High-Dimensional Descriptor Indexing for Large Multimedia Databases

Eduardo Valle
ETIS UMR CNRS 8051
6, ave du Ponceau BP 44F
95014 Cergy-Pontoise cedex France
+33 1 30 73 66 10
mail@eduardovalle.com

Matthieu Cord
LIP6 — UPMC
104, ave du Président Kennedy
75016 Paris
+33 1 44 27 71 39
matthieu.cord@lip6.fr

Sylvie Philipp-Foliguet
ETIS UMR CNRS 8051
6, ave du Ponceau BP 44F
95014 Cergy-Pontoise cedex France
+33 1 30 73 66 10
sylvie.philipp@ensea.fr

ABSTRACT

In this paper we address the subject of large multimedia database indexing for content-based retrieval.

We introduce **multicurves**, a new scheme for indexing high-dimensional descriptors. This technique, based on the simultaneous use of moderate-dimensional space-filling curves, has as main advantages the ability to handle high-dimensional data (100 dimensions and over), to allow the easy maintenance of the indexes (inclusion and deletion of data), and to adapt well to secondary storage, thus providing scalability to huge databases (millions, or even thousands of millions of descriptors).

We use multicurves to perform the approximate k nearest neighbors search with a very good compromise between precision and speed. The evaluation of multicurves, carried out on large databases, demonstrates that the strategy compares well to other up-to-date k nearest neighbor search strategies.

We also test multicurves on the real-world application of image identification for cultural institutions. In this application, which requires the fast search of a large amount of local descriptors, multicurves allows a dramatic speed-up in comparison to the brute-force strategy of sequential search, without any noticeable precision loss.

Categories and Subject Descriptors

H.2.2 [**Database management**]: Physical design – *access methods.* H.2.4 [**Database management**]: Systems – *multimedia databases.* H.3.1 [**Information storage and retrieval**]: Content analysis and indexing – *indexing methods.*

General Terms

Algorithms, Measurement, Performance, Design, Experimentation.

Keywords

High-dimensional data indexing, k nearest neighbors search, local descriptors, image retrieval, document identification, copy detection, near-duplicate detection.

1. INTRODUCTION

The retrieval of multimedia data often carries two intrinsic challenges: the sheer amount of information and the complexity of this information. In contrast with relational or textual data, the relationship between low-level representation of multimedia and its high-level semantic content is much less direct. Therefore, retrieval must be mediated by the use of *descriptors*, which are usually very high-dimensional and often proliferate at the rate of hundreds per described item.

From both the theoretical and the technical standpoints, the indexing of those descriptors is a very difficult problem. From the theoretical point of view, the well-known "curse of dimensionality" makes the indexing inefficient as dimensionality grows [1][2]. From the technical point of view, the memory hierarchy, the workings of disks and caches, and the integration with DBMS create additional constraints which must be addressed if the scheme is to have any practical usefulness.

In this paper we describe *multicurves*, a very efficient index for high-dimensional data, which addresses the need of multimedia descriptor indexing, in the context of very large databases. Our scheme is based on space-filling curves — a technique which has attracted considerable attention on the field for its ability of creating a "vicinity-sensitive" total order on the data, and thus allowing the adaptation of the efficient and convenient one-dimensional indexing techniques, like the B-Tree, to multidimensional data. The originality of our method is to integrate these curves into a very effective structure, where, through a smart combination of multiple, moderate-dimensional curves, boundary effects are adequately addressed and precision is considerably improved, without losing the benefits of scalability and ease of index maintenance.

We introduce the problem of multimedia descriptor indexing using kNN search on § 2. On § 2.2 we specifically discuss the methods which use space-filling curves. The detailed description of our method, multicurves, follows, on § 3. On § 3.3 we discuss some important considerations of parameterization and integration with DBMS. On § 4, we compare multicurves with two other methods based on space-filling-curves from the state-of-the-art, showing that multicurves presents the best compromise between precision and speed. On § 5 we show multicurves in action. In the context of an application of image retrieval, it allows a dramatically accelerating of the results, without any noticeable performance losses.

2. MULTIMEDIA DATA INDEXING

As we have suggested in the introduction, in what concerns information retrieval, a crucial difference between multimedia data and relational / textual data is that the low-level representation has a very indirect relationship with the semantic content.

Because of that, indexing and retrieval of those data is mediated by *descriptors*, i.e., more compact and semantically-rich representation of the data. Those descriptors appear in a large variety of forms: color and texture histograms, invariant moments, Fourier coefficients, local jets, gradient maps, etc. They often appear in the form of vectors, which are frequently high-dimensional.

To establish the similarity / dissimilarity between documents, their descriptors are compared. Therefore, to answer the user queries, a descriptor matching scheme has to be established, and this is usually done by kNN search (also known as *k nearest neighbors search* or *similarity search*).

2.1 The kNN search

kNN search can be understood as the operation of finding, on a given space, the *k* points which are the nearest to the query point. For a more formal definition, consider that we have:

- a *d*-dimensional domain space *D*;
- a base set *B* with the elements $b_{1,}, b_{2,}, \cdots, b_{n,} \in D$;
- a query $q \in D$;
- and a dissimilarity function $\Delta: D \times D \to \mathbb{R}$;
- the kNN search for the k elements most similar to q consists in finding a set $S = \{s_{1,}, s_{2,}, \cdots, s_{k,}\}$ where

$$\forall s_i \in S \;\; \forall b_j \in B - S, \quad \Delta(q, s_i) \leq \Delta(q, b_j)$$

An obvious solution is the sequential search, where each element of the base is compared to the query, and the *k* most similar are kept. Unfortunately, this brute-force solution is acceptable only for small bases, being unfeasible in our context. The alternative is using multidimensional indexing schemes, which are able to accelerate the search.

