A Robust Technique to Recognize Objects in Images, and the DB Problems it Raises

Laurent Amsaleg, Patrick Gros

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Robust Technique to Recognize Objects in Images, and the DB Problems it Raises

Laurent Amsaleg , Patrick Gros

N˚4081

Novembre 2000

THÈME 3

ISSN 0249-6399 ISRN INRIA/RR--4081--FR+ENG

A Robust Technique to Recognize Objects in Images, and the DB Problems it Raises

Laurent Amsaleg* , Patrick Gros†

Thème 3 — Interaction homme-machine,
images, données, connaissances
Projets Temics et Vista

Rapport de recherche n ° 4081 — Novembre 2000 — 27 pages

Abstract: In the context of content-based image retrieval from large databases, traditional systems typically compute a single descriptor per image based for example on color histograms. The result of a query is in general the images whose descriptors are the closest to the descriptor of the query image. Systems built this way are able to return images that are globally similar to the query image, but can not return images that contain *some of the objects* that are in the query. Recent advances in image processing techniques, however, make this possible by computing local descriptors that are well suited for detecting similar objects in images. Obviously powerful, this fine-grain image recognition also changes the way the retrieval process is performed: instead of submitting a single query to retrieve similar images, multiple queries must be submitted, their partial results post-processed before delivering the answer. This paper first presents a family of local descriptors that support fine-grain image recognition. Our technique is robust: it detects similar objects in color images despite orientation changes (rotation of objects), translations, resolution changes, illumination variations, and partial occlusions. Many multi-dimensional indexes have been proposed to speed-up the retrieval process. These indexes, however, have been mostly designed for and evaluated against image databases where each image is described by a single descriptor. While this paper does not present any new indexing scheme, it shows that the three most efficient indexing techniques known today are still too slow to be used in practice with local descriptors because of the changes in the retrieval process.

Key-words: robust fine-grain image recognition, multimedia indexing, multidimensional indexing, image databases, content-based retrieval, local descriptors

(Résumé : tsvp)

* Équipe Temics, IRISA - CNRS
† Équipe Vista, IRISA - CNRS

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Une technique robuste pour reconnaître des objets dans des images, et les problèmes bases de données que cela pose

Résumé : Les systèmes permettant la recherche d'image par le contenu calculent généralement un seul et unique descripteur par image, par exemple selon un histogramme de couleurs. La réponse à une interrogation est typiquement composée des images de la base dont les descripteurs sont les plus similaires de celui de l'image requête. Ce type de système permet de retrouver au sein d'une grande base les images qui sont globalement similaires à l'image requête, mais ne permet pas de retrouver des images montrant certains objets contenus dans l'image requête. Toutefois, les avancées récentes en traitement d'image rendent cela possible via le calcul de multiples descripteurs locaux qui sont bien adaptés à faire de la reconnaissance d'objets de grain fin. Ce type de reconnaissance change bien évidemment les capacités de reconnaissance d'un système, mais change aussi profondément la manière dont les recherches s'effectuent : au lieu de ne soumettre qu'une seule requête et d'attendre que le système retourne les images similaires, de multiples requêtes consécutives doivent être soumises, et une phase de synthèse des résultats partiels doit être effectuée avant de rendre une réponse. Cet article présente d'abord une famille de descripteurs locaux qui permettent de faire de la reconnaissance robuste d'objets de grain fin : ils retrouvent des objets similaires mais dont la position, l'orientation, l'illumination, le facteur d'échelle diffèrent. Par ailleurs, de nombreuses techniques d'indexation multidimensionnelles ont été proposées pour accélérer les recherches sur disque. Ces index ont été majoritairement conçus et évalués dans le contexte de bases d'images à descripteur unique. Bien que cet article ne présente pas de nouveau schéma d'indexation, il montre que les 3 meilleures techniques d'indexation multidimensionnelles à l'heure actuelle sont encore bien trop lentes lorsqu'elles sont utilisées conjointement avec des descripteurs locaux.

Mots-clé : reconnaissance d'images grain fin, indexation multimédia, indexation multidimensionnels, bases d'images, recherche par le contenu, descripteurs locaux

1 Introduction

Image processing and database techniques are together required to build large content-based retrieval systems. Image processing techniques are needed to extract useful *descriptors* from images. Descriptors are typically vectors of real numbers defining points in a high-dimensional space. Descriptors encode information found in images, and they are used during the search process. The similarity of two images is assumed to be proportional to the similarity of their descriptors, which is measured as the (typically Euclidean) distance between the points defined by the descriptors. Similarity search is therefore implemented as a nearest-neighbor search or as a ε-range search within the feature space.

Traditional methods for computing descriptors include color histograms and correlograms [SS94, HKM+97]. These schemes, used in QBIC [FBF+94], Virage, or Excalibur[1], typically compute a *single* descriptor per image. One descriptor therefore encodes information that is global to one image. In this context, content-based retrieval is performed at a coarse-grain level: the system returns the images that are *globally* similar to the query image. The system can not detect, however, that two images contain similar objects, but at different locations, in front of different backgrounds, from different viewpoints or differently illuminated. To address this problem, modern image processing techniques have recently focused on fine-grain image recognition. Fine-grain – or object – recognition in images typically requires to compute many descriptors per image. These descriptors are often called *local descriptors* because one descriptor encodes information that is local to a (small) area of an image. In general, an object in one image is represented by several descriptors.

The first contribution of this paper is the description of a method designed for fine-grain image recognition that is very robust to changes in images. This method computes local descriptors that are well suited for detecting similar objects in color images despite orientation changes (rotation of objects), translations, resolution changes, illumination variations and partial occlusions. With respect to the region-based image recognition scheme proposed by Natsev, Rastogi and Shim [NRS99], our method is robust to more changes (e.g., their approach is not robust to rotations or illumination changes), is not based on multiple sliding windows and is not based on wavelets which force these windows to be square-shaped.

In general, ignoring our descriptors, fine-grain image recognition impacts the retrieval chain when searching for similar images. First, it increases the size of the database: Instead of storing one descriptor per image, many of them must be stored for each image. The size of the database is typically increased by two order of magnitude. Second, it changes the way similar images are searched: Instead of searching the descriptors that are close to the unique query descriptor (as it is the case with global descriptors), the system starts by computing all the descriptors that describe the query image and then queries the database many times, each time using a different local query descriptor. Each (partial) answer is kept around and once all query descriptors have been used, answers are cross-checked and the set of similar images is eventually returned to the user. For example, in the context of our experiments

[1] Information about these systems can be found at http://www.qbic.almaden.ibm.com/, http://www.virage.com/ and http://www.excalib.com/.

(see Section 4.3), searching for the images containing objects that are similar to the ones in the query image typically generates between 50 and 600 consecutive queries that search in a database that is 50 to 600 times larger. This demanding process increases the impact of the performance problems traditional multi-dimensional index techniques suffer from.

The second contribution of this paper is an initial exploration of the consequences of using local descriptors together with up-to-date database multi-dimensional indexing strategies. While this paper *does not present any new indexing scheme*, it experimentally shows that the three most efficient indexing techniques known today are still too slow to be used in practice with local descriptors. Using today's DB indexing techniques with local descriptors requires to *add-up* the response time of each individual query, which makes the global response time far above what one might tolerate. We therefore list problems and enumerate several potential solutions for building efficient content-based retrieval systems supporting fine-grain image recognition as suggested by modern image processing techniques.

The remainder of this paper is structured as follows. Section 2 describes and evaluates our local descriptors tailored for robust object recognition. Section 3 gives an overview of the multi-dimensional indexing techniques used in databases. Section 4 evaluates the performance of the three most efficient techniques known today on a large base of high-dimension data in the context of local descriptors. Section 5 presents some open issues and an initial set of solutions before concluding.

2 Local Descriptors for Robust Object Recognition

The descriptors we present in this section are an extension to color images of the fine-grain recognition scheme for grey-level images originally proposed by [FtHRKV94] and extensively used and evaluated by [SM97]. We built on this scheme because it is highly robust to grey-level image transformations: it detects similar elements in images despite orientation changes (rotations), translations, resolution changes, illumination variations, partial occlusions, changes of backgrounds or viewpoints, etc... Coping with color images instead of grey-level images has a deep impact on the way the descriptors handle (i.e., absorb) the variations of illumination. The main goal of this section is to present the original recognition scheme proposed by [FtHRKV94] and our extension to color images. First, however, we briefly present an interesting example of a system using local descriptors published in the DB literature.

