Storage and Management of Similar Images*

Geneviève Jomier, Maude Manouvrier
LAMSADE - University Paris-Dauphine
Place du Mar. de Lattre de Tassigny
75775 Paris Cedex 16 - France
Fax. (33 1) 44 05 40 91
e-mail: {*jomier,manouvrier*}*@lamsade.dauphine.fr*

Marta Rukoz§
University of Central Florida
School of Computer Science - P.O. Box 162362
Orlando Fl 32816-2362 - USA
Fax. (407) 823-5419
e-mail: *mrukoz@flynn.ciens.ucv.ve*

Abstract *Numerical images are becoming more and more important and an increasing emphasis on multimedia applications has resulted in large volumes of images. However, images need a large memory space to be stored, so their efficient storage and retrieval generate challenges to the database community. This paper proposes a new algorithm for an efficient storage of sets of images. It is based on a version approach used in databases. It shows how to store and operate on similar images; two images are defined as similar if the quad-trees encoding them have only few different nodes. A data structure called Generic Quad-Tree (GQT) is proposed. It optimizes the memory space required to store similar images and allows an efficient navigation among them. An Image Tree stores the ancestors and descendants of an image, like a version hierarchy. Using the Image Tree, the Generic Quad-Tree allows an image to share common parts with its ancestors and descendants. The GQT approach and some algorithms for reading, modifying or removing images from the Generic Quad-Tree are described. Examples using black and white images and gray scale images are presented.*

Keywords: *Generic Quad-Tree, operations on quad-trees, image representation, image comparison, optimization of memory space, image processing application.*

1. Introduction

It is commonly required in image processing and management systems to store different similar images representing the same "reality". Measures on image similarity have been proposed on color [20,6,26,27], texture [33,34,32] or shape [28,8,9]. These similarity measures are mainly used in image databases to retrieve the images which are the most similar to an image example.

In this paper, a new measure is defined on image similarity. It is based on a distance computed from the differences between the quad-trees encoding images. This measure on image similarity, called *Q-similarity*, is defined in order to optimize the memory space required to store images. Such similar images appear in image processing systems [1], where new images are generated as the result of an operation or a sequence of operations applied on an initial image where only few areas are modified [14]. In Geographic Information Systems [32], similar images occur when pictures of an area are taken from time to time or at fixed dates to follow its global evolution, or the evolution of some of its parts. In medical domain, images representing the same kind of pathology may be considered as similar [28]. For instance [25] shows the result of retrieval by similarity of brain images obtained using MRI (Magnetic Resonance Image).

Image storage and retrieval are big challenges because of the increasing number of images, particularly in medical domain. For instance, a typical radiology department currently generates between 100,000 and 10,000,000 images per year, requiring about 1.5 terabytes [16]. As the size of an image is large or very large [10], in kilo or megabytes, and as the number of images to be managed may be large too, various processes of compression and compacting have been proposed. They use different kinds of coding [17,36,37] or wavelets [7,21]. As opposed to these approaches where the compression or compacting process is applied to each image separately, the data structure proposed in this paper and called Generic Quad-Tree (GQT) [11], uses Q-similarity between images to improve their

*This work was supported by the CNRS in France and by CONICIT (accord numbers 5485 and 7202) and CDCH in Venezuela.

§On leave from Centro de Computación Paralela y Distribuida - Fac. de Ciencias - Universidad Central de Venezuela.

storage. The Generic Quad-Tree considers images organized in quad-trees. A quad-tree is an efficient data structure used for representing 2D images [29,30]. The GQT approach allows an efficient memory space optimization by sharing common parts of images. It allows operations on similar images, like comparison of images, or comparison of the same region in different images, for instance to follow consequences of processing on different images or on the same area in different images. Moreover, the set of images is not supposed to be organized in a total order.

This paper is organized in the following way. Section 2 describes a medical application, where the present work takes place and briefly recalls the principles of quad-trees for image representation. Section 3 describes the principles of the Generic Quad-Tree approach. Section 4 describes the primitive operations on images stored in a Generic Quad-Tree. Section 5 presents examples of Generic Quad-Tree implementations. Section 6 compares this approach with others proposals. Section 7 concludes and gives some research directions to go farther.

2. Work context

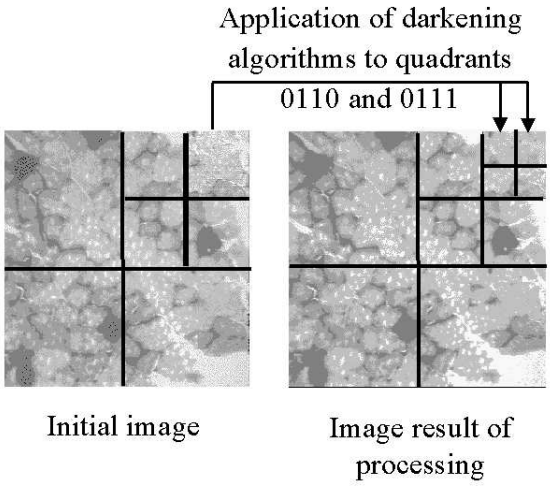

Figure 1: Modification of an image area.

This paper proposes a structure to manage and to store images commonly occurring in image processing applications. To be more precise, let us describe a medical application, which is used as a reference in the SIMA project [5] [18]. The purpose of the processing is to "improve" images according to criteria predefined by biologists. It comes down to make some elements appear more distinctly, to show some characteristics or salient features, or emphasize differences. Figures 1 to 4 present examples of image processing operations. The quadrants cut up in images are recursively identified according to a Z function as shown in figure 5.

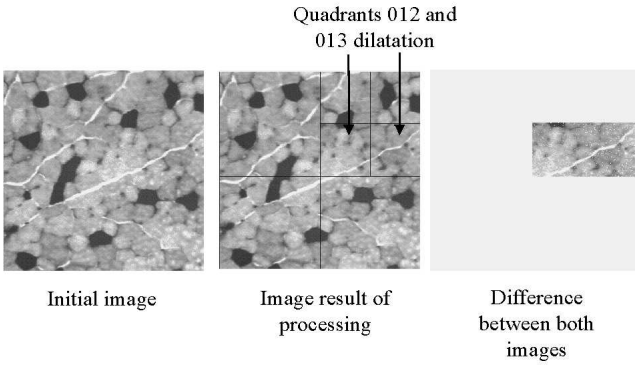

Figure 2: Example of image area dilatation.