However, the efficiency of those schemes depends greatly on the dimensionality of the elements. The search time can be made to grow only logarithmically to the size of the base, but at the expense of introducing a hidden constant which will grow exponentially with the dimensionality. Typically, for over 10 dimensions, it will be impossible to perform *exact* kNN search faster than the sequential method. This phenomenon is known as "curse of dimensionality" and expresses the difficulty in partitioning the data or the space in an efficient way when dimensionality is very high [1][2].

Multimedia descriptors at dimensionalities around 30 are commonplace, and those at around 100 are not unusual. Dimensionalities of 400 and above are not unheard of. For spaces like those, kNN search is extremely challenging.

kNN search is an important topic of research, which finds many applications in addition to descriptor matching, and not surprisingly there is a vast literature about the subject. For a comprehensive introduction and state of art on the subject the reader is referred to [3][4]. Here, however, we are only interested in those methods of practical interest in our specific context. The method should:

- perform well for high-dimensional data, presenting a good compromise between precision and speed;
- adapt well to secondary-memory storage, which in practice means that few random accesses should be performed;
- ideally, the index generated should by dynamic, i.e., allow data insertion and deletion without subsequent performance degradation.

Surprisingly few methods are able to accomplish those requirements: many methods assume implementation in main memory (and thus, cheap random access throughout the index), other methods have prohibitively high index building times (with a forced rebuilding if the index changes too much), and so on.

LSH, or locality-sensitive hashing, uses locality-preserving hashing functions to index the data. The method uses several of those "hash tables" at once, to improve reliability [5].

Another interesting method is MEDRANK, which projects the data into several random straight lines. The one-dimensional position in the line is used to index the data [6].

The PvS is an improvement on MEDRANK, which uses segmentation and re-projection in random straight lines. Points are grouped together only if they are simultaneously near in multiple straight lines [7].

A common pattern of all those methods is the use of multiple subindexes. Conceptually what happens is that a sub-query is performed on each of those subindexes, and the method, then aggregates the results. Intuitively, the idea is to increase the chances of a well-succeeded search by using several parallel structures.

LSH is backed by a solid theoretical background, which allows predicting the approximation bounds for the index, for a given set of parameters. Setting those parameters however is not easy, as there are several of them, some of which are quite critical (like the radius of analysis, which sets a bound beyond which we allow ourselves to ignore the points). Also, the number of hash tables necessary to have good performance is high (30 or more), which represents a cost in terms of storage and of random accesses performed during the search.

MEDRANK is also backed by a theoretical background, but the guaranteed bounds are much looser. The greatest advantage of the method is the fact the data are projected into one-dimensional lines, and can then be managed by very efficient structures, like B-Trees. In empiric tests, however, the method has failed to index multimedia descriptors efficiently, mostly due to the lack of correlation between the distance in the straight lines and the distance in the high-dimensional space [7].

Trying to overcome the troubles of MEDRANK, PvS projects the data onto one straight line, segments the data and then reprojects them into another straight line. The process can be repeated several times, until segments of appropriate size are obtained. Points will fall in the same segment only if they are near in all lines, simultaneously. The major difficulty with PvS is the need of redundant coverage, to avoid boundary effects during segmentation. This makes the index doubly redundant (the points are present in each subindex, but also may be replicated unpredictably within the segments of a subindex), making the maintenance of the index rather complex.

We became acquainted with MEDRANK and seduced by the possibility of simply using a sorted list in the subindexes, since this allows us to create a fast, dynamic and easy to implement index. At the same time, we wanted to use multiple dimensions in the subspace represented in the subindexes, to minimize the distortion of the distance function. The solution to conciliate those two objectives was given by the use of space-filling curves.

2.2 kNN search with space-filling curves

Fractal geometry presents many unexpected results: the concept of "fractional" dimensionality, the existence of figures of finite area but infinite perimeter, and the possibility of relatively simple rules generating very complex ("chaotic") sets. Not less surprising is the existence of continuous mappings from the unit interval [0; 1] and any unit hypercube $[0; 1]^d$. Simply stated, this shows that there exist continuous curves which completely cover any *d*-dimensional hypercube. Those curves are called *space-filling curves* (Figure 1).

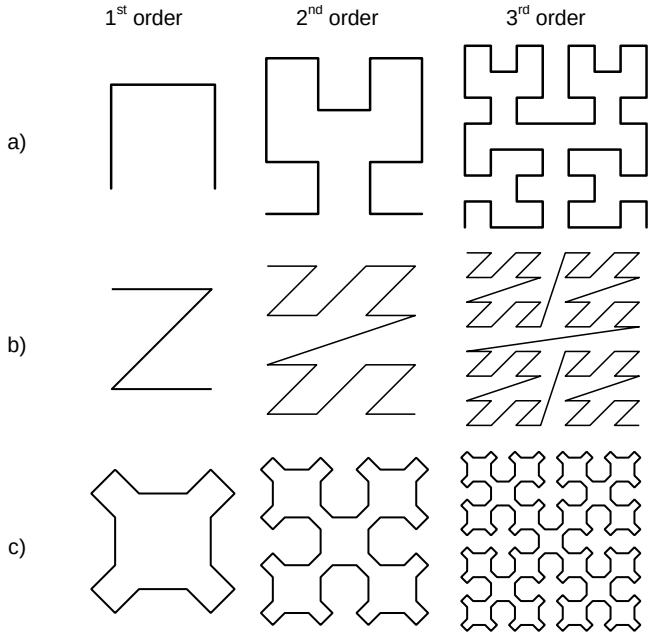

Figure 1: A few examples of 2D space-filling curves. a) Hilbert; b) Lebesgue or "Z-order"; c) Sierpiński. Three iterations of each curve are shown: the actual space-filling curve is the infinite limit of those iterations

Most space-filling are constructed by a recursive procedure, in which the space is progressively divided in smaller cells, which are then traversed by the curve. In this case, we say that the *order of the curve* is the number of iterations which have been carried out in the infinite series of a space-filling curve, i.e., the degree of refinement obtained so far. The space is filled by the curve to which the series converges. A curve of a given *finite* order is said to be an *approximation* of the named space-filling curve.

