AN IMAGE SYSTEM FOR CINDI

BEIHUA ZHANG

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

APRIL 2004

Your file Votre référence
ISBN: 0-612-91152-7
Our file Notre référence
ISBN: 0-612-91152-7

Abstract

An Image System for CINDI

Beihua Zhang

Content Based Image Retrieval (CBIR) becomes possible and necessary as computer graphics, machine learning, knowledgebase and database technologies mature. Though more research needs to be done in different aspects of CBIR, such as abstracting accurate semantic meaning from an image, current CBIR systems are actually well configured into modules, which can be re-used in different applications. The purpose of this thesis is to build a CBIR system, and incorporate Semantic Header for Multi-Media (SHMM), using these modules.

SHMM aims to be a web oriented image library with an automatic image addition and classification mechanism to support sample image based similarity search, and semantic description search as well as allowing the registered users to add image to the database. As a sample CBIR system, SHMM has its feature extraction layer, which supports colour and texture feature extraction that generates a 16-dimensional vector value. Based on this vector, a 16-dimensional SR tree is constructed. Using the nearest neighbor search technology on the SR tree, similarity search is supported. With backend database support, semantic description search, which is based on the keyword of the semantic meaning of image, is also implemented in SHMM. When a new image is added to the system, SHMM will automatically scan its feature, suggest the semantic description based on the similarity search result in the library, and wait for the user's response. Image contributor can accept the system's suggestion or inform system administrator via email to create new semantic description category in the system.

Acknowledgements

I express deep gratitude to my supervisor Professor B.C.Desai for his extraordinary and continuous support throughout my studies at Concordia University. Like many others, I consider myself blessed to be under his supervision. He was always gracious in giving me the time and strength even amidst tight schedule.

I am thankful to Professor N. Shiri, Professor L.Kosseim, and Professor G.Grahne for their encouragement during the entire course of my master degree. I am also thankful to the staff in the Department of Computer Science, especially Halina.

Last but not the least, I express deep veneration to my dearest parents for teaching me the importance of good education, from which everything else springs.

List of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Image database and image retrieval have been subjects of extensive research during the last 20 years. Research has focused on the design of robust image processing and recognition techniques from its beginning. Interest in this area is rapidly growing with the continuous evolution of telecommunication and computer technologies. The most popular and challenging jobs in the development of image databases include image processing for feature extraction, identification of salient objects, development of efficient data models to allow content-based indexing and retrieval, query formulation, and fuzzy query processing.

In the early 1990s, Content-Based Image Retrieval (CBIR), a technique for retrieving images on the basis of automatically-derived features such as colour, texture and shape, appeared after a workshop held by the US National Science Foundation (USNSF). The workshop clearly stated, "Computer vision researches should identify features required for interactive image understanding, rather than their discipline's current emphasis on automatic techniques" [31].

With the advent of powerful but inexpensive computers and storage devices and with the availability of the World Wide Web, CBIR, not yet a commercial success, has already been applied into different areas, such as Military, GIS, Digital Library, and medical applications.

1.2 The problem

This thesis is motivated by digital library requirements: classifying different types of images, adding new images as well as retrieving similar pictures from the image database. These requirements could be divided into the following sub-functions.

Setup image database: At the start of the project, storage strategy and content representation schema of images, as well as database schemas, should be decided. Because image databases in digital library applications could be large, system scalability and quick access need to be considered.

Query by Image Content: Query by examples and semantic description search should be supported.

Classify new images: Both automatic and semi-automatic classification should be provided as to support similarity search based on these classifications.

1.3 The Goal

This thesis presents a CBIR system based on a content representation schema called Semantic Header for Multi-Media (SHMM), which attempts to build an image library described in section 1.2. SHMM is a web-based, 2 layer CBIR system, which supports both syntactic and semantic descriptions of image content. With the simplicity of the design and well-accepted algorithms, SHMM serves to demonstrate what a CBIR system is and how to build a CBIR system.

1.4 Outline

Chapter 2 presents a survey of contemporary intelligent image database systems and provides the current state of the art in CBIR.

Chapter 3 describes the problem domain and examines various CBIR systems with their detailed differential approaches in modules and layers. The intent is to choose an approach most suitable for the project.

Chapter 4 introduces the SHMM system and discusses in detail its major components, their interactions, as well as algorithms.

Chapter 5 discusses the result achieved from the SHMM system and how these results differ from other systems.

Chapter 6 gives our conclusions, the contribution of this thesis and suggests for possible directions for future work on the SHMM.

Chapter 2

State of the Art survey

2.1 What is CBIR?

The earliest use of the term *content-based image retrieval* can be traced back to 1992, when the US National Science Foundation (USNSF) first outlined the major tasks for automatic retrieval of images from a database by colour and shape feature. The term has since been widely used to describe the process of retrieving desired images from a large collection on the basis of features (such as colour, texture and shape) that can be automatically extracted from the images themselves. The features used for retrieval can be either primitive or semantic, but the extraction process must be predominantly automatic. Retrieval of images by manually-assigned keywords is definitely not CBIR as the term is generally understood – even if the keywords describe image content.

CBIR differs from classical information retrieval in that image databases are essentially unstructured, since digitized images consist purely of arrays of pixel intensities, with no inherent meaning. One of the key issues with any kind of image processing is the need to extract useful information from the raw data (such as recognizing the presence of particular shapes or textures) before any kind of reasoning about the image's contents is possible.

CBIR draws many of its methods from the field of image processing and computer vision, and is regarded by some as a subset of that field. While research has been expanded in breadth and depth, CBIR differs from these fields principally through its emphasis on the retrieval of images with desired characteristics from a collection of significant size. Image processing covers a much wider field, including image enhancement, compression, transmission, and interpretation [2]. While there are grey areas (such as object recognition by feature analysis), the distinction between mainstream image analysis and CBIR is usually definitive. An example may make this clear. Many police forces now use automatic face recognition systems. Such systems may be used in one of two ways. Firstly, the image in front of the camera may be compared with a single individual's database record to verify his or her identity. In this case, only two images are matched, a process few observers would call CBIR. Secondly, the entire database may be searched to find the most closely matching images. This is a real example of CBIR.

2.2 CBIR modules

Though more research needs be done in different aspects of CBIR, such as abstracting accurate semantic meaning from an image, efficiently grouping raw data, current CBIR systems are actually well configured into modules [1]. Generally, a CBIR system has three fundamental modules based on functionalities. Content Expression Module, denoted as CEM (2.2.1) is the most important and also the most difficult part of CBIR, which handles not only physical meanings of images but also semantic meanings. It normally can be subdivided into 3 layers: including Feature Extraction Layer (FEL), Object Recognition Layer (ORL), Spatial Modeling and Knowledge Representation Layer (SMKRL). There are some exceptional applications where FEL and ORL are combined as syntactical feature extraction layer [3]. Index Structure Module, denoted as ISM (2.2.2), creates indices and data structures based on the content abstracted from Content Expression module. It is easier and more stable in techniques. R* tree family [7, 8] and SS tree [9] is usually the choice, among which SR [10] tree stands out due to low CPU load and number of disk reads. User Interface Module, denoted as UIM and discussed in section 2.2.3 is the only interaction between system and users; the interaction could be queries, classifications, and answer refinements. Each of the given modules is described in the following sections.

2.2.1 Content Expression module

Content Expression module aims to extract both syntactic and semantic meaning from images so as to provide enough knowledge for future image retrieval. This module largely affects or even determines the remaining two modules in system design and therefore it requires the most critical strategic decision procedure within a CBIR system.

A simple example may illustrate the importance of this module. The simplest way to measure the similarity of two images is to take a difference of colour values at each pixel, known as the pure physical meaning of the image, and average all the differences over the entire image. With such a feature selection, there is really no need to extract semantic meaning from images. However, as image matching is performed on the raw image data, designing an effective indexing and search scheme becomes almost impossible. Furthermore, systems using such pixel-wise comparisons obviously can retrieve only the images that have a layout very similar to that of the sample image. This inevitably leads to a poor database query scheme that has very limited image retrieval capabilities.

At the other extreme, suppose that the image's semantic content is fully interpreted, and a complete set of descriptive keywords is generated as a result of the image interpretation. In this case, the voluminous raw image data are reduced into abstractions that retain the critical information embedded in the image. With these descriptive keywords, an effective indexing and search scheme can be easily constructed by using database techniques. Thus a powerful database query scheme that enables users to retrieve images by referring to any specific parts and contents of an image becomes available. With high-level image features, however, feature defining and capturing, both in syntax and semantics, becomes problematic because none of the contemporary computer vision and image understanding techniques are able to automatically interpret, and fully annotate the contents of general images.

To get as much syntactic and semantic meaning from an image as it is possible, most traditional approaches to image data management use multilevel abstraction mechanisms. Generally, there could be three layers of the Content Expression module: this depends on whether it is required to extract either the primitive or semantic feature from an image, as illastrated in Figure 1. Figure 1 was first introduced by [3] in 1999 as the model to examine the pioneer CBIR systems that appeared in the early 1990's and after that it became an accepted model for CBIR systems.

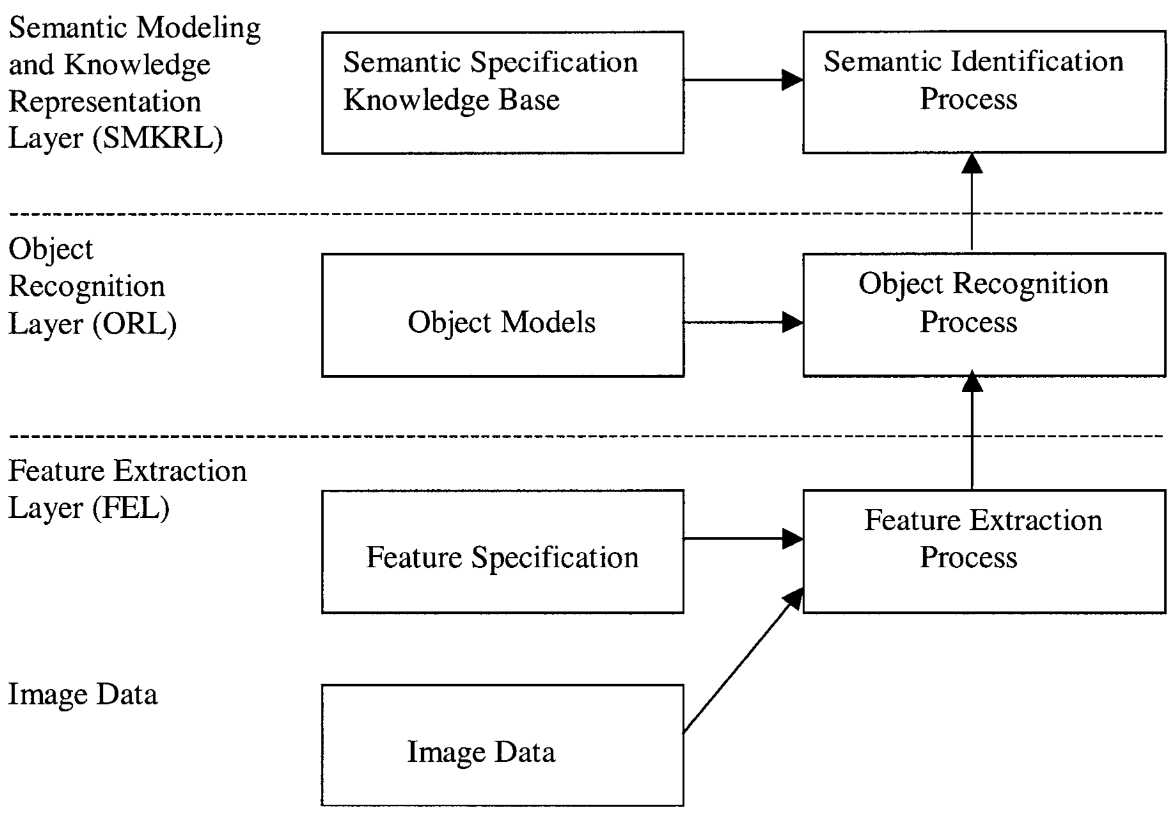

Semantic Modeling and Knowledge Representation Layer (SMKRL)
Semantic Specification Knowledge Base
Semantic Identification Process
Object Recognition Layer (ORL)
Object Models
Object Recognition Process
Feature Extraction Layer (FEL)
Feature Specification
Feature Extraction Process
Image Data
Image Data

Figure 1: Component of a CBIR from [3]

2.2.1.1 Feature Extraction Level (FEL)

A typical approach in FEL is to extract the set of features, which is specified by the system, and then to seek these features in the image. Normally, image features include colours, textures, shapes, and edges. Different applications may concentrate on extraction of one or several features, while colour histogram and textures are the most popular choices. These features are mapped into a multidimensional feature space that can allow similarity-based retrieval of images. Features are in the form of encoded vectors, which provides the basis for indexing and searching mechanisms of an image database. Logically, features in an image can be classified as global or local [21]. Global features generally emphasize "coarse-grained" [3] matching techniques. The global feature extraction techniques transform the whole image into a "functional representation", where finer details within individual parts of the image are ignored. Global features for image carry out uniform processing over the whole image for the chosen feature. Therefore, global features are well suited for processing the type of queries that deal with images as single entities during the matching process. In local features approach, also called "fine-grained" [3], the images are segmented into a collection of smaller regions, with each region representing a potential object of interest. Local features can be used to identify salient objects in an image and to extract more information about the finer details in the image. Though local features in FEL might deal with some simple semantic object, such as shape, generally FEL deals with syntactic meaning of images and is also called low-level image process.

As shown in Figure 1, for example, both image data and feature specification are the input for Feature Extraction Process, known as the key function of FEL. Normally feature specifications include the feature definitions, the feature's implementation strategies and algorithms. Image data in the Figure 1 represents raw data for each original image. The output of Feature Extraction Process, which is also the output in FEL, is the input for Object Recognition Process, which also takes object models as an input.

Since colour is the basic and the important feature in feature extraction process, we present the approaches used below.

Colour Space

A colour space is a method by which we can specify, create and visualize colour [34]. As humans, we may define a colour by its attributes of brightness and colourfulness. A computer may describe a colour using the amounts of red, green and blue phosphor emission required to match a colour. A printing press may produce a specific colour in terms of the reflectance absorbance of cyan, magenta, yellow and black inks on the printing paper [38].

A colour is thus usually specified using three co-ordinates, or parameters. These parameters describe the position of the colour within the colour space being used. They do not tell us what the colour really is since the meaning of the colour depends on what colour space is being used. An analogy to this is that we may be told the directions to visit Concordia from Mont Royal. These directions only mean anything if we are on Mont Royal or know how to get there; otherwise, the instructions are meaningless.

There are many common and computer related colour spaces defined to derive colour features from images, among which are RGB [16], XYZ [17], L*a*b* [20], HVC [23]. HVC is the most typical approach.

- **RGB colour system**

  RGB colour system represents the colour of R (red), G (green), and B (blue), corresponding to the pixel value of images. However, RGB colour system cannot represent the human perspective of colour. First, an RGB tristimuli does not carry direct semantic information about a colour. People cannot visualize a colour given its RGB tristimuli. Second, equal geometric distances in the RGB colour space do not generally correspond to equal perceptual changes in colours. Finally and most importantly, the R, G and B values at each pixel change remarkably when a uniform region is unevenly illuminated, making image segmentation using the RGB colour space very vulnerable to illumination changes. Also this approach is device dependent, which means image understanding depend on the direction of viewing, direction of illumination, and intensity. This dependence will greatly influence the performance of a CBIR system.

- **CIE XYZ colour system**

  The CIE colour theory is based on RGB, and it introduces the additive mixture of three imaginary primaries X, Y, Z. These primaries cannot be seen by a human eye or produced, because they are too saturated. However, this fact will not influence the rightness of colour expression since any perceived colour can be described mathematically by the amounts of these primaries. Because XYZ system is a linear combination of R, G and B values, the XYZ colour system inherits all the dependences on the imaging conditions from RGB colour system. The following conversion matrix is one of the most frequently used formulas [17] (based on NTSC RGB):

Table 1: Transforming Matrix from RGB to CIE XYZ

X = 0.607*Red + 0.174*Green + 0.201*Blue

Y = 0.299*Red + 0.587*Green + 0.114*Blue

Z = 0.000*Red + 0.066*Green + 1.117*Blue

However this colour system has an obvious gap between human perspective and machine process.

- **CIE L*a*b* colour system**

  It is a theory based on XYZ colour system, where the colour feature L* correlates with the perceived luminance, colour feature a* correlates with the red-green content of a colour and b* reflects the yellow-blue content. The transformation format is as follows [20]:

Table 2: Transforming Matrix from CIE XYZ to CIE L*a*b*

$$\text{if } Y/Yo > 0.008856 \quad L^* = 116*(Y/Yo)^{1/3} - 16$$

$$\text{if } Y/Yo \le 0.008856 \quad L^* = 903.3*(Y/Yo)$$

$$a^* = 500*((X/X0)^{1/3} - (Y/Y0)^{1/3})$$

$$b^* = 200*((Y/Y0)^{1/3} - (Z/Z0)^{1/3})$$

X, Y, Z are the primaries in XYZ colour system, while X0, Y0, Z0 are values of a nominally white object-colour stimulus.

As is easy to tell, this theory uses a non-linear transformation, which means that the system becomes unstable when intensity is small [20]. Also this colour system is dependent on viewing directions, object geometry, highlights, direction of illumination, intensity, and colour of the illumination.

- **HVC colour system**

  HVC colour system comes from Munsell [22] colour coordinate system [23], which is famous for its successful imitation of human colour perception. It represents a colour in terms of hue (H), which indicates the types of the colour, value (V), which tells the total amount of light, and chroma(C) that describes how much white light is mixed with the colour (purity). There are several important advantages in using HVC colour system. First, as the luminant and chromatic components of a colour are completely separated in the HVC colour space, it enables us to derive an effective model to handle images under non-uniform illumination conditions. Furthermore, [5] proves that "colours in the HVC colour system are arranged in such a way that equal geometric distances correspond to equal perceptual differences, making it the ideal colour space for colour clustering purpose in CBIR".

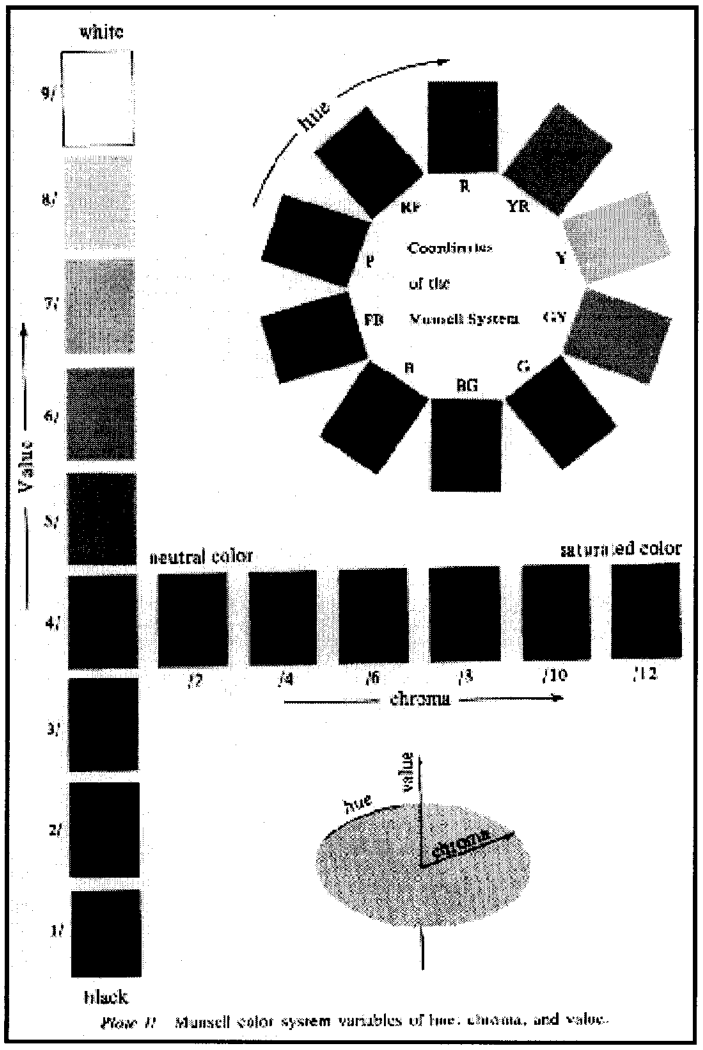


Figure 2: Schematic diagram of the three axes of Munsell color system from [19]

Figure 2 demonstrates the schematic diagram of the three axes of the Munsell colour system by giving examples from the real colour world.

