RISE: A Robust Image Search Engine

Debangshu Goswami & Sanjiv K. Bhatia
Department of Mathematics & Computer Science
University of Missouri – St. Louis
St. Louis, MO 63121
Email: debangshu@yahoo.com, sanjiv@cs.umsl.edu

***Abstract*— Image database indexing is used to improve the efficiency of image retrieval in response to a query expressed as an example image. The query image is processed to extract information that is matched against similar information from images stored in a database to retrieve a set of matched images. The matching process is achieved by a search engine. This paper advances Robust Image Search Engine (RISE). RISE is based on a technique that facilitates content similarity based retrieval of images using an index built on color components of selected blocks of images. It uses a measure of distance between the query and the images in the database. RISE builds on the JPEG indexing algorithm, extending it to formats other than JPEG and into color spaces, in addition to using a relational database. We have implemented RISE as a browser-based search engine, deployed as a java servlet.**

I. Introduction

The search of images in an accurate and fast manner, along with a high signal-to-noise ratio, has become even more challenging in this era of internet when millions of people are looking for various images at any time. Certainly, the need for efficient automated retrieval systems has dramatically increased in recent years due to the world wide web and multimedia technology. In current real world image databases, the prevalent retrieval techniques involve human supplied text annotations to describe image semantics.

The text annotation method to describe images has several drawbacks. The semantic complexity of an image can lead to different descriptions, resulting in content and/or language mismatches [1]. A content mismatch occurs when a seemingly insignificant object or characteristic may be omitted in annotation and, later, a user searches for that omitted information. Language mismatches may occur when different words are used to describe the same object. Another drawback is that 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. The manual annotation method is severely limited as it is time intensive and thus impractical for large databases.

A human can easily discern objects in an image and may be able to assign semantic tokens to describe those objects. However, an object can be described by different tokens due to the richness of language and two different persons may use different words to describe the same object. Adding the context and background of each observer may increase the number of possible descriptions which makes it extremely hard to create tokens to satisfy each possible query. At the present time, the technology has not advanced to the point for a computer to discern an image, or to unambiguously describe its contents. This is due to the limitations of current image processing and object recognition techniques. However, we do have mathematical models that allow us to segment images and capture the distribution of colors and shades in an image in an automatic manner.

In this paper, we present the Robust Image Search Engine, or RISE. RISE is based on a technique to develop an index for an image database using a subset of JPEG coefficients in a compressed image [2]. It creates an average of color components of each 8×8 pixel blocks in a JPEG image, which is the same as the DC-coefficient of each block after applying discrete cosine transform (DCT). The DC coefficient for each color component is used to develop an index. The structure of the index is based on partitioning these color averages into a quad tree structure [3].

In the next section, we present the problem background in detail and review the literature. In Section III, we present the basics of JPEG compression which provides the concepts behind computing color component average. In Section IV, we 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 Section V, we illustrate the query process. Section VI describes the implementation status of RISE. We conclude by presenting a summary and future plans for RISE in Section VII.

II. Background

In recent years, many content-based image retrieval (CBIR) methods have been developed to address the growing need for efficient image management systems. Generally, CBIR systems define methods for extracting characteristics from images, known as *signatures*, and employ rules to compare images using these signatures. In most of these systems, the user is expected to supply 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. Most CBIR systems can be broadly categorized into one of three areas: histogram-based, color layout-based, or region-based. Some systems, such as QBIC [4], Walrus [5], and our system – RISE, have characteristics of more than one category. In the remainder of this section, we present a description of each of the three categories and conclude with an overview of the algorithms used in RISE.

A. Histogram-Based Systems

A histogram of an image conveys the relative quantities of colors in an image. Each pixel in the image is represented as a weighted amount of three colors: red, green, and blue. Each of these three colors is quantized into m divisions, yielding a total of m^3 composite colors, or bins of histograms. The total numbers of pixels that correspond to each bin are counted. The signature of the image is a vector containing these counts and may then be 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. First, these systems disregard the shape, texture, and object location in an image, leading to a high rate of return of semantically unrelated images. For instance, the histograms of the two images in Figure 1 are identical, yet the images

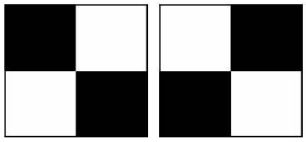

Fig. 1. Two images with opposite colors but identical average histogram intensities

are obviously different. Also, the color quantization leads to additional source of error [6].