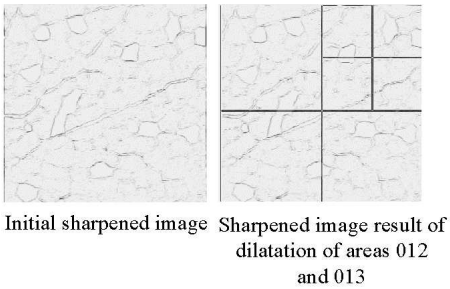

Figure 3: Sharpened images and areas dilatation.

Figure 1 presents an image (on the left) containing cells. In the right area of this image (at the beginning of the arrow), biologists cannot correctly determine the presence of cells. In order to improve the quality of this area, the quadrant is cut in four parts. Each sub-quadrant is differently darkened.

Figures 2, 3 and 4 represent different steps of an image

[5] SIMA Project number S1-9500710 CONICIT Venezuela.

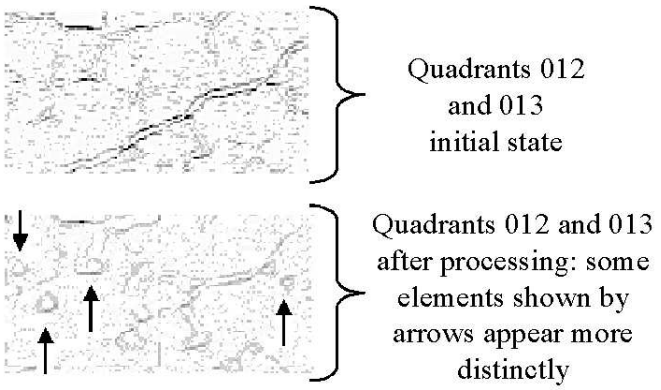

Figure 4: Objects detection after area dilatation in sharpened images of figure 3.

processing. First, the image is recursively subdivided in four quadrants (see figure 2). A dilatation [31] is applied to areas identified by 012 and 013, in order to close irregular cell shapes. Then, a sharpening algorithm is applied to the image. A zoom on areas 012 and 013 allows biologists to see more distinctly some elements (see arrows on figure 4) which are not visible in the initial image.

In order to succeed, biologists work in a trial and error process, using operations which generate a new image as a result. These operations can be applied to the whole image or to a well chosen part of it (see figures 1-2) because an operation can improve some parts of an image and damage some others. Thus it is better to improve locally some parts of an image and to recompose the entire image using its improved parts. Moreover, it is common to modify some image element manually. At the end of the process of "improving" an image, a fair number of images have been created, specially when different experts have been processing in parallel from the same initial image. Throughout the process, the created images must be saved in order to allow starting again from an intermediate image if the result of a sequence of operations is satisfying, or if trying another sequence of operations could generate a better result.

Biologists use a quad-tree to cut images in regions (or quadrants). A quad-tree is a hierarchical structure built by recursive divisions of the space in four disjoint quadrants. H. Samet [29] gives a detailed description of this structure. Quad-trees are used for different types of data, like curves, surfaces or volumes [30]. The most widely known quad-tree allows to cut an image in regions (or quadrants) according to a criterion (see figure 5). An image is recursively cut in four disjoint quadrants or squares of the same size so that a node of the quad-tree represents each quadrant. The root node represents the initial quadrant containing the whole image. If an image is not homogeneous - according to the criterion -, the quad-tree root has four son nodes representing the four first level image quadrants: northwestern, northeastern, southwestern and southeastern. A node is a leaf when its corresponding image quadrant is homogeneous; otherwise the node is internal. As a consequence, the leaf nodes of a quad-tree are not all at the same level. Each internal node has exactly four sons.

Different functions are used for associating an identifier with a quadrant [29]. Here we use a Z function, as shown on figure 5. A quad-tree node uses the identifier of the quadrant it represents. For example, in figure 5, the number 0 identifies the initial quadrant representing the whole image. Numbers 0, 1, 2 or 3, following their parent node identifier 0, identify the four first level image quadrants. Recursively, subquadrants of an image quadrant n are identified by nk where $k \in [0, 3]$.

Quad-trees can be implemented using pointers to nodes. This kind of implementation, called hierarchical, is costly in memory space [35]. To avoid this problem, different techniques of linear storage of quad-tree have been proposed. A linear representation of a quad-tree is a list of values, which saves the hierarchical tree structure. It is generally used for black and white images. Node values, and particularly leaf node values, are encoded following depth-first order [13] or

width-first order [17].

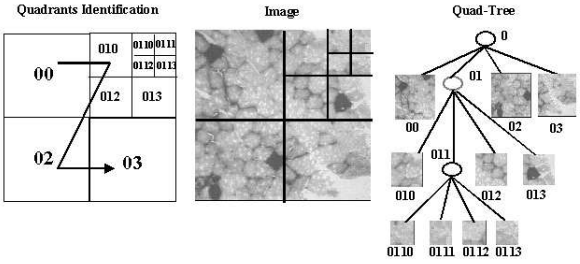

Figure 5: An example of image quad-tree.

3. The Generic Quad-Tree approach

A Generic Quad-Tree (GQT) is a new tool for representing and managing similar images [11,12]. It is based on the Database Version Approach [5]. This structure minimizes the memory space of a set of images and speeds up several operations applied to images like image comparison. In fact, GQT is more than a storage structure. It is designed to be inserted in an environment for image management and processing [19,18], because it allows user to extract one or several images easily and to work on them using his/her image processing tools in his/her working environment. He/she can modify preexisting images, insert new images or delete some of them, compare images, extract images, to build image sequences, etc.

In this section and in the following one, the Generic Quad-Tree approach is presented in a simple particular case where each point has only two values: *black* and *white*. This choice of simplification is to make clear the description of the approach. However, it may be applied to any other case of storage of set of images using quad-trees.

The GQT approach is based on a principle of sharing parts of image quad-trees, which is presented in subsection 3.1. The similarity between images, used in this paper, is defined in subsection 3.2. The images of the database are organized in a special structure defined in subsection 3.3. Then, the Generic Quad-Tree is presented in subsection 3.4 and the end user view of images is described in subsection 3.5.

Below, when no confusion is possible, image and image identifier are indifferently used.

3.1. Sharing parts of image quad-trees

The Generic Quad-Tree approach is based on a principle of sharing of quadrant values between images. Let S be a set of images. If a quadrant q has the same value in a set of images $S' \subset S$, this value is stored only once and is associated with the set of image identifiers of S'. In that case the sharing is called *explicit*, because the identifier of each image sharing the value is explicitly present in the list.