For an in-depth introduction to the subject of the space-filling curves, their mathematical properties and the proofs of the most important theorems, the reader is referred to [8].

The use of space-filling curves to perform the kNN search in multidimensional spaces is not new. Apparently Faloutsos was the first author to explicitly refer to the concept of space-filling curves [9], though implicit use of the Z-order curve (through the notion of "bit-interleaving") can be traced back to 1966 [10]. Faloutsos was also the first to suggest that other curves could perform better than the Z-order, first proposing the Gray-code curve and then the Hilbert curve [9][11]. Skubalska-Rafajłowicz and Krzyżak make a seemingly independent development of the use of the curves to perform kNN, using the generalized form of the Sierpiński curve [12].

All those methods are conceptually very simple. They map the high-dimensional points in the curve, obtaining a one-dimensional coordinate, called **extended-key**, which represents its relative position in the length of the curve. The extended-key is used to perform a one-dimensional similarity search, which is straightforward. The hypothesis is that points which are near to each other in the curve always correspond to points that are near to each other in the space, so it is a good heuristic to take the nearest points in the curve as the nearest points in the space.

Unfortunately, the converse of the hypothesis is not true, in the sense that near points in the space are not always near in the curve. This is because of the boundary effects, which tend to put very far apart points in certain regions of the curve (Figure 2). The matter is seriously aggravated as the dimensionality increases.

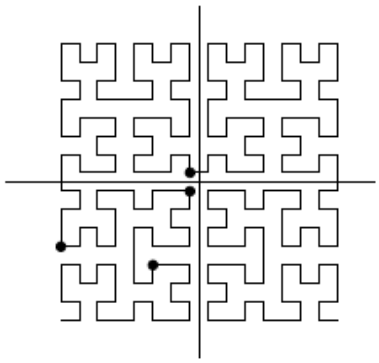

Figure 2: The problem with boundary effects on the space-filling curves. The points in the centre of the space are further apart in the curve than the points in the lower-left quadrant

To conquer the boundary effects, different authors propose different solutions, usually through the use of multiple curves [13][14][15]. A notably different approach is suggested by Mainar-Ruiz and Pérez-Corés, suggesting using multiple (slightly disturbed) instances of the same point in only one curve [16].

3. THE MULTICURVES

As we have seen, the greatest problem of the use of space-filling curves comes from the boundary effects, and different methods propose different solutions, usually through the simultaneous use of multiple curves.

Our method, **multicurves**, is also based on the use of multiple curves, but with the important distinction that each curve is only responsible for a subset of the dimensions. So not only we gain the intrinsic advantages of using multiple curves (i.e., points that are incorrectly separated in one curve will probably stay together in another), but also, we lower the boundary effects inside each one of the curves.

Multicurves has a simple structure: divide the dimensions of the data elements among a certain number of space-filling curves. On each curve map the corresponding projections of the data, compute their extended-keys, and put each pair <extended-key, element> in a list sorted by extended-key. Each one of those sorted lists is a **subindex** (Figure 3).

The search is done the same way: the query is decomposed into several projections (corresponding to the dimensions associated to each subindex) and each projection has its extended-key computed. Then, for each subindex, we explore the elements whose

extended-keys are the nearest to the corresponding query extended-key.

Any recursive space-filling curve can theoretically be used, but in comparison with other space-filling curves, the Z-order curve or the Gray-code curve, the Hilbert curve has better clustering properties, which is important for the kNN search and other applications (see [17]). Other important properties of the curve, which resulted in its widespread application, are its "fairness" (symmetric behavior on all dimensions) and the absence of distant jumps [18].

Before we present the construction and search algorithms, let us make a few definitions:

- *d*: the dimensionality of the data elements;
- *m*: the number of bits needed to represent each dimension of the data elements, which will correspond to the order of the Hilbert curve to be generated;
- *c*: the number of curves to build;
- d_i: the dimensionality of the i^{th} curve;
- *A*: an association between the dimensions of the data and dimensions of the curves (as illustrated in Figure 4).

3.1 Index Construction

The construction algorithm for multicurves (shown in Figure 5) is relatively simple, since the underlying structure of the index is just a set of sorted lists, one for each Hilbert curve generated. Each data element is decomposed into *c* projections, accordingly to the association of dimensions *A*. Those projections are used to compute the extended-key in each curve. The pairs <extended-key, data element> are then inserted in the curves.

points is the list of data elements to build the index
point[i] is the value of the i^{th} dimension of point
d[i] is the dimensionality of the i^{th} curve (d_i)
A[][] is an association between the dimensions of the data space and the subindexes space such that A[i][j] indicates the j^{th} dimension (from the data space) that should be attributed to the i^{th} subindex, i.e., if M[2][3] = 10, it means that the 3^{rd} dimension of the 2^{nd} subindex is in fact the 10^{th} dimension of the data space
curves[] is an array of c sorted lists
projection[] is the projection of point[] in the subspace of the curve

```
BuildMulticurves(c, d[], A[], m, points)
  → Returns array of sorted lists
1.  For curve ← 1 to c do
2.      curves[curve] ←
            new empty sorted list of pairs <extended_key, point>
            sorted by extended_key
3.  Next
4.  For each point in points do
5.      For curve ← 1 to c do
6.          projection[] ← new array with d[curve] elements
7.          For dimension ← 1 to d[curve] do
8.              projection[dimension] ←
                    point[A[curve][dimension]]
9.          Next
10.         extended_key ←
            HilbertExtendedKey(d[curve], m, projection)
11.         Put the pair <extended_key, point> into
        the list curves[curve]
12.     Next
13. Next
14. Return curves[]
```

