



Efficient Shape Matching and Retrieval at Multiple Scales

Evangelos Milios
Euripides Petrakis

Technical Report CS-1998-11

December 2, 1998

Department of Computer Science
4700 Keele Street North York, Ontario M3J 1P3 Canada

Efficient Shape Matching and Retrieval at Multiple Scales

*Evangelos Milios**

Euripides G.M. Petrakis[†]

Dept. of Computer Science

Dept. of Electrical and Computer Engineering

York University

Technical University of Crete

Toronto Canada

Chania, Crete, Greece

December 2, 1998

Abstract

We propose a novel approach for the recognition, clustering and retrieval of shapes. The heart of our methodology is a shape matching algorithm based on dynamic programming, that operates implicitly at multiple scales, but avoids the heavy computational cost of the explicit curvature scale space representation. Our method provides for clustering, visualization and browsing of a data set, as well as for indexing, achieving up to three orders of magnitude speed-up over sequential scanning. We illustrate the application of our method to real two-dimensional static hand gesture data. We also demonstrate the superiority of our approach over traditional approaches to shape matching and retrieval, such as Fourier descriptors, Geometric and Sequential moments. Our evaluation is based on human relevance judgements following a well established methodology from the information retrieval field.

1 Introduction

Object recognition is an important problem in computer vision and has received considerable attention in the literature. Most approaches to object recognition are *model-based* [1] emphasizing the accuracy of recognition. However, the increasing amounts of image data in many application domains has generated additional interest for real-time management and retrieval of shapes [2]. There, the emphasis is not only on accuracy, but also on efficiency (i.e., speed) of retrieval. Addressing such issues has become object of intensive research activities in both the Computer Vision and Database research fields over the past few years [3, 4]. As observed in [5], there

*e-mail: eem@cs.yorku.ca. Corresponding author. This work was supported by a grant from the Natural Sciences and Engineering Research Council of Canada.

[†]e-mail: petrakis@ced.tuc.gr. This work was carried-out while the author was visiting York University.

is a need for increased communication between the two research communities to deal with the above issues. Combining results from both areas is an important next step. This article is a contribution in this direction.

There are three general goals common to all object recognition systems:

Robustness: The system must be able to recognize or retrieve similar objects even if they are noisy or distorted.

Accuracy: It must be able to recognize or retrieve similar objects with as few errors as possible.

Efficiency: It must be *fast*.

Object recognition is based on appropriate representations obtained from all objects during a preprocessing stage. These representations are then used by matching algorithms to determine similarities between objects. In most cases, objects are considered similar if they have similar shapes. Then, the performance of any object recognition system ultimately depends on the types of shape representations used and on the matching algorithms applied.

Below are some criteria for a reliable shape representation:

Uniqueness: A representation must uniquely specify a shape.

Robustness: It must be robust against noise.

Proportionality: Small shape distortions should result in small variations in the representations.

Invariance: A representation must be invariant to translation, scale, rotation and symmetric transformations of the shape.

Scalability: A representation must contain information about the shape at many levels of detail so that, similar objects can be recognized even if they appear at different view-scales (resolution).

Efficiency: A representation must be computationally efficient.

Two shapes are never exactly the same. Then, given their representations, a matching algorithm must determine how similar they are. The definition of appropriate similarity or distance criteria (measures) is a key issue here. A shape matching algorithm must take advantage of the properties of its underlying representation.

Traditionally, to determine which shapes in a collection are similar to a given query shape, all stored shapes have to be matched with the input or query shape. In this case, time responses increase linearly with the number of stored shapes, therefore they may become very slow especially when the shape database is large. Response times can be speeded up significantly by incorporating techniques into shape matching and search that quickly

narrow down the search to only a small number of promising shapes. This can be achieved in two ways: (a) Indices are computed from all the shapes in a collection and stored separately forming an index structure; this index structure is searched first to narrow down the search and (b) Matching a query with a stored shape is performed hierarchically in stages, eliminating non-promising shapes from the early stages of matching.

A new category of methods based on the representation of a shape's inflection points at various scales, called *Curvature Scale Space (CSS)* representation, have been proposed. These methods bring together all the attractive properties for reliable shape representation referred to above (in particular representation at various levels of detail), except that of efficiency: Computing the *CSS* representation is very time intensive. In this work we tackle this problem and we propose a shape matching algorithm which is both fast and accurate.

Specifically, we deal with the following general problem:

- We have a collection of N images representing two-dimensional closed shapes (i.e., image hand gestures in our case).
- Given a prototype shape, we want to find the n most similar shapes or all shapes below a distance threshold t .
- We need to respond to such queries *asymptotically faster* than sequential scanning.

We assume that objects are segmented into closed contours. For our purposes, we take images under good lighting conditions and low noise. Segmentations are carried out by taking the polygonal approximation of their boundary after thresholding. However, automatic object segmentation, in the general case, is difficult and it is outside the scope of this paper.

In summary, the contributions of this work in the following:

- We propose a shape matching algorithm which is at least as accurate as the previous *CSS*-based matching algorithms but much faster. We have identified instances where our method reports a match between two shapes while previous methods fail to find one.
- Our method maps shapes into low-dimensionality points allowing visualization, clustering and browsing of a collection of shapes.
- Our method allows for indexing of shapes at multiple scales, while achieving up to three orders of magnitude speed-up over sequential scanning.
- We establish the superiority of our method over traditional shape matching methods such as Fourier descriptors, Sequential and Geometric moments.

- We introduce to the Computer Vision community a well established method from information retrieval for the empirical evaluation of retrieval results obtained by many competing methods.

We tested our methodology on a data set of 980 two-dimensional hand gesture images but our method can be applied on any kind of shapes.

The rest of this work is organized as follows: A review of related work in the areas of Computer Vision and DataBases is presented in Section 2. The proposed matching algorithm is discussed in Section 3. Our approach to indexing and retrieval is presented in Section 4. The conclusions and the issues for future research are discussed in Section 6.

2 Survey of Related Work

In the following, we present related work in the area of Computer Vision and Database research.

2.1 Shape Matching

Natural shapes [6, 7] are rarely rigid or describable by a small set of transformation parameters [8]. They are typically characterized by smooth boundaries and continuous variations, making it difficult to apply to them feature-based techniques [9]. Similarity of natural shapes is not easy to capture computationally, in spite of the human ease in visually identifying it.

Multiscale representations of shape are considered the most promising, due to their ability to take into account the evolution of shape as it is subjected to more and more smoothing. Different forms of scale space descriptions have been proposed [6, 10, 11, 12, 13].

One class of contour tracking and matching methods relies on physical models of the deformation and is based on minimization of an energy function, without first extracting a symbolic representation of the shapes [14, 15, 16, 17]. More recently, matching of contours has been addressed with dynamic programming together with detection of contours in image sequences. In [18, 19] dynamic programming is used to minimize a cost function that accounts for displacement of a contour in a pair of images from an image sequence. In [20], dynamic programming is used to fit a closed curve template to an image (deformable template matching).

Another class of matching methods relies on symbolic entities extracted from shape contours [21, 22, 23, 12]. Dynamic programming has been a popular approach for matching such symbolic entities [24, 25, 26, 27]. In [24] the inability of dynamic programming to combine contour segments is mentioned, as well as the fact that differing resolutions in the matched contours will lead to reduced performance. In [26] deletions and insertions

of features (corners in a polygonal representation) as well as smoothing of features (i.e. dropping corners) is incorporated in the dynamic programming scheme. This type of smoothing lends a primitive multiple-scale character to the method. In [25] dynamic programming is used to guide the application of grammar rules that transform one shape into another, in the spirit of [7]. In [27], matching proceeds both forward and backward from a support match between two features (landmark points) that are maximally similar. Features are extracted based on their persistence across scales. However, there is no matching of features at multiple scales. Therefore, multiple scales are used as a preprocessing stage only.

Building upon the previously mentioned work, Ueda and Suzuki [23] propose a sophisticated dynamic programming (DP) algorithm, which can group segments together in order to come up with appropriate correspondences. For example, if one shape is slightly noisier than the other, it is possible for a single convex segment to be broken up into a sequence of two convex segments with a concave segment between them. The algorithm in [23] is capable of combining the three (or more) segments in the noisier shape and associating them with the single segment of the less noisy shape. The algorithm of [23] uses the scale space representation of [11] to constrain the possible merges, i.e. it accepts merges that are only present at coarser scales of the scale space representation.

Our algorithm is a substantial extension of the dynamic programming algorithm in [23], in the following ways:

- The algorithm in [23] performs a best-only search in a DP table as it looks for minimum-cost paths in the dynamic programming framework. We have identified instances, in which a best-only search strategy fails to find a valid match between two shapes, although one exists. To address this deficiency, we have extended the dynamic programming algorithm to perform k -best search, and we have demonstrated experimentally that for a small k (e.g., 5), the additional space and time requirements are modest and the algorithm can solve matching problems where the original one fails.
- We have implemented a different set of cost measures from the original algorithm, and we have demonstrated improved performance with them. We present an intuitive justification for the new cost measures and we show experimental results.
- We propose that the scale space restriction be removed from the original algorithm. The scale space computation has two drawbacks. First, it tends to diffuse the effects of a feature far away from its location as coarser and coarser scales are considered. As a result, the locality of the features is lost at coarser scales. Second, it is computationally expensive. We present a formulation of the algorithm that does not use scale space to restrict search for segment merges, and we demonstrate that the results are comparable to those of the original formulation.