If a tree order is introduced in the set of images S, each image, except the tree root, has a unique parent and an indefinite number of children. Thus the following rule of *implicit sharing* may be introduced: *except if the identifier of an image i is explicitly associated with another value v, image i shares the value with its parent image.*

If the image tree is stored, this implicit rule of sharing allows a very compact representation of a set of images when a large number of images share quadrant values along the Image Tree branches

3.2. Similarity between images

The *Q-similarity* between two images is defined as the number of nodes having the same identifier but different values, divided by the cardinal of the union of their nodes.

More formally, let $S(i,i')$ be the set of nodes having the same identifier but different values in quad-trees of images i and i'. Let $U(i,i')$ be the union of the set of node identifiers existing in the quad-tree of image i and the set of nodes identifiers existing in the quad-tree of image i'. Let $Card(S(i,i'))$ (resp. $Card(U(i,i'))$) be the cardinal of $S(i,i')$ (resp. of $U(i,i')$), $Card(U(i,i')) \neq 0$. The Q-similarity between images i and i', $d(i,i')$, is computed according to equation 1.

$$d(i,i') = \frac{Card(S(i,i'))}{Card(U(i,i'))} \quad (1)$$

- $d(i,i') \in [0,1]$, it is a distance [23].

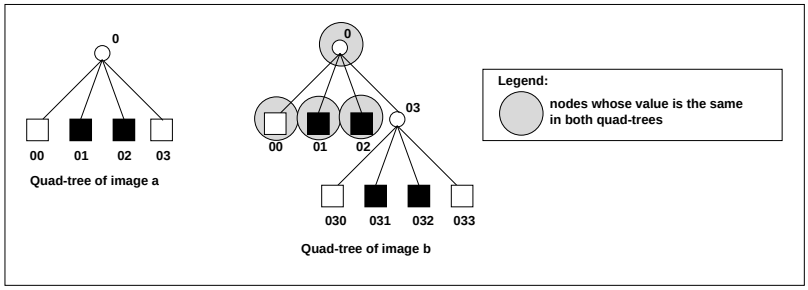

Figure 6: The Q-similarity computation between images a and b.

- $d(i, i') = 0 \iff$ image i and image i' share all their nodes.
- $d(i, i') = 1 \iff$ image i and image i' don't share any node.

For example, the Q-similarity between image a and image b (whose quad-trees are represented in figure 6) is equal to: $d(a, b) = Card(S(a, b))/Card(U(a, b)) = 5/9$

- $Card(S(a, b)) = 5$ because nodes 03, 030 to 033 have the same identifier but a different value in the quad-trees of image a and image b. Nodes 030 to 033 have a special value, *doesn't exist*, in the quad-tree of image a.
- $Card(U(a, b)) = 9$ because nodes 0, 00 to 03 and 030 to 033 appear in the union of node identifiers of both quad-trees.

3.3. Image Tree

Common values of quad-tree nodes are stored once only in the Generic Quad-Tree and are associated with a set of image identifiers. The explicit sharing of node values optimizes the memory space used for image storage. The implicit sharing diminishes the number of image identifiers associated with the stored values.

To use the rule of implicit sharing, explained above in section 3.1, all the images represented using the Generic Quad-Tree approach are organized in a tree structure called *Image Tree*. An image j is inserted in the Image Tree as a child of an image i if the Q-similarity between i and j has the smallest value in the database: $\forall i' \in, i \neq i', d(i, j) \leq d(i', j)$. Image insertion is detailed in section 4.3.

As the Image Tree is only devoted to save memory space, it is ignored by end users. The way end users view the image set is explained below in section 3.5.

Figure 7 represents an Image Tree with four images a, b, c and d.

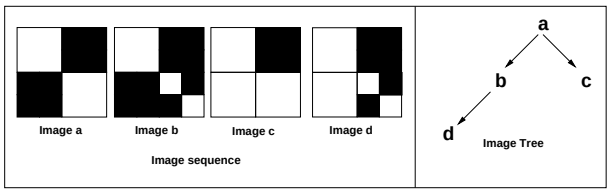

Figure 7: Images are organized in an Image Tree.

3.4. Generic Nodes

Similar images are stored in a *Generic Quad-Tree*. Its nodes are called *generic nodes* or *fat nodes*. For each node appearing in an image quad-tree and representing a quadrant, there is a node with the same identifier in the Generic Quad-Tree. A generic node n represents all nodes n of the quad-trees of images belonging to the database. It contains the whole information necessary to rebuild the value of the node with the same identifier n in each image quad-tree.

Each generic node may be seen as a table with two columns and one or several lines (see figure 8). Each line l of a generic node n contains a list of image identifiers and a value v of quad-tree node. The meaning is: v is the value of node n in each image quad-tree whose identifier i appears in line l (see generic node 02 in figure 8; it contains two lines, while generic node

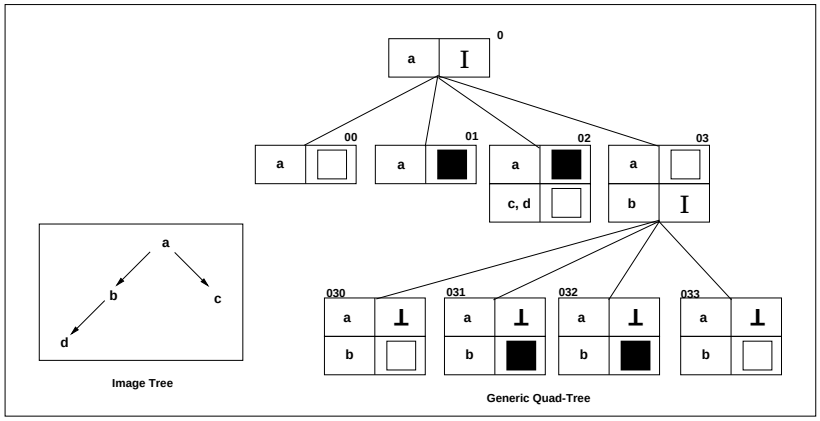

Figure 8: The Image Tree and the Generic Quad-Tree of the images represented in figure 7.

01 has only one line).

Moreover, applying the sharing rule (see section 3.1), all the nodes n of the quad-trees representing images descendant of image i implicitly share the value v; this implicit sharing is stopped by a descendant image identifier appearing in another line of generic node n, i.e. associated with another value v' (see generic node 01 in figure 8, all the images share the value *black*).

The value of a generic node is either $\perp$, meaning the node *does not exist* in quad-trees of images appearing in the corresponding line (see generic node 030 in figure 8), either I, meaning the node is internal - it has four sons - (see generic node 0 in figure 8), either a *black* square if it is a black leaf, either a *white* square if it is a white leaf.