In Walrus [NRS99], each image is broken-down into square sliding windows of varying sizes. The signature of a window is computed by using the s^2 coefficients from the lowest frequency band of the Haar wavelet transform for the window. These signatures are clustered, and the centroid of each cluster is then used as the descriptor for the region. This method provides invariance towards translations, scaling effects, partial occlusions and to color shifts, i.e., when the values of the pixels are shifted from (r, g, b) to $(r+\alpha, g+\beta, b+\gamma)$, where α, β and γ are constant values. Color shifts, however, do not correspond to any natural variation of the illumination conditions (variation of illumination intensity). In addition, wavelets force the sliding windows to be square-shaped, which forbids their scheme to be

invariant to rotations. Last, their scheme evaluates similarity on a common area criterion, but does not consider the degree of similarity of the areas. Therefore, their scheme will favor large regions that are roughly similar to small regions that are almost identical.

2.1 The local differential invariants

In [FtHRKV94], Florack et al. described a new methodology for computing highly robust descriptors for fine-grain grey-level image recognition. Before going into details, their scheme can be roughly outlined as follows.

Computing the local descriptors that encode information about a single image is done in three steps. First, specific points in the image, called interests points, are selected. The number of points in one image varies, since it depends on the shape of the signal of that image. Second, the signal around each interest point is characterized by its convolution with a Gaussian function and its derivatives up to the third order. Third, these derivatives are mixed to enforce invariance properties and to make descriptors robust to the changes mentioned above. A descriptor is typically a vector of real numbers having 7 or 9 dimensions.

Descriptors are then inserted into an index. To know which image a descriptor has been computed from, image identifiers are stored together with the descriptors. Computing descriptors over all the images is done off-line. The similarity retrieval proceeds as follows: interest points are first identified in the query image, and the corresponding local descriptors are then computed. Each descriptor of the query is used to probe the index. The index returns similar descriptors found in the database (with respect to a nearest-neighbor or a ε search), from which it is possible to determine the id of the associated image. It is therefore easy to count the number of time each image id is returned by the index during the whole retrieval process. At the end of this process (i.e., once all the local descriptors of the query have been used to probe the index), the counters allow to rank the candidate images by decreasing similarity.

We now present in more details the computation process of our descriptors.

2.1.1 Extracting Interest Points

Interest points are determined such that it is very likely that a point found in one image will be also found in another image that slightly differs from the first one. Schmid, who extensively used Florack's method, compared several point extractors in [SMB98] and showed that the extractor introduced by Bigün [BGW91] and improved by Harris [HS88] had the best behavior.

This extractor looks for 2D singularities in the signal, and is based on the computation of the eigenvalues of the matrix

$$\mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}} \otimes \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

where I_x and I_y are the convolution of the signal with the two derivatives $\partial G/\partial x$ and $\partial G/\partial y$ of a Gaussian function. In general, there are many points in each image, typically between 50 and 600.

Dufournaud [DSH00] pointed-out that descriptors could not be invariant to scale factors if the points extracted were not invariant to scaling. Therefore, he proposed to extract the points at different scales by varying the variance of the Gaussian function used. This allows to cope with scale factors up to 7, rather than up to 2 as for the original method.

2.1.2 Computing Local Descriptors for Grey-Level Images

Once the points are extracted, the descriptors are computed using the signal around each of the points. This computation is done in two steps. During the first step, the signal is convoluted with a Gaussian function and its 9 first derivatives (the derivatives up to the third order). These smoothed derivatives provide a basic description of the signal. During the second step, the derivatives are mixed together to enforce invariance properties and to make descriptors robust to the changes mentioned above. We describe below the way these derivatives are mixed.

Gaining Translational and Rotational Invariance. Translational invariance is obtained by the fact that the descriptors are computed around each interest point.

The angle of rotation of images can be algebraically eliminated from the ten derivatives, providing nine resulting quantities invariant to rotations. If the smoothed signal is denoted by I, and its derivatives by I_x, I_y, I_{xx}..., these 9 invariant quantities are:

$$\begin{array}{c} I \\ I_i I_i \\ I_i I_{ij} I_j \\ I_{ii} \\ I_{ij} I_{ji} \\ \varepsilon_{ij} \left(I_{jkl} I_i I_k I_l - I_{ikk} I_i I_l I_l \right) \\ I_{iij} I_j I_k I_k - I_{ijk} I_i I_j I_k \\ -\varepsilon_{ij} I_{jkl} I_i I_k I_l \\ I_{ijk} I_i I_j I_k \end{array} \tag{1}$$

The function ε_{ij} is defined by $\varepsilon_{xy} = -\varepsilon_{yx} = 1$, $\varepsilon_{xx} = \varepsilon_{yy} = 0$. The Einstein's notation used in the formulas corresponds to a summation over each index: for instance, $I_i = \sum_i I_i = I_x + I_y$ and $I_{ij} I_{ji} = \sum_i \sum_j I_{ij} I_{ji} = I_{xx} I_{xx} + 2 I_{xy} I_{xy} + I_{yy} I_{yy}$.

Gaining Photometric Invariance. The descriptors are invariant to the variations of illumination that are modeled by $I \mapsto aI + b$. This model, although simple, describes quite accurately what happens when the global intensity of the illumination varies slightly. It is possible to withdraw these two new parameters a and b and to obtain 7 rotational and

photometric invariants: First withdraw the two first quantities of (1) and divide each of the seven last ones by the appropriate power of $I_i I_i$ such as to obtain ratios of degree 0 with respect to I and its derivatives.

Gaining Scale Invariance. Invariance to scale is achieved by adopting a multi-scale approach: the computation of invariants is repeated for various values of the variance of the Gaussian, and all the resulting values are used to describe the image. The values of the variance used here are related to the ones used during the extraction of the interest points: the variance used to extract a point gives the variance to compute the associated descriptor.

2.1.3 Extension to Color Images

The method used to compute the local descriptors for grey-level images is very robust, as evaluated by [SM97]. We therefore extended this method to cope with color images. Each pixel of a color image is defined by 3 values which can be coded in many ways [Poy97]: RGB, HSV, Lab, ... The RGB system is chosen because it facilitates the extension of the descriptors to color images.

Extending the local descriptors does not change the way the interest points are extracted. We still use Harris's detector. The signal, however, is now characterized by 30 derivatives (10 per channel). Coping with color deeply changes the way these derivatives must be mixed to gain invariance towards rotation and illumination variations. We detail these changes below.

Gaining Rotational Invariance. Rotational invariance is obtained by withdrawing the angle of rotation. 3×9 invariants can be computed with equations (1) applied on each channel, and two other ones are chosen among the three following quantities (for numerical reasons, it is wise to keep all three values in practice):

$$R_x G_x + R_y G_y \qquad R_x B_x + R_y B_y \qquad G_x B_x + G_y B_y \tag{2}$$

Scale invariance is obtained by a multi-scale approach similar to the one used with grey-level images.

Gaining Photometric Invariance. Photometric invariance is more complex with colors because different illumination models can be considered. A general model is $(R', G', B')^T = M(R, G, B)^T + V$ where M is a 3×3 matrix and V a vector of dimension 3. Other models are obtained by using a diagonal or scalar matrix for M and by possibly removing the vector V. According to the number of parameters of the model, the dimension of the descriptors varies between 18 and 29.

A very common model is obtained when M is diagonal. In this case, the three channels remain independent when the image is transformed. The descriptors can thus be computed on each channel like they were in the grey-level case, and by adding two more dimensions using the formulas in (2). This provides descriptors of dimension 24.

