RISE: A Robust Image Search Engine

Debangshu Goswami Sanjiv K. Bhatia Ashok Samal

In this article we address the problem of organizing images for effective and efficient retrieval in large image database systems. Specifically, we describe the design and architecture of RISE, a Robust Image Search Engine. RISE is designed to build and search an image repository, with an interface that allows for the query and maintenance of the database over the Internet using any browser. RISE is built on the foundation of a CBIR (Content Based Image Retrieval) system and computes the similarity of images using their color signatures.

The signature of an image in the database is computed by systematically dividing the image into a set of small blocks of pixels and then computing the average color of each block. This is based on the Discrete Cosine Transform (DCT) that forms the basis for popular JPEG image file format. The average color in each pixel block forms the characters of our image description. Organizing these pixel blocks into a tree structure allows us to create the words or tokens for the image. Thus the tokens represent the spatial distribution of the color in the image. The tokens for each image in the database are first computed and stored in a relational database as their signatures. Using a commercial relational database system (RDBMS) to store and query signatures of images improves the efficiency of the system. A query image provided by a user is first parsed to build the tokens which are then compared with the tokens for images in the database.

During the query process, tokenization improves the efficiency by quantifying the degree of match between the query image and images in the database. The content similarity is measured by computing normalized Euclidean distance between corresponding tokens in query and stored images where correspondence is defined by the relative location of those tokens. The location of pixel blocks is maintained by using a quad tree structure that also improves performance by early pruning of search space. The distance is computed in perceptual color space, specifically $L^*a^*b^*$ and at different levels of detail. The perceptual color space allows RISE to ignore small variations in color while different levels of detail allow it to select a set of images for further exploration, or discard a set altogether.

RISE only compares the precomputed color signature images that are stored in an RDBMS. It is very efficient since there is no need to extract complete information for every image. RISE is implemented using object-oriented design techniques and is deployed as a web browser-based search engine. RISE has a GUI (Graphical 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 by using JAI (Java Advance Imaging) tools, which obviates the dependence on a single image file format. In addition, the use of RDBMS and Java also facilitates the portability of the system.

1 Introduction

The accurate maintenance and fast search of images continues to increase in complexity due to proliferation of content, specially over the Internet. A number of users continue to create innumerable digitized images 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 can find them accurately and efficiently with ease.

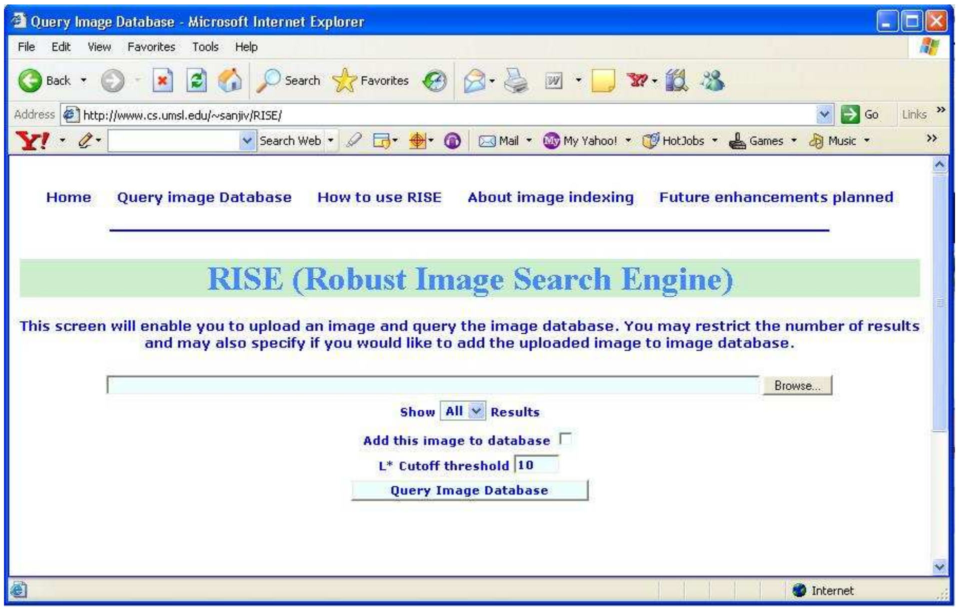
In current real world image databases, the prevalent retrieval techniques involve human supplied text annotations to describe image semantics. This method has been classified as *text-based image retrieval*, or TBIR. TBIR uses a description of the scene in natural language and attempts to match the *tokens* from this description against the tokens or tags associated with each image in the database. TBIR is fraught with several serious drawbacks. First, the semantic complexity of an image leads to different descriptions, resulting in content and/or language mismatch [4, 17]. 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 hence, is impractical for large databases.

