5-2000

Using Raster Sketches for Digital Image Retrieval

James Carswell

USING RASTER SKETCHES FOR DIGITAL IMAGE RETRIEVAL

By

James D. Carswell

B.Tech. (Survey Engineering) Ryerson Polytechnical Institute, 1986

M.S. (Geodetic Science) The Ohio State University, 1988

A THESIS

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

(in Spatial Information Science and Engineering)

The Graduate School

The University of Maine

May 2000

Advisory Committee:

Peggy Agouris, Assistant Professor in Spatial Information Science and Engineering, Advisor

M. Kate Beard-Tisdale, Chair and Professor in Spatial Information Science and Engineering

Max J. Egenhofer, Professor in Spatial Information Science and Engineering

Steven A. Sader, Professor in Forest Resources and Forest Engineering

Anthony Stefanidis, Research Assistant Professor, National Center for Geographic Information and Analysis

USING RASTER SKETCHES FOR DIGITAL IMAGE RETRIEVAL

By James D. Carswell

Thesis Advisor: Dr. Peggy Agouris

An Abstract of the Thesis Presented
in Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy
(in Spatial Information Science and Engineering)
May, 2000

This research addresses the problem of content-based image retrieval using queries on image-object shape, completely in the raster domain. It focuses on the particularities of image databases encountered in typical topographic applications and presents the development of an environment for visual information management that enables such queries. The query consists of a user-provided raster sketch of the shape of an imaged object. The objective of the search is to retrieve images that contain an object sufficiently similar to the one specified in the query.

The new contribution of this work combines the design of a comprehensive digital image database on-line query access strategy through the development of a feature library, image library and metadata library and the necessary matching tools.

The matching algorithm is inspired by least-squares matching (lsm), and represents an extension of lsm to function with a variety of raster representations. The image retrieval strategy makes use of a hierarchical organization of linked feature (image-object) shapes within the feature library. The query results are ranked according to statistical scores and the user can subsequently narrow or broaden his/her search according to the previously obtained results and the purpose of the search.

Acknowledgements

This work was partially supported by the National Science Foundation through grant number SBR-8810917 for the National Center for Geographic Information and Analysis and through CAREER grant number IRI-9702233.

For their support, collaboration and assistance with my thesis, I would like to thank my advisor Peggy Agouris, my research supervisor Anthony Stefanidis, and the reviewing committee members Kate Beard, Max Egenhofer, and Steve Sader.

Thanks to my family for always supporting me in whatever I do and to all the friends who have encouraged me during these years.

And finally, a very special thanks to my wife Michela and baby Nicola for being on my side and believing in my abilities. Her love and patience throughout my career at UMaine has been essential.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Advancements in sensor/scanner technology have resulted in the availability of constantly increasing volumes of digital imagery. The geosciences and spatial information engineering have been greatly affected by this development. Digital photogrammetric applications in particular have become more robust, moving from the experimental use of few images to large-scale projects, which employ numerous images [Carswell and Hasani, 1992]. Indeed, we have reached a point where digital images have practically substituted analog ones (e.g. prints, diapositives, or negatives) as the popular medium for the extraction of precise and up-to-date spatial information like digital elevation models (DEMs), or features with their precise 3-D coordinates. The increased volume of digital imagery necessitates the development of novel methods to efficiently retrieve specific images from large digital image databases. Simple filename-based approaches or querying on metadata information alone are no longer adequate in terms of intelligent data management. This, of course, is due to image filenames usually appearing in the form of photo/flight number ID and therefore not being suitable for conveying image content. Metadata on the other hand are better at describing the contents of an image, but have the requirement of time consuming operator intervention in the database population phase while still not completely accounting for all image-object interaction. Instead, a better alternative would be to base image retrieval on operator queries about image properties and content in its inherent raster domain.

Until now, advancements in digital photogrammetry and digital image analysis for topographic applications have addressed mostly the devising of automated strategies and methods for the extraction of information from digital images [Carswell and Vanderlan, 1993]. This reflects an effort to improve the performance of traditional tasks through automation. However, the use of digital imagery offers unparalleled potential for substantial scientific/practical advancements through the identification of novel tasks and application concepts. Tasks, which were simply not feasible until now, are becoming realistic, and this allows us to revolutionize the way in which the science of digital photogrammetry evolves and practice proceeds. For example, image retrieval from large databases is receiving increased attention the last few years in the computer vision community with quite substantial results [Cohen and Guibas, 1996; Gudivada and Raghavan, 1995a; Mehrotra and Gray, 1995; Ogle, 1995; Pickard and Minka, 1995]. The transfer of such advancements to topographic applications is so far theoretically weak and therefore a challenging yet rather promising task.

In this thesis, theoretical considerations for the retrieval of specific images from large image databases are supported by algorithms developed for user query-by-sketch operations. Together, this is presented with an emphasis geared towards those users of image data in the fields of remote sensing and digital photogrammetry. For example: the intelligence community; mapping agencies; and satellite imaging vendors. Applications for the medical imaging and graphic design disciplines as well as automated updating of geo-databases, automated control point detection and image geo-positioning without control (i.e. using objects of a GIS database) can also be envisioned as possible beneficiaries of such a system.

1.1 Problem Statement

The subject of this thesis is “image retrieval by content in the raster domain”. “Image”, in this sense, is defined as the digital equivalent of an aerial/satellite photograph, and “content” is the shapes of the individual objects that appear in the image. Where “shapes” of “objects” are together defined as the actual boundary edges of the visible features that appear in any given image. The complete term “image retrieval by content” therefore implies that a user will generate a query, i.e. provide a hand drawn sketch (created in any standard PC Paint type program) of a specific *image-object* boundary (also referred to as *feature*), to use as input, and retrieve all the relevant images in the database that contain this query feature.

Since the advent of digital scanners and sensors, beginning in the late 1970’s, [Chang and Reuss, 1978; Zloof, 1975] image database querying has become a major area of research. Most of the efforts during this time have focused on analyzing and comparing the low-level properties of digital imagery. These include: color, in the form of histogram matching; texture, in the form of image coarseness and contrast matching and; composition, where an image is divided into homogeneous regions of color or texture and the relative positions of these regions analyzed [Carson et al., 1997; Flickner et al., 1995; Forsyth et al., 1996; Frankel et al., 1996; Gupta et al., 1991; Ogle, 1995; Pentland et al., 1996; Sclaroff et al., 1997]. While these approaches may be useful for some applications, such as “retrieve for me all the images of jungle” or indeed “retrieve for me all the images of tigers in a jungle”, they are designed to query an image database based on general, low-level image characteristics. They have been shown to perform well with general queries on diverse, multimedia type image databases but they do not

take into account the actual *shapes* of features contained within the imagery. Therefore, such approaches cannot query against the specific object shapes contained within the imagery of a typical aerial/satellite image database. A consequence of this general weakness in dealing with specific image-object shape information is the inability to analyze the differences in scale between identical features, a principal requirement when using raster sketches of query objects for digital image retrieval.

The expression "image retrieval by content" in this thesis means to retrieve images matching the higher level image characteristics, more specifically, actual shape information (synonymous with outline or edges) of features contained within the imagery, e.g. the outline of a building. Furthermore, we are interested in doing so completely in the raster/spatial domains. This distinction is important as both the query sketch and the database images are stored in the same raw raster format without any vectorizing of the image-objects (feature digitizing) performed or other such image-object information known beforehand. The spatial component also implies that there has not been any processing applied to the imagery (such as Fast Fourier Transforms) that transform the raster image into the frequency domain before further operations begin.

The majority of work in the area of feature matching has shown some success through matching the image-objects in the vector domain [Blaser, 1998; Chang, 1997; Cohen and Guibas, 1996; Jagadish, 1991; Mehrotra and Gray, 1993]. This requires that the raster imagery be converted into scenes of vector objects, often together with attributes and other semantics such as topology before they can be queried. As the process of converting raster imagery into vector scenes of objects is not yet fully automated, this task remains tedious and should not be considered as given. Accordingly,

matching in the vector domain is not applicable to many typical users of digital imagery, for example, those concerned with detecting change on the latest imagery available, before it becomes obsolete.

Queries of an aerial and satellite image database must support the information gathering needs of a typical end user. The end user we consider is one who is interested in retrieving raw images containing specific features, in "real-time", from an image database containing enormous amounts (thousands) of digital imagery. Real-time, in this sense, means while the operator waits (i.e. remains on-line) at his terminal while the relevant images are being retrieved. This is opposed to a batch type process that is run in the background or overnight before results are returned.

From a theoretical point of view, to obtain an "optimal" performance in an image information environment, some operations are better or easier done in the raster domain while others are more efficient in the vector domain. Defining spatial relations, for example, the topological (disjoint, touching, overlapping, etc.), directional (above, below, north, south, etc.) and metric (distance) relationships between objects is easier in the vector domain, where properties of the individual objects are known beforehand, [Egenhofer and Herring, 1990; Güting, 1994], while feature matching is usually done in the raster domain [Agouris and Schenk, 1996; Gonzalez and Woods, 1992; Mehrotra and Gray, 1995]. It is well known that the integration of the two domains is still an open problem.

This thesis proposes a novel solution to querying image databases by matching raster features to imagery completely in the raster (and henceforth assumed spatial unless otherwise noted) domain using single features (more specifically their shape) as the sole

matching primitive. The core matching module is a revised least-squares feature matching algorithm developed to accomplish the matching process on binary (black & white feature outline) images. This is accompanied with a novel development of a feature library that organizes and links query objects with their images in the database, thus enabling the retrieval of relevant imagery with the operator on-line, i.e. with an acceptable minimal amount of wait time. The feature library allows us to reduce the search space of a query from a database of images to an abridged group of features. Due to the dynamic nature of the image database, it is also required that the feature library be autonomous in that it be able to automatically maintain its contents and links to the image database.

Our purpose is to extend existing theory on querying image databases to include user-generated sketches of the input query feature. A typical user-generated sketch is created via any standard PC Paint program. This type of program, available as standard application software with all PC operating system software, allows a user to draw (sketch), using the graphic editing tools available, outlines of any design. These outlines include lines and arcs that can be combined to form open or closed polygonal shapes. There is also an option to "fill" the shapes, to make a solid object, but this is not the input required or indeed allowed for the image retrieval environment presented in this thesis. Instead, all that is required as our input query to the image database is the outline (or boundary or edge) of the object created within this simple drawing program. It should be noted that although tools for drawing lines and arcs are used to generate the input query feature, what is actually stored by this drawing program is in fact in raster format. For example, there are no end points to the lines, or interest points on the arcs, all points on

the lines and arcs of the query feature are just individual pixels that are either turned "on" (white) or "off" (black). In fact, it is also possible to draw (sketch) solely by toggling the pixels, that together make up the outline or edge of the input query feature, on and off individually. The final format therefore of this user-generated sketch of the input query feature is termed "binary-raster". Binary, because there are only two allowable states that the individual edge pixels of the sketched query feature can exist in, i.e. either they are on or off, and raster because every pixel in the sketch has its physical location within the sketch, together with its state, stored individually in the final query feature (also referred to as "query template") file.

To date, there have been few proposals to query image databases utilizing user provided sketches of image-objects as the basis for the query in the raster domain [Agouris et al., 1999a]. Most have either relied on matching previously vectorized objects [Blaser, 1998; Mehrotra and Gray, 1993; Nishida, 1999], or on matching Fourier descriptors in the frequency domain [Daoudi et al., 1999; Gadi et al., 1999; Kauppinen et al., 1995; Persoon and Fu, 1977]. Thus, this thesis differs from the current research by extending existing image retrieval theory to allow for user provided sketches of the outlines or edges of features as the basis for the query and to carry out the matching on binary imagery (rather than on color, or gray scale pixels, or sets of coefficients). Also, the image retrieval environment described in this thesis differs significantly from existing work in that it does not require that the images be manually or semi-automatically preprocessed in any way. However it should be noted that pre-processing on the imagery has been performed in a fully automatic fashion in batch mode where image-edge files were created in the background at the time of initial image population of the

database. Additionally it must be said that had the images been manually pre-processed, they could have been incorporated into the image retrieval environment described in this thesis. In fact, improving the imagery manually or semi-automatically by further enhancing their image-object boundaries, removing spurious edges or other such "noise" through the use of various image processing techniques would have resulted in improved matching consistency, as will be shown later.

User provided sketches of features, created within any standard PC Paint program, allow for graphical queries and give more freedom and power to the user. This graphical functionality for sketching is available in addition to the user's capability to extract existing features from database imagery or the feature library during the query building process. The assumption of large, dynamic image databases (containing many thousands of images with new imagery being added daily) of large imagery (many megabytes per image) to be queried is also a theoretical requirement and is addressed through the feature library indexing method mentioned earlier. It will be shown that it is far more effective to match query sketches to a linked library of previous query templates than to each individual image in the database, particularly where there is a requirement for real-time results.

The approach proposed by this thesis is initially designed to query for individual, man-made objects (e.g. airplanes, buildings, etc.) but does propose (in the Conclusions, Chapter 7) options for extending its capabilities to include matching configurations of objects (e.g. an airplane next to a building, multiple buildings, etc.).

1.2 Research Questions and Objectives

Some fundamental questions that this research will attempt to answer include:

- is it possible or not to query image databases based on object shape completely in the raster domain and;
- can the retrieval of images, which match the query feature, be accomplished effectively in real-time (i.e. on-line).

As there are no proposals to answer such fundamental questions in the current literature, they will be identified as the first two primary objectives of this research.

Another question to be answered is;

- can an image retrieval environment be developed to work with a database of aerial/satellite images.

If so, it will need to address the issue of working with extremely complex scenery containing variable scales and high and low contrasting foregrounds and backgrounds together with sometimes very noisy, and often blurry and occluded image-objects. At present, the majority of image retrieval systems (see Chapter 2) require images with well defined objects placed on uniform backgrounds or images without scale differences between them. None of the current systems propose to tackle this difficult problem of matching the inherently noisy and scale, tone, texture, color varying properties of aerial and satellite imagery.

Accordingly, while there is one primary goal of this thesis, namely the development of a query environment capable of using raster sketches of object shape for digital image retrieval, there are three primary objectives to be met on the road to this goal:

- **Objective 1 –** To develop a feature matching algorithm that can take as input a user provided raster sketch of a particular imaged object, e.g. the outlines (edges) of a building. The algorithm should systematically match this sketch to a database of images and return a prioritized list of images that contain this sketched feature.
- **Objective 2 –** To develop an algorithm that will organize the user-generated sketches in an efficient and structured manner. The image database querying should be optimized through the development of this hierarchical (tree like) *feature library* that contains an exhaustive but independent and organized grouping of input query features. These features should be linked to their relevant images within the database, and therefore should allow for effective, real-time querying of the image database.
- **Objective 3 –** To combine the first two objectives into an implementation where optimization testing of matching strategy combined with feature library organization can be compared and analyzed.

1.2.1 Thesis Hypothesis

The hypothesis for this thesis states that:

- The proposed modified least-squares method for matching sketched object shapes outperforms traditional least-squares matching on raw binary-raster imagery.

Traditional least-squares matching, where the conjugate position of a query template window is determined within an image window, is designed to function on gray-scale raster imagery where every pixel contains information. The approach it uses for determining the direction in which to shift the query template within the image requires that the gray-scale gradients at each pixel location are analyzed and compared between

the two windows. As every pixel contributes information to aid in this decision, it has been shown to be a very robust method for achieving this purpose. However, it has also proven to be a very computationally intensive solution to the problem of image matching, requiring good initial approximations of query template positioning within the image [Carswell, 1988; Greenfeld and Schenk, 1989; Agouris, 1992]. For this reason alone, traditional least-squares matching is not used on raw imagery, i.e. where neither image-object information nor any interior or exterior orientation parameters are known a-priori, as it would require stepping through the image pixel by pixel to ensure the "true" conjugate position is found. Therefore, to use this approach for querying an entire image database, where the user is waiting on-line for a prioritized list of query results, is not practical.

Other shortcomings of the traditional least-squares approach include its inability to handle occlusions of image-objects due to its inherent nature of query template re-positioning within the image where it expects every pixel to contribute in the final decision concerning the shift direction. Contemporaneously, typical shift distances calculated through traditional least-squares are in the order of fractions of a pixel, which implies of course that large corrections cannot be accommodated. For implementation on binary-raster imagery therefore, where the input query criteria consists of user provided sketches that do not contain image-object information at every pixel location, traditional least squares will fail due to this deficiency.

1.2.2 Research Challenges

To prove the hypothesis put forth by this thesis, two major research challenges will need to be addressed. Namely, we will need to show that:

- querying an image database can be performed in the raster domain with user-generated sketches as the input query template.
- effective, real-time querying of raster imagery is possible through the creation and organization of a hierarchical library of features.

The main reason that the first of these two challenges exists is because of the identifiable lack of theoretical research in this area. As mentioned previously, most of the current work relies on extensions into the vector domain, which implies intensive operator intervention in the image pre-processing stage before a single query can be made. The criteria for testing whether this challenge has been met will be through the analysis of the query results. For example, does the developed algorithm in fact return the correct images to a users query?

The second challenge exists because it differs from current thinking where raw raster image files are considered inefficient for on-line querying. It is important because it proposes that large databases of image files, linked together with an intelligent feature indexing strategy, can be queried in real-time, i.e. while the operator waits on-line. The criterion for testing whether this challenge has been met will be through the analysis of the feature library itself. For example, does the feature library structure itself in hierarchical tree-like manner where all unnecessary duplicate query features are removed?

1.3 Motivation and Applications

The major motivation for this research is the lack of theoretical solutions to the current problem of retrieving images from a large, dynamic aerial/satellite image database using the shape of user-generated sketches as the matching primitive. Much

effort has been spent on indexing and otherwise optimizing the querying of textual databases [Frankel et al., 1996; Stonebraker, 1990], and recently progress is being made in the area of querying configurations of objects in vector GIS databases through the analysis of their spatial relations [Blaser, 1998]. In the raster world there have been successful implementations of color/textural based image retrieval systems [Flickner et al., 1995; Kelly et al., 1995] but until now, there have been no proposals specifically intended for retrieving aerial/satellite imagery based on queries of the shape of objects contained within them, completely in the raster domain.

Another motivation for this research was to provide the theoretical foundation for potential application to the wider array of disciplines outside the geo-oriented arena. Visual information management is a concern to any industry that utilizes photography in any of its many guises. For example, in medical imaging where various scans of internal organs can be queried against for specifically shaped abnormalities, or for the graphic design industry where images containing certain objects or configurations of objects are required for a particular advertising project.

The intended audience for this specific research comprises the remote sensing and digital photogrammetry communities. Together, they are dealing with an ever growing database of aerial and satellite mapping, reconnaissance and other imagery. The types of queries of an organized image database are infinite but a sample query from these users might be to search the database for all the images at a specific scale (e.g. 1:10,000), taken on a certain date (e.g. Aug. 4, 1998), in a particular format (e.g. infrared), of an individual country (e.g. Afghanistan), and containing specific objects (e.g. buildings, tanks, and planes). With the volume of digital imagery increasing daily for these and every user of

photographic information, a "thesaurus orientated" mechanism for organizing and querying this huge raster data store is imperative.

1.4 Major Contribution of Thesis

The two major contributions of this thesis are:

- Performing matching using raster features for queries on a database of digital imagery.

This is contrasted to the color/textural based alternatives presented in the current literature. Also, it extends existing theoretical models to accommodate application to aerial/satellite imagery, which can be very diverse in scale and structure, even for images depicting the same scene. This is an important theoretical advance of this research, i.e. it matches on real images taken under real conditions, not artificial images taken in the lab of simple, well defined scenes.

- The development of the *feature library* – a novel approach for organizing and linking input query sketches to a database of digital imagery.

It is through the feature library that raw imagery is rendered suitable for on-line querying. This is an important point as it diverges from current research in that the idea of querying raw imagery had previously been thought of as not efficiently possible. This research demonstrates that it is not necessary to know any a-priori image-object information about the database. Queries of objects are possible in the raster world without object segmentation, extraction or identification.

1.5 Organization of Remaining Chapters

The remaining chapters begin with a literature review on spatial database querying in the raster domain, giving an overview on the current research status in this field of research (Chapter 2). This is followed by a detailed description of the major scientific contributions of feature matching and feature library indexing in this research (Chapters 3), a detailed look at how the modified least squares matching algorithm handles raster query features (Chapter 4), and a detailed account of the feature library structure and organization (Chapter 5). A presentation of the proposed image query-by-sketch approach and overall experimental results showing the varying degrees of query retrieval efficiency obtained through modifications of the feature matching strategy and feature library organization are presented next (Chapter 6). Finally the conclusions will describe the advantages and limitations of this approach and provide ideas for future developments (Chapter 7).

Chapter 2

Related Research and Background

This chapter reviews the current literature and gives an overview of the related research into general visual information management systems. The information presented here is required background for understanding the current state of the art handling of raster representations of data. It describes the main differences between raster and vector spatial databases, categorizes the existing literature into descriptions of color/text/texture based matching, shape based matching and combinations of these two approaches. It finishes with an overview of the peculiarities with query template matching in raster space. The chapter does not however contain any new work related to the image query-by-sketch approach proposed by this thesis.

2.1 Raster Space

Spatial databases can be divided into two basic groups: those that contain sets of objects in vector form; and those that contain images or image-objects in raster form [Güting, 1994]. Terms to describe these two basic groups range from pictorial or image to geometric, geographical or spatial. The main difference between them is that in vector space, objects are usually well defined, attributed and their spatial relations determined before querying begins, whereas in raster space, typically this is not the case. The distinction is important because when object extents and spatial relations are well-defined a-priori, the operations and techniques for manipulating and querying their properties are different than for raw objects on a raster image. In raster space, where edge enhancement and image-object extraction can be performed (at least semi-automatically), it is typically

done without semantic or other information describing their spatial or functional properties. A spatial database in raster space therefore, in contrast to its vector counterpart, rarely contains semantic information related to the contents of the imagery it stores.

However, when a raster spatial database does contain additional textual, information about the imagery it contains, it is referred to as "metadata". Currently, there isn't a generally accepted standard format that image metadata should take, although an attempt is being made to address this problem [FGDC, 1997]. Metadata therefore, in the sense used in this research, consists of a listing of potential values for a set of attributes which describe general properties of the image itself, but not about any image-object or other details that the image may contain [Agouris et al., 1999b]. These attributes may include such additional information as: date and time of image acquisition; date and time of introduction in the database; scale/resolution; location of the image, expressed in hierarchically arranged geographic entities like state, country, city, etc.; and/or sensor information and/or imagery type, e.g. black & white, color, color infrared, etc. When available, metadata information is used to thin the pool of potential matches before querying by image "content" begins.

Briefly, a digital image is an array of light intensity or brightness and for each x,y location in the image array, there is stored the brightness value. For black and white (8 bit) imagery the range of brightness values are 0 (black) to 255 (white). For color (24 bit) imagery, the ranges stored for the same x,y location are the same (0 to 255) for the three color bands red, green and blue. For a human operator, the different objects and color/texture patterns and semantics contained within an image are quite obvious,

however this object/pattern recognition process is proving to be quite challenging to translate into automated computer code. Semantic information requires feature extraction, segmentation and object and context recognition, none of which approach full automation with today's technology. Current research into this area therefore is quite active, especially within the digital photogrammetry/remote sensing/image processing community and the fields of artificial intelligence and GIS.

The accepted steps in the process from raw image to "intelligent" image are many and most require some operator (i.e. human) intervention [Gonzalez and Woods, 1992]. This can be either in the form of accepting/rejecting an automated image processing algorithm's result (e.g. histogram stretching or feature representation) or manually controlling/performing the process all together (e.g. feature recognition and interpretation). However, for an image query and retrieval system to be effective, it cannot wait for human input/decisions from semi-automatic algorithms, given the amount of imagery flowing into such an environment on a daily basis. Instead, the image database itself must be fully automatic and self-maintaining, as the approach presented in this research is intended to be. This thesis proposes to show therefore, contrary to current research direction, that raw imagery is indeed suitable for on-line querying.

2.2 Image Retrieval in Raster Space

At present, the majority of visual information management systems in the literature can be grouped almost entirely into a single category. That being query by image "content" where the term content refers to image or image-region color, text and/or texture. There are however a few cases that incorporate this definition of content to include shape and fewer cases still where an attempt is made to match on feature shape

alone. One example of this last case defines shape as the intensity surface of an object, derived by the application of Gaussian filters, which, among other things, is impossible for a user to sketch [Ravela and Manmatha, 1997]. Compared to matching image-object outlines, the results of intensity surface matching are very difficult for an operator to determine or interpret. For example, the answer to the simple question "why was a particular result returned to a query and not another", can be quite perplexing. This is not the definition of the term "shape" referred to in this research, where the shape of an object is defined as the outline of its actual visual appearance. In order for the user to interactively sketch queries, this is imperative.

Table 2.1 presents a direct comparison of the defining characteristics between the major visual information management systems (VIMS) described in the current literature. The last entry (Image Query-by-Sketch) in Table 2.1 is the image retrieval approach presented in this thesis.

	Color, Text, and/or Texture Queries	Vector Shape Queries	Raster Shape Queries	Manual/ Semi-auto image pre-processing	Automatic image pre-processing	Aerial/ Satellite imagery	Multimedia type imagery
Chabot	✔			✔			✔
Candid	✔				✔		✔
VisualSeek	✔				✔		✔
Cypress	✔				✔		✔
Jacob	✔				✔		✔
ImageRover	✔				✔		✔
Yahoo Image Surfer	✔				✔		✔
Lyco Media Search	✔				✔		✔
WebSeer	✔				✔		✔
WebSeek	✔				✔		✔
FIBSSR		✔			✔		✔
Nishida		✔		✔			✔
Fourier Descriptors			✔		✔		✔
QBIC	✔	✔		✔	✔		✔
PICTION	✔	✔			✔		✔
PhotoBook	✔	✔		✔	✔		✔
Virage	✔	✔		✔	✔		✔
Image Query-by-Sketch	✔		✔		✔	✔	✔

Table 2.1 : Comparison Between “Content” Based VIMS

2.2.1 Color, Text and Texture Based Matching

Some cases of image retrieval systems automatically extract keywords (metadata) about the imagery through the analysis of the location (URL) where the image is found on the WWW or from the text in which the image is imbedded. Other semantic and/or metadata information in the form of general image color, texture, dimension, file type, size, and date can also be extracted automatically and indeed are used by some image retrieval systems, e.g. ImageRover, WebSeer, VisualSeek, Lyco Media Search, Yahoo Image Surfer, and others [Athitsos et al., 1997; Frankel et al., 1996; Sclaroff et al., 1997; Smith and Chang, 1996]. This definition of image "content" however is not the same as that used in this thesis. For example, none of the above mentioned systems take into account the actual shape of the objects contained within the imagery. This is understandable of course since the process for generating (extracting) the objects (features) from a raw raster image is not straight forward, i.e. not yet fully automated.

Chabot, (http://http.cs.berkeley.edu/~ginger/chabot.html)[Ogle, 1995] by UC Berkley, takes a database orientated approach to image querying and retrieval by incorporating an object-relational database called Postgres (also developed at UC Berkley) that stores both the images and textual data by allowing for user defined functions and data types [Stonebraker, 1990]. The images require manual preprocessing on input in the form of adding descriptive text. This of course can be a problem due to the different ways a photo could be annotated by different users. For example, the same photo could be annotated as a "1968 Fastback Mustang 428" or as a "red car", depending on operator interpretation. Chabot adds color analysis to the already existing keyword descriptor as an additional matching primitive and proposes to add matching based on

texture, shape and line in the future. However, at present, it does not go far enough to enable querying at the feature level.

Candid [Kelly et al., 1995] is a system developed to retrieve images from an image database based on a query image where a "global signature" is first derived from various image features such as localized texture, shape, or color information for both the query and the database images. The resulting query signatures are then compared against each other using probability density functions of feature vectors. Although shape is described as one of the matching primitives, it is not used in the same context as that proposed by this thesis. Here, shape and texture are combined to produce a feature vector that can be compared to texture feature vectors from other database images using Euclidean distance, thereby allowing the retrieval of images with "similar" textures. This approach therefore is not designed to retrieve images that contain features with a particular shape based on their edge outlines. It is more for retrieving images with the same global "look and feel" as the query image.

VisualSeek [Smith and Chang, 1996] is an approach for retrieving images based on color and texture. Regions within the query scene of similar color and texture are extracted and placed within minimum bounding rectangles and are then compared against similarly decomposed images from within the image database. Comparisons are made between the size and color of the extracted regions but not the shape. Further processing determines the spatial relations between the MBRs within the query scene and those contained within the matching candidate images. As shape is not used in this approach, it is yet another "content" based image retrieval system where matching is based on color, text, and texture and is therefore not suitable for the purposes identified in this research.

In [Carson et al., 1997; Forsyth et al., 1996], a new image representation, "blobworld", and a retrieval system, "Cypress", based on this representation is presented. Here, the authors correctly state that there are currently no systems that automatically classify images or recognize objects. Therefore, the authors propose a new method, which uses Expectation Maximization on color and texture jointly to provide an image segmentation as well as a new image representation, and finds maximum likelihood parameter estimates when there is missing or incomplete data. As this is another color/texture based image query system, it is similar to those systems presented previously in that it does not offer the user any capability to query on sketched shapes of image-objects.

Jacob [Ardizzone and La Cascia, 1997], organizes and retrieves by "content" still digital images or digital video sequences from an image/video database. To do this, image and image sequence contents are first described and coded. No user action is required during the database population step as the system automatically splits a video into a sequence of shots, extracts a few representative frames (called r-frames) from each shot and computes r-frame descriptors based on color, texture and motion. Queries based on one or more "features" are possible. The use of the term "features" therefore is not synonymous with that described previously and used in this research and so the matching potential of this system for shape based queries is non-existent.

ImageRover [Sclaroff et al., 1997] is designed as a WWW image search engine together with Yahoo Image Surfer, Lyco Media Search, WebSeer [Frankel et al., 1996], and WebSeek [Smith and Chang, 1996]. It uses search robots to scan the WWW for images imbedded in web documents and to automatically extract image information such

as color and orientation. Each image is divided into 6 regions and checked for this information (color and orientation) in each of these "sub-images" producing a 2x6 image index vector for each image. Work is in progress to extend ImageRover to include texture and faces in the index vector but as yet there are no plans to include shape. Therefore it is unsuitable for detecting isolated features and at present can only be used to search for general patterns of color together with their orientation.

The Yahoo Image Surfer (http://ipix.yahoo.com/) and Columbia University's WebSeek (http://disney.ctr.columbia.edu/WebSEEk) categorize their imagery into classes, e.g. animals, sports, etc. semi-automatically with text labels attached. The text labels of the various classes can be queried as keywords and then the color histograms of the imagery contained within a selected class can also be queried against. No support for feature shape querying is offered and the semi-automatic nature employed to organize the image database is impractical in an evolving image database environment.

Lycos (http://www.lycos.com/picturethis/) and the University of Chicago's WebSeer (http://infolab.cs.uchicago.edu/webseer/) are two other examples of keyword based search systems with the difference in that the keywords are extracted automatically from the image URL or the text imbedded with the document that contains the image. Although no longer on-line (due to the disbanding of the research team), when operational, WebSeer automatically extracted additional information on the image "content" like the image color and dimension, the file type, size, date and whether or not the image contains any (and how many) faces. WebSeer's main design objective was to find and distinguish between web photographs vs. graphics or icons based on these automatic observations. However, it required initial manual training of the system to

determine what is a photo and a graphic, and did not attempt to query on shape whatsoever.