Figure 8 represents the Image Tree and the Generic Quad-Tree of the images represented in figure 7. Generic node 0 contains only one line with value I. This means that all nodes 0 are internal in the quad-trees of images a, b, c and d. Value I is explicitly associated with image a . This value is implicitly associated with images b, c and d, descendants of a in the Image Tree, because they don't appear in any other line of generic node 0. On the other hand, generic node 02 contains two lines. The first line means: node 02 is *black* in the quad-tree of image a. By implicit sharing, as image b is not appearing in any line and is child of image a in the Image Tree, node 02 is *black* in the quad-tree of image b. However, the quad-trees of images c and d do not share this value, because identifiers c and d appear in the second line of generic node 02. The value of node 02 in the quad-trees of images c and d is *white*. Nodes $03k$, $k \in [0,3]$, do not exist in the quad-tree of image a and image c. The value explicitly associated with image a in generic nodes $03k$ is $\perp$. The quad-tree of image c, child of image a in the Image Tree, implicitly shares this value.

3.5. End user view of images

At this point it must be stressed that end users identify images in a way relevant to their applications. The external identification (a, b, c and d on the preceding examples) which is the only one known by end users, is translated to internal identifiers optimizing search through the Generic Quad-Tree. To optimize Generic Quad-Tree algorithms, the internal identification is built in such a way that from the identifier of an image in the Image Tree, all its ancestors can immediately be deduced [5,15].

This point is very important because the physical implementation of similar images is completely separated from the way end users see their images. More precisely, end users can read images and their corresponding quad-trees from the Generic Quad-Tree and order them in the way they want. For instance they can build sequences of images ordered according to specific criteria. As a consequence, it is useful to associate meta-data, containing information on each image: date of creation, author, process of creation, etc., with each image. Image meta-data are used for image retrieval in several contexts, for instance, digital libraries.

4. Operations on images

The primitive operations on images stored in a Generic Quad-Tree are:

1. Image reading,
2. Image modification,
3. Image insertion,
4. Deletion of an image,
5. Comparison, union and intersection of images.

4.1. Image reading

To read an image i from a Generic Quad-Tree, the Generic Quad-Tree is read from its generic root node, identified by 0. The value of this node for image quad-tree i is determined according to the sharing rule presented in section 3.1. If the value of a node n is I for image quad-tree i, the node is internal in the quad-tree of i and the generic nodes, children of n, are read. Otherwise, the node n is a leaf of image quad-tree i and the value found out is its color.

For example, the following steps describe the reading of image c in the Generic Quad-Tree of figure 8:

- Generic root node 0 is read. Image c identifier does not appear in node 0. Thus image c shares the value of node 0 with image a, parent of c in the Image Tree. Value I, meaning internal node, is associated with image a. As a consequence, node 0 is internal in the quad-tree of c. The four nodes, children of node 0, are read (see following items).
- Implicit sharing appears in nodes 00 and 01. The system deduces, using the sharing rule, that nodes 00 and 01 are respectively *white* and *black* in the quad-tree of c.
- The value *white* explicitly associated with c in generic node 02 corresponds with the value of node 02 in the quad-tree of c.
- Finally, image c and its parent image a implicitly share the value *white* of generic node 03.

4.2. Image modification

Modifying an image updates its quad-tree. Two cases occur, illustrated by the modifications of image b in figure 9 compared to figure 7.

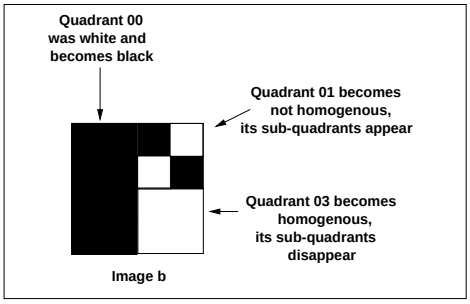

Figure 9: Modification of image b.

1. The modification preserves the image quad-tree structure and only values of leaf nodes are changed. For example in figure 9, quadrant 00 of image b is changed from *white* (see figure 7) to *black* (see figure 9). As a consequence leaf node 00 is modified in the quad-tree of b.
2. The modification alters the image quad-tree structure:
   (a) nodes may disappear because quadrants which were not homogeneous become homogeneous. For example nodes $03k$, $k \in [0,3]$, disappear in the quad-tree of b, because quadrant 03 in image b becomes homogeneous.
   (b) or nodes may appear because quadrants, which were previously homogeneous, are no more homogeneous. For example nodes $01k$, $k \in [0,3]$, appear in the quad-tree of image b, because an homogeneous quadrant (01) becomes non homogeneous.

All cases may occur simultaneously, for different nodes of the original image.

4.2.1. Value modification of a quad-tree node

Performing a value modification of quad-tree node n generates changes in the Generic Quad-Tree. The new value $v.new$ of node n for the modified image quad-tree i must be implicitly or explicitly associated with i. However, to avoid the propagation of the modification to the possible descendant images of i, the implicit sharing between i and its image children, in the Image Tree, must be cut. Thus each image child of i, not explicitly associated with a value in generic node n before the modification, must be explicitly associated with the old value $v.old$ that it was implicitly sharing with i before the modification.

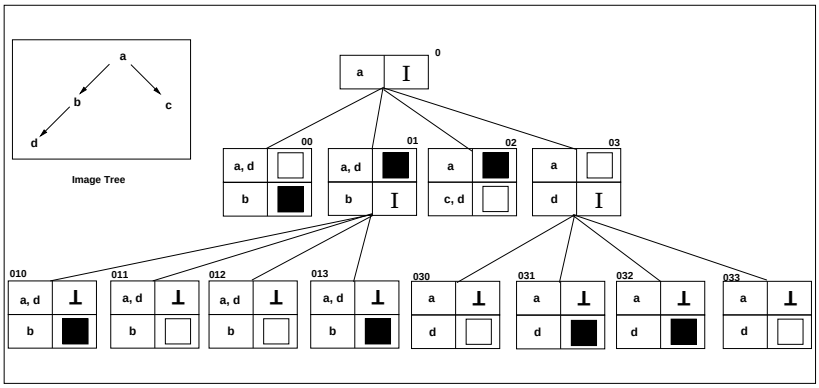

Figure 10: The Generic Quad-Tree after image b modification.

For instance, before the modification of image b, image b and image a were implicitly sharing the value *white* of generic node 00 (see figure 8). After the modification of image b (see figure 9), the value *black* of generic node 00 is explicitly associated with image b in figure 10. Image d, child of b in the Image Tree, is explicitly associated with value *white*, because it was the value of node 00 in the quad-tree of b before the modification and because the quad-trees of images b and d were implicitly sharing this value.

