INFORMATION TO USERS

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

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

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

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

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

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

UMI®
800-521-0600

Applying CBIR to Interior Decorating

Joanne Pilkington

A Major Report

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

August 1998

ABSTRACT

Applying CBIR to Interior Decorating

Joanne Pilkington

This project explores the use of content based image retrieval in the domain of interior decorating. Content based image retrieval is the analysis and retrieval of images based on their content and not on a textual label or a keyword as in traditional searches. In this project, features important in interior decorating, such as colour and texture, are extracted from images and used to compare and search the image database which consists of over 600 images. The images in the database represent items used for interior decorating such as wallpapers, borders, fabrics, floorings, and carpets. Queries on the database can be based on an example image or by selecting from colour and/or texture choosers.

A comparison is done on the performance of different variations of the colour and texture features and of various distance metrics used to measure the similarity between the query and database features. The colour features extracted from the images include: the average colour in the YIQ, HSV, and RGB colour spaces, colour histograms in the HSV and RGB (8 bin and 64 bin) colour spaces, a cumulative colour histogram in the HSV colour space, and the moments of colour distribution in the HSV and RGB colour spaces. The texture features extracted from the images include: contrast, coarseness and four variations of directionality. Six distance metrics are used. They are: L_1, L_2, weighted L_2, L_∞, weighted sum, and scalar distance.

The objective of this project was to find the most appropriate combinations of features and distance metrics for use in an interior decorating application. Test results show that the most effective types of colour and texture features are the colour histogram and the directionality features. All the variations of the colour histograms and distance metrics performed very well except the L_2 distance metric when applied to the RGB colour histograms. This combination of feature and distance metric did not perform well for the lower ranked matches. The histogram version of the directionality feature variations performed much better than the vector variations, except in very simple cases where the images contained patterns in only one definite direction (horizontal, vertical, or diagonal) in which case the vector variations performed almost as well as the histogram variations.

Acknowledgements

I would like to thank my supervisor, Dr. Ching Suen, who never seemed to give up hope in me although I was far away and did not provide any real indication as time went on that I was progressing and would eventually finish this project.

I would like to thank my friends and family for their understanding for when I could not spend time with them because I was working on this project and for their support in continuously asking me how I was progressing which made me feel guilty when I was not working on it. This always helped to push me back to work.

Lastly, thanks to Mike for all his love, support, and candy!

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1. Content Based Image Retrieval

Digital libraries of text, sound, image, video and other data are rapidly growing in size and availability. New technologies for organizing and retrieving by content in databases of still digital images or digital video sequences are required. The problem of how to store large numbers of digitized images and retrieve pictures from such a collection is an active area of research and overlaps many fields within computer science including computer graphics, image processing, pattern recognition, information retrieval, and databases. This is what makes this topic so interesting and an ideal project because you are required to gain knowledge in so many areas.

1.1.1. Background

Content based image retrieval (CBIR) is the analysis and retrieval of images based on their content. This technology complements and extends text or keyword search. Text based search criteria combined with content based criteria can be combined to achieve good results. Text, however, is difficult or impossible to compute automatically from the pixel data and is best entered manually by the user. As well, keywords are not always adequate in searching for images. They may not be precise enough. For example, there can be many shades of "green". It is difficult to describe many visual characteristics such as textures and colour in words, and people often use different words to describe the same visual phenomena.

Traditional database searches consist of exact matching or range searches. Exact search does not allow for navigation and refinement which is possible with content based image retrieval. With traditional database searches one can not construct queries such as: "show me more images with colour similar to this one". These types of queries are possible with CBIR technology.

Examples of applications and markets where content based image retrieval technology would be useful include:

- digital libraries
- electronic publishing: advertising agencies, libraries, museums, art galleries
- satellite images, computer-aided design (CAD)
- trademark searching
- medicine, radiological information systems
- face matching for law enforcement agencies
- on-line catalogs and shopping
- internet publishing and searching
- remotely sensed image management for defense
- environmental image analysis
- weather forecasting
- interior decorating

It is the latter application, interior decorating, this project focuses on. The next section explains why this application is well suited to content based image retrieval.

1.2. Project Overview

The most successful CBIR systems seem to be the ones that have focused on one application or were customized for each application and describe methods of selecting images within one domain. This project focuses on the interior decorating application and describes methods of selecting images within that domain.

In this project, features important to interior decorating such as colour and texture are extracted from images and used to compare and search the image database which consists of over 600 images representing items in interior decorating such as wallpapers, borders, fabrics, etc. Queries

can be based on an example image or by selecting from colour and/or texture choosers. A comparison is done on the performance of different variations of the colour and texture features and of various distance metrics used to measure the similarity between the query and database features. The objective is to find the most appropriate combinations of features and distance metrics for use in an interior decorating application.

1.2.1. Interior Decorating Application

Interior decorating is an ideal application for the current state of the art in computer vision today because the content is limited to parameters that are feasible to compute such as colour distribution, texture, etc. For this application we do not need to attempt to derive more complex descriptions for objects such as "dog" or "house" which are currently beyond the reach of pattern and image understanding technology, at least for a commercial system.

In this project, interior decorating samples of items such as wallpaper, borders, fabrics, etc. are referred to as swatches. In the interior decorating application there is no need to query by shape or object because interior decorating swatches that contain objects, for example a horse, are not very common. As well, it is highly likely that the title of the item would contain a keyword describing the object it contains. Even if the title does not contain the keyword, these patterns are very specific requests and there would not be too many of them so it would not take long for the user to search for these manually.

No manual or semi-manual pre-processing is required to locate objects when populating the database because only colour and texture features are extracted from the images. Database population can be done completely automatically with no human intervention.

Advantages of using CBIR

Traditionally, when decorating a room or house the consumer must search through many books from different manufacturers for each item they are looking for. This could involve trips to many different stores. The user can not easily compare samples found at the different stores. Even at one store such as Home depot, there are numerous swatch books to search through. Books are also generally organized by colour or by pattern but not by both. Sunworthy, a company that produces wall coverings, claims that they provide "Easy selection - Easy decorating. For fast simple selection, Sunworthy has arranged the best of their best borders according to six colour categories." If you are looking a for a particular colour or a particular type of pattern this categorization would hardly be sufficient. By applying CBIR to interior decorating, the consumer could potentially search for all types of items from all manufacturers and search by colour and

texture all at the same time. The user can easily view selections from each type of item side by side. The consumer could even have a sample of an item to match which could be scanned in and used as an example on which to base queries. The work of this project could also be incorporated into existing interior design software which allows you to design the room and view it all together.

Disadvantages of using CBIR

Colour rendition can vary from computer to computer. Scanning, digitizing and the monitor may change the colour - but if the scanner, digitizer and the monitor are always the same for each image, then the images can be compared to one another accurately. However, the colour may not be exactly the same as the real item, so the user would still probably want to see the real swatch before making a purchase. A commercial application could easily provide indexes and easy lookup of the real swatches once the user had made their choices.

The computer screen is relatively small which makes viewing of swatches more difficult especially if the swatch has a large pattern. The relative size of patterns can also be a problem when comparing swatches. This problem can be eliminated if the scanning and digitizing of the images is done at the same zoom level for all swatches.

With regards to texture, the computer screen lacks the tactile sense of texture. Some swatches of wallpaper for instance and certainly carpet have a tactile texture. Once the user had narrowed down their choices they could then go and feel the real item.

1.2.2. Home Decorating / Interior Design Software

There are many commercial software packages available for the applications of home decorating and interior design. Prices can range anywhere from $15 US to $60 US and higher. Following is a list of a few of these applications:

- Visual Home by Sierra (http://www.visualhome.com)
- Planix Home Design Suite 3D by Autodesk (http://www.drafix.com/structure/products/phds_specs.htm)
    - specializes in interior design s/w which permits the user to select carpets, paint colour and other interior styles
- 3D Home Interiors by Broderbund (http://www.3dhome.com/3dhi/home.html)
- Complete Home Designer by Alpha Software (http://www.alphasoftware.com/chd/index.html)

- Image Your Interior Design by Visual Applications, Inc.
    - It allows you to place digital photographs of real products onto a picture of your home, wall, or room and see what it is really going to look like.
    - http://www.visapp.com or http://www.showoff.com

The goal of this project is not to provide the same capabilities as these commercial applications. The goal of this project is to build content based image retrieval software that could run stand alone and/or could potentially be incorporated in a commercial interior decorating application like one of those listed above.

Of all the commercial software packages that were surveyed, none of them incorporated content-based image retrieval. Content based image retrieval could be used in these types of applications to enhance the image search capability. To be of greatest value, the image databases in these applications should contain images of interior decorating swatches from as many manufacturers as possible. Some of the applications listed above already contain items from as many as 50 manufacturers. You could even go so far as to allow the user to customize a swatch if it is not exactly the right colour or pattern. The customizations could then be sent to a computer which operates the presses/printers that produce the wallpapers/fabrics.

The content based image retrieval software built in this project could also run stand alone. It could be used for applications such as interactive home shopping over the internet.

1.3. Outline

This section provides a brief outline of what the remaining chapters and appendices in this paper contain.

The next chapter, Features, starting on page 7, gives a detailed description of each of the features that are extracted from the interior decorating images stored in the database. The Image Retrieval chapter, starting on page 19, describes the different distance metrics used to compute the similarity between query features and database image features and also describes the different types of queries that can be made on the image database. The Design chapter, starting on page 28, provides a high level view of the system design including the architecture, the database design, the user interface and query designs. The Results chapter, starting on page 46, compares the performance of the different features and distance metrics in the context of an interior design application. Finally, the Conclusion, on page 59 summarizes the keys findings of this project and briefly describes some possible future work.

Appendix A on page 65 illustrates the different query options available to the user. Appendix B on page 66 contains some miscellaneous implementation notes such as how to install and run the application.

2. Features

An image feature is a piece of semantic information extracted from the image. Features are either associated with the entire digital image or with specific regions of interest within the image. Due to the nature of the interior decorating domain and the types of images associated with this domain, this project focuses on features associated with the entire image. Thus, the features extracted from the images in this project are all variations of colour and texture features. This section describes each of the features used in this project.

2.1. Colour Features

Before describing the colour feature, a little must be said about colour spaces.

2.1.1. Colour Spaces

A colour space is a means of uniquely specifying a colour. There are a number of colour spaces common used. The colour space used depends on the particular industry and/or application involved. For example as humans we normally determine colour by parameters such as brightness, hue, and colourfulness. On computers it is more common to describe colour by three components, normally red, green, and blue.

There are generally ways of converting (transforming) from one colour space to another although in most cases the transformation is nonlinear. Some colour spaces for example can represent colours which cannot be represented in others.

Three different colour spaces are used in the features in this project, they are: YIQ, HSV, and RGB. Each of these colour spaces is described below along with a brief description of gray-scale.

YIQ Colour Space

The YIQ colour space was developed for the NTSC composite colour television standard. The YIQ colour model is a recoding of the RGB colour model used for television transmissions. 'Y' is the luminosity value (it is the value which controls the intensity on black and white monitors); the 'Y' value can be used to ensure that two different colours will be distinguishable on a black and white display device. 'I' is the in-phase colour value which contains orange-cyan colour hue information and 'Q' is the quadrature colour value which contains green-magenta colour hue information.

The range of the three Y, I, Q components are from (0, -152, -133.11) to (255, 152, 133.110).

HSV Colour Space

The Hue/Saturation/Value model was created by A. R. Smith in 1978. It is based on such intuitive colour characteristics as tint, shade and tone (or family, purity and intensity). The coordinate system is cylindrical, and the colours are defined inside a hexcone. The hue value H describes the pure colour and runs from 0 to 360°. Hue is that attribute of a colour by which we distinguish red from green, blue from yellow, etc. There is a natural order of hues: red, yellow, green, blue, purple. The saturation S is the degree of strength or purity and ranges from 0 to 1. Purity is how much white is added to the colour, so S=1 makes the purest colour (no white). The brightness V also ranges from 0 to 1, where 0 is the black. Value indicates the lightness of a colour.

The human eye is more sensitive to luminance components than to chrominance components.

RGB Colour Space

The most popular colour space is RGB. In this colour space, the R (red) component represents the amount of red in the pixel, the G (green) component represents the amount of green in the pixel and the B (blue) represents the amount of blue. The RGB colour model is used for monitors and scanners.

The RGB colour space can be visualized by a unit cube. Each colour (red, green, blue) is assigned to one of the three orthogonal coordinate axes in 3D space. An example of such a cube is shown below in Figure 1 along with some key colours and their coordinates.

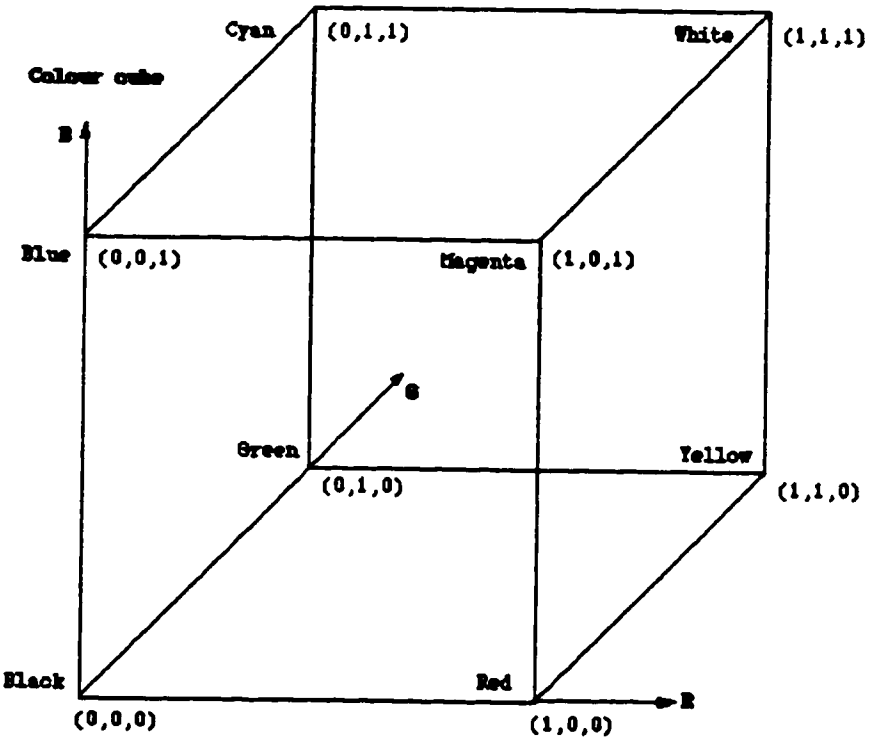


Figure 1. Unit colour cube

Along each axis of the colour cube the colours range from no contribution of that component to a fully saturated colour. The colour cube is solid, any point (colour) within the cube is specified by three numbers, namely, an (r,g,b) triple. The diagonal line of the cube from black (0,0,0) to white (1,1,1) represents all the grays, that is, all the red, green, and blue components are the same. A more "colourful" view of the shell of the colour cube is shown in Figure 2.

