4-19-2006

RISE: A ROBUST IMAGE SEARCH ENGINE

Debangshu Goswami
University of Missouri-St. Louis, debangshu@yahoo.com

RISE: A ROBUST IMAGE SEARCH ENGINE

by

DEBANGSHU GOSWAMI
Bachelor of Engineering

A THESIS

Submitted to the Graduate School of the

UNIVERSITY OF MISSOURI- ST. LOUIS
In partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

in

COMPUTER SCIENCE

May, 2006

Abstract

This thesis advances RISE (Robust Image Search Engine), an image database application designed to build and search an image repository. RISE is built on the foundation of a CBIR (Content Based Image Retrieval) system. The basic goal of this system is to compute content similarity of images based on their color signatures.

The color signature of an image is computed by systematically dividing the image into a number of small blocks and computing the average color of each block using ideas from DCT (Discrete Cosine Transform) that forms the basis for JPEG (Joint Photographic Experts Group) compression format. The average color extracted from each block is used to construct a tree structure and finally, the tree structure is compared with similar structures already stored in the database.

During the query process, an image is given to the system as a query image and the system returns a set of images that have similar content or color distribution as the given image. The query image is processed to create its signature which is then matched against similar signature of images already stored in the database. The content similarity is measured by computing normalized Euclidean distance between the query image and the images already stored in the database.

Preliminary work in this area was done by Climer and Bhatia [4, 5]. They described a method specific to gray scale JPEG images that used JPEG coefficients to build indices, which could be used for searching images. My thesis builds on their work and here I have described how the concepts described in their paper could be made more general and extended to other image formats and also applied to color images. Besides, I also researched how the system could be made more efficient by using various efficient information storage techniques such as using an RDBMS (Relational Database Management System) to store image data. I have used object-oriented design techniques in my implementation and deployed RISE as a web browser

based search engine. RISE has a GUI (Graphic User Interface) front end and a Java servlet in the back end that searches the images stored in the database and returns the results to the web browser. RISE enhances the performance of image operations of the system using JAI (Java Advance Imaging) tools.

RISE only compares the color signature of each image that is precomputed and stored in RDBMS. Hence, its methodology is substantially efficient because there is no need to extract complete information for every image. It uses $L^*a^*b^*$ color space for comparisons in perceptual color space.

I build a quad tree structure from image information. The quad tree structure of images attributes to extensive early pruning. Also, I save the quad trees in an RDBMS. Use of RDBMS ensures faster response in terms of access and retrieval of image data. Use of RDBMS and Java also facilitates portability. Robustness of this systems stems from the fact that this may be applied to all popular image formats and it could be easily extended to any image format. RISE supports most popular image formats such as BMP, JPEG, GIF and TIFF.

Contents

List of Figures

Chapter 1

Introduction

The accurate maintenance and fast search of images has become more and more challenging in this era of internet when innumerable digitized images are being created every second by digital devices such as cameras, scanners, satellites, scientific experiments and home entertainment systems. The extremely large number of images makes it a Herculean task to build an image database, let alone searching for an image that we actually need. This leads us to the vital question regarding how these images should be organized so that we could find them accurately and efficiently with ease.

In current real world image databases, the prevalent retrieval techniques involve human supplied text annotations to describe image semantics. This method is fraught with several serious drawbacks. First, the semantic complexity of an image leads to different descriptions, resulting in content and/or language mismatch [3, 19]. A content mismatch occurs when a seemingly insignificant object or characteristic is omitted in an annotation and, later, a user searches for that omitted information. Language mismatches occur when different words are used to describe the same object. Second, many applications, such as weather forecasting or law enforcement, involve a significant number of similar images that would result in an unmanageable number of matches for a given query. Finally, this method is severely limited as it is time intensive, involves a human element to annotate images, and thus impractical for large databases.

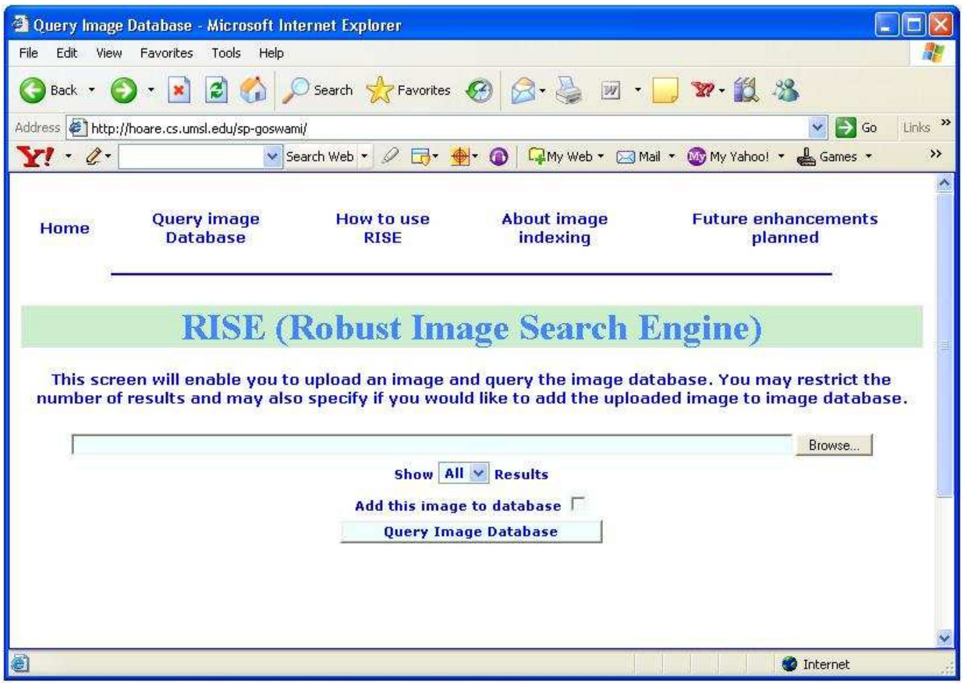
The semantic complexity for manual annotation confounds the effort to automate image indexing and retrieval. While humans can easily discern objects in an image, they are unable to describe its full semantic content in an unambiguous language that can be tokenized for automatic indexing and retrieval [4, 5]. At the present time, the technology has not advanced to the point for a computer to discern an image or to recognize and describe its contents unambiguously. This is due to the limitations of the current edge detection and object recognition techniques. Yet, the need for efficient automated retrieval systems has dramatically increased in recent years due to the advances in world wide web and multimedia technology.

Current attempt to solve this problem has led to the technique of content based image retrieval. A content based image retrieval (CBIR) system retrieves images based on their contents. There are two query methods in CBIR: query-by-example and query-by-memory [21]. In query by example, a user selects an example image as the query. In query by memory a user selects image features from his/her memory (such as color, texture, spatial attributes, and shape) to define a query.

In recent years, many CBIR techniques have been developed to address the growing need for efficient image management systems. In most of these systems, a user can supply or construct a query image, or enter text to initiate a search for matching images. Generally, these systems return a number of images with the closest match to the query. Some systems, such as QBIC [6], WALRUS [11], and my system RISE, may use more than one method to compute the query as described later in this thesis.

