1998

# Texture image retrieval using fuzzy image subdivision.

Donato. Ingratta
*University of Windsor*

# INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

**The quality of this reproduction is dependent upon the quality of the copy submitted.** Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

# Texture Image Retrieval Using Fuzzy Image Subdivision

by

**Donato Ingratta**

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through the School of Computer Science in Partial
Fulfillment of the Requirements for the Degree
of Master of Science at the
University of Windsor

Windsor, Ontario, Canada
1998

# Abstract

Large image databases containing a wide variety of imagery are increasingly more common. Because of their diversity and size, they are often poorly indexed. As a result, new automated techniques which index and search images using visual image properties, such as color and texture, have emerged. Texture is used to charaterize image regions and requires the regions of an image to be identified prior to indexing. Two region extraction methods: subdivision and segmentation are used for this task, and are used to compute direct-match and object-based queries, respectively. Direct-match queries are less costly to compute, and indexing is done without user intervention, resulting in automated content-based image retrieval systems (CBIR). In this thesis, we investigate the use of a new subdivision method, fuzzy image subdivision, to address the undesirable dependence on object position in direct-match texture queries. We implement our approach for querying images on the web and compare the retrieval performance with the current direct-match texture method based on rectangular partitioning, which does not take into account changes in object position.

*Per mei genetori and mia nonna, con amore*

# Acknowledgements

Many thanks to my supervisor Dr. Joan Morrissey for her guidance and support in bringing this body of work to a successful conclusion. Thanks also goes to my fellow graduate students in computer science for their friendship and helpful suggestions. Special thanks to Alexander Dimai for providing me with the source code to his fuzzy image subdivision and color-moment algorithms. Lastly, but not least, my heart felt thanks to my parents and grandmother, without whom this thesis would not of been possible.

# Contents

# List of Figures

# List of Tables

# Chapter 1 INTRODUCTION

Owing to the rapid increase in the size of current image databases, new techniques for efficient and effective retrieval from large image databases using image content have been proposed. Known as content-based image retrieval (CBIR), these techniques are unique in that they employ image analysis to retrieve images in response to queries on generic image properties, such as color, texture and shape. Various techniques which query images on one or more of these properties have been developed [SB91b, SD96, NBE93a, FF91, OS95, KZL94, CBGM97, Smi95]. Basically the techniques differ in terms of query specification, the types of features used to characterize the image properties, and the level of automation achieved in implementation. CBIR is desirable for the following reasons: first, indexing can be done automatically, thus eliminating common errors in manual indexing, such as misspelled and inaccurate keywords; and secondly, querying is more ad hoc, so that users of all skill levels may perform CBIR searches, even when their familiarity with the images in the database is poor.

Typically, query specification in CBIR is performed using the image-by-example model in which the user provides an image example (or a graphical representation, such as a user sketch or painting) and asks the system for images that are nearest or most similar to it. This type of retrieval is called *similarity retrieval* [PF94] and is different from traditional database searches which are

based on exact matching. In recent years, texture-similarity image retrieval has received much interest [KZL94, ZLSW95, PPS94, NBE93b, Smi95] and is used to retrieve images independently of color and shape. In this thesis we develop a new texture-similarity approach based on *direct-matching*. One notable problem in texture retrieval is the identification of textured regions (regions extraction). Texture characterizes the regions of an image and is computed in image regions. Thus prior to indexing (feature extraction), the textured regions of an image must be first identified. This can be done in two ways [KZL94]:

- the image is divided into blocks of the same size and shape.
- divide the image into its structural units or objects of interest.

The former method called *image subdivision* is computed cheaply without the need for user assistance. The latter method called *image segmentation* requires user assistance, and is computation and storage intensive. Which method is used determines what types of queries may be performed on the images. These query types are [FMW97]:

- direct-match queries
- object-based queries

In direct-match queries [KZL94, NBE93b], the texture regions are identified using a predefined set of image regions (e.g. grid). The image example is subdivided at query time and the database images are subdivided at database population.

Images are then matched by comparing the texture in their corresponding image regions. This is illustrated in figure 1 using a 2x2 grid subdivision.

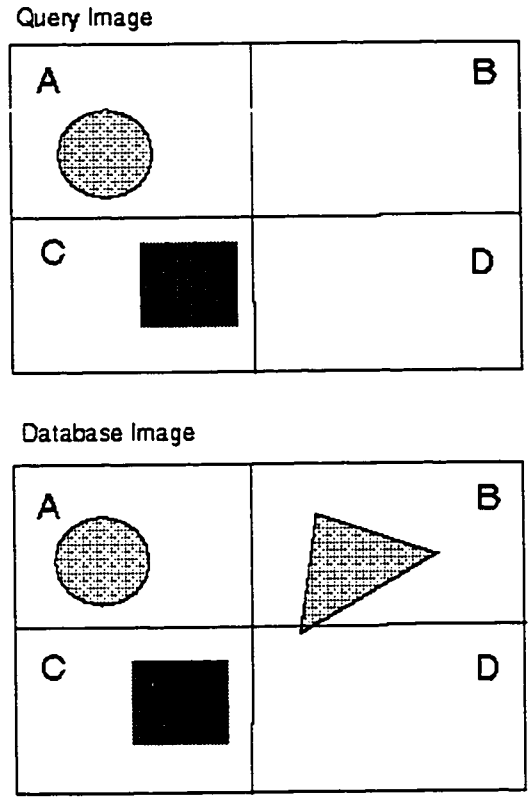



Figure 1 Direct-match query: regions are matched in-situ.

In the above example, regions A, C and D have the highest texture similairty. In object-based queries [LBN94, FMW97], the texture regions are identified using manual or semi-manual extraction (e.g. object outlining or edge following). An object is extracted from the image example at query time and all database objects are extracted at the time of database population. Images are then matched by

comparing the texture of the query object to that of all database objects. This is depicted in figure 2 where object A in the query image is matched against objects A, B and C of the database image. In the example below, objects A and object B have the highest texture similarity. Because direct-matching does not required

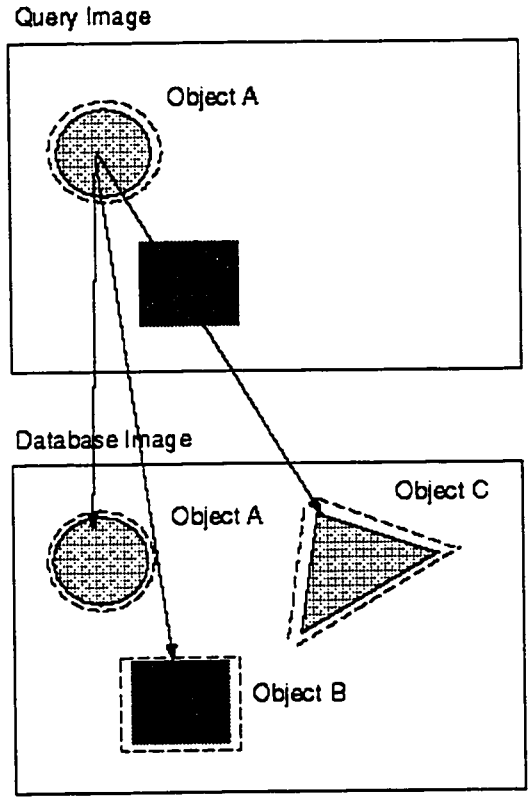



Figure 2 Object-based query: outlined object in query image is matched with each database object

user assistance in region extraction, it is used to develop automated CBIR systems. Object-based systems, on the other hand, are used to develop semi-automated systems, in which texture regions are extracted manually or semi-automatically.

Direct-match systems are preferred when the images are from diverse domains and the user's familiarity with the images is low to high. Object-based systems are preferred when the images are from the same domain and the user's familiarity with images is high.

However direct-match queries suffer from the problem of position dependence. Minor changes in object position, arrangement and configuration will defeat most direct-match image comparisons. Finding features which are invariant under these changes is one goal of the feature invariance problem. Ideally, in direct-matching, features which are also invariant to changes in viewing angle and lighting conditions are needed for optimal retrieval. Currently, no work has been done to address the problem of position dependence in direct-match retrieval using texture-similarity. However, the color-similarity approach in [DS96] has achieved moderate success in reducing the dependence on position in direct-matching of color regions, using a new subdivision method based on fuzzy regions. In fuzzy image subdivision, the regions are defined using fuzzy sets and help diminish the undesirable effects on retrieval of certain changes in object position.

## 1.1 The Thesis Statement and Topics to be investigated

A new direct-match image retrieval approach based on texture-similarity is developed in this thesis. Our approach employs fuzzy image subdivision (FIS) [DS96] to extract the texture regions of an image, and the Tamura texture features

(coarseness, contrast, directionality and line-likeness) [TMY78] to characterize the texture content in the regions. The objective of our study is the following: **to investigate whether fuzzy image subdivision may be used to improve the retrieval performance in direct-matching based on texture-similarity**.

In order to establish that FIS is useful and beneficial, we have attempted to study the following questions:

- ☐ For what types of images is FIS useful in texture-similarity retrieval?
- ☐ How can FIS be incorporated into a direct-match retrieval method based on texture-similarity?
- ☐ What improvements can be expected if FIS is used in direct-match retrieval? Specifically what are the effects on
    a. feature invariance with respect to changes in object position?
    b. index storage requirements?
    c. image-by-example specification model?

To answer these questions we have implemented a texture-retrieval system based our approach, the QBT (Query-by-Texture) system. The use and benefits of fuzzy image subdivision in texture-similarity image retrieval has not previously been investigated. We carried out experiments where the retrieval performance of our system (in terms of precision and two new performance indicators) is compared with the system in [KZL94] (the current state of the art in direct-match texture-

similarity retrieval) which extracts the Tamura features using rectangular or grid partitioning. A FIS region-based texture-query specification tool based on the image-by-example strategy is also designed and implemented for retrieval from the world-wide-web.

## 1.2 Thesis Overview

In chapter two the basic concepts in texture-similarity retrieval based on direct-match and object-based queries are discussed. We also review texture extraction techniques and two direct-match retrieval systems based on rectangular partitioning reported in the literature. The limitations of rectangular partitioning in direct-matching are also discussed. Chapter three discusses the fuzzy image subdivision method and outlines the algorithm. Chapter four gives the details of our versions of the Tamura texture algorithms which are modified for extraction using fuzzy image subdivision. Chapter five gives the design and implementation details of QBT (Query-by-Texture), our texture-similarity image retrieval system. Chapter six discusses the test results of 50 texture image queries used to evaluate our system in terms of retrieval effectiveness feature extraction efficiency against the system in [KZL94]. Finally, chapter six gives the conclusions of this study.

# Chapter 2 REVIEW OF LITERATURE

## 2.1 Content-based image retrieval

Content-based image retrieval (CBIR) uses general properties such as color, texture and shape to locate images in response to queries based on the image content. For instance, images can be represented as a collection of shapes, and the properties of each shape in an image can be used as parameters in content-retrieval searches. A number of systems have developed which can query images on one or more of these generic image properties [SB91b, GZCS94, OS95, NBE93a]. Content-retrieval searches are quite different from the searches in keyword based retrieval which have traditionally been used to retrieve images. In key-word retrieval the search parameters are key-words derived from the text descriptions that are manually assigned to each image by the user.

To be useful and retrievable, the content information in each image needs to be indexed. In other words, various *features* from each property (color, texture and shape etc.) need to be identified and attached to each image. The features of each property can be automatically extracted from the image content using image processing and pattern recognition programs. These programs can perform dynamic feature extraction and can be used to index images without user input. A content-based image retrieval system has two fundamental components:

1. An automated feature extraction subsystem used to automatically extract features which quantitatively describe the generic properties of the image content.
2. A visual-querying interface to allow the generic image properties which are inherently graphical to be specified using graphical tools and GUI-based menus.

Interest in content-based image retrieval has increased considerably due to the recent increase in the size of current image databases. Current image databases may contain terabytes ($10^{12}$) of image data stored in hierarchical storage systems which maintain the images off-line in optical disk arrays. CBIR systems are ideal for handling the vast amounts of image data in these large databases for the following reasons:

1. *Indexing can be automated.* Generic image properties such as color and texture can be algorithmically computed from images — eliminating the need to manually index the images. For example, a color histogram counting the number of pixels of each color, can be quickly extracted from an color image using a simple computer program. Color histograms are used to retrieve images with similar color distributions.
2. *Domain-independent querying.* Since images are indexed using generic properties (color, texture and shape), queries can be made even when the user's

familiarity with the images in the database is poor. For example, images containing a sunset over a lake can be queried by using color layout information as in " Find me images with 60% yellow and orange content at the top-left and 90% blue content in the bottom". In key-word based systems, the user must be familiar with the indexing terms used in order to have a chance of obtaining good results.

3. *Similarity-retrieval.* Retrieval in content-based queries is based on approximate matching and not exact. Images are ranked according to how "similar" they are in color, texture and shape to the user's query. The images which are closest to the user's query are returned in ranked similarity order. Similarity retrieval is used to constrain the search results so that the user does not have to browse too many images to find the desired ones. In key word retrieval, the use of non-specific terms such as "Find me images of a lake" will often lead to an unmanageable amount of browsing.

## 2.2 Computing texture queries

Querying images on texture content (texture-retrieval) is used to locate images independent of color and shape. The texture features of an image are computed from the brightness distribution (gray-scale image) within the textured regions of the image, and are invariant to region orientation and shape. Thus texture extraction requires preprocessing to identify the textured regions within an image.

The queries in texture-retrieval are specified visually using examples of texture such as the texture in an indicated image or in a texture sample selected from a GUI-based texture picker. Some examples of texture from the Brodatz texture photo album are given in figure 3 [Smi95]. In addition, a user constructed example

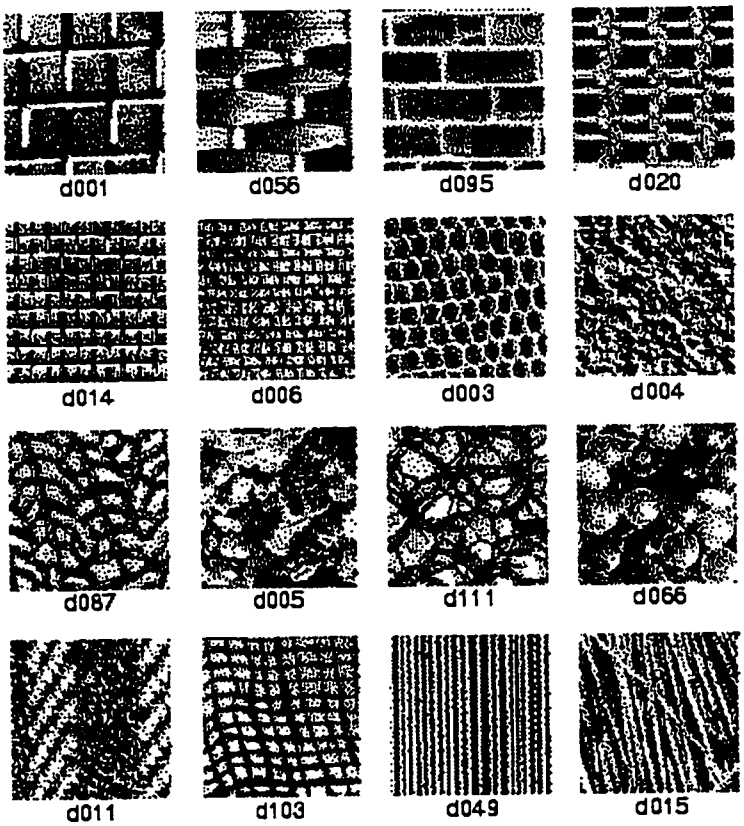



Figure 3 Texture examples from the Brodatz photo album

may be used such as a sketch or painted representation of the texture image. Querying of this form is called query-by-example. In all cases, texture features are extracted from the texture example (in the same way they are extracted from

the database images), and the database images similar in texture to the indicated example are retrieved.

An image may contain both non-textured and textured regions. Thus special considerations are made in indexing the texture content of an image. The non-textured regions must be identified and indexed using other types of properties such as color and shape. Each textured region is identified and indexed using its own set of texture features. The textured regions within an image are identified in two ways [KZL94]:

1. Divide the image into small blocks. This is called *image subdivision* and creates a *regular* partition of the image into rectangles. A partition based on differently shaped and sized regions may also be used and results in an *irregular* partitioning of the image. The texture within each block or region is assumed to be uniform.
2. Identify the texture boundaries of each region. This is called *image segmentation* and the result consists of a polygonal approximation to each region outline. The texture within each region must be reasonably uniform for this method to work.

Both methods have advantages and disadvantages. Subdivision is inexpensive to compute and can be done without user input. However in most cases subdivision will not produce a set of regions which accurately identify the textured

regions within an image. On the other hand, segmentation accurately identifies the texture regions but involves significant amounts of computation and user input. Segmentation algorithms which emulate human outlining (e.g. edge following) rely on the user providing a starting point on the boundary for each region [Rus95]. Images containing natural scenes (real images) such as the outdoors, wildlife, and animals are especially difficult to segment. They contain many textured regions which are too small to segment properly. Current edge and line finders are designed for detecting discontinuities between large homogeneous regions and do not work satisfactorily on small regions [HSD73]. And even to the extent that the boundaries of small regions are determined, the data structures used to store them would require an unreasonable amount of computer memory, since the boundary descriptions are not longer economical [Baj73]. Because of the reliance on user assistance and the cost associated with segmentation, subdivision is used in texture-content retrieval to develop systems which have automatic indexing capabilities.

## 2.3 Matching images using texture region similarity

The images which are returned in texture-content queries are found by comparing the textured regions of each database image to the textured regions of the query example. Each pair of regions is matched, one from the database image and the other from the query example, in turn, and a dissimilarity score of their texture

features is computed. The dissimilarity score is computed using a texture distance function which is devised to give large values when the regions are dissimilar in texture and low values when they are similar. The texture feature set of of each region is compared as a *feature vector* in a multidimensional vector space. Here each dimension corresponds to a different texture feature. Thus the dissimilarity between two feature vectors can be computed as the distance between them. For example, Euclidean distance is typically used to compute this distance. The total dissimilarity score of the database image to the query example is then computed as a function of the region-to-region dissimilarity scores based on texture distance.

The algorithm used to match the regions between the query example and database image can work in two ways. The following is summarized from [Smi95]:

1. The regions in the query example and database image are fixed. This is called *direct-matching* and the regions are a fixed set of regions defined by subdivision. Each region of the query example is matched in-place with its corresponding region of the database image.
2. The regions in the query and database image are not the same. This is called *object-based matching* and the regions or objects are identified using segmentation. The regions of the query example are not compared in-place with the regions of the database image. Each query region is allowed to

"float" and is matched against each region in the database image.

The algorithms which implement each of the above matching schemes are based on 3 components [Smi95]:

1. A region-to-region match function or texture dissimilarity function. This function computes a match value or texture dissimilarity score between a query example region and a database image region based on texture distance. If the feature vector $\vec{F}$ computed from each region has $k$ features $\{f_1, f_2, \ldots, f_k\}$ then this distance can be computed using Euclidean distance or some other distance function. One common distance function is the *city-block* function computed as,

$$d_{city}\left(\vec{F}_I, \vec{F}_Q\right) = \sum_{i=1}^{k} \left| f_i^I - f_i^Q \right| \tag{1}$$

where $\vec{F}_I$ and $\vec{F}_Q$ are the query example and database image region feature vectors, respectively.

2. An image-to-query example match function or image dissimilarity function. This computes the total match or overall texture dissimilarity score between the database image and query example based on the region scores. One match function which is typically used is computed as the average region-to-region match values:

$$D_{image}(I, Q) = \frac{1}{n} \sum_{i=1}^{n} d_{region}(I_i, Q_i) \tag{2}$$

where $I$ and $Q$ are the database image and query example, respectively, and $n$ is the total number of regions compared

3. A retrieval threshold. This real value determined by the user is the maximum database-to-query example dissimilarity value which can be included in the result set of similar database images. Thus the returned set of similar database images $R$ can be defined by:

$$R = \{I \mid D_{image}(I, Q) \leq t\} \tag{3}$$

where $t$ is the dissimilarity retrieval threshold .

Using these components, the general steps of the direct-matching and object-based matching algorithms are [Smi95]:

1. Direct-matching algorithm:
   a. Score each region $Q_i$ in the query example with its corresponding region $I_i$ in the database image using the region-to-region texture dissimilarity function.
   b. Take the total score between the database image $I$ and query example $Q$ as the average region-to-region match score. The image-to-query example match function is computed as

$$D_{image}(I, Q) = \frac{1}{n} \sum_{i=1}^{n} d_{region}(I_i, Q_i) \tag{4}$$

where $n$ is the number of regions in both the query example and database image.

c. Repeat step *a.* until all database images have been processed.

d. Rank the database images in ascending order of the total score computed in *b.*

e. Return all database images where

$$D_{image}(I, Q) \leq t \tag{5}$$

where $t$ is the retrieval threshold.

Object-based matching algorithm:

a. Score each query region $Q_i$ in the query example with each region $I_j$ of the database image. This is computed using the region-to-region match function.

b. Take the total score between the database image $I$ and query example $Q$ as the average region-to-region match score. This is computed as

$$D_{image}(I, Q) = \frac{1}{m \times n} \sum_{i=1}^{m} \sum_{j=1}^{n} d_{region}(I_j, Q_i) \tag{6}$$

where $m$ is the number of query example regions and $n$ is the number of database image regions.

c. Repeat step a. until all database images have been processed.

d. Rank the database images in ascending order of the total score computed in *a.*

e. Return all database images where

$$D_{image}(I, Q) \leq t \tag{7}$$

where $t$ is the retrieval threshold.