The histogram-based systems may improve efficiency by using fewer bins than the number of colors. Here, 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 extreme colors 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 [7]. However, histogram quantization may incorrectly use a uniform divisor for all three color channels. Finally, a query image may contain colors that are similar to the colors of a particular image in the index, but may result in a large distance if they are not close enough to fall into the same bins [6].

B. Color Layout-Based Systems

Color layout-based systems extract signatures from images that are similar to low resolution copies of the images. An image is divided into a number of small blocks and the average color of each block is stored. Some systems, such as Walrus [5], utilize significant wavelet coefficients instead of average values in order to capture sharp color variations within a block.

The traditional color layout systems are limited because of their intolerance of object translation and scaling. The location of an object in an image 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 distance to a green house with a blue roof, while a color layout scheme may discard this image. The Walrus system uses a wavelet-based color layout method to overcome the translation and scaling limitation [5]. For each image in the index, a variable number of signatures (typically thousands) are computed and clustered. Each signature is computed on a square area of the image, with the size and location of each square varying according to prescribed parameters. The squares with similar signature are clustered in order to speed up the search during a query. While this system is tolerant of image translation and scaling, the computational complexity is dramatically increased.

C. Region-Based Systems

Region-based systems use local properties of regions as opposed to the use of global properties of the entire image. A fundamental stumbling block for these systems is that objects may cross across multiple regions, each of which inadequately identifies the object. Examples of region-based systems include QBIC [4], SaFe [1], Blobworld [8], and SIMPLIcity [9].

QBIC uses both local and global properties and incorporates both region-based and histogram properties. It identifies objects in images using semiautomatic outlining tools [4]. SaFe is a complex system that automatically extracts regions and allows queries based on specified arrangement of regions. It uses a color-set back-projection method to extract regions [1]. For each region, it stores characteristics such as color, shape, texture, and location. Then, it performs a separate search for each region in the query image. Blobworld is a region-based system that defines regions, or blobs, within an image using the Expectation-Maximization algorithm on vectors containing color and texture information for each pixel [8]. For each blob, it computes and stores 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. 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 [9].

D. RISE

Despite their complexity, all the above systems miss relevant images in the database and may return a number of irrelevant images. An ideal system is one that provides high value for both *precision* and *recall* [10]. *Precision* is the ratio of the number of correct images retrieved to the total 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 has an adverse effect on precision. 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.

RISE is based on using the average of color components of 8×8 blocks. We use the idea of DC component of the DCT-transformed blocks in JPEG coefficients which are essentially the average color components in blocks. The main advantage

of using average is that it can be applied to any image format. It is also more practical since we do not use the AC-components of the DCT-transformed image.

Each image in RISE is represented by a quad tree structure with leaf nodes that contain average color information. We compute statistics for 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 in separate relations. Just like the images in the database, a query image is partitioned into the quad-tree structure and statistics in 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 are 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.

The efficiency in RISE stems from three factors. First, we use elements as the leaves of a quad tree structure to allow extensive early pruning. Second, we use a relational database for storing information for faster data retrieval. Finally, RISE can handle any image format, and not limited to just JPEG. However, we need to convert the image to raster and then divide it into 8×8 blocks and take average of color components of each block. RISE uses an extremely efficient color layout method with region-based characteristics. The signature of the 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, and has good tolerance for scaling.

III. Average of Color Components

RISE uses the ideas from JPEG, and hence, we briefly describe the JPEG standard in this section. The baseline JPEG compression standard is based on the discrete cosine transform (DCT). DCT is applied by level shifting each pixel in the image by subtracting 128 from its components. 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 by weighting functions optimized for the human eye. The resulting coefficients are encoded using a Huffman variable word length algorithm to remove redundancies [11].

