# A Model of Image Representation and Indexing in Image Database Systems*

G. Sheikholeslami, Wenjie Wang and Aidong Zhang
Department of Computer Science
State University of New York at Buffalo
Buffalo, NY 14260

**Abstract**

Image database systems need to be built to support effective and efficient access to image data on the basis of content. In this process, significant features must first be extracted from image data in their pixel format. These features must then be classified and indexed to assist efficient retrieval of image content. However, the issues central to automatic extraction and indexing of image content remain largely an open problem. In this paper, we investigate effective block-oriented image decomposition structures to be used as a fundamental data model for representing image content in large image databases. A hierarchical structure $S^g$-tree is introduced to decompose database images. Each decomposition of an image segment generates $n$ subsegments of equal-size that are uniformly distributed within the image segment. Our analysis illustrates a special case of $S^g$-tree, nona-tree decomposition, when $n$ equals 9, is an effective and efficient block-oriented decomposition structure to facilitate content-based image retrieval. Using $S^g$-tree structure to represent image content in an image database, flexible access to partial contents of each database image can be provided. Various types of content-based queries and efficient image retrieval can then be supported through novel indexing and searching approaches.

## 1 Introduction

Large image database systems need to be built to support effective and efficient access to image data on the basis of content. In this process, significant features must first be extracted from image data in their pixel format. These features must then be classified and indexed to assist efficient retrieval of image content. Methods for accessing alphanumeric data in traditional databases are, in contrast, well understood. In these databases, efficient data access can be facilitated by $B$-trees [7] and other data structures. However, such traditional approaches may not be appropriate in the context of content-based image data retrieval [1, 6, 22, 14, 8, 11, 2, 24]. In this context, visual queries may be posed via image or pictorial examples. A typical visual query might entail the location of all images in a database that contain a subimage similar to a given query icon. Such a query could, for example, be used in a remotely sensed image setting to find a specific kind of spots in related images.

*This research is supported by Xerox Corporation and NCGIA at Buffalo.

To represent image content in databases, approaches which have drawn much recent attention involve formulating feature vectors of images using the extraction of various features of the images, including color, texture, and shape [21, 17, 24, 2]. Furthermore, to support a visual query of finding all images which contain a subimage similar to a given query icon, each feature vector of a database image must correctly represent its subimage features. Thus, image segmentation is necessary to properly implement such feature-based techniques for searching partial image contents. Effective segmentation isolates the important features of database images. Ideally, the results generated through the process of image recognition and analysis would be used to automatically segment image content. However, as image recognition itself is still in its formative stage, tools are not currently available with which to accurately identify such image content in natural images. The automatic retrieval of images on the basis of content thus poses significant difficulties.

To avoid manual segmentation of images in a large image database, Smith and Chang [20] presented an effective block-oriented approach using quad-tree segmentation [6]. Wavelet transforms were used to extract texture features. In their application of the quad-tree structure, the definition of leaf nodes is slightly changed. Before four children are spawned by each parent, conditions for merging are tested. A distance threshold is computed for each child on the basis of extracted texture features. The distances in the feature space are measured from the parent node to each child. If the distances to all four children fall within the thresholds of the children, a single texture would be declared in the parent node, and no further decomposition is necessary. Otherwise, pair-wise grouping of the children will then be performed. That is, if the distance between two neighboring children falls below the thresholds of both, the children are merged as a single child. The quad-tree decomposition is then iterated on each child. The decomposition approach was demonstrated to be effective.

We believe that block-oriented image decomposition provides a novel representation approach to specifying image content for large image databases. In [25, 15, 16, 19], we introduced another effective block-oriented image decomposition approach, the nona-tree structure, for image segmentation in image database systems. In this image decomposition structure, each decomposition on an image segment produces nine subsegments of equal size rather than four. Similarly to [20], various wavelet transforms were used to extract image features for experimental analysis. Experiments were conducted on Brodatz texture images [4] and on the VisTex collection from MIT media Lab to demonstrate the effectiveness of the nona-tree structure. We have shown that the nona-tree structure offers a better block-oriented tree decomposition approach than the quad-tree decomposition.

In this paper, we extend this work to a full analysis of the block-oriented image decomposition and representation approach to be used to support content-based image retrieval in image database systems. A generalized hierarchical structure $S^g$-tree with $g$ representing the degree of the tree structure is introduced to specify block-oriented decomposition of database images. These decomposition structures offer a fundamental data model for specifying image content in large image databases. In the decomposition process, each decomposition of an image segment generates $n$ subsegments of equal-size that are uniformly distributed within the image segment, where $n = (2^g + 1)^2$. Thus, $S^0$-tree represents quad-tree structure and $S^1$-tree represents nona-tree structure. We will conduct computational analysis on the maximum number of possible segments in various block-oriented tree structures and the minimum overlapping coverage between a query icon and the segments of a database image in instances in which the query icon is similar to a subimage of the database image. We will discuss the indexing and searching approaches that can be applied on the feature spaces constructed from various $S^g$-tree decomposition, and analyze their search efficiency. Using $S^g$-tree structure to represent image content in an image database, flexible access to partial contents of each database image can be provided. Various types of content-based queries and efficient image retrieval can then be supported through novel indexing and searching approaches. Our analysis illustrates that nona-tree decomposition, when $n$ equals 9, is an effective and efficient block-oriented decomposition structure to facilitate content-based image retrieval.

The remainder of this paper is organized as follows. Section 2 introduces the block-oriented decomposition and representation approach for images. A segmentation-tree ($S^g$-tree) is defined. In Section 3, we discuss the properties of the $S^g$-tree structure. Section 4 presents the $TV^+$-tree indexing structure for supporting efficient retrieval of image segments. In Section 5, we provide a discussion on the effectiveness and efficiency of the $S^g$-tree structure for image databases. Concluding remarks are offered in Section 6.

## 2 Block-Oriented Decomposition and Representation

A *quad-tree* is a hierarchical image decomposition structure which can provide quick data access for image retrieval [5]. A quad-tree is based on the principle of recursive decomposition of images. Each decomposition of an image segment produces four equal-sized quadrants. Figure 1 demonstrates the positions of the four quadrants within the decomposed segment. The image decomposition process using a quad-tree structure can be described recursively, with the root representing the entire image and its children representing the decomposed segments; these, in turn, become roots for further segmental decomposition. Each internal node has exactly four children.

