Indexing Personal Image Collections: A Flexible, Scalable Solution

Eduardo Valle, Matthieu Cord, Sylvie Philipp-Foliguet, and David Gorisse

Abstract — *The growth of personal image collections has boosted the creation of many applications, many of which depend on the existence of fast schemes to match similar image descriptors. In this paper we present multicurves, a new indexing method for multimedia descriptors, able to handle high dimensionalities (100 dimensions and over) and large databases (millions of descriptors). The technique allows a fast implementation of approximate kNN search, and deals easily with data updating (insertions and deletions). The index is based on the simultaneous use of several moderate-dimensional space-filling curves. The combined effect of having more than one curve, and reducing the dimensionality of each individual curve allows overcoming undesirable boundary effects. In empirical evaluations, the method compares favorably with state-of-the-art methods, especially when the constraints of secondary storage are considered.*[1].

Index Terms — Descriptor indexing, High-dimensional, Image collections, Consumer images, Near-duplicate detection.

I. Introduction

Personal multimedia devices like digital cameras and multimedia-enabled cell-phones have allowed consumers to gather large personal image collections, sometimes reaching tens of thousands of items. Many tools have been created to deal with those collections, providing services like online sharing, photo collage creation, album organization, image identification, etc. [1]–[4]. Those applications often face the challenge of automatically matching and classifying large amounts of visual data. Descriptors are used to summarize data content. The descriptors are a more compact and — hopefully — semantically richer representation than the raw image pixels. They are usually high-dimensional and may appear at the rate of hundreds per document.

A basic operation needed by those applications is the fast matching of similar descriptors. That can be provided by indexing schemes, but descriptor indexing is challenging from both the theoretical and the technical points-of-view, since the well-known "curse of dimensionality" makes it inherently inefficient. Furthermore, the memory hierarchy brings additional constraints to the implementation.

Many high-dimensional indexing methods have been proposed [5], but few are able to address the practical concerns of very large, very high-dimensional descriptor databases. To be successful, two basic ideas are usually considered: on one hand, querying simultaneously multiple subindexes; on the other hand, reducing drastically the dimensionality of each subindex [6]–[8].

In this paper we propose *multicurves*, an index able to handle high-dimensional image descriptors. Multicurves is based on space-filling curves — a technique which has attracted substantial attention on the field for its ability to create a "vicinity-sensitive" total order on the data, thus allowing the adaptation of the efficient one-dimensional indexing techniques to multidimensional data. The originality of multicurves is to integrate those curves into a very effective structure, where a careful combination of multiple moderate-dimensional curves leads to a large precision improvement.

The paper is structured as follows: in Section II.A, we discuss the subject of image descriptor indexing, exploring some general issues and then reviewing the state of the art in Section II.B. In Section III we describe multicurves, introducing the method and detailing the algorithms for construction and search. In Section IV, we show the empirical evaluation of multicurves, comparing it to other state-of-the-art high-dimensional indexing techniques. In Section III.D we show an interesting application of the method to the task of image identification, i.e., the matching of an original image and its (distorted) copies.

II. Indexing Multimedia Descriptors

A. Image Descriptor Databases

The challenge of multimedia retrieval comes, in great part, from the large semantic gap between how the data are coded and what they represent. Multimedia retrieval seldom uses the raw representation of data, and is instead mediated by the use of descriptors. Descriptors appear in a large variety of forms (color and texture histograms, invariant moments, Fourier coefficients, local jets, gradient maps, etc.) but are often of vector nature, and frequently, high-dimensional.

Besides their elevated dimensionality, multimedia databases are usually very large and the scenario has worsened by the use of local descriptors, which (as we discuss in Section III.D) proliferate at the rate of hundreds per document.

B. Large Scale High-dimensional Indexing

Descriptor matching is usually performed by an operation called kNN search (or *k* *n*earest *n*eighbors search), which finds, in the database, the descriptors most similar to the query descriptor. The simplest solution to kNN search is sequential

[1] E. Valle was supported by a CAPES/COFECUB, and a FAPESP Grant.

E. Valle is with Computing Institute, UNICAMP, São Paulo, Brazil. (e-mail: mail@eduardovalle.com).

S. Philipp-Foliguet and D. Gorisse are with Équipes Traitement de l'Information et Systèmes, 95014 Cergy-Pontoise France. (e-mails: sylvie.philipp@ensea.fr, david.gorisse@ensea.fr).

M. Cord is with the LIP6 lab., UPMC — Paris VI, France (e-mail: matthieu.cord@lip6.fr).

scan. Unfortunately, this brute-force solution is acceptable only for the smallest of databases, being unfeasible in most contexts. The alternative is to use indexing, in order to accelerate the search. However, the performance of indexing depends greatly on the dimensionality of the data. Search time can be made to grow only logarithmically with the size of the base, but at the expense of introducing a hidden constant, which grows exponentially with dimensionality [5][9].