In this thesis, I present the development and implementation of RISE(Robust Image Search Engine) to store and search images based on the contents of the images. RISE uses a technique to develop a color signature of an image using concepts similar to ones used to build JPEG coefficients for the JPEG image format. RISE provides interaction with the user using a GUI (Graphical User Interface), as shown in Figure 1.1. The GUI may be used to upload an image to perform a query. RISE scores over other JPEG-based techniques in that it may be applied to many image formats. The technique involves dividing a given image into 8×8 blocks and then computing color signature of each 8×8 block by computing average of color components. Conceptually, this is equivalent to the DC value in the DCT-transformed image coefficients in JPEG. Next, I form a quad tree structure [17] with the color signatures. It is easy to note that an image may be abstracted in terms of probability distribution function in different colors. Hence, we are able to derive a color signature for an image that is unique to a given image.

In the next chapter, I discuss perpetual significance of color and various color spaces as relevant to my thesis. In chapter 2, I discuss what color signature is and I present the problem background in detail and review the literature. Then I present the basics of computing color signature and JPEG compression which is the concept behind computing color component average. In chapter 3, I show the development of the quad tree corresponding to the color component average and the information to be retained at each node of the quad tree. In chapter 4, I discuss how quad tree information is stored and retrieved from RDBMS. In chapter 5, I illustrate the query process. In chapter 6, I discuss my implementation. Finally, I conclude by presenting summary and future plans regarding the system in chapter 7.



RISE (Robust Image Search Engine)

This screen will enable you to upload an image and query the image database. You may restrict the number of results and may also specify if you would like to add the uploaded image to image database.

Show All Results

Add this image to database ☐

Figure 1.1: GUI Interface

Chapter 2

Perception of color

Color is the perceptual result of light in the visible region of the electromagnetic spectrum, incident upon the retina. The visible region has wavelengths in the region of 400 nm to 700 nm. It is normally characterized by the total amount of energy (or radiance) that flows from a light source and is expressed in a spectral power distribution (SPD), often in 31 components each representing a 10 nm band. The color energy is sensed inside human eye by cones in the retina that respond to the Red (R), Green (G), and Blue (B) components at different rates to produce color sensation for the brain. [15] Color spaces are mathematical models used to specify, create and visualize color. Different models will represent the same color in different ways. Various color spaces (also called color models or color system) have been proposed. The most commonly used spaces are the RGB (red, green, blue), the CMY (cyan, magenta, yellow), CMYK(cyan, magenta, yellow, black), HSI(hue, saturation, intensity) and $L^*a^*b^*$ color space. In essence, a color model is a specification of a coordinate system and a subspace within that system where each color is represented by a single point in three dimensional space.

2.1 RGB Color space

In RGB color space, each color appears in its primary spectral components of red green and blue. The RGB color space can be geometrically represented as a 3 dimensional cube. The coordinates of each point inside the cube represent the values of red, green and blue components, respectively. RGB color space matches nicely with the fact that the human eye is strongly perceptive to red, green and blue primaries. But absorption of these three colors is not same due to anatomy of human eye. Hence, RGB color space is suitable for color display, but not good for color scene segmentation and analysis because of the high correlation between the R, G, and B components. High correlation means that if the intensity changes, all the three components may change.

2.2 $L^*a^*b^*$ Color space

$L^*a^*b^*$ is the most complete color model, developed by CIE (Commission Internationale d'Eclairage), to describe all the colors perceived by the human eye. It is an absolute color model that separates the luminance, the overall brightness, from colors. This color model represents colors in three dimensional space as a cylinder. In this cylinder L^* axis represents luminance channel L. The contrast between red and green, which is represented as channel a, represents axis a^*. The contrast between blue and yellow, which is represented as channel b, represents axis b^* as shown in Figure 2.1. These three channels (L^*, a^* and b^*) attribute to the unique name for this color model.

$L^*a^*b^*$ is a powerful color space that can cause dramatic changes in our perception of an image by creating and destroying contrast easily and quickly. The separation of luminance channel from colors makes it possible to do image operations such as sharpening without accidentally adding color casts.

The range of luminance channel lies between 0 and 100. If a^* and b^* values are zeros, for L^* 0 yields black and 100 yields white. The range of channel a lies between -86 to +86, where

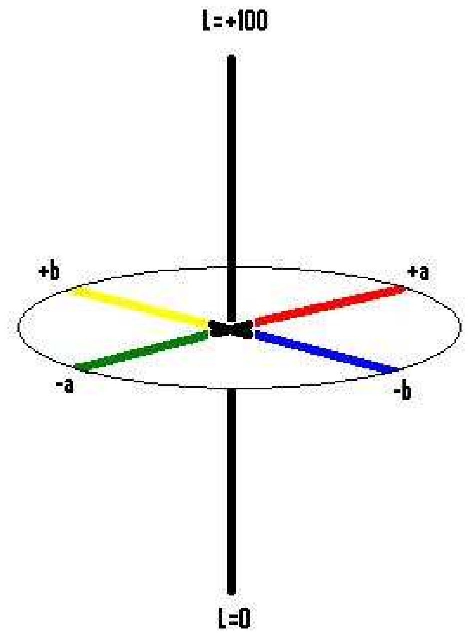


Figure 2.1: Lab color space in 3D co-ordinates

negative values indicate green and positive values indicate red. The range of channel b lies between -108 and +94 where negative values indicate blue and positive values indicate yellow. It is easy to derive the numeric range of these channels from the conversion formula described in the following section.

The $L^*a^*b^*$ color model has been created to serve as a device independent, absolute model to be used as a reference. It provides a linear response to human visual perception abilities. In the next section, I present the conversion of a pixel from RGB to $L^*a^*b^*$ color space.

2.3 Conversion to $L^*a^*b^*$ color space from RGB

The conversion from RGB to $L^*a^*b^*$ color space is based on a standard. This standard is based on international agreement on primaries for high definition television (HDTV). These primaries are closely representative of contemporary monitors used in computing and computer graphics. The transform based on Rec. 709 [8] to convert RGB to $L^*a^*b^*$ is presented next.

The transform is based on CIE XYZ which is a special case of tristimulas values, or set of three linear light components that embed the spectral properties of human color vision. The components are computed as:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

The X,Y,Z values are converted to $L^*a^*b^*$ as follows:

$L^* = 116[f(Y/Y_n) - 16]$

$a^* = 500[f(X/X_n) - f(Y/Y_n)]$

$b^* = 200[f(Y/Y_n) - f(Z/Z_n)]$

Here X_n, Y_n and Z_n are the CIE XYZ tristimulus values of the reference white point known as D_{65} in Rec. 709. A white point is one of a number of reference illuminants which serves to define the color white.

$X_n = 242.36628$

$Y_n = 255.0$

$Z_n = 277.63228$

The function f is defined as:

$$f(t) = \begin{cases} t^{1/3}, & \text{if } t > 0.008856 \\ 7.787t, & \text{otherwise} \end{cases}$$

The division of the $f(t)$ function into two domains is done to prevent an infinite slope at $t = 0$. $f(t)$ is assumed to be linear below some $t = t_0$, and is assumed to match the $t^{1/3}$ part of the function at t_0 in both value and slope. In other words:

$t_0^{1/3} = at_0 + b$

$1/(3t_0^{2/3}) = a$

The value of b is chosen to be 16/116. The above two equations can be solved for a and t_0:

$a = 1/(3\delta^2) = 7.787037$

$t_0 = (\delta^3) = 0.008856$

In the next section, I present the use of $L^*a^*b^*$ color space to compute the color signature of an image.