Assume that the colour image is represented in 24 bits per pixel, which means 8 bits each for R, G and B, respectively. There are several ways to mathematically transform the RGB to the HVC colour system. Because CIE L*a*b* colour space is known for its good perceptual correspondence and simple computation [20], the formula that NTSC RGB values are first transformed into CIE XYZ, and then changed to CIE L*a*b*, and then altered to HVC values [23] is as follows:

Table 3: Transforming Matrix from CIE L*a*b* to HVC

$$\mathbf{X = 0.607*Red + 0.174*Green + 0.201*Blue}$$
$$\mathbf{Y = 0.299*Red + 0.587*Green + 0.114*Blue}$$
$$\mathbf{Z = 0.000*Red + 0.066*Green + 1.117*Blue}$$

$$\mathbf{L^* = 116*(Y/Yo)^{1/3} - 16}$$
$$\mathbf{a^* = 500*((X/X0)^{1/3} - (Y/Y0)^{1/3})}$$
$$\mathbf{b^* = 200*((Y/Y0)^{1/3} - (Z/Z0)^{1/3})}$$

$$\mathbf{H = arctan(b^*/a^*)}$$
$$\mathbf{V = L^*}$$
$$\mathbf{C = (a^{*2}+b^{*2})^{1/2}}$$

X, Y, Z are the primaries in XYZ colour system, while X_0, Y_0, Z_0 are values of a nominally white object-colour stimulus.

Image Features

Image features in CBIR systems include colours, textures, shapes, and edges. Different applications may concentrate on one or several features extraction while colour histogram and textures are the most popular choices.

- **Colour Histograms**

Colour histogram is a way to group colour values within an image so as to generate colour meaning for it. To get the colour meaning from an image, the traditional way is to equally subdivide the image into a large number of small bins, and then count the number of pixels each bin contains. The major problem of the histograms created by this method is that colours from two adjacent histogram bins reveal only trivial differences. A small change in illumination conditions or the influence of noise can cause pixels to drift from one bin to another. And therefore, though two images are similar in colour, they might have two different sets of histogram bins. While introducing colour understanding to CBIR systems, it is possible that we can use human perspective to describe image colour histograms.

There are two major rules in creating colour histograms. First, matching of two histograms should result in an approximate, rather than exact matching of the correspondence images. For CBIR, approximate matching is more useful than exact matching because users are generally more interested in the image similar to, rather than exactly the same as, the sample image when querying the database. Second, the histogram must be able to capture macro colour configuration, rather than trivial colour details of an image. It is quite understandable if we look at human colour perception. It has been proved that human vision system is relatively insensitive to small colour difference [34]. Colour names that are widely used in our daily lives, such as "Red","Green","Blue", actually represent groups of colours with a wide spectrum range. Therefore, human beings evaluate the similarity between images based on overall impression, and trivial colour differences are not of much importance in the process.

Colour histogram is one of the most important techniques for content-based image retrieval [36] because of its efficiency and effectiveness. All colour histogram-based image retrieval systems are composed of at least three operations: a suitable colour space with the specific number of cells in it, an algorithm of how to count pixel number for cells in colour space, and an algorithm of how the pixel number values in the similarity matching.

- **Textures**

Texture is a blanket term used to describe the brightness variations observed in all surfaces [24]. It conveys information about pixel and surrounding. The texture of an image is the distinctive physical composition or structure of something, especially with respect to the size, shape, and arrangement of its parts. The key issue in texture detection is to distinguish texture from non-texture regions. Saying that distinguishing texture from non-texture regions is the issue also attributes to the fact that texture detection and image content recognition are a pair of "chicken and egg" problems. It is easy to understand that complete texture detection cannot be achieved without recognition of the image content while image content recognition relies highly on image segmentation, which includes texture detection. Furthermore, there is no clear boundary between texture and non-texture regions. Hence, a texture detection method has to be made domain-dependent and application-dependent.

There are two main points inside a texture based image retrieval system. One is the texture rule, which consists of many parameters to describe a texture pattern. The other is the algorithm or more specifically, the metric, to demonstrate the similarity between textures. Texture feature describes the content of many real-world images, for example, clouds, trees flowers. Therefore, it has been widely implemented in CBIR systems.

2.2.1.2 Object Recognition Layer (ORL)

This layer focuses on analyzing features extracted from the feature extraction layer and provides solutions on how to group them. The process involves matching features with object models stored in a knowledge base. In general, an object model is a template describing a specific object. Based on the flexibility of definition, there are two types of template matching: fixed template matching and deformable template matching [29]. Approaches based on fixed templates are useful when object shapes do not change with respect to the viewing angle of the camera. For example, application for X-ray image database could use this technique since target images have a fixed viewing angle. Even if image intensities change frequently and sometimes there is noise in images, as far as templates do not change, systems with this approach can relieve from these influence via applying image subtraction techniques. Approaches based on deformable template are more suitable for cases where objects in the database may vary due to rigid and nonrigid deformations. A typical example could be an application for face recognition. A certain level of fuzziness and imprecision in object recognition is inevitable and needs to be incorporated in the similarity measure in order to increase the success rate of queries and not to exclude good candidates; such fuzzy matching is implemented in almost all applications of CBIR. The main difference between implementations in this level is the method used to build the related knowledge base, Frame based or Constraints based. In the Constraints-based methodology, the domain knowledge is represented using a set of constraints in conjunction with formal expressions such as predicate calculus or graphs[3]. Such predicates are augmented with procedural information. A constraint is usually a relationship between two or more objects that needs to be satisfied. This approach is suitable for systems with sufficient description information of images. In Frame-based methodology, frames are used to represent knowledge relevant to particular objects, situations, or concepts [3]. A frame usually consists of a name and a list of attribute-value pairs. A frame can be associated with a class of objects or with a class of concepts and frame abstractions allow encapsulation of file names, features, and relevant attributes of image objects. Though a lot of research has been done in ORL and some successful implementations of several prototypes have been reported, such as [13,14], lack of precise models for object representation and the high complexity of image processing algorithms make the development of CBIR still a challenging task.

2.2.1.3 Spatial Modeling and Knowledge Representation Layer (SMKRL)

The main function of this layer is to maintain the domain knowledge of representing spatial semantics associated with image databases. Semantics at this level are based on spatial actions describing the relative locations of multiple objects, which have been extracted and recognized by object recognition layer (ORL). Such semantics are used for high-level indexing and content-based retrieval of images. An example involving such semantics is a range query, which involves spatial concepts such as "left, right, close by, smaller than, behind, between". A geographic information system (GIS) is probably the most common application implementing spatial semantics and content-based retrieval based on range queries. The general approach for modeling spatial semantics is based on identifying spatial relationships among objects when they are recognized and marked by ORL. Various knowledge-based techniques are applied in representing spatial relationships, such as *semantic networks* [28]. Semantic networks are used extensively in artificial intelligence applications. "*Semantic networks* are graphs of nodes representing concepts that are linked together by arcs representing relationships between these concepts."[3]. Contrary to the traditional approach that identifies an object each time it is used in a relation, in semantic network, it defines each concept or object once, and uses pointers for cross references. Therefore the semantic network approach achieves efficiency in the processing time and storage space. Techniques to recognize objects in ORL, for example, constraints-based approach and frames-based approach, can also be applied in this layer to describe the spatial relationships between objects [3].

The problem of content expression in CBIR has been actively studied for decades; however, extracting knowledge from images and representing such knowledge is a challenging problem and requires further research. Therefore development of a general-purpose, automatic image database system capable of supporting arbitrary domains is a challenging task due to the limitations of existing image processing knowledge representation models.

2.2.2 Index Structure module

Usually CBIR systems represent images by using certain image feature spaces, define similarity metrics on those feature spaces and retrieve image in an order similar to the user-provided reference images. Under this scenario, each image in a CBIR system is represented as a point in multi-dimensional feature space, and a distance between two feature points, either Euclidean or city block distance, is frequently used as a similarity metric. For two points A $(x_1, x_2, \dots, x_n)$ and B $(y_1, y_2, \dots, y_n)$ that represent two images in an *n* dimensional feature space for a CBIR system, the Euclidean distance between A and B is $((y_1 - x_1)^2 + (y_2 - x_2)^2 + \ldots + (y_n - x_n)^2)^{1/2}$, while city block distance for A and B is $|y_1 - x_1| + |y_2 - x_2| + \ldots + |y_n - x_n|$. To provide similarity-based image search, the system must have search schemes as the nearest neighbor search and the range search. The nearest neighbor search finds the n-nearest neighbors to a reference, while the range search retrieves all the images within a region of the feature space. Index structure techniques have matured over the last few decades. Among those that have been proposed to support these two search schemas, there are some major differences [5]:

- **Dynamic or Static**

  A dynamic index structure supports dynamic index updates, which include insertion and deletion. On the other hand, a static index structure requires that the complete dataset be given beforehand, and no insertion and deletion occurs after the index of the given dataset is created.
- **Memory resident or Disk-resident**

  A memory-resident indexing structure creates indices in computer memory, while a disk-resident indexing structure stores indices on hard disk. Disk-resident indexing structure can accommodate a large-scale dataset.
- **High dimensional or Low dimensional**

  Low-dimensional index structure, which is usually classical indexing method such as B-Tree [30] and its variants, can index only one dimensional data points. High-dimensional index structure, which is recent indexing structure such as R-Tree [7] and its variants [8], can support multiple-dimensional indices. The dimension ranges from 2 – 20 or even higher.

For a CBIR system, an image database must be able to support dynamic database updates and also to accommodate hundreds of thousands of images, each of which might be represented by feature vectors of between 10 to 100 dimensions. Therefore, an index structure in a CBIR system must have the capabilities of dynamic updates, efficient disk-based implementation, and high-dimensional indexing. Many index structures could be implemented such as KDB tree [6], R tree [7], R* tree [8], SS tree [9], SR tree [10], among which SR tree stands out for its outstanding performance in CPU time and the number of disk reads.

KDB tree

KDB tree was one of the first structures proposed to support high-dimensional data indexing and disk resident implementations [6]. It is a height-balanced tree and its tree structure is constructed by dividing the search space into sub regions with coordinate plane recursively.

A KDB tree consists of the following two types of nodes:

Non-leaf nodes: These contain a collection of (*region, np*) pairs. *np* is the address of a child node, and *region* refers to a subspace which can be expressed as $I_0 * I_1 * \ldots * I_{k-1}$, where k is the number of dimensions and I_i is the interval of the subspace along dimension *i*.

Leaf nodes: These contain a collection of (*point, dp*) pairs, where *point* is the coordinates of a point in the *k*-dimensional data space, and *dp* is the address of the database record that is associated with *point*.

In a KDB tree, each leaf node provides a bucket in which to store one data point, while each non-leaf node represents a region of the space, which is the result of data space partitioning. When a non-leaf node P is overfilled, the region represented by P is further partitioned, resulting in the splitting of P. Therefore, there is no overlap between any pair of regions; thus each data point has only one path between itself and the root. Figure 3b shows how a traditional KDB tree is constructed. The non-leaf nodes 1,2,3 represent subspaces while leaf nodes, point A, B, C, D, E, F, G and H stand for the value vectors of the images to be indexed. In CBIR systems, each point represents a value vector for an image. Here, points A, B, C, D, E, F, G and H represent the value vector for each image respectively which will be generated after FEL or ORL. To create indices on images is to create indices on these value vectors. Originally, only one space is created, known as being related to the root node. During the process of adding a new leaf to the index structure, which represents adding a new image to the image database, non-leaf node splits when it is overfilled. To process similarity search on images is to process nearest neighbor or range search among the created indices structure. For range search, based on an example image's value vector and so formed query space, system retrieves all the images with their spaces having intersection with the query space. For example, if we have a new image with its query space having intersection with space 1 (The space of an example image can only have one intersection with spaces in KDB tree), image A, B and C will be retrieved as the result for the range search. For nearest neighbor search, first, based on the value vector of a sample image, a sphere of selected radius, which is also known as the definition of similarity, will be generated. Afterwards, any image with its value vector falling into the sphere will be retrieved. Despite the advantage that the data retrieval time in KDB tree depends only on the height of the tree, a KDB tree does not define the minimum number of entries in a node; therefore it cannot ensure the minimum storage utilization, which reduces the performance on range queries and nearest neighbor queries [10].

R tree & R* tree

R tree was initially proposed to index data objects of non-zero size in high-dimensional spaces. Although the R tree and R* tree are originally designed for rectangles, it can be used solely for points and known to be effective also as a point access method for multi-dimensional index [7].

Let M be the maximum number of entries that will fit in one node, and m ($2 \leq m \leq M$) be the minimum number of entries in a node. An R tree is constructed as follows:

1. A non-leaf node contains between m and M entries of the form (*rect*, *np*), where *np* is the address of a child node, and *rect* is the minimum bounding rectangle that covers rectangles of all the entries in the child node.
2. A leaf node contains between m and M entries of the form (*point, id*), where *id* refers to a data object in the database, and *point* is the coordinates of a point in the k-dimensional data space, where k is the number of dimensions of the whole domain.
3. The root node has at least two children unless it is a leaf.

4. All leaves appear on the same level.

An R tree is built based on the clustering of all points in a data space. Each non-leaf node P corresponds to a cluster in the space with its minimum bounding rectangle representing the extent of the cluster. When a node is overfilled, the cluster represented by the node is partitioned, and the node is split into two nodes. Figure 3c shows how a traditional R tree is constructed. The non-leaf nodes 1,2,3 represent subspaces while leaf nodes, points A, B, C, D, E, F, G and H stand for the value vectors to be indexed. In CBIR systems, A, B, C, D, E, F, G and H are the value vector for each image respectively.

In R tree, each node in the tree must store between m and M entries; therefore, the worst-case space utilization for all the nodes except the root is m/M. However, since an R tree allows overlapping of minimum bounding rectangles in different nodes, there can be more than one path between any data object and the root. Hence, the search algorithm must traverse more than one path to search for the desired data. Therefore, the search time of a point query depends on the amount of overlap and is not determined by the height of the tree [10]. A simple example might help to understand this scenario. Suppose there is an R tree constructed as in Figure.3c, we have a search requirement B, whose query space falls into the intersection between space 1 and space 3. Therefore, to do the search, not only we have to examine space1, but also have to check space 3. The extreme case would be that all the space in the non-leaf nodes has intersections, and the search value is unfortunately falls in that part.

R* tree is the most successful variant of the R tree, which shares the same tree structure with the R tree. The optimization comes from the minimization of the area of enclosing rectangles in nodes while processing data insertion and node split algorithms [8].

SS tree

SS tree is an index structure designed for similarity indexing of multidimensional point data. Having a similar configuration to R* tree, SS tree employs minimum bounding spheres rather than minimum bounding rectangles for the region shape. Hence, the key difference between SS tree and R* tree is that SS tree divides points into isotropic neighborhoods by utilizing bounding spheres while the R* tree divides points into small regions by utilizing bounding rectangles.

A SS tree is constructed as follows:

1. Non-leaf nodes contain the form (*sphere* , *np*), where *np* refers to a child node, and *sphere* is the minimum bounding sphere that consists of (*centroid, radius*). The *centroid* is the mean value of the feature vectors in the child node, and the *radius* is the distance from the *centroid* to the further feature vector.
2. Leaf nodes contain entries of the form (*vect, data*), where *data* holds the data for the leaf, and *vect* holds the data's feature vector.

Figure 3d shows how a traditional SS tree is constructed. The non-leaf nodes 1,2,3 represent subspaces while leaf nodes, points A, B, C, D, E, F, G and H stand for the value vectors to be indexed. In CBIR systems, A, B, C, D, E, F, G and H are the value vector for each image respectively.

The utilization of bounding spheres has been proven to yield a higher node branching factor, and a better data clustering in a high dimensional data space [9].

SR tree and its advantage

Bounding rectangles and bounding spheres have both merits and demerits. Experimental results show that bounding rectangles divide points into small volume regions with long diameters while bounding spheres divide points into larger volume regions but with shorter diameters [10]. For nearest neighbor search queries, bounding spheres are more advantageous than bounding rectangles because the lengths of region diameters have more influence to the performance on nearest neighbor queries than the volumes of regions [10]. With the desire to divide points into regions both with small volumes and with short diameters, the SR-tree (Sphere/Rectangle) is developed to combine utilization of bounding spheres and bounding rectangles.

Let M be the maximum number of entries that will fit in one node, and m ($2 \leq m \leq M$) be the minimum number of entries in a node. A SR tree is constructed as follows:

Leaf node: A leaf node contains between m and M entries of the form (*p, data*), where *data* holds the data of the leaf, and *p* holds the data's pointer.

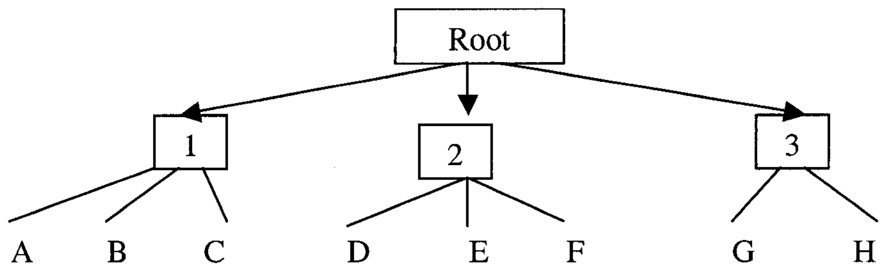
$$\begin{aligned} L &: (E_1, \ldots, E_n) \qquad (m_L \leq n \leq M_L) \\ E_i &: (\boldsymbol{p},\ \boldsymbol{data}) \end{aligned}$$

Non-leaf nodes: A non-leaf node contains between m and M entries of the form (*S,R, w, np)*, where *np* is the pointer to a child node, *w* is the total number of data entries stored in the sub tree pointed to by *np*, *S* and *R* are the minimum bounding sphere and minimum bounding rectangle of the child node, respectively.

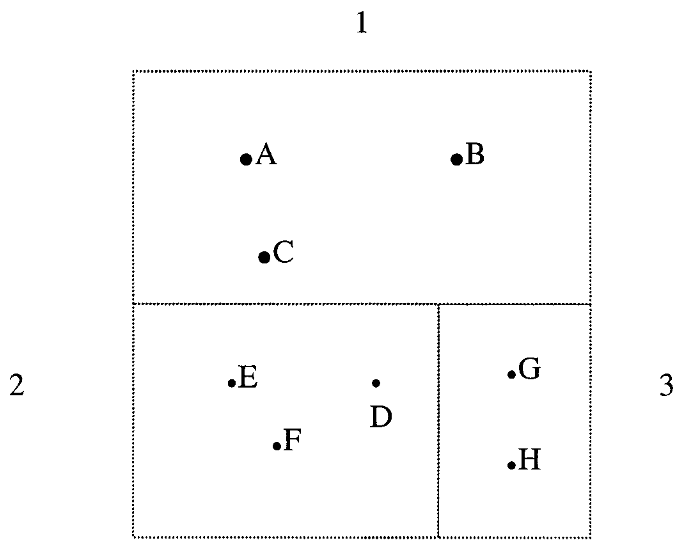
$$\begin{aligned} N &: (C_1, \ldots, C_n) \qquad (m_N \leq n \leq M_N) \\ C_i &: (\boldsymbol{S}, \boldsymbol{R}, w, \mathit{child\_pointer}) \end{aligned}$$

Figure 3e shows how a traditional SR tree is constructed. The non-leaf nodes 1,2,3 represent subspaces while leaf nodes, points A, B, C, D, E, F, G and H stand for the value vectors to be indexed. In CBIR systems, A, B, C, D, E, F, G and H are the value vector for each image respectively.