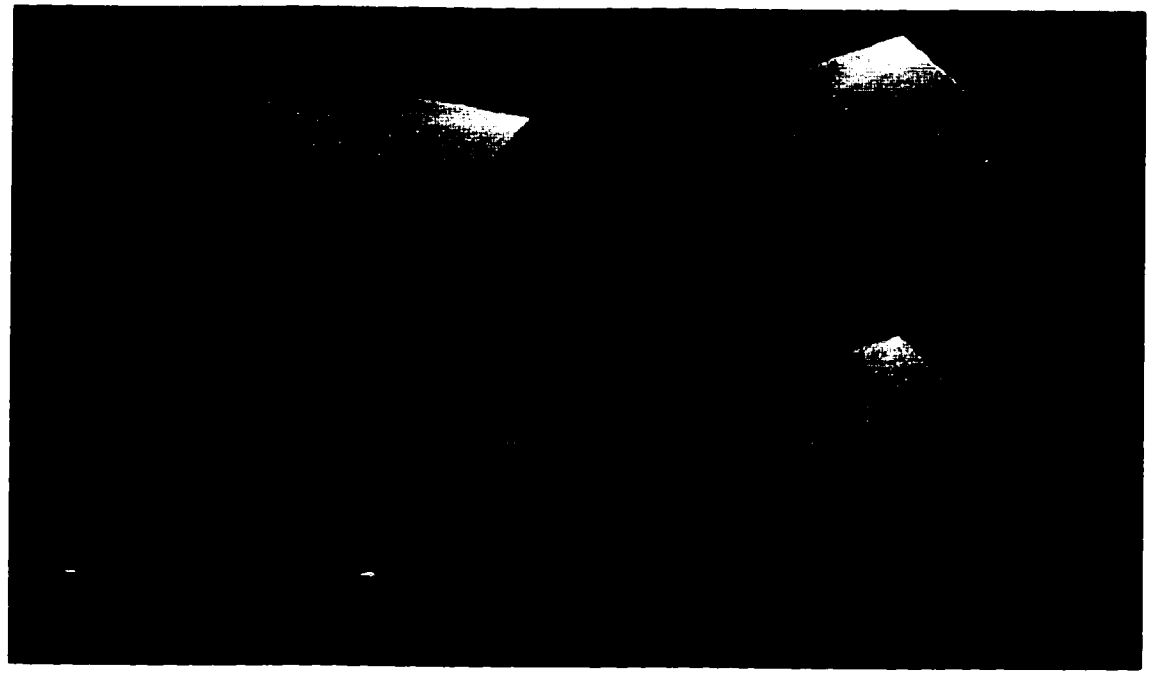


Figure 2. Coloured shell of the RGB cube

Gray-scale

A gray-scale picture only has one colour component: luminance. The higher the value, the lighter the colour of gray. The value 0 represents the colour black and the value 255 represents the colour white.

2.1.2. Average Colour Feature

The average colour feature can be computed in any colour space. This project computes the average colour in the YIQ, HSV, and RGB colour spaces. Computing the average colour in different colour spaces will focus on different aspects of colour. It is not expected that image retrieval using the average colour feature in one colour space will yield exactly the same results with a query using the average colour in another colour space.

Average colour is computed by summing the values of a given colour component for all the pixels and dividing by the number of pixels. This is repeated for all three colour components. The result is a three element vector. Each element in the vector represents the average of one of the colour components in the colour space.

2.1.3. Moments of Colour Distribution

In [8], Stricker introduced the idea of using colour distribution features. It is known from probability theory that a probability distribution is uniquely characterized by its central moments. If the colour distribution of an image is interpreted as a probability distribution, then the colour distribution can be characterized by its central moments as well. Stricker proposed that the average, standard deviation, and third root of skewness be used because all the values would then have the same units which makes them somewhat comparable.

If the i-th colour channel (e.g. R channel if the RGB colour space is being used) at the j-th image pixel is p_{ij}, then the values of colour distribution are:

$$E_i = 1/N \sum_{j=1}^{N} p_{ij} \quad , \quad \sigma_i = \left(1/N \sum_{j=1}^{N} (p_{ij} - E_i)^2 \right)^{1/2} \quad , \quad s_i = \left(1/N \sum_{j=1}^{N} (p_{ij} - E_i)^3 \right)^{1/3}$$

2.1.4. Colour Histograms

One of the most common colour image features is the colour histogram. Colour histograms are a way to represent the distribution of colours in images where each histogram bin represents a colour in a suitable colour space (e.g. RGB, HSV). A colour histogram counts how much of each colour occurs in the image.

To compute a colour histogram, the colour space must first be divided up into non-overlapping subspaces and each subspace is assigned to a histogram bin. An image's colour histogram is computed by traversing the image pixel by pixel, quantizing each pixel into one of the bins and incrementing the count of that bin. After traversing the image, the histogram contains the number of pixels within each colour subspace. The histogram is then normalized by converting each bin value to a percentage representing the relative number of pixels contained in that colour subspace.

Different histograms can be computed for the same image based on colour space, number of bins or type of histogram. This project examined the use of four different colour histograms (HSV, HSV cumulative, small RGB, and RGB) each of which is described below.

HSV Histogram

The HSV histogram used in this project is a 24 bin histogram. The bins in this histogram are divided among the three different HSV colour components as follows: the first 16 bins cover the

range of hue (H), the next 4 bins cover the range of saturation (S), and the last 4 bins cover the range of value (V). The H, S, and V components range from (0, 0, 0) to (2π, 1, 1) respectively.

HSV Cumulative Histogram

A cumulative histogram is a simple variation of the regular colour histogram. The cumulative colour histogram

$$\bar{H}(M) = (\bar{h}_1, \bar{h}_2, \ldots, \bar{h}_n)$$

of the image M is defined in terms of the colour histogram $H(M)$ as follows:

$$\bar{h}_j = \sum_{k=1}^{j} h_k$$

Cumulative colour histograms are always completely dense vectors even if only a few colours of the discrete colour space appear in each image.

In [8], Stricker shows that cumulative colour histograms together with the L-metrics are more robust with respect to the quantization parameter of the histograms than the L-distances applied to regular colour histograms.

8 bin RGB histogram

The 8 bin RGB colour histogram divides up the RGB colour space into 8 bins, each one representing a cube-shaped subspace of the RGB space. Visually, each bin represents a sub-cube that sits in one of the 8 corners in the RGB colour cube shown in Figure 1 on page 9 and Figure 2 on page 10. Table 1 specifies the bin ranges in terms of RGB values.

Bin #	Colour	RGB range
0	Black	(0, 0, 0) - (127, 127, 127)
1	Blue	(0, 0, 255) - (127, 127, 128)
2	Green	(0, 255, 0) - (127, 128, 127)
3	Cyan	(0, 255, 255) - (127, 128, 128)
4	Red	(255, 0, 0) - (128, 127, 127)
5	Magenta	(255, 0, 255) - (128, 127, 128)
6	Yellow	(255, 255, 0) - (128, 128, 127)
7	White	(255, 255, 255) - (128, 128, 128)

Table 1. RGB Histogram - 8 bin ranges

The purpose of this simple histogram is for very crude colour matching and is the basis for the *Some/Mostly* colour queries (see Some/Mostly Colour Queries on page 44 for more details on this query type).

64 Bin RGB Histogram

The 64 bin RGB colour histogram is very similar to the 8 bin version except the RGB colour space is divided into 64 sub-cubes instead of 8. Visually, each axis in the colour cubes in Figure 1 on page 9 and Figure 2 on page 10 is divided into 4. This histogram is used in QBIC [1][2][3][4] and in [5].

2.2. Texture Features

Texture has the following properties:

- repetitive , either:
  - natural and somewhat irregular patterns
  - man-made patterns (such as a brick wall or woven wire) where the texture primitives are rather well defined and more regular
- oriented and directional,
- grainy (randomly positioned micro-patterns that lack directionality, e.g. coffee beans and pebbles), and
- difficult to comprehend and describe symbolically.

Textural properties can be decomposed into parameters such as coarseness, contrast, directionality, etc. The texture features chosen in this project come from Tamura's work in [6]. Tamura approximated in computational form six basic textural features, namely, coarseness, contrast, directionality, line-likeness, regularity, and roughness. In [6], very successful results were obtained with coarseness, contrast and directionality therefore these textural features were chosen for this project.

Note that all textural features are computed from a gray scale version of an image, therefore before any computations are done, it is necessary to convert to gray scale.

2.2.1. Coarseness

The coarseness is the most fundamental textural feature. When two patterns differ only in scale, the magnified one is coarser. For patterns with different structures, the bigger its element size and/or the less its elements are repeated, the coarser it is felt to be.

The coarseness feature is computed as follows:

Step 1: The averages at every point are taken over neighborhoods whose sizes are powers of two, e.g. 2 x 2, 4 x 4, ..., 32 x 32. The average over the neighborhood size 2^k x 2^k at the point (x,y) is

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

where $f(i,j)$ is the gray-level at (x,y). In this project neighborhoods were calculated for k = 1,2 ,3,4, and 5. So at the end of this step, you end up with 5 matrices of the same size (in terms of pixels) as the image, one for each neighborhood size.

Step 2: For each point, the differences between pairs of averages corresponding to pairs of non-overlapping neighborhoods are taken just on opposite sides of the point in both horizontal and vertical orientations. For example the difference in the horizontal case is

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

After this step, the result is 10 matrices of the same size as the original image: 2 for each neighborhood size - one for horizontal orientation and one for vertical orientation.

Step 3: At each point, pick the best neighborhood size which gives the highest output value:

$$S_{best}(x, y) = 2^k$$

where k maximizes E in either direction, i.e.,

$$E_k = E_{max} = \max(E_1, E_2, \ldots, E_L)$$

If more than one neighborhood size has the maximum E, then the largest size is taken for S_{best}. Additionally, at each point, if there exist some k such that $k > k_{max}$ and $E_k \geq tE_{max}$, then the largest k for S_{best} is taken where t is a constant less than 1; otherwise the original S_{best} is kept. Tamura found that a value of t = 0.9 gave the best results, therefore this value was used in this project.

After this step, the result is one matrix of equivalent size to the original image where each element in the matrix is the maximum of all the $E_{k,h}$ and $E_{k,v}$ for that point, i.e. for each point a comparison is made to 10 values for that same point in the 10 matrices from step 2.

Step 4: Finally, the average of S_{best} over the image is taken to be the coarseness feature F_{crs}:

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

where m and n are the width and height of the image.

In all the calculations above, it is important to note that boundary strips of an image within the width of the largest operator size 2^L cannot be processed.

2.2.2. Contrast

The simplest method of varying picture contrast is by stretching or shrinking its gray scale. The contrast knob on a television set is a practical realization of this idea. Thus, based on this idea, when two patterns differ only in gray-level distribution, the difference between their contrast can be measured. However, it is not so simple as that because at least four factors can influence the contrast difference between two texture patterns. They are:

1. dynamic range of gray-levels,
2. polarization of the distribution of black on white on the gray-level histogram or ratio of black and white areas,
3. sharpness of edges, and
4. period of repeating patterns.

The contrast measurement used by Tamura approximated only factors 1 and 2, i.e. describing contrast in the narrow sense based on the gray-level distribution. However, good results were obtained so approximations of factors 3 and 4 were left to further studies.

The standard deviation σ about the mean of the gray-level probability distribution is used to approximate factor 1. It also reflects factor 2 to some extent because it is a measure of the dispersion of the distribution. For factor 2, a measure of polarization is required. The kurtosis α_4 is used for this purpose and is defined as follows:

$$\alpha_4 = \mu_4 / \sigma^4$$

where μ_4 is the fourth moment about the mean and σ^2 is the variance. These two approximations for factors 1 and 2 are combined to give a measure of contrast as follows:

$$F_{con} = \sigma / (\alpha_4)^n$$

where n is a positive number. Experimentally, Tamura found that $n = 1/4$ yielded the best results therefore the same value was used for purpose of this project.

2.2.3. Directionality

Directionality involves both element shape and placement. In [6], Tamura measures the total degree of directionality of the texture pattern and is not interested in the orientation of the texture pattern. In the work done by Tamara, two patterns which differ only in orientation have the same degree of directionality. It was felt that for this project, i.e. for the interior decorating application, it would not be sufficient to measure only the degree of directionality. In this application it is important that patterns which differ in orientation be seen as having different directionality. For this reason, a slight variation of the measurement defined in [6] was used. This was simply achieved by eliminating the last step of that procedure.

Directionality is measured using a histogram of local edge probabilities against their direction angle. It has been shown that this histogram represents sufficiently global features of the image such as long lines and curves. The directionality feature is computed as follows:

Step 1: The gray scale image is processed to detect edges using a 3 x 3 gradient edge detector. This method applies two 3 x 3 kernels to the neighborhood of each pixel to estimate the horizontal and vertical brightness differences, Δ_H and Δ_V. Conceptually, a kernel is applied to an image by sliding the kernel over the image and summing the products of the values in the kernel with the brightness values under them. The result is the derivative, or brightness slope of the pixel under the center of the kernel. For the purpose of comparison, two different operators are used in this project, the Sobel operator and the Prewitt operator. Both of these operators are shown below.

-1	0	1	1	1	1	1	0	-1	1	2	1
-1	0	1	0	0	0	2	0	-2	0	0	0
-1	0	1	-1	-1	-1	1	0	-1	-1	-2	-1
horizontal			vertical			horizontal			vertical		
Prewitt						Sobel					

Step 2: This method utilizes the fact that gradient is a vector, so it has both magnitude and direction. The magnate $|\Delta G|$ and the local edge direction θ are approximated as follows:

$$|\Delta G| = (|\Delta_H| + |\Delta_V|)/2$$

$$\theta = \tan^{-1}(\Delta_V/\Delta_H) + \pi/2$$

The resultant θ is a real number ($0 \leq \theta < \pi$) measured such that the horizontal direction occurs at these extremes 0 and π, the vertical direction occurs at $\pi/2$ and diagonal is everything in between.

Step 3: The directionality histogram H_D is obtained by quantizing θ and counting the points with the magnitude $|\Delta G|$ over the threshold t as follows:

$$H_D(k) = N_\theta(k) / \sum_{i=0}^{n-1} N_\theta(i), \qquad k = 0, 1, \ldots, n-1$$

where $N_\theta(k)$ is the number of points at which

$$(2k-1)\pi/2n \leq \theta < (2k+1)\pi/2n \quad \text{and} \quad |\Delta G| > t$$

Thresholding $|\Delta G|$ by t helps to prevent the counting of unreliable directions which can not be considered edge points. In [6], Tamura used $n = 16$ and $t = 12$ and found that the shape of each histogram H_D was not sensitive to the value of t. The same values were used for the purpose of this project.