2.2.2 Shape Based Matching

[Gudivada and Raghavan, 1995b] identify two main research directions for content based image retrieval (CBIR). That is the "primitives" approach where object boundaries and centroids are extracted manually a-priori and the "logical" approach where abstract representations of the images in the form of attributes and semantics are also manually extracted beforehand. Although the research in this thesis does use the primitive approach to CBIR, it differs from both these directions by not manually or semi-automatically pre-processing the imagery in the database, for example to extract objects or spatial information. The remainder of this section will describe the methods used for CBIR in those image retrieval systems that propose to match on shape.

One of the better attempts at shape based matching is FIBSSR (Feature Index-Based Similar-Shape Retrieval), described in [Mehrotra and Gray, 1995]. Here the outlines or edges of an object are decomposed into line segments connected by interest points of maximum curvature or vertices. Each line segment and corresponding endpoints are called a "feature" in this context. A group of features in this sense compose a shape and are used to search a shape database looking for shapes (objects) with similar features. This is done by first encoding all the "features" of a shape into a "basis vector" which removes transformation variances such as scale, rotation and translation. For high contrast images of shapes this approach shows promise but for noisy aerial and satellite imagery, too many spurious edges would confuse any automated process designed to extract shape boundaries. Indeed this application is limited to

querying shape databases of simple or articulated objects like scissors or other such tools on uniform backgrounds. Although overlapping shapes are considered by eliminating interest points in the query image that correspond to shapes found by previous queries and then re-generating a new query feature with the remaining points, it does not consider the topology of disjoint shapes, the most common configuration in aerial imagery.

In [Nishida, 1999], a method for image retrieval from image databases is proposed using structural feature indexing. The features are first approximated with polygons and then their convex/concave parts and quantized directions are indexed. This of course requires working in the vector domain so is not directly comparable to working in raster space other than it being another approach at solving the shape based query/image retrieval problem. It does demonstrate however that an indexing strategy increases the efficiency and robustness for shape/index retrieval from image databases.

Much work has been done in shape retrieval using Fourier Descriptors [Daoudi et al., 1999; Gadi et al., 1999; Kauppinen et al., 1995; Persoon and Fu, 1977], which implies working in the frequency domain (as opposed to the spatial domain). The foundation of frequency domain techniques is the convolution theorem [Gonzalez and Woods, 1992]. Here, edges can be accentuated by using a function that emphasizes the high-frequency components of an image. For example, in [Gadi et al., 1999], "fuzzy logic" is proposed as a possible solution to the shape/image retrieval problem because of the inherent difficulties of extracting region/object boundaries. In the spatial domain, automation of the boundary extraction problem is indeed a major impediment to progress in matching on shape information alone and is why color and texture have instead been used

extensively. This is why the approach proposed in this thesis doesn't attempt to first extract feature boundaries but instead matches completely in the raster/spatial domains.

An advantage of using the frequency domain is that Fourier Descriptors are invariant to the basic similarity transformations such as translation and rotation. Hence, the same shape appearing at a variety of positions and orientations, would all yield the same set of descriptors. The shortcoming of this approach however is that it does not scale, so the same object at different scales would yield different descriptors. This poses a problem when dealing with aerial imagery in particular as even adjacent photos can differ in scale due to changes in ground elevation or flying height. Another issue is when images are returned from the query, the user doesn't know exactly why or where the query shape matched the image. This is because the shapes have been previously transformed into coefficients, with obscure meaning to most users.

2.2.3 Combined "Content" Based Matching

IBM's QBIC [Flickner et al., 1995] is designed as an image query system based on image "content" that includes color, shape, texture, sketches, example images and camera and object (image subsets) motion. It is organized into a database population component and a graphical query component. On the database population side, the images are pre-processed to extract "features" (features in QBIC are the properties of either an image or image-subset (object) that describe content) describing their content and then storing these features in a database. On the graphical query side, a sample query consists of retrieving images that match to "find all images that are 30% blue and 15% red". Texture is described as image or image-subset coarseness, contrast and directionality and shape descriptors are area, circularity, eccentricity, major axis direction

and a set of tangent angles around the object perimeter. The images are annotated with text information and the objects are pre-extracted manually or semi-automatically with area flooding and outlining using spline snakes. Automatic object extraction is available for simple images containing easily separable background and (few) foreground objects.

QBIC (http://www.qbic.almaden.ibm.com/) is by far the most referenced system in the literature and indeed has been awarded "best on the web" honors, among others. However, it still falls short of achieving full automation by failing to solve the basic problems of automatic image segmentation and object representation and description, or alternatively proposing a strategy to populate their database without some operator intervention. The automation that QBIC does achieve in extracting objects from their backgrounds for simple scenes is commendable but again is only useful for specific applications, for example, those that do not require aerial or satellite imagery. For this type of imagery, that is arriving in quantity on a daily basis, operator assisted pre-processing is undesirable.

PICTION [Srihari, 1995] is a "content based" image retrieval system where an integrated text/image database of people is queried for similarity. The main application of this approach is to query newspaper photos for human faces. Therefore, "content" in this system refers to whether or not the image contains people. This is determined by passing filters over the image to extract three edges that could represent a face combined with textual analysis of the photo caption. Identifying people in newspaper photos is a very challenging task and this approach does show promise for this application. It is however quite a different problem than identifying features or groups of features from

aerial or satellite imagery that contain no textual captions and an infinite number of possible object shapes.

In PhotoBook [Pentland et al., 1996], the authors provide three general approaches to constructing semantics-preserving representations. These are appearance-specific descriptions applied to face and keyframe databases, texture descriptions applied to texture-swatch and keyframe databases, and shape descriptions applied to hand-tool and fish databases. As the first two approaches do not deal with shape, only the third approach will be explored. The general idea for semantic-preserving image compression is to "first transform portions of the image into a canonical coordinate system that preserve perceptual similarities, and then to use a lossy compression method to extract and code of that representation". When searching for objects, this approach uses variations on the Karhunen-Loeve transform to derive optimally compact representations for either appearance or shape [Jagadish, 1991]. This transform is known to provide an optimal-compact linear basis for a given class of signal. In addition, this transform uses the eigenvectors of the covariance matrix of the set of image features. These eigenvectors can be thought of as a set of parametric variations from the mean or prototypical appearance that altogether characterizes all of the variations between images of the object and the object's prototypical appearance. For shape of image, eigenvectors describe the intrinsic symmetries for the object in a unique and canonical manner. The transformed shapes used by this system were in fact a rectangular approximation of the true shape of the object and did not handle rotation, occlusion, or the composition of multiple shapes. Also, the system was implemented on a database of tools and fish outlines on a uniform background. Thus, determining rectangular approximations and eigenvectors for objects

in noisy aerial images containing many edges from different objects would find this approach infeasible.

[Bach et al., 1996], describes the Virage image search engine as a general image retrieval system, it mainly focuses on the visual features of color, texture, and shape [Flickner et al., 1995; Pentland et al., 1996]. The definition of shape in Virage therefore is approximated similar to PhotoBook above, i.e. with rectangles. Virage is an indexable collection of vectors for data types, with a collection of vectors representing a single category of image information called a primitive. A primitive is a semantically meaningful feature of an image. In this system, image indexing is performed after several preprocessing operations, such as smoothing and contrast enhancement. Each primitive-extraction routine takes a preprocessed image, and depending on the properties of the image, computes a specific set of data for that primitive. A vector of the computed primitive data is stored in a proprietary data structure. Since different computational processes extract primitives, they belong to different topological spaces and each has different distance metrics defined for them. A primitive encompasses a given feature's representation, extraction, and comparison function. When a query is submitted, the search system computes the one or more similarity distances for a pair of primitive vectors. This is performed in two steps; for each primitive, a similarity distance is computed, and the similarity distances are combined with weights by a distance function that forms a final score that is used to rank results by similarity. Virage does not support spatial relationships among image-objects and due to its scale dependent shape approximating rectangles, does not fully realize the users need for a truly sketch based image query system.

2.3 Image Matching Techniques

When matching pixels of raster data from a query sketch to a database image, traditionally there have been three fundamental approaches: that is area-based matching where the gray levels of the query patch are used as matching primitives; feature-based matching where image characteristics are used as the matching primitives; or hybrid techniques that combine the strengths of the first two methods. [Doorn et al., 1990]

2.3.1 Area-Based Matching

Gray level or area-based matching involves the extraction and precise matching of conjugate windows or patches of pixels. These patches (one from the input image and one from the database image) are in the form of two dimensional arrays of gray scale values. The best match between these two patches is where their collective difference in gray values is minimized [Ackerman, 1984].

One method to digitally correlate the images is to directly compare the sums of the squares of the density differences of corresponding pixels [Gruen and Baltsavias, 1987]. When the gray levels of these corresponding pixels are near the same value then the conjugate position has been found.

A drawback to this method is that corresponding pixels will have different radiometric and possibly geometric qualities even if they represent the same image-object. They will have these differences because the two images were not taken from exactly the same exposure station at exactly the same moment in time. Therefore the contrast and brightness of the same image-object point and the geometry of same image-object outline will never be exactly the same from image to image. Distortions between the two patches, due to differing image perspectives and illumination, are therefore

modeled by geometric and radiometric transformations. Typically, an affine transformation is used to describe the geometric relationship (rotation, translation, scale) combined with added parameters to eliminate the radiometric differences.

Another limitation of area-based matching is that it requires good approximations for the initial patch locations [Baltsavias, 1991]. The "pull-in" range of this approach is defined as the maximum distance the input patch can be initially positioned in the image patch, from its true conjugate position, and still be able to converge on its correct match. In cases where the exterior orientation parameters of an image are known a-priori, this range can be accommodated. But where raw imagery is used, finding conjugate matches using this technique is extremely costly from a computational standpoint, even for small input patches, as sequential stepping through the entire image is necessary to ensure all potential conjugate positions have been tested.

2.3.2 Feature-Based Matching

Feature-based matching is an approach to simulate how a human photogrammetric operator extracts and processes information from a digital image and therefore is a model based on a matching strategy that proceeds from *coarse* to *fine* [Greenfeld, 1987]. For example, a human operator, given the chore of matching an input image with a database image, would at first view the two images independently and then together to get a general feel for their contents and alignment. The operator would then determine if the two images share any common ground, followed by matching up any of their prominent features before moving on to the analysis of any minor details the imagery might contain in common.

Feature-based matching accomplishes the above by matching specific "features" of an input image to those of a database image through increasing levels of pixel resolution, with matches at coarser resolutions transferred to and refined at higher levels of resolution [Greenfeld, 1987]. Features in this sense include interest points, edges and corners which are extracted using specially designed operators for each feature type. Lists of features with their locations, strengths and orientations are created and maintained for the input and database images respectively. Therefore, unlike area-based matching, good initial locations are not required before matching can begin. With geometric and/or statistical tests, the lists of features are sequentially compared and the best matches recorded [Agouris, 1992; Forstner, 1986; Grimson, 1985]. Multiple and/or erroneous matches are eliminated through a global consistency check using the image coordinates of matched features and a priori information on the exposure geometry of the two images. Unfortunately, this pre-knowledge of image/feature orientation/position is not known in a raw raster image database, or for randomly orientated, user-generated raster query sketches. Subsequently, although this technique has proven useful for automatically creating DTM's from digital stereo pairs of aerial imagery, it is not directly suitable for the application intended in this research.

2.3.3 Combined Area/Feature-Based Matching

The two matching methods described briefly above both have advantages that can be exploited in a combined area/feature-based matching approach. Here, feature detection can be used, on previously oriented imagery, to identify conjugate areas rich in information. Once located, these areas provide the required good initial approximations

for further refinement with area-based matching [Carswell, 1988; Greenfeld and Schenk, 1989]. Table 2.2 presents a direct comparison of these two techniques [Agouris, 1992].

Area-Based Matching	**Feature-Based Matching**
Good initial approximations are necessary	Good approximations are not required
High precision	Lower precision
Low reliability: susceptible to erroneous matches if wrong initial approximations	High reliability: matched pairs are most likely truly conjugate features
Can produce a dense regular grid of matched points	Matched points have a sparse and irregular distribution
Sensitive to geometric distortions and radiometric noise	Less sensitive to geometric distortions and radiometric noise
Ambiguous matches in areas of low contrast or repetitive texture	

Table 2.2 : Comparison Between Area-Based and Feature-Based Matching Methods.

2.4 Least-Squares Matching

This research uses the idea of least-squares matching similar to that used in the area based matching case in that it is based on the analysis of dissimilarities between an input query template (user-generated sketch) and an image-edge file window. When a query template is compared to an image-edge file, the template pixels vote to stay put (or move) according to their similarity (respective dissimilarity) to corresponding image-

edge pixels. Moves can be performed in the +/- x and y directions, in one of 5 options: left, right, up, down or stay put. This resembles the comparison of gray values in least squares matching and the use of image gradients to identify shifts, rotations, and scaling. Similar to the traditional least-squares approach, the final solution is obtained after a set of iterations, and by analyzing voting patterns, we can accommodate occlusions. The main difference between the feature matching approach in this thesis and traditional least-squares is that the matching is carried out on edge information from binary imagery only, instead of on gray scale differences between patches of pixels. As such, the conventional least-squares approach, where every pixel is considered, is not strictly necessary as only those pixels where information (i.e. the edge pixels) exists need to be considered. Therefore, we avoid computationally expensive matrix manipulations and the requirement for good initial positioning of the query template within the image-edge patch. An overview of least-squares matching however is included for convenience.

Traditional least-squares matching offers a robust method for establishing correspondences among image windows. Its mathematical background, based on least-squares principles, permits its successful extension for application in a multiple image matching scheme [Agouris and Schenk, 1996], or even for the establishment of correspondences in sets of 3-dimensional images [Maas et al., 1994].

Assuming *f(x,y)* to be the reference query edge template and *g(x,y)* to be the actual image patch, a matching correspondence is established between them when

$$f(x,y) = g(x,y) \qquad (1)$$

However, considering the effects of noise in the actual image, the above equation becomes

$$f(x,y) - g(x,y) = e(x,y) \tag{2}$$

with *e(x,y)* being the error vector.

In a typical least-squares matching method, observation equations can be formed relating the gray values of corresponding pixels. They are linearized as

$$f(x,y) - e(x,y) = g^o(x,y) + \frac{\partial g^o(x,y)}{\partial x} dx + \frac{\partial g^o(x,y)}{\partial y} dy \tag{3}$$

The derivatives of the image function in this equation express the rate of change of gray values along the *x* and *y* directions, evaluated at the pixels of the patch. The two patches are geometrically related through an affine transformation

$$x_i = a_{11} + a_{12}x + a_{21}y \tag{4}$$

$$y_i = b_{11} + b_{12}x + b_{21}y \tag{5}$$

The affine transformation parameters are the unknowns, which allow the repositioning of the image window to a location that displays better radiometric resemblance to the reference template. They are introduced in the derivative terms $(\partial g / \partial x, \partial g / \partial y)$ of the linearized observations above as

$$f(x,y) - e(x,y) = g^o(x,y) + g_x da_{11} + g_x x_0 da_{12} + g_x y_0 da_{21} + g_y db_{11} + g_y x_0 db_{12} + g_y y_0 db_{21} \tag{6}$$

The resulting observation equations are grouped in matrix form as

$$-e = Ax - l \quad ; \quad P \tag{7}$$

In this system, l is the observation vector, containing gray value differences of conjugate pixels. The vector of unknowns x comprises the affine transformation parameters, while A is the corresponding design matrix containing the derivatives of the observation equations with respect to the transformation parameters, and P is the weight matrix. A least-squares solution allows the determination of the unknown parameters as

$$\hat{x} = (A^T PA)^{-1} A^T Pl \tag{8}$$

Through the adjusted transformation parameters we determine a new position in the image as conjugate of the template. The robust mathematical foundation of least-squares matching allows us to obtain meaningful statistical measures for the accuracy of the matching process.

The a posteriori variance of unit weight

$$\hat{\sigma}^2{}_o = \frac{V^T PV}{df} \tag{9}$$

is an excellent measure of the overall accuracy. The residuals vector V expresses the deviation between observations and their adjusted values (and is actually an evaluation of e in Equation 7 once we obtain estimates of the unknown parameters x). The degree of freedom (df) of the system expresses its redundancy and equals the difference between formed equations and unknown parameters. An expression of the a posteriori variance is used as a score index reflecting the confidence level of a match.

The above system can also be increased to incorporate additional parameters. By taking advantage of the diffusion equation of the Gaussian function [Lindeberg, 1994], according to which

$$\frac{\partial g(x, s_x)}{\partial s_x} = \frac{1}{2} \frac{\partial^2 g(x, s_x)}{\partial x^2} \tag{10}$$

the derivative with respect to the scale parameter is equivalent to the second derivative of gray values with respect to the spatial coordinate, allowing us thus to directly introduce it in the linearized least-squares matching observation equations (with the second derivatives of gray values as corresponding coefficients in the Jacobian matrix A of equation 7). Thus, first derivatives of gray values express positional/rotational

differences between two conjugate windows, while second derivatives express scale differences among them.

2.5 Tree Data Structures

Tree structures are typical approaches to organize data into some form of a hierarchical index that facilitates efficient information search and retrieval combined with the desire to save storage space by aggregating data having same or similar values. A tree is a collection of elements called "nodes" along with a relation ("parenthood") that places a hierarchical structure on the nodes [Aho et al., 1987].

The simplest of the tree structures is known as the binary search tree. It is used for representing large sets of elements that are ordered in some linear order. The important property of a binary search tree is that all elements stored in the left sub-tree of any node x are less than the element stored at x, and all elements stored in the right sub-tree of x are greater than the element x. This storage property of binary search trees makes testing for membership in the set simple. For example, to determine whether x is a member of this set, first compare x with the element r at the "root" of the tree, i.e. the parent element. If $x=r$ we are done and the answer to the membership query is "true". If $x<r$ then x can only be a descendant of the left child of the parent, if x is present at all. Similarly, if $x>r$, then x could only be a descendant of the right child of the parent. (Figure 2.1)

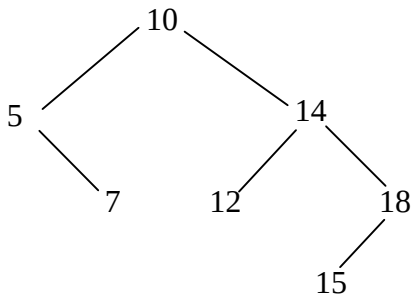


Figure 2.1 : A Binary Search Tree

It can be seen from such a structure however that one side of the tree can have far fewer nodes than the other. This is due to the time of element insertion into the tree. For example, if we are representing the set of real numbers between 0 and 100 and the 1st element we insert in to the tree is the number 10, then the right side of the tree will end up being far more populated than the left. This results in unequal search and retrieval times when future queries, insertions or deletions are performed.

To counter this, the balanced tree implementation of element sets is used. For example, if we have a tree of six nodes, as in Figure 2.2, and want to insert the element 1, instead of inserting it such that four elements reside to the left of the root element and two elements reside to the right of the root element, a re-ordering of the tree is performed. The tree will then contain an equal number of elements on either side of the root element, which allows for equal search and retrieval times for future queries (Figure 2.3).

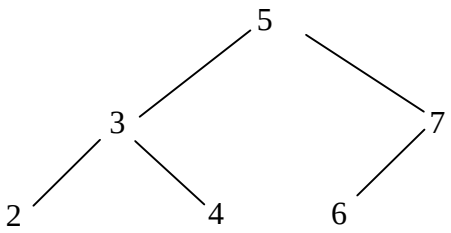


Figure 2.2 : Unbalanced Binary Tree

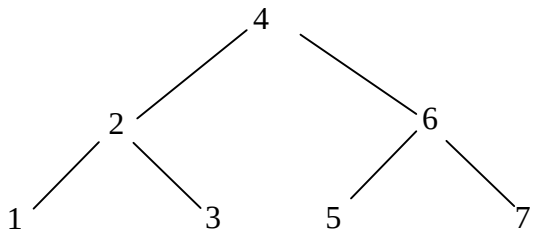


Figure 2.3 : Balanced Binary Tree

Therefore, even for the simplest of tree index structures, the order or temporal aspect of element insertion is important and re-ordering is required to balance the tree for optimal query processing. In this research, we differ from the standard tree structure above in that we have multiple "root" (parent) nodes (in fact, multiple nodes at every level in the tree). The multiple parent nodes arise because of the range of values acceptable at each node, instead of using a single value to determine an elements position within the tree. As will be described later, this results in a pronounced affect regarding the final tree structure, due to this temporal aspect of feature insertion, and subsequent tree balancing requires far more complex element re-ordering operations.

The term "quadtree" is used to describe a class of hierarchical data structures whose common property is that they are based on the principle of recursive decomposition of space [Samet, 1990]. The prime motivation for the development of the quadtree is the desire to reduce the amount of space necessary to store data through the use of aggregation of homogeneous blocks. An example of a quadtree representation of data is for a two-dimensional binary image where successive subdivisions into quadrants, sub-quadrants, etc. are performed until blocks are obtained that consist entirely of 1s or entirely of 0s.

Variations of the basic quadtree concept have been proposed to store, query, and retrieve different kinds of information. For example, point quadtrees are used for sets of points where non-uniform cell subdivisions are made continuously until only one point resides within each cell [Finkel and Bentley, 1974]. Point quadtrees were initially defined to represent point data in two dimensions but can be used for any (k) dimension. However for higher dimensions, k-d trees are usually used to avoid the large branching factor [Bentley, 1975]. Similar to the simple binary tree approach, both the point quadtreee and the k-d tree have the property that their shape is dependent on the order in which the points are added to them.

As an example of an application for k-d trees, in [Beis and Lowe, 1997], feature vectors are generated for objects and an index structure created based on the k-d tree approach. The goal was to recover from the index the most similar model shapes to a given query image shape. Random viewpoints were used to generate approximately 103 images per object in the database. From these images, groups of 5 edge-length ratios and 5 angles between the edges were extracted and stored in 10-d feature space. Attempts to identify a query object then began by extracting similar edge/angle representations from the query object and searching through the database until the nearest neighbor to this particular grouping of angles/edges is found, thus identifying the query object. This approach showed promise in search and retrieval time but because of its 3-d objects and the preprocessing overhead of extracting (i.e. vectorizing) edges and angles, it is not the same problem considered in this thesis, where no image-object vector information is known a-priori.

2.6 Summary

This concludes the chapter on background information. It began with a description of raster space and an overview of the image retrieval systems found in the current literature. In general, most of these systems can be grouped into the category of "content" based image retrieval where the term content refers to a combination of image color or texture. The remaining systems belong to a second group where manual preprocessing of the database imagery is required, in the form of vectorizing the image-objects, before database queries can be performed. This was followed by an overview of image matching in raster space with an emphasis placed on the least-squares approach. The chapter finished with a brief introduction to tree data structures and how they are used to store data for nearest neighbor queries. Beginning with Chapter 3, the new work related directly to this research is presented.

Chapter 3

An Approach for Digital Image Retrieval Using Raster Sketches

From the previous chapter, it can be seen that the typical image database query systems developed to date are either color/text/texture based systems or require substantial manual effort during the image pre-processing stage. These requirements, while quite useful, limit the query system to retrieving images based on lower level image "content" rather than on higher level shape information and/or to retrieving images that have been manually pre-processed in some way (e.g. to extract image-object information). In contrast, this thesis introduces an approach that is not encumbered by either of these restrictions and therefore is unique in its design. The remainder of Chapter 3 will describe the scientific contribution and theoretical foundation of this new and innovative work.

3.1 Objective & Contribution

The main objective of this chapter is to give a brief overview of the complete image query and retrieval environment proposed by this thesis. More specifically, it will highlight on each of the three individual components, namely the image library, the metadata library, and the feature library, that together make up the comprehensive digital image database. Further, the linking between each of these components and the need for both off-line and on-line matching will be discussed. Finally, a description concerning the flow of information and decision making for a typical database query will be presented.

The major scientific contributions of this thesis lie in the feature library indexing strategy and in the development of a feature shape-based similarity matching algorithm. These two fundamental innovations replace the traditionally slow and computationally intensive least-squares approach for object similarity matching in the raster domain, and introduce a structured feature library mechanism that facilitates efficient on-line querying of raw imagery.

The shape-based querying mechanism is also distinct from the color/textural based alternatives presented in the current literature since it uses actual image-object boundary outlines as its sole matching primitive. This allows for more specific queries on image content, i.e. on the shape of actual features depicted in the scene, and not just on a general description of image color or texture. Furthermore, it applies this capability to aerial/satellite images, which can be very diverse in scale and structure, even for images depicting the same scene. This is an important aspect of this research as it moves away from lab imagery where objects are typically well defined compared to their background and therefore imposes certain theoretical challenges. For example, we have to consider rotations, scaling and translations of the query template on traditional aerial/satellite imagery complete with the usual image noise and subsequent spurious edge information.

A matching algorithm developed for an aerial/satellite image query environment will need to be versatile in that it be able to handle both the comparison of query input to existing lists of previous query templates, and to the actual image-edge files themselves. The main difference between these two requirements being the template shifting strategy employed, which has a direct influence on the time required to locate a conjugate match

and subsequently on the algorithms ability to retrieve correctly matched images in real-time.

The structured feature library is another major contribution that is not presented elsewhere in the literature. It is a novel approach for organizing image-object data, in the form of previous shape queries, and through it shows that raw imagery is indeed suitable for on-line querying. This is an important point as it diverges from current research in that the idea of querying raw imagery had previously been thought of as not efficiently possible.

This research proposes therefore that it is not necessary to know any a-priori image-object information about the database. Queries of image-objects are possible in raster space without object segmentation, extraction or identification. Chapters 4 and 5 are devoted to describing in more detail these two fundamental innovations and requirements of a shape-based image query and retrieval environment.

3.2 Theoretical Model of an Image Query and Retrieval Environment

Image information systems incorporate many individual components and processing stages. For example, Chang [Chang, 1985] refers to a typical system (an approach which we are not adopting) as one that consists of three components. The first component is the raw image, which is analyzed and image-objects recognized through image enhancement, normalization, segmentation and pattern recognition techniques. The next component in this system would convert image-objects into image knowledge structures, i.e. assign topological, direction and distance measures to them. Finally, the third component requires domain knowledge specific to each user application that enables intelligent interrogation and use of the data. This ideal system proposed by

Chang uses feature indices, i.e. "hot spots" within the image with semantics attached thereby resulting in "smart images". The user would merely roll his cursor over the image and obtain all sorts of attribute information and other properties concerning the image area in question. To accomplish this, he proposes super computers performing feature extraction work with massively parallel architecture to support the generation of these active feature indexes. However, ideas envisioned with today's technology in mind and not requiring next generation or otherwise unobtainable hardware (to the average user) are more likely to be realized and used in practice.

The first component describing the image information system above could be further subdivided and described as having eight fundamental steps [Gonzalez and Woods, 1992]. These include:

- Image acquisition, where digital cameras/sensors or conventional cameras and scanners are used;
- Preprocessing, where imaging techniques like contrast stretching, noise removal, edge enhancement, etc. are employed to improve image appearance to ensure success of future processing steps;
- Segmentation, where the image is partitioned into homogeneous regions;
- Representation & description, where edges/features are extracted;
- Recognition & interpretation, where the extracted features are identified/labeled/attributed;
- Processing, where various software used for analysis/classification is run;
- Communication, which involves the transferring of data between systems and users; and finally

- Display, where media like slides, monitors, photos, transparencies, etc. are used to convey the results.

As most of these eight steps, together with the final two components of the image information system mentioned previously, are not yet fully automated, i.e. they require at least some operator intervention, they are not included in the design approach taken by this research.

The intention of the approach proposed by this thesis therefore is to function fully automatically in the raster domain, i.e. without operator intervention at any stage, except for when composing a query to the comprehensive digital image database itself. For example, in the image acquisition step, it is only assumed that the image is in digital format before it can be processed in the proposed image query and retrieval environment of this thesis.

The only required pre-processing steps on the digital imagery in this proposed environment are that a generic edge enhancement filter is applied to each image and the resulting image thresholded such that only black or white pixels remain (values 0 or 255). These two steps are required to produce the edge-image representation (of a given raw image), on which the shape queries are processed. The edge-image therefore will contain only the boundary outlines of the image-objects inherent to its corresponding raw image. The two pre-processing steps involved in generating the edge-image are fully automated with today's technology and are performed in a batch process (e.g. in the background) as each new raw image is inserted into the database. Together, they constitute the only pre-processing requirements to populate the proposed comprehensive digital image database before querying can begin.

There are many reasons for fully automated and self-maintaining image query systems. They include:

- Decreased cost of setup and operation;
- Ease of use and maintenance and;
- Speed of bringing new database images on-line.

The main driving force behind the level of automation in this research is that it is designed for the imaging user with hundreds or even thousands of digital images arriving from various sources on a daily basis with temporal necessity the overriding priority. In this case, the workload involved in pre-processing the imagery manually in any way, before being able to insert it into the database or to query its contents, is too time prohibitive.

Converting image-objects into image knowledge structures, i.e. storing objects along with their topology, direction, and distance, is not necessary in this proposed approach. Where some approaches begin at the object level, i.e. they require that image-objects be pre-identified (i.e. vectorized) before the image's contents can be queried, we try to go lower to the image level and work solely in the raster domain. However, it should be noted that vector shape queries on image content could potentially be accommodated by the approach proposed in this thesis by first converting the vector query shapes to raster form.

The remainder of this chapter will discuss the architecture of the raster query and image retrieval environment proposed by this thesis. The components that make up this environment include:

- The image library,
- The metadata library,
- The feature library,

} Comprehensive Digital Image Database

- Off-line query processing,
- On-line query processing.

} Query Interface

} **I.Q.**

Together, these components describe the environment where feature queries on the comprehensive digital image database are performed. Henceforth, this environment, i.e. the approach presented in this thesis for using raster queries for digital image retrieval, will be given the abbreviated term, **I.Q.** (**I**mage **Q**uery-by-Sketch) (Figure 3.1).

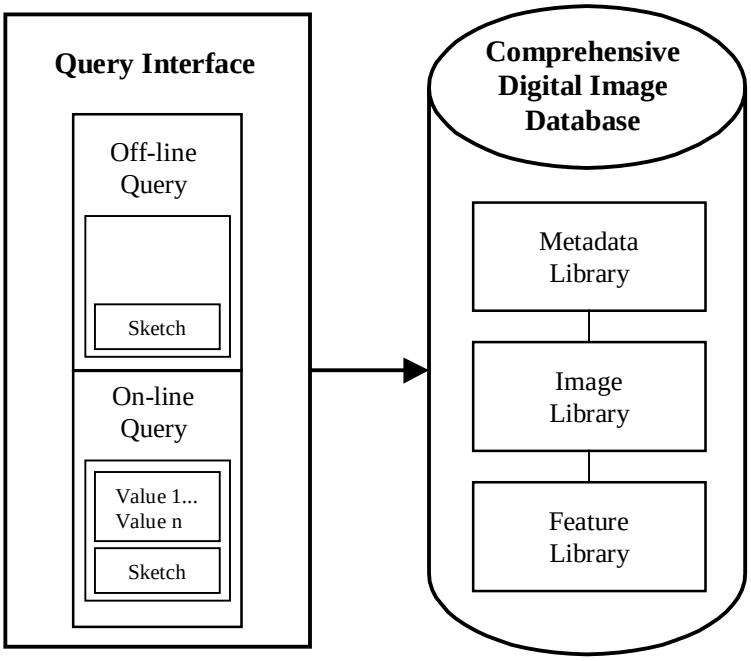


Figure 3.1 : The I.Q. Raster Query and Image Retrieval Environment.

3.3 Image Library

The image library component of **I.Q.** (highlighted in Figure 3.2) will contain an assortment of black & white (8 bit) digital aerial/satellite images. As each new image gets inserted into the database, three automatic pre-processing steps will be performed. The result is that an original raw image gets inserted along with its edge representation image. Both images are linked one-to-one and together comprise one entry in the image library (Figure 3.3). They will be stored within the image library in separate directories and have similar root filenames, e.g. a typical raw image filename would be *253482.tif* and its edge-image counterpart would be *253482_edg.tif*.