2.4 Color signature computation methods

The visual perception is a powerful source of information about the world around us. As a consequence, a considerable part of the data that we collect comes in the shape of visual material such as photographs and video. However, while the costs of collecting and storing all these images are dwindling, searching such databases efficiently for specific material is becoming increasingly challenging. It is indeed difficult to categorize images unambiguously [13]. Most of us probably would prefer to tell the computer to fetch all images of a particular object instead of getting to the tedious details about a picture. One step closer to that goal is to be able to provide an image to the computer and ask it to fetch all similar images. This leads us to the question regarding how a digital image could be quantized into a set of numbers so that we can use a computer to search images based on its contents.

An image is a composition of various intensities of colors. These intensities can be transformed into a set of numbers, called a color signature, that quantizes intensities of various colors of overall image or of a given region of the image. In other words, we can use the distribution of color components to compute color signature of an image. However, as explained in the following section, only consideration of distribution of color does not give a true representation of an image. Two images could be different as in Figure 2.2 and yet have the same color distribution. This is a problem with computing signature we are trying to ameliorate. Therefore, it is essential to take various regions of the image into account as well. This is where the concept of computing the color signature comes in place which considers both the color and the region of an image where that color comes from. In general, color signature

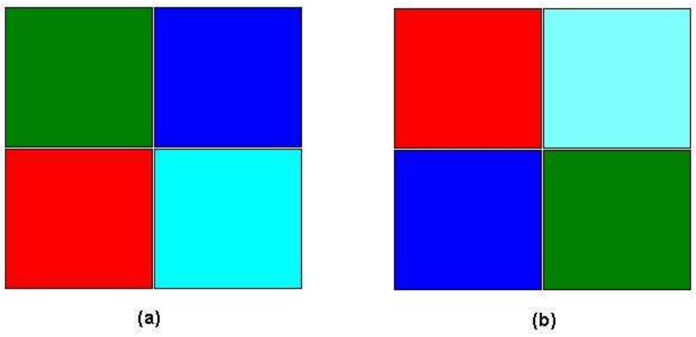


Figure 2.2: Two images with opposite colors, yet identical histograms average intensities

computation methods may be divided into three broad categories - histogram-based, color layout-based and region-based. In the remainder of this section, I present the descriptions of each of these three categories.

2.4.1 Histogram based systems

A histogram of an image conveys the relative quantities or probability distribution of colors in an image. Inside an image, each pixel in the image is represented as a weighted amount of three primary colors: red, green, and blue. A histogram is computed by quantizing each of these three color channels into m divisions. Thus, we can have m^3 composite colors, or bins. The total numbers of pixels that correspond to each bin are tallied and the signature of the image is a vector containing the number of pixels in the image for colors corresponding to bins. The signature is then used to build an index. When a query image is presented, its signature is extracted and compared with entries in the index.

The histogram-based systems suffer from several limitations. These systems disregard the shape, texture, and object location information in an image, leading to a high rate of return of semantically unrelated images. For instance, the histograms of the two images in Figure 2.2 are identical, yet the images are obviously different. Furthermore, the color quantization step leads to additional sources of errors [9].

For efficiency, there usually are far fewer bins than all possible colors, so many colors may get quantized into a given bin. The first problem is that similar colors that are near the division line may get quantized into different bins and for a given bin, the set of colors in the range for the bin may be quite different. Second, given a set of three color channels, perceptual sensitivity to variations within colors is not equal for all three channels [10]; however, histogram quantization incorrectly uses a uniform divisor for all three channels. Finally, a query image may contain colors that are similar to the colors of a particular image in the index, but a large distance may result if they are not close enough to fall into bins [9] that are close to each other.

2.4.2 Color layout-based systems

Color-layout based methods extract signatures from images that are similar to low resolution copies of the images. In these systems, an image is divided into a number of small blocks and the average color of each block is stored as signature. Some systems, such as WBIIS [23] and WALRUS [11], utilize significant wavelet coefficients instead of average values in order to capture sharp color variations within a block. Signatures derived in the WBIIS system include coefficients derived from a fast wavelet transform with Daubechies wavelets and their standard deviation [23].

Traditional color layout-based systems are limited because of their intolerance to object translation and scaling. The object location is frequently helpful in identifying semantically similar images. For instance, the histogram of a query image containing green grass and blue sky may have a close similarity to a bluish house with a green roof, while a color layout scheme would discard this image. Also, these systems lack the desirable feature of tolerance of object translation.

The WALRUS system uses a wavelet-based color layout method to overcome the translation and scaling limitations [11]. For each image in the database, the system computes and clusters a variable number of signatures (typically thousands). Each signature is computed on a square area of the image, with the size and location of each square varying according to some prescribed parameters. This system performs clustering on squares with similar signatures in order to reduce the number of signatures to be searched during a query. While this system is tolerant of image translation and scaling, its computational complexity is dramatically increased. WALRUS is essentially a color layout scheme; however, it incorporates some region based system attributes.

2.4.3 Region-based systems

Region based methods use local properties of regions (ideally objects) as opposed to the use of global properties of an entire image to compute the signature. A fundamental stumbling block for these systems is that objects are frequently divided into multiple regions, each of which may inadequately identify the object. Examples of region based systems include QBIC [6], SaFe [20], Blobworld [2], and SIMPLIcity [22].

The QBIC system uses both local and global properties and incorporates both region based and histogram properties. It identifies objects in images using semiautomatic outlining tools [6].

SaFe is a complex system that automatically extracts regions and allows queries based on special arrangements of the regions. It automatically extracts regions using a color set back-projection method [20]. Characteristics such as color, shape, texture, area, and location, are stored for each region. A separate search is performed for each region in the query image. Blobworld is a region based system that automatically defines regions, or blobs, within an image using the Expectation-Maximization algorithm on six dimensional vectors containing color and texture information for each pixel [2]. For each blob, this system stores six pieces of information that include the anisotropy, orientation, contrast, and two dominant colors. SIMPLIcity is a region-based system that partitions images into predetermined semantic classes prior to extracting the signature [22]. Signature construction and distance formulations are varied according to the semantic class. The k-means algorithm and Haar wavelet are used to segment the image into regions [22].

In the next section Ill describe the JPEG scheme to compress an image, and its use in RISE.

2.5 JPEG and DCT

JPEG derives its name from Joint Photographic Experts Group and is a well established standard for compression of color and grayscale images for the purpose of storage and transmission [10, 14, 18]. The JPEG compression results in an image that is considerably similar in quality to the original image while using far less bytes, typically from 20:1 to 25:1 ratio of compression without a perceivable quality degradation. The compression itself is lossy which means that some information in the image may be lost depending on the desired amount of compression. The minimal subset of the JPEG compression standard, known as the baseline JPEG, is based on the discrete cosine transform (DCT).