In this project, a much smaller histogram is also computed which contains only 3 elements, each containing the percentage of bias in the horizontal, vertical, and diagonal directions respectively. This histogram is called a directionality vector to distinguish it from the histogram previously described. This is a very crude measurement of direction. The diagonal element lumps together any slopes to the left and right. This directionality vector was used in [5].

In summary, in this project four variations of the directionality feature are computed for each image: Prewitt-histogram, Sobel-histogram, Prewitt-vector, and Sobel-vector.

3. Image Retrieval

3.1. Distance Metrics

CBIR queries rely on similarity metrics rather than exact matching. Distance metrics produce a relative distance between two image features. For this project, distance metrics were chosen from the current research in CBIR technology. Some metrics are used for more than one feature and some features use more than one distance metric. The distance metrics used to calculate the differences between each of the features are summarized in Table 2. Each distance metric is described in detail in this section.

Feature Type	Feature	Distance Metric
Average Colour	RGB	L_2
	HSV	L_2
	YIQ	L_2
Moments of Colour Distribution	RGB	Weighted Sum (A,B,C)
	HSV	Weighted Sum (A,B,C)
Colour Histogram	HSV	L_1, L_2
	Cumulative HSV	L_1, L_2, L_∞
	Small RGB (8 bin)	L_2, Weighted L_2
	RGB (64 bin)	L_2, Weighted L_2
Texture	Contrast	Scalar
	Coarseness	Scalar
	Direction Vector	L_1, L_2
	Direction Histogram	L_1, L_2

Table 2. Distance metric to feature mapping

3.1.1. L_1 distance metric

The L_1 distance metric is used to compute the difference d between two n element vectors, e.g. histograms, H and I as follows:

$$d_{L_1}(H, I) = \sum_{j=1}^{n} |h_j - i_j|$$

In [8], Stricker states that a retrieval which is based on the L_1-distance produces many false negatives, i.e., it does not retrieve all the images with perceptually similar colour histograms. This happens because perceptually similar colour histograms may be a large L_1 distance apart from each other. Changes in lighting, which may result in a slight shift in the colour histogram, cause the L_1-metric to misjudge the similarity completely. Due to its unreliability, this metric was limited to computing the distance for the HSV colour histogram and the HSV cumulative colour histogram features, i.e. the same features that were used in [8].

3.1.2. L_2 distance metric

The L_2 (also known as Euclidean) distance metric is also used to compute the difference d between two n element vectors, e.g. histograms, H and I as follows:

$$d_{L_2}(H, I) = \sqrt{\sum_{j=1}^{n} (h_j - i_j)^2}$$

This metric can be used as the similarity metric for any feature whose value is a vector. In this project that includes the average colour, colour histogram and directionality features.

In [8], Stricker states that "Using a metric similar to the L_2-metric results in false positives, i.e., histograms with many non-zero bins are close to any other histogram and thus are retrieved always." What this means is that the L_2 metric only works well on sparse matrices. The metric introduced next, the weighted L_2 metric, attempts to improve upon the L_2 metric by introducing weights.

3.1.3. Weighted L_2 distance metric

The weighted L_2 distance metric used in [10] attempts to solve some of the drawbacks of the L_1 and L_2 metrics. A weight factor is used to put less emphasis on those bins with small or zero percentages and greater emphasis on those bins with significant content. This should reduce the problem of false positives due to contribution of insignificant differences between histograms.

This metric is used to compute the difference d between two n element colour histograms, e.g. H and I, as follows:

$$d_{wL_2}(H, I) = \sum_{j=1}^{n} w_j \sqrt{(h_j - i_j)^2}$$

where

$$w_j = \begin{cases} h_j \text{ if } h_j, i_j > 0 \\ 1 \text{ if } h_j \text{ or } i_j = 0 \end{cases}$$

where h_j and i_j are the j-th colour relative pixel frequency of the query and database images respectively, n is the number of colours in the histogram and w is the weight factor used. For a particular colour, if both the histogram bins are non-zero, the weight w_j used is h_j since we want to take the relative proportion of colour j in that image. If either of the corresponding histogram bins has a value of zero (which means the colour is absent) then w_j is 1. In this case the relative difference of the two bins is used as a push factor to separate the two images in the similarity metric.

This metric is used for the regular RGB histogram and the small RGB histogram.

3.1.4. Colour Similarity Matrix distance metric

The colour similarity matrix distance metric is described here however it has not yet been implemented in this project. It is on the list of enhancements described in Appendix B - Implementation Notes on page 69.

This metric, defined by Niblack *et al.* in [2], is used to measure colour histogram distance. It makes use of the colour similarities of the bins in the colour histograms. The metric is defined as follows:

$$d_{sim}(H, I) = \sqrt{(H-I)^T A (H-I)}$$

where T is the transpose operator. The entries a_{jk} in A describe the similarity between colour j and colour k.

With this metric we can correctly compute that orange images are similar to red images, and that a half-red/half-blue image is different from an all-purple one. This method accounts for both the perceptual distance between different pairs of colours (e.g. orange and red are less different than orange and blue), and the difference in the amounts of a given colour (e.g. a particular shade of red).

3.1.5. L_∞ distance metric

The L_∞ distance metric can be used to compute the difference d between two n element vectors, e.g. histograms, H and I as follows:

$$d_{L_\infty}(H, I) = \max_{1 \le j \le n} |h_j - i_j|$$

This metric was used in [8] to measure similarity between HSV cumulative colour histograms. This metric was also used for the HSV cumulative colour histogram in this project.

3.1.6. Weighted sum distance metric

The weighted sum metric is used to compute the similarity of the moments of colour distribution features. For example, if H and I are the colour distributions of two images with r colour channels (e.g. 3 channels if using RGB or HSV colour spaces) and the average, standard

deviation, and skewness of the two images are E_i, F_i, σ_i, ς_i, s_i and t_i respectively, then the weighted sum distance is computed as follows:

$$d_{mom}(H, I) = \sum_{i=1}^{r} (w_{i1}|E_i - F_i| + w_{i2}|\sigma_i - \varsigma_i| + w_{i3}|s_i - t_i|)$$

where $w_{kl} \geq 0$, $(1 \leq l,k \leq 3)$, are weights.

The weights in d_{mom} can be used to tune the metric for a given application. For example, often when working in the HSV colour space, it is desirable to match hue more strictly than the saturation and the value. This can be achieved by setting all the weights for the moments of the hue channel to a higher value than the other weights. The weight matrices used in this project are found in Table 3 and Table 4 below for the HSV and RGB colour spaces respectively.

	H	S	V		H	S	V		H	S	V
average	1	2	1	average	1	2	3	average	1	2	1
std. dev.	1	2	1	std. dev.	1	2	3	std. dev.	2	4	2
skewness	1	2	1	skewness	1	2	3	skewness	2	4	2
		A				B				C	

Table 3. Weight matrices for moments of HSV colour distribution distance computations

In the HSV colour space, since the range of the hue is between 0 and 2π, and the ranges of the saturation and the value are between 0 and 1, the weight matrices given in Table 3 emphasize the matching of the hue. The matrices in Table 3 are the same as those that were used in [8].

	R	G	B		R	G	B		R	G	B
average	1	1	1	average	1	1	1	average	1	1	1
std. dev.	1	1	1	std. dev.	2	2	2	std. dev.	2	2	2
skewness	1	1	1	skewness	2	2	2	skewness	3	3	3
		A				B				C	

Table 4. Weight matrices for moments of RGB colour distribution similarity computations

The weight matrices for the RGB colour space in Table 4 do not emphasize one colour component over another because each colour component has the same range. The matrices however do vary the emphasis between the average, standard deviation and skewness.

It is possible that two non-identical colour distributions have a similarity value of 0 because only a small subset of the moments of the colour distribution are used. A retrieval based on d_{mom} may

produce false positives because the moments contain no information about the correlation between the colour channels. Stricker [8] uses moments only for the purpose of indexing and not as a metric because of their limitations. In spite of this, I still decided to try using the moments for retrieval.

3.1.7. Scalar distance metric

The scalar distance metric is simply the absolute value of the difference between two scalar values, e.g. x, y, as follows:

$$d_s = |x - y|$$

This metric is used to measure the distance between the contrast and coarseness features which are both scalars.

3.2. Fast Searching

System performance must not slow down proportionately as image collections grow. To maintain performance indexing, clustering, and filtering should be designed into the matching methods. In a CBIR query, features from the database are compared to corresponding features from the query to determine which images are a good match. For a small database, sequential scanning of the features followed by similarity computations is usually adequate. However, as the database grows, this combination can be too slow. Two techniques, indexing and filtering, can be used to speed up queries.

3.2.1. Indexing

In the practice of CBIR, there are at least three difficult issues to overcome in the area of image feature indexing:

- Image features are typically of high dimension requiring complex high-dimensional indexing. Even efficient data structures for database indexing, like R^*-trees, work well up to 20 dimensions only. Data structures for fast access of high-dimensional features for spatial relationships must be invented.
- Traditional indexing assumes exact matching or range searches; similarity matching complicates the indexing structure.
- It is difficult to combine high-dimensional, similarity-based indexing methods to efficiently support queries composed of multiple features.

Given the complex nature and problems associated with image feature indexing, this project did not attempt to tackle indexing. Filtering, described in the next section, is much more feasible at this level.

3.2.2. Filtering

The idea behind filtering is that a computationally fast filter is applied to all images in the database and only items that pass through the filter are operated on by the second stage, which computes the true similarity metric. Filters should eliminate non-relevant images while ensuring that none of the relevant ones are dismissed.

Although no explicit filtering was implemented in this version of the project, two filters are considered for future work: average colour filter and largest bin filter.

Average Colour Filter

The average colour filter is applicable to a query by example that uses colour histogram matching. Filtering is done by first computing the distance between average colours followed by the evaluation of the histogram similarities only on those images that pass through the filter. The average colour distance metric requires only three multiplications, three subtractions, and two additions, i.e. it is independent of the number of histogram bins.

Largest Bin Filter

The largest bin filter can be used for indexing into a large database efficiently. This filter, called *Incremental Intersection* in [9], is also applicable to a query by example that uses colour histogram matching. The theory behind this filter is that most information is carried by the largest bins of the histogram. With this filter, only the largest bins from the example image and the database image are compared, and a partial histogram intersection value is computed.

In [9], Incremental Intersection is split into two phases, an off-line phase (i.e. at database population time) and an on-line phase (i.e. at query time). In the off-line phase:

1. For each colour in the histograms, group the bins representing that colour from each of the database image histograms.
2. Sort each group by the relative pixel count contained in the bins.

In the on-line phase:

1. Sort the query image histogram bins by the relative pixel count contained in the bins.
2. For the B largest query image bins, starting with the largest, match the query image bin to all the database image bins of the same colour that have the same relative pixel count or larger.

With the database of images in [9], it was found that after examining 10 bins, this method matched the query images to the database images without error. Most images contain at most five or six different colours, therefore the largest histogram bins capture a good percentage of the image contents, while the smaller bins are more likely to be noise.

3.3. Query Options

With CBIR, an image database can be queried using an example image as the basis for the query, i.e. query by example (QBE), or by specifying the features to match (e.g. average colour), which is referred to as query by picker (QBP) in this project.

Appendix A - Query Choices on page 65 summarizes the different query options available to the user in this project. The graph in that appendix should be read from left to right.

3.3.1. Query by Example (QBE)

The basis of a query by example is a sample image. The user can specify which characteristics (features) to match in the image and what weights to put on each feature as well as what distance metric to use when comparing the sample image to the database images.

This type of query is particularly useful when the user already has a sample swatch (e.g. a piece of upholstery) and would like to match it to other interior decorating items such as wallpaper or borders. In this scenario, the swatch could be scanned, digitized, and loaded into the CBIR application to be used as a sample image. The user can then select the feature(s) in that image to base the search on, e.g. the user may only be interested in matching the pattern (texture) in the image and not the colours.

3.3.2. Query by Picker (QBP)

Query by picker consists of the user selecting specific values of features to be matched. The user can specify queries such as:

- get all the images with approximately 20% green and 40% red, and I don't care about the other colours
- get all the images with horizontal stripes
- get all the images that are mostly black
- get all the images which have an average colour of blue.

The user makes selections by choosing colours from a colour chooser and/or choosing textures from a texture chooser. The texture chooser is a list of sample gray-scale images representing a variety of textures. The user can not actually choose specific values for texture features, e.g. a specific value for contrast, as these are difficult to comprehend and describe symbolically.

3.3.3. Query Refinement

Query refinement consists of selecting an image from the results of one query to be used as the example for a subsequent query. Query refinement provides the user with an easy way to keep narrowing down searches.

4. Design

This chapter provides a high level view of the design of the CBIR system built for this project. First, the programming language used to implement this project is described and an explanation is given as to why this language was chosen. The system architecture is then described. Following that, more details are provided on the database, the user interface and the query design.

4.1. Java

This project was implemented using Java. Java is a simple high-level object oriented programming language. Java is unusual in that each Java program is both compiled and interpreted. With a compiler, a Java program is translated into an intermediate language called Java bytecodes, i.e. the platform-independent codes interpreted by the Java interpreter. With an interpreter, each Java bytecode instruction is parsed and run on the computer. Compilation happens just once; interpretation occurs each time the program is executed.

Java bytecodes can be thought of as the machine code instructions for the Java Virtual Machine (Java VM). Every Java interpreter, whether it's a Java development tool or a Web browser that can run Java applets, is an implementation of the Java VM.

Java bytecodes help make "write once, run anywhere" possible. A Java program is compiled into bytecodes on any platform that has a Java compiler. The bytecodes can then be run on any

implementation of the Java VM. For example, the same Java program can run on Windows NT, Solaris, and Macintosh.

Java also includes the Java Application Programming Interface (Java API). The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries (packages) of related components.

As a platform-independent environment, Java can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time bytecode compilers can bring Java's performance close to that of native code without threatening portability.

Java was chosen for this project because it helped to do the following:

- Write once, run anywhere: 100% Pure Java programs run consistently on any platform with a Java interpreter because they are compiled into machine-independent bytecodes.

- Get started quickly: Although Java is a powerful object-oriented language, it is easy to learn, especially for programmers already familiar with C or C++. Before starting this project, I was familiar with C and C++ but had no experience with Java.

- Write better code: The Java language encourages good coding practices, and its garbage collection helps to avoid memory leaks. Java's object orientation and its wide-ranging, easily extendible API let you reuse other people's tested code and introduce fewer bugs.