Figure 5: The construction algorithm for multicurves

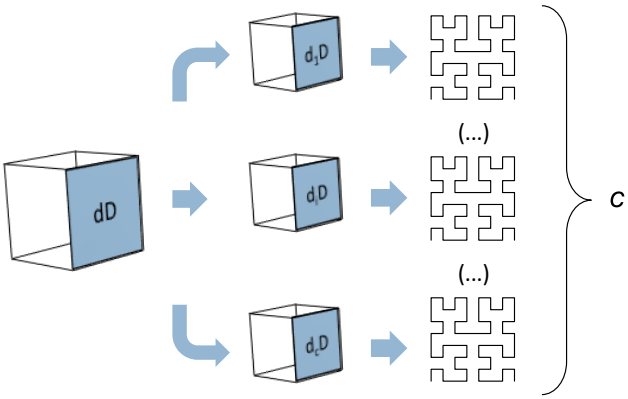

Figure 3: The base structure of multicurves: using simultaneously several moderately-dimensional space-filling curves

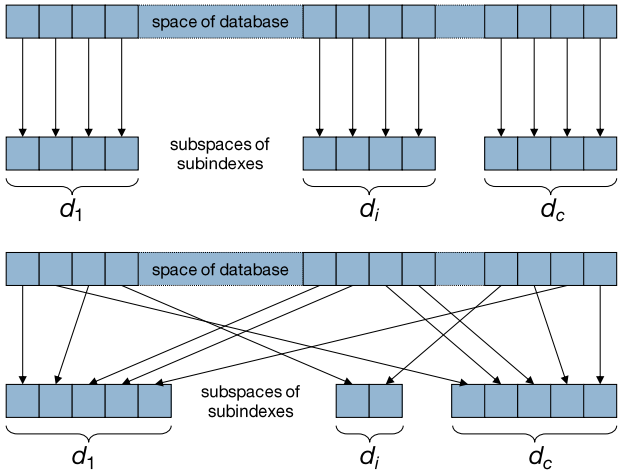

Figure 4: Two sample parameterizations of the multicurves showing different choices for the parameters *c*, d_i, and the association *A*, leading either to a regular scheme (a), or not (b).

The complexity of the construction depends on the underlying structure used to implement the sorted lists. The computation of the projections of each element in steps 7–9 takes, at worst, $O(d)$ operations. The computation of the extended-key using the Butz algorithm on line 10 makes $O(md)$ bit operations. Assuming an efficient sorted list structure, insertions will take $O(\log n)$ steps. Since we are building c lists, algorithm should take at worst, $O(c(nmd + n\log n))$ steps.

3.2 Search

The search algorithm is also simple (Figure 6). We choose, beforehand, the number of elements to examine in each subindex. Then we project the query onto the same *c* subspaces used to build the index. We find, on each subindex, the elements nearest to the corresponding projection, and keep the *k* nearest to the query.

The complexity analysis of the search comparison is easy: the construction of the extended-key using the Butz algorithm makes $O(md)$ bit operations. The time spent looking for the values nearest to the extended-key depends on the underlying data structure, but generally it can be assumed to take at most $O(\log\ n)$ steps. This step is prone to be expensive, for here we are

forced to make at least one random access to the data. The complexity of steps 9–15 is known beforehand and grows linearly with the number of elements to be examined. The expensive operation here is the computation of the distances, which takes $O(d)$ arithmetic operations, for the p-norm distances (like the Euclidian distance).

The dimensionality has also a "hidden" influence, in that it increases linearly the amount of data which must be transferred by the algorithm. This is non-negligible when we use secondary storage.

The whole operation (steps 2–17) is repeated once for every subindex, which means a linear growth with this parameter.

In summary, the time spent on the search grows linearly with the number of points to be examined in each index, the number of indexes and the number of dimensions in the data. It also grows logarithmically on the number of elements in the database.

```
depth is the number of data items to examine in each curve
query[i] is the value of the ith dimension of query

SearchMulticurvesDist(c, d[], A[], m, curves[], depth, k, query)
  → Returns a list of k nearest neighbours

1.    best ← new empy sorted list of pairs <distance, point>
          sorted by distance
2.    For curve ← 1 to c do
3.        projection[] ←new array with d[curve] elements
4.        For dimension ← 1 to d[curve] do
5.            projection[dimension] ← query[A[curve][dimension]]
6.        Next
7.        extended_key ←
          HilbertExtendedKey(d[curve], m, projection)
8.        candidates ←
          list with the depth points nearest to extended_key in
          the sorted list curves[Curve]
9.        For each candidate in candidates do
10.           distance ← distance from candidate to query
11.           If distance < last distance in best then
12.               Put pair <distance, candidate> in best
13.           If best has more than k entries then
14.               Remove last entry in best
15.       Next
16.   Next
17    Return best
```

Figure 6: The search algorithm for multicurves

3.3 Discussion

In order to use the Hilbert curve in any intended application, it is necessary to compute a mapping between the coordinates in the hyperdimensional space and the one-dimensional ordinal position in the curve (i.e., the *extended-key*). There is an algorithm which uses little memory, is iterative, and can map any curve using only $O(md)$ bit operations [19].

Any adequate data structure can be used to store the sorted lists, the choice being based on practical considerations. Normally some flavor of B-tree [20] should be the best solution for most database applications. In our test implementations, for the sake of simplicity, we have chosen a two-level indexing, with the first level fitting entirely in main memory.

The most important parameters for the construction of the multicurves are the number of subindexes and the association between the dimensions. We propose to set as the highest value which efficiency will allow (which cannot be much higher than 10, anyway, since each additional index means a new random access to the data). Currently, the association of dimensions we use is a simple balanced partition among the curves.

The parameterization for the search is simple: only one parameter must be decided which is the number of elements to examine in each subindex.

Those parameters (number of subindexes and number of elements to examine) are a subject of empirical evaluation in the next section.

Since we have established that it is critical to limit the number of random accesses, it is absolutely critical that the subindexes store copies of the descriptors and not pointers to the descriptor information. The reason for this is clear: if the subindex has just a pointer to the data (as indexes usually do), when we arrive at the loop 9–15 of the algorithm in Figure 6, we will have to make one additional random access for each candidate, which is unacceptable.

This of course, is not a specific problem of multicurves: most multimedia indexing methods face this implementation challenge: the use of pointers to the data, which in the majority of cases works perfectly for relational data, incurs into unacceptable high costs for multimedia descriptors.