To apply DCT, each pixel in the image is level shifted by 128 by subtracting 128 from each value. Then, the image is divided into 8×8 size blocks and DCT is applied to each block, yielding DCT coefficients for the block. These coefficients are quantized using weighting functions optimized for the human eye. The resulting coefficients are encoded using a Huffman variable word length algorithm to remove redundancies [10].

The compression process is started by dividing the rectangular image canvas into 8×8 blocks and the DCT is applied to each block to separate the high and low frequency information in the block. Application of DCT results in the average value (or DC component) in location (0,0) of the 8×8 block while the other locations of the 8×8 block contain the AC terms. The AC terms are made up of higher frequency components of the block. The maximum value for an AC term, the dominant coefficient, represents the highest change in the cosine wave between any two adjacent pixels in the block. The DCT coefficients in each block are quantized to get scaled coefficients by dividing each value with a quantization coefficient from the quantization table developed by the ISO JPEG [14]. The coefficients in the quantized block are rearranged using a zigzag ordering to create a vector. The zigzag pattern approximately orders the basis functions from low to high spatial frequencies [14]. The vectors resulting from the zigzag ordering contain the DC coefficient corresponding to the original 8×8 block in the first location of the block. The baseline JPEG standard requires these vectors of DCT coefficients to be run length encoded, using Huffman encoding to remove redundancy, for storage and transmission.

Since I wanted to include different types of image formats, I could not use JPEG Coefficients which would have required an extra step to convert a given image to JPEG. I have decided to use the concept of computing JPEG Coefficients instead. As explained above, a given image is abstracted into a set of 8×8 pixel blocks. Thus we need to compute an average of 8×8=64 pixels for each block.

Average color for a given block i in RGB color space is derived as:

$$(r, g, b)_i = \left(\frac{1}{64} \sum_{j=0}^{63} R_j, \frac{1}{64} \sum_{j=0}^{63} G_j, \frac{1}{64} \sum_{k=0}^{63} B_j \right)$$

where

R_j, G_j and B_j are the red, green and blue component of jth pixel in the ith block

Average color for a given block i in $L^*a^*b^*$ color space is derived as:

$$(L^*, a^*, b^*)_i = \left(\frac{1}{64} \sum_{i=0}^{63} L_j^*, \frac{1}{64} \sum_{j=0}^{63} a_j^*, \frac{1}{64} \sum_{k=0}^{63} b_j^* \right)$$

where

L_j^*, a_j^* and b_j^* are the L^*, a^* and b^* component of jth pixel in the ith block and are converted from the red, green and blue components of the pixel as explained in section 2.3

Chapter 3

Quad tree structure representation of average color components

A quad tree is a data structure in which each internal node has up to four children. This data structure can be used efficiently to represent an image at different levels of detail (LOD). A point region (PR) quad tree is a type of quad tree where each node must have exactly four children, or be a leaf, having no children. The PR quad tree represents a collection of data points in two dimensions by decomposing the region containing the data points into four equal sub quadrants, that are further subdivided into four sub quadrants each and so on, until no leaf node contains more than a single point. For simplicity, I consider the PR quad tree as quad tree.

In the previous chapter, I have shown how I divide the entire image into 8×8 blocks to extract the average color of each 8×8 block. This effectively implies that the entire image is composed of 8×8 pixel blocks. I consider the average value of 8×8 block to be the smallest addressable unit of the image instead of each pixel. Here I describe the construction of the quad tree structure using the information from each 8×8 block.

The quad tree in RISE is a full tree such that each node contains exactly four children or none (Figure 3.1). Moreover, all the leaf nodes are at the same level in the tree, and represent

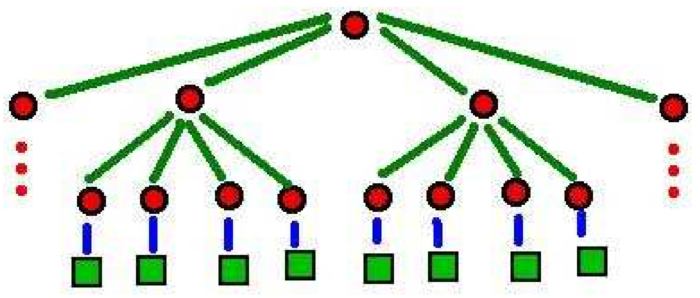


Figure 3.1: Quad tree

the 8×8 pixel block in the original uncompressed image. To develop the quad tree, first I scale a given image into 512×512 pixels. Thus, the length of each side of the new square image is a power of 2. A quad tree as in Figure 3.1 can be derived from the square block by recursively dividing each side by 2 as shown in Figure 3.2.

3.1 Leaf nodes of the quad tree

Each leaf node of the quad tree corresponds to a tristimulus color vector to describe average of color components of the block that has been computed from an 8×8 pixel block in the original image. I have used the term tristimulus color vector as RISE is not limited on just the RGB color space but can work in any color space.

In addition to the average of the color components, the node also retains the name of the image for identification purpose.

3.2 Internal nodes of the quad tree

The internal nodes contain information extracted from aggregates of 8×8 average color components. They are used to make similarity comparisons between images at a higher level than the individual blocks. Just like in leaf nodes, I retain the name of the image for identification purpose. Lastly, the non leaf nodes also contain the links to their four children, identified

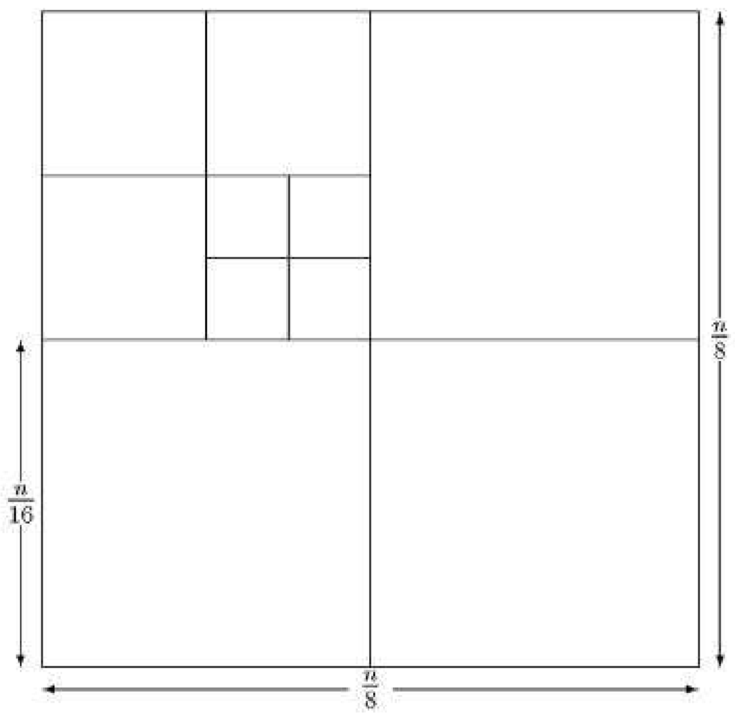


Figure 3.2: The quad tree of an image.

```
public class QuadTree {
    Point blockStart;            /* Starting coordinates of the block         */
    int blockHeight;             /* Height of JPEG coefficients block         */
    int blockWidth;              /* Width of JPEG coefficients block          */
    int size;                    /* Size of the side of enclosing square block */
    float avgCoefficient;        /* Average coefficient value in the block    */

    String imageName;            /* Name of the image (for retrieval usage)   */
    QuadTree upperLeftChild;     /* Upper left child                          */
    QuadTree upperRightChild;    /* Upper right child                         */
    QuadTree lowerLeftChild;     /* Lower left  child                         */
    QuadTree lowerRightChild;    /* Lower right child                         */
```

Figure 3.3: Definition of a node in the quad tree.

respectively as upper left child, upper right child, lower left child and lower right child. The complete definition of each node is presented in Figure 3.3. It may be noted that a leaf node uses the same structure as in Figure 3.3 by assigning a null value to the four children. The quad tree itself is illustrated in Figure 3.1. In this figure, we start with the set of 8×8 average color components forming a $\frac{n}{8} \times \frac{n}{8}$ square corresponding to an n×n image at the root.