In an earlier approach by Mokhtarian and Mackworth [28] matching is performed through “*interval trees*” which are computed by tracking the scale space representation from coarser to finer scales. In [23] it is demonstrated, however, that small shape changes may cause major structural changes in the interval tree and this may lead to matching errors.

The problem of indexing based on scale space representations is difficult due to the complexity of the underlying *CSS* representations as well as due to the large amounts of data in such representations. In [23] neither the problem of shape indexing nor of efficiency of retrieval is addressed. Mokhtarian [29] base their matching on the maxima of curvature zero-crossing contours of the *CSS* image. Sequential scanning is implicitly assumed as the retrieval mechanism in both methods.

Additional work on shape indexing include, methods based on multidimensional indices [30, 31, 32], and methods based on hashing [33, 34]. However, none of these methods works in scale space. Finally, the method by Günsel and Tekalp [35] works in the modal feature space, does not require the extraction of connected boundaries but, does not support indexing and cannot handle heavily deformed shapes.

2.2 Spatial Access Methods (SAMs)

We can achieve faster-than-sequential searching by using spatial access methods. These are based on file structures to manage a large collection of f -dimensional points (or rectangles) stored on the disk so that, *range queries* can be efficiently answered. A range query specifies a region (e.g., hyper-rectangle or hyper-sphere) in the address space, requesting all the data objects that intersect it. If the data objects are points (as eventually happens in our application), the range query requires all the points that are inside the region of interest. An example of a range query on point data is “*retrieve all the cities that are 200 km away from Brussels*”. Spatial access methods can also handle “*nearest neighbor*” [36] and “*all-pairs*” (or “*spatial-join*”) [37] queries. For clarity, in this paper we focus on range queries only.

Several spatial access methods have been proposed forming the following classes: (a) Methods that transform rectangles into points in a higher dimensionality space [38]; (b) Methods that use linear quad-trees or, equivalently, the “*z-ordering*” [39] or other “*space filling curves*” [40]; and finally, (c) Methods based on trees (k-d-trees [41]). One of the most characteristic methods is the R-tree [42].

Methods referred to as “*metric trees*” are based on the idea of indexing using distance information. Vantage-Point (VP) trees [43] and Geometric trees (GNAT) [44] are characteristic examples. Most of these methods require expensive preprocessing for building a tree index structure. An alternative to metric trees is FastMap [45], a fast algorithm that transforms data entities (e.g., shapes in our application) into multidimensional points.

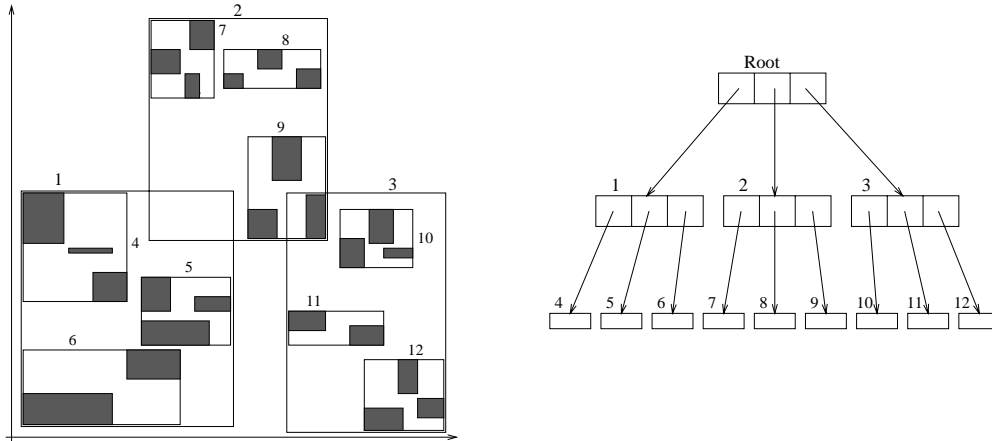


Figure 1: Data (dark rectangles) organized in an R-tree.

In this work, we used Fastmap in conjunction with R-trees for indexing. The reasons for our choices are justified below.

2.3 R-trees

The R-tree can be envisioned as an extension of the B-tree for multidimensional objects. R-trees have the desirable property to remain robust for high-dimensionality spaces.

A geometric object is represented by its Minimum Bounding Rectangle (MBR). Non-leaf nodes contain entries of the form (ptr, R) where ptr is a pointer to a child node in the R-tree; R is the MBR that covers all rectangles in the child node. Leaf nodes contain entries of the form $(object - id, R)$ where $object - id$ is a pointer to the object description, and R is the MBR of the object. The main idea in the R-tree is that father nodes are allowed to overlap. This way, the R-tree can guarantee good space utilization and remains balanced. Figure 1 illustrates data rectangles (in black) organized in an R-tree (left); the file structure for the same R-tree is also shown (right); the nodes correspond to disk pages.

Extensions, variations and improvements to the original R-tree structure include the packed R-trees [46], the R^+ -tree [47], the R^* -tree [48], and the Hilbert R-tree [49]. Recent extensions for high-dimensions include the X-tree [50] and the SR-trees [51]. We used R-trees solely because of availability; any spatial access method would do, like, e.g., R^* -trees and X-trees.

2.4 FastMap

FastMap is an algorithm that takes in a set of N data items (e.g., images), together with a distance function $D()$, and map these data items into points in some f -dimensional space (f is user defined), such that distances

are preserved. It has the following attractive properties:

- It is as fast as it can be, since it is *linear* on the number of items. (No mapping algorithm can be better than linear, unless it operates on a sample of the database, which FastMap can do, too.) Specifically, it needs $\mathcal{O}(fN)$ distance calculations.
- It is dynamic: New data can be mapped after the original data have been mapped, without having to redo the mapping from the beginning. The mapping for a new item (or query) takes $2f$ distance calculations, that is, constant on the size of the database.
- The algorithm does not require the distance measure to be Euclidean. However, on non-Euclidean distances, we may have false dismissals, as we see later.

3 Shape Matching

The shape matching algorithm that lies at the core of our methodology takes in two shapes and computes: (a) Their distance; the more similar the objects are, the lower the value of the distance function and (b) The correspondences between similar parts of the two shapes. In retrievals, only distances between shapes are used. However, the correspondences help assess the plausibility of the distance computation, if necessary.

3.1 Definitions

Let A and B be the two shapes to be matched. The first step is to segment both shapes. A segment is a part of a shape contour between two consecutive inflection points and it is either convex or concave.

The major problem with segmented representations is that small perturbations to the shape can yield large changes in the segmentation. Therefore the matching algorithm must be robust to segmentation changes. The standard fix is to represent the shape at multiple scales of resolution (smoothing), and either use a full scale-space representation for matching [28], or have the algorithm choose different scales for different parts of the shape [23]. In both approaches the major cost is the computation of the scale-space representation. Furthermore, the standard scale-space representation (obtained by smoothing with Gaussian filters of varying scales) has the characteristic that the more prominent features tend to shift the position of and finally absorb the less prominent ones, something that may be undesirable in cases when the smaller features have perceptual significance. Our shape matching algorithm follows the paradigm of [23], but avoids the expense of computing the full scale-space representation. Merging of neighboring segments is key to both algorithms, with the difference that [23] only allows segments to merge in matching only if they merge in the standard scale-space representation. Our

algorithm allows all segment merges, and relies only on the minimization of the overall matching cost to select the merges.

Figure 2 shows two two-dimensional silhouettes of hand gestures, extracted from real video images. The large circles indicate inflection points that are transitions from convex to concave and the small circles from concave to convex, when the shapes are traversed in a clockwise fashion.



Figure 2: Shape examples

The number of segments in shapes A and B are N and M respectively; elements of A and B are indexed by i and j respectively; inflection points are denoted by p_i and p_j ; all i subscripts are modulo N and all j subscripts are modulo M . Let

$$A = a_0, a_1, \dots, a_{N-1}$$

$$B = b_0, b_1, \dots, b_{M-1}$$

be the convex/concave segment sequences of the two shapes with a_i being the segment between two consecutive inflection points p_i and p_{i+1} ; similarly for b_j . Then, $a(i - n|i)$, $n \geq 0$, is the sequence of segments $a_{i-n}, a_{i-n+1}, \dots, a_i$; similarly for $b(j - m|j)$, $m \geq 0$.

The algorithm searches for segment correspondences from the finest to coarser scales by merging an odd number of consecutive segments at the finest scale, if such a merging replacement can lead to the minimization of a cost function. Each merging should be a recursive application of the grammar rules $CVC \rightarrow C$ and $VCV \rightarrow V$, where C and V denote convex and concave segments respectively [7, 25]. This is enforced by the algorithm. A *complete match* is a correspondence between groups of consecutive segments in order, such that no segments are left unassociated.

The goal is to find the *best* association of segments in shape A to segments in shape B . The problem of finding the best correspondences between the two shapes is formulated as a minimization problem which is solved efficiently by Dynamic Programming (DP): A table of partial costs is built and the optimal matching is searched in the form of a path in the DP table that minimizes a dissimilarity cost. The method involves building various paths in the DP table matching groups of segments from the two shapes and, finally, choosing the path that leads to a complete match and has the minimum cost.

The DP table has $2N$ columns and $2M$ rows, corresponding to segments of shape A and shape B respectively repeated twice (to force the algorithm to consider all possible starting segment pairs between the two shapes). A link between cells (i_{w-1}, j_{w-1}) and (i_w, j_w) denotes the matching of segments $a(i_{w-1} + 1|i_w)$ with $b(j_{w-1} + 1|j_w)$. A *path* is a linked sequence of cells $(i_0, j_0), (i_1, j_1), \dots, (i_w, j_w)$, not necessarily adjacent, indicating a partial match. Function $\psi(a(i_{w-1} + 1|i_w), b(j_{w-1} + 1|j_w))$ represents the dissimilarity cost between its two arguments and is defined later in this section.