This phenomenon is known as "curse of dimensionality" and expresses the difficulty of partitioning the data or the space in an efficient way when dimensionality is very high [10]. Basically, as dimensionality grows, several counter-intuitive phenomena appear, all of them detrimental to the working of indexes [9]. As far as we know, for over a dozen dimensions, no method can reliably perform exact matching faster than a simple sequential search. This poses a challenge, since descriptors with hundreds of dimensions are usual.

To solve this dilemma, the methods may trade-off exactness for speed. This means that they will find the matching descriptor with good probability, but not for sure. In order to be of practical interest in the context of large-scale multimedia, the scheme must:

- perform well for high-dimensional data, presenting a good trade-off between exactness and speed;
- adapt well to secondary memory, which in practice means that few random accesses must be performed;
- be dynamic, i.e., allow easy data insertion/ deletion, without significant performance degradation.

Despite the abundant literature on multidimensional indexing (for an in-depth account of the bibliography see [5]), surprisingly few methods are able to accomplish those requirements: many assume implementation in main memory (and thus, cheap random access throughout the index), other have prohibitive building times, and so on.

A class of methods which does achieve those goals tries to transform the n-dimensional indexing problem into a one-dimensional indexing problem by using space-filling curves.

C. *Indexing using Space-Filling Curves*

Space-filling curves are fractals whose topological dimension is one, but whose Hausdorff dimension is the same as their embedding dimension. They are thus able to provide a continuous mappings from the unit interval [0; 1] to any unit hypercube $[0; 1]^d$. They were introduced by Peano [11] and Hilbert [12]. Most space-filling curves are constructed by a recursive procedure, in which the space is progressively divided into smaller cells, which are then traversed by the curve. In the limit, the curve fills the entire space (Fig. 1).

Though the study of those curves and of their surprising properties is fascinating in itself, what concerns us here is their ability of inducing a "vicinity-sensitive" total order to the data. What we mean by that is that the curve gives the multidimensional data a total order which, locally and with high-probability, preserves the neighborhood relations of the space (putting near in the curve data which are near in the space).

The use of space-filling curves to perform kNN search is not new. Apparently Faloutsos [13] was the first to explicitly refer to the concept of curves, though earlier authors already used the idea of *bit shuffling*, *bit interlacing* or *bit interleaving*. There is a space-filling curve concept "hidden" in bit shuffling techniques, because interleaving the bits of the individual spatial coordinates induces the appearance of a fractal curve known as "Z-order curve". Faloutsos and Roseman were 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 [14].

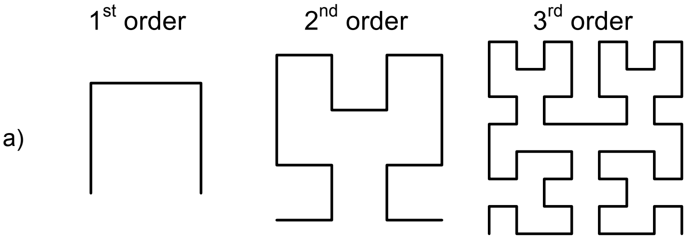

Fig. 1. The recursive space-filling Hilbert curve. Three iterations are shown: the actual space-filling curve is the infinite limit of those iterations. The curve allows mapping an n-dimensional space onto a 1-dimensional line in a way which, locally, preserves the neighborhood relations.

All those pioneering methods were conceptually very simple; they mapped the high-dimensional elements in the curve and then performed a straightforward similarity search using the one-dimensional position in the curve.

A good heuristic is to take the nearest elements in the curve as the nearest elements in the space. The hypothesis is that points that are near to each other in the curve always correspond to points that are near to each other in the space. Unfortunately, the converse is not true, in the sense that near points in the space are not always near in the curve. This is because of boundary effects, which tend to put very far apart points in certain regions of the curve. The matter is seriously aggravated as the dimensionality increases (Fig. 2).

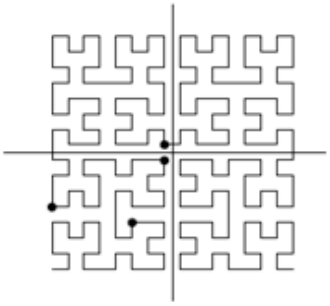

Fig. 2. The problem with boundary effects on the space-filling curves. The points in the centre of the space are further apart in the curve than the points in the lower-left quadrant. On those circumstances, the "neighborhood-preserving" property of the mapping is violated.

In order to conquer the boundary effects, Megiddo and Shaft suggest the use of several curves at once, hoping that, for any given query, at least one of them will not pose boundary problems. They present the idea in a very general way, without describing which curves should be used and how they should be made different [15]. Shepherd et al. develop on this idea, by specifically recommending the use of several identical Hilbert curves where different copies of the data elements are mapped, after random transformations (rotations and translations) [6]. Whether or not the set of transformations could be optimized is left unanswered.

Finally, Liao et al. solve the problem of choosing the transformations, by devising the necessary number of curves and the optimal set of translations to obtain a bounded approximation error in the kNN search [7].

