



This is to certify that the  
thesis entitled

OBJECT SHAPE AND STRUCTURE FOR IMAGE  
MATCHING AND RETRIEVAL

presented by


NAVEED SARFRAZ KHAN KHATTAK

has been accepted towards fulfillment  
of the requirements for the

Master of  
Science

degree in

Computer Science



Major Professor's Signature

13 Dec 2002

Date

LIBRARY

Michigan State University

PLACE IN RETURN BOX to remove this checkout from your record.  
TO AVOID FINES return on or before date due.  
MAY BE RECALLED with earlier due date if requested.

DATE DUE	DATE DUE	DATE DUE

**OBJECT SHAPE AND STRUCTURE FOR IMAGE MATCHING  
AND RETRIEVAL**

**By**

**Naveed Sarfraz Khan Khattak**

**A THESIS**

**Submitted to  
Michigan State University  
in partial fulfilment of the requirements  
for the degree of**

**MASTER OF SCIENCE**

**Department of Computer Science**

**2002**



ABSTRACT

OBJECT SHAPE AND STRUCTURE FOR IMAGE MATCHING AND  
RETRIEVAL

By  
NAVEED SARFRAZ KHAN KHATTAK

This thesis reports results on the use of shape features for content-based image retrieval. Edges, lines, corners, and straight ribbons are extracted from a query image. These features and their attributes, along with various relations are used to match representations of other images in a database via a graph-matching algorithm. Images can be matched to images or to hand-drawn sketches.

Experimental results on images with man made structures and aerial images are promising. Images can be matched to images of hand drawn sketches. In addition to the content-based retrieval, the matching method is also applicable to object recognition and localization. The algorithm is also suitable for estimation of scene changes in a continuous movie. Presently, the algorithm is developed in MATLAB. The current program must be translated to 'C/C++' in order to run much faster and should be tested on a much larger set of images.

Copyright by  
NAVEED SARFRAZ KHAN KHATTAK  
2002

## ACKNOWLEDGMENTS

I would like to express my gratitude to all those who gave me the possibility to complete this thesis. I want to thank National University of Science and Technology for their financial support and for encouraging me to complete my thesis.

I would also like to express my special thanks to my supervisor Dr. George Stockman for his advice and guidance during this study and to my committee members Dr Sakti Pramanik and Dr Jiajuo Qi.

I would like to give my special thanks to my parents, wife and children whose patient love enabled me to complete my thesis.

I am also thankful to all those who helped me to complete my thesis, especially Irafan Asalam and Khurram Waheed.

## TABLE OF CONTENTS

List of Tables .....	ix
List of Figures .....	x
<b>1. INTRODUCTION.....</b>	<b>1</b>
<b>2. LITERATURE REVIEW .....</b>	<b>6</b>
<b>2.1 Current CBIR Techniques .....</b>	<b>6</b>
<b>2.1.1 Color Based Retrieval .....</b>	<b>6</b>
<b>2.1.2 Texture Based Retrieval .....</b>	<b>7</b>
<b>2.1.3 Shape Based Retrieval .....</b>	<b>7</b>
<b>2.1.4 Retrieval by other Types of Primitive Feature .....</b>	<b>8</b>
<b>2.2 Practical Applications of CBIR.....</b>	<b>11</b>
<b>3. METHODOLOGY.....</b>	<b>12</b>
<b>3.1 Data Representation.....</b>	<b>12</b>
<b>3.2 Image Matching Process.....</b>	<b>14</b>
<b>4. FEATURE EXTRACTION MODULE.....</b>	<b>16</b>
<b>4.1 Edge Detection .....</b>	<b>16</b>
<b>4.1.1 Canny Edge Detector .....</b>	<b>18</b>
<b>4.1.2 Experiment and Results with the Canny Edge Detector .....</b>	<b>19</b>
<b>4.2 Line Detection .....</b>	<b>25</b>

4.2.1	Line Joining .....	34
4.2.2	Line Detection Results .....	35
4.3	Corner Detection .....	38
4.3.1	Corner Detection Observations.....	40
4.3.2	Corner Detection Results.....	42
4.4	Ribbon Detection .....	43
4.4.1	Ribbon Detection .....	44
4.4.2	Type of Ribbons.....	44
4.4.3	Ribbon Detection Results.....	45
5.	IMAGE REPRESENTATIONS .....	49
5.1	Basic Definitions .....	50
5.1.1	Adjacency Matrix.....	51
5.2	Line Structure Representation.....	54
5.2.1	Line Slope.....	55
5.2.2	Edge Gradient.....	55
5.2.3	Distance between Line Segments. ....	57
5.2.4	Relative Line Orientation. ....	57
5.2.5	Behavior of Line Descriptors and Relationships.....	58
5.3	Corner Structure Representation.....	59
5.3.1	Angle of a Corner. ....	60

5.3.3	Corner's Orientation.....	61
5.3.4	Corner's Precincts.....	62
5.3.5	Angle between Corners.....	64
5.3.6	Behavior of Corner Descriptors and Relationships.....	65
5.4	Ribbon Structure Representation.....	65
5.4.1	Width of a Ribbon.....	67
5.4.2	Types of a Ribbon.....	67
5.4.3	Relative Distance between Ribbons.....	68
5.4.4	Ribbon's Orientation.....	69
5.4.5	Behavior of Ribbon Descriptors and Relationships.....	70
6.	GRAPH MATCHING ALGORITHM.....	71
6.1	Graph Matching.....	71
6.2	Proposed Graph Matching Algorithm.....	72
6.3	Line-Graph Matching.....	73
6.3.1	Matching Algorithm.....	73
6.3.2	Matching Results for Line Segments.....	78
6.4	Corner-Graph Matching.....	84
6.4.1	Matching Algorithm.....	84
6.4.2	Matching Results for Corners.....	87
6.5	Ribbon-Graph Matching.....	92

<b>6.5.1</b>	<b>Matching Algorithm .....</b>	<b>92</b>
<b>6.5.2</b>	<b>Matching Results for Ribbons .....</b>	<b>95</b>
<b>6.6</b>	<b>Combined Results of Graph Matching Algorithm .....</b>	<b>100</b>
<b>7.</b>	<b>DISCUSSION AND FUTURE WORK .....</b>	<b>114</b>
<b>7.1</b>	<b>Analysis .....</b>	<b>114</b>
<b>7.2</b>	<b>Conclusion .....</b>	<b>117</b>
<b>7.3</b>	<b>Future Work.....</b>	<b>118</b>
	<b>APPENDEX-A.....</b>	<b>119</b>
	<b>APPENDEX-B.....</b>	<b>126</b>
	<b>BIBLIOGRAPHY.....</b>	<b>130</b>

## LIST OF TABLES

<b>Table 3-1 Features and their Parameters .....</b>	<b>15</b>
<b>Table 4-1 Confidence Rating for corner detection of hand-drawn images.....</b>	<b>43</b>
<b>Table 4-2 Confidence Rating for ribbon detection.....</b>	<b>46</b>
<b>Table 5-1 Line descriptors.....</b>	<b>57</b>
<b>Table 5-2 Behaviour of Line Descriptors and Relationships .....</b>	<b>59</b>
<b>Table 5-3 Decision table for arm's orientation of a corner .....</b>	<b>63</b>
<b>Table 5-4 Type of bounds corner is making .....</b>	<b>63</b>
<b>Table 5-5 Behavior of Corner Descriptors and Relationships .....</b>	<b>65</b>
<b>Table 5-6 Behavior of Ribbon Descriptors and Relationships.....</b>	<b>70</b>
<b>Table 6-1 Different threshold used in Line Structure Graph Algorithms.....</b>	<b>78</b>
<b>Table 6-2 Different threshold used in Corner Structure Graph Algorithms .....</b>	<b>87</b>
<b>Table 6-3 Different thresholds used in Ribbon Structure Graph Algorithms.....</b>	<b>92</b>
<b>Table 7-1 Results for exact matching .....</b>	<b>115</b>
<b>Table 7-2 Results for similar matching .....</b>	<b>116</b>
<b>Table 7-3 Graph sizes (number of vertices) for different data-structures and images .....</b>	<b>116</b>
<b>Table 7-4 Run-time for image retrieval.....</b>	<b>117</b>



## LIST OF FIGURES

**“Images in this thesis are presented in color”**

<b>Figure 1-1 Query image of Small Building and results of our retrieval system.....</b>	<b>4</b>
<b>Figure 3-1 Overall Methodology for Image Comparison.....</b>	<b>14</b>
<b>Figure 4-1 Process for feature extraction .....</b>	<b>16</b>
<b>Figure 4-2 Edge image of different shapes with default parameter <math>\sigma=1.0</math> .....</b>	<b>19</b>
<b>Figure 4-3 Edge image of different shapes with <math>\sigma=2.0</math> .....</b>	<b>20</b>
<b>Figure 4-4 (a) Test Image with Gaussian noise of 0.01. (b)Edge image with <math>\sigma=1.0</math>..</b>	<b>21</b>
<b>Figure 4-5 (a) Test Image with Gaussian noise of 0.02. (b) Edge image with <math>\sigma=1.0</math>.</b>	<b>21</b>
<b>Figure 4-6 (a) Test Image with Gaussian noise of 0.05. (b) Edge image with <math>\sigma=1.0</math>.</b>	<b>22</b>
<b>Figure 4-7 Edge image with <math>\sigma=3.0</math>.....</b>	<b>22</b>
<b>Figure 4-8 (a) Thick W with original scanner noise. (b) Edge image of Thick W with <math>\sigma=1.0</math>.....</b>	<b>23</b>
<b>Figure 4-9 (a) Taj Mahal with original scanner noise. (b) Edge image of Taj Mahal with <math>\sigma =1.0</math> .....</b>	<b>23</b>
<b>Figure 4-10 (a) Small Building with original scanner noise. (b) Edge image with <math>\sigma=1.0</math>.....</b>	<b>24</b>
<b>Figure 4-11 The parameters <math>d</math> and <math>\theta</math> used in the equation 4.2 of a straight line.....</b>	<b>27</b>
<b>Figure 4-12 Accumulator Array showing the peaks .....</b>	<b>29</b>
<b>Figure 4-13 Test image .....</b>	<b>29</b>
<b>Figure 4-14 Accumulator Array showing the merged peaks .....</b>	<b>32</b>

<b>Figure 4-15 (a) line detection before applying algorithm 4.1 and (b) line detection after applying algorithm 4.1 .....</b>	<b>33</b>
<b>Figure 4-16 (a) Vertical overlapping edge (b) Diagonal overlapping edge .....</b>	<b>34</b>
<b>Figure 4-17 (a) and (b) Successful line detection.....</b>	<b>36</b>
<b>Figure 4-18 (a) and (b) Incomplete line detection .....</b>	<b>37</b>
<b>Figure 4-19 (a) Lincoln Memorial and (b) Big Taj, line length = 5 .....</b>	<b>37</b>
<b>Figure 4-20 (a) Nairobi 1 and (b) Small Taj, line length = 5 .....</b>	<b>38</b>
<b>Figure 4-21 Test image with all the corners detected .....</b>	<b>40</b>
<b>Figure 4-22 Test Pattern with one corner missing (lower left corner of the lower middle rectangle) .....</b>	<b>41</b>
<b>Figure 4-23 Corner detection results for (a) Lincoln 25 corners and (b) Taj Mahal 60 corners .....</b>	<b>41</b>
<b>Figure 4-24 (a) is Type-1 ribbon and (b) is Type-2 ribbon.....</b>	<b>45</b>
<b>Figure 4-25 Thin W ribbon detection.....</b>	<b>46</b>
<b>Figure 4-26 (a), (b) and (c) Shows the detected ribbons .....</b>	<b>47</b>
<b>Figure 4-27 Nairobi 2 showing some successful ribbon detection.....</b>	<b>48</b>
<b>Figure 4-28 Showing some successful ribbons of Nairobi 2.....</b>	<b>48</b>
<b>Figure 5-1 Pictorial representation of a graph <math>G(V,E)</math>.....</b>	<b>52</b>
<b>Figure 5-2 Adjacency Matrix of Figure 5-1 .....</b>	<b>53</b>
<b>Figure 5-3 Adjacency Matrix of Figure 5-1 .....</b>	<b>53</b>
<b>Figure 5-4 Pictorial representation of line structure. ....</b>	<b>54</b>
<b>Figure 5-5 (a) Test image, (b) Lines extracted from (a).....</b>	<b>56</b>

<b>Figure 5-6 Line Orientation .....</b>	<b>58</b>
<b>Figure 5-7 Pictorial Representation of Corner Structure .....</b>	<b>59</b>
<b>Figure 5-8 Corners detected by the corner detection algorithm.....</b>	<b>60</b>
<b>Figure 5-9 Corner Orientation.....</b>	<b>61</b>
<b>Figure 5-10 Example of corner bounds.....</b>	<b>62</b>
<b>Figure 5-11 Pictorial Representation of Ribbon Structure .....</b>	<b>66</b>
<b>Figure 5-12 Ribbon Width.....</b>	<b>67</b>
<b>Figure 5-13 Distance between ribbon A and ribbon B .....</b>	<b>68</b>
<b>Figure 5-14 Ribbon Orientation.....</b>	<b>69</b>
<b>Figure 6-1 Query image Drawn Taj and retrieved images.....</b>	<b>80</b>
<b>Figure 6-2 Query image Taj Mahal and retrieved images .....</b>	<b>81</b>
<b>Figure 6-3 Query image Small Building and retrieved images .....</b>	<b>82</b>
<b>Figure 6-4 Query image Spartan Village Apartments and retrieved images .....</b>	<b>83</b>
<b>Figure 6-5 Query image drawn Taj and retrieved images .....</b>	<b>88</b>
<b>Figure 6-6 Query image Taj Mahal and retrieved images .....</b>	<b>89</b>
<b>Figure 6-7 Query image Small Building and retrieved images .....</b>	<b>90</b>
<b>Figure 6-8 Query image Spartan Village Apartments and retrieved images .....</b>	<b>91</b>
<b>Figure 6-9 Query image drawn Taj and retrieved images .....</b>	<b>96</b>
<b>Figure 6-10 Query image Taj Mahal and retrieved images .....</b>	<b>97</b>
<b>Figure 6-11 Query image Small Building and retrieved images.....</b>	<b>98</b>

<b>Figure 6-12 Query image Spartan Village Apartments and retrieved images .....</b>	<b>99</b>
<b>Figure 6-13 Query image drawn Taj and retrieved images .....</b>	<b>101</b>
<b>Figure 6-14 Query image Taj Mahal and retrieved images .....</b>	<b>102</b>
<b>Figure 6-15 Query image Small Building and retrieved images.....</b>	<b>103</b>
<b>Figure 6-16 Query image Spartan Village Apartments and retrieved images .....</b>	<b>104</b>

## **1. Introduction**

An important problem in the field of computer vision is the automatic recognition of objects in two-dimensional images for the classification of scenes. Another problem is that of matching images based upon similarities between object features. Because of the recent increase in the wide range of applications depending upon the resolution of such problems, such as multimedia database searches and augmented reality, the solutions to these problems should have a high degree of flexibility and automation. Matching images in a multimedia database offers an example of such a problem. The database may contain hand-drawn images, images taken from cameras and photographs containing objects with varying geometric properties, colors, and textures, as well as a wide range in image resolution and size. The extraction and subsequent classification of object features must be general enough to deal with the varied contents of the database and must ensure that reliable matching takes place. At the same time, the system should strive for automation so that user interaction is kept to a minimum. The shape of a single object and the various spatial constraints among multiple objects in an image are examples of object classification. Shape and spatial constraints are important data in many applications, ranging from complex space exploration and satellite information management to medical research and entertainment. Image retrieval can be categorized into exact match searching and similarity based searching. For either type of retrieval, the dynamic aspects of image content require expensive computations and sophisticated methodologies in the areas of image processing and database systems. In order to overcome these problems, several schemes for data modeling and image representation have been proposed [1,2]. In general, each of these schemes builds a symbolic image for

each given physical image, and symbolic images are then used in conjunction with index structures as proxies for image comparisons to reduce the search space. One of the traditional indexing methods for image retrieval is text-based. Although text annotation is a practical technique, this task is labor intensive, language dependent, vocabulary controlled, human subjective in nature and also, cannot predict future use. In some cases, it is rather difficult to characterize certain important real world concepts, entities, and attributes by means of text only. The shape of a single object and the various spatial constraints among multiple objects in an image are examples of such concepts for image comparison. Another indexing method is content-based similarity.

Content-based image retrieval systems rely on similarity measures. The four major classes of similarity measures are based upon color, texture, shape, and object and relationship similarity [3]. Once a measure of similarity is determined, the corresponding actual images are retrieved from the database. Due to the lack of any unified framework for image representation, storage, and retrieval (see [4] for information on the emerging MPEG-7 standard), these symbolic representation schemes and retrieval techniques have greatly facilitated image management.

The Query By Image Content system commonly known as the QBIC system by IBM is an example of recent work in content-based image retrieval that is based upon color and texture similarity [5]. QBIC is able to perform database queries in terms of color percentage matching, color histogram matching, color layout similarity and texture. While a system such as this will perform well when the most important image feature is color, it will suffer when object shape is the dominant feature. The Veggie Vision system

by IBM provides a good example of the performance that can be achieved when the four similarity measures are combined in one system [6]. This system uses color, texture, shape, and size histograms for the identification of produce. The success the Veggie Vision system has achieved, confirms the value for the integration of shape similarity with that of color and texture.

This thesis will concentrate on extracting shape structures with the intent of developing better representations of scenes and better means of matching in the domain of scenes with significant man-made object content. With a view towards the recognition of man-made objects in scenes, the natural features to use as the basis of the matching algorithms are lines, corners, and ribbons. The retrieval process has been divided into three sub processes; the feature extraction process, data representation, and graph matching algorithm. Feature extraction and data representation are offline processes whereas graph matching is an online process.

The Canny edge detector and Hough Transform are used for feature extraction. These features such as lines, corners and ribbons are then processed for extraction of relative attributes, for example distance, angle, and orientation, for matching purposes. All this information is then stored into a graph data structure separately for each image in MAT file format used by MATLAB for storing data. A detailed discussion on these processes may be found in Chapter 4 and 5.

Our image retrieval system requires a query image for the image retrieval process. The query image is then processed for feature extraction and representation. The extracted features are then compared with the features of all the images already stored in

the database and the images are displayed in descending order basing on percentage of similarity. Figure 1-1 is one of the examples of image retrieval. In this example, a query image of a Small Building was submitted and the program successfully retrieved all the images of that Small Building taken from different views.

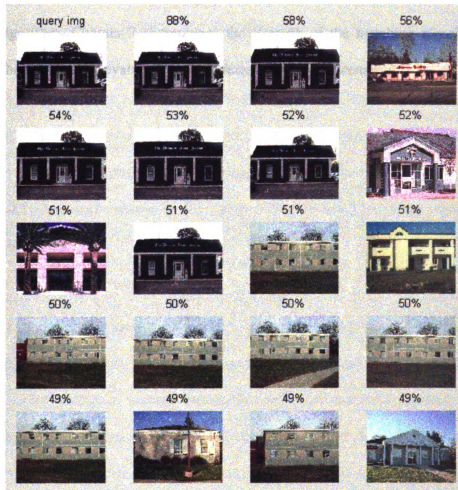


Figure 1-1 Query image of Small Building and results of our retrieval system.

The core work of this thesis is presented in Chapter 3 to Chapter 6. Chapter 3 provides an overview of the image retrieval system and explains the logical



decomposition of our system into three main modules: feature extraction module, features representation module and graph matching module. I will give a detailed description of the techniques used for edge, line, corner, and ribbon detection in Chapter 4, and discussing the results of these detectors applied to both hand-drawn images and manmade object. Chapter 5 and 6 are devoted to graph data structure development and the graph matching algorithm. Chapter 7 summarizes the strength of the matching algorithm, as well as its shortcoming by evaluating it in different worst-case scenarios.

## **2. Literature Review**

### **2.1 Current CBIR Techniques**

Content Based Image Retrieval 'CBIR' works on a principle of retrieving images from a database by comparing features automatically extracted from the images themselves. The commonest features used for CBIR are color, texture or shape. A typical CBIR system allows users to create queries by submitting an example of the type of image being sought, though some offer alternatives such as selection from a palette or sketch input. The system then retrieves images from the database whose feature values match those of the query most closely and displays thumbnails of these images on the screen. Some of the more commonly used types of features used for image retrieval are described below.

#### **2.1.1 Color Based Retrieval**

Color is one of the most widely used features in content-based image retrieval systems. The color feature is relatively robust to background complications, image resolution and orientation. Color histogram is most commonly used to extract color features. Several other methods for color-based image retrieval systems have been described in the literature, but most are variations on the same basic idea. Each image added to the collection is analyzed to compute a color histogram, which shows the proportion of pixels of each color within the image. The color histogram for each image is then stored in the database. At search time, the user can either specify the desired proportion of each color (75% olive green and 25% red, for example), or submit an example image from which a color histogram is calculated. Either way, the matching process then retrieves those

images whose color histograms match those of the query most closely. The matching technique most commonly used, histogram intersection, was first developed by Swain and Ballard [7]. Variants of this technique are now used in a high proportion of current CBIR systems. Methods of improving on Swain and Ballard's original technique include the use of region-based color querying [8].

### **2.1.2 Texture Based Retrieval**

"A variety of techniques have been used for measuring texture similarity; the best techniques rely on comparing values of second-order statistics calculated from query and stored images" [10]. Fundamentally, these calculate the relative brightness of selected pairs of pixels from each image. From these Tamura et al[9], calculated measures of image texture such as the degree of contrast, coarseness, directionality and regularity. Liu et al[10] calculated *periodicity*, *directionality* and *randomness*. Texture queries can be made in a manner similar to color queries, by supplying an example query image. The system then retrieves images with texture measures most similar in value to the query. Ma and Manjunath [11] introduced a new technique of texture thesaurus that retrieves images based on similarity.

### **2.1.3 Shape Based Retrieval**

Shape is an important feature in content-based image retrieval. However, shape is not a well-defined concept mathematically. Because of uncertainty in shape representations, the retrieval system may work well only in certain environments. Shapes can be represented either globally such as with aspect ratio, perimeters and moments, or locally such as with boundary segments. An alternative method for shape matching is elastic deformation of templates by Pentland et al [12]. Queries to shape retrieval systems are

made either by identifying an example image to act as the query, or as a user-drawn sketch [13].

Shape matching of three-dimensional objects is a more challenging task – particularly where only a single 2-D view of the object in question is available. While no general solution to this problem is known, some useful inroads have been made into the problem of identifying at least some instances of a given object from different viewpoints. One approach has been to build up a set of 3-D models from the available 2-D image, and match them with other models in the database [14]. Another is to generate a series of alternative 2-D views of each database object, each of which is matched with the query image [15]

#### **2.1.4 Retrieval by other Types of Primitive Feature**

One of the oldest established means of accessing pictorial data is retrieval by its position within an image. Accessing data by spatial location is an essential aspect of geographical information systems and efficient methods to achieve this have been around for many years [16]. A similar technique was proposed by Gudivada and Raghavan [17] to allow users to search for images containing objects in defined spatial relationships with each other.

Several other types of image feature have been proposed as a basis for CBIR. Most of these depend on complex transformations of pixel intensities. These techniques aim to extract features, which reflect some part of image similarity. The most well known technique of this type uses the wavelet transform to model an image at several different resolutions. Promising retrieval results have been reported by matching wavelet features computed from query and stored images [18].

Retrieval by appearance is another method that gives interesting results. Two versions of this method have been developed, one for whole-image matching and one for matching selected parts of an image. The part-image technique involves filtering the image with Gaussian derivatives at multiple scales [19], and then computing differential invariants; the whole-image technique uses distributions of local curvature and phase [20].

The advantage of all these techniques is that they can describe an image at varying levels of detail (useful in natural scenes where the objects of interest may appear in a variety of appearances), and avoid the need to segment the image into regions of interest before shape descriptors can be computed.