In the following, the term *options* is used for the number of K best choices stored at each cell of the dynamic programming table. The cell at the intersection of column i and row j will be referred to as $cell(i, j)$. Each cell contains the cost array $g[k_w]$ and associated bookkeeping data $t_1[k_w], t_2[k_w], index[k_w], g_n[k_w], g_m[k_w]$, where k_w varies from 0 to $K - 1$ and refers to the k_w -th best path, or partial match, up to and including $cell(i_w, j_w)$. Specifically, $g[k_w]$ holds the cost of the path, $t_1[k_w]$ and $t_2[k_w]$ hold the number of unmatched segments in shapes A and B respectively for the path and $index[k_w], g_n[k_w]$ and $g_m[k_w]$ hold the back links for the path and allow the backward tracing of the path. If $g_n[k_w] = n_w$ and $g_m[k_w] = m_w$ for $cell(i_w, j_w)$, then the previous cell and option in the path is $cell((i_{w-1}, j_{w-1}) = cell(i_w - 2n_w - 1, j_w - 2m_w - 1)$ and $index[k_w]$ respectively, where n_w and m_w are nonnegative integers. Notice that, $i_w = i_{w-1} - 2n_w - 1$ and $j_w = j_{w-1} - 2m_w - 1$, where $n_w, m_w \geq 0$, since the number of segments which are merged is always odd.

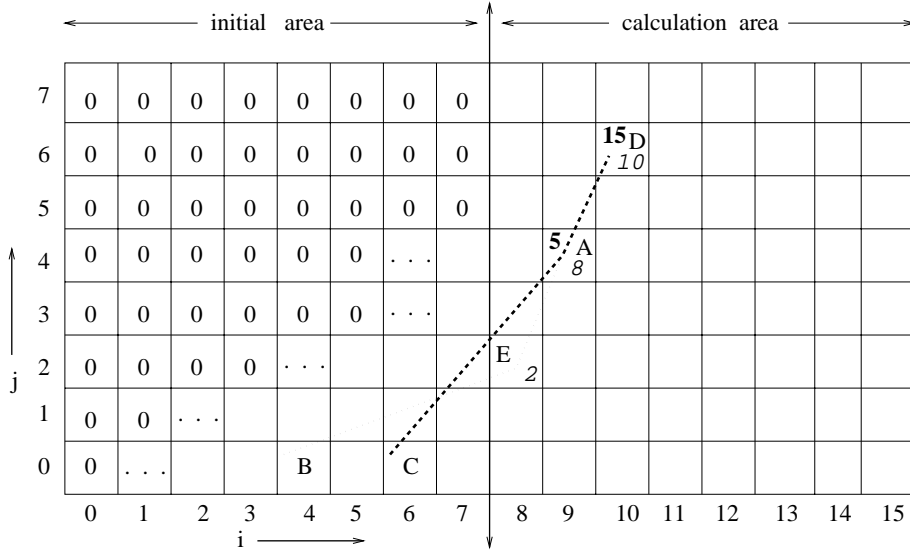


Figure 3: Example of a DP table.

The DP table consists of two distinct areas, the initial value area (left half) and the calculation area (right half). In the initial value area all g terms are initialized to zero, t_1 to N and t_2 to M , implying that each of these cells can act as the first cell in a path. The calculation area is computed and finally the optimal path is searched. All paths of length greater than zero lie in the calculation area. A path is complete when its corresponding t_1 and

t_2 at the final cell of the path, both become zero simultaneously. Figure 3 illustrates an example of a DP table computed for the matching of two shapes with 8 and 4 segments respectively. Two incomplete paths are shown ending at $cell(11, 7)$.

3.2 The algorithm

The main idea in the algorithm is to fill the DP table cells and then search for and trace back the optimal complete path. Filling the cells involves the computation of the K elements of cost array $g(k_w)$ in each cell and the associated bookkeeping information. Figure 4 outlines the algorithm.

```

for  $i_w = N - 1, N, \dots, 2N - 1$  do
  for  $j_w = 0, 1, \dots, 2M - 1$  do
    fill the  $K$  options in  $cell(i_w, j_w)$ ;
    check if a complete path has been found;
  end for
end for
select the complete path with the lowest cost over the whole DP table;
retrace corresponding shape match by following backward links;

```

Figure 4: Outline of the shape matching algorithm.

The *for* loop for j_w does not run over all the indicated values, but only over those values that do not involve convex to concave segment associations, which are not possible.

To describe the filling of $cell(i_w, j_w)$ of the DP table with values, we need the following auxiliary functions:

- $xmin(list, x)$ extracts the x smallest terms from $list$.
- $listat(i_w, j_w, n_w, m_w)$ is the list of the costs of all options at $cell(i_w - 1, j_w - 1)$, augmented by the dissimilarity cost $\psi(a(i_w - 1 | i_w), b(j_w - 1 | j_w))$ of extending them to $cell(i_w, j_w)$.
- $fmin(f, i)$ returns the integer pair (n, m) that leads to the i -th smallest result in the evaluation of a given function $f(n, m)$.

The function f required for filling the k_w option of $cell(i_w, j_w)$ is

$$f(n, m) = g(i_w - 2n - 1, j_w - 2m - 1, k_w) + \psi(a(i_w - 2n | i_w), b(j_w - 2m | j_w)), \quad n, m \geq 0. \quad (1)$$

where $g(i_w, j_w, k_w)$ is the value of $g[k_w]$ for $cell(i_w, j_w)$.

The cost ψ in the above equation is the cost of association of segments $a(i_w - 2n | i_w)$ with $b(j_w - 2m | j_w)$ and consists of a merging cost component and a dissimilarity cost component (see Section 3.3).

With the above definitions, we fill the cost array $g[k_w]$ at $cell(i_w, j_w)$ with the K values computed and collected by the following loop:

```

for  $x = 0, 1, \dots, K - 1$  do
    collect  $xmin(listat(i_w, j_w, fmin(f, x)), K)$ ;
end for

```

Function $fmin(f, i)$ searches for n 's over the range $[0, \frac{N-1}{2}]$ and for m 's over the range $[0, \frac{M-1}{2}]$ to form acceptable pairs (n, m) . Additional constraints on acceptable pairs (n, m) are that either $t_1 = t_2 = 0$ (a complete match has been found), or $t_1 > 0, t_2 > 0$ (there exist unmatched segments in both shapes).

A constraint in the original method proposed in [23] is the curvature scale space constraint on (n, m) that $a(i_w - 2n|i_w)$ and $b(j_w - 2m|j_w)$ must actually merge in the scale space representation at some scale, not necessarily the same for the two shapes. In our method, we have removed this constraint, and we rely on the merging costs alone to select the optimal merges.

3.3 Cost Components

The cost term ψ in Equation 1 can be rewritten as $\psi(a(i_{w-1} + 1|i_w), b(j_{w-1} + 1|j_w))$. Its computation must rely on geometric properties of the segments themselves. This cost term consists of three additive components [23]:

$$\begin{aligned}
 \psi(a(i_{w-1} + 1|i_w), b(j_{w-1} + 1|j_w)) = & \tag{2} \\
 & MergingCost(a(i_{w-1} + 1|i_w)) + \\
 & MergingCost(b(j_{w-1} + 1|j_w)) + \\
 & \lambda DissimilarityCost(a(i_{w-1} + 1|i_w), b(j_{w-1} + 1|j_w)),
 \end{aligned}$$

where λ is a constant that represents the relative importance of the merging and dissimilarity costs. In the experiments λ was set to 1.

The first two terms in Equation 3 represent the cost of merging segments $a(i_{w-1} + 1|i_w)$ in shape A and segments $b(j_{w-1} + 1|j_w)$ in shape B respectively while, the last term is the cost of associating the merged $a(i_{w-1} + 1|i_w)$ with the merged $b(j_{w-1} + 1|j_w)$.

Requirements for reliable cost computation are the following:

- Merging should follow the process grammar rules [7, 25], i.e. each allowable merging should be a recursive application of the grammar rules $CVC \Rightarrow C$ and $VCV \Rightarrow V$. This is enforced by the DP algorithm.

- Absorbing a “visually prominent” segment (i.e., a long segment with high curvature) into a merged segment of the opposite type should incur a high cost. To specify this requirement, we need to define *visual prominence* in geometric terms.
- The partial cost components arising from different features of the shape should be combined into a total cost in a meaningful way.

The heuristic cost computations that follow attempt to satisfy the above requirements.

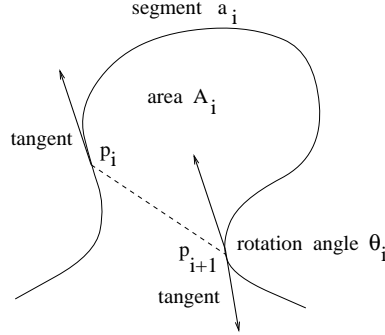


Figure 5: Geometric quantities for defining the prominence of a segment

We now define geometric quantities (features) needed in the specification of visual prominence of a segment according to Figure 5.

Rotation Angle θ_i is the angle traversed by the tangent to the segment from inflection point p_i to inflection point p_{i+1} and shows how much a segment is curved.

Length l_i is the length of segment a_i .

Area A_i is the area enclosed between the line segment (chord) and the arc between the inflection points p_i and p_{i+1} .