- Develop programs faster: Development time may be as much as twice as fast versus writing the same program in C++ because fewer lines of code are written with Java and Java is a simpler programming language than C++. Comparisons of program metrics (class counts, method counts, and so on) suggest that a program written in Java can be four times smaller than the same program in C++.

Appendix B - Implementation Notes on page 66 contains the version number of the Java Development Kit, Java Virtual Machine, and Java APIs used in the implement the CBIR system for this project.

4.2. Architecture

Figure 3 on page 31 illustrates the architecture of the CBIR system implemented for this project.

The system consists of two programs. The first, CBIR, is used to execute content based queries on the database of interior decorating images. The second, CBIR Database Loader, is used to populate the database in batch mode. The execution of these two programs start in the **main** methods of the **CBIR** and **CBIRdbLoader** classes respectively. Only one of these programs should run at any given time because only one program should access the database at a time to avoid corruption of the database.

4.2.1. Programs

CBIR

The CBIR program is implemented by the **CBIR** class which contains only a **main** method to start the execution of the Content Based Image Retrieval application. The core application managers and the user interface windows are created. The core application managers, which are described later in this section, consist of the CBIR Engine, the Image Analyzer, the Query Manager, the Storage Manager and the Distance Measurer. The user interface windows, described in detail in the User Interface section on page 35, consist of the main window containing the Search by Example and Search by Picker panels, the Results Displayer and the Choices Displayer. Of the user interface windows created, only the main window is shown to the user at start up.

When the CBIR program is run, on the command line you must specify the path to the CBIR database and yes or no indicating whether or not to run in debugging mode.

CBIRdbLoader class

Similar to the CBIR program, the CBIR Database Loader is implemented by the **CBIRdbLoader** class which contains a **main** method to start the execution of the batch mode CBIR database population program. Unlike the **main** method in the **CBIR** class, the **CBIRdbLoader main** method only creates those core application managers required for database population, namely the Image Analyzer and the Storage Manager, and no user interface windows are created.

The **CBIRdbLoader** class contains a method, **loadCBIRdb**, which is invoked to populate the CBIR database in batch mode. When the CBIR Database Loader is run, on the command line you

must specify the path to the source of images, the path to the CBIR database and yes or no indicating whether or not to run in debugging mode.

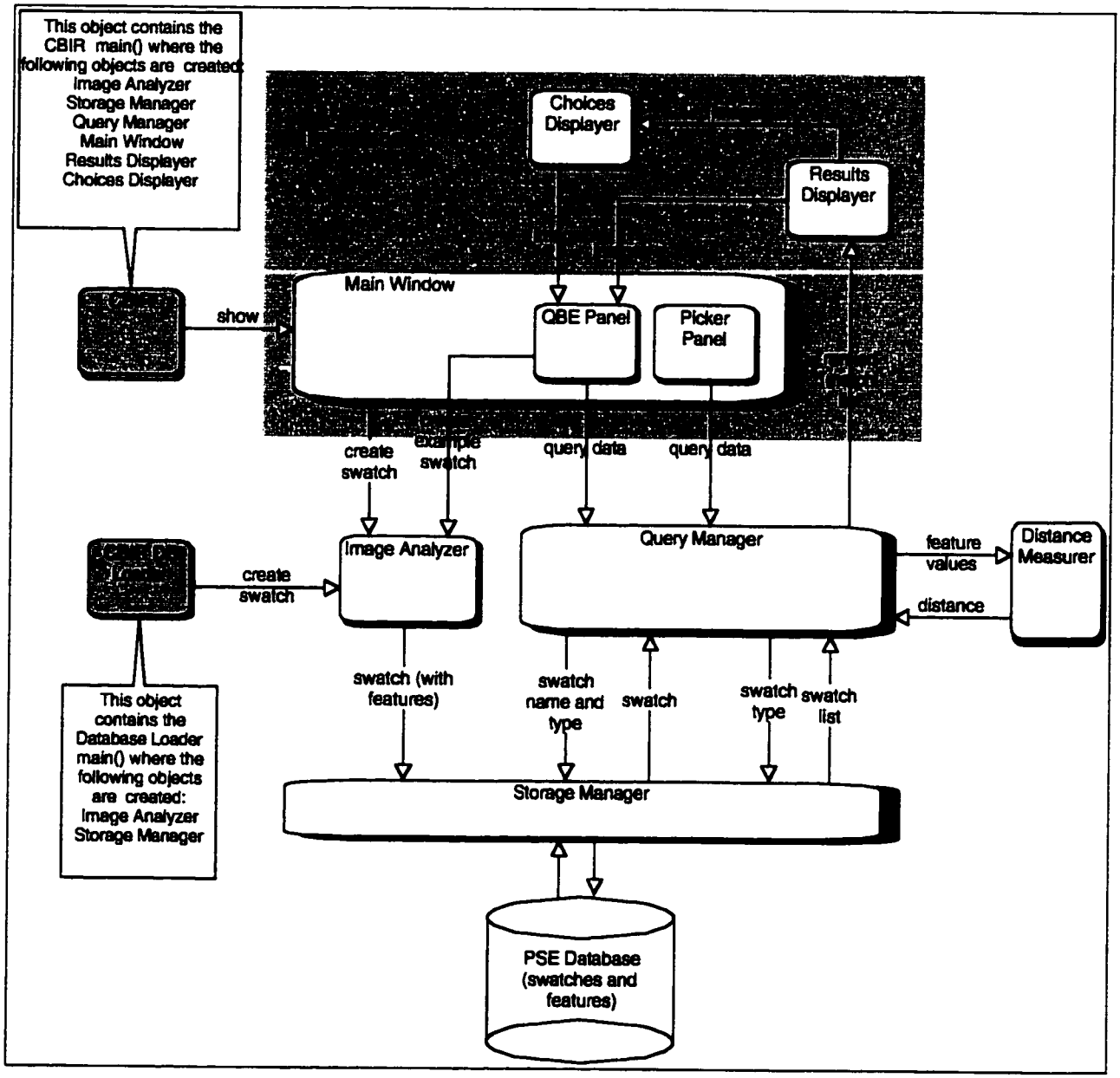


Figure 3. CBIR system architecture

4.2.2. Swatches and Features

Not shown in Figure 3 on page 31 are the **Swatch** and **Feature** objects which are described below. It is the Swatch and Feature objects that are stored in the CBIR database.

Swatches

A swatch is a sample item (image) used for interior decorating and is implemented by the **Swatch** class. A Swatch object consists of an image name, the type of swatch (e.g. wallpaper or border) and a list of image **Feature** objects.

Features

Features are implemented by an abstract parent class, **Feature**, and many subclasses, one for each type of feature described in the Features section on page 7. A feature object consists of an attribute which represents the value of the feature for the **Swatch** in which it is contained and a **calculate** method which contains the algorithm to calculate the feature.

4.2.3. Core Application Managers

CBIR Engine

The CBIR Engine, implemented by the **CBIRengine** class, is the most central object of the application. When the CBIR Engine object is created, the main application window is created. The CBIR Engine handles all menu item selections in the main window. The **CBIRengine** class contains many static attributes (e.g. one for each core application manager and main user interface components) which are used by the other objects in the application to communicate amongst each other.

When the CBIR program is run, a **CBIRengine** object is created. When the CBIR Database Loader is run, a **CBIRengine** object is not created because no user interface is required, but the static attributes of the **CBIRengine** class are still available to allow communication between the core application managers.

Image Analyzer

The Image Analyzer, implemented by the **ImageAnalyzer** class, is the core application manager responsible for the creation of the **Swatch** objects and the **Feature** objects. When a **Swatch** is created, a **Feature** object is created for each of the image features and the features are calculated.

Requests to create a **Swatch** can come from the CBIR Database Loader while loading the database in batch mode and from the CBIR Engine when the user has selected the **Add image** menu item or the **Load example** menu item in the main window. If the **Swatch** is to be stored in the database, then the Image Analyzer invokes the Storage Manager to store the **Swatch** in the database.

Storage Manager

The Storage Manager, implemented by the **StorageManager** class, is the core application manager responsible for the interface to the database of swatches. At program startup, the Storage Manager is invoked to open the database. At application exit, the Storage Manager is invoked to close the database. The Storage Manager can be requested to store a swatch, retrieve a particular swatch or retrieve a list of swatches of a given type.

Query Manager

The Query Manager, implemented by the **QueryManager** class, is the core application manager responsible for the execution of queries on the CBIR database. Based on the user's selection of features and distance metrics in the Search by Example or Search by Picker panels, the Query Manager executes the query. The Query Manager uses the Storage Manager to retrieve swatches from the database, the Distance Measurer to measure the similarity between query features and database swatch features, and the Results Displayer to display the results of the query.

Distance Measurer

The Distance Measurer, implemented by the **DistanceMeasurer** class, is responsible for the calculation if the similarity between query features and database swatch features. The DistanceMeasurer class is an abstract class which contains only static methods, one for each of the distance metrics described in the Distance Metrics section on page 19.

4.3. Database

This section describes the underlying database management system used in this project as well as the contents of the database.

4.3.1. Persistent Storage Engine (PSE)

The underlying database used in this project is called Persistent Storage Engine (PSE) by Object Design, Inc.. PSE is a persistent storage engine for Java. It provides persistent storage for Java objects. Persistent data is available to programmers in such a way that it appears as familiar, normal Java objects. PSE was chosen as the underlying database because with this database persistent Java objects and regular Java objects are manipulated in the same way and behave in the same way. The result is that storage and retrieval of objects is almost transparent. When retrieving an object from the database you do not have to read a record from a table and then create the object that the record represents as would be required in a traditional relational

database. Similarly, when storing an object, you do not have to create a record from the object and then insert it into a table as would be required in a traditional relational database. With PSE, to store an object you simply have to make the object persistent by making it accessible from another persistent object. Database roots, which are simply objects designated as a root, provide starting points for navigation among persistent objects.

PSE is designed for applications that require persistent Java support for as much as 100 MB of data accessed by one user. The PSE runtime library is written entirely in Java. It uses approximately 300 KB of disk space and runs entirely within the application process. It supports transactions and provides access to objects without reading the entire database.

PSE provides an application programming interface (API) that allows a program to

- Create, open, close, and destroy databases.
- Start, commit, and abort transactions.
- Read and write database roots.
- Store objects in a database and retrieve and update those objects.
- Store collections of objects with indexed look-up.

A PSE database consists of two files, <db name>.odb and <db name>.odt. PSE uses the standard Java file input/output API to access a database. There is no client/server distinction. There is no PSE server. PSE is just accessing a file on the file system. PSE has no special privileges. If PSE can read or write a database file, then so can any other application using ordinary file I/O.

4.3.2. Database Content

The database contains an updatable collection of swatches. Table 5 summarizes the type and number of the different swatches stored in the database.

Swatch Type	# of Swatches
wallpaper	170
border	165
fabric	200
carpet	21
flooring	80
Sub-total	**636**
texture	45
Total	**681**

Table 5. Database swatch content breakdown

The swatches of type wallpaper, border, fabric, carpet and flooring represent the images that are searched during a query. The actual images which the swatches represent are not stored in the database, only the name of the image, which is the same as the image file name, is stored.

The database also contains swatches of type texture. Texture swatches are not searched during queries but are used as sample textures which can form the basis of a query by picker using texture features.

The images in the database are large enough to contain sufficient legible detail to represent the content unambiguously. The maximum image width and height allowed is 270 and 215 pixels respectively. Both giff and jpeg image formats (file extensions .gif and .jpg) are accepted by the application. Currently the database only contains references to jpeg images because jpeg images tend to require less storage space. The storage space required by the images currently referenced by the database ranges from about 2K to 12K depending on the image size and content detail.

The database files (cbir.odb and cbir.odt) require approximately 2.5 megabytes of storage space and the images themselves require approximately 2.7 megabytes of storage space for a total of approximately 5.2 megabytes.

4.4. User Interface

The user interface (UI) is composed of three windows (Main, Results, Choices) and two choosers (Colour, Texture). Each one is described below.

4.4.1. Main Window

The main window is composed of two panels, the Search by Picker panel (captured in Figure 4) and the Search by Example panel (captured in Figure 5). You can toggle between the two panels by clicking on the tabs at the top of the window.

The main window contains four menus: **File**, **Image**, **Window**, and **Help**.

Currently, the only item implemented on the **File** menu is **Exit** which exits the application. There are two items on the **File** menu which are disabled, **New** and **Open**. These menu items will be used for a future enhancement which will provide the capability to save and retrieve a set of images chosen by the user from query results (see Application Enhancements on page 69 in Appendix B for more details).

The Image menu contains three items: **Add**, **Remove** and **Load example**. **Add** can be used to add a new swatch or a new texture sample to the database. The user is prompted for the name of the image to be added. **Remove** is disabled because it is not currently implemented. It is intended to provide the capability to remove a particular swatch or texture sample from the database. **Load example** can be used to load an image to be used as the example image in a query by example.

The **Window** menu contains two items: **Results** and **Choices**. Selecting one of these items will bring the corresponding window of the same name to the front.

The **Help** menu contains three items: **Topics**, **Using Help**, and **About CBIR**. All of these help items are disabled. Currently, the only on-line help available in the application is the tool tips that pop up when the mouse pointer passes over a user interface component. The **Help** menu items will be implemented as part of a future enhancement.