## 2.4 Texture Extraction

Texture extraction is a widely studied topic and has application in many areas, such as remote sensing, parts inspection and computer graphics [KZL94]. Features are extracted which quantify basic texture properties such as: texture element shape (e.g. line-like or blob-like), size (e.g. granularity) and the spatial arrangement of texture elements (linear, bidirectional) [Baj73]. A texture element is the basic structural unit in a texture pattern. The pattern can be attributed to arranging the texture elements (pattern unit) according to a placement rule [TMY76]. This is the structural view of texture and is not useful in practical texture extraction. Instead, statistical methods are used which are based on the observation that values of certain local properties occur repeatedly throughout the textured region [Zuc76]. Thus texture is measured in terms of various statistics of local properties in the 2–D image space.

In most statistical methods the values of basic texture properties are computed in local neighborhoods over the textured region. A neighborhood is an area centered around each pixel that is typically square. Each pixel in the neighborhood is connected and has 8–nearest neighbor pixel pairs as shown in figure 4.

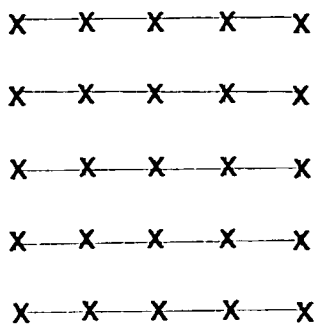


Figure 4 A square 5 x 5 neighborhood

The earliest statistical texture extraction method is based the gray-level difference operator [WDR76]. At each pixel, the gray-level (pixel intensity value) difference between the pixel and a nearest-neighbour pixel separated by a fixed distance $d$ is computed. Alternatively, the average gray-level difference between neighborhoods with separation $d$ can also be used. These values are used to construct a histogram $H_G$ of size $n$ where $n$ is the number of distinct gray-level or average grey-level difference values computed. The elements in $H_G(g)$ are defined as the relative frequency of each distinct grey-level value or average grey-level difference value $g$.

From $H_G$, the texture element size (coarseness) can be derived by measuring the spread in the values of $H_G$ [TMY76]. For example, if a texture is coarse, and $d$ is small compared to the texture element size, the pairs of pixels at separation $d$ should usually have similar gray levels, so that the values in $H_G$ should usually be small (i.e. the values in $H_G$ should be concentrated near $g = 0$). Conversely, for

a fine texture with $d$ comparable to the element size, the gray levels at distance $d$ should often be quite different (i.e. the values in $H_G$ should be more spread out). Thus a good way to analyze texture coarseness would be to compute $H_{\Delta G}(i)$, for various magnitudes of $d$, some measure of the spread of values in $H_{\Delta G}(i)$ away from the origin. The mean is one such measure and is computed as follows [WDR76]

$$\text{MEAN} = \frac{1}{n}\sum_{i}^{n} iH_{\Delta G}(i) \tag{8}$$

This is small when the $H_{\Delta G}(i)$ are concentrated near the origin and large when they are far from the origin.

### 2.4.1 The MRSAR model

The MRSAR [KZL94] model (multi-resolution simultaneous autoregression model) is based on the SAR model (simultaneous autoregression model) which measures texture granularity (texture element size) for a single image resolution. To better measure texture granularity, the MRSAR model computes the SAR features at various lower level images. The lower level resolutions are obtained by successively low-pass filtering and subsampling previous level images. Specifically, two model parameters (texture features) $\sigma$ and $\theta(r)$ are computed for each image resolution. $\sigma$ measures texture granularity in which a higher $\sigma$ implies more granularity and vice versa. The parameters $\theta(r)$ at each pixel $p = (p_x, p_y)$ indicate the pixel's dependence on its neighbors over a $4 \times 4$ square neighborhood

$\Omega$. For vertically oriented textures $\theta(0.1)$ and $\theta(0.-1)$ will have higher values. This dependence is expressed as,

$$g(p) = \mu + \sum_{r \in D} \theta(r) g(p+r) + \epsilon(p) \tag{9}$$

where $\mu$ is the mean image brightness, and $D$ is the set of neighbors in a square neighborhood. $\varepsilon(p)$ is an independent Gaussian random variable with zero mean and variance $\sigma^2$. Computing the SAR model texture features requires estimating parameters $\sigma$ (texture granularity) and $\theta(r)$ by using least squares estimation (LSE) or maximum likelihood estimation (MLE).

The collection of model parameters for each lower image resolution is chosen as the texture features. The 4 resolution MRSAR model has been found to give the best results [KZL94].

### 2.4.2 Tamura's texture features

Tamura's texture features [TMY78] have the highest correspondence to visual texture perception reported in literature. In particular, texture is described in terms of three parameters: coarseness, contrast and directionality. Coarseness measures the scale of texture elements (e.g. pebbles versus boulders); contrast measures the vividness of texture patterns; and directionality indicates whether texture has a preferred direction like grass or is non-directional like a smooth object. The algorithms to compute each feature are given below. [KZL94].

#### 2.4.2.1 Contrast

The contrast measure $F_{\text{con}}$ is a function of the distribution of pixel brightness. It is computed as

$$F_{\text{con}} = \sigma/\alpha^{\frac{1}{4}} \tag{10}$$

where $\sigma$ is the standard deviation of pixel brightness and $\alpha$ is the kurtosis. The kurtosis $\alpha$ is computed as,

$$\mu_4/\sigma^4 \tag{11}$$

where $\mu_4$ is the fourth moment about the mean. This provides a measure of the global contrast in the image [KZL94].

#### 2.4.2.2 Coarseness

The coarseness measure $F_{crs}$ is derived from a size detector which computes moving averages over different neighborhoods sizes e.g. $1 \times 1, 2 \times 2, \ldots, 32 \times 32$ at each pixel. The best size is chosen as follows: At pixel $p(p_x, p_y)$, the moving average over a window of size $2^k \times 2^k$ is

$$A_k(p_x, p_y) = \sum_{i=x-2^{k-1}}^{x+2^{k-1}-1} \sum_{j=y-2^{k-1}}^{y+2^{k-1}-1} g(p_x, p_y)/2^{2k} \tag{12}$$

where $g(p_x, p_y)$ is the brightness at pixel $p(p_x, p_y)$. Differences between the moving averages in horizontal and vertical directions are computed as

$$E_{k,h}(p_x, p_y) = \left| A_k\left(x + 2^{k-1}, y\right) - A_k\left(x - 2^{k-1}, y\right) \right| \tag{13}$$

for the horizontal and similarly for the vertical case. At each pixel, the best size $k$ is the one which maximizes $E$ in either direction

$$S_{\text{best}}(x, y) = 2^k \tag{14}$$

The overall coarseness measure is defined as,

$$F_{\text{crs}} = \frac{1}{m \times n} \sum_{i}^{m} \sum_{j}^{n} S_{\text{best}}(i, j) \tag{15}$$

where $m$ and $n$ are the dimensions of the image. The computation of this measure can be modified to apply to small objects or regions. In order to reduce the amount of computation for this measure, a lower image resolution can be used with fewer window sizes.

#### 2.4.2.3 Directionality

The directionality measure $F_{\text{dir}}$ is obtained by computing the gradient magnitude,

$$|\Delta G| = (|\Delta_H| + |\Delta_V|)/2 \tag{16}$$

and gradient angle

$$\theta = \arctan(\Delta_V / \Delta_H) + \frac{\pi}{2} \tag{17}$$

at each pixel. The horizontal and vertical differences $\Delta_V$ and $\Delta_H$ are computed using $3 \times 3$ operators. A histogram of $\theta$, $H_D$ is constructed by quantizing and counting those pixels for which magnitude $|\Delta G|$ is larger than a threshold. The

histogram will be flat for non-directional images and strongly peaked for highly directional images. The overall directionality is measured as the sum of the widths (sharpness) of peaks in the histogram i.e.

$$F_{\text{dir}} = \sum_{p}^{n_p} \sum_{\phi \in w_p} (\phi - \phi_p)^2 H_D(\phi) \tag{18}$$

where $n_p$ is the number of peaks, $\phi_p$ is the $p$th peak position of $H_D$ and $w_p$ is the range of the $p$th peak between valleys. Only those peaks are considered for which certain ratios between the peak and surrounding valleys are satisfied.

## 2.5 Texture-retrieval systems based on rectangular subdivision

In this section, we examine two texture-similarity image retrieval systems based on rectangular subdivision. Each system divides the database images into rectangular regions forming a grid of *partition units*. The grid is superimposed on each image by dividing its horizontal and vertical dimensions into equidistant intervals. The number of intervals in each direction may be different. When features are computed, the location of each pixel relative to the partition unit is determined. Pixels belonging to a partition unit will only contribute to the features extracted for that partition unit. The size of the partition units is determined by the number of horizontal and vertical partition units and the image size. Some

typical grid sizes are given in figure 5.

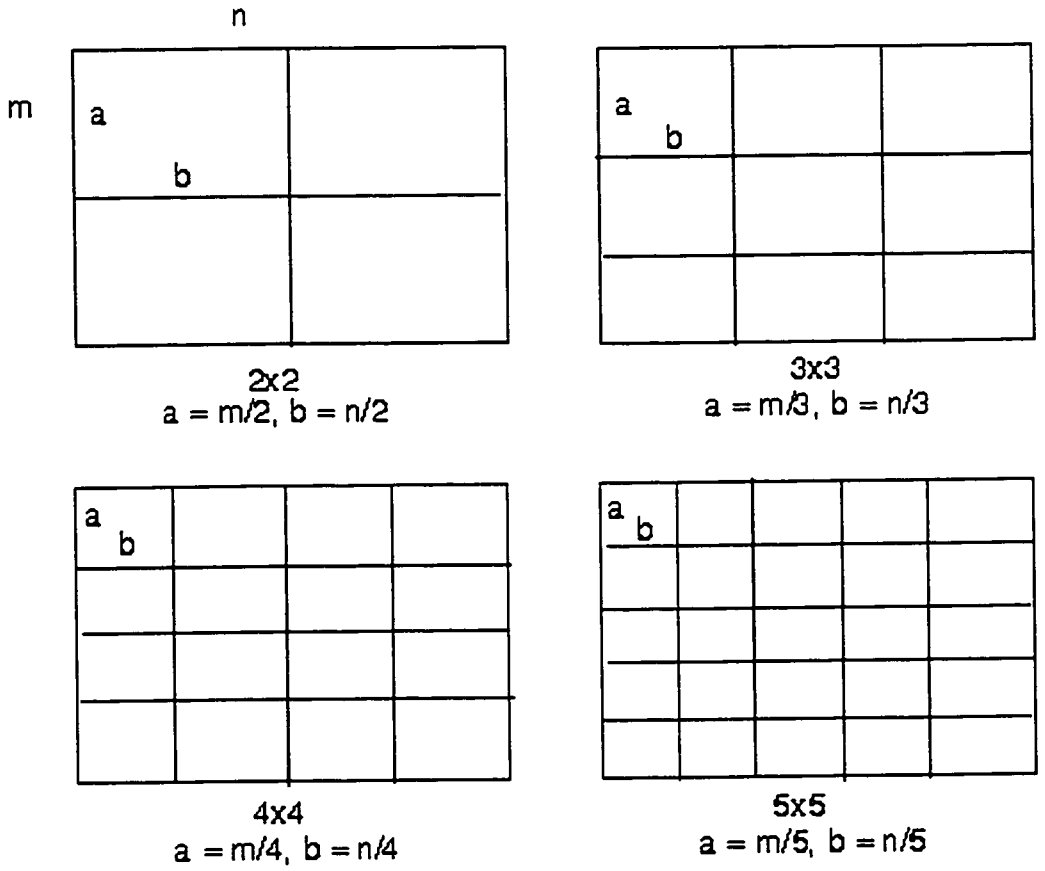



Figure 5 Examples of rectangular partition sizes

### 2.5.1 Partitioned-based query-by-paint

The partition-based query-by-paint method [ABF95] is part of a larger system called QBIC (Query by image content) [NBE93a] which supports querying on other image properties such as color layout and shape. Preprocessing divides each database image into a grid of 6 vertical by 8 horizontal blocks. Tamura's texture features are computed and stored for each grid block for each database image.

Queries are specified using the query-by-paint method. The user draws or paints an approximate color/texture drawing. This is done by selecting a color and/or texture sample and outlining the image area where these features are desired. When the user completes the drawing, Tamura's texture features are extracted from the image, with the exception that unspecified parts in the drawing are excluded. The extracted features are then processed into distinct query "objects". An object is a connected component (8–connected in the grid structure) of grid blocks where some query data is specified [ABF95].

Retrieval is based on direct-matching. The matching algorithm computes a score for a given image by scoring each query object against the image, and taking the average query object score. The query object score is the average score of each of its grid elements. The grid element score is computed as a function of the element-to-element score over a corresponding neighborhood in the image.

This is computed using:

$$d_{\text{grid}(i,j)} = \min_{(x,y \in \text{nbr}(i,j))} \{w(x,y) d_{\text{elem}}(x,y)\} \tag{19}$$

where $w(x,y)$ is a weight which takes the value 1 at the centre of the neighborhood and decreases as the distance from the center increases. For example, to score the query object element at grid location $(i,j)$, the grids of the image at locations $\{i-1, i, i+1\} \times \{j-1, j, j+1\}$ are searched. The location $(i,j)$ is weighted more heavily to favor correct spatial alignment, and the final score is the minimum of the individual scores. The element-to-element distance score $d_{elem}(x,y)$ is computed using 3D Euclidean texture distance. This is computed as

$$d_{elem}(x,y) = \sqrt{(C_x - C_y)^2 + (O_x - O_y)^2 + (d_x - d_y)^2} \tag{20}$$

where $C_x$ and $Cy$ are the contrast measurements; $O_x$ and $O_y$ are the coarseness measurements; and $D_x$ and $D_y$ are the directionality measurements for query block $X$ and image block $Y$.

### 2.5.2 Partitioned-based query-by-image example

The system in [KZL94] supports texture-similarity retrieval using Tamura's texture features and the MRSAR model features. The database images are divided into a grid of 16 × 16 rectangular partition units. Both Tamura's features and the MRSAR features are computed in each partition unit of each database image and are stored in a separate index. Query-by-image example is the only supported

query specification method. The parameters of the MRSAR model are estimated using maximum likelihood estimation for three different reduced image resolutions $128 \times 128, 64 \times 64, 32 \times 32$. Both model parameters $\sigma$ and $\theta(r)$ are computed in a symmetric neighborhood $\{(0,1),(1,0)(1,1),(1,-1)\}$ centered at each pixel.

Retrieval is based on direct-matching. A database image is scored against the query image by computing the average Mahalanobis texture distance [KZL94] over corresponding partition units in the database and query image. This distance is computed as

$$D_{\mathbf{mahal}}(B^1, B^2) = (B^1 - B^2)C^{-1}(B^1 - B^2) \tag{21}$$

where $B^1$ and $B^2$ are the MRSAR texture features in corresponding partition units of the query and image, respectively, and $C$ is the feature covariance matrix.

In retrieval based on Tamura's features, the texture distance between corresponding pairs of partition unit is computed using simplified Mahalanobis texture distance. This is computed as,

$$D_{\mathbf{simpl}}(B^1, B^2) = \sum_{i=1}^{3} \frac{(T_i^1 - T_i^2)}{c_i} \tag{22}$$

where $T^1$ and $T^2$ are the Tamura's features in corresponding partition units of the query and image, respectively, and $c_i$ is the variance of feature $i$.

### 2.5.3 Limitations of rectangular subdivision in direct matching

Because the regions in direct-matching are compared in-place, the position of objects relative to the regions in subdivision is important. If two images contain

the same objects but the objects are arranged differently in each image then they will not be considered to match. Thus even small changes in object position such as small translations which shift portions of an object into another region will cause most direct match comparisons to fail. To improve the results in direct region matching, it is necessary to reduce this dependence on object position. One way to do this is to use fuzzy regions to subdivide the image. In rectangular subdivision, each region is defined by a standard set. This means that each region point has equal influence. In terms of set membership this is analogous to saying each point has equal membership. Fuzzy sets on the other hand radically depart from this and allow points to have graded membership. Thus some points in a region may have more influence or greater membership than others in the region.

In direct-matching, the ability to vary the weight of each pixel in a region can be used to control its influence in the region's feature measurements. In rectangular subdivision, each pixel has the same weight in the feature measurements. By allotting the most influence to pixels in the centre of each region and by allowing the regions to overlap, the effects of small object translations and rotations near the region boundary can be marginalized. Using graded membership in fuzzy regions, two regions can overlap at their boundary by allowing points on either side of the boundary to have partial membership in both regions. In the example given in figure 6, the points in $D$ excluding those adjacent to the region boundary are given full membership of 1. The points adjacent to the region boundary are

assigned a weight of 0.5 in region $D$ and also in the adjacent regions $A$ and $E$. Similarly, the points in regions A and E adjacent to the region boundary of region $D$ also have a weight of 0.5 in region $D$. Here the texture estimator $F$ is defined

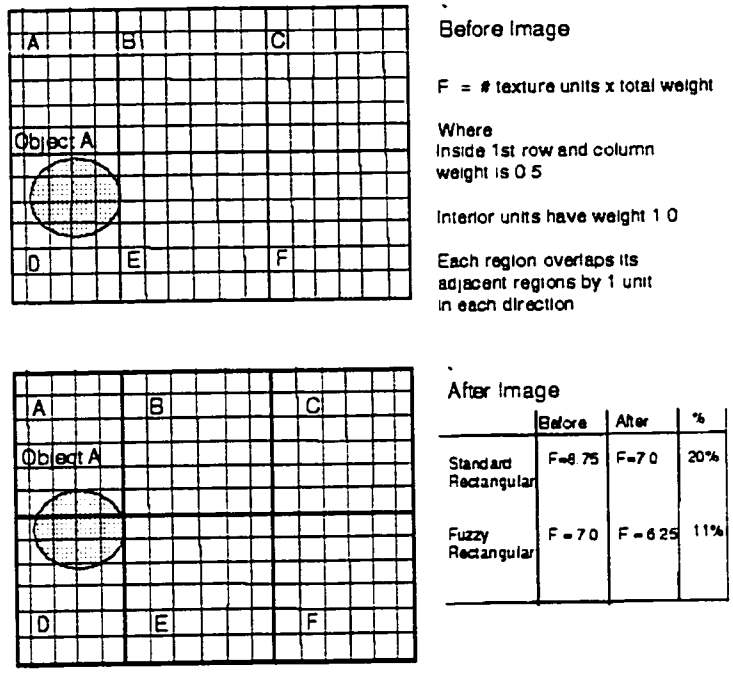



Figure 6 Standard Vs. Fuzzy rectangular partitioning

as the number of shaded points per region times the total weight of the shaded points. The effects of translating Object A in the after image is significantly less pronounced in the fuzzy region-based calculation of $F$ than in the standard region-based calculation of $F$. Thus the effects of small object position changes in direct-matching can be better accounted for using fuzzy regions.

Large object rotations about the image centre in multiples of 90 degrees can be accounted for using centrally arranged image regions (as seen in figure 7).

Here the changes due to the major object rotations (90, 180 and 270 degrees) are handled in the image-to-image similarity function using a scoring technique called minimum permutation similarity scoring [DS96]. By scoring the query image against 3 rotated versions of the database image (90, 180 and 270 degrees), any object rotations in the query image with multiples of 90 degrees about the image centre can be accounted for. This is illustrated in figure 7. Here Object A in the query image is rotated by 90 degrees from region *D* into region *A* in the database image. The minimum permutation similarity scoring technique correctly compares region *D* in the query image with region *A* in the database image resulting in the highest similarity score. On the other hand, conventional direct-matching will result in a low similarity score by matching region *D* strictly in place.

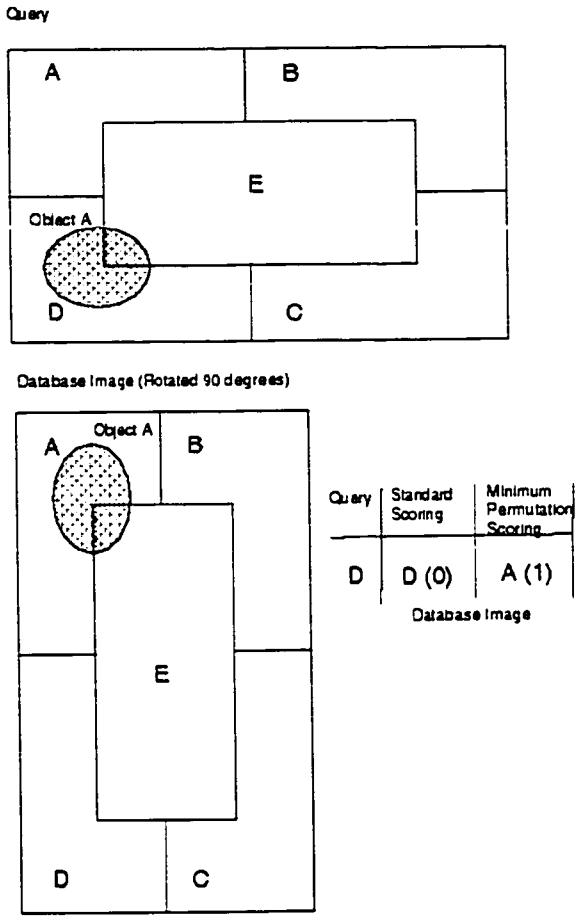



Figure 7 A centrally partitioned image with overlapping rectangles

# Chapter 3 FUZZY IRREGULAR IMAGE SUBDIVISION

To address the undesirable dependence on object position in direct-matching, we use the fuzzy irregular subdivision (FIS) method proposed in [DS96]. Instead of using rectangular regions, this method irregularly divides the images into a centre region and the four corner regions as seen in figure 8. From these regions we will define our texture features. In this chapter, we give the details of this method.