The set of coefficients is divided into four sets of $\frac{n}{16} \times \frac{n}{16}$ coefficients each. I continue to subdivide each set into four subsets recursively until I get a set containing only one 8×8 average color component. At each node in the tree, I calculate the information in the node (Figure 3.3).

The nodes in the tree are at distinct levels that can be identified by the size of the image. I save the nodes in a database table column LEVEL_nn depending upon their level in the tree, where *nn* indicates the level as measured from root. It is easy to see that all the segments at a given level are of the same size. The nodes are saved by performing a level order traversal of the quad tree with the following collating sequence: upper left, upper right, lower left, and lower right.

In the next chapter, I describe the organization and storage of the quad tree structure in a relational database management system (RDBMS).

Chapter 4

Relational Database Management System

A relational database management system (RDBMS) reliably manages large volumes of data while delivering high performance. The high performance is achieved by faster storage and retrieval of data. In this thesis I have used the terms RDBMS and database synonymously. Using an RDBMS to store information is an effective way of delegating issues of efficient storage and retrieval of data to a database. Using a conventional database to store information has many advantages. Firstly, this ensures that the wheel is not reinvented. Secondly this enables us not to worry about efficient use of expensive hardware devices and lets us concentrate more on logic of our program and presentation of our data. Finally, this makes the software portable since hardware or operating system specific information are not a part of the application, it could be easily migrated across platforms if necessary without changing code. Also, I have used (Structured Query Language) SQL to access database. This is standardized across databases and so, I am not limited in my choice of database.

However, use of a database is not cost free. We incur overheads when we use a database because we need to connect to a database first before using it and this connection process may take even more time if it involves authentication. However this connection time is fairly constant. So, for a large database this trade-off may seem negligible when we consider the benefits of using RDBMS.

Another problem is that it takes time to send a request to database and receive data. This may become significant if a large number of records are fetched from database. However, this problem may be resolved by devising technique to minimize the number of accesses to database, for example, by bringing larger chunk of data in each access and thus reducing the impact of making too many accesses to the database.

Since I foresee my image database to grow large in volume with the passage of time, I deemed it fit to use a database to store information to achieve higher order of efficiency, robustness and portability. In the following section, I present a description of database characteristics used in RISE.

4.1 Database Data types

Each attribute of an object manipulated by a database has a data type associated with it. Example of data types as specified in the ANSI (American National Standards Institute) specification [1] include among others `CHARACTER`, `CHARACTER VARYING` and `INTEGER`. Data type of a value associates certain properties with the value. These properties cause the database to treat of one data type value differently from another. For example two `INTEGER` values may be added but two `CHARACTER` values may not. A brief description of data types of interest in RISE is presented below.

`CHARACTER` data type: This ANSI data type specifies a fixed-length character string.

`CHARACTER VARYING` data type: This ANSI data type specifies a variable-length character string.

`INTEGER` data type: This ANSI data type specifies an integer.

4.2 Database Structures

Structures are well-defined objects (such as tables) that store or access the data in a database. Structures and the data contained within them can be manipulated by database operations.

A table is the basic unit of data storage in a database. The tables of a database hold all of the user-accessible data. Table data is stored in rows and columns. Every table is defined with a table name and set of columns. Figure 4.1 displays an example of a table structure. I have used this table to store a quad tree.

In my example IMAGE_NAME is the first column, LEVEL_0_L is the second column and so on and so forth. Each column is given a column name (for example, IMAGE_NAME for first column on in Figure 4.1) , a datatype (for example, `CHARACTER VARYING` for first column on in Figure 4.1) , and a width (100 for first column in Figure 4.1)

Once a table is created, valid rows of data can be inserted into it. The tables rows can then be queried, deleted, or updated. Figure 4.2 displays data in rows. For the sake of simplicity I have displayed only a few column values and restricted the display to a few rows.

Indexes are optional objects that are the primary method of providing faster access to data [12]. An index contains an entry for each value that appears in the indexed column(s) of the table and provides direct, fast access to rows. A common way to implement indexing is through B-trees.

4.3 Structured Query Language (SQL)

SQL is a data sublanguage. The purpose of SQL is to provide an interface to a RDBMS and all SQL statements are instructions to the database. Therefore, SQL is the set of statements designed to access data in a database. This particular aspect of SQL sets it apart from other general purpose programming languages such as C and Java. Among the features of SQL are the following:

```
QUAD_TREE
 Name                 Type
------------------- --------------
IMAGE_NAME          CHARACTER VARYING(100)
LEVEL_0_L           INTEGER
LEVEL_0_a           INTEGER
LEVEL_0_b           INTEGER
LEVEL_1_L           CHARACTER(1)
LEVEL_1_a           CHARACTER(1)
LEVEL_1_b           CHARACTER(1)
LEVEL_2_L           CHARACTER(16)
LEVEL_2_a           CHARACTER(16)
LEVEL_2_b           CHARACTER(16)
LEVEL_3_L           CHARACTER(64)
LEVEL_3_a           CHARACTER(64)
LEVEL_3_b           CHARACTER(64)
LEVEL_4_L           CHARACTER(256)
LEVEL_4_a           CHARACTER (256)
LEVEL_4_b           CHARACTER (256)
LEVEL_5_L           CHARACTER(1048)
LEVEL_5_a           CHARACTER(1048)
LEVEL_5_b           CHARACTER(1048)
LEVEL_6_L           CHARACTER(4096)
LEVEL_6_a           CHARACTER(4096)
LEVEL_6_b           CHARACTER(4096)
```

Figure 4.1: A table definition

```
IMAGE_NAME      LEVEL_0_L LEVEL_0_A LEVEL_0_B LEVEL_1_L LEVEL_1_A LEVEL_1_B
--------------- --------- --------- --------- --------- --------- ---------
ankita.bmp             74         5         6  46474A50  05040405  050A0405
usa-flag.gif           81        16         6  47554F57  0F10140C  F8090E08
ankita.jpg             75         2        11  47484B50  02000104  0D100906
img1.jpg               47        72        61  2F2D2E30  4847484A  3D3B3C3E
sample.jpg             70        -9       -16  3C474B48  05F7F2ED  FCF1ECE7
```

Figure 4.2: Table rows

It processes sets of data as groups rather than as individual units.