Dissimilarity cost computation: This cost computation should assign a higher cost to segments (or groups of segments) with large differences in more than one feature. The dissimilarity cost of associating segments of shape A with segments of shape B is computed using the following equation:

$$DissimilarityCost = W \max_{all\ features\ f} \{d_f\}, \quad (3)$$

where $f = l, \theta$ or a .

We choose the *max* operation instead of product (as in [23]) because in the product a small cost in terms of one feature can cancel the effect of a high cost in terms of another feature, something that may lead to a visually implausible outcome. The max operation addresses this problem.

Factor W equals the number of features for which d_f is greater than $0.75 \times \max\{d_f\}$, where $f = l, \theta, a$. Thus, if all three features have uniformly large d_f , then the dissimilarity cost is multiplied by 3.

The term d_f , for $f = \theta$ is defined as

$$d_f = \left| \frac{\Theta_a - \Theta_b}{\Theta_a + \Theta_b} \right|, \quad (4)$$

where $\Theta_a = \sum_{s=i_{w-1}+1}^{i_w} \theta_s$, and $\Theta_b = \sum_{s=j_{w-1}+1}^{j_w} \theta_s$, and θ_s being the rotation angle of segment with index s of shape A and shape B respectively.

The term d_f , for f being l (length) or area (a), is defined as

$$d_f = \left| \frac{f_a}{F_a} - \frac{f_b}{F_b} \right|, \quad (5)$$

where $F_a = \sum_{s=0}^{N-1} f_s$, $f_a = \sum_{s=i_{w-1}+1}^{i_w} f_s$ of shape A and similarly for F_b, f_b of shape B .

Merging cost computation: Let the types of the segments being merged be $CVC \dots VC$, where C is convex and V is concave (the opposite case is obtained by switching C and V). The merging cost is defined as follows:

$$MergingCost = \max_{all\ features\ f} \{w_f c_f\}, \quad (6)$$

where subscript f refers to a feature (length, area or rotation angle).

We choose the above maximization formulae instead of sum of products of terms comparing consecutive segments (as in [23]) for the following reasons: We used max instead of product, because in a product a small cost in terms of one feature can cancel the effect of a high cost in terms of another. The reason for abandoning the sum of consecutive segments is because it implies that the plausibility of merging several segments can be reduced to the similarity of consecutive segments, which may not necessarily be true. Consider, for example, the case of a short and flat segment next to a long and curved one. In this case, it is plausible to merge the two, however the merging cost in [23] will be high. Another drawback of the use of a sum is that the merging cost increases with the number of segments merged, even if several very short segments are being merged into a large one.

For f being length or area:

$$c_f = 1 - \frac{\sum_C\ segs\ of\ group\ f - \sum_V\ segs\ of\ group\ f}{\sum_{all\ segs\ of\ shape\ f}}. \quad (7)$$

For f being rotation angle:

$$c_f = 1 - \frac{\sum_C\ segs\ of\ group\ f - \sum_V\ segs\ of\ group\ f}{\sum_C\ segs\ of\ group\ f + \sum_V\ segs\ of\ group\ f}. \quad (8)$$

The intuition behind these formulae is that they measure the visual prominence of the features of the absorbed segments (of type V) relative to the absorbing segments (of type C). All costs c_f are within the interval $[0, 2]$. Cost c_f is close to 0 if the convex segments visually dominate the concave ones (hence it is plausible to absorb the concave ones), while it is close to 2 if the concave segments visually dominate the convex ones (hence it is not plausible to perform the merge, therefore the merging cost should be high).

For f being any feature (length, area, rotation angle) the weight term is

$$w_f = \frac{N}{2} \frac{\sum_V \text{segs of group } f}{\sum_V \text{segs of shape } f}. \quad (9)$$

The intuition behind the weight term is to measure the visual prominence of the absorbed segments within the shape as a whole. Factor $\frac{N}{2}$ is heuristic, and it could instead be incorporated into the relative weight λ between the merging and dissimilarity costs, as defined next.

3.4 The Ueda-Suzuki Algorithm

The algorithm by Ueda and Suzuki [23] is a special case of the above K -option DP algorithm with $K = 1$, and with the additional constraint that segment merges must actually appear in coarser scales. The motivation for introducing the K -option method is that, by the use of only one option at each cell, the optimal path is sometimes missed. This happens because the (n, m) pair chosen at $cell(i, j)$ is the one which leads to the least cost path only up to $cell(i, j)$. This path may not be the one that leads to the formation of a complete path. It is possible that after the entire DP table is computed, there exist only incomplete paths, in which case the algorithm fails to produce a match. Also, the best choice at $cell(i, j)$ may lead to an expensive match of the remaining unmatched segments.

This is illustrated in figure 3, which shows the DP table for matching two shapes with 8 and 4 segments respectively. Cells in the table are marked as follows: $(4, 0)$ as B , $(6, 0)$ as C , $(8, 2)$ as E , $(9, 4)$ as A and $(10, 6)$ as D . The number in bold at a cell represents the cost of the path up to that cell using the $K = 1$ option (dashed line). Oblique numbers represent the cost along the alternate path ($K = 2$, dotted line). Alternate path $BEAD$ is ignored by the $K = 1$ option algorithm because of a least cost choice made at A . This problem is solved by the K -option DP method where the choice of a best subpath is not made at cell A but deferred until a later point when more information is available.

Examples of this kind of failure of the 1-option algorithm in a real situation of matching the silhouettes of two hand gestures, along with two additional optimizations of the 1-option algorithm as presented in [23] are described in [52].

3.5 Experiments with Matching

The goal of this set of experiments is to illustrate the superiority of our proposed matching algorithm over the algorithm by Ueda and Suzuki [23].

In the generalized algorithm presented above, we have the following options:

- Matching involving the computation of the *CSS* representation or matching without it.
- Matching using the cost functions of [23] or the new proposed in the previous section.
- Matching with the best-search option ($K = 1$) as in [23] or with the K -best-search option ($K > 1$) proposed in the previous section.

The algorithm in [23] corresponds to the use of *CSS*, the original cost computations and $K = 1$. In the experiments described below, we used $K = 5$ to avoid the situations in which the $K = 1$ algorithm fails to find a solution, where one exists. Among the several combinations of the above three options, in the following, we discuss the most characteristic ones:

Method 1: Matching using the Ueda-Suzuki algorithm [23].

Method 2: Matching with the cost definitions of [23], searching for the best 5 paths ($K = 5$) and without the *CSS* representation.

Method 3: Matching with our proposed cost definitions, searching for the best 5 paths using *CSS*.

Method 4: Matching with our cost definitions, searching for the best 5 paths in the DP table and matching without the *CSS* representation. This is essentially our proposed algorithm.

Figure 6 shows a model database of two-dimensional silhouettes of hand gestures, extracted from real video images. In the following, the two silhouettes of Figure 2 are used as queries and are matched against the above 17 model silhouettes. They are referred to as *query 1* (left silhouette) and *query 2* (right silhouette).

Table 1 from [52] shows the results of retrieving all the model shapes using *query 1*. Table 2 shows the same experiments when *query 2* is used. In these tables, the models have been sorted by increasing cost, hence the top one is the most similar according to the respective algorithm used. The cost values have been normalized so that the minimum cost for each algorithm equals to 1. Notice that, for query 1, only methods 3 and 4 retrieve correct best match.

Apart from the failure of the Ueda-Suzuki algorithm (method 1) to identify the correct match, we also notice that this algorithm is more “uncertain” in its inferences than the new cost versions, as in the latter the cost of

the closest incorrect match is at least 20% larger than the cost of the correct match. Our experiments confirm that relaxing the CSS constraint, while using the new cost measures, improves performance compared with the original cost computations with the CSS constraint while saving a lot of computation time.

<i>method 1</i>		<i>method 2</i>		<i>method 3</i>		<i>method 4</i>	
<i>model</i>	<i>cost</i>	<i>model</i>	<i>cost</i>	<i>model</i>	<i>cost</i>	<i>model</i>	<i>cost</i>
4	1	12	1	3	1	3	1
17	2	3	3	12	18	12	18
13	2	9	14	17	32	17	28
9	2	17	17	2	33	2	51
3	6	8	17	8	33	13	61
12	8	13	21	10	42	15	64
5	22	10	22	9	49	4	72
16	25	11	25	15	52	8	75
1	46	2	29	11	55	9	75
8	49	7	33	13	60	16	76
7	54	15	36	5	63	11	83
10	61	5	41	16	66	7	84
2	64	6	50	14	67	10	90
14	77	4	59	7	68	6	90
15	82	14	69	6	75	1	95
11	100	16	78	1	84	5	98
6	100	1	100	4	100	14	100

Table 1: Matching query 1 to the model database. Method 1 corresponds to the Ueda-Suzuki method while, method 4 corresponds to our proposed method. Methods 2 and 3 correspond to intermediate settings of the matching options.

Figure 8 shows the results of matching query 1 with the the same model (model 8). In this experiment we illustrate the segment correspondences between the two shapes reported by all methods. The large circles indicate inflection points that are transitions from convex to concave, and the small circles from concave to convex, when the shapes are traversed in a clockwise fashion. Lines connecting inflection points in the two shapes define the associations between shape parts computed by the algorithm.

Figure 7 illustrates the results obtained by matching query 2 with model 3. Again, all methods report only correct associations but not always the same.