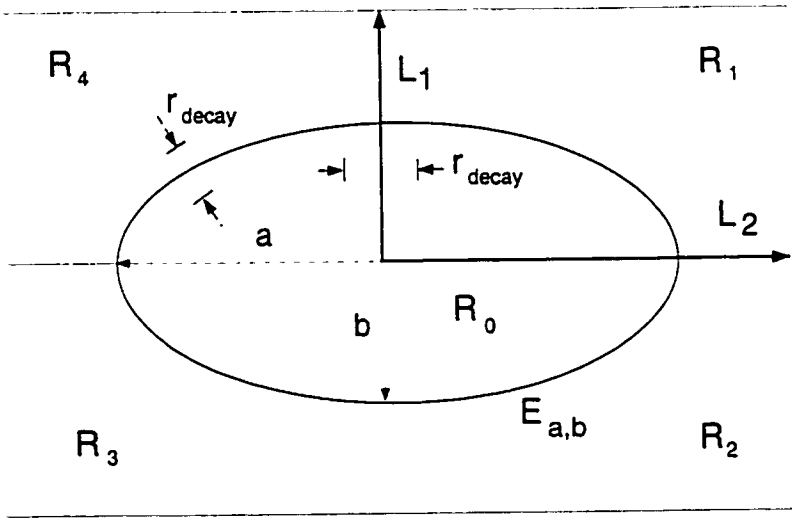



Figure 8 Regions in fuzzy irregular subdivision

Because the regions partially overlap, features computed from these regions become insensitive to small changes in object position. The regions also provide a more meaningful partitioning than rectangular regions. The images in our database

are images of real scenes, such as photographs of wildlife, natural water resources, buildings and people[1] in two different formats: portrait and landscape formats. In both formats the main objects are always placed in the centre. From this two assumptions can be made [DS96]:

1. The centre of the image is very important (i.e. contains the most important pictorial information).
2. It is not necessary to achieve invariance of the features of the regions with respect to arbitrary rotations and translations (which is unrealistic and almost impossible to achieve).

Thus the feature invariance problem with respect to object position can be simplified by restricting the set of transformations to which we want the features to be invariant to small rotations around the center and small translations (denoted by $T_{\text{small}}$), and larger rotations around the centre in multiples of 90 degrees (denoted by $T_{90}$ [DS96].

## 3.1 Region definition

The centre region $R_0$ is defined by an ellipse $E_{a,b}$ with semi-axis $a$ and $b$ centered in the image. Since the corner regions are symmetrically arranged around the centre it is only necessary to define one of them. The others can be obtained by the obvious reflections. With the origin of the coordinate system in the top-left

[1] The images are available via ftp from the Chabot Project at the University of California, at Berkeley

corner of the image, the top-right region, $R_1$, is defined by two straight lines (see figure 7):

$$\begin{aligned} l_1 &: x = \frac{1}{2}(n + r_{\text{decay}}) \\ l_2 &: y = \frac{1}{2}(m + r_{\text{decay}}) \end{aligned} \tag{23}$$

where m and $n$ are the horizontal and vertical image dimensions, respectively. The parameter $r_{\text{decay}}$ in the equations above is user defined and defines the width (in pixels) of the fuzzy borders of the regions in figure 8. This is seen as the dotted lines surrounding each region. Within the fuzzy borders, the image intensity in each region falls off with *cosine-like* decay. This allows the texture features of the regions to be less sensitive to the transformations in $T_{\text{small}}$. Even with this restricted set of transformations, absolute invariance of the features of the regions is still impossible. Instead the next best option is to make the features continuously depend on the transformations in $T_{\text{small}}$ [DS96]. This is achieved by defining each region as a fuzzy set.

Fuzzy sets unlike standard sets define set membership to be a graded continuum in the unit interval $[0, 1]$. Thus region membership is expressed mathematically as,

$$f_{\text{region}}(P) = \{u | u \in [0, 1].\ P \in \text{region}\} \tag{24}$$

where $P$ is a pixel belonging to a region in FIS. Using fuzzy sets to define each region allows the contribution made by each pixel in the features of the region to be

manipulated by weighting the pixel by its fuzzy membership value. Thus feature invariance in the regions to small changes in object position can be achieved by weighting pixels near the region boundary less strongly (i.e. membership <1) (See §2.5.3). The pixels in the centre of each region are assigned a weight of 1 ($f_{\text{region}}(p) = 1$) so that they have the most influence in the feature measurements of the region. This has the effect of focusing the feature measurements in each region on the centre area of the region.

The fuzzy membership functions for the centre and top-right corner regions are given below. A distance function is used in each region membership function to facilitate a smooth decrease in membership for values outside the region. The fuzzy membership for the centre region $R_0$ is given by [DS96]:

$$R_0(P) = \begin{cases} 1 & \text{for } P \text{ inside } E_{a,b} \\ \frac{1}{2}\left(\cos\left(d_{\text{ellipse}}(P, E_{a,b})\frac{\pi}{2}\right) + 1\right) & \text{for } P \text{ outside } E_{a,b} \end{cases} \tag{25}$$

where $d_{\text{ellipse}}$ is a distance function which approximates the distance between $P$ and the ellipse $E_{a,b}$. The fuzzy membership function for the top-right region $R_1$ is defined by [DS96]:

$$R_1(P) = \begin{cases} 0 & \text{if } x \leq \frac{1}{2}\left(m - r_{\text{decay}}\right) \text{ or } y \leq \frac{1}{2}\left(n - r_{\text{decay}}\right) \\ 1 - R_0(P) & \text{if } x \geq \frac{1}{2}\left(m + r_{\text{decay}}\right) \text{ and } y \geq \frac{1}{2}\left(n + r_{\text{decay}}\right) \\ \frac{1}{2}\left(\cos\left(d_{\text{surround}}(P, l_1, l_2)\frac{\pi}{r}\right) + 1\right).(1 - R_0(P)) & \text{otherwise} \end{cases} \tag{26}$$

where $d_{\text{surround}}$ computes the minimum distance between $P$ and the lines $l_1$ and $l_2$

## 3.2 The FIS Algorithm

The FIS algorithm computes a table of weights for each region using the pixel weighting scheme described above. The weights are also incorporated in each region feature measurement to map each image pixel to the region to which it belongs. Thus for pixels $P$ lying outside a given region their weight in the region is defined to be 0 (i.e. $F_{\text{region}}(P) = 0$). Consequently, only the pixels belonging to a region will take part in its features measurements.

A table $LUT_{R_l}(x, y), l = 0, 1, 2, 3, 4$ of size $m \times n$ is computed for each region containing the weights for each image pixel with respect to a region. The image pixels in each table are weighted differently based on the weights assigned by the region membership function. To compute each $LUT_{R_l}$, the pixels are processed in scanline order (i.e. left-to-right and then top-to-bottom). The pixels coordinates are given with respect to the origin in the centre of the image. The transformation below is used to transform each image pixel to this coordinate system:

$$P'(x, y) = P\left(x - \frac{n}{2}, y - \frac{m}{2}\right) \tag{27}$$

Input to the algorithm consists of the length of the semi-major axis $n_a$ (specified as a fraction of the image width $n$) and the length of the semiminor-axis $n_b$ (specified as a fraction of the image height $m$). The input parameters are depicted in figure 9 and the algorithm is outlined below [DS96]

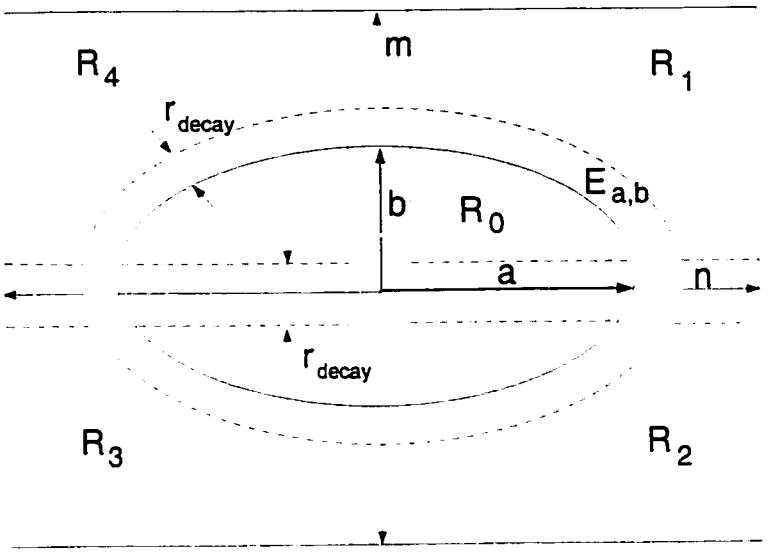



Figure 9 The parameters used in computing FIS region weights

For each image pixel in scanline order $P(x, y)$ do the following steps:

1. Compute the weight $LUT_0(x, y)$ of $P(x, y)$ in the centre region $R_0$.

   a. Calculate the semi-axis radius $r_a$ and $r_b$ of $a$ and $b$, respectively:

$$r_a = n_a \cdot \frac{n}{2}$$
$$r_b = m_b \cdot \frac{m}{2} \tag{28}$$

   and the distance $d_{\text{ellipse}}(x, y)$ of $P(x, y)$ from the ellipse $E_{a,b}$

   b. If

$$\left(\frac{x}{r_a}\right)^2 + \left(\frac{y}{r_b}\right)^2 \leq 1 \tag{29}$$

   then

$$LUT_0(x, y) = 1 \tag{30}$$

(i.e. if the pixel lies inside the ellipse then this weight is 1)

c. if

$$-\frac{r_{\text{decay}}}{2} \leq d_{\text{ellipse}}(x, y) \leq \frac{r_{\text{decay}}}{2} \tag{31}$$

then

$$LUT_0(x, y) = \frac{1}{2}.\cos\left(d_{\text{ellipse}}(x, y).\frac{\pi}{r_{\text{decay}}}\right) \tag{32}$$

(i.e. if the pixel lies inside the fuzzy border then this weight is *cos* like).

d. if

$$d_{\text{ellipse}}(x, y) > \frac{r_{\text{decay}}}{2} \tag{33}$$

then

$$LUT_0(x, y) = 0 \tag{34}$$

(i.e. if the pixel is both outside the ellipse and its fuzzy border then this weight is 0)

2. Compute the weight $LUT_1(x, y)$ of $P(x, y)$ in the top-right corner region $R_1$

a. if

$$\left(x \leq -\frac{r_{decay}}{2}\right) \bigvee \left(y \leq -\frac{r_{decay}}{2}\right) \tag{35}$$

then

$$LUT_1(x, y) = 0 \tag{36}$$

(i.e. if the pixel lies to the left of the left fuzzy border or below the bottom fuzzy border its weight is 0).

b. if

$$\left(x \geq \frac{r_{decay}}{2}\right) \bigwedge \left(y \geq \frac{r_{decay}}{2}\right) \tag{37}$$

then

$$LUT_1(x, y) = 1 - LUT_0(x, y) \tag{38}$$

(i.e. if the pixel is to the right of the left fuzzy border and above the bottom fuzzy border its weight is one minus its weight relative to the ellipse).

c. if

$$-\frac{r_{\text{decay}}}{2} \leq x \leq \frac{r_{\text{decay}}}{2} \tag{39}$$

and

$$-\frac{r_{\text{decay}}}{2} \leq y \leq \frac{r_{\text{decay}}}{2} \tag{40}$$

then

$$LUT_1(x, y) = \max\left[\frac{1}{2}\left(1 - LUT_0(x, y).\left(\cos\left(\min\left(d_{\text{left}}, d_{\text{bottom}}\right).\frac{\pi}{r_{\text{decay}}}\right) + 1\right)\right)\right] \tag{41}$$

where

$$d_{\text{left}}(x, y) = \frac{r_{decay}}{2} - x \tag{42}$$

and

$$d_{\text{bottom}}(x, y) = \frac{r_{decay}}{2} - y \tag{43}$$

(i.e. if the pixel is in the left or bottom fuzzy border its weight is *cos* like

3. Repeat steps 1 and 2 until all image pixels have been processed.

4. The weights for the other corner regions $R_l$, $l = 2,3,4$ are computed using the obvious reflections of the pixel weights computed for $R_1$. These are given below.

   a.

$$LUT_2(x,y) = LUT_1(n-x,y) \tag{44}$$

   b.

$$LUT_3(x,y) = LUT_1(n-x,m-y) \tag{45}$$

   c.

$$LUT_4(x,y) = LUT_1(x,m-y) \tag{46}$$

# Chapter 4 TEXTURE FEATURES FOR FUZZY REGION-BASED EXTRACTION

From each of the fuzzy regions defined in fuzzy irregular subdivision, we propose to extract four texture features: fuzzy-region based coarseness, contrast, directionality and line-likeness. We collectively refer to these as the fuzzy-region based texture features (FRBT). The algorithms for these features are based on the Tamura texture feature counterparts [TMY78] having been modified to incorporate the fuzzy weighting scheme in the FIS algorithm. Thus from each of the five fuzzy regions, we extract four texture features yielding a feature vector with four texture feature components. Consequently, the resulting index entry for each image will contain 20 feature values (i.e. 4 components x 5 regions). In the sections below we outline the algorithms to compute each FRBT feature.

## 4.1 Fuzzy region-based coarseness

In each region we need to compute a measure of coarseness. We propose to weigh each best window size (the size detector) computed at each pixel in the original coarseness algorithm by the pixel's weight (the fuzzy membership value) in each region. Thus in each fuzzy region we use the pixel weighted sum of the best window sizes to compute the average window width, the final coarseness measure in the region. The motivation is this: to take the largest contribution from gray averages computed over windows (square neighbourhoods

of size $2^k \times 2^k$, $k = 1, 2, ..., 5$) centered at pixels in the centre area of each region and not near the region boundary. This will focus the coarseness measurement on the centre of the region and mitigate the dependence on small changes in object position near the region boundary. Moreover, these changes are also mitigated since the contributions from pixels near the region boundary in adjacent regions will also be considered given the regions slightly overlap. The algorithm to compute the FRBT coarseness measure $F^l_{crs}$ in region $R_l$ is outlined below (the image pixels are processed in scanline order):

1. For each pixel $(x, y)$, compute the average grey-level in square neighborhoods with sizes $1 \times 1, 2 \times 2, ..., 32 \times 32$

   a. The average grey-level at $(x, y)$ over the window $2^k \times 2^k$ is computed as,

   $$A_k(x, y) = \sum_{i=x-2^{k-1}}^{x+2^{k-1}-1} \sum_{j=y-2^{k-1}}^{y+2^{k-1}-1} G(i, j)/2^{2k} \tag{47}$$

   where $G(i, j)$ is the gray-level intensity at $(x, y)$

2. For each image pixel $(x, y)$ take the difference between averages in non-overlapping neighborhoods on opposite sides of $(x, y)$ in both the horizontal and vertical directions. The difference in the horizontal direction between two neighborhoods with size $2^k \times 2^k$ is calculated as:

   $$E_{k,h}(x, y) = \left| A_k\left(x + 2^{k-1}, y\right) - A_k\left(x - 2^{k-1}, y\right) \right| \tag{48}$$

   a. The best window size at $(x, y)$ is selected as the window size $2^k$ which yields the largest average grey-level difference in either direction at $(x, y)$.

The best window width $2^k$ is then accumulated in $R_l$ as

$$ACCUM_l(x,y) = LUT_l(x,y).2^k \tag{49}$$

where $LUT_l(x,y)$ is the fuzzy weight of $(x,y)$ in $R_l$

3. Take the average of the weighted best window widths computed in step 2 to be the final coarseness measure in $R_l$. This is computed as,

$$F^l_{crs} = ACCUM_l(x,y) / \sum_{i=0}^{m-1} \sum_{y=0}^{n-1} LUT_l(x,y) \tag{50}$$

## 4.2 Fuzzy region-based contrast

In the original algorithm, the contrast in an image is computed using the variance of its gray-level distribution. To deal with distributions which are biased to black or white (i.e. where a single peak is high), the variance is adjusted by the kurtosis factor [TMY76], a measure of the amount of polarization in the distribution.

To obtain a contrast measure in each fuzzy region, we propose to compute the weighted variance of the gray-level distribution in each region using the fuzzy weights of the pixels in each region. Similarly, the kurtosis factor (which adjusts each region grey-level variance for polarization) is also weighted in each region using the pixel weights of each region. These modifications will accomplish two things: focus the contrast measurement on the centre area of each region, and consider contributions from pixels close to the region boundary in the adjacent

regions. Together they will help mitigate the dependence on small changes in object position near the region boundary. The algorithm for computing the FRBT contrast measures is given below (the image pixels are processed in scanline order.

1. Calculate $WMEAN_l$ the weighted mean of the gray-level distribution in $R_l$ as,

$$WMEAN_l = \sum_{y=0}^{m-1}\sum_{x=0}^{n-1} G(x,y).LUT_l(x,y) / \sum_{y=0}^{m-1}\sum_{x=0}^{n-1} LUT_1(x,y) \qquad (51)$$

where $G(x,y)$ is the gray-level at $(x,y)$, $LUT_l(x,y)$ is the fuzzy weight at $(x,y)$ in $R_l$ and $m$ and $n$ are the image height and width, respectively.

2. Next compute $WVAR_l$ the weighted variance of the gray-level distribution in $R_l$ using the weighted mean computed in step 2. This is computed as,

$$WVAR_l = \sum_{y=0}^{m-1}\sum_{x=0}^{n-1} (G(x,y) - WMEAN_l)^2.LUT_l(x,y) / \sum_{y=0}^{m-1}\sum_{x=0}^{n-1} LUT_l(x,y) \qquad (52)$$

3. Compute $WKURT_l$ the weighted kurtosis factor in $R_l$ as,

$$WKURT_l = W4MOM_l / WVAR_l^2 \qquad (53)$$

where $WVAR_l$ is computed in step 2 and $W4MOM_1$ is the weighted fourth moment about the weighted mean $WMEAN_l$ computed in step 1 (details of this are given in appendix A).

4. The final contrast measure in $R_l$ is computed as:

$$F_{con}(R_l) = \sqrt{WVAR_l} / (WKURT_l)^{\frac{1}{4}} \qquad (54)$$

5. Repeat steps 1–4 for each region $R_l$, $l = 0, 1, \ldots, 4$

## 4.3 Fuzzy region-based directionality

Directionality identifies whether a textured region has a preferred direction. For example, the texture of a grassy field is naturally directional whereas the texture of sand is smooth and non-directional. Tamura's directionality measure is computed from the histogram of local edge direction frequencies versus local edge direction ($H_D$). It is known that $H_D$ represents sufficiently global features of the image such as long lines and simple curves [MMM73]. The algorithm is based on the fact that gradient is vector and thus has both magnitude and direction. The gradient magnitude $|\Delta_G|$ and local edge direction $\theta$ $(0 \leq \theta < \pi)$ are computed at each pixel in the algorithm using equations 16 and 17 in Section 2.4.2.3. The histogram $H_D$ is then computed by quantizing the local edge direction angle into the intervals:

$$(2k-1)\pi/2n \leq \theta < (2k+1)\pi/2n \tag{55}$$

where $k = 0, 1, 2, \ldots, 15$. The $k$ in the left endpoint of each interval is called the *direction code*. The histogram entry at each direction code $k$, $H_D(k)$ is the number of image pixels which have $|\Delta_G| > t$ and edge direction $\theta$ in the interval bounded by the direction codes $k$ and $k+1$. The thresholding of $|\Delta_G|$ is used to prevent the counting of unreliable edge directions.

To obtain a directionality measure in each fuzzy region, we compute a histogram of weighted local edge direction frequencies (denoted $WHIST_l$) against the local edge direction in each fuzzy region. The entries $WHIST_l(k)$ represent the total weight of pixels in the region which have $|\Delta_G| > t$ and edge direction $\theta$ in the interval bounded by direction codes $k$ and $k+1$. The contribution from each image pixel $(x, y)$ with direction code $k$ to the sum $WHIST_l(k)$ is the weight of $(x, y)$ in $R_l$ (i.e. $LUT_l(x, y)$). The motivation here is that the largest contribution will come from pixels in the centre area of the region and thus marginalize the dependence on small changes in object position near the region boundary. Also pixels near the region boundary in the adjacent regions will also be considered to account for such changes in object position. The steps to compute $WHIST_l(k)$ for each region $R_l$ is given below:

*Part A.* For each pixel $(x, y)$ do the following:

1. Compute the horizontal $\Delta_H$ and vertical $\Delta_V$ gray-level differences at $(x, y)$ using the 3 × 3 operators respectively:

$$\begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{matrix} \tag{56}$$

and

$$\begin{matrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{matrix} \tag{57}$$

2. Compute the magnitude $|\Delta_G|$ and the local edge direction $\theta$ at $(x, y)$ as,

$$|\Delta G| = (|\Delta_H| + |\Delta_V|)/2 \tag{58}$$

and

$$\theta = \arctan(\Delta_V/\Delta_H) + \frac{\pi}{2} \tag{59}$$

3. Determine the direction code $k$ of $\theta$ as

$$k = \theta \; DIV \; \frac{\pi}{32} \tag{60}$$

where the operator *DIV* is integer division.

4. Determine the contribution added to $WHIST_1(k)$ as follows:

if

$$|\Delta_G| \geq 12 \tag{61}$$

then

$$WHIST_l(k) = WHIST_l(k) + LUT_l(x, y) \tag{62}$$

Otherwise go to step 5.

5. Repeat steps 1–4 until all pixels have been processed.

6. Repeat steps 1–5 for each region $R_l$. $0, 1, \ldots, 4$

The directionality measure $F^l_{dir}$ in each fuzzy region $R_l$ is computed from the sum of the widths of peaks (sharpness) in $WHIST_l(k)$. The procedure used to identify the peaks in each histogram $WHIST_l$ is described below.