Replication of data, however, introduces the opportunity of inconsistencies and other anomalies. Besides the logical database design, must remain free of those implementation considerations. The best solution, therefore, is to delegate the management of the replication of the data to the physical design of the database, and let the DBMS to take care of the details.

4. EXPERIMENTAL RESULTS

4.1 Experimental Setup

For the evaluation of our system, we have used two image databases.

The **APM** database is composed by SIFT descriptors [21] generated from image transformations of a selection of 100 original images. The images are old photographs (XIX and first half of XX centuries) from the collection of the Arquivo Público Mineiro, the State Archives of the Brazilian State of Minas Gerais. Each image suffered three rotations, four scale changes, four non-linear photometric changes (gamma corrections), two smoothings and two shearings — a total of 15 transformations. Each transformed image had its SIFT descriptors calculated and aggregate into a database of 2 871 300 descriptors. The queries are the SIFT descriptors calculated from the original images, amounting to 263 968 descriptors. The dimensionality of SIFT descriptors is 128.

The **Yorck** database is composed of SIFT descriptors generated from the images of the Yorck Project, a collection of images of works of art. The images were downloaded from the Wikimedia Commons website, reduced to a smaller resolution, and had their SIFT points computed and put on a database — which amounts to 21 591 483 descriptors. The queries are the SIFT descriptors calculated from images of the Yorck Project selected at random and then transformed. One hundred images were selected, of which, 20 were rotated, 20 were resized, 20 suffered a gamma correction, 20 were sheared and 20 suffered a dithering (halftoning) — summing up to 166 315 query descriptors.

The ground truth is the set of the correct nearest neighbors for all query descriptors, according to the Euclidian distance. It was computed using the sequential search, a slow method, but which guarantees exact results. However, not all nearest descriptors are

equally useful for image identification. In fact, the match can only be considered correct, when it corresponds to a real match in the image. To take this into account we eliminate from the ground truth all answers which do not correspond to true matches. This is done by taking in turn every pair of correct query-target images (which are known beforehand), matching the descriptors and keeping only those which are geometrically consistent (using a RANSAC model fitting). After all pairs are processed, the only matches remaining are at the same time discriminating (from the point of view of the descriptor vector) and correct (from the point of view of the visual features).

Performance is measured in two axes: efficacy (the capability of the method to return the correct results) and efficiency (the capability of the method to use as little resources as possible).

To measure the efficacy we use the MAP (mean average precision), a classic efficacy metric of information retrieval, which is defined as "the area underneath a non-interpolated recall-precision curve" [22]. It can be considered a condensed single-value summary of the precision at all recall values.

From the point of view of the user, the most critical efficiency metric is the wall time spent on the search, but using it to compare the methods can be misleading, since it depends heavily on the machine, the operating system, and even on the current load (concurrent tasks) at the time the experiment is performed. We choose, therefore, to compare the methods by counting, for each method, how many target descriptors were accessed per query descriptor.

As we have mentioned in § 3.3, an important parameter for multicurves is the number of subindexes. This is also important for the methods of Mainar-Ruiz et al. and Liao et al. It is important to keep in mind that each new subindex represents additional costs in terms of construction time and disk space occupied. More critical, however: each subindex represents more descriptors acessed and more random access to the data (for multicurves and for Liao et al.), which we want to avoid. Therefore, we want to keep this number reasonably low.

We did not feel it was useful or informative to cover all the parameter scale for all methods — instead, we covered the range adequate to make the general tendencies unequivocal, in order to allow a fair comparison.

4.2 Methods Implemented

We have implemented and tested three methods: our multicurves, the methods by Liao et al. [15] and the one by Mainar-Ruiz and Pérez-Cortés [16]. Those are methods from the recent state-of-the-art in kNN search, which also employ space-filling-curves.

Like us, Liao et al. propose solving the problem of boundary effects on the Hilbert space-filling curves by using several curves at once. But in their work, all curves map the entire dimensionality of the data — the difference between the subindexes is a translation of the data.

Mainar-Ruiz and Pérez-Cortés suggest, instead, using only one curve but multiple representants of the data. Before inserting those representants in the curve, the algorithm disturbs their position, to give them the opportunity of falling into different regions of the curve. In that way, even if the query falls into a problematic region, chances are it will be reasonably near to at least one of the representants. The advantage of this method is that the index is composed by a single sorted list and, therefore, the number of random accesses necessary to perform the search is reduced to the bare minimum. Our reimplementation of this method had a slight

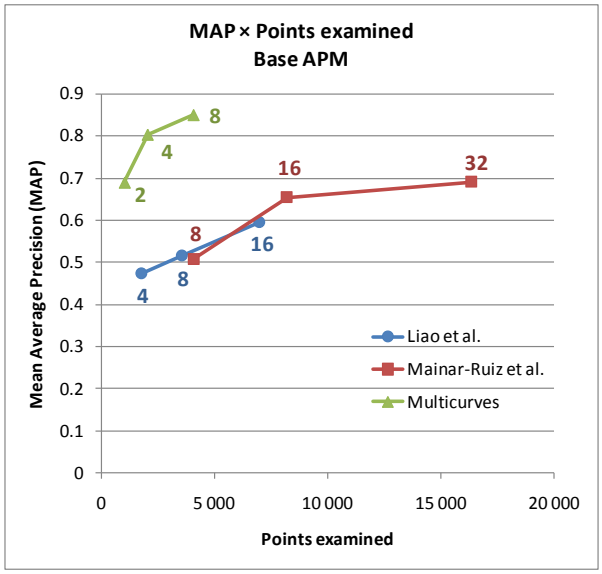

Figure 7: Performance comparison (precision × speed) for the methods on the APM database. The best compromise is towards the upper left corner. The numbers next to the points represent the number of subindexes per index