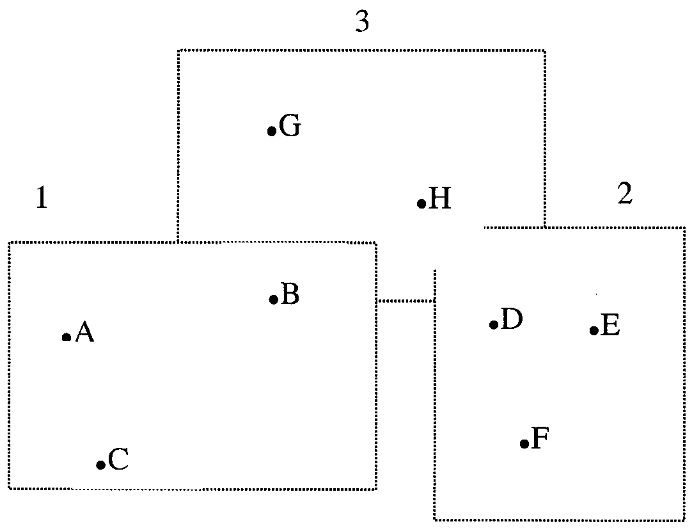
According to performance tests in [10], the SR-tree outperforms the R*-tree and the SS-tree especially for high-dimensional and non-uniform data, which are likely to appear in the actual image applications. Therefore, recent CBIR systems always choose SR tree as their index structure for their multidimensional value vectors, which could be colour, texture, edge, or most commonly a combination of all of these features.



a: Example Data

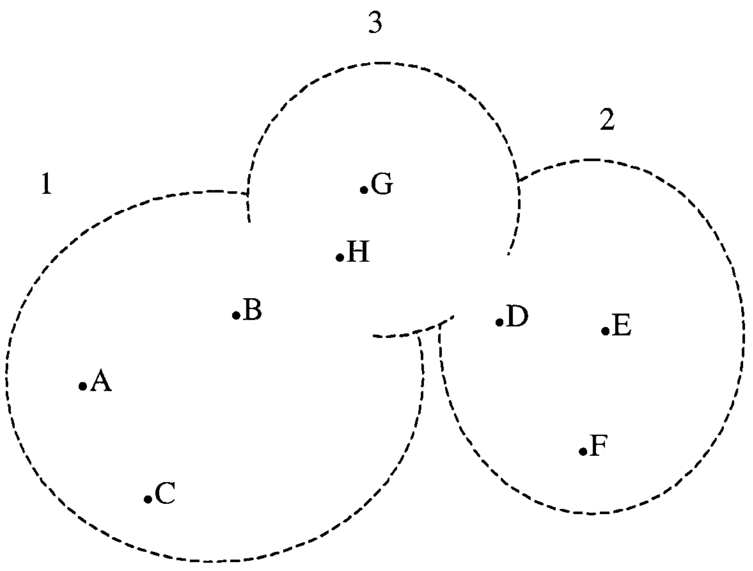


b: Construction of KDB tree

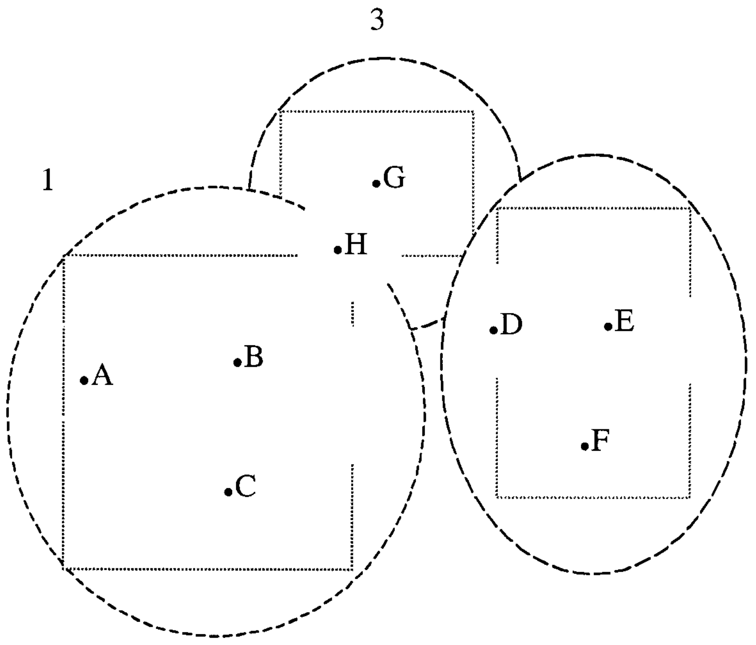


c: construction of R tree

Figure 3: Construction of KDB, R, SS, SR tree



d: construction of SS tree



e: construction of SR tree

Figure 3 cont'd: Construction of KDB, R, SS, SR tree

2.2.3 User Interface Module

Web-based interface is the most popular choice in current CBIR systems for the user due to its convenience in connection over the Internet and its excellent look and feel. From functional point of view, user interface module could be divided into two categories; namely "*Query by image examples*" and "*Interaction*" [26].

Query by image examples is the most well known and also most thoroughly researched schema for user interface module. While providing an example image as an input, a CBIR system will examine it, search for the most similar images in the system, and display the retrieval results in a predefined order. However, since an image is suited to be as an example only when another image with the same objects is the aim of the search, it is simply not sufficient and practical to find example images in all cases. Semantic description search, on the other hand, uses general contents of an image as keywords to provide queries. Though semantic description search has nothing to do with the example image, still we categorize it as "*Query by image examples*" since logically it is more like it than "*Interaction*". The query result has an inherent display dimension, which depends on the variety of queries [1]. In "*Semantic description search*" mode, when the query is exact, (for example, the system only retrieves images hundred percent belonging to the semantic descriptions) the result of the query is a set of images, and the sequence of them does not matter at all. In "*Query by image examples*" mode, for approximate query, which is what a CBIR system usually dealing with, the query result is the images with similarity ranking respect to the query. Therefore, most systems display these images in a 2D grid, in the order of related rankings [27].

Interaction was originally developed for text, and only was applied to image retrieval several years ago. It comes with the idea that the retrieval procedure is no longer a feature matching algorithm but a complex process integrating image classification [1]. During the retrieval process, users give their feedback to evaluate the retrieval results. Depending on when the "*Interaction*" mode is applied, it can be subcategorized as "*Classification*" and "*Answer refinement*" [11].

Classification is important since similarity measurement is hard to describe in machine language, and even it is subjective to change from person to person depending on domain expertise. Compared to the traditional manually or automatically classifying function in text-based systems such as digital library, image classification is more complicated and less efficient due to the difficulties of image interpretation. Therefore, it is usual for CBIR to have both automatic and semi-automatic classification function, which requires user's involvement.

Answer refinement, or relevance feedback is important in interaction process, which could bring into both long term and short term refinement procedures [11]. Long term refinement keeps updating and adjusting the image database organization with the help of user's evaluation of the retrieved images. Short term refinement determines the meaning of similarity gradually with users preferences in the loop. The idea comes mainly because users can normally judge the relevance of a set of images displayed on the screen quickly. Furthermore, only a limited number of current systems are capable of matching users' needs accurately first time round due to the difficulites of image content understanding. The key technique used is a refining algorithm that differs a lot between applications. For query result, as apparent from *Query by image examples* mode, there is an abundance of information available for display [1], such as how many people retrieve the same image and what's their preference of ranking. Therefore, the traditional 2D grid model might not satisfy the requirement, though it is still the current choice in implementations.

2.3 Survey of well-known CBIR systems

Since 1990, a number of CBIR systems have appeared and brought new technology into this area [3]. Also competition among these systems helped their development and led to maturity of CBIR. Among pioneer CBIR systems, the following stand out not only for their success in different areas but also for their contributions to CBIR technology: QBIC by IBM [4], CHABOT [12] by UC Berkeley, KMeD [13] by UC Los Angeles, PICTION [14] by the State University of New York, Buffalo, Photobook [15] by MIT.

QBIC is a comprehensive and complete commercial image database system, which allows querying of the database by sketching features and providing colour information about the desired objects. It employs multiple image features such as colour histogram, text and object shape to retrieve images, and adopts the R* tree to form database indices on the basis of these image properties.

CHABOT is a picture retrieval system for a database that will eventually include over 500,000 digitized multi-resolution images, which is part of the digital library project belonging to the State of California Department of Water Resources (DWR) [12]. It integrates the use of stored text and other data types with content-based analysis of the images to perform "concept queries".

KMeD known as Knowledge-Based Medical Database System has a domain knowledge of descriptions about the shapes of objects with simple structures such as tumor, brain, bones, etc. It has four layers consisting of the raw data layer, feature and content layer, schema layer and knowledge to represent the various aspects of an image objects' characteristics. Besides automatic image processing, human assisted object recognition process is also implemented in the system.

PICTION uses a combination of text annotation and image processing techniques for face recognition and identifying relative positions of people in images. It involves three steps of face recognition: face detection to locate a face inside an image; feature extraction where various parts of a face are detected; and face recognition where the person is identified by consulting a database containing facial models.

PHOTOBOOK is known for its learning mechanism while expressing the meanings of images. It uses collective texture models that interact to explain texture patterns in pictures, observes the user's annotation to pieces of images and then finds similar regions to which to propagate the user's preference on the image understanding.

While these systems contributed to the process of defining the architect of typical CBIR system, we can still use the standard CBIR modules to examine these well-known systems. Among three fundamental modules we discussed in section 2.2, many differences lie in Content Expression module. Therefore, we will focus on the different approaches among these systems.

- **Feature Extraction Level**

  As described in section 2.2.1.1, image features include colours, textures, shapes, and edges; colours and textures are the most popular choices. Choice of features, their extraction mechanisms and the search process are domain specific.

  QBIC uses colour, shape, and texture features for image retrieval; CHABOT uses colour and texture features; Photobook uses shape and texture features; KMeD uses quantitative methods for edge detection to identify shape features and ignores colour feature since medical pictures could be black and white; PICTION uses specific facial shape and ignores colour feature to eliminate application constraints.

- **Object Recognition Layer**

  As discussed in section 2.2.1.2, ORL focuses on analyzing features extracted from the feature extraction layer and provides solutions on how to group them. It matches features with object models stored in a knowledge base. The main difference between applications in ORL is the method to build the related knowledge base, Frame based or Constraint based.

  QBIC does not have ORL since it is almost the oldest CBIR application and ORL requires more specific domain knowledge to work effectively. Therefore QBIC, when processing a user query, automatically compares the example sketch with the stored image features without recognizing the content of an image. CHABOT uses the notion of "concept query" where a concept, like sunset, is recognized by analyzing images using colour features. It uses a frame based knowledge representation of image contents, which is precomputed and stored as attributes in a relational data model. For improving the performance of the system, it uses textual annotation of images by keywords that are manually entered. KMeD also use frame based knowledge representation technique while it is more complex in using schema definition, and human-assisted object recognition process since objects may vary due to rigid or non-rigid deformations as discussed in section 2.2.1.2. PICTION uses constraints based knowledge representation, which is a Natural Language Processing (NLP) module. A set of constraints is derived by the NLP module from the picture captions, and will be used with the face recognized in the picture to identify the spatial relationships among people. Photobook uses frame based knowledge representation; however its pre-analysis features limits the application to be a fixed template matching system while the propagating procedure keeps the model updated and more accurate.

- **Spatial Modeling and Knowledge Representation Layer**

  As mentioned in section 2.2.1.3, there are several popular methods in spatial modeling to define spatial concepts. These are semantic networks, inclusion hierarchies, constraints-based and frames-based.

  QBIC, CHABOT, Photobook do not have SMKRL at all. KMeD has a knowledge base abstraction to represent higher level semantics in level three, which is called type abstraction hierarchy (TAH). TAH is organized using semantic networks to conceptualize the objects and their semantics and to incorporate the domain expert knowledge in order to improve search efficiency in radiological database. PICTION uses constraints to organize the recognized objects.

Even a quick glance at these pioneer systems can easily tell us their varieties in technical approach and system design. Though a CBIR system could be examined in general models, there is definitely no rule what a CBIR system should look like. Furthermore, a CBIR system is determined more by its specific domain than by any specific techniques.

Chapter 3

Problem Domain

3.1 The need for CBIR and major issues in system design

Today's computers are not only empowered with networking capabilities, but also equipped with CD-ROMS, scanners, and various other multi-media devices. With the explosion of image data, a big challenge we are facing is the lack of techniques to efficiently handle the needs of interpretation, storage, and retrieval of them. Currently, many organizations possess large archives of photographs but have not made good use of them since they lack efficient tools for search and retrieval. The ideal solution to this problem is the development of automated methods that are able to recognize the composition and the content of the images, and to make use of the recognized image content to achieve content-based image retrieval. A major obstacle to accomplishing this intelligent image database is that research on machine vision has to provide a solution on general image understanding and object recognition; hence we are not yet at the stage to automate image content recognition by computers.

In developing a complete intelligent image database system, at least the following three important issues must be addressed in system design:

Feature Extraction: Determines what image feature, or combination of image features is to be used for image matching and retrieval purposes. Selects algorithms to capture the determined image feature or to combine the determined image features.

Indexing and Search Scheme: Creates effective indices and data structures based on the determined image features to speed up image retrieval on database.

Database Query Schema: Provides methods that enable users to effectively form database queries.

Before describing how these issues are related to our specific problem domain, we would like to clarify two prerequisites.

Prerequisite 1: Image domain

Image domain is the set of images which CBIR systems are designed to handle. Generally, narrow domain and broad domain are known as two subcategories of the image domain [25].

Narrow domain has a limited and predictable variability in all relevant aspects of its appearance [1]. For example, a set of frontal views of faces recorded against a clear background could form a narrow domain for face recognition. Although each face is unique and has large variability in the visual details, there are obvious geometrical, physical, and color-related constraints governing the domain. The domain would be wider if the image is that of a crowd or an outdoor scene. In that case, variations in illumination, clutter in the scene, occlusion, and viewpoint will have a major impact on the analysis. The advantage of using this domain is clear, as semantic description of images is usually easy and well defined with uniqueness. The disadvantage of this approach is that we will have to refine the raw images from current resources.

Broad domain has an unlimited and unpredictable variability in its appearance even for the same semantic meaning [1]. It might be the case that there are conspicuous objects in the scene for which the object class is unknown or even that the interpretation of the scene is not unique. As it is easy to get a broad class of images under this domain from large photo stocks or other photo archives, semantic description of images becomes the most difficult task and hard to achieve.

In a broad image domain, the gap between the feature description and the semantic interpretation is generally wide. For narrow and specialized image domain, the gap between features and their semantic interpretation is usually smaller, so domain-specific models may help. For example, for a face recognition system, such as PICTION, the model is not available for broad image domains as the required number of computational variables would be enormous. More practical applications have an image domain just between these two extreme spectrums. If a CBIR system works well on a narrow image domain, with more work and modifications, it would also work on wider domains.

Therefore, we choose narrow image domain as the default image domain for our CBIR system discussed below.

Prerequisite 2: Definition of similarity

In each CBIR system, it is necessary to define criteria for similarity of images. Generally either syntactical or semantic features could be selected. Syntactical features are extracted from physical representation of images while semantic features must be assigned to images through image understanding by systems, by users and by contributors of images. Choosing syntactical features as the criteria comes from the idea that they are easy to derive from images. However, the shortcoming is obvious that these syntactical features are in some way different from human perception model. On the other hand, choosing semantic features as the criteria is based on human interpretation of images. The disadvantage of this approach is that it is not easy and cost effective to implement. Our choice is to combine syntactical and semantic approach together so as to make a balance between cost and effectiveness.

3.2 Problem scenario

The problem description could be simplified as follows:

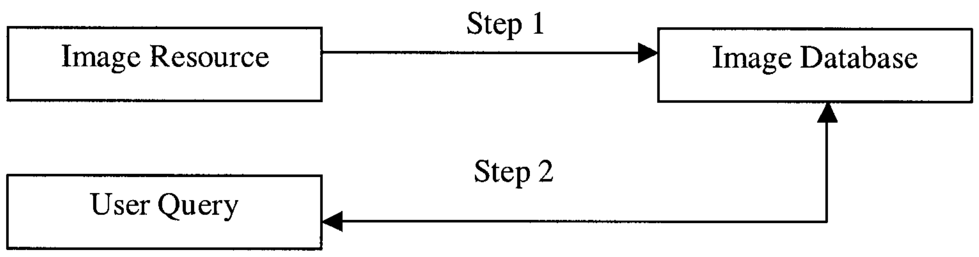


Figure 4: Data flow

Two major steps, which shall be used by all CBIR modules, are:

Step 1:

Objective

Classify images; Create semantic descriptions for each image;

Possible CBIR fundamental modules involved

In order to classify an image, *Content Expression module* is needed with all three layers: Feature Extraction Layer (FEL), Object Recognition Layer (ORL), Spatial Modeling and Knowledge Representation Layer (SMKRL). But how these three layers work together and how this module works may differ from system to system. Since we have already defined two prerequisites in section 3.1, we will only discuss domain-specific models, which means, as a sample application, only several categories of images will be examined (In SHMM, in total, 10 categories are under consideration) and all the similarities are based on a simple combination of syntactic and semantic meaning of images.

Moreover, Index Structure module is needed for future search function in image library. Index content may vary based on the details of images, but the indexing strategy could be the same during the image classification process even though terms used may not be the same.

Also this classification process could be automatic or manual depending on the system need and the accurate rate of classification. A nice solution could be a semi-automatic system. Not only automatic and manual classification is supported, but also during the manual process, the system may suggest the category an image may belong to. A better approach would include a learning process, which trains the classification process when the contributor modifies a classification.

Step 2:

Objective

Through an interface between users and image library, provide similarity search functions.

Possible CBIR fundamental modules involved

User Interface Module (UIM) is the primary component here. No matter what kind of similarity search function it supports, *Query by image examples, Query by keyword or Query by natural language,* the system provides essential introduction information for the operations. For example, in *Query by image examples* mode, random samples from image database shall be provided while in *Query by keyword* mode, keyword information is necessary. Also, since our project is a part of the digital library project, considering the security reason, *User Management* has to be included in UIM.

Chapter 4

SHMM—Semantic Header for MultiMedia

4.1 System diagram

Based on the problem scenario discussed in section 3.2 and with the two prerequisites discussed in section 3.1, we designed our system framework as follows:

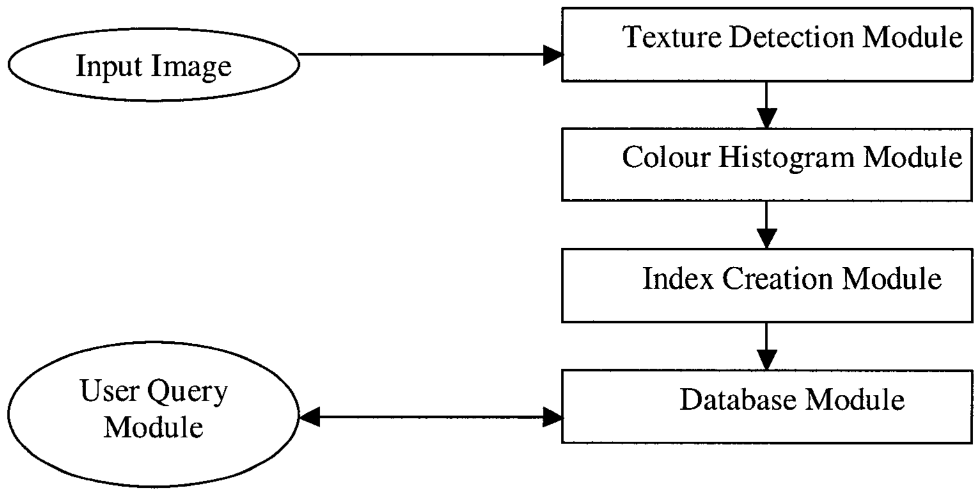


Figure 5: System diagram

This system diagram can be mapped to CBIR fundamental modules as follows. The **texture detection module** and **colour histogram module** serves the same purpose as *Content Expression module* of the CBIR, **index creation** module could be regarded as CBIR's *Index Structure Module* and **User query module** could be mapped to the *User Interface Module.*

There are two categories of input images. For all new non-classified input images, the system will determine image feature vector via **texture detection module** and **color histogram module**, and then this image feature vector will be added to index structure through **index creation module**. Then the semantic description of the image and other input features will be added to image database by the use of **user query module**. For input images that come from organizations or public resources with some semantic classification already done, the main task for adding these pictures into the image database is to map the semantic classification in the resource into semantic descriptions in our database, using traditional data integration techniques. Therefore, in our system, we can only focus on new non-classified images as the original input.