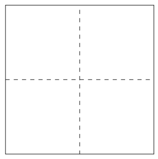


Figure 1: Segments in one step quad-tree decomposition.

A *quin-tree* is a hierarchical image decomposition structure based on a slight modification of the recursive decomposition of images used in quad-trees. Each decomposition of an image segment produces five subsegments of equal size. In addition to the four equal-sized quadrants obtained in quad-tree decomposition, a fifth subsegment equal in size to each quadrant is generated from the central area of the image segment. Figure 2 demonstrates the position of the segment within the decomposed segment. The decomposition process of a quin-tree can also be described recursively, with the root representing the entire image and its children representing the decomposed segments; these, in turn, become roots for further segmental decomposition. Each internal node has at most five children.

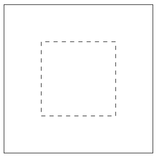


Figure 2: The fifth segment in quin-tree decomposition.

A *nona-tree* is a hierarchical image decomposition structure based on a further modification of the recursive decomposition of images that is proposed in quin-trees. Each decomposition of an image segment produces nine subsegments of equal size. In addition to the five previously mentioned equal-sized segments (four quadrants and one taken from the central area of the image segment), four additional segments, with the same size, are produced from the central areas of the upper, bottom, left, and right halves of the image segment. Figure 3 demonstrates the positions of the four segments within the decomposed segment. The decomposition process of a nona-tree can also be described recursively, with the root representing the entire image and its children representing the decomposed segments; these, in turn, become roots for further segmental decomposition. Each internal node has at most nine children.

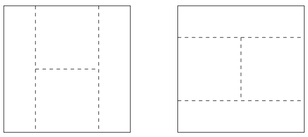


Figure 3: Additional segments in nona-tree decomposition.

The top-left corner location (row, column) of each subsegment $p$ at level 1 in the nona-tree decomposition can be defined as:

$$column = ((p-1)\,\mathbf{mod}3) * \frac{N}{2^2}, \text{ and}$$
$$row = \lfloor \frac{p-1}{3} \rfloor \frac{N}{2^2}, \text{ where } 1 \leq p \leq 9.$$

In general, we introduce a *Segmentation-tree* (denoted $S^g$*-tree*) to be a generalized hierarchical image segmentation structure, where $g$ represents the degree of the tree. $S^g_D$-tree denotes the $S^g$-

tree of image $D$. Given an $S^g$-tree structure, each decomposition of an image segment produces $n = (2^g + 1)^2$ subsegments of equal size ($\frac{1}{4}$ of image segment size) that are uniformly distributed within the image segment. $S^g$-tree is a generalized form of quad- (represented by $S^0$-tree) and nona-tree (represented by $S^1$-tree), not including the quin-tree. Thus, the quad-tree is an $S^g$-tree of degree 0 and the nona-tree is an $S^g$-tree of degree 1. The top-left corner location of subsegments in an $S^g$-tree with $g > 1$ can then be precisely defined as follows.

We consider an image of size $N \times N$ where $N = 2^n$. Size of each segment of $S^g$-tree at level $i$ is the same for different degrees and it is equal to $(\frac{N}{2^i})^2$. We denote the $p^{th}$ segment of $S^g$-tree at level $i$ as $s^g_{ip}$. The row and column coordinates of top-left corner of $s^g_{ip}$ can be computed as follows:

$$column = ((p-1)\mathbf{mod}(2^{i+g} - 2^g + 1))\frac{N}{2^{g+i}},$$
$$row = \lfloor \frac{p-1}{2^{i+g} - 2^g + 1} \rfloor \frac{N}{2^{g+i}},$$

where $0 \leq i \leq L$ with $L$ defining the maximum level of the tree, $0 \leq g \leq n/2$, and $1 \leq p \leq (2^{i+g} - 2^g + 1)^2$.

If an image is of size $2^n \times 2^n$, the highest degree of the $S^g$-tree for the image is $g = \frac{n}{2}$. The maximum level of $S^g$-tree is $L = \lceil log_2 N \rceil$, or $n$.

## 3 $S^g$-tree Properties

This section discusses computational issues of the block-oriented decomposition structures. We will analyze the maximum number of possible segments in block-oriented tree structures and the minimum overlapping coverage between a query image and the segments of a database image in instances in which the query icon is similar to a subimage of the database image.

### 3.1 Number of Tree Segments

As the decomposition of a segment in the quad-tree structure spawns at most 4 segments, we can see that, at each level $i$ of a quad-tree, there are at most $4^i$ possible segments. We now discuss the maximum number of segments in a quin-tree. As in quad-tree decomposition, we can easily claim that, at each level $i$ of a quin-tree, there are at most $5^i$ possible segments. However, there are actually redundant segments in the tree; these start at level two. For example, consider the image given in Figure 4. Let the first level decomposition generate segments 1, 2, 3, 4, and 5, as shown in Figure 4. We now consider the decomposition at level two. The shaded area could appear to be a subsegment of both segments 1 and 2 from the first level of decomposition; it is the fourth subsegment of segment 1 and the first subsegment of segment 2. In fact, among the five subsegments to be generated in segment 2, only subsegment 2 has no redundancy with the subsegments generated in segments 1, 3, 4 and 5. Thus, there are actually 21 rather than $5^2 = 25$

subsegments at the second level of the quin-tree. In other words, there are $25 - 21 = 4$ redundant subsegments.

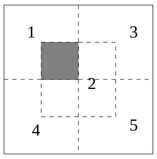



Figure 4: Quin-tree segments.

Let $S_i$ be the number of segments at level $i$. We then have $S_0 = 1$ and $S_1 = 5$. At level $i+1 (i > 1)$, we have

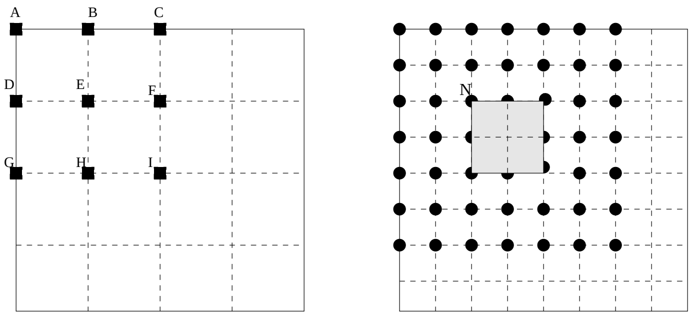
$$S_{i+1} = 4 \times S_i + 1. \tag{1}$$

From Formula 1, we see that, at $i$th level, we have at most $S_i = \frac{4^{i+1}-1}{3}$ rather than $5^i$ subsegments.