Another case is that of a full rank M matrix when no rotational invariance is needed. In this case, the vectors (R, G, B), (R_x, G_x, B_x), (R_y, G_y, B_y)... are all subjects to the same linear or affine transform. The vector V can be withdrawn by not considering the vector (R, G, B). Remains the linear part. The basic invariant in this case are the coordinates of the vectors with respect to three of them chosen as a reference frame. If we choose (for reasons of symmetry) the three vectors (R_{xx}, G_{xx}, B_{xx}), (R_{xy}, G_{xy}, B_{xy}), and (R_{yy}, G_{yy}, B_{yy}) as reference frame, then the invariants are:

$$\frac{\begin{vmatrix} X & R_{xy} & R_{yy} \\ Y & G_{xy} & G_{yy} \\ Z & B_{xy} & B_{yy} \end{vmatrix}}{\begin{vmatrix} R_{xx} & R_{xy} & R_{yy} \\ G_{xx} & G_{xy} & G_{yy} \\ B_{xx} & B_{xy} & B_{yy} \end{vmatrix}}, \quad \frac{\begin{vmatrix} R_{xx} & X & R_{yy} \\ G_{xx} & Y & G_{yy} \\ B_{xx} & Z & B_{yy} \end{vmatrix}}{\begin{vmatrix} R_{xx} & R_{xy} & R_{yy} \\ G_{xx} & G_{xy} & G_{yy} \\ B_{xx} & B_{xy} & B_{yy} \end{vmatrix}}, \quad \frac{\begin{vmatrix} R_{xx} & R_{xy} & X \\ G_{xx} & G_{xy} & Y \\ B_{xx} & B_{xy} & Z \end{vmatrix}}{\begin{vmatrix} R_{xx} & R_{xy} & R_{yy} \\ G_{xx} & G_{xy} & G_{yy} \\ B_{xx} & B_{xy} & B_{yy} \end{vmatrix}} \ldots$$

where (X, Y, Z) is respectively equal to (R_x, G_x, B_x), (R_y, G_y, B_y), $(R_{xxx}, G_{xxx}, B_{xxx})$... The invariants to both rotations and a photometric model with full rank matrix M have not been derived yet, as far as we know.

2.2 Evaluating the Recognition Capabilities of the Descriptors

In general, evaluating the recognition capabilities of descriptors is difficult. Experimental results depend obviously on the intrinsic power of the descriptors, but also on the content of the database. For example, it is really hard to retrieve only sunshines when querying with a sunshine image a database containing solely sunshines and sunrises (they are not so different, after all), but this is almost trivial when the database contains only images of sunshines and of dark tropical forests. The impact of this subtle side-effect can be limited when the database stores images specifically chosen to stress a particular aspect of the descriptors. For example, the robustness to illumination variations of the descriptors can be precisely evaluated if the database includes a series of identical images that differ only by their illumination characteristics. The database we used to evaluate our color descriptors contains such images in addition to other real-life non-specific images.

2.2.1 Evaluation with Grey-Level Images

Schmid extensively evaluated the descriptors proposed by Florack. Her evaluations show clearly the robustness of the descriptors in the context of grey images. The database used for her experiments was made of about 1000 images: 200 pieces of art, 100 aerial images of the downtown of Marseilles (France), and 720 images of 3D objects.[2]

The aerial images are the most challenging. They "look" almost similar (roofs do not differ so much) and images composed of 3D micro structures (chimneys, antennas, cars seen

[2]This database can be browsed at
http://www.inrialpes.fr/movi/pub/Demos/Reconnaissance_Cordelia/en/reconnaitre_objets.html

from above, etc.). In her tests, the query images were not part of the database. These images were taken during a subsequent pass over the city. This changes the viewpoint (the facades of buildings become visible or disappear) and the composition of images (some cars have moved). The plane, however, took all the images in a short time frame, making unchanged shadows.

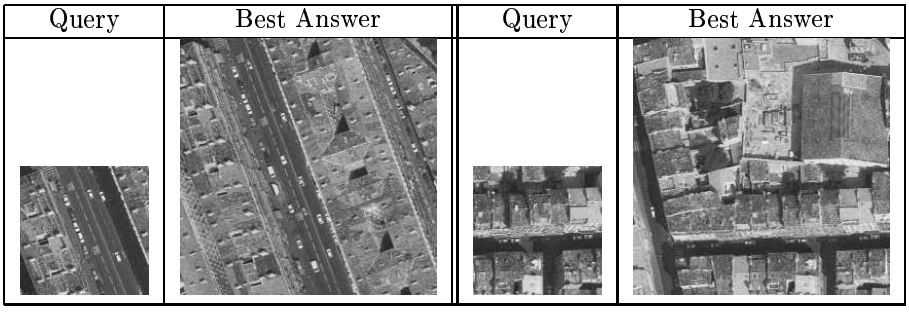

Figure 1: Two queries made with image fragments and the most similar images found by the retrieval system.

To illustrate the recognition power of the descriptors, we insert here two examples extracted from Schmid's work. Figure 1 shows the best answer returned by the system (i.e., the most similar image) when queried by a small image fragment. As stated above, viewpoints differ between query and database images. Queries using fragments provide good results as long as the size of the fragment stays above 10% of the size of the original image. Smaller fragments degrade the results rapidly.

Figure 2 illustrates another interesting result. Several images of a dinosaur seen from very different points of view were stored in the database. 3 different query images were then cooked-up using a complex background on which a view (that is not in the DB) of the dinosaur was superimposed. In addition, the left part of the third query image was deleted (only the tail of the dinosaur remains visible). The system was then asked to return the most similar image for each of these 3 query images. For all queries, the best answer is the image of the dinosaur that is on the right of the Figure (it is the point of view in the DB that is the closest to the one in each query). This result emphasizes the robustness of the method caused by the locality of the descriptors and the by the counting process.

2.2.2 Evaluation with Color Images

To evaluate the recognition power of our extension to color images, we used a database made of the 24 dimension descriptors derived from 1,816 real-life color images. 1,206 images come from 50 seconds of a video. The remaining images come from a database of still images

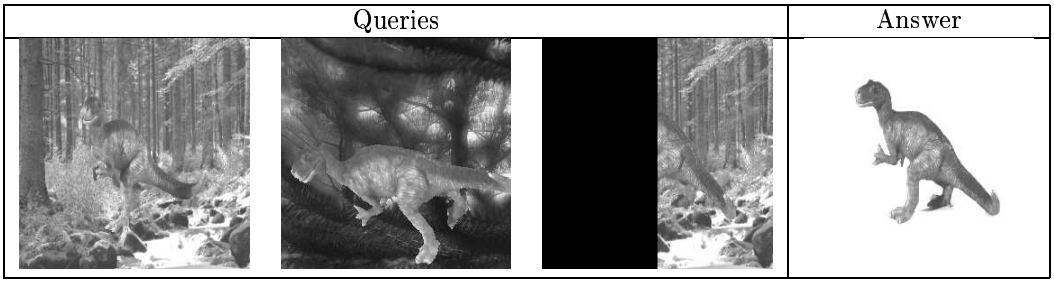

Figure 2: Three queries with cluttered complex scenes and the retrieved image (on the right)

found on the web[3]. The total number of descriptors computed from these images is 413,412 and the database occupies about 40 Mb on disk.

The still image database is composed of several sequences of images where one or two parameters vary slowly. For instance, there are sequences with a variation of the intensity of the light source, sequences taken by a rotating or a translating camera, ... This allows to test specifically the robustness of the descriptors and of the retrieval method towards these parameters. In all the figures below, the number of descriptors that characterize each query image is indicated together with the query image. In addition, the number of descriptors that matched is indicated together with each image returned by the system. Note that the average noise level (i.e., the number of matches whatever the query is) is between 5 and 10.

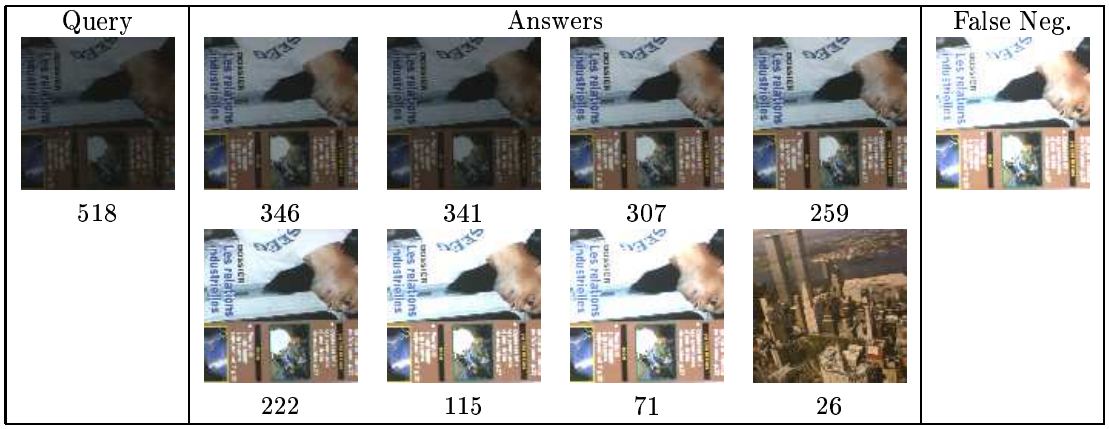

Figure 3: Robustness towards variation of illumination. An example with the query image, the eight most similar images retrieved, and a false negative.