One early system was designed to interpret and retrieve line drawings of objects within a narrow predefined domain, such as floor plans for domestic buildings. The system analysed object drawings, labelling each with a set of possible interpretations and their probabilities. These were then used to derive likely interpretations of the scene within which they appeared. GRIM\_DBMS [21].

One system recently introduced by Qasim Iqbal and J. K Aggarawal [22], used higher level and lower level vision methodology to retrieve manmade objects. Higher level vision was used to extract features such as 'L' junctions, 'U' junctions and parallel groups. Lower-level analysis is performed globally by using Gabor filters to extract texture features. A manmade object region of interest extracted by using perceptual grouping is used as a frame for conducting lower-level analysis, but the method of image retrieval is weak. Instead of comparing color, texture and shape features, these features are weighted and used for comparison.

Object recognition has also been an area of interest to the computer vision community for many years. Techniques are now being developed for recognizing and classifying objects with database retrieval in mind. David et al [23] have attracted considerable publicity for themselves by developing a technique for recognizing naked human beings within images, though their approach has been applied to a much wider range of objects, including horses and trees. Haering et al [24] also developed a method for identifying deciduous trees via their foliage. All such techniques are based on the idea of developing a model of each class of object to be recognized, identifying image regions which might contain examples of the object, and building up evidence to confirm or rule out the object's presence. Evidence will typically include both features of the candidate region itself (color, shape or texture) and contextual information such as its position and the type of background in the image.

In contrast to these fully automatic methods is a family of techniques, which allow systems to learn associations between semantic concepts and primitive features from user feedback. The earliest such system was FourEyes from Minka [25]. This encourages the user to explain selected regions of an image, and then proceeds to apply similar semantic labels to areas with similar characteristics. The system is capable of improving its performance with user feedback. Another approach is the concept of the semantic visual template introduced by S F Chang et al [26]. Here, the user is asked to identify a possible range of color, texture, and shape or motion parameters to express his or her query, which is then refined using relevance feedback techniques. When the user is satisfied, the query is given a semantic label (such as "sunset") and stored in a query database for later use.

## **2.2 Practical Applications of CBIR**

A wide range of possible applications for CBIR technology has been identified e.g.

- Crime prevention
- The military
- Intellectual property
- Architectural and engineering design
- Fashion and interior design
- Journalism and advertising
- Medical diagnosis
- Geographical information and remote sensing systems
- Cultural heritage
- Education and training
- Home entertainment
- Web searching
- Robotics
- Industries

### **3. Methodology**

This chapter focuses on the development of a methodology for feature extraction using higher-level image analysis. Manmade objects like buildings generally have sharp edges and straight boundaries. The prominent characteristics of a building are the apparent regularity and the relationship of its component features. An image containing large manmade objects such as buildings, bridges and roads, will exhibit a large number of significant edges, junctions and parallel lines, compared to an image with predominantly non-manmade objects (such as scene of vegetation). These images are generated by the presence of corners in the object, such as doors, windows, pillars, and the boundaries of the buildings. These higher-level features exhibit apparent regularity and relationship, and are strong evidence of structure in a scene. The presence of these distinguishing features in an image can be utilized for image comparison and subsequently for image retrieval.

#### **3.1 Data Representation**

To compare different images, the following higher-level features are extracted from images:

- Straight lines
- Ribbons (Parallel lines)
- Corners

Manmade objects are generally rigid; therefore, these representations adequately define their shape descriptions. The extraction process is hierarchical. The first stage is extraction of edges from the image, then extraction of straight lines, which tend to be



small fragments that are grouped to form longer lines. Lines are then used to find corners and ribbons. A set of lines terminating at a common point is called a corner. The corner is an important relation. According to the proximity rule of perceptual grouping, the human visual system easily groups corners. In fact, it has been suggested that a major function of eye movement is to determine corners and edges [27].

The straight-line segments are also used to extract parallel lines. A large number of manmade objects contain parallel structures such as pillars, beams, doors and windows. Human vision can rapidly identify parallel lines [28]. A parallel relation is a non-accidental relationship that can be used to infer relationships. These parallel lines are grouped together to find parallel groups, which are then used to extract significant parallel groups [29]. The rationale for the grouping relations described above is the following.

- Spatially closed primitive structures are likely to be related and to reflect meaningful structures.
- Accidental image relations of natural objects may cause some primitive structures. For example, line segments extracted from a cluster of tree leaves may accidentally form a ribbon or corner. Such ribbons or corners tend to be randomly and sparsely distributed and unlikely to form meaningful structures.
- Manmade objects usually consist of regular structures.

### 3.2 Image Matching Process

Before explaining each step in detail, here, I would like to give an overall view of my image retrieval methodology. I have divided my matching process into three main parts namely:-

- Feature Extraction Module
- Features Representation Module, and
- Graph Matching Module

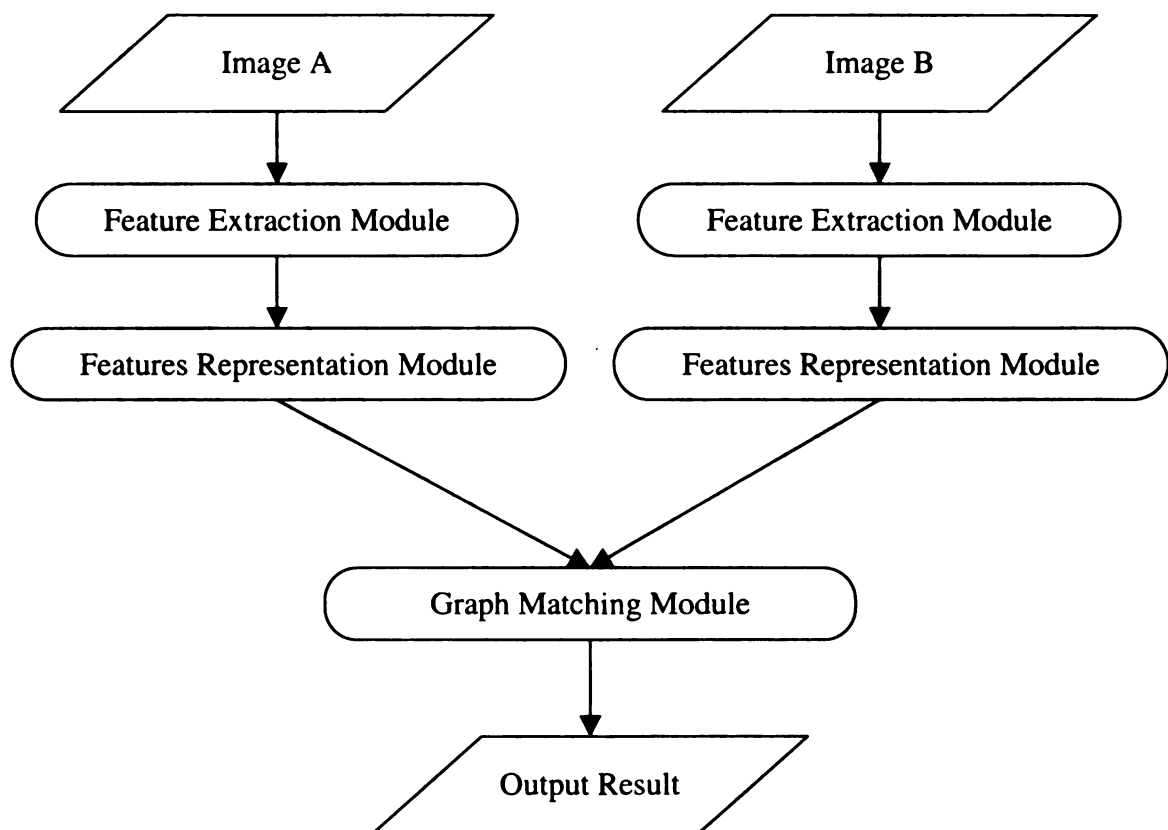


Figure 3-1 Overall Methodology for Image Comparison

Figure 3-1 shows the overall methodology for an image retrieval system. In Figure 3-1 the feature extraction module is actually a combination of a number of other processes (i.e such as Canny edge detector, hough transform, line fit routine that joins small line segments into longer lines, reordering the lines, corner detector and ribbon detector). These processes are responsible for extracting higher-level features such as lines, corners and ribbons from the given gray level images. After extraction of higher-level features, the feature representation process starts and collects several other parameters (Table 3-1) of these features before storing them into graph data structures. When these parameters are calculated, then all these parameters are stored into respective graph data structures. The graph-matching algorithm actually compares the query image with a stored image from the database.

<b>Features</b>	<b>Parameters</b>					
<b>Lines</b>	Slope of Line	Edge Gradient	Line Orientation	Distance between Lines		
<b>Corners</b>	Angle of a Corner	Corner Orientation	Corner Precincts	Distance between Corners	Angle between Corners	
<b>Ribbons</b>	Ribbon Width	Slope of Line	Edge Gradient	Type of Ribbon	Ribbon Orientation	Distance between Ribbons

Table 3-1 Features and their Parameters

## 4. Feature Extraction Module

Feature extraction is one of the most important steps in our image retrieval system as all other steps depend on this process. This process is divided into sub processes in order to analyse each step critically. Figure 4-1 illustrates the processing chain that each image will ultimately pass through for feature extraction.

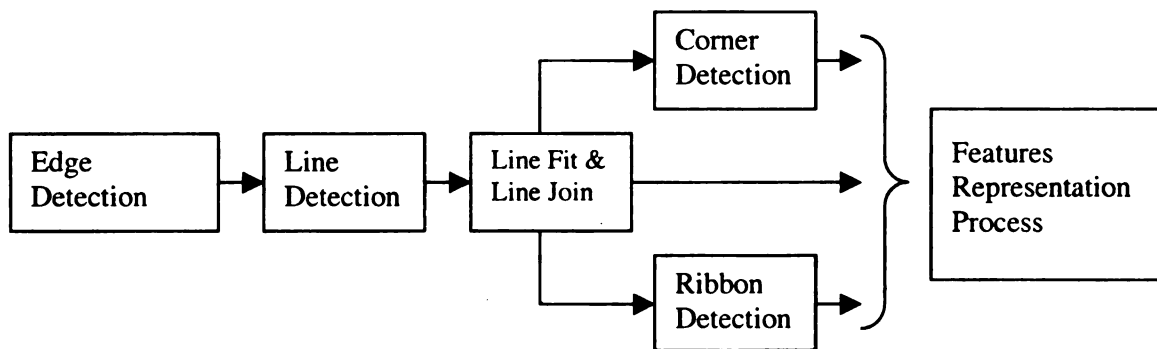


Figure 4-1 Process for feature extraction

Because of this processing sequence and the number of parameters at each stage, one must be conscious of a natural dependency. Namely, no matter how robust the corner and ribbon detectors are, they are dependent upon the line detector, which in turn is dependent upon the edge detector. Thus, the number of parameters that influence the last two detectors may be larger than desired. With this in mind, the edge and line detectors must exhibit acceptable performance over a wide range of images, balanced with the need for parameter optimisation and automation.

### 4.1 Edge Detection

The early stages of vision processing identify features in images that are relevant to estimating the structure and properties of objects in a scene. Edges are one such feature.

Edges are significant local changes in the intensity image and are important features for analyzing images. Edges typically occur on the boundary between two different regions in the image. Edge detection is frequently the first step in recovering information from images. Due to its importance, edge detection continues to be an active research area.

At its simplest, an edge is a sharp discontinuity in gray-level profile. However, the situation is complicated by presence of noise and image resolution. An edge is specified by its magnitude and its direction. A number of linked edge points may be better approximated by a linear segment called an edgel [30]. There are many types of edges; they may be classified into three classes: step edge, roof edge and linear edge. A good edge operator must have the following properties [30]; it must

- Operate locally
- Be efficient
- Be sensitive to the orientation and magnitude of an edge
- Work in the presence of noise
- Be insensitive to threshold value
- A further condition imposed by Canny [Canny 86] is that the operator must not have multiple responses to a single edge.

To detect these different types of edges many edge operators have been suggested in the computer vision literature. i.e Sobel [Prewitt 70], Roberts [Roberts, 65], Krish [Krish, 71], Marr and Hildreth [Marr, 80]. These edge operators possess some of the above listed properties, but not all, and also are good for one type of edge and not good for other types of edge. Canny in 1986 developed an edge operator, which extracted not only step edges but also ridge and roof edges. Chen et al. [31] have evaluated many of

these algorithms for edge detection on brain images obtained from various sources such as computer tomography (CT), magnetic resonance images (MRI), and positron emission tomography (PET). Their results indicate that none of the edge detectors mentioned are universally applicable. Since in our thesis first step in the feature extraction sequence is to extract edges from the grey scale image, the use of the Canny edge detector to satisfy this step is justified as the Canny operator has a consistent response with single smoothing parameter. The Canny edge detector used here is the Canny algorithm described by Stockman and Shapiro [3].

#### 4.1.1 Canny Edge Detector

The original gray scale image  $I[i, j]$  is smoothed with a Gaussian filter  $G[i, j, \sigma]$ , where  $\sigma$  is the spread of the Gaussian that controls the degree of smoothing.

$$S[i, j] = G[i, j, \sigma] * I[i, j] \quad 4.1$$

The gradient magnitude  $M[i, j]$  and direction  $\theta[i, j]$  are then computed from the smoothed image  $S[i, j]$ . The magnitude image is used to provide votes for possible edge pixels. If the magnitude is higher than a given low threshold, then the pixel is classified as a possible edge. Next, the gradient direction is used in a step called non-maximal suppression. The purpose of this step is to thin a possible edge by suppressing any pixel response that is not higher than the two neighbouring pixels on either side of it along the direction of the gradient. Finally, the magnitude of each pixel in the image of possible edges is compared against a given high threshold. If a pixel passes the test, it is classified

as an edge pixel, and the magnitudes of its neighbours are recursively tested against the low threshold.

#### 4.1.2 Experiment and Results with the Canny Edge Detector

The edge detection is the first stage of the segmentation chain; the spread parameter  $\sigma$  is perhaps the most important of all parameters to be encountered. For example, Figure 4-2 shows the edge image of different shapes with default parameter of  $\sigma=1.0$  and Figure 4-3 shows the edge image with  $\sigma=2.0$

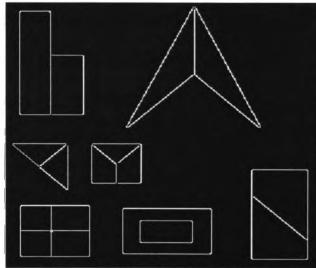


Figure 4-2 Edge image of different shapes with  
default parameter  $\sigma=1.0$

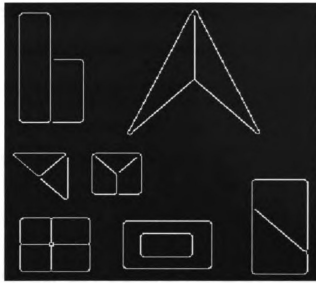
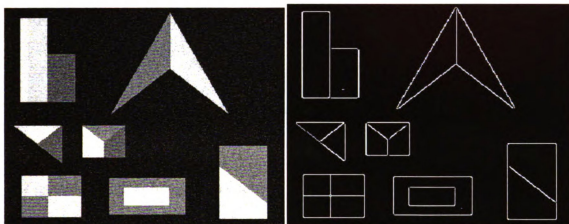


Figure 4-3 Edge image of different shapes with  $\sigma=2.0$

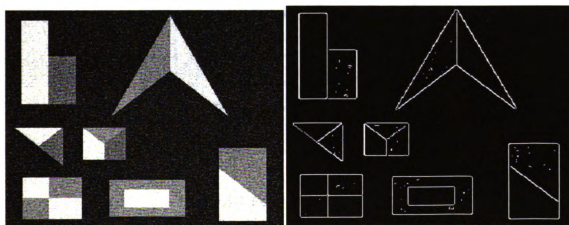
In Figure 4-3 corners are somewhat distorted when spread  $\sigma=2.0$  was used. The smoothing parameter  $\sigma$  may also have to be adjusted in the presence of noise. Figure 4-4 to Figure 4-7 show the results of adding Gaussian noise to the test image. Notice that the adjustment of  $\sigma$  yields the proper edges and these edges are still as 'clean' as those detected from the noise free image using the same  $\sigma$ . The image with medium noise will not suffer from using a small  $\sigma$  because the edges are still 'clean'. The image with high noise will undoubtedly suffer in later stages of detection, due to the jagged edges and noise artifacts. One should not be alarmed at such results, since the amount of noise that has been added is much greater than what is expected in practice. A more reasonable amount of noise can be seen in the original Thick W, Taj Mahal and Small Building images (see Figure 4-8 to Figure 4-10). This noise was added during the scanning



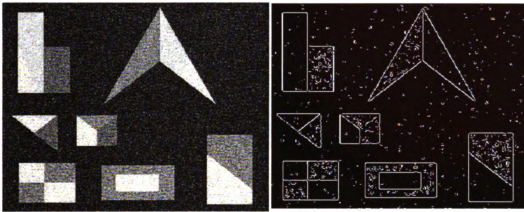
process, and it will be shown that this noise does not affect the overall performance of the detection system.



(a) (b)  
Figure 4-4 (a) Test Image with Gaussian noise of 0.01. (b) Edge image with  $\sigma=1.0$



(a) (b)  
Figure 4-5 (a) Test Image with Gaussian noise of 0.02. (b) Edge image with  $\sigma=1.0$



(a)

(b)

Figure 4-6 (a) Test Image with Gaussian noise of 0.05. (b) Edge image with  $\sigma=1.0$

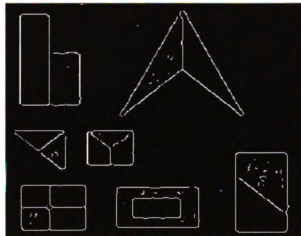


Figure 4-7 Edge image with  $\sigma=3.0$

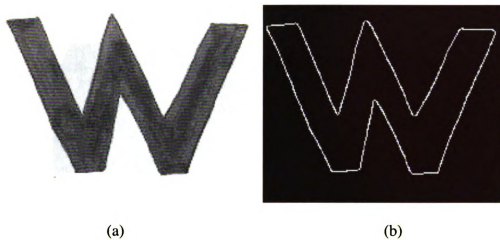


Figure 4-8 (a) Thick W with original scanner noise. (b) Edge image of Thick W with  $\sigma=1.0$

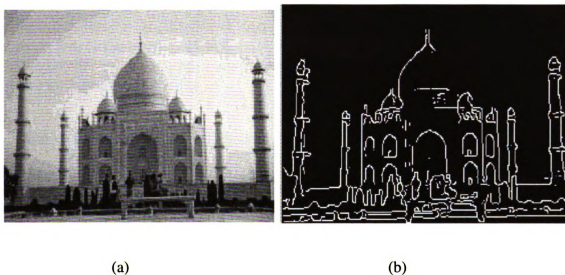


Figure 4-9 (a) Taj Mahal with original scanner noise. (b) Edge image of Taj Mahal with  $\sigma=1.0$



(a)

(b)

Figure 4-10 (a) Small Building with original scanner noise. (b)  
Edge image with  $\sigma=1.0$

## 4.2 Line Detection

The Hough Transform (HT) is used in computer vision and pattern recognition for detecting geometric shapes (in digital images) that can be defined by parametric equations. It is a mapping from the image plane to the parameter space, and essentially consists of a voting process followed by a peak detection stage. The advantages of the transform are its robustness to noise and discontinuities in the image, while the disadvantages are its demand for a large amount of computing time and storage (although with present advancement in computer memory and computing power it's not a problem). The time and space requirements depend on the number of parameters required to describe the pattern, the resolution of accumulator array, and the image resolution. A straight line is the simplest of all geometric patterns and can be described with only two parameters (see Equation 4.2).

Different aspects of the HT have been investigated and reported in the literature [3,30,32,34]. The compute bound nature of the HT has inspired the development of efficient algorithms and implementations of the transform. Performance of the transform with respect to accuracy of detection has been discussed in the literature. A formal mathematical definition of the transform, its properties and relationships appear in [3,30,33,34].

Commonly used parameterizations of a straight line are slope-intercept and the normal forms. When the HT is used for the detection of straight lines in an image, only the parameters of the infinite line are obtained. It does not provide any information regarding the length, position and endpoints of the line segment in the image plane.

H

m

P

cc

de

d.

pr

de

Pr

de

lin

for

Pr

me

sh

per

However, in computer vision application, the length and exact position of a line segment in addition to its normal parameters, are required for locating the line segment. The parameters, the length, and the coordinates of end points of a line segment constitute the complete line segment description. Techniques to find the complete line segment description using the Hough Transform have not been thoroughly studied, although a few algorithms are available in the literature [33]. The available methods are based on the projection of the line on either the x or y axis in the image plane. I also used the technique of least square error fit to more precisely estimate the line parameters.

In order to detect and report on the line detection process, one must first decide on a proper definition of a line. There are certain unavoidable problems that occur when detecting lines in an image. These problems stem from the fact that, even though a digital line is not the same as its mathematical counterpart, the mathematical definition is used for detection purposes. In order to take advantage of both concepts of a line, or more properly a line segment, I define a line to be a set of pixels that fit a mathematical line model. For the purposes of detection, the model used for a line is

$$d = x \cos(\theta) + y \sin(\theta) \quad 4.2$$

where  $d$  is the distance from the line to the origin and  $\theta$  is the angle between the vector perpendicular to the line and the x-axis see Figure 4-11.

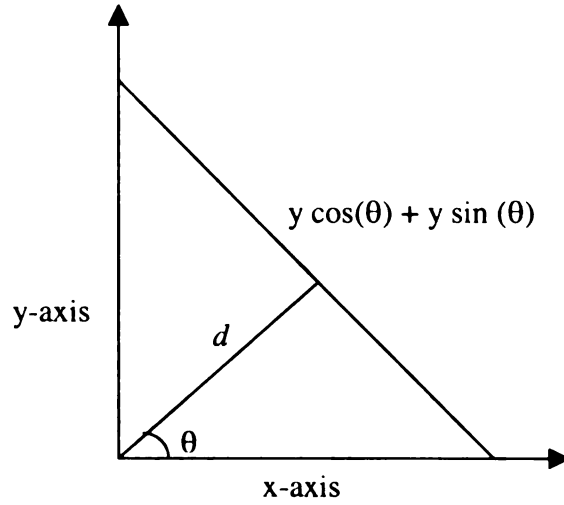


Figure 4-11 The parameters  $d$  and  $\theta$  used in the equation 4.2 of a straight line.

The line detection algorithm employed is a variation of the Hough Transform with the O’Gorman and Clowes method for extracting straight lines. Refer to Chapter 10 of Stockman and Shapiro [3] for a detailed description of this method. The idea behind the Hough transform is to build evidence for the existence of lines using the model

$$d = c \cos(\theta) + r \sin(\theta) \quad 4.3$$

where  $r$  and  $c$  are the respective row and column location of a pixel and  $d$  and  $\theta$  are the length of normal and angle of the normal with respect to the positive  $x$ -axis. The angle  $\theta$  was computed using Sobel  $5 \times 5$  gradient operator. The distance from a potential line to the origin can thus be computed by  $d = |c \cos(\theta) + r \sin(\theta)|$ , so that

$$0 \leq d \leq d' \quad 4.4$$



where  $d'$  is half the length of the main diagonal of the image. The distance  $d$  and angle  $\theta$  are then quantized for each feature point in the image, i.e. initially  $d$  is assigned to the nearest multiple of 3, and  $\theta$  is assigned to the nearest multiple of 10. A  $36 \times \frac{d'}{3}$  accumulator array,  $V$ , is used to gather votes from pixels. For example, if a pixel has a quantized angle  $\theta$  of  $180^\circ$  and a quantized distance  $d$  of 240, the value in  $V[18][80]$  is incremented. An array of point lists is also kept to store the list of pixels that voted for a particular line.