We will now discuss the maximum number of segments existing in a general $S^g$-tree with $g \geq 1$. We first informally consider nona-tree ($S^1$-tree). Clearly, at each level $i$, there are at most $9^i$ possible segments. However, there are redundant segments; these start at level two of the nona-tree. We introduce the concept of a *representative point* for each segment. The representative point of a segment in a nona-tree refers to the uppermost left point of that segment. Thus, in Figure 5 (a), labeled bold points $A$ through $I$ are representative points for the nine subsegments of the given segment. $A$, $C$, $G$, and $I$ are the representative points of segments that are in common with the quad-tree. $E$ is the representative point of segment in common with the quin-tree. $B$, $D$, $F$, and $H$ are representative points of segments introduced by the nona-tree. Thus, at level 1, we have $3 \times 3 = (2^2 - 1) \times (2^2 - 1)$ segments, which is 9. Subsegments at the next level are similarly labeled in Figure 5 (b). Note that some of the points represent multiple segments. For example, the point labeled $N$ in Figure 5 (b) represents the shaded area which appears to be a subsegment of segment $A$, $E$, $B$, and $D$ at level 1. This subsegment is the fourth subsegment of segment $A$, the first subsegment of segment $E$, the third subsegment of segment $B$, and the second subsegment of segment $D$. In Figure 5 (b), there are actually $7 \times 7 = 49$ rather than $9^2 = 81$ subsegments at the third level of the nona-tree, which can also be expressed as $(2^3 - 1) \times (2^3 - 1)$. Thus, there are $81 - 49 = 32$ redundant subsegments. By analyzing these numbers, at the $i$th level, we have at most $(2^{i+1} - 1)^2$ instead of $9^i$ subsegments.

In general, we have the following theorem:

**Theorem 1** *The maximum number of non-redundant segments of an $S^g$-tree at each level $i$ ($i \geq 0$) is $(2^{i+g} - 2^g + 1)^2$.*




(a) (b)

Figure 5: (a) Segment decomposition and representative points at level 1; (b) Segment decomposition and representative points at level 2.

**Proof**: (Sketch) Given an $S^g$-tree, at level $i$ ($i \geq 0$), the maximum number of non-redundant segment points in horizontal direction (or vertical direction) can be calculated as:

$$\frac{\frac{2^i-1}{2^i}}{\frac{1}{2^{g+1}} \times (\frac{1}{2})^{i-1}} + 1 = 2^{i+g} - 2^g + 1.$$

Thus, the maximum number of non-redundant segments of $S^g$-tree is $(2^{i+g} - 2^g + 1)^2$. $\square$

Consequently, the total number of non-redundant segments of $S^g$-tree is

$$\sum_{i=0}^{L} (2^{i+g} - 2^g + 1)^2. \tag{2}$$

The increase of segments from $S^{g_1}$-tree to a higher degree tree $S^{g_1+h}$-tree is

$$\frac{\sum_{i=0}^{L} (2^{i+g_1+h} - 2^{g_1+h} + 1)^2}{\sum_{i=0}^{L} (2^{i+g_1} - 2^{g_1} + 1)^2}. \tag{3}$$

Since

$$(2^{i+g_1+h} - 2^{g_1+h} + 1)^2 \leq 2^{2h} * (2^{i+g_1} - 2^{g_1} + 1)^2, \tag{4}$$

we thus have

$$\frac{\sum_{i=0}^{L} (2^{i+g_1+h} - 2^{g_1+h} + 1)^2}{\sum_{i=0}^{L} (2^{i+g_1} - 2^{g_1} + 1)^2} \leq 4^h. \tag{5}$$

Thus, the increase of segments from quad-tree structure to nona-tree structure is at most four times and the increase of segments from nona-tree structure to a higher degree tree $g$ ($g > 1$) is at most $4^{g-1}$ times.

## 3.2 Subimage Coverage

We will now discuss the overlapping coverage between a query icon and the segments of a database image in instances in which the query icon is contained in (or similar to) a subimage of the database image.

Let $Q$ be a query icon and $D$ be a database image. We assume for simplicity that both query icon and database image are square-shaped of size $L \times L$ and $N \times N$, respectively. We now consider a case when query icon $Q$ equals in size to a quadrant in database image $D$. That is, $L = \frac{N}{2}$. Assume that $D$ contains a subimage $q$ which is similar to $Q$ and $q$ may be anywhere in $D$. Ideally, it is the best to have a segmentation mechanism which can isolate $q$ and store it as a separate image feature for retrieval. However, no generic segmentation mechanism exists for decomposing natural images. We now examine the degrees of coverage between the query icon and the segments generated by the block-oriented image decomposition approaches.

Let $D$ be decomposed into four equal-sized quadrants by applying the quad-tree decomposition. The minimum coverage between these four quadrants of $D$ and $Q$ will be $\frac{1}{4}$ of $Q$, when $q$ is located in the center of $D$. Obviously, when $q$ is located in any other places in $D$, the coverage between $Q$ and any quadrant of $D$ will be larger than the $\frac{1}{4}$ of $Q$. However, in practice, such guarantee of overlapping coverage may not be sufficient for the features of any of these four quadrants to be matched with that of the query image. For example, consider the features of images which are generated by calculating various energy values from the wavelet transformed data [26, 13, 10]. In such a scenario, a segment which covers only $\frac{1}{4}$ of the query icon could hardly have similar features to that of the query icon within a given distance threshold. As a result, this database image may fail to be selected.

Let $D$ now be decomposed into five equal-sized segments by applying the quin-tree decomposition. The minimum coverage between these five segments of $D$ and $Q$ will be $\frac{1}{2}$ of $Q$, when $q$ is located at the central areas of the upper, bottom, left, and right halves of $D$. Similarly, when $q$ is located in any other places in $D$, the coverage between $Q$ and the five segments of $D$ will be larger than the $\frac{1}{2}$ of $Q$. Experiments have demonstrated that retrieval effectiveness with the quin-tree is much higher than with the quad-tree, especially when the query icons are found to be similar to subimages located in the central area of the database images [16]. However, when the query icon is similar to a subimage located at a distance from the central area of the database image and very close to the central areas of the upper, bottom, left, or right halves of the database image, this database image is unlikely to be selected.