The manual annotation adds a semantic complexity to the process that 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 [5]. The problem is exacerbated by the fact that the *tags* considered important by one person in an image may not be relevant to another. This assertion is easily confirmed by making a query on popular web sites for image retrieval such as flickr and google. 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 image analysis 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. Hence, most of the attempts have been to build semantic models to describe images at low levels [3].

The current attempts to solve this problem have led to the technique described as content-based image retrieval (CBIR) research. A CBIR system retrieves images based on their contents. There are two query methods in CBIR: query-by-example and query-by-memory [19]. In query-by-example, a user selects an example image as the query. In query-by-memory, a user selects image features, such as color, texture, shape, and spatial attributes, from his/her memory to define a query.

In recent years, many CBIR techniques have been developed to address the growing need for efficient image management. 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 that closely match the query. Some systems, such as QBIC [6], WALRUS [11], and the system presented here – RISE, may use more than one method to describe the query.

In this article, we present the development and implementation of RISE, Robust Image Search Engine, to organize, index, and search images based on the contents of the images. RISE develops a color signature of an image using concepts similar to ones used to build JPEG coefficients for the JPEG image format. RISE provides user interaction using a GUI (Graphical User Interface), as shown in Figure 1. The GUI may be used to upload an image to perform a query as well as to add an image to the database.


Query Image Database - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://www.cs.umsl.edu/~sanjiv/RISE/ Go Links
Search Web Mail My Yahoo! HotJobs Games Music
Home Query image Database How to use RISE About image indexing Future enhancements planned
Browse...
Query Image Database
Internet

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 ☐

L* Cutoff threshold 10

Figure 1: The Graphical User Interface for RISE

In contrast to many JPEG-based techniques, RISE may be applied to many image formats. The technique involves dividing a given image into 8×8 blocks and then computing the color signature of each 8×8 block as the average of its color components. Conceptually, this is equivalent to the DC value in the DCT-transformed image coefficients in JPEG. Next, RISE forms a quad tree structure [15] with the color signatures at different resolution levels. It should be noted that an image may be abstracted in terms of probability distribution function in different colors. Hence, RISE derives a unique color signature for an image, in terms of the probability distribution of colors at different resolution levels structured in the form of a quad tree.

RISE computes the color signature in perceptual color space instead of the more commonly used RGB color space. We discuss the relevant background for perceptual color space in the next section. In Section 2, we discuss the color signature in detail. Then we present the basics of computing the color signature and the relevant steps from JPEG compression which provided the motivation to compute color component average. In Section 5, we show the development of the quad tree corresponding to the color component average as well as the information to be retained at each node of the quad tree. In Section 6, we discuss the storage and retrieval of quad tree information using an RDBMS. In Section 7, we illustrate the query process. Implementation details are described in Section 8. Finally, we conclude by presenting a summary and future plans for the system in Section 9.

2 Perception of Color

Color is the perceptual sensation of light in the visible region of the electromagnetic spectrum, incident upon the retina. The visible region has wavelengths in the region of 400nm to 700nm. Color can be characterized by the total amount of energy (or radiance) that flows from a light source and is expressed as a spectral power distribution (SPD), often in 31 components each representing a 10nm band. The color energy is sensed inside human eye by cones in the retina that respond to the red, green, and blue components with different weights to produce color sensation for the brain [13].

Color spaces are mathematical models used to specify, create, and visualize color. Different models represent the same color in different ways. The most commonly used color spaces are RGB, CMY, CMYK, HSI, and $L^*a^*b^*$. 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

The RGB color space is based on the system used in human eye where the cones perceive the three colors as primary and other colors as a blend of the three primaries. However, a major difference is that the human eye assigns different weights to those primaries whereas the RGB scheme assigns them the same weight.

The RGB color space is geometrically represented as a 3-dimensional cube. The coordinates of each point inside the cube represent the value of red, green, and blue components, respectively. The linear nature of colors in RGB color space is suitable for color display, but not so for color scene segmentation and analysis because of the high correlation between the primaries. High correlation implies that if the intensity changes, all the three components may change as well.

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

The $L^*a^*b^*$ color model was developed by CIE (Commission Internationale d'Eclairage) to describe the colors as perceived by the human eye. It is an absolute color model that separates the luminance from colors. This model represents colors in three dimensional space as a cylinder. In this cylinder, the linear axis corresponds to L^* and represents the luminance channel L. The contrast between red and green, which is represented as channel a, is represented on the axis a^*. The contrast between blue and yellow, represented as channel b, represents the axis b^* as shown in Figure 2.