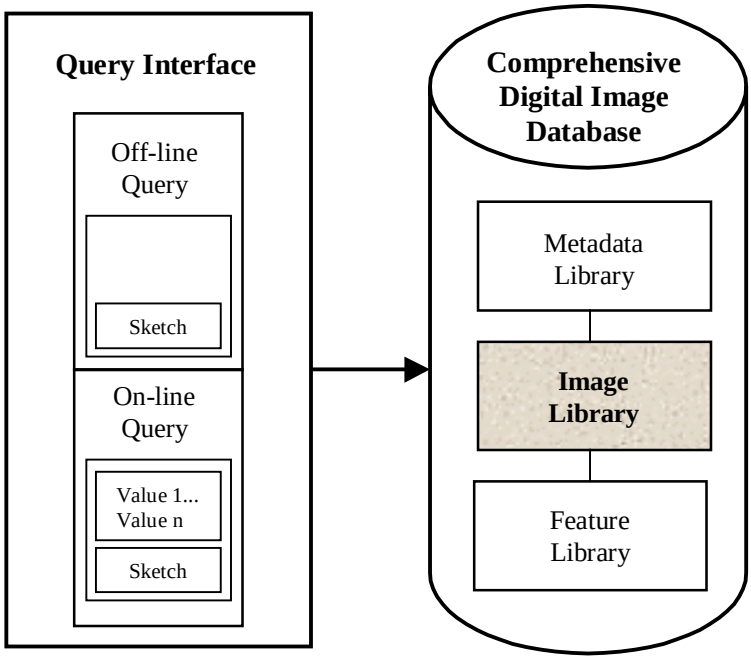


Figure 3.2 : The Image Library Component.

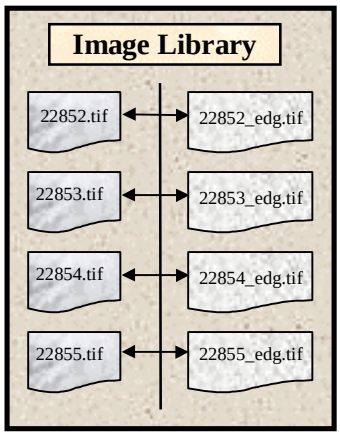


Figure 3.3 : One-to-One Linking Within the Image Library

The first step in the process to generate an edge-image from a raw image is to apply an edge preserving filter that smoothes the low-frequency components while keeping the major (strong) image discontinuities (edges) intact. The result of low-frequency filtering is a smoothed gray scale image showing a more distinct or apparent difference between the major image-objects (foreground) and their background. An edge operator is then applied to this intermediate result to extract the most prominent edges from the background information.

Various low-pass (smoothing) filters such as Laplacian or Median, and hi-pass (edge enhancement) filters such as Sobel, Prewitt, Roberts, Wallis, and Kirsch could be used for this procedure with differing results, i.e. for every image there is a preferred combination of filters that will bring out the best edge representation and indeed even for a single image there may be different combinations of filters that work better in one region and another combination that works better in another region of the image.

A typical edge operator, like the Roberts Cross-Gradient Operator for example, can be envisioned as two filters that are separately passed over the image, pixel by pixel,

with their sums at each location recorded and assigned to the image position of the upper left pixel position of the mask. The output of this edge enhancement operation will need to be thresholded so that all pixels below the threshold value are set to 0 (black) and all pixels above the threshold value are 255 (white). This gives the final edge representation image of Figure 3.4 and is stored together with the original raw image (Figure 3.5) in the image library in the manner described previously.

Figure 3.4 : Edge Representation Image

It can be envisioned that further filtering steps to clean up the excess noise along with the shorter, less prominent, and/or spurious edge information within the image could

also be applied. However, due to time constraints and that these image processing details are not the main focus of this thesis, no further effort was spent on this issue.

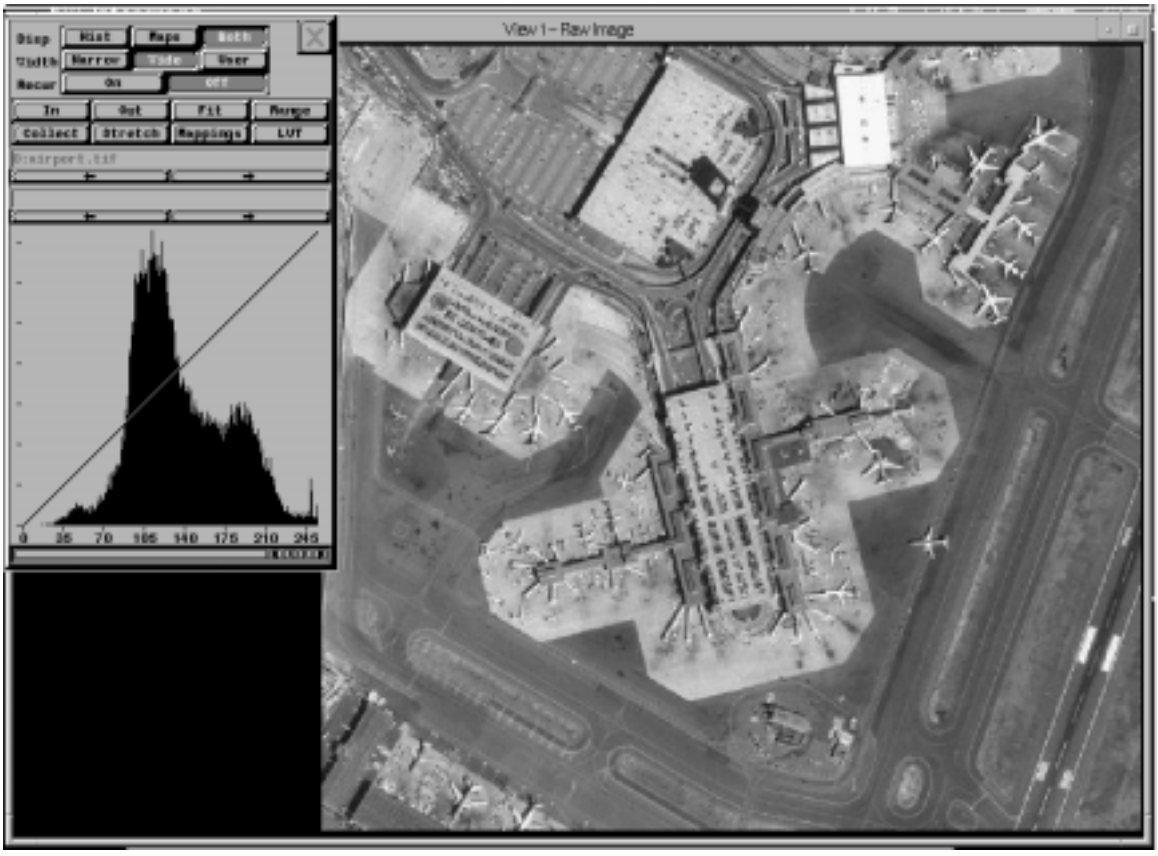


Figure 3.5 : Raw Image

In this research it was decided to compromise and use the same combination of edge preserving/extracting filters for all imagery, giving acceptable results on average, although perhaps not the optimum results for any individual image. The reason for this approach is that the image library is required to be fully self-maintaining and also because there isn't an automated process available (with today's technology) that is capable of dynamically changing its filtering parameters within a single image in order to always separate noise from information consistently. It is not the intention of this research to solve this problem of automatically determining the "best" combination of

filters for highlighting edge information. Unfortunately, this is still a very operator intensive procedure, relying heavily on a user's previous experience. When advancements in this area are realized, they of course can be added to enhance the accuracy of the image-edge files currently generated within **I.Q.**.

3.4 Metadata Library

The metadata library is another component of the comprehensive digital image database (Figure 3.6) and, as used in this research, consists of a listing of potential values for a set of attributes which describe general properties of the image itself [Agouris et al., 1999b], but not about any specific image-object or other details the image may contain.

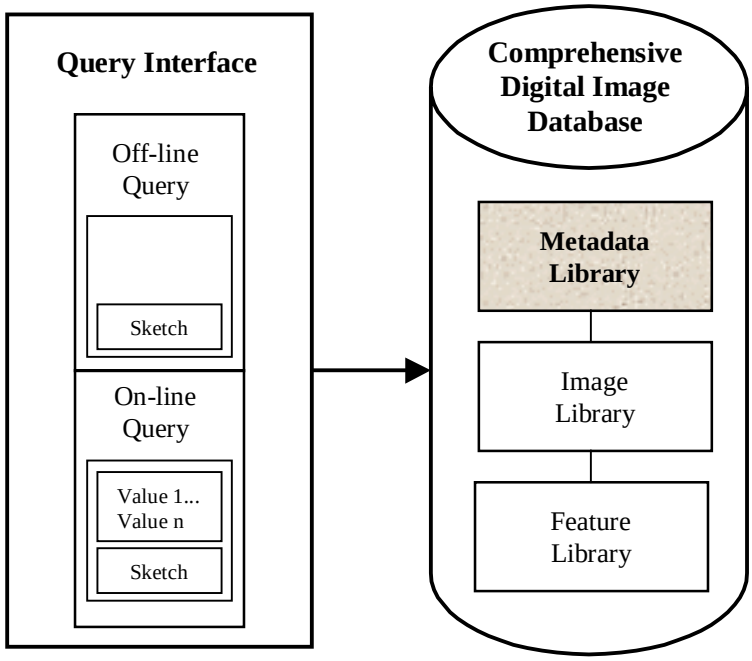


Figure 3.6 : The Metadata Library Component.

Typical metadata for aerial/satellite imagery includes such additional information as: date and time of image acquisition; date and time of introduction in the database; scale/resolution; location of the image, expressed in hierarchically arranged geographic entities like state, country, city, etc.; and/or sensor information and/or imagery type, e.g. black & white, color, color infrared, etc.

Conceptually, metadata space is an *n*-dimensional one if we assume *n* distinct metadata values. A point within this space corresponds to all images of the same area, captured at the same scale, at the same date, with similar sensor. When one or more of these parameters can accept less specific values, we move to "blobs" within the *n*-dimensional metadata space, e.g. photos of various scales of a specific area taken on a specific date form a blob in the metadata space. This blob then represents the scale space of the area at the time of data capture. When defining points (or blobs) of the metadata space we actually define a set of representations of a specific geographic area, and of the features within it. Therefore, when querying the database using metadata information, we narrow our area of interest, before we perform a query against the shapes that we expect to exist in this region.

From a computational standpoint, metadata searches are inexpensive and fast. On the other hand, shape-based searches are in general computationally demanding, but they allow us to move from global image properties, which are conveyed by metadata, to individual features (content) within images. By using metadata properties to narrow the search space for subsequent shape matches we gain computational time without compromising the quality of the query results.

Therefore, when available, metadata information can be considered, similar to the feature library (described in Section 3.5), as a screening device to thin the pool of potential matches before querying by shape begins, thus speeding up the image retrieval time for a given query. However, it should be noted that image metadata is not a requirement for **I.Q.** to process a query. It is used only as a facilitator in that, if available, it can pre-select a sub-group images within the image library or alternatively a sub-group of features within the feature library in which to limit the matching of the query sketch against. Without metadata, the query is processed against all the images in the image library or features in the feature library according to the procedures described in Chapters 4 and 5.

In this research, metadata is entered at image insertion time, before the edge-image generation process (outlined in Section 3.3) begins, and is linked one-to-one with each image in the image library (Figure 3.7). Therefore, if a subsequent query criterion is for all images at scale 1:10000, the query gets processed within the metadata library with the results pointing to individual images in the image library. To assist the user with inserting an image into the image library, a selection of the most commonly entered data values for each metadata category, as taken from other, previously inserted database imagery, can be presented in a pop-up menu for quick selection. If relevant metadata categories or data values are not presented or otherwise not suitable in this pop-up list, the user could interactively add new metadata information to the list. Once the image metadata has been entered, the image is inserted into the comprehensive digital image database where the process of edge-image generation (described in Section 3.3) begins.

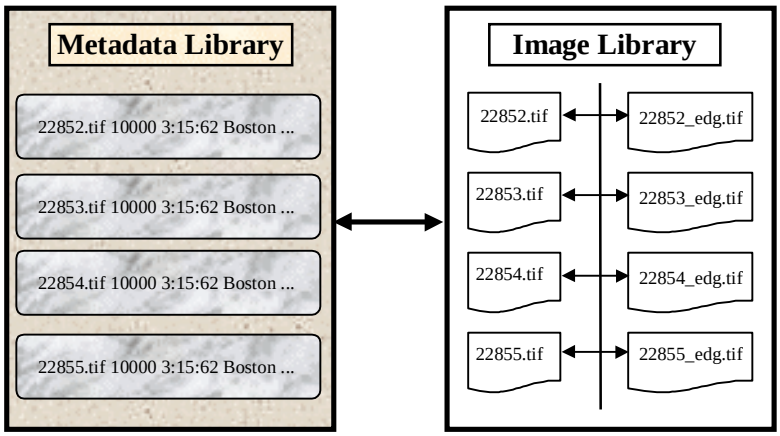


Figure 3.7 : Linking Between the Metadata Library and Image Library

If metadata is used as a matching criterion during the query building process, all images that satisfy the users request will be selected from the image library. For example, if the user only wants to look at imagery taken on a certain date, at a certain scale, and of a certain location, this information will screen out most of the imagery in the database as unsuitable matching candidates. From the remaining images, there are known links to features within the feature library that the query sketch will match against if on-line querying (described in Section 3.7) is invoked. In this case (i.e. when metadata is used to facilitate on-line querying), the query sketch will go directly to the features within the feature library that the selected images point to and begin matching the sketch at the relevant parent levels in the tree. It will ignore testing against any features that may be below the root parent point in the tree if it does not pass the specified matching criteria, thus speeding up the search times considerably.

3.5 Feature Library

The feature library component of the comprehensive digital image database (Figure 3.8) contains a set of distinct features (i.e. image-object shapes) and links to relevant edge-images where such features appear. The role of the feature library is to allow for efficient querying through the optimal organization of image-object data, in the form of previously sketched queries, and to provide the crucial link between this abridged group of raster features and a library of images. Feature-image linking allows us to avoid matching against the actual images, which can be very time consuming even for a small library of imagery.

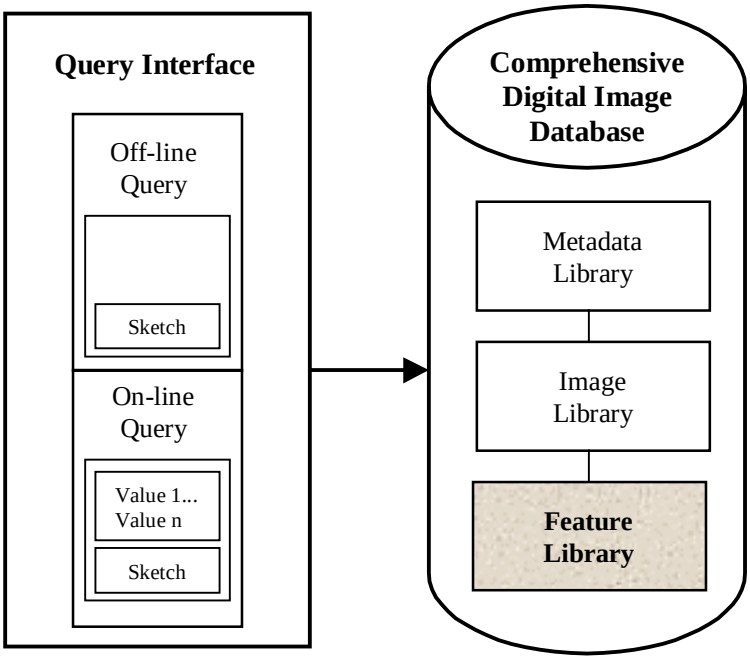


Figure 3.8 : The Feature Library Component

The organization of the feature library data enables it to act like a multi-stage screening mechanism that minimizes the risk of wasting considerable time making passes over extensive data that have no chance of selection. For example, the first screening

criterion will eliminate as potential matching candidates most of the features within the library. A secondary screening criterion will eliminate the next greatest number of alternatives and so on down through the feature library tree hierarchy. This is in contrast to performing a sequential search through an unordered list of features where much time will be wasted unnecessarily matching the query sketch against unrelated or duplicate image-objects.

The feature library is linked many-to-many with the image library (Figure 3.9). That is, one image could be linked to more than one feature within the feature library and one feature could be linked to more than one image in the image library. Due to the dynamic natures of the image library and query building, the feature library is constantly adding, subtracting and otherwise updating its features, links, and internal organization. It will also therefore need to be autonomous in that it automatically maintains its own contents depending on the changing states of these external but integrated components.

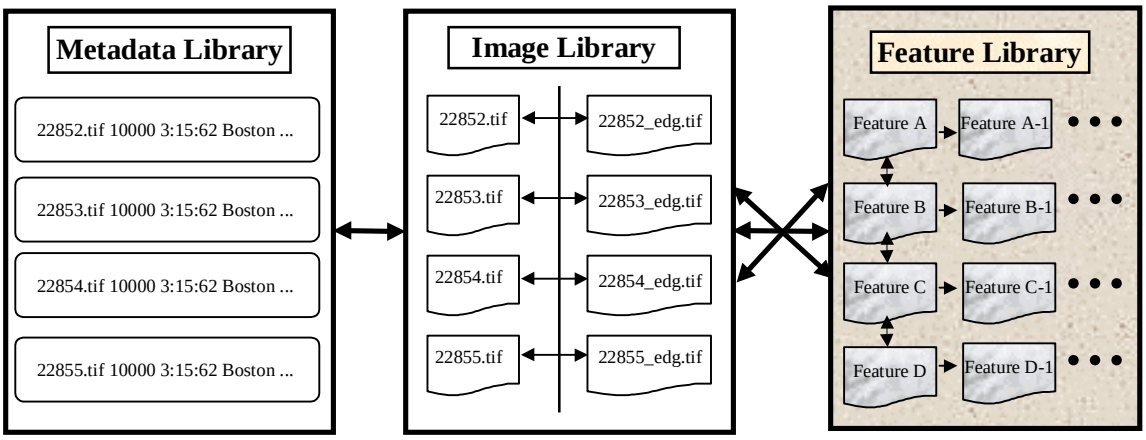


Figure 3.9 : Linking Between the Metadata Library, Image Library, and Feature Library

3.6 Off-line Matching

The off-line matching component of the **I.Q.** environment is a process that matches a query feature to the entire feature library while ignoring its hierarchical structure (Off-line Method 1, Figure 3.10) or to the entire image library (Off-line Method 2, Figure 3.11). In doing so, off-line matching completely ignores any metadata input the user may have entered while composing the query. Subsequently, off-line matching Method 1 and Method 2 bypasses the metadata library component of the comprehensive digital image database and processes the input query using sketch information only. Off-line matching therefore is invoked only in the cases where no acceptable images were returned to the original query.

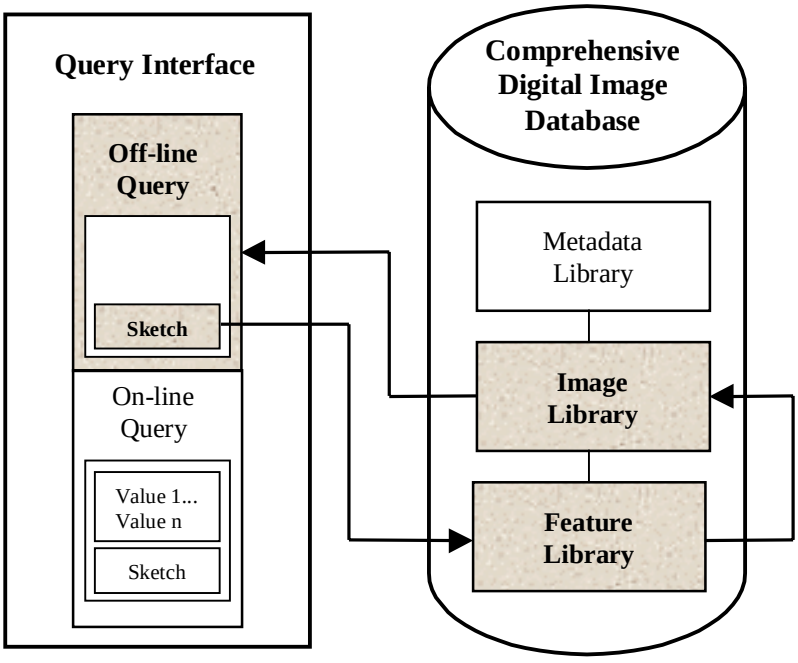


Figure 3.10 : The Off-Line Matching Method 1 Information Flow

Potentially, a situation where unacceptable images are retrieved by the original query could arise if relevant features (i.e. those that match acceptably to the query sketch) within the feature library were inserted as children under a parent feature that did not match sufficiently to the input query sketch. In this case, the query feature would not pass down through the parent feature's branch within the feature library tree hierarchy and therefore would not find its potential match and subsequent link to the image library.

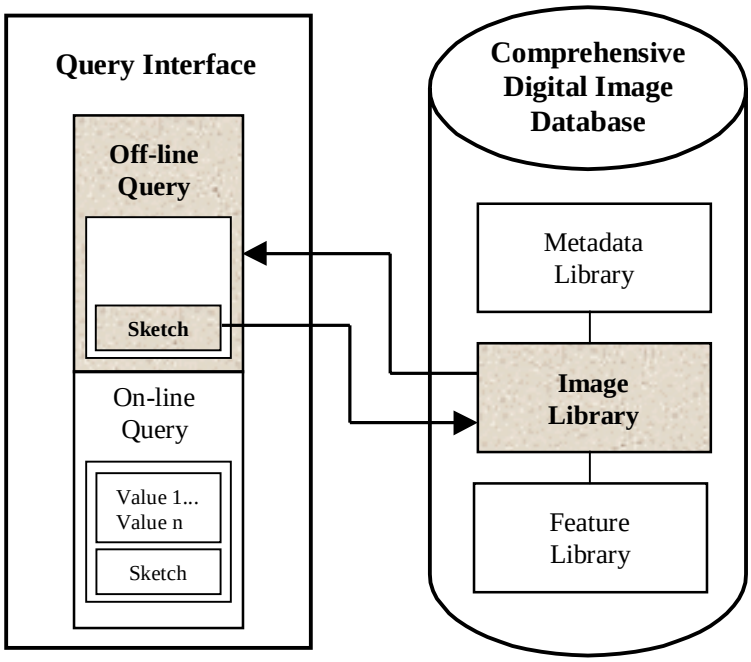


Figure 3.11: The Off-Line Matching Method 2 Information Flow

Off-line matching is also a process that matches all existing and newly inserted library features (as opposed to query features) against all newly inserted images (Off-line Method 3) of the image library. In this case, no query input is provided by the user, and a substantial amount of time is required to complete the matching process (depending on how many images are in the image library and/or features in the feature library).

However, this process ensures that all library features are linked to their proper images (and vice versa).

Off-line Method 3 is usually invoked under two circumstances:

- First, when a new image is inserted into the image library. Upon insertion, a new image gets its edge representation generated, as described earlier, and then all the features in the feature library get matched against this new image to see if and where they match. More specifically, when new images get inserted, Off-line Method 3 matches the existing features only to those new images, and not to the entire image library. If a feature-to-image match is above 50%, a link from this feature to this image (and vice versa) is established along with its location (i.e. the coordinates of the feature centroid within the image), and the coordinates of the minimum bounding rectangle (MBR) of this feature within the image, and;
- Second, when a query feature is inserted into the feature library. The query feature, to begin with, will acquire all the links of its parent but will in turn get tested against all the images in the image library to update these links. Similar to the previous case, if a feature-to-image match is above 50%, a link from this feature to this image (and vice versa) is established along with its location.

3.7 On-line Matching

On-line matching is a process that will retrieve all relevant images from the image library that match to the users input query in real-time, i.e. while the user waits on-line. The wait time required for such a process depends on the number of features in the

feature library and on the final organizational structure the feature library. For example, depending on the number of nodes at a given level in the tree hierarchy, the time required to search for a conjugate match at that level will vary.

On-line matching can be invoked with or without the metadata option being selected during the query composition phase. For example: when a query is processed with the Metadata option, four processes begin sequentially (Figure 3.12):

- First, the metadata library will be searched and image filenames that satisfy the query will be returned;
- Second, the selected images will return the corresponding feature filenames that they are linked to;
- Third, the input query sketch will be matched against this abridged list of features beginning at the parent level of each of these selected features and the results prioritized, and;
- Fourth, the images that the best matched feature library features point (link) to will be returned as the final result to the query.

This is the most efficient approach to querying the comprehensive digital image database and is what makes on-line querying possible (i.e. in real-time).

Some issues to consider while determining which images to return from a query and in what priority need to be addressed. These include the concepts of *same*, *similar*, and *different*. For example, when should a query sketch be considered to match the same to a feature from the feature library (or to an object in an image), or similar, or different? This issue was examined in some detail and is described further in Chapters 5 and 6. In summary, it was discovered that the values chosen for these three matching categories are

both application and user dependent. Therefore, it is left up to the user to determine what matching percentages he will accept to mean that two features should be considered as the same, similar, or different.

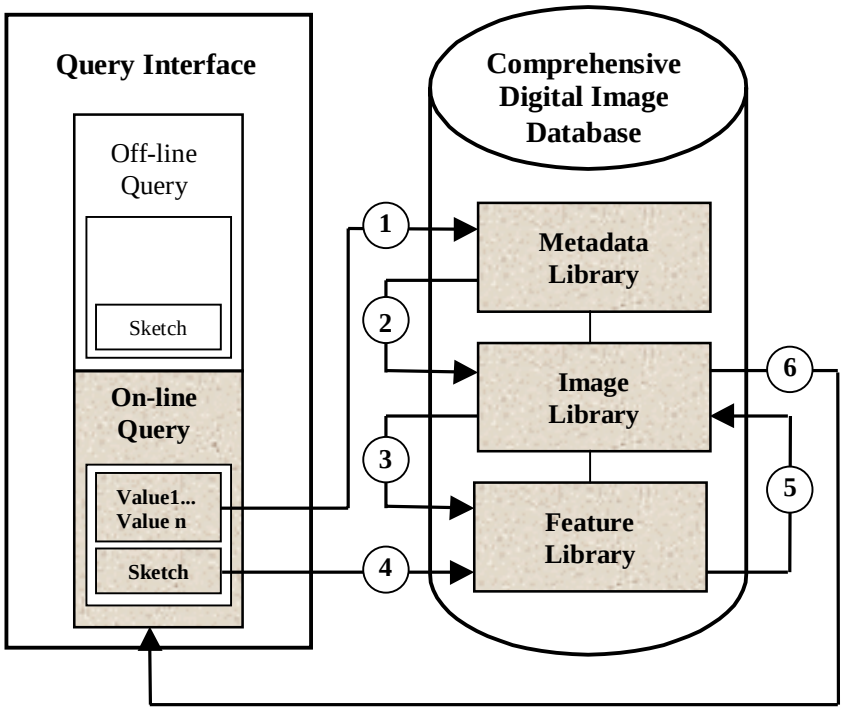


Figure 3.12: The On-Line Matching Information Flow With Metadata

Depending on what value ranges the user chooses for these three categories, the feature library will re-structure accordingly by adjusting its width and depth. This in turn will have a direct affect on both the number of images that get returned to a given query and on the time required to return this result, e.g. a wider tree structure will generally take longer to search through.

Concerning the prioritization issue, where the sequence of returned images to a given query is in question, there are conditions here that will also need to be investigated. For example, consider the case where a query feature matches 70% similar to Feature A

and 60% similar to Feature B. Feature A in turn is linked to an image where it matches only 55% while Feature B is linked and matched to an image at 85%. In this case, the prioritization process will take the links provided by the feature with the highest match first (the image links to Feature A). The prioritization process does not consider that Feature B might have stronger links to its images (than Feature A) because this may or may not make any difference to how well the query feature itself matches to the images.

It is quite possible that a query feature could match better to the linked images than the feature it matched to in the feature library and vice versa. Since this is on-line matching, it is not feasible to check in real-time how well the query feature itself matches to each of the linked images just for the purpose of prioritizing the images returned from a given query. Therefore, the image links to the best-matched feature from the feature library will be returned first in their respective order followed by the links to other images of lesser-matched features (Figure 3.13).

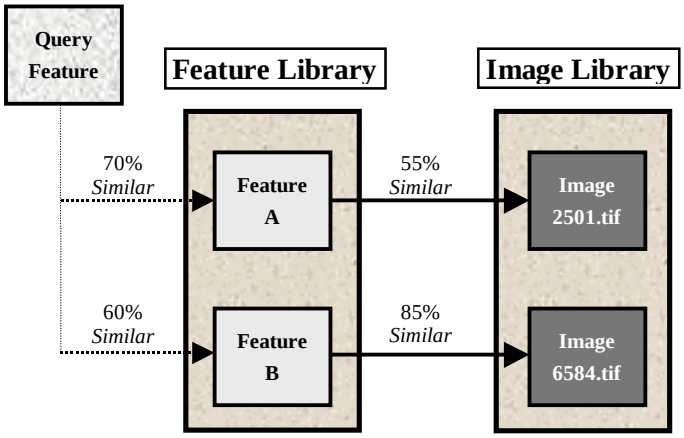


Figure 3.13 : Prioritization of Query Results.

Images linked to Feature A will be returned before images linked to Feature B even though the Query Feature might match better to the images linked to Feature B. This allows for querying and on-line matching to complete the image retrieval process in real-time.

When a query is processed without the Metadata option, due to the user having no restrictions on scale (or on location or on any other metadata information), three processes begin sequentially (Figure 3.14):

- First, the query feature gets matched against the entire feature library beginning with the primary parent level and then subsequently down through the required (i.e. best matched) trees;
- Second, the best matched features in the feature library will be found and their links to the images prioritized, and;
- Third, the prioritized images are returned as the final result to the query.

This approach, which is also considered to be on-line matching, is not as efficient as when metadata is used in the query building process, but is still much quicker than matching the query feature against the image library directly.

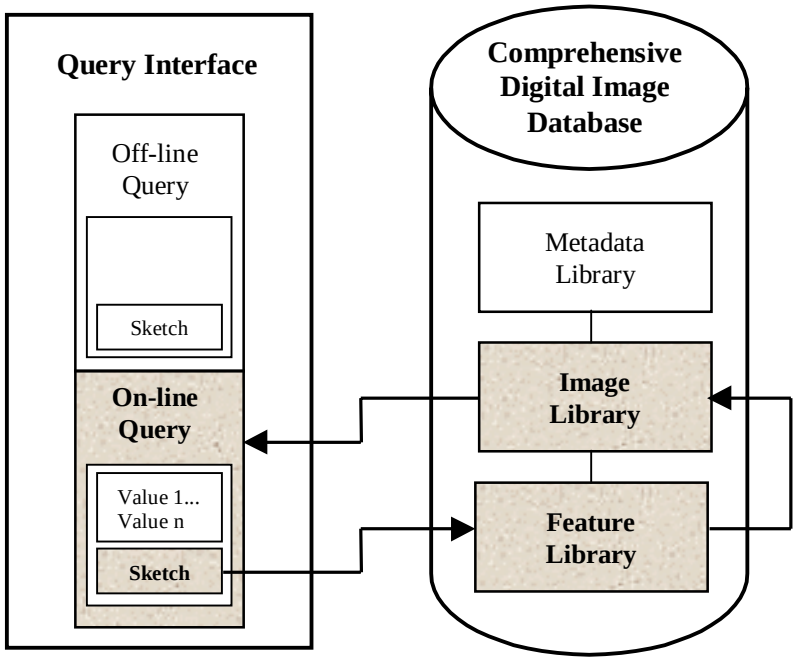


Figure 3.14: The On-Line Matching Information Flow Without Metadata

Both of the querying scenarios just described are considered as on-line matching. They both will return, in a reasonable amount of time, results to a users query for images that contain the requested object or feature. However when neither of the above two methods return the desired images, off-line matching will need to be invoked.

