

An Analytical Study on Image Databases

By

Francine Ming Fang

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of


Master of Engineering in Electrical Engineering and Computer Science


at the Massachusetts Institute of Technology

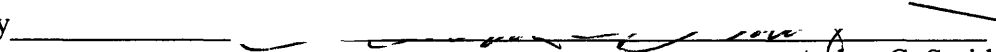
June 5, 1997

Copyright 1997 Francine Ming Fang. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and distribute publicly paper and electronic copies of this thesis and to grant others the right to do so.

Author 
Department of Electrical Engineering and Computer Science
June 5, 1997

Certified by 
Dr. Nishikant Sonwalkar
Thesis Supervisor

Accepted by 
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

RECEIVED
OCT 29 1997

An analytical Study on Image Databases

By
Francine M. Fang

Submitted to the
Department of Electrical Engineering and Computer Science

June 5, 1997

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

Complex information data types such as images are becoming more and more significant in digital communication as modern technology continues to evolve. Because of the increased significance of complex image data types, conventional databases which dealt only with textual and numerical information are no longer adequate. With the help of recent advent of more powerful computers and faster networking capabilities, significant amount of work has began in both the academia and commercial world in searching for better techniques to store and retrieve images in large collections. The goal of this thesis is to understand the current status of these research efforts and learn the technologies in the area of image recognition. Furthermore, it also provides a step-by-step guide for designing image databases and presents a case study on an existing commercial image database. The thesis is intended to serve as an informational resource for researchers and professionals that are interested in this growing field.

Thesis Supervisor: Nishikant Sonwalkar
Title: Director, Hypermedia Teaching Facility

CHAPTER 1 INTRODUCTION.....	5
1.1 TRADITIONAL DBMS'S INADEQUACIES	5
1.2 GENERAL ARCHITECTURE OF AN IMAGE DBMS.....	6
1.3 QUERIES IN IDBMS.....	7
CHAPTER 2 PATTERN RECOGNITION AND IMAGE ANALYSIS TECHNIQUES.....	9
2.1 SEGMENTATION.....	9
2.1.1 <i>Edge Detection</i>	9
2.1.2 Clustering.....	13
2.2 SHAPES OF REGIONS.....	17
2.3 TEXTURE.....	18
2.4 MATCHING.....	22
2.5 NEURAL NETWORKS.....	24
CHAPTER 3 IMAGE DATABASE RESEARCH PROJECTS.....	28
3.1 BERKELEY'S DOCUMENT IMAGE DATABASE.....	28
3.1.1 <i>Multivalent Documents</i>	28
3.1.2 <i>System Limitations</i>	30
3.1.3 <i>Proposed Solutions to the Limitations</i>	31
3.2 MANCHESTER CONTENT-ADDRESSABLE IMAGE DATABASE (CAID).....	35
3.2.1 <i>Architecture</i>	35
3.2.2 <i>The Data</i>	36
3.2.3 <i>MVQL</i>	37
3.3 THE ALEXANDRIA DIGITAL LIBRARY PROJECT.....	41
3.3.1 <i>Image Analysis</i>	42
3.3.2 <i>Similarity Measures and Learning</i>	43
3.3.3 <i>A Texture Thesaurus For Aerial Photographs</i>	44
3.4 PHOTOBOK.....	45
3.4.1 <i>Semantic Indexing of Image Content</i>	45
3.4.2 <i>Semantic-preserving image compression</i>	46
3.4.3 <i>Face Photobook</i>	47
3.5 THE CHABOT PROJECT.....	49
3.5.1 <i>Integration of Data Types</i>	50
3.5.2 <i>POSTGRES</i>	50
3.5.3 <i>Image Retrieval Methods</i>	51
3.5.4 <i>System Analysis</i>	54
CHAPTER 4 COMMERCIAL IMAGE RETRIEVAL SYSTEMS.....	56
4.1 IBM'S QUERY BY IMAGE CONTENT.....	56
4.1.1 <i>Image Feature Extraction</i>	56
4.1.2 <i>Automatic Query Methods for Query-By-Painting</i>	57
4.1.3 <i>Object Identification</i>	60
4.2 THE VIRAGE IMAGE SEARCH ENGINE.....	62
4.2.1 <i>Virage Data types</i>	62
4.2.2 <i>Primitive</i>	63
4.2.3 <i>Schema definition</i>	64
4.3 METADATA.....	68
4.3.1 <i>What is Metadata and Why use it</i>	68
4.3.2 <i>Issues</i>	69
4.3.3 <i>Current Status</i>	69
CHAPTER 5 DESIGNING AN IMAGE DATABASE.....	71
5.1 STORING DIGITAL IMAGES.....	72
5.2 FEATURE EXTRACTION.....	74

5.3	QUERY AND RETRIEVAL	75
CHAPTER 6 CASE STUDY: QBIC.....		79
6.1	TESTING METHODOLOGY	79
6.2	TEST RESULTS	81
6.3	EVALUATION CONCLUSIONS	83
CHAPTER 7 CONCLUSION.....		85

Chapter 1 Introduction

In the last several years, there has been a significant increase in the amount of digital data being produced such as digital images, video, and sound. This expansion of digital mediums created a need for developing management systems that can handle these types of data. Since traditional databases rely solely on the alphanumeric information for search and retrieval, which is adequate for books, papers or any other type of textual publications, they are inadequate for these new data types. These traditional database systems lack the capability and the flexibility to satisfactorily index and search digital images, videos or sounds by content since it is nearly impossible to describe a painting, movie or song completely by using alphanumeric descriptions alone. Clearly, the creation of a new paradigm for digital information management system is necessary.

In general, a database management system provides several fundamental capabilities for sharing information such as querying, recovery, security, fast access, and data integrity. These capabilities can be classified into two categories. One concerns with the issues of the physical storage, compression and error correction of the images. The other deals with the management of the images in storage, such as the issue of indexing, searching and retrieval. However, the focus of this thesis is mainly on the indexing, searching and retrieval part of *image databases* although some issues in data storage are mentioned as well.

1.1 Traditional DBMS's Inadequacies

Most conventional databases support a number of basic data types, such as integer, floating point numbers and character strings²⁵. To support emerging imaging applications, a number of database management systems (DBMS) also support arbitrary long binary strings called Binary Large Objects or **BLOBs**. Although most conventional databases support long fields or BLOBs, most conventional database applications still include only the path or references of images in relational records and stored the actual images on some optical storage system. There are a number of reasons for this strategy. One basic reason is that this strategy meets the requirements for on-line, near-line, and off-line storage of images, including archiving. This means, the data in the databases are of textual or numerical nature only which is clearly insufficient for image databases.

Another function often missing from conventional databases is **content retrieval**. Typically, querying and retrieval in conventional DBMS are based on their text descriptive attributes. That is adequate for document databases since searching often involves the *content* of document texts. With full-text retrieval systems, queries can involve any combination of terms and words contained in text documents. But, this content-retrieval concept cannot be generalized to other data types other than text. For content retrieval on images, more than descriptive phrases are necessary. Ideally, image content retrieval can be done by extracting image features, such as color, shape, texture, and then use these features to guide the queries.

Similarly, in conventional databases, sometime the DBMS allows thumbnail viewing of images. **Thumbnails** are miniature versions of the full size images, so one screen can display many thumbnails at once. These thumbnails help users organize and manage images visually. Users can quickly scan the displayed images to find the desirable one. However, this is no more than like a screen dump of all images stored on the databases. Users may have to view numerous thumbnails before finding the ones they want. The problem here is that the system does not discriminate enough so it returns very imprecise retrieval results to the users.

1.2 General Architecture of An Image DBMS

There are very few existing products or systems that are exclusively image database management systems. Instead, most image databases are constructed within existing conventional database management systems that already include some features and capabilities that can handle images. Therefore, image database developers typically build an image database on top of a conventional DBMS.

The integration of images into a conventional DBMS can include hierarchical storage modules, content-retrieval engines, graphical editors, and other modules. To meet performance requirements, an image DBMS (IDBMS) may include additional components such as search engines, spatial data structures, query presentation, and other “extension” modules. Thus the “system” or engine used by an image database typically consists of several objects or modules from different sources. For example, a core DBMS engine, a storage management module, a content-based search engine and a front-end modules such as the user interface. Figure 1 depicts this general architecture.

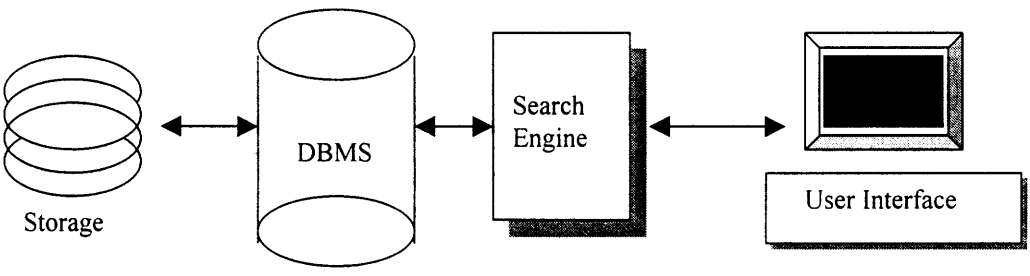


Figure 1: typical architecture of an IDBMS

1.3 Queries in IDBMS

In conventional “text only” DBMS, the search criteria are on relatively *precise* attributes such as name, identification number, etc. The user submits a query based on these precise attributes and then obtains the results in a table. The situation is not so clear-cut for image data type queries. Precise attributes such as title and painter are not sufficient for content-based image retrievals. Most of the image attributes need to be extracted from the image itself such as color, texture and shape. Content-based searches can take many forms. For instance, images can be presented as a collection of shapes and then the properties of each shape or object in an image can be parameters in content-retrieval searches. Similarly, the spatial relationship among various objects can be another possible criterion for content retrieval of images. The type of attributes used for retrieval will depend on the intended application. The following are some content retrieval techniques used for image queries²⁵.

- **Visual queries by example:**

The system provides a visual example in an image database so users can find other images similar to it. For instance, the user might pick a fabric and post the query: *Show me all fabrics of similar texture in different colors*. The query expression thus will involve a GUI environment where the user can pick samples and even compose a query by putting together different objects. An example would be: *Give me all still images that have a lamp on the right-hand corner of a table just as I have shown here in the sample drawing*. In many image database applications, the sample images in a query are actually obtained from a pick list of a “domain,” that is, a set of all possible, typical values for a particular feature or object type. The pick list can be a pick list of colors, textures, shapes, or actual objects, such as facial features in facial-retrieval systems.

- **Weights:**

In a conventional database query expression, all predicate terms have equal weights. For instances, if the predicate expression is *Subject = Correspondence* and *DateCreated = 10/10/93* then the predicate on *Subject* and *DateCreated* must both be satisfied. However, various terms of the predicates can be given different weights by the user. If the query is, say, on “image database”, the user can indicate that “image” is more important than “database” in the retrieval expression. Similarly, this can be applied to image content retrieval where users can indicate, for instance, that the shapes of the objects are more important than the color.

- **Similarity Measure:**

A search expression in image databases can involve terms like *SIMILAR TO* where the argument could be a complex image. The image database should

support and contain predefined terms to undergo this kind of complex similarity evaluation. Algorithms must be developed to determine how similar one image is to another. Furthermore, the query results must be returned within an acceptable amount of response time, and the quality of the images returned in the result must be good. The system must be extendible for development for future functionality.

In order to perform content retrieval for images, relevant features in the images must be extracted. Extracting descriptive attribute values or recognizing objects in image databases can be done manually, automatically, or with a hybrid scheme that extracts some of the features automatically while domain experts manually enter additional key features and attributes of the images. Manual indexing is a difficult and arduous task, so it is not ideal for image DBMS. However, a fully automated feature extraction requires the system developers not only to be experts in image feature extractions but also prophets that can foretell what features are ideal for all applications. Therefore, the hybrid scheme is generally preferred. Automatic and manual indexing techniques are often used at different layers of an image database for image recognition and feature-identification components. However, choosing what features to extract and how they can be extracted efficiently is still a challenge for image database developers whether one automate the feature extraction process or not.

Several research and prototype developments of image database applications are currently underway. But before one can fully understand the imaging methodologies that are applied in these image databases, one needs some basic knowledge on pattern recognition and image analysis techniques. Chapter 2 is devoted to give an overview of the field of pattern recognition and image analysis. In chapter 3, several different image database research projects in academia are then presented followed by chapter 4 that focuses on commercial developments in image databases. Chapter 3 and 4 aim to provide an exposition of imaging database technology, especially on methods for image content retrievals. A guide to designing image databases is then given in chapter 5 while chapter 6 presents the results of an evaluation on a commercial image DBMS.

Chapter 2 Pattern Recognition and Image Analysis Techniques

Prior to being used as inputs to image retrieval programs, images are often modified by going through operations such as segmentation, texture differentiation, and shape recognition so they can become better inputs. These operations extract the images' distinct features to make the image retrieval process easier, faster and more accurate. In this chapter, a brief introduction into several pattern recognition schemes and image analysis technique is given. The areas being explored are segmentation, edge detection, clustering, shape determination, texture differentiation, matching, and neural networks.

2.1 Segmentation

Often, the first step in image analysis is to divide the images into regions that correspond to various objects, each of which is reasonably uniform in some characteristics in the scene. Afterwards, various features such as size, shape, color, and texture can be measured for these regions and these features then can be used to classify the image. A **region** can be loosely defined as a collection of adjacent pixels that are similar in some way, such as brightness, color, or local visual texture. Objects are defined here as non-background regions in the image¹⁶.

One of the simplest ways to segment an image is to first set thresholds based on pixels' color or brightness that can divide the image into several regions and then consider each resulting connected region to be a separate object. For example, if thresholds were set based on pixel's gray level for black-and-white images, the scene can then be divided into light and dark regions. Therefore, one could set n thresholds, which divide the image into $n + 1$ categories of gray level, to produce a new image with gray levels of $0, 1, 2, \dots, n$.

Threshold setting divides the pixels into regions on the basis of their pixels' individual characters, such as gray level, alone. The segmented outcome may be inadequate and unsatisfactory. Another way to segment an image into regions which is not based on single pixel's attributes alone is to look for the discontinuities or edges between the regions and connect them so as to surround, and therefore define, the regions. This is done through a technique called *edge detection*. Another segmentation technique is called *clustering*. It is to look for clusters of similar pixels and allow these groups to expand and merge to fill the regions.

2.1.1 Edge Detection

Edges in pictures are important visual clues for interpreting images. If an image consists of the objects that are displayed on a contrasting background, **edges** are defined to be the transition between background and objects¹⁶. The total change in intensity from background to foreground is called the **strength of the edge**. This change can be used to

detect edges. In the simplest case, first order calculus is used in getting this rate of change. For black-and-white images, the rate of change in gray level with respect to horizontal distance in a continuous image is equal to the partial derivative $g'_x = dg(x, y)/dx$. This partial derivative can be made finite to become

$$g'_x(x, y) = g(x + 1, y) - g(x, y) \quad \text{Equation 1}$$

Similarly for the rate of change in gray level with respect to the vertical distance becomes

$$g'_y(x, y) = g(x, y + 1) - g(x, y) \quad \text{Equation 2}$$

g'_x and g'_y are called the **first differences** which represent the change in gray level from one pixel to the next and can be used to emphasize or detect abrupt changes in gray level in the image. Since the edges of objects in a scene often produce such changes, these operators are called **edge detectors**.

The above edge detectors are quite primitive since it can only detect in horizontal or vertical directions. More sophisticated first difference detectors can operate on a ring of directions as well as on pixels of mixed colors. More sophisticated edge detection is done based on calculating the diagonal differences after performing some rotations. But performing all possible rotations is arduous. For continuous images, the **gradient** is a vector quantity that incorporates both magnitude and direction. The gradient of the continuous black-and-white image $g(x, y)$ is defined to be the ordered pair of partial derivatives

$$\text{grad } g(x, y) = \nabla g(x, y) = \left(\frac{\partial}{\partial x} g(x, y), \frac{\partial}{\partial y} g(x, y) \right) \quad \text{Equation 3}$$

The symbol ∇ represents the gradient. The gradient of the continuous image indicates the rate of change in gray level in the images in both the x and y directions. The analogous gradient definition for finite difference operations is

$$\nabla g(x, y) = (g'_x(x, y), g'_y(x, y)). \quad \text{Equation 4}$$

The magnitude M of the gradient is defined by

$$M = \sqrt{g'_x(x, y)^2 + g'_y(x, y)^2}, \quad \text{Equation 5}$$

and the direction D is defined by

$$D = \tan^{-1}(g'_y(x, y)/g'_x(x, y)) \quad \text{Equation 6}$$

D is an angle measured counterclockwise from the x -axis. If g'_y is negative, D lies between 180 and 360 degrees, otherwise it lies between 0 to 180 degrees. The direction D is the direction of the greatest change in $g(x, y)$, and the magnitude M is the rate at which $g(x, y)$ increases in that direction.

The edge detectors mentioned earlier are based on first difference methods, so they are most sensitive to rapid changes in the gray or color level from one section to another. But sometime the transition from background to object is quite gradual. In these cases, the edges does not produce a large change in gray or color levels between any two touching sections, but it does produce a rapid change in the rate of change of gray or color level between the regions. To look for a large change of value in the gray level changes in black-and-white images, the **second derivative** $\partial^2 g(x, y)/\partial x^2$ is used, which is approximated by taking the first derivative of the first difference image

$$\begin{aligned} g''_x(x, y) &= [g(x+1, y) - g(x, y)] - [g(x, y) - g(x-1, y)] \\ &= g(x+1, y) - 2g(x, y) + g(x-1, y) \end{aligned} \quad \text{Equation 7}$$

Second derivatives are useful only if the gradual gray level changes resemble something like a ramp, a straight line with a constant slope, between the two regions. In real images, it is more likely that the gray level changes will resemble a smooth curve rather than a ramp near the edges of objects. In this case, for two-dimensional images, the edges are often considered to lie at location where the second difference is 0, i.e. where $\partial^2 g(x)/\partial x^2 = 0$ or $\partial^2 g(x)/\partial y^2 = 0$. This is called the **point of inflection**.

2.1.1.1 Laplacian

Just as for the first difference technique, the Laplacian is the counterpart of the gradient for second difference methods. A **Laplacian**¹³ of an image $g(x, y)$ is defined as the sum of its second derivatives,

$$\nabla^2 g(x, y) = \left[\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right] g(x, y) = \frac{\partial^2}{\partial x^2} g(x, y) + \frac{\partial^2}{\partial y^2} g(x, y)$$

Equation 8

The Laplacian operator is well suited for detecting small spots in pictures. Although some edges show up clearly, the many isolated spots show that random noise in the image is amplified also.

2.1.1.2 Hough transforms

Although second difference methods are better at detecting fading edges, it is not ideal for finding edges in incomplete or noisy pictures. Generalized **Hough transforms**¹⁷ are a set of techniques that can detect incomplete simple objects in images. Hough transforms detect edges by finding clusters in a parameter space that represents the range of possible variability of the objects to be detected. These transforms are usually used to infer the presence of specific types of lines or curves in noisy images, especially in cases where only fragments of the true line or curve may actually appear in the image. It also applies to faint edges.

The basic idea is to record parameters for each of the lines defined by any pair of points. These parameters should define the line uniquely, regardless of which two particular points on that line originally defined it. For instance, the slope and the y -intercept are frequently used to define straight lines. Thus, as an example, for n points, the edges can be detected by letting the $n(n-1)/2$ pairs of points define a line and check how many other points fall on or near each of these lines.

To extend to other family of lines, the steps are the same: first, select features that define the lines; second, make a scatter plot of the results from the feature parameters. The parameter space that is being scatter plotted is called a **Hough space**. Each pair of points in the image produces one point in the scatter plot. The lines that are actually present in the image produce clusters, dense regions, or peaks in the scatter plot, and are detected by locating these peaks using clustering techniques mentioned in the next section. To illustrate, the detection of existing circles in the image can be done using the (x,y,r) triplet, where (x,y) are the origin of the circle and r is the radius, and then determine which value of r that has the most points fallen on it. Figure 2 and figure 3 show this process.

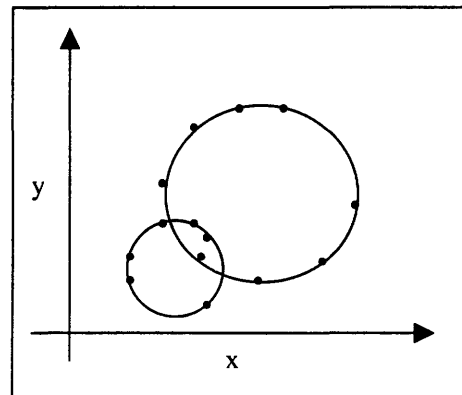
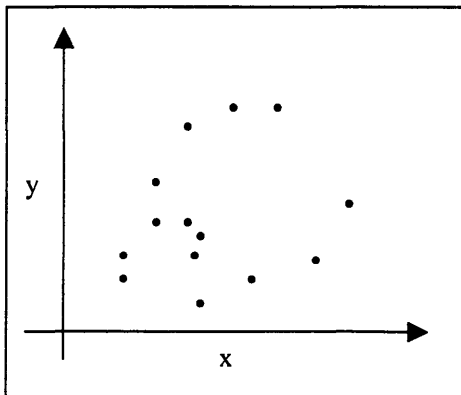


Figure 2. Sampled points in a Hough space (x,y) . Figure 3. Two circle detected for the points.

The methods used to calculate the nearness of other points to a line are least squares and Eigenvector line fitting. **Least squares** technique minimizes the sum of *vertical* squared error of fit, while the **Eigenvector** method minimizes the sum of squared distances between the points and the line, measure from each point in a direction *perpendicular* to the line.

2.1.2 Clustering

In the previous section they talked about edge detectors, now they are going to introduce another technique often used in image segmentation, clustering. Given a set of feature vectors sampled from some population, they would like to know if the data set consists of a number of relatively distinct subsets. **Clustering**⁵ refers to the process of grouping samples so that the samples are similar within each group. These groups are called clusters. The main goal in clustering is to discover the subgroups. In image analysis, clustering is used to find groups of pixels with similar gray levels, colors, or local textures, in order to discover the various regions in the image.

When feature vectors are measured from the samples, finding the clusters is much more than looking for high concentration of data points in a two-dimensional graph. Distinct clusters may exist in a high-dimensional feature space and still not be apparent in any of the projections of the data onto a plane, since the plane is defined only by two of the feature axes. One general way to find candidates for the centers of clusters is to form an n -dimensional histogram of the data, one dimension for each feature, and then find the peaks in the histogram. However, if the number of features is large, the histogram may have to be very coarse to have a significant number of samples in any cell, and the location of the boundaries between these cells are specified arbitrarily in advance, rather than depending on the data.