4.2.2. Deletion of quad-tree nodes

When a modification implies the deletion of nodes (at least four: $n0$, $n1$, $n2$ and $n3$) in an image quad-tree i, the corresponding generic nodes must be updated for image quad-tree i, and only for i, by value $\perp$, meaning *does not exist*. In order not to modify the images descendant of i, image i and its children must not implicitly share the value $\perp$. This operation is performed exactly as explained in section 4.2.1, above. If a generic node $n.k$ ($k = 0..3$) contains only the value $\perp$, this node does not exist anymore in any image quad-tree. As a consequence it may be deleted from the Generic Quad-Tree. The deletion of four nodes, which are brothers identified by $n0$, $n1$, $n2$ and $n3$, in a quad-tree entails a modification of their parent node n: it was internal and becomes a leaf. As a consequence the generic parent node must be updated.

For example, node 03, which was internal (see figure 8) in the quad-tree of image b becomes *white* (see figure 10). In the Generic Quad-Tree, image b is removed from the image identifiers list associated with value I of generic node 03. Thus, image d, child of b in the Image Tree, is explicitly associated with value I of generic node 03. Then, image b identifier is associated with value *white* of generic node 03. As this value exists in generic node 03 and is associated with image a, parent of b in the Image Tree, image b implicitly shares value *white* with image a. After the modification of node 03, nodes $03k$, $k \in [0, 3]$, are removed from the quad-tree of image b. As a consequence, image b implicitly shares the value $\perp$ of generic nodes $03k$ with image a, because image a is explicitly associated with value $\perp$ of generic nodes $03k$ and b is the child of a in the Image Tree. Finally, image d is explicitly associated with the *black* and *white* values of nodes $03k$ before the updating of b.

4.2.3. Creation of quad-tree nodes

Finally, new nodes of image quad-tree (at least four: $n0$, $n1$, $n2$ and $n3$) may appear after updating image i. This operation updates the value of the parent node n, which becomes internal. The value I of the generic parent node n is associated with image i, and only with i (i.e. no side effect on other images descendant of i). If image i and its children in the Image Tree were implicitly sharing a value in the generic parent node n, this implicit sharing must disappear. For each new node $n.k$ ($k \in [0, 3]$) created in the quad-tree of image i, a new generic node $n.k$ is created in the Generic Quad-Tree, if it doesn't exist. First, this node is created empty: value $\perp$ is associated with the root of the Image Tree. Then, the generic node $n.k$ is updated and the corresponding value of node $n.k$ in the quad-tree of i is associated with i in generic node $n.k$ (see section 4.2.1).

This situation is illustrated in figure 10. Nodes $01k$, $k \in [0, 3]$, are created in the quad-tree of b, because region 01 is not homogeneous any longer (see figures

7 and 9). The corresponding generic nodes $01k$ do not exist in the Generic Quad-Tree. They are created with value $\perp$ associated with image a, root of the Image Tree. Nodes $01k$ are updated in order to associate the value *white* or *black* with image b. Image d, child of b in the Image Tree, is explicitly associated with value $\perp$ in generic nodes $01k$, because these nodes do not exist in the quad-tree of image d. A new line is added in generic node 01. This line associates value I with image b. Image d, child of b in the Image Tree, is explicitly associated with the value *black* of generic node 01, because it was the value of node 01 in the quad-tree of b before the updating.

4.2.4. Other modifications

After all these examples, it appears easy to perform other operations by modifications of generic nodes :

1. Modification of a part of an image by copying the corresponding part of another image. In fact copying corresponds to two simultaneous updates.

2. Performing the same modification simultaneously in several images. For instance, it is easy to propagate a modification of node n performed on image i to all images descendant of i. This operation consists in modifying the value of generic node n associated with image i, and in deleting the identifiers of the descendants of image i in generic node n. Thus all the images descendant of i implicitly share the value of node n with image i.

Modifying an image i changes the implicit sharing between i and its descendants. After many image modifications, it may happen that the implicit sharing between images deceases. A reorganization of the Image Tree may improve it. Algorithms for Image Tree reorganization are detailed in [23].

4.3. Image insertion

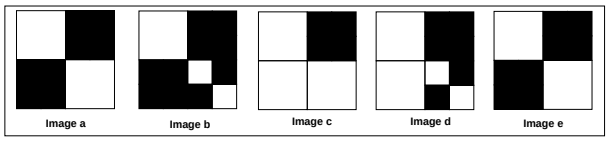

Figure 11: Creation of image e as a logical copy of image a.

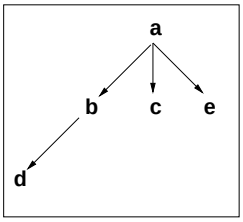

Figure 12: Image Tree modification after the insertion of image e.

Two modes of insertion of a new image in a Generic Quad-Tree are provided, according to users needs. The first mode corresponds to the creation of a new image by modification of a preexisting image without destroying that one. The other mode is the insertion of a new image created outside the Generic Quad-Tree, e.g. by an image-processing tool.

For the first case, users are provided with an operation which creates a new image $x.i$ by logical copy of a preexisting image x. The new image $x.i$ is inserted as the ith child of x in the Image Tree. As it is a copy of x, it has exactly the same content and it shares the value of all its quad-tree nodes with x. As a consequence, no updates of Generic Quad-Tree nodes are required; this is the reason why this operation is called *logical copy*. This case of logical copy of image is illustrated on figure 11: a new image e is created as a child of image a in the Image Tree. Figure 12 represents the Image Tree modification. The Generic Quad-Tree does not change and is still the one represented in figure 10.

In the second case, a new image i' built outside the Generic Quad-Tree must be inserted. It may be produced by an external source or as the result of an image processing operation performed on an image extracted from the Generic Quad-Tree. The quad-tree of i' is built. It is easy to compare the value of each quad-tree node n of image i' with the corresponding generic node n and to detect the node values that image i' and other images will share. If the place of i' in the Image Tree is not well chosen, the sharing will mainly be explicit. To improve the implicit sharing, i' must be inserted in the Image Tree as a child of an image sharing the maximum number of node values with i'. For that, for each node n of i', the list of all images sharing the value of n with i' is built using the information contained in generic node n. The set of images,

sharing the maximum number of node values with i', is determined. The image i parent of i' is chosen in this set and i' is inserted as a leaf in the image Tree. Then the Image Tree and the Generic Quad-Tree are updated to introduce if necessary values of nodes of image quad-tree i' ; They can be unshared, explicitly shared, or implicitly shared with i. Algorithms for insertion of external image are detailed in [23].

4.4. Image deletion

The deletion of an image can be logical and physical.