A departure from those methods was suggested recently by Mainar-Ruiz and Pérez-Cortés [16]. Instead of using multiple curves, they propose using multiple instances of the same element in only one curve. Before inserting those instances 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 instances. This has two advantages: first, a single sorted list structure has to be managed; second, at search time, a single random access is performed. The main drawback is the difficulty in controlling the optimal number of instances per element: the method ignores that some regions in the curve are much more problematic than others, and simply associates each element to the same number of instances.

III. The Multicurves Index

A. Introduction

As we have seen in Section II.C, the greatest problem of the use of space-filling curves comes from boundary effects brought by the existence of "zones of discontinuity" in the curves, where the quasi-order-preserving qualities of the mapping are broken. Different methods propose different solutions, usually through the simultaneous use of multiple curves.

Multicurves is also based on the use of multiple curves, but with the important improvement that each curve is only responsible for a subset of the dimensions.

The dimensionality-reduction makes for an efficient implementation of the subindex, reducing the effects of the "curse of dimensionality". Because of the exponential nature of the "curse" it is more efficient to process several low or moderate-dimensional indexes than a single high-dimensional one. This is explained by the fact that not only we gain the intrinsic advantages of using multiple curves (i.e., elements 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 index creation is simple: one subindex (a sorted list data structure) is created for each subspace of the data. The data points are inserted in all subindexes. For each subindex, first the data point is projected onto the associated subspace; then the projection is mapped onto a one-dimensional *extended-key* using the space-filling curve; finally a pair <extended-key, data> is inserted into the list, which is sorted by extended-key.

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 (Fig. 5).

This scheme presents several advantages:

- the sorted list is the mainly used data structure, and it can be handled efficiently (by a B-tree, for example), resulting in a disk friendly method;
- for the same reason, insertions and deletions can be handled easily;
- the offline pre-processing for index construction is reasonable, consisting mainly of sorting operations;
- almost all accesses to the data in the sorted lists are sequential, resulting in savings in disk operations, where random access is expensive.

Before detailing the algorithms, a few definitions (Fig. 3):

- d: the dimensionality of the data elements;
- c: the number of curves to be used in the index;
- $d[i]$: the dimensionality of the i^{th} curve with $\Sigma\, d[i] = d$;
- A: an association between the dimensions of the data and dimensions of the curves, such that the value on $A[i][j]$ indicates which data dimension corresponds to the j^{th} dimension of the i^{th} curve. E.g., if $A[2][3] = 10$, it means that the 3rd dimension of the 2nd curve is in fact the 10th dimension of the data space (Fig. 3);
- m: the number of bits needed to represent each dimension of the data elements, which will correspond to the order of the curve to be generated;

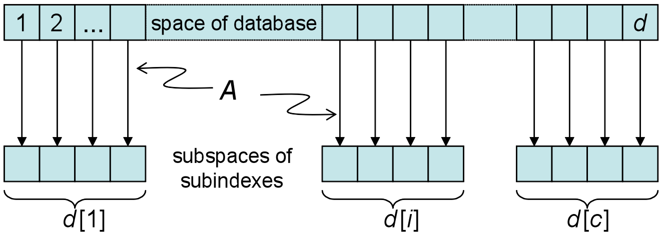

Fig. 3. A sample parameterization of the multicurves showing how the parameters *c*, *d*[*i*], and the association *A* determine the dimensions considered by each subindex.

B. Building the Index

The construction algorithm for multicurves (shown in Fig. 4) is relatively simple, since the underlying structure of the index is just a set of sorted lists, one for each curve.

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, element> are inserted in the curves.

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 on line 10 takes $O(md)$ bit operations (see Section III.D). Assuming an efficient sorted list structure, insertions will take $O(\log n)$ steps, where n is the number of elements in the database. Since we are building c lists, the algorithm should take at worst, $O(c(nmd + n\log n))$ steps.

C. Searching

The search algorithm is also simple (Fig. 5). We choose, beforehand, the number of elements to examine in each subindex (which is called *probe depth*). 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 takes $O(md)$ bit operations (see Section III.D). The time spent looking for the values the 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, where n is the number of elements in the database. 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 Euclidean 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 2–16) is repeated once for every subindex, which means a linear growth with this parameter.

The symbols are explained in Section II.A
points is the list of data elements
point[i] is the value of the i^{th} dimension of point
A[][] is an association between the dimensions of the data space and the subindexes space as explained in Fig.3 .
curves[] is an array of c sorted lists
projection[] is the projection of point[] onto the subspace of the curve
GetExtendedKey() is a function which gives the extended-key in the space-filling curve from the coordinates in the space

```
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 ←
                GetExtendedKey(projection)
11.         Put the pair <extended_key, point> into
            the list curves[curve]
12.     Next
13. Next
14. Return curves[]
```

Fig. 4. The construction algorithm for multicurves. The algorithm decomposes each database point in a set of projections. Each projection is inserted into a sorted list, accordingly to its extended key on a space filling curve.