4 Shape Retrieval

The heart of our approach is the matching algorithm presented above. Given the shape matching algorithm and a database of N shapes, the obvious method to search for similar shapes is sequential scanning. However, this

<i>method 1</i>		<i>method 2</i>		<i>method 3</i>		<i>method 4</i>	
<i>model</i>	<i>cost</i>	<i>model</i>	<i>cost</i>	<i>model</i>	<i>cost</i>	<i>model</i>	<i>cost</i>
6	1	8	1	8	1	8	1
9	9	17	9	17	11	17	21
17	30	11	18	3	47	9	40
12	48	9	20	6	47	2	42
3	55	6	24	7	47	13	45
13	64	12	25	11	48	3	49
2	66	2	34	9	61	7	51
8	69	13	35	13	61	11	53
15	80	4	38	16	64	6	56
7	81	5	43	4	66	12	56
14	86	16	45	12	68	15	59
10	92	10	49	15	75	16	67
11	94	15	53	1	75	4	78
1	95	7	57	10	78	14	82
6	95	3	73	5	83	1	86
5	99	1	94	14	84	10	95
4	100	14	100	2	100	5	100

Table 2: *Matching query 2 to the model database.*

requires always N distance computations. We can speed-up retrievals using indexing.

The main idea behind our approach is to transform each shape to a point in an f -dimensional space. The mapping of shapes to f -dimensional points is achieved through FastMap [45]: The FastMap algorithm accepts as input N shapes, our distance function and f , the desired number of dimensions, and maps the above shapes to N points in an f -dimensional space. The complexity of this mapping is $\mathcal{O}(Nf)$ distance computations. Notice that this operation could (and should) be off-line. Similarly, when a query is given, it is quickly mapped to a point into the above f -dimensional space requiring only $\theta(f)$ distance computations. Then, the problem of database search is transformed into one of spatial search.

To speed-up retrievals, the f -dimensional points are indexed using an R-tree [42]. We used R-trees solely because of availability; *any* spatial access method would do, like e.g., X-trees [50]. Figure 9 illustrates the above sequence of operations. Below we discuss each one of these processing steps separately.

4.1 Shape Indexing

Figure 10 demonstrates the proposed file structure of the data on the disk. Specifically, the file structure consists of the following parts:

- The “*shape file*” holding the original shapes along with their boundary contours.
- The R-tree holding an f -dimensional vector for each stored shape.

4.2 Search Strategy

The user specifies a query shape and a tolerance t and asks for all shapes within that tolerance. The f -dimensional vector of the query is computed first using FastMap. Then, all vectors within distance t are retrieved from the R-tree. As we show later, the R-tree may return *false alarms* (i.e., non qualifying shapes). A post-processing step is required to clean-up the false alarms. The generic search algorithm is as follows:

R-tree search: Issue a range query on the R-tree to obtain a list of promising shapes (their identifiers).

Clean-up: For each of the above obtained shapes, retrieve its corresponding contour from the shape file and compute the distance between this shape and the query. If the distance is less than the threshold t , the shape is included in the response set.

Nearest-neighbor queries, such as “*give me the 10 most similar shapes*”, can also be answered using the algorithm in [36].

5 Experimental Results

We carried out several groups of experiments. The experiments were designed to illustrate the:

Superiority of our shape matching algorithm over traditional algorithms such as those examined in [2]. We experimented with Fourier descriptors, Sequential and Geometric moments.

Speed-up of our method over sequential scan searching. We studied the search time for various values of the tolerance t and of the dimensionality f . We show that our method can be several times faster than sequential scanning.

Accuracy of our indexing method. Our indexing scheme may miss some of the answers that sequential scanning would retrieve. We show that our methodology exchanged a very small loss in accuracy (i.e., 5 - 10%) for several times faster retrievals.

Clustering properties of our method. Similar shapes tend to cluster together and these clusters can be viewed in 2 or 3 dimensions.

To test the efficiency of our methodology we used a data set consisting of 980 synthetic shapes which are generated from the original 17 hand gestures of Figure 6: We took the shape models in pairs and, for each such pair, we produced a number of blend shapes on the process of transforming the one shape to the other using the shape morphing algorithm of [53]. To evaluate our method we took the 17 original model shapes as queries and we computed the average of their responses. Therefore, all measurements below correspond to averages over 17 queries.

5.1 Comparison With Other Methods

The competitors to our method are:

Fourier descriptors [54] : This known to be one of the most successful methods for the recognition of closed shapes. We took the first (lower order) 20 coefficients of the fourier transform.

Sequential moments [55]: This is one of the most effective moment-based methods for closed shapes. For each shape, a representation of 4 moment coefficients is computed from its bounding contour.

Geometric moments [56]: This is the original and the most characteristic representative of a wide class of methods based on area moments. A representation of 7 moment coefficients of the shape is computed from the area it occupies.

In [2], such methods were tested on a database of 500 trademark images. Additional reasons for choosing these methods for our evaluation are: (a) They are translation, rotation and scale invariant (the same as our method) and (b) They all filter out shape detail so that, they can detect similarity at a coarse view-scale. Our method has these advantages too. Notice that matching with such methods is global, as opposed to matching with our method which is local, detecting similarities between shape parts, not necessarily at the same scale for each matched part.

We used human relevance judgements to compute the effectiveness of each method. Two shapes (i.e., a query and a model shape) are considered similar if a human judges that they represent the same figure. To measure effectiveness, for each candidate method we computed:

Precision defined as the percentage of similar shapes retrieved with respect to the total number of retrieved shapes.

Recall defined as the percentage of similar shapes retrieved with respect to the total number of similar shapes in the database. Because we don't have the resources to compare every query with each database shape (i.e.,

this would require, for each method, $17 \times 980 = 16,660$ visual judgements!), for each query, we merged the answers obtained by all candidate methods and we considered this as the database which is manually inspected for relevant entries. This is a valid sampling method known as “pooling method” [57]. Notice however, that this method does not allow for absolute judgements such as “*method A misses 10% of the total similar answers in the database*”. It provides, however, a fair basis for comparisons between methods allowing judgements such as “*method A returns 5% fewer correct answers than method B*”.

Each one of the 17 queries retrieves the best 50 shapes. For answer sets containing between 1 and 50 entries, we computed the average values of precision and recall. These values are represented in a *precision-recall plot*: The horizontal axis corresponds to recall and the vertical axis corresponds to precision. Each method is represented by a curve. Each point in such a curve is the average over 17 queries. The total number of points in each curve is 50 (i.e., we compute precision and recall for answers containing between 1 and 50 shapes). Therefore, the top-left point of the diagram corresponds to the precision/recall values for the best answer (best match) while, the bottom right point corresponds to the precision/recall values for the entire answer set with 50 retrieved shapes.

Figure 11 illustrates the precision-recall diagram for this experiment. For small answer sets returning up to 10 shapes (corresponding to the left-most 10 points of a curve) our method and Fourier descriptors perform about equally in terms of both, precision and recall. Notice that, for such small answer sets, both methods achieve precision close to 1, that is, their answers are almost 100% correct. For larger answer sets, our method performs clearly better than any other method, achieving up to 25% better recall and 20% better precision than the second best method (Fourier descriptors). This result demonstrates that our method is very well suited for database work, where one is interested in retrieving between 10 and 50 shapes.

Both, our method and Fourier descriptors perform much better than moments. An interesting observation here is that, Geometric moments perform better than Sequential moments for answers with more than 30 shapes. Notice that, most literature references (e.g., [55]) report that sequential moments perform always better than Geometric moments, but they examine very small answer sets.

5.2 Response Time

In the following we study the speed-up effect of our indexing scheme over sequential scanning. Times are reported in number of distance computations that is, in number of calls of the matching algorithm. The reasons of this choice are: (a) Response time in seconds vary depending on the implementation of the matching algorithm and of the access structures and (b) The time for distance computations (i.e., 1-2 seconds per matching on a PENTIUM PC 200Mhz) clearly dominates the time for R-tree search (i.e., less than 1 second per query!).

Tolerance	0.1	0.3	0.5	0.7	0.9	1.1	1.3	1.5	1.7	1.9	2.1	2.3	2.5
Shapes	0	0.1	1.9	3.9	7.1	10.0	13.0	18.5	23.8	29.2	34.8	41.8	52.0

Table 3: Average number of retrieved shapes as a function of the tolerance t .

Figure 12 shows the average number of distance computations plotted against the tolerance t for $f = 2, 3, 5, 7$ and 10 dimensions. Table 3 shows the number of retrieved images (after clean-up) as a function of the tolerance t . For $t < 0.5$, that is for queries returning one or two best matches, searching with indexing is up to three orders of magnitude faster than sequential scanning. For $t \leq 2.5$, 52 images are retrieved on the average.

The shape of the curves is justified as follows: Sequential scan, performs always the same number of distance computations. For the proposed method, the shape of the curve resembles as exponential curve. This is expected because the number of shapes which are retrieved from the R-tree increases exponentially with the tolerance t and each retrieved shape yields a shape distance calculation during clean-up.

In all cases, searching with indexing results in faster retrievals but, the speed-up decreases with the dimensionality: As the dimensionality decreases, more points (shapes) are projected from a higher dimensionality space within t distance from the query (false drops) in the lower dimensionality space. The above false drops account for distance computations during clean-up and time responses are slowed-down.

5.3 False Dismissals

The labels in Figure 12 denote average values of accuracy compared to sequential scanning. For example, a value 0.95 denotes that 5% of the answers retrieved by sequential scanning are missed (false dismissals). Notice that, our method achieves high values of accuracy (i.e., greater than 0.90) in most cases. Accuracy drops with the dimensionality but remains as high as 0.90 for answer sets with 50 images. Presumably, a typical user retrieves more than 40 or 50 shapes. In this case, we set $d = 10$. This value assures high accuracy with maximum speed-up. If, however, a user is interested in smaller answer sets with 10 or 20 shapes, then, we would choose $d = 5$. To satisfy both cases of users, we may select to keep both R-tree indices (i.e., with $d = 5$ and $d = 10$) on the disk.