When an image is logically deleted it does not appear any longer to end users. So it does not appear any longer in the set of images and in the Image Tree where the corresponding node of the image quad-tree is no longer visible to users. A deletion indicator, associated with each image in the Image Tree, is used to perform this deletion. When the indicator of i is positioned, i is logically deleted. Nothing else is done. Due to the sharing mechanism, the Generic Quad-Tree remains unchanged: the references to a deleted image in generic nodes still exist, hidden to users. The Image Tree is preserved.

Another step in deletion of image consists in deleting the value corresponding to i in all the generic nodes, if this value is not shared with any other image. This is useful when the deleted values are large on the point of view of memory space.

Another step consists in deleting all explicit or implicit references to i in the Generic Quad-Tree. This is performed by changing to explicit sharing all the cases of implicit sharing between i and its image children and then in deleting the identifier i from all the generic nodes.

The last step deletes image i from the Image Tree. Thus the Image Tree structure is modified with consequences on internal image identifiers. Therefore, this physical deletion is only used in case of archiving or destroying a subset of images stored in the Generic Quad-Tree (for instance a sub-tree).

4.5. Other operations on images

Using Generic Quad-Tree, other operations involving several images and very useful for applications, like comparison, can be easily performed.

The comparison of values of an area in any two images i_j and i_k can be performed by a top-down comparison of the values extracted from generic nodes. It begins at the generic node of the smallest quadrant including the considered area. Similarly the comparison of any two images can be performed.

It is also interesting to extract the regions which are common to several images or which are different. If, from the user point of view, the compared images are ordered according to their validity time, this kind of comparison is useful to study spatio-temporal evolution. This is a hot topic in Geographic Information Systems.

Using the technique of generic node modification illustrated in 4.2 above, and provided that these operations are defined for the values stored in the Generic Quad-Tree, it is easy to compute inside the Generic Quad-Tree an image whose content is the result of:

1. the complementary of an image or a part of an image,
2. the union, intersection or difference of any two images, or of the same region in any two images.

5. Examples of GQT implementations

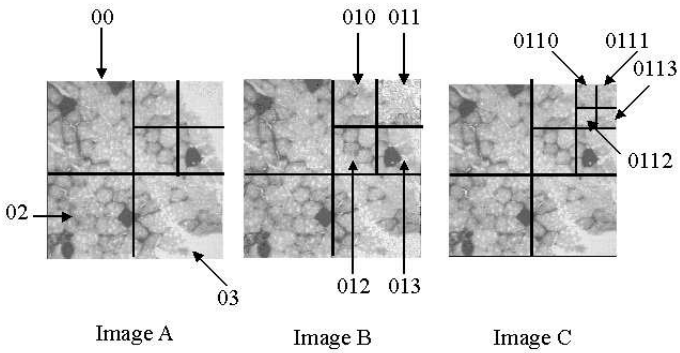

Figure 13: An example of three processed images.

Until now in this paper we have considered that the only constraint on quad-trees is homogeneity of the leaf quadrants according to a predefined criterion.

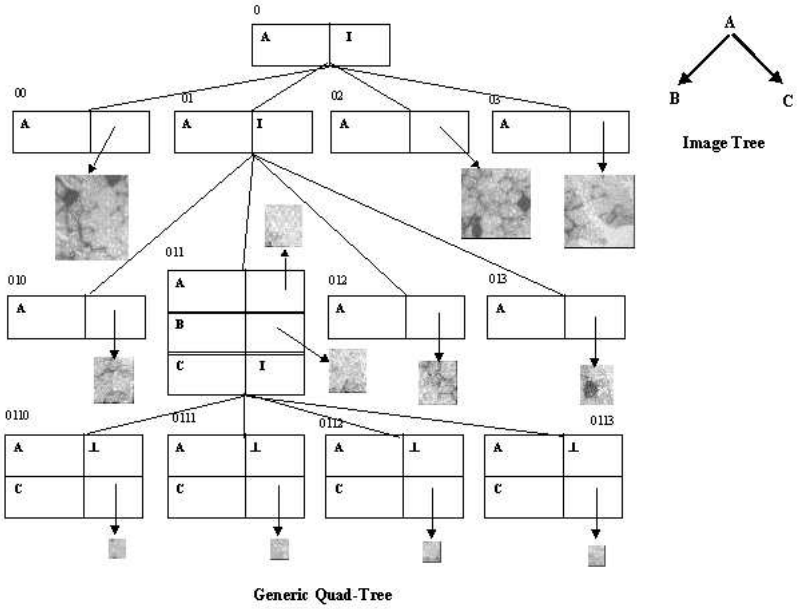

Figure 14: Images of figure 13 stored in a Generic Quad-Tree.

As a consequence, instead of black or white quadrant, gray scale, colors, textures etc. can be used without any change in the algorithms of the Generic Quad-Tree. Moreover, to limit the number of levels in quad-tree divisions a threshold on the quadrant value can be chosen.

In image processing, particularly in medical domain, quad-trees are interesting to delimit image quadrants whose size can be as small as required. For instance, a quadrant may represent an interesting area for a biologist, because it contains specific cells. These quadrants are subject to image operations in order to improve the quality. This situation happens in the medical application of SIMA and is illustrated by figures 13. In order to improve the quality of quadrant 011 in image A two different processes have been carried out starting from image A. Thus, two images, B and C, have been created. The image B has been obtained by application of an image improvement operation on the contrast of quadrant 011 of image A. On the other hand, image C has been obtained by applying darkening algorithms on quadrants 0110 and 0111 of image A. Figure 14 represents the Generic Quad-Tree storing images A, B and C. The quadrants of the images are subject to image operations. Their values, which are smaller images, are too large to be stored inside a generic node. As a consequence the quadrant values are stored outside the Generic Quad-Tree, for instance in files, and the generic nodes contain references to these files, for instance file identifiers.

6. Comparison with other approaches

First, this section rapidly compares the Generic Quad-Tree approach with other approaches using image versions. Then, the Generic Quad-Tree approach is compared in detail to approaches using quad-trees for managing images and for optimizing memory space.

6.1. Approaches using image versions

The authors of [1,14] propose an approach to manage image versions in an image processing system. An image version is defined as the result of processing algorithms on an initial image. Each version of an image i is associated with a specific object containing the history of the image processing of i. All images, all image versions and all operations applied on image versions are stored in a database. Users can reuse an existing image version to apply processing algorithms stored in the database or created outside. They can use an

image history to create new image versions or operate on them. In this approach, each image is considered as a whole and each version corresponds to an entire image. There is no image memory space optimization.