The **Texture Detection Module** scans the input image and searches for texture regions. To make the system easy to implement, we choose a simple algorithm, which will be discussed in section 4.2.1, to form a texture region for an image. After detecting the texture region, the module will take out the part of images inside it and then pass them to the next image module (Colour Histogram Module) for further processing and classification.

The **Colour Histogram Module** takes the texture region masked off from an image as input. Before doing any further analysis, the module will divide the input into 16 segments and these 16 segments will be examined for colour histograms. The output of this module is a 16 dimensional colour vector.

The **Index Creation Module** uses SR tree to store and manage the multidimensional indices for the colour vectors generated from the Texture Detection Module and the Colour Histogram Module. At the same time, it uses the nearest neighbor search to get the most similar picture.

The **Database Module** assigns semantic descriptions for images while maintaining the image database. The input to this module depends on the query involved. For adding new images or classifying images, input comes from the **Index Creation Module,** where the image understandings for new images have been processed and the categories for these images have been suggested, and from the **User Query Module**, where users determine whether to follow the system's suggestion or to change the classification according to the human perception of the image. For retrieving images under the specific category, the input comes only from the **User Query Module**, where user defines the request. In our system, this module is implemented in MICROSOFT ACCESS; however, it can be any other relational database, such as SQL SERVER without any source code and table structure change or any object-oriented database with minor source code change and table structure change.

The **User Query Module** provides the user with a graphical interface, translates the user's database queries into internal index keys and conducts the user feedback to the **Database Module.** *Query by image examples and Query by keyword* functions are supported in our system.

4.2 Algorithms for the modules

Though the modules and functions of how to build one CBIR system have been decided, there are still lots of variations in the algorithms of how to implement them. Both the theoretical and practical reasons are considered when we make the choice. Based on modules, the detailed solutions are as follows:

4.2.1 Texture Detection Module

As discussed in section 4.1, the input for this module is any image, and the output from this module is the part of the original image that lies inside the detected texture regions.

The key element in Texture Detection Module is the texture detection algorithm, which has more than twenty models [1]. The conclusion about textures that we get from the discussion in section 2.2.1 is that we have to make texture detection method be domain-dependent and application-dependent. We perceive the fact that pattern containing high frequency colour transitions causes a significant deterioration for texture detection method; therefore, in our systems, we restrict our image domain to the pictures with such characteristics, for example flowers, buses. That means pictures with dominant colour figures will be more correctly classified than images without dominant colour. We can build a simple but still efficient system that generates one texture region for an image to get the whole idea of what the image looks like. Gong in [5] compares two common techniques called the first and the second order local derivative operators first introduced by Laplace [32]. He concludes that there is no clear evidence that one is better than the other and each solution has its advantages and disadvantages in detecting different type of images. Based on share software CxImage [18], which uses the second order local derivative operator known as Laplacian Filter Kernel [32], we implemented our texture extraction module.

CxImage is a collection of C++ classes to manage virtually any kind of images. It can load, save, display, and transform images in a very simple and fast way. It was developed by Alejandro Aguilar Sierra in 1995. Since then, it has become a shareware and has been expanded by different groups of people. In our installations, we are using the latest version of CxImage 5.53, which was released in January 2003. In our implementation, we do image processing using some of the CxImage functions, refine the picture and pass it to the colour histogram module.

The algorithm is as follows while more details can be found in the appendix:

Step 1: Load the image from storage place, scale the image to a standard size and get the image index for future processing.

To avoid the difference from image quality that is the number of dots in the image, we have to scale the image to a standard size. The texture we get by using CxImage function is a number of dots that describe the general map of what the image looks like. The image index is just like the file handle in the file operation.

Step 2: Get the texture value of the loaded image using Laplacian Filter.

Gonzalez and Woods demonstrated in [37] that Laplacian Filter can be constructed in various ways while they choose

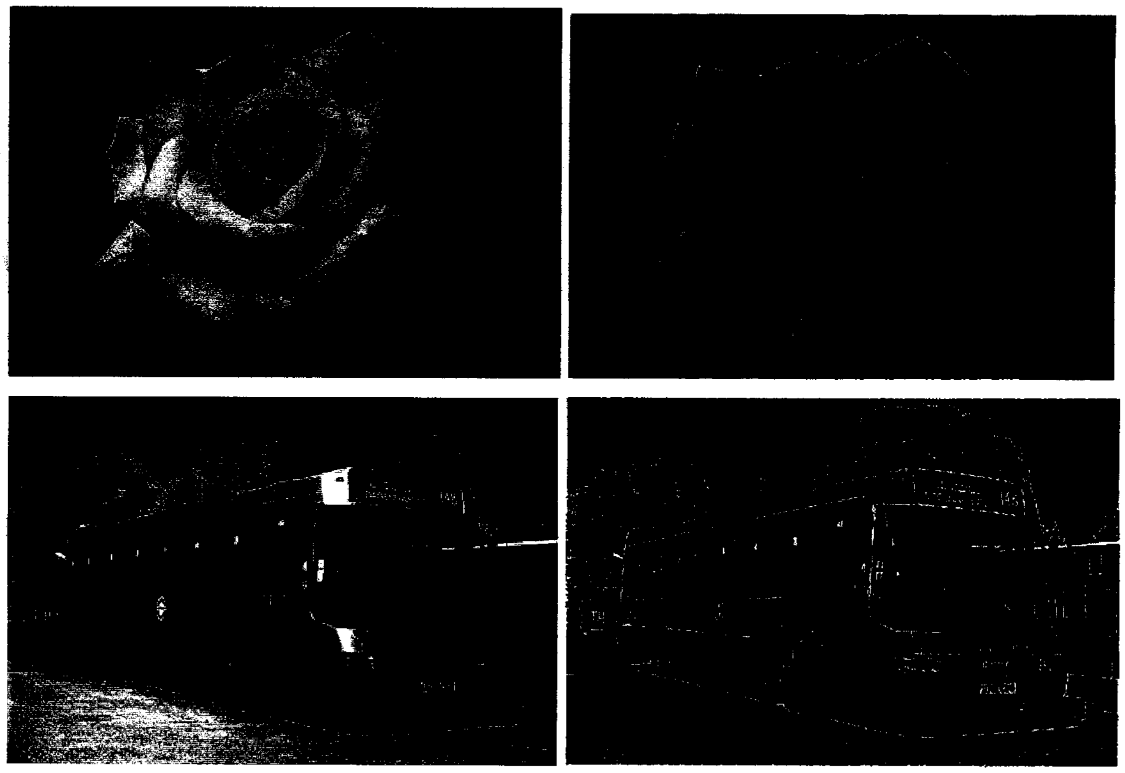
$$\begin{vmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{vmatrix}$$

as a 3-by-3 kernel for edge detection. Therefore in our implementation, we also select this value as the Laplacian Filter Kernel.

Step 3: Get a group of dots that stand for the texture of the image and generate a new image that only contains the texture part of the original picture.

The newly generated image file is a temporary file and will be deleted after getting its colour histogram feature.

In Figure 6, the right hand side (b) represents the texture region of the original image in the left hand side (a). Based on this texture, we divide the image into one texture region and one non-texture region. Afterward, the image inside the texture region will be examined in Colour Histogram Module. This approach we have taken in texture detection module tries to reduce the influence from non-texture region, which sometimes has dominant values in colour.



a) original image b) texture region from original image

Figure 6: Sample result from texture detection module

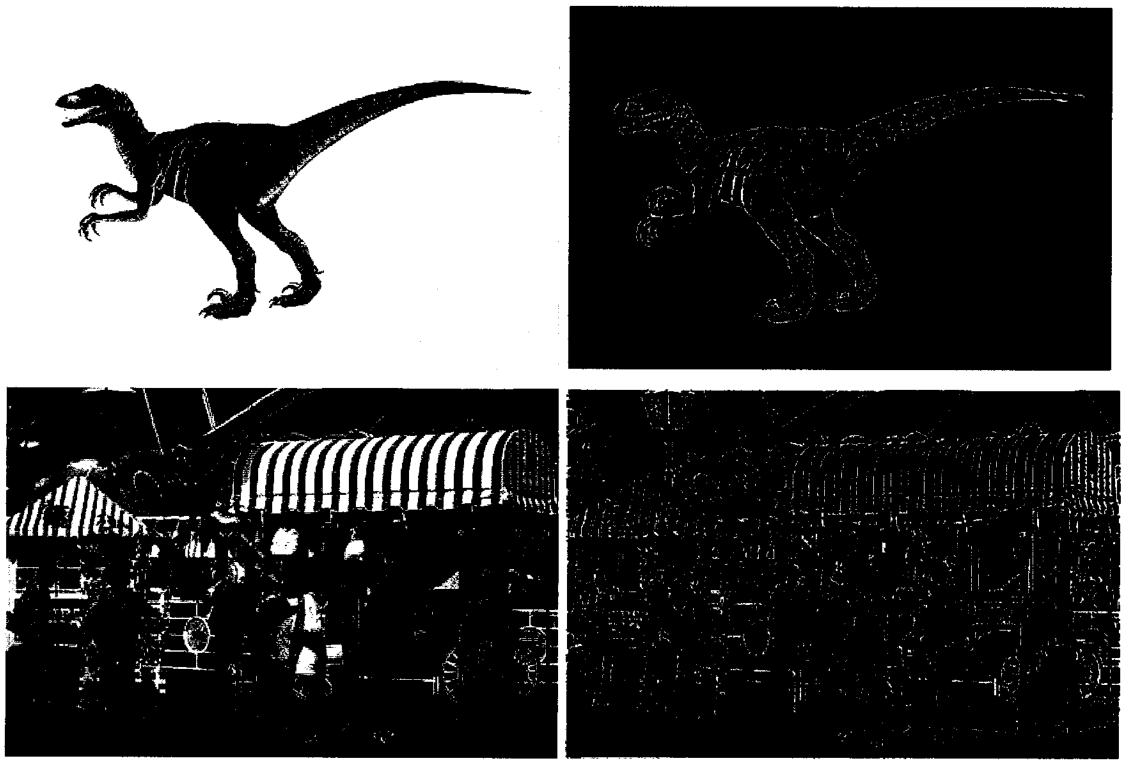

a) original image b) texture region from original image

Figure 6 cont'd: Sample result from texture detection module

4.2.2 Colour Histogram Module

As discussed in section 4.1, the input for this module is the part of the original image that is inside texture regions, and the output from this module is a 16 dimensional colour vector.

Recall the colour theory discussion in section 2.2.1.1, the HVC colour system is well known for its representation of colour along human colour perceptual dimensions; hence here we choose this approach to create colour histograms. When transferring RGB colour into HVC colour and then combining similar HVC colours based on human perception, [5] presents a colour-matching table according to the intensive experimental evaluations. The experimental evaluation of 11-color zone table is as follows.

Table 4: HVC value list from [5]

Color name	**Hue(H)**	**Value(V)**	**Chroma(C)**
Red	0~36	4~9	1.5~30
	36~64	4~9	15~30
Orange	64~112	4~8	9~30
Yellow	80~112	9~10	1.5~30
Skin Color	36~64	4~9	1.5~15
	64~112	4~8	1.5~30
Green	112~196	4~10	1.5~30
Cyan	196~256	6~8	1.5~30
Blue	256~312	4~8	1.5~30
Purple	312~359	4~8	1.5~30
Black		<3	
Gray		4~8	<1.5
		3~4	
White		>9	<1.5

It is really wonderful to have this table after we get HVC values via applying the transferring formula from RGB colour system to HVC colour system. By comparing the transferred HVC values with the sample experimental results, we may easily map the RGB colour world into the real world colour and get the semantic meaning from image colours if adding enough domain knowledge.

However, when we apply this experimental result into our implementation, we find it is hard to achieve this result. Take RGB (159, 2, 23) as an example, which is a traditional red colour. While Applying the HVC exchange formula described in table 1, 2 and 3, we will get the following result, which contradicts the result map in table 4 .

R = 159

G = 2

B = 23

X = 101.484
Y = 51.337
Z = 25.823

L* = 52.022
a* =76.955
b* =29.074

H = 20.697
V = 52.022
C = 82.264

It is obvious that the inaccurate result lays in value V and C. And the more we transfer sample RGB data to HVC colour system, the more we find the contradictions between the results in table 4 and the real data we get from experiment.

Following is a sample RGB colour list from some published materials [32, 33] with the transformation result we get from the formula.

Table 5: Map sample RGB colour into HVC colour system

	X/255	Y/255	Z/255	L*	A*	B*	H	V	C
red	0.176	0.102	0.048	38.1990	46.5875	20.7618	24.0202	38.1990	51.0044
orange	0.386	0.311	0.066	62.5920	25.2955	54.6786	65.1737	62.5920	60.2462
yellow	0.614	0.644	0.112	84.1736	-6.8116	76.3074	95.1010	84.1736	76.6108
dark skin	0.092	0.081	0.058	34.1903	9.3804	9.1174	44.1855	34.1903	13.0813
light skin	0.411	0.376	0.303	67.7248	10.8671	10.0216	42.6823	67.7248	14.7827
green	0.149	0.234	0.106	55.4820	-43.0390	28.5923	146.4025	55.4820	51.6709
cyan	0.149	0.192	0.421	50.9204	-23.3770	-34.5163	235.8913	50.9204	41.6876
blue	0.050	0.035	0.183	21.9444	20.6483	-48.1269	293.2211	21.9444	52.3694
purple	0.059	0.040	0.102	23.6714	23.6522	-25.0475	313.3588	23.6714	34.4501
white	0.981	1.000	1.184	100.0000	-3.1869	-11.5829	254.6162	100.0000	12.0133
black	0.000	0.000	0.000	-16.0000	0.0000	0.0000		-16.0000	0.0000
neutral 8	0.632	0.644	0.763	84.1736	-2.6987	-10.0428	254.9588	84.1736	10.3991

Theoretically, we can prove that table 4 is not accurate.

Recall the transforming formula in table 3;

X = 0.607*Red + 0.174*Green + 0.201*Blue

Y = 0.299*Red + 0.587*Green + 0.114*Blue

Z = 0.000*Red + 0.066*Green + 1.117*Blue

$$L^* = 116*(Y/Yo)^{1/3} - 16$$

$$a^* = 500*((X/X0)^{1/3} - (Y/Y0)^{1/3})$$

$$b^* = 200*((Y/Y0)^{1/3} - (Z/Z0)^{1/3})$$

$$H = \arctan(b^*/a^*)$$

$$V = L^*$$

$$C = (a^{*2}+b^{*2})^{½}$$

where X_0, Y_0, Z_0 are values of a nominally white object-colour stimulus.

Red, Green, Blue value for white is (255,255,255), giving $X_0Y_0Z_0$ for white as (250.41, 255, 301.66).

We will prove using the method of reduction to absurdity.

In table 4, V values for natural red, orange, etc. are between 4 and 10.

That is $4 \leq 116*(Y/Yo)^{1/3} - 16 \leq 10$ (1)

From (1), and the value of Y_0, we get $1.3069 \leq Y \leq 2.8714$ (2)

That is $1.3069 \leq 0.299*Red + 0.587*Green + 0.114*Blue \leq 2.8714$ (3)

$Y = 0.299*Red + 0.587*Green + 0.114*Blue$

$= 0.114*(Red + Green + Blue) + 0.185*(Red + Green) + 0.288*Green$ (4)

$\geq 0.114*(Red + Green + Blue)$ (5)

from (3),(5) , we will get $0.114*(Red + Green + Blue) \leq 2.8714$ (6)

that is $(Red + Green + Blue) \leq 25.1877$ (7)

In RGB colour space, (7) cannot always true. Therefore, we proved that V value range in table 4 is not correct.

Similarly, we can prove that C value range in table 4 is also not correct.

Though the results we get are different from [5], table 4 still has its values:

1. For RGB colour value, H value is accurate in table 4. Therefore, we can still use H value of a colour to justify the colour family it belongs to. Also V value varies when two colours are quite similar and belong to the same colour family. Since C value represents the purity of a colour, it does not contribute a lot to justify the colour family. Therefore, we neglect C value in our implementation.
2. For black, gray and white, they don't have special Hue value feature; therefore, we have to use other way to recognize them. However, thinking of an image from the real world, gray is the colour between white and black and usually these three colours are background colours. And after we refine images with textures, it is possible that we can treat black, white and grey as the same colour group.

Therefore, based on experimental result and analysis, we formed our new colour range as shown in table 6

Table 6: Modified HVC value list

Color name	**Hue(H)**	**Value(V)**
Red	0~64	
Orange	64~112	
Yellow	80~112	
Skin Color	36~64	
	64~112	
Green	112~196	
Cyan	196~256	
Blue	256~312	
Purple	312~359	
Black,Gray,White		>90 <10

Another new approach in our histogram generation module is the way that we group our colour data. [5] takes the pixel counts in the 11-colour zones as the elements in the feature vector formed from histogram generation module. It considered the importance of the number of pixels in the similarity of two images, but it did not take the distribution of these pixels into account. For example, 2 images might have quite similar number of pixels in all 11-colour zones while the sequence of how these colours appear is different. In human understanding, these two images are hard to be viewed as similar in colour; however when using this approach in histogram module, these two images are considered similar. To avoid this shortcoming, we divide the image into 16 segments, and count the dominant colour in each segment. Afterward, we group these 16 dominant colour values in a sequence which has been predefined as from left to right and from top to bottom.

The algorithm details are as follows:

Step 1: Scale any input image to a standard size. With our experimental pictures, we choose 256*256 pixel size of. Though we scale an input image into a standard size, it does not influence the physical storage of the image. Another issue here is the colour loss. There are two kinds of possible change in image pixel size when we scale an input image to a standard size. One is "shrink". While the pixel size of the input image is larger than 256*256, colour loss is inevitable since not only some of the pixels will be lost but also the pixel colour will be modified to match the human perspective. Another operation is "expand". Traditional approach in this operation is to split rows and columns inside an image. Since the input file in this module is an output processed by texture detection module, we can not change the texture that was derived. Therefore, in our implementation, we fix the input image, and add background blocks to meet the size change of pixels.

Step 2: Divide the image into 16 equal segments. Each segment with a pixel size of 64*64 will be processed from Step 3 to Step 5.

Step 3: Get RGB colour for each pixel in a segment that has a pixel size of 64*64. The return type of the colour is a RGBQUARD structure.

Step 4: Transform RGB colour value to HVC colour value applying formulas in table3. While computing H value, to keep its value range from 0 to 360, following formula is applied :

If $a^*>0$ and $b^*>0$	$H = \mathrm{actan}(b^*/a^*)$
If $a^*>0$ and $b^*<0$	$H = 360 + \mathrm{actan}(b^*/a^*)$
If $a^*<0$ and $b^*>0$	$H = 180 + \mathrm{actan}(b^*/a^*)$
If $a^*<0$ and $b^*<0$	$H = 270 + \mathrm{actan}(b^*/a^*)$

Here $a^* =500*((X/X0)^{1/3}-(Y/Y0)^{1/3})$, $b^* =200*((Y/Y0)^{1/3}-(Z/Z0)^{1/3})$(more details in table 3)

Step 5: Compare HVC value with the colour range value in table 6; count the pixel number for each colour and choose one dominant colour for this 64*64 image part. Colour with the largest pixel count is regarded as the dominant colour. Use $(H)^2*V/10000$ as the representation value to represent those dots belonging to dominant colour. Generate the final result with the mathematical average of these values

Step 6: Combine 16 chosen colours into a 16 dimensional colour vector which represents the colour histogram for the image; write this record out into a file that will become an input for index creation module.