The symbols are explained in Section II.A and Fig. 4
probe_depth is the number of data items to examine in each curve
query[i] is the value of the i^{th} dimension of query

```
SearchMulticurves (c, d[], A[][], m, curves[], probe_depth, k, query)
    → Returns a list of k nearest neighbors

1.  best ← new empty 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 ←
            GetExtendedKey(projection)
8.      candidates ←
            list with the probe_depth  points the 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
```

Fig. 5. The search algorithm for multicurves. The query point is decomposed into a set of projections. Each projection is used to gather a number of candidates on the sorted list corresponding to the space filling curve associated to that subspace. At the end, the best candidates are returned.

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

D. Discussion

The dimensionality reduction in each subindex is crucial for the performance of multicurves. Not only it serves the pragmatic purpose of making the index implementation more efficient, but it also allows for better comparison of data in high-dimensional spaces. On those spaces, it is counterproductive to take all dimensions into account all the time because at every local cluster, some dimensions act as outliers. In a nutshell this means that we cannot assume that all dimensions "make sense" throughout the space. This somewhat surprising property of high dimensional data had already been observed in the field of data mining [17]. By putting different dimensions on each subindex, multicurves introduces the possibility of ignoring outlier dimensions.

Any recursive space-filling curve could theoretically be used, but in our implementation we have used the Hilbert curve, which in comparison with other space-filling curves, like the Z-order curve or the Gray-code curve, has better clustering properties, mostly because of the absence of distant jumps. Compared to those curves, however, the mapping between the hyperdimensional coordinates and the extended-key is much more complex. Fortunately, there is an efficient algorithm which uses little memory and can map any curve using only $O(md)$ bit operations [18].

It is not necessary to compute the arbitrary-precision version of the curve. If the data coordinates are quantized in *m* bits, a recursive approximation of order *m* is enough to guarantee no loss of precision.

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 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.

Once the index is built, updating consists simply in inserting and removing data from the lists. The ability to do it without degrading the index performance depends, of course, on the underlying data structure used to implement the lists, but, if B-trees are used, the index will be completely dynamic.

IV. Empirical Evaluation

In this section, we present the empirical performance evaluation of multicurves, including the comparison with other state-of-art methods.

A. Experimental Setup

A standardized experimental setup, accepted by the community, is still lacking for the evaluation of high-dimensional indexing. Therefore, one of the main challenges researchers on the subject face is the choice of their databases, queries, ground truth and metrics.

Though early works tended to use synthetic data, following a uniform random distribution, it is now generally accepted that this is unrealistic and leads to overpessimistic results. Therefore, recent works are usually evaluated on real data.

We have chosen to use databases of SIFT descriptors [19], which serve well to our evaluation purposes: they are high-dimensional (128 dimensions in their standard version), can be embedded in a Euclidean space, and, due to their effectiveness, are very well established, having been used extensively in both research and industrial applications.

We have created two databases. The **Small Database** is composed by the SIFT descriptors generated from image transformations of a selection of 100 original images. 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 aggregated 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 **Large Database** is composed by SIFT descriptors generated from about 10 000 original images and amounts to 21 591 483 descriptors. The queries are the SIFT descriptors calculated from originals 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 Euclidean distance. It was computed using the sequential search, a slow method, but which guarantees exact results.

Performance is measured in two axes: effectiveness (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 effectiveness we use a classic metric: the precision (which measures the fraction of relevant answers found). 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 is misleading, since it depends heavily on the machine, the operating system, the current load (concurrent tasks) at the time the experiment is performed and even on the degree of fine-tuning spent on implementation. We have chosen, therefore, to compare the methods by counting, for each method, how many database descriptors were accessed per query descriptor.

Since we are mainly interested in large-scale (thus, disk-based) contexts, a critical metric is the number of random accesses needed to perform the query. Since this operation involves the physical relocation of the I/O heads of the disk, it incurs in severe performance penalties and must be kept at very small values.

B. Evaluated Methods

We have implemented and tested four methods. We have compared multicurves (explained in Section III) with the state-of-the-art of methods based on space-filling curves by Liao et al. [7] and by Mainar-Ruiz and Pérez-Cortés [16] (both methods are explained on Section II.C). All methods were implemented in Java, using the Java Platform, Standard Edition v. 1.6.

For the sake of completeness, we have also compared our method with the improved version of LSH (Locality Sensitive Hashing) of Datar et al. [8]. This version of LSH is also based on the use of multiple subindexes, implemented as hash tables. Each subindex takes into account just a subset of the dimensions of the data, by a clever use of a series of "locality sensitive" hash functions based on the projection of the data onto straight lines. The method is complex, and the user is referred to the cited article for additional details.