- It provides automatic navigation to the data.
- It uses statements that are complex and powerful individually, and therefore stand alone.

Essentially, SQL lets you work with data at the logical level. We should be concerned with the implementation details only when we want to manipulate data. For example, to retrieve a set of rows from a table, we define a condition used to filter the rows.

SQL provides statements for a variety of tasks, including:

- Querying data
- Inserting, updating, and deleting rows in a table
- Creating, replacing, altering, and dropping objects
- Controlling access to the database and its objects
- Guaranteeing database consistency and integrity

SQL unifies all of the above tasks in one consistent language.

Data is inserted by using `INSERT` statement in SQL. It is inserted by specifying the column values within a `VALUES` clause,

Data is retrieved from tables by `SELECT` statement in SQL. General syntax of a select statement is as follows:

```
SELECT list of columns
FROM table from which to select records
WHERE condition
```

4.4 Storing image signatures to the database

As was already explained, I compare images in $L^*a^*b^*$ color space. At the very root level of the quad tree, represented here as level 0, there is only one node and L^*, a^* and b^* values at this level represent the overall color signature of the image. Therefore, we need to access this number most often for pruning unlikely matches. This is the reason why I have stored this number as an integer. Color signature of the image is used frequently for initial pruning of trees that are too distant from the query node and hence are unlikely to match with the query image. Because this column is used frequently, I have indexed these three columns for faster retrieval. I have used B-Tree indexes for indexing L^*, a^* and b^* value at root level.

However, at levels under the root I have multiple children. So, I am left with the option of adding multiple columns or creating separate rows for each node. First option is impractical because it will create a table with enormous number of columns and will be very difficult to maintain. Adding separate row works but for RISE it resulted in longer response time. Therefore I have separated each of these L^*, a^* and b^* components at each node and stored them as arrays. For example, I have 4 nodes at level 1. So, I have created three arrays of length 4 (ranging from 0 to 3), one for L^* components, one for a^* components and finally one for b^* component. L[0], a[0] and b[0] stores three components of left most child and L[3], a[3] and b[3] stores the three components for right most child. The same technique is extended for lower levels.

As in any tree structure, number of children progressively increases as we progress through each level. For my implementation I have implemented a quad tree with six generations. So, it is easier to note here that my quad tree has one node at the top level but successive levels generate 4^0, 4^1, 4^2, 4^3, 4^4, 4^5, 4^6 nodes.

4.5 Querying the database

I have used SQL (Structured Query Language) for selecting data from database.

The following select statement demonstrates how I have selected data from database.

```
SELECT *
FROM   quad_tree
WHERE  L value in database is
between +15% AND -15% of the L value of input image
```

The above query selects data from quad tree. This selects images from database that have L value within the range of ±15% of the L value of the input image. As evident from the above select statement, I select complete node in a single query statement. This avoids making expensive accesses again and again to the database. Response time is very important to RISE since it has been designed to be a real time application.

4.6 Saving data to database

Data is inserted by specifying the column values within a VALUES clause,

Here is an example of an `INSERT` statement:

```
INSERT INTO
       quad_tree
       (
       image_name,
       level_64_L,level_64_a,level_64_b,
       level_32_L,level_32_a,level_32_b,
       level_16_L,level_16_a,level_16_b,
       level_8_L,level_8_a,level_8_b,
       level_4_L,level_4_a,level_4_b,
       level_2_L,level_2_a,level_2_b,
       level_1_L,level_1_a,level_1_b
       )
       VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
```

Here I insert complete quad tree in a single insert. Serialized byte arrays are populated in the value clause.

Chapter 5

The query process

During the query process, an image is given to the system as a query image and the system returns a set of images that have similar content as the given image. The query image is processed to create its signature which is then matched against similar signature of images already stored in a database. The content similarity is measured by computing Euclidean distance between the query image and the images already stored in the database.

Once the quad tree has been developed for each image in the database and the required information saved in the database tables, the query process is fairly simple. The query in this case is considered to be an image. The image is processed to develop the quad tree of its average color component of blocks as described in previous chapter. Further steps in the query rely completely on the quad tree.

First, I compute distance between the top most level of the image. RISE uses a predefined threshold parameters to select only those images that have a distance less than the threshold. This process is repeated at each level and images with larger distances are pruned. RISE uses Euclidean distance D between image signatures by taking a summation of Euclidean distance between corresponding blocks as follows:

$$D = \frac{1}{RC} \sum_{x=0}^{C} \sum_{y=0}^{R} \sum_{c=L^*,a^*,b^*} \sqrt{(p_1(x,y,c) - p_2(x,y,c))^2}$$

where

R Number of rows in each image

C Number of columns in each image

$p_i(x, y, c)$ Pixel value for color c at position (x, y) in image i

Since the number of nodes at upper level of a quad tree is less, a full scan of the table is not expensive at this level. After initial scan RISE narrows down its choices and only computes distance for a few selected images at lower levels.

It is easy to see that in the case of a perfect match, for example, the same image in query and the database, the Euclidean distance will evaluate to zero. At any time, the images that result in the comparison greater than a pre specified threshold can be ignored from further consideration. However, it may be the case that two images which are not similar to each other in content but are similar in average intensity (the DC component for the image), may result in the comparison evaluation of zero. An example of this case is depicted in Figure 2.2. These images need to be ranked using the information in the lower nodes in the quad tree.

While comparing the lower nodes, I must keep track of the relative location of each sub block at the node in the query as well as the database and must compare only the corresponding nodes. The index has been structured such that each index file contains the nodes at a given level within the quad tree. Traversing the quad tree corresponding to the query in level order, each node is compared with the corresponding node in the image database, with the root node as the reference, and the evaluation of the comparison (the distance between query and reference for the sub block) added to the previous evaluation. At any stage, if the summation of distances during the level-order traversal comparison becomes too large, the image is removed from further consideration. I can also limit the number of images that should be kept under consideration by keeping only the top ranked images in consideration. The overall results of the query are summarized in a ranked list of images that are sorted on the basis of their distance from the queried image. The ranking in the list allows us to present the resulting