In figure 7 the first number represents the number of vector that is used to describe an image. The 16 numbers, separated by a colon (:), are the colour histogram for the image with the image name given at the end of the record.

```
16
0.001735:0.001146:0.000807:0.000409:0.000147:0.000022:0.000000:0.000000:0.00000
0:0.000000:0.000015:0.000137:0.000385:0.000787:0.001125:0.001694:(600.jpg)
0.002212:0.001614:0.001353:0.001185:0.000997:0.000721:0.000462:0.000335:0.00032
9:0.000445:0.000711:0.000974:0.001179:0.001338:0.001594:0.002154:(301.jpg)
0.001961:0.001683:0.001462:0.001138:0.000900:0.000625:0.000358:0.000154:0.00013
8:0.000348:0.000610:0.000878:0.001121:0.001449:0.001671:0.001924:(400.jpg)
0.002327:0.001875:0.001662:0.001475:0.001280:0.001036:0.000729:0.000467:0.00046
0:0.000710:0.001021:0.001263:0.001468:0.001647:0.001858:0.002283:(100.jpg)
```

Figure 7: Colour histogram for Figure 6 from colour histogram module

4.2.3 Index Creation module

As described in section 4.1, the input for this module is a 16 dimensional colour vector, which is given in Figure 7, and the output from this module is the updated index file together with the most similar image as the result of the nearest neighbor search. The input file is a .txt file while the index file is a binary file. SR tree is our choice for index structure with its advantage having been discussed in section 2.2.2.

In [10], Katayama gives a detailed discussion about how to construct SR tree while at the same time presenting related algorithms. Part of the source code can be downloaded from http://research.nii.ac.jp/~katayama/homepage/research/srtree/. (The program has also been archived at http://www.cs.concordia.ca/~bcdesai/grads/bh_zhang/srtree.zip)The detailed algorithms of how we implement our SR tree based on [10] are given below:

Configuration of an SR-tree's nodes

Notation: m and M are the minimum and maximum number of entries in a node.

Leaf node: A leaf node contains between m and M entries of the form (*p, data*), where *data* holds the data of the leaf, and *p* is the pointer to the *data*.

$$\begin{array}{ll} L : (E_1, \ldots, E_n) & (m_L \leq n \leq M_L) \\ E_i : (\boldsymbol{p},\ \boldsymbol{data}) & \end{array}$$

Non-leaf nodes: A non-leaf node contains between m and M entries of the form (*S, R, w, np), where np* is the pointer to a child node, *w* is the total number of data entries stored in the sub tree pointed to by *np*, *S* and *R* are the minimum bounding sphere and minimum bounding rectangle of the child node, respectively.

$$\begin{array}{ll} N : (C_1, \ldots, C_n) & (m_N \leq n \leq M_N) \\ C_i : (\boldsymbol{S}, \boldsymbol{R}, w, \mathit{child\_pointer}) & \end{array}$$

In our implementation, M is selected as 10 while m is 6. Actually we may change these two parameters and can still get the similar experimental result in time consumption and processing space, as reported by Katayama in [10].

Operations on SR tree

SplitNode (quadratic algorithm)

SR tree takes SplitNode algorithm from R tree [10] since SS tree will reset a portion of its entries rather than split it [9]. E1, E2 represent the leaf node for SR tree, and E1.rect represents the bounding rectangle for E1. The data structure for R tree can be found in section 2.2.2.

Step 1: Compose a rectangle R enclosing E1.*rect* and E2.*rect* for each pair of entries E1 and E2. Calculate d = area(R) – area (E1.*rect*) – area (E2.*rect*).

Step 2: Choose the pair with the largest d to be the first elements of the two subspaces.

Step 3: For each entry E not yet in any subspace, calculate d1 = the area enlargement of the covering rectangle of subspace1 for enclosing E.*rect*. Calculate d2 similarly for subspace 2. Choose the entry with the maximum difference between d1 and d2 as the next entry to be distributed.

Step 4: Add the entry selected in Step 3 to the subspace whose covering rectangle will have to be enlarged least to accommodate it. Resolve ties by adding the entry to the subspace with the smaller area, then to the one with the fewer entries.

Step 5: Repeat Step 3 and Step 4 until all the entries are distributed, or one of the two subspaces has M - m + 1 entries.

Search

Suppose T is the root node of an SR tree and S is the search rectangle specified by the user.

Step 1: if T is not a leaf, find all the entries E whose rectangle E.*rect* overlaps S. For each overlapping entry E, set T = E.*np*, and perform this operation recursively.

Step 2: If T is a leaf, find all the entries E whose rectangle E.*rect* overlaps S and retrieve the record that is pointed to by E.*id* from the database.

Insertion

Suppose D is the new entry to be inserted into SR tree.

Step 1: Set N to be the root node.

Step 2: If N is not a leaf, find the entry E in N whose rectangle E.*rect* needs least enlargement to include D.*rect*. Resolve ties by choosing the entry with the smallest rectangle and also the smallest bounding sphere. Set N to be the child node pointed to by E.*np*, and repeat step 2.

Step 3: if N is a leaf, and N has room for another entry, insert D in N. Otherwise, invoke SplitNode to obtain N and NN with contain D and all the old entries in N.

Step 4: Let P be the parent node of N, and let $E^{(N)}$ be N's entry in P. Adjust rectangle $E^{(N)}$.*rect* to enclose rectangles of all the entries in N. Adjust new bounding sphere with the following format.

Let d be the dimension of the data space, $\mathbf{x} = \{Xi \mid I=1,\ldots, d\}$ and r be the center and the radius of the new bounding sphere, respectively, $E_k(k=1,\ldots,n)$ be the k'th entry in the child node, $E_k.X_i$ and $E_k.r$ be the center and radius of entry E_k's bounding sphere, respectively, $E_k.R$ be entry E_k's bounding rectangle, and $E_k.w$ be the total number of data entries stored in the sub tree pointed to by E_k.

$X_i = (\sum_{k=1}(E_k.X_i * E_k.w)) / \sum_{k=1} E_k.w$

$r = \min(d_s, d_r)$

$d_s = \max(\|x-E_k.x\|+E_k.r),\ 1 \le k \le n$

$d_r = \max(\mathrm{MAXDIST}(x, E_k.R)),\ 1 \le k \le n$

MAXDIST(p, R) is the longest distance from point p to rectangle R.

Step 5: If NN is created as a result of a split, create a new entry $E^{(NN)}$ with $E^{(NN)}$.*np* pointing to NN and $E^{(NN)}$.*rect* enclosing all the rectangles in NN. Add $E^{(NN)}$ to P if there is room. Otherwise, invoke SplitNode to produce P and PP containing $E^{(NN)}$ and all old entries in P.

Step 6: Set N=P and NN=PP if a split occurs. If N is the root node, then go to Step 7. Otherwise, repeat from Step 4.

Step 7: If node split propagation causes overflow in the root, create a new root whose children are the two nodes obtained by splitting the old root node.

Deletion

Suppose D is the entry to be removed from the database.

Step 1: Set search range S = D.*rect,* and invoke the search algorithm to find leaf L containing the entry that matches D. Stop if the entry is not found.

Step 2: Remove D from L.

Step 3: Set N = L, and initialize the temporary working area Q to be empty.

Step 4: If N is the root, go to Step 8. Otherwise, let P be the parent of N, and $E^{(N)}$ be N's entry in P.

Step 5: if N has fewer than m entries, delete $E^{(N)}$ from P and store all the entries in N in Q.

Step 6: Otherwise, adjust $E^{(N)}$.*rect* to tightly enclose all the entry rectangles in N. Adjust new bounding sphere with the following format.

Let d be the dimension of the data space, $\mathbf{x} = \{Xi \mid I=1,\ldots, d\}$ and r be the center and the radius of the new bounding sphere, respectively, $E_k(k = 1,\ldots,n)$ be the k'th entry in the child node, $E_k.X_i$ and $E_k.r$ be the center and radius of entry E_k's bounding sphere, respectively, $E_k.R$ be entry E_k's bounding rectangle, and $E_k.w$ be the total number of data entries stored in the sub tree pointed to by E_k.

$X_i = (\sum_{k=1}(E_k.X_i * E_k.w)) / \sum_{k=1} E_k.w$

$r = \min(d_s, d_r)$

$d_s = \max(||x - E_k.x|| + E_k.r), 1 \le k \le n$

$d_r = \max(\text{MAXDIST}(x, E_k.R)), 1 \le k \le n$

MAXDIST(p, R) is the longest distance from point p to rectangle R.

Step 7: Set N = P and repeat from Step 4.

Step 8: Re-install all the entries stored in Q. The principle of the re-insertion is that an entry must be re-inserted into the same level as it was deleted from.

In our implementation, the input file containing the colour vector values is a .txt file and in a format as demonstrated in Figure 7. The first line in the file describes how many dimensions the index will have. In our system, we use 16.

4.2.4 Database Module

As described in section 4.1, this module connects to **Index Creation Module** and **User Query Module** and could take outputs from both modules as input. The main task in this module is to assign semantic descriptions for images while dynamically maintaining its own information. Though we implement this module in MICROSOFT ACCESS, other relational or object oriented database could be the choice.

Database structure

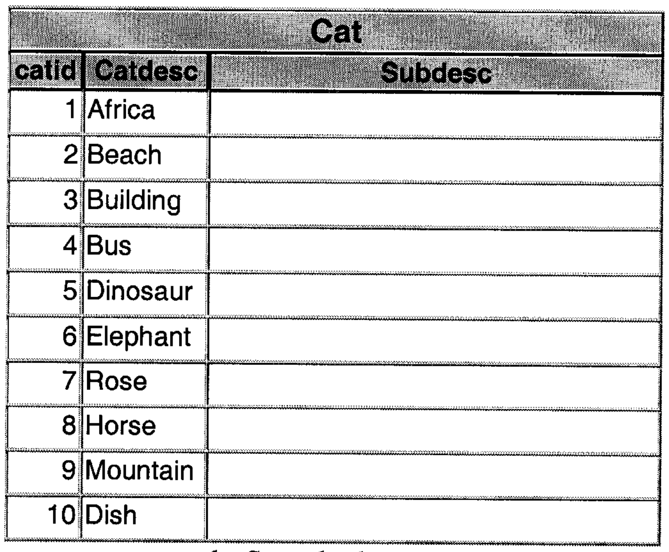
There is only one level of semantic description in the sample database that has been implemented. The system can be expanded easily following the logical relationship in the current model.

Semantic header description: All semantic headers are kept as records in relation "*cat*". The table structure is flexible in expanding and hence we may give subdivision and semantic headers for each category. This expansion is extremely useful when users ask for more detailed classification suggestions. Also, when the user gives feedback of discovering a new category, a new record will be inserted to the table.

Table 7: SHMM-Relation *cat*

catid	AutoNumber	Semantic category ID
catdesc	Text	Semantic category description
subdesc	Text	Description for subcategory

a: Relation Name: cat



Cat		
catid	Catdesc	Subdesc
1	Africa	
2	Beach	
3	Building	
4	Bus	
5	Dinosaur	
6	Elephant	
7	Rose	
8	Horse	
9	Mountain	
10	Dish	

b: Sample data entry

In our test database, 1000 sample pictures are classified into 10 categories with the detailed information in table 7 b.

Image description: Each image has a unique record in the relation "*pic*". Since these records have no relationship with the procedure of CBIR, it doesn't matter how huge it might be. The main usage of this table is to use *picid*, which is generated automatically by the database system, to uniquely identify an image. Picture name are given by contributors and could be the same since different contributors may use the same name to describe different images. Also this table will hold more specific features for images.

Table 8: SHMM-Relation *pic*

Picid	AutoNumber	picture ID
Picname	Text	picture name
Picdesc	Text	picture description
picdate	Text	picture shooting date
photographer	Text	Photographer
contributor	Text	contributor of the picture
picplace	Text	picture shooting place

a: Relation Name: pic

Pic						
picid	picname	picdesc	picdate	Photographer	contributor	Picplace
1	1.jpg	people			user	

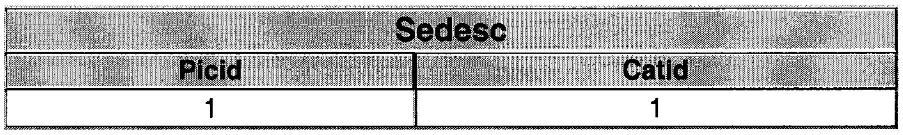
b: Sample data entry for pic

Semantic description for image: The relationship between images and semantic descriptions is maintained as records in relation "*sedesc*". Since there is only one level of semantic description known as category in the sample implementation and there is no overlap between categories, currently the combination of *picid* and *catid* is unique. When expanding the system to at least two level semantic descriptions, the structure might be changed to add more subcategory id in this relation. Another approach is to add a new table to describe directly the relationship between picture and subcategory. The former approach reduces storage space while wasting time in reading irrelevant information if we only care about the subcategory. The latter approach instead, provides efficiency in query process while wasting storage space. This table is useful when suggesting image categories during the image classification process.

Table 9: SHMM-Relation *sedesc*

Picid	Number	picture ID
Catid	Number	category ID

a: Relation Name: sedesc



Sedesc	
Picid	Catid
1	1

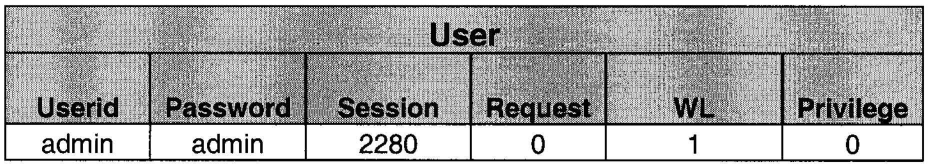
b: Sample data entry for sedesc

User Management: User information and requests are kept as records in relation "*user*". Not only does this table protect system security, but also it determines the function modules users can view in SHMM. Also the session and request fields will play an important role in the user query module.

Table 10: SHMM-Relation *user*

Userid	Text	User ID
Password	Text	Password
Session	Text	Session
Request	Number	Request result or not
WL	Number	Whether in waiting list
Privilege	Number	Viewing Privilege

a: Relation Name: user



User					
Userid	Password	Session	Request	WL	Privilege
admin	admin	2280	0	1	0

b: Sample data entry for user

4.2.5 User Query Module

As discussed in section 4.1, this module provides the user with a graphical interface, translates the user's database queries into internal index keys and conducts the user feedback to the **Database Module.**

In web applications, we seldom use files to transfer information between two pages. However in our SHMM system, there are some backend programs such as index creation program (dynamicbuild.exe) and search for nearest neighbor program (fecthNN.exe).

These programs take a specific file as input and another file as output; therefore, in our implementations, we have to consider the conflicts that the file sharing mechanism brings to the multi-request web application.

The solution is to use relation USER to manipulate the user request. In relation USER, only one record could have value "1" in field REQUEST, while other records have value "0". When running backend programs, all requests with REQUEST value "0 in relation USER" have to wait. After finishing processing one request, the system will choose the smallest session ID to respond to another request. Figure 8 explains how the User Query module handles requests generally.

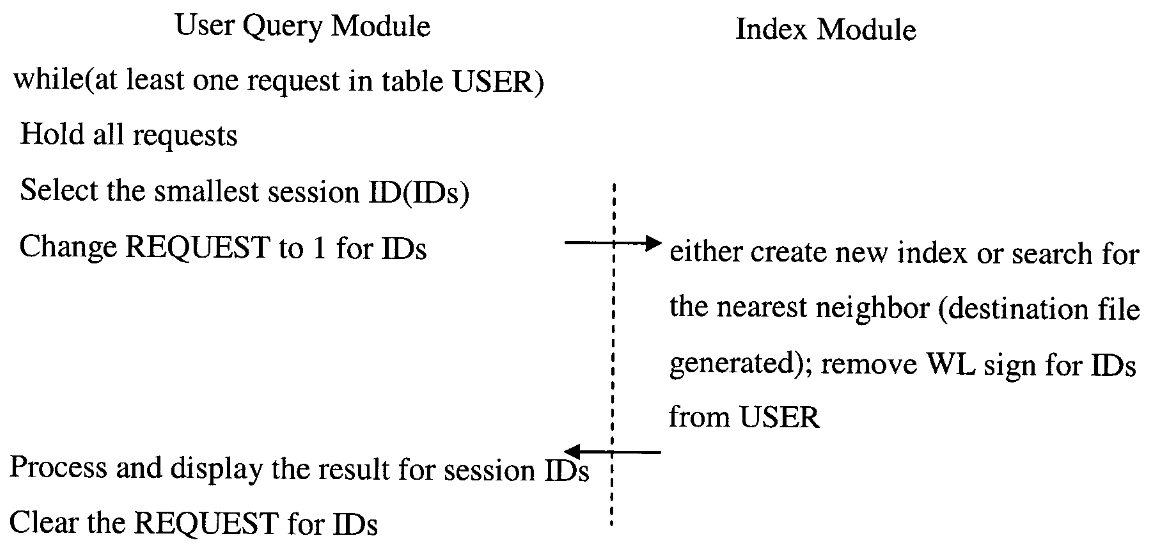
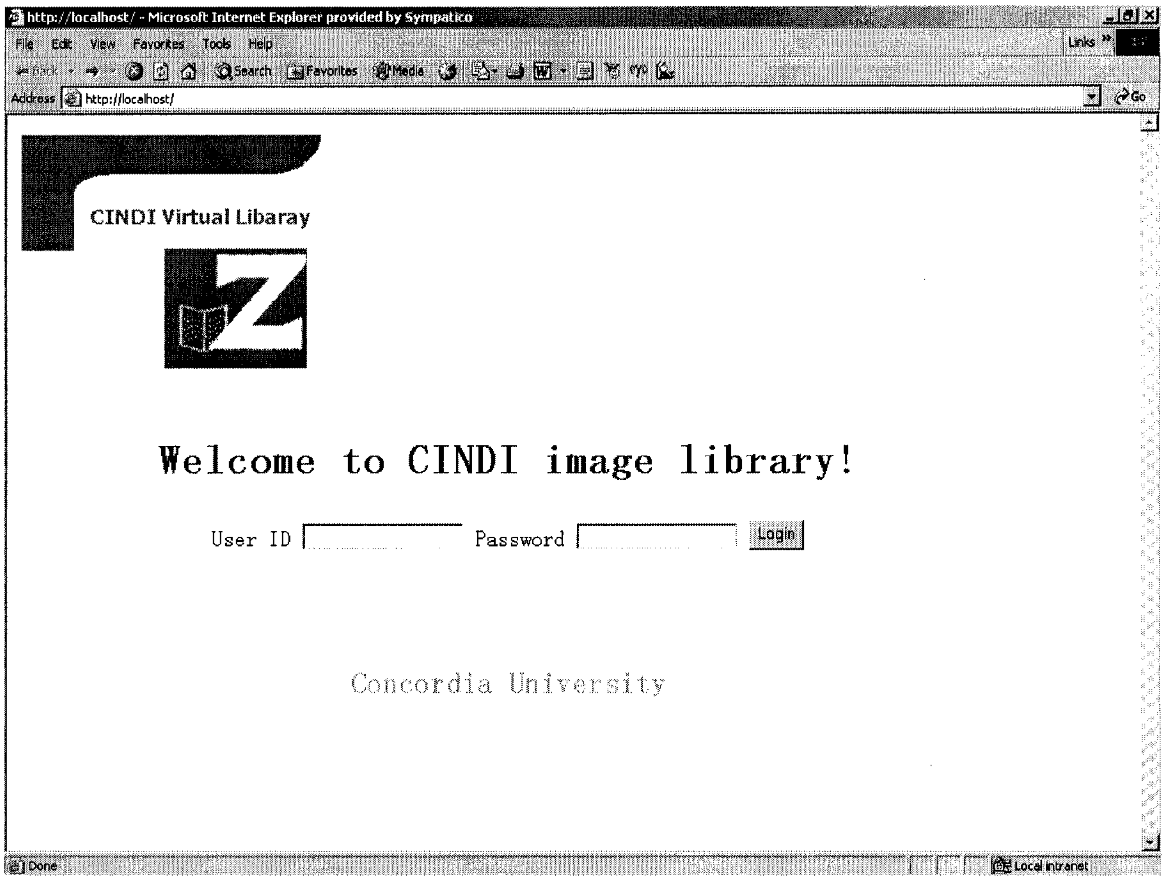

User Query Module
Index Module
while(at least one request in table USER)
Hold all requests
Select the smallest session ID(IDs)
Change REQUEST to 1 for IDs
either create new index or search for the nearest neighbor (destination file generated); remove WL sign for IDs from USER
Process and display the result for session IDs
Clear the REQUEST for IDs

Figure 8: How User Query Module handle multiple requests

The main screen for SHMM is as follows:


http://localhost/ - Microsoft Internet Explorer provided by Sympatico
File Edit View Favorites Tools Help
Address http://localhost/
CINDI Virtual Libaray
Done
Local intranet

Welcome to CINDI image library!