6.2. Overlapping approaches

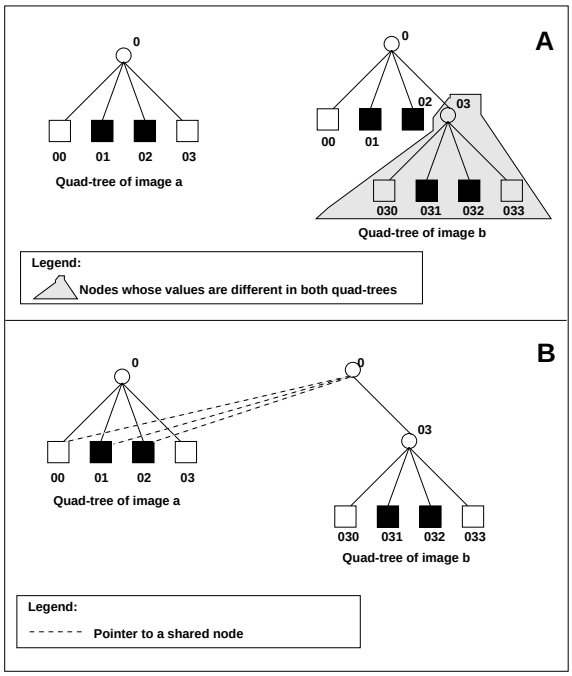

Figure 15: Overlapping between two image quad-trees.

Storage of similar images has been considered in different articles. Several proposals are based on extensions of overlapping between data of the same type [17,36,37]. The purpose of overlapping mechanisms is to share the maximum of common parts between an original data structure and another one [2,3]. This mechanism has been extended to sequences of a given data structure, like B-tree [4,22], R-tree [24] or quad-tree [17,36,37]. The idea is to share the common parts of the quad-trees corresponding to different similar images using overlapping. As a consequence, memory space is saved and the access time to an image is the same with overlapping as without it. In [37], the management of any type of images (binary, colored images, etc.) is allowed. Linear overlapped quad-trees representations are proposed in [17,36], but they only manage black and white images.

In [37], a technique of overlapping to represent sequences of similar images using quad-trees is proposed. When a new image i is inserted at the end of a sequence, its quad-tree overlaps the quad-tree of the preceding image, called $i-1$. The quad-trees representing image i and image $i-1$ share all parts which are equal in both quad-trees. On the contrary, when a leaf node has different values in both quad-trees, all the nodes from the root to the modified node are copied in the quad-tree of image i. This is shown on figure 15.B. The different parts of quad-trees representing both images a (in the left of the figure 15.A) and b (in the right) are represented in gray in the quad-tree of b. Both quad-trees share nodes 00 to 02, because they appear with the same value in both of them. Dotted lines represent references to shared nodes. On the other hand, node 03 has a different value in both quad-trees. Then a new sub-tree is created in the quad-tree representing image b.

The sharing of common parts of quad-trees saves memory space. In the overlapping methods the set of images is organized in sequence. The sharing of parts of a quad-tree is always implicit and is only possible with the quad-tree of the previous image in the sequence. On the contrary, in the Generic Quad-Tree, when the same value appears in the node n of any two different image quad-trees, it is automatically shared, implicitly or explicitly. Therefore the Generic Quad-Tree approach maximizes the sharing of quadrant values.

Image modification has not been considered by authors [37,17,36]. The only operations allowed on images stored in overlapped quad-trees are reading an image and inserting an image at the end of the sequence. Moreover, it is possible to modify or delete the last image of the sequence. On the contrary, for the images stored in a Generic Quad-Tree, there is no limitation on operations allowed: a new image may be inserted as the child of any image in the Image Tree and any image can be modified or deleted. As a consequence, the sharing of values is, in the worst case, equal for the Generic Quad-Tree and the overlapping approaches, and generally better for the Generic Quad-Tree.

The most important difference between the Generic Quad-Tree approach and the overlapping quad-tree methods is that Generic Quad-Tree allows many operations which do not exist for the others: updating an image, comparing images or part of images, fol-

lowing the evolution of an area across images, etc., as explained in 4. A consequence of this peculiarity is that it is efficient for working on one image and for operating simultaneously on any set of images. Thus, for the kind of application, presented in section 2, the Generic Quad-Tree is the most adapted structure.

7. DISCUSSION AND CONCLUSION

The Generic Quad-Tree, presented in this paper, optimizes the storage of similar images organized in quad-tree, avoiding to store several times a part common to several images. The Generic Quad-Tree allows applying operations to images: image reading, image insertion or suppression, modification of an image or several images simultaneously, and image comparison. This type of operation is very important in the domain of image processing.

In the GQT approach, operations are applied on images organized in quad-tree. Quad-tree offers an absolute reference of image regions and allows parallel executions of some operations. It is also possible to visualize different consequences of an image processing in the same regions. In the same way, this type of operation is important in the spatio-temporal domain, to study the evolution of a geographical area.

Among the representations of similar images using quad-trees, the Generic Quad-Tree is specific because each generic node may be used according to two dimensions. One dimension allows navigating inside the quad-tree of an image i: thus it is hierarchical in that sense. The other dimension, across images, allows jumping from one image to another using generic nodes. Thus it is easy, using the Generic Quad-Tree to access simultaneously to the same area in different images, for instance to compare their value or to study the area evolution. Because of sharing the comparison of node n value in different image quad-trees is immediate.

A prototype validating the GQT approach has been implemented at the University Central of Venezuela [12]. Work is currently in progress on the building of accurate indexes of similarity between images according to other definitions of similarity [20,27,34,32] used in image retrieval domain. Mechanisms proposed in the GQT approach have been extended to other data structure (B+-tree, R-Tree) in [23], in order to enlarge the application domain of the Generic Quad-Tree.