We have used E2LSH version 0.1, the publicly available implementation by Andoni, written in C [20]. Unfortunately, this LSH implementation is based on main memory, and rewriting it to disk would demand a very laborious adaptation. We have opted instead to keep it on memory and to measure the number of points accessed, and the number of different tables accessed per query (corresponding to the number of random accesses). All other methods performed on disk.

C. Parameterization

An important parameter for all methods is the number of subindexes (the number of curves for multicurves and the method of Liao et al., the number of hash tables for LSH). The equivalent notion in the method of Mainar-Ruiz et al. (which uses a single curve) is the number of representative instances each database descriptor will have in the index. This decision has an impact on the time spent building the index and on the space it occupies, but chiefly, it influences the number of descriptors accessed and the number of random accesses.

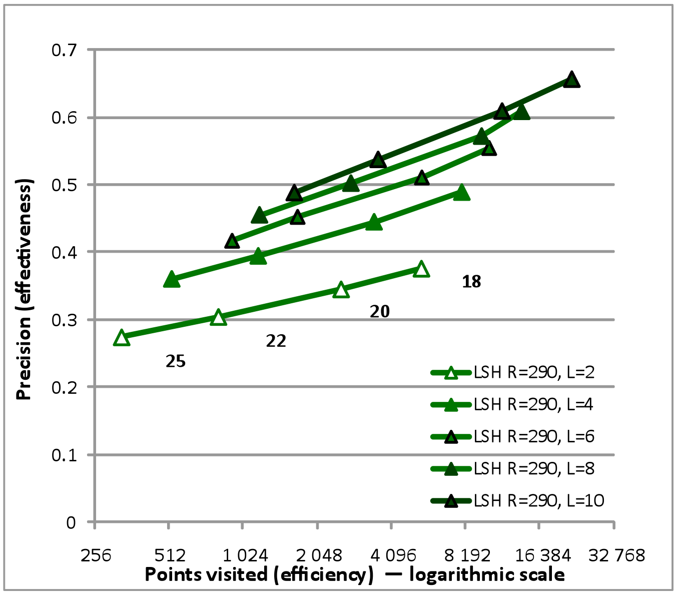

Fig. 6. The compromise between the number of hash tables (L) and the size of the hash key (K, grows from right to left for each data series, from 18 to 25). Experiments performed on the Small database.

TABLE I

IMPACT OF NUMBER OF CURVES ON PERFORMANCE FOR MULTICURVES

# of Curves	Points Visited (Efficiency)	Precision (Effectiveness)
2	1024	0.39
4	2048	0.50
8	4096	0.52
16	8192	0.51

Experiments on the Small Database with Probe Depth = 512.

TABLE II

IMPACT OF PROBE DEPTH ON PERFORMANCE FOR MULTICURVES

Probe Depth (Per Curve)	Points Visited (Efficiency)	Precision (Effectiveness)
512	4096	0.52
1024	8192	0.58
2048	16384	0.65

Experiments on the Small Database with 8 curves.

The other essential parameter, for the space-filling methods is the probe depth, i.e., how far to explore each one of the subindexes. The compromise here is that the more we explore the subindexes, the more we improve the precision, but at the expense of effectiveness. For LSH, there is not a number of elements to explore *a priori*; the number of elements visited is a consequence of two parameters — the selectiveness of the index, and the radius of analysis (a radius that indicates that all potential matches beyond that distance may be safely discarded).

A parameter which affects the evaluation as a whole, is the number of neighbors sought (the k in kNN), since the first neighbors (i.e., the nearest) are easier to find. For the Small database, in which each query image had potentially several matches in the database, we have set k = 20, a margin large enough to fetch the descriptors which will have the most matches. For the Large database, in which each query image had only one match in the database, we have only evaluated the ability of the method to recover the first neighbor.

We have performed our parameterization tests of LSH, on the Small database and covered a large spectrum of its parameters. Three parameters must be set on the version of the LSH we have tested. The number of hash tables, the size of the key used for hashing (which is related to the selectivity — the larger the key, the more selective the hash functions), and the radius of analysis (which can be interpreted as a distance from the query beyond which LSH is allowed to ignore any candidate solution).

The effectiveness × efficiency plot in Fig. 6 shows what happens when we set the radius of analysis (R) and vary the size of the key (K) and the number of hash tables (L). The two latter parameters have inverse effects on the selectivity of LSH: a larger key tends to make each individual hash table very stringent, which can be compensated by introducing more tables. The sweet spot of this compromise is where one obtains the highest precision without visiting a lot of elements (towards the upper-left corner of the graph).

As the plot clearly shows, if one wants to obtain an improvement in precision, the growth in the number of elements visited is much steeper if K diminishes than if L grows, and this is the main reason the parameterization of LSH for main memory tends to use very large values of L, in order to keep K also large. Since this implies a prohibitive number of random accesses, parameterizations intended for disk tend to choose smaller values for both parameters.

The radius of analysis has also an impact on the performance of LSH, but though still sizable, it is not as dramatic. A small radius improves the selectivity of the index, granting a better efficiency, but with the risk of ignoring potential solutions if they lie beyond the radius. We have tested a range of different radiuses, obtaining the best performance at R = 290.