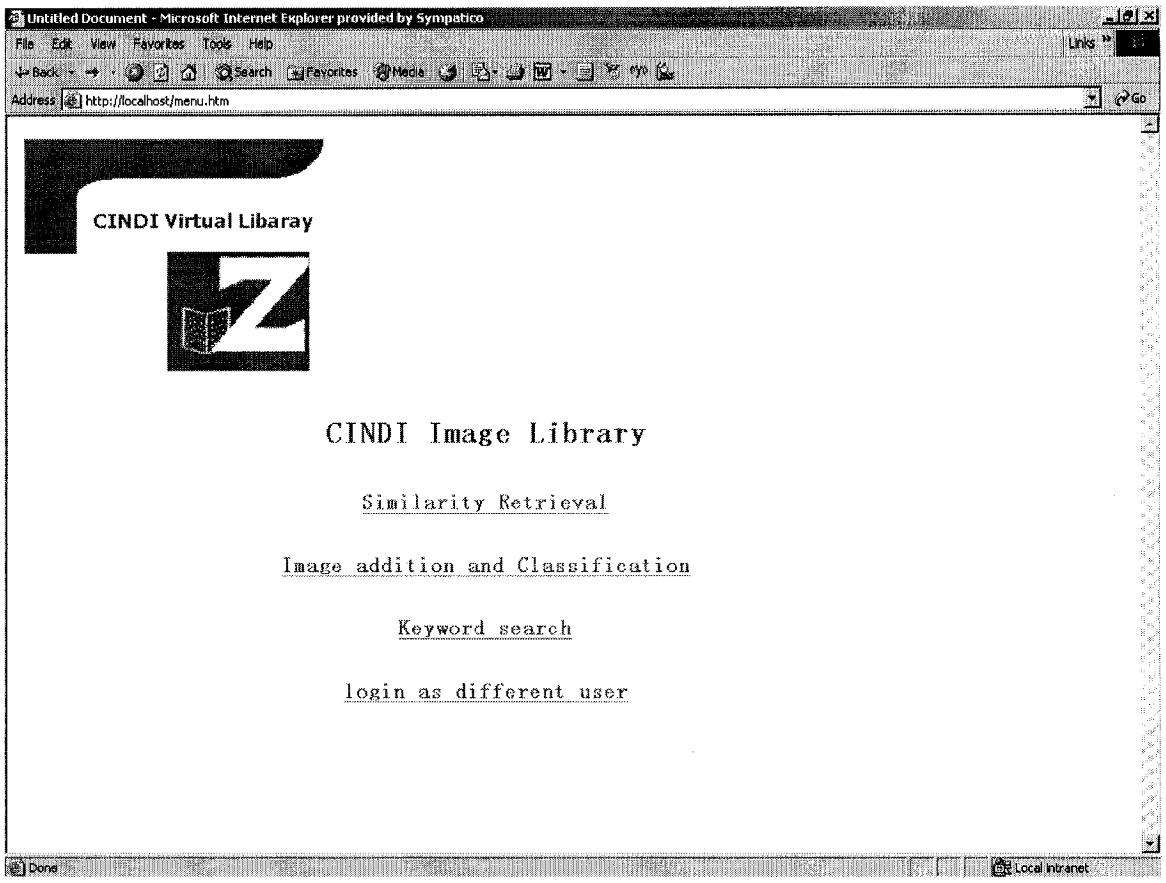
User ID Password Login

Concordia University

Figure 9: SHMM-User Login

Currently two kinds of users are supported in SHMM: system administrator and registered user.

After inputting correct user id and password, main menu will be seen as Figure 10:


Untitled Document - Microsoft Internet Explorer provided by Sympatico
File Edit View Favorites Tools Help
Address http://localhost/menu.htm
CINDI Virtual Libaray

CINDI Image Library

Similarity Retrieval

Image addition and Classification

Keyword search

login as different user

Figure 10: SHMM-Main Menu

In the main menu page, users can select similarity retrieval, image addition and classification, keyword search or logout and login as different user.

Clicking "**Similarity Retrieval**", the interface is as follows:

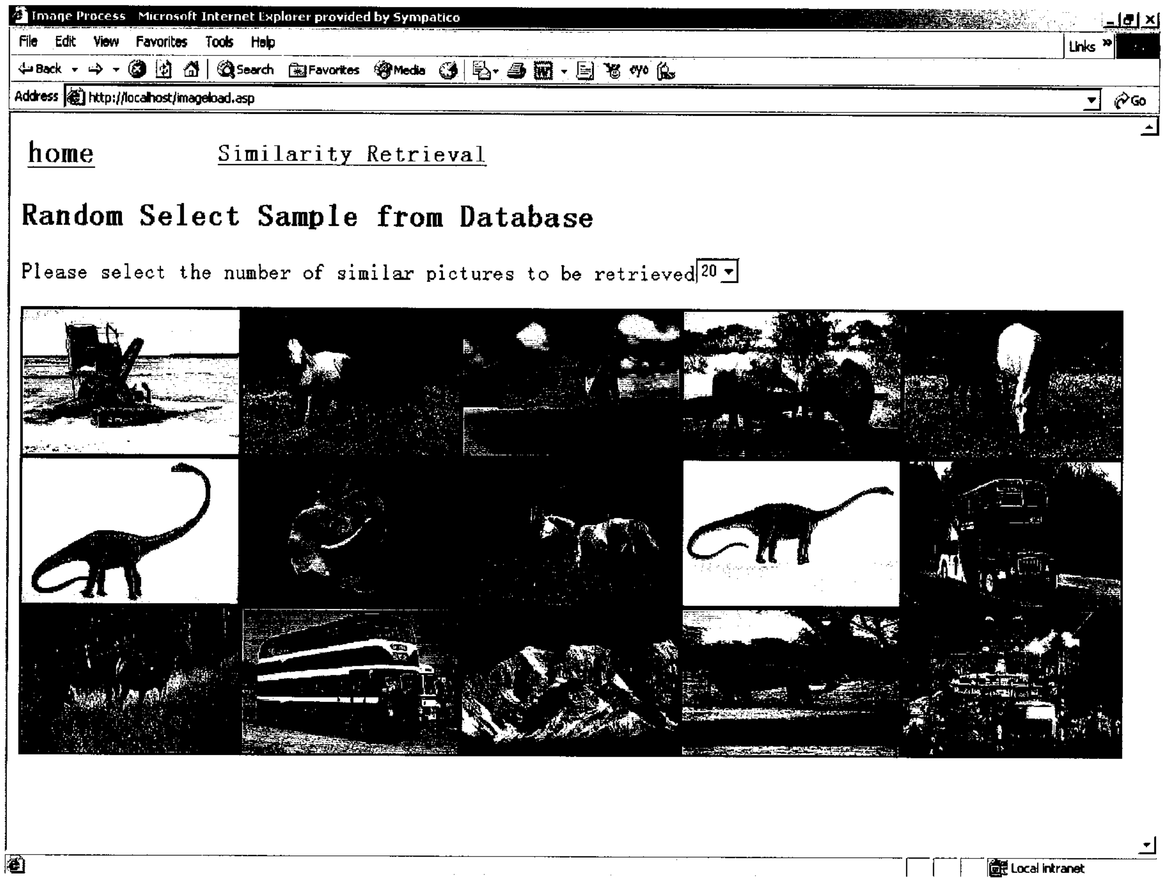


Image Process - Microsoft Internet Explorer provided by Sympatico
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/imageload.asp
Go
Links
Local intranet

home Similarity Retrieval

Random Select Sample from Database

Please select the number of similar pictures to be retrieved 20

Figure 11: SHMM-Similarity Retrieval 1

This interface provides *Query by image examples* function where sample images are randomly selected from the database. Before clicking the selected sample image, users can change the number of results the similarity search might return while the default value in the system is 20.

Once the selected image is clicked, assuming it to be the image with the yellow rose in Figure 11, the result will be given in Figure 12:

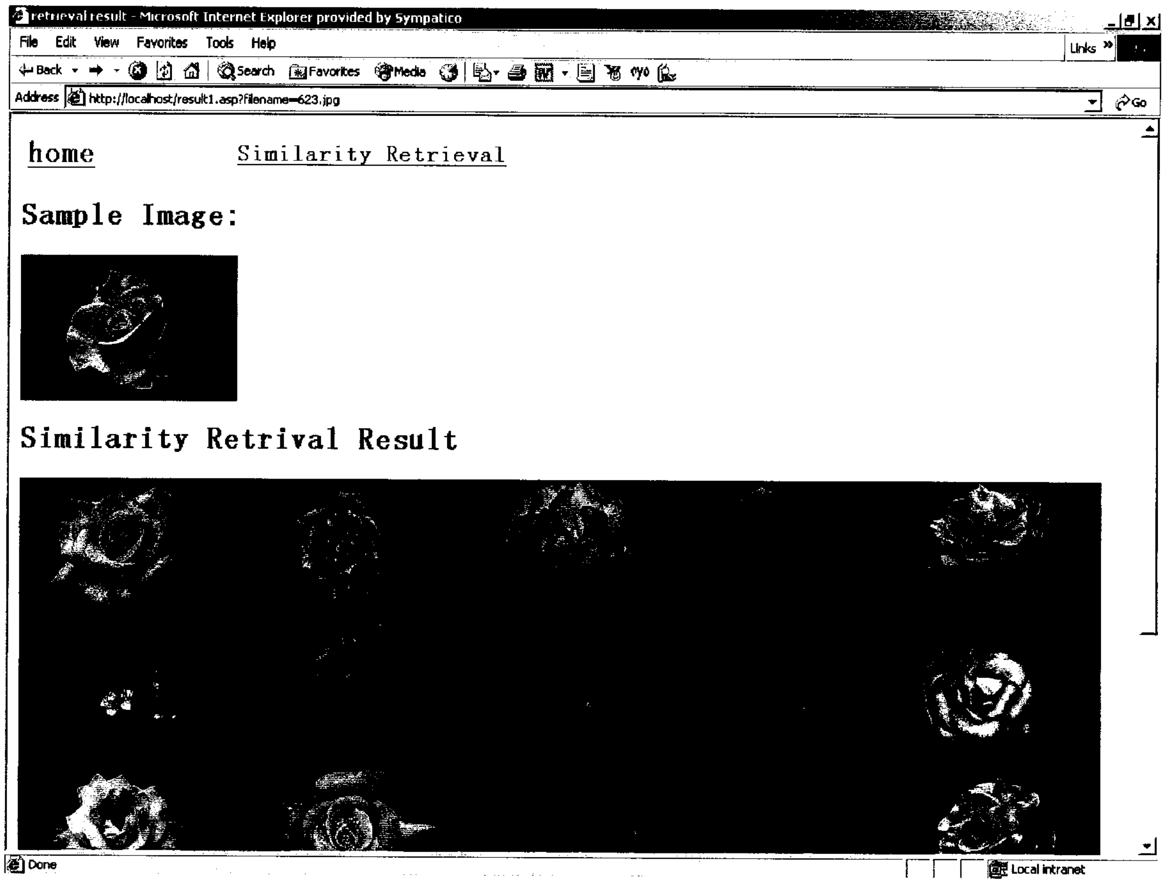

retrieval result - Microsoft Internet Explorer provided by Sympatico
File Edit View Favorites Tools Help
Address http://localhost/result1.asp?filename=623.jpg
home
Similarity Retrieval
Sample Image:
Similarity Retrival Result
Done
Local intranet

Figure 12: SHMM-Similarity Retrieval 2

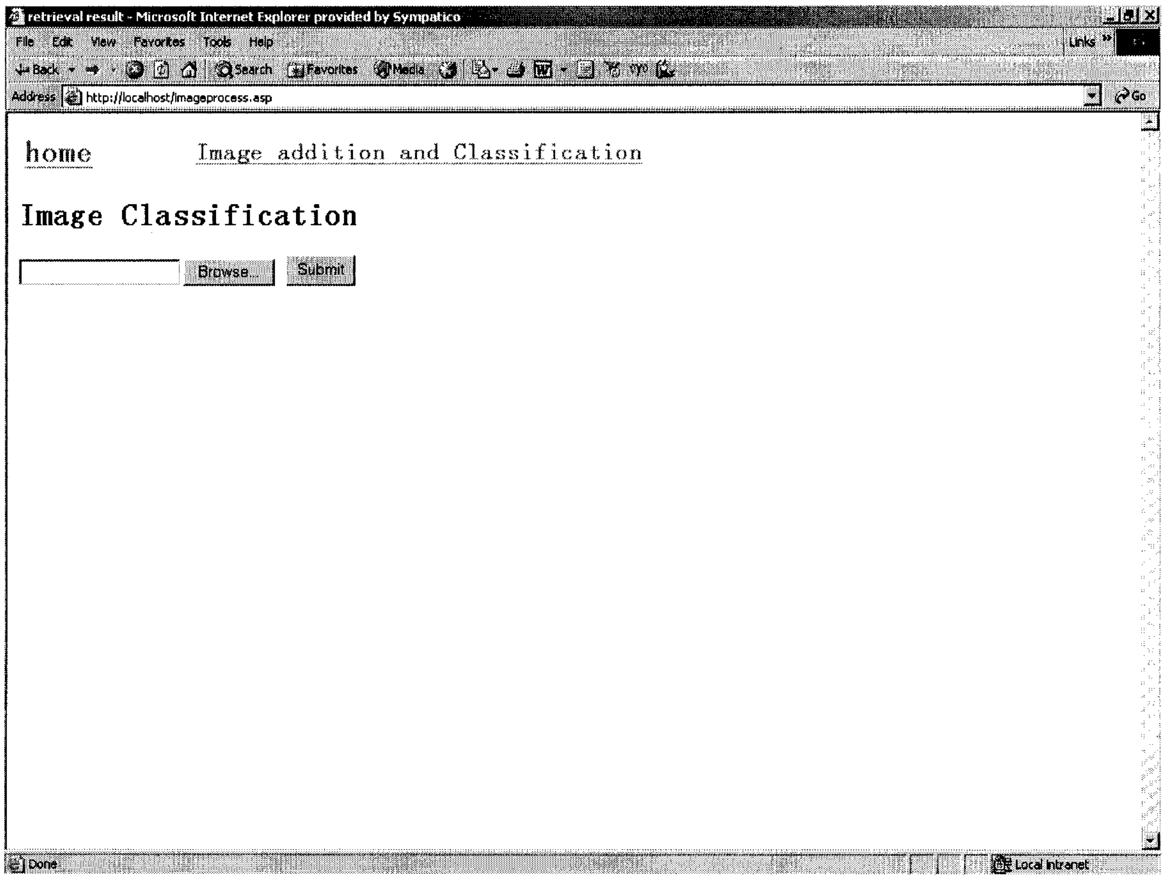
The process in both front and back ends of the system includes:

Step1 Get the picture id and the number of returned retrieved images.

Step2 Pass the variable value in Step1 to Index Creation Module, and use the nearest neighbor search to get the query result.

Step 3 Apply the result getting in Step 2 to Database Module, and show the result in the user query module.

The main screen for "**Image addition and Classification**" is as follows:


retrieval result - Microsoft Internet Explorer provided by Sympatico
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/imageprocess.asp
Go
Links
Done
Local intranet

home Image addition and Classification

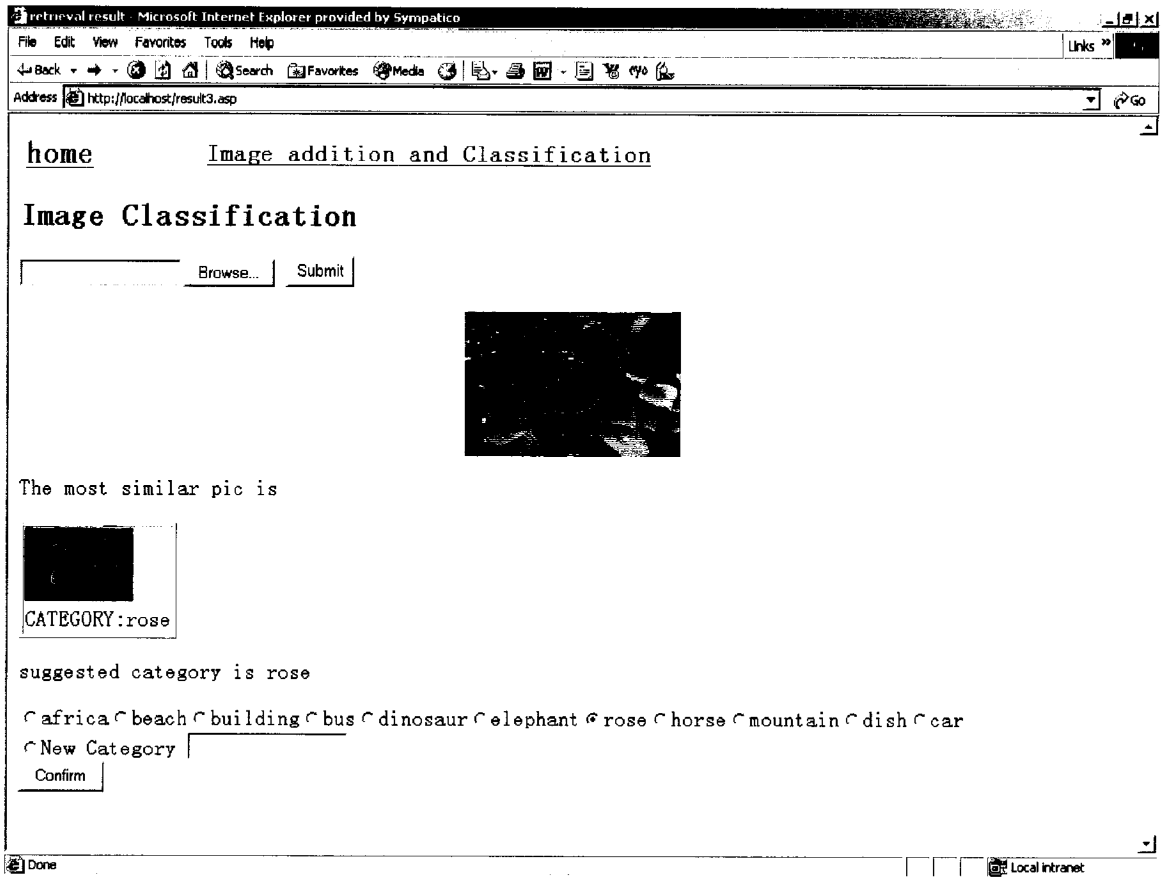
Image Classification

Figure 13: SHMM-Image addition and Classification 1

The classification process procedure includes:

Step 1: Select the image contributor /user would like to add. Use "browse" button to find the picture and use "submit" button to confirm. Currently, the system only supports image files with .jpg format.