REFERENCES

1. M. Aritsugi, M. Tabata, H. Fukatsu, Y. Kanamori, and Y. Funyu. Manipulation of Image Objects and Their Versions under CORBA Environment. In *DEXA '97*, pages 86–91, Toulouse, France, 1997.
2. F.W. Burton, M.M. Huntbach, and J.Y.G. Kollias. Multiple Generation Text Files Using Overlapping Tree Structures. *The Computer Journal*, 28(4):414–416, Aug. 1985.
3. F.W. Burton, J.G. Kollias, D.G. Matsakis, and V.G. Kollias. Implementation of overlapping B-trees for time and space efficient representation of collections of similar files. *The Computer Journal*, 33(3):279–280, June 1990.
4. M.J. Carey, D.J. DeWitt, J.E. Richardson, and E.J. Shekita. *Object-Oriented Concepts, Databases, and Applications*, chapter 14 - Storage Management for Objects in EXODUS, pages 341–369. Addison Wesley - ACM Press, 1989.
5. W. Cellary and G. Jomier. Consistency of Versions in Object-Oriented Databases. In *VLDB*, Brisbane (Australia), 1990.
6. J.M. Corridoni, A. Del Bimbo, and E. Vicario. Painting Retrieval Based on Color Semantics. In *Images Databases and Multi-Media Search*, Series on Software Eng. and Knowledge Eng. (8), pages 13–24. World Scientific, 1997.
7. G.M. Davis. A Wavelet-Based Analysis of Fractal Image Compression. *IEEE Trans. on Image Processing*, 7(2):141–153, Feb. 1998.
8. A. Del Bimbo and P. Pala. Shape Indexing by Multi-scale Representation. In *Images Databases and Multi-Media Search*, Series on Software Eng. and Knowledge Eng. (8), pages 59–74. World Scientific, 1997.
9. C. Djeraba, I. Savory, M. Barere, and S. Marchand. Content Based Image Retrieval Model in an Object Oriented Database. In *Images Databases and Multi-Media Search*, Series on Software Eng. and Knowledge Eng. (8), pages 263–275. World Scientific, 1997.
10. C. Goble. *The Handbook of Multimedia Information Management*, chapter 12 - Images Database Prototype, pages 365–404. Prentice Hall, 1997.
11. G. Jomier, M. Manouvrier, and M. Rukoz. Stockage et gestion d'Images par un Arbre Quaternaire Générique. In *15èmes Journées Bases de Données Avancées (BDA'99)*, Bordeaux (France), 1999.
12. G. Jomier, M. Manouvrier, M. Rukoz, J. Ramirez, and Y. Valero. MIS: Un prototipo de un sistema de Manipulacin de Imágenes Similares. In *XXV Conf. Latinoamericana de Informática (Panel'99)*,

Asunción (Paraguay), 1999.

13. E. Kawagushi and T. Endo. On a method of binary picture representation and its application to data compression. *IEEE Transactions Pattern Anal. Mach. Intell.*, 2(1):27–35, 1980.
14. S. Kawashima, M. Tabata, Y. Kanamori, and Y. Masunaga. Versioning Model of Image Objects for Easy Development of Image Database Applications. In *DEXA '96*, pages 194–200, Zurich, Switzerland, 1996.
15. A. Keller and J. Ullman. A version numbering scheme with a useful lexicographic ordering. In *ICDE*, pages 240–248, Taipei (Taiwan), 1995.
16. F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast nearest neighbor search in medical image databases. In *VLDB*, pages 215–226, Mumbai (Bombay), India, Sept. 1996.
17. Tsong-Wuu Lin. Compressed quadtree representations for storing similar images. *Image and Vision Computing*, 15:833–843, 1997.
18. C. Lon, M. Rivas, and M. Rukoz. Una Herramienta en java par Aplicaciones Distribuidas de Tratamiento de Imgenes Biomdicas. In *Memorias de la XXIV Conferencia Latinoamericana de Informtica (PANEL'98)*, Ecuador, oct. 1998.
19. C. Lon and M. Rukoz. Sistema Distribuido para Tratamiento de Imgenes. In *Memorias de la XXI Conferencia Latinoamericana de Informtica (PANEL'95)*, Brasil, juil. 1995.
20. H. Lu, B-C. Ooi, and K-L. Tan. Efficient Image Retrieval by Color Contents. In *First Int. Conf. on Applications of Database (ADB-94)*, Vadstena (Sweden), June 1994.
21. M.K. Mandal, S. Panchanathan, and T. Aboulnasr. Choice of Wavelets for Image Compression. In *Lecture Notes in Computer Science*, volume 1133, pages 239–249. World Scientific, 1996.
22. Y. Manolopoulos and G. Kapetanakis. Overlapping B+-Trees for Temporal Data. In *Proc. of Jerusalem Conf. on Inf. Technology (JCIT 90) - IEEE Computer Science Press #2078*, pages 491–499, Israël, Oct. 1990.
23. M. Manouvrier. *Objets Similaires de Grande Taille dans les Bases de Données.* PhD thesis, Université Paris IX Dauphine (France), januray 2000.
24. M.A. Nascimento and J.R.O. Silva. Toward Historical R-trees. In *Proc. of ACM Symposium on Applied Computing (SAC'98)*, pages 235–240, Feb. 1998.
25. C. Nastar. Indexation d'Images par le Contenu : un Etat de l'Art. In *COmpression et REpreésentation des Signaux Audiovisuels (CORESA'97)*, France, 1997.
26. N. Nes, C. van den Berg, and M. Kersten. Database Support for Image Retrieval Using Spatial-Color Features. In *Images Databases and Multi-Media Search*, Series on Software Eng. and Knowledge Eng. (8), pages 293–300. World Scientific, 1997.
27. B.C. Ooi, K-L. Tan, T.S. Chua, and W. Hsu. Fast image retrieval using color-spatial information. *The VLDB Journal*, 7:115–128, 1998.
28. E.G.M. Petrakis and C. Faloutsos. Similarity Searching in Medical Image Databases. *IEEE Transactions on Knowledge and Data Engineering*, 9(3), mai 1997.
29. H. Samet. The Quadtree and Related Hierarchical Structures. *Computing Surveys*, 16(2):187–260, 1984.
30. H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison Wesley, 1989.
31. J. Serra. *Image Analysis and Mathematical Morphology, vol-2. Theoretical Advances.* Academic, 1988.
32. G. Sheikholeslami, A. Zhang, and L. Bian. A Multi-Resolution Content-Based Retrieval Approach for Geographic Images. *GeoInformatica*, 3(2):109–139, june 1999.
33. J.R. Smith and S-F. Chang. Quad-Tree Segmentation for Texture-Based Image Query. In *Proc. ACM Multimedia*, San Fransisco, CA. USA, Oct. 1994.
34. J.R. Smith and S-F. Chang. VisualSEEk: a fully automated content-based image query system. *ACM Multimedia '96*, Nov. 1996.
35. I.P. Stewart. Quadtrees: Storage and Scan Conversion. *The Computer Journal*, 29(1):60–75, Feb. 1986.
36. T. Tzouramanis, M. Vassilakopoulos and Y. Manolopoulos. Overlapping Linear Quadtrees: a Spatio-temporal Access Method. In *ACM GIS'98*, Washington D.C., Nov. 1998.
37. M. Vassilakopoulos, Y. Manolopoulos, and K. Economou. Overlapping Quadtrees for the Representation of Similar Images. *Image and Vision Computing*, 11(5):257–262, June 1993.