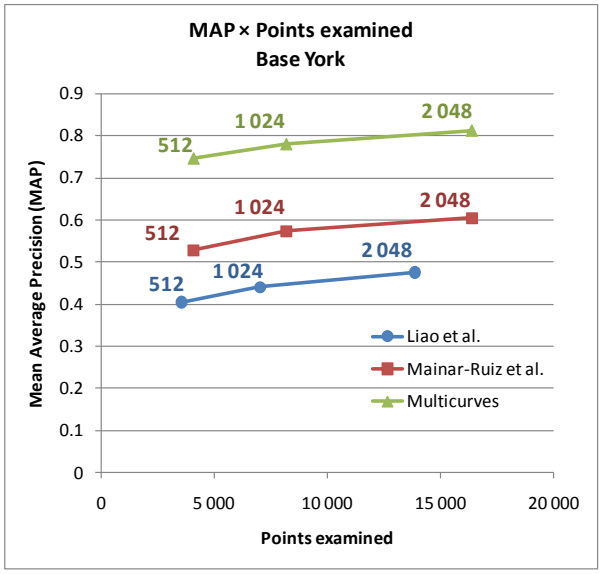

Figure 8: Performance comparison (precision × speed) for the methods on the Yorck database. The best compromise is towards the upper left corner. The numbers next to the points represent the amount of points examined per subindex

adaptation: while originally a generalized Sierpiński curve is used, we use the Hilbert curve (both curves are very similar in terms of their boundary domains).

All methods were implemented in Java, using the Java Platform, Standard Edition v. 1.6.

4.3 Results

Figure 7 shows the performance comparison between the three methods in the APM database. In the horizontal axis, representing

efficiency, we plot the number of points examined per query. In the vertical axis, representing efficacy, we plot the MAP obtained. The best compromise is towards the left upper corner.

To check how the methods depend on the number of subindexes, we vary this parameter (indicated next to each data point). In Mainar-Ruiz et al. there is, of course, only one sorted list, and this number corresponds to the number of representants given to each original point.

By a large margin, multicurves presented a better compromise for all interval of parameters tested. Either for a given number of subindexes, or a given number of points examined, it is much more precise. For 8 subindexes, which is the parameter which covers all three methods, the MAP of multicurves is 70% greater than that of the other methods. The methods by Liao et al. and Mainar-Ruiz et al show roughly comparable compromises of efficacy × efficiency in this database.

In what concerns the number of subindexes, though the interval studied is small, there seems to be a fast law of diminishing returns for multicurves. Anyway, practical considerations prevent us from having much more than 10 subindexes (for at this point we already have a tenfold increase in storage needs and 10 disk random access, at least, per query). For this particular database, the interval between 4–8 subindexes seems to be the "sweet spot".

The experiments performed on the Yorck database (Figure 8) keep the number of subindexes set at 8, but vary the number of elements examined in each subindex, which is indicated next to each data point. Keeping the same axes as before, we plot the performance of the three methods.

Multicurves has again the best compromise by a large margin. For the same number of descriptors examined, it increases precision by roughly 35% in comparison to the method by Mainar-Ruiz et al. In this database, which is about 10× larger than the previous, the method by Liao et al. lagged behind.

Multicurves seems to be relatively insensitive to the number of descriptors examined per subindex: doubling this value increases the precision by 3% in this database. This suggests that small values should be employed for most applications.

5. APPLICATION: VISUAL DOCUMENT IDENTIFICATION

5.1 Document Identification

Document identification or *copy detection* consists in taking a query document and finding the original from where it derives, together with any relevant metadata, such as titles, authors, copyright information, etc. It is an important operation to institutions possessing large documental collections, both to answer the needs of the users, which often want to establish the provenance of unidentified documents, and to detect copyright violations.

The task is challenging for visual documents, since we are interested in recovering more than exact pixel-by-pixel copies: even if the document has been subjected to a series of deformations, we still want to identify them (Figure 9). The set of transformations varies from application to application but usually includes translations, rotations, scale changes, photometric and colorimetric transformations, cropping and occlusions, noise of several kinds, and any combination of those.

Image identification systems are a specialization of content-based image retrieval (CBIR) systems, proposed to solve the problem of copy detection. Contrarily to traditional textual search, those schemes work in the absence of annotations, and in contrast

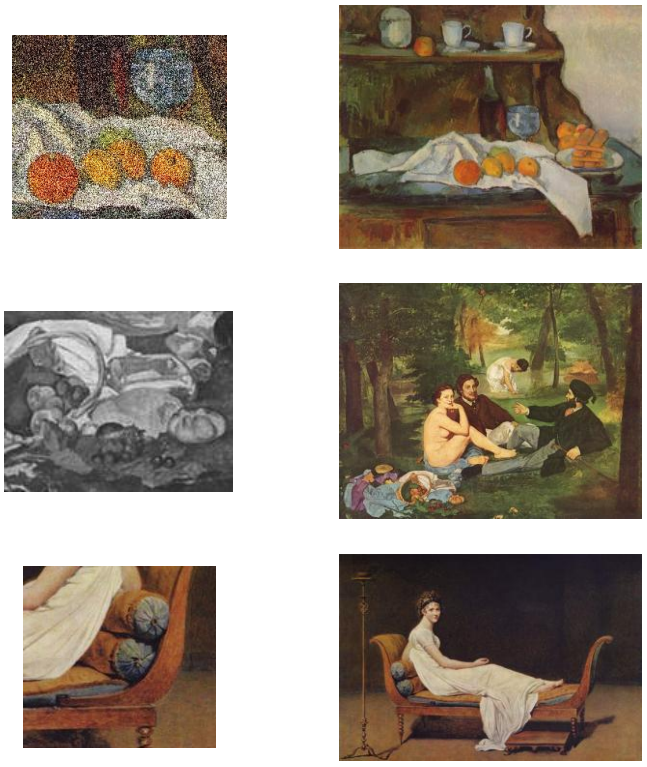

Figure 9: Visual document identification — we should be able to retrieve the original images (right) from the queries (left) even after strong transformations

to watermarking, they can identify documents outside the controlled chain of distribution (which will lack the watermarks).