Once the accumulator array has been filled, it is searched for the line with the greatest number of votes. Then, the set of pixels that voted for the line are processed to find neighbors whose angles are within ten degrees of the given angle. If such a neighbor is found, it is deleted from its point list, added to the current point list, and the value in its accumulator array is decremented. This process is referred to as the Merge stage of the Hough Transform. The process of choosing the greatest bin and merging is continued as long as the size of the bin in the accumulator array is greater than a chosen length parameter. This parameter dictates the length of the lines that can be detected.

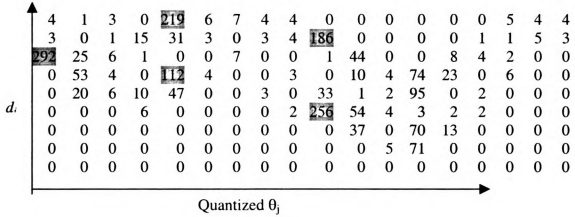


Figure 4-12 Accumulator Array showing the peaks

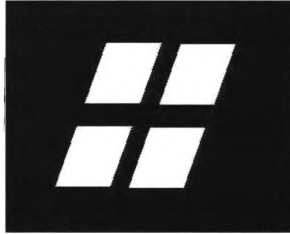


Figure 4-13 Test image

Figure 4-12 is a portion of accumulator array for Figure 4-13. The array size in Figure 4-12 is  $d_i * \theta_j$  where  $i=1$  to  $\text{maximum\_distance} / 3$  and  $j=1$  to 36. The digits marked with dark color shows the peaks in accumulator array, which are quite significant and can be detected easily, but if there is some diagonal line in the image plane that has some

irregular slope it will be hard to detect. The votes marked with light color actually represent a diagonal line and we can observe from the accumulator array that votes are spread from top to bottom in different peaks and it very hard to detect complete diagonal line.

```

Merge the peak in accumulator array A
A is accumulator array
[x , y] = size of A
for d = 1 to x
    for  $\theta$  = 1 to y
        if (A[d ,  $\theta$ ] > threshold)
            [i , j] = Search_Highest_Peak( A, d ,  $\theta$ )
            k = i;
            while A(i,j) > 0
                Merge A(k,j) and A(i+1,j); // also merge point list
                A(i+1,j)=0;
                i=i+1;
            end while
            while A(i,j) > 0
                Merge A(k,j) and A(i-1,j); // also merge point list
                A(i-1,j)=0;
                i=i-1;
            end while
        end if
    end for
end for

Search_Highest_Peak( A, k,  $\ell$  )
    Temp = A[k,  $\ell$ ]
    i=k;
    j= $\ell$ ;
    while A[i,j] > 0
        if A[i , j] > Temp
            Temp= A[i,j];
        end if
        i=i+1;
    end while
    i=k;
    while A[i,j] > 0
        if A[i , j] > Temp
            Temp= A[i,j];
        end if
        i=i-1;
    end while
    return k,  $\ell$ 

```

Algorithm 4.1 Algorithm to merge peaks in the accumulator array

The algorithm that I designed in Algorithm 4.1 can detect diagonal lines and it's a modified version of the technique proposed by M. Atiquzzaman and M.W Akhtar [34]. The algorithm searches for the peak in accumulator array  $V(d, \theta)$  from left to right and up to down by incrementing  $\theta_j$  and  $d_i$  respectively. When it finds the first peak in the accumulator array that is greater then the threshold, it calls a routine to search for the highest peak in the  $d$  direction, searching from up to down and down to up and returns the position of the highest peak. Now the calling program searches all the adjacent peaks up and down and merges them into the highest peak making it more prominent.

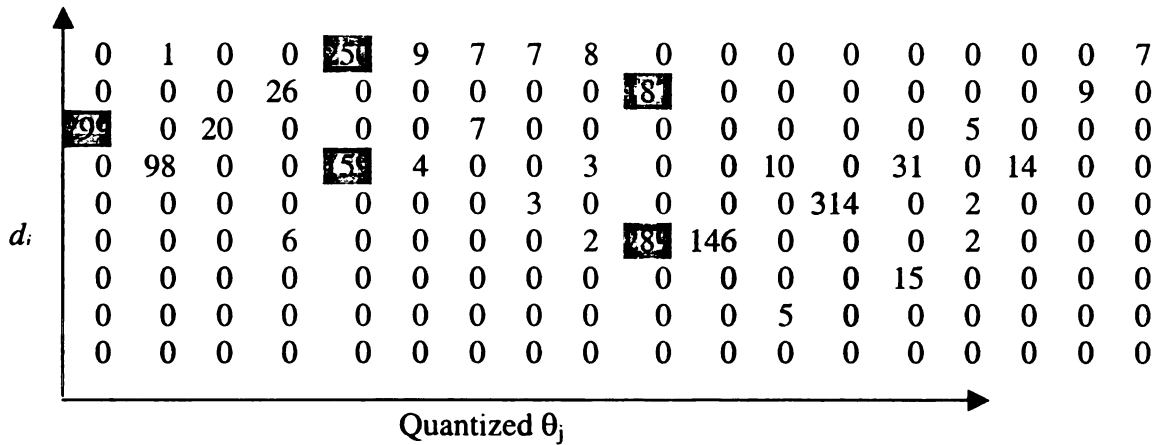


Figure 4-14 Accumulator Array showing the merged peaks

I run this algorithm on Figure 4-12, which is a portion of accumulator array of Figure 4-13 by setting threshold to 4. The resultant accumulator array, is shown in Figure 4-14. The algorithm scans from left to right and up to down. The first peak in Figure 4-12 is 4 since it equals the threshold, therefore the second part of the algorithm will activated and will search for the highest in that column from up to down. When the algorithm finds the

highest peak in the column, such as 292, it starts merging the columns until it finds a zero. Now the peak in the first column becomes 299. Similarly, the rest of the smaller peaks are merged into other larger peaks, which makes them quite prominent.

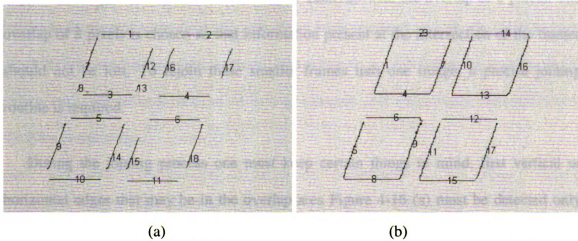


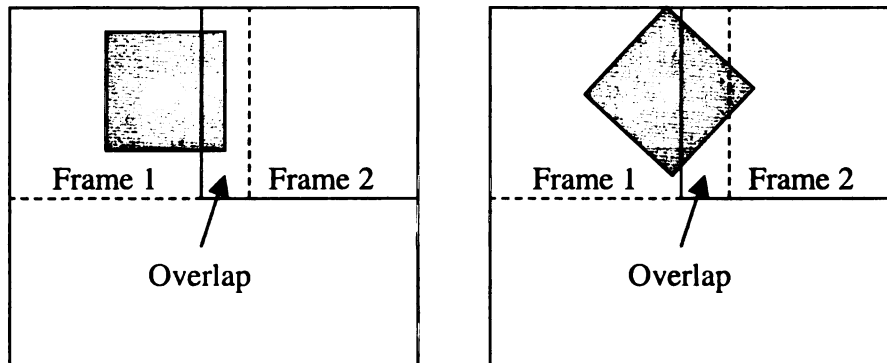
Figure 4-15 (a) line detection before applying algorithm 4.1 and (b) line detection after applying algorithm 4.1

In Figure 4-15 (a) all the diagonal lines are not completely detected. Diagonal lines 7 and 8 should have been merged together but they were not, similarly lines 12 and 13 should have been merged. Horizontal lines are completely detected, whereas diagonal lines are not completely detected. The reason for incomplete detection is the same as peaks are spread in accumulator array. Line segments 7 and 8 in Figure 4-15 (a) are merged into single line segment 1 in Figure 4-15 (b), similarly line segments 12 and 13 in Figure 4-15 (a) are also merged into single line segment 7 in Figure 4-15 (b).

### 4.2.1 Line Joining

It is well known that HT may be confused in detecting lines when the image size is more than 250x250. To be consistent in line detection, images greater than 192x192 are required to be processed in smaller frames of 128x128 with the overlap of 8 pixels. The overlap of 8 pixels is chosen so that information present at the intersection of the frames should not be lost. To rejoin these smaller frames into one image, a precise joining routine is required.

During the joining process one must keep certain things in mind, first vertical or horizontal edges that may be in the overlap area Figure 4-16 (a) must be detected only once and those diagonal lines Figure 4-16 (b), which continue from frame 1 to frame 2, must be joined perfectly.



(a) (b)  
Figure 4-16 (a) Vertical overlapping edge (b) Diagonal overlapping edge

Not only does this problem require the line-joining routine but the real images such as Taj Mahal and Small Building may also require least square error fit and line-joining

routines to join any broken line segments detected by the Hough Transform. In practice, the line segments obtained from the Hough Transform may be fragmented and must be grouped together to obtain longer line segments. To accomplish this task, a set of closely bunched and similarly oriented straight-line segments must be found and joined together. I used least squares error fitting to join those lines segments, which are close to each other. Before joining these line segments, line segments are labeled in sequential order to reduce the computational time, as we have to compare only neighboring lines and not all the lines in the image.

$$K(L_i, L_j) < \delta_n \quad 4.2$$

$K$  is function that calculates the least squares fit and joins pair of lines  $[L_i, L_j]$  if least square error is less then the threshold  $\delta_n$ , where  $i$  and  $j$  are line labels and  $i \neq j$  and  $j \in [i - 5, i + 5]$ .

#### 4.2.2 Line Detection Results

The results of the line detection procedure for all of the test images can be found in Appendix 3. The line detection for every image in this appendix uses the default parameter of 5 pixels for the line length, any line segments less than 5 pixels are taken as noise. In this section, I will discuss the results on specific images that exemplify both the positive and negative aspects of the line detector, and in particular, view how the line length parameter influences the number of lines that are correctly detected.



The first images that I will investigate are hand-drawn images and test patterns. Since these images are not as complicated as natural scenes, one should expect that the line detector would perform with a high degree of accuracy. Figure 4-17 (a) (b) shows the results of line detection on two of the test images. I consider the line detection for these images to be highly accurate. Every line that one may expect to be detected has been detected. Figure 4-18 (a) (b) shows line detection results that are partly successful. In Figure 4-18 (a), the lowermost and upper right horizontal lines are missed, and in Figure 4-18 (b), the lowermost horizontal line is split into two lines and the lines in the minarets are also not joined. The reason that these potential segments escaped detection is because their individual bins did not receive enough votes to warrant line fitting.

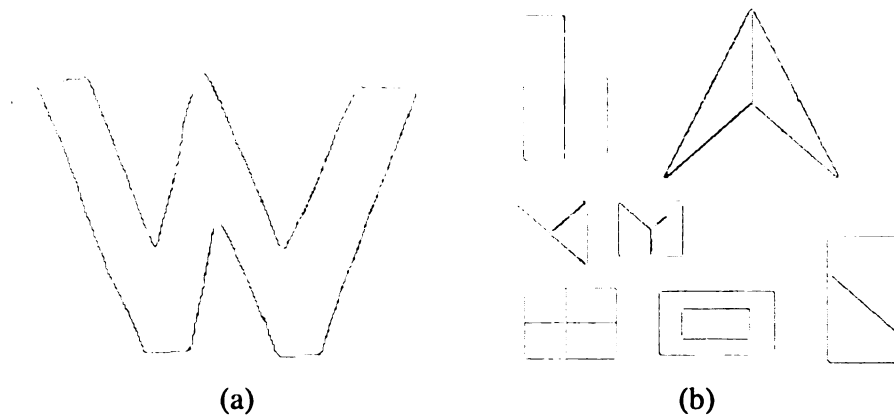


Figure 4-17 (a) and (b) Successful line detection



(a) (b)  
Figure 4-18 (a) and (b) Incomplete line detection

As with the test images, the set of scene images exhibit both successes and failures with respect to line detection. Some of the positive line extraction examples can be seen in Figure 4-19 (a) and (b). In particular, the vertical lines formed by the columns in the Figure 4-19 (a) and the broken lines of the Figure 4-19 (b) have been successfully identified.



(a) (b)  
Figure 4-19 (a) Lincoln Memorial and (b) Big Taj, line length = 5

The line extraction results from Figure 4-20 (a) and Figure 4-20 (b) shows some successful and unsuccessful line detection results. While many lines are correctly identified in Figure 4-20 (a), some of the most important lines, namely those formed by the crossbeams, are not detected. This can be attributed to the soft shadows present beneath the crossbeams. The dismal lack of line detection in Figure 4-20 (b) is due to the size of the image, only 150 x 150 pixels, and the distance of the structure from the viewer.

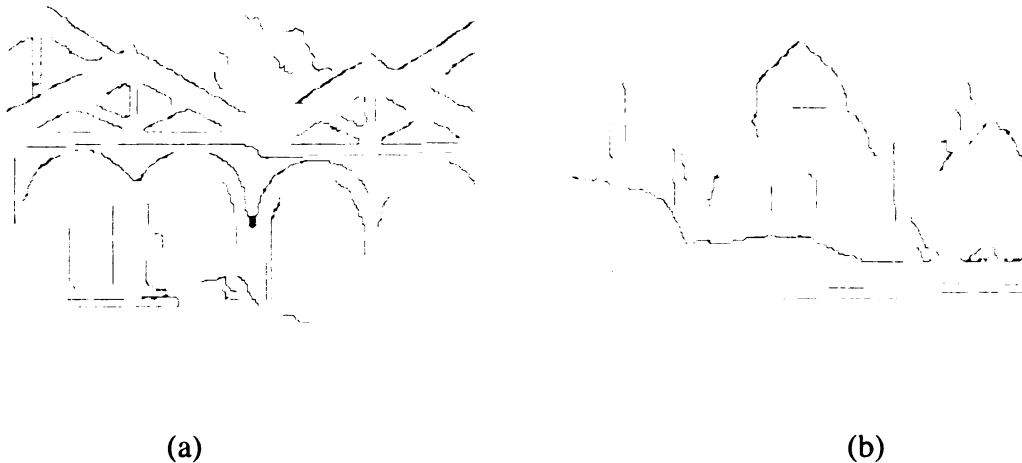


Figure 4-20 (a) Nairobi 1 and (b) Small Taj, line length = 5

### 4.3 Corner Detection

A corner is defined to be the point of intersection of two distinct proximal line segments making a shape of “L”. Here we will not consider corners (“ $\perp$ ”). The attributes of a corner consist of an angle and a location. Since lines are fit in a non-quantized sense, it is possible to define the location of a corner with sub-pixel accuracy. In order to detect corners, the line lists found in the line detection stage must be processed.

$$K(L_i, L_j) < \delta_n \quad 4.3$$

$K$  is function that finds the corner between pair of lines  $[L_i, L_j]$  if the threshold is less than  $\delta_n$ , where  $i$  and  $j$  are line labels and  $i \neq j$  and  $j \in [i - 10, i + 10]$ .

$m = (y - y_1) / (x - x_1)$  is the slope of the line  $L_i$  with two end points  $(y, x)(y_1, x_1)$ . First, the slope of each line  $L_i$  is computed with equation 4.4 and then compared against the slope of  $L_j$  (where  $i \neq j$  and  $j \in [i + 10, i - 10]$ ). If the slopes are commensurate, the point of intersection is computed. Because the points of interest occur as the intersection of line segments rather than lines, the computed intersection point must be compared against the initial and terminal points of each line segment. If the point is within a certain distance, the corner threshold, of either the initial or the terminal point of both segments, or if the point lies on both segments, the pair is classified as a corner. The default distance for the corner threshold is ten pixels. Essentially, the corner threshold is the distance that each line segment is extended in both directions before testing for intersections.

After detecting the corner point, the measure of the corner angle is computed.

$$\text{Angle}_{L_i L_j} = \tan^{-1} ((m_j - m_i) / (1 + m_i * m_j)) \quad 4.4$$

where  $m_i$  is slope of  $L_i$  and  $m_j$  is slope of  $L_j$ , and  $i \neq j$

### 4.3.1 Corner Detection Observations

Since corner detection is dependent upon both the edge detection and line detection, there are three parameters that affect its performance: the Gaussian spread, the line length tolerance, and the corner threshold. Because the number of parameters is three, the reasons for incorrect corner detection may be difficult to pinpoint, and it may be difficult to find the best configuration of these parameters for a particular image. Upon referring to Appendix 4, the reader can verify that, for the majority of the examples, both false positives and false negatives (non-detection of corners) can be attributed to faulty line detection. A few exceptions to this, as well as some of the more successful attempts at corner detection can be seen in Figure 4-21 to Figure 4-23.

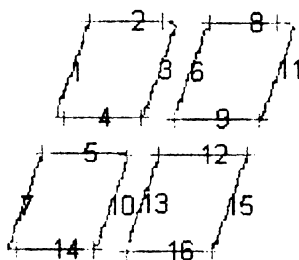


Figure 4-21 Test image with all the corners detected



Figure 4-22 Test Pattern with one corner missing (lower  
left corner of the lower middle rectangle)



(a)

(b)

Figure 4-23 Corner detection results for (a) Lincoln, 25 corners and (b)  
Taj Mahal, 60 corners

In Figure 4-21 all of the possible corners are detected but in Figure 4-22 one corner at the left bottom corner of the lower middle rectangle is missing and it is due to the corner threshold. In Figure 4-23 (a), the Lincoln memorial scene displays very strong corner

evidence in the upper left and upper right hand sides of the building. While this is encouraging, there are also corners that are incorrectly detected due to the edges of the columns extending into the strong line at the bottom of the building. This can be directly attributed to the default corner threshold. However, if the threshold were to be lowered, the strong corners at the top of monument may not be detected. The corner detection for the Figure 4-23 (b) Big Taj scene also displays mixed results. Both the points of intersection on the minarets and those on the windows are promising, but all of the window corners are not successfully detected, and there are many false positives amongst the crowd of people.

#### **4.3.2 Corner Detection Results**

The general observations that were made in the previous section provide evidence as to the strengths and weaknesses of the line-driven corner detector. In this section, the results of the corner detector are quantified so that an objective report of its accuracy can be made. The accuracy of the detector is determined by the confidence ratio, which is defined by

$$R = (C-F)/E \quad 4.6$$

Where C is the number of corners correctly detected, F is the number of false alarms, and E is the number of corners expected. Clearly, the number of corners that one “expects” to find in an image is not well defined. While some might define a corner to be a point with high curvature, others, including myself, consider a corner to be the potential point of intersection between two line segments. It will be impossible to make an

objective decision about expected corners in scene images. Therefore, only confidence ratings for test and hand drawn images will be reported. The corner detection results for these images are found in Table 4-1. The results are reported with the default values  $\sigma = 1.0$ , line length = 5, and corner threshold = 10.

<b>Image</b>	<b>Detected Corners</b>	<b>False Alarms</b>	<b>Expected Corners</b>	<b>Confidence Rating</b>
<b>Parallelogram</b>	16	0	16	100%
<b>Test Image</b>	35	4	32	97%
<b>Thick W</b>	9	1	12	67%
<b>Thin W</b>	6	0	6	100%
<b>Line Taj</b>	16	0	25	64%

Table 4-1 Confidence Rating for corner detection of hand-drawn images

#### 4.4 Ribbon Detection

Stockman and Shapiro define a ribbon as “an elongated region that is approximately symmetrical about its major axis” [3]. This definition encompasses many varied objects, including picture frames; bottles, columns, and any 2-dimensional object that possesses symmetry. When the images of such objects are projected onto a plane, the symmetry is translated into curves with reflective symmetry about an axis. When an actual cylindrical object is projected onto a plane, its symmetry is translated into parallel lines. The ribbon detection algorithm will only identify those ribbons that are the result of projected cylinders and rectangular prisms, namely straight ribbons. In this section, I will describe the algorithm for straight ribbon detection and explore the advantages and potential difficulties of ribbon detection.



#### 4.4.1 Ribbon Definition

For the purpose of identifying straight ribbons, a more practical definition of a ribbon must be given. To this end, I define a straight ribbon to be two lines whose edge gradients differ by  $180^\circ \pm 10^\circ$ , have approximately the same length, approximately the same slopes and the width should be  $\leq 1.5$  times the line length. To be more precise, given two lines,  $L_1$  and  $L_2$ , they form a ribbon if and only if the following conditions are satisfied:

- $170^\circ \leq | \theta_1 - \theta_2 | \leq 190^\circ$  where  $\theta_1$  and  $\theta_2$  are the respective average edge gradient of  $L_1$  and  $L_2$ .
- $l(L_i) \geq \frac{l(L_j)}{2}$  for  $1 \leq i, j \leq 2$  where  $l(L)$  is the length of line segment  $L$ .
- The projected line  $L_1$  should overlap  $L_2$  and vice versa.

A detected ribbon then inherits the attributes of each of its lines, as well as a width, length, and one of two possible types. The length of the ribbon is defined to be the average of the lengths of the two lines. The width of the ribbon is the distance between the lines.

#### 4.4.2 Type of Ribbons.

Ribbons are divided into two different types: Type-1 and Type-2. If the edge gradient  $\theta_1$  at line  $L_1$  is greater than the edge gradient  $\theta_2$  at line  $L_2$ , the ribbon is Type-1, otherwise it is Type 2. Essentially, a Type-1 ribbon is the one in which the gradient vectors of the

lines pointing away from each other, and a Type-2 ribbon is one with gradient vectors pointing towards each other. Figure 4-24 gives examples of the two different types.

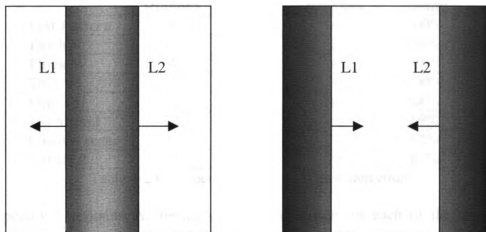


Figure 4-24 (a) is Type-1 ribbon and (b) is Type-2 ribbon (L1 must be left of L2)

#### 4.4.3 Ribbon Detection Results

The confidence rating that was defined for corner detection can also be used to determine the general accuracy of the ribbon detection algorithm. As with corner detection, the concept of an expected ribbon is not well defined, and the determination of such ribbons in images is fuzzy at best. For this reason, confidence ratings will only be reported for the class of hand-drawn images and images that have some obvious ribbons (see Table 4-2).

<b>Image</b>	<b>Detected Ribbons</b>	<b>False Alarms</b>	<b>Expected Ribbons</b>	<b>Confidence Rating</b>
<b>Test Pattern</b>	8	0	8	100%
<b>Thick V</b>	2	0	2	100%
<b>Thick W</b>	4	0	4	100%
<b>Thin W</b>	4	0	4	100%
<b>Line Taj</b>	9	0	9	100%
<b>Taj Mahal</b>	24	6	26	69%
<b>Lincoln scene</b>	44	29	24	62%
<b>Narobi 2</b>	21	5	26	61%

Table 4-2 Confidence Rating for ribbon detection

Appendix 5 contains the results of ribbon detection for each of the test images. Images that yielded perfect ribbon detections are Test Pattern, Thick V, Thick W, Thin W and Line Taj. In the Thin W image, one would expect the detection of four distinct ribbons and all the four ribbons have been detected successfully as shown in in Figure 4.25.



Figure 4-25 Thin W ribbon detection

Ribbon detection for the remaining hand-drawn images performs as one might expect: if the corresponding lines are detected, then a ribbon is detected. Non-detection only occurs when there is incomplete line evidence.

The results of applying the ribbon detector to the set of scene images show varying levels of success. While most of the obvious ribbons are detected, some surprising detections are made as well. Figure 4-26 displays some of the more promising detection results. In Figure 4-26 (a) Nairobi 1, the strong type 1 ribbon for is clearly detected, as are the minarets in Figure 4-26 (b) Taj Mahal. The ribbons for all the minarets in Figure 4-26 (b) Taj Mahal have been detected successfully but the main door and some of the window's ribbons are not detected. The most encouraging example can be found in the Figure 4-26 (c) Lincoln scene. In this image, vertical ribbons of both types are detected.