The application of DCT separates the high and low frequency information in the block. It results in the average value (or DC component) in location (0,0) of the 8×8 block while the other locations of this block contain AC terms, or higher frequency components of the block.

Since we wanted to include different types of image formats, we could not use JPEG coefficients which would have required an extra step to convert a given image to JPEG. In RISE, we use the JPEG coefficients only for the JPEG images. For all other formats, we use the average color for the 8×8 blocks in the image. The average color is used to build the quad tree data structure as described in the next section.

IV. Quad Tree Structure Representation

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

In the previous section, we showed the computation of the average color for each 8×8 block in an image. This effectively implies that the entire image is considered to be constructed of 8×8 pixel blocks. We consider the average value of 8×8 block to be the smallest addressable unit of the image, instead of each pixel. Here, we describe the construction of a quad tree structure using the information from each 8×8 block. The quad tree in our system is a full tree such that each node contains exactly four children or none (Figure 2). Moreover, all

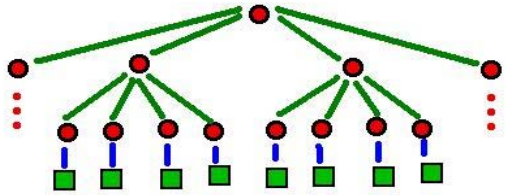

Fig. 2. Quad tree

the leaf nodes are at the same level in the tree, and represent the average value of the corresponding 8×8 block in the original uncompressed image. To develop the quad tree, first we scale the given image to 512×512 pixel image. Thus, the length of each side of the new square image is a power of 2. A quad tree as in Figure 2 can be derived from the square block by recursively dividing each side by 2.

Each leaf node of the quad tree corresponds to a vector of average of color components of the block that has been computed from an 8×8 pixel block in the original image. In addition to the average of the color components, the node also retains the name of the image for identification purpose.

The non-leaf 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, we retain the name of the image for identification purpose. Lastly, the non-leaf nodes also contain links to their four children, identified as upper-left child, upper-right child, lower-left child, and lower-right child.

It may be noted that a leaf node uses the same structure as an internal node, assigning a null value to each of its

four children. We start with the set of 8×8 average color components forming a $\frac{n}{8} \times \frac{n}{8}$ square corresponding to an $n \times n$ image at the root. The set of coefficients is divided into four sets of $\frac{n}{16} \times \frac{n}{16}$ coefficients each. We continue to subdivide each set into four subsets recursively until we get a set containing only one 8×8 color component. At each node in the tree, we calculate the information in the node.

The nodes in the tree are at distinct levels corresponding to the size of the image. We save the nodes depending on their level in the tree in a table in a database named as `IMAGE_DB_LEVEL_`nn where nn indicates the size of the segment at that level. 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.

V. The Query Process

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 Section IV. Further steps in the query rely completely on the quad tree.

First, we compute distance between the topmost level of the query image from each image in the database. We use 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 distances are pruned. The difference between two images is given by a summation of Euclidean distance D between corresponding pixels and is computed as follows:

$$D = \frac{1}{mn} \sum_{x=0}^{m} \sum_{y=0}^{n} \sum_{c=R,G,B} \sqrt{(p_1(x,y,c) - p_2(x,y,c))^2}$$

where m and n are the number of columns and rows, respectively, and R, G, B indicate the color values for red, green, and blue channels, respectively.

Since there are fewer nodes at the upper level in the tree, a full scan of the table is not expensive at upper levels. After initial scan, we narrow down our choice and only compute distance for a few 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 evaluates to zero. At any time, the images that result in a distance greater than a pre-specified threshold from the query image, 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 Euclidean distance zero (see Figure 1). These images need to be ranked using the information in the lower nodes in the quad tree.

While comparing lower nodes, 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 table in relational database 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 to 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) is added to the previous evaluation. At any stage, if the summation of distances during the level-order traversal comparison is too large, the image is removed from further consideration. We 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, sorted on the basis of Euclidean distance of images from query. The ranking in the list allows RISE to present the resulting images in decreasing order of relevance with respect to the query. In addition, we can also limit the number of images that are presented to the user.