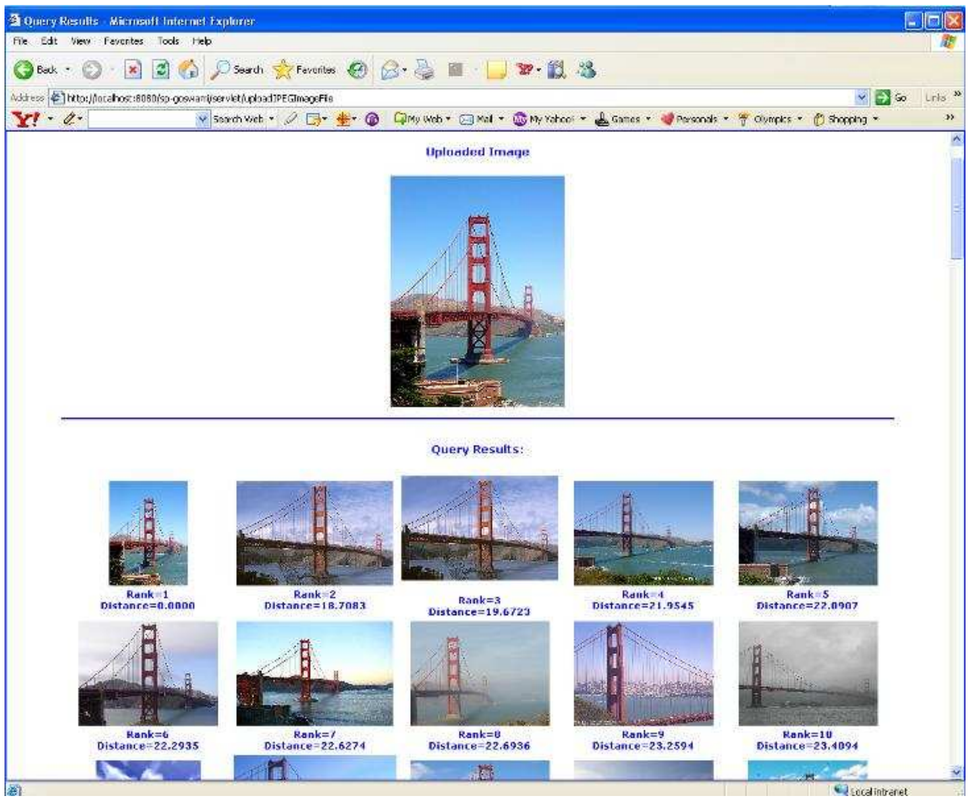


Figure 5.1: Query results: Case 1

images in decreasing order of relevance with respect to the query. In addition, I can also limit the number of images that are to be presented to the user.

I have formulated three specific test cases to demonstrate the query results of RISE. First, when the query image exists in the image database and many similar images also exist. Here the query result should find the exact match and also return those images which are very similar. Second, when the query image exists in image database and also a slight variation of that image exists in the database. Query result should return the exact match and also should pick the slight variation from database. Finally, the case when the query image does not exist in the database. Here we need to study the types of results being returned from the database.

The query result of first test case is displayed in Figure 5.1. Here, the query image is the Golden Gate Bridge. It is easy to note that the query results constitute images of the Golden Gate Bridge. But distance and hence rank changes when angle and color of sky is different.

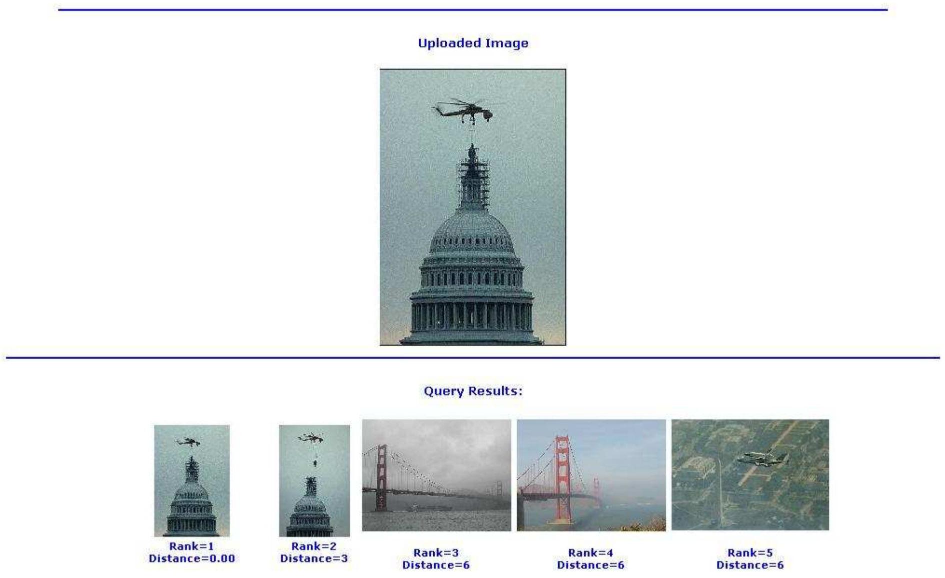


Figure 5.2: Query results: Case 2

The query result of second test case is displayed in Figure 5.2. Here, the query result shows the exact match at rank 1. Exact match is indicated by a distance of zero. The close match was returned as the second image with a distance of 3. There are other images in the query result which are unrelated to the query image.

The final case is when the query image does not exist in the database. This is being displayed in Figure 5.3. This is a satellite image of southern Jordon. It is interesting to note that the first five images returned are also aerial images. The similar background color of these images is responsible for close ranking.

5.1 Precision and recall

Despite their complexity, most of the existing CBIR systems miss relevant images in the database and may return a number of irrelevant images. An ideal system will be one that provides high value for precision and recall [16]. These terms originate in the information retrieval literature. Precision is defined as the ratio of the number of correct images retrieved to the number of images in the retrieved set. Recall is the ratio of the number of correct

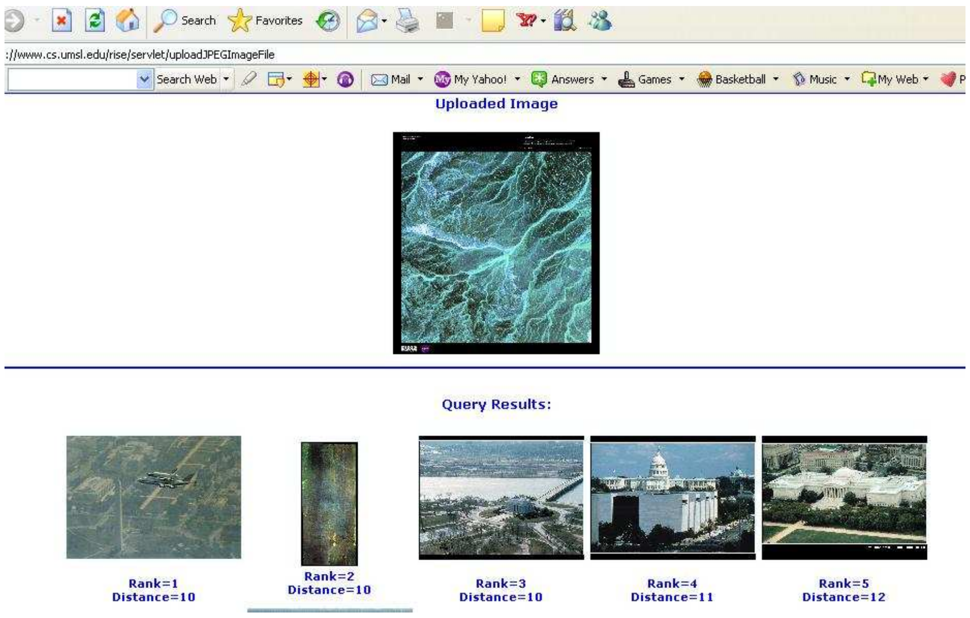


Figure 5.3: Query results: Case 3

images retrieved to the number of relevant images in the database. It is easy to see that a recall of 1 is achieved if we retrieve all the images in the database. However, that will have an adverse effect on precision which is akin to signal to noise ratio. Maximizing precision may adversely affect recall by omitting some relevant images. Precision and recall capture the subjective judgment of the user and may provide different values for different users of a system. It has been commented that even manual browsing is not error free [7].