For example, we try to add an image of a rose to the database. After clicking the "submit" button, we can see Figure 14 as follows. There is a similar interface for "Query by Example" to upload an image for which you are looking for a similar image.


retrieval result - Microsoft Internet Explorer provided by Sympatico

Address http://localhost/result3.asp

home Image addition and Classification

Image Classification

Browse... Submit

The most similar pic is

CATEGORY:rose

suggested category is rose

africa beach building bus dinosaur elephant [✓ rose] horse mountain dish car

New Category

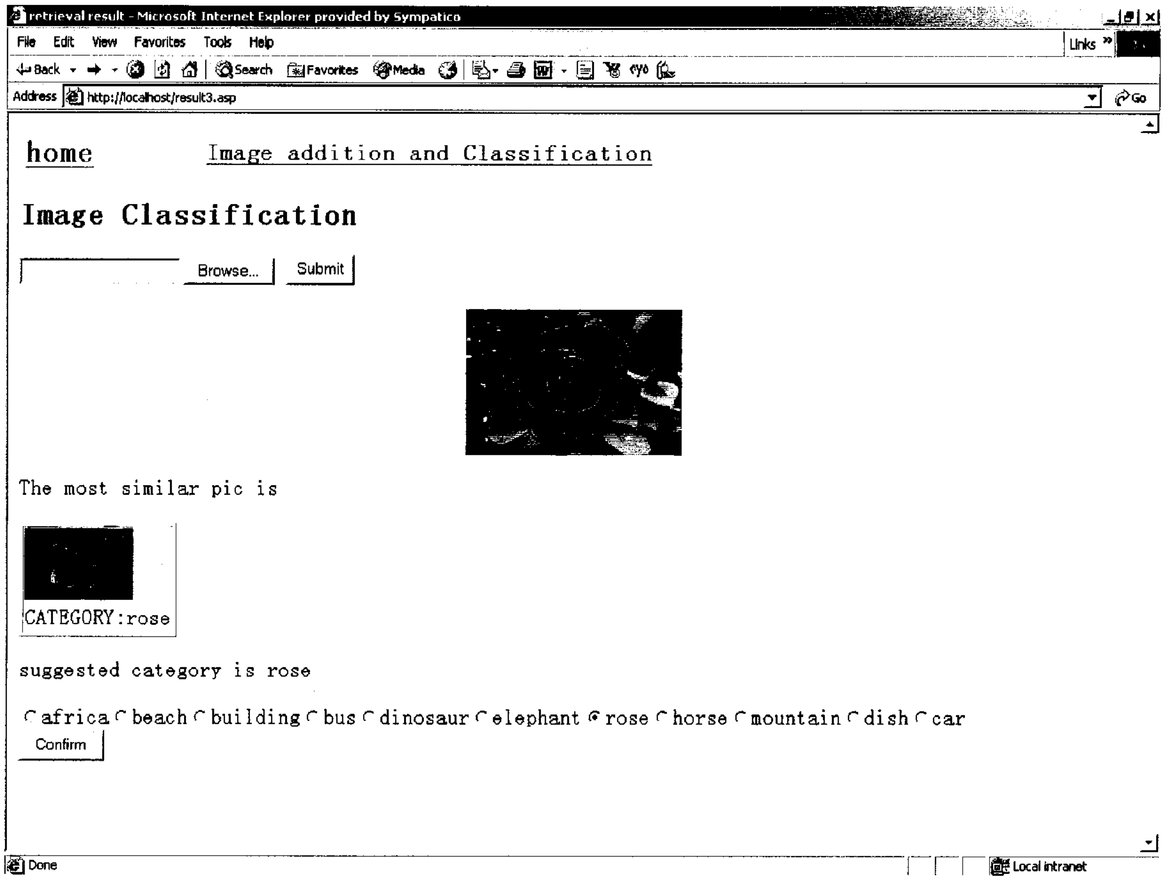
Confirm

Figure 14: SHMM-Image Edition and Classification 2

The following steps are involved in getting the suggested category.

1) Pass the image to the Texture Detection Module and Colour Histogram Module, respectively, and get a 16-dimensional vector.
2) Use the 16-dimensional vector to query Index Creation Module. After applying nearest neighbor search, get the most similar image in the database.
3) Applying the result generated in 2 to Query Database Module, get the suggested category and semantic header for the image waiting for classification.
4) Display the result from 3 in User Query Module and wait for the final decision from users/contributors.

If the user has the system administration privilege, he (she) may choose the New Category option and add a new category to the system, as shown in Figure 14. If the image contributor uses this system, he (she) cannot add new category to the system directly due to system security while he (she) may use email or phone administrator about adding the preferred new category. The sample interface is given in Figure 15.



retrieval result - Microsoft Internet Explorer provided by Sympatico

Address http://localhost/result3.asp

home   Image addition and Classification

Image Classification

The most similar pic is

CATEGORY:rose

suggested category is rose

○ africa ○ beach ○ building ○ bus ○ dinosaur ○ elephant [✓ ● rose] ○ horse ○ mountain ○ dish ○ car

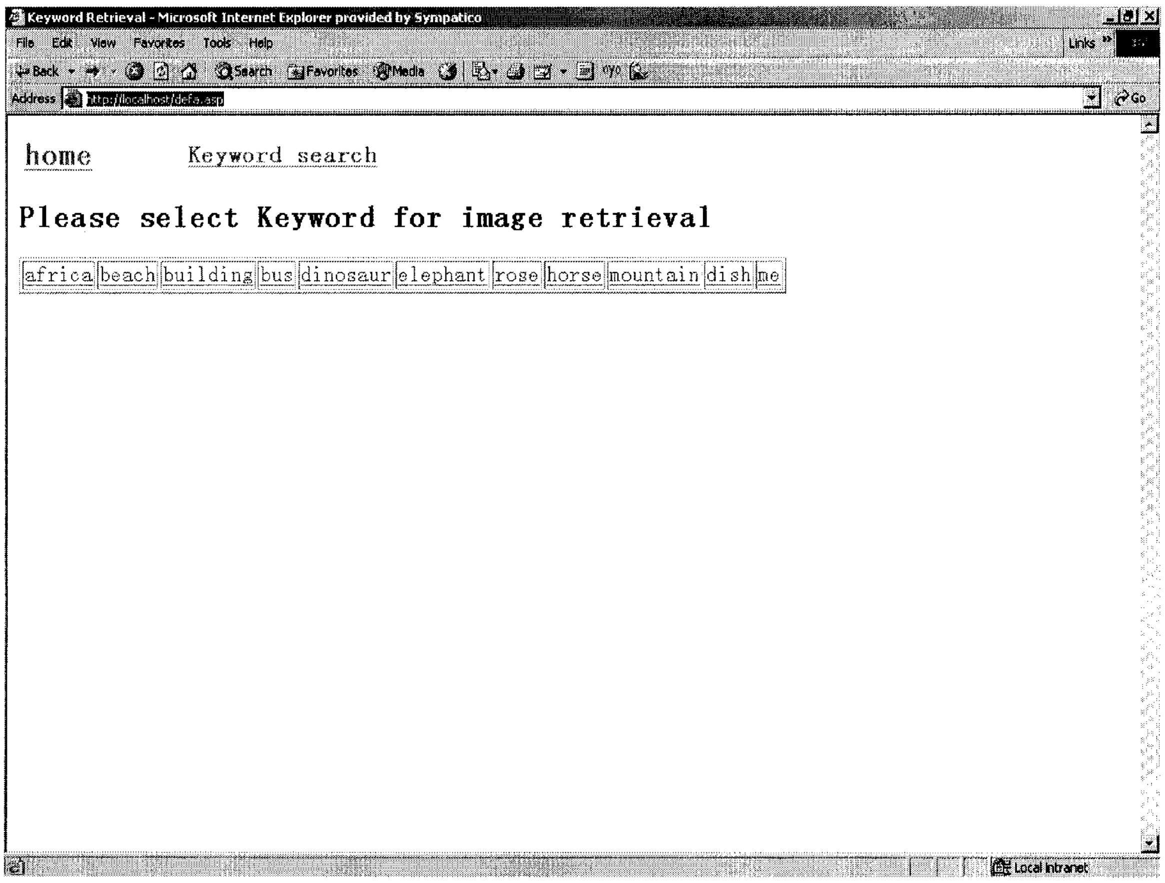
File Edit View Favorites Tools Help
Back Search Favorites Media
Browse...
Submit
Confirm
Done
Local intranet

Figure 15: SHMM-Image Edition and Classification 3

Once user makes the choice, and click "confirm button", the new image will be added to the image database. The required process includes:

1) Get the picture name together with the location, 16-dimensional colour vector, picture category and semantic description from User Query Module.
2) Copy the source image to the image database, insert 16-dimensional colour vector of this image into index structure through Index Creation Module, and insert semantic description of this image into image database through Database Module. In case any part of processing fails, the whole process will be rolled back to maintain the integrity of the system.
3) Transfer system feedback to User Query Module for further actions.

The main screen for "**Keyword Search**" is as follows:


Keyword Retrieval - Microsoft Internet Explorer provided by Sympatico
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/defa.asp
Go
Links
Local intranet

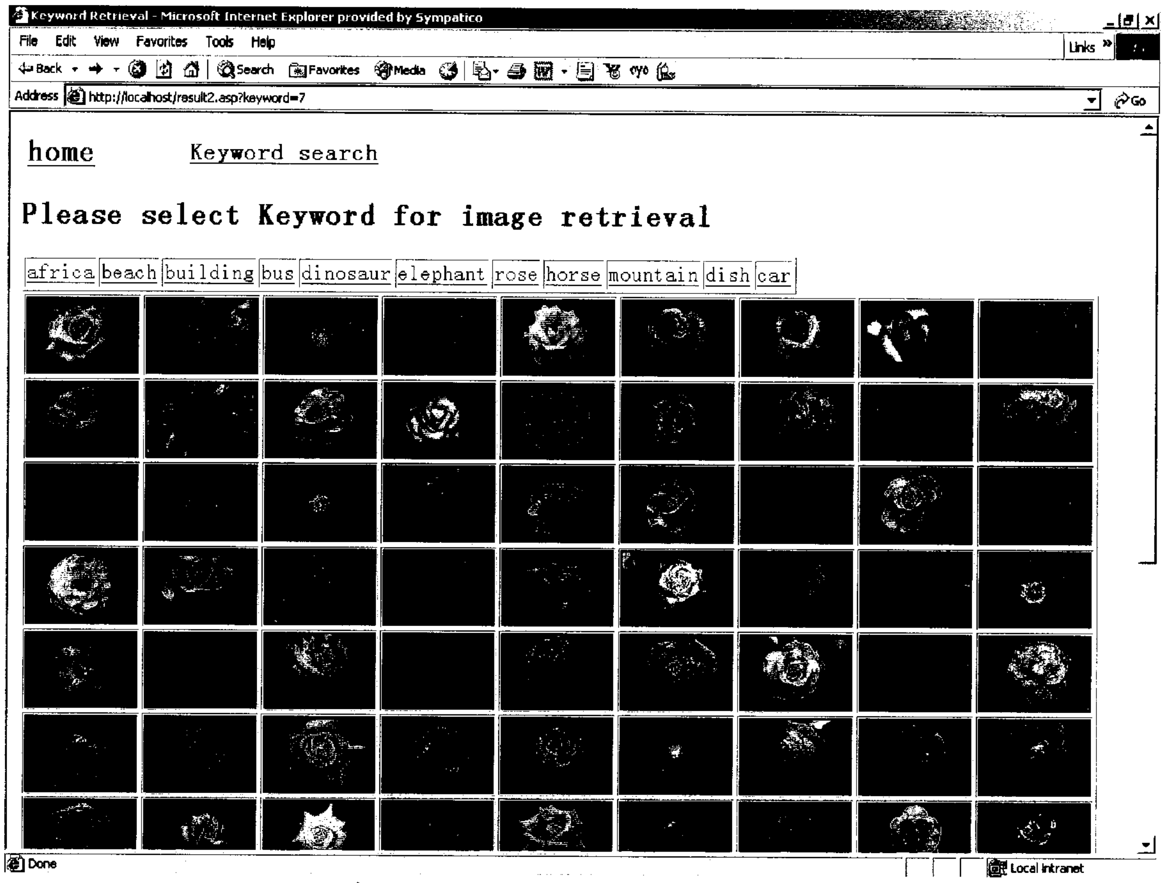
home Keyword search

Please select Keyword for image retrieval

africa | beach | building | bus | dinosaur | elephant | rose | horse | mountain | dish | me

Figure 16: SHMM-Keyword Search 1

This interface provides *Query by keyword* function. While automatically searching the image database for category and semantic description information, the system first provides all possible key words for user query selection. Of course, when the system has a large volume of keywords, we can only display the first layer keywords, or even in some extreme case the limited quantity of keyword. In current stage, we only implemented the category as the keyword; it is the same way to implement the semantic descriptions with subcategory as key word. When the user selects one keyword, the images under this category or more specifically that have this meaning in content are being shown in the order of creation date. For example, we choose the keyword *rose,* and all images classified in this category are displayed as in Figure 17.


Keyword Retrieval - Microsoft Internet Explorer provided by Sympatico
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/result2.asp?keyword=7
Go
Links
Done
Local intranet

home Keyword search

Please select Keyword for image retrieval

africa | beach | building | bus | dinosaur | elephant | rose | horse | mountain | dish | car

Figure 17: SHMM-Keyword Search 2

Users can also do *Query by image examples* in this interface via clicking any picture they have seen. The process procedure is almost the same as it does in "Similarity Retrieval" except that the number for retrieving results is kept as default value 20, and users cannot make any change there.

Chapter 5

Experimental Results and Discussions

The system is implemented on a PC with Microsoft Windows 2000. While using Personal Web 5.0 for web publishing, the web page is written in ASP. Also C, MFC have been used in different modules. Two-phase tests have been done on SHMM. First, the testing sample images come from the joint project from Stanford University and Penn State University started from 1995 known as SIMPLICITY (http://wang.ist.psu.edu/docs/related/) with totally 1000 images equally distributed in 10 categories. Secondly, the 10000 testing sample images come from Internet and have no clear classifications. The system performance is evaluated by the accuracy of similarity search results and the accuracy of image classification. Session 5.1 and Session 5.2 will discuss the performance based the testing for 1000 images. Session 5.3 will focus on the performance comparison between two-phase tests while session 5.4 will give an overall performance discussion.

5.1 Results for Similarity Search

Generally, the results vary a lot for various input images.
For images with one dominant texture and almost uniform colour inside that texture region, such as category of "roses", "buses", "dinosaur", the result is excellent. We can get at least 16 out of 20, usually 18 out of 20, and even hundred percent matching is not rare. The sample result is given in figure 19:

The first image is the sample image and the outcome is 8 out of 10. The reason why the other two images appear is simple. As discussed in the *Query by image examples* process procedure in section 4.2.5, this result lies in the values in the index structure. While the index value to represent an image is a 16-dimensional colour vector, it is obvious that the suspect images must have similar colour vector as the sample one. Considering the way we get the colour vector, we might have to admit that this kind of mistake is not avoidable. In Texture Detection Module, only one texture region is detected; therefore, suspect images such as 2 and 8 (numbered from left to right) have almost the same textures as the sample image. Taking this result as an input to Colour Histogram Module, first we divide the image into 16 parts, then select one dominant colour in each part, and form the colour vector based on these values. Since there is quite similar colour in each part, while processing nearest neighbor search in SR tree, this process will generate the result shown in Figure 18. However in reality, images 2 and 8 are not similar to the sample image.

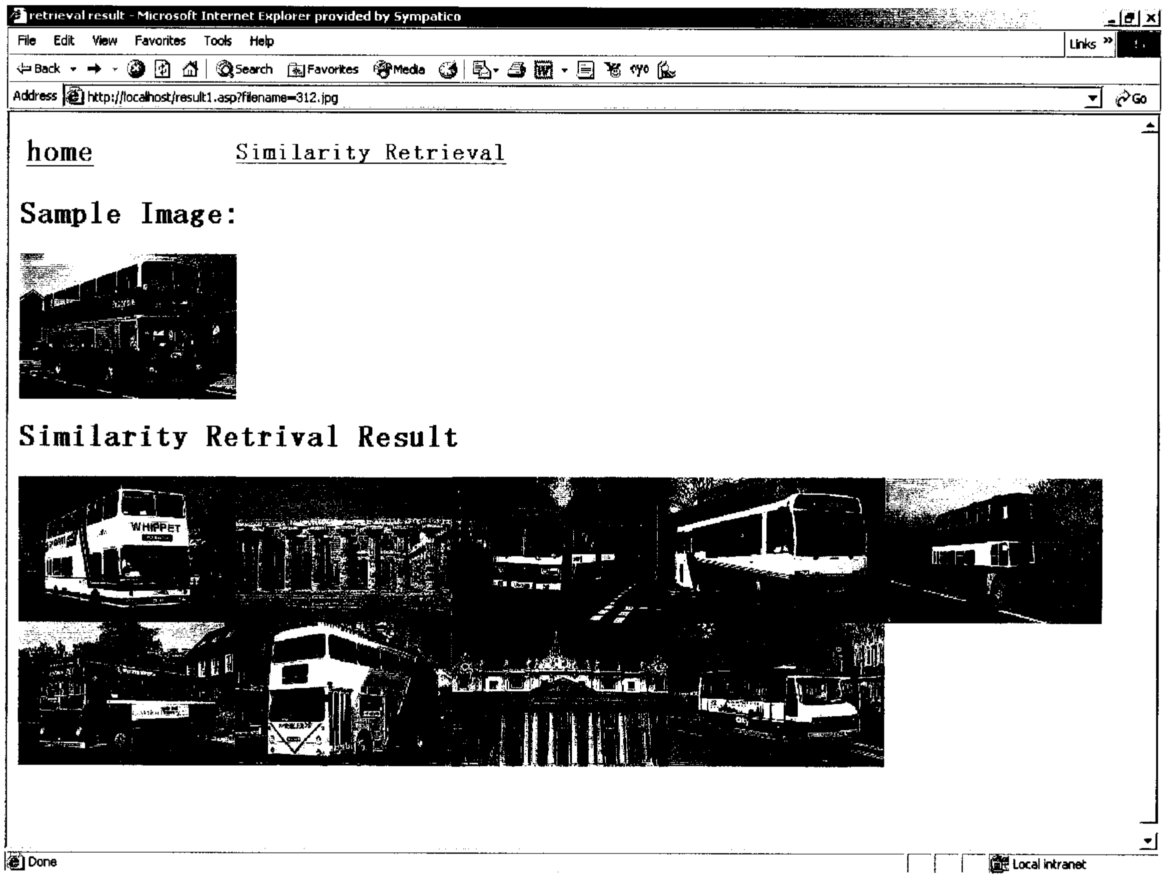
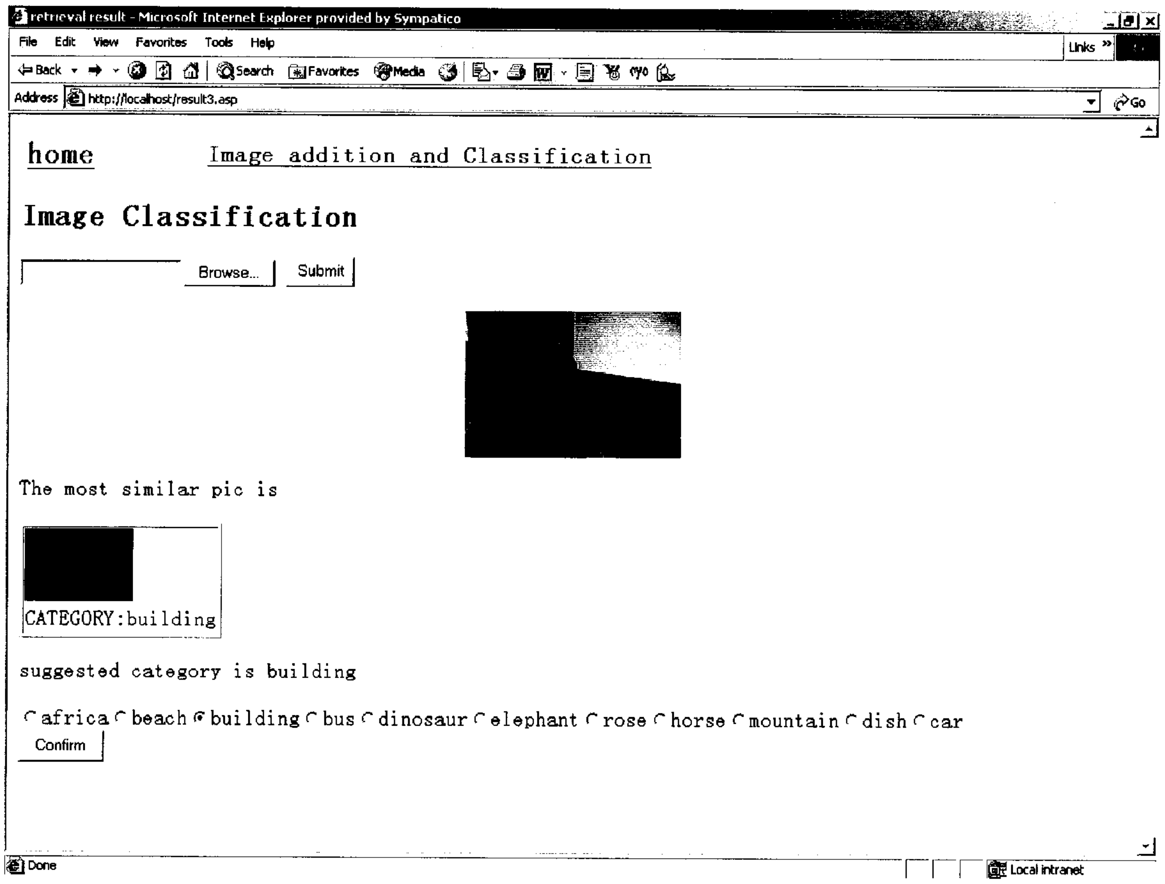

retrieval result - Microsoft Internet Explorer provided by Sympatico
File Edit View Favorites Tools Help
Links
Back Search Favorites Media
Address http://localhost/result1.asp?filename=312.jpg
Go
home
Similarity Retrieval
Sample Image:
Similarity Retrival Result
Done
Local intranet

Figure 18: Sample result from Similarity Retrieval

5.2 Results for Image Addition and Classification

The result also varies a lot from the input. As discussed above, for images with one dominant texture and almost uniform colour inside that texture region, we can get better result. And on the other hand, if images are full of trivial partial characters, the classification might be a disaster.