5.4 Clustering

FastMap transforms shapes into f -dimensional points preserving much of the distance information creating clusters with similar shapes. When $f = 2$ or 3 then the above target space can be visualized. This diagram can

be found especially useful for the:

Classification of unknown shapes: We apply FastMap to transform an unknown shape to a multidimensional vector. The unknown shape belongs to a class if it is mapped within its cluster.

Browsing of the contents of shape collection and for the retrieval of similar shapes: Once a query is mapped to a point on the above diagram, one could search manually for similar shapes by browsing shapes in the neighborhood of the query.

Figure 13 illustrates the FastMap diagram for $f = 3$ of 44 shapes belonging to 4 disjoint classes. We observe 4 clusters since the shapes of each class tend to cluster together.

6 Conclusions

We propose a novel approach for shape matching at multiple scales and for handling similarity retrieval in a shape database. The heart of our approach is an algorithm for shape matching in scale space which has the following advantages:

- Corrects failures of previous algorithms (i.e., [23]).
- Employs a newly defined set of cost measures which showed improved performance.
- Does not assume the scale space representation during matching thus saving lot of computation time.

With respect to retrieval, we proposed an indexing method which:

- Achieves up to three orders of magnitude speed-up over sequential scanning with very high accuracy.
- Provides for clustering, visualization and browsing of a shape data set.

Additional contributions of this work are that:

- We demonstrate the superiority of our approach over traditional approaches to shape matching and retrieval, such as Fourier descriptors, Geometric and Sequential moments.
- We introduce to the Computer Vision community a well established methodology for the evaluation of the retrieval results obtained by more than one competing methods.

Current research is directed towards extending our matching algorithm for open curves. A very promising research direction is the study of data-mining algorithms [58] on the point-transformed set of shapes, to detect regularities and patterns within a shape collection.

Acknowledgements

The authors are grateful to Prof. Christos Faloutsos for valuable discussions on Fastmap, Pantelis Elinas for his help in the experiments, Zusheng Rao for valuable discussions and for a shape morphing program to generate the database. Marita Roussou and Panos Economopoulos designed and implemented C++ code of professional quality for the shape matching algorithm.

References

- [1] Roland T. Chin and Charles R. Dyer. Model-Based Recognition in Robot Vision. *ACM Computing Surveys*, 18(1):67–108, 1986.
- [2] Babu M. Mehtre, Mohan S. Kankanhalli, and Wing Foon Lee. Shape Measures for Content Based Image Retrieval: A Comparison. *Information Processing and Management*, 33(3):319–337, 1997.
- [3] M. De Marsicoi, L. Cinque, and S. Levialdi. Indexing Pictorial Documents by Their Content: A Survey of Current Techniques. *Image and Vision Computing*, 15(1):119–141, 1997.
- [4] Venkat N. Gudivada and Vijay V. Raghavan. Modeling and Retrieving Images by Content. *Information Processing and Management*, 33(4):427–452, 1997.
- [5] Ramesh Jain, Alex Pentlant, and Dragutin Petkovic. NSF-ARPA Workshop on Visual Information Management Systems, June 1995. <http://www.virage.com/vim/vimsreport95.html>.
- [6] E. Saund. Symbolic Construction of a 2D Scale-Space Image. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, pages 817–830, August 1990.
- [7] M. Leyton. A Process Grammar for Shape. *Artificial Intelligence*, 34:213–247, 1988.
- [8] W. Grimson. *Object Recognition by Computer: The Role of Geometric Constraints*. MIT Press, Cambridge, Massachusetts, 1990.
- [9] F. Stein and G. Medioni. Structured Indexing: Efficient 3-D object Recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14:125–145, 1992.
- [10] K. Siddiqi and B. Kimia. Parts of Visual Form: Computational Aspects. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 17(3):239–251, 1995.
- [11] A. Witkin. Scale Space Filtering. In *Proceedings 8th IJCAI, (Karlsruhe, West Germany)*, pages 1019–1022, 1983.
- [12] F. Mokhtarian. Silhouette-Based Object Recognition through Curvature Scale Space. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 17(5), May 1995.
- [13] F. Mokhtarian, S. Abbasi, and J. Kittler. Efficient curvature-based shape representation for similarity retrieval. In *European Signal Processing Conference (EUSIPCO)*, pages 597–600, 1998.
- [14] A. Witkin, D. Terzopoulos, and M. Kass. Signal Matching through Scale-Space. In *Proceedings of AAAI-86*, pages 714–719, 1986.

- [15] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *Int. J. Computer Vision*, 1(4):321–331, 1988.
- [16] B. Vemuri and R. Malladi. Constructing Intrinsic Parameters with Active Models for Invariant Surface Reconstruction. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 15(7), July 1993.
- [17] F. Leymarie and M. Levine. Tracking Deformable Objects in the Plane using an Active Contour Model. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 15(6), 1993.
- [18] D. Geiger, A. Gupta, L. Costa, and J. Vlontzos. Dynamic programming for detecting, tracking, and matching deformable contours. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 17(3) and 18(5):294–302, March and May 1995 and 1996.
- [19] L. Floreby. A multiscale algorithm for closed contour matching in image sequence. In *IEEE Int. Conf. on Pattern Recognition*, pages 884–888, 1996.
- [20] H. Tagare. Deformable 2D template matching using orthogonal curves. *IEEE Trans. on Medical Imaging*, 16(1):108–117, February 1997.
- [21] L. Shapiro. A Structural Model of Shape. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 2, March 1980.
- [22] D. Wuescher and K. Boyer. Robust Contour Decomposition using a Constant Curvature Criterion. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 13(1):41–51, 1991.
- [23] Naonori Ueda and Satoshi Suzuki. Learning Visual Models from Shape Contours Using Multiscale Convex/Concave Structure Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(4):337–352, April 1993.
- [24] J. Gorman, R. Mitchell, and F. Kuhl. Partial shape recognition using dynamic programming. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 10(2):257–266, March 1988.
- [25] E. Milios. Shape Matching using Curvature Processes. *Computer Vision, Graphics and Image Processing*, 47:203–226, 1989.
- [26] R. Mehrotra and W. Grosky. Shape matching utilizing indexed hypotheses generation and testing. *IEEE Trans. on Robotics and Automation*, 5(1):70–77, February 1989.
- [27] N. Ansari and E. Delp. Partial shape recognition: A landmark-based approach. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 12(5):470–483, May 1990.
- [28] Farzin Mokhtarian and Alan Mackworth. Scale-Based Description of Planar Curves and Two-Dimensional Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(1):34–43, 1986.
- [29] Farzin Mokhtarian, Sadegh Abbasi, and Josef Kittler. Robust and Efficient Shape Indexing through Curvature Scale Space. In *Proceedings of British Machine Vision Conference*, Edinburgh, UK, 1996.
- [30] Paul G. Gottschalk, Jerry L. Turney, and Trevor N. Mudge. Efficient Recognition of Partially Visible Objects Using a Logarithmic Complexity Matching Technique. *The International Journal of Robotics Research*, 8(6):110–131, December 1989.
- [31] Andrea Califano and Rakesh Mohan. Multidimensional Indexing for Recognizing Visual Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(4):373–392, 1994.

- [32] Jeffrey S. Beis and David G. Lowe. Shape Indexing using Nearest-Neighbour Search in High-Dimensional Spaces. In *Proceedings of International Conference on Computer Vision and Pattern Recognition*, 1997.
- [33] Alan Kalvin, Edith Schonberg, Jacob T. Schwartz, and Micha Sharir. Two-Dimensional, Model-Based, Boundary Matching Using Footprints. *The International Journal of Robotics Research*, 5(4):38–55, 1986.
- [34] Fridtjof Stein and Gerard Medioni. Efficient Two Dimensional Object Recognition. In *In Proceedings of 10th International Conference On Pattern Recognition*, pages 13–17, Atlantic City, New Jersey, June 1990.
- [35] B. Günsel and M. Tekalp. Shape similarity matching for query-by-example. *Pattern Recognition*, 31(7):931–944, July 1998.
- [36] Nick Roussopoulos, Steve Kelley, and F. Vincent. Nearest Neighbor Queries. *Proc. of ACM-SIGMOD*, pages 71–79, May 1995.
- [37] Thomas Brinkhoff, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. Multi-step processing of spatial joins. *ACM SIGMOD*, pages 197–208, May 1994.
- [38] K. Hinrichs and J. Nievergelt. The Grid-File: A Data Structure to Support Proximity Queries on Spatial Objects. Technical Report 54, Institut für Informatik, ETH, Zurich, July 1983.
- [39] Jack A. Orenstein. Spatial Query Processing in an Object Oriented Database System. In *ACM Proceedings SIGMOD 86*, pages 326–336, Washington, May 1986.
- [40] C. Faloutsos and S. Roseman. Fractals for Secondary Key Retrieval. *Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 247–252, March 1989. also available as UMIACS-TR-89-47 and CS-TR-2242.
- [41] J.L. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *CACM*, 18(9):509–517, September 1975.
- [42] A. Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of ACM SIGMOD*, pages 47–57, June 1984.
- [43] P. N. Yianilos. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, pages 311–321, 1993.
- [44] Sergey Brin. Near Neighbor Search in Large Metric Spaces. In *Proceedings of the 21st VLDB Conference*, pages 574–584, Zurich, Switzerland, 1995.
- [45] C. Faloutsos and K.-I. Lin. FastMap: A Fast Algorithm for Indexing, Data Mining and Visualization of Traditional and Multimedia Datasets. In *SIGMOD RECORD, Proceedings of the 95 ACM SIGMOD International Conference on Management of Data*, pages 163–174, San Jose, California, June 1995.
- [46] N. Roussopoulos and D. Leifker. Direct spatial search on pictorial databases using packed R-trees. *Proc. ACM SIGMOD*, May 1985.
- [47] Timos Sellis, Nick Roussopoulos, and Christos Faloutsos. The R⁺-tree: A Dynamic Index for Multidimensional Objects. In *Proceedings of 13th International Conference on VLDB*, pages 507–518, England, September 1987.
- [48] Nobert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of the 1990 ACM SIGMOD*, pages 322–331, Atlantic City, NJ, May 1990.