*Part B.* For each local peak $\phi$ in $WHIST_l(k)$ do the following:

1. if

$$WHIST_l(v_{n,n+1})/WHIST_1(\phi_{n+1}) < 0.5 \tag{63}$$

and

$$WHIST_l(v_{n+1,n})/WHIST_l(\phi_{n+1}) < 0.5 \tag{64}$$

and

$$WHIST_l(\phi_{n+1})/WHIST_l(\phi_n) > 0.2 \tag{65}$$

then $\phi_n$ is a peak. Otherwise $\phi_n$ is not a peak. $v_{n,n+1}$ and $v_{n+1,n}$ are the positions of valleys from local peak $\phi_n$ to the next local peak $\phi_{n+1}$, and vice versa, respectively. These are shown for a directional region in figure 10.

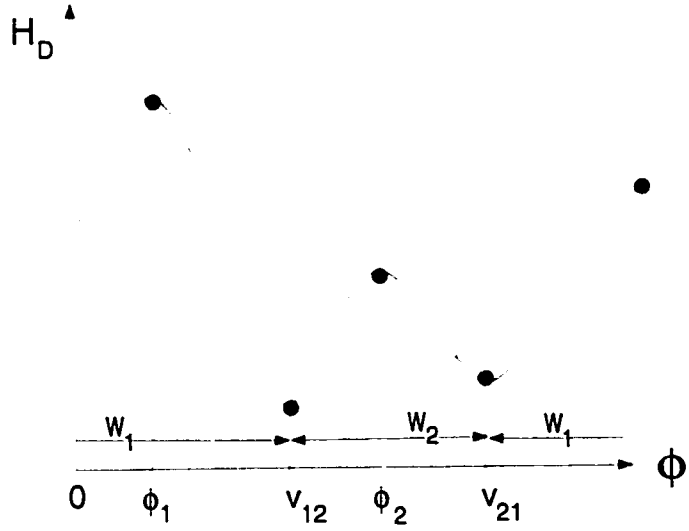



Figure 10 Definition of symbols for directional case

2. The final directionality measure is computed by,

$$F^l_{dir} = 1 - n_p \cdot \sum_{p}^{n_p} \sum_{\phi \epsilon w_p} (\phi - \phi_p)^2 . HIST_l(\phi) \tag{66}$$

where $n_p$ is the number of peaks in $WHIST_l$, $\phi_p$ is the $p$th peak position of $WHIST_l$ and $w_p$ is the range of $p$th peak between valleys.

3. Repeat steps 1 and 2 for each $WHIST_l$, $l = 0, 1, ..., 4$

## 4.4 Fuzzy region-based line-likeness

Line-likeness describes an element of texture composed of lines. The opposite of line-like is blob-like. Tamura's line-likeness algorithm is based on the idea that when the direction and the neighboring direction at two points are nearly equal, the set of edge points can be considered a line [TMY76]. Line-likeness is computed from the local direction occurrence matrix $P_{D_d}$. The elements $P_{Dd_l}(i, j)$ represent the relative frequency with which two neighboring points, separated by distance $d$ in the edge direction occur in the image, one with direction code $i$ and the other with direction code $j$ [TMY76]. This is illustrated in figure 11. The direction codes are computed as in the previous section using equations 59, 60 and 61. Again thresholding is used to prevent the counting of unreliable directions.

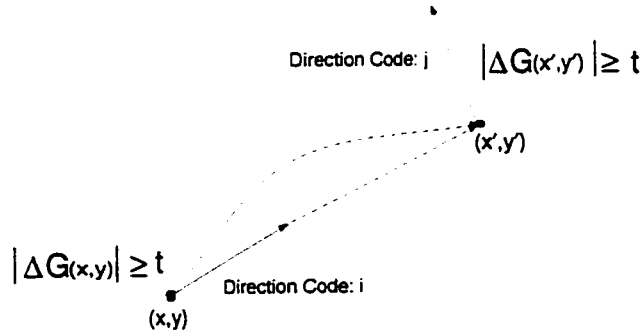



Figure 11 Entry $P_{Dd}(i, j)$ of direction co-occurrence matrix

The edge directions included in $P_{Dd_l}(i,j)$ are the angles formed between a pixel and its four nearest-neighbour pixels in figure 12. This results in a $4 \times 4$

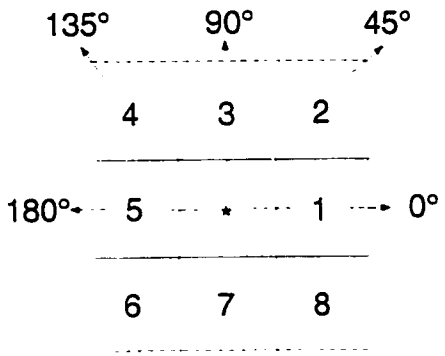



Figure 12 Nearest-neighbour pixel locations and edge directions

direction occurrence matrix of size. To compute a measure of line-likeness in each fuzzy region, we compute a direction occurrence matrix $WMAT_l$ in each fuzzy region. The entries $WMAT_l(i,j)$ represent the total pixel weight with which two neighbouring pixels separated by distance $d$ along the edge direction occur in the region $l$, one with direction code $i$ and the other with the direction code $j$. For the neighboring pixel pair $(x_1, y_1)$ and $(x_1, y_1)$, the pixel weight is chosen as,

$$\min\left(LUT_l(x_1, y_1), LUT_l(x_2, y_2)\right) \tag{67}$$

where $LUT_l(x_1, y_1)$ and $LUT_l(x_2, y_2)$ are the weights of the pixels, respectively. Again, the motivation here is that the largest contribution will come from pixels in the centre area of the region and thus help mitigate the dependence on small changes in object position near the region boundary. Also pixels near the region boundary in the adjacent regions will also be considered to account for such

changes in object position. The steps to compute the direction co-occurrence matrix $WMAT_l(i,j)$ for the region $l$ are given below:

*Part A.* For each pixel $(x,y)$ do the following:

1. Compute the direction code $i$ at $(x,y)$ using the local edge gradient and edge direction as in the previous section.
2. Compute the nearest neighbor $(x',y')$ at distance $d$ from $(x,y)$, along the edge direction $i$. $(x',y')$ is found using the following translations for $d = 4$ (in pixels):
   a. if $i = 0$ then $(x',y') = (x+4,y)$
   b. if $i = 1$ then $(x',y') = (x+4,y+4)$
   c. if $i = 2$ then $(x',y') = (x,y+4)$
   d. if $i = 3$ then $(x',y') = (x-4,y+4)$
3. Compute the direction code $j$ at $(x,y)$.
4. The entry $WMAT_l(i,j)$ is then updated as,

$$WMAT_l(i,j) = \min\left(LUT_l(x,y), LUT_l(x',y')\right) \tag{68}$$

5. Repeat steps 1–4 until all pixels are processed.

*Part B.* The line-likeness measure $F^l_{line}$ in region $l$ is computed by summing the entries $WMAT_l(i,j)$ so that co-occurrences in the same direction are weighted

by +1 and those in the perpendicular direction −1. This is computed as

$$F_{lin}^{l} = \sum_{i=0}^{3}\sum_{j=0}^{3} WMAT_l(i,j).\cos\left[(i-j)\frac{2\pi}{4}\right] / \sum_{i=0}^{3}\sum_{j=0}^{3} WMAT_l(i,j) \qquad (69)$$

where $n = 4$ and $WMAT_l(i,j)$ is the local direction co-occurrence matrix for region $l$.

# Chapter 5 System Design and Implementation

In this chapter, we discuss the design and implementation of the QBT (Query-by-Texture) system, a texture-similarity image retrieval system based on our approach. The system retrieves images using direct-matching on the fuzzy region-based texture features (FRBT) developed in Chapter 4. The QBT system is developed in C++ for Windows NT and provides an HTML form interface for access from the world-wide-web (WWW). An HTML form interface offers platform independent access from typical web browsers, such as Netscape's Navigator and Microsoft's Internet Explorer, available for all major platforms (e.g. UNIX, Macintosh, Windows 95 and Windows NT).

The QBT system consists of the following components:

1. A world-wide-web (WWW) based query-by-example interface, *QBT Form*, for formulating fuzzy region-based texture queries. The interface displays image examples in a hypermedia document called a *photobook*.
2. A world-wide-web based direct-match search engine, *QBT CGI*, which processes the fuzzy-region based texture queries and returns a hypermedia document containing the search results (a ranked set of images matching the user's example image).
3. A batch program, *QBT Tool*, used to create and index the image database using the FRBT features, and create the photobook of image examples.

My aim is to show (through tests devised in Chapter Six) that the FRBT features and the direct-match search algorithm (SSDM) developed in this chapter perform significantly better than Tamura's original features in direct-matching based on rectangular partitioning. The proposed direct-match algorithm is based on the minimum image permutation similarity technique [SD96] described in Chapter 2. The use of this technique together with fuzzy image subdivision helps to reduce the undesirable dependence on object position in direct-matching.

## 5.1 The QBT Design

The QBT design is based on the world-wide-web client/server model. A client/server design partitions a system into two parts [KB96]: a *client*, which is responsible for all interactions with the user and accepts requests for processing, and a *server*, which is a central location for all functionality used in processing the requests made the client. Normally, the client and server are connected over a network, allowing remote access from many concurrent clients. The clients and the server communicate using a standard data protocol, such as HTTP (Hyper-text transfer protocol) which is the data protocal used on the world-wide-web. The WWW client/server model consists of three main components [Gun95]:

1. A *browser* (client) used to request and display documents containing text, images and sound stored locally or on a remote *web server*.

2. A web server which handles the browser's requests for documents and returns them using the HTTP data protocol.

3. A network over which the browser and web-server communicate. This can be either a local (intranet), regional (extranet) or a world-wide network (internet).

Most of the information available on the WWW is stored in *hypermedia* documents. These documents are created in HTML (Hyper-Text Markup Language), which is a syntax of style and format tags used to control the presentation of text, graphics and sound. The browser interprets the HTML and is responsible for displaying the formated content. The language is named after a special tag, called a "hyper-link" which links two hypermedia documents together. Hypermedia documents can be nested to an arbitrary depth, to create a network of documents called a web. The links allow users to navigate the web in a transparent and "point and click" fashion. Each document link contains a URL (uniform resource locator) used by the browser to locate the document in the web a seamless manner.

A hypermedia document's contents may be static — created in advance and stored on the web server — or virtual — created from the output of a program executing on the web server. The program output used in virtual documents must be first converted into HTML by a common gateway interface (CGI). CGI is the part of the web server which allows it to communicate with other programs on the server. Using CGI, a program may linked to a hypermedia document via a

special URL and its results displayed in a browser. This is illustrated in figure 13 [Gun95].

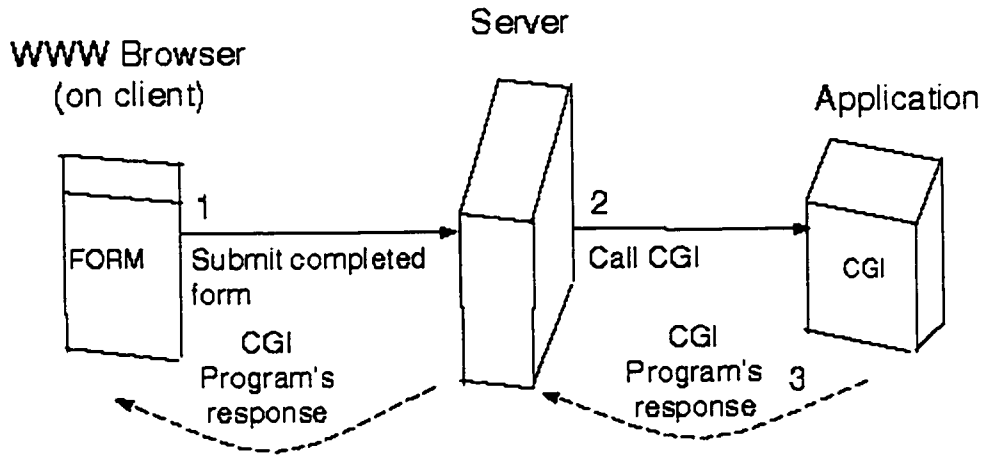



Figure 13 The CGI architecture

A virtual document is requested using an HTML form, a graphical user interface embedded within a hypermedia document. The form is used to collect information from the user which is then submitted to the web server, along with the request for the virtual document. The web server recognizing that the request is for a program and not a static document, then proceeds to call the program, passing it the submitted form data.

Queries are performed in QBT using a form interface. The QBT form interface consists of a static and dynamic area. Each area is displayed in area of the browser

called a *frame*. The static frame contains the form, allowing the user to specify and submit texture queries to the QBT CGI search engine. The QBT form interface is shown in figure 14.

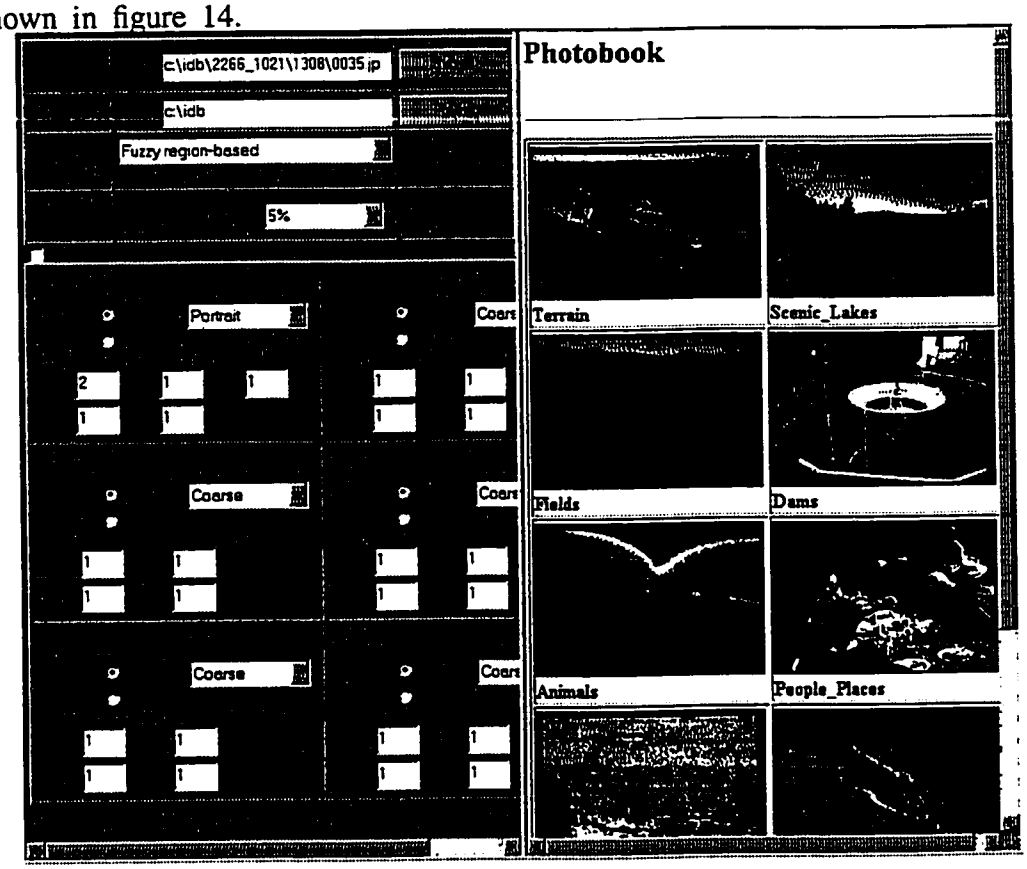



Figure 14 The QBT form interface

The texture query is comprised of the following information:

1. **The relative weight of each fuzzy region.** These weights emphasize or de-emphasize the importance of each region in the direct-match texture query. The region weights may be set individually using a set of text fields or

altogether using region weight presets portrait, landscape, aerial and macro available in a list box.

2. **The feature weights in each region.** These weights emphasize or de-emphasize the importance of each texture feature (coarseness, contrast, directionality and line-likeness) in the region-to-region match values. The feature weights for each region may be user assigned or set using several feature weight presets (coarse, fine, flat, color) available in a list box. Each preset describes a region which is homogeneous in the indicated texture. For example, the coarse preset applies to regions which are predominately coarse, and so on with the other presets. The color preset is for regions which contain hetergenous texture in which none of the other presets apply. The color preset when applied to all 5 regions is used to the query the image by color.
3. **The name and path of the image example and image database.** These are specified in appropriately name text fields.
4. **Retrieval cutoff percentage.** This determines the "cutoff" rank number, specified as a percentage of the total number of images in the database. Images with rank number above the cutoff rank number are discarded from the returned set of images (e.g. images outside of the top 10% are filtered out). A selection list is provided with the available cutoff rank percentage values.
5. **The subdivision method.** The QBT form interface is specifically designed for texture-retrieval based on fuzzy image subdivision. Direct-match texture

queries based on rectangular partitioning are also supported. However, the region and feature weights do not apply in the latter queries.

The dynamic frame has two purposes. First, the photobook — a collection of image examples — is displayed in the dynamic frame for the user to browse. Secondly, it is used to hold the set of images returned from a texture query. The dynamic frame is scrollable to view information which cannot be completely displayed at the same time. The photobook is redisplayed after each query by clicking on the link located in the top-right corner of the static frame. A photobook hypermedia document is created for each image database. It consists of a top-level page with links to several categories of images stored in other hypermedia documents. The image categories are from various domains (e.g. animals, people and places, terrain etc.) and are used to select an image example. The photobook is created manually. The user chooses the image categories and selects a few representative images from the database for each category. The Photobook document is then produced from these images using the QBT tool, which constructs the hypermedia document web underlying the Photobook. An example of a Photobook with the categories terrain, lakes, fields, people and is shown in figure 15.

# Photobook

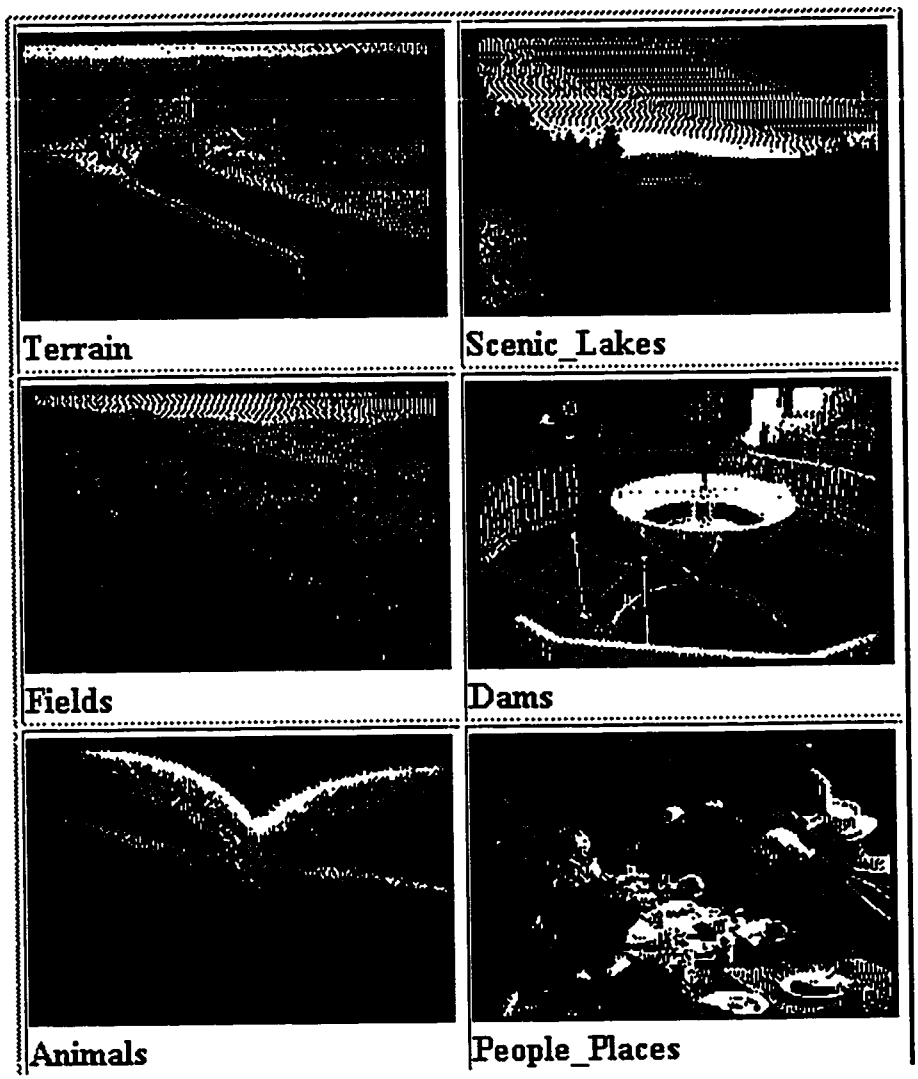



Figure 15 Photobook of image examples

To compose a query, the user completes the QBT form with the region and feature weights and then submits the form for processing. The image example may be selected from the Photobook examples or may be some other image. Choosing a Photobook image will reduce the response time because the image is already indexed in the database, avoiding the costly feature extraction operation. The query processing cycle is depicted in figure 16. The query results are displayed in a two column table where similarity to the image example descrease from top-left to bottm-right

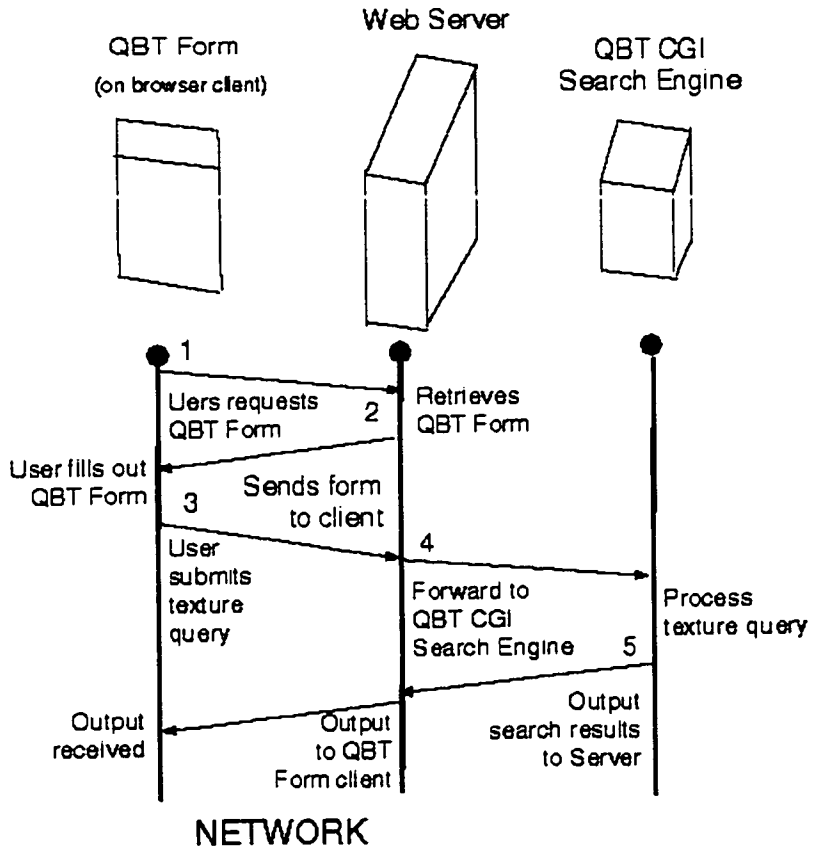



Figure 16 The QBT architecture

### 5.2.1 Database population

Database population begins by creating a directory on secondary storage, where the images are to reside. The images are stored in files in the JPEG image format. This may be created on a fixed or removable hard disk. The database directory may also contain sub-directories, which may be nested to a finite depth.