Chapter 6

Implementation

RISE is based on using the average of color components of 8×8 blocks in $L^*a^*b^*$ color space. I used the idea of DC components of DCT used to compute JPEG coefficients which are essentially the average of color components of 8×8 blocks. Advantage of using average is that it can be applied to any image format. Each image in the database is represented by a quad tree structure with leaves that contain relevant color signature information. I compute statistics concerning each node in the quad tree and include those in the index along with the identification and location information. The quads at any level in the tree, starting from the root, are of the same size. All the quads at a given level in the tree are collected in a relational database table, with quads at different levels collected in separate relations. Just like the images in the database, a given query image is partitioned into the quad tree structure and statistics from different nodes are compared against the statistics from the quads of same size from the index relations. The comparison is quantified as a distance measure that can be used to determine the similarity of the query to different images in the data base. Using a threshold, some of the images can be ignored from further comparison yielding further improvements in retrieval efficiency. Finally, an adjustment of threshold yields the desired number of images that match the query image. The efficiency of RISE stems from three factors. First of all, I use elements as the leaves of a quad tree structure that allows extensive early pruning. Secondly, I have used a relational database for storing information which makes data retrieval faster. Finally, this is applicable to any image format. I just need to convert the image to $L^*a^*b^*$ and then divide it into 8×8 blocks and take average of color components.

RISE uses an extremely efficient color layout method with region based characteristics. The signature of query image is based on its global characteristics, however, regional properties of the indexed images are searched. At the present time, it is tolerant of limited object translation.

RISE has been designed using object oriented paradigm. The GUI in front end and the backend objects used for searching are implemented as a Java servlet on Sun SPARC system and hosted in a Tomcat web server. I have used Oracle as the relational database for storing quad trees. The initial GUI input screen may be seen in Figure 1.1. I have used JAI(Java Advanced Imaging) interface for all image related operations. This makes the processing extremely fast as JAI is already optimized for those operations.

A quad tree was computed for each image and index information were stored in the database. This part runs as a command line. The user may upload an image using any standard web browser and query the image database. The output is displayed as a ranked list of images from the database.

Currently, the output in the browser displays set of retrieved images, their distance from the query image and a rank.

Chapter 7

Summary

In this paper, I present an image database indexing system for efficient storage and retrieval of images in response to a query expressed as an example image. RISE can be classified as a content-based image retrieval system and is very efficient. The efficiency stems from indexing images using JPEG coefficient like average of three color components which can be applied to any given image format to compute image signature in the form of a quad tree. Use of quad tree structure allows extensive early pruning during query so that any images that are not promising for the query are eliminated from consideration at an early stage. The system accepts all popular image formats by extracting RGB raster data. RISE converts these raster information to $L^*a^*b^*$ color space and computes average of color components. These values become the leaf nodes of the quad tree. Four 8×8 blocks comprise a 16×16 block of the image. The nodes on the level next to the leaf nodes each contain the average of four leaf nodes (corresponding to a 16×16 block). This procedure is repeated until the root node is constructed, containing the average value for the entire image. This tree is the signature of the image that is stored in the index. A query image is processed to produce a quad tree, and the roots of the trees in the index are compared with the query root using a Euclidean distance formula. For distances within a given tolerance, the second level of the corresponding trees are compared. This process repeats down to the leaf nodes, and selected images are ordered and returned to the user. The system has been implemented in Java with JAI interface on a Sun workstation and Oracle was the relational database used to save quad trees.

Bibliography

[1] Database language foundation. *ANSI*, (X3.135), 1999.

[2] S. Belongie, C. Carson, H. Greenspan, and J. Malik. Color- and texture-based image segmentation using the expectation-maximization algorithm and its application to content-based image retrieval. In *ICCV98: Proceedings of the Sixth International Conference on Computer Vision*, pages 675–682, 1998.

[3] S.-F. Chang, J. R. Smith, M. Beigi, and A. Benitez. Visual information retrieval from large distributed on-line repositories. *Communications of the ACM*, 40(12):63–71, December 1997.

[4] S. Climer. Image database indexing using compressed data. *MS Thesis, UMSL*, 2001.

[5] S. Climer and S. K. Bhatia. Image database indexing using JPEG coefficients. *Pattern Recognition*, 35(11):2479–2488, November 2002.

[6] M. Flickner, et. al. Query by image and video content: The QBIC system. *IEEE Computer*, 28(9):23–32, September 1995.

[7] D. Forsyth, J. Malik, and R. Wilensky. Searching for digital pictures. *Scientific American*, 276(6):88–93, June 1997.

[8] ITU-R Recommendation BT.709. Basic parameter values for the HDTV standard for the studio and for international programme exchange. Technical Report BT.709 [formerly CCIR Rec. 709], International Telecommunications Union, 1211 Geneva 20, Switzerland, 1993.

[9] G. Lu and B. Williams. An integrated WWW image retrieval system. In *Australian* WWW *Conference*, April 1999.

[10] J. D. Murray and W. vanRyper. *Encyclopedia of Graphics File Formats (2nd ed.).* O'Reilly, Sebastopol, CA, 1996.

[11] A. Netsev, R. Rastogi, and K. Shim. WALRUS: A similarity retrieval algorithm for image databases. In A. Delis, C. Faloutsos, and S. Ghandeharizadeh, editors, *SIGMOD 1999: Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 395–406, Philadelphia, PA, June 1999. ACM Press.

[12] Oracle Press. *Oracle9i Database Concepts Release 1 (9.0.1)*, July 2001. No. A88856-02.

[13] D. E. Pauwels. Content-based image retrieval and processing. 2005.

[14] W. B. Pennebaker and J. L. Mitchell. *JPEG Still Image Data Compression Standard.* van Nostrand Reinhold, New York, 1993.

[15] C. Poynton. A guided tour of color space. In *New Foundations for Video Technology: Proceedings of the SMPTE Advanced Television and Electronic Imaging Conference*, pages 167–180, San Francisco, CA, February 1995.

[16] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, 1983.

[17] H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, June 1984.

[18] A. N. Skodras. Direct transform to transform computation. *IEEE Signal Processing Letters*, 6(8):202–204, August 1999.

[19] J. R. Smith and S.-F. Chang. Automated image retrieval using color and texture. Technical Report 414-95-20, Columbia University, Department of Electrical Engineering and Telecommunications Research, New York, NY 10027, July 1995.

[20] J. R. Smith and S.-F. Chang. Integrated spatial and feature image query. *International Journal of Multimedia Systems*, 7(2):129–140, March 1999.

[21] E. L. van den Broek, P. M. F. Kisters, and L. G. Vuurpijl. Design guidelines for a content-based image retrieval color-selection interface. In *Chi 2004 Proceedings*, Vienna, Austria, June 2004.

[22] J. Z. Wang, J. Li, and G. Wiederhold. SIMPLIcity: semantic sensitive integrated matching for picture libraries. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(9):947–963, 2001.

[23] J. Z. Wang, G. Wiederhold, O. Firschein, and S. X. Wei. Content-based image indexing and searching using Daubechies' wavelets. *International Journal on Digital Libraries*, 1(4):311–328, 1997.