(a) Nairobi 1



(b) Taj Mahal



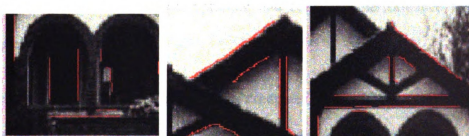
(c) Lincoln scene

Figure 4-26 (a), (b) and (c) Shows the detected ribbons



Figure 4-27 Nairobi 2 showing some successful ribbon detection

The scene in Figure 4-27, Nairobi 2, has strong ribbon evidence in the crossbeams, and one would hope that they would be detected, but this is not the case. The problem is that some of the lines in question were never detected. Other interesting ribbons are displayed in the Figure 4-28.



(a) (b) (c)  
Figure 4-28 Showing some successful ribbons of Nairobi 2

## 5. Image Representations

There are many types of features such as global, local and relational features that can be used for object recognition and matching. Global features are region based and can be obtained either by considering all points inside the region, or only those points on the boundary of the region. In any of the case, the purpose is to find the locations, intensity characteristics, color, and spatial relations of these points. Local features usually represent a small area of a region. Some local features are corners, and boundary segments. Relational features are based on relative position of different entities, such as distance between features, relative angle, orientations and precincts. These features are very important in object recognition as even change in lighting conditions, view angle and noise will not greatly influence these features. Line segments and orientation were used for object recognition by B.Huet et al. [35]. We will use relative features to define our data structures in order to get optimum and consistent performance from graph matching algorithm.

Graphs are important data structures for computer vision. They are widely used to represent the neighborhood relations that exist in an image. Relational graph matching with model based segmentation for human detection was used by Ozer et al [36] for the decision of human presence in the image as well as posture recognition. A similarity based aspect graph approach was used by Christopher et al [37] to recognize 3D objects. Section 5.1, contains the basic definitions from the graph theory that will be used in the rest of this Chapter. Section 5.2 discusses the line structure representation, Section 5.3

discusses the corner structure representation and Section 5.4 is devoted to ribbon structure representation.

## 5.1 Basic Definitions

The definitions from graph theory that are contained in this section can be found in standard textbooks on graph theory such as [38,39,40,41].

Definition 5-1 A graph  $G=(V,E)$  consists of two sets: a finite set  $V$  of elements called **vertices** and finite set of elements called **edges**. Each edge creates a binary relation between a pair of vertices.

Definition 5-2 The vertices  $v_i$  and  $v_j$  associated with an edge  $e$  are called the **end vertices** of  $e$ :  $e$  is said to be incident to its end vertices. An edge is denoted as  $e=(v_i, v_j)$

Definition 5-3 Two **edges** are **adjacent** if they have a common end vertex.

Definition 5-4 Two **vertices** are **adjacent** if they are the end vertices of same edge.

Definition 5-5 A **path**  $K$  in a graph is finite alternating sequence of vertices and edges, such that

- Vertices  $v_{i-1}$  and  $v_i$  are the end vertices of the edge  $e_i$ , where  $1 \leq i \leq k$
- All edges are distinct
- All vertices are distinct

Vertices  $v_0$  and  $v_k$  are called **end vertices of the path**, and we refer to it as  $v_0$ - $v_k$  path. The number of edges in the path is called **length of the path**.

**Definition 5-6** A graph  $G$  is connected if there exists a path between every two vertices in  $G$ .

**Definition 5-7** Consider a graph  $G=(V, E)$ .  $G'=(V', E')$  is a **sub-graph** of  $G$  if  $V'$  and  $E'$  are, respectively, sub set of  $V$  and  $E$  such that an edge  $(v_i, v_j)$  is in  $E'$  only if  $v_i$  and  $v_j$  are in  $V'$ .

**Definition 5-8** An  $n$ -vertex graph with  $n(n-1)/2$  edges is a **complete** graph.

**Definition 5-9** The adjacency matrix of an  $n$ -vertex graph  $G = (V, E)$  is an  $n \times n$  matrix  $A$ . Each element of  $A$  is either zero or one. If  $G$  is an undirected graph and  $V=\{1,2,3,\dots,n\}$  then the elements of  $A$  are defined as follows:

$$A(i, j) = \begin{cases} 1 & \text{if } (i, j) \in E \text{ or } (j, i) \in E \\ 0 & \text{otherwise} \end{cases}$$

### 5.1.1 Adjacency Matrix

Since I used MATLAB 6.12 for my implementation, it is, therefore important to highlight some of the MATLAB features. MATLAB as defined by Math works group [42] “MATLAB is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It



allows both *programming in the small* to rapidly create quickly and dirty throw away programs, and *programming in the large* to create complete large and complex application programs”. MATLAB does not support pointers [42], therefore, it is hard to implement a graph data structure in its classic manner. MATLAB is well known for matrix manipulation and operations. The graph data structures can also be represented in matrix form (Adjacency Matrix). Therefore, I will refer to an adjacency matrix as a graph data structure.

Now let see some examples so that it's clearer.

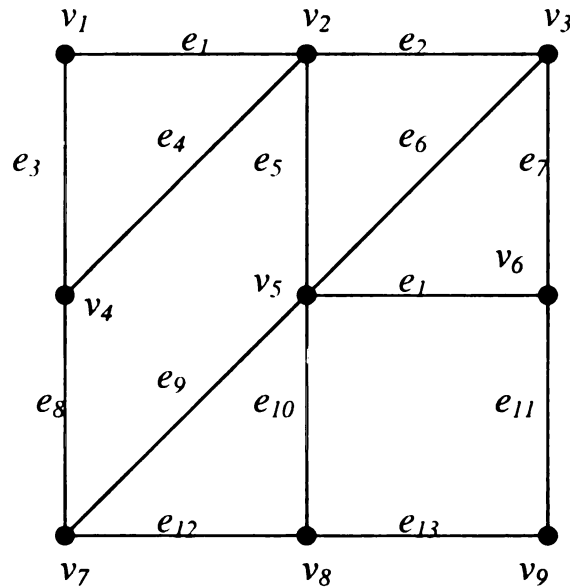


Figure 5-1 Pictorial representation of a graph  $G(V,E)$ .

Figure 5-1 shows the example of a graph. Vertices are represented by black spots ( $\bullet$ ), and edges by straight lines. The graph in Figure 5-1 has nine vertices:  $v_1$  to  $v_9$ . Therefore, the adjacency matrix will have nine rows and nine columns. The adjacency matrices of Figure 5-1 are shown in Figure 5-2 and Figure 5-3.

$$G = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Figure 5-2 Adjacency Matrix of Figure 5-1

$$G = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 5-3 Adjacency Matrix of Figure 5-1

Figure 5-2 and Figure 5-3 are adjacency matrices of Figure 5-1. The problem with adjacency matrices is, it takes  $n \times n$  memory space and if we want to have sub pixel accuracy such as distance between two lines or a corner point, then we have to declare float type and float takes 8 bytes of memory space to store each value. So the total memory space that matrix will occupy in the above-mentioned case is  $8n^2$ , where  $n$  is the number of vertices. To reduce this we have to, first declare non-float type such as unsigned character type, which takes one byte of memory space and value ranges from 0 to 255, and secondly we can eliminate the lower diagonal of Figure 5-2 as it is identical to the upper diagonal. To further reduce memory storage we can convert the adjacency matrix shown in Figure 5-3 into a sparse matrix [40,42].

## 5.2 Line Structure Representation.

I have divided a line structure into two groups depending upon its properties, descriptor and relationship. Descriptors of a line represent different properties of line such as line slope and edge gradient, and relationships represent distance between line segments and line orientations.

For each of the descriptors and relationships we have to calculate the spatial relationship as described in detail in subsequent sections. A descriptor is a good candidate for vertex and relationship is a good candidate for an edge in a graph. A pictorial representation of line structure is as shown in Figure 5-4

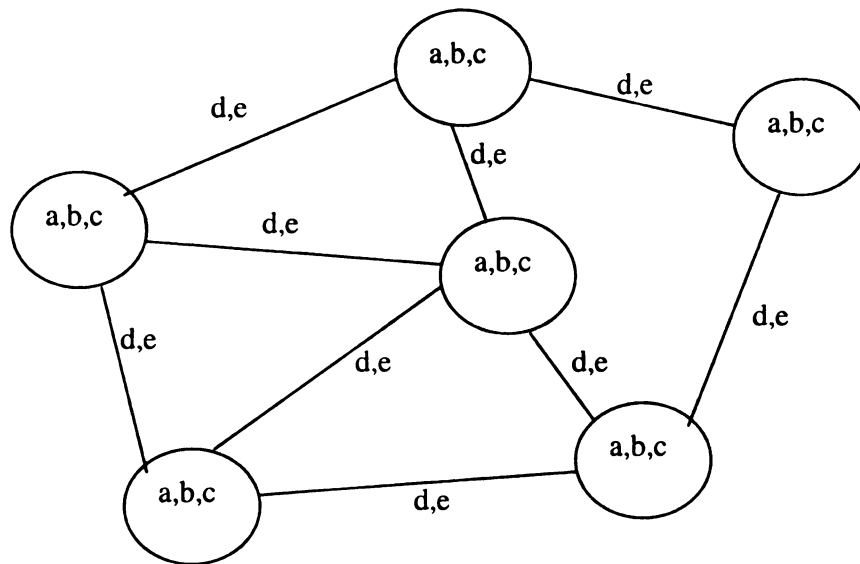


Figure 5-4 Pictorial representation of line structure.

Figure 5-4 'a' 'b' and 'c' are properties of vertices and represent line length, line slope, edge gradient respectively and 'd' and 'e' are properties for an edge of the graph and represent distance between two lines and relative line orientation respectively.

### 5.2.1 Line Slope.

Line slope is used as one of the attributes of line segment.

### 5.2.2 Edge Gradient.

Edge gradient is one of the primitives in computer vision. It has been used as a property of lines. Since I am using a graph-matching algorithm to match different lines and line structures to achieve accurate matching results, edge gradient is used as one of the descriptors of a line. Edge gradient normally changes with the changes in lighting conditions, therefore, for consistent matching results a high threshold is used for comparison.

To compute the average edge gradient Equation 5.3 is used.

$$\mathcal{L}_{iG} = \frac{\sum_{n=m}^{S-3} A(n) + \sum_{n=m}^{E-3} A(n)}{LineLength - 6} \quad 5.3$$

where  $\mathcal{L}_{iG}$  represents the average edge gradient, 'i' represents the line number, 'S' and 'E' represent start and end points of a line respectively, 'm' is a midpoint of the line, 'n' is a pair of row and column coordinates of the line and 'A' is the gradient of the edge. The Sobel gradient operator normally distorts the gradient at the end-points of an edge. The degree at which it degrades the gradient depends on the size of the operator. I am using 5 x 5 Sobel gradient operator to get more accurate and consistent average gradient, average gradient is computed between end-points.

Figure 5-5 (a) is a test image and Figure 5-5 (b) shows the line segments resulting from the Hough Transform and line joining routine.

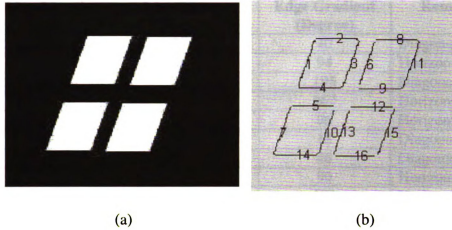


Figure 5-5 (a) Test image, (b) Lines extracted from (a)

In Table 5-1 descriptors such as line lengths, line slopes and edge gradients are shown for Figure 5-5 (b). Line lengths for the horizontal lines are 30 or 32, whereas their line slope is  $0^\circ$ . The only thing that differentiates among these horizontal lines is their edge gradient. The edge gradient divides these lines into two distinct groups. The first group has the edge gradient less than  $100^\circ$  and the second group has the edge gradient more than  $200^\circ$ . Similarly one can see diagonal lines. Line lengths range from 42 to 48 and line slope ranges from  $73^\circ$  to  $79^\circ$ . The differences in line lengths and line slopes are less than 10 pixels and  $10^\circ$  respectively, which is less than the threshold that is set for the matching (thresholds for these descriptors are discussed in Chapter 6). Here also edge gradient plays an important role by dividing these lines into four groups. Diagonal lines, 1 and 7, 6 and 13, line 3, 10, 11 and 15 are grouped together. Similarly Horizontal lines 2, 5, 8 and 12 are grouped together and lines 4, 9, 14, and 16 are grouped together.

Line #	Line Length (Pixels)	Line Slope (Degree)	Edge Gradient (Degree)	Remarks
1	46	77	40	Diagonal
2	32	0	94	Horizontal
3	48	79	195	Diagonal
4	32	0	267	Horizontal
5	32	0	96	Horizontal
6	42	73	18	Diagonal
7	47	77	46	Diagonal
8	32	0	90	Horizontal
9	32	0	270	Horizontal
10	44	74	200	Diagonal
11	48	79	190	Diagonal
12	32	0	95	Horizontal
13	43	75	25	Diagonal
14	32	0	269	Horizontal
15	46	74	198	Diagonal
16	30	0	267	Horizontal

Table 5-1 Line descriptors

### 5.2.3 Distance between Line Segments.

Distance between lines is another important parameter that can refine our matching process. To compute the distance, Equation 5.4 is used.

$$\mathcal{L}_{ijD} = \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2} \quad i \neq j, j \in [i - 10, i + 10] \quad 5.4$$

where  $\mathcal{L}_{ijD}$  represents the line distance adjacency matrix, 'i' and 'j' represents the line numbers, 'D' represents the distance between line 'i' and line 'j' in pixels and  $(x_i, y_i)$  and  $(x_j, y_j)$  are the mid points of line 'i' and 'j' respectively.

### 5.2.4 Relative Line Orientation.

Line orientation means relative positions of the lines, such as 'left', 'right', 'up' and 'down'. To relate their positions, midpoints are used.

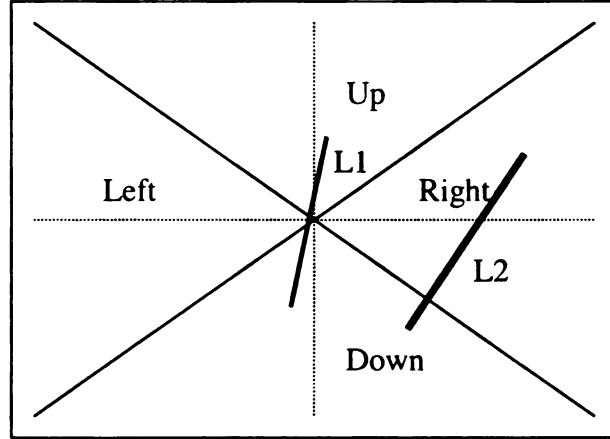


Figure 5-6 Line Orientation

In Figure 5-6, 360° is divided into four equal angles of 90° each. Up is from 45° to 135°, left is from 135° to 225°, down from 225° to 315° and right from 315° to 45°. In Figure 5-6 line L2 is right of line L1.

$$\mathcal{L}_{ijO} = \mathbf{K}(L_i, L_j) \quad i \neq j, j \in [i - 10, i + 10] \quad 5.5$$

where  $\mathcal{L}_{ijO}$  is a line orientation adjacency matrix, 'i' and 'j' are line segment label numbers and  $\mathbf{K}$  is a function that returns the orientation of line 'i' with respect to line 'j'.

### 5.2.5 Behavior of Line Descriptors and Relationships

Predicted behavior of line descriptors and relationships are shown in Table 5-2. However, the behavior of these descriptors and relationships will be analyzed and reported in Chapter 7.

<b>Descriptors/ Relationships</b>	<b>Rotation Invariant</b>	<b>Scale Invariant</b>	<b>Illumination Invariant</b>	<b>Occlusion Invariant</b>
Line Slope	No	Yes	Yes	Yes
Edge Gradient Direction	No	Yes	No	Yes
Edge Gradient Magnitude	Yes	Yes	No	Yes
Distance between Line Segments	Yes	No	Yes	No
Relative Line Orientation	No	Yes	Yes	Yes

Table 5-2 Behaviour of Line Descriptors and Relationships

### 5.3 Corner Structure Representation.

A geometric descriptor of the corner structure is 'angle of a corner', and relations that a corner makes with other corners are:-

- Distance between Corners.
- Corner's Orientation.
- Corner's Precincts.
- Angle between Corners.

The attributes of corner structure are shown in Figure 5-7.

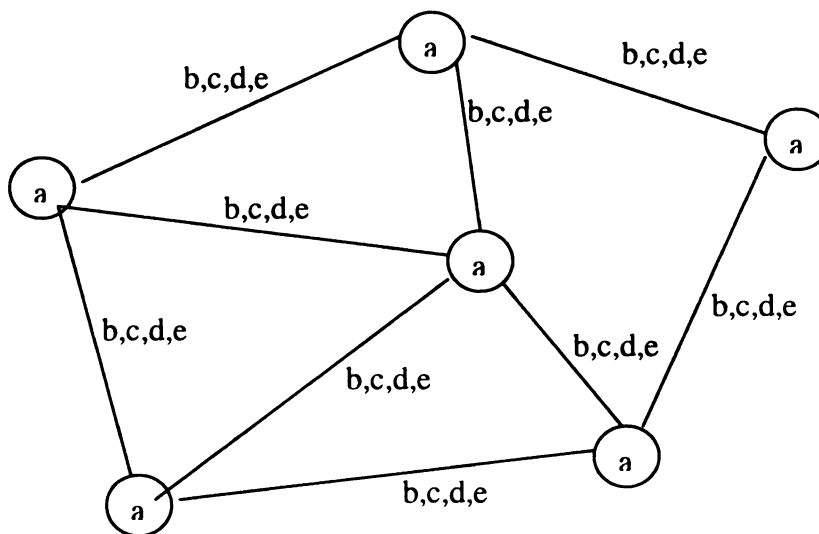


Figure 5-7 Pictorial Representation of Corner Structure



In Figure 5-7 'a' is a property of vertices and represents the angle of a corner and 'b', 'c', 'd' and 'e' are properties of the edge of the graph and represent distance between two corners, corner orientation, corner precincts and angle between corners respectively.

### 5.3.1 Angle of a Corner.

The procedure to detect a corner and compute its angle was defined in Section 4.3. The results obtained from Equation 4.5 are stored in the adjacency matrix  $\mathcal{T}_{ijA}$ .

### 5.3.2 Distance between Corners.

The distance between corners is an important relationship that differentiates between corners of doors and windows and those of a building. As one can see in the Figure 5-8 (a) and (b), corners of a door and windows are relatively closer than corners of the building. We require that objects should match under small changes in the view and resolution. For instance Figure 5-8 (a) and (b) show the same building but there is change in the view and scale. Here the resolution of the image is same.

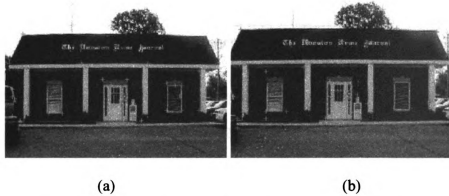


Figure 5-8 Corners detected by the corner detection algorithm

To compute the distance between different corners Equation 5.6 is used.

$$\mathcal{T}_{ijD} = \mathbf{K}(\mathcal{T}_{iA}, \mathcal{T}_{jA}) < \delta \quad i \neq j, j \in [i - 10, i + 10] \quad 5.6$$

where  $K$  is function that computes the distance between corners  $\mathcal{C}_i$  and  $\mathcal{C}_j$  and returns Euclidian distance between two corners. If this distance is less than the threshold distance  $\delta$  then this distance is stored in adjacency matrix  $\mathcal{C}_{ijD}$

### 5.3.3 Corner's Orientation.

As already discussed in Section 5.2.4 for line orientation, corner orientation is also used to refine the matching results. Unlike the distance between corners, corner orientation will not often be affected by view and resolution.

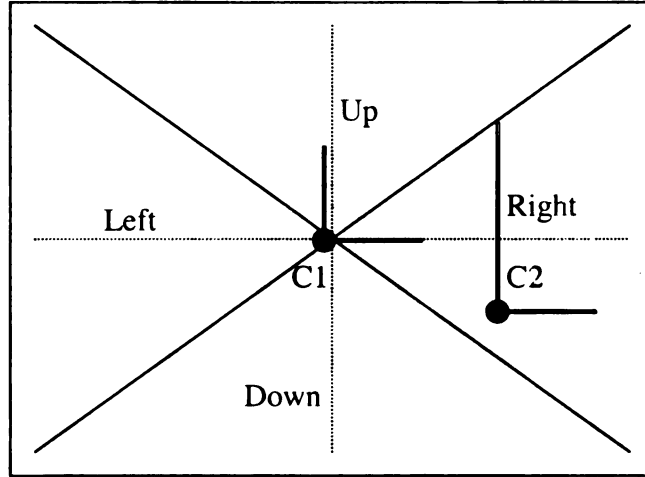


Figure 5-9 Corner Orientation

In Figure 5-9,  $360^\circ$  is divided into four equal angles of  $90^\circ$  each. Up is from  $45^\circ$  to  $135^\circ$ , left is from  $135^\circ$  to  $225^\circ$ , down from  $225^\circ$  to  $315^\circ$  and right from  $315^\circ$  to  $45^\circ$ . In Figure 5-9 corner C2 is right of corner C1.

$$\mathcal{C}_{ijO} = K(\mathcal{C}_i, \mathcal{C}_j) \quad i \neq j, j \in [i - 10, i + 10] \quad 5.7$$

where  $\mathcal{C}_{ijO}$  is a corner orientation adjacency matrix, ' $i$ ' and ' $j$ ' are corner label numbers and  $K$  is a function that returns the orientation of corner  $\mathcal{C}_i$  with respect to  $\mathcal{C}_j$ .

### 5.3.4 Corner's Precincts.

One of the most important relationships among corner structures is represented by the corner precincts. Precincts mean boundary, the boundary of a corner that arms (lines) make. In Figure 5-10 precincts are shown with extended dotted lines. We assumed that any two corners of a window or door would close each other in their boundary as in Figure 5-10 corner C1 and corner C2 close each other in their bounds.

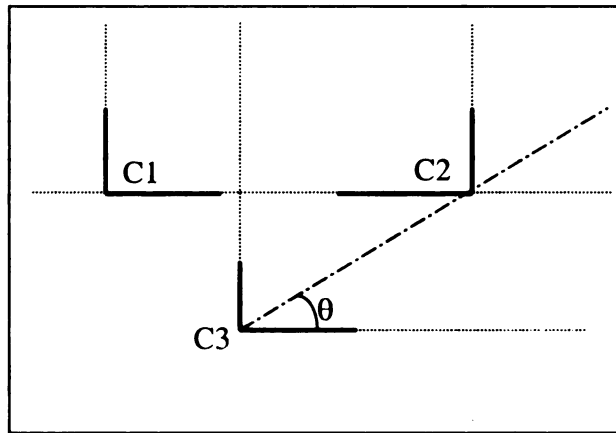


Figure 5-10 Example of corner bounds

In Figure 5-10, C1, C2 and C3 are three corners. Dark thick lines show the line segments and dotted lines show the bounds of these corners. C1 and C2 bound each other, whereas C3 bounds C2 but C2 does not bound C3.

$A_1 \rightarrow B$		$A_2 \rightarrow B$		Decision
Dot Product Angle	Cross Product Sign	Dot Product Angle	Cross Product Sign	
$\theta_1$	+1	$\theta_2$	+1	Outside
$\theta_1$	+1	$\theta_2$	-1	Inside
$\theta_1$	-1	$\theta_2$	+1	Inside
$\theta_1$	-1	$\theta_2$	-1	Outside

Table 5-3 Decision table for arm's orientation of a corner

Corner A	Corner B	Type	Code
Outside	Outside	> <	1
Outside	Inside	> >	2
Inside	Outside	< <	3
Inside	Inside	< >	4

Table 5-4 Type of bounds corner is making

Corner point 'A' and 'B' can be any corners shown in the Figure 5-10.  $A_1$  and  $A_2$  are the distant endpoints of arms of corner 'A'. Suppose 'A' is 'C1' and 'B' is 'C2'. Now first we have to decide if corner point 'B' is inbound to corner point 'A' or not. Dot product and cross product are taken between, corner point 'B' and arm ' $A_1$ '. Corner point 'A' is taken as origin. Similarly, dot product and cross product are taken between, corner point 'B' and arm ' $A_2$ '. If both the signs of cross product are same, the corner point 'B' is out side the bounds of 'A', if signs are opposite, then corner point 'B' is inbound (see Table 5-3). Similarly, we can find, if corner point 'A' is inside the bounds of corner point 'B' or not. After getting both the results, Table 5-4 is consulted for the type of bounds. In this example, both the signs are opposite for 'C2'; therefore, corner 'C2' is inbound to corner 'C1' and similarly 'C1' is inbound to 'C2'.