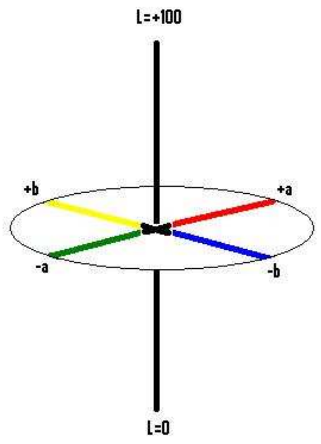


Figure 2: Lab color space in 3D co-ordinates

$L^*a^*b^*$ is designed to be a perceptually linear color space. That is, the components of the color space change in a linear manner with a change in color. It is particularly suited to applications that need to change contrast easily while retaining the basic color information. It achieves this by separating the luminance channel from chrominance making it possible to perform image processing operations such as sharpening without accidentally adding color casts.

In $L^*a^*b^*$, the range of luminance channel lies between 0 and 100. If a^* and b^* values are zeros, $L^* = 0$ yields black while $L^* = 100$ yields white. The range of channel a^* lies between -86 to +86, where 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 using 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, we present the conversion of a pixel from RGB to $L^*a^*b^*$ color space.

2.3 Conversion from RGB to $L^*a^*b^*$ Color Space

The conversion from RGB to $L^*a^*b^*$ color space is based on Rec. 709 standard [8]. 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

$$\begin{aligned} L^* &= 116(f(Y/Y_n) - 16) \\ a^* &= 500(f(X/X_n) - f(Y/Y_n)) \\ b^* &= 200(f(Y/Y_n) - f(Z/Z_n)) \end{aligned}$$

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.

$$\begin{aligned} X_n &= 242.36628 \\ Y_n &= 255.0 \\ Z_n &= 277.63228 \end{aligned}$$

The function $f()$ is defined as

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

We divide $f(t)$ into two domains to prevent an infinite slope at $t = 0$. We assume $f(t)$ to be linear when $t \leq t_0$, and match the $t^{1/3}$ part of the function at t_0 in both value and slope. In other words:

$$\begin{aligned} t_0^{1/3} &= at_0 + b \\ 1/(3t_0^{2/3}) &= a \end{aligned}$$

The value of b is chosen to be $\frac{16}{116}$. The above two equations can be solved for a and t_0:

$$\begin{aligned} a &= 1/(3\delta^2) &= 7.787037 \\ t_0 &= (\delta^3) &= 0.008856 \end{aligned}$$

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

3 Color Signature Computation Methods

Visual perception is a powerful source of information about the world around us. 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. Most of us 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 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, just the distribution of color does not give a true representation of an image. Two images could be different (see Figure 3), and yet have the same color distribution. Therefore, it is essential to take spatial organization of the image into account as well. Hence, a colors signature must consider both the color information as well as the location at which the color is found in the image. In general, color signature computation methods may be divided into three broad categories – histogram-based, color layout-based and region-based. In the remainder of this section, we present the description of each of these three categories.

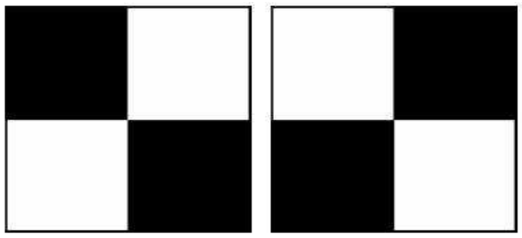


Figure 3: Two images with opposite colors, yet identical histograms average intensities

3.1 Histogram-Based systems

A histogram of an image conveys the relative frequency or probability distribution of colors in an image. Inside an image, each pixel 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 3 are identical, yet the images are obviously different. Furthermore, the color quantization step leads to additional sources of errors [9].

The histogram-based systems may use far fewer bins than all possible colors for reasons of efficiency. Thus, many colors may get quantized into a single 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 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 that are close to each other [9].

3.2 Color Layout-Based Systems

The 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 the signature of the image. Some systems, such as WBIIS [21] 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 [21].

The 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 be close to that of a bluish house with a green roof. These two images would not appear close in a color layout scheme. 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, WALRUS 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. WALRUS performs clustering on squares with similar signatures in order to reduce the number of signatures to be searched during a query. While this scheme is tolerant of image translation and scaling, its computational complexity is dramatically increased. WALRUS is essentially a color layout scheme; however, it incorporates some attributes of region-based systems.