VI. Implementation Status

The image database system described in this paper has been implemented as a java servlet on a Sun SPARC system and hosted on a Tomcat web server. We have used mySql as the relational database for storing quad trees. The initial input screen is presented in Figure 3.

A quad tree is computed for each image and index information is stored in the database. The user may upload an image through any standard browser and query the image database. The output is displayed as a ranked list of images from the database. Currently, the output contains a list of images, their distance from the input image, and rank. The results from a query are displayed in Figure 4.

VII. Summary and Future Work

In this paper, we have presented RISE – an image database indexing system for efficient storage and retrieval of images in response to a query expressed as an image. RISE can be classified as an efficient CBIR system. The efficiency stems from indexing images using JPEG coefficient-based average of color components which can be applied to any given image format and using a quad tree for image signature. RISE also allows extensive early pruning during query so that images that are not promising for a query, are eliminated from consideration at an early stage.

RISE accepts most of the popular image formats by extracting the average of color components from the RGB raster data to build a quad tree. This tree is the image *signature* that is stored in the index. A query image is processed to produce a quad tree. The query tree and stored trees are compared in a level-order fashion, using Euclidean distance, pruning the images from further comparison if the distance exceeds a pre-specified threshold. The system has been implemented in Java with Java Advanced Imaging tool on a Sun workstation and mySql relational database used to save quad trees.

At present, we are working on changing the comparison of images to the perceptual L*a*b* color space instead of the RGB color space [7]. We'll also like to improve the efficiency of the system through faster database access, using proper indexing and partitioning of the database tables.

The efficiency of RISE results from indexing images and creating a tree that allows extensive early pruning. These concepts could be applied to other CBIR projects.

REFERENCES

[1] J. R. Smith and S.-F. Chang, "Integrated spatial and feature image query," *International Journal of Multimedia Systems*, vol. 7, no. 2, pp. 129–140, March 1999.

[2] S. Climer and S. K. Bhatia, "Image database indexing using JPEG coefficients," *Pattern Recognition*, vol. 35, no. 11, pp. 2479–2488, November 2002.

[3] H. Samet, "The quadtree and related hierarchical data structures," *ACM Computing Surveys*, vol. 16, no. 2, pp. 187–260, June 1984.

[4] M. Flickner, et. al., "Query by image and video content: The QBIC system," *IEEE Computer*, vol. 28, no. 9, pp. 23–32, September 1995.

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

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

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

[8] 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*, 1998, pp. 675–682.