### 5.3.5 Angle between Corners.

The angle between two corners also plays an important role in matching the corner points. C2 is making  $0^\circ$  with C1 in Figure 5-10, whereas C1 is making  $180^\circ$  with C2 and C1 and C2 relate with code type 4. From this we can conclude that if the difference between the two angles is  $180^\circ$  and code type is 4, then it is strong evidence that these corners are of some windows or doors.

$$\mathcal{T}_{ijC} = \mathbf{K}(\mathcal{T}_i, \mathcal{T}_j) \quad i \neq j, j \in [i - 10, i + 10] \quad 5.7$$

where  $\mathcal{T}_{ijC}$  is a adjacency matrix, ' $i$ ' and ' $j$ ' are corner label numbers and  $\mathbf{K}$  is a function that returns the angle between corner  $\mathcal{T}_i$  and  $\mathcal{T}_j$ . function  $\mathbf{K}$  uses dot product to calculate the angle.

### 5.3.6 Behavior of Corner Descriptors and Relationships

Predicted behavior of corner descriptors and relationships are shown in Table 5-5. However, the behavior of these descriptors and relationships will be analyzed and reported in Chapter 7.

Descriptors/ Relationships	Rotation Invariant	Scale Invariant	Illumination Invariant	Occlusion Invariant
Angle of a Corner	Yes	Yes	Yes	Yes
Relative Corner's Orientation	No	Yes	Yes	Yes
Distance between Corners	Yes	No	Yes	Yes
Corner's Precincts	Yes	Yes	Yes	Yes
Angle between Corners	Yes	Yes	Yes	Yes

Table 5-5 Behavior of Corner Descriptors and Relationships

### 5.4 Ribbon Structure Representation.

The detection of ribbons is an important step towards the ultimate goal of image matching using the graph-matching algorithm. The ribbon structure required by the matching algorithm mainly uses descriptors of the line structure because a ribbon is defined as parallel lines whose edge gradients differ by  $180^\circ$ , are in the same relative

position and at least some of the portion of lines overlap each other. The descriptors used for ribbon structure are:-

- Width of a Ribbon.
- Line Slope.
- Edge Gradient.
- Type of Ribbon.

Except for 'type of ribbon' and 'width of a ribbon' the rest of the above descriptors are the same as already explained in Section 5.2 and will not be discussed here. The relationship between ribbons are:-

- Distance between Ribbons.
- Ribbon's Relative Orientation.

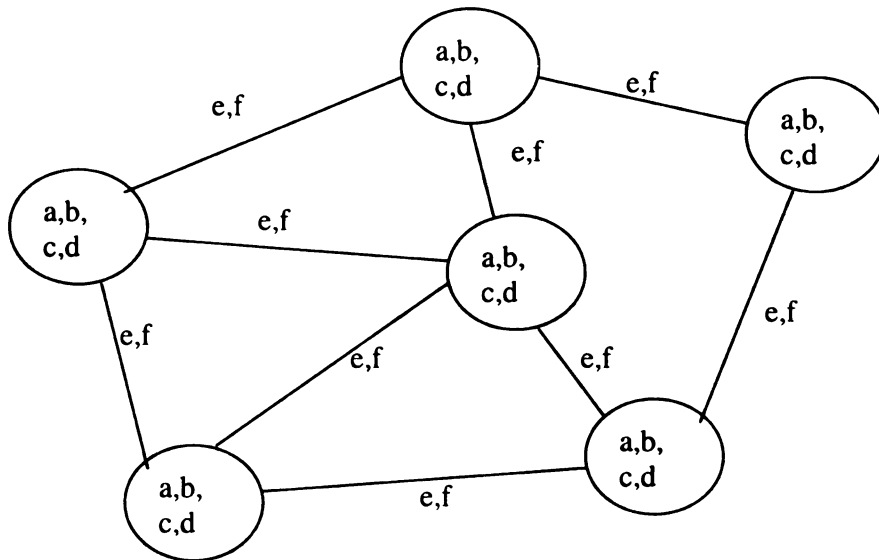


Figure 5-11 Pictorial Representation of Ribbon Structure

In Figure 5-11 'a' 'b' 'c' and 'd' are the properties of vertices and represents width of a ribbon, line slope, edge gradient and type of ribbon respectively and 'e' and 'f' are properties for edge of the graph and represent distance between two ribbons and ribbon's orientation respectively.

#### 5.4.1 Width of a Ribbon.

Figure 5-12 shows two ribbons 'A' and 'B' with the shaded area between two lines. In ribbon 'A' and 'B' some of the portion of lines are overlapping each other. The difference in edge gradient is  $180^\circ$ . In Figure 5-12, ribbon width 'w' is the shortest distance between two parallel lines.

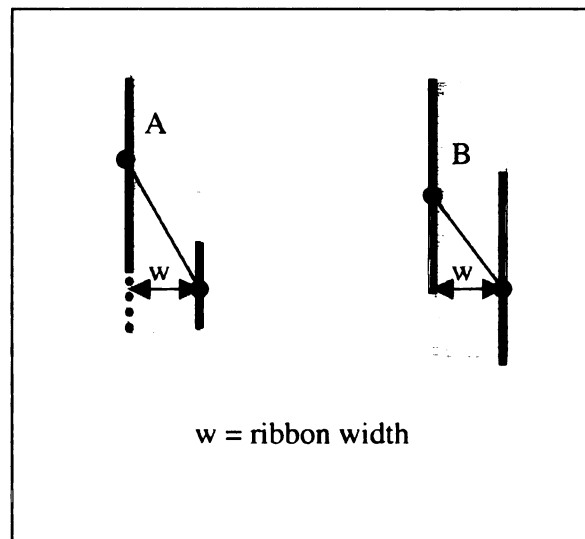


Figure 5-12 Ribbon Width

#### 5.4.2 Types of a Ribbon.

The types of a ribbon are already described in Section 4.4.2.



### 5.4.3 Relative Distance between Ribbons.

Relative distance between two ribbons can be defined as the shortest distance between midpoints of ribbons. In Figure 5-13 distance 'd' is the shortest distance between two ribbons.

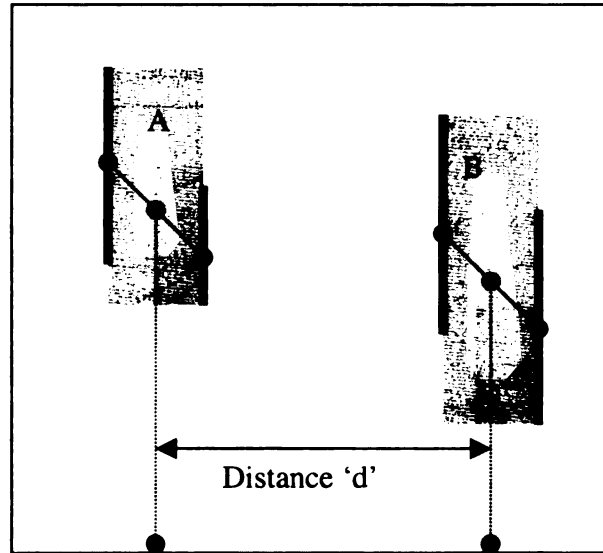


Figure 5-13 Distance between ribbon A and ribbon B

Equation 5.8 shows how to generate the adjacency matrix for distance between ribbons.

$$R_{ijC} = \mathbf{K}(R_i, R_j) \quad i \neq j, j \in [i - 10, i + 10] \quad 5.8$$

where  $R_{ijC}$  is an adjacency matrix, 'i' and 'j' are ribbon label numbers and  $\mathbf{K}$  is a function that returns the distance between ribbon  $R_i$  and  $R_j$ .

#### 5.4.4 Ribbon's Orientation.

Like corners and line orientation, orientation of a ribbon also plays vital role in graph matching as it helps in narrowing the match. Ribbon orientation means position of the ribbons relative to other adjacent ribbons, in terms of positions such as 'left', 'right', 'up' and 'down', taking midpoints of each ribbon as reference.

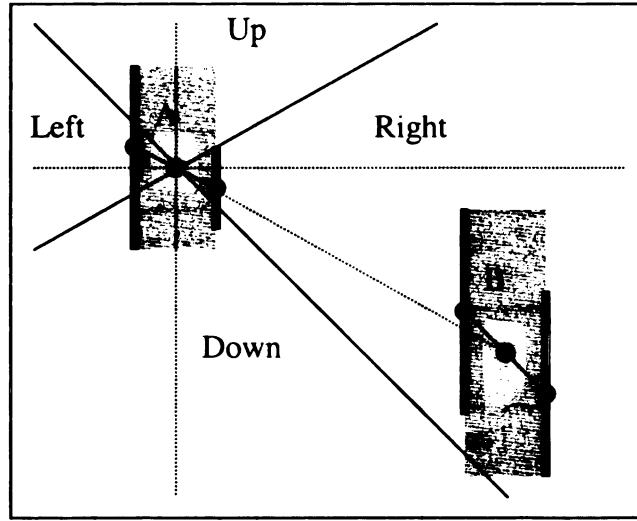


Figure 5-14 Ribbon Orientation

In Figure 5-14  $360^\circ$  is divided into four equal angles of  $90^\circ$  each. Up is from  $45^\circ$  to  $135^\circ$ , left is from  $135^\circ$  to  $225^\circ$ , down from  $225^\circ$  to  $315^\circ$  and right from  $315^\circ$  to  $45^\circ$ . In Figure 5-14 ribbon A is right of ribbon B.

$$R_{ijO} = \mathbf{K}(R_i, R_j) \quad i \neq j, j \in [i - 10, i + 10] \quad 5.9$$

where  $R_{ijO}$  is a ribbon orientation adjacency matrix, ' $i$ ' and ' $j$ ' are ribbon label numbers and  $\mathbf{K}$  is a function that returns the orientation of ribbon  $R_i$  with respect to  $R_j$ .

#### 5.4.5 Behavior of Ribbon Descriptors and Relationships

Predicted behavior of ribbon descriptors and relationships are shown in Table 5-6. However, the behavior of these descriptors and relationships will be analyzed and reported in Chapter 7.

Descriptors/ Relationships	Rotation Invariant	Scale Invariant	Illumination Invariant	Occlusion Invariant
Width of a Ribbon	Yes	No	Yes	Yes
Line Slope	No	Yes	Yes	Yes
Edge Gradient	No	Yes	No	Yes
Type of Ribbons	Yes	Yes	Yes	Yes
Distance between Ribbons	Yes	No	Yes	Yes
Relative Ribbon Orientation	No	Yes	Yes	Yes

Table 5-6 Behavior of Ribbon Descriptors and Relationships

## **6. Graph Matching Algorithm**

In most of the image retrieval algorithms, researchers have been using only one of these image features in isolation such as lines and corners with only two attributes. Lines with two of its attributes, line length and gradient, was used by Benoit Huet and Edwin R. Hancock [43] for sensitivity analysis for the problem of recognizing line patterns from large structural libraries. For object based image retrieval Yi Tao and William I. Grosky [44] also used a single feature (corner) with two attributes, location and color histogram at the centre of mass. This thesis develops a graph-matching algorithm using three structures, (1) lines with four attributes (2) corners with five attributes and (3) ribbons with six attributes. For details of these features and their attributes see Chapter 5 and Table 3-1. Although graph matching is comparatively slower than other techniques used for matching, at the same time the accuracy in graph matching is superior. In this chapter I will try to prove that with such a large number of feature points, their attributes and relationships graph matching is robust.

The matching algorithm is divided into three routines; each routine is used for matching one of the features. This chapter is divided into four sections. The overall methodology of graph matching is discussed in Section 1. In the remaining three sections, each matching algorithm is discussed along with the results obtained.

### **6.1 Graph Matching**

The term graph matching refers to the process of comparing two or more graphs with each other. In our case, we are matching a query graph with the database graph. Now before explaining the matching algorithm some of the basic terms used in graph matching

are introduced. If  $G_1$  is the graph for a database image and  $G_2$  is the graph for query image ( $G_1 = \{ V_1, E_1 \}$  and  $G_2 = \{ V_2, E_2 \}$ ), then we can match graph  $G_1$  with  $G_2$  and  $G_2$  with  $G_1$ . Since the image is represented by three sets of structures (lines, corners and ribbons), a separate graph  $G$  is defined for each structure. For details refer to Chapter 5.

## 6.2 Proposed Graph Matching Algorithm.

The algorithm shown in Algorithm 6-1 is a general algorithm that will give the framework for how the graph matching will be done in subsequent sections.

```

Input: Graph G1 and Graph G2
Output: List of matching Vertices
For each vertex  $V_i$  of graph G2
Find matching vertex  $V_j$  of graph G1 (by attributes)
  If (True)
    For all adjacent edges of G2 and G1
    Compare vertices
    If (True)
      Increment Vote
    End If
  End For
  Save vertices of G1, G2 and Votes
End If
End For

```

Algorithm 6-1 General graph matching algorithm

The algorithm takes two inputs, graph  $G_1$  and graph  $G_2$ , as graph data structures and compares each vertex of graph  $G_1$  with every vertex of graph  $G_2$  and return an array of matched vertices. The complexity of the algorithm is  $n \times m$ , where  $n$  is the number of vertices of  $G_1$  and  $m$  is the vertices of  $G_2$ . In this section I proposed the general algorithm for graph matching problems. Now using the same concept, I will develop

particular graph matching algorithms that will match the images using specific image structures.

### 6.3 Line-Graph Matching.

The particular goal in this section is to incorporate the line structure that is defined in Chapter 5 into the matching algorithm. Before we proceed to experiment with the line matching, it is important to understand the way the matching algorithm works and some of the notation that is being used in the subsequent paragraphs.

If  $G = (V, E)$  is a graph then  $V = \{ \mathcal{I}_{iG}, \mathcal{I}_{iS} \}$  are the descriptors of vertices and  $E = \{ \mathcal{I}_{ijO}, \mathcal{I}_{ijD} \}$  are the relationships that vertices make with each other;  $\mathcal{I}_{iG}, \mathcal{I}_{iS}$  represent the edge gradient and line slope respectively, and  $\mathcal{I}_{ijO}, \mathcal{I}_{ijD}$  represent the line orientation and distance between two line segments, where 'i' and 'j' represent the vertices and  $i \neq j$ .

#### 6.3.1 Matching Algorithm

Algorithm 6-2 shows the line matching routine. It requires two inputs, graph G1 and graph G2, as line structures and returns the line-matching result. For comparison of vertices Algorithm 6-2 calls sub-routine `match_vertex(.)` given in Algorithm 6-4. `match_vertex(.)` compares all descriptors of vertices  $V_i$  of graph G1 with every descriptor of vertex  $V_j$  of graph G2. If a match found, it returns '1'; else it returns '0'. Based on this result, `match_edge(.)` sub-routine Algorithm 6-3 is called. The `match_edge(.)` subroutine compares all the descriptors of edges  $E_{ij}$  of graph G1 with every descriptor of edges  $E_{ij}$  of graph G2. Now if any edge of  $V_i$  matched with  $V_j$ , `match_edge(.)` calls the `match_vertex(.)` sub-routine for matching the other end of the edge, and on successful

vertex match the algorithm increments the variable 'vote'. The loop will continue until it compares all the edges of  $V_i$  with  $V_j$ . At the end it returns the 'vote' it gathered for successful matches. Getting control from the `match_edge(.)` sub-routine, the `LineMatchingRoutine(.)` stores lines  $L_i$  and  $L_j$  in the 'match' array, provided the vote is greater than zero.

```

//G=(V , E) where V={  $\mathcal{L}_{iL}, \mathcal{L}_{iG}, \mathcal{L}_{iS}$  } and E={  $\mathcal{L}_{ijO}, \mathcal{L}_{ijD}$  }
LineMatchingRoutine( G1 , G2 )
{
n1=size of graph G1
n2=size of graph G2
vote=0
k=1
for i=1 to n1
  for j=1 to n2
    if(match_vertex ( Vi , Vj) )// match vertex Vi with Vj
      vote=match_edge(Ei( i , : ) , Ej( j , : ) , i , j )
      if vote > 0
        match( k , : ) = [Li Lj vote] //Array that store the matching.
                                results
        k=k+1
      end if
    end if
  end for
end for

FinalMatch = line_sel_by_vote( match )
return (FinalMatch)
}

```

Algorithm 6-2 Line Matching Routine



```

e1 is edges of  $V_i$  of  $G1$ 
e2 is edges of  $V_j$  of  $G2$ 
I is number of vertex  $V_i$ 
J is number of vertex  $V_j$ 
match_edge( e1 , e2 , I , J )
{
  vote=0
  for i=I-5 to I+5 // compares maximum of 10 edges
    for j=J-5 to J+5 // compares maximum of 10 edges
      if ( $\mathcal{L}_{iO1} = \mathcal{L}_{jO2}$  &  $|\mathcal{L}_{iD1} - \mathcal{L}_{jD2}| < \delta n$ )
        if ( match_vertex (  $V_i$  ,  $V_j$  ))
          vote=vote+1
        end if
      end if
    end for
  end for
  return (vote)
}

```

Algorithm 6-3 Edge matching sub-routine.

```

Match_vertex( $v_1$  ,  $v_2$  )
{
  if ( $|\mathcal{L}_{1S} - \mathcal{L}_{2S}| < \delta 1$  &  $|\mathcal{L}_{1G} - \mathcal{L}_{2G}| < \delta 2$ ) // compare the
                                                    //vertices
    return (1)
  else return (0)
}

```

Algorithm 6-4 Vertex matching sub-routine

```

line_sel_by_vote( LinesMatch )
{
x = size of LinesMatch
for i=1:5
    j=1
    while j<x
        if (LinesMatch(j,1) == LinesMatch(j+1,1)) // compare the line # for G1
            if (LinesMatch(j,3) > LinesMatch(j+1,3)) // compare the votes
                LinesMatch(j+1,:) = [ ] // delete the next line
            else LinesMatch(j,:) = [ ] // delete the line
            end
            x=x-1
        end
        j=j+1
    end
end
LinesMatch=sortrows(LinesMatch,[2]) // sort array by row # 2
x =size of LinesMatch
for i=1:5
    j=1
    while j<x
        if (LinesMatch(j,2)= LinesMatch(j+1,2)) // compare the line # for G2
            if (LinesMatch(j,3) > LinesMatch(j+1,3)) // compare the votes
                LinesMatch(j+1,:) = [ ]; // delete the next line
            else LinesMatch(j,:) = [ ]; // delete the line
            end
            x=x-1;
        end
        j=j+1;
    end
end
return (LinesMatch )
}

```

Algorithm 6-5 Line selection sub-routine

After completing the comparison between all the lines of graph G1 and G2, sub-routine 'line\_sel\_by\_vote(.)' given in Algorithm 6-5 is called. This sub-routine examines the 'match' array and if any line has more then one match it will delete all those with non maximum votes and will return the final line matching results to the parent routine. Thresholds used in the algorithms are given in Table 6-1

Algorithms	Thresholds	Explanation	Default Setting
Algorithm 6.3	$\delta n$	Distance between two lines	10 pixels
Algorithm 6.4	$\delta l$	Slope of the line	$10^\circ$
Algorithm 6.4	$\delta 2$	Edge gradient of the line	$30^\circ$

Table 6-1 Different threshold used in Line Structure Graph Algorithms

### 6.3.2 Matching Results for Line Segments

In this section, we will investigate the robustness and quality of the graph-matching algorithm. Experiments were conducting on an image database consisting of 97 images of different resolutions. In some of the images Gaussian noise was added. Images and their resolutions are given in Appendix A. The images in the database are divided into six groups depending upon their shapes and structural types. Four query images were taken from these groups and the graph-matching algorithm for lines was applied. The results are shown in Figure 6-2, Figure 6-3 and Figure 6-4. The first image shows the query image and the remaining images are the result from the graph matching algorithm. Retrieved images are shown in descending order from left to right and up to down depending upon percentage of similarity.

In Figure 6-1, hand drawn Taj was given as query image. The graph-matching algorithm for line structure successfully retrieved all the images of hand drawn Taj, even

those with the Gaussian noise and occlusion. The Taj Mahal images were also retrieved along with some other images. This was the most successful experiment as out of eight hand drawn Taj images, seven were successfully retrieved. In Figure 6-2 the Taj Mahal was given as query. In this experiment, the algorithm successfully retrieved four out of nine Taj Mahal images. The good thing about this experiment is, it retrieved Taj Mahal images of different scale and resolution, view and noise level. Results from experiment 3 are shown in Figure 6-3. In this experiment, a Small Building was given as query and three images were successfully retrieved from its group. In experiment 4 the query image Spartan Village Apartments, was from the same group of Small Buildings. This time the algorithm retrieved six relevant images from the same group.