3.3 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 [18], Blobworld [2], and SIMPLIcity [20].

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 arrangement of regions. It automatically extracts regions using a color set back-projection method [18]. It then stores characteristics such as color, shape, texture, area, and location, for each region. It performs a separate search 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, it 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 [20]. It varies signature construction and distance formulations according to the semantic class. It uses the k-means algorithm and Haar wavelet to segment the image into regions [20].

RISE uses the distribution of colors in different areas of the images to compute the signature at multiple resolutions. It is based on computing the average color from the DC component of the DCT transform used in JPEG compression. In the next section, we describe the JPEG scheme to compress an image, and its use in RISE.

4 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 storage and transmission [10,12,16]. 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 discrete cosine transform (DCT).

To apply DCT, each pixel in the image is level shifted by 128 by subtracting 128 from its 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 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 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 [12]. The coefficients in the quantized block are rearranged using a zigzag ordering to create a vector. The zigzag pattern approximately orders the basis functions on spatial frequencies [12].

The vector resulting from the zigzag ordering contains 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 RISE can work with different types of image formats, it does not use JPEG coefficients which would have required an extra step to convert a given image to JPEG. Instead, RISE uses the concept of computing JPEG coefficients. As explained above, a given image is abstracted into a set of 8×8 pixel blocks. Then, RISE computes an average of 8×8 pixels for each block.

RISE converts each pixel in the block to $L^*a^*b^*$ and then, computes the average color for the ith block as

$$\begin{bmatrix} L_i^* \\ a_i^* \\ b_i^* \end{bmatrix} = \begin{bmatrix} \frac{1}{64}\sum_{j=0}^{63} L_j^* \\ \frac{1}{64}\sum_{j=0}^{63} a_j^* \\ \frac{1}{64}\sum_{j=0}^{63} b_j^* \end{bmatrix}$$

5 Quad Tree Representation of Average Color Components

A quad tree is a data structure in which each internal node has up to four children. It can be used efficiently to represent an image at different levels of detail (LOD). There are different types of quad trees. A *point region* (PR) quad tree is one 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, we will refer to the PR quad tree simply as quad tree.

In the previous section, we have shown how RISE divides the entire image into 8×8 blocks and computes the average color these blocks. This effectively implies that the entire image is composed of 8×8 pixel blocks. Instead of individual pixels, RISE considers the average value of a pixel block to be the smallest addressable unit of the image. Here we 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 4). Moreover, all the leaf nodes in the quad tree are at the same level in the tree, and represent an 8×8 pixel block in the original uncompressed image. To develop the quad tree, RISE scales a given image into a 512×512 pixel image. This is performed to make the image signature uniform across all the images. Thus, the length of each side of the new square image is a power of 2. A quad tree, as in Figure 4 can be derived by recursively dividing each side by 2 as shown in Figure 5.

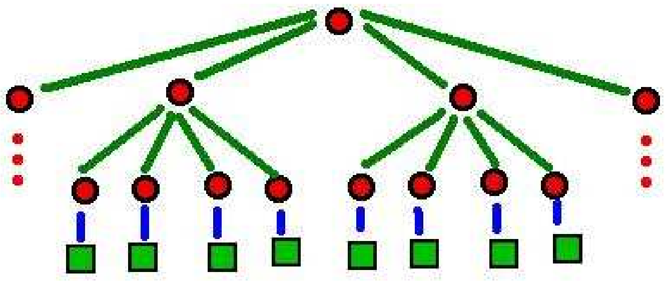


Figure 4: A quad tree in RISE

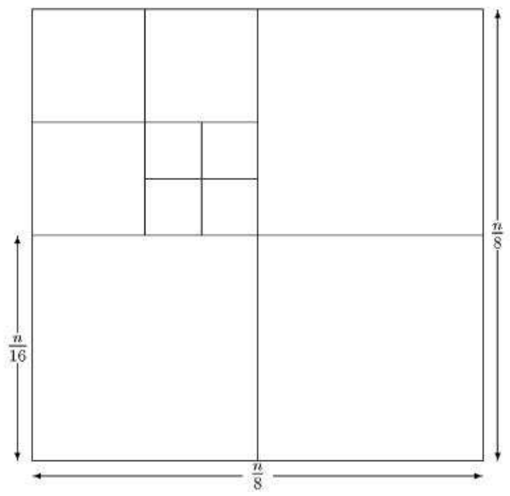


Figure 5: The quad tree of an image.