We now examine the degrees of coverage between the query icon and the segments generated by general $S^g$-trees. Let $2d$ denote the distance between the representative points of neighboring segments at level 1 of $S^g$-tree. For an image of size $2L \times 2L$, $d$ can be calculated as:

$$2d = \frac{1}{2^{g+1}} \times 2L = \frac{1}{2^g} L,$$

$$d = \frac{1}{2^{g+1}} L.$$

We define *covering segment* of a query icon as a segment of $S^g$-tree which has the maximum overlap with the query icon. *Coverage area* of a covering segment is the area in which the representative point of the corresponding query icons are located. If the representative point of a segment $s$ of $S^g$-tree is located at $(x, y)$, and the distance between two neighboring representative points is $2d$, then the coverage area of segment $s$ is the region $(X1, Y1) - (X2, Y2)$ (that is the rectangle whose top-left corner is $(X1, Y1)$, and bottom-right corner is $(X2, Y2)$) where

$$X1 = max(0, x - d), \quad X2 = min(L, x + d), \text{ and}$$

$$Y1 = max(0, y - d), \quad Y2 = min(L, y + d).$$

Figure 6 shows the coverage area of segments 1, 2, and 5 of $S^1$-tree (nona-tree).

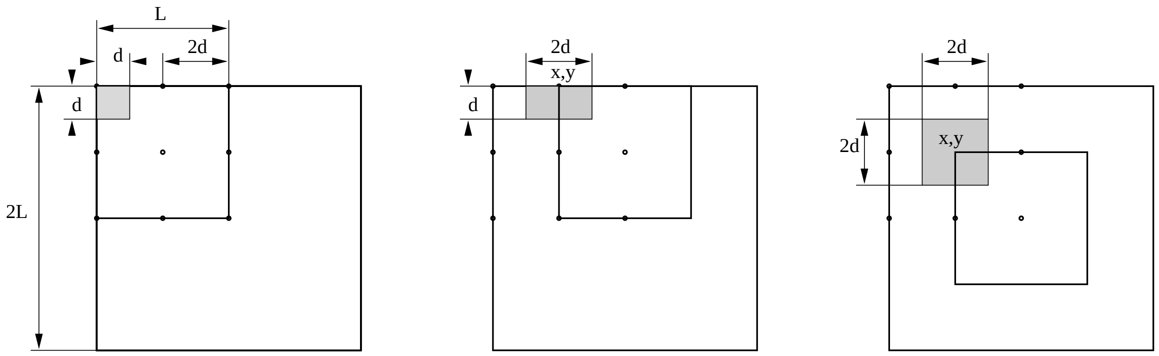



Figure 6: Coverage area of segments 1, 2, and 5 of $S^1$-tree (nona-tree) are shown in shaded areas.

**Theorem 2** *For a database image $D$ of size $2L \times 2L$ and a query icon $Q$ of size $L \times L$, the minimum subimage coverage of $Q$ by a segment of $S^g_D$-tree is $((2^{g+1} - 1)/2^{g+1})^2$ fraction of $Q$ ($L^2$).*

**Proof:** (Sketch) Assume segment $s$ of $S^g_D$-tree is the covering segment of query icon $Q$. Figure 7 shows segment $s$ as a square of size $L \times L$ with solid line. Coverage area of segment $s$ is shown as the shaded area in Figure 7. Since the covering segment of query icon $Q$ is segment $s$, the representative point (top-left corner) of $Q$ is located in coverage area of $s$. Figure 7 shows two of possible query icons as squares of size $L \times L$ with dashed lines.

In horizontal direction, both query icon $Q$ and segment $s$ have the size of $L$, and the maximum difference between their location is $d$, so the minimum overlap between them will be $L - d$. In other words, the minimum coverage of segment $s$ for query icon $Q$ in horizontal direction is $L - d$. Similarly, for vertical direction it can be shown that the minimum coverage of segment $s$ for query icon $Q$ is $L - d$. Hence, the size of minimum subimage coverage will be:

$$(L - d)(L - d) = (L - \frac{1}{2^{g+1}}L)(L - \frac{1}{2^{g+1}}L) = (1 - \frac{1}{2^{g+1}})^2 L^2 = (\frac{2^{g+1} - 1}{2^{g+1}})^2 L^2. \tag{6}$$

$\square$

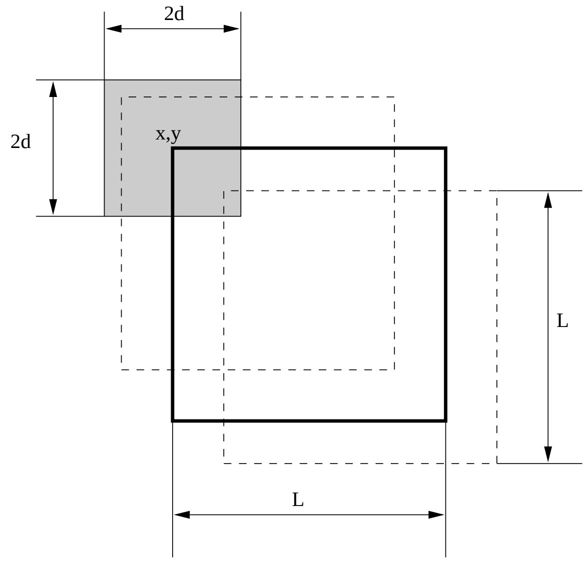



Figure 7: Solid line square is a segment of $S^g$-tree, and the shaded area is its coverage area. Two of possible query icons are shown as dashed line squares.

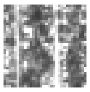
Following Theorem 2, if a database image $D$ contains query icon $Q$ as its subimage in nona-tree decomposition, then there exists a segment in database image $D$ which covers at least $\frac{9}{16}$ of query icon $Q$.

Note that Theorem 2 is independent of the *level* of $S^g$-tree and can be applied to any level of this tree.