For the comparison with the other methods we retained the series with R = 290 and K = 22.

Compared to LSH, the parameterization of multicurves is more straightforward. Just two parameters have to be set: the number of curves, at construction time, and the probe depth (number of elements examined in each curve), at search time.

Table I shows the plot of effectiveness × efficiency as the number of curves grows. Effectiveness reaches a maximum at 8 curves, where the compromise between the number of subindexes and the representativeness of each subindex is the best. For efficiency reasons, we have to keep the number of subindexes fairly low (10 being an upper limit in practice), because not only the number of points visited is directly proportional to the number of subindexes, but also (and most important) each subindex implies a random access.

The probe depth also has a considerable effect on the effectiveness, since the further we travel in a subindex, the better the chances we compensate for the lesser, local, boundary effects of the space-filling mapping (Table II).

D. *Method Comparison*

We start by comparing the performance of all methods (multicurves, LSH, Mainar-Ruiz et al. and Liao et al.) as the number of subindexes changes (for Mainar-Ruiz, which always uses a single subindex, we varied the number of representative instances assigned to each data point). This comparison is shown in Fig. 7.

The superiority of multicurves and LSH over the other methods is immediately apparent, as they reach a considerably better compromise between efficiency and effectiveness. The "sweet spot" for both methods is in the region around 1000–2000 points visited and precision of 0.4–0.5.

Nevertheless the advantage of multicurves only becomes unambiguous when one takes into consideration the number of random accesses performed. In fact, for the parameterization in the "sweet spot" mentioned, multicurves performs half the number of those expensive operations (indicated in the small numbers next to the data points).

To see how the methods behave in a larger scale context, we performed a comparison in the Large Database, including multicurves, Mainar-Ruiz et al. and Liao et al. (as we have explained in Section IV.B, the available LSH implementation is RAM-based, and thus, cannot deal with a database so large).

This time, we have kept the number of subindexes (representative instances, for Mainar-Ruiz et al.) fixed at 8, and varied the probe depth. The results (Fig. 8) confirm the superiority of multicurves among the space-filling curve based methods.

V. Near-Duplicates in Personal Image Collections

Several applications have been recently proposed to deal with consumers image collections, including autosummarizations and collages [1], organization of photo albums [2][3], identification of locations and point of view of photos [3]. We have chosen the problem of near-duplicate image identification [4]. Near-duplicate detection is useful for many tasks: retrieving lost metadata, finding intersections between sub-collections, removing duplicate removal in retrieval results, finding the relative importance of a scene in summarizations, saving disk space, etc. It also illustrates well the gains provided by the fast matching of high-dimensional descriptors.

A. *Image Identification and Copy Detection*

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, dates, etc. It is an important operation both to institutions and to individual users possessing large documental collections.

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. 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.

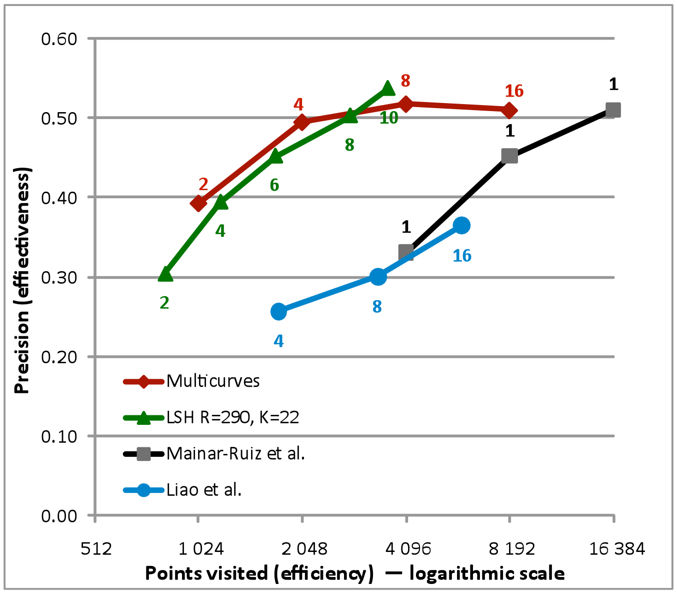

Fig. 7. Comparison of all methods in the Small database. Multicurves and LSH have the best efficiency × effectiveness compromise, but Multicurves performs considerably less random accesses (small numbers).