2.1.2.1 Hierarchical Clustering

Hierarchical clustering⁶ refers to a clustering process that organizes the data into large groups, which contain smaller groups, and so on. A hierarchical clustering helps to manage large feature sets and may be grown as a tree or **dendrogram**. The finest grouping is at the bottom of the dendrogram, the coarsest grouping is at the top of the dendrogram. In between, there are various numbers of clusters. Figure 4 shows an example of a dendrogram.

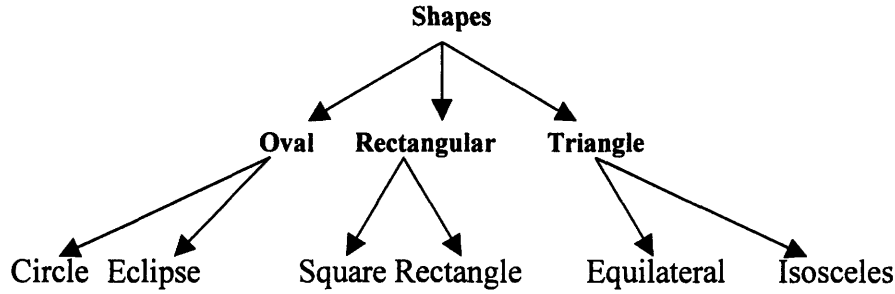


Figure 4. An example of a dendrogram.

Agglomerative Clustering⁷ is a type of hierarchical clustering that builds the dendrogram from the bottom up. In this section, the similarity between clusters are measured by the distance between the clusters in their feature space. In other words, each object, such as pixels, is represented as a point in some multi-dimensional feature space. The location of each point is determined by the object's feature attributes. The similarity of the objects is represented as the distance between pairs of samples.

There are four basic algorithms of Agglomerative clustering: single-linkage, complete linkage, average linkage and the Ward's method. The **Single-Linkage Algorithm**⁷ is also known as the minimum method or the nearest neighbor method. Similarity is obtained by defining the distance between two clusters to be the smallest distance between two points such that one point is in each cluster. Formally, if C_i and C_j are clusters, the similarity between them S_{SL} is defined as

$$S_{SL}(C_i, C_j) = \min_{a \in C_i, b \in C_j} dist(a, b), \quad \text{Equation 9}$$

where $dist(a, b)$ denotes the distance between the samples a and b .

The **Complete-Linkage Algorithm**¹¹ is also called the maximum method or the farthest neighbor method. The similarity between two clusters is determined by measuring the largest distance between a sample in one cluster and a sample in the other cluster. The most similar clusters are the two clusters that produced the smallest of the maximum differences of all possible pairs of clusters. Formally, if C_i and C_j are clusters, the similarity between them S_{SL} is defined as

$$S_{SL}(C_i, C_j) = \max_{a \in C_i, b \in C_j} dist(a, b), \quad \text{Equation 10}$$

where $dist(a, b)$ denotes the distance between the samples a and b .

Both single-linkage and complete linkage clustering algorithms are susceptible to distortion by deviant observations. The **Average-Linkage Algorithm**¹¹ is the compromise of the two. It is also known as the unweighted pair-group method using

arithmetic averages (UPGMA), and it is one of the most widely used. In this algorithm, similarity is obtained by defining the distance between two clusters to be the average distance between a point in one cluster and a point in the other cluster,

$$S_{SL}(C_i, C_j) = \frac{1}{n_i n_j} \sum_{a \in C_i, b \in C_j} dist(a, b), \quad \text{Equation 11}$$

where n_i and n_j are the number of elements in cluster C_i and C_j respectively.

The **Ward's Method**¹¹ is also called the minimum-variance method. It finds two clusters that are most similar to each other by choosing the pair that produces the smallest squared error for the resulting set of clusters. The square error for each cluster is defined as follows. If a cluster contains m samples x_1, \dots, x_m and each sample is measured for d features, then x_i is the feature vector (f_{i1}, \dots, f_{id}) . The square error for the sample i is

$$\sum_{j=1}^d (x_{ij} - u_j)^2 \quad \text{Equation 12}$$

where u_j is the mean value of feature j for the samples in a cluster.

$$u_j = \frac{1}{m} \sum_{i=1}^m x_{ij}. \quad \text{Equation 13}$$

The square error E for the entire cluster is the sum of the squared errors of the samples.

$$E = \sum_{i=1}^m \sum_{j=1}^d (x_{ij} - u_j)^2 = m\sigma^2 \quad \text{Equation 14}$$

The vector composed of the means of each feature, $u = (u_1, \dots, u_d)$, is called the mean vector or **centroid** of the cluster. The square error for a set of clusters is defined to be the sum of the square errors for the individual clusters.

2.1.2.2 Partitional Clustering

The Agglomerative clustering creates a series of nested clusters. This is not ideal when what is desired was to create one set of clusters that partitions the data into similar groups. That is called **Partitional Clustering**⁸. Samples close to one another are assumed to be similar and the goal is to group data that are close together. The number of clusters to be constructed can be specified in advance.

Partitional clustering produces divisive dendrograms, or dendrograms formed from top-down, where one initial data set is divided into at least two parts through each iteration of the partitional clustering. Partitional clustering is more efficient than agglomerative clustering since it can divide into more than two at a time, only the top data set is need to start, and doesn't have to complete the dendrogram.

Forgy's Algorithm¹⁴ is one of the simplest partitional clustering algorithms. The inputs are the data, a number k indicating the number of clusters to be formed, and k sample points called **seeds** which are chosen at random or with some knowledge of the scene structure to guide the cluster selection. The seeds form the cluster centroids. There are three steps to the Forgy's algorithm.

1. Initialize the cluster centroids to the seed points.
2. For each new sample, find the closest centroid to it and cluster it together, then recompute the centroids when all samples are assigned a cluster.
3. Stop when no samples change clusters in step2.

Instead of choosing seed points arbitrarily, the Forgy's algorithm can begin with k clusters generated by one of the hierarchical clustering algorithms and use their centroids as initial seed points. One of the limitations of the Forgy's algorithm is that it takes a considerable amount of time to produce stable clusters. However, restrict number of iterations, state a minimum cluster size, and supply new parameters may help in reaching the outcome faster.

The **K-means algorithm**⁹ is similar to Forgy's algorithm. The inputs are the same as Forgy's algorithm. However, it recomputes the centroid as soon as a sample joins a cluster, and it only makes two passes through the data set.

1. Begin with k clusters, each containing one of the first k samples. For each remaining $n - k$ samples, where n is the number of samples in the data set, find the closest centroid to it, and cluster them. After each sample is assigned, recompute the centroid of the altered cluster.
2. Go through the samples the second time, and for each sample, find the centroid nearest it. Put the sample in the cluster identified with this nearest centroid. No recomputing of centroids is required at this step.

Both Forgy's and K-means algorithms minimize the squared error for a fixed number of clusters. They stop when no further reduction occurs. They do not consider all possible clustering however, so they achieve a local minimum squared error only. In addition, they are dependent on the seed points. Different seed points can generate different seed results, and outcomes may be affected by the ordering of points as well.

Isodata algorithm¹⁰ is an enhancement of the approach mentioned before. Like the previous algorithms, it tries to minimize the square error by assigning samples to the nearest centroid, but it does not deal with a fixed number of clusters. Rather, it deals with k clusters where k is allowed to range over an interval that includes the number of clusters requested by the user. Furthermore, it discards clusters with too few elements. It merges

clusters if the number of clusters grow too large or if clusters are too close together. It splits a cluster if the number of clusters is too few or if the clusters contain very dissimilar samples. In addition, isodata algorithm does not take a long time to run.

2.2 Shapes of Regions

Once objects are discovered in an image, the next step is to determine what the objects are. The shapes of the object themselves are important to their recognition. A shape has many measures that can define it, and a pure shape measure does not depend on the size of the object. For example, a rectangle that is 8 *cm* long and 4 *cm* wide has the same length-to-width ratio, sometimes called the **aspect ratio**, as a rectangle that is 2 *cm* long and 1 *cm* wide. This ratio is dimension-less, and it only depends on the shape of the object, not size.¹⁸

A ratio of any two lengths of an object could be used as a size-invariant shape measure. One of the important lengths often used is its **diameter**, which can be defined as the distance between the two pixels in the region that is farthest apart. For a rectangle, its diameter would be the length of its diagonal. Likewise, the **path length** of an object could be defined to be the length of the shortest path that lies entirely within the object, between the pair of object points for which this path is longest. The path length of the letter "S" is the length of the path required to write S. Similarly, the **width** of an object can be defined to be the width of the narrowest rectangle that encloses the object. Since rectangles only has two parameters or two degrees of freedom that describe them, its length and width, so one measure of shape will suffice to classify its shape. For more complex shapes or objects with higher degrees of freedom, more shape measures will be required to determine their shapes.

Another example of dimension-less measure of shape is the perimeter squared divided by the area of a region P^2/A . For circles, this shape measure equals to

$$P^2/A = (2\pi r^2)/\pi r^2 = 4\pi = 12.57 \quad \text{Equation 15}$$

for circle of all sizes, and has larger values for all other shapes. For instance, $P^2/A=16$ for all squares and $P^2/A=20.78$ for all equilateral triangles. But P^2/A can approach infinity for long thin or highly branched objects. As a result, P^2/A is modified to $4\pi A/P^2$ so it can be used to produce a shape measure called circularity C , that ranges from 0 to 1.

$$C = 4\pi A/P^2 \quad \text{Equation 16}$$

An object in a continuous image is **convex** if any line segment connecting any two points on the outer boundary of the object lies completely within the object. When an object is not convex, the number of concavities, or their total area divided by the area of the object, can also be used as a shape measure.

Some other interesting shape measures use the concept of **symmetry**. For example, symmetry classes have been used as features in classifying diatoms, which are single-celled algae³³. These microscopic plants are important environmentally because they lie at the bottom of the food chain of a large proportion of all aquatic life. These diatoms show three types of symmetry. Crescent-shaped diatoms have one horizontal line of symmetry. Elliptical diatoms have two lines of symmetry, but if an S-shaped diatom is rotated 180 degrees, it will match its previous position, so it has rotational symmetry.

In breast X-rays, non-cancerous tumors tend to be round while cancerous ones tend to be “spicular” or star-shaped. When a radius is drawn from the center of a rounded object to its edge, it will meet the edge at an angle of about 90 degrees, but for spicular objects, the angle will be considerably smaller, on the average. A shape feature based on this concept has been used in the diagnosis of breast cancer¹.

The medial axis of a region, or the portion of a region left after a technique called thinning, can also serve as a simplified description of the shape and size of an object. Moreover, various parameters based on the branching patterns of the objects can also be used as features for describing shapes. Many other measures of shape are possible. Generally, the nature of the problem determines the types of shape measures that are appropriate to it.

2.3 Texture

The visual perception of texture in an image is created by having regular or random variation in the gray level or color. Various texture features measure the aspects of that variability in different ways. Although human readily recognize a wide variety of textures, they often have difficulty describing the exact features that they use in the recognition process¹⁹.

Features based on texture are often useful in automatically distinguishing between objects and in finding boundaries between regions. If a region in an image consisted of many very small sub-regions, each with a rather uniform set of gray level – such as a mixture of salt and pepper - they would say it had a **fine** texture. But if there regions were large, they would say it had a **coarse** texture. Thus one measure of texture is based on **piece size**, which could be taken to the average area of these regions of relatively constant gray level. Figure 5 shows examples of coarse and fine texture samples.

This piece size average can be taken over some predefined region or object or it can be taken in a localized region about any point in the image to measure its texture. Moreover, the average shape measure of these pieces, such as P^2/A , can be used as a texture measure as well.



Figure 5. (a) an example of fine texture. (b) an example of coarse texture.

Another measure of texture, which still depends on the neighborhood of a pixel, is the **variance** in gray level in a region centered at that pixel. However, this variance measure does not require segmentation of the images into regions first. For example, in a 5×5 region, this variance is defined as

$$T_v(x, y) = \frac{1}{25} \sum_{s=-2}^2 \sum_{t=-2}^2 \left[g(x+s, y+t) - \bar{g} \right]^2 \quad \text{Equation 17}$$

where

$$\bar{g} = \frac{1}{25} \sum_{s=-2}^2 \sum_{t=-2}^2 g(x+s, y+t). \quad \text{Equation 18}$$

Similarly, the standard deviation of the gray level in a region can be used as a measure of texture as well.

Run length is another type of measurement that texture can be classified by. First, the number of distinct gray levels in an image should be limited. Limiting the number of gray level can be done by using thresholds for example. Then the image is scanned line by line, and the length in pixels of each run of a certain gray level is counted. One then can use the average of all these lengths in a region as a measure of the coarseness of its texture. As an example, a three-parameter measure of texture can be based on the average run lengths of dark, medium, and light pixels. Furthermore, parameters of the run-length distributions, such as the variance in the length of the dark run, can be used to detect subtle differences in texture. The relative numbers of light, medium, and dark runs or of various sequences, such as light followed by dark runs, can also be used, depending on how these features vary among the classes to be discriminated.

2.3.1.1 Co-occurrence Matrices

A slightly more complicated two-dimensional measure of texture is based on gray level **co-occurrence** matrices²⁰. The matrices keep track how often each gray level

occurs at a pixel located at a fixed geometric position relative to each other pixel, as a function of its gray level. The $(3,17)$ entry in a matrix for right neighbors, for examples, would show the frequency or probability of finding gray level 17 immediately to the right of a pixel with gray level 3. Figure shows a 3×3 image and its gray level co-occurrence matrix for right neighbors. The 2 in the matrix indicates that there are two occurrences of a pixel with gray level 4 immediately to the right of a pixel with gray level 1 in the image.

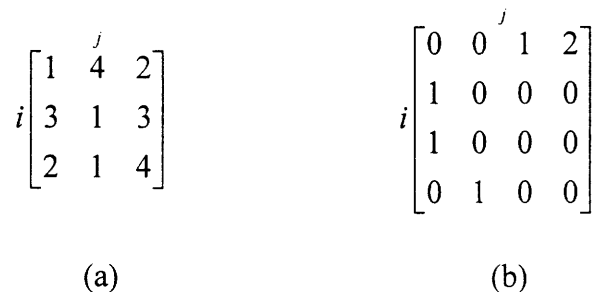


Figure 6. (a) the original image, the entries indicate the pixels (i,j) 's gray level. (b) the *right* co-occurrence matrix of the original image, the entries indicate the frequency of a pixel with gray level i being to the right of a pixel with gray level j .

Each entry in a co-occurrence matrix could be used directly as a feature for classifying the texture of the region that produced it. When the co-occurrence matrix entries are used directly, the number of discrete gray levels may need to be reduced. For example, in a 512×512 pixel images with 256 gray levels, the co-occurrence matrix would be a 256×256 matrix. So the average entry in the matrix will be slightly less than 4, if nearby neighbors are compared. Given this small number, most of the entries probably would be very noisy features for determining the class membership of the region. If the gray levels were divided into fewer ranges, the size of the matrix would be reduced, so there would be larger, less noisy entries in the matrix.

There are many possible co-occurrence matrices. Each different relative position between two pixels to be compared creates a different co-occurrence matrix. A co-occurrence matrix for top neighbors would probably resemble one for right-side neighbors for a non-oriented texture such as the pebbles. However, these two-occurrence matrices would probably differ significantly for oriented textures such as bark or straw. Therefore, comparing the resulting co-occurrence matrices for images can also give important clues to its classification.

If the edges between the texture elements are slightly blurred, nearby neighbors may be very similar in gray level, even near the edges of the pieces, In such cases, it is better to base the co-occurrence matrix on more distant neighbors. For example, the matrix entry m_{ij} could represent the number of times gray level j was found 5 pixels to the right of the gray level i in the region.

Rather than using gray level co-occurrence matrices directly to measure the textures of image or regions, the matrices can be converted into simpler scalar measures of texture. For example, in an image where the gray level varies gradually, most of the nonzero matrix entries for right neighbors will be near the main diagonal because the gray levels of neighboring pixels will be nearly equal. A way of quantifying the lack of smoothness in an image is to measure the weighted average absolute distance d of the matrix entries from the diagonal of the matrix:

$$d = \frac{1}{M} \sum_{ij} |i - j| m_{ij}, \quad \text{Equation 19}$$

where

$$M = \sum_{ij} m_{ij} \quad \text{Equation 20}$$

If all the nonzero matrix entries were on the diagonal, $|i-j| = 0$ for each entry, with the result that $d = 0$. If the neighbors tend to have very different gray levels, most of the entries will be far from the diagonal, and d will be large. Other functions of the matrix have been used to classify terrain²³.

The total length of all the edges in a region can also be used as a measure of the fineness or complexity of a texture. A good source of texture features is the data. Examination of typical scenes to be segmented or objects to be classified will show how they vary in their textures, and features should be chosen that quantify this observed variation.

2.3.1.2 Fourier Transform

There are occasions where global information may be desirable in classify textures such as how often certain features repeat themselves. In these cases, they can consider the images to be waveforms, and they are looking for repetitive elements in the waveforms. The same transform methods that were used for processing waveforms can then be generalized to analyze the variation of the gray levels in two-dimensional images. Although these transform methods, such as the discrete **Fourier transform**²¹, is more useful for analyzing sound waves than it is for images because sound waves tend to be more periodic than most images. In this type of analysis, rapid changes in an image generally correspond to the high frequencies in the waveform while the gradual changes correspond to the low frequencies in the waveform.

Consider an $N \times M$ image for which a typical gray level is $g(x, y)$. The definitions of the two-dimensional cosine and sine transforms are

$$H_1(f_x, f_y) = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} g(x, y) \cos(2\pi f_x x / N + 2\pi f_y y / M)$$

$$H_2(f_x, f_y) = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} g(x, y) \sin(2\pi f_x x / N + 2\pi f_y y / M) \quad \text{Equation 21}$$

The amplitude spectrum is computed as

$$A(f_x, f_y) = \sqrt{H_1(f_x, f_y)^2 + H_2(f_x, f_y)^2}$$

Equation 22

2.4 Matching

Often in image recognition systems, there is a large collection of base objects, objects that unknown objects can be compared to so they can be identified. The comparison of the known base object models with an unknown is called **matching**¹².

Matching techniques are often used in the analysis of aerial pictures. Matching is useful in comparing two aerial images of the same area that are taken at different periods of time. In general, one of the images must be shifted, rotated, and scaled so that corresponding objects in the two images coincide. The required combination of transformations required to match one image with another can be considered to be a **linear transformation** of the coordinate system, from (x', y') to (x, y) , defined by

$$x = a_1 x' + b_1 y' + c_1$$

$$y = a_2 x' + b_2 y' + c_2 \quad \text{Equation 23}$$

The coefficients of the transformation can possibly be found from the heights, positions, and headings of the airplane that were taking the pictures, or by imaging a standard test object in both pictures as the orientation point. Trial and error may also be required to find the best-matching position. A parameter such as the sums of the average absolute difference in gray levels between corresponding pixels in the two images can be calculated for various transformations, and the transformations that gives the best match can be chosen in this way.

2.4.1.1 Graph Matching

Graphs are a very powerful data structure for many tasks in image analysis. They are often used for the representation of structured objects. In image recognition, known objects are often represented by means of graphs and stored in a database. If both known and unknown objects are represented by graphs, the detection or recognition problem then becomes a **graph matching**³ problem.

Generally, the term graph matching refers to the process of comparing two or more graphs with each other. There are several classes of graph matching. In the graph **isomorphism** problem, they want to determine if two given graphs are isomorphic to each other. Two graphs are isomorphic to each other if they are structurally identical. In the **subgraph isomorphism** problem, they are given two graphs g_1 and g_2 , and want to find out if g_2 contains a subgraph that is isomorphic to g_1 . More generally, **bidirectional subgraph isomorphism** between g_1 and g_2 means the existence of subgraph g'_1 and g'_2 of g_1 and g_2 , respectively, such that g'_1 and g'_2 are isomorphic to each other. Finally, in error-tolerant graph matching, they want to find a graph that is the most similar to the one compared to.

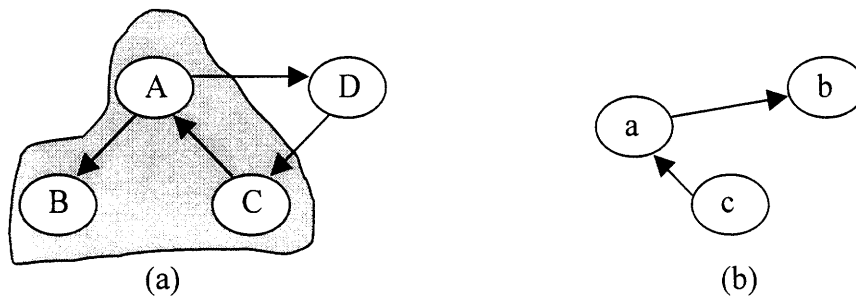


Figure 7: the subgraph enclosed by the gray area in graph (a) is isomorphic to the graph (b) where node A corresponds to a, node B to b, and node C to c.

There are many variations to the algorithms that perform these graph matching problems, the best ones are based on tree searching with backtracking. Each path from the root to the leaves in the tree represents a possible matching of the graphs or subgraphs. The best match is chosen by iteratively minimizing an objective function that represents the distance of the current solution to the correct solution. Graph matching problems are shown to be NP complete problems. This means that all available methods have exponential time complexity. Therefore, although graph matching can return a satisfactory comparison result, it usually takes too long or too much memory space. Most graph matching algorithms utilize some lookahead techniques and prune dead ends in the search tree as early as possible.

2.5 Neural Networks

The reason that human brain can perform so efficiently is that it uses parallel computation effectively. Thousands or even millions of nerve cells called neurons are organized to work simultaneously on the same problem. A **neuron** consists of a cell body, **dendrites** which receive input signals, and **axons** which send output signals. Dendrites receive inputs from sensory organs such as the eyes or ears and from axons of other neurons. Axons send output to organs such as muscles and to dendrites of other neurons¹¹.

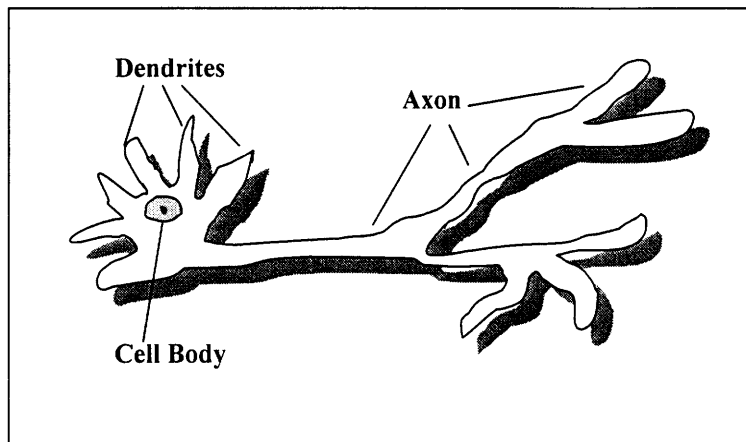


Figure 8: A diagram of a biological neuron.