- [49] Ibrahim Kamel and Christos Faloutsos. Hilbert R-tree: An Improved R-tree Using Fractals. In *Proceedings of VLDB Conference*, pages 500–509, Santiago, Chile, September 1994.
- [50] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-tree : An Index Structure for High-Dimensional Data. In *Proceedings of the 22rd VLDB Conference*, pages 28–39, 1996.
- [51] Norio Katayama and Shinichi Satah. The SR-Tree: An Index Structure for High Dimensional Nearest Neighbor Queries. In *Proceedings of ACM SIGMOD*, pages 369–380, Tuscon, Arizona, May 1997.
- [52] J. Baid and E. Milios. Deformed Shape Recognition using Dynamic Programming. Technical report, Department of Computer Science, York University, 1995.
- [53] Thomas W. Sederberg and Eugene Greenwood. A Physical Based Approach to 2-D Shape Bending. *Computer Graphics*, 26(2):25–34, 1992.
- [54] Timothy P. Wallace and Paul A. Wintz. An Efficient Three-Dimensional Aircraft Recognition Algorithm Using Normalized Fourier Descriptors. *Computer Graphics and Image Processing*, 13:99–126, 1980.
- [55] L. Gurta and M. D. Srinath. Contour Sequence Moments for the Classification of Closed Planar Shapes. *PR*, 20(3):267–271, 1987.
- [56] Ming-Kuei Hu. Visual Pattern Recognition by Moment Invariants. *IRE Transactions in Information Theory*, IT-8:179–187, 1962.
- [57] D.K. Harman, editor. *The Second Text Retrieval Conference*. National Institute of Standards and Technology Gaithersburg, MD, 1994.
- [58] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules in Large Databases. *Proc. of VLDB Conf.*, pages 487–499, September 1994.

Contents

1	Introduction	1
2	Survey of Related Work	4
2.1	Shape Matching	4
2.2	Spatial Access Methods (SAMs)	6
2.3	R-trees	7
2.4	FastMap	7
3	Shape Matching	8
3.1	Definitions	8
3.2	The algorithm	11
3.3	Cost Components	12
3.4	The Ueda-Suzuki Algorithm	15
3.5	Experiments with Matching	16
4	Shape Retrieval	17
4.1	Shape Indexing	18
4.2	Search Strategy	19
5	Experimental Results	19
5.1	Comparison With Other Methods	20
5.2	Response Time	21
5.3	False Dismissals	22
5.4	Clustering	22
6	Conclusions	23

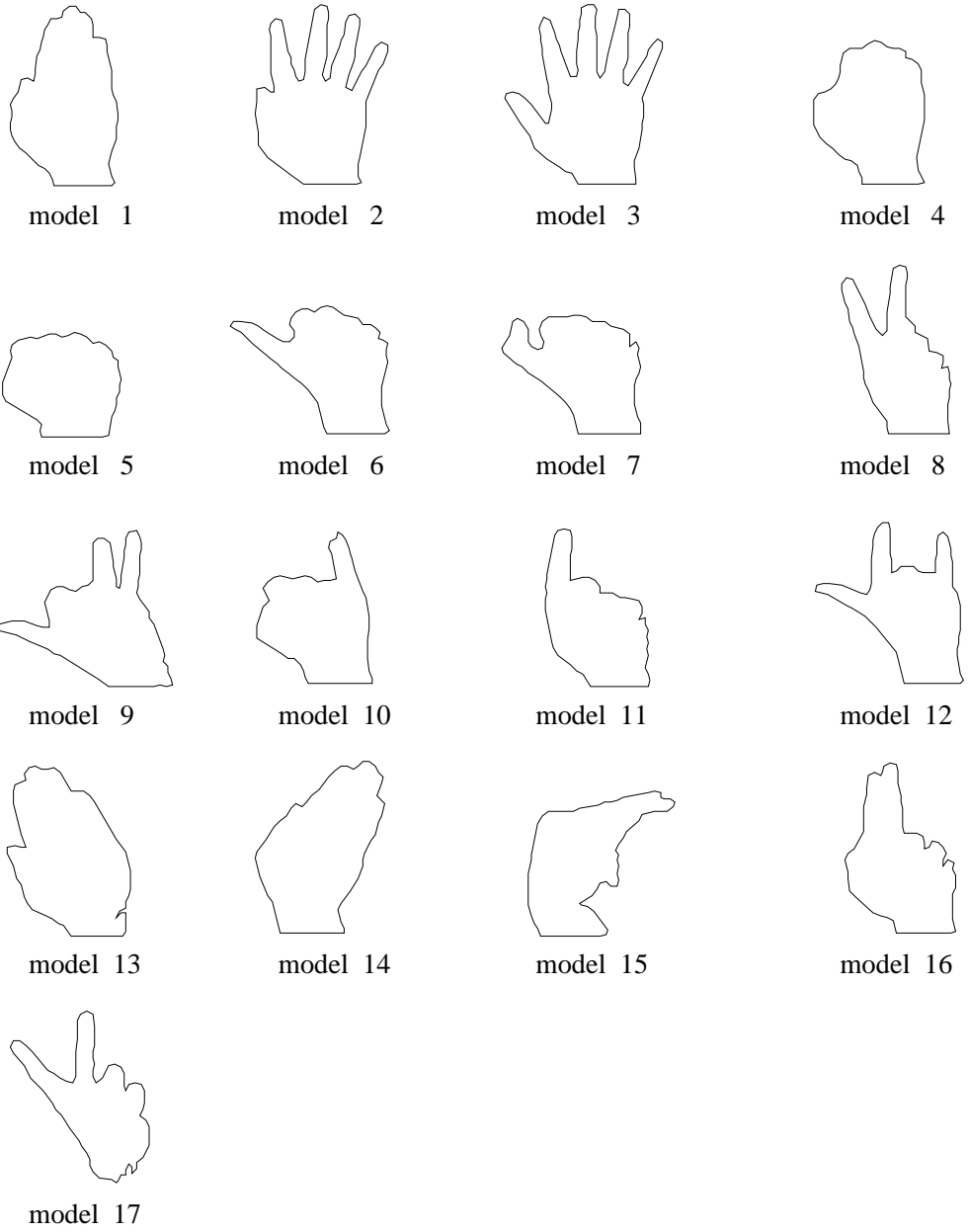


Figure 6: *Model database.*

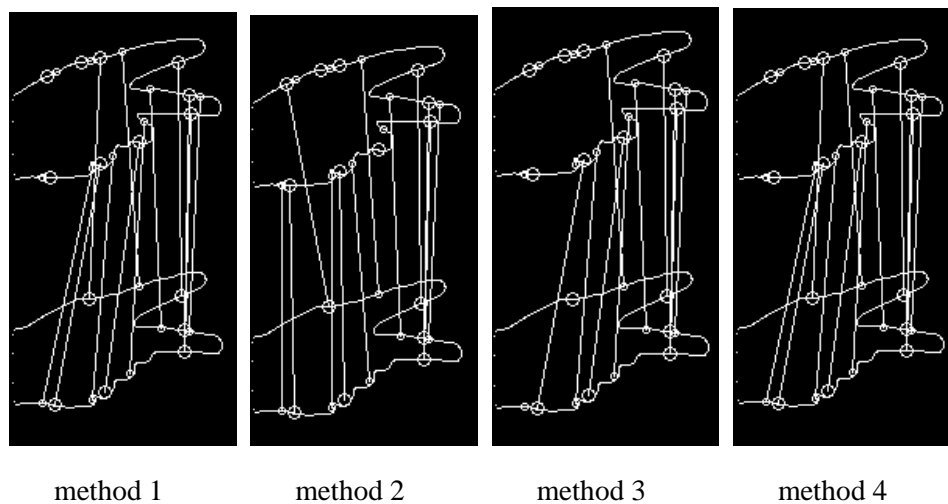


Figure 7: Matching results illustrating the segment associations reported by the matching algorithm for query 2 corresponding to four different settings.

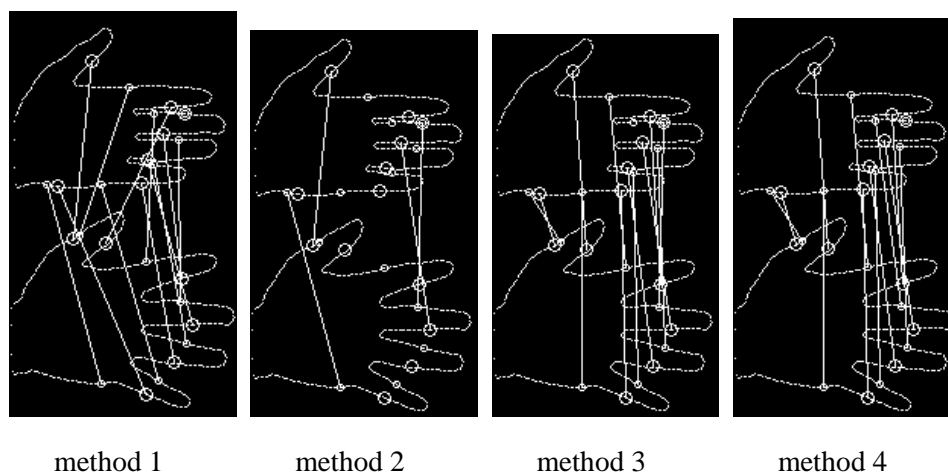


Figure 8: Matching results illustrating the segment associations reported by the matching algorithm for query 1 corresponding to four different settings.