[3]http://www.inrialpes.fr/movi/pub/Images/index.html

Evaluation of Photometric Invariance. Robustness towards intensity variation of the main light source was tested using a sequence of 9 images presenting this variation. The first image of the sequence was used as a query, and 7 of the other images were retrieved as the most similar, as illustrated by Figure 3. The brightest image of the sequence is a false negative: although it is very similar, it is quite saturated and many descriptors were thus unusable.

Another interesting photometric variation occurs when the spectrum of the light source varies. Such variations are handled correctly by a illumination model having a diagonal matrix when it remain of small magnitude. This can be seen on the results shown on Figure 4. For greater variations, a model with a full rank matrix is needed.

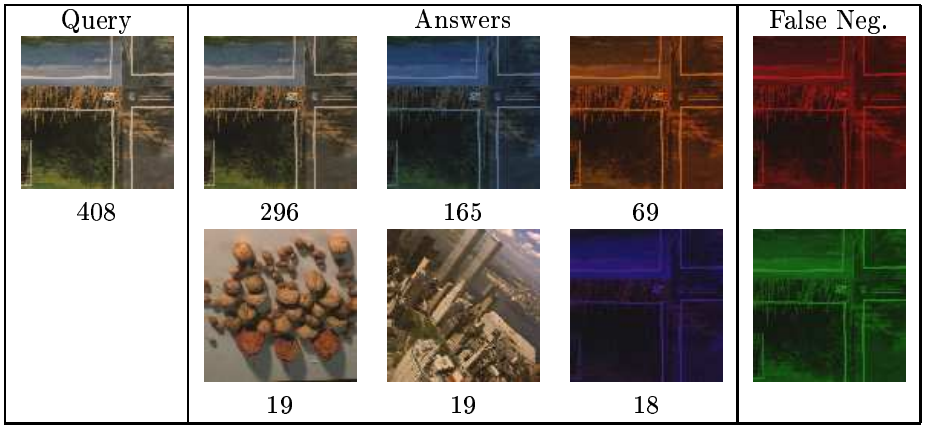

Figure 4: Robustness towards light spectrum variations. An example with the query image, the six most similar images retrieved, and two false negatives.

The last case we studied is the motion of the light source (e.g., the sun which moves during the day). Being robust to that motion enables for example to query the system using a morning image of a landscape and to get evening images of that same landscape. The results in this case are irregular and depend on the intensity of the shadows. As far as they are not too dark, there is still enough information to compare light and dark objects. Some results are illustrated by Figure 5 using a sequence of 7 images.

Evaluation of Geometric Invariance. We present here only one result using a sequence where the camera is moving with a very general motion. The sequence is composed of 20 images and we chose the eleventh one as a query. 14 images of this sequence where found as being the most similar to the query image, as illustrated by Figure 6.

Evaluation of the Robustness to Changes in the Composition of Images. In this test we used a sequence of images in which various objects are added or removed. Then,

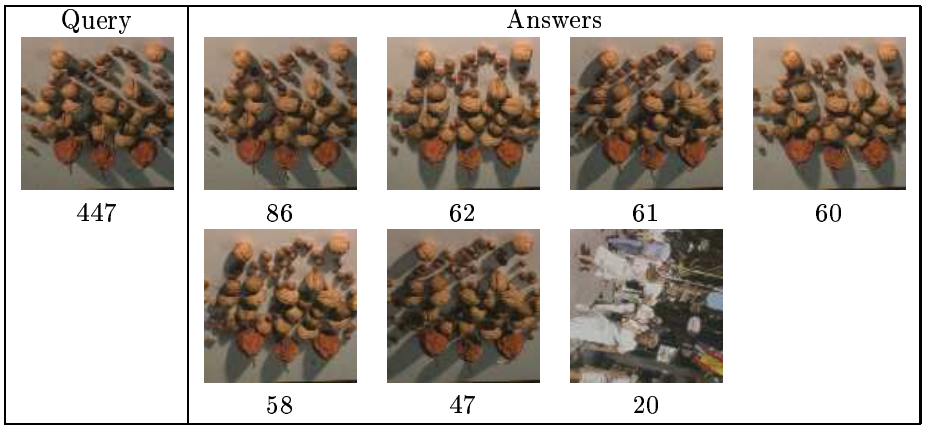

Figure 5: Robustness towards light source motion. An example with the query image and the seven most similar images retrieved.

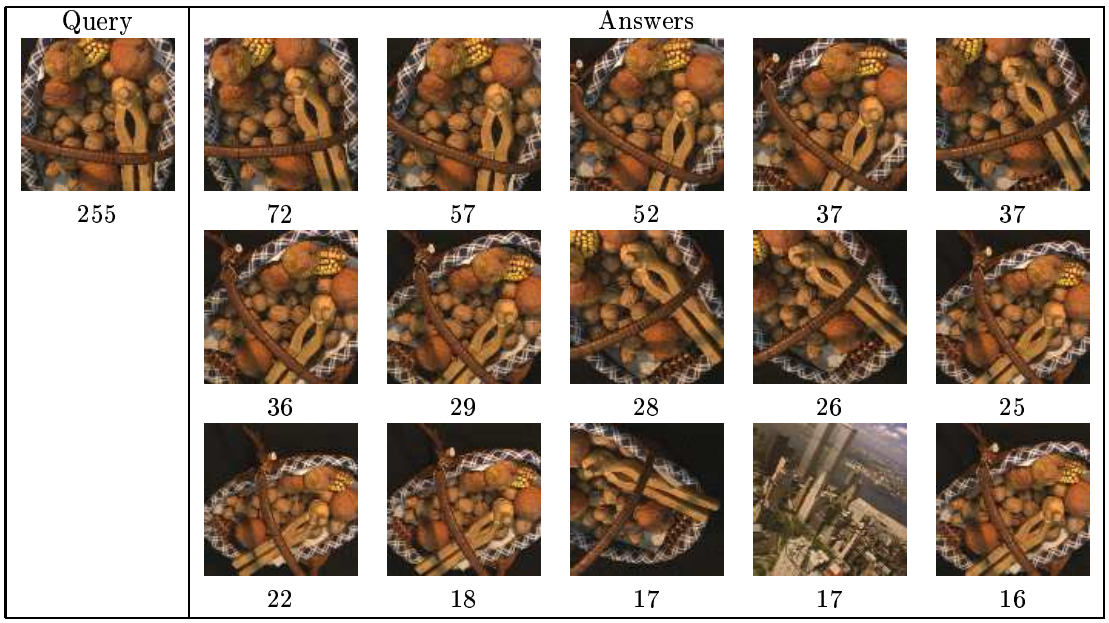

Figure 6: Robustness towards geometric variations. An example with the query image and the 15 most similar images retrieved.

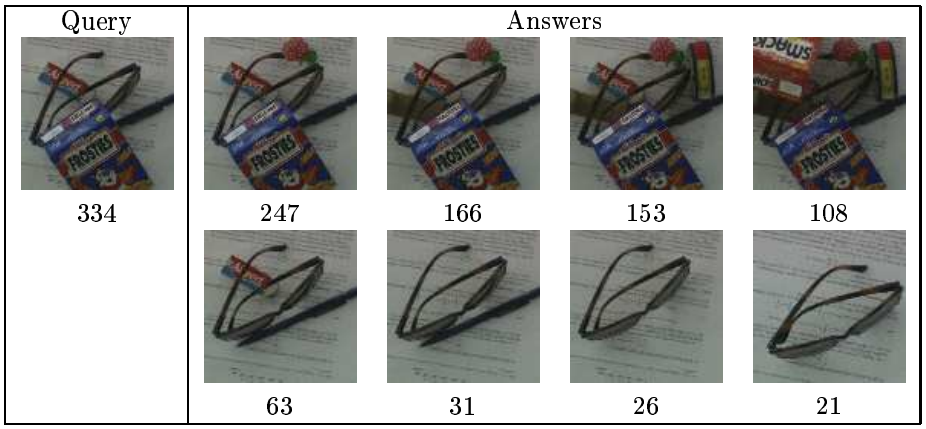

Figure 7: Robustness towards variations in the composition of images. An example with a query image and the eight most similar images retrieved.

we picked one of these images to be the query, and asked the system to return the most similar images. An example of this is given by the Figure 7. Images with additional objects are preferred over images with missing objects because they usually allow more matches between descriptors.