Tens of billions of neurons are present in the human brain. A neuron typically receives input from several thousand dendrites and sends output through several hundred axonal branches. Because of the large number of connections between neurons and the redundant connections between them, the performance of the brain is relatively robust and is not significantly affected by the many neurons that die every day. The brain shows remarkable rehabilitation by adaptation of the remaining neurons when damages occur.

The neurons in many regions of biological brain are organized into layers. A neuron in a layer usually receives its inputs from neurons in an adjacent layer. Furthermore, those inputs are usually from neurons in a relatively small region and the interconnection pattern is similar for each location. Connections between layers are mostly in one direction, moving from low-level processing to higher coordination and reasoning.

An early attempt to form an abstract mathematical model of a neuron was by McCulloch and Pitts in 1943. Their model receives a finite number of inputs x_1, \dots, x_M . It then computes the weighted sum, $S = \sum_{i=1}^M w_i x_i$ using the weights w_1, \dots, w_m , which are the weights for each of the x_i inputs respectively. After obtaining S , it then outputs 0 or 1 depending on whether the weighted sum is less than or greater than a given threshold

value T . Summations is indicated by a sigma and thresholding is shown by a small graph of the function. The combination of summation and thresholding is called a node in an artificial neural network. Node inputs with positive weights are called excitory and node inputs with negative weights are called inhibitory. Figure 9 shows an artificial neuron representation.

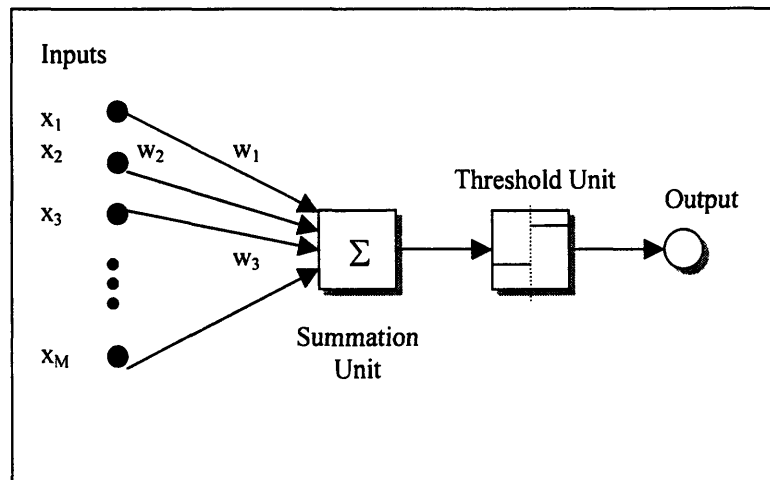


Figure 9. An artificial representation of the neuron. ¹¹

The action of the model neuron is output 1 if

$$w_1x_1 + w_2x_2 + \dots + w_Mx_M > T \quad \text{Equation 24}$$

and 0 otherwise. For convenience in describing the learning algorithms presented later in this section, they rewrite Equation 24 as the sum

$$D = -T + w_1x_1 + w_2x_2 + \dots + w_Mx_M \quad \text{Equation 25}$$

In which output is 1 if $D > 0$ and output is 0 if $D \leq 0$.

A difficult aspect of using a neural net is training, that is finding weights that will solve problems with acceptable performance. In section, they will discuss the back-propagation algorithm for training weights, and later in the section, they will discuss Hopfield's network. Once the weights have been correctly trained, using the net to classify samples is straightforward.

2.5.1.1 Back-Propagation

The **back-propagation procedure**³⁶ is a relative efficient way to train the neural nets. It computes how much performance improves with individual weight changes. The procedure is called the back-propagation procedure because it computes changes to the weights in the final layer first, reuses much of the same computation to compute changes to the weights in the penultimate layer, and ultimately goes back to the initial layer.

The overall idea behind back-propagation is to make a large change to a particular weight, w , if the change leads to a large reduction in the errors observed at the output nodes. For each sample input combination, you consider each outputs' desired value, d , its actual value, o , and the influence of a particular weight, w , on the error, $d - o$. A big change to w makes sense if that change can reduce a large output error and if the size of that reduction is substantial. On the other hand, if a change to w does not reduce any large output error substantially, little should be done to that weight.

Note that any changes to the weight of particular node will also affect and possibly change the weights of nodes that are close to it. This is the propagation effect. This is the reason why this procedure computes the desirable weights of the last layer of nodes first and then works backward to the first layer.

2.5.1.2 Hopfield Nets

A distinctive feature of biological brain is their ability to recognize a pattern give only an imperfect representation of it. A person can recognize a friend that he or she has not seen for years given only a poor quality photograph of that person; the memory of that friend can then evoke other related memories. This process is an example of **associative memory**. A memory or pattern is accessed by associating it with another pattern, which may be imperfectly formed. By contrast, ordinary computer memory is retrieved by precisely specifying the location or address where the information is stored. A **Hopfield**²⁷ net is a simple example of an associative memory. A pattern consists of an n -bit sequence of 1s and -1s.

While the nets discussed in the previous sections are strictly feed-forward (a node always provides input to another node in the next higher layer), a Hopfield net every node is connected to every other node in the network.(See figure 10.) Each connection has an associated weight. Signals can flow in either direction on these connections, and the same weight is used in both directions. Assigning weights to a Hopfield net is interpreted as storing a set of example patterns. After the storage step, whenever a pattern is presented to the net, the net is suppose to output the stored pattern that is most similar to the input pattern.

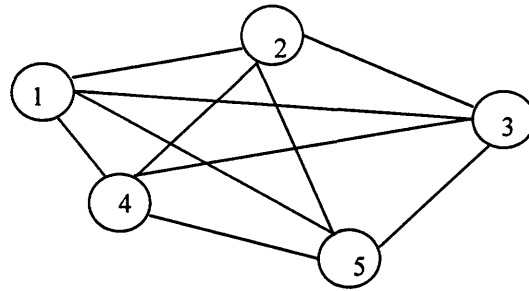


Figure 10. A Hopfield graph.

In the classification step, the n nodes of a Hopfield net are initialized to the n feature values of a pattern. Each node represents one of the n features. Each node concurrently computes a weighted sum of the values of all the other nodes. It then changes its own value to +1 if the weighted sum is non-negative or to -1 if the weighted sum is negative. When the net ceases to change after a certain point in the iteration process, the net is said to have reached a stable state. In practice, a stable state will be detected if the state of the net does not change for some fixed time period depending on the speed and size of the net. If the net functions correctly, it converges to the stored pattern that is closest to the presented pattern, where the distance between two bit strings x and y is the number of positions in which the corresponding bits of x and y disagree. In practice, however, if the net capacity is not large enough, the net may not produce the stored pattern closest to the input pattern. Experimental results show that a Hopfield net can usually store approximately $0.15n$ patterns, where n is the number of nodes in the net. If too many patterns are stored, the net will not function correctly.

Although using a nearest neighbor algorithm to compute the stored pattern closest to the input pattern might be more straightforward, one attractive feature of the Hopfield net is that all the nodes in the net can operate asynchronously in parallel. Suppose that the nodes asynchronously compute the weighted sum of the other nodes and that they re-compute and change their states at completely random times, such that this occurs on the average once every time unit.

Hopfield net have been used to recognize characters in binary images. Although a major advantage of Hopfield nets is that all the nodes can operate asynchronously, a major disadvantage is that every node might be connected to every other node. This can become unwieldy in a net with a large number of nodes.

There are numerous uses of neural nets. Neural net is currently one of the most effective way researchers have found to simulate intelligence in computers. Neural nets are used in areas of optical character recognition, face recognition, machine vision and of course pattern recognition.

Chapter 3 Image Database Research Projects

3.1 Berkeley's Document Image Database

One of the many possible applications of image databases is to handle electronic publications. When paper documents are converted into scanned digitized images, these documents can be easily translated into text by optical character recognition (OCR) products. The UC Berkeley Environmental Digital Library Project²⁶ is one example of such systems. It is one of the six university-led projects that were initiated in the fall of 1994 as part of a four-year digital library initiative sponsored by the NSF, NASA and ARPA. The Berkeley collection currently contains about 30,000 pages of scanned documents. The documents consist of environmental impact reports, county general plans and snow survey reports.

The documents enter the database system through a production scanning, image processing and Optical Character Recognition operation that is currently based on commercial technology. The scanner produces 600 ppi TIFF files at 15-20 pages/minute. The OCR software is an "omni-document" system (Xerox ScanWorX) whose capabilities are typical of those found in other state-of-the-art products and research systems. A metadata record is manually created for each document, containing a typical set of fields such as title, author, publication year, document type, etc. These records are entered into a relational database and fielded search is provided via a form-based interface. In addition, the ASCII files produced by OCR are used for full-text retrieval (currently using FreeWAIS 5.0).

The collection can be accessed through the World-Wide-Web at <http://elib.cs.berkeley.edu/>. Most of the testbed services are available using standard HTML browser, only a few advanced capabilities require a Java-compliant browser. The basic display representation for a retrieved document is the scanned page image, specifically a 75 ppi, 4 bit/pixel GIF file. A simple page-based viewer provides standard next-page / previous-page / goto-page navigation. For rapid browsing, the recognized text of a document can be viewed as a continuous ASCII scroll with embedded links to the page images, a format called hyper-OCR. It also has an advanced viewer that is based on the **multivalent document** (MVD) paradigm. The MVD paradigm will be presented later. One of the features on the MVD browser is the use of word location information provided by ScanWorX. It highlights search terms in the scanned pages and provides text "select-and-paste" from the page image to a text input buffer. Figure 11 shows the web interface for the system.

3.1.1 Multivalent Documents

*Multivalent Documents*³⁵ is a new paradigm for the organization of complex digital document content and functionality. The multivalent view of a document contrasts with monolithic data formats that attempt to encompass all possible content types of the

document in a single, complicated specification, and provide for interaction with them with concomitant complicated editors, often of limited extensibility. This means that, especially for specialized documents, the monolithic type of systems can be overwhelming and lacking simultaneously to the users.

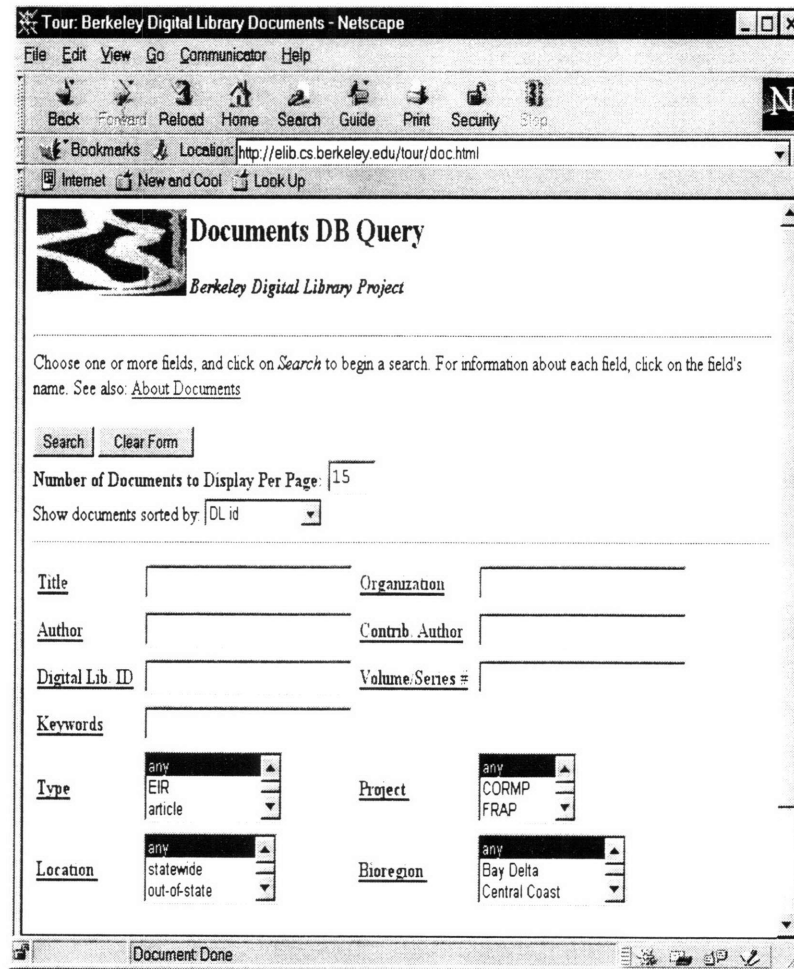


Figure 11. The Berkeley Document Image Database's Web Interface

In contrast, the multivalent approach “slices” a document into layers of more uniform content and then have different operators for each layer acting on the respective uniform content. Additional layers can be implemented when desired. Functionality to these layers is provided by relatively lightweight, dynamically-loaded program-objects, called “behaviors” that manipulate the content. Behaviors of one layer can communicate with other layers and behaviors. Figure 12 shows an example of one document’s semantic layers and the behaviors.

In conceiving of documents as interacting layers of content and functional behaviors that unite to present a single conceptual document, this paradigm is able to introduce new capabilities and benefits. For example, by representing a document as

multiple components, people besides the original document's author can add additional content and behaviors at later time. This also allows new techniques in dealing with documents to be added whenever it is wanted. This paradigm is flexible since these additions can be made locally, without special cooperation of the server of the base document. In addition, by keeping the pieces of content simple, the behaviors that manipulate them are relatively straightforward to define. The components are also more easily reusable.

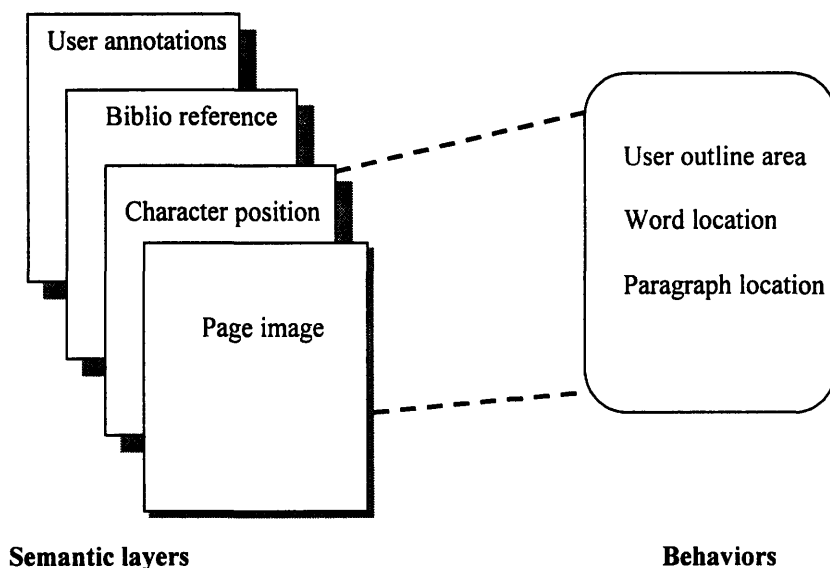


Figure 12. At left, the content of a scanned document conceptualized as semantic layers of information. At right, the programmatic or behavior implementation of a layer.³⁵

3.1.2 System Limitations

The document recognition capabilities of the Berkeley system appear capable of supporting a quite useful level of document indexing and retrieval services. However, there are two important areas that the Berkeley researchers have faced challenges in attempting to develop more advanced services.

First, character recognition accuracy for their system tends to vary significantly across documents and is sometimes unacceptably low. The current OCR technology's accuracy is usually adequate on passages of plain English text. However, the error rate is frequently too high to use the text in new documents (e.g. via the MVD select-and-paste mechanism) without thorough proofreading. Furthermore, many of the testbed documents contain tables of statistical data that the users want to import into spreadsheets, new reports, etc. This material must be transcribed essentially without error in order to be useful. Thus, the error rate is often much too high for ideal operation.

Second, the current omni-document recognition systems recover almost none of the semantic structure of the information encoded in a document. Most high-end commercial systems such as their ScanWorX are oriented toward document reconstruction, which aims to recover, in addition to the basic text flow, information about the typographic style (fonts, column width, etc) and logical structure (sections, headers, footnotes, table of contents, etc) of the document. However, the logical structure of a document is often different than the semantic structure of the information within the document. For example, the logical elements of a report describing fish species might be sections, subsections, the appendix, glossary, bibliography, etc. The information about a fish, on the other hand, might be naturally organized in terms of attributes such as weight, color, fin shape, etc, with no direct correspondence between these attributes and the document elements. The people at Berkeley would like to have their system to extract logical elements from the documents.

The distinction between document structure and content structure is especially important to digital library since most of the users would want to retrieve information to answer a query, rather than to retrieve documents per se. The answer to a query may not be on a single document, or even in textual form. Supporting semantic information retrieval requires that the collection be indexed and accessed on the basis of content, rather than simply document structure. Thus, “omni-document” recognition technology is clearly inadequate in providing much support for true content-based access.

3.1.3 Proposed Solutions to the Limitations

Since the general goal of the document recognition research component of the Berkeley project is to improve the accuracy and structuring of the information extracted from scanned documents, the researchers have been actively working on solving the limitations mentioned above. The technical approach that are currently being following is the use of document-specific decoders within the **document image decoding**²² (DID) framework. DID is an approach to document image analysis that is based on an explicit communication theory view of the processes of document creation, transmission and interpretation.

3.1.3.1 High-Accuracy OCR

A distinctive feature of the DID methodology is its emphasis on document-specific bitmap templates as character models, rather than the “omni-font” feature-based models used by other approaches. The basic is to train a set of templates using a small subset of the pages to be recognized the text well and then use these templates to decode the remaining pages. The training are typically prepared by manual correction. If the document is large enough, the cost of creating the specialized templates will be more than offset by a reduced need for post-recognition proofreading and error correction.

A quantitative performance evaluation of this approach has been completed using DWR Bulletin 155 (B155). This document was selected because of the availability of the WordPerfect source file, which was used as ground truth in assessing OCR performance. The template estimation procedure was applied to a set of 20 page images from B155, using the corresponding WordPerfect source as the training transcription. The training data contained about 40,000 glyphs and 212 different characters (where characters in different fonts are considered distinct). The resulting templates were used to decode 375 additional pages of material from B155, which contained 543,779 glyphs.

The table below summarizes OCR character error rates (substitution, deletions, insertions) using these templates in DID decoders with 3 different language models. The recognition performance of ScanWorX is also given for comparison. The "DID unigram" decoder allows any of the 212 possible characters to follow any other. This is the weakest possible language model and puts the entire recognition burden on the character templates. The "DID bigram" model is the minimal bigram model that includes all of the bigrams that occur in the document, as determined from the WordPerfect source. Finally, the "DID uni+bigram" decoder is a modification of the unigram decoder in which a bigram model, trained on the training data, is used for selected fields of the tables.

Decoder	Substitutions	Deletions	Insertions	Errors (%)
ScanWorX	2149	1069	1061	4279(0.79%)
DID unigram	430	73	80	583(0.11%)
DID uni+bigram	289	72	68	429(0.079%)
DID bigram	87	57	53	197(0.036%)

Table 1. Summary of OCR character error rates using templates in DID decoders. ²²

The character error rate of the DID unigram model is factor of 7 less than that of ScanWorX while the error rate using the unigram/bigram hybrid is a factor of 10 less. The "ideal" bigram model provides more than a factor of 20 improvement.

3.1.3.2 Structured Information Extraction

Automatically recovering structured information from documents is challenging because such structures are document-specific and the range of possibilities is open-ended. As a result, researchers at the Berkeley project expect that some amount of manual effort will be always necessary to configure a recognition system to properly transcribe new types of documents. The goal of DID in this regard is to support the automatic generation of custom recognizers from declarative specifications. The overall vision is summarized in figure 13. The input to a decoder generator is a document model that describes aspects of the document content and appearance relevant to extracting the desired information. Typical elements of this model include specification of the language, information structure, layout, character shape and degradation process. The decoder

generator converts the model into a specialized document recognition procedure that implements **maximum a posteriori** (MAP) image decoding with respect to the model.

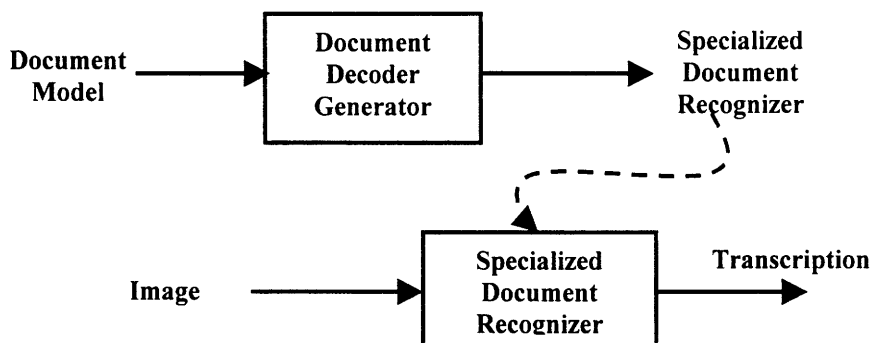


Figure 13. Automatic generation of decoders from document models.²²

Most of this project's efforts have focused on a special class of finite-state Markov image source models. The significance of these model is that MAP decoding can be performed in time that is linear in the size (number of pixels) of the image. Markov sources have proven extremely useful for decoding material that can be viewed as essentially a single column of formatted text lines. A more general class of models, based on stochastic attributed context-free grammar, has been defined that appear powerful enough to describe images containing multiple columns, tables, embedded graphics, etc. However, the computational cost of using these models is high so the researchers have decided not to pursue these options for now.

In order to investigate document-specific structured information extractions, the researchers at Berkeley have initiated a series of decoding case studies using selected documents from the Berkeley collection. The objective is to create a number of examples of **advanced structured documents** (ASD) that provided content-based access to information extracted from the scanned pages. These documents are not necessarily intended to be reconstructions of the corresponding paper documents, but rather new online documents that provide better electronic retrieval.

A total of 5-10 ASD experiments are conducted. Thus far, only three has been completed containing information about water district acts, California dams and Delta fishes. The experiment results can be accessed via the Berkeley project home page or directly (<http://elib.cs.berkeley.edu/kopec/>). The decoders for these documents have the pipeline structure shown in figure 14. Lexical analysis treats each page image as a simple column of text lines and estimates the position, character label and font of each printed glyph. The lexical analyzers are compiled from formal Markov source models as in figure 13 and use document-specific templates constructed as described before. The parser module uses the labeled glyph positions and knowledge of the relationship between semantic structure and page format to identify and label semantic elements and generate the output transcription (e.g. SQL, HTML). Hand-coded "throwaway" routines were used

for the three completed parsers, but they expect future parsers to be generated automatically from attributes grammars as the required technology becomes refined.