5.1 Leaf Nodes of the Quad Tree

Each leaf node in the quad tree corresponds to a tristimulus color vector in $L^*a^*b^*$ color space to describe the average of color components of the block that has been computed from the 8×8 pixel block in the original image. The definition of a leaf node is presented in Figure 6.

```
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 6: Definition of a node in the quad tree.

In addition to the average of the color components, the node in the quad tree also retains the name of the image for identification purpose, as well as the location of the block in the image using pixel coordinates. The location is defined to be the lower left pixel position of the block.

5.2 Internal Nodes of the Quad Tree

The internal nodes in a quad tree contain information extracted from aggregates of data in the 8×8 pixel blocks in the subtree below that node. They are used to make similarity comparisons between images at a higher level than the individual blocks. Just like in leaf nodes, RISE retains the name of the image for identification purposes, in addition to the location of the aggregate block. Lastly, the internal nodes also contain the links to their four children, identified 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 6.

It may be noted that a leaf node uses the same structure as an internal node by assigning a null value to the four children. The quad tree is built in a bottom-up manner by aggregating lower level nodes. We start at the leaf nodes that correspond to the 8×8 pixel block. Then, a set of four neighboring 8×8 blocks are aggregated into a second level node. The process continues till we are left with just the root node of the quad tree, corresponding to the average tristimulus values for the entire image.

If we look at it in a top-down manner, the set of coefficients is divided into four sets of $\frac{n}{16} \times \frac{n}{16}$ coefficients each. RISE continues to subdivide each set into four subsets recursively until it gets a set containing only one 8×8 average color component. At each node in the tree, it calculates the information in the node (Figure 6).

The number of nodes in the tree at each levels can be identified by the size of the image. RISE saves each node in a relational table with attributes LEVEL_nn depending upon the node's level in the quadtree, where nn indicates the level starting at root at level 0. It is easy to see that all the image 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 section, we describe the organization and storage of the quad tree structure in an RDBMS.

6 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 article, we 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 enables us to not worry about efficient use of expensive hardware devices and lets us concentrate more on logic of our algorithm and presentation of our data. This also 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. Finally, we have used SQL (Structured Query Language) to access the database. This is standardized across databases and so, RISE is not limited in the choice of database.

The use of an RDBMS is not cost free. We incur overhead 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. For a large database, this trade-off may seem negligible when we consider the benefits of using RDBMS.

Another problem in using a database is that it takes time to send a request to RDBMS and receive the results. This may become significant if a large number of records are fetched from database. However, this problem may be resolved by devising techniques to minimize the number of accesses to database, by bringing larger chunks of data in each access and thus reducing the impact of access to the database.

In the following subsection, we present a description of database characteristics used in RISE.

6.1 Database Data types

Each attribute of an object manipulated by a database has a data type associated with it. The data types have been standardized by ANSI (American National Standards Institute) [1] and among others, include `CHARACTER`, `CHARACTER VARYING` and `INTEGER`. Data type of a value binds certain properties to the value. These properties cause the database to treat a data type value differently from another. For example, two `INTEGER` values may be added but two `CHARACTER` values may not.

6.2 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 are manipulated by database operations.

A table or relation is the basic unit of data storage in a database. The tables of a database hold all of the user-accessible data. Rows of a table correspond to records in a database, while the columns represent the attributes in the relation. Every table is defined with a table name and set of attributes. Figure 7 displays an example of a table structure used by RISE to store the quad tree.

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 7: A relational table definition in RISE

In our example, each attribute in the relational table definition is described as a record. Let us consider the first record (row) in Figure 7. The first attribute in the table defined by Figure 7 is IMAGE_NAME, the second attribute is LEVEL_0_L, and so on. The second column in Figure 7 describes the properties of this attribute. For example, the properties of the attribute IMAGE_NAME are its data type (`CHARACTER VARYING`), and its width (100).

Once the relation is created, we can insert valid rows of data into it. The records in the relation can then be queried, deleted, or updated. Figure 8 displays data in rows. For the sake of simplicity, we have displayed only a few attribute values and restricted the display to a few rows.

A relational database provides faster access to data by using an optional object called an *index*. An index contains an entry for each value that appears in the indexed column(s) of the relation and provides direct, fast access to rows. A common way to implement indexing is through B-trees. In the next subsection, we describe the storage of image signature in an

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 8: Table rows

RDBMS.

6.3 Storing Image Signatures in the Database

RISE compares images in the $L^*a^*b^*$ color space and uses the quad tree structure to improve the performance. At the root of the quad tree, represented as level 0, there is only one node and L^*, a^*, and b^* components 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 RISE stores 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 attribute is used frequently, we have indexed the three corresponding columns for faster retrieval. RISE uses B-Tree indexes for indexing L^*, a^*, and b^* values at the root level.

At levels under the root, the quad tree has multiple children. So, we have the option of adding multiple columns or creating separate rows for each node. The first option is impractical as it creates a table with enormous number of columns and will be difficult to maintain. Adding separate rows works but for RISE, it resulted in longer response time. Therefore, we have separated the L^*, a^*, and b^* components at each node and stored them as arrays. For example, RISE has 4 nodes at level 1. So, we have created three arrays of length 4 (ranging from 0 to 3), one array for each of the L^*, a^*, and b^* components, using the collating sequence for quad tree nodes described in Section 5. The same technique is extended for lower levels using larger arrays.

As in any tree structure, the number of children progressively increases through each level of the quad tree. RISE uses a quad tree with six levels. It is easy to see that the quad tree has one node at the top level but successive levels have 4^1, 4^2, 4^3, 4^4, 4^5, and 4^6 nodes.

6.4 Querying the Database

RISE uses SQL to select data from the RDBMS. The following statement demonstrates the selection of data from database.

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

This query selects images from database that have their L^* value within the range of $\pm 15\%$ of the L^* value of the input image. As evident from this statement, RISE selects complete nodes in a single query statement. This avoids making repeated expensive accesses to the database. This also improves the response time tremendously.

6.5 Saving Data in Database

RISE inserts data into the database by specifying the attribute values within a `VALUES` clause, as illustrated below.