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

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

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

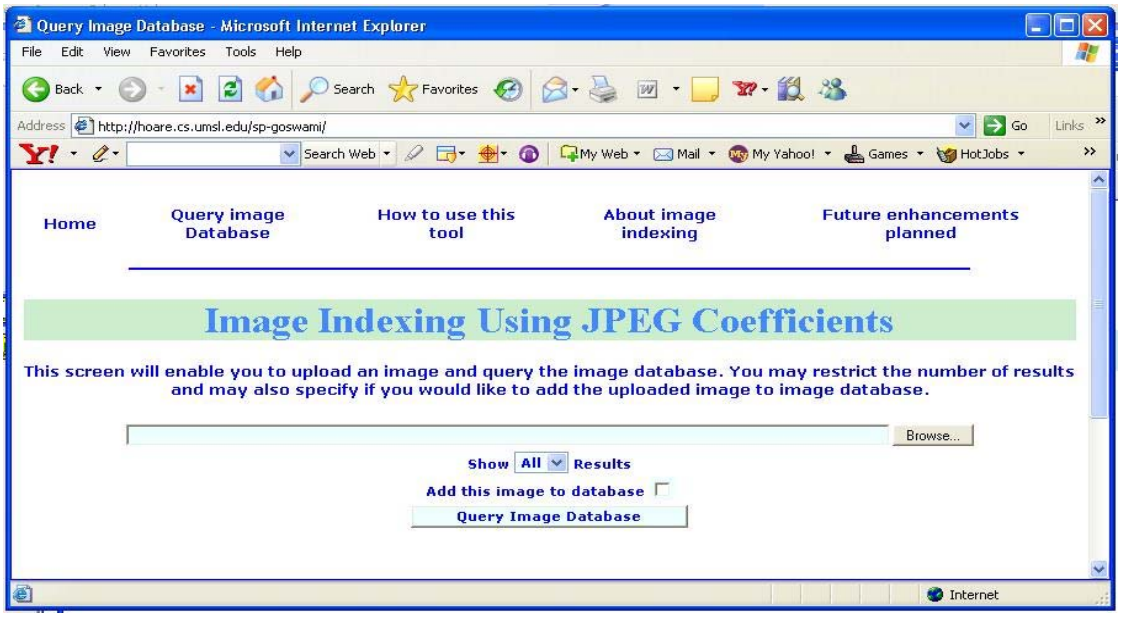

Fig. 3. Input screen

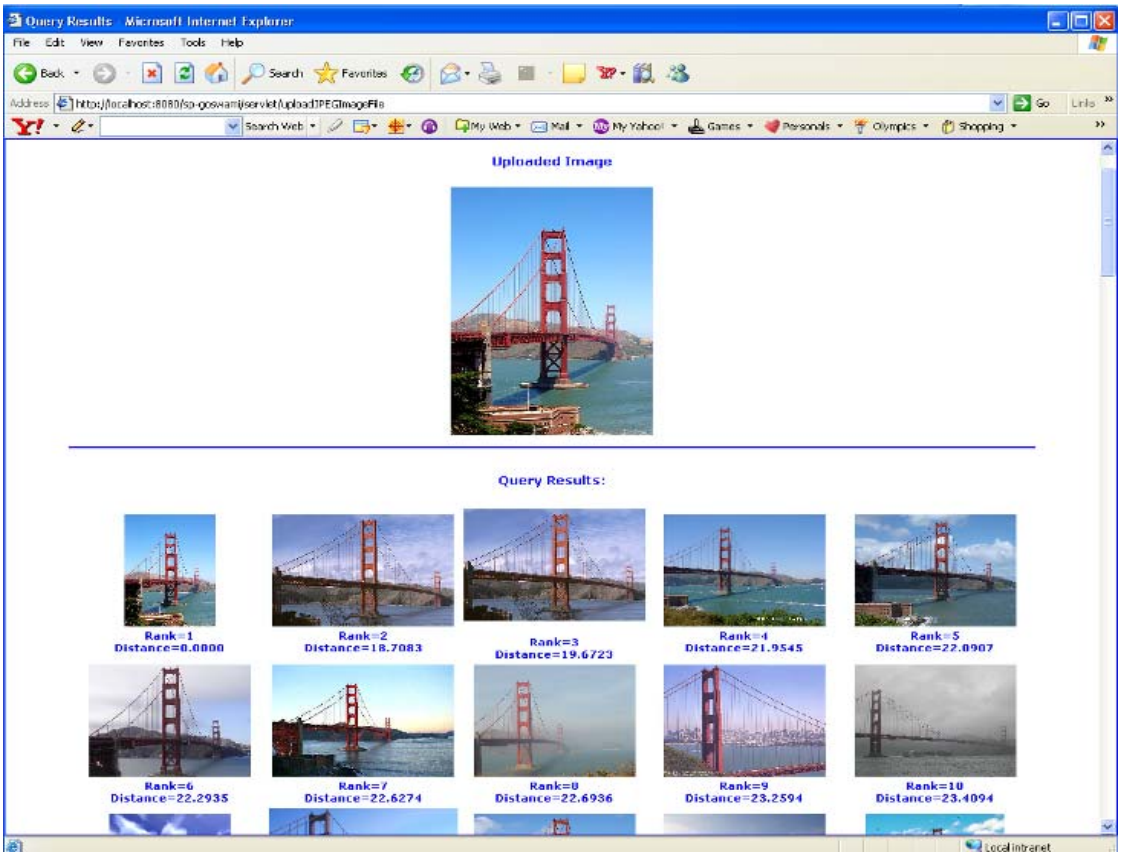

Fig. 4. Query results for Golden Gate Bridge image; database contains 177 images with 15 images of Golden Gate Bridge