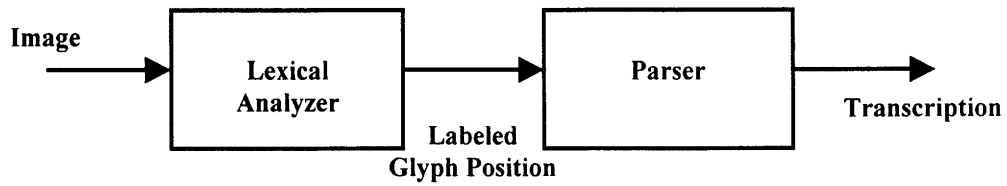


Figure 14. Custom decoder architecture.²²

3.2 Manchester Content-addressable Image Database (CAID)

The **Manchester Content Addressable Image Database (CAID)**²⁷ is one of the earliest database systems that dealt with querying images based on content issues. Unlike the Berkeley project mentioned in the previous chapter, the input images to this system are still pictures captured by video cameras. CAID is developed by the Multimedia group at Manchester University in the United Kingdom. The aim of the Multimedia Group is to investigate the management of raster images, text and simple composite items.

CAID uses primitive image features as an “index” for image content. The primitive features such as lines and spots are computed as part of the data collection and storage part of the database. The system stores pre-computed feature tokens, which are obtained by conventional processing of the input images, into a database which is accessed by a specialized query language called **Manchester Visual Query Language (MVQL)**. MVQL extends the usual notion of a query language to include creation and refinement of groupings based on the stored attributes of the pre-computed features. The user can describe which arrangements of the primitives are comparatively specific to the content sought

3.2.1 Architecture

In the CAID system, the user interacts with a user interface that gives access to an input part of entering image instances, a query part for formulating queries and an output part for presentation of query results and production of hard copy output. Image instances are stored in the database for access by the interpreters and retrieval by the user interface’s output part. An interpreter is provided for each image and interpretation results are stored in the database. The query engine resolves user’s queries according to the interpretation results. The following figure depicts of this design.

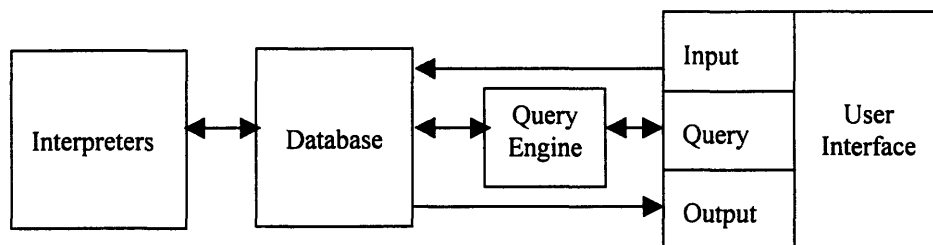


Figure 15. An architecture for a generic CAID.²⁷

The user interface has a frame buffer linked to a video camera. When the user selects the store button from the user interface window, the image is grabbed from the buffer and a label is assigned to the inserted image. Once images are entered, they must be interpreted and their features stored in the database. Once the new features are formed, the image is available for querying.

Similarly, when the “Do Query” button on the user interface is pressed, the query is sent to the query engine which parse it into MVQL search requests and formats the results into a label set. A label set is a ranked set of labels. The labels are used to retrieve the images in the database.

3.2.2 *The Data*

A typical image stored in this system contains a number of objects in some background, along with a scale marker. A **scale marker** is an easily identifiable object of a known size that is used to standardize scalar measurements across images. The objects, scale marker and noise are separated from the background of an image using a standard thresholding technique to produce a modified presentation of the original image. The modified image is then scanned for connected regions of dark pixels called blots. Each blot is stored in the database as an individual object. Each blot in the image is identified as object, noise or scale marker by the image interpreter. A blot is considered to be noise if it is smaller than 750 square pixels in area. It is considered to be a scale marker if its aspect ratio, circularity and transparency lie within predefined ranges. Any other blot is considered to be an object.

The following are the attributes used to describe the blots:

Longest chord: The longest line that can be drawn between any two points on the perimeter, defined by its end points.

Longest chord perpendiculars: the two longest perpendiculars that can be drawn from the longest chord to the perimeter, one on each side. Defined by their end points.

Upright bounding box: the minimum upright rectangle that will fit round the blot, defined by its height and width.

True bounding box: the minimum rectangle that fits round the blot, defined by its four corners, its center and its orientation to the x -axis.

Area: defined in terms of number of pixels.

Perimeter: the length of the perimeter.

Hull area: the area of the blot’s convex hull (analogous to an elastic band stretched round the perimeter)

Length: the length of the longest chord.

Width: the combined length of the longest chord perpendiculars.

Aspect ratio: Length/Width

Transparency: the ratio $\frac{Hull\ area - Area}{Hull\ area}$, transparency is minimum (0) for a convex shape such as 'o' and maximum (1) for a skeletal shape such as 'T'.

Circularity: the ratio $\frac{4\pi \times Area}{Perimeter}$, which gives an idea of how convex a blot is; which gives an idea of how incised a blot is; it is minimum (0) for a line and maximum (1) for a circle.

Irregularity: irregularity is the quantity $\frac{1}{N} \left[\sum_{i=1}^N |\Delta\alpha| - 2\pi \right]$ where N is the number of pixels around the perimeter and $\Delta\alpha$ is the angle change in moving from one pixel to another; it is used to describe the unevenness of a blot's perimeter.

Convex maxima count: the number of major convex extremities on a blot's perimeter.

Concave maxima count: the number of major concave extremities on a blot's perimeter.

Class: each blot is given a shape classification based on clustering in a multi-dimensional attribute space. This is used as an aid for query by example.

3.2.3 MVQL

The feature data is accessed in CAID via a specialized query language, the Manchester Visual Query Language (MVQL)³¹. In MVQL, the image analysis procedure is entirely implicit in the query. Thus it allows the users to intentionally form groups of objects that are of interest to the application in mind.

The idea is to sidestep the problem of creating explicit visual models by providing a query language that enables the user to specify desired spatial and other groupings of primitive features. These groupings are characteristics of an implicit visual model that exists only in the mind of the user. It is not necessary for the user to devise methods to find instances of it. As long as the groupings the user specified are sufficiently characteristic, the CAID will retrieve only image regions of interest.

Starting from sets of primitives, the three main types of operations in MVQL defined are:

1. a pair operation, PAIR – it operates on a pair of sets.
2. a partitioning operation, BIN - dividing up a set into non-overlapping subsets called bins.
3. A selection operation, SELECT – selects elements that meets certain criteria.

These correspond broadly to the join, project and select operations in a relational database. In SQL terms, the equivalencies are

<u>SQL</u>	<u>MVQL</u>
SELECT... FROM A,B WHERE C AND D	PAIR((A,B),{C}) SELECT D
SELECT... FROM A GROUP BY B HAVING C	BIN(A,{B}) SELECT C

3.2.3.1 MVQL Design Ideas

The visual objects are processed in sets in MVQL, with operations taking a set as input and yielding a set of objects as output. Retrieval of lexical values, such as the cardinality of a set, and display of objects in sets are also provided for. All the queries start from feature data that is generated as part of the data entry procedure. In order not to prejudge queries, the data is not partitioned in advance of queries.

The data model chosen is a labeled directed graph with three types of links between nodes, corresponding to three types of relation. Links of the first type form a hierarchical tree structure and they are taken to be synonymous with the class (in object-oriented sense) of the object node. From nodes in this tree, a second type of link represents the attached lexical attributes, which are inherited by child nodes. A third type of link distinguishes which group the attached features belong to. Links are labeled with names (such as Image, Angle) by which they can be referred to.

3.2.3.2 More Detailed Description of MVQL

MVQL queries act on sets of objects which are the primitive features from the images, to yield subsets of objects groups. Operators are available within the language to perform computations on objects' attributes. These include the usual algebraic operators, so for example, if a set of arcs were stored with attributes position (x,y) , orientation (θ) , curvature $(1/R)$, then an attribute <center of curvature> could be computed by using the

expression $(x + R * \sin(\theta), y + R * \cos(\theta))$ in MVQL. Pascal syntax is used for expressions. Further operators allow computations on sets of groups. For example, the COUNT operator yields the number of elements in a group. A more complex operator would be LEAST_SQUARE_REGRESSION which applied to a set of groups of objects with position attribute (x,y) would perform a least squares fit to the positions to yield orientation attributes.

Some of the information that is stored in the structure of the graph can be referenced as if it were an attribute of the connected node. In particular, a reference to an ancestor class as an attribute of the object yields the identifier of the object's ancestor. This permits the manipulation of sets of heterogeneous objects.

Subsets of objects are obtained by using the operators in a Boolean-valued expression which must be true for all the objects in a subset. So for example, given a set S_1 , they can form a subset S_2 with

$$S_2 := \text{SELECT } (S_1, \text{COUNT}() > 10)$$

So that S_2 contains only those groups with more than ten elements.

A more complex example might be

$$S_2 := \text{SELECT } (S_1, (\text{COUNT}() > 10 \text{ AND } (\text{VARIANCE}(\textit{Angle}) < 2.0)))$$

When a grouping of features is created the user indicates at which point in the tree the group is to be attached, using the keyword ATTACHMENT_NODE.

The MVQL PAIR operator creates a set of objects which have properties computed from their constituents. In the following example, the constituents are called FIRST and SECOND. Like wise, the partition operator BIN forms groups with properties (e.g. MEAN, MAX, SUM, COUNT) computed over the elements in the bin. In the following, suppose the database contains arcs detected in images as primitives. Each arc is assigned position coordinates (x,y) on the arc itself, rather than at the center of curvature, orientation (*Angle*) and radius of curvature (*Radius*).

A simple MVQL query for this database can be formed from SELECT operation. For example, the query

$$\text{LargeArcs} := (\text{Arc}) \text{ SELECT } \pi * \text{SQR}(\text{Radius}) > 50$$

Defines LargeArcs as the set of Arcs forming parts of circles with area greater than 50 square pixels.

To illustrate forming a set of pairs, consider the query

$$\text{ArcPairs} := \text{PAIR}((\text{Arc}, \text{LargeArcs}), \{\text{FIRST}[\textit{Angle}] = \text{SECOND}[\textit{Angle}]\}, \text{SELECT } \textit{distance} < 10)$$

Where they use the macro

$Distance = \text{SQRT}(\text{SQR}(\text{FIRST}[x] - \text{SECOND}[x]) + \text{SQR}(\text{FIRST}[y] - \text{SECOND}[y]))$

as an aid to legibility.

The expression in curly braces is the pair-partition, which in this case has the effect that the processing. It will first form blocks of Arcs of the same orientation and compute the distance only between elements in the blocks. The resultant set is a set of pairs of closely spaced parallel arcs, and the number of such pairs is reported on the console.

To illustrate the BIN operator consider the task of grouping Arcs that belong to the same image and fall in the same quadrant of that image. If their image was 256x256 pixels image then the expression $X \text{ DIV } 128$ take on the values 0 and 1 in the left and right halves of the images, and $Y \text{ DIV } 128$ would take on the value 0 and 1 in the top and bottom halves. So a partition of the data by image and by these expressions would yield the grouping required. As an example, the set

`ArcClusters := BIN((Arc), {image(X DIV 25) (Y DIV 25)}) SELECT COUNT > 10`

Thus, Arcs are grouped by image and by position in a 10 by 10 grid of squares superimposed on the image. The resultant set is a set of groups of arcs, each group being associated with one square in one image, and each group having at least ten members. Set reduction is accomplished by a SELECT operation. The query language allows the operations to be nested and in this way quite complex groupings can be specified fairly simply.

3.3 The Alexandria Digital Library Project

The Alexandria Digital Library (ADL)²⁸ project was established in 1994, with funding for four years as one of the six projects of the NSF/NASA/ARPA Digital Library Initiative (DLI). Participants in ADL include the Map and Imagery Laboratory, Departments of Computer Science, Computer and Electrical Engineering, and Geography, and the National Center for Geographic Information and Analysis at the University of California, Santa Barbara; NCGIA sites at the State University of New York at Buffalo and the University of Maine and several corporations, libraries, and agencies.

The primary goal of ADL is to design, implement, and deploy a digital library for spatially-indexed information. A digital library supporting such materials is desired because spatially-indexed information is an extremely valuable resource in many applications but is currently costly to access. Furthermore, although a growing amount of such information is available in digital form, it is still inaccessible to most individuals. The uniqueness of the Alexandria projects is that it attempts to provide a framework for putting these collections online, providing search and access to these collections to broad classes of users through the Internet.

A significant part of their collection includes maps, satellite images, and air photos. For example, the Maps and Imagery library at the University of California, Santa Barbara, contains over 2 million of historical valuable aerial photographs. A typical air photo can take over 25 MB of disk space and providing access to such data by selecting images based on content is a challenge.

First, image features that are needed for search and indexing the database are selected first. Currently, the project has used texture as a basis for content based search. Texture features are extracted as the images are ingested into the database. These texture features are then used in creating an image texture thesaurus which will facilitates faster retrieval of patterns during query time. The salient components include:

- Segmentation based on texture flow.
- Learning similarity measures for comparing patterns in texture feature space.
- Use of hierarchical vector quantization techniques for fast indexing.

3.3.1 Image Analysis

Texture Feature for Content-Based Search

ADL uses Gabor filtered outputs of an image for texture analysis. Some prior experimental results has indicated that the Gabor texture features provide very good pattern retrieval accuracy. This scheme has been used to annotate large aerial photographs (typically 5Kx5K pixels) in the current ADL collection by performing the feature extraction on non-overlapping subimages. A subimage pattern is processed through a bank of Gabor filters at different scales and orientations. The mean and standard deviation of the filtered outputs is then used to construct a feature vector to represent that pattern.

Image Segmentation and Grouping

The purpose of image segmentation is to partition the image into a number of homogeneous regions. An image segmentation scheme that is appropriate for large images and database retrieval applications has been designed. The proposed scheme utilizes the Gabor texture features extracted from the image tiles and performs a coarse image segmentation based on a spatial interaction model for texture flow. From The feature vector, a local texture gradient is computed between each tile and its surrounding neighbors. This local gradient contains information about where the salient texture boundaries are located and how strong the boundary energies are. After a few iterations the propagation reaches a stable state. At the next stage, the local piece-wise boundaries are connected to form an initial set of image regions, and at the end, they use a conservative region merging algorithm is to group similar regions.

After the image is segmented, the mean μ_{mn} and standard deviation σ_{mn} of Gabor filtered outputs of each image region are used to construct the corresponding region texture features, which can be easily obtained by combining texture features in the individual tiles of the region:

$$\mu_{mn} = \frac{1}{N} \sum_{k=1}^N \mu_{mn}^{(k)}$$
$$\sigma_{mn} = \sqrt{\left(\frac{1}{N} \sum_{k=1}^N (\mu_{mn}^{(k)})^2 + (\sigma_{mn}^{(k)})^2 \right) - (\mu_{mn})^2}$$

where $\mu_{mn}^{(k)}$ and $\sigma_{mn}^{(k)}$ are features of the tile k, and N is the total number of tiles belonging to this region.

Currently, each large aerial photograph is first partitioned into 64x64 tiles for texture feature extraction. This results in about 6,400 tiles per image. After segmentation and grouping, there are about 100-200 regions for each large aerial photo. Thus the search process can be performed more efficiently on the segmented image data.

3.3.2 Similarity Measures and Learning

In any content-based image retrieval system, the search for similar image patterns can be considered as a two-step process. The first step extracts the feature vectors to annotate the underlying image information, and the second step is to search through the database to retrieve similar patterns. The similarity measure in the feature space has often been implicitly used to capture the similarity between the original image patterns. However, similarity is a subjective measure. This is particularly true when the image features correspond to low level image attributes such as texture, color, or shape. Simple distance measure such as the normalized Euclidean distance on these features may or may not produced the desired results in a database with maps and aerial pictures.

In order to better the retrieval performance of the texture image features, a learning algorithm was proposed. In this learning algorithm, a hybrid neural network is used to learn the similarity measure in the feature space. The neural network goes through two stages of training. The first training stage performs an unsupervised learning using the Kohonen feature map to capture the underlying feature distribution. The second stage performs a supervised learning using learning vector quantization which partitions the original feature space into clusters of visually similar patterns based on the information provided by human observers.

Once the network is trained, the search and retrieval process is performed in the following way:

- When a query pattern is presented to the system, the network first identifies a subspace of the original feature space which is more likely to contain visually similar patterns.
- The final retrievals are then computed using a simple Euclidean distance measure with the patterns which belong to the corresponding sub-space.

In addition to retrieving perceptually similar patterns, an additional advantage of this clustering approach is that it provides an efficient indexing tree to narrow down the search space. The cluster centers are then used to construct a visual texture image thesaurus, as explained below.

3.3.3 A Texture Thesaurus For Aerial Photographs

A texture thesaurus can be visualized as an image counterpart of the traditional thesaurus for text search. The thesaurus contains a collection of **codewords** which represent visually similar clusters in the feature space. A subset of the air photos was used as training data for the hybrid neural network algorithm described earlier to create the initial indexing tree. Within each subspace, a hierarchical vector quantization technique was used to further partition the space into many smaller clusters. The centroids of these clusters were used to form the codewords in the texture thesaurus, and the training image patterns of these centroids were used as icons to visualize the corresponding codewords. In their current implementation, the texture thesaurus is organized as a two-level indexing tree which contains 60 similarity classes and about 900 codewords.

When an air photo is ingested into the database, texture features are extracted using 64x64 subpatterns. These features are then grouped to form regions. Features are used to compare with the codewords in the thesaurus. Once the best match is identified, a two-way link between the image tile or region and the corresponding codeword is created and stored. During query time, the feature vector of the selected pattern is used to search for the best matching codeword, and by tracing back the links to it, all similar patterns in the database can be retrieved.

The current ADL implementation provides both tile-based and region-based search capabilities using texture image primitives. Tile-based retrieval is useful in searching for local image features such as highway intersections. Region-based search is appropriate for larger geographic features. Currently, they have 40 large aerial photographs in the database which contain about 280,000 image tiles and 6,000 regions. Using the texture dictionary approach, searching for similar image tiles take only couple of seconds. Contrast this with a sequential search which would have taken 5-10 minutes.

Metadata in ADL

When the ADL researchers were designing the schema for Alexandria's metadata, two standards were followed. Both were important for representing spatial metadata. The first standard was the USMARC, a national standard for libraries since the 1960s. The second standard was the Federal Geographic Data Committee (FGDC) Standard. This federally mandated standard was in use since early 1995, with the intent of making a standard set of geospatial metadata. These two standards now have reached an agreement such that FGDC fields which did not exist in USMARC are now added to USMARC.

Neither standard alone fully represent both digital spatial data and analog materials. As a result, both FGDC and USMARC were merged into the Alexandria project's schema. During the original design of the schema, Alexandria used the USMARC local-use fields to create additional fields for analog spatial data not found in either standard. The schema currently has 455 attributes in 81 tables.

3.4 Photobook

The Photobook³⁴ is a set of interactive tools developed at the MIT Media Lab for browsing and searching images. Direct search on image content is made possible by the use of semantics-preserving image compression, which reduces images to a small set of perceptually-significant coefficients.

3.4.1 *Semantic Indexing of Image Content*

The researchers working on the Photobook think the problem of image databases is that to make it a user- and purpose-independent, one must annotate everything in the images and all the relations between them. Text databases avoid this problem by using strings of characters (e.g. words) that are a consistent encoding of the database's semantic content. Thus questions about the database's semantic content can be answered by simple comparing sets of text strings. Because this search is efficient, users can search for their answers at query time rather than having to pre-annotate everything.

To accomplish the same thing for image databases, Photobook researchers believe one must be able to efficiently compare the images themselves, to see if the images have the same or similar semantic content. The user must be able to select "interesting" or "significant" events, and then have the computer efficiently search for more instances of these events. These researchers believe the key to solving this image database problem is in semantic-preserving image compression: compact representations that preserve essential image similarities. The definition of "semantically meaningful" is that the representation gives the user direct access to the parts of the image content that are important for their application.

The Photobook philosophy is that having a semantic-preserving image compression method allows one to quickly search through a large number of images because the representations are compact. It also allows one to find those images that have perceptually similar contents by simply comparing the coefficients of the compressed image code.

All the other image database systems that have been described previously assume that the images had been fully "predigested" into appropriately segmented and grouped line drawings. Then a variety of image indexing methods are used, based on shape, color or combination of such features. This approach is to calculate some approximate invariant statistic, like a color histogram or invariants of shape moments, and use that to stratify the image database. The key problem with this approach is that significant semantic information may get lost in the process. For instance, are apples, Ferraris, and tongues really "similar to each other" because they have similar color histogram? Indexing gives a way to limit search space, but does not answer "looks like" questions except in constrained datasets.

The difference between these methods and the Photobook's is that these methods emphasize computing a key or index into the image, where as Photobook wants to "completely" represent the perceptually salient aspects of the image. Generally speaking, the coefficients these earlier efforts have produced are not sufficiently meaningful to reconstruct the perceptually salient features of the image. In contrast, the models Photobook presents give coefficients that can reconstruct the images.

Another important consequence of perceptual and semantic completeness that Photobook researchers argue is that one can efficiently ask a wide range of questions about the image. For instance, it requires only a few matrix multiplies per image to calculate indices such as color histograms or moment invariants from their coefficients. If one starts with relatively complete representation, then one is not limited in the types of questions one can ask; whereas if one start by calculating indices then one maybe limited to queries about those particular invariants only.

3.4.2 *Semantic-preserving image compression*

The general idea on designing "semantic-preserving" image compression algorithm is to first transform portions of the image into a canonical coordinate system that preserves perceptual similarities, and then to use a lossy compression method to extract and code the most important parts of that representation. By careful choice of transform and coding methods, this approach may produce an optimally-compact, semantic-preserving code suitable for image database operations

Note that because different parts of the image have different characteristics, one has to use a variety of representations, each tuned for a specific type of image content. This necessity of multiple content-specific representations means that one must also have an efficient, automatic method for developing "basis function" specific to object or texture classes. For representing object classes, which require preservation of detailed geometric relations, the Photobook researchers used an approach based on the Karhunen-Loeve transform. For characterization of textures, they use an approach based on the Wold decomposition.

To employ the strategy of semantic-preserving image compression for image database search, one must be able to determine which image data belongs to which content-classes as the data are preprocessed for entry into the database. While this remains a difficult problem in general, the following approach was developed. The method is to recast the problem as one of detection rather than segmentation. The basic idea is that if a content-specific representation can accurately describe some portion of an image, then it is likely to be an appropriate representation of the image data. Thus one can detect instances of models by asking how well the models can describe each part of the image. Although not a real-time process on current workstations, this computation is sufficiently efficient to be incorporated in the image preprocessing step.