Figure 6-1 Query image Drawn Taj and retrieved images

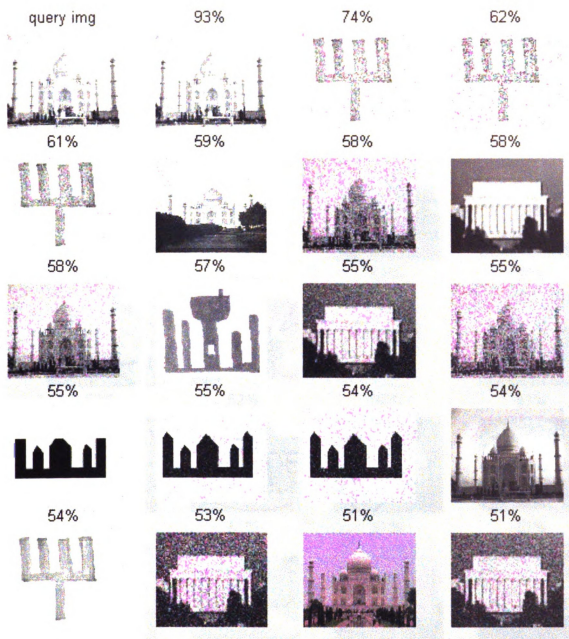


Figure 6-2 Query image Taj Mahal and retrieved images

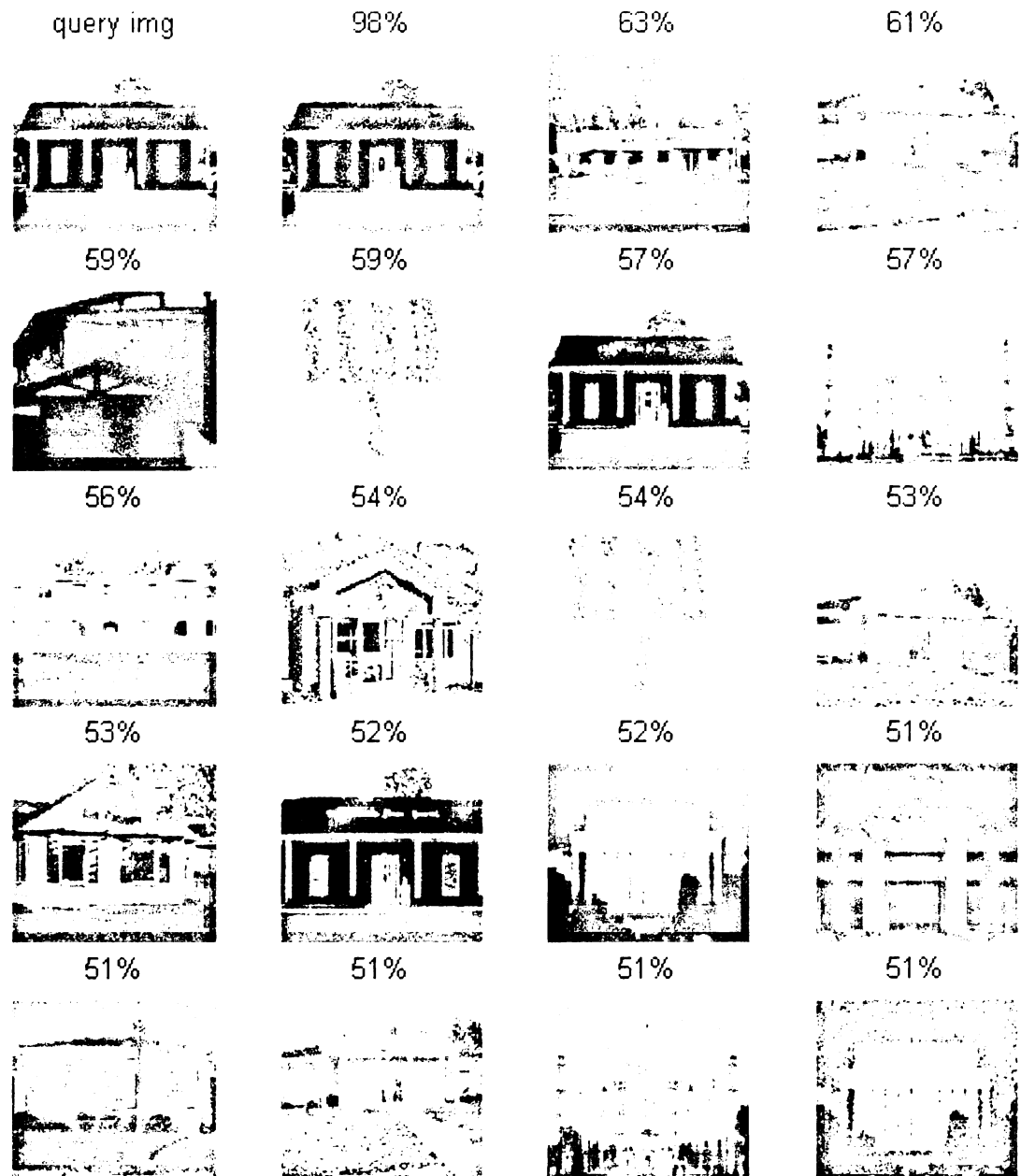


Figure 6-3 Query image Small Building and retrieved images

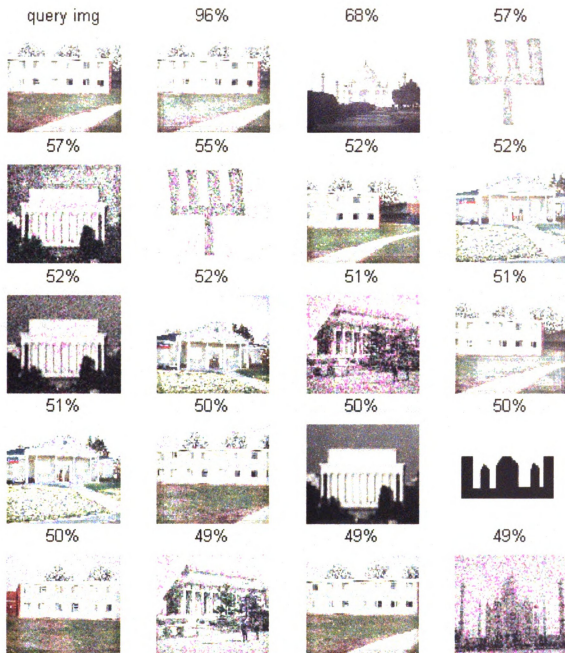


Figure 6-4 Query image Spartan Village Apartments and retrieved images



## 6.4 Corner-Graph Matching.

The particular goal in this section is to incorporate the corner structure that is defined in chapter 5 into the matching algorithm and to study the effect. Before we proceed to experiment with the corner matching, it is important to understand the way the algorithm works and some of the notation that is being used in the subsequent paragraphs

If  $G = (V, E)$  is a graph then  $V = \{ \tau_{iA} \}$  is the descriptor of vertex and  $E = \{ \tau_{ijD}, \tau_{ijO}, \tau_{ijP}, \tau_{ijC} \}$  are the relationships that vertices make with each other.  $\tau_{iA}$  represents the corner angle and  $\tau_{ijD}, \tau_{ijO}, \tau_{ijP}, \tau_{ijC}$  represent the distance between corners, corner's orientation, corner's precinct and angle between corners. where  $i \neq j$ .

### 6.4.1 Matching Algorithm

The graph-matching algorithm for corner structures is identical to the line structure except the attributes of vertices and edges are used for comparison. For detailed explanation, see Section 6.3.1.

```

//G=(V , E) where V={  $\tau_{A.}$  }and E={  $\tau_D, \tau_P, \tau_O, \tau_C$  }
CornerMatchingRoutine( G1 , G2 )
{
n1=size of graph G1
n2=size of graph G2
vote=0
k=1
for i=1 to n1
  for j=1 to n2
    if(match_vertex ( Vi , Vj ) // match vertex Vi with Vj
      vote=match_edge(Ei( i , : ) , Ej( j , : ) , i , j )
      if vote > 0
        match( k , : ) = [Ci Cj vote] //store the matching results
        k=k+1
      end if
    end if
  end for
end for

FinalMatch = corner_sel_by_vote( match )
return (FinalMatch)
}

```

Algorithm 6-6 Corner matching routine

```

match_edge( e1 , e2 , I , J )
{
vote=0
for i=I-5 to I+5
  for j=J-5 to J+5
    if (  $\tau_{iP1} == \tau_{jP2}$  &  $\tau_{iO1} == \tau_{jO2}$  &
      |  $\tau_{iD1} - \tau_{jD2}$  | <  $\delta 1$  & |  $\tau_{iC2} - \tau_{jC2}$  | <  $\delta 2$  )
      if ( match_vertex ( Vi , Vj ))
        vote=vote+1
      end if
    end if
  end for
end for
return (vote)
}

```

Algorithm 6-7 Edge matching sub-routine

```

Match_vertex( $v_1$  ,  $v_2$  )
{ if (  $|\mathcal{T}_{1A} - \mathcal{T}_{2A}| < \delta n$  ) // compare the vertex
  return (1)
  else return (0) }

```

Algorithm 6-8 Vertex matching sub-routine

```

corner_sel_by_vote( CornerMatch )
{
x = size of CornerMatch
for i=1:5
  j=1
  while j<x
    if (CornerMatch (j,1) == CornerMatch (j+1,1)) // compare the corner #
                                                    // for G1
      if (CornerMatch (j,3) > CornerMatch (j+1,3)) // compare the votes
        CornerMatch (j+1,:) = [ ] // delete the next corner
      else CornerMatch (j,:) = [ ] // delete the corner
      end
      x=x-1
    end
    j=j+1
  end
end
CornerMatch =sortrows(CornerMatch ,[2]) // sort array by row # 2
x =size of CornerMatch
for i=1:5
  j=1
  while j<x
    if (CornerMatch (j,2)==CornerMatch (j+1,2)) // compare the corner #
                                                    // for G2
      if (CornerMatch (j,3) > CornerMatch (j+1,3)) // compare the votes
        CornerMatch (j+1,:) = [ ]; // delete the next corner
      else CornerMatch (j,:) = [ ]; // delete the corner
      end
      x=x-1;
    end
    j=j+1;
  end
end
return (CornerMatch )

```

Algorithm 6-9 Corner selection sub-routine

Algorithms	Thresholds	Explanation	Default Setting
Algorithm 6.7	$\delta n$	Corner angle	$10^\circ$
Algorithm 6.8	$\delta 1$	Distance between corners	10 pixels
Algorithm 6.8	$\delta 2$	Angle between corners	$10^\circ$

Table 6-2 Different threshold used in Corner Structure Graph Algorithms

#### 6.4.2 Matching Results for Corners

In this section, we will investigate the robustness and quality of the graph-matching algorithm for corners. We conducted four experiments on the same query images as was done in Section 6.3.2. The results from the first experiment are shown in Figure 6-5. This experiment was one of the best experiments and gave a 100% recall. All the images from the group were successfully retrieved. In the second experiment, as shown in Figure 6-6, the algorithm successfully retrieved six out of nine Taj Mahal images. Retrieved images of Taj Mahal are of different scale and resolutions, view and with Gaussian noise. In experiments three and four the results are much better than the experiments three and four of graph-matching algorithm for lines. The results from these experiments are shown in Figure 6-7 and Figure 6-8 respectively.

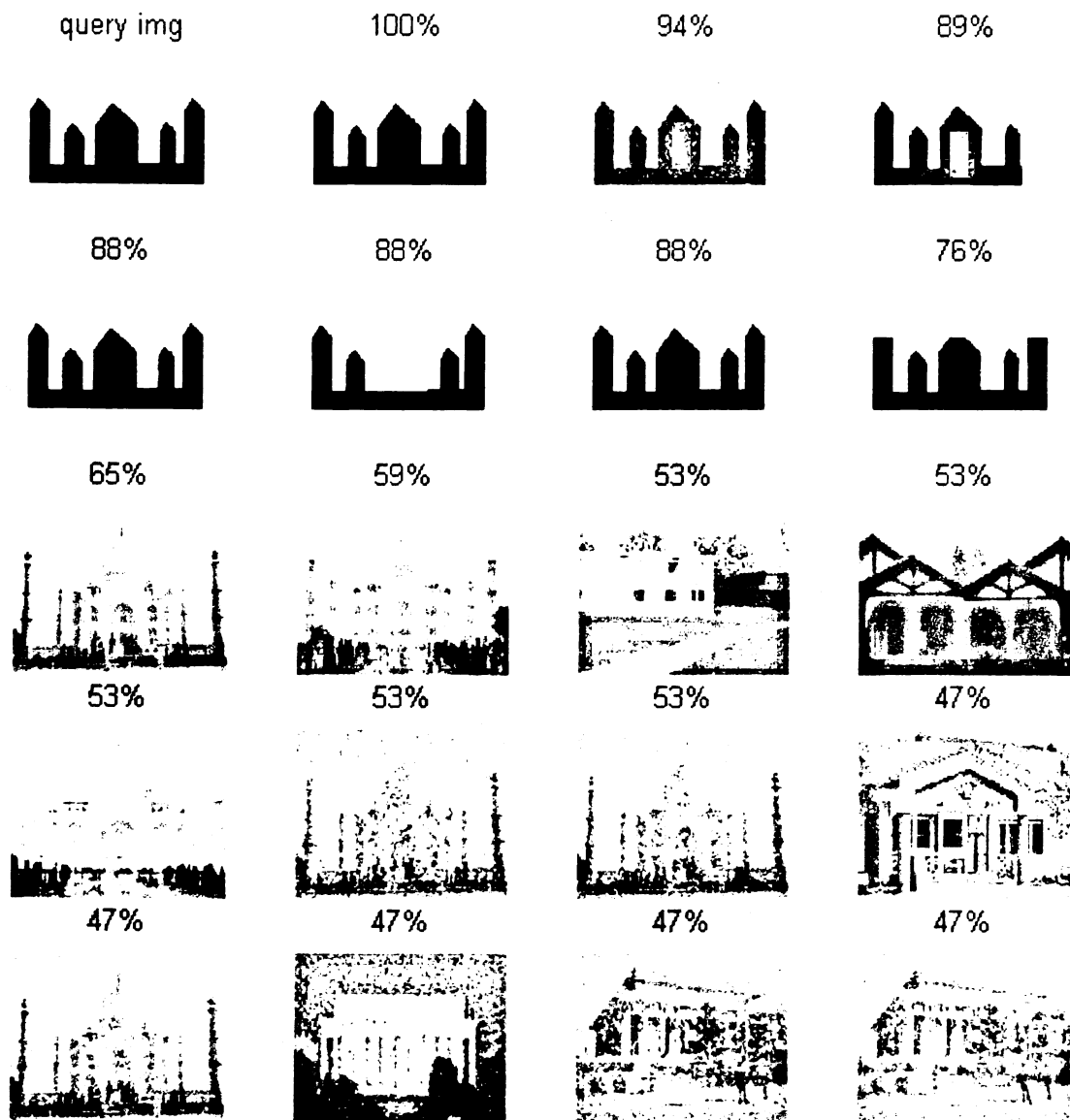


Figure 6-5 Query image Drawn Taj and retrieved images



Figure 6-6 Query image Taj Mahal and retrieved images



Figure 6-7 Query image Small Building and retrieved images



Figure 6-8 Query image Spartan Village Apartments and retrieved images



## 6.5 Ribbon-Graph Matching.

The particular goal in this section is to incorporate the ribbon structure that is defined in Chapter 5 into the matching algorithm and to study the effect. Before we proceed to experiment with the ribbon matching, it is important to understand the way the algorithm works and some of the notation that is being used in the subsequent paragraphs.

If  $G = (V, E)$  is a graph then  $V = \{ \mathcal{L}_{iL}, \mathcal{L}_{iG}, \mathcal{L}_{iS}, R_{iT} \}$  are descriptors of vertices and  $E = \{ R_{ijD}, R_{ijO} \}$  are the relationships that vertices makes with each other.  $\mathcal{L}_{iW}, \mathcal{L}_{iG}, \mathcal{L}_{iS}, R_{iT}$  represent the width of a ribbon, edge gradient, line slope and ribbon type respectively, and  $R_{ijD}, R_{ijO}$  represent the distance between ribbons, and ribbon's orientation, where  $i \neq j$ .

### 6.5.1 Matching Algorithm

The graph matching algorithm for ribbon structure is identical to the line structure and corner structure algorithms except for the attributes of vertices and edges. For details see section 6.3.1. Thresholds used in algorithms are given in Table 6-3.

Algorithms	Thresholds	Explanation	Default Setting
Algorithm 6.11	$\delta_1$	Distance between ribbons	20 pixels
Algorithm 6.12	$\delta_2$	Width of a ribbon	10 pixels
Algorithm 6.12	$\delta_3$	Slope of lines	$10^\circ$
Algorithm 6.12	$\delta_4$	Edge gradient of lines	$30^\circ$

Table 6-3 Different thresholds used in Ribbon Structure Graph Algorithms

```

//G=(V , E) where V={  $\mathcal{L}_L, \mathcal{L}_G, \mathcal{L}_S, R_T$  } and E={  $R_D, R_O, R_C$  }
RibbonMatchingRoutine( G1 , G2 )
{
n1=size of graph G1
n2=size of graph G2
vote=0
k=1
for i=1 to n1
  for j=1 to n2
    if(match_vertex ( Vi , Vj ))// match vertex Vi with Vj
      vote=match_edge(Ei( i , : ) , Ej( j , : ) , i , j )
      if vote > 0
        match( k , : ) = [R i R j vote] //store the matching results
        k=k+1
      end if
    end if
  end for
end for

FinalMatch = ribbon_sel_by_vote( match )
return (FinalMatch)
}

```

Algorithm 6-10 Ribbon matching routine

```

match_edge( e1 , e2 , I , J )
{
vote=0
for i=I-5 to I+5
  for j=J-5 to J+5
    if (  $R_{iO1} = R_{jO2}$  &  $|R_{iD1} - R_{jD2}| < \delta 1$  )
      if ( match_vertex ( Vi , Vj ))
        vote=vote+1
      end if
    end if
  end for
end for
return (vote)
}

```

Algorithm 6-11 Edge matching sub-routine

```

Match_vertex( $v_1$  ,  $v_2$  )
{
  if (  $R_{1T} = R_{2T}$  &  $|\mathcal{I}_{1W} - \mathcal{I}_{2W}| < \delta_2$  &  $|\mathcal{I}_{1S} - \mathcal{I}_{2S}| < \delta_3$  &  $|\mathcal{I}_{1G} - \mathcal{I}_{2G}| < \delta_4$ )
  // compare the vertex
    return (1)
  else return (0) }

```

Algorithm 6-12 Vertex matching sub-routine

```

ribbon_sel_by_vote( RibbonMatch )
{
  x = size of RibbonMatch
  for i=1:5
    j=1
    while j<x
      if (RibbonMatch (j,1) == RibbonMatch (j+1,1)) // compare the Ribbon #
                                                    // for G1
        if (RibbonMatch (j,3) > RibbonMatch (j+1,3)) // compare the votes
          RibbonMatch (j+1,:) = [ ] // delete the next Ribbon
        else RibbonMatch (j,:) = [ ] // delete the Ribbon
        end
        x=x-1
      end
      j=j+1
    end
  end
  RibbonMatch =sortrows(RibbonMatch ,[2]) // sort array by row # 2
  x =size of RibbonMatch
  for i=1:5
    j=1
    while j<x
      if (RibbonMatch (j,2) ==RibbonMatch (j+1,2)) // compare the Ribbon #
                                                    // for G2
        if (RibbonMatch (j,3) > RibbonMatch (j+1,3)) // compare the votes
          RibbonMatch (j+1,:) = [ ]; // delete the next Ribbon
        else RibbonMatch (j,:) = [ ]; // delete the Ribbon
        end
        x=x-1;
      end
      j=j+1;
    end
  end
  return (RibbonMatch )
}

```

Algorithm 6-13 Ribbon selection sub-routine

### **6.5.2 Matching Results for Ribbons**

In this section, we will investigate the robustness and quality of the graph-matching algorithm for ribbons. We conducted four experiments on the same query images as was done in Section 6.3.2 and Section 6.4.2. The results from the first experiment are shown in Figure 6-9. This experiment was the best experiment among four experiments conducted and gave 100% recall. All the images from the group were successfully retrieved. In the second experiment, as shown in Figure 6-10, the algorithm successfully retrieved three out of nine Taj Mahal images. Retrieved images of Taj Mahal are of different scale and resolutions, view and with gaussian noise. In experiment three and four the results are much also better than the experiment three and four of graph-matching for lines and corners. The results from these experiments are shown in Figure 6-11 and Figure 6-12 respectively. The overall results from the graph matching algorithm for ribbons are much better than for lines and corners.

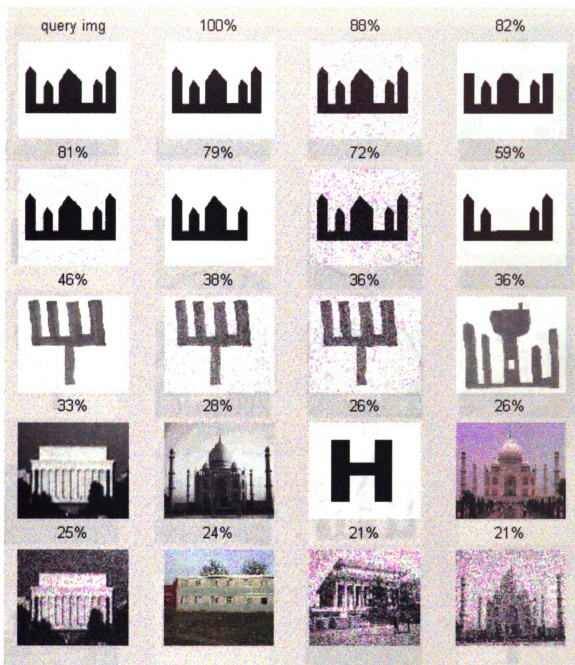


Figure 6-9 Query image drawn Taj and retrieved images

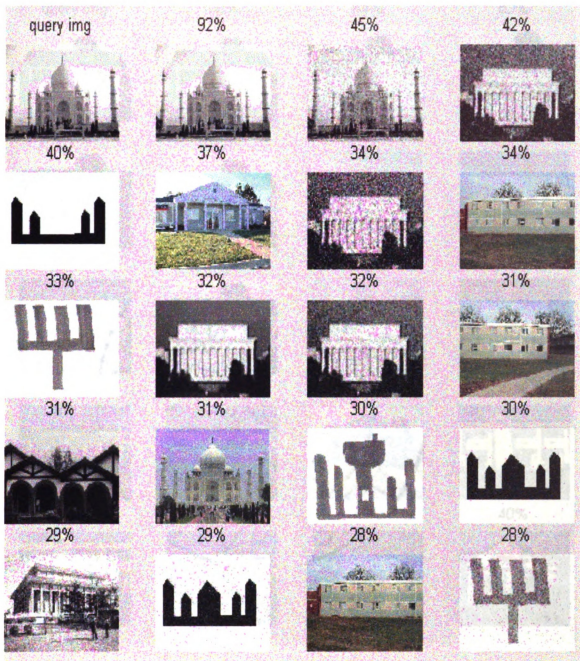


Figure 6-10 Query image Taj Mahal and retrieved images



Figure 6-11 Query image Small Building and retrieved images



Figure 6-12 Query image Spartan Village Apartments and retrieved images



## **6.6 Combined Results of Graph Matching Algorithm**

To justify our goal of employing a graph matching algorithm to retrieve images from a database consider Figure 6-13 to Figure 6-16. The results obtained from line structure, corner structure and ribbon structure are weighted by  $1/3$  and then added to obtain the combined results. The results indicate that by combining individual results we can improve precision.