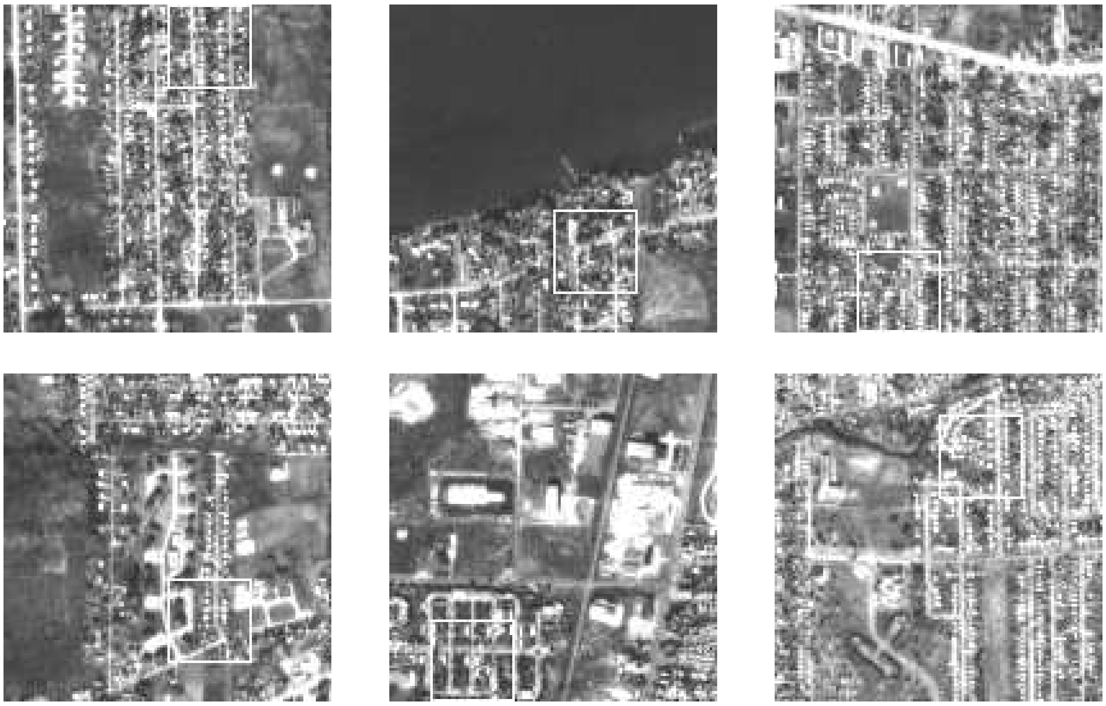
### 3.3 Matching Effectiveness

To apply the block-oriented tree structures in image retrieval from image database systems, all images in the database are decomposed into their block-oriented tree hierarchies. Each node in the tree is associated with an image segment and its feature vector. Given a query icon, its feature vector is also calculated. The matching between the feature vectors of the query icon and database images is then proceeded with a given distance threshold. In a large image database environment, conducting the matching procedure for each tree structure individually is not efficient and practical. We tested our system on a database of 405 airphoto images of size $128 \times 128$ pixels. Figure 8 shows a query icon of size $32 \times 32$ pixels and the top 6 retrieved images when nona-tree is used. The matched segment of each image is shown within the white box.

In general, the higher the degree of $S^g$-tree is, there will be more number of image segments in each database image to be compared. Thus, it is more likely to find a better segment matching to the query icon, resulting in a more effective retrieval. But on the other hand, the large number of extra image segments needs more storage space. Moreover, given an image icon, it is not practical



a) Query icon



b) Retrieved images

Figure 8: Retrieval results: a) query icon; b) retrieved images.

to search the $S^g$-tree of all individual database images. To address the latter issue, an efficient indexing must be designed on all image segments. We will discuss this issue in the next section.

## 4 Indexing for Efficient Search

We will now investigate an indexing approach on image segments to support efficient retrieval. A feature space is first organized which contains all generated feature vectors of image segments from database images. As we have shown earlier, higher degrees of $S^g$-tree decomposition generate a comparatively larger number of image segments than do lower degrees of an $S^g$-tree. More precisely, the increase of segments from $S^g$-tree to $S^{g+1}$-tree is at most four times of the segments generated by using $S^g$-tree. Obviously, the larger number of image segments generated in higher degree of $S^g$-tree structure may cause searches to be more time consuming. However, this problem can be circumvented by an indexing technique involving the feature vectors generated from the image segments.

## 4.1 $TV^+$-tree Indexing

Consider a feature space which contains all generated feature vectors of image segments. The feature vectors are then indexed. A novel indexing technique, termed $TV$-tree, has been designed to generate an index on a set of feature vectors [12]. This approach is a variation of the $R$-tree family [9, 18, 3].

As a member of the $R$-tree family, the $TV$-tree was proposed especially for the high dimensional data indexing. This indexing technique defines a well balanced indexing tree structure. It supports dynamic updating. The $TV$-tree is also fast in retrieval because it uses "active dimensions", which will largely reduce the calculations when searching through the tree. During the construction of the indexing tree, the $TV$-tree technique uses less dimensions of the feature space than the $R$-tree and the $R^*$-tree do. As nodes are closer to the root, fewer dimensions are used. As the tree descends, more dimensions of the feature space may be used. The total number of calculations will be far less than that of the $R$-tree when searching is performed.

We observed that it is difficult to directly apply the $TV$-tree and other $R$-tree variants in the applications in which some component features are more important than other component features. For example, consider feature vectors $v_1 = (300, 230, 125, 70)$, $v_2 = (300, 230, 125, 54)$, and $v_3 = (300, 220, 125, 70)$. Based on the closeness measurement of "distance" defined in [12], we can see that the distance between $v_1$ and $v_3$ is smaller (equal to 10) than that between $v_1$ and $v_2$ (equal to 16). Thus, $v_1$ and $v_3$ are considered more similar. However, if the first and second features in these three feature vectors are considered more important than the third and fourth feature components, then $v_1$ and $v_2$ should be considered more similar.

Another problem in applying the $TV$-tree lies in the fact that it only indexes the first $\alpha$ dimensions (called "active dimensions" in [12]) when there is no exact matching of the feature values in the feature vectors. This is because the number of dimensions indexed will change only when exact matching occurs. In our database, the feature vectors normally have real values. The chance of exact matching is very low. Using $TV$-tree technique will generate an indexing tree with heavy overlapping nodes. In our proposed approach, we will improve $TV$-tree to overcome these problems.

In order to use the $TV$-tree more effectively, we propose a generalization of the $TV$-tree, which is termed *$TV^+$-tree*. The goal of $TV^+$-tree indexing structure is to have the following properties:

- It should support high dimensional data indexing. Our feature vectors generated from remote sensing images contain as many as 60 feature components.