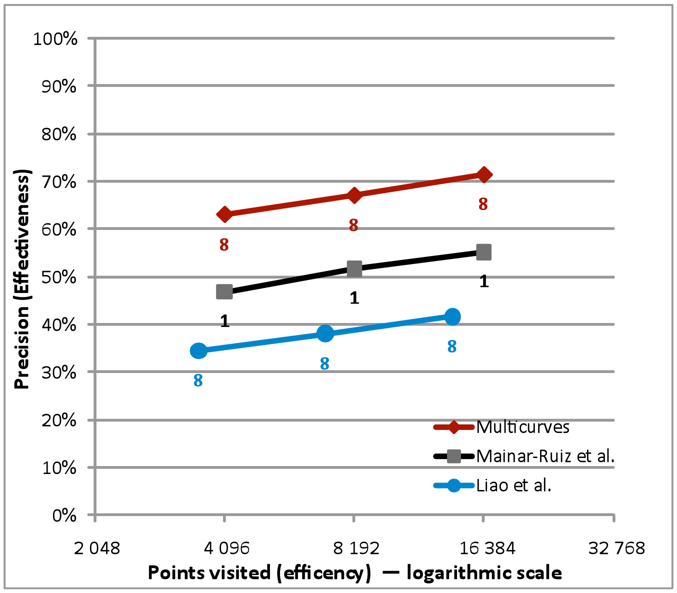

Fig. 8. Comparison of the space-filling based methods in the Large database. Multicurves has the best efficiency × effectiveness compromise.

Image identification systems are a specialization of content-based image retrieval (CBIR) systems, proposed to solve the problem of copy detection. Like all CBIR systems, they 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. When a single descriptor must capture the information of the image, we say it is a *global descriptor*. When the descriptors are associated to different features of the image (regions, edges or small patches around points of interest), they 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 Euclidean 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 have not shown enough precision on the task of image identification, except for slight transformations. In all comparisons, local-descriptor methods have performed better [4][24][25].

Local-descriptor image and video identification are application scenarios where multicurves shows all its advantages. Because of the high number of query descriptors, query times must be low. 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 large size of the databases demands a scalable, disk-friendly and easy to update indexing technique.

B. *Evaluation*

We have tested multicurves in an image identification context, for the Large Database, containing over 10 thousand images. The system architecture follows a classic scheme: we compute the descriptors for every image in the database, and then store 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 [26]), 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 [19], 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 (and typically, much more) are guaranteed to be correctly matched between query and target.

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.

VI. Conclusion

When the database is small enough to fit in main memory, it is reasonable to assume that random access is cheap. In that context, the time spent on descriptor matching is often dominated by the computation of distance functions. The breakthrough of methods like LSH is the ability to dramatically reduce the number of elements examined (and thus, distances computed), saving much CPU time. They introduce, however, the cost of performing a large number of random accesses, making their adaptation to disks very challenging.

In a secondary memory context, it is critical to reduce those accesses, since they involve the physical relocation of the hard disk I/O head, an operation which takes the time equivalent to millions of CPU cycles. In this context, the advantage of multicurves becomes clear, since it provides good precision with a small number of subindexes, and thus, avoids making many random accesses.

Multicurves possesses all desiderata to thrive in a large scale database context: besides being disk-friendly, it is simple to implement and easily accepts updates (due to the fact it is backed by simple sorted lists), and it has a good compromise between precision and speed.

As future work, we would like to explore alternative ways to distribute the dimensions among the subindexes (other than a simple partitioning) and to provide a theoretical model of the approximation bounds of multicurves.

References

[1] C. Rother, L. Bordeaux, Y. Hamadi, A. Blake. "AutoCollage" in *Proc. 33rd Int. Conf. and Exhib. on Comp. Graph. and Interactive Techniques (SIGGRAPH 2006)*. Boston – MA, USA, 2006.

[2] N. Snavely, S. Seitz, R. Szeliski. "Photo tourism: Exploring photo collections in 3D" in *Proc. 33rd Int. Conf. and Exhib. on Comp. Graph. and Interactive Techniques (SIGGRAPH 2006)*. Boston – MA, USA, 2006.

[3] P. Corcoran, G. Costache. "Automated sorting of consumer image collections using face and peripheral region image classifiers," IEEE Trans. on Consumer Electronics, v. 51, n.3, 2005.

[4] Y. Ke, R. Sukthankar, and L. Huston, "An efficient parts-based near-duplicate and sub-image retrieval system," *Proc. 12th ACM Int. Conf. on Multimedia*, New York, NY, USA, 2004.

[5] H. Samet, *Foundations of multidimensional and metric data structures* (The Morgan Kaufmann series in computer graphics). San Francisco, CA: Morgan Kaufman, 2006.

[6] J. Shepherd, X. Zhu and N. Megiddo, "A fast indexing method for multidimensional nearest neighbor search," in *SPIE Conf. on Storage and Retrieval for Image and Video Databases VII*. San Jose, CA, 1999.

[7] S. Liao, M. Lopez and S. Leutenegger, “High dimensional similarity search with space filling curves,” in *Proc. IEEE Int. Conf. on Data Eng.* Heidelberg, Germany, 2001.

[8] M. Datar, N. Immorlica, P. Indyk and V. Mirrokni, “Locality-sensitive hashing scheme based on p-stable distributions,” in *Proc. 12th Annual Symp. on Computational Geometry*. Brooklyn – NY, 2004.