Some Results with the Images Extracted from a Video. When one image from a video is used to query a database storing that video, the images found the most similar are usually the neighboring images belonging to the same shot in the video. This is confirmed by our results as illustrated on Figure 8.

3 Database Techniques for Multi-Dimensional Indexing

The descriptors presented above are robust and well suited to detect similar elements in color images. It is therefore natural to integrate them in a large content-based retrieval system. In this case, database indexing techniques are needed to enhance similarity-based searches. We thus give in this section an overview of the techniques used in databases for indexing still images. We first present the traditional approaches. We then focus on the two strategies that provide today the most efficient support for multi-dimensional searches. We chose these two strategies to built our own system. Their performance, however, is far too slow to make the system usable in practice, as detailed in the next section.

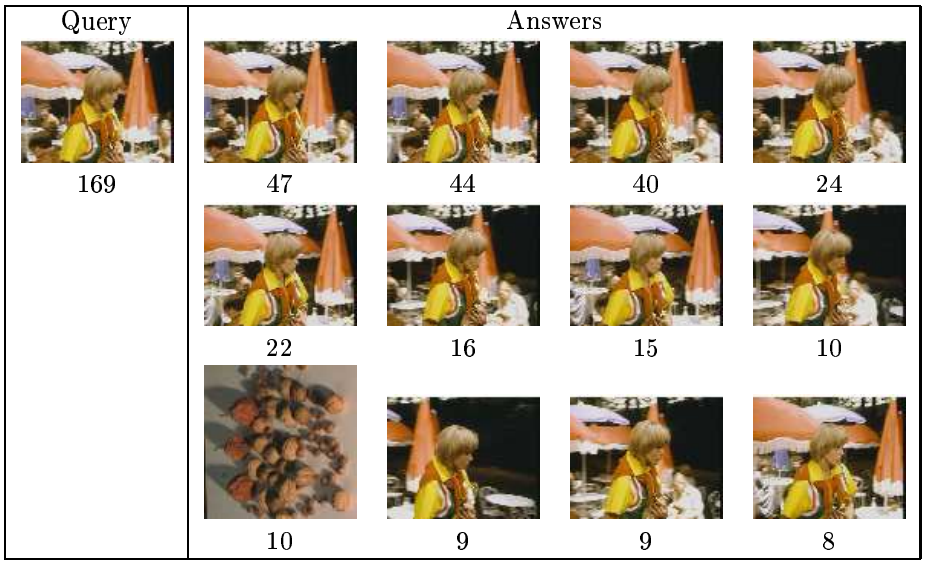

Figure 8: Querying with an image extracted from a video: the most similar images come from the same shot (except image #9).

3.1 Traditional Approaches

Still images indexing techniques can be classified in two families: *data-partitioning index methods* that divide the data space according to the distribution of data, and *space-partitioning index methods* that divide the data space along predefined lines regardless to the actual values of data and store each descriptors in the appropriate cell. All techniques fill the data space with descriptors or with approximations of them.

Data-partitioning index methods all derive from the seminal R-Tree [Gut84], in which minimum bounding rectangles and overlapping are key concepts. Various strategies can be used to determine which rectangles should be merged or kept separated at each level of the tree [BKK96]. The SS-Tree [WJ96] is an extension of the R-Tree that relies on spheres instead of rectangles. The SR-Tree [KS97] specifies its regions as the intersection of a bounding sphere and a bounding rectangle since this reduces the space volume into which searching.

In general, tree construction is usually achieved by splitting nodes in overflow into two equally filled nodes, i.e., at the 50%-quantile. This fill factor enforces balanced trees and maximizes disk usage. [BBK98], however, demonstrates that in the general case, using a 50%-quantile leads to the unexpected effect that, in high-dimensional spaces, the probability of accessing every index page gets close to 1. It is therefore likely that the whole index as

to be scanned during a search process. The resulting access pattern to disk pages severely hampers the search performance since it is totally random.

Space-partitioning techniques like grid-file [NHS84], K-D-B-Tree [Rob81], LSD^h-Tree [Hen98] typically divide the data space along predetermined lines regardless of data clusters. Actual data are subsequently stored in the appropriate cells. These techniques are known to become inefficient when the dimension of data gets above 10 to 16 dimensions. They also face the problem of indexing large volumes of empty space. For example, dividing each of the 30 dimensions of a data space in two distinct regions creates 2^{30} cells. This number is by far greater than the typical number of points defined in a data space. In addition, when the query point is near a cell boundary, the search process may have to lookup many cells in the neighborhood, increasing the search cost. Furthermore, it is likely that most of these cells are empty. The evaluation of the most recent approaches like [AGGR98] or [HK99] show that they are efficient with low-dimensions and for a small amount of noise (ε-search).

3.2 VA-File and Pyramid-Tree

All the techniques presented above generally work well for low-dimensional spaces. Their performance, however, is known to degrade as the number of dimensions of the descriptors increase, e.g., above 10 to 16, as evaluated by [WSB98]. This phenomenon is known as the *dimensional curse*. In other words, navigating within the index becomes more costly than a simple sequential scan in high-dimension spaces when searching nearest-neighbors.

Two innovative approaches, the Pyramid-Tree [BBK98] and the VA-File [WSB98], however, have been recently proposed to tackle head-on the dimensional curse phenomenon: they have been designed specifically such that their behavior does not dramatically degenerates when the indexed data has many dimensions. These two strategies provide today among the most efficient support for multi-dimensional similarity search. We therefore used them for our performance evaluations (see Section 4).

The VA-File approach comes from the observation that the brute-force sequential scan proves to be competitive in high-dimensions (it is often the fastest search technique). Therefore, their approach tries to boost the sequential search by eliminating many useless comparisons using rough approximations of descriptors. Their method manages two different sets of data: a file storing all the descriptors, and another file storing the geometrical approximations of these descriptors. The performance of this method is at its best when this latter file fits in main memory.

To compute the geometrical approximations of the descriptors, the method first splits each dimension d_i in 2^{b_i} slices (coded using b_i bits), such that all slices are equally full. All d dimensions are sliced this way. The intersection of slices define 2^b cells, where $b = \sum_i b_i$, numbered from 0 to $2^b - 1$. To fill the index, all the descriptors are then read, and the approximation of a descriptor is given by the cell number into which it falls. The file storing the geometrical approximations of the descriptors therefore associates a descriptor id to a cell number. Only cells in which at least one descriptor falls are kept in the file, avoiding

the problem of managing many empty cells as mentioned above. Note that this file is small since cell numbers are typically smaller than descriptors.

During a search, the algorithm first uses approximations to determine which cells are close to the query point, and scans them in an increasing order of distance. Starting from the closest cell, the algorithm sequentially fetches the associated descriptors and performs distance calculations. This process is repeated until n nearest-neighbors are found, or as soon as it is detected that the next candidate cell stores only descriptors that can not be closest than the current n-th neighbor. Restricting the search to close cells and ordering their investigation filters out cells that can not be part of the result, and therefore filters out all the irrelevant descriptors. This reduces the number of records to fetch and the number of comparisons and calculations to perform with respect to the traditional sequential scan.

Berchtold, Böhm et Kriegel propose, with the Pyramid-Tree [BBK98], a method that divides a space $[0,1]^d$ in $2 \times d$ pyramids. The top of each pyramid is placed at the center of the data space $(0.5, \ldots, 0.5)$. The base of each pyramid has a surface of $d-1$ dimensions. Each pyramid is assigned a different number. Each pyramid is then cut in slices that are parallel to its base. The nature of pyramids is such that the slices close to their tops are smaller than the ones near their bases. This division of the data space has the interesting property to create a number of cells that increases linearly (and not exponentially) with the number of dimensions.