The sample would be as follows:


retrieval result - Microsoft Internet Explorer provided by Sympatico
File Edit View Favorites Tools Help
Links
Back Search Favorites Media
Address http://localhost/result3.asp
Go
Browse...
Submit
Confirm
Done
Local intranet

home Image addition and Classification

Image Classification

The most similar pic is

CATEGORY:building

suggested category is building

africa beach [✓ building] bus dinosaur elephant rose horse mountain dish car

Figure 19: Sample result from image addition and classification

Since we only generate one texture region in an image, it is hard to describe the difference between people in a building and the inner side of a building. If we keep all the current algorithms and still want the system to classify the above two categories, the success rate will be certainly low.

The texture for the sample image is shown in Figure 20-a while the most similar image in the database has the following texture given in Figure 20-b. After examining the colour feature, which even from the human perception is quite similar, the colour vectors for these two images are eventually similar. Therefore, after the nearest neighbor search inside index structure, we get the answer, which is obviously not a quite reasonable classification.

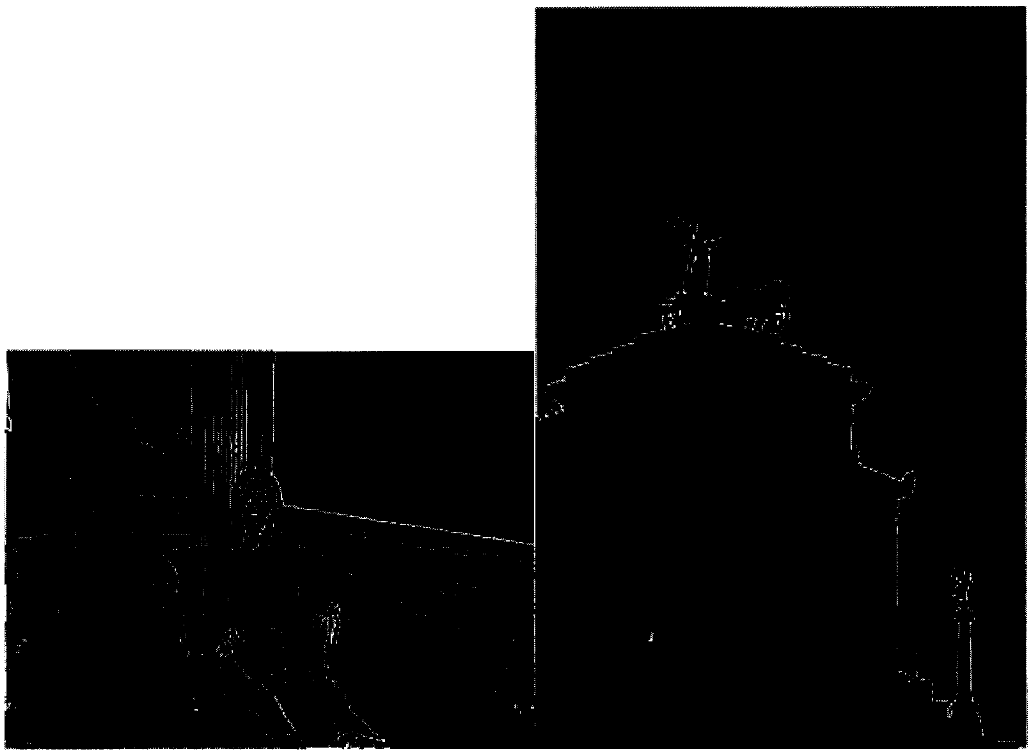


Figure 20: Result comparison

5.3 Performance comparison between two-phase tests

After having successfully finished the system test for 1000 images, we expand our testing to 10000 images. While randomly select images from Internet, we also have got several categories of images within our image domain. Our experimental results show that images with dominant colour inside texture region have better performance in SHMM than images lack of frequency colour transitions though the overall performance is worse than the first phrase testing. Looking back to these random images from Internet, we find that image quality and image size have influences to the performance. In our case, the quality of these 10000 images is not as good as it is in the phrase one and the image size for the 10000 images is smaller than the first phrase test. Also we can conclude that colour feature is not enough for a CBIR system.

5.4 Discussions

Though CBIR has been the hot research area in the past ten years, and models to well discipline modules as pointed out, how to build a good CBIR system and what a CBIR looks like are still unresolved. Through a literature survey on contemporary CBIR systems, our SHMM system chooses the most typical and also simple approach in each module of a CBIR. Despite the shortcoming of unstable performance due to different input image types, our experimental evaluations have shown that SHMM is effective and flexible in forming database queries, as well as retrieving user desired images.

Chapter 6

Conclusions and Future Work

6.1 Conclusions and Contributions

Through SHMM, we demonstrate how to build a CBIR system. SHMM aims to be a web oriented image library with an automatic image classification mechanism to support similarity search, and keyword search. As a standard CBIR system, SHMM has its feature extraction layer, which supports colour and skeleton feature extraction so as to generate a 16-dimensional vector value, based on which, a 16-dimensional SR tree is constructed. Using the nearest neighbor search technology inside the SR tree, similarity search is supported. With backend database support, the semantic description layer, which constructs the keyword of the semantic meaning of image, can be selected as a search requirement. When a new image is entered in the system, SHMM can automatically scan its feature, and based on the similarity images in the library, the system can provide the suggested semantic description for the new image.

The SHMM presented in this document serves as a sample and efficient CBIR system for image understanding especially for the pictures with one dominant texture and almost uniform colour inside that texture region. With the flexible module driven design, SHMM is capable of addressing complicated multi-layer CBIR requirement while expanding feature definition, defining knowledge representation model and adding domain knowledge.

6.2 Future Work

Future work for the SHMM might lies in three main directions:

1) Introduce more efficient texture detection algorithm or optimize the current algorithm to get more texture regions from an image.

2) Introduce segmentation detection into SHMM so that more detailed image understanding could be processed and hence even partial image matching might be supported in the future.
3) Add object and spatial representation to SHMM, so that more semantic meaning would be derived from images.
4) Migrate current system to the platform of Linux plus PHP plus MY SQL so as to integrate with CINDI system.

References:

[1] Arnold W. M. Smeulders, Macel Worring, Simone Santini, Amarnath Gupta, Ramesh Jain, “Content-Based Image Retrieval at the End of the Early Years”, IEEE transactions on pattern analysis and machine intelligence, pp.1349-1380, December 2000

[2] Rafael C. Gonzalez, Richard E. woods, “Digital Image Processing”, Addison Wesley, 1992.

[3] Wasfi Ai-Khatib, Y. Francis Day, Arif Ghafoor, P. Bruce Berra, “Semantic Modeling and Knowledge Representation in Multimedia Database”, IEEE transactions on knowledge and data engineering, pp. 64-80, Jan/Feb. 1999.

[4] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, “Query by Image and Video Content: the QBIC System”, Computer, pp.23-32, Sept.1995.

[5] Yihong Gong, “Intelligent Image Databases: Towards Advanced Image Retrieval”, ISBN 0-7923-8015-0, Kluwer Academic Publishers SECS421,1998.

[6] J. T. Robinson, “The KDB tree: a Search Structure for Large Multidimentional Dynamic Indices”, Proceedings of ACM SIGMOD, pp.10-18, April 1981.

[7] D. A. White, R. Jain, “R-trees: a Dynamic Index Structure for spatial Searching”, Proceedings of ACM SIGMOD, pp.47-57, Jun. 1984.

[8] N. Beckmann, H. P. Kriegel, R. Schneider, B. Seeger, “The R* tree: an Efficient and Robust Acess Method for Points and Rectangles”, Proceedings of ACM SIGMOD, pp.322-331, May 1990.

[9] D. A. White, R. Jain, “Similarity Indexing with the SS tree”, Proceedings of 12th International Conference On Data Engineering, pp.516-523, Feb. 1996.

[10] Norio Katayama, Shin’ichi Satoh, “The SR tree: an Index Structure for High-Dimensional Nearest Neighbor Queries ”, Proceedings of ACM SIGMOD, pp.369-380, May 1997.

[11] Zijun Yang, Xia Wan, C. C. Jay Kuo, “Interactive Image Retrieval: Concept, Procedure and Tools”, IEEE 32nd Asilomar Conference, pp.261-265, Nov.1998.

[12] V. E. Ogle, M. Stonebraker, “Chabot: Retrieval from a Relational Database of Image”, Computer, pp.40-48, Sept.1995.

[13] C. C. Hsu, W. W. Chu, R. K. Taira, "A Knowledge-Based Approach for Retrieving Images by Content", IEEE Transactions on Knowledge and Data Engineering, pp.522-532, Aug 1996.

[14] R. K. Srihari, "Automatic Indexing and Content-Based Retrieval of Captioned Images", Computer, pp.49-56, Sept.1995.

[15] J. R. Smith, S. F. Chang, "Exploring Image Functionalities in WWW Applications Development of Image/Video Search and Editing Engines", Proceeding of ICIP, pp.26-29, Oct.1997.

[16] J. Tajima, "Uniform Color Scale applications to Computer Graphics", Computer Vision. Graphics, and Image Processing, pp.305-325, 1983.

[17] D.B.Judd, G. Wyszecki, "Color in Business, Science and Industry, second Edition", Jon Wiley&Sons, Inc, 1963.

[18] Shareware CxImage introduction and software download, "http://www.xdp.it", archived at http://www.cs.concordia.ca/~bcdesai/grads/bh_zhang/cximage.zip.

[19] Hall.E. L, "Computer Image Processing and Recognition", Academic Press New York , 1979.

[20]M.Ikeda, "Fundamentals of Color Technology", Asakura Publishing Tokyo, Japan, 1980.

[21] E.Remias, G.Sheikholeslami, A.Zhang, F.Syeda_Mahmood, "Supporting Cotent-Based Retrieval in Large Image Database Systems", Multimedia Tools and Application, pp.153-170, March 1997.

[22] Munsell, A. H, "A Color Notation", Munsell Color Company, Baltimore, MD, 1923.

[23] M.Miyahara, Y.Yoshida, "Mathematical transform of (r,g,b) color data to Munsell(h,v,c) color data", SPIE Proceedings in Visual Communication and Image Processing, Volume 1001, pp.650-657,1988.

[24] Hideyuki Tamura, Shunji Mori, TaKashi Yamawaki, "Textural featues corresponding to visual perception", IEEE Transactions on Systems, Man, and Cybernetics, SMC-8(6), pp.460-472, June 1978.

[25] A.W.M.Seulders, M.L.Kersten, T.Gevers, "Crossing the Divide between Computer Vision and Data Bases in search of Image Databases", Proc. Fourth Working Conference Visual Database Systems, pp.223-239, 1998.

[26] M.L.Pao, M.Lee, "Concepts of Information Retrieval", Libraries Unlimited, March 1989.

[27] S.Santini, A.Gupta, R.Jain, "User Interfaces for Emergent Semantics in Image Databases", Proc.Eighth IFIP Working Conference Database Semantics (DS-8), 1999.

[28] S.L.Tanimoto, "Elements of artificial Intelligence Using Common LISP", Computer Science Press, 1990.

[29] L.A.Rowe, J.Boreczky, C.Eads, "Indexes for User Access to Large Video Databases", Proc.IS and T/SPIE Int'l Symp Electronic Imaging: Science and Technology, pp.150-161, Feb.1994.

[30] R.Bayer, E.Mccreight, "Organization and Maintenance of Large Ordered indexes", Acta Informatica, pp.173-189, 1972.

[31] R.Jain, "Workshop Visual Information Management Systems", Proc.US NSF, pp.57-75, 1992.

[32] Eric W. Weisstein, "Laplacian." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/Laplacian.html, archived at http://www.cs.concordia.ca/~bcdesai/grads/bh_zhang/laplacian.html.

[33]C S McCamy, H Marcus, J G Davidson, "A Color Rendition Chart, Journal of Applied Photographic Engineering", Volume 11, number 3, pp.95-99, 1976.

[34] Günter.Wyszecki, W.S.Stiles, "Colour Science—Concepts and Methods, Quantitive Data and Formulae", WS Wiley and Sons INC NY, 1982.

[35] R.W.G. Hunt, "Measuring Colour (second edition)", Ellis Horwood, 1991.

[36] M. J. Swain, D. H. Ballard, "Color Indexing", International Journal of Computer Vision, 7(1), pp.11–32, 1991.

[37] Rafael C. Gonzalez, Richard E. Woods, "Digital Image Processing (2nd Edition)", Addison-wesley publishing, Jan.2002.

[38] Adrian Ford, Alan Robert, "http://www.poynton.com/PDFs/coloureq.pdf", archived at http://www.cs.concordia.ca/~bcdesai/grads/bh_zhang/coloureq.pdf.

Appendix: Algorithms in SHMM

Algorithms in SHMM can be divided into 5 parts based on system diagram illustrated as Figure 5 in the thesis. While main algorithms lie in the image processing and index module, we will go through algorithms in these parts denoted as A1 for Texture Detection Module, A2 for Colour Histogram Module, A3 for Index Creation Module. How these modules work with Database Module and User Query Module has been illustrated in the thesis; for the 3 modules, we will focus on the algorithms as well as input and output.

.

A1 and A2 are implemented in getfeature.cpp while A3 is implemented in fetchNN.cpp, storeRecord.CPP, dynamicBuild.CPP.

A1:
Input: filename
Output: global CxImage object imageTx

```
Void scanTx(filename)
{
  if (!imgTemp.Load(filename,0)) return;
 //iWidth, iHeight is the standard size of the image. Here is 256*256
 If (!imgTemp.Resample(iWidth, iHeight)) return;
 imgTemp.FFT2(&imgTemp, &imgTemp, &rfft1, &ifft1);
 pIndex = rfft1.GetPixelIndex();

// get the texture dots from an image
 long kernel[]={-1,-1,-1,-1,8,-1,-1,-1,-1};
 ifft1->Filter(kernel,3,pIndex,0);

// get the texture image
//compare each dots with the edge dots in the texture
pImage = rfft1.GetBits();
//blocksize =16
for(iRow = 0; iRow < iHeight; iRow += iBlockSize)
{
 for(iCol = 0; iCol < iWidth; iCol += iBlockSize )
                {
                        // Block Processing
                 for(iLoopRow = 0; iLoopRow < iBlockSize; iLoopRow++ )
                 {
                  for(iLoopCol = 0; iLoopCol < iBlockSize; iLoopCol++ )
                        {
                                Get the filtered value in the same position
                                If value exists
                                count this dot in the texture,
                                combine this not with the last confirmed texture dot
                                add to dImage
                        }
                }
                }
}
dImage.FFT2(&dimage, &dimage, & imageTx, & imageTx);
}
```

A2:

Input: imageTx

Output: global file object picfeature.txt

```
Void colorHis()
{
  //resize the image to 256*256, rewrite Resample function in CxImage
 If (!imageTx.Resample(iWidth, iHeight)) return;
 //blocksize=64
 for(iRow = 0; iRow < iHeight; iRow += iBlockSize)
 {
 for(iCol = 0; iCol < iWidth; iCol += iBlockSize )
      {
```

```
                // Block Processing
                  reset pixel counter for each colour (9 counters totally)
                  reset  representation value for each colour(9 values)
                  for(iLoopRow = 0; iLoopRow < iBlockSize; iLoopRow++ )
                  {
                  for(iLoopCol = 0; iLoopCol < iBlockSize; iLoopCol++ )
                        {   R,G,B  =  imageTx.GetPixelColor(iRow*  iBlockSize  +  iLoopRow
                            .iCol*     iBlockSize +     iLoopCol)
                            transfer RGB to HVC
                            use H, V value to classify the colour group
                            in the newly classified colour group ,counter = counter +1
                            value= value+ $(H^2*V/10000)^2$
                        }
                }
            decide the dominant colour through pixel counter
            represent this dominant colour by $(value)^{1/2}$ /counter
      }
    }
  delete imageTx;
  get  16 colour value, export to picfeature.txt
  }
```

A3:

Action1: build index (create a new index file or insert new values to a index file)

Input: picfeature.txt , picindex

Output: updated picindex

```
void storeRecord (picfeatureFile, picindex)
{
        picFeature.open(picfeatureFile);
        if( !picFeature ){return ;}
        //check whether the destination exists
        if picindex not exist
        {       //dynamicBuild an empty index file
        create binary file picindex ;
        create an empty root node;
        define the maximum and minimum number for leaf node, 10 and 6 respectively
        binding the root node with HnSphere(sphere), HnRect(rectangle)
}
while( picFeature.eof() !=0 )
        {
                read a line;
                //insert the new vector into picindex
                create 16 dimensional vector;
                allocate this vector into picindex by choosing the smallest  HnSphere,
                HnRect
                if available position exists in the entry leaf node
                 {   insert this vector to the leaf node
                     while(not root node)
                    {
                            modify counter, HnSphere, HnRect at its parent node
                            modify pointer if necessary at its parent node
                     }
                 }
                else
                {//split the leaf node
                while (no request in splitting stack)
                {
                        create a new node with its HnSphere, HnRect covering the whole
                        entry needed to be deployed;
                        take the new node as a parent node and stack a new split
                        requirement if necessary;
                        rearrange entries to be mostly equally distributed in the new
                        parent node;
                        delete this request from splitting stack;
                }
                insert this vector to the leaf node
                 while(not root node)
                {
                        modify counter, HnSphere, HnRect at its parent node
                         modify pointer if necessary at its parent node
                }
            }
        }
}

Action2: find the nearest neighbor for the input vector
Input: value vector, picindex
Output: global picresult.txt
void fetchNN(sampleVector, picindex)
{
        //set the comparing range, radius for rectangle and sphere
        set HnRange for HnRect;
        set HnSphere.radius;
        put root node to the visiting list
         while (not empty in visiting list)
        {
                get child node T from the top node in visiting;
                 if not leaf node
                if ((HnRect(sampleVector)overlapsHnRect(T))     or
                (HnSphere(sampleVector)overlaps HnSphere(T)));
                        {
                                put to the visiting list
                        }
                else
                retrieve any entries under T that has overlaps range among predefined values
        }
        sort the result based on the range and radius differences;
        export value handles in the result to picresult.txt
        update USER table while set WL=0 for record where REQUEST=1
}
```