[9] C. Böhm, S. Berchtold and D. Keim, “Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases,” *ACM Computing Surveys (CSUR)*, vol. 33, n. 3, pp. 322–373, Sept. 2001.

[10] R. Bellman, *Adaptive control processes: a guided tour.* Princeton, NJ: Princeton University Press. 1961.

[11] G. Peano, “Sur une courbe, qui remplit toute une aire plane,” *Mathematische Ann.*, vol. 36, n. 1, pp. 157-160, 1890.

[12] D. Hilbert, “Über die stetige Abbildung einer Line auf ein Flächenstück,” *Mathematische Ann.*, vol. 38, n. 3, pp. 459-460, 1891.

[13] C. Faloutsos, “Gray codes for partial match and range queries,” *IEEE Trans. on Soft. Eng.*, vol. 14, pp. 1381–1393, Oct. 1988.

[14] C. Faloutsos and S. Roseman, “Fractals for secondary key retrieval,” in *Proc. 8th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Syst*. Philadelphia, PA, 1989.

[15] N. Megiddo and U. Shaft, *Efficient nearest neighbor indexing based on a collection of space filling curves*. IBM Almaden, San Jose, CA, Research Report RJ 10093, 1997.

[16] G. Mainar-Ruiz and J-C. Pérez-Cortés, “Approximate nearest neighbor search using a single space-filling curve and multiple representations of the data points,” in *Proc. 18th Int. Conf. on Pattern Recognition*. Wan Chai, Hong Kong, 2006, pp. 502–505.

[17] C. Aggarwal, “Redesigning distance functions and distance-based applications for high dimensional data,” *ACM SIGMOD Record*, vol. 30, n. 1, pp. 13–18, 2001.

[18] A. Butz, “Alternative algorithm for Hilbert's space-filling curve,” *IEEE Trans. on Computers*, vol. C-20, 1971.

[19] D. Lowe, “Distinctive image features from scale-invariant keypoints,” *Int. J. of Computer Vision*, vol. 60, n. 2, pp. 91–110, 2004.

[20] A. Andoni and P. Indyk, *E2LSH v. 0.1 User Manual*. June 2005.

[21] E. Valle, M. Cord and S. Philipp-Foliguet, “High-dimensional descriptor indexing for large multimedia databases,” in *Proc. 17th ACM Int. Conf. on Inform. and Knowledge Manage.* Napa, CA, 2008, pp. 739-748.

[22] G. Shakhnarovich, T. Darrell and P. Indyk, (eds.), *Nearest-neighbor methods in learning and vision: Theory and practice*. Cambridge, MA: The MIT Press, 2005.

[23] E. Valle, M. Cord, and S. Philipp-Foliguet, “Fast identification of visual documents using local descriptors,” in *Proc. 8th ACM Symp. on Document Eng.* São Paulo, SP, Brazil, 2008, pp. 173–176.

[24] P-A. Moëllic and C. Fluhr, “Imageval official results,” in *ImagEVAL Workshop*. Amsterdam, Netherlands, 2007.

[25] E. Valle, M. Cord, and S. Philipp-Foliguet, “Content-based retrieval of images for cultural institutions using local descriptors,” in *Geometric Modeling and Imaging — New Trends*. London, UK, 2006, pp. 177–182.

[26] M. Fischler and R. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM*, vol. 24, n. 6, pp. 381–395, 1981.

Eduardo Valle is a B.Sc. and a M.Sc. in Computer Sciences by the Federal University of Minas Gerais (UFMG) and a Ph.D. in Computer Sciences by the University of Cergy-Pontoise. Currently he is post-doctorate researcher at the Computing Institute of the State University of Campinas UNICAMP, working on scalability issues of machine learning and content-based retrieval. He is particularly interested in the applications related to cultural heritage.

Matthieu Cord obtained his Ph.D. degree in Image Processing in 1998 at the University of Cergy-Pontoise, France, and was a post-doc in 1999 at the Katholieke Universiteit Leuven, Belgium. He has joined the ETIS labs in France to create the image indexing research group. In 2004, he has joined the University of Paris 6, where he is a full professor position. He has been recently nominated to the highly selective French Research Institute (IUF). His research interests include computer vision, image processing, machine learning and applications to multimedia information retrieval and multimedia processing.

Sylvie Philipp-Foliguet is a full professor at the National School of Electronics (ENSEA) of Cergy-Pontoise, France, since 1988. She manages the MIDI (Multimedia Indexing and Data Integration) team of the ETIS labs (Information Processing and Systems). Her research domains are image segmentation and interpretation. She has developed a fuzzy segmentation method, methods for inexact graph matching and statistical learning, and worked on applications concerning indexing and retrieval of images, videos and 3D objects.

David Gorisse is an M.Sc. in Computer Sciences by the University of Cergy-Pontoise and M.Sc in electrical engineering and telecommunications by ISEN. Currently he is a Ph.D. Student in Computer Sciences in MIDI team of ETIS at the University of Cergy-Pontoise.