3.8 Query-by-Sketch in a Comprehensive Digital Image Database

When querying the comprehensive digital image database, several conditions will need to be addressed. For example, querying with and without metadata should be allowed. This type of condition should be allowed in such an environment to counter the instances where relevant imagery bypasses the metadata filter. An image could potentially slip through the metadata search "net" if no information about a particular attribute was entered at image insertion time. For example, if a particular image didn't have an attribute value entered for the date of exposure, a metadata search on this criterion would fail to include this image in the query result.

Therefore a query in the **I.Q.** environment can consist of any combination of metadata and sketch, i.e. there are five cases.

- **Case 1:** a query could be built and run against the image library using only metadata information (Figure 3.15). A sample query in this case could be to return all the images that were taken on a certain date by a certain sensor.

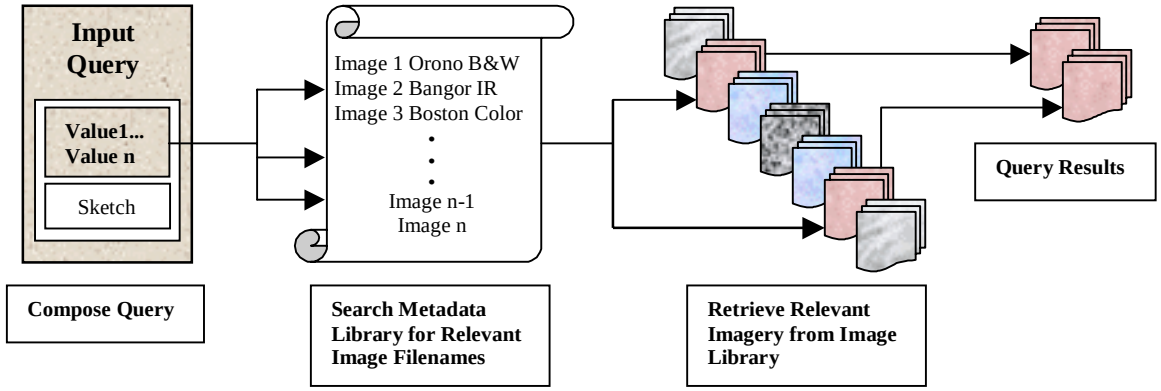


Figure 3.15: Metadata Only Query Information Flow

- **Case 2:** a query could consist of metadata information and a sketch (Figure 3.16). A sample query in this case would be to use the results provided in Case 1 to select a subset of features from the feature library to match the user provided sketch against. The images linked to the best-matched features from this subset will be retrieved from the image-library as the results to the query.

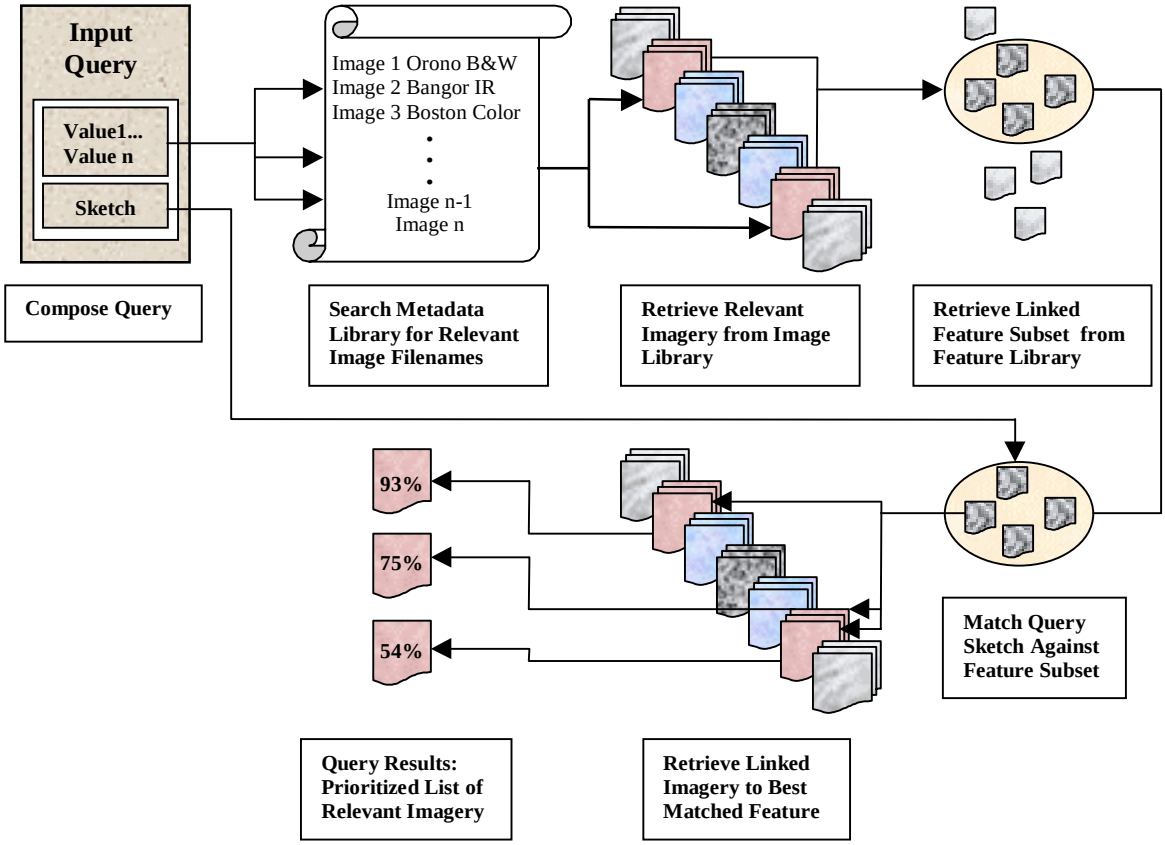


Figure 3.16: Metadata and Sketch Query Information Flow

- **Case 3:** a query could consist of sketch information only and be performed solely on the image library (Figure 3.17). A sample query in this case is that you want to find all the imagery that contains a certain object regardless of scale, location, date, etc. of the imagery, while at the same time ignoring the feature library completely. This type of query is performed off-line (Off-line Method 2) due to the time needed but makes certain that all imagery in the image library is being considered. This type of query, for example, takes into consideration that if no metadata information is entered for some of the imagery at image insertion time, they can not be considered as results to a metadata search.

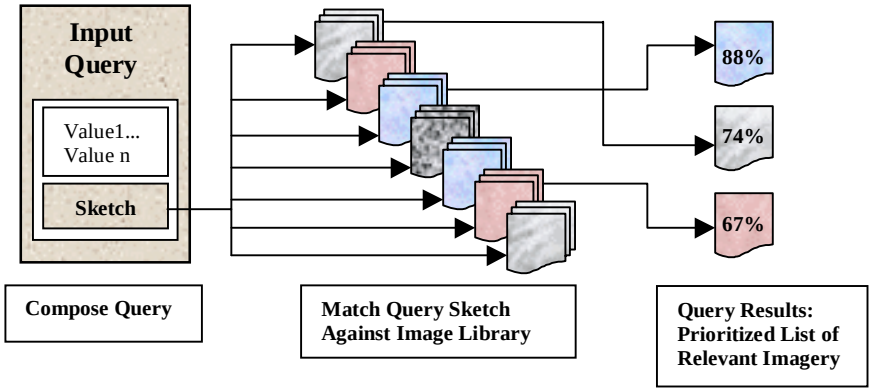


Figure 3.17: Sketch Only Query Without Feature Library Information Flow

- **Case 4:** a query consists of sketch information only and the feature library is considered but not its underlying organizational structure (Figure 3.18). What would happen in this case would be that the query feature would get matched offline (Off-line Method 1) sequentially to all the features in the library. A possible scenario where this type of matching would be invoked is in the cases where no acceptable images were returned to the original query due to the relevant features within the library being inaccessible. Library features could become inaccessible in some cases because the organizational structure of the library is dependent on the values chosen for the matching parameters “same”, “similar”, and “different” and on the temporal order of feature insertion into the library’s tree-like structure (discussed further in Chapter 5).

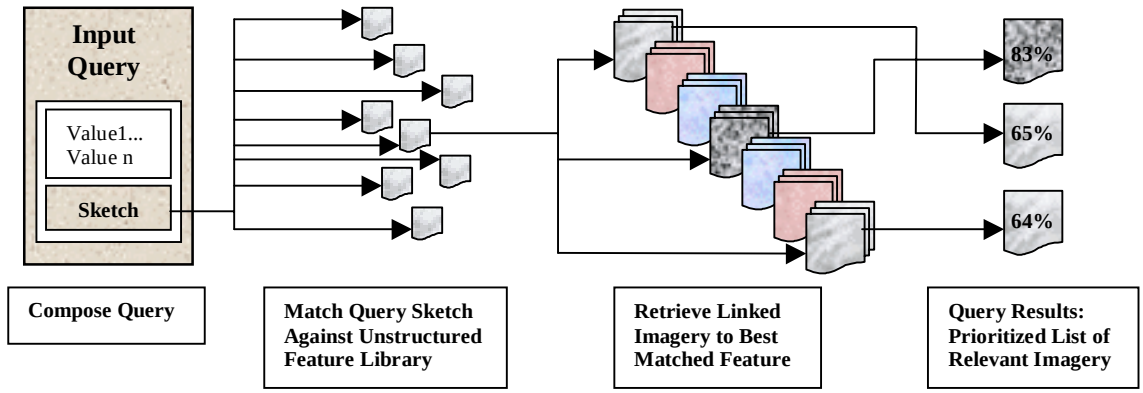


Figure 3.18: Sketch Only Query Ignoring Feature Library Structure Information Flow

- **Case 5:** a query consists of sketch information only and the feature library, together with its organized structure, is considered (Figure 3.19). A sample query in this case is that you want to find all the imagery that contains a certain object regardless of scale, location, date, etc. of the imagery, similar to Case 3, but you want to perform this query on-line (i.e. in real-time). This type of query would first match the query template feature against all the features at the parent level of the feature library and then start progressing down through the relevant trees of the best-matched parents. The links to the images of the best matched library features are returned as the results to the query and the query feature gets inserted, if the user wishes, into the feature library at that point.

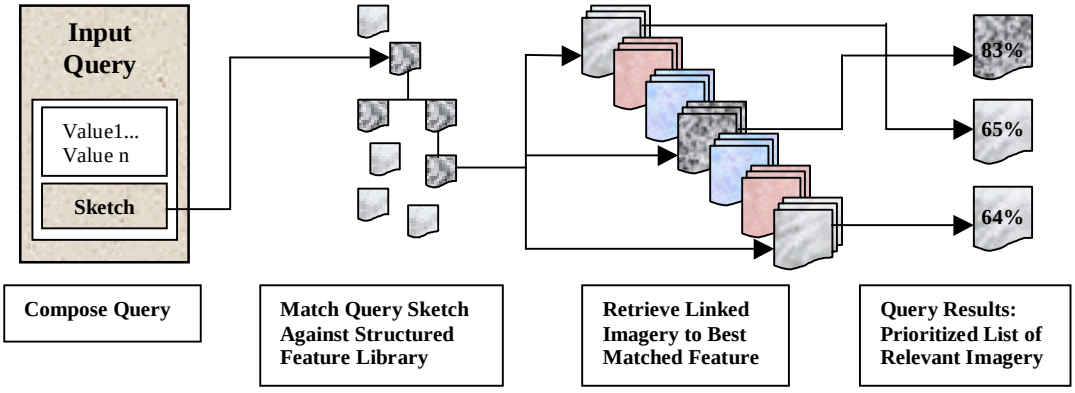


Figure 3.19: Sketch Only Query With Structured Feature Library Information Flow

3.9 Feature Matching and Linking

In order to allow the proposed image query and retrieval environment to perform on-line matching, it is necessary to link previous user sketches (queries) with their respective matches in the image library. In doing so, further sketch-based queries on the feature library will result in a list of prioritized images returned to the user. Also, further metadata queries will result in a list of images being returned that point to a prioritized list of features in the feature library, on which sketch-based matching can begin. The result of this matching concludes in turn with a list of prioritized images that match to the combined metadata/feature input query.

It is from the matching process itself (described in detail in Chapter 4) that the links between the feature library and image library are created and maintained. The variables returned from the matching process are the feature name, the image name, the matching percentage, the row and column that the centroid of the feature matched at, and the row and column values for the minimum bounding rectangle of the feature within the image. Sample values for each of these variables might consist of (Table 3.1):

Feature	Image	%	Row	Col	MBRminrow	MBRmincol	MBRmaxrow	MBRmaxcol
Building1	22852	74	487	1920	450	1850	530	1990
Building1	29342	70	2190	1530	2100	1500	2253	1600
Building1	43852	65	3278	550	3200	500	3300	600
Plane7	39653	85	1522	2425	1496	2400	1555	2465
Plane4	22852	90	1700	886	1650	800	1778	925

Table 3.1 : Image-Link File.

An image-link file is created containing an entry for every feature in the feature library. This file can be considered as a view of the matching results grouped according to feature names, as in the above table. This view contains an ordered list of the images that matched to the features above 50%. Every time off-line matching is invoked and new links are found, they get inserted into this file in their proper location with higher matched images listed first. The features themselves get appended to this view upon insertion into the feature library.

Similarly, each image in the image library has a feature-link view of the matching results grouped according to image names (Table 3.2). This view contains an ordered list of the features that it matched to above 50%.

Image	Feature	%	Row	Col	MBRminrow	MBRmincol	MBRmaxrow	MBRmaxcol
22852	Plane4	90	1700	886	1650	800	1778	925
22852	Building1	74	487	1920	450	1850	530	1990
43852	Building1	65	3278	550	3200	500	3300	600
39653	Plane7	85	1522	2425	1496	2400	1555	2465
29342	Building1	70	2190	1530	2100	1500	2253	1600

Table 3.2 : Feature-Link File.

The variable entries in this file consist of the same columns as for the image-link files except for the ordering. Every time off-line matching is invoked and new features are found to match greater than 50%, they are added to this file in their proper location with higher matched features listed first. The images themselves get appended to this view upon insertion into the image library.

3.10 Chapter Summary

This chapter presented the theoretical framework underlying the approach adopted in this thesis for digital image retrieval using raster queries. It includes a description of the three major components that make up the comprehensive digital image database, namely the image library, the metadata library, and the feature library. The distinction between on-line and off-line matching is presented along with the concept of feature matching and linking and a description of the 5 types of queries that are indeed possible to process in such an environment. Within the proposed environment, queries can be performed with or without the metadata option and with or without consideration to the feature library organizational structure. Input query sketches can consist of an existing member of the feature library, an extracted image-object taken directly from an image in the image library, or a newly sketched query feature submitted by the user.

Chapter 4

Modified Least-Squares Feature Matching

Both the on-line and off-line matching algorithm employed by the image query and retrieval environment presented in this thesis is based on the modification of the least-squares matching (lsm) technique described previously in Chapter 2. In traditional least-squares matching, the difference in gray values between a patch of template pixels and a patch of image pixels is the criterion used to determine the accuracy of the match. Limitations of this approach are that it can be a highly computation intensive operation, even for small patches of pixels, and that good initial approximations for positioning the template patch within the image are required for determining a correct match.

4.1 Objective & Contribution

The main objective of this chapter is to explain in more detail how the modified least-squares matching approach functions. It was decided to use lsm in this thesis as the matching foundation due as much to its robust nature in establishing correspondences among conjugate image windows, as to its other previous successful extensions, including; matching on multiple image windows [Agouris and Schenk, 1996], feature-wise global matching solutions [Gruen et al., 1995], and on establishing correspondences in sets of 3-D images [Maas et al., 1994].

Conceptually, the modified matching algorithm is a variation of (lsm), modified to function with image-edge files. One of the obvious differences between the modified approach and the traditional least-squares method therefore is the theoretical extension of

matching an input query template, in the form of a binary raster sketch, on binary raster imagery (Figure 4.1).

The major scientific contribution of this modified least-squares approach is this extension to function with binary raster data and subsequently without good initial approximations of the conjugate positions.

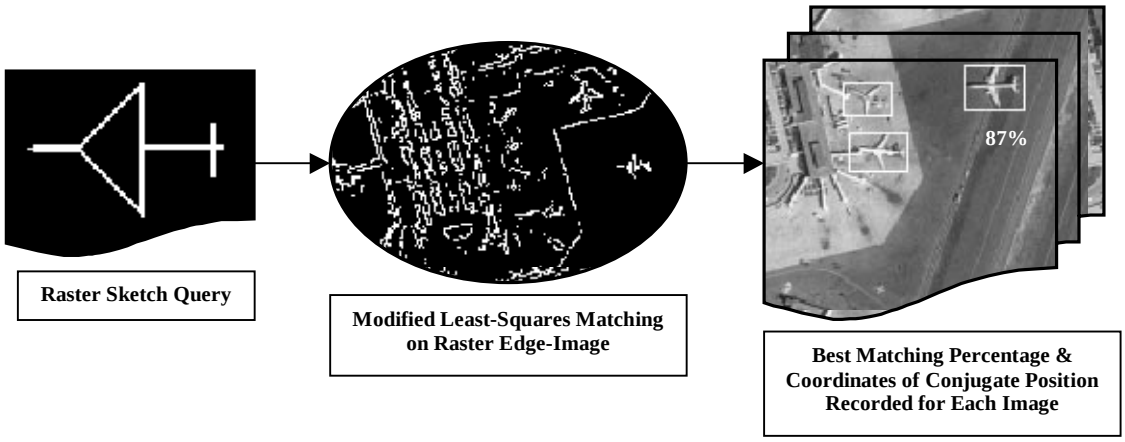


Figure 4.1 : The Query Matching Workflow

4.2 Feature Matching

In our approach, we use edges instead of gray scale pixels as the metric for template to image similarity. Therefore, it is not necessary to test full patches of pixels against each other. Instead, we reduce the patch into its information content by considering only those pixels that contain image-object information, i.e. only those that constitute image-object edges, are required to test for similarity. This is possible because the image itself is binary and therefore consists only of edge representations. The matching process then reduces to checking only if the input query template edge pixels overlay edge pixels from the image - the answer can only be either a “yes” or “no”. If

yes, this template edge pixel "votes" to stay where it is. If no, then this template edge pixel votes to move. The question now is, in which direction and how far.

4.2.1 The Unknown Transformation Parameters

In traditional least-squares the affine transformation is used to calculate the new position of the query template patch within the image. It incorporates unknown transformation parameters that take into account template translation, scaling and rotation. The direction to shift the query template then is determined by testing the gray scale difference between neighboring pixels within the template against the gray scale difference between neighboring pixels within the image. This amounts to testing the template gradient against the image gradient at each pixel location within the template.

The result of this test is a decision as to where and how far to shift the template within the image to acquire a better match. Typical distance values for shifting the template can be anywhere from fractions of a pixel to a few pixels in any direction before re-sampling takes place and the unknown transformation parameters and new shift values are re-calculated (Figure 4.2).

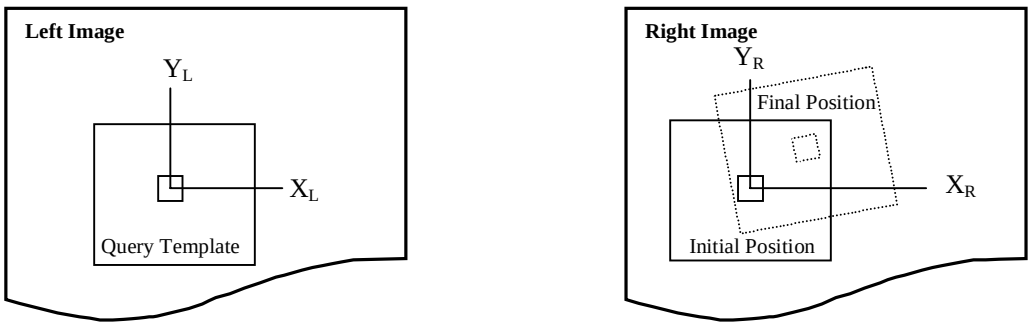


Figure 4.2 : Left Image Template and its Conjugate Right Image Window after Iteration

Since the template never shifts more than a few pixels at a time in any one direction, if the initial positioning of the template within the image is not already close to its correct conjugate position, the template may never find its proper final match. Traditional least-squares template matching therefore, although robust and highly accurate, is very sensitive to the initial approximations as its "pull-in" range is quite restrictive.

In contrast to traditional least-squares, our matching algorithm is a modification of lsm to function using object outlines instead of gray values, thereby avoiding computationally expensive matrix manipulations. We obtain a solution by analyzing the dissimilarities between a query template (user provided sketch) and an edge-image window. By using edge information only, the calculation of the unknown transformation parameters reduces to checking in each of the cardinal directions (for each edge pixel in the template that votes to move) for the nearest image edge pixel. Summing up and averaging all directions and distances for all edge pixels that vote to move gives the distance values and direction to shift the template feature.

Typical distance values range from 0 to 10s of pixels in any direction with the maximum distance allowed being half the dimension of the template feature itself. This distance is always an integer amount because the edge pixels are either "on" or "off" and there is no averaging of groups of neighboring pixels allowed. This allowance for shifting of the template feature 10s (or more) of pixels in any direction means that initial approximations for positioning the template within the image are not required. If our developed edge matching algorithm determines that the template is not in a suitable position within the image, it will "move to where the image content is" on its own.

4.2.2 Observations

In traditional least squares, the observations are the gray levels of the images themselves. This methodology allows for finding correspondences between two images, between an image and a map, and between an image component (window) and a query template (sketch). In our approach, again since object edges are used instead of gray scale pixels, gray level observations are replaced with feature existence/absence observations. When a query template is compared to an edge-image, the template is divided into four quadrants. Each quadrant is matched separately to a corresponding edge-image area, and template pixels vote to stay put (or move) according to their similarity (respective dissimilarity) to corresponding edge-image pixels. This resembles the comparison of gray values in least-squares matching and the use of image gradients to identify shifts, rotations, and scalings.

4.2.3 An Iterative Solution

In traditional least-squares, the solution is obtained by solving for and using the unknown parameters of the affine transformation to determine a new position in the image as the conjugate of the template. This is an iterative procedure, i.e. the least-squares matrix equation will have to be iterated until the unknown transformation parameters do not change (within a certain tolerance) or until the iteration number reaches a previously specified maximum.

In our approach, to determine which direction and how far to move the query template while matching within an edge-image, the template is divided into quadrants centered on the centroid pixel of the template feature. Each quadrant is matched separately to the edge-image and the sum of the “votes” for each pixel of the feature in

the quadrant is recorded as to whether it votes to stay or move - and if it votes to move, in which direction and how far. Image pixels are checked in the +/- x and y directions. This means each quadrant votes to move in one of 4 cardinal directions; left, right, up, down together with how far it wants to move plus 2 other possible voting outcomes; staying where it is or shifting completely to another position within the edge-image. Therefore, with 6 possible outcomes for each of the 4 quadrants, there are 6^4 (or 1296) possible outcomes to consider for each new position the template takes up within the edge-image (Figure 4.3).

The process of determining direction and distance to shift the query template through the tabulation of its edge pixels votes continues until an arbitrary limit (e.g. 20 iterations) has been reached or until more pixels vote to stay put than to move. This method also allows for occlusions of up to half of the template feature to be detected as the feature can shift its origin, centroid pixel right up to the border of the edge-image.

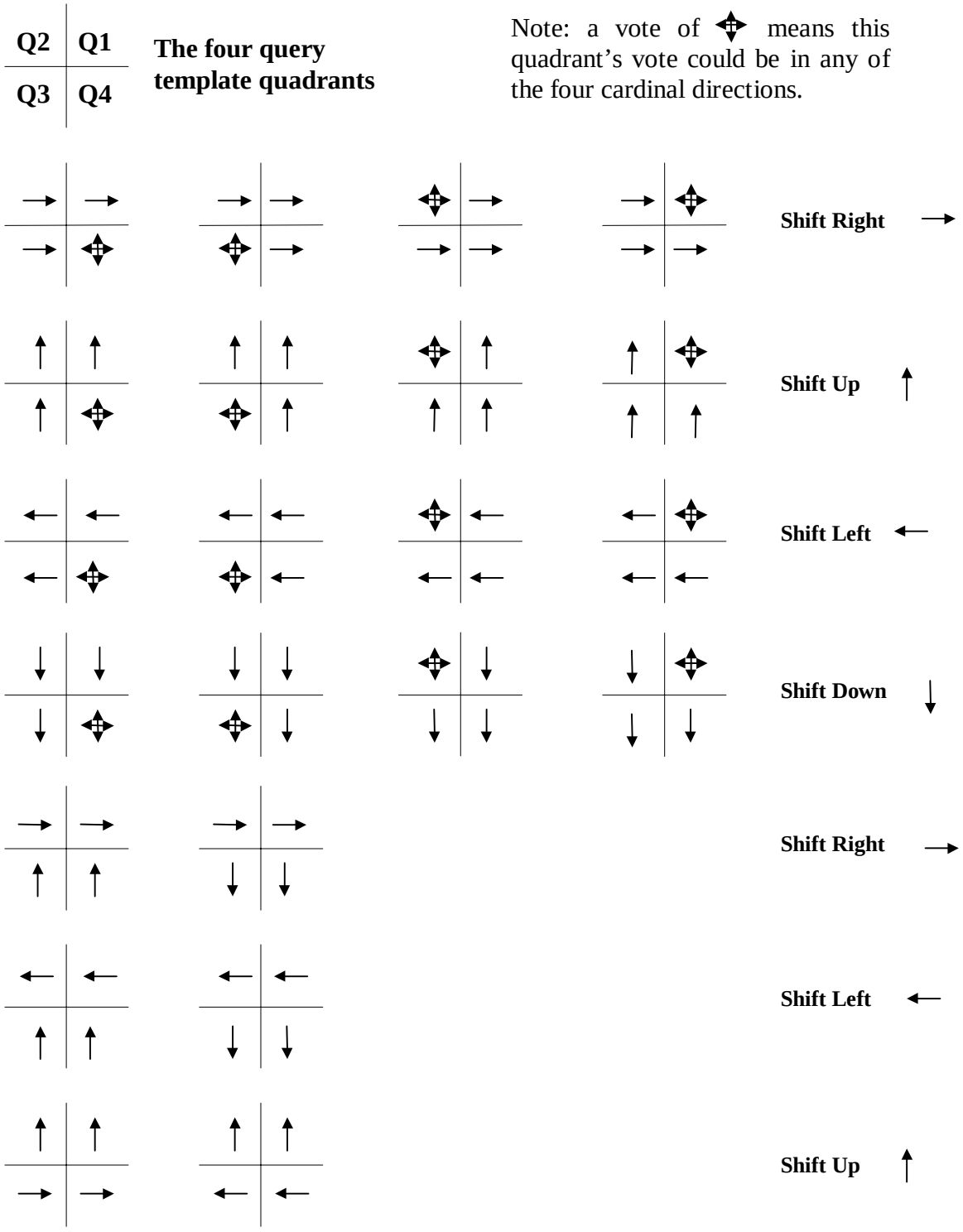


Figure 4.3 : Quadrant Voting and Feature Shifting.

Note: The voting is presented in the order in which it is tested.

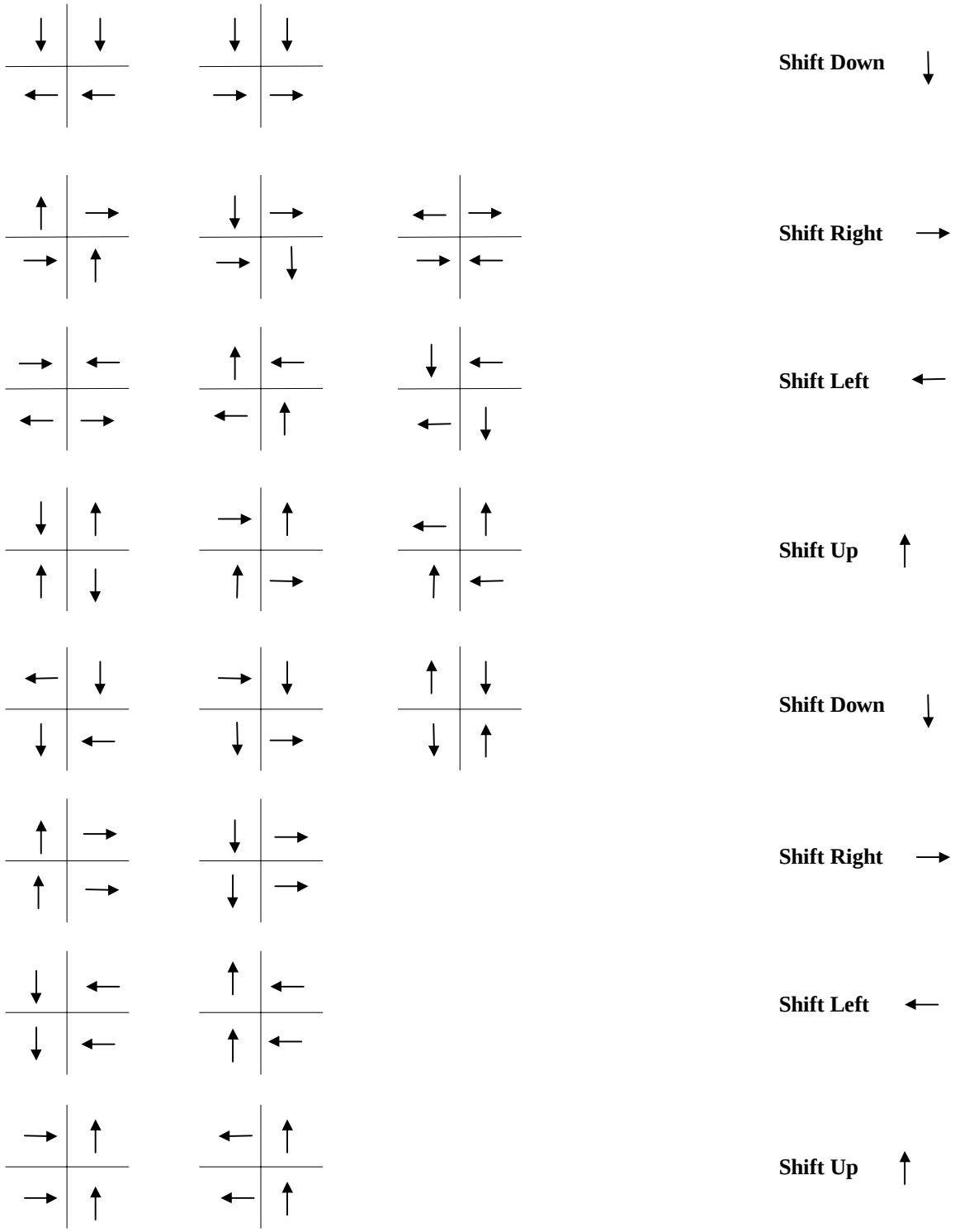


Figure 4.3 cont. : Quadrant Voting and Feature Shifting.