```
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(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
```

In this statement, RISE inserts a complete quad tree with a single statement. The `VALUES` clause allows for population of serialized byte arrays.

7 The Query Process

During the query process, a user uploads an image to RISE as a query image. In response, RISE returns a set of images that it determines as having similar content as the given image. RISE processes the query image to create its quad tree signature that is then matched against similar signature of images already stored in the database. The content similarity is measured by computing Euclidean distance between the query image and the images 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 image is processed to develop the quad tree of its average color component of 8×8 blocks at different resolutions as described in Section 5. Further steps in the query rely completely on the quad tree.

RISE starts the process by computing the distance between the top most level of the image (root of each quad tree) in database. RISE uses a predefined threshold parameter to select only those images that have a distance less than the threshold. This process is repeated at each level and images with larger distance 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 and C are the number of row blocks and column blocks at that level in the quad tree and $p_i(x,y,c)$ is the average color c in the block at position (x,y) in image i.

Since the number of nodes at upper levels 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 the 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 database, the Euclidean distance evaluates to zero. At any time, the images with a distance 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 was depicted in Figure 3. These images need to be ranked using the information in the nodes at lower levels of the quad tree.

While comparing the nodes at lower levels of the tree, we 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 larger than the threshold, the image is removed from further consideration. RISE 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 query image. The ranking in the list allows RISE to present the resulting images in decreasing order of relevance with respect to the query. In addition, RISE can also limit the number of images that are to be presented to the user.

In the next subsection, we comment on the performance of RISE, in terms of standard retrieval measures of precision and recall.

7.1 Evaluation of Query Results

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 [14]. 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 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 a number of 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 due to this subjective nature of performance evaluation [7]. Therefore, we use simple representative queries to evalute the performance of our system.