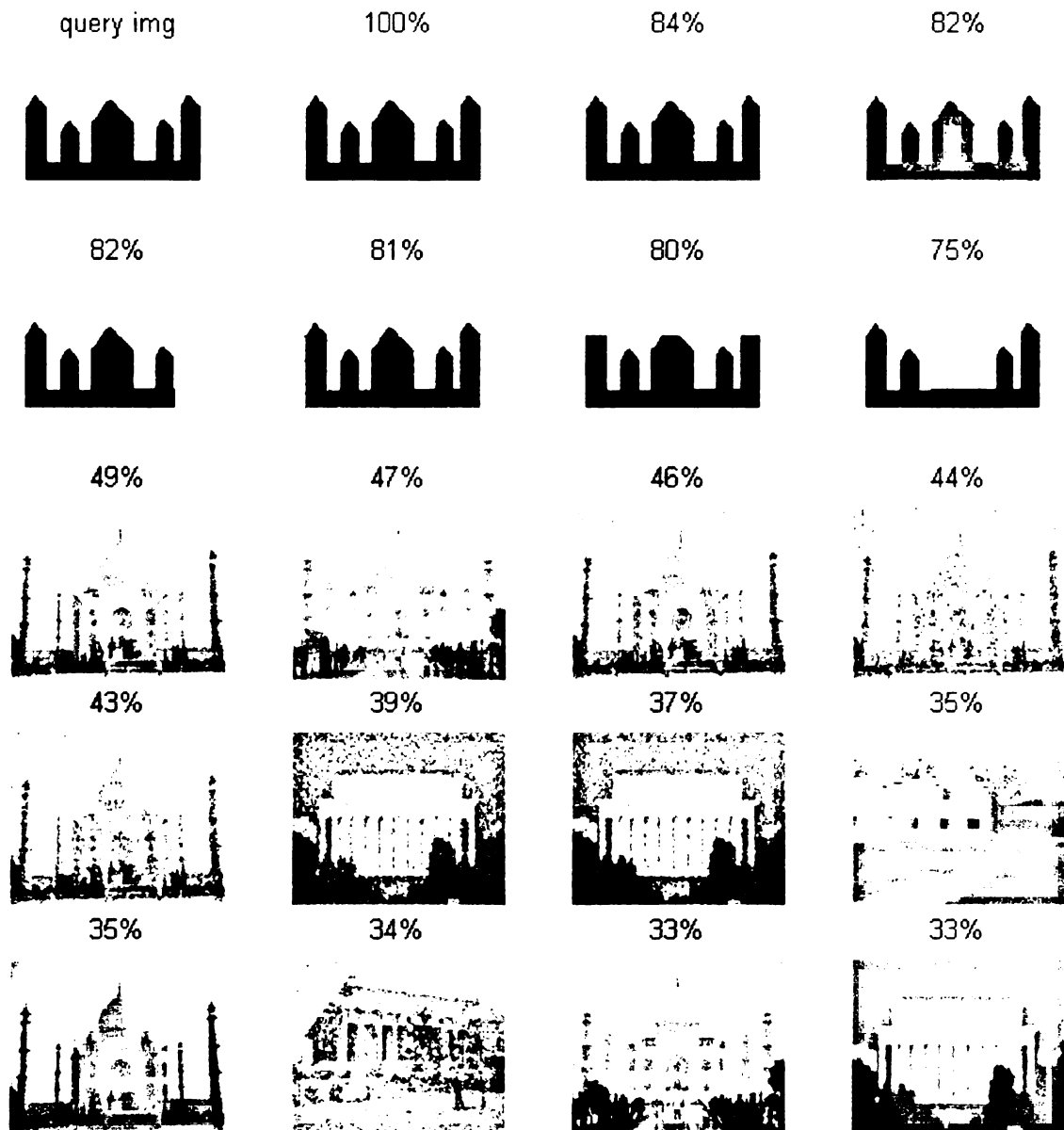


Figure 6-13 Query image drawn Taj and retrieved images

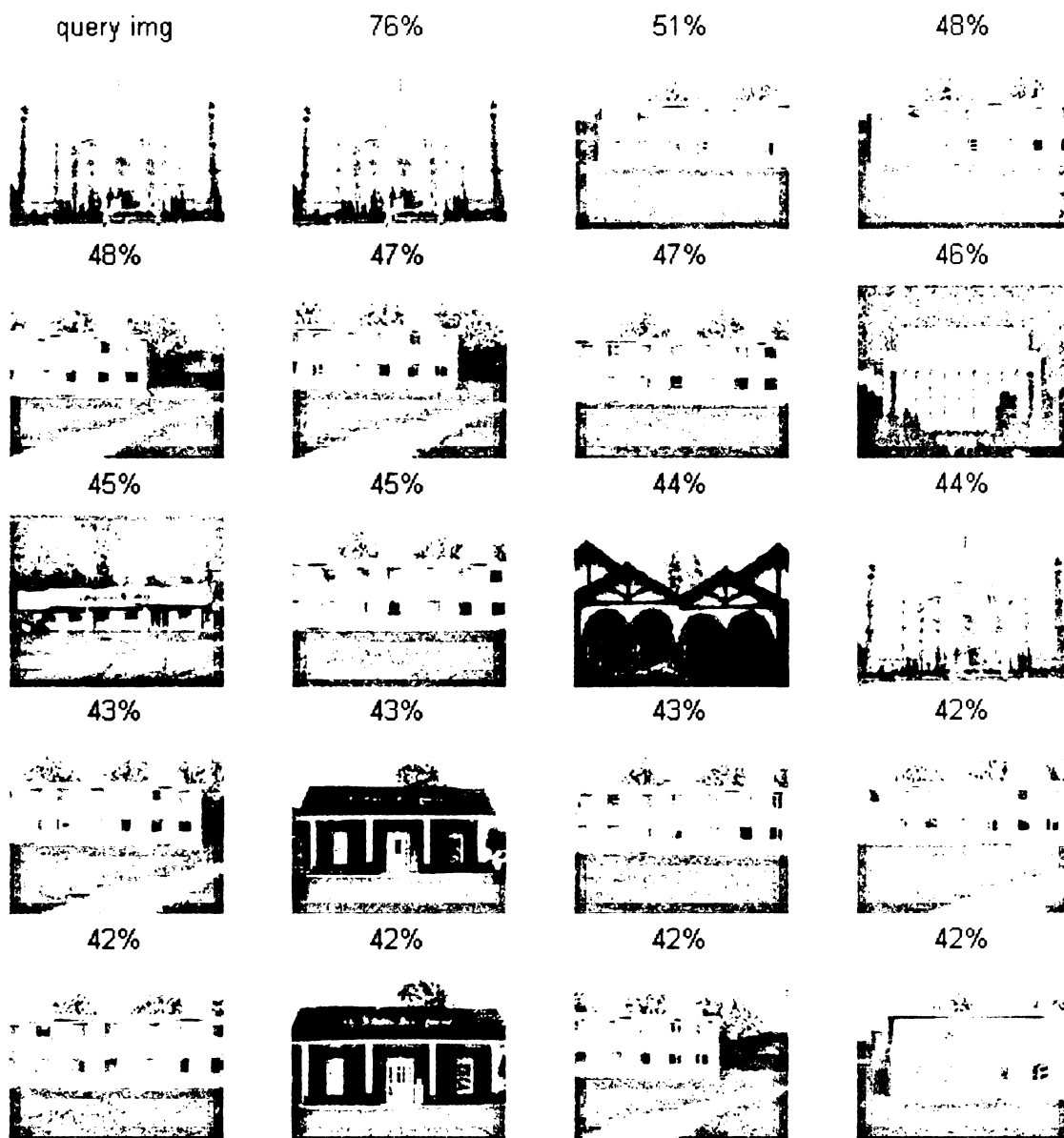


Figure 6-14 Query image Taj Mahal and retrieved images

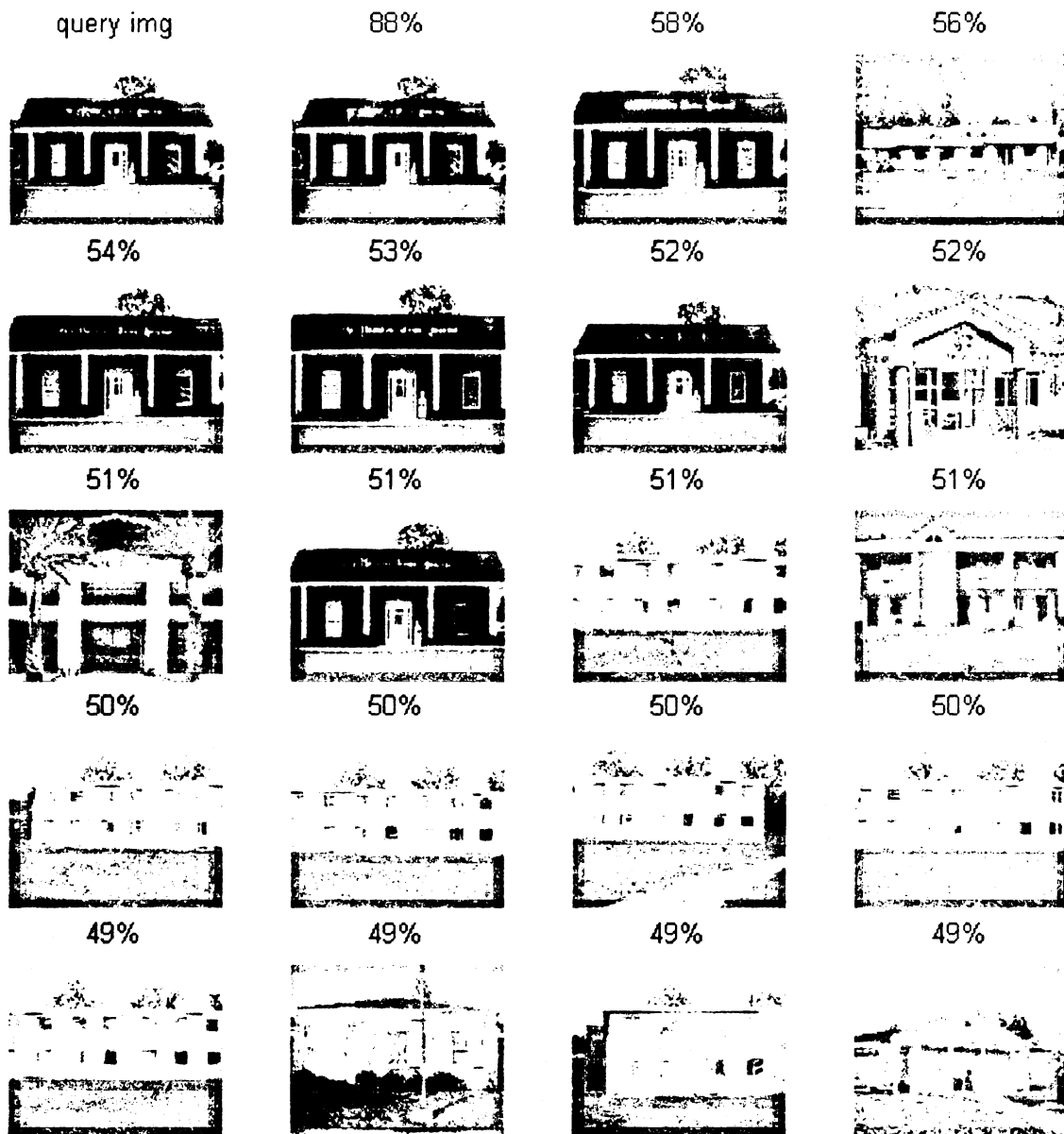


Figure 6-15 Query image Small Building and retrieved images

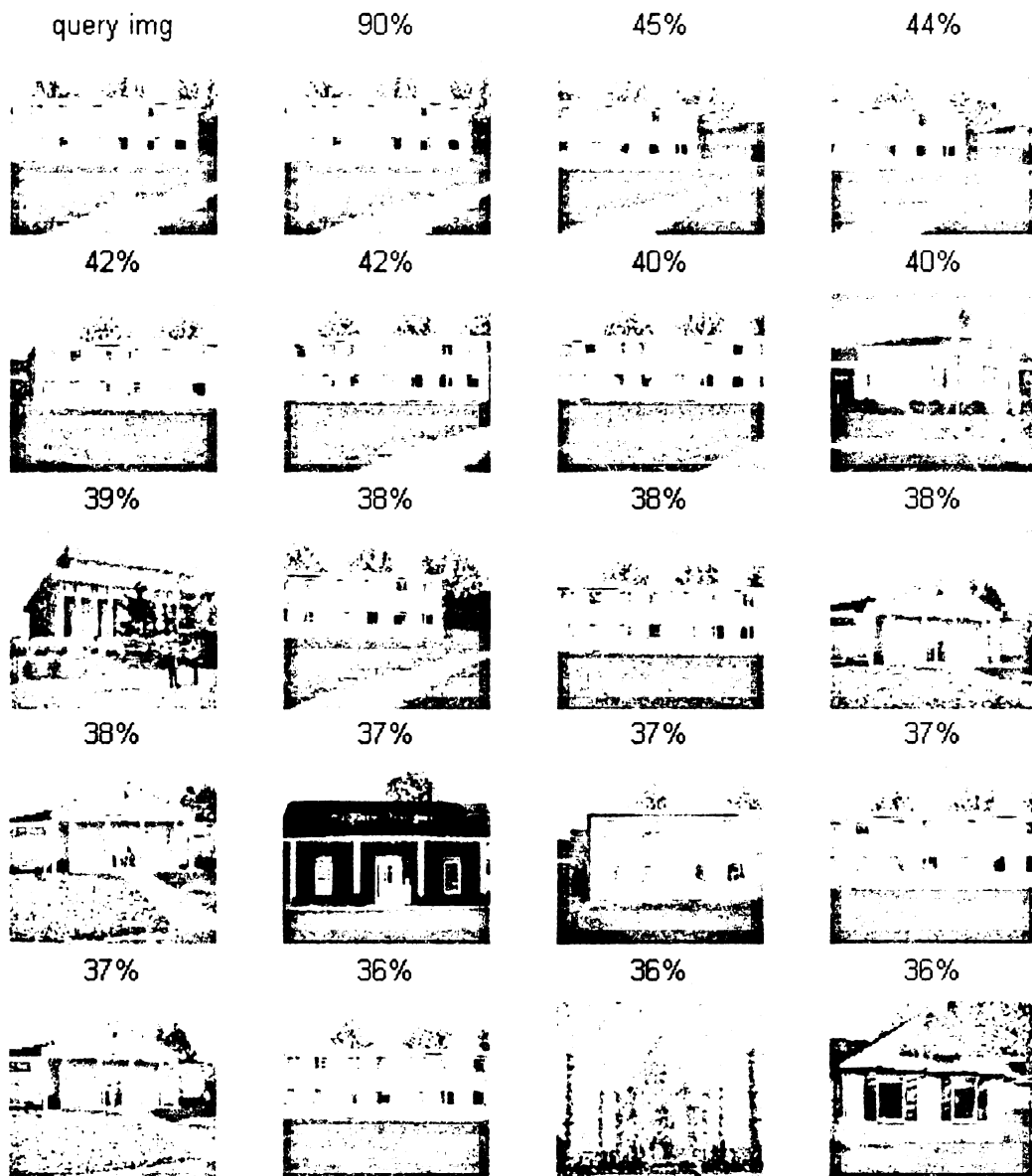


Figure 6-16 Query image Spartan Village Apartments and retrieved images

## **6.7 Results of Aerial Images**

Fifteen aerial images were added to the image database. Three separate experiments were conducted for matching the lines, corners and ribbon structures respectively. Results are shown in Figure 6-17 to Figure 6-24. Results show that the line and the ribbon matches have better performance than the corner match. All the images were retrieved within the 25 % image similarity.