Dividing the data space in sliced-pyramids enable to map any point of the multi-dimensional space into a pair (pyramid number, height in the pyramid). Because of this mapping, a B^+-Tree index can be used instead of a multi-dimensional index structure. B^+-Trees are known to be very efficient for this type of data and for range queries. A given slice of a specific pyramid is stored as a page of the B^+-Tree. In addition to their efficiency, B^+-Trees are known to nicely cope with concurrent updates and can be made failure resistant. These two properties are very desirable and often lack to other solutions.

4 Performance Evaluations

This section summarizes the evaluation of the performance of the VA-File, the Pyramid-Tree and the sequential scan. As mentioned earlier, these 3 techniques proved to be efficient in the context of nearest-neighbor or ε-range searches with global descriptors within a multi-dimensional space. We therefore measured their performance when used together with local descriptors. The first experiment shows the performance of the techniques when the dimension of the descriptors increases. The second experiment shows the impact of the size of the database on the response times. Last, the third experiment, which is the most relevant to this paper, shows the influence of the (large) number of descriptors forming a single query on the response times. We first describe our experimental setup.

4.1 Experimental Environment

We used the source code of the VA-File and of the Pyramid-Tree provided by their respective authors to perform our performance evaluations.[4] We also implemented our own version of the sequential search. All the algorithms were ran on a SUN Ultra 5 workstation running SunOS 5.7. Its CPU is a 333 MHz UltraSPARC-IIi, with 384Mb of main memory and 8Gb of local secondary storage. All the response times reported here have been obtained using `getrusage()`.

We analyzed the codes of the VA-File and of the Pyramid-Tree to insert at the appropriate places timer start and stop instructions. We slightly changed the metric used by the Pyramid-Tree to compute the distances between points in the data space: it was L_∞ and we changed it to L_2. Without this patch, the nearest-neighbors returned by the Pyramid-Tree would not have been identical to the ones returned both by the VA-file and the sequential search. This patch seems to have no significant impact of the response-time.

4.2 Implementation of a Sequential Search Strategy

In addition, we implemented a sequential search strategy. This implementation assumes that all the query descriptors fit in main memory.[5] All query descriptors are read at once at the beginning of the search. Then, each descriptor of the database is read sequentially and compared against all the query descriptors. The sets of neighbors for each query descriptor are maintained dynamically.[6] The result is delivered once the end of the database is reached. This implementation behaves somehow like a join in which the smaller relation is fully fetched before reading tuple after tuple the larger relation. Our implementation of the sequential search is less than 300 lines of C++.

4.3 Overview of the Database

For the different experiments shown afterward, two databases where created. The first one has already been described (see Section 2.2.2). It is made of 413,412 descriptors of 24 dimensions derived from 1,816 real-life color images. The distribution of the descriptors along each dimension is far from being uniform. For example, the second component varies from about -20 to about 36, and 99.14% of the values are between -1 and +1. Since many performance evaluations published in the literature assume uniformity of data distribution, we generated our second database in which the $24 \times 413,412$ values have been picked between 0 and 1 using a random uniform generator. Queries use images and random vectors that are different from those stored in the databases.

[4] We are grateful to Roger Weber who graciously gave us his implementation of the VA-File. The source code of the Pyramid-Tree is available on the Web page of Stefan Berchtold (`http://www.stb-gmbh.de/~berchtol/`)

[5] There are on average 150 descriptors in each query, and this occupies only about 15K in main memory in the case of 24 dimensions (150*24*sizeof(float) = 150*24*4 = 14,400.)

[6] If we assume that 10 nearest-neighbors are maintained for each query descriptor, then, about 100K are required to store the all these neighbors when 150 descriptors are in the query.

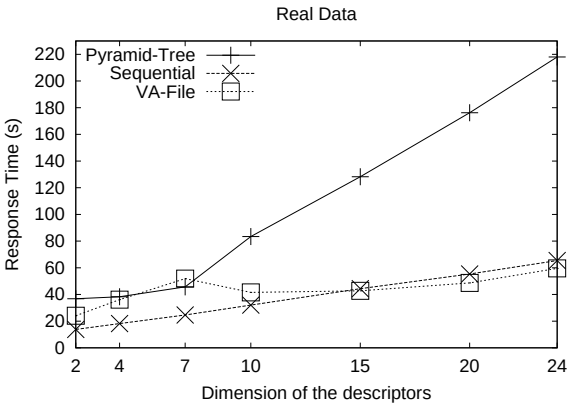

Figure 9: Real data

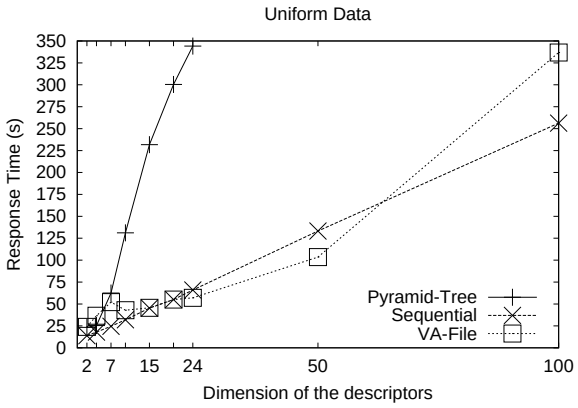

Figure 10: Uniform distribution

Figures 9 and 10: Database storing 413,412 descriptors, 150 descriptors in each query, *increasing dimension of descriptors*

4.4 Experiment 1: Influence of the Dimensionality of Data

The first experiment shows the influence of the dimensionality of data on the performance of the three technique we study here. For this experiment, we first computed the 413,412 descriptors having 24 dimensions using real data from the database described above. These descriptors were then truncated to 2, 4, 7, 10, 15, 20 et 24 dimensions. The other dimensions are used to get intermediate results. 413,412 descriptors following a uniform distribution have then be randomly generated for the same dimensions, but also for greater dimensions (up to 1,000) for experimental purpose. The sizes of the resulting databases is given by Table 1.

dimension	size	dimension	size	dimension	size
2	3,307,296	15	24,804,720	100	165,364,800
4	6,614,592	20	33,072,960	250	413,412,000
7	11,575,536	24	39,687,552	500	826,824,000
10	16,536,480	50	82,682,400	1000	1,653,648,000

Table 1: Size in bytes of the databases for various dimensions of the 413,412 descriptors stored

Once the databases created, a query containing 150 descriptors was computed using an image outside the database or new random numbers. We then truncated them to the appropriate dimensions in order to create the requests that will query the real and the synthetic

databases. The response times given by the Figures 9 and 10 are the cumulative response times of 150 consecutive databases interrogations, each returning 10 nearest-neighbors.

The performance of the algorithms using real data are illustrated by Figure 9. In this case, the performance of the Pyramid-Tree severely degrades above 7 dimensions. Beyond, the response time of this technique is too big to remain competitive. The VA-File and the sequential search clearly exhibit better performance, and degrade less rapidly when the dimension of the data increases. The performance of the sequential search is linear with the dimension, which is normal and without surprises. Sequentially searching 150 descriptors among 413,412 descriptors takes approximatively 14 seconds in a 2-dimensions data space, and about 66 seconds for 24 dimensions.

The performance of the VA-File and of the sequential search are rather similar, except when the number of dimensions is small. In this case, for 2 dimensions, a VA-File search takes about 24 seconds, and about 52 seconds in 7 dimensions (25 seconds are needed in 7 dimensions for the sequential search). When the number of dimensions grows, the VA-File can divide its data space in smaller cells, thereby augmenting the efficiency of its filtering strategy. On the other hand, less dimensions makes the filtering less selective, and exploiting the approximations in addition to computing many actual distances is part of the observed overhead.