Like all CBIR systems, image identification systems use descriptors to establish the similarity between the images. But instead of stimulating generalization, exploration and trial-and-error, typical goals of semantic-oriented CBIR systems, they are tuned to emphasize the exactness of image identification and to tolerate transformations which completely disrupt the appearance of the image (such as conversion to grayscale or dithering).

The images may be described either by one descriptor or a set of descriptors. In the former case, when a single descriptor must capture the entire information of the image, we say it is a *global descriptor*. In the later case, the descriptors are associated to different features of the image (regions, edges or small patches around points of interest) and are called *local descriptors*.

Systems based on local descriptors adopt a criterion of vote count: each query descriptor matches with its most similar descriptors stored in the database (using a simple distance, like the Euclidian distance). Each matched descriptor gives one vote to the image to which it belongs. The number of votes is used as a criterion of similarity.

Local descriptor based systems are unsurprisingly much more robust. Because the descriptors are many, if some get lost due to occlusions or cropping, enough will remain to guarantee good results. Even if some descriptors are matched incorrectly, giving votes for the wrong images, only a correctly identified image will receive a significant amount of votes. Unfortunately, the multiplicity of descriptors brings also a performance penalty, since hundreds, even thousands of matches must be found in order to identify a single image.

Systems based on global descriptors [23] have not shown enough precision on the task of image identification, except for slight transformations. In all comparisons, local-descriptor methods have performed better [24][25][26].

Local-descriptor image and video identification are application scenarios where multicurves shows all its advantages. The elevated number of query descriptors makes a fast indexing scheme an absolute requirement for the system is to have practical interest. Furthermore, the approximation of the results induced by the index is not serious, because the loss of a few matches is unlikely to affect the final results. Finally, the enormous size of the databases demands a scalable, disk-friendly and easy to update indexing technique.

5.2 Results

We tested multicurves in an image identification context, for the Yorck Project database, containing over 10 thousand reproductions of paintings. The system architecture follows a classic scheme: we compute the descriptors for every image in the database, and then stock and index those descriptors. When a query image is presented, its descriptors are computed and matched to the 10 nearest descriptors in the database. To get rid of false positives and improve the solution, we apply a geometric consistency step (using a robust model fitting technique [27]), discard all inconsistent matches and then count the votes. The images are ranked by number of votes and presented to the user. The descriptor used is SIFT [21], which has a dimensionality of 128.

One hundred images were selected and suffered intense transformations, which included rotation, size reduction, gamma correction, shearing and dithering. The task consisted in using those images as queries to locate their originals.

First, we have run the system using the exact sequential search to match the descriptors. Since our query images have a large number of descriptors, it is unsurprising that we obtain perfect results (the original is always found), since at least a few dozens of descriptors is correctly matched between query and target every time (and typically, much more).

Then, we have run the system using multicurves with 8 subindexes and examining 512 descriptors per subindex to match the descriptors. Each correctly identified image has lost, on average, about 20% of its votes, but those were so many to begin with, that this did not result in changes in the final ranking, which was still perfect. Running time, however, was between 20 and 25 times shorter.

These results are a testimony of both the robustness of the local-descriptor architecture, and the potential efficiency gains provided by multicurves in those architectures.

6. CONCLUSION

In this paper we have proposed multicurves: a new method for the indexing of multimedia data, based on the use of multiple, moderate-dimensional space-filling curves. Multicurves addresses several requirements rarely considered by multidimensional access methods: small cost for index construction, easy index maintenance, scalability, compatibility with secondary memory storage and easy implementation. When used to perform approximate kNN search, it presents very good performance, comparing very favorably with other state-of-the-art methods based on space-filling curves, both in terms of efficiency and efficacy.

We have also presented a local-descriptor based information-retrieval architecture were multicurves demonstrates its excellent results. In those architectures, a large number of local descriptors is used to retrieve the document, and the eventual mismatching of a few of them is unlikely to harm the final results. In the application we have chosen to evaluate — image identification — multicurves allowed a speed-up of up to 25, without any noticeable precision losses.

Multicurves can be embedded in a DBMS to provide fast access to multimedia data. In a well-conceived architecture, the DBMS will manage all the replications and redundancies at the physical layer, and the logical design may remain untarnished of those considerations.

Our current work is to establish the bounds of approximation of multicurves, to give it a theoretic framework as solid as that of LSH or MEDRANK. In the process, we are also investigating if there are convenient ways to determine the (near-)optimal values for the number of curves (c) and for the association between the dimensions of the data and the dimensions of the subindexes (A).

Since most local-descriptor schemes incur in an excessive number of query descriptors, we are also investigating what is the best compromise between having a very precise matching over a few carefully chosen query descriptors, and a very approximate matching of all query descriptors.

7. ACKNOWLEDGMENTS

The images in the APM database were lent by the Arquivo Público Mineiro. The images in the Yorck database were generously put on public domain by the German company Directmedia Publishing, and made available to the public thanks to the continuous efforts of the Wikimedia Foundation.

Eduardo Valle is sponsored by the CAPES Foundation through the CAPES / COFECUB Program.

8. REFERENCES

[1] Böhm, C., Berchtold, S. and Keim, D. A. 2001. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Computing Surveys (CSUR), 33, 3 (September 2001), pp. 322–373. DOI= 10.1145/502807.502809

[2] Weber, R., Schek, H.-J. and Blott, S. 1998. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In Proceedings of the 24rd International Conference on Very Large Data Bases (New York, NY, USA, August 24–27, 1998). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 194–205.

[3] Samet, H. J. 2006. Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann, San Francisco, CA, USA.

[4] Shakhnarovich, G., Darrell, T. and Indyk, P. 2006. Nearest-Neighbor Methods in Learning and Vision: Theory and Practice. The MIT Press, Cambridge, MA, USA.