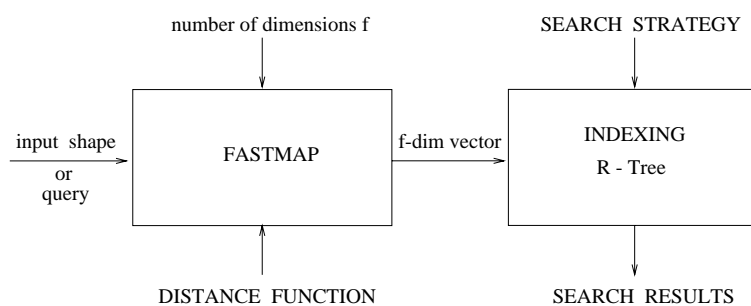


Figure 9: Mapping of shapes to f -dimensional points.

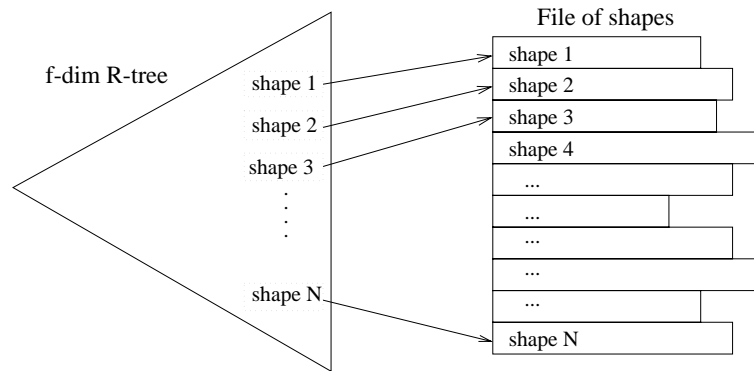


Figure 10: File structure.

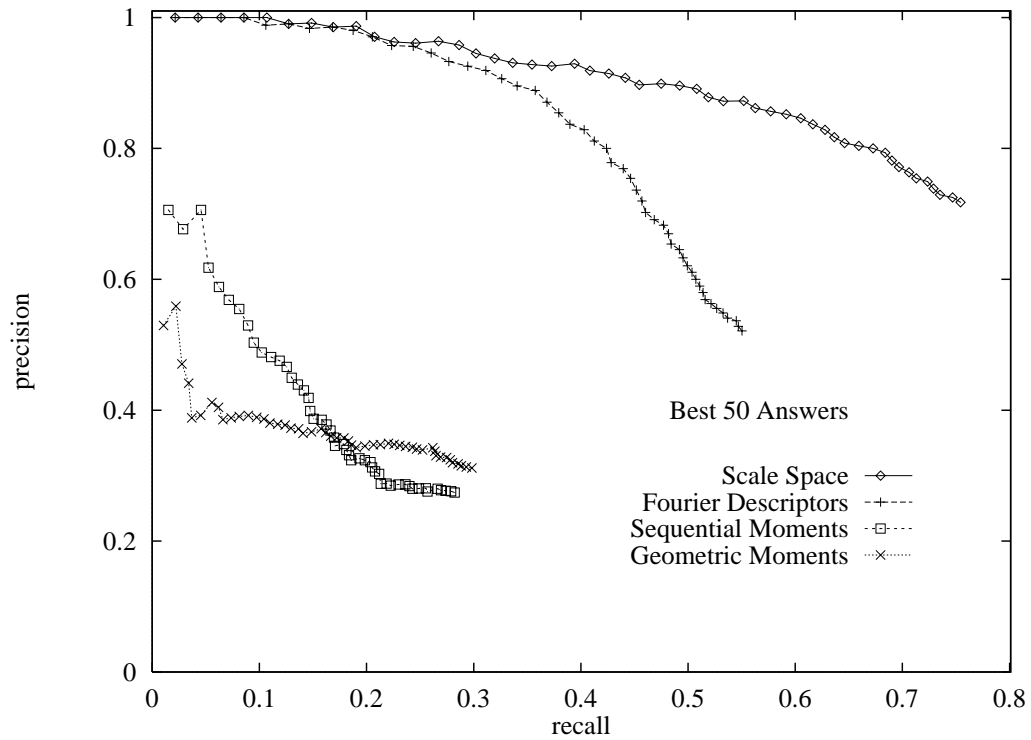


Figure 11: Precision-Recall diagram for (a) Our shape matching algorithm, (b) Fourier descriptors, (c) Sequential moments and (d) Geometric moments.

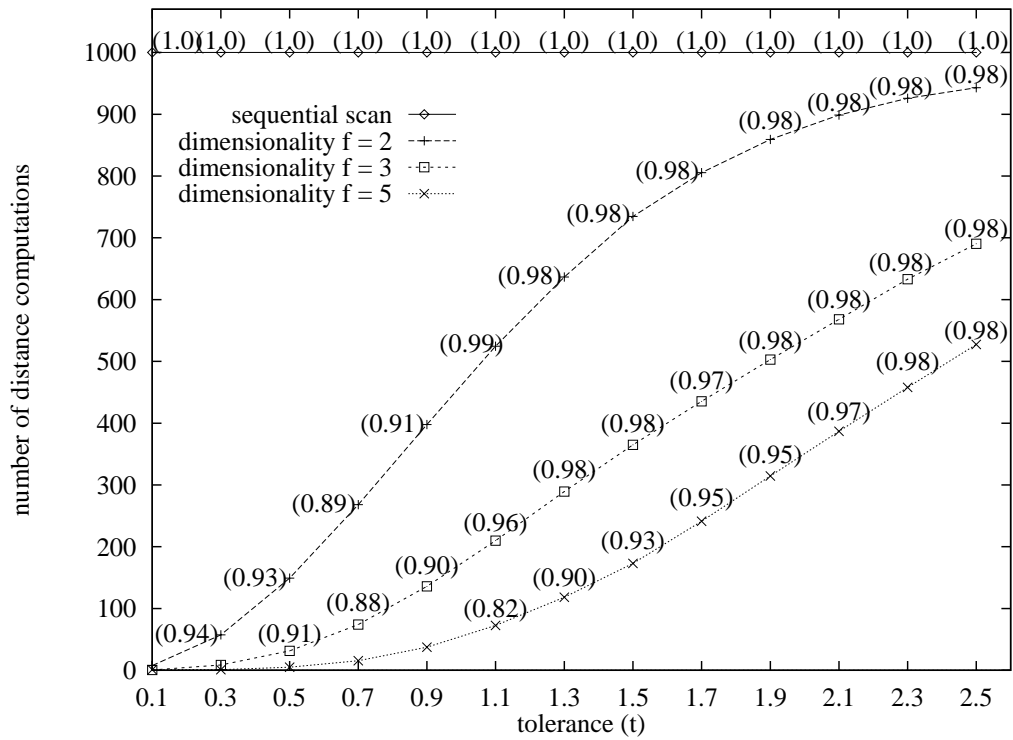


Figure 12: Average number of distance computations as a function of the tolerance t , corresponding to sequential scanning and dimensionality $f = 2, 3, 5, 7$ and 10 . The labels in parentheses denote average values of accuracy.

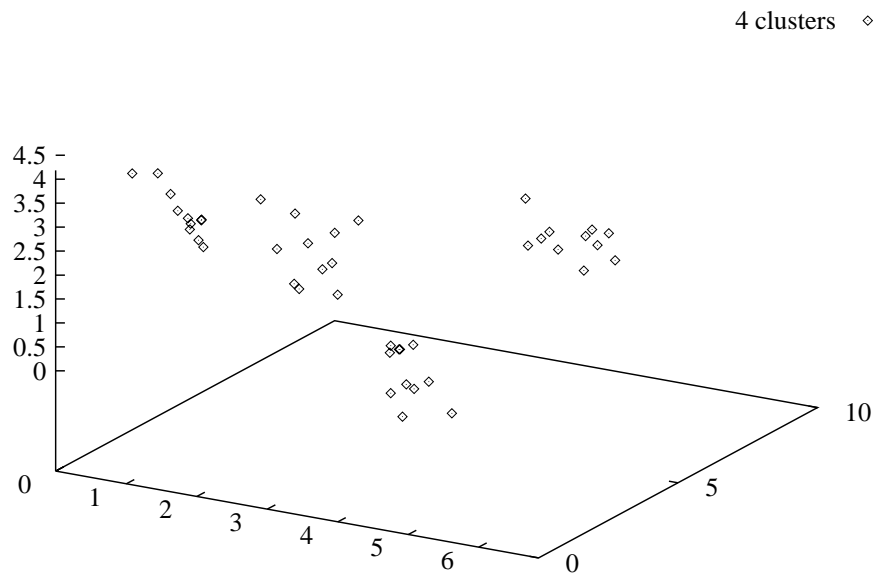


Figure 13: 3-dimensional *FastMap* diagram of 44 hand gestures belonging to 4 classes.