We have formulated three test cases to demonstrate the performance of RISE. In the first case, the query image exists in the image database and many similar images also exist. Here, RISE should find the exact match and should also return those images that are similar. In the second case, the query image exists in the image database and so does a slight variation of that image. In this case, RISE should return the exact match and also should pick the slight variation from the database. Finally, in the third case, the query image does not exist in the database. Here we need to study the types of results being returned from the database carefully to determine the performance of RISE.

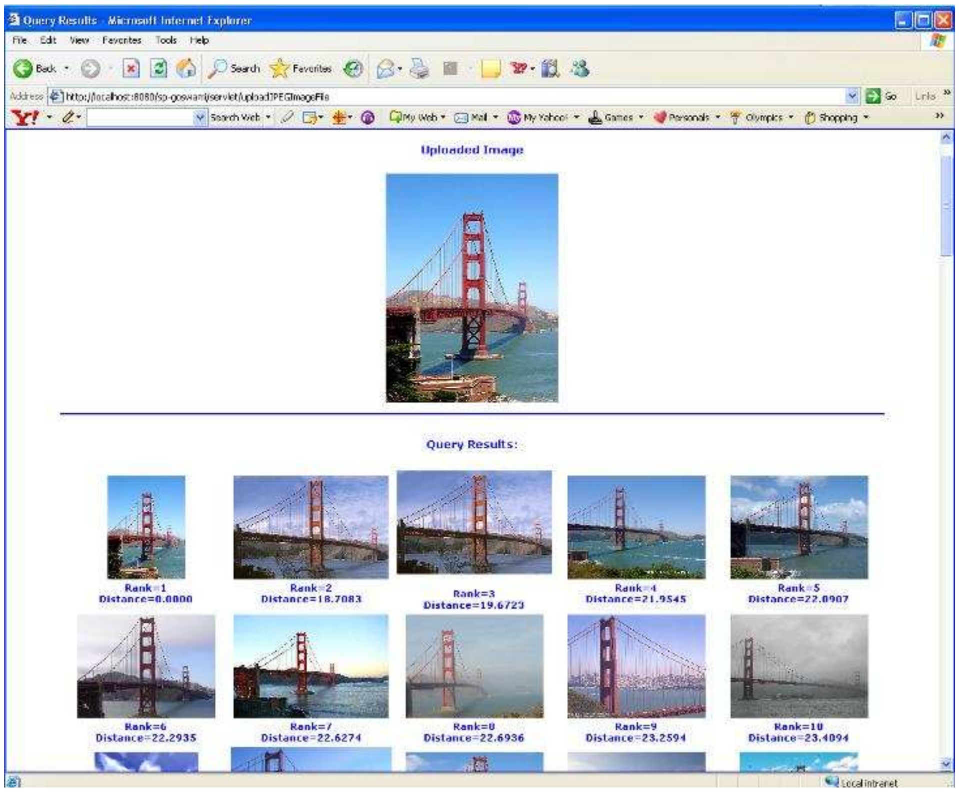


Figure 9: Query results: Case 1

The result of first query is displayed in Figure 9. Here, the query image is the Golden Gate Bridge. It is easy to note that the results constitute images of the Golden Gate Bridge. However, the distance and hence, the rank changes when angle and color of sky is different.

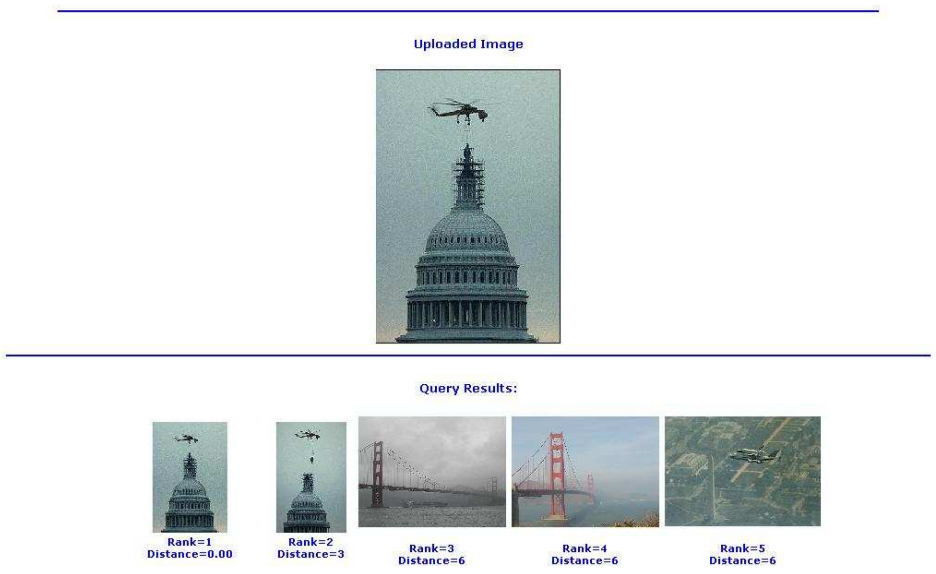


Figure 10: Query results: Case 2

The result from the second query is displayed in Figure 10. Here, the 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 several unrelated images in the result.

The result of the final query (the image not in the database) is shown in Figure 11. The query 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 their relatively high ranks.

8 Implementation of RISE

RISE is based on using the average of color components of 8×8 blocks in $L^*a^*b^*$ color space. We have used the idea of DC components of DCT as used to compute JPEG coefficients. The DC component is essentially the average of color components of 8×8 blocks. The 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. RISE computes the statistics at each node in the quad tree, and includes 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.

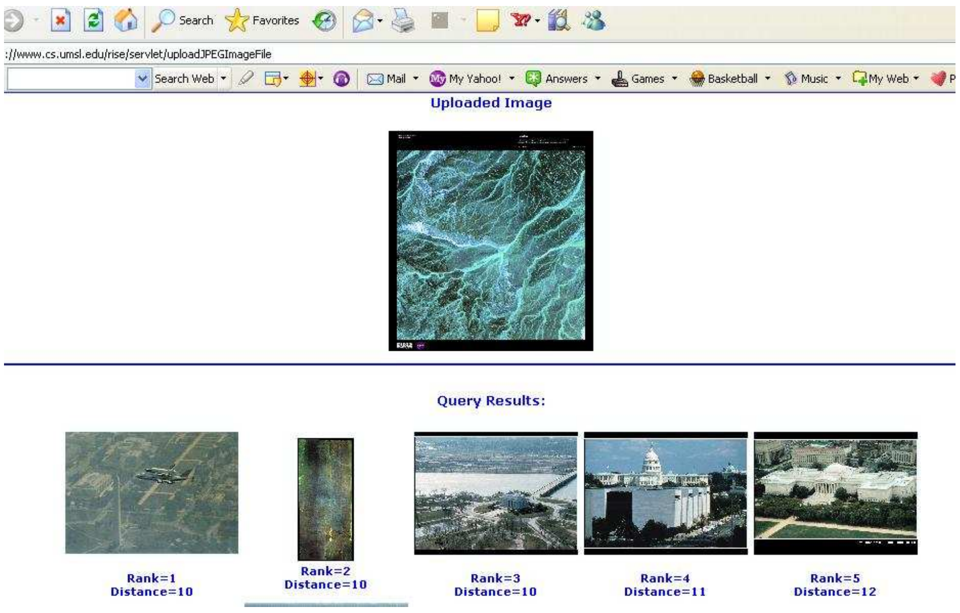


Figure 11: Query results: Case 3

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 processed into the quad tree structure and statistics from different nodes are compared against the statistics from the quads of same size in the indexed 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, its use of the quad tree structure allows extensive early pruning. Second, it uses a relational database for storing information which makes data retrieval faster. Finally, RISE can process images in any image format. We just need to convert the image to $L^*a^*b^*$, divide it into 8×8 blocks, and take the average of the 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, RISE uses regional properties of the indexed images to perform the search. 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 back-end objects used for searching is implemented as a Java servlet on a Sun SPARC system and hosted in a Tomcat web server. We have used Oracle as the relational database for storing quad trees. The initial GUI input screen may be seen in Figure 1. We 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.