- It should support both efficient and effective data retrieval. That is, retrieval must be fast, and high precision and recall [23] should be obtained. It should also support dynamic updating, which means new images can be inserted and out of date images can be deleted, when needed.

As we can see, if the two problems of the $TV$-tree mentioned above can be solved, we will have an indexing tree structure that satisfies the above requirements. In order to overcome the two problems of the $TV$-tree, we will firstly redefine the term "similarity", and then formulate a strategy to increase the possibility of exact matching of features in the feature vectors to be indexed.

We introduce the concept of *similarity* based on the weights on the feature vector components. In many applications, some features play a more important role than others. We define the similarity based on the weighted features as follows. Let two feature vectors be $v_1 = \langle x_1, ..., x_n \rangle$ and $v_2 = \langle y_1, ..., y_n \rangle$. When comparing the similarity of feature vectors $v_1$ and $v_2$, we consider the importance of individual features. If feature $x_i$ is more important than feature $x_j$ ($i \neq j$ and $i, j \in \{1, ..., n\}$) and feature vectors $v_1$ and $v_2$ have the same difference in value between $x_i - y_i$ and $x_j - y_j$, that is,

$$x_i - y_i = x_j - y_j,$$

then the difference in $x_i - y_i$ should be considered more important than that of $x_j - y_j$. Using a weight function $w_i$, we can more precisely represent the distance between the two feature vectors. The distance or "similarity" between two feature vectors is defined as:

$$d(v_1, v_2) = \sqrt{\sum_{i=1}^{n} w_i (x_i - y_i)^2}. \tag{7}$$

If the features in the feature vectors are ordered such that the features with smaller subscripts are always more important than those with bigger subscripts, an example of the weight function $w_i$ could be

$$w_i = 1/i^2, \quad i = 1, ..., n. \tag{8}$$

By using the above definition of "similarity" based on the weighted features, we can get a better precision when retrieving images in our application, because the most important features are always evaluated with the highest weights. Note that different applications may require different weight functions in order to apply (7) effectively. Also, the $TV^+$-tree degrades to be a $TV$-tree if the weight function is chosen to be 1.

Using the $TV^+$-tree indexing on database images, the feature vectors are grouped into leaf nodes of the indexing tree and the description of their *minimum bounding region* (MBR) [12] is stored in the parent node. In case an MBR contains too many elements, additional features will be used to split the MBR. Parent nodes are then recursively grouped until the root is constructed. Each parent node contains an MBR which covers all descendants of that branch. We assume that each node (may not include root) contains between $m$ and $M$ ($m < M$) children or data items.

As an example, consider the following four feature vectors to index: $f_1 = (300, 230, 125, 70)$, $f_2 = (300, 230, 125, 54)$, $f_3 = (300, 220, 125, 70)$, and $f_4 = (300, 220, 125, 54)$. We are also given

$M = 3$, $m = 2$, and $\alpha = 4$. Using the $TV$-tree in [12], $f_1$ and $f_3$ will be inserted into the same internal node, and $f_2$ and $f_4$ into another internal node, as shown in Figure 9 (a). This is because the distance between $f_1$ and $f_3$ is 10, while the distance between $f_1$ and $f_2$ is 16. Thus, in the $TV$-tree, $f_1$ and $f_3$ are considered more "similar" to each other than $f_1$ and $f_2$ would be. Similarly, $f_2$ and $f_4$ are more "similar", while $f_3$ and $f_4$ are less "similar". However, if the importance of the features in these feature vectors is ordered such that for $x_i, x_j \in f_k$ and $i < j$, $x_i$ is more important than $x_j$, we then get the wrong results using the $TV$-tree. Through the weight function used in $TV^+$-tree, this importance ordering can be represented. Assume that we assign the weights as $w_i = 1/i^2$. We will get a tree that has two internal nodes: one contains $f_1$ and $f_2$, with a distance 4, and the other contains $f_3$ and $f_4$, also with a distance 4. Figure 9 (b) shows the $TV^+$-tree indexing for $f_1, f_2, f_3$ and $f_4$. The distance between $f_1$ and $f_3$ now becomes 5, and so does the distance between $f_2$ and $f_4$. The result of the $TV^+$-tree structure is apparently better than the $TV$-tree structure. It should be noted that, in general, we can use any appropriate weight function.

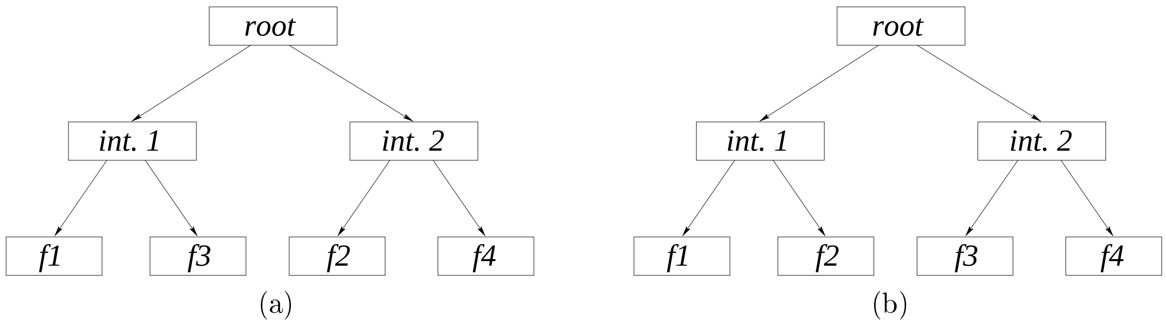



Figure 9: (a) Indexing tree grown by the $TV$-tree. $Int.1$ and $int.2$ stand for internal nodes; (b) Indexing tree grown by $TV^+$-tree.

In order to increase the possibility of exact matching, we can simply round up real valued features to integer valued features. This can be done even in case of small or narrow distributed feature values. Once the exact matching problem is solved, other problems like heavy overlapping will be solved accordingly. The MBR overlapping may still exist. This can only be fixed by introducing new algorithm when splitting nodes, like the way that the $R^*$-tree is implemented [3].

The $TV^+$-tree structure can improve the precision of the image data retrieving as well as the searching speed for given image icons. We will discuss this issue in the next subsection.