3.4.3 Face Photobook

The eigenvector technique is used in this image system. It is performed on each head shot to produce several eigenvectors collectively called as eigenface that serves as the lowest representations of picture. Then the eigenvector technique is extended to yielding eigeneyes, eigen noses and eigenmouths. These eigenfeatures were chosen because studies indicate that these particular facial features represent important landmarks for human eye fixation, especially in an attentive discrimination task. The belief is that by incorporating an additional layer of description in terms of these facial features will assist the recognition process significantly. This approach can be viewed as a modular or layered representation of a face, where a coarse (low-resolution) description of the whole head, eigenface, is augmented by additional (higher-resolution) details in terms of salient facial features, eigenfeatures. Figure 16 shows an example of the markings of eigenfeatures.

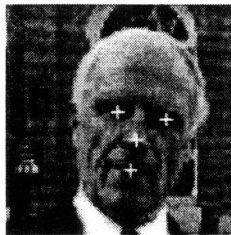


Figure 16. An example of eigenfeatures²⁹

In the eigenfeature representation of faces, the equivalent "distance-from-feature-space" (DFFS) is effectively used for the detection of features. Given an input image, a feature distance-map is built by computing the DFFS at each pixel. The global minimum of this distance map is then selected as the best feature match.

This eigenfeature modular description is also advantageous for image compression and coding purposes. A modular reconstruction which automatically blends reconstructions of each feature on top of the eigenface reconstruction yields much better results than just the eigenface alone, since the position and spatial detail of these regions are preserved the quality of the reconstruction is improved.

The utility of this layered representation (eigenface plus eigenfeatures) was tested on a small face database. A representative sample of 45 individuals with two views per person, corresponding to different facial expressions were selected. These sets of images were partitioned into a training set and a testing set. Then the recognition rates as a function of the eigenface-only, eigenfeature-only and the combined representation were calculated. The outcome of the test was supportive of the researcher philosophy that a combined strategy outperforms the other two. However, the experiment was conducted on a small set of data and there were several interesting points. One is that the eigenfeatures alone were sufficient in achieving an asymptotic recognition rate equal to

that of the eigenfaces. Moreover, that in the lower dimensions of eigenspace, eigenfeatures outperformed the eigenface recognition. So, readers so draw their own conclusions here.

However, a potential advantage of the eigenfeature layer that should not be overlooked is the ability to overcome the shortcomings of the standard eigenface method. A pure eigenface recognition system can be fooled by gross variations in the input image (hats, beards, etc). They performed another test with pictures of the same people in hats and with facial hair. In this test, the results made clear that the modular representation can easily outperform the eigenface-only and eigenfeature-only models.

3.5 The Chabot Project

The **Chabot**³² project is initiated to study the storage and retrieval of images for the State of California Department of Water Resources (DWR). DWR has a large collection open to the public that contains not only pictures of State Water Project Facilities but images of California natural resources as well. The purpose of the Chabot project was to facilitate the task of processing DWR's image requests as much as possible. Prior to the project, these images were stored on Photo-CDs and limited descriptive data were stored in a low functional relational database, so searches were done using key word lookups to find an ID number. Since the requests can vary from where the ID number for the image is already known to a very general request of "scenic pictures" of lakes and rivers, the search task can be tedious.

The project goal is to integrate image analysis techniques into the retrieval system so requests for images do not depend solely on the stored textual information. Since the textual information are already available, the Chabot researchers just needed a more powerful relational database. The researchers also wanted a DBMS that can support a variety of complex data types including text, numerical data, relative and absolute time, and geographical location. Ideally, the retrievals should be possible on any combination of the complex data types that are associated with the images, as well as with the content of the images themselves. Therefore, the retrieval system should be flexible enough to quickly fetch images by ID number, but also able to handle more complex queries that combine several of the attributes of the image. To process a query such as "Find a picture of a sunset taken near San Francisco during 1994", the retrieval system should be able to search on multiple data types such as geographical location ("San Francisco"), time ("after 12/31/93 and before 1/1/95"), and content ("a sunset").

An object-relational database called POSTGRES, which is also developed at UC Berkeley, was chosen to be the underlying database. POSTGRES was chosen because it has tools for complex data types, a rich query system, and extensible types and functions. POSTGRES was also desirable because of it allows user-defined functions and types which can be used to perform run-time image analysis.

Because of the size of the DWR collection, queries that are too general might return a result set of unmanageable size, so steps must be taken to increase the precision of retrievals, thereby reducing the set of images that a user must browse to find the images of interest. More importantly, since the primary data type of this database is the image, standard querying by stored descriptive data will not always yield satisfactory results. Therefore, the system should integrate stored textual information with image content information. What the researchers wanted is to have the user register a conceptual description like "sunset" with the retrieval system. Then the system should respond by initiating the appropriate functions to analyze the content of the images stored in the database that meet the user's expectation of what constitutes a "sunset". This is precisely the approach Chabot took.

3.5.1 Integration of Data Types

Each image in the DWR collection is accompanied by a sizeable amount of metadata. Below is a sample entry for one image from DWR's existing database:

0162 A-9-98 6/1/69 SWP Lake Davis Lahontan Region (6) Grizzly Dam, spillway and Lake Davis, a scenic image. DWR 35 mm slide Aerial 2013 0556 18

In the example, "0162" is the first four digits of the CD number; "A-9-98" is the DWR ID, followed by the date the photo was taken (6/1/69), the category ("SWP"), the subject ("Lake Davis"), the location ("Lahontan Region (6)"), a description of the image, the organization ("DWR"), the type of film used, the perspective of the photo, the last eight digits of the Photo-CD, and finally, the number of the image on the photo-CD.

Furthermore, the schema for the Chabot project was designed to fit with those of other research projects in progress at Berkeley -- a collection of technical reports and a video library. The image class in their database is called PHOTOCOD_BIB, for "Photo-CD Bibliography", which inherits the attributes "title" and "abstract" from the DOC_REFERENCE class, which is shared by the technical report and video object classes. As shown below, the PHOTOCOD_BIB class contains "bibliographical" information about the image object, such as the ID number, the name of the photographer, the film format, the date the photo was taken, and so on.

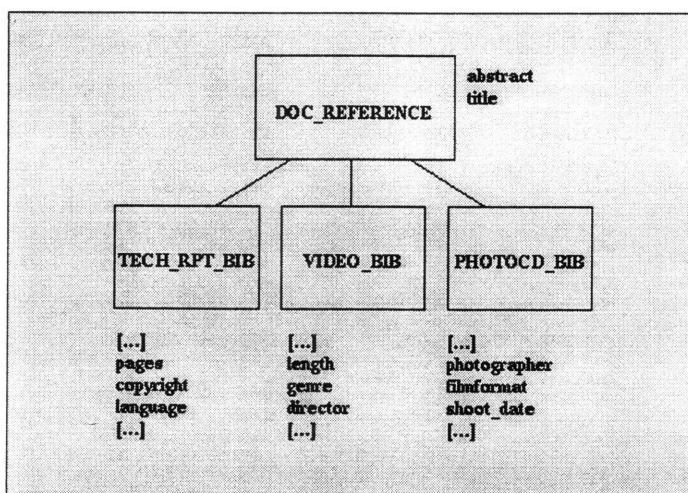


Figure 17. The image class in Chabot³²

3.5.2 POSTGRES

As mentioned earlier, Chabot is using POSTGRES, an object-relational DBMS developed at the University of California, Berkeley, to store the images and textual data. POSTGRES was chosen because in addition to the standard relational database features, it provides features not found in traditional relational DBMS's, such as:

Object-oriented properties: classes can be defined for objects in a POSTGRES database and attributes can be inherited among classes. For example, the PHOT OCD_BIB class mentioned in the previous section on the schema.

Complex types: it provides a flexible assortment of data types and operators such as time (absolute and relative), variable-length arrays, and images. In addition, users can define new data types for a database, along with operators that are particular to the type. For example, a type "PhotoCD" can be defined that includes operators to manipulate the image at runtime.

User-defined indices: a secondary index can be defined using access methods specified by the user. The index can be implemented either as a B-tree or as an R-tree. Partial indices that include a qualifying operator can be extended incrementally. For image analysis, an index can be created for all pictures that are predominantly red, for example, using the stored color histograms for each image.

User-defined functions: functions written in C can be registered with a POSTGRES database. The first time the function is invoked, POSTGRES dynamically loads the function into its address space; repeated execution of the function causes negligible additional overhead since it remains in main memory. For the Chabot database, they wrote a function that analyzes at retrieval time color histograms that have been previously computed and stored in the database.

Even though POSTGRES is the underlying database, it is not the storage for the actual images. Instead each of their images is in Photo-CD format in five different resolutions, ranging from a "thumbnail" (128x192 pixels) to the highest resolution of 2048x3072 pixels. The size of each image is from 4 to 6 MB. Since DWR's goal is to allow on-line access to both images and data, it must provide reasonably fast browsing of the stored images over a network. A random access medium such as a magnetic disk that is fast enough for remote browsing was too expensive to store the large number of images they are storing; cheaper alternatives such as tape may be so slow that on-line browsing is virtually impossible. Their solution is to use a two-level storage scheme. They use magnetic disk for storing the thumbnail images and text needed for browsing the database and they archive the large multi-resolution image files on a tertiary device, a Metrum VHS-tape jukebox. The Metrum holds 600 VHS tapes, each tape having a 14.5 GB capacity. With a total capacity of 10.8 TB, the Metrum is more than adequate as a repository for the DWR image library. The average time for the Metrum to find a tape, load it, and locate the required file is about 2 minutes - too slow for browsing a set of images but fast enough for filling a request from a DWR client once the desired image has been identified.

3.5.3 Image Retrieval Methods

A graphical point-and-click Motif-like interface for Chabot was implemented. The interface is designed to prevent accidental corruption of data while browsing the

database; the main screen gives the user three options: find, edit, and load. The database can be modified only via the edit and load screens and user authorization for these screens is required. The find screen is for running queries and for browsing the database.

The user can build queries by clicking on the appropriate buttons and typing text into the entry fields next to the search criteria. Pull-down menus, indicated by a downward pointing arrow next to the entry field, are provided for some search criteria, those that have limited options such as "Region", "Film Type", "Category", "Colors", and "Concept". The user selects one or more of these fields and then clicks on the button labeled "Look Up" to initiate the query, and a Postquel query, the query language for POSTGRES, is constructed and issued to the database. For example, using the search criteria from the find screen shown, the Postquel query would be:

```
retrieve (q.all) from q in PHOT OCD_BIB where
    q.shoot_date>"Jan 1 1994" and
    q.location~"2" and
    MeetsCriteria("SomeOrange",q.histogram)
```

This query requests all images in the database that were taken after January 1, 1994 in the San Francisco Bay region, and that have some of the color orange in them. When a query such as the one above is processed, the result set of data is displayed in a pop-up "Query Result" window. The user can then print the data, save it to a file, or click on a "Show Image" button, to display the selected images. Up to 20 images can be displayed at once.

3.5.3.1 MeetsCriteria

To implement concept queries, two capabilities that POSTGRES provides were used. One is the storage of pre-computed content information about each image (a color histogram) as one of the attributes in the database, and the other is the ability to define functions that can be called at run-time as part of the regular querying mechanism to analyze this stored information. The function "MeetsCriteria" is the underlying mechanism that is used to perform concept queries. It takes two arguments: a color criterion such as "Mostly Red" and a color histogram. The user selects a color criterion from a menu on the find screen, and a call to MeetsCriteria is incorporated into the query using the selected color.

For the histograms, the system quantified the colors in the images to a very small number so that run-time analysis is speeded up. Tests were conducted on histograms containing 20 elements that were computed using Floyd-Steinberg quantization. The results indicate that quantifying as few as 20 colors allows the system to find the predominant colors in a picture for the "Mostly" queries while still providing a glimpse of the minor colors for the "Some" queries. For example, a picture of a field of purple flowers having tiny yellow centers qualifies as "Mostly Purple", but they can also retrieve this picture using the search criterion "Some Yellow".

The POSTGRES query executor calls the function MeetsCriteria for each histogram in the database, checking to see whether it meets the criterion that is presented. POSTGRES's query optimization facility is used to minimize the search set of histograms. The function returns true if the histogram meets the criterion, false if it does not. Although the method for finding histograms that meet the criterion varies according to which color is being checked, in general the algorithm employs two metrics: compliance and count.

Compliance: Each of the colors in the histogram is checked to see whether it complies with the values that have been pre-defined for the requested color. For example, in the RGB model the color white is represented by (255,255,255) for (red, green, blue); a color whose RGB values are all above 241 qualifies as "white" in their approach.

Count: As they check each color in the histogram for compliance, they keep two counts: the number of colors in the current histogram that have matched the criterion, and the number of pixels contained in the matching colors as a function of total pixels in the image. The former count is used when they are looking for "Some" colors; in the "Some Yellow" example, they get a true result if just one or two of the twenty colors in the histogram qualify as yellow. They use the total pixel count for the "Mostly" matches: more than 50% of the total pixels of an image must be "red" in order for the image to meet the "Mostly Red" criterion.

3.5.3.2 Concept Queries

In addition to using color directly for content analysis, Chabot let users compose higher level content-based queries to the database that embody contextual information such as "sunset" and "snow". These queries are called **concept queries**. The Concepts selection on the find screen of the interface lists the concept queries that are available, each of which has been previously defined by the user.

Selecting a concept from the pull-down menu generates a Postquel query that incorporates a combination of search criteria that satisfy the concept. Typically MeetsCriteria is used in these queries for color analysis in combination with some other textual criteria. For example, when "sunset" is chosen from the Concepts menu, the following query is sent to the database:

```
retrieve (q.all) from q in PHOT OCD _BIB where
    q.description ~ "sunset" or
    MeetsCriteria("MostlyRed",q.histogram) or
    MeetsCriteria("MostlyOrange",q.histogram)
```

In this case, the user has defined the concept "sunset" as including images that have the stored keyword "sunset" associated with them, or images that have red or orange

as their predominant color. Concept queries can be used in conjunction with other criteria. The query "Find pictures of Lake Tahoe at sunset" would be generated by choosing "sunset" from the Concept menu and setting the Location to "Lake Tahoe".

The users are allowed to define new concepts and add them to the Concepts menu by first selecting criteria on the find screen that should be included in the new concept. Clicking on the "Define Concept" button on the find screen brings up a dialog box prompting the user for the name of the new concept. The Postquel query can be edited, after which the user presses the "Define" button to register the new concept. The query is written to a file in the user's home directory, so that the new concept is immediately available, and future invocations of the browser will include it as well. The editing capability can also be used to add postquel constructs that may not be otherwise available, such as disjunctive conjunctions. The user can edit the concept file, make copies of the file available to other users, and incorporate others' concepts in the file.

3.5.4 System Analysis

To test the content analysis, the Chabot researchers measured the recall and precision of the concept queries that are shown in the image retrieval method section. Recall is the proportion of relevant materials retrieved, while precision quantifies the proportion of retrieved materials that are relevant to the search. For each concept query, all the images in the collection were identified by hand that are thought to be similar prior to the test. Various implementations of the concept using different combinations of content-based and stored textual data were tested. Recall and precision for each implementation were then recorded.

The following table shows the results from one of the test queries that is representative of the findings, the concept "yellow flowers". For this concept, 22 pictures in the collection were identified to be relevant; then the "yellow flowers" function in seven different ways were implemented using different combinations of search criteria. As shown below, queries 1-3 used keyword search only, queries 4 and 5 used only content-based information, and queries 6 and 7 used a combination of keyword and content-based data.

#	Keywords	Color Content	Retrieved	Relevant	Recall	Precision
1	"flower"	None	55	13	59.1	23.6
2	"yellow"	None	11	5	22.7	45.4
3	"flower" and "yellow"	None	5	4	18.1	80.0
4	None	Some yellow(2)	235	16	72.7	6.8
5	None	Some yellow(1)	377	22	100	5.8
6	"flower"	Some yellow(2)	7	7	31.8	100
7	"flower"	Some yellow(1)	15	14	63.6	93.3

Table 2. Results of a yellow flower test query. ³²

In this test, two different methods for finding yellow were tried. SomeYellow (2) means there were at least two yellowish colors in a 20-element histogram. SomeYellow (1) means that only one yellow color is needed for the picture to be counted as having "some yellow". As shown for query 5, pictures can be retrieved with 100% recall if the color definition is broad enough, but the precision is too low. The 377 images retrieved from query 5 would require the user to browse nineteen screens of thumbnails (each screen displays 20 images) to find the pictures of yellow flowers. Using the coarse definition for yellow in conjunction with the keyword "flower" gives the best result: query 7 has a recall of 63.6% with a very high precision of 93%.

In some of the tests, a good deal of experimentation was necessary to find the right combination of color content and keywords. For example, for the "sunset on a lake" concept they made several passes through the database testing various colors and keywords before identifying all the necessary criteria that would result in the best recall and precision rates. Thus, the success of the concepts that users define will depend to some degree on their familiarity with the images in the collection. On the other hand, concepts like "yellow flowers" and "purple flowers" are more straightforward and are more easily implemented, especially if care is taken to include some textual information in the concept along with the content-based criteria.

In summary, the researchers found that retrieving images on keywords alone or on content alone produced unsatisfactory results. For example, recall and precision are in inverse proportion: when they retrieve a high percentage of the relevant images, as retrieving all "Mostly Red" images in order to find sunsets, they also retrieve many more images that are not sunsets. But if the search criteria were more restricted by using carefully chosen keywords so that precision increases, fewer of the relevant images are retrieved. For the Chabot's intended application, the best results are achieved when both content and some other search criteria were used.

Chapter 4 Commercial Image Retrieval Systems

4.1 IBM's Query By Image Content

There aren't many commercial image databases that are available in the markets right now. In this chapter, two top-of-the-line image databases that are currently being marketed are introduced. The first system is IBM's Query By Image Content (QBIC)³⁰ system. QBIC is one of the first commercial systems that allow content-based querying. It allows a variety of queries on the color, texture, shape, and position of image and the object they obtain. For example, QBIC allows a query of the form: "Find images that are 17% blue and 11% white".

4.1.1 Image Feature Extraction

QBIC allows queries to be based on objects or the image as a whole. The object queries are based on the properties of the outlined objects, and the image queries are based on properties computed globally over the entire scene. The properties extracted from either objects or images are color, texture, shape and location.

Histogram color is based on a k -dimensional (usually 64 or 256) color histogram of each item. To do the query, a query histogram q is specified by the user (explicitly or by selecting an item and asking for others similar to it), and the distance of the color histogram r for each database item to q is computed as $(r - q)^T A(r - q)$. The histograms r and q are k -dimensional vectors whose i -th element is the percent of color i , and the a_{ij} of matrix A is the color similarity of color i to color j .

Texture features are based on modified versions of the coarseness, contrast, and directionality features. Coarseness measures the scale of the texture (pebbles vs. boulders), contrast describes the vividness of the pattern, and directionality describes whether the image has a favor direction (like grass) or is isotropic (like a smooth object). For queries, distance between a query item and database item is measured as a 3D weighted Euclidean distance between the query texture feature and the database item feature.

Location features are the normalized (x,y) coordinates of an object centroid within its image, and the location distance measure is 2D Euclidean distance. For shape, they used a combination of heuristic shape features (area, circularity, eccentricity, major axis orientation) plus a set of algebraic moment invariants that together give a 20 element shape vector. At query time, shape distance is computed as Euclidean distance in this space.

So far, nothing mentioned above makes QBIC unique among the other systems that have been presented already. However, QBIC's uniqueness is that it also allow sketch queries. Sketch queries let a user to retrieve an image based on a rough sketch of the major lines and edges in an image. They use edge detectors to compute and store an

edge map for each image in the database. In the next section, the two main methods of querying by sketches or painting are described.

4.1.2 Automatic Query Methods for Query-By-Painting

QBIC has two query methods that allow a user to retrieve images similar to an approximate color drawing. These methods are referred to as “query-by-painting”. The user paints, using drawing tools similar to those in common graphics programs, a simplified color image and uses this as the query specification. QBIC retrieves images with similar colors in a similar spatial arrangement. For each of the two matching methods, the following describes how the stored features are used and then how the match engine works.

Partition-based Query-by-Painting

The partition-based approach is done by preprocessing each image and then computes a set of features relative to a rectangular partitioning superimposed on the image. Ideally, the individual partitions should be roughly square. However, for simplicity they typically use one that divides the images into a grid of 6 vertical x 8 horizontal or 9 vertical x 12 horizontal blocks.

For each block, the average color and the five most frequently occurring colors and their frequencies (i.e., a partial histogram) are computed. The standard texture features for each sub-image are also computed and stored. This data – average color, top five colors, and texture parameters are computed and stored for each grid block of the partition for each image.

At query time, the user draws or paints an approximate color/texture drawing. Typically, the user completes the drawing, features for the query are extracted from the image. The features are the same as those described in the previous section, with the exception that unspecified parts of the query drawing area are excluded. These features are processed into distinct query “objects”. An object is a connected component (8-connected in the grid structure) of grids where some query data is specified.

In the matching algorithm, the score or “distance” for a given image in the database is computed by scoring each query object against the image, and averaging the scores for the objects. The score for a query object is the average of scores for each of its grid elements, and the score for a grid element is computed as a function of the element-to-element score over a corresponding neighborhood in the image:

$$d_{grid}(i, j) = \min_{(x,y) \in nbr(i,j)} \{w(x, y)d_{elem}(x, y)\} \quad \text{Equation 26}$$

where $w(x,y)$ is a weight which takes the value of 1 at the center of the neighborhood and decreases as the distance from the center increases. For example, to score the query

object element at grid location (i, j) , they may search image location $\{i-1, i, i+1\} \times \{j-1, j, j+1\}$, weighting location (i, j) more heavily to favor correct spatial alignment, and use the minimum of these individual scores.

The element-to-element distance score is a weighted linear combination of the 3D Euclidean average color distance, color histogram distance, and 3D Euclidean texture distance for each element:

$$d_{elem}(i, j) = w_{avg}d_{avg}(i, j) + w_{hist}d_{hist}(i, j) + w_{txt}d_{txt}(i, j) \quad \text{Equation 27}$$

These weights can be adjusted at query time to allow the different feature components to emphasize or de-emphasize.

Region-based Query-by-Painting

A second approach to query by painting does not rely on a fixed grid placed on the image. It uses an approximate segmentation of each database image and of the query drawing into a hierarchical set of colored rectangles, and bases the matching on these hierarchical sets.

For each image, the rectangle hierarchy are generated as follows. The colors occurring in an image are clustered into a small number of colors, typically 5 to 15, depending on the image. At each iteration, a pair of clusters P and Q is collapsed into a single cluster if the mutual rank of P and Q falls below a preset threshold. The mutual rank of P and Q is $m+n$, where Q is the n th closest cluster to P and P is the m th closest cluster to Q . "Closest" here is in the terms of some color metric on the average color of the cluster, e.g. Euclidean distance in RGB. Color distance and spatial distance are combined during this clustering stage by forbidding the collapsing of two clusters P and Q if the spatial correlation of P and Q falls below a preset bound. The spatial correlation of P and Q is a measure of the similarity of the spatial distribution of the image into rectangular cells and calculating

$$\frac{(\sum_c p_c q_c)^2}{\sum_c p_c^2 \sum_c q_c^2} \quad \text{Equation 28}$$

Where P_c is the number of pixels corresponding to cluster P that occur in cell c . The iteration stops when no pair of clusters can collapse.