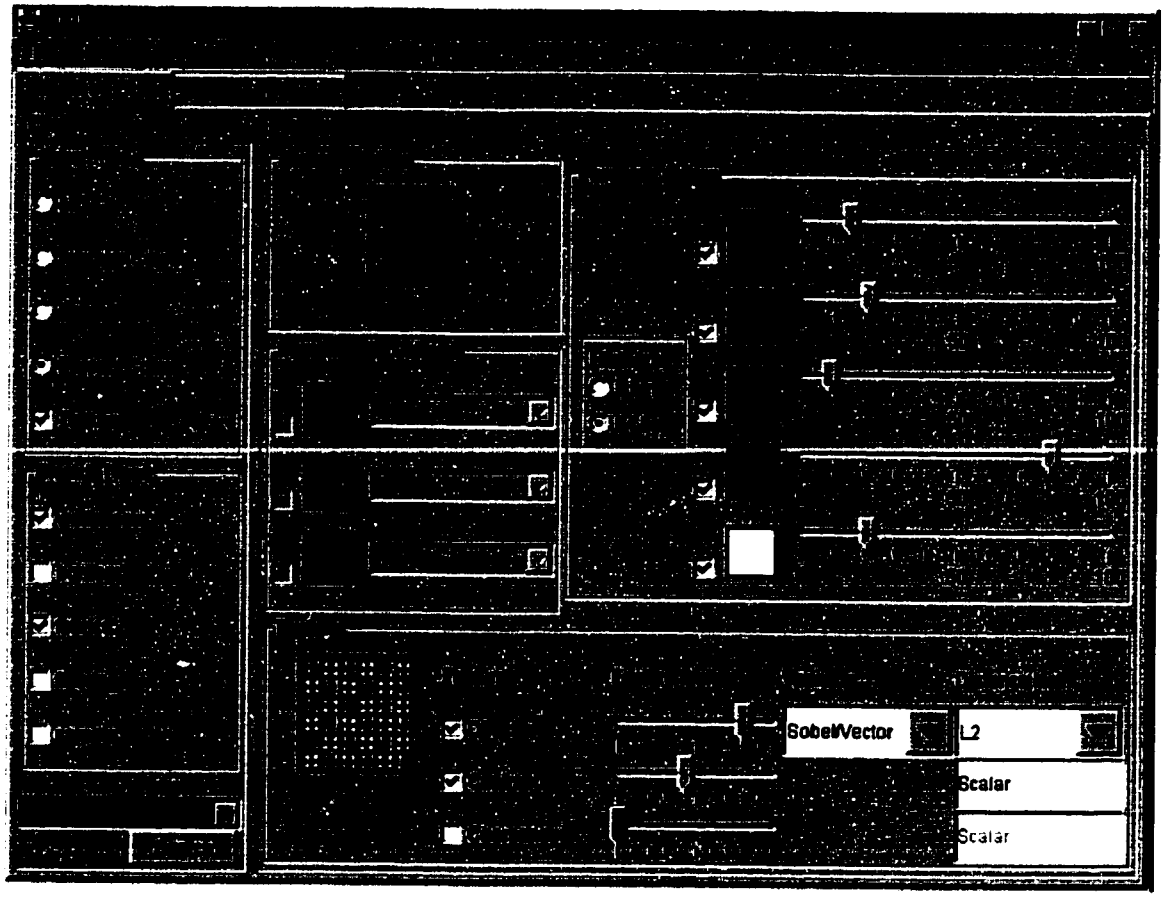


Figure 4. Screen capture of Search by Picker panel in the main window

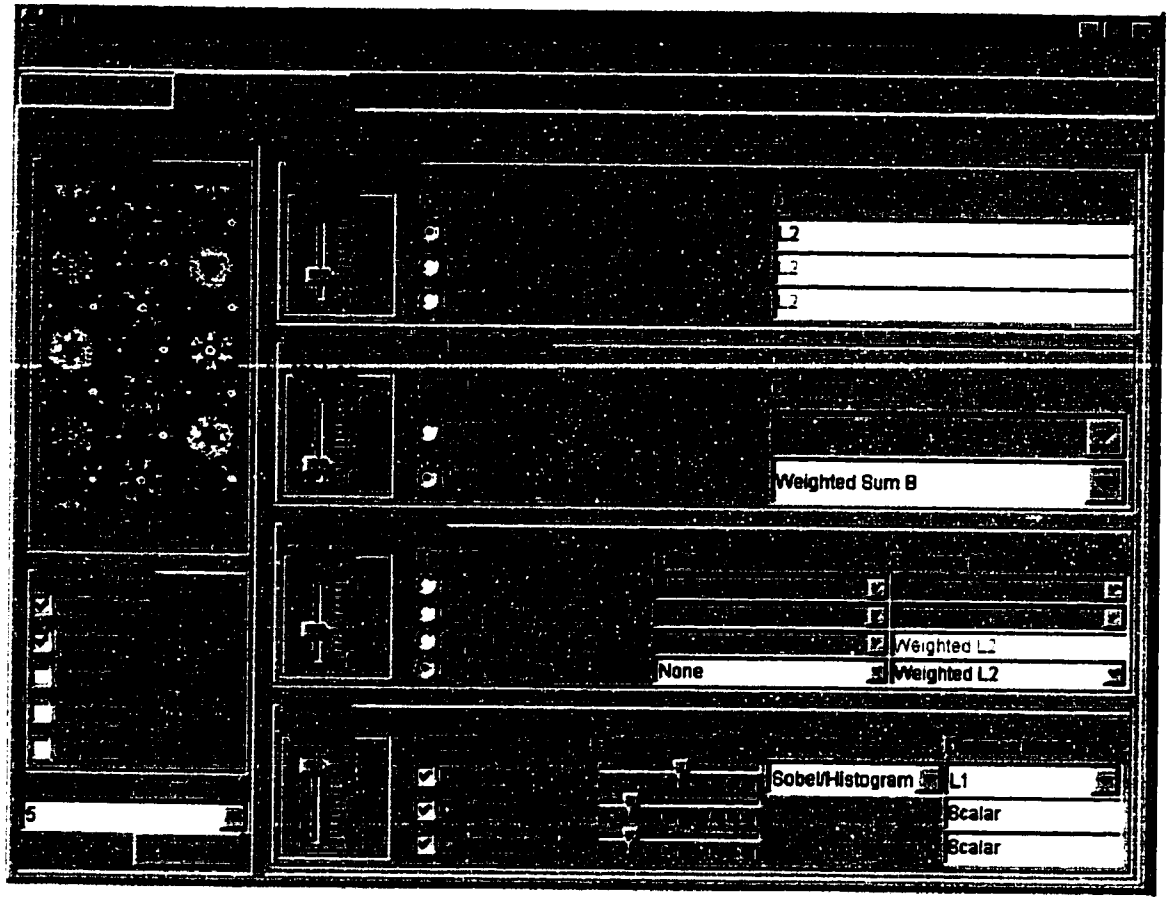


Figure 5. Screen capture of the Search by Example panel in the main window

4.4.2. Search Panels

The Search by Picker and Search by Example panels are implemented by the PicPanel and QBEPanel classes respectively. Figure 4 and Figure 5 contain screen captures of these panels. These panels contain all the options available for the respective query type. The user must choose what types of swatches to search, which features to search on, and in some cases which variation of the feature to use and which distance metric to use to measure the similarity between the query features and the database swatch features.

In both panels, before clicking on the search button to execute the query, the user must select which swatch type(s) to search and the number of matches to return (of each swatch type). In both panels the user can choose to search by colour and/or texture.

Search by Picker Panel

In the Picker panel, searches are based on the specific feature values selected by the user. This is done by clicking on one of the colour choices and/or the texture choice in top left of the panel. Within the colour category, the user can search by choosing the average colour, or up to 3 colours which the image must contain some or mostly of, or the percentage of up to 5 colours that the image must contain. For percent colour, the user has the option of choosing to use the 8 bin RGB colour histogram or the 64 bin RGB colour histogram. To select a specific colour or texture, the user clicks on the appropriate blue outlined square and the corresponding chooser (either Colour or Texture) pops up (see Choosers on 42 for more details). Once the user has selected a colour or texture, it is displayed in the blue outlined square that was clicked on.

Important: It is not possible to make any selections in the picker areas of the panel until the corresponding search type is selected in the top left of the panel. Similarly, the blue outlined squares in the Some/Mostly and Percent Colour picker areas are disabled until the check box immediately beside them is selected. As well, the check box immediately beside the different texture features must be selected to search by that feature and to change its weight, variation and/or distance metric.

Search by Example Panel

In the Search by Example Panel, searches are based on an example image selected by the user. This can be done by loading a new image (Load Example option on the Image menu) or by selecting an image displayed in the Results or Choices windows to be used as an example (see Results and Choices Windows on page 40 for more details on how this is done).

The user must also select which features in the example image to search by: average colour, moments of colour distribution, colour histogram, and/or texture. For the colour features, the user must choose which variation of the feature to use in the search and which distance metric to use. Within texture, the user can choose to search by direction, contrast, and/or coarseness. Similar to the colour features, the user can choose the variation of the feature and distance metrics to be used for the texture features. The weight sliders are used to vary the emphasis put on that particular feature during the search.

Important: To search by a particular feature type, the weight must be greater than zero. Similarly, to search by one or more texture features, their weights must also be greater than zero.

4.4.3. Results and Choices Windows

The Results and Choices windows are used to display images representing swatches. Both windows contain a scroll pane for each swatch type. Figure 6 on page 41 shows an example of the Results window displaying the results of a query that searched for images with an average colour of blue.

The Results window is used to display the query results. The Choices window is used to display images that the user has selected to save from the Results window. Each time a query is executed, the retrieved images are displayed in the Results window, overwriting any previous results. Note that only the previous results of the swatch types selected for the current query are overwritten. The user can copy any image from the Results window to the Choices window. This process saves them so that they can be compared with subsequent query results or used as inputs to subsequent queries.

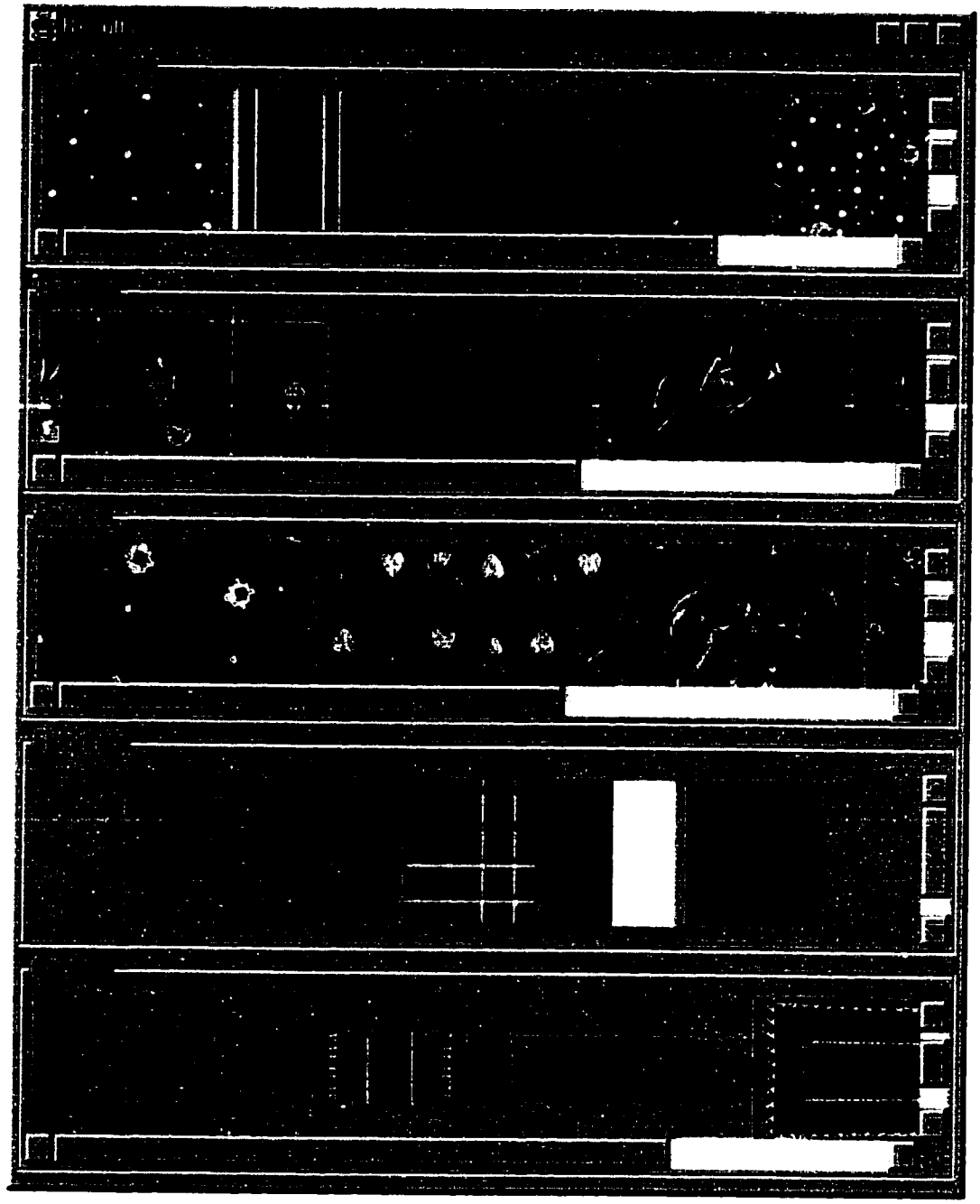


Figure 6. Example of Results window (query: average colour = blue)

The images in the Results and Choices windows are active menu buttons that can be clicked to pop up a menu with a list of options. The image popup menu in both the Results and Choices windows has an option called "Example". Selecting this option will copy the image to the example image area in the Search by Example panel where it can be used as the example for the next query.

The popup menu in the Results window also gives the user an option called "Hold" which copies the image to the Choices window. The popup menu in the Choices window also gives the user the option called "Remove" which deletes the image from the Choices window.

The popup menu in both the Results and Choices windows also contain a disabled option called "Fullsize". The intent of this option is to display a full size version of the image. This option has not been implemented yet (see Application Enhancements in Appendix B on page 69).

4.4.4. Choosers

A chooser pop ups when the user clicks on a blue outlined square in the Search by Picker panel. There are two types of choosers, Colour and Texture, which are described in this section.

Colour Chooser

The Colour chooser is implemented using the ready made JColorChooser class that is provided in the Java API. The Colour chooser provides the option of specifying a colour by choosing colour components in the HSV or RGB colour spaces. Figure 7 below shows screen captures of both options.

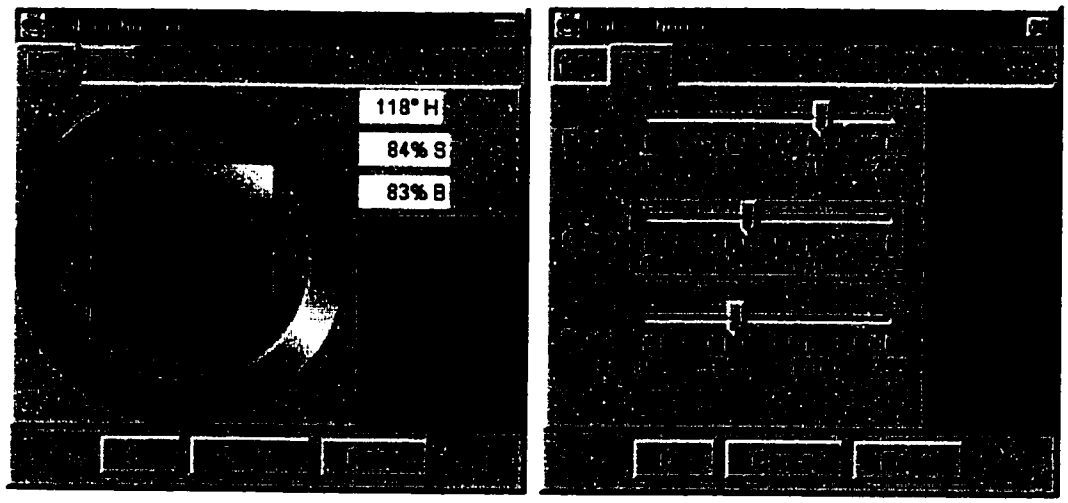


Figure 7. Colour chooser

Texture Chooser

The Texture chooser, implemented by the **TexturePicker** class, consists of a list of sample gray scale images which represent a variety of textures. The images are displayed in a scroll pane. The user selects a texture sample by clicking on the image. Figure 8 on page 43 is a screen capture of

the Texture chooser. Sample textures can be added to this list by selecting the **Texture** option from the **Add** menu item on the **Image** menu in the main window.