The performance corresponding to the experiments that use uniform data are given by Figure 10. This Figure does not show any response time of searches for data having more than 100 dimensions since they become too high to remain significant. In this Figure, the Pyramid-Tree is again the technique having the worst response time. Below 15 dimensions, the sequential search performs better than the VA-File, for similar reasons as the ones mentioned above. When data has 50 dimensions, the VA-File returns its answer (recall that 150 consecutive queries must be submitted before returning the answer) in about 104 seconds while the sequential search needs 134 seconds. In this case, the VA-File strongly benefits from the uniformity of data, from the geometrical approximations and from its filtering strategy. Above 50 dimensions, the sequential search becomes faster than the VA-File. With 100 dimensions, the sequential search needs 256 seconds and the VA-file 336. These results are confirmed by those given in the article presenting the VA-File (see their Figure 12). This article says that above 45 dimensions, the approximation files becomes too large to fit in main memory, increasing the number of I/Os and the overall response time.

Regardless of the nature of the data stored in the database (real or uniform), the response times needed to search the 10 nearest-neighbors of 150 descriptors in a rather small database are big: around a minute for both the sequential scan and the VA-File with 24 dimensions. These response times are above what one might tolerate if these techniques were part of a real system. The next experiment investigates further the influence of the size of the database on the response times.

4.5 Experiment 2: Influence of the Database Size

The Figures 11 and 12 show the impact of the size of the database on the response times of the 3 techniques we evaluated. For this experiment, we reused the 413,412 descriptors

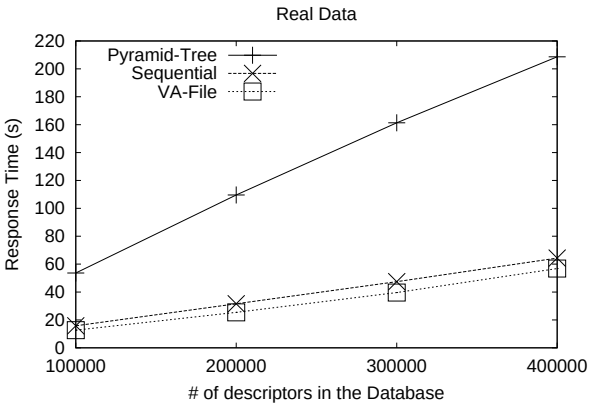

Figure 11: Real data

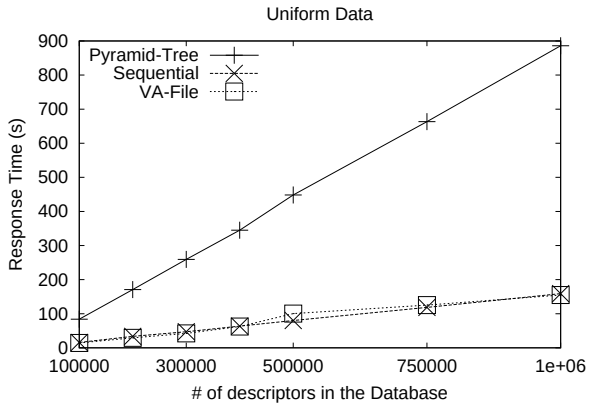

Figure 12: Uniform distribution

Figures 11 and 12: 24 dimensions descriptors, 150 descriptors in each query, *increasing the size of the database*

previously computed with 24 dimensions, and generated new databases by keeping only 100,000, 200,000, 300,000 and 400,000 of them. The requests are made of the same 150 descriptors in 24 dimensions as above. We could not easily create larger databases since the amount of real data we could use was limited. It is easy, however, to create uniform databases of arbitrary sizes. We therefore created such databases, and the larger we generated contains 1,000,000 descriptors (96Mb), and this could correspond to more than 6,500 images if we assume that an image is described by 150 local descriptors on average.

Figure 11 shows the case with real data. What was observed in the previous experiment can be found here again. That is, the Pyramid-Tree is more expensive than other techniques, and that the VA-File is slightly better than the sequential. Still, 15 seconds are needed to perform a search of a database made of only 100,000 descriptors.

Figure 12 shows the case with uniform random data. For a database made of 1,000,000 descriptors the response time for the sequential scan or for the VA-File is of about 3 minutes (160 seconds). This result clearly forbids the use of these techniques in a real system indexing millions of images (and therefore many more descriptors). Furthermore, our request has only 150 descriptors, and they are, in the general case, more numerous. The next experiment focuses on this problem, and shows the influence of the number of descriptors in a request on the response times.

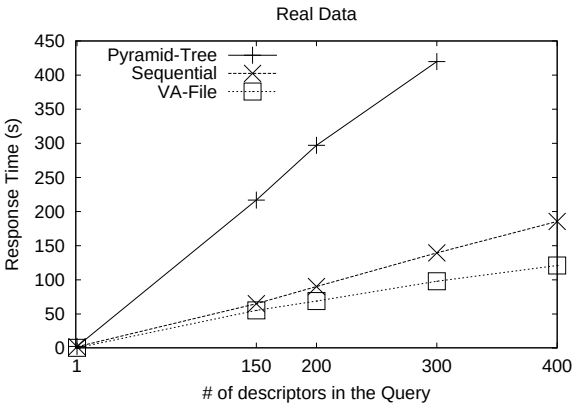

Figure 13: Real data

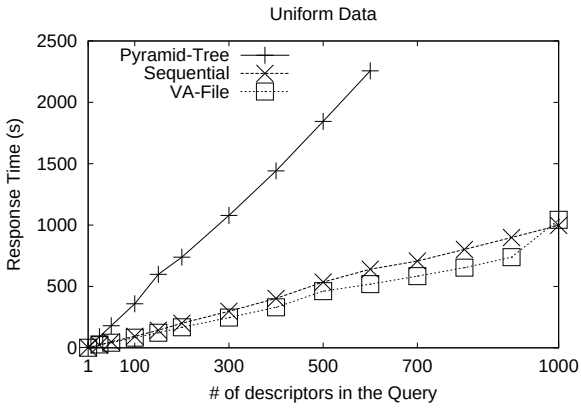

Figure 14: Uniform distribution

Figures 13 and 14: Database storing 413,412 descriptors having 24 dimensions, *increasing the number of descriptors in each query*

4.6 Experiment 3: Impact of the Number of Descriptors in a Request

All the descriptors used in this experiment have 24 dimensions. To generate the queries used here, we searched in our real database images for which 100, 200, 300 and 400 descriptors were computed. We also made-up an artificial query that has only one descriptor since this is the typical case for which the database techniques have been designed for. Forging synthetic queries is trivial, and the largest one had 1,000 descriptors.

The results of this experiment given by the Figure 13 (for real data) and by the Figure 14 (for uniform distribution) show again that only the VA-file and the sequential scan remain interesting. The response time, however, rapidly grows with the number of descriptors in the query. For example, 400 real descriptors cause the response time to jump to 121 seconds for the VA-File and to 185 for the sequential. With uniform data, 997 and 1,042 seconds are needed respectively for the VA-File and the sequential with 1,000 descriptors.

The number of descriptors in each query is directly related to the number of interest points detected in the query image (see Section 2). This number can clearly be very big depending on the image and on the detection strategy. It is crucial that the cost of a query having many descriptors does not increase as illustrated by this experiment.

5 Conclusion and Perspectives

The dimensional curse phenomenon makes existing multi-dimensional indexing techniques barely efficient when content-based retrieval is performed on a (traditional) global similarity

criteria. Fine-grain image recognition, supported by local descriptors, magnify even more this phenomenon. While performance problems are clearly seen in our experiments, worse results are expected if the techniques were used in a more realistic environment, where the size of the image bank is far bigger than our database (up to several Gb), where the descriptors have many more dimensions (couple hundreds) or where the number of descriptors used for one query is much greater (thousands).

To fully exploit the power of fine-grain image recognition, it is therefore crucial to come-up with new indexing techniques specifically designed to efficiently support the use of local descriptors. We therefore list in the following several directions for future investigations aimed at tempering the above mentioned effects.

Numerous local descriptors per queries creates redundancy. When local descriptors are used, recognition is based on multiple consecutive searches, each returning information which, once accumulated and post-processed (cross-checking), gives the final answer. Some images stored in the database will belong to this final answer because several descriptors of the query matched with several descriptors associated to these images. There is therefore a certain form of redundancy in the information used during the complete search process (because all these query descriptors are associated with a *single* query image) and in the information returned (because an image is found similar since many of its descriptors match). It is possible to use this redundancy in (at least) two ways.