Then, for each color, the connected components of the pixel population having that color are identified. For each connected component a bounding rectangle is calculated. This is taken to be the rectangle having the smallest area amongst all tilted or rotated bounding rectangles.

The bounding rectangles for a given color are successively clustered into groups of geometrically close rectangles. For each group, a tilted, bounding rectangle of least

area that encloses the group is calculated. This process is done repeatedly until one rectangle remains. The result is a hierarchical tree structure having a rectangle at each node that bounds the rectangles at its child nodes. Each leaf node corresponds to an initial bounding rectangle. Each root node corresponds to a cluster color.

As the rectangles are clustered, if the area of the rectangle corresponding to a node is less than a fixed multiple of the total area of the connected components of its children, then the node becomes a leaf (i.e., the tree is pruned). This pruning reduces the size of the feature set for this image and speeds up the query response time.

At query time, the region features of the query are matched against the region features for each image in the database. To do this, for each color region of the query, a distance to each color region of the image is calculated as a weighted sum of the distance between the colors themselves and the distance between their associated trees (the tree distance calculation is described below). The total distance between the query and the image is the average of the distance between each query color region and its closest image color region. The comparison of a query tree against an image tree uses an asymmetric distance function $d_{rect}(Q, R)$ that measures how well a particular image rectangle R accounts for a particular query rectangle Q . The distance function is computed as

$$d_{rect}(Q, R) = \frac{\alpha \text{area}(Q/R) + \beta \int_{x \in Q} \|(\text{closest point in } R \text{ to } x) - x\|^2 dx}{\text{area}(Q)}$$

Equation 29

This equation is the weighted sum of two terms: α times the percentage of area of Q falling outside R plus β times the average squared distance of points in Q to their nearest points in R . The weighting factor α and β are preset parameters. This function returns 0 if the query rectangle lies entirely inside the image rectangle, and is bounded below by β times the minimum squared Euclidean distance, $d_{min}(Q, R)$, from any point in R to any point in Q .

The comparison of a query tree against an image tree begins by matching the query root node with the image root node. Then it proceeds recursively, matching each child of a query node against the closest child of the matching image node. If either the image or query node is a leaf, the matching proceeds keeping this node fixed and traversing the rest of the corresponding tree. Because $d_{min}(Q', R') \geq d_{min}(Q, R)$ for any rectangle $Q' \subseteq Q$ and $R' \subseteq R$, a lower bound on the total matching distance between the leaves of a query tree Q and the leaves of an image tree R can be calculated as

$$\beta \sum_{n \in N} d_{min}(\text{rectangle}(n), \text{rectangle}(\text{match}(n))) \cdot (\text{number of leaves in } Q \text{ descendent from } n)$$

Equation 30

where the sum is taken over any set N of nodes in Q such that no element of N is a descendant of any other. $\text{Match}(n)$ is the node of R calculated as matching the node n of Q during the recursive matching algorithm described above. This lower bound can be used to abort a matching at the point it exceeds the minimum matching distance among previous trial matches of query tree Q against image trees.

Of the two spatial color based matching methods, the region method tends to be more robust to shifts of the query color regions with respect to those of the image because the matching distance increases smoothly as the query regions move from their optimal matching positions.

4.1.3 Object Identification

In the previous section, two query methods were described which require no manual intervention at database population time. Such methods are still not able to accurately match on detailed object shapes, small objects, etc. For more precise matching, object identification at population time is still required. Because this can be tedious, they want to facilitate this step with semi-automatic methods. This section will describe two such methods, an enhanced floodfill tool and an edge follower.

Enhanced Floodfill

Floodfill algorithms for identifying an image object are common and available in both research and commercial systems. A typical algorithm works by having the user select a pixel or small area, thereby defining the initial set of object pixels. Any pixel touching an object pixel and whose value is within some range r , say plus or minus 10, of the value of the initial set of object pixels is added to the set, and this process is repeated until no pixels are added. The value of r is typically set by the user with a slider or keyboard entry. The proper setting is dependent on the image material surrounding the object, the uniformity and contrast between the object and the background, and other factors. For each identified object the user may need to set or change it. Also, floodfill methods may “leak”, that is expand outside the object boundary.

Enhancements are added to the basic floodfill methods to solve these problems. First, they let the user select one or more background areas (pixels, or groups of pixels) in addition to selecting the initial object pixels from which to start the floodfill. The background pixels are compared with the starting pixel(s) and a threshold is determined automatically based on the contrast between the background and object. In their system, the default is to set r at 80% of the minimum difference between object and background pixels. In this way, they have replaced the adjustment of a slider or numeric field with a mouse click. Secondly, they allow “pruning” of a filled object. The user can draw a line or curve on a floodfilled object, dividing it into two portions, and delete one portion by

clicking on it. Finally, a user can draw a “blocking line” (or curve) to block the expansion of the floodfill into an area. These additions have provided us with a more effective outlining tool.

Snake-based Edge Following

The floodfill tool allows a user to identify objects by clicking on pixels both inside and outside the object. It is thus an area-based tool. They also have an edge based tool. While a user draw a boundary curve along an object, this tool interactively adjusts to the boundary to better match the object edge. In this way the user does not have to follow all the boundary details, thus expediting the outlining process.

The method is based on snakes, introduced by Kaas as energy minimizing curves guided by external and internal forces. In their case, they formulate the problem so that the user drawn contours are represented as splines that are attracted to strong image edges. Parameterizing the problem in terms of a small number of control points yields a minimization procedure that is fast enough to give them an interactive, rubber-banding effect as the user draws with the mouse. This feed back allows users to quickly guide the snake on the desired path.

A B-spline curve is defined by its control points x_0, x_1, \dots, x_N as

$$f(t) = \sum_{i=0}^N x_i N_i(t) = \bar{x}^T \bar{N} \quad \text{Equation 31}$$

where \bar{N} is the vector of the B-spline basis function with element $N_i(t)$. A simple use of a B-spline curve in an energy-minimization formulation or deformable snake is

$$E_{image} = \int |\nabla I(\bar{x}^T \bar{N})| dt \quad \text{Equation 32}$$

where \bar{x} are the unknown B-spline control points that define the snake and ∇I is the gradient image. The problem is to find the control points that minimizes E_{image} , the integral of the magnitude of the gradient image over the path defined by the control points. This is a non-linear minimization problem.

To perform the computation for E_{image} , they pre-compute the gradient image (∇I) using a recursive filter proposed by Deriche for a fast computation of a smoothed gradient. The smoothing helps by reducing noise and widening edges. The integration is done by summing the digital gradient image along the B-spline path. The evaluation speed is dominated by the speed at which they can render a B-spline. For this they use a fast rendering algorithm which generates a poly-line that can be rendered/summed with the gradient image.

4.2 The Virage Image Search Engine

As presented earlier, several imaging systems have emerged in both research and commercial sectors that are doing “content-based” image retrieval. However, each of these systems is relatively heterogeneous to each other. One system can not be easily substituted for another to perform the same intended queries. This is what is unique about the *Virage image search engine*²⁴ approach. It is an attempt to provide an open framework for building any imaging systems. A brief description of this framework is given below.

4.2.1 Virage Data types

In general, a content-based image retrieval system creates an abstraction of the raw visual information in the form of features, and then operates only at the level of these abstracted features. Virage has categorized the features to the following data types: values, distributions, indexed values and indexed distributions. A **value** is a set of vectors that may represent some global property of the image. The global color of an image, for example, can be a vector of RGB values, while the dominant colors of an image can be defined as the set of k most frequent RGB vectors in an image. A **distribution**, such as color histograms is typically defined on an n -dimensional space which has been partitioned into b buckets. Thus, it is a b -dimensional vector. An **indexed value** is a value local to a region of an image or it can be multi-dimensional as in the orthonormal bounding box coordinates covering an image segment. An **indexed distribution** is a local pattern such as the intensity profile of a region of interest, and can be derived from a collection of b -dimensional vectors by introducing an index.

The idea is that the vectors form a uniform base type for features which in term represent image content. The primary data type in the Virage Engine is an indexable collection of vectors. The primary operations on this abstract data type are:

Create collection – this operation creates an empty collection of vectors. The specific form of the collection (e.g., list, set, multiset) is dependent on the implementation.

Create vector – this operation takes a collection, an image and a function as arguments and extracts a specific feature vector from the image. The computed vector is placed in the named collection.

Extract – this is a generic operation to access an element of the collection. Its arguments are the collection and a user-specified function to specify the element of interest in the collection. This is a non-destructive operation and leaves the initial collection unaltered. This function can utilize any indexing schemes available to the operation.

Distance – this operation compares two vectors and returns a measure of the distance between them. It is required that each vector of a collection has corresponding distance function for comparison.

Combine – this operation creates a new vector by combining two given vectors with an admissible user-specified function.

Delete vector – this operation is required to free the memory associated with a particular vector.

Delete collection – this operation deletes the collection from transient and/or persistent memory.

4.2.2 Primitive

In the Virage Engine, the primitive is a collection of vectors represents a single category of image information. Thus, a primitive is a semantically meaningful feature of an image. Thus color, texture, and shape are all general image primitives. Of course, not all primitives will be applicable across all images. For instance, a color primitive may have no relevance with respect to X-ray imagery. In practice, a primitive is specified as a 6-tuple of the following values:

Primitive_id – a unique primitive identifier.

Label – a category name for the primitive.

Compute_function – this function accepts the image data and computes its visual feature data and stores it in a buffer.

Distance_function – this function returns the similarity score for its associated primitive. The query operation of the engine call this function with two data buffers (previously created with *compute_function()*) to be computed. The score which is returned must be in the range from [0.0...1000.0]. For maximum discrimination, the spectrum of distances returned for this primitive should be spread over this range evenly or in a reasonably smooth distribution.

Swap_function – the engine takes full responsibility for handling the byte order difference between hardware platform, regardless of byte-order differences. Each primitive supplies this function which will do the byte-order conversion for its own data. The engine will automatically use this function when necessary, to provide consistent performance across any platform.

Print_function – this function is used to print out the desired information of the associated primitive.

After a primitive is defined, it is registered with the Virage Engine using the *RegisterPrimitive()* function. In addition to the above primitive information, an estimated cost of comparison may also be supplied for the primitive, to aid in query optimization by the engine.

4.2.3 Schema definition

A Virage Engine schema is defined as a 2-tuple: a schema id, and an ordered set of primitives. Similar to primitives, the Virage Engine is notified of a new schema by a *RegisterSchema()* function. The primitives Ids referenced here must have previously been defined using *RegisterPrimitive()*, or must be one of the built-in primitives. The order in which the primitives are referenced dictates the order in which their functions are called during feature extraction, but not during query processing. This ordering allows primitives to work synergistically and share computational results. A single application is allowed to define and use multiple schemas. The Virage Engine operates as a stateless machine and therefore does not manage the data. Hence the calling application must manage the storage and access of the primitive data computed from any schema. The following paragraphs describe the types of operations available to applications utilizing the Virage Engine.

Before an application can determine the similarity between an image description and a set of candidate images, the image must be analyzed by the engine. The resulting feature data is returned to the caller to be used in subsequent operations. Naturally, if an image is to be a candidate in future operations, the feature vector should be stored in a persistent manner, to avoid re-analyzing the image.

Analyze_image – this function accepts a memory buffer containing the original image data. It performs an analysis on the image by invoking the analysis functions of each primitive. The results of this computation are placed in memory and returned to the caller, along with the size of the data. Maintenance and persistent storage of this data is the calling application's responsibility. Eventually, these structures are passed into the image comparison entry points.

Destroy_features – this function is used to free the memory associated with a visual feature that was previously returned from *analyze_image()*. Typically, this is called after the application has stored the data using the associated persistent storage mechanism.

Any image retrieval application requires the ability to determine the similarity between the query description and any of the candidate images. The application can then display the computed similarity value of all of the candidate images, or convey only the most similar images to the user. To do this, similarity scores are computed by the Virage Engine for the relevant candidate images. An application will call the comparison function provided by the Virage Engine. These functions will return a score structure, which indicate the similarity between the images being compared. The score structure contains an overall numerical value for the similarity of the two images, as well as a

numerical value for each of the primitives in the current schema. This allows applications to use the values of the individual primitive comparisons, if necessary. Following are some of the functions relating to the management of the score structures.

Create_scores – this function is used to create a scores structure for later use by the function *compare_into_scores()*. It is a cache of primitive level scoring information that can be efficiently reused to compute a new overall score given a new set of weights. The application is responsible for managing the returned score structure. The function *destroy_score()* must be called to deallocate the memory for this structure.

Get_score – this function is used to access individual score values from the score structure. It provides the mechanism for retrieving the scores for individual primitive values within the structure.

Refresh_scores – this function will compute an overall visual similarity distance given a cached scores structure returned from *compare_into_scores()* and a new set of weights. This computation is very efficient compared to re-computing all of the score data from the feature vectors.

Destroy_scores – this function deallocates a previously allocated score structure created by *create_scores()*.

When two images are compared by the engine, each of the primitives in the current schema are compared to give individual similarity values for that primitive type. Each of these scores must then be used to provide an overall score for the comparison. In certain situations, these individual primitive scores may need to be combined differently, depending on the desired results. By altering the ways these individual scores are combined, the application developer has the ability to indicate relative importance between the various primitives. For example, at times the color distribution of an image will be much more important than its texture characteristics. There may also be cases where only some of the available primitives are required in order to determine which images should be considered the most similar. Applications are given flexibility in how the overall score is computed through use of the weight structure. The application has control over the weight values for any given comparison through the weights structure, and the following functions:

Create_weights – this function is used to allocate a weight structure for use in the compare functions. The associated *schema_id* will determine the specific format of the structure.

Destroy_weights - this function is used to free the memory previously allocated with *create_weights()*.

Set_weight – this function sets the weight in the weights structure identified by the given *primitive_id*, which identifies the primitive whose weight is to be set. The value should

be a positive floating point number. In general, weights are normalized before use by calling *normalize_weights()*.

Get_weights – this function is used to extract an individual weight value from a weight structure.

Every application will have unique requirement in the way they determine which image are to be considered most similar, and how to efficiently manage a changing set of results. Certain applications may need to do an exhaustive comparison of all images in the candidate set, while others are only “interested” in a certain set which are most similar to the query description. Certain applications or situations may also require the ability to quickly manipulate the relative importance of the primitives, using the individual primitive scores and weights, as discussed above. Some of more common comparison functions which address these requirements are described below:

Compare – this is the simplest entry point for computing the overall visual similarity for two given images, represented by their respective visual features. The caller passes in a weight structure and *compare()* computes and return the weighted overall score, which is a numerical value in the range [0.0...100.0]. This function can be used when a score is required for every candidate image. If only the top N are required, the function *compare_less_than()* may be more appropriate. Subsequently, if scores are desired for additional sets of weights, then use *compare_into_scores()*.

Compare_less_than – this function can be used for optimized searches in which the scores of every single candidate image are not requires. A threshold similarity distance is passed in to indicate that any image whose score is above the threshold is not of interest for this search. As soon as the engine determines that the image is outside the range, it terminates the similarity computation and returns a flag to indicate that the threshold has been exceeded. This provides a significant performance boost when top N style searches are sufficient. Again, it is the application’s responsibility to determine the appropriate threshold values for each comparison.

Compare_into_scores- this function also performs comparison, but it returns a score data structure that caches scoring information form each primitive. The score data, like the visual feature data, must be managed by the application. It can be used effectively in situations where a ranking of candidate image is desired for several sets of weights. One companion function to this is the *refresh_scores()* function which can quickly compute a new overall score given one of these score structures and a desired set of weights. It is also possible for the developer to use *get_score()* to extract component scores form the structure and perform a custom combination of these to achieve a final score.

The Virage Engine sounds quite promising and seems to provide the fundamental capabilities that are require for building image content-based retrieval systems. Informix has adapted Virage’s technology into their database technology for image retrieval. A demo for the Virage Engine can be found at <http://www.virage.com> and figure shows the Web interface for the Virage image search engine.

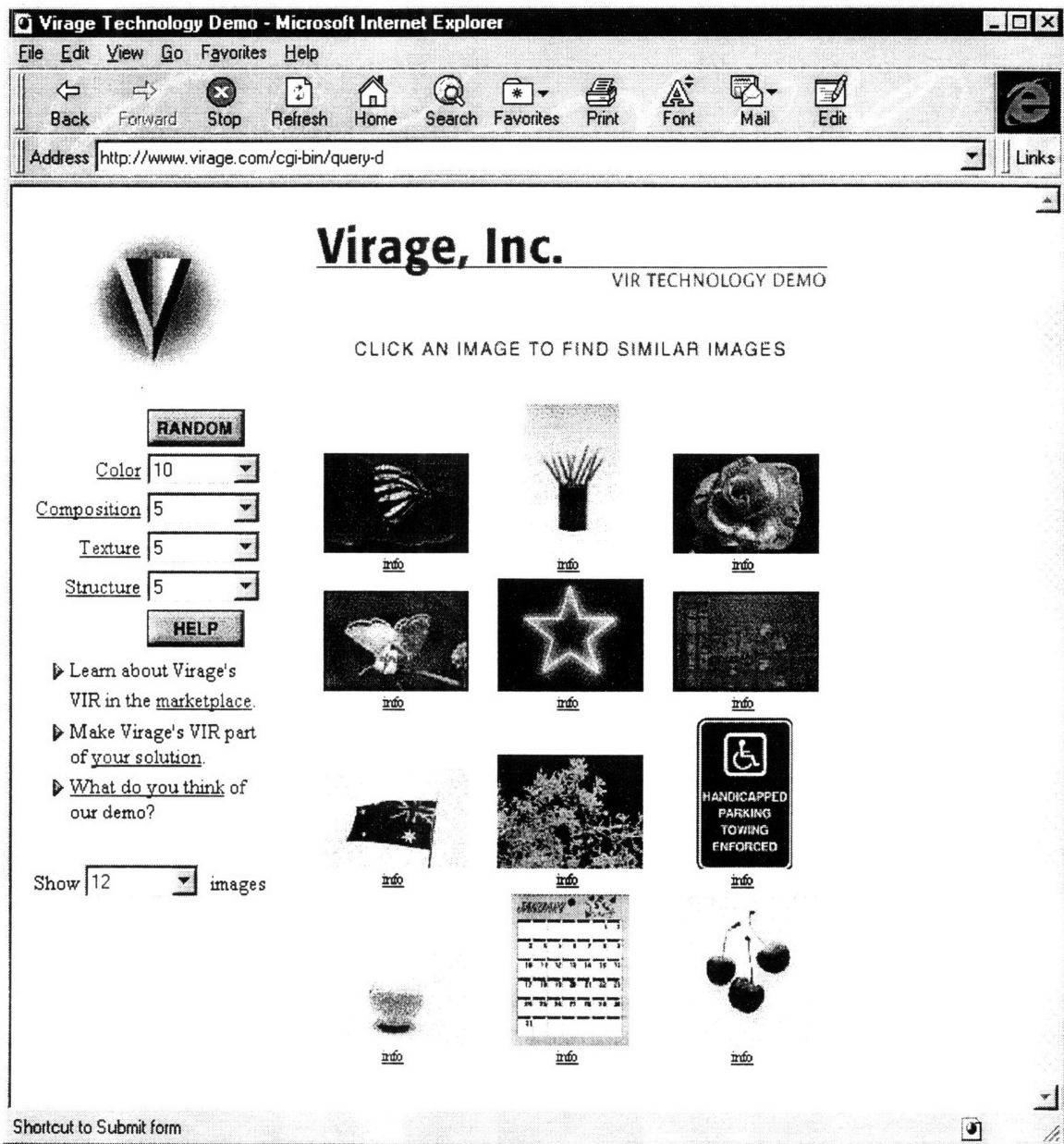


Figure 18. The Web user interface to Virage's Image Search Engine Demo.

4.3 Metadata

In the traditional library, the card catalog has provided the primary mechanism for search and information discovery. People can lookup titles, authors, subjects and keywords on the books and search for the ones they want from this catalog information. In the image databases, this process can be generalized as well. Now is possible to envision a process of automating image cataloging, based on computer analysis of each image object's contents. The term commonly used to refer to such catalog information is *metadata*. In this chapter, the current development of metadata for image databases is examined.

4.3.1 What is Metadata and Why use it

A definition of **metadata**⁴ devised by Francis Bretherton is that metadata is "information that makes data useful". Such metadata might include three subsets:

1. information necessary for the successful handling and use of the data, such as its format or location;
2. information that generalizes or abstracts the data to identify salient characteristics of its content, and thus supports the functions of search and browse;
3. information typically not available in the content that affects usefulness, such as information on quality.

In image databases, the first and third are in principle content-independent, not obtainable from the content; the second is content-dependent, but often the generalization and abstraction function is sufficiently complex that it cannot be automated. For example, a map legend is pure metadata. The legend contains information about the publisher of the map, the publication data, the map's scale and its accuracy, among many other things. Metadata is simply a descriptive information on information, or data about data. Metadata are a "common" set of terms and definitions to use when documenting certain types of data such as books or maps.

Creating metadata may seem burdensome, but there are numerous benefits that easily outweigh the trouble. Using metadata increase the value of such data by facilitating data sharing through time and space. People who use the same metadata standards can share information continents or decades apart. Metadata makes information sharing possible and easy. Books that were cataloged ages ago can still be searched using that cataloged information now. If there is a disciplined, requirements-based approach to indexing images, it would make interoperability of image databases much easier to achieve.

4.3.2 Issues

An image metadata standard seems like a great idea, but there are many obstacles in the way. For example, the analogy to the traditional card catalog seems to provide an obvious basis for defining image metadata, but the traditional card catalog is built on a rigid structure of information granularity, since all information objects in the traditional library are individually bound books. In the digital world, granularity of information is harder to define. For example, one might want to merge individual satellite images into a seamless view of the Earth's surface, or to separate the several layers of information shown on a single picture into several separate objects.

In addition, unlike traditional book cataloging, image database is unlikely to rely entirely on the metadata for searching. An image database needs to allow direct search of the contents of an information object, particularly for domain-specific information or minor detail that a catalog information would normally lack. For example, for the geographical information retrieval community, their focus is on spatial objects and therefore would place great importance on geographic elements of metadata. On the other hand, to the general document library community, with its emphasis mainly on bound volumes of text, geographic extent is a comparatively minor element and therefore would be burdensome to store as metadata.

Furthermore, while some elements of metadata might be derived automatically through analysis of content, or inserted by the software that creates the object, it is clear that this cannot apply to all elements, such as those related to successful handling of the object. Here lies the issue of what features should be included in an image metadata standard. An agreement between all possible domains will be hard to reach since it is unlikely that any certain manageable set of features as a metadata standard can uniformly benefit all possible image retrieval systems.