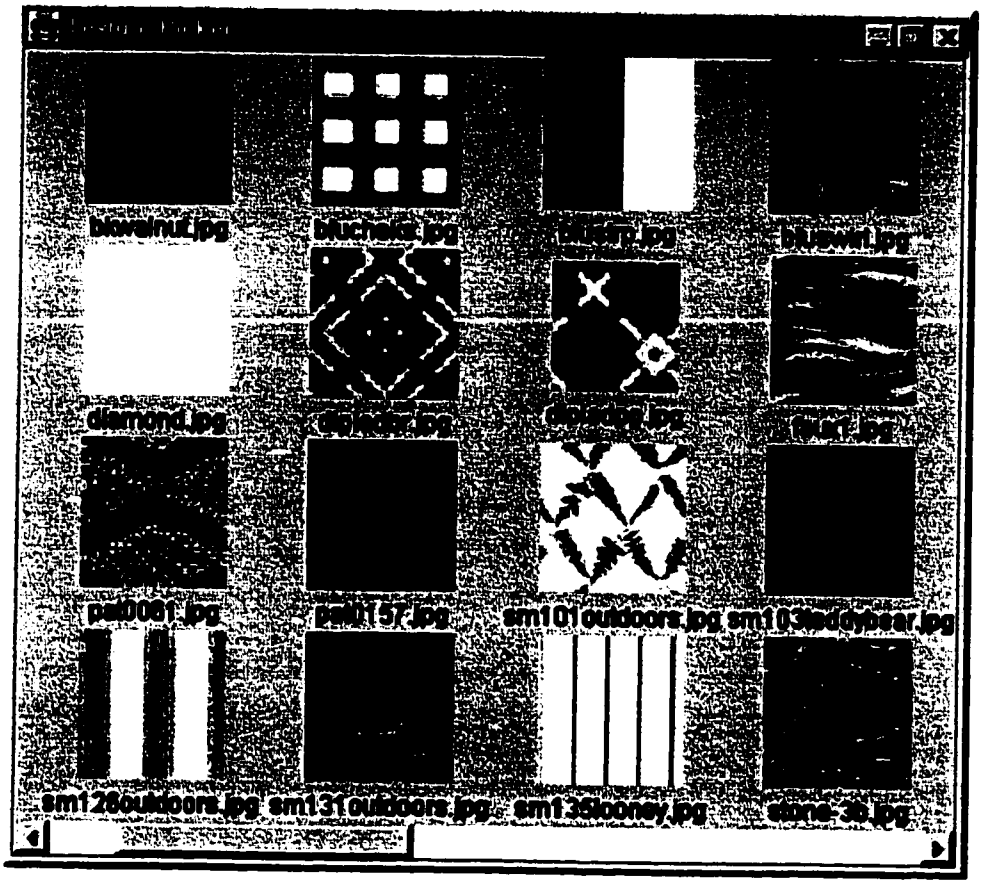


Figure 8. Screen capture of the Texture chooser

4.5. Query Design

This section provides some miscellaneous details regarding how the queries are implemented. These details are important to know when executing a query and analyzing the results of that query.

4.5.1. Weights and Total Distance

When measuring the similarity between the query features and the features in a database swatch, the total distance is a weighted combination of the individual colour and texture distances. A weight is assigned to a feature by using the weight slider associated with that feature. The higher the weight, the more emphasis will be put on that feature during the search because the distance

measure of that feature will be multiplied by its weight before being added to the total distance. If a weight is 0 then that feature will not be considered in the query.

4.5.2. Query Results

Query results are displayed in order of matching commencing with the best match first. A best match is the one with the lowest total distance. There are a couple of cases however where it is not necessarily the best matches that are displayed. This occurs when the query consists of Some/Mostly colours or Percent colours only. For both of these query types, an image either meets the criteria specified or does not. There is no distance metric associated with these types of queries. In these cases, the query searches for all the swatches that meet the criteria and displays only the maximum number of matches requested by the user. Note that if n is the number of matches to display, it is the last n swatches found meeting the criteria which are displayed.

If texture is also being queried along with Some/Mostly colour or Percent colour then all the swatches meeting the colour criteria are passed on to the texture part of the query which are then ranked by the texture distance.

Some/Mostly Colour Queries

An image meets the criteria specified in a Some/Mostly colour query if for each colour specified as Some (up to 3 colours) and each colour specified as Mostly (only 1 colour allowed) the corresponding bin in the colour histogram containing that colour has a relative pixel frequency between 2% and 20% for Some and between 50% and 100% for Mostly. The small RGB colour histogram is used as the basis of this query type because the intent of this query type is to narrow down the search to a broad range of colours and not to be too specific. For more specific queries, the Percent colour query should be used.

Percent Colour Queries

An image meets the criteria specified in a Percent colour query if for each colour specified (up to 5 colours) the corresponding bin in the colour histogram containing that colour has a relative pixel frequency within +/- 10% of the selected percentage and no less than 2% for that colour.

The small RGB colour histogram or the 64 bin RGB colour histogram is used as the basis of this query type. The user chooses histogram to use. The HSV colour histogram is not appropriate for this type of query because the HSV histogram bins are broken up by colour component, i.e. 16 bins for H, 4 bins for S, and 4 bins for V, which is not intuitive at all for the user. You can not

easily pick a colour and then determine what bin it belongs in because the colour is distributed over 3 bins.

Number of Matches

The user can specify that the query return a maximum of 1, 5, 10, 15, 20 or 25 matches. The selected number of matches to return applies to each swatch type selected. For example, if the user specifies that 5 matches be returned and that the wallpapers and borders are searched then the 5 best matched wallpapers and the 5 best matched borders will be returned. With Some/Mostly and Percent colour queries, it is possible that there are not as many swatches meeting the specified criteria as the number of matches requested to be returned. In a case like this, only those swatches meeting the specified criteria are returned.

5. Discussion of Results

The purpose of this section is to discuss the performance and usability of the CBIR application in this project as it is applied to the interior decorating domain.

Performance is made up of two aspects: retrieval accuracy and efficiency. Each of the swatch features and distance metrics will be discussed with regards to these two performance aspects.

Usability is discussed briefly with regards to the use of combinations of different features in queries.

5.1. Retrieval Accuracy

Retrieval accuracy refers to the relevance of retrieved images to a query as perceived by a user. This can be quite subjective. For example, one user may believe that an one image is more coarse than another image while another user may perceive it the opposite way.

5.1.1. Colour

Average Colour

In a query by picker, searching by average colour seems to produce good results when there are images in the database that have a high percentage of colours close to the average colour chosen. Otherwise, you get some unexpected results. For example, Figure 9 shows the results of a query

for wallpapers with the average colour of pure red (i.e. R=255, G=0, B=0). Figure 10 shows the results of a query for wallpapers with the average colour of pure green (i.e. R=0, G=255, B=0).

Figure 9. Query by picker for wallpapers - average colour = red

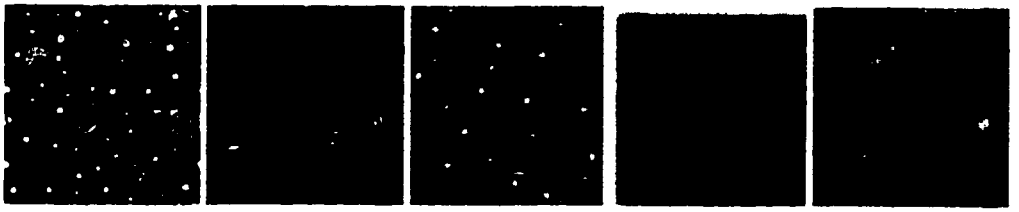
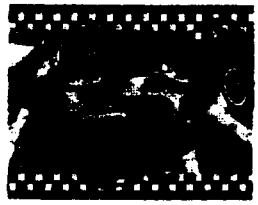


Figure 10. Query by picker for wallpapers - average colour = green

There are many wallpapers in the database that are mostly red so the first query returns good results. However, there are not that many wallpapers in the database that are mostly green so for the second query, the last image appears to have virtually no green but the average of its colours is close to green. This is not obvious at all for a user.

In a query by example, using average colour can be even more counter-intuitive in some situations because the query is dependent on the example image chosen. If the example image is mostly one colour, then it is usually obvious what the average colour is. However, if the example image contains many colours, none of which stand out from the rest then it is not obvious what the average colour is and hence it is not obvious what is actually being searched for. For example, Figure 11 shows the best 5 border matches for a query by average RGB colour using the image that is displayed in the top part of the figure as the example.



Example Image

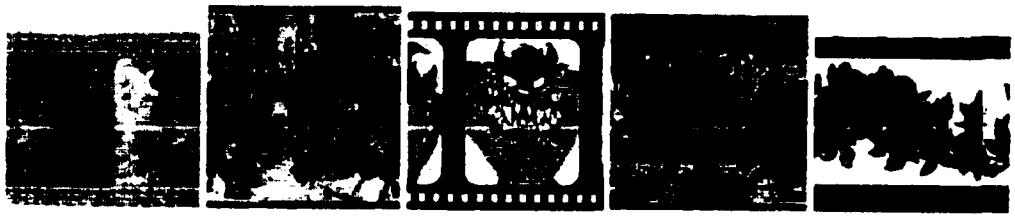
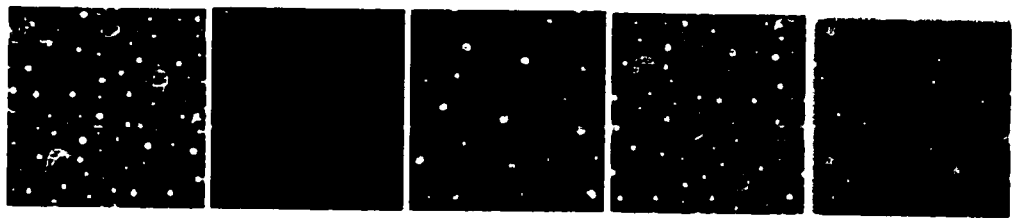


Figure 11. Query by example on borders - average RGB colour

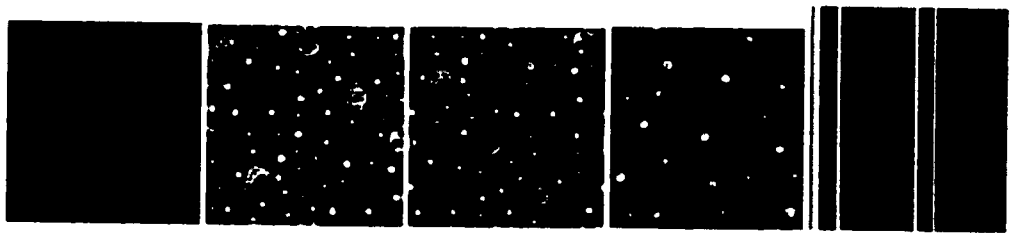
You can see from the results in Figure 11 that only the fourth image from the left seems to have the same colours as the query image. However, the first three images have a closer average colour.

The performance of the average colour features in the three different colour spaces were also compared. Figure 12 shows sample results obtained from three average colour queries based on the same example image but using different colour spaces.

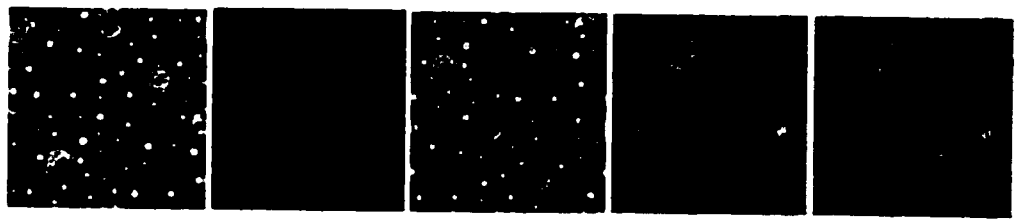
Example Image



RGB Colour Space



YIQ Colour Space



HSV Colour Space

Figure 12. Query by example - average colour in RGB, YIQ, and HSV colour spaces

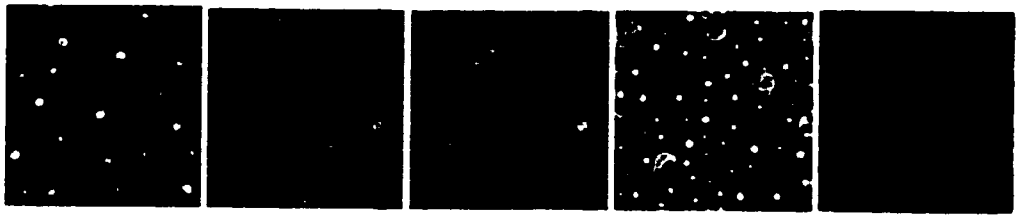
As the example in Figure 12 demonstrates, average colour queries using different colour spaces tend to return the same best matches, at least for the first couple of matches. Beyond that there is some variation. These results are quite consistent. I have been unable, however, to see any pattern with regards to what each of the colour spaces seem to focus on. Therefore, I am unable to draw any conclusions with regards to which of the colour spaces is most suitable for an interior decorating application.

All average colour queries use the L_2 distance metric, therefore no distance metric comparisons were made for average colour.

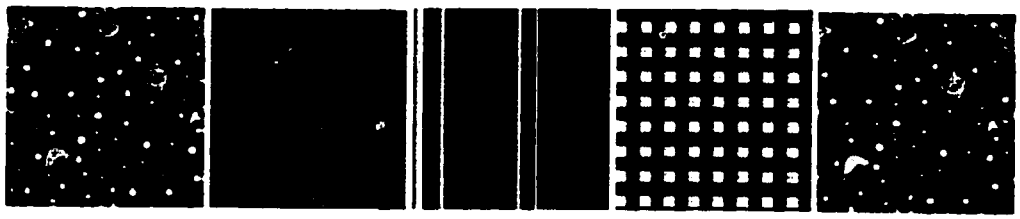
Moments of Colour Distribution

For the moments of colour distribution features, comparisons are made between the results using the RGB versus HSV colour spaces and within one colour space comparisons are made between the different weighted sum distance metrics that are used. Figure 13 shows results of a sample moments of colour distribution query using the same example image that is shown in Figure 12.