A directory path $F$ is an ordered list of directories $D_1, D_2, ..., D_n$, such that $D_{i+1} \in D_i$ where $i = 1, 2, ..., n-1$, and $F \in D_n$. $D_1$ is the *root directory* of directory path, and $D_n$ is called the *parent directory* of $F$. The parent directory where the database images are stored is called the *database name*. The database name has a *database path* which is the same as its directory path.

The database images may also be stored in one or more sub-directories. However, the last directory of each sub-directory path must only contain images. The resultant structure is called a *database folder tree*. The path to each image in the tree is called a *folder path*, and its parent directory in the path is called the *folder name*. An example of a database folder tree showing the relationship between folders and images is given in figure 17. The database name is boldfaced and the folder names are in uppercase; the image files are all in lower case.

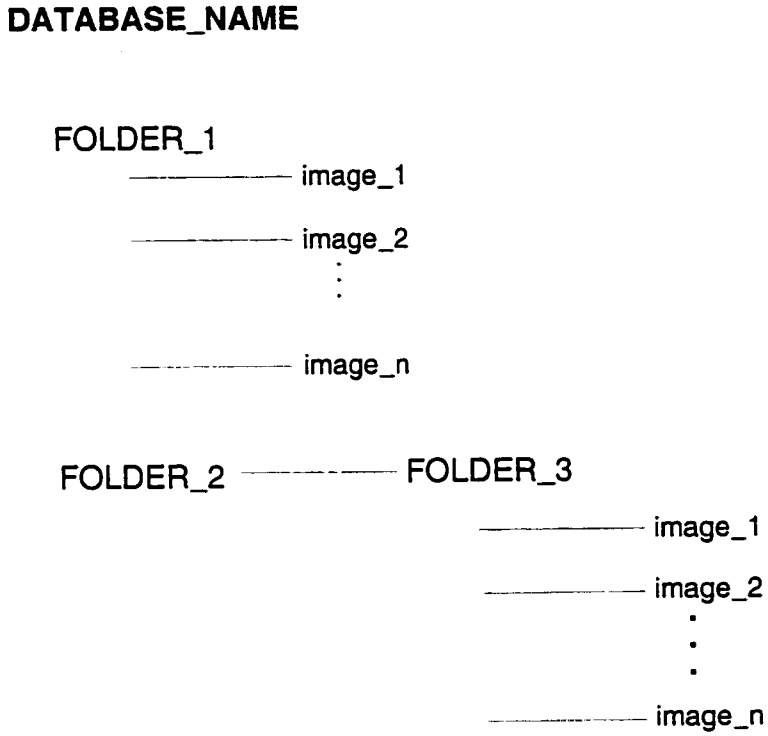



Figure 17 An example of a database folder tree

To reduce the overhead in managing the images, a convenient file naming scheme is used. The images are each named by a 4–digit number, followed by a "." and the 4 character file extension, "jpeg" (the standard JPEG file extension). Within each folder the 4–digit numbers are consecutive but they are not consecutive across folders. Each image is uniquely identified by its file name and folder path.

In order to make the images accessible to the QBT system, the images must be first cataloged. This process involves enumerating each path and image within the database folder tree using a catalog file. For each path in the database folder tree,

the folder path, the total number of images in the folder name, and the file name with the lowest ordinal number is stored. The catalog file layout corresponding to the database folder tree example is given in figure 18. From this information, the filename and folder path of each image in each database folder may be generated.

```
\DATABASE_NAME 0 0

\DATABASE_NAME\FOLDER_1 0 n

\DATABASE_NAME\FOLDER_2 0 0

\DATABASE_NAME\FOLDER_2\FOLDER_3 0 n
```

Figure 18 The catalog file layout for the database folder tree example

The catalog is implemented by the `Catalog` and `Folder` class. Each database folder tree path and its enumeration data is stored in a new `Folder` object. The `Folder` objects read from the catalog are stored in a linked list in a `Catalog` object. The `Catalog` and `Folder` class declarations are given below; only the main data members and methods are shown.

```
class Catalog
{
  char catalogPath[MAX_PATH];
  int numFolders;
  Folder& databaseFolders;
public:
```

```
  AddFolder()
  ReadCatalogFile()
  GetNextFolder()
}
```

The ReadCatalogFile() reads each folder description from the catalog file into a new Folder object. The numFolders data member counts the number of folders read. Each new Folder object is added to the catalog object by the AddFolder() method.

```
class Folder
{
friend: class Catalog

  char folderPath[MAX_PATH];
  int numImages;
  int startImageNum;
  Folder& nextFolder;
public:
  GetNumImages();
  GetNextImage();

}
```

The GetNumImages() and GetNextImage() methods are used to read the information contained in each folder. The latter method reconstructs the image filename to be indexed in the feature extraction operation (see next section).

The last step in database population is to create the photobook of image

examples used in querying the database. The photobook images are stored in a directory referred to as the *photobook parent directory*. The name of this directory is called the *photobook name*. The image examples are stored within sub-directories called *photobook categories*, which correspond to the image categories in the image database (e.g. people and places, terrain, agriculture etc.). A photobook category however cannot be nested and may only contain images. The image examples are selected by the user from images in the database. The selected images are then manually copied into their photobook categories. Once the photobook has been populated, a photobook hypermedia document, used to view the image examples in each category, is generated using the QBT Tool. Details of the QBT Tool are given in the appendix.

### 5.2.2 Feature extraction

From each cataloged image, an index record is created in the feature extraction operation. Each index record contains the texture and color features extracted from each of the five fuzzy regions. The index records are stored in a index file located in the database name directory. Each index record consists of five region descriptions which each contain the FRBT (fuzzy region-based texture) and FRBC (fuzzy region-based color) features of a single region.

The feature extraction operation as with database population is an off-line operation. Both operations are performed using the QBT tool. When a new image

database is created or when images are added to an existing database folder tree, the QBT Tool is used to create or re-create the catalog and index files.

The QBT Tool is implemented by the DB class which provides the framework for all three QBT operations (i.e. database population, feature extraction and image query). The main data members and methods of the DB class are given below.

```
class DB
{
  char databasePath[MAX_PATH];
  char catalogFilePath[MAX_PATH];
  char indexFilePath[MAX_PATH];
  char indexVariantFilePath[MAX_PATH];
  char photobookFilePath[MAX_PATH];
  Catalog databaseCatalog;
  SubMethod indexMethod;
public:
  DBConnect();
  DBCreateCatalog();
  DBCreatePhotobook();
  DBCreateDatabaseIndex();
  DBExecuteQuery();
}
```

During the feature extraction operation, the `databaseCatalog` member (an object of the `Catalog` class) contains the database folder records stored in the catalog file. The database folders are read into memory by the `DBConnect()` method, a mandatory call which specifies the name and path of the image data-

base prior to starting any of the database operations. Before loading the catalog, this method establishes the names and NOS paths of the catalog (`catalogFilePath`), photobook (`photobookFilePath`), index file (`indexPath`), and other system files used in the query operation.

The feature extraction operation produces a system file, called an *index file*. An index is to used efficiently find things in a file or database [FZ88]. In this case the index is used to locate the images stored in the database folders of a database folder tree. I use a simple index which is an ordered linear sequence of index records[FZ88]. The index records each contain one or more *key fields* and a *reference field*. The key field stores the standardized form of the key (here a texture or color feature) on which the images are searched. The reference field stores information on where to find the image containing the color or texture feature stored in the associated key field.

The index created by the QBT Tool is a *multi-key* index. Rather than index each color and texture feature in a separate single-key index, one multi-key index is used for all color and texture features. Multi-key indexes are preferred for multi-feature searches because single-feature indexes would be cost prohibitive for a large database [KB96]. In addition, multi-key access strategies, such as *k-d trees*, *multidimensional tries* and *grid files* can speed up the search by using all features of the multi-key. A review of multi-dimensional access techniques is

given in [KB96]. However such techniques are not used here.[2] As a result, all QBTqueries require an exhaustive and sequential search of the index.

The index record layout is shown in figure 19. The record layout consists of a _FOLDER_NAME label delimiting each new database folder record, followed by a database folder path. The lines which follow are the five region descriptions. The texture and color feature values $f_t^{r,c}$ are stored on a separate lines (where $r$ is the region number, $c$ is the dimension and $t$ is the feature type). The first three lines contain the color features: average color, variance color and covariance color. The color features (FRBC) are computed using the method in [SD96]. Details of this method are also found in the appendix.

The final line contains the FRBT features. Thus each index record consists of a reference field and 65 key fields (9 color + 4 texture) x 5 regions.

The `DBCreateDatabaseIndex()` method extracts the color and texture features from each database image, and creates the index record. This method implements the FRBT algorithms developed in chapter five, and the FRBC algorithms in [SD96].

---

2 My interest here is to improve retrieval effectiveness not query response time

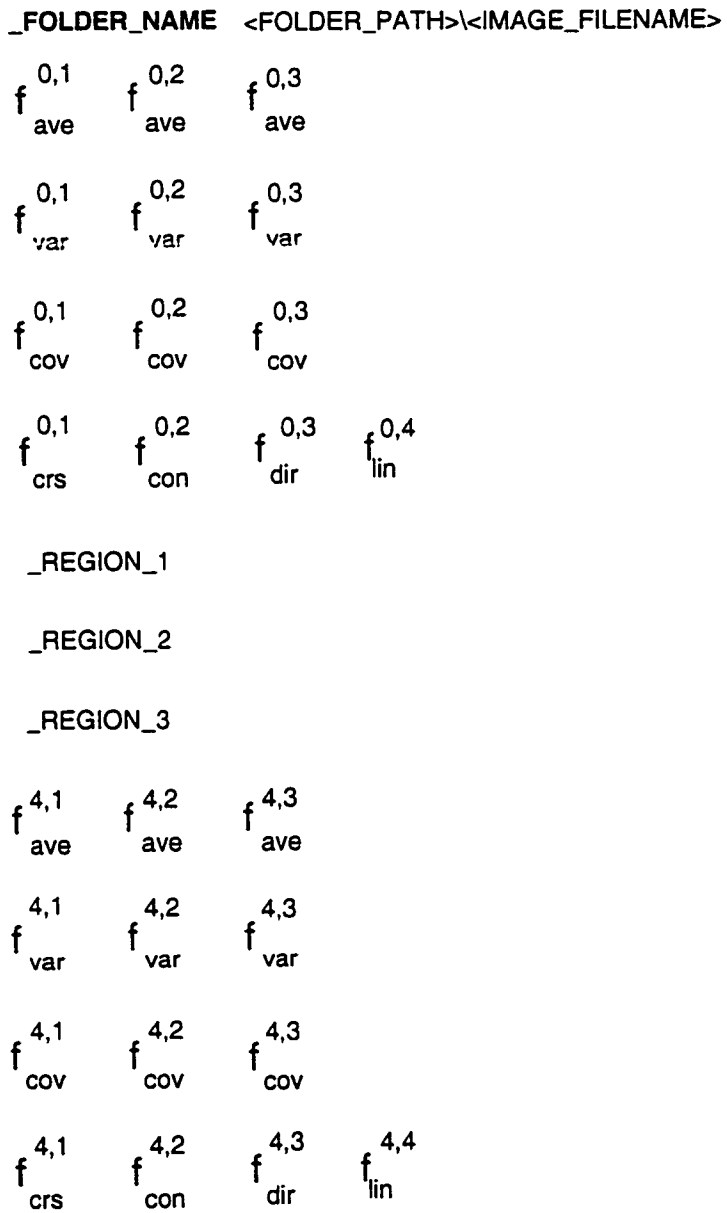



Figure 19 The index record structure

The steps of the `DBCreateDatabaseIndexMethod()` are given below. Several other classes participate in the feature extraction operation which are

briefly described below. A more detailed description of each class follows.

1. Create objects of the support classes:

   a. `Image image`. Object used to read each database image into memory and to access the gray-level value at each pixel.

   b. `Index indexRecord`. The index record in memory.

   c. `SubMethod subMethod`. Stores the fuzzy image subdivision parameters (e.g. $r_a$ semimajor-axis radius, $r_b$ semiminor- axis radius and $r_{\text{decay}}$ decay width.

   d. `FISWeightTable weightTables`. An object which computes the FIS fuzzy region weights. The 5 region weight tables are stored in a $m \times n \times 5$ `Matrix3D` object.

   e. `IndexFile indexfile`. A secondary storage class object which creates and writes to the index file.

   f. `Folder folder`. An object used to store the current database folder being processed.

2. Get the next database folder from the catalog object `databaseCatalog` and get its start image sequence number (`start_seq_no`) and total number of images (`no_folder_images`).

   a. `folder = databaseCatalog.GetNextFolder()`

   b. `start_seq_no = folder.start_seq_no`

c. `no_folder_images = folder.size`

3. Get the next image filename `name` to be indexed in `folder` where `name` is constructed as `{folder.folderName() ++ '\' ++ ToString(i,4), i= start_seq_no,..., start_seq_no + no_folder_images-1}`. "++" is the string concatenation operator and `ToString(i,n)` converts an integer to a 4–character string left padded with zeros.
4. Read the image `name` into the `image` object.
5. Compute the FRBT (texture) and FRBC (color) features from each fuzzy region. The features are stored in the `indexRecord` using a $4 \times 4 \times 5$ `Matrix3D<double>` object. An accumulative total of each texture feature is maintained in the `indexRecord` object which is later used to compute the variance of each feature in each region over all images indexed.
6. Write the index record out to the index file on secondary storage.
7. Repeat steps 3–6 until all images in the current database folder `folderare` processed.
8. Repeat steps 2–7 until all database folders in the catalog are processed.
9. Save feature variants computed by the `indexRecord` object to the feature variant file. This file is stored with the other system files under the database name directory.

A data structure fundamental to the supporting classes used in the `DBCreateDatabaseIndexMethod()` method is the `Matrix3D` template class. A template defines a family of classes using a single generic implementation [Eck93]. For example, a family of matrix classes for different element types (e.g. char, int and double) can be defined using a single template class. Here the template class parameterizes the element type using a template parameter. A new matrix class is created by instantiating objects of the generic matrix class with a new element type. The compiler ensures that the generic matrix class operations work for the new element type. The `Matrix3D` template class defines a three-dimensional matrix and is used in several QBT classes including `Image`, `Index` and `FISWeightTable` classes. The element type can be of any object type. A one–dimensional counterpart, the `Vector` class is also used extensively in QBT. Both these classes were written by and used by permission from Alexander Dimai, another researcher in the CBIR area.

The `Image` class stores the image data (pixels) in a $m \times n$ `Matrix` (a 2D matrix class) object of type *char* (*m* and *n* are respectively the image height and width) which contains the color values of the image. Each color value has 3 components *R*(red), *G* (green) and *B* (blue) which are stored in 3 consecutive bytes for each pixel. A row of color values is called a scanline and represents a single row of pixels. The `ReadImage()` method reads the scanlines in top-to-bottom order.

The `Index` class is composed of a $4 \times 4 \times 5$ `Matrix3D` object of type *double* (i.e. 4 feature vectors of dimension 4 for each of the 5 fuzzy regions). Each $4 \times 4$ 2D matrix contains the color and texture feature values of a region. The features are computed by the `FillColorAndTextureFeatures()`. This method is discussed in more detail at the end of the section. The feature variants in each region are computed during the feature extraction operation using a $5 \times 4$ `Matrix` of `StdMoment` objects. The `StdMoment` class is a statistical moment generator (i.e. the statistical moments of a random variable) used in computing the variance of each feature in each region (i.e. 4 feature values from 5 regions) over all database images. See the appendix for the class details.

The `SubMethod` class consists of a number of strings and numeric values which describe and store the parameters of the image subdivision method used in feature extraction (e.g. rectangular partitioning or fuzzy image subdivision etc.). The `FISWeightTable` class is composed of a $m \times n \times 5$ `Matix3D` object of type `double`. Each $m \times n$ 2D matrix of double values stores the fuzzy region weight table (FIS table) of a fuzzy region. See Chapter Two for more details on the FIS tables and algorithm.

The FRBT feature extraction algorithms are implemented using a hierarchy of texture classes. This is shown in figure 20. At the root of this hierarchy is the `TE` (Texture Extraction) base class. From this base class, a class for each texture feature is derived. The `TE` base class defines a number data members

(such as an `Image` and `FISWeightTable` object) inherited by each derived texture class that are common in each feature extraction operation. Each FRBT

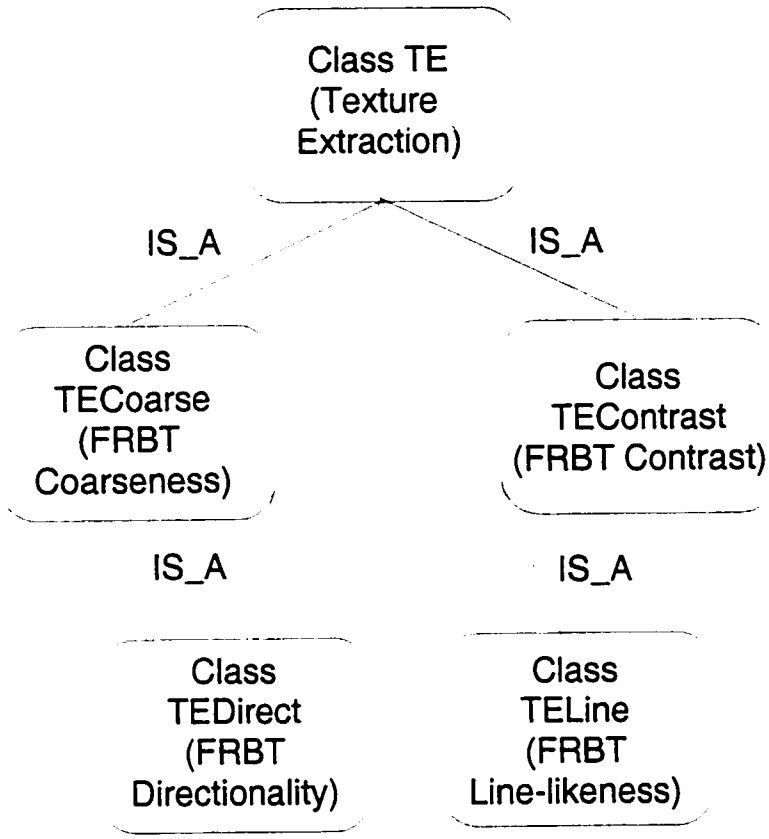



Figure 20 The TE class hierarchy

texture feature in the `FillRegionTextureAndColor()` method is computed using an object of its corresponding derived texture class. This is seen in the steps outlining the `FillRegionTextureAndColor()` method.

1. Declare an object of each derived texture class.

   a. `TECoarse crsObj`

b. `TEContrast conObj`

c. `TEDirect dirObj`

d. `TELine lineObj`

2. Call the common interface methods `Apply()` and `Measure()` on each `TE` class object. The former method initializes the object and prepares the image for extraction; the later method computes the respective feature in each fuzzy region. This is shown for the crsObj object (coarseness) below.

   a. `crsObj.Apply()`

   b. `crsObj.Measure()`

3. Finally read the feature values in each region, update the `featureVariant` accumulator objects (i.e. each is an object of the `StdMoment class`) with the feature values, and store the feature values in the `indexValuesObj` object (an object of class `Matrix3D<double>`). This is done again by a common interface method (i.e. `Read()`) shown for only the `crsObj` below. The coarseness feature value is stored in the first component of the texture feature vector (i.e. 4th row of the index record) of each region description.

   a. `indexValues.SetValue(4,1,0) = crsObj.Read(0)`

   b. `indexValues.SetValue(4,1,1) = crsObj.Read(1)`

   c. `indexValues.SetValue(4,1,2) = crsObj.Read(2)`

   d. `indexValues.SetValue(4,1,3) = crsObj.Read(3)`

e. `indexValues.SetValue(4,1,4) = crsObj.Read(4)`

The `TE` class has 3 virtual methods `Apply()`, `Measure()` and `Read()`. The behavior of these methods is redefined in each feature class for the specific requirements of each derived feature class. A brief description of the implementation of these common interface methods in each derived class is given next.