4.3.3 Current Status

There is something that most researchers can agree on. The contents of image metadata should address the following three dimensions. The first is *structure*, defined as the information needed to understand the arrangement of bits and bytes in the object. The second is *context*, defined as the meaning of the bits and bytes in more abstract, conceptual terms, and the general properties of the object that describe its relationship to the broad domain of other objects. And the third is *content*, defined as the specific detail contained in the object. For example, a remote sensing scene might be classified as follows:

- structure: the file format;
- context: an image (ID) collected on (date, time);
- content: has 80% cloud cover, shows a country scene.

To find the image, people normally rely on structure and context. But to evaluate the image both content and context are needed. Similarly, to use the image people usually require structure and content. Even though it is hard for people from all image domains to agree on a specific set of content information to be stored, but there are some domain-independent metadata that may be helpful to all types of usage.

From the current trend, it seems various niches will emerge in which metadata efforts will be comparatively successful. For example, metadata for general documents, metadata for maps, metadata for music scores, etc. But the issues of compatibility still needs to be addressed. Perhaps, the future domains of metadata may be defined by access conditions, such that users pay access may support more extensive metadata than open access, for example. For now, discussion of image metadata only within the geographical information retrieval community makes sense, since it deals with quite structured geographical and spatial information on its data.

There are a couple standards being formed for geographical related images. For instance, the Federal Geographic Data Committee (FGDC) recently adopted content standards for graphical metadata. According to an Executive order signed by President Clinton on April 11, 1994, all Federal agencies will begin to use these standards to document newly created geospatial data as of January, 1995 after receiving the appropriate training. These standards provide a consistent approach and format for the description of data characteristics. The standards were developed over a two-year period, with extensive review by professionals at all levels of government. They provide a way for data users to know. Because large amounts of Federal data will be available in these standards, data managers from State and local governments and private industry will have an incentive to adopt these standards to document their own data.

The FGDC is also sponsoring the creation of a National Geospatial Data Clearinghouse which will point users toward the best spatial data for a particular project. The intent is not to centralize all geographic data in one location, but to provide links through the Internet to distributed sites where data are produced or maintained. Managers who document data using the metadata standards will provide these metadata to the National Geospatial Data Clearinghouse so that users can easily find data. Easier access to data will mean that a company's customers or an agency's cooperators could be increased. . The initiative taken by the Federal Geographic Data Committee (FGDC) in promoting its Content Standards for Digital Geospatial Metadata can be found at: (<http://GeoChange.er.USGS.gov/pub/tools/metadata/standard/metadata.html>).

Chapter 5 Designing an Image Database

Digital image management is becoming a fast growing field because of the growing popularity of imaging peripherals such as digital cameras and scanners. These technologies aided by the opportunities opened by the Web can help anyone, any institution, or any company create, store, manage and produce images. There are many ways to manage an image database and the management methodology should be dependent on the intend application. In general, image management involves the following three steps:

1. *Store the digital images.*
2. *Extract features from the images.*
3. *Query and retrieve the images.*

The following figure shows the flow of these three steps.

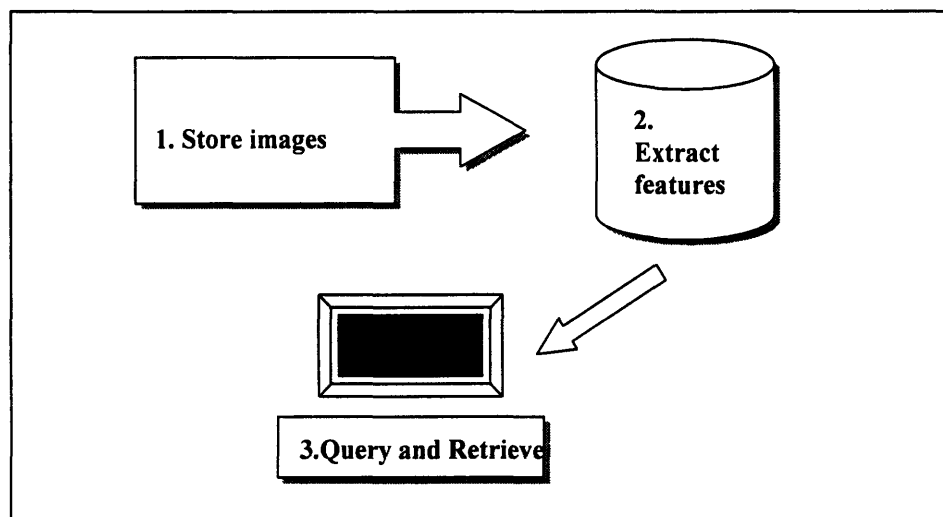


Figure 19. A flow chart illustrating the three general steps in designing an image database

Since the technology one chooses for image management systems depends on the intended use, there are many issues that one should consider before implementing an image management system. In this chapter, the issues involved in designing an image database are discussed and some recommendations are given as well.

5.1 Storing Digital Images

Before one can manage images digitally, one must store the digitized raw images. However, before choosing an image storage device, people should first determine what is the highest resolution of the digitized image that the intended application will need and how large the collection will be. Then the type of storage devices should be chosen based on the answer to these two questions. Once a type of storage device is selected, then one has to make further considerations such as how important is the read and write performance of the storage device for the application in mind.

There are three general categories for images: document images, graphics or photographs, and high-density pictures. Examples of document images are scanned articles, slides and transparencies. Graphics or photographs are exactly what they are, the typical graphics and photographs that are commonly seen on the web. On the other hand, high-density images are satellite, aerial or x-ray images where the resolution is fairly high. The file size for a digitized documents or photograph depends on what the scanner or digital camera's accuracy is. Typically, digital cameras and scanners produce outputs at 600 ppi (pixel per inch) with 24 bits per pixel. Therefore the typical file size of a scanned document or photograph is in the ranges of tens to hundreds of kilobits. However, x-ray pictures can be from 1436x1755 to 2048x2487 pixels per image at 16 byte per pixel producing a range from 5 to 10 Megabytes per picture.

There are three general storage technologies that should be considered when designing an image database². They are magnetic disk systems, optical disk systems and high performance magnetic tape systems.

Magnetic disk systems: they are random access devices such as regular hard disks and RAID arrays, storing data up to multiple GBs per drive with an access delay in the low 10's of ms.

Optical disk systems: devices such as optical jukebox and drives, storing multiple GBs of data per platter with an access time on the order of 20 to 100's ms.

High performance magnetic tape systems: devices include tape drives with automated robotic access systems. Each tape can store 10's of GBs of data with an access time on the order of seconds. Robotic tape libraries store up to several thousand tapes with an access time on the order of 10 seconds.

The following table summarizes the above three image storage technologies:

Type of Device	Access Delay	Capacity
Magnetic disk	10's ms	1 – 20 GBs
Optical disk	20 to 100's ms	20 – 1,000 GBs
Magnetic Tape	1,000s ms	10 – 1,000+ GBs

Table 3. Summary of the three types of digital image storage devices.

Most image database systems would probably be using the optical disk systems and optical technology is the most interesting area for image storage, therefore it is worth some investigation. There are several drives for optical storage systems: rewritable magneto-optical, magneto-optical WORM (write once, read many), recordable CD and regular CD-ROM. As mentioned earlier, which medium to choose from depends on the quantity of data and the requirement on read/write performance. The following is a table presenting the difference in read/write performance and access time for these four types of optical drives.

Drive Type	Maximum Capacity	Transfer Rate (MB per sec.)	Access Time (ms.)	Rewritable ?	Write Convenience
Magneto-Optical Rewritable	2,600	3.800	19 to 35	Yes	Logically function like a hard drive.
Magneto-Optical WORM	2,600	3.800	19 to 35	Write-once	Logically function like a hard drive.
CD-Recordable (4x speed)	650	0.612	250 to 300	Write-once	Premastering stage required. Limited incremental writing.
CD-ROM Player (6x Speed)	650	0.918	150 to 200	Read-only	Zero capability.

Table 4. 5.25" Magneto-Optical vs. CD-Recordable/CD-ROM.²

As the table shows, magneto-optical drives have a larger storage capacity and faster access time than the compact disk drives. It is quite ideal for image systems that require fast raw image retrieval and contain a relatively large collection of images. Therefore, if the image database one has in mind would use optical jukeboxes as storage medium, then magneto-optical media should be seriously considered even though it costs more than regular CD-ROMs. Magneto-optical media should not be overlooked for storage cost because in the long run, lower system productivity is going to cost more. Furthermore, magneto-optical technology also scales better than CD-ROMs. If one's image collection is expected to expand and grow in the future then the extra dollars are worth it..

An interesting technology that is new to image storage that is worth mentioning is the PhotoCD CD-ROM, a proprietary storage device by Eastman Kodak. The PhotoCD

Image Pac compression system enables a master PhotoCD to hold 100 images without losing image quality. Each image Pac has five resolutions, the largest of which is 2,048 pixels by 3,072 pixels. They also have a Pro PhotoCD which does a higher-quality scanning job and holds approximately 24 images, each at six resolutions with the largest resolution at 4,096 pixels by 6,144 pixels.

When designing an image database, one must pick a system that is ideal for the application in mind. One has to consider the file sizes of the images as well as the quantity of the image collection. Once the size of the collection is determined, one then has to consider what the reading and writing time requirements are for the intended use. For small collections of document images or photograph, a large magnetic hard disk or some PhotoCDs might be adequate. On the other hand, in places of hospitals or satellite observation stations, perhaps a storage hierarchy is required where all the images reside on magnetic tape systems with some recent pictures in jukeboxes and magnetic disks.

5.2 Feature Extraction

After having the raw images stored digitally, features must be extracted from the images in order to enable content queries. The decision one has to make in this step is determining what are the most relevant features that should be extracted. Feature extraction takes time and the effectiveness of the query outcomes relies heavily on what feature sets are used. As a result, designers of image databases usually invest quite an amount of time into feature selections.

There are five general features that most imaging systems extract for image content. They are colors, shape, texture, eigenvectors and spatial relationships. Besides these features, other operations may also be required to assist the accuracy and efficiency of these feature extraction procedures. For example, most images undergo segmentation and edge detection first before any feature characteristics are evaluated. Segmentation and edge detection separates the objects in the pictures from the background and thus put an effective boundary on the space where imaging procedures should be performed so the system can yield better or optimal results for feature extraction.

In addition to image content features such as colors, shape, texture, eigenvectors and spatial relationships, for best query result, text annotations should also be inserted for each of the images to further limit the search space. What text annotation should be made will also have to be determined by what the underlying goal for application is. For example, for face recognition systems, the name and biographical information of the photographed person would certainly be of use for retrieval and identification purposes.

It is very likely that an imaging system would extract several features for each image. The time for feature extraction can be quite long for a large collection of images but it usually means that the consequent query time will be shorter. The better and the more relevant the feature sets are, the faster the query and retrieval process would be. Therefore, it is worth taking the time to do good feature extractions to ensure better

performance at query time. The following table lists the average approximate computation time based on the number of pixels in the images for general feature extraction or pre-processing procedures.

Imaging Process	Computation Time	Imaging Process	Computation Time
Segmentation	$O(n \log n)$	Shape	$O(n^2)$
Edge detection	$O(n \log n)$	Texture	$O(n^2)$
Color analysis	$O(n)$	Eigenvectors	$O(n^2)$
Spatial relations	$O(n)$		

Table 5. Summary of the average computation time for some imaging processes.

The table reflects only the general methods for each of the imaging processes. More sophisticated algorithms of course can take longer or shorter computation time than the ones given above. The n in the table represents the number of pixels in the given image.

As stated before, what features are selected for extraction depends on the application. For example, colors are generally useful for most imaging systems, however, for x-ray picture, colors play no real importance on content retrieval (colors here mean the RGB values not the gray scale). Similarly, eigenvectors are usually not necessary for general-purpose content retrieval systems, but for face or fingerprint recognition systems, they are crucial. Another interesting observation that is worth mentioning is that even though the feature extraction process can take some time for each of the images, it is still much faster than making text annotation entries. Alphanumeric information associated with the images is usually entered manually. This manual entry process can take from seconds to minutes which is significantly slower by several order of magnitude than the automatic feature extraction process. Overall, this is another indication why image content retrieval systems is more superior than just plain text searching.

5.3 Query and Retrieval

Once the features have been extracted from the images, these features need to be stored to assist the retrieval process later. Usually people store this information in databases. There are three types of databases, relational, object-oriented, and object-relational.

Relational database is the most popular one today. It stores data in tables and has a structured way to search the targeted items by determining the “relations” between the stored items and the query request. Structured Query Language (SQL) is the relational database’s structured search method. Relational database is ideal for text based data, but it generally is not sophisticated enough for the complex image data and can not support complicated query methods. On the other hand, object-oriented database are better suited for complex image data since it mimics the way people handle images, as objects, not

entries in a table. It also has inheritance and more advanced programming structure that gives the image system developers the capability to create functions to tailor the query methods to meet their needs. However, often the imaging application the designer has in mind already has an existing image collection with textual annotations in a traditional relational database. If these textual annotations are still valid and can serve as an important criterion for image queries, object-relation databases will be better suited for these applications than object-oriented databases since these textual annotations can not be as easily transitioned to an object-oriented database. Therefore, object-relational database seems to be the best choice for storing the image features since it combines the strength of both relational and object-oriented databases.

There are three categories for image query and retrieval methods. The following table summarizes these three categories. The most primitive is the associated textual information query. It is no more than searching images based on alphanumeric information alone. Just as primitive is the visualization method, where it is just a display of multiple smaller versions of the stored images for the users to browse and choose. Then there is the feature based methods where image content searches are possible.

Search Category	Characteristics	Search Methods
Associated Textual Info.	Annotations, OCR, year, geographical location, name, etc.	Keywords, SQL.
Visualization Methods	Thumbnails, mosaic of multiple images.	Browsing, scrolling.
Feature Based Technique	Spatial relationships, color, shape, texture, and eigenvectors.	Distance measurement, statistical analysis, neural networks, graph matching, spatial query languages.

Table 6. Three image search method categories.

The search method category that is of the most interest to image database designers is obviously the feature based techniques. The techniques to be used for similarity comparison depend on the image features selected from step 2. For example, the texture of an image is usually represented as a feature vector. Comparing two images' texture means determining the similarity between the two corresponding texture feature vectors. The method used to determine this similarity is usually Euclidean distance measurements. This approach computes the distances between each of the corresponding elements in the two vectors and then returns the average of these element-wise distances as the texture similarity measure between the two images as a whole. On the other hand, image's color feature is usually represented as histograms. Similarity between histograms are better suited by using statistical analysis such as Probability Density Distribution (PDF) that can analyze the *distribution* of the features. Therefore, if distribution of a

feature is important in determining image feature's similarities, then statistical analysis will be a better method to use than distance measurements.

Sometimes it is important to add more intelligence into the query systems to carry out more complicated searches such as face or fingerprint pattern matching. In these cases, neural networks and graph matching techniques are more ideal since they can consider more heterogeneous factors simultaneously and be trained to make the desired match. The downside to these approaches is that training and finding the right combination of factors can take a very long time. Training a neural network is considered an art. It often takes months to train a neural network to output at an acceptable level of performance. Similarly, graph matching right now is a NP-complete problem, which means no one has been able to design a graph matching algorithm that has a guarantee performance in polynomial time.

For spatial relational type of queries, a query language with spatial constraints is ideal. There are several research projects that are examining this venture. The difficulty lies in confining the number of corresponding objects in the comparison images and determining the spatial relation between all of them. Determining the spatial relationship for each object with respect to all the other objects in the image grows exponentially with the number of objects. Likewise, for each object in one image, the system has the hard task of determining which object in the other image that this particular object should correspond with. To summarize what has been stated, the following table shows the feature based search methods' performances.

Methods	Computation Time	Remarks
Distance Measurement	$O(n)$	Good for feature vectors.
Statistical Analysis	$O(n)$	Good for feature histograms
Spatial Query Language	$O(n \log n)$	Finding the corresponding object between images is hard.
Neural Networks	Approx. $O(n^2)$	Selecting the right feature combinations and weights is hard. Training takes a long time.
Graph Matching	$O(n^2)$	Finding the corresponding nodes between graphs is difficult. It is a NP-complete problem.

Table 7. Summary of the feature based search methods. The n represents the number of elements in the feature space.

In conclusion, there are three main steps in designing an image database. One is deciding the storage device for the digitized images. Two is to select the relevant features to extract. Three is to determine which retrieval technique to use based on the selected feature sets. In deciding which storage device to use, one should first determine what is the largest file size the images in the collection can have and what is the expected quantity of the image collection. Then, one can chose from either magnetic disks, optical

disk, or magnetic tape storage technologies based on that size information and the wanted read/write requirements. For selecting image features for extraction, one should consider what features would be the most relevant and useful for the query process. For example, color and texture are not as relevant as eigenvectors for fingerprint applications. In the matching and query step, the methods chosen will again be dependent on the selected feature set. Overall, the most widely used methods for general purpose image databases are distance measurements such as near-neighbor algorithms, and statistical analysis such as PDFs. For more sophisticated imaging query process, neural networks are the chosen technique. In all three steps, the most influential factor in the decision making process is the same, what is the intended use for the application. Taking time on this designing process to really understand and evaluate the needs of the intended application is important since the success of the image system under development significantly depends on a well thought-out architecture. The table below summarizes the findings discussed in this chapter.

Storage Device	Capacity	I/O access time	Rewritable?
Magnetic Disk	1-20 GBs	10's ms	Yes
Optical Disk	20-1,000 GBs	20to 100's ms	Some
Magnetic Tape	10-1,000+ GBs	1,000s ms	Yes

Image Type	Spatial relations	Color	Shape	Texture	Eigenvector
Computation time	$O(p)$	$O(p)$	$O(p^2)$	$O(p^2)$	$O(p^2)$
Scanned Documents	No	No	Yes	No	No
Color photographs	Yes	Yes	Yes	Yes	No
Black & White photos.	Yes	No	Yes	Yes	No
X-rays	Maybe	No	Yes	Yes	No
Fingerprint or Face shots	Yes	No	Yes	No	Yes
Retrieval Method(s)	Spatial SQL	Distance Meas. Statistical Anly.	Distance Meas. Graph matching	Distance Meas. Statistical Anly.	Distance Meas. Graph matching Neural Network
Retrieval comput. time	$O(f)$	$O(f)$ $O(f)$	$O(f)$ $O(f)$	$O(f)$ $O(f)$	$O(f)$ $O(f)$ $O(f)$

Table 8. A Decision table that summarizes the design issues and findings. The p represents the number of pixels in the image and the f represents the number of features that are used.

Chapter 6 Case Study: QBIC

IBM's *Query By Image Content* or so called *QBIC* system is one of the only two existing commercial image databases. As a courtesy of IBM, the author was able to obtain an evaluation copy of this system. A retrieval performance analysis on this system was conducted using a large set of images that the author has individually selected. In this chapter, the testing process, the outcome of the evaluation and the conclusions reached from this experience are presented.

6.1 Testing Methodology

QBIC currently has executable binaries available for AIX 3.2.5 or 4.1, Linux Slackware 3.0 with gcc 2.7.7, OS/2 Warp, Windows NT 3.51 / 4.0, and Windows 95. QBIC also requires an HTTP server for the web user interface that they have developed. The QBIC system is installed on to a Pentium 120 machine with 16 megabytes of RAM running Windows 95. The HTTP server used is also from IBM called Internet Connection Server (ICS). In general, the installation process was straight forward although it required running DOS commands unlike the push-button one step installation process that most PC users are used to.

For the test image collection, 460 JPEG images has been gathered of sizes that range from 30 to 80 kilobytes. Each of the 460 JPEG images falls into only one of the 46 pre-chosen categories. Therefore, there are 46 groups of images, each group has 10 images in it. These groupings or categories represent what a user may want to find in the image database. The idea is to evaluate how well the search methods that QBIC has provided serve the needs of the user to retrieve the images for each category. The categories were chosen to be as different as possible and the images in each groups were chosen to be as similar as possible. The following lists some of the categories used.

<i>Arches</i>	<i>People's Face</i>	<i>Brown Bears</i>	<i>Butterflies</i>
<i>Cactus</i>	<i>Cats</i>	<i>Red cars</i>	<i>Coins</i>
<i>Red flowers</i>	<i>Houses</i>	<i>Aircrafts</i>	<i>Medals</i>
<i>Rocks</i>	<i>Mountains</i>	<i>Oceans</i>	<i>Space views</i>

After an image collection is made, QBIC creates a catalog for the collection for indexing and retrieval using the *QBCreateCat* method. Running *QBCreateCat* from DOS, it took a total of 46 minutes to catalog the collection of 460 images. About the first 12 minutes were used to create thumbnail images for each of the pictures in the collection. The rest was used to create catalogs for each individual picture. It comes to about 2 seconds to create one thumbnail image and 4.5 seconds to catalog one picture.

After cataloging the images, the system is ready for image content queries on these images. The picture below shows QBIC's web user interface. As one can see, QBIC has a button labeled "*Random*" which randomly selects pictures from the collection for

display. It also has a pull-down menu in the center that displays three available image content search methods: *color percentage*, *color layout* and *texture*. To do a content search, first select one of the three search methods and then click on one of the displayed images. *Color percentage* retrieves images based on their color histograms. For example, if the selected example image has 10% white, 15% blue and 20% yellow, then QBIC will try to retrieve other pictures that have a similar color composition. For *color layout*, if the sample image has the top half of the picture mostly blue and the bottom half of the picture mostly brown, then it will retrieve other pictures that have the similar top blue and bottom brown color layout. As for *texture*, it uses the example image's texture to compare with all the other images and returns what it considers to be the closest texture matches.