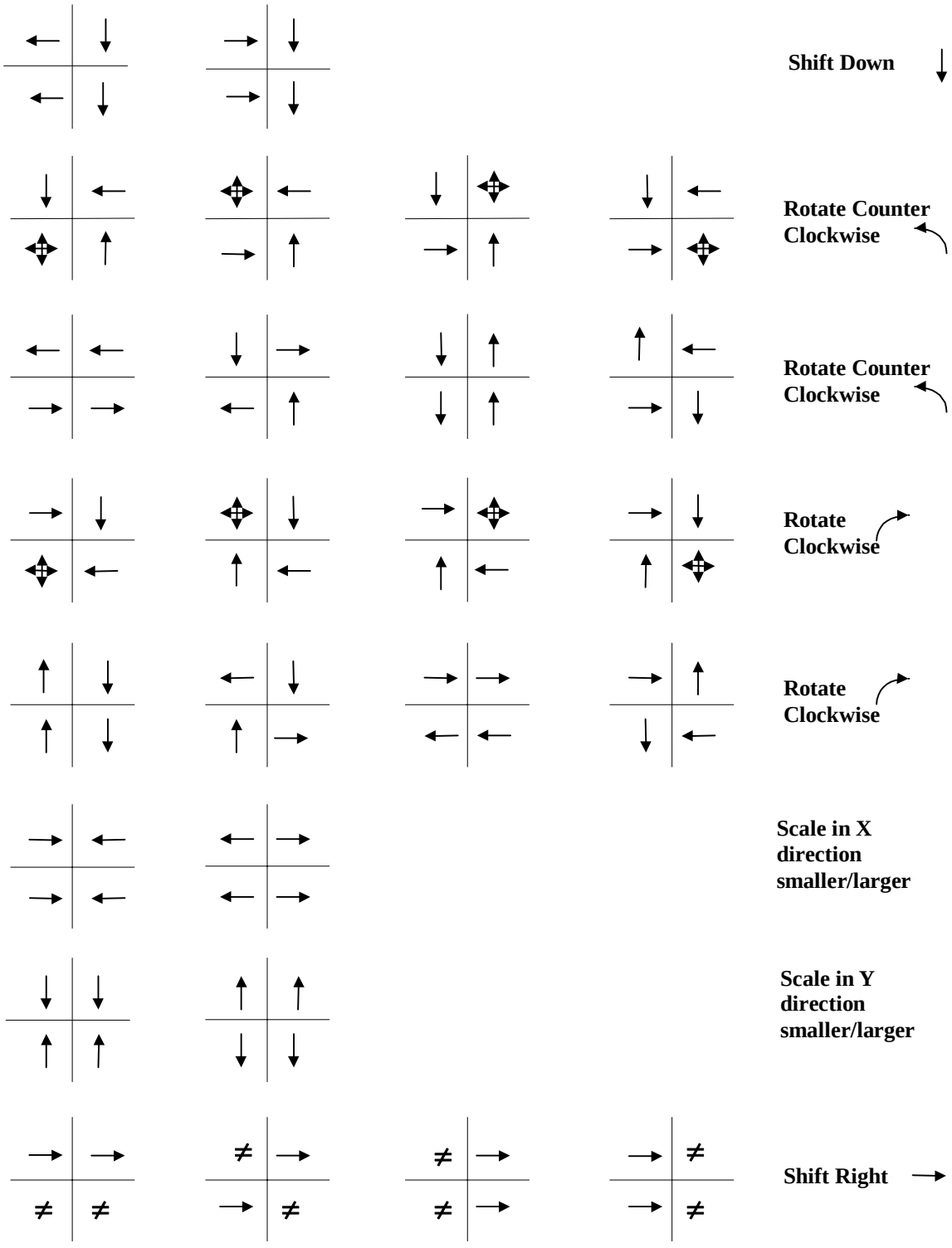


Figure 4.3 cont. : Quadrant Voting and Feature Shifting.

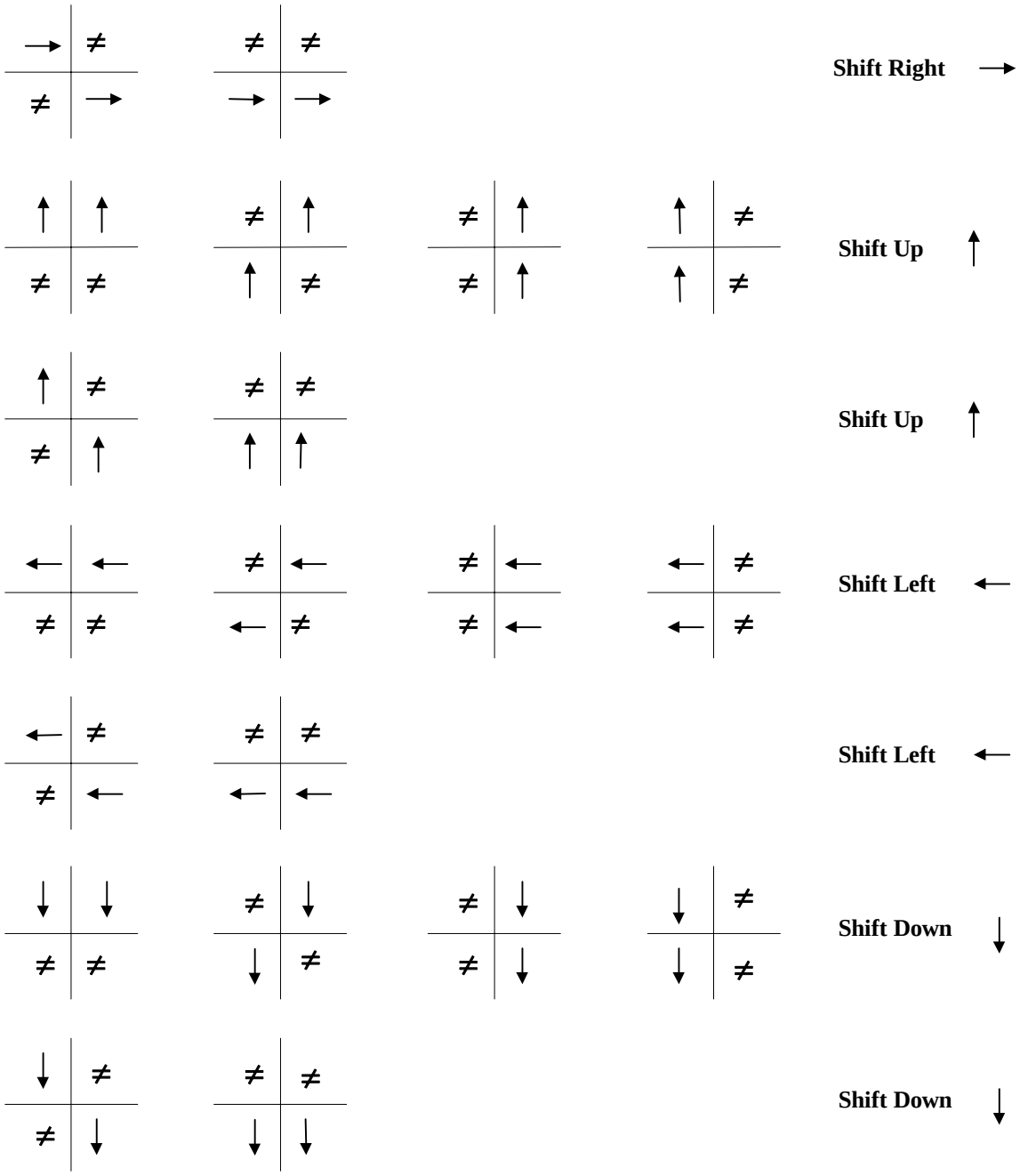


Figure 4.3 cont. : Quadrant Voting and Feature Shifting.

4.2.4 Accuracy Measures

In traditional least-squares, the robust mathematical foundation allows us to obtain meaningful statistical measures for the accuracy of the matching process in the form of the *a posteriori variance of unit weight*. Here, the residuals vector, expressing the deviation between observations and their adjusted values, and the degrees of freedom, or difference between formed equations and unknown parameters are combined in an equation that is used as a score index, reflecting the confidence level of a match.

In our approach, once the template query feature has settled onto a match, its accuracy is determined from its matching percentage, i.e. by how many of its pixels continue to vote to stay put compared to the total number of pixels that constitute its edges. For example, if a query feature has 100 edge pixels and after it has scaled, rotated, and translated itself onto its final conjugate position, 70 of these pixels are positioned on edge information from the image, it will be considered as matching 70% to this particular image. This matching percentage is recorded and sorted for each image in the image library.

4.3 Implementation Concerns

To further assist the self propelled movement of the query template throughout the edge-image, the file is subdivided into fractions depending on the pixel dimensions of the input query feature. For example, if we assume the feature being searched for is smaller than $1/9^{th}$ of the total edge-image size, the edge-image could be subdivided into 9 equal sub-regions (Figure 4.4). If the feature being searched for is larger than this, then the image could be subdivided into quarters, etc.. For example, when matching the query feature to a member of the feature library (during on-line matching), there will be just 1

sub-region and its dimension will be identical to the library feature's dimension. Decisions on how to subdivide the image could also include the scene complexity, with a more complex scene being subdivided more to ensure no detail is missed and to speed up the searching process.

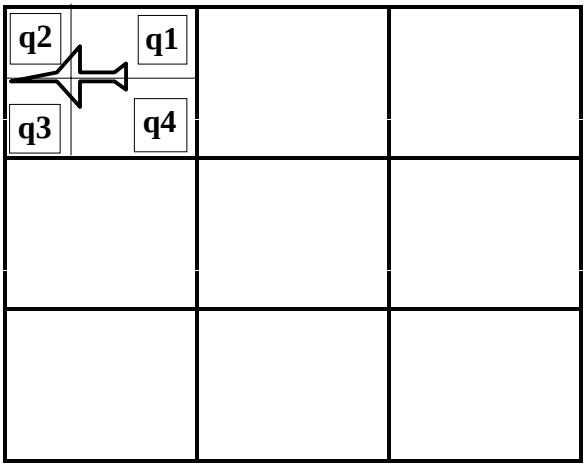


Figure 4.4 : Example of Query Template Quadrants and Image Sub-Regions.

Sub-region boundaries are set up so the template feature doesn't waste time searching around in an area of the image which may have no relevant data and to avoid searching over the same image area more than once. By subdividing the image into these regions, the feature is restricted from translating uncontrolled over the image. Its movements are confined to the sub-region and each sub-region is checked sequentially only once for matches. The best template matching percentage for all the sub-regions is taken as the best match for that particular template/image pair.

When the query feature's edge pixels are searching for a direction and distance to move to, they will only do so for a distance of half the dimension of the sub-region. If they do not find a suitable match within this distance, then those pixels vote to move the

entire feature to a new position within the sub-region a distance of ½ the dimension of the query template away. Once the query feature encounters the boundary of one of the sub-regions, it drops down ½ the dimension of the template in the y direction and repeats its movement along the x direction until it either finds some features in the image to match to or again hits the boundary of the sub-region. When the entire sub-region has been searched in this way, either successfully or unsuccessfully, the feature will then jump to the beginning of the next sub-region and repeat the process. This is in contrast to the traditional least-squares approach to template matching where the template is restricted to moving only fractions of this amount.

At the completion of checking all the sub-regions, the position of the best match from all the regions is identified as the best match for the image. To avoid missing a match that may occur at the borders of two sub-regions, additional sub-regions are added to overlap these areas in both the x and y directions. For example, if the edge-image was originally divided into 9 sub-regions, two more overlapping regions would be added in both the x and y directions thereby creating a 5x5 overlapping division of the edge-image. This would result in 25 separate sub-regions requiring to be searched (Figure 4.5).

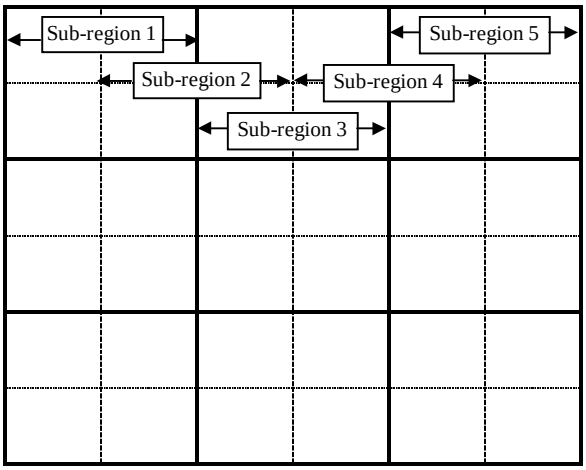


Figure 4.5 : Example of Overlapping Image Sub-Regions

For scaling the template feature, there are two approaches:

- Independent axis scaling is recommended when the user digitizes his own feature to match, as a freehand sketch will most probably not contain a uniform scale. This will allow for the feature to scale by different amounts in the x and y directions.
- Global scaling is recommended for the matching process if the user chooses an existing feature, either from the feature library or from an image, to search the image library against. Global scaling assumes a constant scale change in all directions. This is usually the case when images are scanned at different resolutions or taken with different focal lengths or flying heights. For example, if the developed matching algorithm determines, through analyzing the quadrant shifts, that scaling in the x direction only is required, the entire feature will scale the same amount in all directions.

In both cases of scaling, each quadrant remains intact. That is to say, when the template feature pixels within the quadrant scale, all the pixels move in the same direction the same amount. It can be seen therefore that in the case of scaling the feature larger, there will now be spaces between the features pixels from one quadrant to the next after expansion. This is logical, as data cannot be inserted into the newly created empty spaces because this detail did not exist in the original scan. One method to circumvent this lack of data, and an area of further research, may be to use fractal image compression techniques, where, independent of how large an image is scaled (or zoomed in), there is always new detail shown.

The analysis of the quadrant shift directions and distances proceeds first through translation, then rotation then scale. The amount of translation is taken as the average distance calculated from the 4 quadrants. For rotation, the transformation of coordinates for the query feature pixels are calculated using Equation 11 and Equation 12.

$$row' = row * \cos(-\alpha) + col * \sin(-\alpha) \tag{11}$$

$$col' = col * \cos(-\alpha) - row * \sin(-\alpha) \tag{12}$$

where α is the angle of rotation between the x' axis and the positive x axis. The rotation angle begins at 180°, is negative in the counter clockwise direction, and positive in the clockwise direction. If upon the next iteration the quadrant shifts determine a rotation angle in the opposite direction, then the previous rotation angle is halved and the sign changed. This halving and sign changing (if necessary) of the rotation angle continues until an arbitrary limit (e.g. 20) of iterations is met or until the template feature settles onto its final matched position. This process of arbitrarily choosing an initial rotation angle, then continue to halve it until the final rotation angle is determined solves

the problem of calculating an initial rotation angle when only a clockwise or counter clockwise direction is given.

For all instances of translation, rotation and scaling, there are "strong" and "weak" cases. Many of the total 1296 different direction combinations mentioned earlier can be grouped into similar cases within the weak and strong definitions and others can be ignored altogether. For example: if all four quadrants vote to move in the same direction, this obviously is considered as a strong case. If three of the four quadrants vote to move in the same direction and the fourth quadrant votes to move in one of the other directions, this is also considered as a strong case to move in the direction indicated by the three quadrants that agreed with each other. Then there are the cases where only two of the quadrants vote to move in the same direction and the other two vote to move elsewhere (other than equal to each other). This is considered as a weak case and is handled only after all the strong cases have been tested for both translation and rotation.

After the strong cases for translation are tested for, the strong cases for rotation are then considered. These include the cases where all four quadrants vote to rotate in the same direction. In the cases where only three of the quadrants vote to rotate consistently in the same direction and one quadrant points to another direction, the template feature will rotate only if this case doesn't already match one of the strong cases for translation.

The order of testing for query feature movement within the image therefore is critical for maintaining matching efficiency. After the strong and weak cases for rotation have been tested, the cases where only two of the four quadrants point to the same direction and the other two point elsewhere (but at the same time the quadrants taken together don't fall into any of the previous rotation categories), are considered.

The final transformation category to test for consists of those combinations of directions that equal to scaling in either axis direction independently or together. The remainder of the combinations of quadrant voting will consist of cases where all four quadrants point to different directions that do not correspond to one of the translation, rotation, and scaling. In this case, the template feature will shift to a new position within the sub-region and begin the matching process anew.

4.4 A Query Template Matching Example

The following example will explain the matching process between a query feature and an edge-image from the image library (Figure 4.6).

- The user will search the feature library for a suitable template feature to modify, or search the image library for a suitable template feature to extract from an existing image and modify, or draw from scratch a template feature sketch.
- Once the template feature has been located/sketched, its number of rows and columns are recorded along with the total number of edge pixels in the sketch. The centroid of the template feature is calculated using the center of mass of the feature. The (row,col) coordinate of this pixel will be used as the origin for template translation, scaling and rotation within the image. The centroid will also be used as the origin for the template quadrant system of edge pixel voting.
- The template feature gets superimposed on the image in position “sub-region 1”. The testing of the feature edge pixels begins and the votes for each pixel to either stay or move are recorded. If the pixel in question votes to move, it

is determined in which direction and how far with higher weights given to shorter distances. After all the template feature edge pixels have been tested, the sum of all these directions and distances are determined separately for each of the four quadrants within the template feature.

- From analysis of the edge pixel voting patterns, a decision is made to translate/scale/rotate the template feature within the sub-region. Once shifted into its new position, the template feature re-calculates all its pixel's votes and new distance values and directions are determined. The searching is stopped when the number of pixels in the template deciding to stay where they are is greater than the number wanting to move to a new location, or the iteration maximum is reached.
- If the searching determines that there are no good matches found in sub-region 1 or that a good match was indeed found, it jumps to sub-region 2 and so on until all sub-regions have been checked independently. The best-matched position for all sub-regions is recorded and the best one of these is used as the final matching percentage and position for the template feature in this particular edge-image.

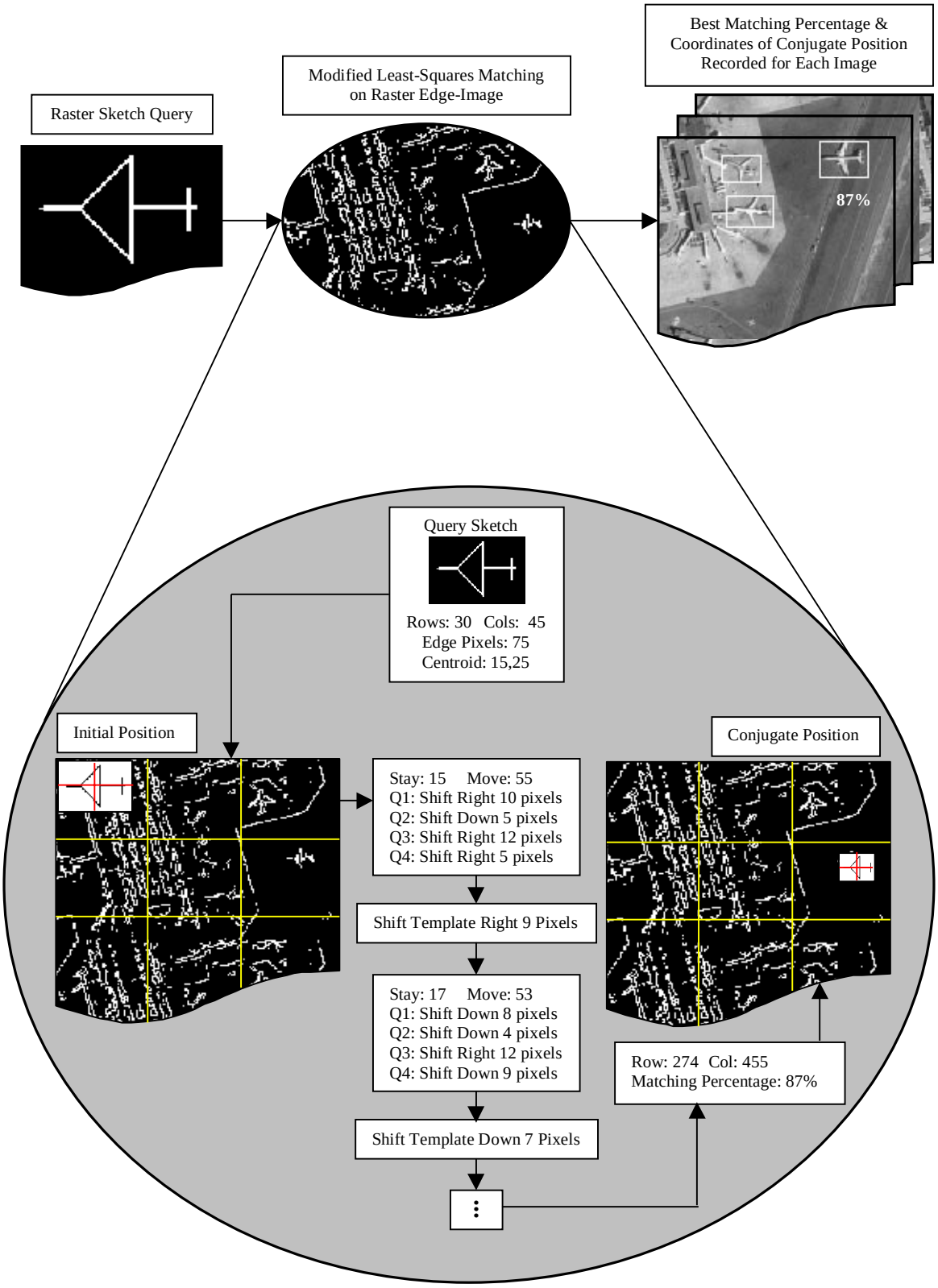


Figure 4.6 : Example of Raster Query Processing on Edge-Image from Image Library

4.5 Chapter Summary

In this chapter, a description of the matching process within **I.Q.** is presented. In general, a query feature is divided into quadrants at its centroid and then superimposed on an image. The edge pixels in each of the quadrants then vote as to which direction and the distance they would like to move. An average direction and distance is determined for each quadrant separately and then analyzed to give an overall position shift for the template feature. This shift can be one of a translation, a rotation or a scaling. The process repeats for a maximum number of iterations or until a match is found. The template then moves to another location (sub-region) within the image where the matching procedure begins anew. The final matching position for the template feature within the image is taken as the best matched position from all the sub-regions.

The modified least-squares matching algorithm developed in this thesis is a novel theoretical extension of traditional lsm to function with binary raster data and subsequently without good initial approximations of the conjugate positions.

Chapter 5

Feature Library

The feature library is a hierarchical organization of raster query feature templates. It is through the exploitation of the unique **I.Q.** feature library that on-line query and retrieval of raw raster imagery can be realized. This is accomplished through linking the features in the feature library to imagery in the image library (Figure 5.1).

Some issues to consider when determining which images need to be linked to which features need to be addressed. These include the concepts of *same*, *similar*, and *different*. For example, when should a query sketch be considered to match the same to a feature from the feature library (or to an object in an image), or similar, or different?

We suggest that the "cut-off" values chosen for these three matching categories are both application and user dependent. Different applications require different types of imagery and perhaps the shape of objects contained within them do or do not differ very much. In either case, the range of matching percentages selected to mean that two features should be considered as the same, similar, or different will vary. Also, for the same application, some users might wish to restrict the imagery returned from a query by allowing for near perfect matches only when considering if two features are the same while other users will accept imagery having a much broader range of matching percentage values. As an example of how the values for these ranges could be selected, one might choose to limit the number of features allowed on the parent level, e.g. by broadening the *different* value range.

Depending on what value ranges the user chooses for these three categories, the feature library will re-structure accordingly with adjustments to its width and depth. This in turn will have a direct affect on both the number of images that get returned to a given query and on the time required to return this result.

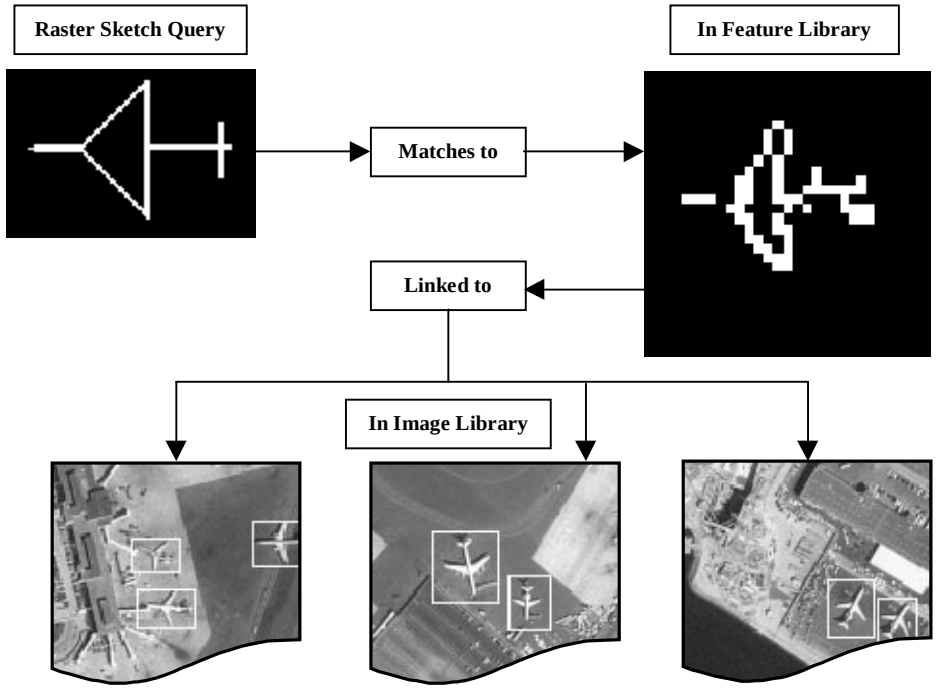


Figure 5.1 : Raster Query Workflow Via Feature Library Linking

5.1 Objective & Contribution

The main objective of this chapter is to introduce the novel approach taken by this thesis to establish an organized, hierarchical structuring of shape templates in a feature library. The overall structure of the feature library will be presented, illustrating its multi level hierarchical organization. A typical query process will be described in detail, showing how a new query feature template returns the linked imagery query results and subsequently how it gets inserted into the tree, becoming the feature library's newest

member. Finally, the concept of feature housecleaning will be introduced, explaining the need of and methods for ensuring feature library consistency by testing for and removing any unnecessary duplicate members.

The major scientific contribution of the feature library indexing strategy is its structured, progressive filtering mechanism that facilitates efficient on-line querying of raw raster imagery. To accomplish this, matching results at a specific node in the tree are used to eliminate complete branches and alternatively, to consider only a few branches. This permits us to reduce the search space from a large database of imagery to a progressively refined and limited group of feature outlines.

5.2 Feature Library Organization

The feature library is an organized arrangement of distinct feature outlines (i.e. image-object shapes) and links to image files where such features appear. The role of the feature library is to provide the crucial link that allows us to reduce the search space of a query from a large image database to an abridged group of features. In order for the query to be efficient, the feature library needs to be optimal. The optimality criteria are three: the members of the library should be *exhaustive* (thus being able to describe all possible input features); the members of the library should be *independent* (avoiding unnecessary duplications); and the members of the library should be *organized* (such that related members are grouped and stored dependently). The three properties, when satisfied, are equivalent to an ideal library, which is approaching a structured base spanning the space of shapes.

Feature library organization employs three parameters: a matching percentage above which objects are considered to be the *same*, a percentage range within which

objects are assumed to be *similar*, and a matching percentage below which objects are considered to be *different*. Combined, the three parameters define the degree to which the feature library approaches the ideal library outlined previously. High values for *same* ensure the independence of elements at the same level of the library tree. In Figure 5.2, and for the remaining examples in this chapter, the 80-100% range is used to define *same*, the 50-79% range to define *similar*, and matching percentages below 50% define what is meant by *different*. These values were chosen as they seemed a good fit to the data generated by the experimental results, explained further in Chapter 6.

Therefore, all features on the same level anywhere within the organized feature library structure are considered as *different* if they match less than 50% to each other. Child features at any level are considered *similar* if they match between 50% and 79% inclusive to their respective parents. All unnecessary duplicates, i.e. those features that match 80% or greater to an existing feature in the library are removed as they will be considered as the *same*.

During a query process, a query sketch is first matched against the features at the parent level in the tree. The process then progresses to child, grandchild and other levels as necessary. Using the above intervals for illustration, the hierarchical levels of the feature library are organized in the following manner:

- **The Primary Parent Level :** The primary parent level is the level against which every new query shape is first tested (Figure 5.3). The features at this level are the roots of the multiple trees that comprise the feature library. Individual features within the parent level are considered *different* if their mutual similarity percentages are below 50%. If the matching percentage between a query feature and a parent feature

is the same, the links the parent feature has to the images in the image library are returned as the results to the query. If more than one parent feature matching same to the query feature exists, the links of the top matching parent (highest matching percentage) are returned first. In practice, this case of multiple parent matches is unlikely due to the mutual dissimilarity between the parent features.

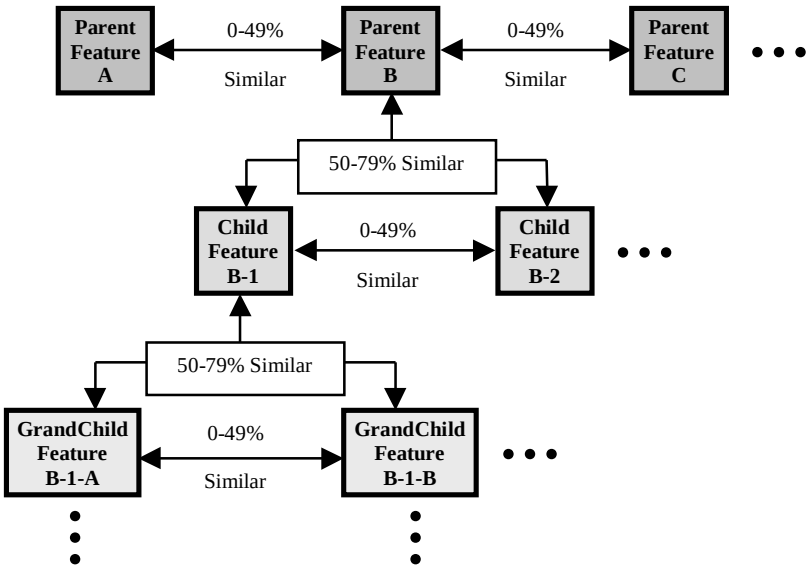


Figure 5.2 : Example of Feature Library Hierarchy

If the matching percentage is different to all existing parent level features, the query feature is considered as different to the parent feature(s) and is a candidate for insertion into the feature library at the primary parent level (Figure 5.4). Subsequent off-line processing is then required to establish its links to images in the image library.

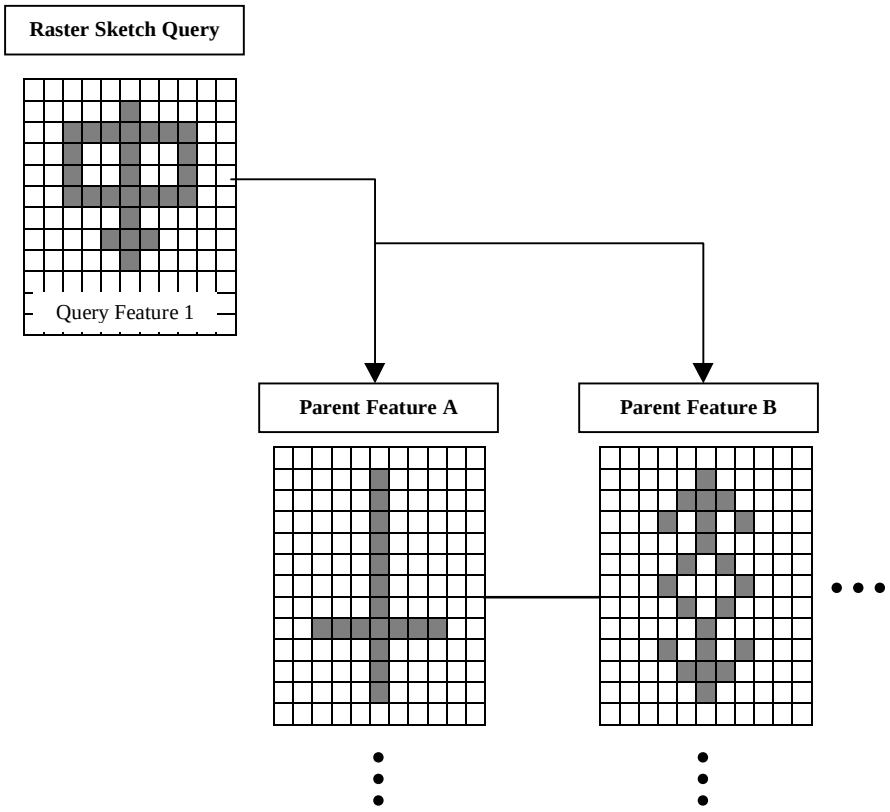


Figure 5.3 : Query Feature Matching at the Primary Parent Level.