The `TECoarse` class computes the fuzzy region-based coarseness values in each fuzzy region. The main member of the class are shown below. The `Apply()` method computes the average gray-level over neighbourhoods of size $2^k \times 2^k$, $k = 1, 2, \ldots, 5$ at each pixel. The 5 gray-level averages at each point are stored in a `Vector` of `struct WindowAverage` (a record in C/C++) using a *double* field for each window size. The `Measure()` method computes the best window size at each pixel (i.e. the size detector) in each region. The best window size values in each region are weighted by the fuzzy weight of each pixel in the region. The `Read()` method computes and returns the average weighted window size (the coarseness measure) for each region. The `TECoasre` class is given below.

```
class TECoarse
{
  Matrix3D<double> weightTables;
  Image inputImage;
  Vector(struct WindowAverages) windowAveGrayLevels;
  Vector<double> regionCoarseMeasures;
```

```
public:
  double Read(int);
  Apply(FISWeightTable&,Image&);
  Measure();
}
```

The TEContrast class computes the fuzzy region-based contrast measures. Its main members are shown below. The Apply() method computes the weighted gray-level variance in each region (the gray-levels are weighted by the fuzzy pixel weights with respect to each region). The region gray-level variances are accumulated in regionGrayLevelVariance, a Vector of StdMoment objects. The latter method also computes the weighted kurtosis value (regionKurtosis) in each region. The Measure() method combines the standard deviation in each region with the kurtosis value for the final contrast measure. The Read() method simply returns this value for a particular region.

```
class TEContrast
{
  Matrix3D<double> weightTables;
  Image inputImage;
  Vector<StdMoment> regionGrayLevelVariance;
  Vector<StdMoment> regionKurtosis;
  Vector<double> regionContrastMeasures;
public:
  double Read(int);
  Apply(FISWeightTable&,Image&);
  Measure();
```

```
}
```

The `TEDirect` class (see below) computes the fuzzy region-based directionality measures. The `Apply()` method computes the local edge vs. edge direction histograms in each region. These are stored in a $k \times 5$ `Matrix<double>` where $k$ is the number of edge directions. The region histogram reflects the total edge weight for each edge direction. The `Measure()` method computes the sharpness of the peaks in each region histogram. This is used to arrive at the final directionality measure in each region returned by the `Read()` method. For more details see Chapter Four.

```
class TEDirect
{
  Matrix3D<double> weightTables;
  Image inputImage;
  Matrix<double> edgeDirectHistogram;
  Vector<double> regionDirectMeasures;
public:
  double Read(int);
  Apply(FISWeightTable&,Image&);
  Measure();
}
```

The `TELine` class computes the fuzzy region-based line-likeness measures. The main class members are given below. The `Apply()` method computes

the local-edge direction co-occurrence matrix for each region. This is stored in `edgeDirectionHistogram`, a $4 \times 4 \times 5$ `Matrix<double>` object (4 is the number of edge directions considered). Here the region histograms represent the total weight in which two pixels occur in the region with two particular direction codes. The `Measure()` method sums the histogram entries, weighting the entries in the same direction by +1 and those in the perpendicular direction by −1. The final measure divides this result by the total weight of all entries in the region. The `Read()` method returns this value for a particular region.

```
class TELine
{
  Matrix3D<double> weightTables;
  Image inputImage;
  Matrix<double> edgeCooccurrenceHistogram;
  Vector<double> regionLineMeasures;
public:
  double Read(int);
  Apply(FISWeightTable&,Image&);
  Measure();
}
```

### 5.2.3 Image query

Once the database is created, populated and indexed, the database can then be queried. For details on how to submit a texture query using the QBT form see section 5.1. In this section, we discuss the back-end processing which takes

place between the web server and QBT CGI search engine. This begins once the submitted form is received by the web server. The CGI search engine is then called by the web server to process the form data (i.e. the texture query). The CGI search engine executes the query and responds by returning the search results directly to the QBT form for display. The QBT search engine unlike the web server is not a continuously running process. The web-server invokes a separate search engine process for each new texture query. The search engine process terminates once the back-end processing is finished.

The backend processing steps are summarized below:

1. The form data is validated for missing information. If the data is valid the search engine then attempts to connect to the database specified in the query.
2. If the connection is successful, the database catalog, index and feature variant files are located and opened for reading. Failure to locate one or more of these files will result in the query being aborted.
3. The QBT texture and color features are extracted from the image example and an index record is created. The features are extracted using the same method used to index each database image (the `FillTextureAndColor-Feature()` method in Section 5.2.2.
4. The database index is searched exhaustively and sequentially using similarity searching. The algorithm in [DS96] based on minimum region permutation

similarity scoring has been modified here to incorporate texture similarity based on the FRBT features. In the modified algorithm, serial search direct-matching (SSDM), several enhancements to query specification have been made. These include intuitive region and feature weight presets used in the evaluation of texture similarity.

5. An image similarity score is computed for each database image in the SSDM algorithm. The image similarity function is based on minimum permutation similarity scoring which compares the image example to 4 versions of each database image (rotated by 0, 90, 180, and 270 degrees). Each rotated database image version is scored using direct-matching. The minimum score of the 4 versions is taken as the final image-to-image similarity score between the image example and database image.

6. The "best-matching" images are filtered. Instead of returning only those images which have an image-to-image similarity value below a specified threshold, a variation to similarity searching called *cutoff-rank filtering* [DS96] is used. In cutoff-rank filtering, two pieces of information, the index reference and similarity score of each database image is stored using an intermediate list. The list is sorted in decreasing order of similarity. Then based on a cut-off rank number, $r_{cutoff}$ specified by the user, the images are filtered based on rank. The rank of an image is its position in the intermediate list (rank number). The lower rank number and thus the higher the similarity

score, the better the image ranks. Images with rank numbers below the cutoff rank number are regarded as the "best-matching" images and those which are above the cutoff rank number are considered the "worst-matching" images.

7. Finally, the set of image references and similarity scores with rank numbers below $r_{cutoff}$ are returned in a hypermedia document. The images are presented in a two column HTML table in the QBT form.

An example of the intermediate list generated in Step 6 is shown below.

```
\MY_DATABASE\FOLDER2\0005.jpeg  0.09

\MY_DATABASE\FOLDER2\0010.jpeg  0.10
\MY_DATABASE\FOLDER1\0004.jpeg  0.15
\MY_DATABASE\FOLDER1\0030.jpeg  0.25
...

...
```

Each line contains a database image folder path and image filename, along with its similarity score. The HTML source used to create the hypermedia document in step 7 with the filtered results shown above is given below. The first 4 lines is the HTTP header which indicates the HTTP version, content type (text/html) and content length (2000 bytes).

```
HTTP/1.0 200 OK Server:
NCSA/1.4.2 MIME-version: 1.0
Content-type: text/html
Content-length: 2000
```

```
<HTML>
<BODY>
<TABLE>
<TR>
<TD>
<IMG SRC="MY_DATABASE\FOLDER2\0005.jpeg"> Rank: 1  Score: 0.09
</TD>
<TD>
<IMG SRC="MY_DATABASE\FOLDER2\0010.jpeg"> Rank: 2  Score: 0.10
</TD>
</TR>
<TR>
<TD>
<IMG SRC="MY_DATABASE\FOLDER1\0004.jpeg"> Rank: 3  Score: 0.15
</TD>
<TD>
<IMG SRC="MY_DATABASE\FOLDER1\0030.jpeg"> Rank: 3  Score: 0.25
</TD>
</TR>
.
.
.
</BODY>

</HTML>
```

The response is transferred as a large text stream, one line at a time to the QBT form. The lines are written to standard output which is redirected to the QBT form via the web server. The HTTP header and HTML body are separated by a blank line. The <HTML></HTML> tags begin and end the HTML document. The <BODY></BODY> tags enclose its contents. The <TABLE></TABLE> tags define an HTML table. Each <TR></TR> tag pair defines a new row, and each

<TD></TD> tag pair defines a new table cell. The table cells each contain 3 pieces of data: a database image reference and its similarity score. The index image reference is also displayed with the image. This reference can then be used to resubmit the texture query (this time with a returned image as the image example) to iteratively refine the search results. The region and feature weights may also be adjusted in the resubmitted query.

### 5.2.4 The SSDM Algorithm

The SSDM algorithm performs an exhaustive and sequential search of the image database index. Each database index record is compared to the image example index record to compute a similarity score for each database image. The similarity function is modelled after the one used in [DS96] which computes a color similarity score based on minimum region permutation similarity scoring (see chapter 2). I have expanded the original similarity function to also consider texture-similarity based on the FRBT features.

Also enhanced in our version of the similarity function is the use of region and feature weights in query specification. We will see later in this section how the use of region and feature weight presets plays an important role in this respect.

The SSDM algorithm is partly implemented in the `DBExecuteQuery` method (`DB` class) which creates the image example index record and initiates the search. Two important search related classes are used in this method: the

IndexSearch class and the CutoffRankFilter class. These two classes are shown below. The IndexSearch class performs the actual similarity search on the database index and returns a list of index references and similarity scores using an object of the CutoffRankFilter class. The latter class is then responsible for filtering the images and returning the "best ranking" images in a hypermedia document. Class IndexSearch

```
{
private:
  int indexType;
  int queryType;
  char* queryImageName;
  ifstream indexFileObj
  ifstream variantFileObj
  CutoffRankFilter& searchResultsObj
  Matrix<double> indexVariantObj

public:
  IndexSearch();
  ~IndexSearch();
  ReadIndexHeader(Vector<double>&);
  SearchIndex(Index&, Matrix3D<double>&);
  GetIndexRecord(Index&, char*);

  GetMinPermScore(Index&, Index&, Matrix3D&);Class CutoffRankFilter

{
private:
  int noImages;
  int cutoffRank;
  Vector<char*> imageNames;
  Vector<double> simScores;
public:
```

```
  CutoffRankFilter(int, int);
  ~CutoffRankFilter()
  AddImage(char*, double);

  Sort();
  FilterResults & Filter();
}
```

The steps of the SSDM algorithm are outlined below. Where appropriate we indicate the methods of the above two classes relevant to each step.

1. Let $E$ denote the image example index record (`ImageIndex` class object); $D$ denote a database image index record; W denotes a $6 \times 3 \times 5$ 3D matrix containing the composed region and feature weights specified in the texture query; I a list storing the intermediate search results (`CutoffRankFilter` object); and $R_{cutoff}$ the cutoff rank number specified in the texture query.
2. Create the image example index record $E$. The texture and color features are extracted from the image example using the `FillColorAndTexture` method (`ImageIndex` class).
3. Read the next database index record into $D$. The index is processed sequentially from start to end.
4. Compute the image-to-image similarity score between $E$ and $D$ using the `GetMinPermScore` method (`IndexSearch` class). The weight matrix $W$ controls the weight of each region and its features in the similarity score.
5. Append the similarity score computed in step 4 and the index reference stored

in $D$ to the list of intermediate search results $I$.

6. Repeat steps 3–5 until all database index records have been processed.
7. Sort $I$ in ascending order of similarity scores.
8. Filter out the worst ranking images based on $R_{cutoff}$ using the `FilterImages()` method of the `CutoffRankFilter` class.
9. The best ranking image references and similarity scores remaining at the end of step 8 are formatted into a 2 column HTML table. The table of search results is then transferred using HTTP (hyper-text transfer protocol).

### 5.2.5 The region and feature matrix

Before moving on to the details of the similarity function (the `GetMinPermScore` method) some discussion on the role of the region and feature weights in query specification (collectively stored in a 3D matrix) is warranted. The 3D weight matrix described below play a critical role in assessing image similarity based on texture and color. The weight matrix consists of 5 sets of weights each stored in a $6 \times 3$ `Matrix` (which together are stored in a $6 \times 3 \times 5$ Matrx3D object), and each storing the region and feature weights of a region. The first three rows of each matrix contain the composed color feature weights (average color, variance colore and covariance color). They are called composed weights because each value in each region matrix is the product of the feature weight and the region weight. The first 3 rows contain the color weights (average color, vari-

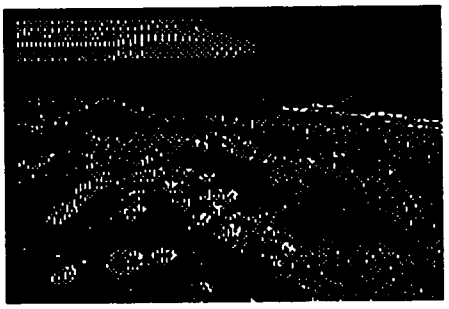
ance color and covariance color). The last two rows contain the texture feature weights (fuzzy region-based coarseness, contrast, directionality and line-likeness). The region weights are integers ranging between 0–10 and the feature weights are reals between 0–10. Setting the region weight to 0 (therefore the product of each feature weight and region weight is 0) will exclude the entire region and its features in the direct-match similarity computation (i.e. the region-region score is 0). For more finer control of features participating in the region-to-region score, the feature weights can be individually set in each region. Setting a feature weight to 0 will excludes just the feature in the region-to-region score.

To assist in specifying region and feature weights, several region and feature weight presets are provided in the QBT Form. Six region weight presets are available: portrait, landscape, macro, top and bottom. Table 1 gives the weights used in each preset along with the regions emphasized.

| | | |
|---|---|---|
| Portrait | {2, 1, 1, 1, 1} | Emphasis on centre |
| Landscape | {1, 1, 1, ,1 ,1} | Equal emphasis |
| Macro | {1, 1, 1, 1, 1} | Close-ups |
| Top | {0, 0, 0, 1, 1} | Emphasis on top corners |
| Bottom | {0, 1, 1, 0, 0} | Emphasis on bottom corner |
| Left | {0, 0, 1, 1, 0} | Emphasis on left corner |
| Right | {0, 1, 0, 0, 1} | Emphasis on right corner |

Table 1 Region weight presets

The feature weight presets are based on simple texture descriptions which characterize the texture in a region as either predominantly *coarse*, *fine* or *flat*. In each feature weight preset, a *positive* and *negative* texture feature is emphasized for the region. The idea here is that each is an effective discriminator in the region but on opposites sides of the spectrum. The coarse preset for example emphasizes coarseness (positive feature) and line-likeness (negative feature). The

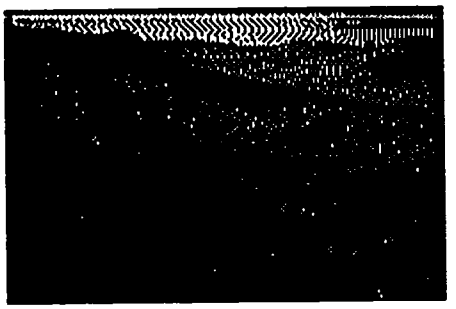


| | Feature weight preset | Region weight preset |
|---|---|---|
| R0 (centre) | coarse | 1 |
| R1 (bottom-right) | fine | 1 |
| R2 (bottom-left) | fine | 1 |
| R3 (top-left) | flat | 1 |
| R4 (bottom-right) | flat | 1 |

Figure 21 Coarse region example

coarse preset would apply to the centre and bottom-left regions of the image given in figure 21. The coarseness is high and line-likeness is low in these regions. The negative feature in the coarse preset will help prevent matching with smooth

regions (e.g. flat) in which coarseness is artificially high and line-likeness is 0. The difference in line-likeness (i.e. 0 versus low) will reduce the match quality when a coarse and smooth region are matched.



| | Feature weight preset | Region weight preset |
|---|---|---|
| R0 (centre) | fine | 1 |
| R1 (bottom-right) | fine | 1 |
| R2 (bottom-left) | fine | 1 |
| R3 (top-left) | flat | 1 |
| R4 (bottom-right) | flat | 1 |

Figure 22 Fine region example

The flat feature preset is used in smooth regions with low texture. Here the contrast (positive feature) and line-likeness feature (negative feature) are emphasized in the region. In figure 22, the flat preset is used to describe the top-left and bottom-right regions of the image.

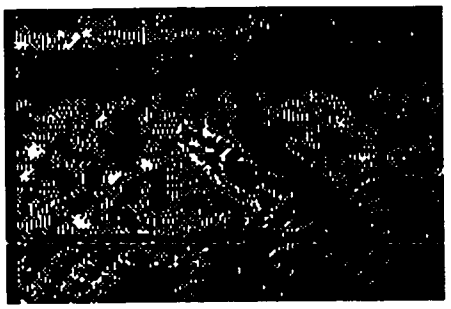
The idea behind the positive and negative feature weighting scheme is that when matching two regions with completely different texture profiles, we would

like the separation between the positive and negative features to be as large as possible, thereby increasing the texture dissimilarity between the regions. For example, in matching a coarse region with a fine region, the positive and negative features are optimally far apart (i.e. high coarse vs. low coarse and low line-likeness vs. high line-likeness. Table 2 show the interplay between the positive and negative features when matching the possible region combinations.

| | Coarse | Fine | Flat |
|---|---|---|---|
| Coarse | high vs. high coarse (+.ve)<br>low vs low line-likeness (-.ve) | high vs. low coarse (+.ve)<br>low vs high line-likeness (-.ve) | high vs. coarse (+.ve)<br>low vs nil line-likeness (-.ve) |
| Fine | high vs. low line-likeness (+.ve)<br>low vs high coarse (-.ve) | high vs. high line-likeness (+.ve)<br>low vs low coarse (-.ve) | high vs. nil line-likeness (+.ve)<br>low vs high coarse (-.ve) |
| Flat | low vs high contrast (+.ve)<br>nil vs low line-likeness (-.ve) | low vs high contrast (+.ve)<br>nil vs high line-likeness (-.ve) | low vs low contrast (+.ve)<br>nil vs nil line-likeness (-.ve) |

Table 2 Comparison of negative and positive feature presets

Finally the color preset is used in regions with multiple textures which cannot be described using a single texture preset. The only way to match these types of regions is to use the color preset. This is the only time when color features participate in the scoring.



| | Feature weight preset | Region weight preset |
|---|---|---|
| R0 (centre) | fine | 1 |
| R1 (bottom-right) | fine | 1 |
| R2 (bottom-left) | fine | 1 |
| R3 (top-left) | flat | 1 |
| R4 (bottom-right) | flat | 1 |

Figure 23 Emphasis on coarseness in bottom region

The region and feature weights are used as query specification tools i.e. image example construction. With practice, the weights can be used to narrow the search for similar images by emphasizing regions and features which are the most important in determining image similarity. For the image example in figure 23, the user can refine the search for snakes in the bottom of the image by focusing on the bottom regions using the bottom region weight preset and coarse feature weight preset. This would help prevent matching with images which have similar features only in the top regions. In another example, the user constructs an image example from another image (see figure 24). In this example, the user contructs

an image of a beach scene using an image of a lake with grass below. To specify that sand is required in the lower regions instead of grass, the flat preset is used in these regions instead of the fine preset.

| | Feature weight preset | Region weight preset |
|---|---|---|
| R0 (centre) | fine | 1 |
| R1 (bottom-right) | fine | 1 |
| R2 (bottom-left) | fine | 1 |
| R3 (top-left) | flat | 1 |
| R4 (bottom-right) | flat | 1 |

Figure 24 Example of query construction

### 5.2.6 Details of the image-to-image similarity function

The `GetMinPermScore()` method computes an image-to-image score based on texture and color distance using the minimum permutation image similarity scoring technique (see Section). For each database image, the technique computes an image-to-image similarity score for each rotation of the database image in $T_{90}$ (0, 90, 180 and 270 degrees). Thus three other orientations in ad-

dition to the original database image orientation (0 degrees) are compared with the image example. Although an index record can be created for each new database image orientation (90, 180 and 270 degrees) not in the database index; what is done instead is to shift the database image index record region descriptions (the index record stored for the image corresponds to the original orientation) in the direct-match correlation to reflect each rotation. Each shift in the region descriptions of the database image yields a new *region description permutation.* A region description permutation $P_n(D)$ is defined as an ordered list of region descriptions. Recall that an index record is comprised of five region descriptions $\{r_0, r_1, r_2, r_3, r_4\}$ which store the index features corresponding to the regions $R_0$ (centre), $R_1$ (bottom-left), $R_2$ (bottom right), $R_3$ (top-left) and $R_4$ (top-right).

The default region description permuation $P_0(D)$ is defined as $\{r_0, r_1, r_2, r_3, r_4\}$ (i.e. the original ordering of the region descriptions). The other permutations are found by shifting the outside region descriptions (i.e. $r_i, i = 1, 2, 3, 4$ one position to the left (with rotation through the last region description) for each 90 degree rotation of the database image. The region description permutation $P_1(D)$ corresponding to the database image rotated 90 degrees, for example, is $\{r_0, r_2, r_4, r_3, r_1\}$. The subscript in $P_n(D), n = 0, 1, 2, 3$ indicates the number of 90 degree rotations and thus the number of left shifts through the last region description. Thus four intermediate image-to-image similarity scores corresponding to the region description permutations $P_n(D), i = 0, 1, 2, 3,$ are

computed. The final image-to-image similarity score is the minimum of these scores. The minimum permutation score corresponds to the orientation which best matches the image example. This will account for changes in the image example from portrait to landscape formats and vice-versa (i.e. database images identical or very similar to the image example but differing in format).

The image-to-image score is the sum of the region-to-region scores computed in the direct-match correlation between the region descriptions of the image example and a given database image region description permutation. The default region description permeation is always used for the image example. Recall that in a direct-match correlation the regions are matched in place so that the *i*th region description of the image example index record is scored against the *i*th region description of the database image permuted index record. The region-to-region score is the sum of the texture and color distance between two region descriptions. Recall that each region description contains 3 color feature vectors (average color, variance color and covariance color) and 1 texture feature vector (the FRBT texture features).