RISE computes a quad tree for each image and stores the index information in the database. This part runs as a command line interface. The user may upload an image using any standard web browser and query the image database. RISE displays the output as a ranked list of images from the database. Currently, the output in the browser displays a set of retrieved images, their distance from the query image, and their rank.

9 Summary

In this article, we have presented the design and implementation of RISE, an image database indexing system for efficient storage and retrieval of images in response to a query expressed as an example image. RISE is an efficient content-based image retrieval system. The efficiency stems from indexing images using JPEG coefficient-like average of three color components in perceptual color space. The techniques can be applied to any image format to compute image signature.

RISE uses a quad tree to structure the signature of an image for query and storage. The 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 the raster information to $L^*a^*b^*$ color space and computes the average of color components. The average 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.

RISE processes a query image to produce a quad tree, and compares the roots of the trees in the index with the query using the Euclidean distance. For distances within a given tolerance, the second level of the corresponding trees are compared. This process is repeated down to the leaf nodes. The selected images are ordered and returned to the user. The system has been implemented in Java with JAI interface on a Sun workstation using an Oracle relational database to save quad trees.

References

[1] American National Standards Institute. 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 International Conference on Computer Vision*, pages 675–682, 1998.

[3] A. Bosch, X. Munoz, and R. Marti. A review: Which is the best way to organize/classify images by content. *Image and Vision Computing*, 2006. doi:10.1016/j.imavis.2006.07.015.

[4] 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.

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

[13] 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.

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

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

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

[17] 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.

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

[19] E. L. van den Broek, P. M. Kisters, and L. G. Vuurpijl. Design guidelines for a content-based image retrieval color-selection interface. In *Proceedings of the Conference on Dutch Directions in HCI*, Amsterdam, Holland, 2004.

[20] J. Z. Wang, J. Li, and G. Wiederhold. SIMPLIcity: Semantics-sensitive integrated matching for picture libraries. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23, 2001.

[21] 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.