[5] Indyk, P. and Motwani, R. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In Proceedings of the 13th annual ACM symposium on Theory of computing (Dallas, Texas, USA, May 24–26, 1998). ACM, New York, NY, USA, pp. 604–613. DOI= 10.1145/276698.276876

[6] Fagin, R., Kumar, R. and Sivakumar, D. 2003. Efficient similarity search and classification via rank aggregation In Proceedings of the 2003 ACM SIGMOD international confe-

rence on Management of data (San Diego, California, USA, June 09–12, 2003). ACM, New York, NY, USA, pp. 301–312. DOI= 10.1145/872757.872795

[7] Lejsek, H., Ásmundsson, H. F., Björnþór, J. and Amsaleg, L. 2005. Efficient and effective image copyright enforcement. Technical report, Institut national de recherche en informatique et en automatique, I. d. r. e. i. e. s. aléatoires, Rennes, France.

[8] Sagan, H. 1994. Space-filling curves. Springer-Verlag, Berlin / New York.

[9] Faloutsos, C. 1986. Multiattribute hashing using Gray codes. In 1986 ACM SIGMOD International Conference on Management of data (Washington, D.C., USA, May 28–30, 1986). ACM Press, New York, NY, USA, pp. 227–238. DOI= 10.1145/16894.16877

[10] Morton, G. M. 1966. A computer oriented geodetic data base and a new technique in file sequencing. Technical Report, IBM Ltd., Ottawa, Ontario, Canada.

[11] Faloutsos, C. and Roseman, S. 1989. Fractals for secondary key retrieval. In Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (Philadelphia, Pennsylvania, USA, March 29–31). ACM Press, New York, NY, USA, pp. 247–252. DOI= 10.1145/73721.73746

[12] Skubalska-Rafajłowicz, E. and Krzyżak, A. 1996. Fast k-NN Classification Rule Using Metrics on Space-Filling Curves In Proceedings of the International Conference on Pattern Recognition (Vienna, Austria, August 25–29, 1996). IEEE Computer Society, Washington, DC, USA, pp. 121–126. DOI= 10.1109/ICPR.1996.546736

[13] Megiddo, N. and Shaft, U. 1997. Efficient nearest neighbor indexing based on a collection of space filling curves. Research Report, IBM Almaden Research Center, San Jose, California, USA.

[14] Shepherd, J. A., Zhu, X. and Megiddo, N. 1999. A fast indexing method for multidimensional nearest neighbor search. In SPIE Conference on Storage and Retrieval for Image and Video Databases VII (San Jose, California, USA, January 26-29, 1999). pp. 350–355. DOI= 10.1117/12.333854

[15] Liao, S., Lopez, M. A. and Leutenegger, S. T. 2001. High Dimensional Similarity Search With Space Filling Curves. In Proceedings of the International Conference on Data Engineering (Heidelberg, Germany, April 02–06, 2001). IEEE Computer Society Washington, DC, USA, pp. 615–622. DOI= 10.1109/ICDE.2001.914876

[16] Mainar-Ruiz, G. and Pérez-Cortés, J.-C. 2006. Approximate Nearest Neighbor Search using a Single Space-filling Curve and Multiple Representations of the Data Points. In Proceedings of the 18th International Conference on Pattern Recognition (Wan Chai, Hong Kong, August 20–24, 2006). IEEE Computer Society, Washington, DC, USA, pp. 502–505. DOI= 10.1109/ICPR.2006.275

[17] Moon, B., Jagadish, H. V., Faloutsos, C. and Saltz, J. H. 1996. Analysis of the clustering properties of Hilbert space-filling curve. Technical Report, University of Maryland at College Park, College Park, Maryland, USA.

[18] Mokbel, M. F., Aref, W. G. and Kamel, I. 2003. Analysis of Multi-Dimensional Space-Filling Curves. Geoinformatica, 7, 3 (September 2003), pp. 179–209. DOI= 10.1023/A:1025196714293

[19] Butz, A. R. 1971. Alternative Algorithm for Hilbert's Space-Filling Curve. IEEE Transactions on Computers, C-20, 4 (April 1971), pp. 424–426.

[20] Bayer, R. and McCreight, E. M. 1972. Organization and maintenance of large ordered indexes. Acta Informatica, 1, 3 (September 1972), p. 173–189. DOI= 10.1007/BF00288683

[21] Lowe, D. G. 2004. Distinctive Image Features from Scale-Invariant Keypoints International Journal of Computer Vision, 60, 2 (November 2004), pp. 91–110. DOI= 10.1023/B:VISI.0000029664.99615.94

[22] Voorhees, E. M. and Harman, D. 2001. Overview of TREC 2001. In TREC 2001 (Gaithersburg, MD, USA, 2001). National Institute of Standards and Technology.

[23] Chang, E. Y., Wang, J. Z., Li, C. and Wiederhold, G. 1998. RIME: A Replicated Image Detector for the World-Wide Web. In Multimedia storage and archiving systems III (Boston, MA, USA, November 2–4, 1998). SPIE, Bellingham, USA, pp. 58–67.

[24] Ke, Y., Sukthankar, R. and Huston, L. 2004. An efficient parts-based near-duplicate and sub-image retrieval system. In Proceedings of the 12th ACM international conference on Multimedia (New York, NY, USA, October 10–16, 2004). ACM, New York, NY, USA, pp. 869–876. DOI= 10.1145/1027527.1027729

[25] Moëllic, P.-A. and Fluhr, C. 2007. ImagEVAL Official Results. In ImagEVAL Workshop (University of Amsterdam, Amsterdam, Netherlands, July 12, 2007).

[26] Valle, E., Cord, M. and Philipp-Foliguet, S. 2006. Content-Based Retrieval of Images for Cultural Institutions Using Local Descriptors. In Geometric Modeling and Imaging — New Trends (London, UK, July 5–7, 2006). IEEE Computer Society, Washington, DC, USA, pp. 177–182. DOI= 10.1109/GMAI.2006.16

[27] Fischler, M. A. and Bolles, R. C. 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, 24, 6 (June 1981), pp. 381–395. DOI= 10.1145/358669.358692