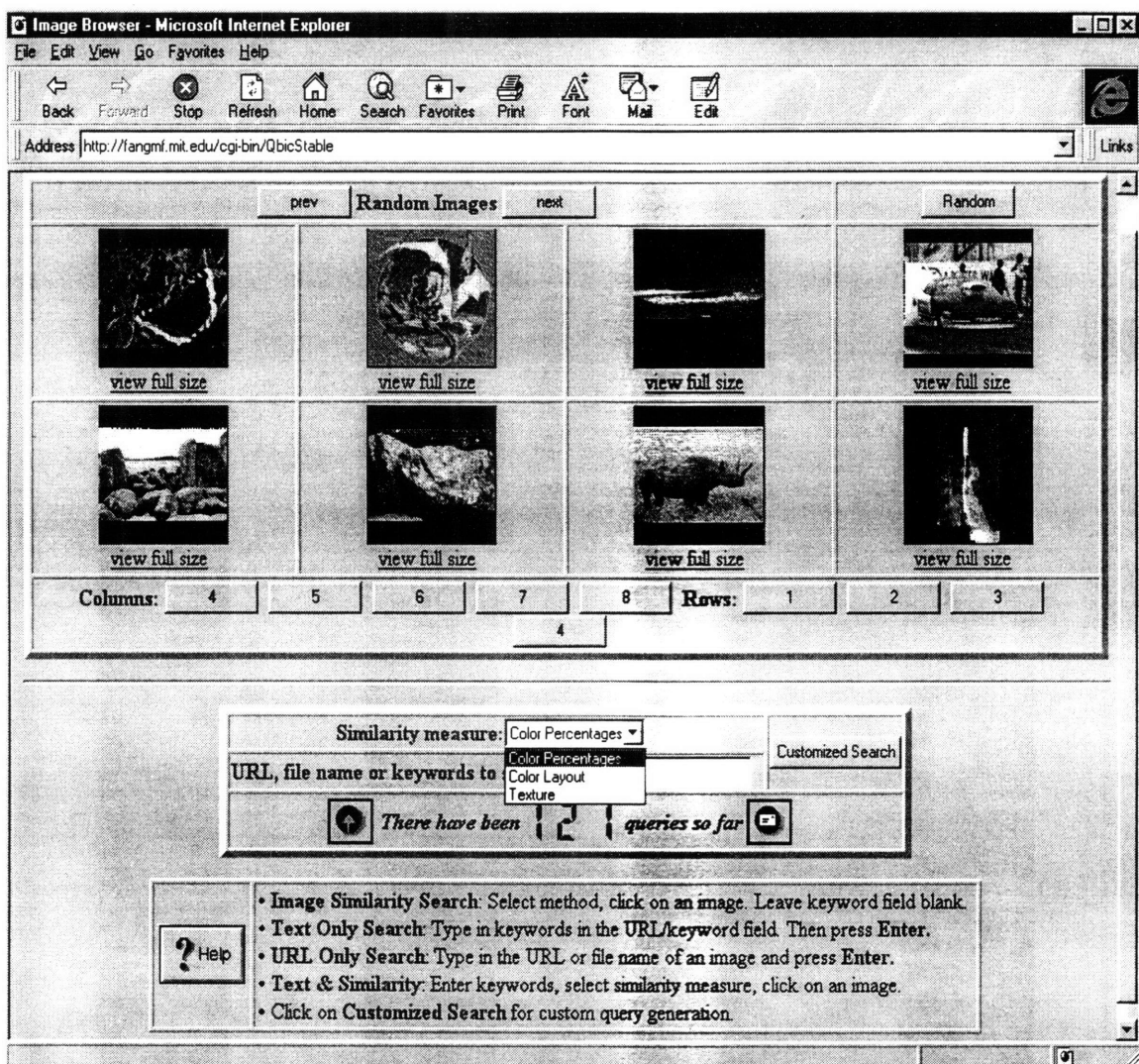


Figure 20. An image of the QBIC Web User Interface.

The evaluation was conducted in the following manner. One image from each of the 46 groups is chosen to be the token example image for that group. Then each of the three search methods were performed using those token images as examples. Then the number of relevant group members that were retrieved as well as the total number of images that were retrieved are recorded. The results are discussed in the next section.

6.2 Test Results

To test this content retrieval analysis, the recall and precision of the conducted queries are measured. Recall is the proportion of relevant material retrieved, while precision quantifies the proportion of the retrieved materials that are relevant to the search. Relevance is measured by seeing if the returned image belongs to the same group as the example image or not. The following table presents the average recall and precision for each of the three QBIC search methods.

SEARCH METHOD	RECALL	PRECISION
Color Percentage	32%	16%
Color Layout	38%	19%
Texture	24%	12%

Table 9. Recall and Precision for the test results.

The outcome is not as satisfactory as have been expected. The obvious reason why is that the content searches were based on only one image feature. However, the results seem to indicate that color layout is a better image retrieval criterion than color percentage and texture. Furthermore, they also indicate that using texture alone for image retrieval is not ideal.

There is one other observation that the author has made. For certain types of images, color percentage is the best retrieval method and vice versa for the other two as well. For example, for categories where the images contains a variety of colors such as butterflies or medals, color percentage search method does significantly better than the other two methods. Figure 20 shows the returned results of using color percentage on an example medal image. As for color layout, it is best for pictures that have very distinct color boundaries such as pictures of mountains with the sky showing, and ocean pictures. Figure 21 shows the returned images for an example ocean image using color layout. Similarly, texture is best for close-up pictures of rocks and wood where the texture is relatively uniform through out the picture. Figure 21 shows the returned results on a example wood image using the texture retrieval method.

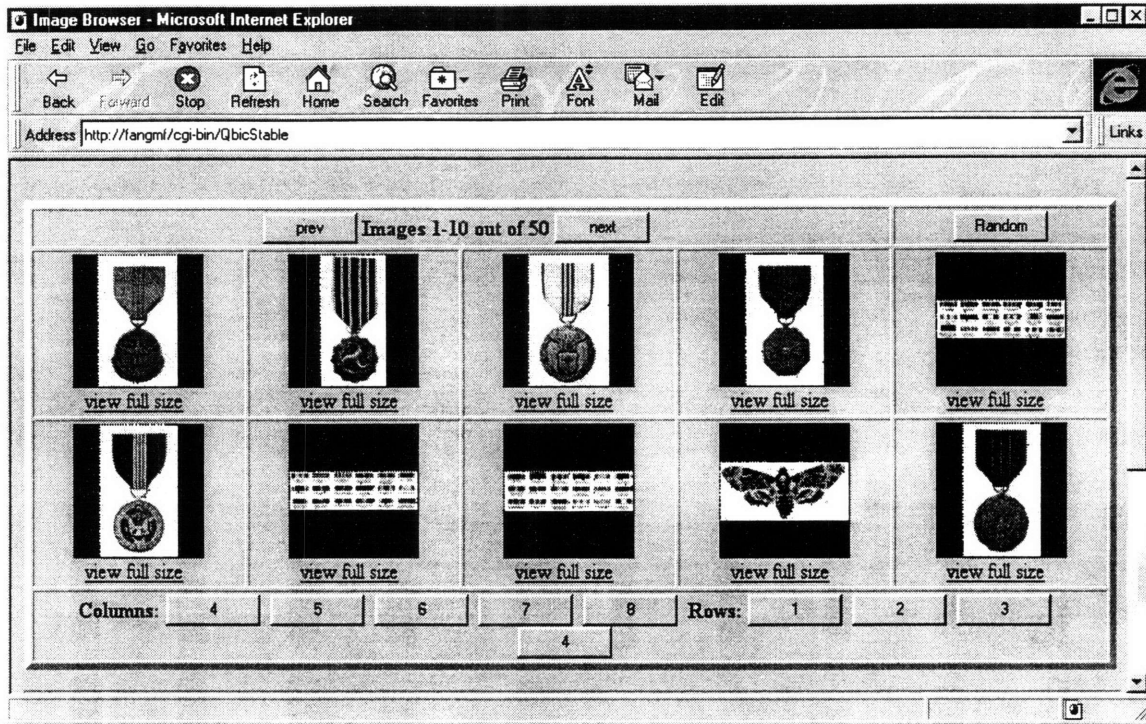


Figure 21. The returned results of example medal image in the upper left corner using color percentage.

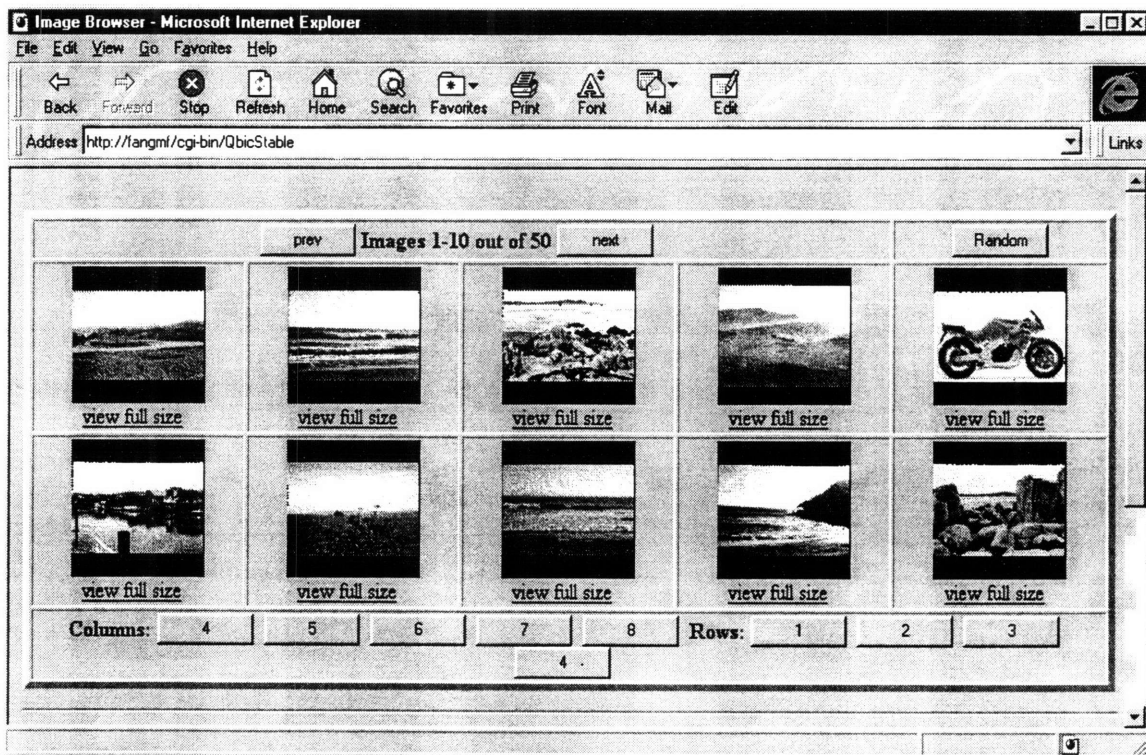


Figure 22. The returned results of example ocean image in the upper left corner using color layout.

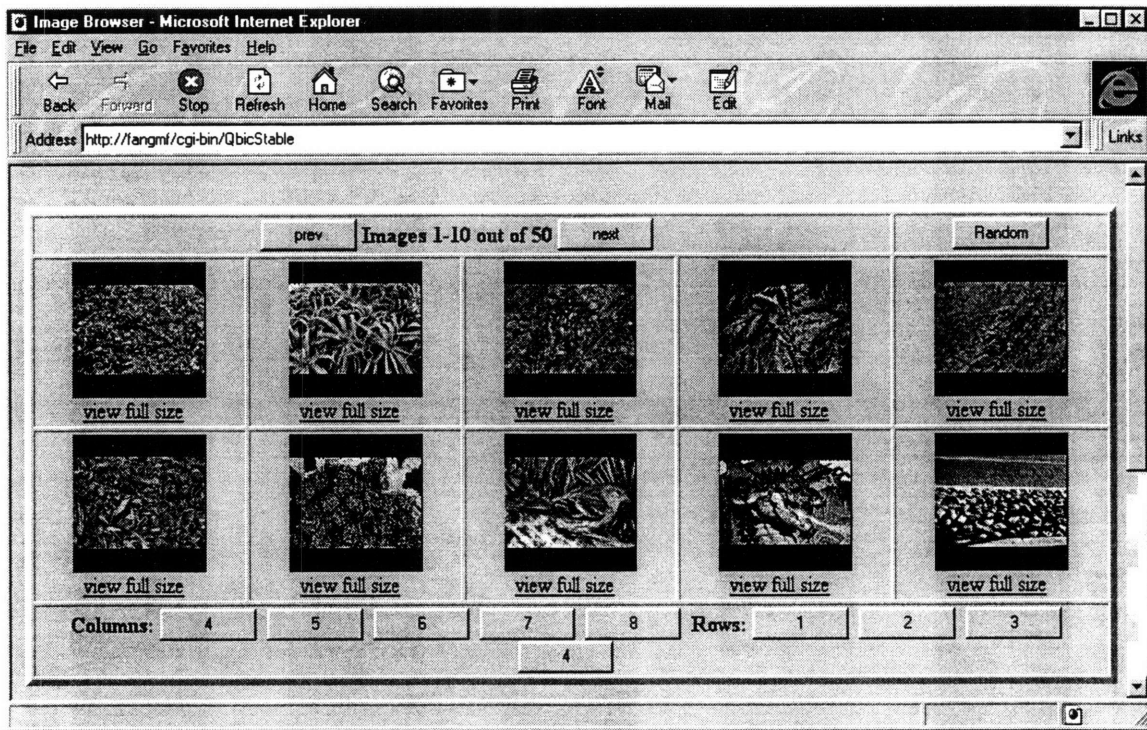


Figure 23. The returned result of an example wood image in the upper left corner using texture.

From the outcome of the testing, all three search methods are concluded to be quite well implemented by IBM. This observation is made by noticing that even though the images returned are mostly not relevant, one can still see the similarities in either the color percentage, color layout or texture. If somehow these methods can be combined, the retrieval accuracy will certainly increase.

6.3 Evaluation Conclusions

QBIC is representative of the state-of-the-art technology in commercial image databases. From this evaluation experience, several conclusions have been made. One is that although IBM, as one of the leaders in the area of image databases, has made many progresses in this area, the current technology is still inadequate in serving the users needs in content-based image searches.

Another is that there are three recommendations to be made to the QBIC developers. First is that if QBIC is to market this application for personal computers, the installation and cataloging process should not be DOS base procedures. Most PC users are used to the touch and feel of Window based applications. It would be a short and worthwhile investment to making a Window based installation and cataloging interface. Second is that there seems to be no provided means to added, update, or delete images in a collection that has been cataloged already. It would be undesirable for user to re-catalog everything each time the collection has changed. Most image database applications would

acquire some capability in adding, updating, and deleting images. After all, it is suppose to be a database. Third, this recommendation concerns with the search methods. The overall recall and precision rate based on my evaluation is rather low, but as stated before, each search method is well implemented. Hence if IBM can somehow combine the three methods to perform a single search based on some weighted scale, the results would be surely be much better. This seems like a trivial step to implement and should not take long to accomplish such task.

Finally, image retrieval based purely on image features may not be the best way to build an image database. Although inserting text annotations for images may be tedious and burdensome, textual information certainly has very relevant and important value. There must exist a combination of textual information and feature based query that can yield an optimal search outcome. It would be rewarding for image database designers to proceed in this direction.

Chapter 7 Conclusion

Image database is currently a very exciting research field. In this thesis, the fundamentals in the area of pattern recognition and image analysis such as image segmentation, edge detection, shape recognition, texture determination, etc were examined. The newest developments in both academic and commercial research efforts in content based image retrieval have also been investigated. Although the raw image data might be different, such as scanned documents, satellite pictures and photographs, in general, all these academic and research image systems conform to the following paradigm. Image features are pre-computed during the insertion phase into the system. These features represent the images and they are typically characteristics such as color histograms, edges, object shapes and texture. These features are then stored in a database to be used later for comparison purposes. Content-based query are then generally done by finding images that are feature-wise similar to a given example image. Users either select this example image from the image collection or construct one using some graphic tool. The system then computes a similarity score, relative to the example image, for each image in the database and returns the results in some ranked order. There are various variations of this general scheme, such as allowing users to specify weights on the feature search criteria, or specify “concepts” like in the Chabot project.

Although this general scheme described above seems to be adequate for these research and commercial efforts, there are several areas that the current image database projects are lacking. For example, even the most sophisticated query methods still require large amount of effort from the user. These query methods combine several feature-based comparisons to produce an overall similarity measure score based on some weights that are generally specified by the users. Only after numerous trials and errors can the users find the right weights for the desired images. Furthermore, users have to repeat this strenuous weights-correcting process for each different query on images since it is unlikely that one set of weights can yield optimal results for all possible image queries a user would have. Some research effort is needed for automating this weights-correcting process.

Likewise, metadata is another area that additional considerations should be given by the researchers. Metadata can make interoperability between imaging systems possible. Many benefits can come with having a metadata standard for image databases. The current challenge lies in determining what information should be included in an image metadata standard. An agreement on what image features should be included is hard to come by since many imaging systems are tailored for specific applications. However, the author believes that the researchers should still be able to have the authoring information that is common to all images, such as copyright information and the file format, be standardized.

Image database development is just as exciting as it is challenging as a research field. Many things have to be carefully considered before proceeding on to the implementation phase. In this thesis, three general steps in designing an image database, *storing the images, feature extraction, query and retrieval* are discussed. The decisions one makes in

each of these three steps all depend on the intended application and are driven by a number of constraints such as computation time versus accuracy.

An evaluation on a state-of-the-art commercial image database is also presented in this thesis. From this evaluation experience, it was evident that much progress still needs to be made in the area of image databases. There are many challenges that still await the research community to face. However, as the society evolves more and more toward the digital age, the author is confident that people will one day search images just as easily as they search for text publications now!

Future research efforts should focus on two main areas: image database's user friendliness and interoperability between image databases. Currently, the users have the burden of choosing parameters and weights for image content retrieval. Algorithms should be developed for automating this process. New image features should also be postulated to pursue the possibility of more effective image retrieval methods. Similarly, the combination of alphanumeric description and feature-based content processing is another area that is currently overlooked. Neither word portrayal nor image features alone can successfully capture the essence of images that users often look for. Additional studies into combining text phrases and image content for image search engines seems like a logical next step. As for interoperability between image databases, standards are needed. Standardizing metadata for images should be possible since many authoring information are common to all types of images regardless of application. Furthermore, open frameworks for image database development are also necessary for cooperation and sharing of pictorial data. Thus, further research efforts should be invested in making metadata standards as well as image database development frameworks.

References

1. Ackerman, L. and Gose, E.E. "Classification of xeromammogram lesion by computer", *Cancer*, 30 (1972), 1025-1035. 18
2. Batoy, A., "8 things to consider when considering document image storage & retrieval product options," *Advanced Imaging* v.11 n 3 March 1996, pages 26-29. 74, 75
3. Bunk and Messmer, "Efficient Attributed Graph Matching and Its Application to Image Analysis" in *Image Analysis and Processing, Lecture Notes in Computer Sciences*. 1995 24
4. Goodchild, M., "Report on a Workshop on Metadata held in Santa Barbara, California," University of California, Santa Barbara, CA, November 8, 1995. 70
5. Gose, Johnsonbaugh, and Jost, *Pattern Recognition and Image Analysis*, page 199-227 13
6. Gose, Johnsonbaugh, and Jost, *Pattern Recognition and Image Analysis*, page 200-209 13
7. Gose, Johnsonbaugh, and Jost, *Pattern Recognition and Image Analysis*, page 201-205 14, 15
8. Gose, Johnsonbaugh, and Jost, *Pattern Recognition and Image Analysis*, page 210-223. 15
9. Gose, Johnsonbaugh, and Jost, *Pattern Recognition and Image Analysis*, page 213-215. 16
10. Gose, Johnsonbaugh, and Jost, *Pattern Recognition and Image Analysis*, page 215-218. 17
11. Gose, Johnsonbaugh, and Jost, *Pattern Recognition and Image Analysis*, page 227-231 25, 26
12. Gose, Johnsonbaugh, and Jost, *Pattern Recognition and Image Analysis*, page 284. 23
13. Gose, Johnsonbaugh, and Jost, *Pattern Recognition and Image Analysis*, page 303-309. 11
14. Gose, Johnsonbaugh, and Jost, *Pattern Recognition and Image Analysis*, page 311-213 16
15. Gose, Johnsonbaugh, and Jost, *Pattern Recognition and Image Analysis*, page 3249-254. 27
16. Gose, Johnsonbaugh, and Jost, *Pattern Recognition and Image Analysis*, page 329-338. 9, 23
17. Gose, Johnsonbaugh, and Jost, *Pattern Recognition and Image Analysis*, page 352-357. 12
18. Gose, Johnsonbaugh, and Jost, *Pattern Recognition and Image Analysis*, page 362-265. 17
19. Gose, Johnsonbaugh, and Jost, *Pattern Recognition and Image Analysis*, page 372-379. 18
20. Gose, Johnsonbaugh, and Jost, *Pattern Recognition and Image Analysis*, page 374. 20
21. Gose, Johnsonbaugh, and Jost, *Pattern Recognition and Image Analysis*, page 379-388. 22
22. Grimson, W.E.L., *Object recognition by Computer: the role of Geometric Constraints*, MIT Press, Cambridge, MA, 47-76, pp. 447-454, 1990 32, 33, 34, 35

23. Haralick, R.M. "Statistical and structural approaches to texture", Proceedings of the IEEE, 67 (1979), 786-804. 22
24. Jain, R. etc, Virage Inc., "Virage image search engine: an open framework for image management," in Storage and retrieval for Still Image and Video Databases IV, I. Sethi, R. Jain, Editors, Proc. SPIE 2670, page 76-88. 64
25. Khoshafian, S.and Baker A.B. Multimedia and Imaging Databases 5, 7
26. Kopec, G, "Document Image Decoding in the Berkeley Digital Library", in Proceedings IICIP-96 vol. 2, IEEE International Conference on Image Processing. 1996 29
27. M. O'Docherty, A Multimedia Information System with Automatic Content Retrieval, University of Manchester, Manchester, UK. 1993. 36
28. Manjunath, B.S. and Ma, W.Y, "Browsing Large Satellite and Aerial Photographs", Department of EECE, University of California, Santa Barbara, CA 93106 42
29. MIT Media Laboratory Vision and Modeling Group, Face Recognition Demo Page, <http://www-white.media.mit.edu/vismod/demos/facerec/index.html>. 48
30. Niblack, W. etc. IBM Almaden Research Center, "Automatic and semiautomatic methods for image annotation and retrieval in QBIC, " in Storage and retrieval for Image and Video Databases III, W. Niblack, R. Jain, Editors, Proc. SPIE 2420, page 24-35. 57
31. Oakley, Davis, Shann and Xydeas. "Evaluation of an application-independent image information system", in Storage and Retrieval for Image and Video Databases II, Wayne Niblack, Ramesh C. Jain, Editors, Proc. SPIE 2185, pages107-118 (1994). 38
32. Ogle, V.E.and Stonebraker, M., "Chabot: retrieval from a Relational Database of Images," University of California, Berkeley. 50, 51, 56
33. Patrick, R and Reimer, C.W., "The diatoms of the United States, exclusive of Alaska and Hawaii", vol. 1, 1996, vol. 2, 1975, Academy of Natural Sciences. 18
34. Pentland, Picard and Sclaroff, "Photobook: tools for content-based manipulation of image databases," in Information Systems: Applications and Opportunities, Peter J. Costianes, Editor, Proc. SPIE 2368, pages 37-50 (1995). 46
35. Phelps, T and Wilensky, R, "Toward Active, extensible, Networked Documents: Multivalent Architecture and Application", in Proceedings of ACM Digital Libraries '96, Bethesda, Maryland. 1996. 30, 31
36. Winston, P.H., Artificial Intelligence, Addison-Wesley Publishing Company, 1993, 1993, Pages 448-453. 27