First, the search process can be restricted such that it checks only the descriptors that are in the same cell than the one in which each query descriptor falls.[7] This avoids the typical and mandatory lookup of all neighboring cells during nearest-neighbors searches, which is known to be expensive since many cells must be visited. If the search process returns, for *each* query descriptor, only the nearest-neighbors that are *in the same query cell*, and ignores other potential neighbors that are in adjacent cells, than the search cost would be reduced. The result of each query, however, is clearly a rough approximation of what would be returned if the normal search process was enforced. The quality of this approximation is improved as the time goes by since many query descriptors are used to get the images that are similar to *one* image. Cross-checking what is returned by each individual search is a natural way to consolidate the final answer and fully uses the observed redundancy.

This search process therefore needs only to determine the cell in which one query descriptor falls, fetch sequentially the descriptors stored in that cell, and computes actual distances.[8] This simple strategy is repeated until all query descriptors are used this way. It has the interesting property to trade accuracy (of the final result) for efficiency.

The second way to use the redundancy is to stop the search before having used all query descriptors. In this case, the search is greedy, and each partial result returned by each individual query is immediately processed and updates the (in-progress) final result. When

[7] The meaning of the word *cell*, in this section, includes the notions of leaves for tree-based indexing schemes, and the traditional cells defined in space-partitioning index methods, etc...

[8] It is unlikely that *all* query descriptors fall in empty cells. If too many empty cells are found, then the search can switch-back to its regular behavior.

this current (not yet complete) final result has a high probability to be the complete final answer, the search is stopped, the remaining query descriptors are skipped, and the result is returned to the user.

Note that this second search strategy is orthogonal to the first one mentioned above. Both might be combined to search only the relevant cells (ignoring adjacent cells) for a limited number of query descriptors.

Exploit the distribution of data to accelerate the queries. Not all descriptors carry the same amount of information: some are associated to many images, some others are more rare. Therefore, the matching of two descriptors returns a more or less discriminative information, making the associated database image to be more or less likely part of the final result. In this case, a Bayesian formalism may help in determining the probability for each match to help converging towards the final result. In addition, it is possible to sort the descriptors in the query such that it starts with the descriptors that are the most informative. It is therefore possible to stop the search as soon as the probability of having the final answer is high enough, or as soon as the search starts using the descriptors that do not help converging. This technique has the interesting property to refine the search as the time goes by. In addition, the search accuracy can easily be made controllable by the user.

Change the management of memory to benefit from consecutive queries. Traditional techniques assume that a single search within the database is sufficient to return the final answer. Therefore, was is fetch in memory during a search benefits to the next query only by chance: if the second query is lucky enough to use some of the data brought in memory by the first query, than its response time is enhanced because some data is already cached. A better mechanism can be designed when local descriptors are used. In this case, we know in advance that a large amount of consecutive queries will be submitted to the database. Therefore, it may be interesting to pick the next query descriptor with respect to what is already in the cache. That is, the next descriptor used to query the database can be the one which is the most likely to have its nearest neighbors *already in memory*, brought in by previous descriptors. Therefore, instead of consuming all the query descriptors sequentially as the natural search process does, descriptors are picked in a memory conscious way. In addition, since all the query descriptors are known before hand, prefetching might be used to remove cache misses from the critical path.

Use several low-dimension indexes instead of a unique high-dimension index. It is known that the cost of content-based retrieval grows fast when the dimension of data increases. It is therefore potentially interesting to evaluate if querying many low-dimension indexes in parallel instead of a unique high-dimension index gives good results. These "small" indexes must be constructed in such a way, and their use must be such that the result they return is identical to what would return a regular index.

A small index would store the same descriptors as the ones stored by the large index. These descriptors, however, would be truncated and would keep only specific dimensions

chosen with care. A particular dimension might be kept by more than one small index. A query would then have to be transformed in multiple sub-queries, each interrogating a given (small) index. If these index are physically stored on different machines, then large grain parallelism is possible.

This scheme tries to limit the problem of dimensionality curse by enforcing multiple interrogations of low-dimension data for which efficient indexing schemes exist. On the other hand, additional processing steps are needed, and the global size of the database (i.e., the size occupied by all the descriptors, and not by the images) is increased. These disadvantages must be put in perspective with the potential enhancements provided by the parallelization and the efficiency of each sub-query.

References

[AGGR98] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. of the ACM SIGMOD, June 2–4, Seattle, Washington, USA*, pages 94–105, 1998.

[BBK98] Stefan Berchtold, Christian Böhm, and Hans-Peter Kriegel. The Pyramid-Tree: Breaking the curse of dimensionality. In *Proc. of the ACM SIGMOD, June 2–4, Seattle, Washington, USA*, pages 142–153, 1998.

[BGW91] J. Bigün, G. H. Granlund, and J. Wiklund. Multidimensional orientation estimation with applications to texture analysis and optical flow. IEEE *Transactions on Pattern Analysis and Machine Intelligence*, 13(8):775–790, August 1991.

[BKK96] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-tree : An index structure for high-dimensional data. In *Proc. of the 22nd VLDB, September 3–6, Mumbai (Bombay), India*, pages 28–39, 1996.

[DSH00] Y. Dufournaud, C. Schmid, and R. Horaud. Matching images with different resolutions. In *Proceedings of the Conference on Computer Vision and Pattern Recognition, Hilton Head Island, South Carolina, USA*, volume 1, pages 612–618, June 2000.

[FBF+94] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3:231–262, 1994.

[FtHRKV94] L.M.T. Florack, B. ter Haar Romeny, J.J Koenderink, and M.A. Viergever. General intensity transformation and differential invariants. *Journal of Mathematical Imaging and Vision*, 4(2):171–187, 1994.

[Gut84] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. of the ACM SIGMOD, Boston, Massachusetts, June 18–21*, pages 47–57, 1984.

[Hen98] Andreas Henrich. The LSD^h-tree: An access structure for feature vectors. In *Proc. of the 14th ICDE, February 23–27, Orlando, Florida, USA*, pages 362–369, 1998.

[HK99] Alexander Hinneburg and Daniel A. Keim. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In *Proc. of the 25th VLDB, September 7–10, Edinburgh, Scotland, UK*, pages 506–517, 1999.

[HKM+97] J. Huang, S. Ravi Kumar, M. Mitra, W.J. Zhu, and R. Zabih. Image indexing using color correlograms. In *Proceedings of the Conference on Computer Vision and Pattern Recognition, Puerto Rico, USA*, pages 762–768, June 1997.

[HS88] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, pages 147–151, 1988.

[KS97] Norio Katayama and Shin'ichi Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *Proc. of the ACM SIGMOD, May 13–15, Tucson, Arizona, USA*, pages 369–380, 1997.

[NHS84] Jürg Nievergelt, Hans Hinterberger, and Kenneth C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Transactions On Database Systems*, 9(1):38–71, 1984.

[NRS99] Apostol Natsev, Rajeev Rastogi, and Kyuseok Shim. Walrus: A similarity retrieval algorithm for image databases. In *SIGMOD 1999, Proc. of the ACM SIGMOD, June 1-3, 1999, Philadephia, Pennsylvania, USA*, pages 395–406, 1999.

[Poy97] C.A. Poynton. Frequently asked questions about color, 1997.

[Rob81] John T. Robinson. The K-D-B-Tree: A search structure for large multidimensional dynamic indexes. In *Proc. of the ACM SIGMOD, Ann Arbor, Michigan, April 29–May 1*, pages 10–18, 1981.

[SM97] C. Schmid and R. Mohr. Local grayvalue invariants for image retrieval. IEEE *Transactions on Pattern Analysis and Machine Intelligence*, 19(5):530–534, May 1997.

[SMB98] C. Schmid, R. Mohr, and Ch. Bauckhage. Comparing and evaluating interest points. In *Proceedings of the 6th International Conference on Computer Vision, Bombay, India*, pages 230–235. IEEE Computer Society Press, January 1998.

[SS94] M. Stricker and M. Swain. The capacity of color histogram indexing. In *Proceedings of the Conference on Computer Vision and Pattern Recognition, Seattle, Washington, USA*, 1994.

[WJ96] David A. White and Ramesh Jain. Similarity indexing with the SS-tree. In *Proc. of the 12th ICDE, February 26–March 1, New Orleans, Louisiana*, pages 516–523, 1996.

[WSB98] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. of the 24th VLDB, August 24–27, New York City, New York, USA*, pages 194–205, 1998.

Contents

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399