Texture distance is computed as simple Mahalanobis distance [KZL94] between the texture features of the region descriptions being scored. This is computed as ,

$$d_{texture}(e,d) = \sum_{i=1}^{4} \frac{\left(t_i^e, t_i^d\right)^2 \cdot w_i}{v_i^e} \tag{70}$$

where $e$ and $d$ are regions descriptions of the image example and database image respectively; $t_i^e$ and $t_i^d$ are the $i$th texture feature components of the region descriptions $e$ and $d$, respectively. $w_i^e$ is the composed feature weight of the $i$th texture feature and $v_i^e$ is the variance of the $i$th texture feature in $e$. Color distance is computed as a function of the average and covariance color distances. These are computed using the following weighted distance functions in [DS96]. Average color distance is computed as

$$d_{ave}(e,d) = \sqrt{\sum_{i=1}^{3} w_i \left(A_i^e - A_i^d\right)^2} \tag{71}$$

where $A_i^e$ and $A_i^f$ are the $i$th components of the average color feature vectors of the region descriptions $e$ and $d$. $w_i$ is the composed weight of the $i$th average color component. Covariance color distance is computed as

$$d_{cov}(e,d) = \sqrt{\sum_{i=1}^{3} w_i^e \left(C_i^e - C_i^d\right)^2} \tag{72}$$

where $C_i^e$ and $C_i^d$ are the $i$th components of the covariance color feature vectors of the region descriptions $e$ and $d$. $w_i$ is the composed weight of the $i$th covariance color component. Based on these distances the total color distance between $e$ and $d$ is computed as

$$d_{color}(e,d) = d_{ave}(e,d) + d_{cov}(e,d) + d_{ave}(e,d).d_{cov}(e,d) \tag{73}$$

Lastly, the steps of the `GetMinPermScore()` method are outlined below.

1. Input consists of the image example and database image index records $E$ and $D$, respectively, and the weight matrix $W$ containing the composed region and feature weights.

2. An image-to-image score $S_{image}(E, P_n(D))$ is computed between the image example region descriptions (e.g. $P_0(E)$) and each database image region description permutation $P_n(D)$ where $n = 0, 1, \ldots 3$. $n$ is the number 90 degree rotations and$P_n(D)$ is the set of database region descriptions defined by

$$\{d_0, d_{(i+n) \bmod 4} | d_i \in P_0(D) - \{d_0\}\} \tag{74}$$

For corresponding region descriptions $e$ and $d$ in $P_0(E)$ and $P_n(D)$, the region-to-region score is computed as,

$$R(e, d) = d_{texture}(e, d) + d_{color}(e, d) \tag{75}$$

3. The image-to-image similarity scores between $P_0(E)$ and $P_n(D), n = 0, 1, 2, 3$ are computed as the sum of their region-to-region scores. This is computed as,

$$S_{image}(P_0(E), P_n(D)) = 1 - \left[ R(e_0, d_0) + \sum_{k=1}^{4} R(e_k, (P_n(D))_k) \right] \tag{76}$$

where $(P_n(D))_k$ is $k$th region description in $P_n(D)$

4. Repeat Step 2 until all database image region description permutations have been processed.

5. The final image-to-image similarity score is the minimum of the image-to-image similarity scores computed in steps 2–3. This is given by

$$S_{min}(E, D) = \min \{S_{image}(P_n(E), P_n(D)), n = 0, 1, 2, 3, 4\} \quad (77)$$

### 5.2.7 Details of the FilterImages method

Below we outline the cutoff rank filtering [KZL94] steps implemented in the `FilterImages` method (`class CutoffRankFilter`).

1. Input to the method consists of the cutoff percentage $p_{cutoff}$ defined as the percentage of total database images compared to be returned from the search, and the list of sorted index references and similarity scores returned from step 7 of the SSDM algorithm.
2. Then the cutoff rank number $r_{cutoff}$ is computed as

$$r_{cutoff} = n \times p_{cutoff} \quad (78)$$

and is used to filter the best-ranking images.

3. The rank of each image is determined by its position in the sorted list of database references and similarity scores. The images in this list decrease in match quality (similarity score) from top to bottom. The cutoff rank number determines the list position after which the image match quality is below what meets the texture query. Finally the best-matches are filtered using, $\{R_i | R_i \in R, i \leq r_{cutoff}\}, \{S_i | S_i \in S, i \leq r_{cutoff}\}$ where $R_i$ and $S_i$

are the $i$th database image reference and similarity score, respectively; $R$ and $S$ are respectively two associative lists containing the index references and similarity scores.

# Chapter 6 TEST RESULTS

In this chapter, the results of 3 tests are presented and are used to do a performance comparison between the QBT system and the system in [KZL94] (the reference method). The latter system is the current state-of-the-art in texture-similarity image retrieval based on direct-matching. In the first test, the overall retrieval effectiveness is compared using *precision*, a standard measure of retrieval effectiveness in information retrieval.

The second test examines the dependence of each system on the object position changes in $T_{small}$. A new performance indicator, $R_{small}$ ($T_{small}$ recall number), is proposed to evaluate the retrieval effectiveness of each system with respect to these transformations. Finally, the last test examines the dependence of each system on the object position changes in $T_{90}$. Again, a new performance indicator $R_{90}$ ($T_{90}$ recall number) is used to assess the retrieval effectiveness of each system with respect to these transformations.

## 6.1 Overall performance

The test image database consists of 3350 192x128 8–bit color JPEG images. Prior to testing, 50 test images representing a cross section of the images in the test image database are selected. Table 3 lists the number of test images selected

from each category. The 50 test queries are executed using each method and the results in each run are used in the analysis of each test.

| Category | # of images |
|---|---|
| Scenic Lakes | 5 |
| Fields | 5 |
| Dams | 5 |
| Animals | 8 |
| Agriculture | 7 |
| People | 5 |
| Structures & Machinery | 6 |
| Forest | 4 |
| Terrain | 5 |

Table 3 Number of test images from each image category

Here we evaluate the overall retrieval effectiveness of each system using precision. Precision is defined as [KB96],

$$P = \frac{\#\text{of relevent images retrieved}}{\#\text{ of images returned}} \tag{79}$$

where the relevant (or similar) images of each test image are determined manually.

In direct-match texture retrieval, images are deemed similar if they contain similarly textured objects in nearly the same locations. The objects, however, may differ in color, shape, and orientation. An object is a region with homogeneous texture. In QBT, we have also tried to make direct-match retrieval insensitive as much as possible to changes in object position. The size of each test result in

both methods is 52 images (calculated using the test image database size and a cutoff percentage of 1.5%). The QBT precision values of the 50 test queries are displayed in figure 25.

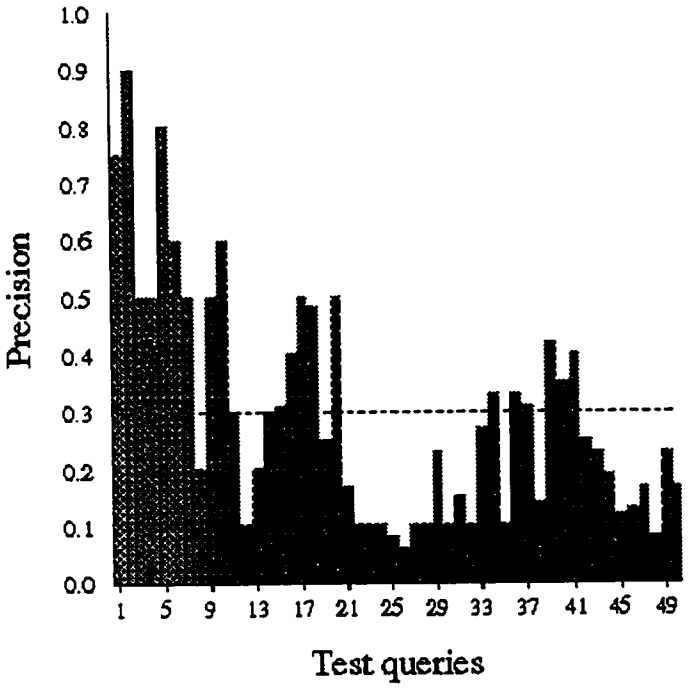



Figure 25 QBT precision values

The dotted line corresponds to average precision. Figure 26 compares the average precision results in each image category.

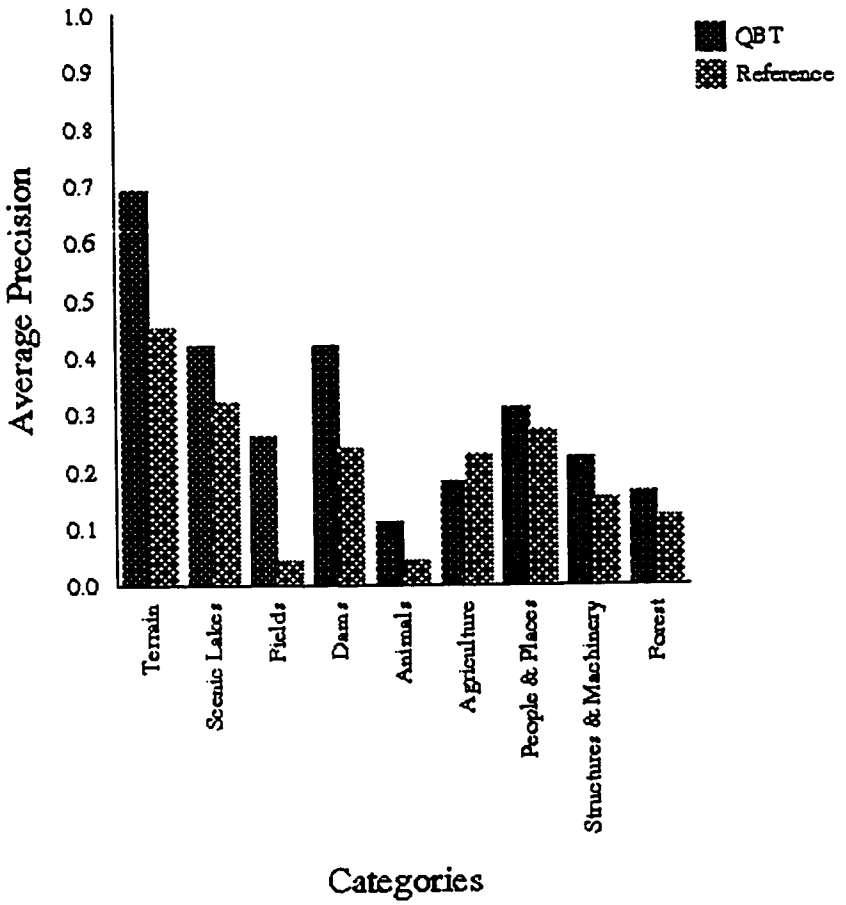



Figure 26 Average precision results per image category

In all but one category (agriculture) our method performed significantly better than the reference method.

Figure 27 shows the difference between the precision values of both methods. The dotted line corresponds to average precision difference. The precision values

indicate a 66% increase in average precision for the QBT system.

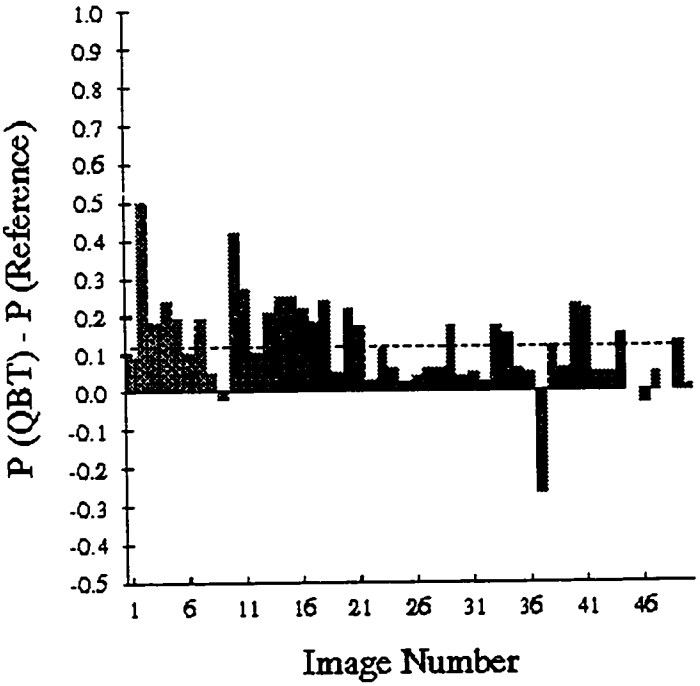



Figure 27 Difference in precision values between both methods

Clearly the above precision results indicate that the overall performance of QBT is superior to that of the reference method. For only 3 out of 50 test images, the reference method performed significantly better than our system. This superior performance is due to a number of factors which are explained next. First, the lack of feature weights in the reference method controlling the influence of each feature in the similarity function precludes the ability to fine tune their discrimination ability for each test image. The Tamura texture features do not always yield accurate feature measurements for all textures. For textures which the Tamura texture feature values do not accurately reflect the given texture, the

features should not be used in the similarity function. The feature presets used in QBT querying based on a positive and negative feature weighting scheme adequately addresses this need. In addition, the region weights available only in QBT querying helps focus retrieval on the most important textured regions or objects.

The last two factors are related to the fuzzy regions and the design of the similarity function which together address the problem of position dependence in direct-matching. Indeed, many of the similar images returned in the QBT test queries are transformed versions of the test image (e.g. $T_{small}$ and $T_{90}$). More on this in the next two sections.

## 6.2 Testing $T_{small}$ performance

One way to diminish the dependence on object position in direct-matching is to extract features using fuzzy regions instead of standard regions which have sharp boundaries. Fuzzy regions are used to focus the feature vectors on the centre of a region so that small changes in object positions near the boundary are marginalized. Small translations in $[-20, 20]$ in both the x and y axes (measured in pixels), and small rotations in $[-20, 20]$ degrees around the image centre can be accounted for using the fuzzy pixel weighting scheme in fuzzy image subdivision.

We test the effectiveness of both systems with respect to these specific changes in object position using two tests. In the first, we examine the dependence on

$T_{small}$ of the similarity function (the image-to-image similarity function) in each system. This is accomplished using a separate image database which contains only translated and rotated versions of a single test image. 10 rotations ranging from –20 to 20 degrees in 2 degree increments and 10 translations in the x and y axis in 2 pixel increments of the original test image (query 23) are added and indexed into this database. We then match the original image with the transformed versions using 2 texture queries, one using the QBT system and the other using the reference method. The similarity values of both queries illustrating the dependence of each similarity function on the transformations in $T_{small}$ are displayed in figures 28, 29 and 30.

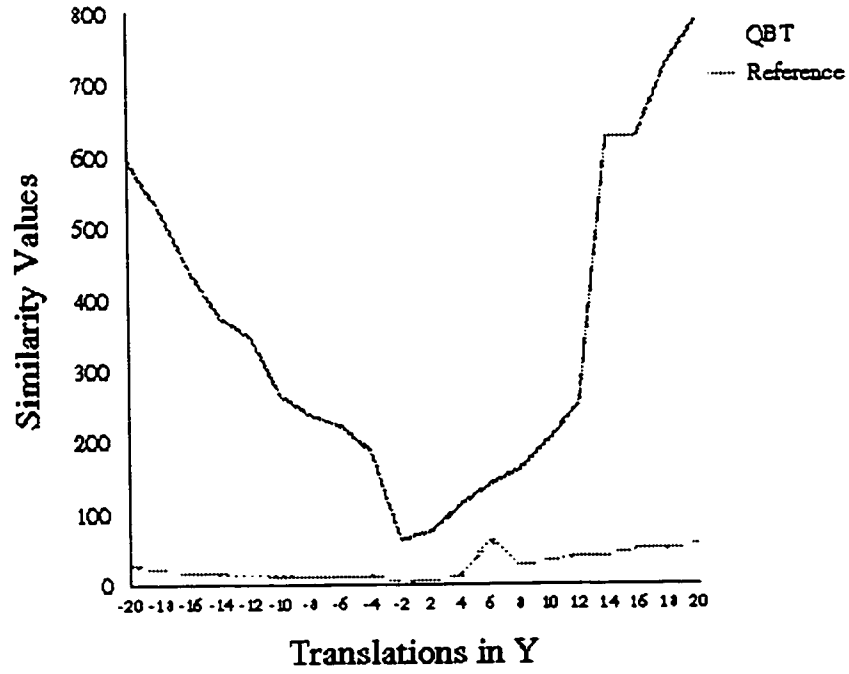



Figure 28 Dependence on small translations in y

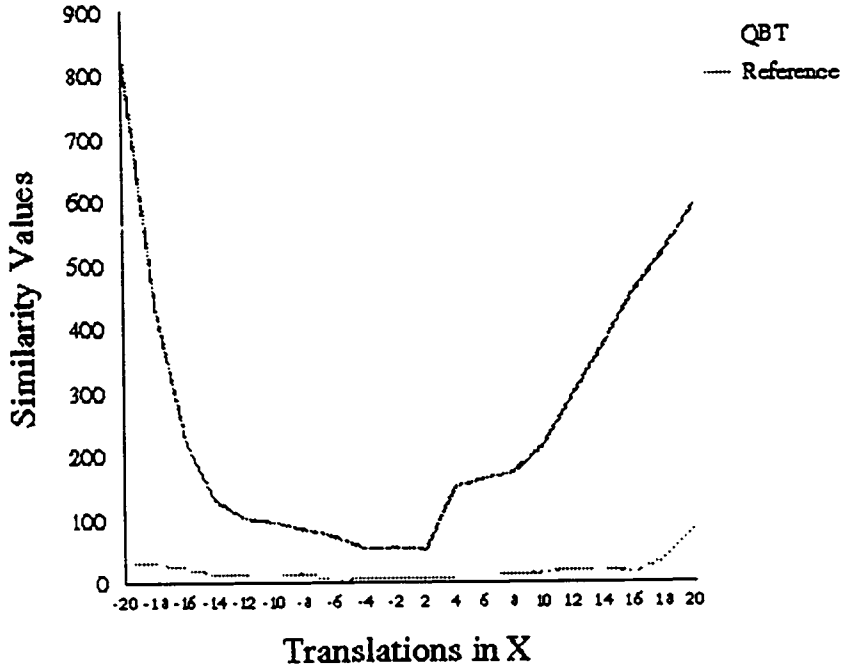



Figure 29 Dependence on small translations in x

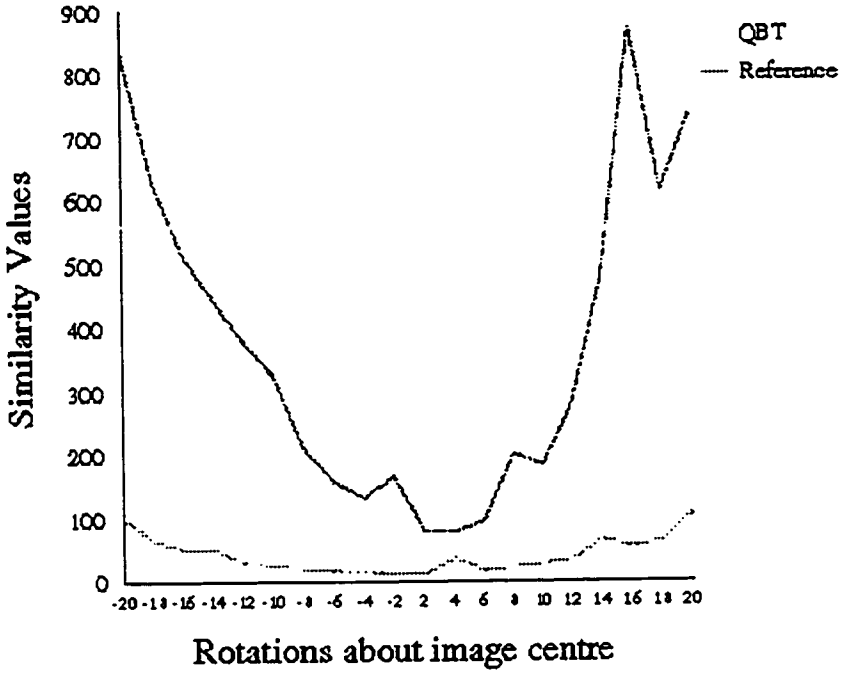



Figure 30 Dependence of similarity function on small rotations

Recall one of our objectives was to design a similarity function which changes smoothly over the restricted transformations in $T_{small}$. Clearly, our similarity function is more stable with respect to these changes than that of the reference method. To examine the effects of $T_{small}$ on retrieval, we define a special performance indicator for this in the second test. The $T_{small}$ recall number ($R_{small}$) is defined as the minimum of the ratio of the total number of $T_{small}$ transformed test image versions retrieved to the total number of transformed test versions in the database and 1. This is computed as,

$$R_{small} = \min\left(\frac{\text{Total \# of } T_{small}\text{test versions retrieved}}{\text{Total \# of } T_{small}\text{test versions}}, 1\right) \qquad (80)$$

For each of the 50 test images we create 3 translated versions and 3 rotated version. The transformed versions are indexed and added to the test image database using both indexing methods. We cap the $T_{small}$ recall value to 1 since the transformed versions of other test images may be returned along with those of the original test image.