Query Feature 1 is tested against all the features at the primary parent level of the feature library. It matches best with Feature A, i.e. the matching percent is 14/26=53.8%

- **The Child Level :** if the query feature matched in the 50-79% range to any parent feature, it is considered "similar" to the parent and is then tested against this parent's respective child features. In the case of multiple parent candidates, we select the one with the highest percentage first and continue with other candidates in order. If there are as yet no child features for the selected parent feature, the query feature gets inserted into the feature library as a descendent of the best-matched parent. Its image links are added and prioritized through off-line matching. If child features already reside at this level for any of the selected parents, the query feature gets matched against each of them. If the matching percentage is in the 80-100% range, the query

feature is considered as the "same" as the corresponding child feature and the links of this child feature to the images in the image library are returned as the results to the query. If the matching percentage is in the 0-49% range, the query feature gets inserted into the feature library at the child level of the best-matched parent, and additional links are added and prioritized through off-line matching.

- **The Grandchild Level :** If after matching in the 50-79% range to a parent feature, the query feature matched in the 50-79% range to any child feature, the query feature is tested against any grandchild features. Similar to the above levels, the relationship between child and grandchild follows that of parent and child. The processes already described take place at this library level to identify the "same" grandchild, to establish a new grandchild or to move further down the tree to the level of great-grandchildren. Combined, these similarity tests have to be satisfied within branches of the tree structure form the *insertion-testing* criterion. The children levels may be alternatively referred to as *n-child levels*, with the child being 1-level, grandchild being 2-level, etc.

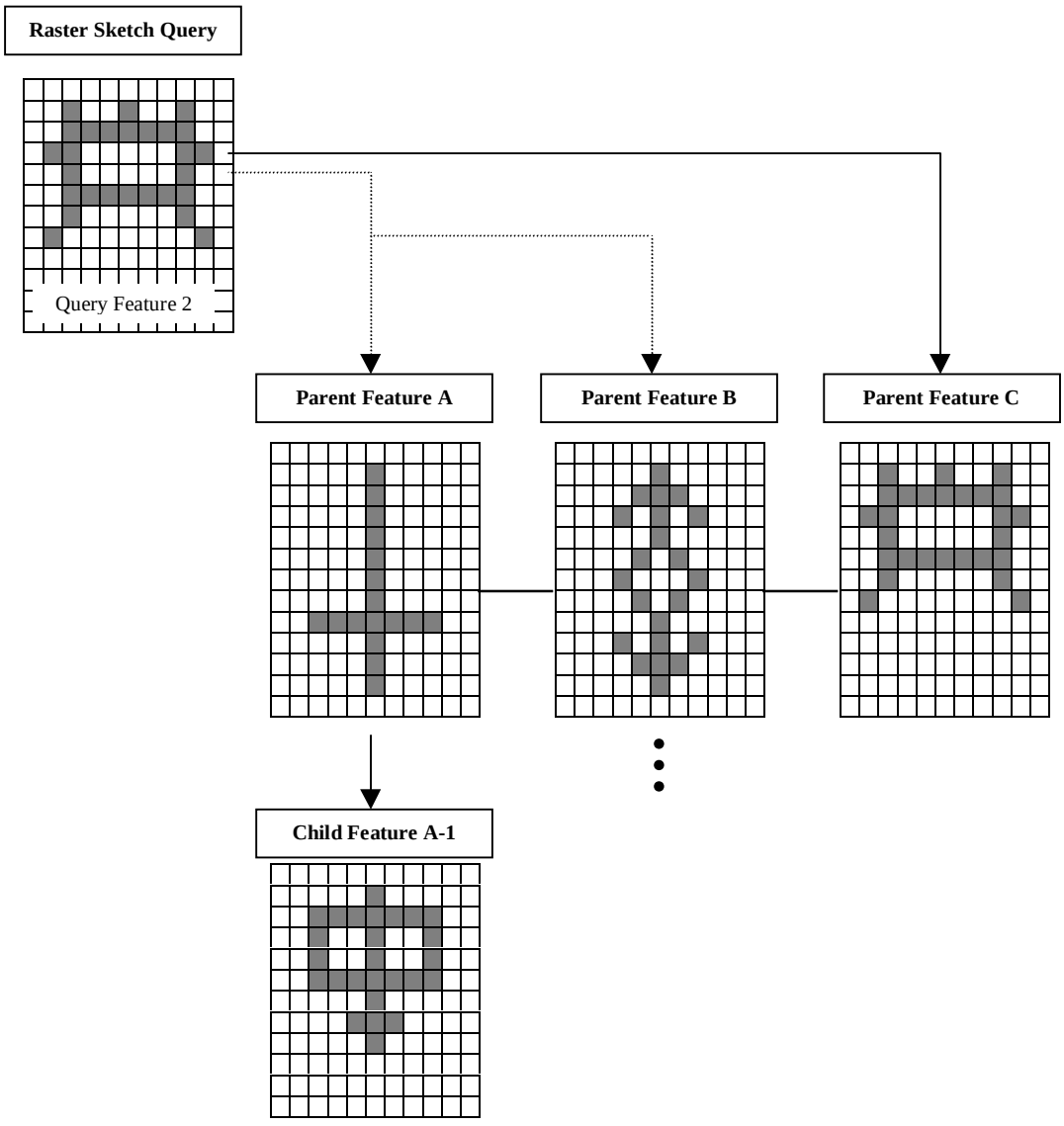


Figure 5.4 : Query Feature Insertion at the Primary Parent Level.

Query Feature 2 matched less than 50% to all existing features at the parent level so is inserted as Parent Feature C. Note also that Query Feature 1 had previously been inserted as a descendent (Child Feature A-1) of Parent Feature A.

5.3 Implementation Concerns

This "insertion testing" approach to query feature population of the feature library ensures that all child/n-child features will match at least 50% (using the lower bound for *similar* described earlier) to their parents - a fundamental requirement of the feature library. It can be envisioned however that a new query feature might match above 50% to a particular child/n-child feature but less than 50% to any of the current parent features, as in Figure 5.4. This type of query correctly results in the query feature being inserted into the feature library at the primary parent level. However, the child/n-child feature that matched better than 50% to this newly inserted parent feature might now be residing under the "wrong" parent, i.e. a parent feature that it matches less well to than the newly inserted parent feature. In this case the child feature must shift its position within the feature library to reside under the "better" parent in the tree hierarchy (Figure 5.5).

Another example of a potential inconsistency in the feature library could occur when two children of different primary parents match closer together than to any other feature currently within their respective tree hierarchies. Depending on the temporal aspect of feature insertion, this phenomenon could produce unnecessary duplicates in some cases (Figure 5.6) and necessary duplicates in others (Figure 5.7). This is due to the allowable range of percentages considered for "same" (i.e. 80-100%) and "similar" (i.e. 50-79%) matching. If the feature library did not allow for this degree of uncertainty in its matching, i.e. by allowing for exact matching only, this phenomenon would not exist. These and other potential inconsistencies are rectified by applying general feature

"housecleaning" (which utilizes the temporal aspect of feature insertion, discussed further in Section 5.4) to the feature library off-line.

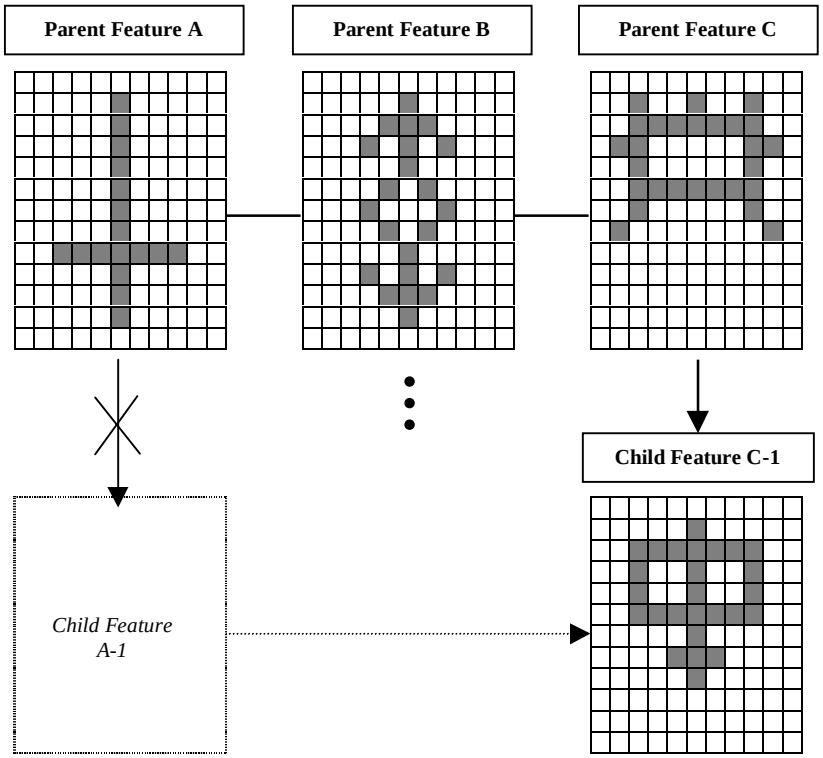


Figure 5.5 : Shifting Feature Positions.

Child Feature A-1 is tested against newly inserted Parent Feature C and finds a better match, i.e. 19/26=73.1%. It shifts position within the feature library to reside as Child Feature C-1. All of its previous image library links remain intact.

The matching parameters mentioned here are intended to serve as example values and are not unique. They can be chosen experimentally or even arbitrarily. The percentages may become lower or higher and can be considered as tuning parameters of the approach, with their variations affecting the structure of the feature library in a predictable and organized manner.

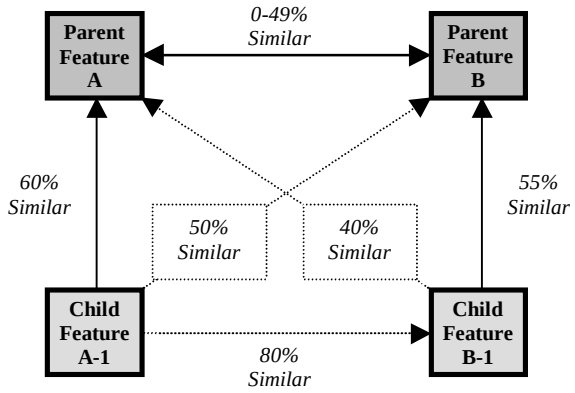


Figure 5.6 : Unnecessary Duplications.

Parent Features A and B already reside in the library. Child Feature A-1 matches 60% to A and 50% to B. It therefore gets correctly inserted as Child Feature A-1. Child Feature B-1 matches 40% to A and 55% to B so gets correctly inserted as Child Feature B-1 without testing against A-1. Child Feature A-1 is 80% similar to B-1 and 50% similar to B so should be removed from the feature library.

By narrowing the high (equivalent to *same*) matching range, e.g. using the range 90-100% instead of 80-100% we increase the number of distinct entries at each level of the tree structure, making the tree wider and less deep. This will increase the search space and make the processing time longer. However, it would also allow the queries to become very specific. For example, the queries might be: "retrieve images containing features which look *exactly* like the query sketch", as opposed to "retrieve images containing feature that look like the query sketch".

Widening the high range, e.g. using a 70-100% matching range will result in fewer entries in the database and will make the tree deeper and narrower. In turn, this produces faster queries with less accurate results. Narrowing or widening the middle (*similar*) matching range will have comparable effects to the structure of the feature library.

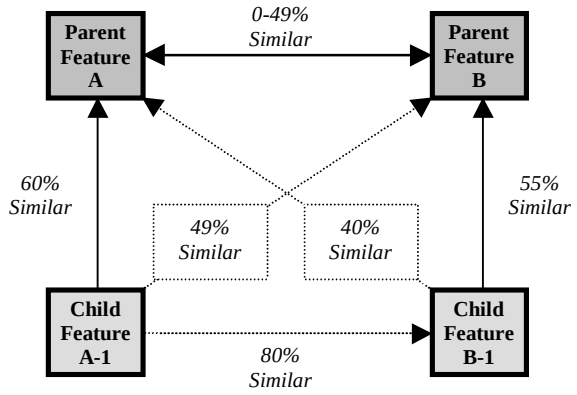


Figure 5.7 : Necessary Duplications.

Parent Features A and B already reside in the library. Child Feature A-1 matches 60% to A and 49% to B. It therefore gets correctly inserted as Child Feature A-1. Child Feature B-1 matches 40% to A and 55% to B so gets correctly inserted as Child Feature B-1 without testing against A-1. Child Feature A-1 is 80% similar to B-1 but 49% similar to B so is not allowed to shift its position under B in the tree hierarchy - thus maintaining library consistency.

5.4 Feature Housecleaning

Feature housecleaning is a process that maintains the consistency of the feature library tree hierarchy. It is run by default every night but could be invoked each time the user inserts a new feature into the library, if desired. The user should be aware however that while feature housecleaning is in progress, the feature library is off-line to further querying until completed. Feature housecleaning ensures that each of the parents are themselves unique and that all n-child features are residing under their proper parent. It minimizes the theoretical possibility that unnecessary "duplicate" features could exist somewhere in the tree hierarchy (i.e. n-child features that are between 80% and 100% similar). This possibility of unnecessary duplicate features exists because:

- The feature library is not symmetric. A case can be envisioned where Parent Feature X matches less than 50% to Parent Feature Y without the reverse being true.
- The feature library is not transitive. Another case could have Feature X matching greater than 50% to Feature Y and Feature Y matching greater than 50% to Feature Z but Feature X matching less than 50% to Feature Z.

To check for these conditions, feature housecleaning utilizes a self-organizing table called the "Temporal Feature Index" (TFI). The TFI (Table 5.1) is a two dimensional cross-referencing table of feature filenames together with their respective matching percentages. Newly inserted features into the feature library are simply appended to this table and all their matching percentages recorded as they occur, thereby recording their relative insertion time. The time of insertion is important as it reduces both the number of features to test and the number of features to test against. For example, it is not necessary to re-test Child Feature A-2-A against Parent Feature A as this matching percentage is already known and recorded previously at time of insertion.

Feature housecleaning searches the TFI top down beginning with the first feature found (i.e. Feature A in Table 5.1). It takes this feature and matches it against all other features inserted after it, provided there does not already exist a matching percentage for it in the table.

If the matching percentage is in the 80-100% range to any of the subsequently added features, Feature A will want to remove itself from the feature library. If the matched feature is a parent level feature, the image links for Feature A get passed to this "same" parent feature before deleting itself from the library. If the matched feature is not

a parent feature, then the matching percentages for all the nodal features above this match beginning at the primary parent level are tested. If all pass the 50-79% similarity test, the feature is an unnecessary duplicate and therefore passes its links to this "same" feature and removes itself from the feature library. If the parent/child feature had n-children of its own, they are tested in turn and re-inserted into the tree similar to their shifting parent. Therefore, the insertion testing criteria is invoked for each feature in the TFI, with all of its children (if any) also re-inserted following it before the next feature's turn in the TFI.

Temporal Feature Index

Feature/ %	A	B	A-1	C	C-1	A-2	A-2-A	C-2	C-3	C-3-A	C-3-B
A	**100**										
B	**44**	**100**									
A-1	**60**	**23**	**100**								
C	**33**	**12**		**100**							
C-1	**28**	**56**		**61**	**100**						
A-2	**66**	**23**	**39**	**55**	**33**	**100**					
A-2-A	**76**	**55**	**29**	**51**	**49**	**70**	**100**				
C-2	**45**	**11**		**77**	**38**			**100**			
C-3	**65**	**34**	**19**	**67**	**21**	**32**	**61**	**20**	**100**		
C-3-A	**55**	**54**	**17**	**50**	**45**	**27**		**45**	**56**	**100**	
C-3-B	**53**	**37**	**5**	**53**	**26**	**7**		**40**	**57**	**42**	**100**

Table 5.1 : Temporal Feature Index for Figure 5.8.

If all insertion tests are satisfied, a child feature will shift its position in the tree hierarchy under a new "parent" - providing it is less than 50% similar to any existing children already occupying this level. This eliminates the possibility of a shifting feature ending up in a position within the tree "worse-off" than where it was before (Figure 5.8).

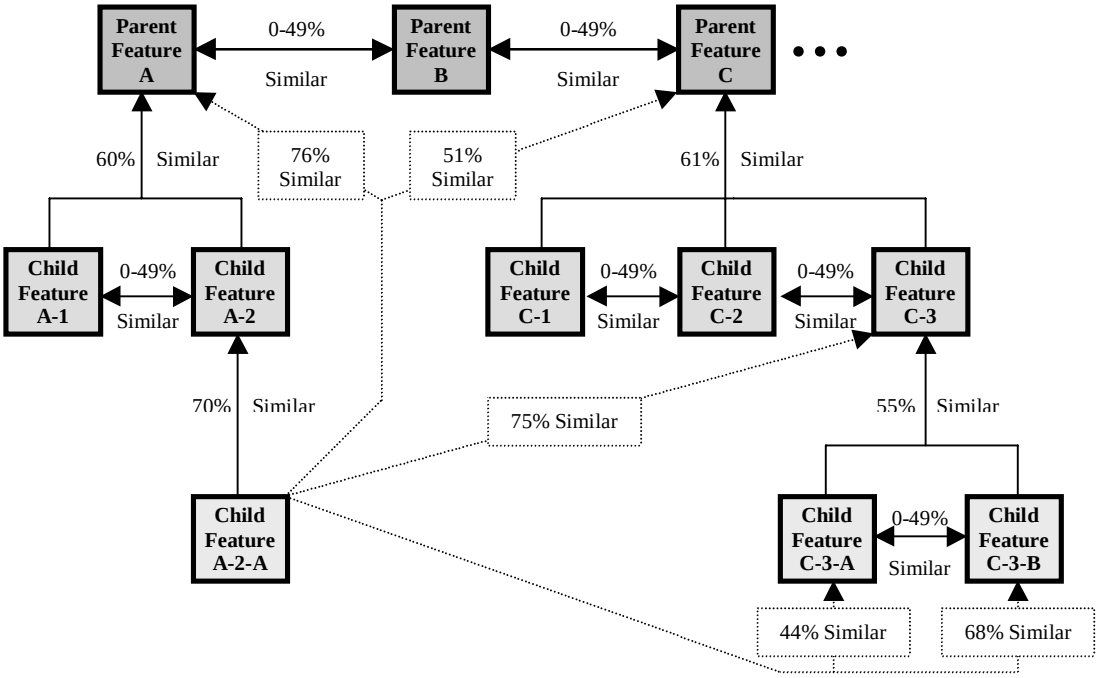


Figure 5.8 : Feature Housecleaning.

Feature A-2-A matches better to C-3 than to A-2 and should perhaps be a descendent of this "parent". However, because C-3 has a child (C-3-B) that matches above 50% to A-2-A but less than its current parental match of 70%, it doesn't shift its position within the tree as this would make the feature library inconsistent. Had Feature A-2-A matched above 70% to C-3-B, it would have shifted position with the tree hierarchy as a child of C-3-B.

It also ensures that the internal consistency of the library is maintained, i.e. that all features reside under their best matched "parent", all things considered (i.e. satisfying the insertion testing criteria). Feature housecleaning terminates once it has completed one pass through the TFI table.

This approach to feature housecleaning applies the highest weight to the matching percentage of a direct parent. Other approaches could apply a higher weight to the root (i.e. primary parent) or consider the average matching percentage of all the nodes

composing the path from the root to the target feature in the corresponding tree. Depending on which approach is taken and on the matching percentages used to determine the levels of feature similarity, the feature library will either be deeper or broader in structure. For example, if the "similar" matching range is increased, the tree will be deeper but narrower. One optimum combination of matching percentages with organizational strategy would be the one that allows for the shortest search time for any given query. Any overall best strategy will need to be verified through experimental results as mathematical models for determining least cost paths through data with varying degrees of uncertainty unfortunately do not yet exist.

5.5 Chapter Summary

In this chapter, we introduce an organized and hierarchical feature library to manage shapes in a tree structure. It is through the organization and exploitation of the feature library that allows for on-line querying of raw raster imagery. A defining characteristic of the library is its range of matching percentages acceptable at each node and from this, its subsequent multiple root nodes. In regard to this range of acceptable values allowed at each level in the hierarchy, the feature library varies from a typical tree indexing structure. The resulting tree is without a global ordering due to the fact that the matching relations between library features are neither symmetric nor transitive and that the temporal order of feature insertion into the library affects its structure. Any unnecessary duplicate features are removed from the feature library and its overall structure re-arranged through the implementation of a feature housecleaning algorithm. This is a recursive procedure that re-inserts all the features, followed by their respective

children, within the library through the analysis of their matching percentages stored in the Temporal Feature Index (TFI).

In Chapter 6, the experimental results of this unique approach to using raster sketches for digital image retrieval will be presented.

Chapter 6

Experimental Results

In this chapter, the implementation of the **I.Q.** methodology will be discussed in more detail together with experimental results. The modified least-squares matching algorithm and the feature library working together comprise the essential components of the image query-by-sketch approach proposed by this thesis. This approach outperforms traditional least-squares matching due to an adaptation to operate with raw binary-raster imagery. It should be noted however that the modified least-squares method and the feature library are indeed separate components of the complete query system and therefore can be tested as such. For example, the matching algorithm can be tested on template features or complete images without consideration for the feature library implementation, and likewise, the feature library can be tested for efficiency of its organizational structure without reference to the matching module. Such a separation of components allows for improvements or enhancements to each independently of the other. This capability facilitates the overall testing process and also allows for the two major elements of this thesis to be analyzed individually.

6.1 Traditional Least-Squares Template Matching

In Section 2.3, the inherent characteristics of traditional least-squares area-based matching were described. These include its reliance on gray-scale imagery, and that it requires good approximations for the initial query template positioning within the image.

Gray-scale imagery is a pre-requisite for traditional least-squares to function properly as it requires that every pixel within the query template and the image contains information. This brightness value information, contained within each pixel, is needed in order to calculate the gradient at each pixel location (Figure 6.1). Pixel gradients in the x and y directions, instead of the actual pixel gray-scale or brightness value are typically used for matching purposes in order to minimize the radiometric differences between identical objects. Corresponding pixels from the query template and the image will have different radiometric and possibly geometric qualities even if they represent the same image-object because the two images were not taken from exactly the same exposure station at exactly the same moment in time. Therefore the contrast and brightness of the same image-object point and the geometry of the same image-object outline will never be exactly the same from image to image.

9	**7**	**8**	**3**	**5**
4	**2**	**5**	**3**	**8**
2	**7**	**4**	**2**	**3**
2	**1**	**6**	**3**	**3**
7	**1**	**4**	**9**	**7**

Figure 6.1 : Pixel Gradient Calculation

If the array of brightness values in Figure 6.1 were to be considered as the gray-scale query template, only the shaded pixels would have a gradient in both the x and y directions. Most other pixels will not have a gradient in one of the two principle directions and the corner pixels will not have a gradient in either direction. The gradient of the central pixel would be calculated as:

$$\dot{g}_x = 2 - 7 = -5$$

$$\dot{g}_y = 6 - 5 = 1$$

It is due to this gradient calculation that occluded features cannot be found within the image. For example, if an imaged feature was only half contained within the image, (or otherwise occluded by cloud cover, overhanging trees or buildings) traditional least-squares would not be able to find it for the simple reason that the query template is restricted to remaining completely within the image space (Figure 6.2). This is because all pixels within the query template and underlying image are required for the calculation of shift distance/direction, a fundamental assumption of the traditional least-squares solution. If any pixel information is missing or does not contain significant brightness value information due to the occluding object, traditional least-squares will fail.

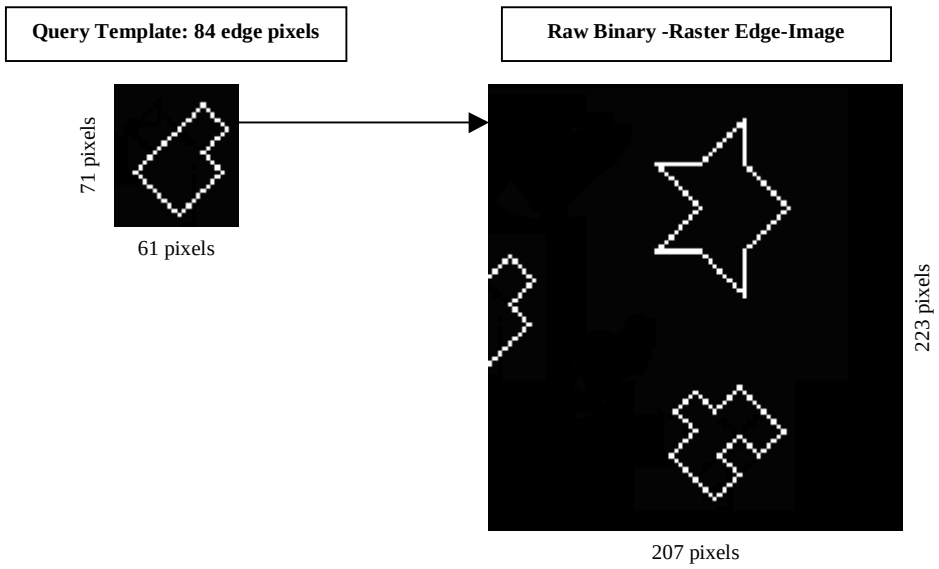


Figure 6.2 : Occluded Image Feature

Image resampling is another characteristic of traditional least-squares matching. It is the process of assigning new gray levels to the pixels within the image where the query template has just been shifted. The first sampling of gray levels for the image will be those pixels directly under the overlaid query template. However, these gray levels

will only be the gray levels of those image pixels for the first phase of iterations. After the x and y shift values have been calculated at the end of the first phase of iterations, there will be new gray levels tested from the image. If the two shifts were integer values, the new gray levels would simply be the gray levels of the new pixels that the query template has shifted onto. What happens in practice is that the shift values are not integer but are instead real values. This leads to shifts of partial pixel distances that requires a resampling of the image pixels before the next iteration can be processed. For example, the new shifted location must take as its gray level a percentage of the gray levels that surround it with the inverse of the distances to these surrounding pixels used as the weights for determining what percentage of a gray level it will be given.

After all image gray levels (that underlie the shifted query template) have been resampled, the iterations will re-start at finding the best transformation parameters for this new position. Once new transformations parameters are found, new shift values are calculated and resampling is repeated. All of these procedures will cease when the x and y shift values are less than a preset tolerance, typically set to $1/10^{th}$ of a pixel. The final position of the query template (when the preset tolerance is reached) is the final coordinates of the conjugate match within the image.

In the example of Figure 6.2, traditional least-squares would require a good initial approximation of the conjugate feature position for it to function properly. This is because the "pull-in" range for this approach is limited to a distance of about half (or less) the dimension of the input query template [Baltsavias, 1991]. In our case of raw imagery, where neither image-object information nor interior or exterior orientation parameters for the image are known a-priori, good initial approximations cannot be

provided. Traditional least squares therefore would have to sequentially step the query template through the entire image, pixel by pixel, in order to test against all possible conjugate positions, an extremely time consuming process. For this example image, it would require 22,192 initial re-positionings (and corresponding gradient calculations and matrix manipulations) plus any subsequent shifting and resampling operations. Assuming 1" per initial re-positioning, this translates into more than 6 hours of processing time for this very small sample image alone. This amount of processing overhead is the main argument behind the contemporary view that raw raster imagery is unsuitable for real-time querying.

Stepping the input query template through the binary-raster image of Figure 6.2 would result in a first position match (top left corner of image) of just over 98%. With such a high matching percentage, the subsequent shift distances calculated for the x and y directions are insignificant which renders the resampling of the new, shifted query template's position's underlying image pixels, meaningless. This is an example of a typical problem encountered with traditional least-squares matching on binary-raster imagery and is an important illustration of why it is not suitable for such purposes.

Along with large corrections in translation, traditional least-squares cannot accommodate equally large corrections in scale or rotation. For example, input query template A in Figure 6.3 appears in the image rotated 180º. Traditional least-squares methodology is incapable of handling such large corrections due to an intrinsic property of the mathematical model that the query template function is a minor geometric transformation of the image function as both are assumed to be images of the same object related to the same co-ordinate system. Input query template B would suffer a similar

fate, as it cannot scale larger than its own dimensions and therefore would not find its conjugate match in the image.

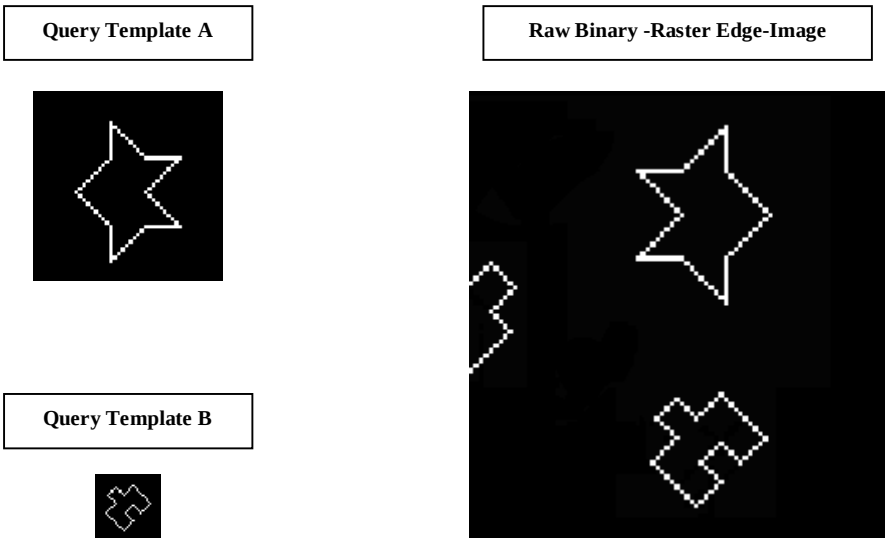


Figure 6.3 : Large Corrections Cannot be Accommodated

From the above explanation, it can be seen that traditional least-squares lacks the flexibility required to function effectively with raw binary-raster imagery. The following sections will describe how the modified least-squares approach proposed in this thesis behaves under similar operating conditions.

6.2 Modified Least-Squares Template Matching

Various individual feature templates and edge-imagery were used to test the robustness of the modified least-squares feature matching algorithm. For example, one edge-image used was a controlled environment for testing purposes only (Figure 6.4). It is actually a collection of sample shapes with all image noise removed. A sample query template feature for this image could be the outline of an "L" shaped building complex.

Using this query template/edge-image combination, a typical, straightforward matching process is described in (Figure 6.5).