## 4.2 $TV^+$-tree Querying

Once we have built up the $TV^+$-tree indexing of the feature vectors, content-based image retrieval can be facilitated. To conduct the similarity search in the $TV^+$-tree indexing for a given query icon, the search process starts from the root and examines each branch that overlaps with a given search region until the leaf MBRs are selected. Given a query icon $Q$, we first extract the same features from $Q$ as that from database images and form a query feature vector. Because these features will be compared to the features in the indexed feature vectors, we need to weigh them in the same way as we have done to the indexed feature vectors. Thus, when comparing this query feature vector to the feature vectors in the indexing tree, we use the same strategy as we have done when we insert new feature vectors into the indexing tree. That is, we weigh the features and get the correct distance. When searching is performed, the query feature vector will be compared to the indexing tree MBRs at each level of the tree. Because the same weight function is used, the query feature vector will be directed to a more precise subtree than the $TV$-tree will do. Recursively searching the subtrees, we can eventually obtain the top most similar images to the icon $Q$. The total number of images returned depends on a threshold indicating how much overlapping between the query icon and the database images will be considered to be "similar".

### 4.3 Efficiency Improvement

We now compare the search efficiency on feature spaces that are generated from various degrees of $S^g$-trees. Let the database contain $\mathcal{N}$ images. Let the number of points in the feature space generated using an $S^g$-tree structure be $N_g$. In general, if each internal node in an indexing tree has $\lambda$ children and the tree is fully balanced, the height $h$ of the tree will be:

$$h = log_{\lambda} N_g. \tag{9}$$

The index tree has its maximum height when $\lambda = m$, that is, $log_m N_g$. Following Formula (5), the number of points in the feature space generated using an $S^{g+h}$-tree can be approximately estimated to be $4^h * N_g$. We can thus calculate the difference of the maximum heights of indexing trees corresponding to $S^{g+h}$-tree and $S^g$-tree:

$$log_m(4^h * N_g) - log_m N_g = log_m 4^h = h * log_m 4. \tag{10}$$

Formula (10) indicates that the increase of maximum height of the indexing tree generated from $S^g$-tree to $S^{g+h}$-tree is at most $h * log_m 4$. Thus, using the above indexing technique can ensure that the retrieval efficiency of query icons using the nona-tree structure is comparable to that of using the quad-tree structure, with maximum increase of $\lceil log_m 4 \rceil$ in height.

If there are $\lambda$ children in each internal node of index tree of a database of size $N_g$, then to search the index tree, $\lambda * log_{\lambda} N_g$ comparisons will be performed. The parameter $\lambda$ ($m \le \lambda \le M$), depends on the values of $m$, $M$, disk page size, and distribution of entries in the nodes of index tree. We define $D_c$ as the difference in the number of comparisons when searching for segments in $S^{g+h}$-tree and $S^g$-tree. When using index, $D_c$ will be as follows

$$D_c = \lambda * log_{\lambda}(4^h * N_g) - \lambda' * log_{\lambda'}(N_g). \quad (11)$$

Assuming that the value of $\lambda$ for $S^{g+h}$-tree and $S^g$-tree is the same ($\lambda = \lambda'$) , we will have

$$D_c = \lambda * log_{\lambda}4^h = h * \lambda * log_{\lambda}4. \quad (12)$$

As an example we consider nona-tree and quad-tree (h=1), when number of quad-tree segments ($N_g$) is 1,000,000. Assume $\lambda = \frac{m+M}{2}$, that is, all the internal nodes contain the average number of children between $m$ and $M$. Value of $m$ and $M$ depends on disk page size and number of features. Table 1 shows $D_c$ for different page sizes when we have 20 real valued features. In this setup when disk page size is 2048 bytes, total number of comparisons in searching the index tree of quad-tree is 80. Searching the index tree of nona-tree needs just 8 more comparisons ($D_c = 8$). However, when we do a *serial search* on the segments generated by nona-tree, we need 3,000,000 more comparisons ($D_c$ = 3,000,000). It shows that searching an $S^g$-tree of higher degree using proposed index tree will add little cost to the search time, while providing access to more accurate segments that user is looking for.

| Page size (byte) | 512 | 1024 | 2048 | 4096 |
|---|---|---|---|---|
| $D_c$ | 4 | 6 | 8 | 13 |

Table 1: $D_c$ for different disk page sizes when using index tree.

By carefully defining "similarity", rounding up the feature values in feature vectors to be indexed, we can build an indexing tree that works under much more realistic situations. Even though rounding up the feature values in feature vectors may cause overhead, fast retrieval speed will still be guaranteed. This is because that all of the data preprocessing can be done at the database constructing time. Various types of queries can be supported based on the above indexed image database. For example, in a biomedical image, an anomaly such as a tumor in a diagnostic image can be captured in an iconic texture representation. The block-oriented image decomposition provides an effective method for searching iconic queries. In the real-world applications, a user may ask for top $N$ (say 20) relevant images. To apply this approach to on-line queries, various thresholds must be determined to support such queries. An efficient approach is to simply sort all the feature vectors in the selected MBRs in the order of their distance with the feature vector of the query icon, and then return the top $N$ images.

## 5 Discussion

We have conducted substantial experimental analysis on the retrieval performance of quad-, quin-, and nona-tree decomposition [16, 19]. Our experimental results indicate that nona-tree decomposition offers the best content-based image retrieval performance among the three structures studied. Based on the above analysis, we can see that nona-tree decomposition generates four times as many segments as that of quad-tree decomposition. However, the increased data amount introduced by the nona-tree structure can be offset by applying a good indexing technique to the feature vector space. We will now discuss the issue of whether the nona-tree structure should be further expanded. Theoretically, the block-oriented approach to image decomposition can continue to expand the nona-tree structure to a higher degree of $S^g$-tree ($g > 1$). Within such an expanded structure, if a portion of a decomposed image matches a query icon, there will then exist a segment in the decomposed image which covers more than $\frac{9}{16}$ of the query icon, as illustrated in Theorem 2. However, to guarantee evenly distributed segment decomposition and equal coverage of query images, we must add a greatly increasing number of the segments at each level of tree decomposition. To expand the nona-tree structure, the smallest number of subsegments to be decomposed for the image at the first level is 25 ($S^2$-tree). The smallest number of subsegments to be decomposed for the image at the first level to further expand the $S^2$-tree structure is 81 ($S^3$-tree). Based on Formula (5), we see that such an expansion will cause the feature space to grow at a rate of four times from each degree of the $S^g$-tree to one degree higher, this increase may seriously affect storage efficiency for large image databases.