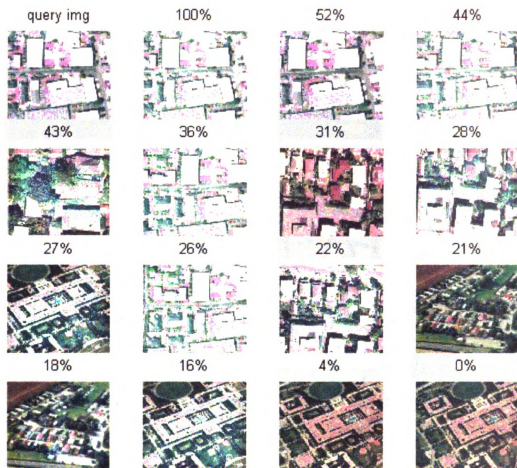


Figure 6-17 Query image 'img92' -line matching results

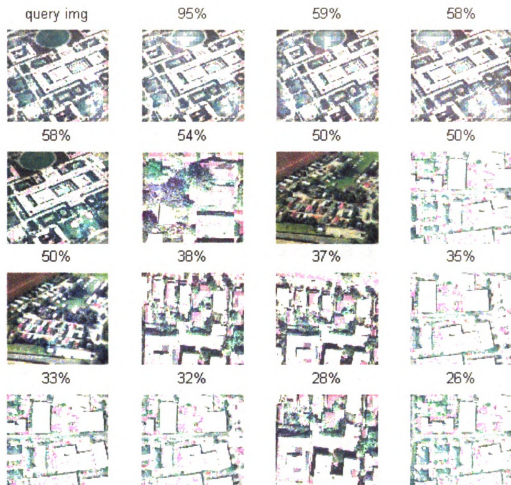


Figure 6-18 Query image 'img88' – line matching results



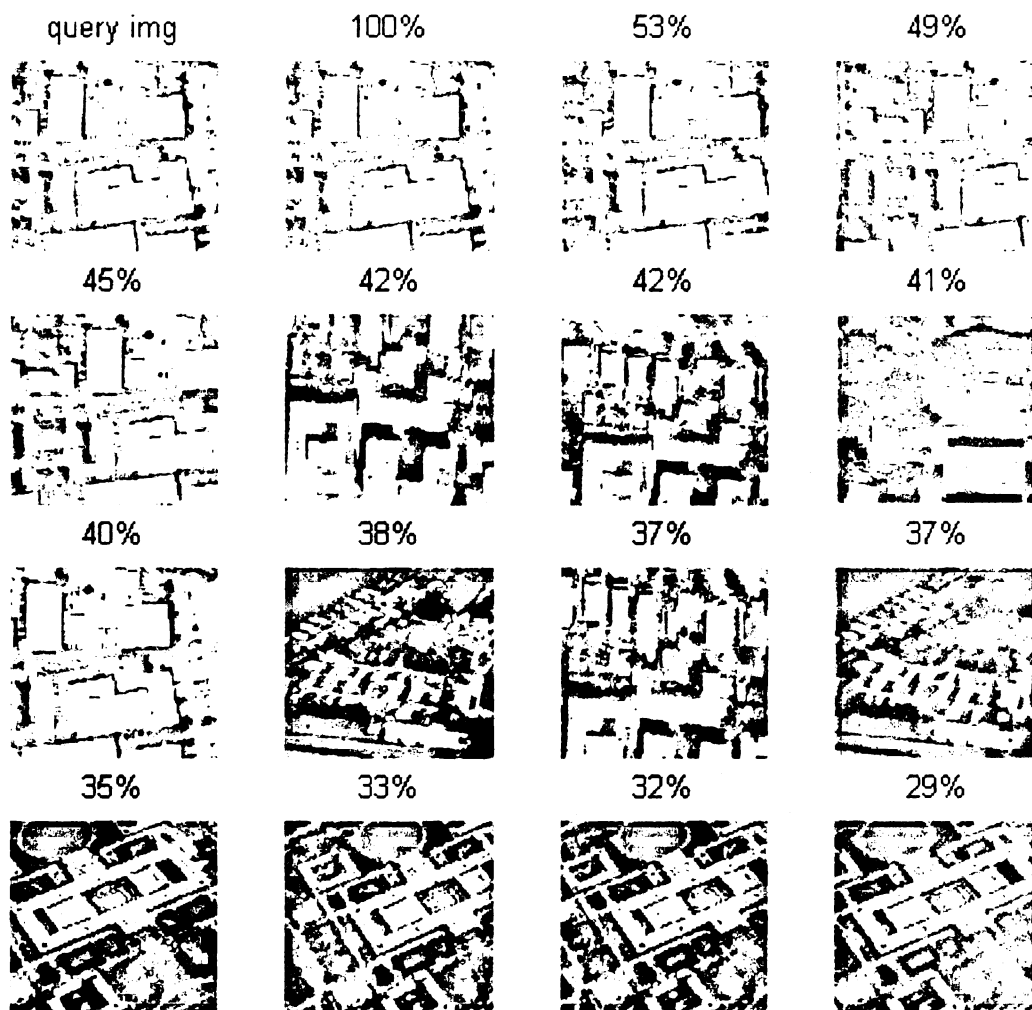


Figure 6-19 Query image 'img92' – corner matching results

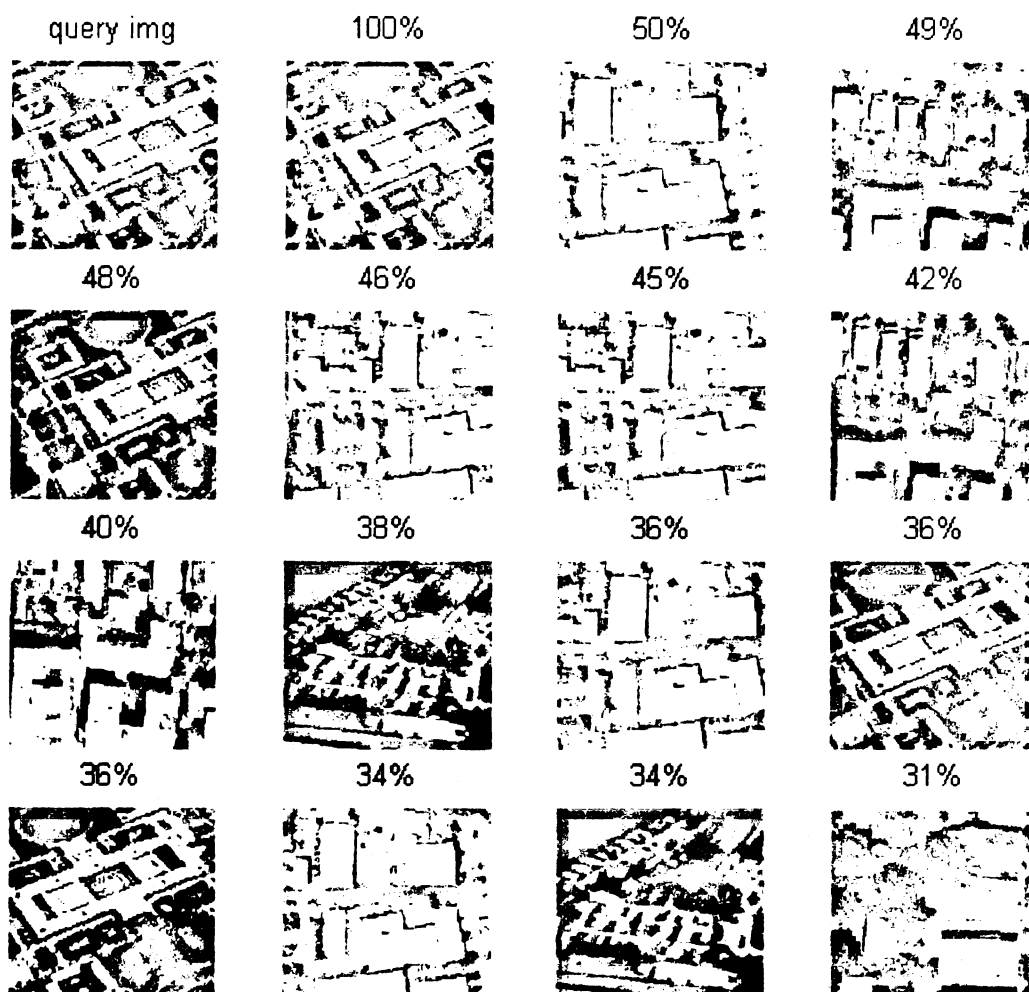


Figure 6-20 Query image 'img88' – corner matching results

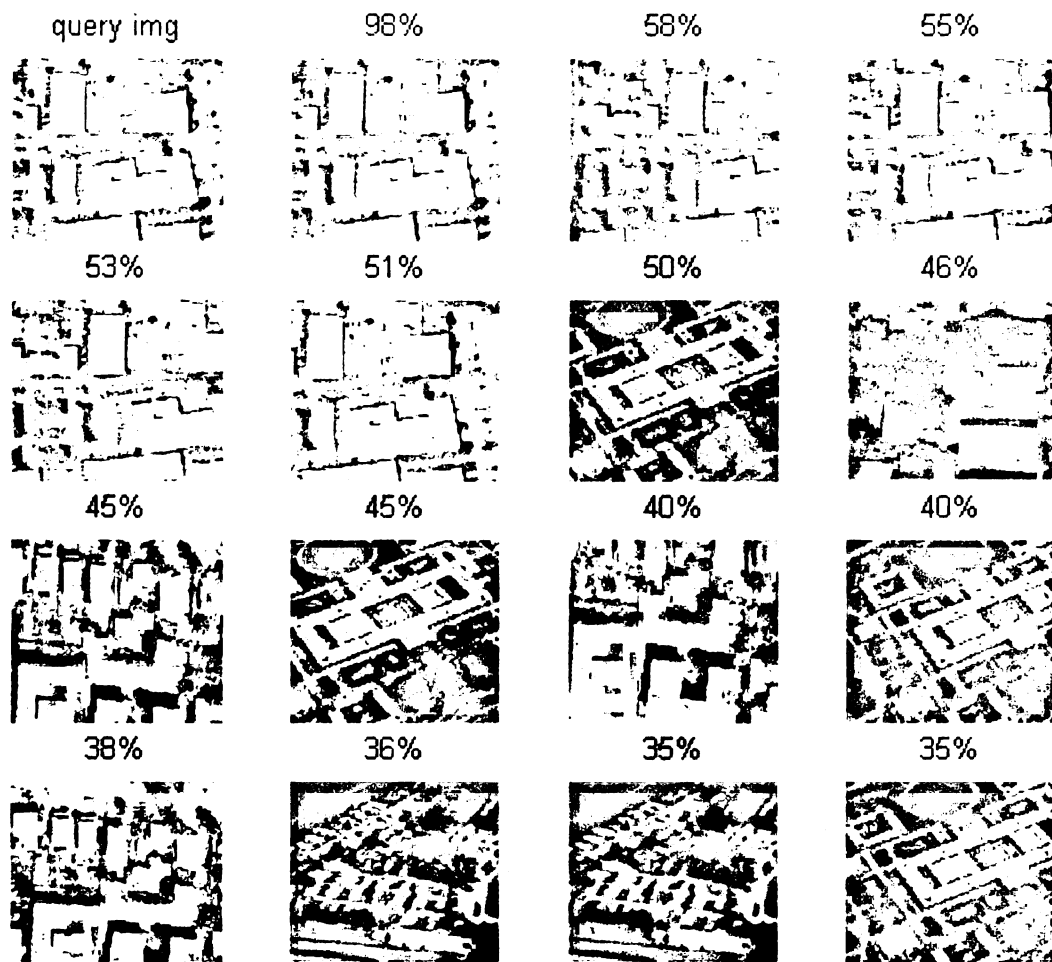


Figure 6-21 Query image 'img92' – ribbon matching results

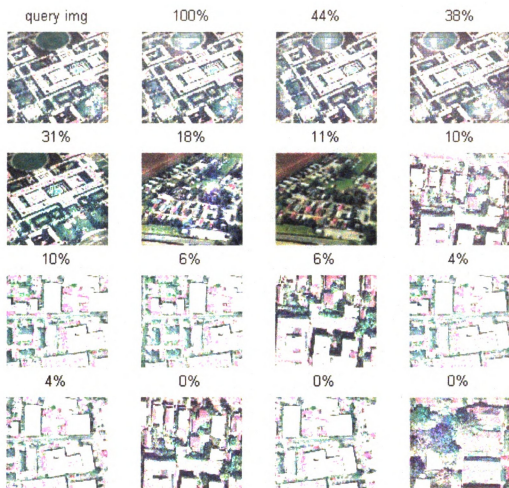


Figure 6-22 Query image 'img88' – ribbon matching results

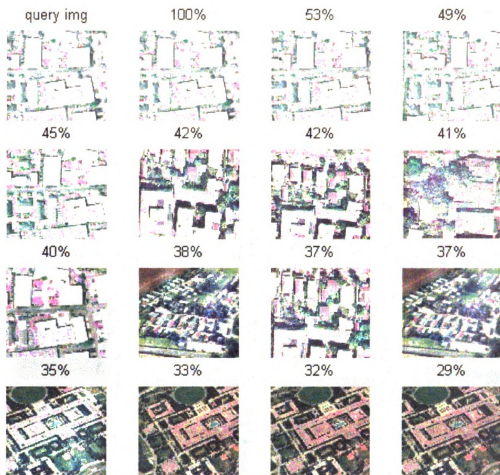


Figure 6-23 Query image 'img92' – combined results

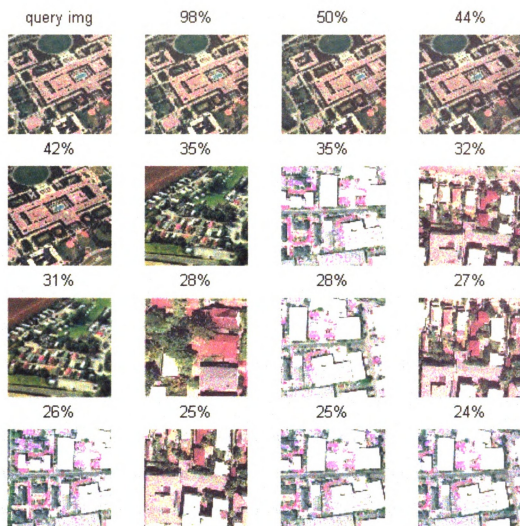


Figure 6-24 Query image 'img88' – combined results

## **7. Discussion and Future Work**

### **7.1 Analysis**

The overall performance of the graph-matching algorithm is comparable to other CBIR matching algorithms. Although it was assumed that the performance of the proposed matching algorithm would solely depend on the feature detection process, meaning that feature extraction would be of vital importance, results show that this is not the case. As we added different levels of noise to some of the images, it negatively affected the feature extraction results but still correct images were retrieved. For instance, we added Gaussian noise of 1%, 2% and 4% to the Drawn Taj and Taj Mahal but still our graph-matching algorithm successfully retrieved these images. To test our matching algorithm against perspective/view tolerance, we also added images with different perspectives in our database, such as the Spartan village apartments, Fifth Third Bank and Small Building. The results are very encouraging. When we submit one of the images from Spartan village apartments as the query, the matching algorithm successfully retrieved eleven out of thirteen relevant images and for the small building query, six out of seven relevant images were retrieved. We also tested the algorithm for intensity variations and were successful in retrieving images of different brightness, e.g., see results for Taj Mahal and others. The algorithm was also tested for occlusion, for which we removed some of the minarets from Drawn Taj and submitted the query; the algorithm still successfully retrieved the relevant images.

Results from all the three structures used in our graph-matching algorithm are quite promising and the combination results are even better. Although possible combinations of two features were not tried, it is safe to assume that results will be better than the

individual results. Table 7-1 and Table 7-2 summarize the results using a combination of the three features. Table 7-1 shows the results for exact matching. By exact matching, we mean that the same building with different perspectives or with added noise is used in the query. The first row of the table shows four percentages of similarity for which we evaluate image retrieval results. The remaining five show the percentage of images recalled at each of the four levels of similarity. For instance, for the Drawn Taj query, one of the seven relevant images was retrieved at the 100 % matching level, (14 %) whereas all seven relevant matches were retrieved at the 75 % matching level (100 %).

Table 7-2 shows that a similarity level of 25 % may have to be used to achieve a high recall value.

<b>Similarity</b>	<b>100%</b>	<b>75%</b>	<b>50%</b>	<b>25%</b>	<b>Total Images</b>
<b>Query Images</b>					
Drawn Taj	14%	100%	100%	100%	7
Fifth Third Bank	25%	25%	25%	75%	4
Small Building	-	12%	12%	88%	8
Spartan Village Apartments	-	8%	8%	84%	13
Taj Mahal		25%	25%	50%	4

Table 7-1 Results for exact matching

Table 7-2 shows the results for those images that are similar in shape. For instance, the Drawn Taj is similar to the Taj Mahal and 54% of relevant images were retrieved within 75% of similarity and 100% were retrieved within 25% of similarity. Again, 25 % similarity level will recall a significant percentage of target images. These results also show that precision will go down if exact matches are wanted.



<b>Similarity</b>	<b>100%</b>	<b>75%</b>	<b>50%</b>	<b>25%</b>	<b>Total Images</b>
<b>Query Images</b>					
Drawn Taj	8%	54%	54%	100%	14
Fifth Third Bank	11%	11%	11%	45%	9
Small Building	-	6%	67%	84%	15
Spartan Village Apartments	-	8%	4%	84%	13
Taj Mahal	-	12%	12%	25%	8

Table 7-2 Results for similar matching

Like every algorithm, our graph-matching algorithm also has some weak points. The graph-matching algorithm is slow compared to some other types of matching algorithms, such as histogram matching. The complexity of the graph-matching algorithm is  $n \times m$ , where  $n$  and  $m$  are the number of features of test and query images, respectively. In Table 7-3 some of the data-structures and sizes are shown. From this, one can anticipate the relative compute time required for the corresponding matching of graphs. Table 7-4 gives total time in seconds for comparison of query image with 97 images from database.

<b>Query Images</b>	<b>Line Structure</b>	<b>Corner Structure</b>	<b>Ribbon Structure</b>
Drawn Taj	21	17	16
Fifth Third Bank	187	76	55
Small Building	96	41	60
Spartan Village Apartments	128	47	28
Taj Mahal	136	60	24

Table 7-3 Graph sizes (number of vertices) for different data-structures and images

<b>Query Images</b>	<b>Time Taken for Matching in Seconds</b>
Drawn Taj	165
Fifth Third Bank	3140
Small Building	1940
Spartan Village Apartments	2100
Taj Mahal	1880

Table 7-4 Run-time for image retrieval

## 7.2 Conclusions

Content Based Image Retrieval (CBIR) is an important problem for computer vision and computer science. This thesis presents a new geometric method for a Content Based Image Retrieval system that internally uses graph-matching for image identification and retrieval. The retrieval system is divided into three sub processes, feature extraction, data representation and graph-matching algorithm. Feature extraction and data representation are pre-processors, while the graph-matching is done online. Initially features such as lines, corners and ribbons are extracted and saved in a graph data-structure. Subsequently, when a query is submitted to the graph-matching algorithm, it first undergoes feature extraction. The algorithm then uses the extracted query feature space along with the existing data-structure database for image matching and retrieval purposes.

The graph-matching algorithm that is proposed in this thesis is robust to the presence of noise, occlusion, small changes in perspective, and image brightness. The performance of the algorithm against such adverse effects has been measured via MATLAB simulations. Further, the proposed algorithm can detect abrupt changes in view and perspective; therefore, it can be also used for automatic view change detection in

continuous video. The current implementation is too slow to be practical. Improvements are suggested below.

### **7.3 Future Work**

Although all the desired goals of this work have been met, several new issues have been noticed, which can be dealt with in future research. To improve the retrieval performance of this prototype image retrieval system, color histogram matching and texture matching need to be integrated into the algorithm implementation. Further, one can enhance the existing algorithm for rotation invariance, if needed by the user. The ability to match the rotated images is critical for some aerial imagery applications; the current algorithm can handle only small rotations in perspective for ground geometric structures such as buildings.

The performance of the algorithm for video is yet to be evaluated. Further, it is evident from the current results that the system can be used for object recognition and localization. It can be used for security purposes where finding and matching of exact buildings or other geometric features in a database of aerial imaging data may be required. Of course, runtime performance must be improved. Reprogramming in C and using hierarchical matching are two sure methods for significantly speeding up matching.

## **Appendix A**



img1  
192x192x16M jpeg



img10  
176x192x16M jpeg



img11  
176x192x16M jpeg



img12  
221x192x16M jpeg



img13  
221x192x16M jpeg



img14  
221x192x16M jpeg



img15  
221x192x16M jpeg



img16  
206x182x16M jpeg



img17  
180x181x16M jpeg



img18  
179x192x16M jpeg



img19  
358x363x16M jpeg



img2  
192x192x16M jpeg



img20  
160x157x16M jpeg



img21  
115x109x16M jpeg



img22  
115x109x16M jpeg



img23  
115x109x16M jpeg



img24  
115x109x16M jpeg



img25  
115x109x16M jpeg

# E

img26  
115x109x16M jpeg



img29  
182x182x16M jpeg



img31  
192x192x16M jpeg



img34  
256x144x16M jpeg



img37  
289x192x16M jpeg



img4  
192x192x16M jpeg

# F

img27  
115x109x16M jpeg



img3  
192x192x16M jpeg



img32  
192x192x16M jpeg



img35  
256x144x16M jpeg



img38  
300x199x256 jpeg



img40  
289x192x16M jpeg

# H

img28  
115x109x16M jpeg



img30  
192x192x16M jpeg



img33  
256x144x16M jpeg



img36  
256x144x16M jpeg



img39  
289x192x16M jpeg



img41  
289x192x16M jpeg



img42  
192x192x16M jpeg



img43  
320x240x16M jpeg



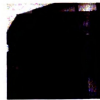
img44  
320x240x16M jpeg



img45  
230x237x256 jpeg



img46  
192x192x16M jpeg



img47  
192x192x16M jpeg



img48  
320x240x16M jpeg



img49  
320x240x16M jpeg



img5  
192x192x16M jpeg



img50  
320x240x16M jpeg



img51  
320x240x16M jpeg



img52  
320x240x16M jpeg



img53  
320x240x16M jpeg



img54  
320x240x16M jpeg



img55  
320x240x16M jpeg



img56  
320x240x16M jpeg



img57  
320x240x16M jpeg



img58  
320x240x16M jpeg



img59  
320x240x16M jpeg



img6  
192x192x16M jpeg



img60  
320x240x16M jpeg



img61  
275x182x16M jpeg



img62  
270x338x16M jpeg



img63  
300x192x16M jpeg



img64  
300x192x16M jpeg



img65  
300x192x16M jpeg



img66  
300x192x16M jpeg



img67  
300x192x16M jpeg



img68  
295x187x16M jpeg



img69  
320x240x16M jpeg



img7  
192x192x16M jpeg



img70  
320x240x16M jpeg



img71  
320x240x16M jpeg



img72  
320x240x16M jpeg

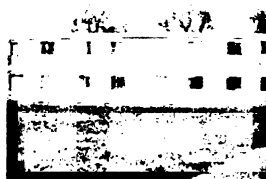


img73  
320x240x16M jpeg

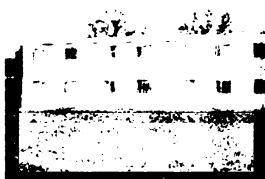


img74  
320x240x16M jpeg

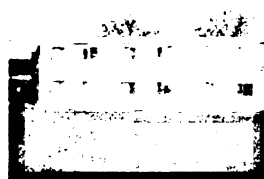




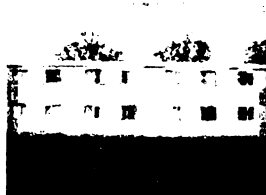
img75  
320x240x16M jpeg



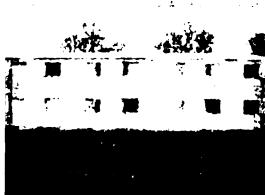
img76  
320x240x16M jpeg



img77  
320x240x16M jpeg



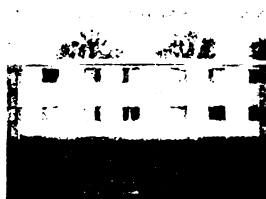
img78  
320x240x16M jpeg



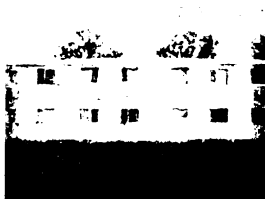
img79  
320x240x16M jpeg



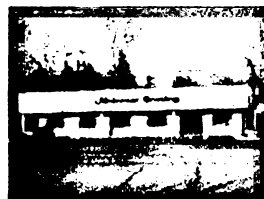
img8  
176x192x16M jpeg



img80  
320x240x16M jpeg



img81  
320x240x16M jpeg



img82  
320x240x16M jpeg



img83  
129x128x16M jpeg



img84  
241x241x16M jpeg



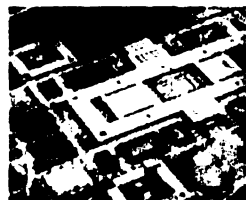
img85  
248x234x16M jpeg



img86  
248x234x16M jpeg



img87  
235x179x16M jpeg



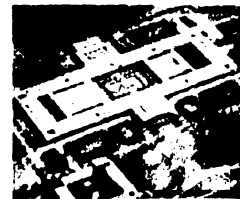
img88  
223x181x16M jpeg



img89  
216x179x16M jpeg



img9  
176x192x16M jpeg



img90  
213x179x16M jpeg



img91  
194x179x16M jpeg



img92  
205x185x16M jpeg



img93  
190x186x16M jpeg



img94  
194x189x16M jpeg



img95  
205x189x16M jpeg



img96  
241x187x16M jpeg



img97  
245x193x16M jpeg

## **Appendix B**

### Line Detection Results

### Corner Detection Results

### Ribbon Detection Results



Img 1 (a)



Img 1 (b)



Img 1 (c)



Img 12 (a)



Img 12 (b)



Img 12 (c)



Img 16 (a)



Img 16 (b)



Img 16 (c)



Img 29 (a)



Img 29 (b)



Img 29 (c)



Img 37 (a)



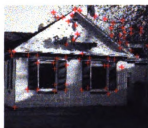
Img 37 (b)



Img 37 (c)



Img 48 (a)



Img 48 (b)



Img 48 (c)



Img 52 (a)



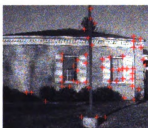
Img 52 (b)



Img 52 (c)



Img 57 (a)



Img 57 (b)



Img 57 (c)



Img 60 (a)



Img 60 (b)



Img 60 (c)



Img 65 (a)



Img 65 (b)



Img 65 (c)



Img 73 (a)



Img 73 (b)



Img 74 (c)

# Bibliography

---

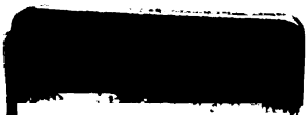
1. Ahmad I. and Grosky W. I. , Spatial Similarity-Based Retrievals and Image Indexing by Hierarchical Decomposition, Proceedings of the International Database Engineering and Application Symposium, Montreal, Canada, August 1997, pp. 269-278
2. Chang S. K. and Liu S. H. , Picture Indexing and Abstraction Techniques for Pictorial Databases, IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 6, Number 4 (July 1984), pp. 475-484.
3. Stockman G. and Shapiro L. Computer Vision, Prentice Hall, Inc. Upper Saddle River, New Jersey, 2001.
4. Information on the emerging MPEG-7 standard, MPE97 [http://drogo.cselt.stet.it/mpeg/faq/faq\\_mpeg-7.htm](http://drogo.cselt.stet.it/mpeg/faq/faq_mpeg-7.htm)
5. W. Niblack, R. Barber, W. Equitz, M.D. Flicknet, D. Glasman, D. Petkovic, and P. Yanker. 1993. The QBIC project: Querying images by content using color, texture, and shape. SPIE Proc. Storage and Retrieval for Image and Video Databases, 173-187
6. Bolle, R., J. Connell, N. Haas, R. Mohan, and G. Taubin. 1996. VeggieVision: a produce recognition system. Proc. IEEE Workshop on Applications of Computer Vision
7. Swain, M J and Ballard, D H (1991) "Color indexing" International Journal of Computer Vision 7(1), 11-32
8. Carson C S, Belongie ,Hayit Greenspan and Jitendra Malik (1997) "Region-based image querying" in Proceedings of IEEE Workshop on Content-Based Access of Image and Video Libraries, San Juan, Puerto Rico, 42-49
9. H. Tamura, S. Mori, and T. Yamawaki (1978) "Textural features corresponding to visual perception" IEEE Transactions on Systems, Man and Cybernetics 8(6), 460-472
10. Liu, F and Picard, R W (1996) "Periodicity, directionality and randomness: Wold features for image modelling and retrieval" IEEE Transactions on Pattern Analysis and Machine Intelligence 18(7), 722-733
11. Ma W Y and Manjunath, B S (1997) "Netra: a toolbox for navigating large image databases" Proc IEEE International Conference on Image Processing (ICIP97), 1, 568-571

- 
12. R. W. Piccard A. Pentland and S. Sclaroff "Photobook: tools for content-based manipulation of image databases" *International Journal of Computer Vision* 18(3), 233-254
  13. Chan, Y and Kung, S Y (1997) "A hierarchical algorithm for image retrieval by sketch" in *First IEEE Workshop on Multimedia Signal Processing* , 564-569
  14. Chen (1996) "Indexing to 3D model aspects using 2D contour features" in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, San Francisco , 913-920
  15. Sven Dickinson, Alex Pentland and Suzanne Stevenson (1998) "Viewpoint-invariant indexing for content-based image retrieval" in *IEEE International Workshop on Content-based Access of Image and Video Databases (CAIVD'98)* , Bombay, India, 20-30
  16. N. Roussopoulos, C. Faloutsos, and T. Sellis. (1988) "An efficient pictorial database system for PSQL" *IEEE Transactions on Software Engineering*, 14(5), 639-650
  17. Gudivada V N and Raghavan V V (1995a) "Content-based image retrieval systems" *IEEE Computer* 28(9), 18-22
  18. Charles E. Jacobs, Adam Finkelstein, David H. Salesin, "Fast Multiresolution Image Querying" *Proceedings of SIGGRAPH 95*, Los Angeles, CA (ACM SIGGRAPH Annual Conference Series, 1995), 277-286.
  19. Ravela, S and Manmatha, R (1998a) "Retrieving images by appearance" in *Proceedings of IEEE International Conference on Computer Vision (IICV98)*, Bombay, India , 608-613
  20. Ravela, S and Manmatha, R (1998b) "On computing global similarity in images" in *Proceedings of IEEE Workshop on Applications of Computer Vision (WACV98)*, Princeton, NJ , 82-87
  21. Rabbitti, F and Stanchev, P (1989) "GRIM\_DBMS: a graphical image database management system" in *Visual Database Systems* (Kunii, T, ed), Elsevier, Amsterdam, 415-430
  22. Qasim Iqbal and J. K. Aggarwal "Retrieval by classification of images containing large manmade objects using perceptual grouping" in *Pattern Recognition* 35 (2002) 1463-1479
  23. David A. Forsyth, Jitendra Malik, Margaret M. Fleck, Hayit Greenspan, Thomas Leung, Serge Belongie, Chad Carson, Chris Bregler (1997) "Finding pictures of objects in large collections of images" in *Digital Image Access and Retrieval: 1996*



- 
- Clinic on Library Applications of Data Processing (Heidorn, P B and Sandore, B, eds), 118-139. Graduate School of Library and Information Science, University of Illinois at Urbana-Champaign.
24. Niels Haering, Zarina Myles, Niels da Vitoria Lobo (1997) "Locating deciduous trees" in Proceedings of IEEE Workshop on Content-Based Access of Image and Video Libraries , San Juan, Puerto Rico, June 1997, 18-25
  25. Minka, T (1996) "An image database browser that learns from user interaction" MIT Media Laboratory Technical Report #365
  26. S. Chang and W. Chen and H. Sundaram (1998) "Semantic visual templates: linking visual features to semantics" in IEEE International Conference on Image Processing (ICIP'98), Chicago, Illinois 531-535
  27. T.O. Binford, Inferring surfaces from images, Artif. Intell. 17 (1981) 205-244.
  28. D.G. Lowe, Perceptual Organization and Visual Recognition, Kluwer Academic Publishers, Hingham, MA, 1985.
  29. H.Q. Lu, J.K. Aggarwal, Applying perceptual organization to the detection of man-made objects in non-urban scenes, Pattern Recognition 25 (8) (1992) 835-853.
  30. Hussain, Digital Image Processing, Ellis Horwood Ltd, England 1991.
  31. Chin-Tu Chen, Jin Shin Chou, Wei Chaung Lin, and CA Pelizzari. Edge and surface searching in medical images. 1988.
  - 32 Ramesh Jain, Rangachar Kasturi, Brain G. Schunck, Machine Vision, McGraw-Hall, Inc. 1995.
  33. M.W. Akhtar and M. Atiquzzaman, "Determination of line length using Hough Transform," Electronics Letters, vol.28, no. 1, pp-94-96, January 2, 1992
  34. M. Atiquzzaman and M.W. Akhtar, "A robust Hough transform technique for complete line segment description" Real Time Imaging, vol. 1, 1995, pp. 419-426.
  35. B.Huet, ADJ.Cross, ER.Hancock, "graph matching for shape retrieval", Neural Information Processing System 98, pp 896-902, 1998
  36. Burak Ozer, Wayne Wolf, Ali N. Akansu, "A Graph Based Object Description for Information Retrieval in Digital Image and Video Libraries", Accepted for publication, Journal of Visual Communication and Image Representation, Academic Press, 2002
  37. Christopher M. Cyr and Benjamin B. Kimia 3D Object Recognition Using Shape Similarity-Based Aspect Graph ICCV 2001

- 
38. Sartaj Shani, "Data Structure, Algorithms, and Application in C++", Mc Graw Hill 1998
  39. Michel Gondran and Michel Minoux, "Graph and Algorithms", A Wiley-Interscience Publication, 1984
  40. Alan George, John R. Gilbert and Joseph W.H. Liu, "Graph Theory and Sparse Matrix Computation", Springer-Verlag, 1993
  41. W.T. Tutte, "Graph Theory and Related Topics", Academic Press New York, 1979
  42. MathWorks Group, "Technical Documentation for MATLAB". The MathWorks, Inc., 1984-2000
  43. Huet B. and E.R. Hancock, "Structural Sensitivity for Large-Scale Line-Pattern Recognition", Third International Conference on Visual Information Systems (VISUAL99), page 711-718, 2-4 June, 1999, Amsterdam, The Netherlands.
  44. Yi Tao and William I. Grosky. Object-Based Image Retrieval Using Point Feature Maps. Proceedings of the 8th IFIP 2.6 Working Conference on Database Semantics (DS8), Rotorua, New Zealand, January 5-8, 1999, pp. 59-73.



MICHIGAN STATE UNIVERSITY LIBRARIES



3 1293 02331 8664