Figure 6.4 : Edge-Image of Sample Shapes

The query feature begins its search within the image at the top left position. It works its way through the sub-regions, as described in Chapter 4, until it finally arrives in the sub-region where the feature exists, i.e. at the bottom right of the image. Once the query feature gets translated into this final sub-region, it iterates until the best matching position is found. A description of the actual matching process within this last sub-region follows (Figure 6.6):

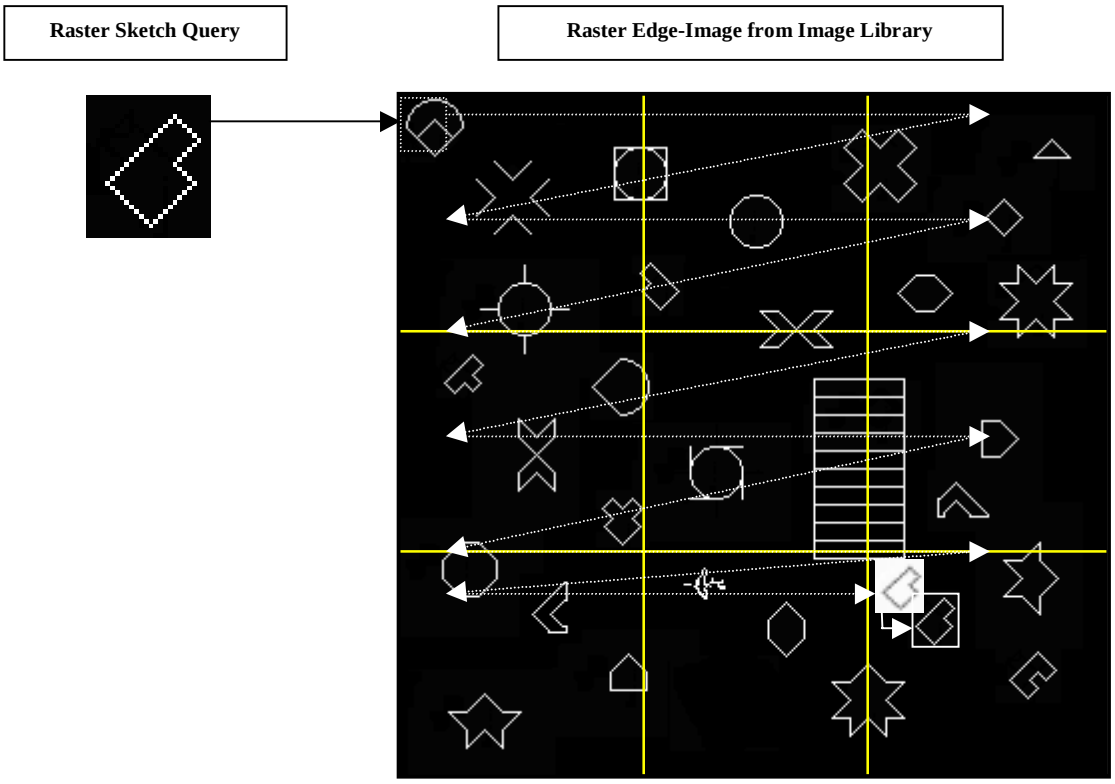
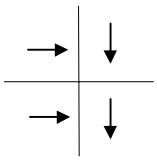


Figure 6.5 : Example of Typical Query Input and Feature Matching Process

Note: a detailed description of the matching process for the final (bottom right) sub-region is presented in Figure 6.6.

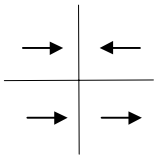
Program Output **Event Summary**

```
Start iter=1,mp=0.000000, iter-1=0,mp-1=0.00
move=43, stay=3
q1dir=4, q2dir=1, q3dir=1, q4dir=4
q14dir=4 and other dir=1, movedown 9 pixels
End iter=1,mp=6.522000, iter-1=0,mp-1=0.00
```



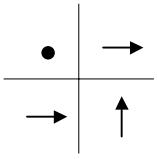
Iteration 1
- 43 pixels vote to move
- 3 pixels vote to stay
- shift down 9 pixels
- match=6.5%

```
Start iter=2,mp=0.000000, iter-1=1,mp-1=6.52
move=46, stay=0
q1dir=3, q2dir=1, q3dir=1, q4dir=1
3 outof 4 dir=1, moveright 7 pixels
End iter=2,mp=0.000000, iter-1=1,mp-1=6.52
```



Iteration 2
- 46 pixels vote to move
- 0 pixels vote to stay
- shift right 7 pixels
- match = 0%

```
Start iter=3,mp=0.000000, iter-1=2,mp-1=0.00
move=18, stay=28
q1dir=1, q2dir=5, q3dir=1, q4dir=2
q13dir=1 and other dir not same, moveright 2
pixels
End iter=3,mp=60.869999, iter-1=2,mp-1=0.00
```

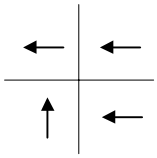


Iteration 3
- 18 pixels vote to move
- 28 pixels vote to stay
- shift right 2 pixels
- match = 60.9%

Figure 6.6 : Matching Process For the Final Sub-Region.

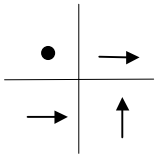
Program Output **Event Summary**

```
Start iter=4,mp=0.00000, iter-1=3,mp-1=60.86
move=28, stay=18
q1dir=3, q2dir=3, q3dir=2, q4dir=3
3 outof 4 dir=3, moveleft 2 pixels
End iter=4,mp=39.130001, iter-1=3,mp-1=60.86
```



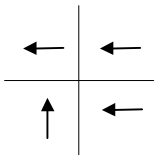
Iteration 4
- 28 pixels vote to move
- 18 pixels vote to stay
- shift left 2 pixels
- match = 39.1%

```
Start iter=5,mp=0.00000, iter-1=4,mp-1=39.13
move=18, stay=28
q1dir=1, q2dir=5, q3dir=1, q4dir=2
q13dir=1 and other dir not same, moveright 2
pixels
End iter=5,mp=60.869999, iter-1=4,mp-1=39.13
```



Iteration 5
- 18 pixels vote to move
- 28 pixels vote to stay
- shift right 2 pixels
- match = 60.9%

```
Start iter=6,mp=0.00000, iter-1=5,mp-1=60.86
move=28, stay=18
q1dir=3, q2dir=3, q3dir=2, q4dir=3
3 outof 4 dir=3, moveleft 2 pixels
shift left ARBITRARILY, 1 pixel
End iter=6
```

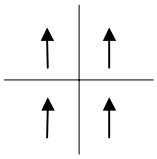


Iteration 6
- 28 pixels vote to move
- 18 pixels vote to stay
- shift left 2 pixels
- Arbitrary shift to break out of loop

Figure 6.6 Cont. : Matching Process For the Final Sub-Region.

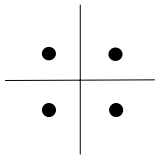
Program Output **Event Summary**

```
Start iter=7,mp=0.000000, iter-1=6,mp-1=0.00
move=46, stay=0
q1dir=2, q2dir=2, q3dir=2, q4dir=2
q1234dir=2or5, moveup 1 pixels
End iter=7,mp=0.000000, iter-1=6,mp-1=0.00
```



Iteration 7
- 46 pixels vote to move
- 0 pixels vote to stay
- shift up 1 pixel
- match = 0%

```
Start iter=8,mp=0.000000, iter-1=7,mp-1=0.00
move=0, stay=46
q1dir=5, q2dir=5, q3dir=5, q4dir=5
q1234dir=5, shift entire template
End iter=8,mp=100.000000, iter-1=7,mp-1=0.00
```



Iteration 8
- 0 pixels vote to move
- 46 pixels vote to stay
- shift 0 pixels
- match = 100%

Figure 6.6 Cont. : Matching Process For the Final Sub-Region.

Note: in this example the template feature enters and breaks out of a repeating loop before finding its final best match. This conjugate position for the template feature is illustrated in the final sub-region of Figure 6.5.

Using a single 180MhZ Pentium Processor computer, this feature matching approach takes about 2'30" to complete on a typical 512x512 block of image space. On a single feature within the feature library, the same matching procedure takes only about 2".

This significant difference in time required to match a query template against a single feature as opposed to a complete image illustrates the need and importance of

incorporating an organized feature library into an overall, image database query strategy. In addition, due to the features in the library being singular and free of image noise, the matching algorithm could operate with a greater success rate where combinations of scaling and rotating the query template are required. More on situation handling of the modified least-squares matching algorithm is presented in the next section.

6.3 Matching Accuracy

A collection of 48 tests of the matching accuracy were performed to see how well the implemented feature matching algorithm worked in practice using "real world" features and images (Table 6.1). Various features from imagery in the image library were extracted, and in some cases modified, and tested against the same library of images. The results of the tests are grouped into three categories:

- The "Positive" group describes if the "correct" images were returned to a query with the corresponding matching percentage inserted in Table 6.1. Correct in this sense means that the image(s) where the feature resides was retrieved somewhere in the top 10 prioritized list of images. This number was chosen somewhat arbitrarily although it does represent 10% of the imagery in the image library. Therefore, if the correct image is returned somewhere within the top 10% of the total amount of imagery contained in the image library, it will be considered as a successful or positive result to the query.
- The "Commission" group is defined as those instances when "incorrect" imagery (i.e. those that do not contain the object being searched for) did get returned as a result to the query. Commission errors therefore are very subjective and would vary from user to user as each has their own idea on

what “similar” features should look like. To include this group in the results would require extensive user testing with an averaging of their subjective results. Therefore, in the interest of time and that analyzing subjective behavior associated with individual preferences is beyond the scope of this thesis, this group is not included in the testing.

- The “Omission” group is defined as those instances when “correct” imagery (i.e. those that do contain the object being searched for) did not get returned as a result to the query. Therefore, this group describes if the correct images were not returned in the top 10 list of prioritized images with the corresponding matching percentage inserted in Table 6.1.

The “Position” column in the table indicates the actual position the image occupied in the prioritized list of matches returned from the query.

From the test results, the implemented matching algorithm returned the correct image to the query, somewhere within the top 10 list of prioritized images, 77% of the time, and to the top 20 list, 85% of the time. The conditions under which the matching algorithm failed to find the correct images for a query are varied, and will be discussed in the next section on matching limitations.

Test #	Feature Name	Positive (%)	Omission (%)	Position
1	221870_f1	67.4		1
2	221870_f2	100		1
3	221870_f3	39.3		2
4	221870_f4	28.8		10
5	221870_f5	100		1
6	225870_f1	44.8		6
7	225870_f2		18.8	26
8	225882_f4	38		1
9	225882_ff		32.4	21
10	233898_f	61.9		9
11	237898_f	34.6		7
12	241890_f	52.6		1
13	241894_f	60.3		1
14	241894_f2	46.2		4
15	241898_f		30.8	19
16	241902_f	39.1		1
17	241902_f2	27.9		6
18	241906_f1	32.5		5
19	241906_f2	46.4		7
20	241906_f3	43.2		10
21	241906_f4		40.4	17
22	241906_f5		36.2	16
23	241918_f1	52.1		3
24	241918_f2		33.3	26
25	241918_f3	51.2		8
26	245886_f	37.6		1
27	245890_f	35.3		10
28	245902_f		26.6	15
29	245902_ff	29.4		5
30	245906_f		22.4	19
31	245910_f	32.2		1
32	245910_ff	30.4		1
33	249886_f	39.9		1
34	249906_f	35.3		5
35	249910_f	55		8
36	253894_f	97.7		1
37	2m_1_f	46.2		6
38	2m_1_ff		29.2	22
39	2m_2_f		23.3	28
40	2m_3_f	62.8		1
41	2m_3_ff	29.7		4
42	2m_4_f	37.9		1
43	2m_5_f	61.4		3
44	2m_6_f	80.8		1
45	2m_7_f		27.1	20
46	2m_8_f	73.8		1
47	2m1_f3	22.3		2
48	2m1_f4	49.3		4
		37/48=77%	11/48=23%	

Table 6.1 : Summary of Matching Accuracy Tests.

6.4 Matching Limitations

As expected, it was discovered during the implementation phase of this research that the translation, rotation and scaling operations on the template feature tend to be sensitive to image noise. For example, the presence of spurious edges around an image-object may cause the voting for the pixel shifts to change dramatically. This potentially could lead to erroneous over-corrections, or back and forth shifting inside a loop.

To counter the noisy image problem, filters could be applied to the edge-images to rid them of edges shorter than a given length. Or, the image could be further divided into more sub-regions, with the shifting template advancing in smaller increments throughout the matching process. Both approaches were tested with improved success rates for template feature matching.

In the first case, artificial test images were created like that used in Figure 6.4. For the second case, where the image is further subdivided, it too resulted in better matching accuracy but of course produced the counter effect of increasing the time needed to match a template feature to an image. Solving the problem with more sub-regions is in essence approaching a traditional least-squares solution, i.e. it provides good initial approximations by increasing the number of initial template positioning within the sub-regions.

Another limitation, found during the implementation phase, was the methodology used when determining the amount of angular rotation to apply to the template feature. For example, the idea of halving the rotation angle upon the next iteration did not always produce correct results in practice. The theory of quadrant voting used in this research

does in fact handle the rotation problem, through closer analysis of the distance and direction averages, but was beyond the scope of our implementation.

In practice, due to noise in the images or other factors, after the template applied its first rotation, the subsequent iteration sometimes produced disruptive results. For example: on the subsequent iteration, the template feature would vote to shift, which would misalign the rotation origin; or the template feature would vote to scale, which changed the locations of the pixels in the quadrants.

In addition to the above two conditions, two more cases of diminished matching accuracy were discovered during the implementation and testing phase of the research. One such instance occurred when the template matching would not settle onto its final matching position and instead entered into a shifting loop. Here, due to the arrangements of the pixels within the quadrants, a template shift would be determined of a certain amount of pixels in a given direction. After moving to this new position and recalculating the pixel votes for each of the quadrants, it is determined that a shift back to the previous template position is required, and so on. This situation is analogous to the difficulties that traditional least-squares matching has with repetitive patterns.

In order to break out of this shift/reverse shift loop, the last three template matching iterations were monitored. If a repetition was found, for example, if the matching algorithm determined to first shift 3 pixels to the right, then 3 pixels to the left, and then 3 pixels to the right again, the template would be arbitrarily shifted a distance of 1 pixel in any direction and the pixel positions recalculated. An example of this exact problem was presented in Figure 6.6.

A final limitation of the matching strategy was found when trying to match template features where the centroid of the feature lies outside the feature boundaries. This situation amounted to one or more of the quadrants having very few or even no pixels available to vote on distance/direction shifting. A solution to this problem, where the template feature's pixels are not (roughly) evenly distributed amongst the four quadrants, would be to first look upon the pixels of the feature as points on a line and then calculate the best fit line to them. The angle between this line and the vertical axis could be determined and the quadrant system could be rotated to align with this new orientation. This has the effect of distributing the feature pixels evenly throughout the four quadrants, which in turn allows for more intelligent shifting decisions.

6.5 Feature Library Testing

Experiments comparing the image retrieval time required with and without the feature library implementation showed quite significant results. Search and retrieval times for the feature library implementation were tested against a sample database of 94 features. These features comprised a grouping of both manually sketched and extracted objects from existing imagery. Typical shapes were circular, representing cooling towers, rectangular, representing buildings, and unique shapes like outlines of airplanes and other visible image-objects.

Testing the matching algorithm for a single template feature against a complete but unstructured feature library, i.e. one that has not been organized into a hierarchical tree structure, resulted in search times averaging around 2" per feature. This is in contrast to matching the same feature against a single aerial image, which took about the 2'30" to complete (per image). A substantial saving in search times therefore is realized

through matching query sketches to linked features rather than to the original images, which was to be expected.

The next step was to organize the 94 features into their proper tree hierarchy according to the rules outlined in Chapter 5. To accomplish this, a recursive algorithm was developed that processed two complete passes through the TFI (see Appendix A). The first pass through the list of features parallels initial feature insertion into the feature library. It resulted in a tree structure that was 60 features wide and 3 deep with 10 features being removed (deleted) as unnecessary duplicates. After processing the second pass, i.e. simulating feature housecleaning, the tree reduced to a structure of 48 features wide and 3 deep with 5 additional features being removed as unnecessary duplicates. The search times therefore were roughly cut in half (i.e. 48 x 2"), demonstrating the need for an organized feature library structure, as the minimum number of features to test against was reduced from 94 to 48. Additional testing into the tree, if required, did not take significant amounts of time, as it was only three features deep with typical search times averaging around 2" per feature.

The final structure of the feature library, with the same 94 features, depends on the values chosen for the variables "same" and "similar" and on the order of feature insertion into the library. For example, the above test used a same percentage lower bound of 80 and a similar percentage lower bound of 50. If however the values were changed to 70% for same and 30% for similar, the final structure of the feature library would find 27 features removed as unnecessary duplicates with 48 features on the parent level and 6 levels deep. Modifying these variables therefore affects the form of the

library but not retrieval efficiency, i.e. given a range of values acceptable at each node, the library will always re-organize into its most efficient state.

When the order of feature insertion into the library was reversed, the resulting structure had 16 features removed as unnecessary duplicates and 47 features on the parent level. The test results show therefore that the final organization of the feature library also depends on the sequence of feature insertion. This is logical, as the matching between library features is neither symmetric nor transitive, i.e. although there is a local ordering of the features, a global ordering among the features cannot be defined, resulting in possibly different tree structures (depending on the sequence of insertion).

Also, there does occur the case where one feature could be a child of either of two (or more) parents since it matches exactly the same to both. Therefore, depending on the order of insertion, this feature could end up under a different parent. Similarly, depending on the order of insertion, a feature could get removed, or not, as an existing feature that matches the "same" to it could be in a path that is not reachable. However, this is considered as a necessary duplicate so is acceptable in this case.

Tree structure variation therefore is due then to the order of feature insertion into the library, to the acceptable range of values allowable at each node and to the dynamic nature of the library itself, where features are constantly being added in the form of user queries.

In order to guarantee that a "canonical form" can be found, i.e. one that has the maximum number of removed features, all possible permutations of feature insertion must be tested - which is time prohibitive because with n features, this is in the order of n! tests, and therefore not feasible.

The feature library, with its inherent organization and links to imagery in the image library, is essential if the goal of a raster image query and retrieval environment is to return results to the query in real-time. As an illustration with the complete feature library would be too space prohibitive, an example of feature library organization with a handful of those features follows. In Figure 6.7, a grouping of polygon shapes of n-order (i.e. n=3,4,5,6,8,10) are inserted before a selection of extracted image-object shapes taken from imagery in the image library. In Figure 6.8, the order of insertion into the feature library is reversed, i.e. the extracted shapes are inserted before the polygon shapes.

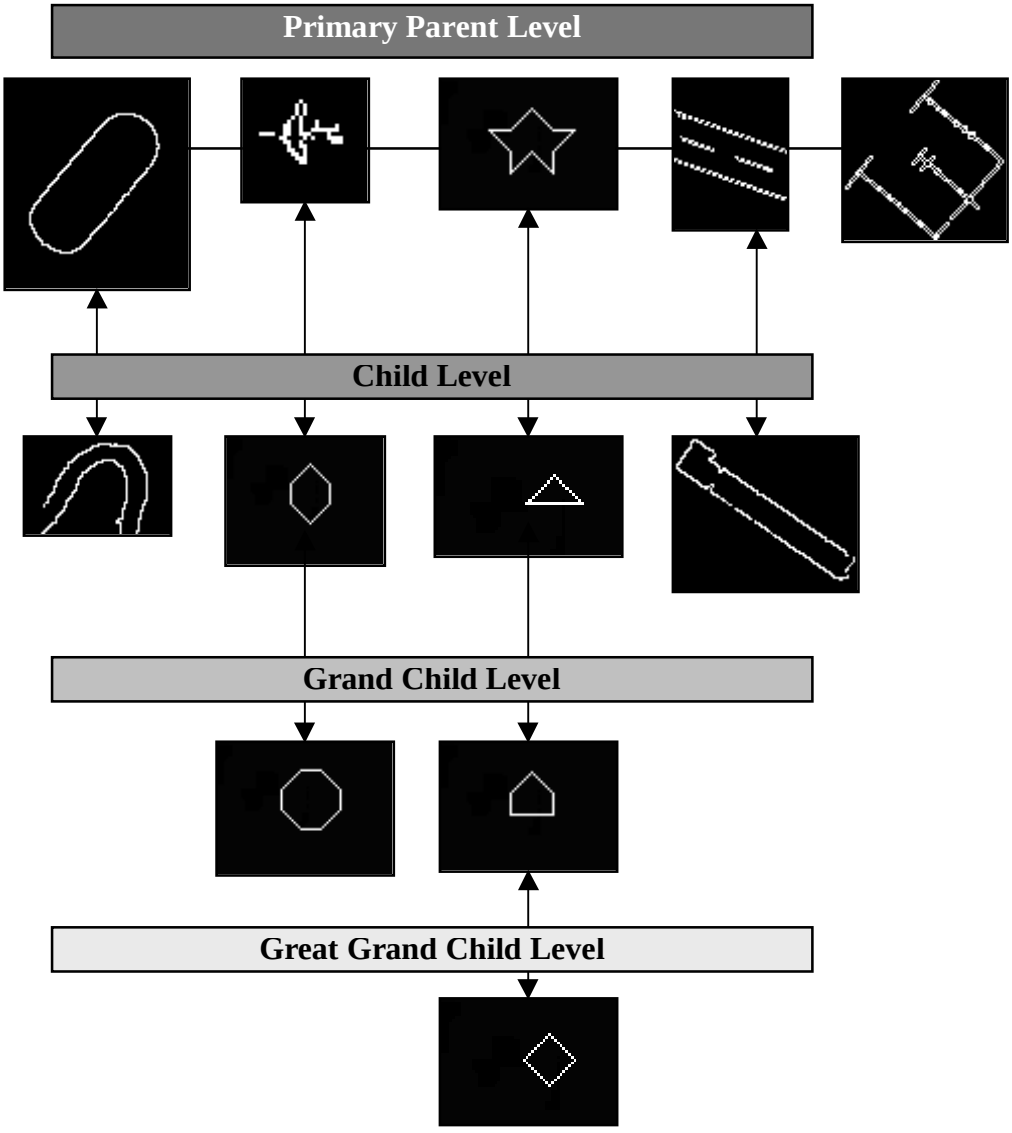


Figure 6.7 : Feature Library Hierarchy when Polygonal Shapes are Inserted Before Extracted Shapes

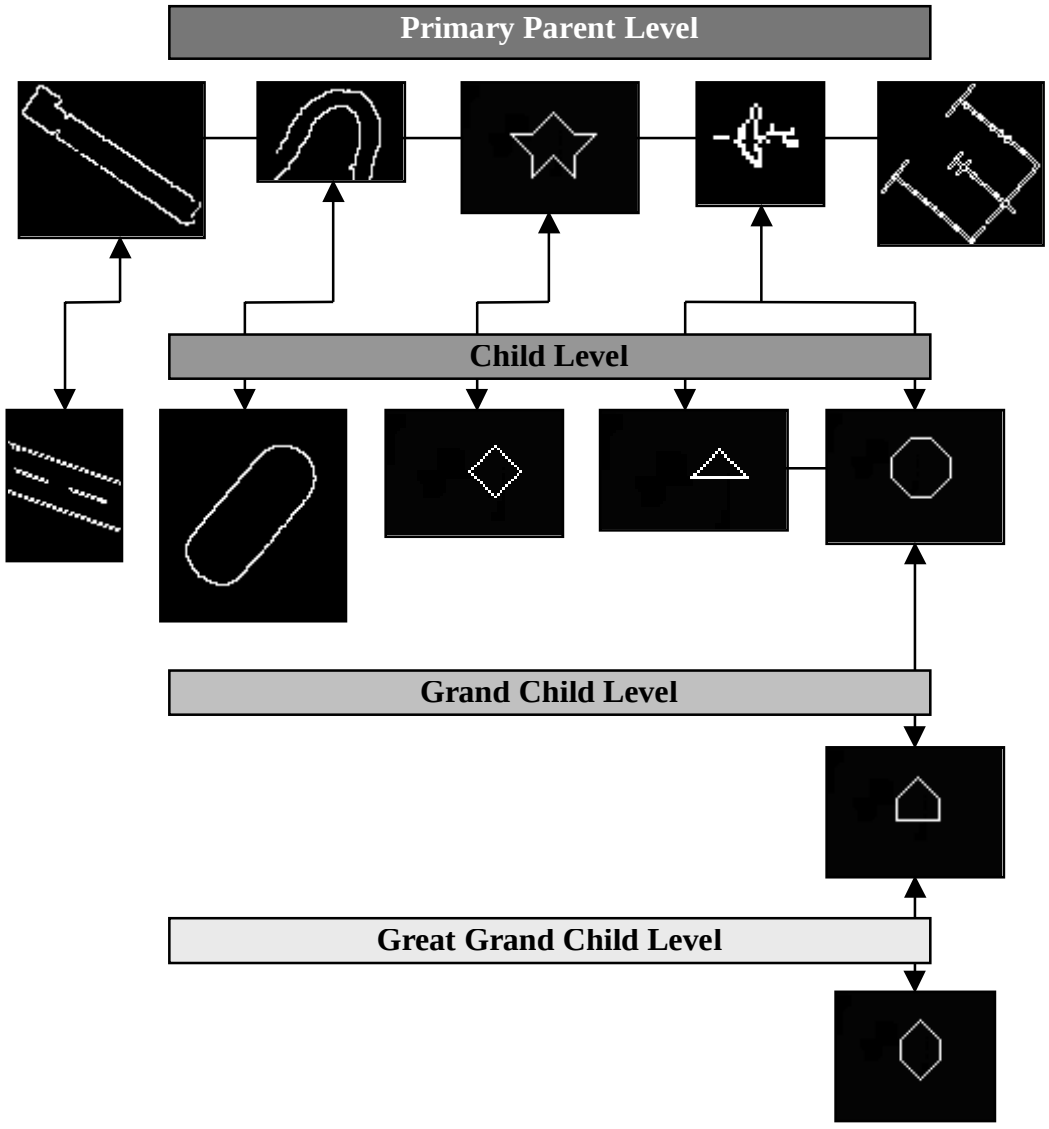


Figure 6.8 : Feature Library Hierarchy when Polygonal Shapes are Inserted After Extracted Shapes

From the above tests, it can be seen how the temporal aspect of feature insertion has a marked affect on the organization of the feature library. For example, some of the parent-child relationships are just reversed between the two figures. In Figure 6.7, the majority of the polygonal figures organized under the star parent feature, while in Figure 6.8, the star feature, while remaining at the parent level, has only one child. This is

entirely due to the sequence of feature insertion into the feature library. Also notice that in both cases, the polygonal shapes tended to group together, while the majority of the extracted features tended to group at the parent level.

In addition to feature sequence of insertion, it was shown that by modifying other variables, the feature library structure is similarly affected. While keeping the order of feature insertion constant, the overall structure and organization of the feature library can be adjusted directly by the values chosen for the variables "same" and "similar". For example, it was thought that if the value for "same" were changed from 100% down to 51%, at some point a drop off would occur in the amount of features considered as unnecessary duplicates. This cut off value therefore might be a good starting point to begin studying where the difference between two shapes can be considered as being either the same or similar. Tests with varying quantities for these values were performed to see if there were any general trends that could be deduced from the data structure (Figure 6.9).

Although not a substantial reduction, the result of this test did in fact show a small drop off in the percentage of features remaining in the feature library when the value for "same" was around 78%. From then on downwards to 50%, a decrease in number of library features seemed to fall off steadily. This would indicate that for matching values of "same" above about 80%, the feature library considers those features as unnecessary duplicates and therefore removes them from the library.

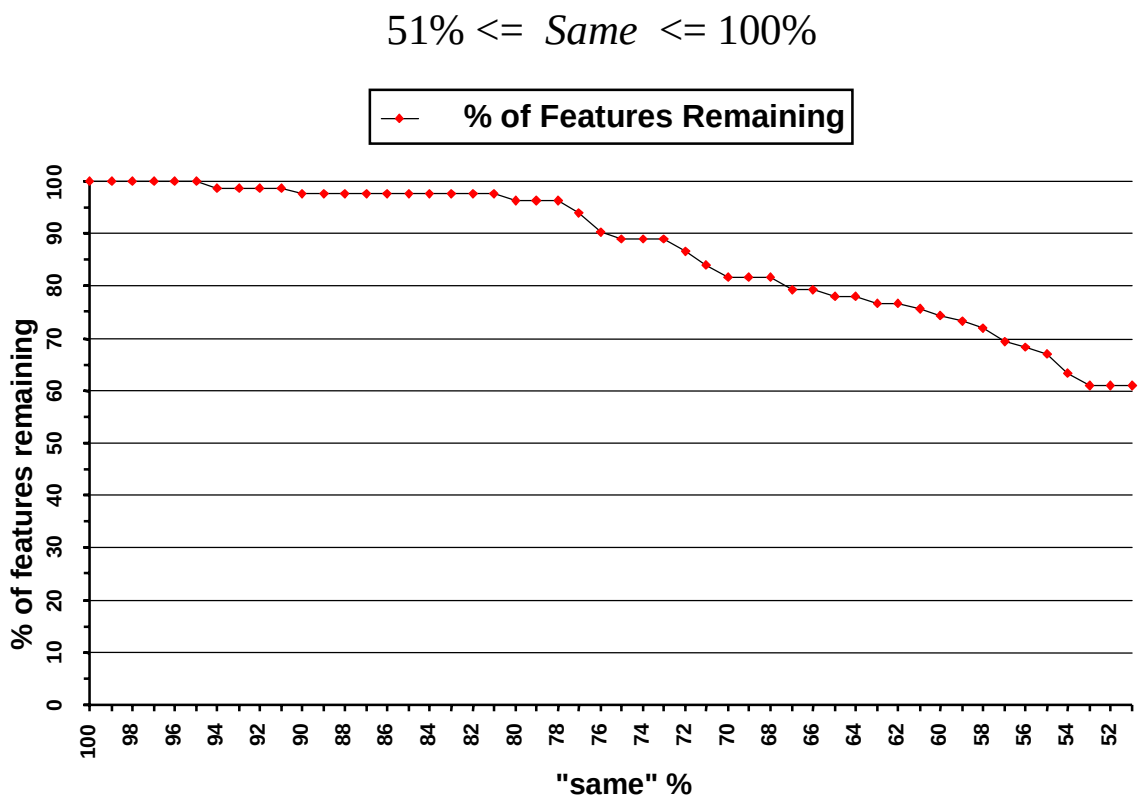


Figure 6.9 : Number of features remaining in the feature library while changing the value of "same", keeping "similar" constant at 50%.

The value of "similar" was modified between 20% and 79% to see what effect this would have on the overall structure of the feature library, i.e. the width and depth (Figure 6.10)

From this test, it was shown that there was no significant value for "similar" that produced a noticeable change to the slope of the graph. With a constant decrease in the value for "similar" a proportional constant decrease in the amount of features remaining on the parent level likewise occurred, and subsequently the depth of the structure increased to accommodate these additional features. For example, with a value of 20%

for “similar”, the final library structure was 7 levels deep with 13 features on the parent level.