Thus, a balance between image retrieval effectiveness and storage efficiency must be achieved in choosing a specific block-based decomposition strategy. Our experimental results indicate that, for certain query icons, the nona-tree structure is more effective than either the quad- or quin-tree structures. In addition, although the nona-tree structure increases the number of segments, retrieval efficiency can be maintained by using a good indexing technique.

## 6 Conclusions

In this paper, we have presented a fundamental data model for specifying image content in large image databases. This image data model involves recursively block-oriented image decomposition on images. A generalized hierarchical structure $S^g$-tree with $g$ representing the degree of the tree structure has been introduced to specify block-oriented decomposition of database images. We have conducted various computational analysis on the maximum number of possible segments in various block-oriented tree structures and the minimum overlapping coverage between a query image and the segments of a database image in instances in which the query icon is similar to a subimage of the database image. We have also discussed the indexing and searching approaches that can be applied on the feature spaces constructed from various $S^g$-tree decomposition, and analyzed their search efficiency. Using $S^g$-tree structure to represent image content in an image database, flexible access to partial contents of each database image can be provided. Various types of content-based queries and efficient image retrieval can then be supported through novel indexing and searching approaches. Our analysis illustrates that nona-tree decomposition is an effective and efficient block-oriented decomposition structure to facilitate effective content-based image retrieval.

## References


[1] Manish Arya, William Cody, Christos Faloutsos, Joel Richardson, and Arthur Toga. QBISM: A Prototype 3-D Medical Image Database System. *IEEE Data Engineering Bulletin*, 16(1):38–42, 1993.

[2] J.R. Bach, S. Paul, and R. Jain. A Visual Information Management System for the Interactive Retrieval of Faces. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):619–628, 1993.

[3] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and Robust Access Method for Points and Rectangles. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 322–331, Atlantic City, NJ, May 1990.

[4] P. Brodatz. *Textures: A Photographic Album for Artists and Designers*. Dover, New York, 1966.

[5] S.K. Chang. *Principles of Pictorial Information Systems Design*. Prentice Hall, 1989.

[6] S.K. Chang, C.W. Yan, Donald C. Dimitroff, and Timothy Arndt. An Intelligent Image Database System. *IEEE Transaction on Software Engineering*, 14(5):681–688, May 1988.

[7] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. The Benjamin/Cummings, 1994. Second edition.

[8] T. Gevers and A.W.M. Smeulders. *An Approach to Image Retrieval for Image Databases*, volume 720. Lecture Notes in Computer Science, Springer-Verlag, Berlin.

[9] A. Guttman. R-Trees: A Dynamic Index for Geometric Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1984.

[10] Michael L. Hilton, Bjorn D. Jawerth, and Ayan Sengupta. Compressing Still and Moving Images with Wavelets. *Multimedia Systems*, 2(5):218–227, December 1994.

[11] Tai-Yuan Hou, Peiya Liu, Arding Hsu, and Ming-Yee Chiu. Medical Image Retrieval by Spatial Features. In *IEEE Conference on Systems, Man, and Cybernetics*, 1992.

[12] K.-I. Lin, H.V. Jagadish, and C. Faloutsos. The TV-Tree: an Index Structure for High-Dimentional Data. *The VLDB Journal*, 3(4):517–542, October 1994.

[13] K. Perez-Lopez, A. Sood, and M. Manohar. Selecting Image Subbands for Browsing Scientific Image Databases. In *Proceedings SPIE Conference on Wavelet Applications*, Orlando, April 1994.

[14] F. Rabitti and P. Savino. Automatic Image Indexation and Retrieval. In *Conference on Intelligent Text and Image Handling*, 1991.

[15] E. Remias, G. Sheikholeslami, and Aidong Zhang. Block-Oriented Image Decomposition and Retrieval in Image Database Systems. In *Proceedings of the 1996 International Workshop on Multi-media Database Management Systems*, Blue Mountain Lake, New York, August 1996.

[16] E. Remias, G. Sheikholeslami, Aidong Zhang, and Tanveer F. Syeda-Mahmood. Supporting Content-Based Retrieval in Large Image Database Systems. *The International Journal on Multimedia Tools and Applications*, 4(2):153–170, March 1997.

[17] M. Flickner H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by Image and Video Content: The QBIC System. *IEEE Computer*, 28(9), 1995.

[18] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-tree: A Dynamic Index of Multidimensional Objects. In *Proceedings of the 18th VLDB conference*, 1987.

[19] G. Sheikholeslami, E. Remias, J. Guo, and Aidong Zhang. Image decomposition and representation in large image database systems. *The Journal of Visual Communication and Image Representation.* (scheduled to appear in Vol. 8, No. 2, June 1997).

[20] John R. Smith and Shih-Fu Chang. Quad-Tree Segmentation for Texture-Based Image Query. In *Proceedings of ACM Multimedia 94*, pages 279–286, San Francisco, California, October 1994.

[21] Hideyuki Tamura, Shunji Mori, and Takashi Yamawaki. Texture Features Corresponding to Visual Perception. *IEEE Transactions on Systems, Man, and Cybernetics*, 8(6), June 1978.

[22] A. Turtur, F. Prampolini, M. Fantini, R. Guarda, and M.A. Imperato. IDB: An Image Database System. *IBM Journal of Research and Development*, 35(1):88–96, January 1991.

[23] Cornelis J. van Rijsbergen. Retrieval Effectiveness. In Karen Sparck Jones, editor, *Information Retrieval Experiment*, pages 32–43. Butterworths, 1981.

[24] Jian Kang Wu and Arcot Desai Narasimhalu. Identifying Faces Using Mutiple Retrievals. *IEEE Multimedia*, 1(2):27–38, 1994.

[25] Aidong Zhang, Biao Cheng, and Raj Acharya. An Approach to Query-by-texture in Image Database Systems. In *Proceedings of the SPIE Conference on Digital Image Storage and Archiving Systems*, pages 338–349, Philadelphia, October 1995.

[26] Aidong Zhang, Biao Cheng, Raj Acharya, and Raghu Menon. Comparison of Wavelets and Fractals in Texture-based Image Retrievals. In *Proceedings of the SPIE Conference on Visual Data Exploration and Analysis III*, pages 116–125, San Jose, January 1996.