Figure 31 shows the difference in the $T_{small}$ recall values of both methods. The QBT system in most of the test queries performed significantly better than the reference method, and in all cases was at least as good as the reference method. Overall, the QBT values indicate a 16% improvement in retrieval performance

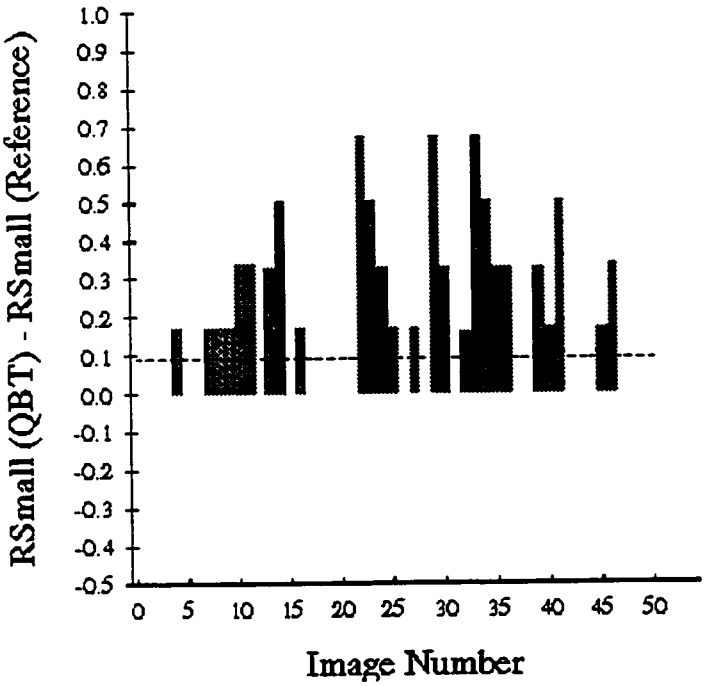



Figure 31 Difference in $R_{small}$ values between the two methods

with respect to $T_{small}$. Finally in figure 32, we compare the $R_{small}$ values in

each image category. In only two of the categories (Terrain and Dams) did the

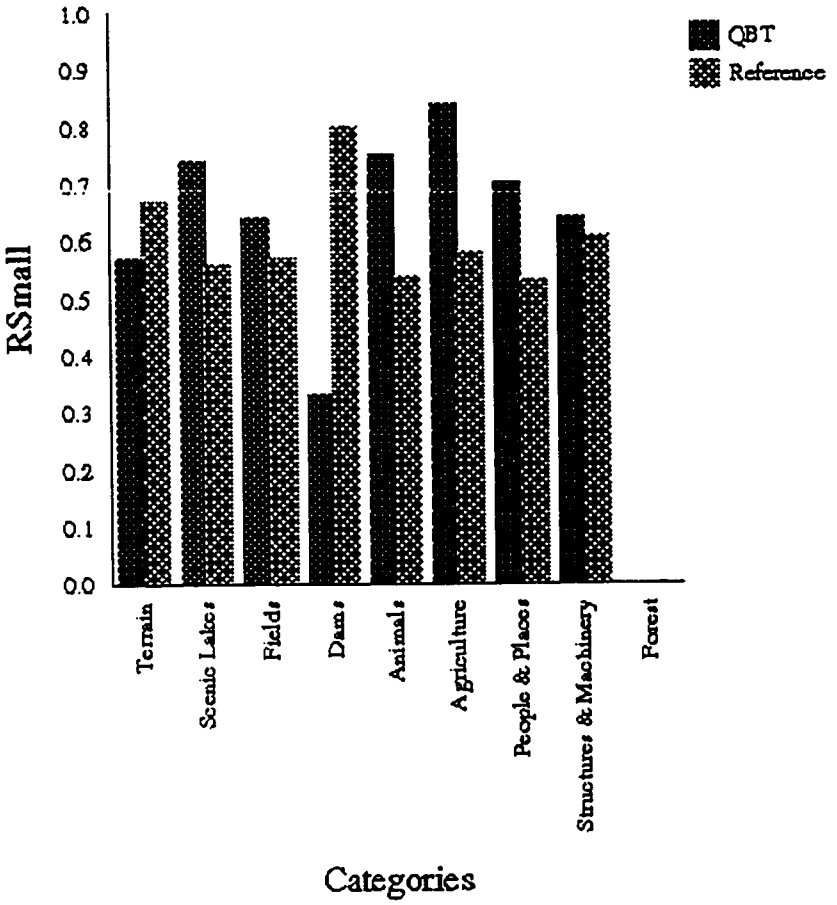



Figure 32 Comparison of $R_{small}$ in each category

reference method perform significantly better then our method

## 6.3 Testing $T_{90}$ Performance

The other way to diminish the dependence on object position in direct-matching is to use a similarity function based on minimum permutation similarity scoring [SO95]. This technique minimizes the effects of large rotations in

multiples of 90 degrees around the image centre (i.e. $T_{90}$) and is designed for centrally arranged image regions.

To test the effects of $T_{90}$ object position changes on retrieval, we rotate each test image for each rotation in $T_{90}$. The rotated test images are indexed and added into the test image database using both indexing methods. We define a performance indicator, the $T_{90}$ recall number, to assess the impact on retrieval performance due to position changes in $T_{90}$. This is given by,

$$R_{90} = \min\left(\frac{\text{Total \# of } T_{90}\text{test versions retrieved}}{\text{Total \# of } T_{90}\text{test versions}}, 1\right) \tag{81}$$

The $T_{90}$ recall number is defined as the ratio of the total number of $T_{90}$ transformed test image versions retrieved to the total number of $T_{90}$ transformed test image versions in the database. We cap this measure at 1 since the transformed test versions of other test images may also be retrieved with transformed versions of the original test image. The difference in the $T_{90}$ recall number values of the two methods is displayed in figure 33. In all but 1 query did the reference

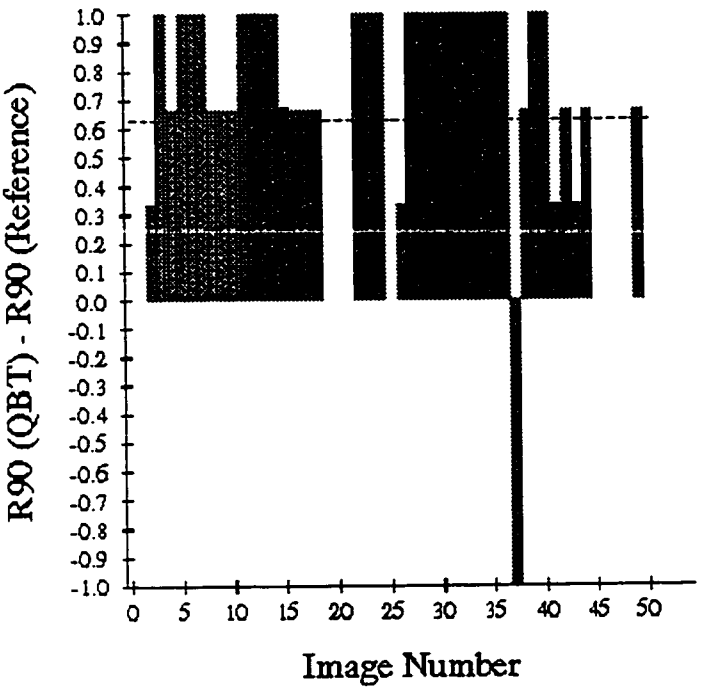



Figure 33 Difference in $R_{90}$ values between the two methods

method perform significantly better then our method. The QBT $T_{90}$ number values indicate a 100% increase in performance with respect to changes in $T_{90}$. Finally, in figure 34 we compare the $R_{90}$ values for each category.

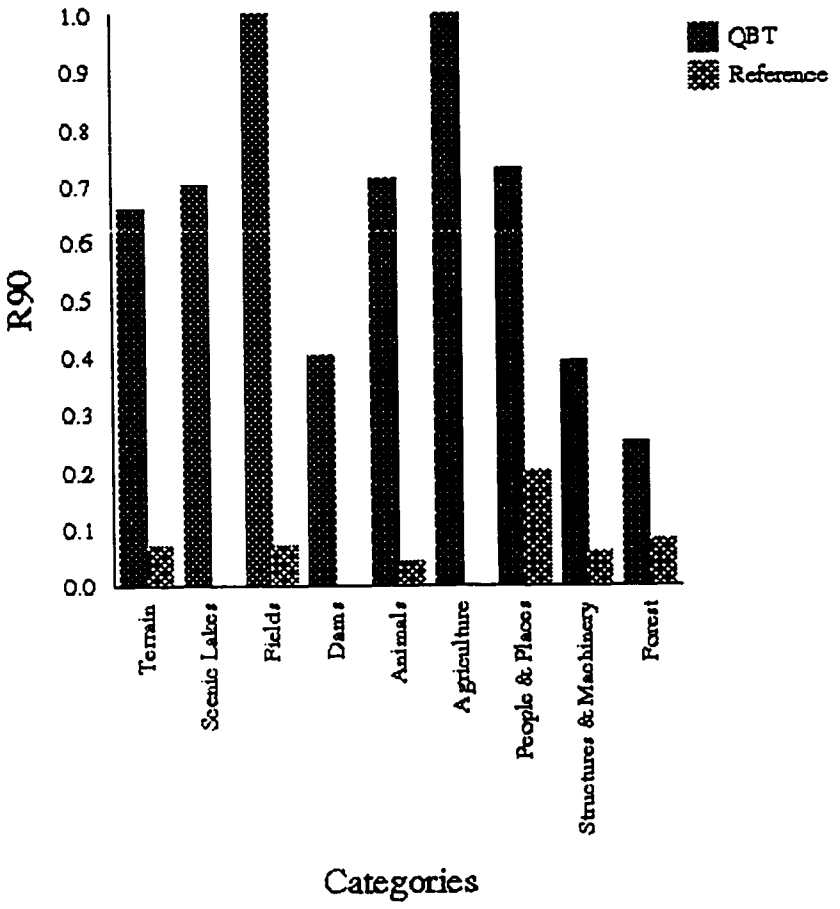



Figure 34 Comparison of $R_{90}$ for each category

Here the QBT system performed significantly better than the reference method in all categories.

# Chapter 7 CONCLUSION

This thesis has shown that fuzzy image subdivision (FIS) [SO95] is superior to rectangular-partitioning [KZL94] in direct-match image retrieval based on the Tamura texture features. The contributions made in this thesis can be summarized as the following: to show how FIS is incorporated into a texture-similarity approach based on the Tamura texture features; in particular, the modifications to the original feature algorithms required for fuzzy-region extraction, and the design of a suitable image-by-example query specification model and texture-similarity function.

To demonstrate that FIS is superior to rectangular-partitioning, we addressed three specific problems associated with texture-similarity image retrieval based on direct-matching:

- ☐ feature invariance with respect to object position changes in $T_{Small}$ and $T_{90}$
- ☐ overall retrieval effectiveness using precision
- ☐ improved query specification model

For the first goal, we have simplified the problem of feature invariance w.r.t. to position changes by working only with the restricted set of transformations in $T_{small}$ and $T_{90}$. For the real images used in our test image database, these are the most typical transformations which occur (changes between portrait and landscape

formats). Two tests, $R_{small}$ ($T_{small}$ recall number) and $R_{90}$ ($T_{90}$ recall number) were devised to evaluate the effectiveness of our system and the reference method in dealing with these object-position changes in direct-match retrieval. In short our system yielded a 16% increase in recall of transformed versions of the query image in $T_{small}$ and a 100% increase for transformed versions of the query image in $T_{90}$. Recall that these experiments were performed by selecting 50 test images and creating several transformed versions of each test image using the transformations in $T_{small}$ and $T_{90}$. We also plotted the similarity values (a measure of texture image similarity) of the transformed test image versions in both tests for each method. It is clear from these graphs that are similarity function is significantly more stable w.r.t. the changes in $T_{small}$ and $T_{90}$. Secondly, we have shown that our system also improves the overall precision results over the reference method. In 50 test images, our system yielded a 66% increase in precision.

Thirdly, we have developed a new image-by-example, query specification model based on direct-matching and the feature weighting scheme (+.ve/-.ve weighting) proposed in this thesis. The test results demonstrate that this weighting scheme improves the discrimination power of the Tamura texture features in fuzzy region-based retrieval. We believe that this weighting scheme is partly responsible for the improvements in overall precision reported above.

# BIBLIOGRAPHY


[ABF95] J. Ashley, R. Barber, and M. Flickner. Automatic and semi-atuomatic methods for image annotation and retrieval in QBIC. *Storage and Retrieval for Images and Video Databases*. 2420:24–35, 1995.

[Baj73] R. Bajcsy. Computer description of textured surfaces. 3rd Int. Joint Conf. Artificial Intelligence, pages 572–579, 1973.

[BGS92] Elisabetta Binaghi, Isabella Gagliardi, and Raimondo Schettini. Indexing and fuzzy logic-based retrieval of color images. In E. Knuth and L.M. Wegner, editors, *Visual Database Systems III*, pages 79–91. Elsevier Science Publishers B.V. North Holland, 1992.

[Bro66] P. Brodatz. *Textures*. New York: Dover, 1966.

[CBGM97] C. Carson, S. Belongie, H. Greenspan, and J. Malik. Region-based image querying. *CVPR97 Workshop on Content-Based Access of Image and Video Libraries*, 1997.

[Dow93] James Dowe. Content-based retrieval in multimedia imaging. *Storage and Retrieval for Images and Video Databases*, 1908:164–67, 1993. Preface to chapter on Content-based retrieval.

[DRW97] L. S. Davis, A. Rosenfeld, and J. S. Weszka. Region extraction by averaging and thresholding. *IEEE Transactions on Systems, Man, and Cybernetics*, pages 383–388, May 1997.

[DS96] Alexander Dimai and Markus Stricker. Spectral covariance and fuzzy regions for image indexing. Technical report, Swiss Federal Institute of Technology, ETH, April, 1996. BIWI-TR-173.

[Eck93] Bruce Eckel. *C++ Inside and out*. Osborne McGraw Hill, 1993.

[FF91] B. V. Funt and G. D. Finlayson. Color constant color indexing. Technical Report 91–09, Sch, School of computer science, Simon Fraser University, Vancouver B.C., Canada, 1991.

[Fin92] G. D. Finalyson. Color Object Recognition. Master's thesis, School of Computer Science, Simon Fraser University, Vancouver, B.C. Canada, April 1992.

[FMW97] D. Forsyth, J. Malik, and R. Wilensky. Searching for digital pictures. *Scientific American*, June 1997.

[FSN95] M. Flickner, H. Sawhney, and W. Niblack. Query by image and video content: The QBIC system. *IEEE Computer*, 28(9), 1995.

[FZ88] Micheal J Fold and B. Zoellick. *File Structures*. Addison Weesley, 1988.

[Gud95] Venkat N. Gudivada. Content-based image retrieval systems. *IEEE Computer*, 28(9):18–22, 1995.

[Gun95] Shishir Gundavaram. *CGI Programming on the world-wide web*. O'Reilly & Associates: Dover, 1995.

[GZCS94] Yihong Gong, Hongjiang Zhang, H.C. Chuan, and M. Sakauchi. An image database system with content capturing and fast image indexing abilities. In *IEEE International Conference on Multimedia Computing and Systems*, pages 121–130. Boston, Mass, May 1994.

[HH94] B. Holt and L. Hartwick. Retrieving art images by image content: The UC Davis QBIC project. In *Aslib proceedings, Oct. 1994*, volume 46, pages 243–248, 1994.

[HSD73] R. M. Haralick, K. Shanmugam, and I. Dinstein. Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3:610–621, 1973.

[HSK95] Wynne Hsu, Chua T. S., and Pung H. K. An integrated color-spatial approach to content-based image retrieval. *ACM Multimedia*, pages 305–313, November 1995.

[Jul62] B. Julesz. Visual pattern discrimination. *IRE Trans., Info. Theory*, IT 8:84–92, Feb. 1962.

[Jul65] B. Julesz. Texture and visual perception. *Scientific American*, 212:38–54, Feb. 1965.

[Jul75] B. Julesz. Experiments in the visual perception of texture. *Scientific American*, 232:34–43, Apr. 1975.

[KB96] S. Khoshafian and A. B. Baker. *Multimedia and Imaging Databases*. Morgan Kaufmann Publishers, Inc, 1996.

[KCHSR74] JR. K. C. Hayes, A. N. Shah, and A. Rosenfeld. Texture coarseness: further experiments. *IEEE Transactions on Systems, Man, and Cybernetics*, pages 467–472, September 1974.

[KZL94] A. Kankanhalli, H. J. Zhang, and C. Y. Low. Using texture for image retrieval. *The 3rd International Conference on Automation, Robotics and Computer Vision (ICARCV '94)*, (6), November 1994.

[LBN94] D. Lee, R. Barber, and W. Niblack. Query by image content using multiple objects and multiple features: User interface issues. In *Proc. ICIP'94 IEEE international conference on image processing*, volume II, pages 76–80, 1994.

[MMB77] O. R. Mitchell, C. R. Myers, and W. Boyne. A max-min measure for image texture analysis. *IEEE Computer*, pages 408–414, April 1977.

[MMM73] S. Mori, Y. Monden, and T. Mori. Edge representation in gradient space. *Computer Graphics and Image Processing*, 2:321–325, 1973.

[Mul93] J. N. Muller. *Computerized Document Imaging Systems: Technology and Applications*. Artech House, Boston, 1993.

[NBE93a] W. Niblack, R. Barber, and W. Equitz. The QBIC Project: Querying images by content using color, texture, and shape. *Storage and Retrieval for Images and Video Databases*, 1908:173–81, 1993.

[NBE93b] W. Niblack, R. Barber, and W. Equitz. The QBIC project: Querying images by content using color, texture and shape. Technical report, Almaden Research Center, San Jose, CA., 1993. IBM Research report RJ 9203 (81511),.

[OS95] Virginia E. Ogle and Micheal Stonebraker. Chabot: Retrieval from a relational database of images. *IEEE Computer*, 28(9):40–48, 1995.

[PF94] E. G. M. Petrakis and C. Faloutsos. Similarity searching in large image databases. Technical report, Department of Computer Science, University of Maryland, 1994.

[PNF96] D. Petkovic, W. Niblack, and M. Flickner. Recent applications of IBM's query by image content (QBIC). Technical report, Almaden Research Center, San Jose, CA., 1996. IBM Research report RJ 10006 (89095).

[PPS94] A. Pentland, R. W. Picard, and S. Sclaroff. Photobook: Tools for content-based manipulation of image databases. *Storage and Retrieval for Images and Video Databases*, 2185:34–47, 1994.

[Rob95] M. F. Robek. *Information and Records Management: Document-based Information systems*. Glencoe, New York, 1995.

[Rus95] J. C. Russ. *The image processing handbook*. CRC Press, 2nd edition, 1995.

[RV] A. Rosenfeld and G. J. Vanderbrug. Coarse-fine template matching. *IEEE Transactions on Systems, Man, and Cybernetics*, February 1977:104–107.

[SB91a] G. Micheal Schneider and Steven C. Bruell. *Concepts in data structures and software development.* West, 1991.

[SB91b] M. J. Swain and D. H. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

[SD96] Markus A. Stricker and Alexander Dimai. Color indexing with weak spatial contraints. SPIE conference. Feb. 96. San Jose. California, 1996.

[SH93] Ramin Samadani and Cecilia Han. Computer-assisted extraction of boundaries from images. *Storage and Retrieval for Images and Video Databases*, 1908:219–25, 1993.

[SH94] Harpreet S. Sawhney and James L. Hafner. Efficient color histogram indexing. In *Proc. ICIP'94 IEEE internation conference on image processing*, volume II, pages 66–70, 1994.

[Smi95] John R. Smith. *Integrated Spatial and Feature Image Systems: Retrieval, Analysis and Compression.* PhD thesis, Columbia University, 1995.

[SO95] Markus Stricker and Markus Orengo. Similarity of color images. *Storage and Retrieval for Images and Video Databases*, 2420:381–92, 1995.

[Str92] Markus A. Stricker. Color and geometry as cues for index. Technical report, Department of Computer Science, University of Chicago, 1992. CS 92–22.

[Str94a] Markus Stricker. The capacity and the sensitivity of color histogram indexing. Technical Report 94–05,, Department of Computer Science, University of Chicago, 1994.

[Str94b] Markus A. Stricker. Bounds for the discrimination power of color indexing techniques. *Storage and Retrieval for Images and Video Databases 3*, 2420:15–24, 1994.

[Swa93] Micheal J. Swain. Interactive indexing into image databases. *Storage and Retrieval for Images and Video Databases*, 1908:95–103, 1993.

[Tho77] W. B. Thompson. Textural boundary analysis. *IEEE Transactions on Computers*, pages 272–275, March 1977.

[TMY76] H. Tamura, S. Mori, and T. Yamawaki. Psychological and computational measurements of basic textural features and their comparison. 3rd Int. Joint Conf. Pattern Recognition, pages 273–277, Nov. 1976.

[TMY78] H. Tamura, S. Mori, and T. Yamawaki. Textural features corresponding to visual perception. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-8(6), June 1978.

[TOH95] H. Treat, E. Ort, and J. Ho. Searching images using Ultimedia Manager. IBM Santa Teresa Lab, San Jose, CA, 1995.

[WAL+93] J. K. Wu, Y. H. Ang, P. C. Lam, S. K. Moorthy, and A. D. Narasimhalu. Facial image retrieval, identification and inference system. *ACM Multimedia '93*, June 1993.

[WDR76] J. S. Weszka, C. R. Dyer, and A. Rosenfeld. A comparative study of texture measures for terrain classification. SMC-6(4):269–285, April 1976.

[ZLSW95] H. J. Zhang, C. Y. Low, S. W. Smoliar, and J. H. Wu. Video parsing, retrieval and browsing: an integrated and content-based solution. *ACM Multimedia '95*. November 1995.

[ZT75] A. L. Zobrist and W. B. Thompson. Building a distance function for gestalt grouping. *IEEE Transactions on Computers*, C-4(7):718–728, July 1975.

[Zuc76] S. W. Zucker. Towards a model of texture. *Computer Graphics and Image Processing*, 5:190–202, June 1976.

# VITA AUCTORIS

**Donato Ingratta** was born in 1968 in **Leamington, Ontario**. He graduated from **Leamington District Secondary High School** in 1986. From there he went on to the **University of Windsor** where he obtained a B. Sc. in mathematics in 1993 and a B. Sc. in Computer Science in 1994. He is currently a candidate for the Master's degree of Science at the University of Windsor and hopes to graduate in the Spring of 1998.