The top half of Figure 13 shows that the query results for the weighted sum B and C in the RGB colour space which were the same. The only difference between weighted sum B and C is that C puts a little more emphasis on the skewness. The same top 5 images were also returned but in a different order when the weighted sum A was used. Weighted sum A puts equal emphasis on the all three moments while weighted sums B and C put less emphasis on the average.



RGB colour space - Weighted sum B & C



HSV colour space - Weighted sum A

Figure 13. Query by example - moments of colour distribution in RGB & HSV colour spaces

The bottom half of Figure 13 shows the query results for the weighted sum A in the HSV colour space. The top 5 matches using weighted sum C were almost identical except the positions of the 4th and 5th images were swapped. Using weighted sum B, four of the same top 5 matches were returned but in a different order. The 5th match above (mostly red one) was ranked 2nd when

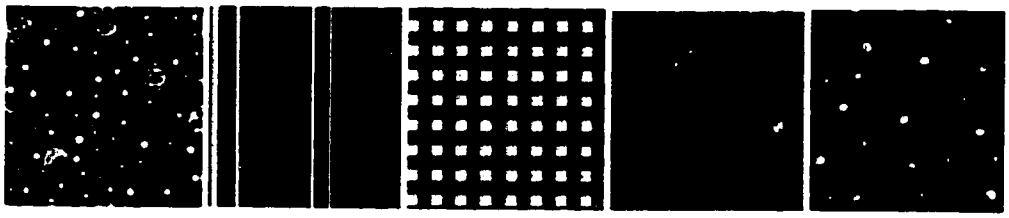
weighted sum B was used. Weighted sum A puts a greater emphasis on the saturation component while weighted sum B puts even more emphasis on the value component.

Notice that the images for both the RGB and HSV colour spaces are not the same as what was returned for the average colour queries in Figure 12. This is because of the standard deviation and skewness factors that are introduced as well as the average. The average colour focuses on the actual colours in the image while the standard deviation and skewness are a measure of the range of colours and the symmetry of the colour distribution, both of which are independent of the actual colours in the image.

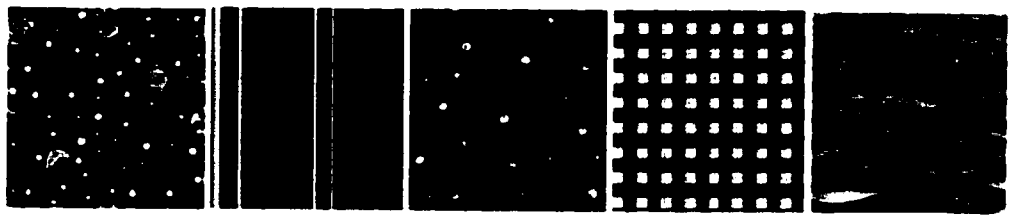
At first I found the 5th match to be quite surprising because it is mostly red and not blue. I then realized that the standard deviation and the skewness are virtually independent of the actual colour of the image. So when greater emphasis is put on these features, the results seem to be less and less what one would expect. For this reason, I do not believe that the moments of colour distribution features would be very effective in an interior decorating application. They are more suited to finding images that are the same.

Colour Histogram

Four colour histograms were tested: HSV, cumulative HSV, small RGB, and RGB. Figure 14 shows the results of a sample colour histogram query using the same example image that is shown in Figure 12.



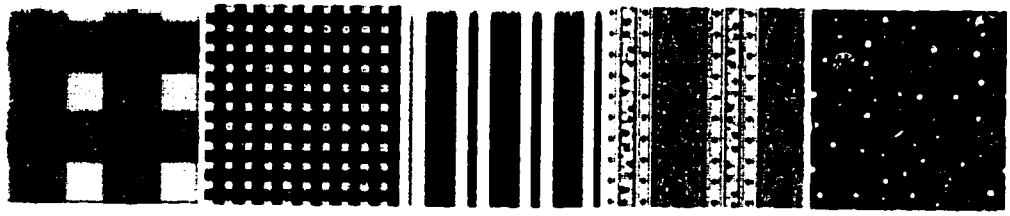
HSV colour histogram - L_1 distance metric



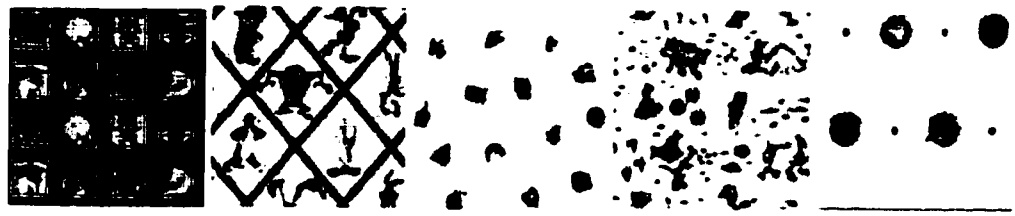
Small RGB colour histogram - L_2 distance metric

Figure 14. Query by Example - colour histograms - top 5 matches

The top 5 matches for this query were almost all the same for the four types of colour histograms regardless of which distance metric was used. Significant differences in the results only started occurring after the top 5 matches, i.e. matches #6 to #10. Figure 15 shows some sample results of matches #6 through #10.



RGB colour histogram - L_2 distance metric



RGB colour histogram - Weighted L_2 distance metric

Figure 15. Query by Example - colour histograms - matches #6 to #10

You can see in Figure 15, that the weighted L_2 distance metric produces much better results than the simple L_2 metric for the RGB colour histogram when you are comparing histograms that are less and less similar. It seems that the weighted L_2 distance metric really does put more emphasis on those bins with significant content in the query image. As you descend the ranked list of matches, the matched images contain the same colours but not necessarily in the same proportion as the query image. This observation was also consistent when the small RGB histogram feature was used.

Overall, the HSV histograms (regular and cumulative) performed well even when comparing histograms that are less similar (i.e. matches #6 to #10 in the example above).

Some/Mostly

The Some/Mostly colour queries performed well as long as you are conscious that the underlying feature used is the 8 bin RGB colour histogram. Because the RGB colour space is only separated into 8 colour bins, a lot of colours fall into one bin. This is most deceiving in the case of the colours white and black. A lot more colours fall into the white bin and the black bin than you

would originally expect. However, keeping this in mind, the Some/Mostly queries can be quite useful when first starting a search.

One disadvantage however with this type of query is that there is no ranking of the results that are returned. Instead, a list of unordered images are returned that meet the criteria specified by the user. Only up to the number of matches requested are returned, however, there may be many more images in the database that meet the criteria.

Percent Colour

The Percent colour queries perform very well. Again, if you have chosen to use the 8 bin RGB colour histogram as the underlying feature, you must be conscious of the fact that quite a wide range of colours are associated with each bin. This is not the case, however, if the 64 bin RGB colour histogram is used.

Percent colour queries suffer from the same disadvantage as the Some/Mostly queries, i.e. there is no ranking of the results and only the number of results requested by the user are returned even though there may be more images that meet the specified colour percentages. This limitation can be solved by implementing the matching algorithm differently. Instead of matching images whose colours are within +/- 10% of the colours specified by the user, an alternative would be to actually measure the distance between the colour percentages specified by the user and the corresponding bins in the database image histogram. Any of the distance metrics used for colour histograms could be used but would only be applied to the histogram bins corresponding to the colours specified by the user. Using this method it would be possible to rank the results.

5.1.2. Texture

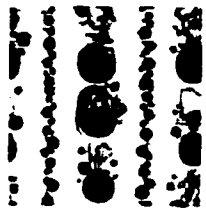
The texture features chosen for this project were based on the best results found in [6]. For the texture features, determination of whether or not a given image should be in the retrieval set for a query is very subjective. This is especially true for the contrast and coarseness features, the directionality feature is a little more intuitive. Because of this, the results discussed here are focussed more on directionality than coarseness and contrast.

When querying by picker, the user is provided a list of sample gray scale images representing a variety of different textures. User can select a sample texture to use and then select which features, i.e. directionality, contrast, and/or coarseness, of the sample texture to measure against. The sample textures are actually gray scale versions of selected images in the database of swatches. So one would expect that if the appropriate swatch type is being searched, the number

one ranked match should be the corresponding colour version of the sample texture image. Tests showed that this was true.

There was no significant difference found between the performance of the directionality features that used the Sobel operator versus those variations which used the Prewitt operator. There was, however, a difference in the performance between the vector and histogram variations of directionality. The vector variation is suitable for matching very simple directions, e.g. horizontal, vertical or diagonal but not a combination of these. The histogram variations of the directionality feature, however, performed extremely well on all images.

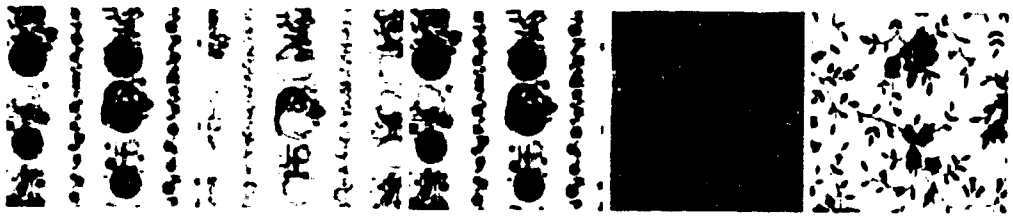
When querying by example using all three texture features (i.e. direction, contrast, and coarseness), one would expect to retrieve all images in the database that have exactly the same pattern as the example image but just different colours. This is not actually the case because contrast varies with the intensity (darkness versus lightness) of the colours and the coarseness measurement is also somewhat affected by intensity. Better results are obtained if you search by directionality and coarseness but not by contrast. The best results, however, are obtained when you search only by directionality. When you search only by directionality, all the images with exactly the same pattern are ranked at the top of the list of matches. An example of this can be seen in Figure 16 which shows the results from three sample queries: the first used all three texture features, the second used directionality and coarseness, and the third used only directionality. The example image appears at the top of the figure.



Example Image

Texture: directionality, coarseness and contrast

Texture: directionality and coarseness



Texture: directionality

Figure 16. Query by example - texture features

Similar results are obtained for all the variations of the directionality feature regardless of which distance metric is used (the queries in Figure 16 used the Prewitt histogram variation of the directionality feature). The only exception is that the Sobel vector variation seems to perform a little better than the Prewitt vector variation.

5.2. Usability

Although querying by combinations of features in parallel seemed like a good idea at first, after using the system for a while I have come to realize that it is not so useful after all, at least the way it is currently implemented.

For example, to search for all images with blue stripes this could be done using query by picker and setting the average colour to blue or specifying Some/Mostly colours as blue or some Percent colours as blue and selecting a sample texture with stripes from the texture chooser. The results obtained will not be what one expects. The results seem to suggest that only the colour part of the query was considered and the texture part was ignored. This is because the possible range of the distance for the colour part of the query is much greater than the possible range for the texture part of the query. I attempted to improve the situation by increasing the weight on the texture feature however this did not make much difference at all. So then I increased the maximum possible weight for the texture feature by an order of magnitude. The results improved but they were still not satisfactory.

This problem can be solved in three ways:

1. Provide the capability to serialize queries for different features, e.g. in the example above the user could base the query on the stripes first, then use the query results as the "database" of images for the following query in which the user would search for the colour blue.

2. Normalize the distances such that the range of possible distances is the same for all features regardless of the distance metric used.

3. Adjust the maximum weights for each feature such that when the feature distance is multiplied by the weight, it is possible to achieve the same distance range for all features.

I believe that both solutions 1 and 2 described above would be very beneficial while solution 3 is just an indirect method of achieving solution 2 and also put the onus on the user to keep adjusting the weight values when the query results are not as expected.

5.3. System Efficiency

This section compares the different image features based on time to analyze and time to measure similarity. All the results discussed in section were obtained while running the CBIR system on a portable PC with a 166MHz MMX processor and 64 Megs of RAM.

5.3.1. Feature Calculation Times

When adding a new swatch to the database, the total time required to calculate all the features associated with that new image is approximately 2 minutes. This is based on an average size image of about 120 x 120 pixels. The breakdown of the calculation times for the different features is approximated in Table 6.

Feature	Time to calculate (approx.)
Colour (all)	20 sec
Contrast	20 sec
Coarseness	45 sec
Directionality (all 4)	40 sec
Total	**2 minutes**

Table 6. Feature calculation times

As you can see in Table 6, colour features are very quick to calculate - approximately 20 seconds is required to calculate all of the colour features. Calculation of texture features is slower, with coarseness taking by far the longest: it takes about 40 seconds to calculate coarseness, 20 seconds to calculate contrast and 40 seconds to calculate all four directionality variations. It took approximately 22 hours to populate the database with all the images (636 of them).

The same amount of time is required whether adding a new image or loading an example, i.e. approximately 2 minutes, because in both cases all the features are calculated for that new image. It takes a little less time to add a texture sample because in that case only the texture features are calculated.

Note that no effort was made to be more efficient by reusing calculations from one feature to the next - the feature calculations are completely independent - which is easier to maintain. Considering that the bulk of the database population is done offline, improving the analyze time is not of high priority.

However, the calculation time for the coarseness feature was cut down from about 2 minutes to 45 seconds. This was done by setting the largest K for the neighborhood size to be 4 (i.e. a 16x16 neighborhood) instead of 5 (i.e. a 32x32 neighborhood). See Coarseness on page 14 for the details on how coarseness is calculated. No significant difference in performance was found with regards to retrieval accuracy.

5.3.2. Query Times

Most of the queries are quite quick. Most time is actually spent reading the images and displaying them. Obviously, the more matches the user requests and the more swatch types included in the search the longer it will take to display the images.

Queries using the 64 bin RGB colour histogram take the longest to execute. We can see here how these queries could benefit by filtering by average query or largest bins first before comparing histograms (see Fast Searching on page 24 for a discussion on filtering and indexing).

6. Conclusion

6.1. Results Summary

The results from the tests conducted as part of this project show that the most effective types of colour and texture features for an interior decorating application are the colour histogram and the directionality features. All the variations of the colour histograms and distance metrics performed very well except the L_2 distance metric when applied to the RGB colour histograms. This combination of feature and distance metric did not perform well for the lower ranked matches. The histogram version of the directionality feature variations performed much better than the vector variations, except in very simple cases where the images contained patterns in only one definite direction (horizontal, vertical, or diagonal) in which case the vector variations performed almost as well as the histogram variations.

The other features tested in this project, i.e. average colour, moments of colour distribution, contrast and coarseness, are not as useful for an interior decorating application because the results from queries based on these features are often not what is expected intuitively although the results are correct as far as the algorithms and calculations are concerned.

6.2. Future Work

Many improvements can be made to this project. Some of these improvements were not part of the original project because of a lack of time while others were only thought of while conducting

experiments, making observations and drawing conclusions. Following is a list of possible improvements:

- add an average colour filter and a largest bin filter for colour histogram matching
- implement percent colour queries using distance metrics instead of "meets criteria", i.e. instead of setting a percentage range for colour, measure the distance between the user provided values and values in the database image
- provide the capability to serialize queries of different feature types
- normalize the ranges of the distance measurements
- incorporate this CBIR application into a commercial interior decorating software package

7. References

7.1. Papers

1. M. Flickner *et al.*, **Query by Image and Video Content: The QBIC System**, *Computer - Innovative Technology for Computer Professionals*, September 1995, pp. 23-32.

2. W. Niblack *et al.*, **The QBIC Project: Querying Images by Content Using Color, Texture, and Shape**, *Research Report RJ 9203 (81511)*, IBM Research Division, New York, February 1993.

3. J. Hafner, H.S. Sawhney, W. Equitz, M. Flickner, W. Niblack, **Efficient Color Histogram Indexing for Quadratic Form Distance Functions**, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 17, No. 7, July 1995, pp. 729-736.

4. C. Faloutsos *et al.*, **Efficient and Effective Querying by Image Content**, *Research Report RJ 9453 (83074)*, IBM Almaden Research Center, August 1993.

5. S. Landis, **Content-Based Image Retrieval Systems for Interior Design**, *CS718 Project*, Fall 1995.

6. H. Tamura, S. Mori, and T. Yamawaki, **Textural Features Corresponding to Visual Perception**, *IEEE Trans. On Systems, Man, and Cybernetics*, Vol. 8, No. 6, June 1978, pp. 460-473.

7. **The Java Tutorial**, available at http://java.sun.com/docs/books/tutorial

8. M. Stricker and M. Orengo, **Similarity of Color Images**, *SPIE Vol. 2420*, pp. 381-392.

9. M.J. Swain and D.H. Ballard, **Color Indexing**, *International Journal of Computer Vision*, Vol. 7, No. 1, 1991, pp. 11-32.

10. J.K. Wu *et al.*, **CORE: a content-based retrieval engine for multimedia information systems**, *Multimedia Systems*, Vol. 3, Feb. 1995, pp. 25-41.

11. R. Aalmoes, P.Bosch, **Overview of Still-picture and Video Compression Standards**, *Pegasus paper 95-3*, University of Cambridge Computer Laboratory, Dec. 21, 1995.

12. J. Pilkington, **Content-based Image Retrieval**, *Final Report for Advances in Human-Computer Communications course*, Concordia University, April 1996.

13. W. Y. Ma and B. S. Manjunath, **Texture Features and Learning Similarity**, *1996 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 425-430.

14. B. S. Manjunath and W.Y. Ma, **Texture Features for Browsing and Retrieval of Image Data**, *CIPR TR-95-06*, University of California at Santa Barbara, July 1995.

15. V. E. Ogle and M. Stonebraker, **Chabot: Retrieval from a Relational Database of Images**, Computer - Innovative Technology for Computer Professionals, September 1995, pp. 40-48.

16. M. Borchani, **Caraterisation d'images par les attributs visuels: texture, couleur et forme**, Rapport d'activite SIP, University of Paris, February 1996, pp. 17-20.

17. R. W. Picard and F. Liu, **A New Wold Ordering for Image Similarity**, *1994 International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Adelaide, South Australia, pp. V129-V132.

18. P. M. Kelly, M. Cannon and D. R. Hush, **Query by image example: the CANDID approach**, Los Alamos National Laboratory, Los Alamos, New Mexico and University of New Mexico, Albuquerque, New Mexico.

19. E. Binaghi, I. Gagliardi, and R. Schettini, **Indexing and Fuzzy Logic-Based Retrieval of Color Images**, *IFIP Transactions*, A-7, 1992, pp. 79-94.

20. R. W. Picard and T. P. Minka, **Vision texture for annotation**, *Multimedia Systems*, Vol. 3, February 1995, pp. 3-14.

21. R. W. Picard and T. Kabir, **Finding Similar Patterns in Large Image Databases**, *1993 International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Minneapolis, Minnesota, pp. V161-164.

22. A. Pentland, R.W. Picard, and S. Sclaroff, **Photobook: Content-Based Manipulation of Image Databases**, International Journal of Computer Vision, Vol. 18, No. 3, 1996, pp. 233-254.

23. J. Ashley *et al.*, **Automatic and Semi-Automatic Methods for Image Annotation and Retrievals in QBIC**, *Research Report RJ 9951 (87910)*, IBM Research Division, Almaden Research Center, April 1995.

24. D. Lee *et al.*, **Query by image content using multiple objects and multiple features: user interface issues**, *1994 Proceedings of the IEEE International Conference on Image Processing (ICIP)*, Austin, Texas, pp. 76-80.

25. **PSE/PSE Pro for Java API User Guide** for Release 1.2 of PSE and PSE Pro for Java.

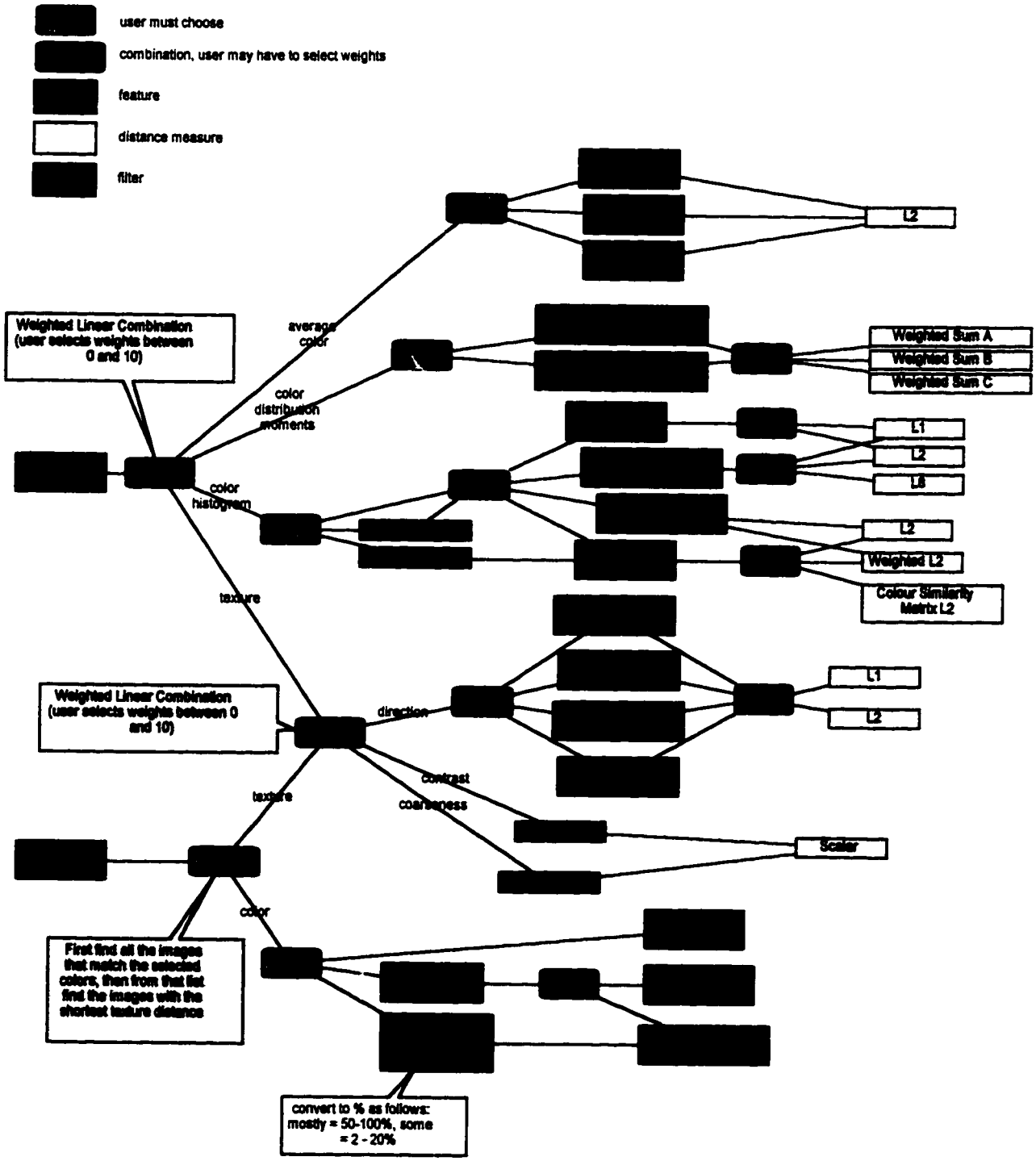
7.2. Sources of Images

Following is a list of sources for the images contained in the database of this project:

- www.imp-wall.com/patterns/index.html
  - jpeg format
  - thumbnails & fullsize of each
  - 24 wallpapers: country1-6.jpg, faux1-6.jpg, floral1-6.jpg, stripes1-6.jpg
  - 40 borders: country1-6.jpg, architectural1-4.jpg, contemporary1-6.jpg, folkart1-6.jpg, kids1-6.jpg, novelty1-6.jpg, traditional1-6.jpg
- Print Artist 4.0 in the Sierra Home 97 Collection by Sierra
  - all the textile images that appear in the Print Artist 4.0 manual on pages 129-133
  - images were found on the Print Artist CD in /pa4/jpg
  - jpeg format
  - fullsize (I shrunk them down to 15% to create thumbnail versions)
  - 200 fabrics (textiles)
- http://www.visualhome.com/v11/Collections/Generic/rugs10-1.htm
  - from Visual Home Digital Collections download center
  - carpets: from the Rugs Bonus Pack, I got 21 images that I had to crop and covert from bitmap to jpeg
- Custom Home 3D Version 4.0 in the Sierra Home 97 Collection by Sierra
  - I got 80 images off the CD to use for floorings, on the CD the images are found in /Material in the folders: Abstract, Finishes, Geometric, Stone, Textured, Stone, Tile and Woods
  - the images were very small bitmap images - I increased the size by 50% and converted them to jpeg format

- http://www.decdim.com/ (info@decdim.com)
  - borders: 44 Looney Tunes, 41 outdoors, 40 teddybear
  - wallpapers: 63 Looney Tunes, 37 outdoors, 46 teddybear
  - fullsize version of all the images above are available but I did not bother downloading them

8. Appendix A - Query Choices



9. Appendix B - Implementation Notes

This appendix contains some important details regarding the implementation and installation of the CBIR system for this project as well as a brief list of the current limitations and some suggested enhancements.

9.1. Installation

This section provides instructions on how to install the CBIR system for this project to run on your computer.

Step 1: The CBIR system requires a Java Virtual Machine to run. You can obtain everything you need from the Java Runtime Environment (JRE) version 1.1.6 which is freely available at http://java.sun.com/products/jdk/1.1/jre/. Download the appropriate version for your computer and follow the installation instructions that comes with the software. **Important:** the CBIR system will not run without this software.

Step 2: Installing from floppy diskettes: Copy the contents of all 5 diskettes into a folder (directory) on your hard drive where you would like to install the system. You should end up with 5 zip files (classes.zip, db.zip, swatch1.zip, swatch2.zip and swing.zip) in that folder.

Installing from a compact disk (CD): Copy all the contents of the CD into a folder (directory) on your hard drive where you would like to install the system. **Then, skip to step 5.**

Step 3: Extract the contents of each zip file into the same folder where you copied them to. **Important:** when extracting the zip files, make sure the option to use the folder names is selected (otherwise the directory structure will not be created properly).

Step 4: Delete the zip files.

Step 5: On a Windows platform, include the following lines in your autoexec.bat file to set two important environment variables. On a platform other than Windows, set the environment variables as per standard procedures for that platform. Modify CBIR_HOME and JRE_HOME to be the path to the CBIR folder where you installed the CBIR system and the path to the location where you installed the Java Runtime Environment respectively:

```
set CBIR_HOME=C:\CBIR

set JRE_HOME=C:\JRE\1.1
```

Step 6: Restart your system (this step is required to initialize the values of the new environment variables defined in step 5).

Installation is now complete and you are ready to run the system (see the next section on running the system).

9.2. Running the System

CBIR application

Once you have completed the installation you are now ready to run the application. To run the application in Windows, change folders to where the CBIR system was installed and double click on **cbir.bat**. To run the application on an operating system other than Windows, copy the contents of the **cbir.bat** file to the command line.

CBIR database loader

Instructions are provided here on how to run the database loader but it should not be run because the database has already been populated.

Step 1: Prepare a folder which will contain the source images to be loaded into the database. That folder should contain folders named as follows: Wallpapers, Borders, Carpets, Floorings, Fabrics, and Textures. Place the images to be added in one of these folders, e.g. if the image represents a wallpaper sample then put the image in the Wallpapers folder.

Step 2: In the folder where the CBIR system was installed, edit the file called loaddb.bat. The second argument after CBIRdbLoader should be modified to be the path to the image folder prepared in step 1.

Step 3: To run the database loader in Windows, change folders to where the CBIR system was installed and double click on **loaddb.bat**. To run the application on an operating system other than Windows, copy the contents of the **loaddb.bat** file to the command line.

9.3. Third Party Software

Following is a list of the third party software used to implement, run, or is incorporated in the CBIR system for this project.

- Java Development Kit (JDK) - version 1.1.6
- Java Foundation Classes (JFC, aka Swing) - version 1.0.2
- Java Runtime Environment (JRE) - version 1.1.6
- ObjectStore PSE (Persistent Storage Engine) for Java - release 1.2
- Collections package by Doug Lea - version 0.96b
- Numerical Library for Java (JNL) by Visual Numerics

9.4. Current Limitations

Following is a list of some limitations in the current implementation of the CBIR system for this project:

- When an image is added to the database, it is moved from its source location to the database location instead of being copied.
- A file contention problem occurs sporadically when adding new images - it seems like sometimes it is slow to return permission to write the file after reading it. The current workaround is a busy loop that keeps trying to move the file until it is successful.

9.5. Application Enhancements

Following is a list of suggested enhancements to improve the existing CBIR application in this project:

- Add the colour similarity matrix distance metric (see description on page 22).
- Add the capability to request a random set of images from the database, from which the user could choose an image to start a search.
- Add the capability to remove an image from the database.
- Add the capability to save and load a set of choices, i.e. the contents of the Choices window). Choices could be saved as a "room" and the user would provide a name such as "master bedroom". A previously saved set of Choices could be loaded on another day to continue the search.
- Provide a full size image viewer. When images are added to the database, both the full size and thumbnail versions would have to be provided. This applies to images added on line or in batch mode.
- Include a more comprehensive on line help system.