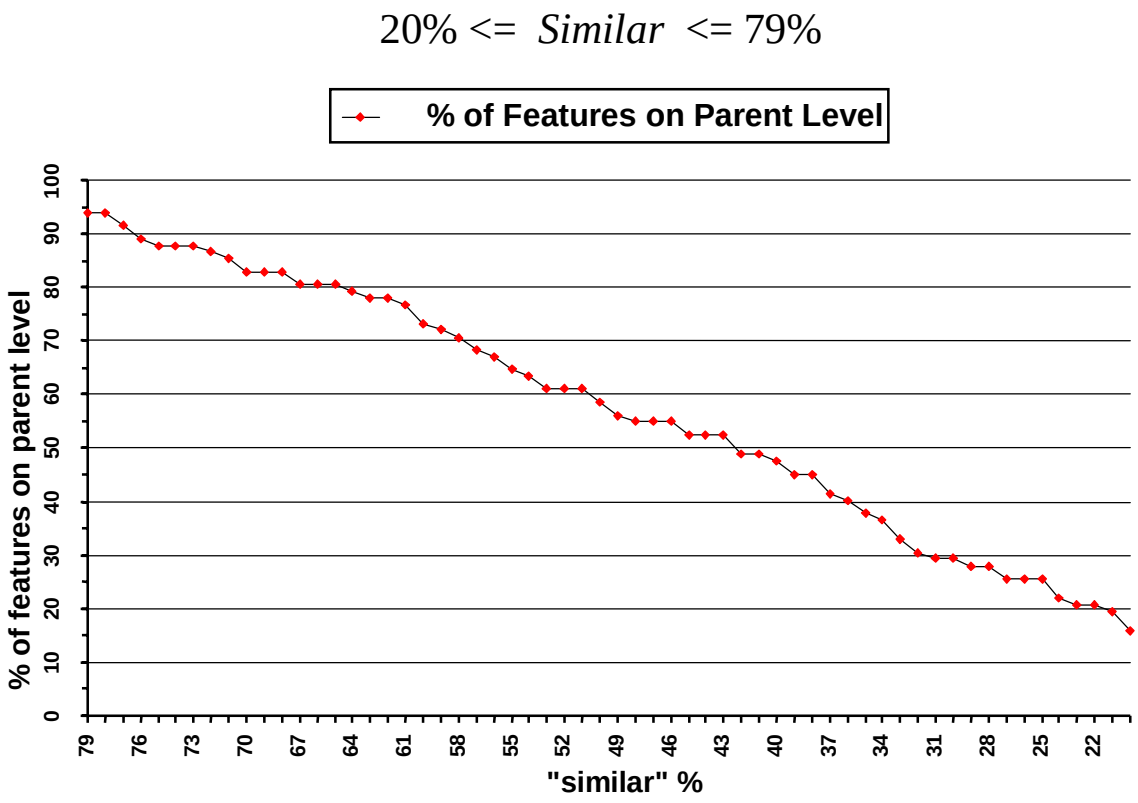


Figure 6.10 : Number of features remaining on the parent level in the feature library while changing the value of “similar”, keeping “same” constant at 80%.

6.6 Feature Library Limitations

Different users would consider different values for what is meant by the term “same” and “similar”. Also, for different types of applications, another set of different values could be valid choices as the collection of features within the library could have substantially different shapes to those found for a typical geographic application. It was decided therefore that the best way to overcome this subjective decision was to present a sample template feature to the user and a sample of about a dozen modified versions of

the template shape. The user could then click on which features in the sample he/she felt was the same as the template, or similar, or different. From this operation, the feature library could determine the average matching percentages the template had to each of the 3 groups and set the values for "same" and "similar" from them. Each user of course would probably choose different shapes for each of the categories and would therefore end up with a different organization of the feature library.

However, in the end, the final structure of the feature library is not so important as long as it retrieves to the user the images he/she is searching for. Due to this basic assumption, another solution to the question of what values should occupy the "same" and "similar" variables is to ask the user how much time he/she is prepared to wait for a query to return results. Depending on the time indicated, for example if the user chooses a maximum wait time of 10", the feature library should organize into a structure that is not more than about 10 features wide and about 10 deep. Using this constraint as input, values for "same" and "similar" could be calculated by the feature library that would give such a structure.

6.7 Summary

In this chapter, an explanation of the shortcomings of traditional least-squares matching on raw binary-raster imagery, together with the experimental results for the image query-by-sketch approach proposed in this thesis were presented. The implementation phase was divided into two separate but related modules. This allowed for the revised least-squares template feature matching algorithm to be developed and tested first without requiring a completed feature library implementation, and also for enhancements to either of the two modules independently of the other.

It was found that the matching algorithm performed more consistently in the case of single feature matching than in the case of complete image matching due mainly to the lack of image noise inherent to members of the feature library. Also, for single feature matching, it did not require initial approximations in order to find the correct match, an important improvement compared to the standard least-squares approach. However, when matching on noisy aerial images, it did require the image to be sub-divided into more sub-regions, which approximates the improved initial position requirements of the traditional least-squares case.

There were various limitations on matching performance discovered throughout testing. These included difficulties in determining the amount of angular rotation for the template, situations where the template would fall into a shift/reverse shift loop, and where the centroid of the template feature would fall outside the feature boundary thus leaving unequal amounts of pixels in the four quadrants. These issues were addressed through closer analysis of quadrant voting distances and, shifting the template feature 1 pixel in an arbitrary direction, and rotating the vertical quadrant axis to align with the best fit line through the points in the template.

For the feature library, a recursive algorithm was developed that organized the query features into a hierarchical tree like structure. By changing the values used for "same" and "similar", the shape (i.e. its width and depth) of the tree could be manipulated. It was proposed that the values for these two variables could be determined by presenting the user with a sample template feature and a collection of library features that it could match too. The user would then decide which of the library features

appeared the “same”, “similar” or “different” to the template, from which the feature library algorithm could then determine the corresponding values for these variables.

Another solution for handling this issue could have the user inputting the amount of time he/she is willing to wait for a query to return results and the tree could re-organize its maximum width/depth limits accordingly (keeping in mind match times of 2” per feature) through the manipulation of these two “same” and “similar” percentages.

Beginning in Chapter 7 the Conclusions of this research will be drawn. They include the advantages and limitations of this proposal and some ideas for future research.

Chapter 7

Conclusions

In this thesis, a proposal for querying digital image databases through the use of sketched image features completely in the raster domain has been presented. The approach combines a modified least-squares matching algorithm with a structured feature library. Together they offer a substantial improvement, in matching effectiveness and time, over the traditional least-squares approach for on-line querying of raw binary-raster imagery. An implementation of the proposed theory demonstrated its feasibility and highlighted areas for future work. In the following sections, the advantages and limitations of the proposal are discussed and some remarks on possible future developments are given.

7.1 Limits of the Traditional Least-Squares Approach

Several points were made in the previous chapter concerning the unsuitability of the traditional least-squares method for matching on raw binary-raster imagery. In fact, due to its reliance on good initial approximations, limited scale and rotation range, and gray scale imagery, it was shown to be completely ineffective for even the most basic of queries. For example, in order for traditional least-squares matching to accommodate the inherent characteristics present in raw binary-raster imagery, it will need to be modified to comply with the requirements that:

- It must function without initial approximations, i.e. without a sequential search through the image;

- It must function with rotations of up to +/-180º, i.e. without a limit to its range of rotation angles;
- It must function with scale increases larger than the input query template dimensions, i.e. without a limit to its range of scale values;
- It must be able to locate conjugate image-objects irrespective of initial query template positioning, i.e. without a limit to its pull-in range;
- It must be able to locate semi-occluded objects, i.e without conjugate features being fully contained within the image;
- It must function with edge pixel information only, i.e without every pixel containing gray-level information;
- It must search an entire image database in a fraction of the time it currently requires to match a single image, i.e. without excessive amounts of processing time.

A summary of the defining characteristics or differences between traditional least-squares and the modified approach can be found in Table 7.1. These results confirm that the proposed modified least-squares method for matching sketched object shapes outperforms traditional least-squares matching on raw binary-raster.

Traditional Least-Squares Matching	Modified Least-Squares Matching
Good initial approximations are necessary	Initial approximations are not required
High precision (shifts of fractions of a pixel)	Lower precision (shifts of integer amounts only)
Low reliability: susceptible to erroneous matches if wrong initial approximations	Higher reliability: matched pairs are more likely to be truly conjugate features
Limited rotation range	No limit to range of rotation
Limited scale range increases of query feature	No limit to range of scale increases
Limited pull-in range	No limit to its pull-in range
Cannot find semi-occluded objects	Can find objects up to 50% occluded
Cannot function with edge information only	Designed to function with edge information only
Requires excessive amounts of processing time per image	Can return multiple imagery to query in "real-time"
Sensitive to geometric distortions and radiometric noise with ambiguous matches in areas of high noise and spurious edge content	

Table 7.1 : Comparison Between Traditional LSM and Modified LSM.

7.2 Advantages of the Modified Least-Squares Approach

The limitations to traditional least-squares matching listed in the previous section were addressed by the image query and retrieval environment presented in this thesis. Additionally, until now the majority of current image query research requires that image-objects be vectorized at least to some extent. This of course requires some form of pre and/or post human intervention, as fully automated algorithms for consistent feature extraction from raw imagery do not yet exist. Also, those systems that do work in the raster domain require that the spatial domain be transformed into the frequency domain where translation and rotation can be modeled but not scale. The main advantage of this proposal therefore is that it provides the ability to query digital image databases completely in the raster and spatial domains, without any manual intervention in the form of pre or post processing of the imagery.

This task was accomplished through the modification of traditional least-squares matching theory to function effectively with raw binary-raster edge representations. This modification overcomes the constraints implicit with matching gray-scale imagery using the traditional least-squares method, for example, its reliance on good initial approximations and a limited pull-in range, by requiring no initial approximations with an unlimited pull-in range. It therefore increased the overall matching efficiency and freed the user from providing any a-priori image-object information.

Just an improvement in matching efficiency alone however will not make an image query and retrieval environment complete. Due to the size and quantity of images in an image database, it is not feasible to query on them directly. A novel approach to allow for on-line querying therefore was introduced where all previous query features are

inserted into a hierarchically structured feature library. The library organized and linked the query features to the images in the database through off-line matching. A typical query would proceed by matching the query feature against the features already stored in the library and return a prioritized list of the images linked to the best-matched features. The query feature itself would then get inserted into the feature library, get linked to images and be available for future interrogations of the image database. The feature library therefore allows us to reduce the search space of a query from a database of images to an abridged group of features.

The feature library is dynamic and self-maintaining. Every new addition gets tested against every existing element in a feature housecleaning operation that is designed to remove any unnecessary duplicates. This ensures an efficient and optimal feature library where the members approach our objective of an ideal library, i.e. related members are grouped and stored dependently (*organized*), are able to describe all possible input features (*exhaustive*) without unnecessary duplications (*independent*).

7.3 Limits of the Proposal

During the implementation phase of this research, it was noticed that the matching algorithm worked as expected, e.g. that it is affected by excessive image noise. The voting strategy employed considers all edge information within its pull in range, giving higher weight to closest edges. If this closest "edge" turned out to be image noise, the matching algorithm would still consider them for the scaling and rotation conditions set out in the procedure. If the centroid position of the template feature lay outside the feature boundaries or if there wasn't a roughly even distribution of pixels scattered throughout the four template quadrants, the template feature could potentially scale or

shift when in fact a rotation was required, or vice versa. To counter this effect, it is proposed to further subdivide each of the template feature's quadrants into additional quadrants where the voting patterns in each of the quadrants are analyzed and transformed individually. In theory, template feature quadrants could continue to be further subdivided into sub-quadrants in this way until each quadrant contains only 4 pixels, whereby it approaches the traditional least-squares paradigm.

An obvious limitation of the approach is that it does not consider the semantic aspects of the query. For example, the proposed implementation would not consider the two objects in Figure 7.1 to be different, either semantically or geometrically, in fact they would be considered about 95% the "same". This is because the approach proposed by this thesis works completely in the raster domain, i.e. based on shape information only and does not consider the semantic context of objects.

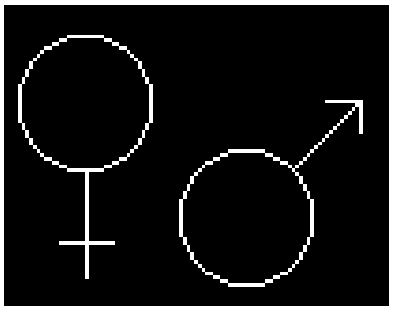


Figure 7.1 : Two Almost Identical Shapes With Completely Different Semantics.

Concerning limitations to the feature library, there might be discrepancy between different users when considering whether two features are the "same", "similar", or "different". This creates the situation where a final convergent structure to the feature library can not be predicted. The structure is also affected by constant querying of the

image library, i.e. as new query features get inserted into the feature library, the process of feature housecleaning will re-structure it accordingly to ensure that each feature is residing under its best matched parent and that there are no unnecessary duplications.

However, this dynamic aspect of the feature library, i.e. its ability to self-organize and maintain its contents is one of its advantages. The final structure of the library is not, in general, of concern to the user. What is important is that the proper images get returned to the query in real-time. Similar to other tree data structures, the final structure of the feature library is dependent on the sequence of feature insertion, and due to the lack of a global ordering, a canonical form cannot be determined.

7.4 Future Developments

The directions for future extensions of this research are many and varied. For example:

- The implementation phase could be complimented with the development of a graphical user interface;
- The matching process could be extended to include querying for configurations of image-objects;
- Semantic information could be included as a query criteria;
- Testing the matching algorithm with temporal queries for change detection;
- Considerations for querying on vector and heterogeneous databases could be examined.

7.4.1 GUI Development

The ultimate purpose for the image query-by-sketch proposal in this thesis is to function as a query mechanism for large image database with minimal interference by the operator. Possible user environments include both internet and intranet application. Users will be able to query and update, from remote workstations, the images stored on the image database server.

To accomplish this, an interface will need to be constructed that allows for these functionalities. For example, for the query operations, an initial screen acting as a window into the image database (Figure 7.2) will let the user examine the contents of the data base and allow for object extraction from the existing images. Extracted features can then be used as is or modified as the template feature for further queries (Figure 7.3).

The Metadata Menu could be the next screen accessed by the user (Figure 7.4) or bypassed completely. Choosing the metadata option will narrow the search space of the image database by eliminating those images, and therefore those feature library features, that do not meet the query criteria.

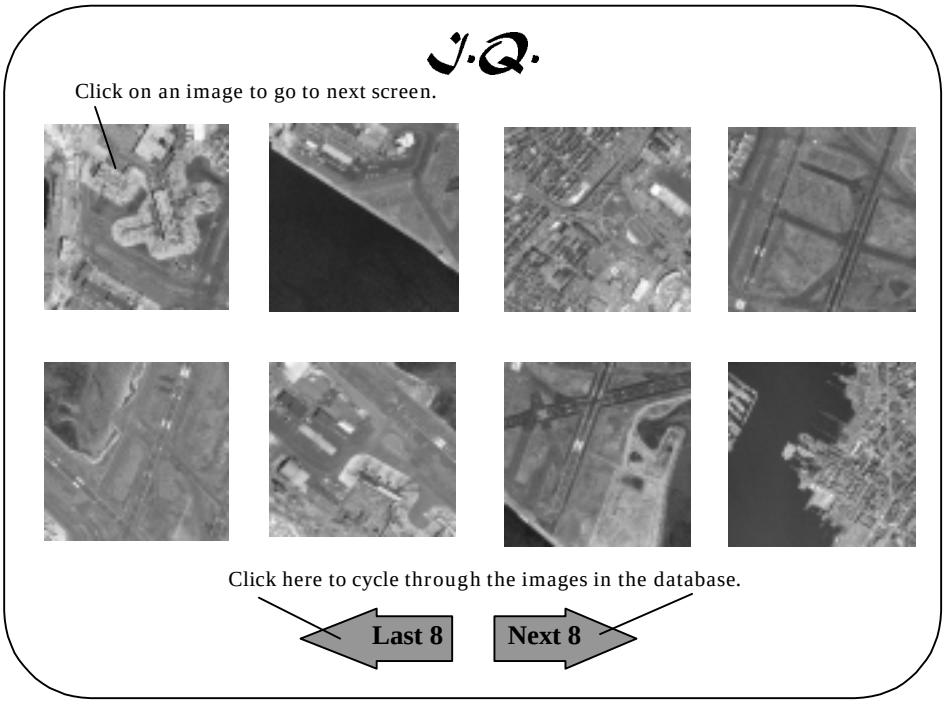


Figure 7.2 : The Initial Screen of the Image Query Interface.

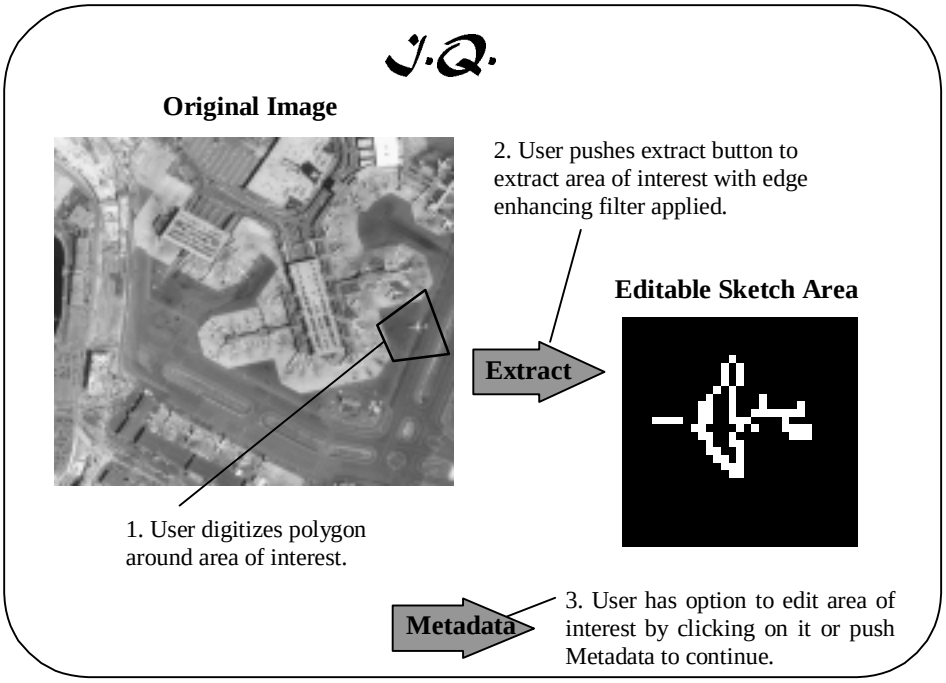


Figure 7.3 : Extract an Existing Feature to Edit or Sketch New Query Feature

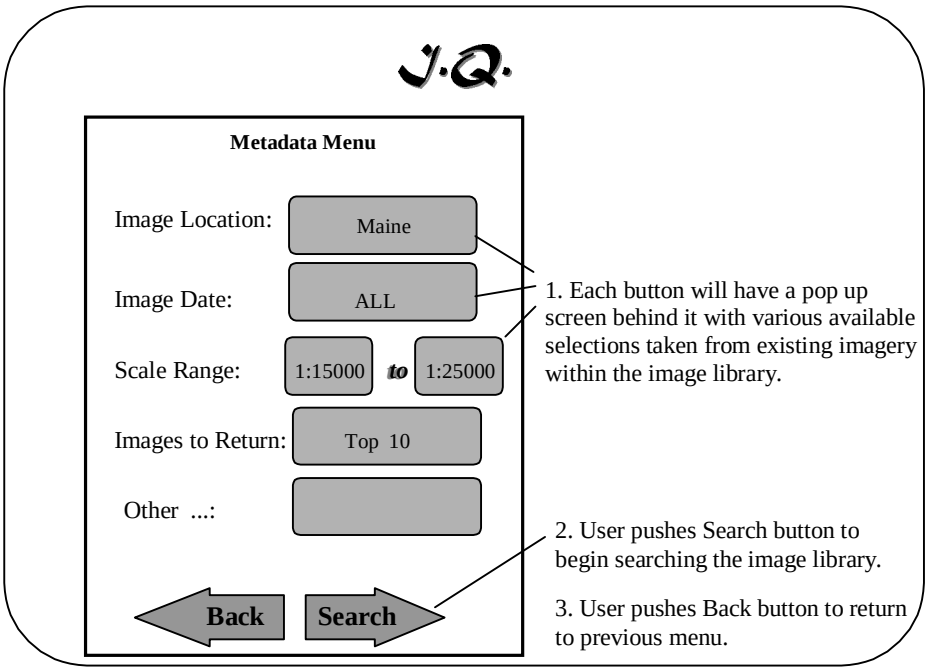


Figure 7.4 : Select or Input Relevant Metadata

A final screen containing the prioritized results from the query, with options to further edit the query feature or refine the metadata will be presented next (Figure 7.5).

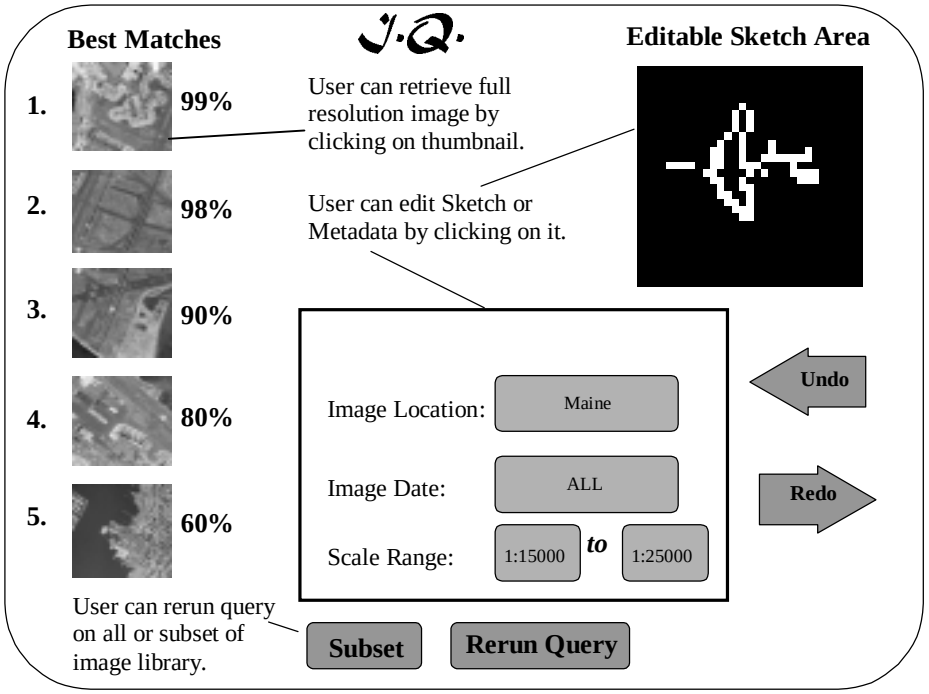


Figure 7.5 : Prioritized Query Results

7.4.2 Matching Configurations of Objects

Configurations can be matched within the query environment by adding an extension to the query building interface. What is required is that the individual objects within the sketch be recognized as such, and the pixels that make up their respective edges stored in independent arrays. This can be accomplished by clicking on an icon within the sketching environment that indicates when new feature digitizing commences and terminates. In addition, another icon can be used to group or separate existing objects in the sketch. Note that the objects continue to remain in raster form, i.e. an object is considered as the minimum bounding rectangle (MBR) array of pixels that encompasses it, some of which are turned "on" (to indicate object boundaries) while others are turned "off" (to indicate background information).

Once the query scene has been built, the individual MBR component objects that make up the scene are separable and the 9 intersection matrix [Egenhofer and Herring, 1991] can be determined for each pair of objects in the scene.

The individual sketch objects that make up the scene are then processed against the feature library and for each sketch object a prioritized list of candidate images is proposed. An analysis of the returned images for each object in the scene shows that certain images are included more than once. For each image that was returned by all of the objects in the configuration, the topological relations between the involved objects will be determined.

As described previously, when a feature gets matched to an image, its centroid coordinates within the image are recorded as well as the top left and bottom right coordinates of the query feature's minimum bounding rectangle, after scaling and rotation

have taken place (if required). Therefore, topological relations on the image are also determined through the use of the matched query features MBRs instead of the actual image-objects because the image is in raster (non-vectorized) format and therefore no a-priori information is known about any image-object (in particular, their boundaries). Also, it is straight forward to determine the MBR containing the pixels composing the translated/rotated/scaled query object and using their MBRs speeds up the topological building process and thus allows for topology to be built and queried in real-time.

To summarize then, given a configuration of image-objects, built by the user, the system will automatically compute topological, orientation, and distance relations between those objects before the actual search begins. Next, the search will commence by first retrieving all the images that match to the individual objects of the query scene. From this subset of images, only those images that have a match above a given threshold for all of the objects in the query scene will be further considered. From these remaining images, the topological/orientation/distance relations between the matched objects will be computed and compared to that of the query scene.

7.4.3 Semantic Information

Once queries have been made on the image database, additional semantic information could be stored on the objects contained within the images. For example, if a traditional query returned an image with a rectangular outline of a building, this additional data could be stored as a "hospital" in a semantic library that is connected to the imagery library within the comprehensive digital image database (Figure 7.6). A typical query using semantic information then would include, on the metadata form, an

option for the user to specify what type (or purpose, etc.) of objects should be contained within the images returned from the query, not just the shape.

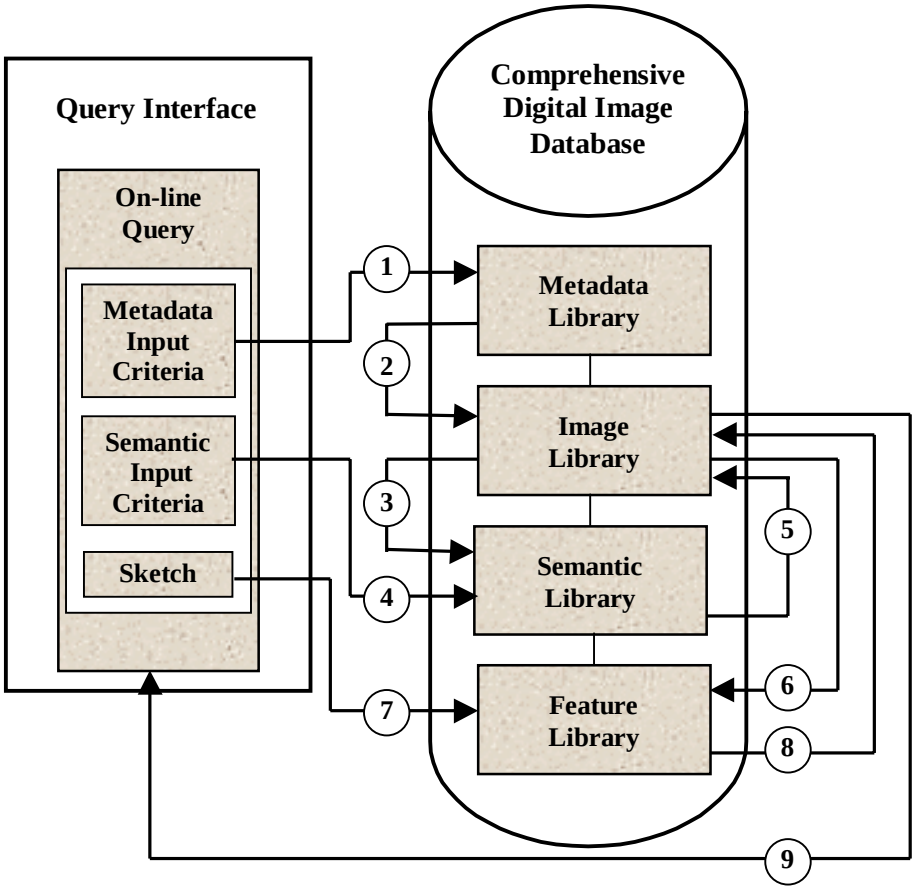


Figure 7.6 : Conceptual Model of I.Q. Environment that Includes Semantic Information.

For example, if the user decided to retrieve all images with hospitals that match to a particular shape, the query would:

- Process the metadata library and retrieve all images that match to the specified metadata criterion (e.g. scale, or location, etc.).
- Follow the links from this subset of images to the semantic library where it would identify which of these images contained hospitals of any shape;

- Use the semantic input criteria to further narrow down the list of images to search within for the required query shape;
- Use this further subset of imagery to select, through feature linking, a refined subset of features within the feature library to match the query sketch against, and;
- Feature library matching and subsequent image retrieval will proceed as described previously.

7.4.4 Temporal Queries

The technique described in Section 7.3.2 could be modified to allow for temporal reasoning on the database. By loosening up object/relation constraints and by analyzing matching percentages, we will be able to detect temporal changes in some areas, such as: the elimination of some objects, changes in object shape, and change in location.

For example, given two images of the same area, by analyzing shape similarity results, we will be able to detect the elimination of objects and changes in shape, and changes in topological/orientation relations could help detect changes in location. Statistical methods could be used for this analysis for predicting behaviors of dynamic objects. The graphic interface, proposed in Section 7.3.1, could be enhanced to provide a menu that assists the user in formulating his temporal queries in an intuitive way.

7.4.5 Querying Heterogeneous Data: Vector to Raster Considerations

In many application fields, users need to retrieve information from heterogeneous spatial databases, i.e. a database that contains various data types and formats. Although our interest is on raster formats for their inherent spatial accuracy, if we want to access

and manipulate the objects within a scene, a vector representation is also required. Therefore, both raster and vector descriptions of scenes and individual objects should be included in the database, and linked, to allow switching from one representation to the other depending on the required task. Textual information on these scenes and objects could be stored in the form of metadata and semantic properties.

The image query-by-sketch methodology proposed in this thesis could also be applied to querying for vector objects. To accomplish this, the vector query object and the various "map sheet" regions of the database would first be rasterized. The query can then be processed similar to the raster case.

Thus, our method could be extended to querying heterogeneous data. In general, query criteria could consist of one or more of the following: a sketch in raster or vector format of individual features (image-objects) or configurations of objects; metadata information on images or maps; and the semantic properties of objects.

The enhanced spatial query module will search and retrieve images and maps that match to user specified query criteria. A spatial query language will need to be developed to assist users in building their queries. The alphabet for this language could comprise the features/objects contained in the feature library as well as any interactive sketches the user might create. The constructs of the language will consist of configurations of objects built by the user. These configurations are built graphically and are characterized by the spatial relations between objects.

Bibliography

Ackerman, F., 1984. Digital Image Correlation: Performance and Potential Application in Photogrammetry. Photogrammetric Record, 11(64): 429-439.

Agouris, P., 1992. Multiple Image Multipoint Matching for Automatic Aerotriangulation, The Ohio State University, Columbus, Ohio.

Agouris, P., Carswell, J. and Stefanidis, A., 1999a. An Environment for Content-Based Image Retrieval from Large Spatial Databases. ISPRS Journal of Photogrammetry & Remote Sensing, 54: 263-272.

Agouris, P., Carswell, J. and Stefanidis, A., 1999b. Sketch-Based Image Queries in Topographic Databases. Journal of Visual Communication and Image Representation, 10: 1-16.

Agouris, P. and Schenk, T., 1996. Automated Aero-triangulation Using Multiple Image Multipoint Matching. *Photogrammetric Engineering & Remote Sensing*, 62(No. 6): 703-710.

Aho, A.V., Hopcroft, J.E. and Ullman, J.D., 1987. Data Structures and Algorithms. Addison Wesley Publishing Company.

Ardizzone, E. and La Cascia, M., 1997. Automatic video database indexing and retrieval. Multimedia Tools and Applications.

Athitsos, V., Swain, M. and Frankel, C., 1997. Distinguishing Photographs and Graphics on the World Wide Web, IEEE Workshop on Content-Based Access of Image and Video Libraries, Puerto Rico, pp. 10-17.

Bach, J.R. et al., 1996. The virage image search engine: An open framework for image management, SPIE - Symposium on Electronic Imaging: Science and Technology - Storage and Retrieval for Still Image and Video Database IV, pp. 76-87.

Baltsavias, E.P., 1991. Multiphoto Geometrically Constrained Matching. 49, ETH - Institute for Geodesy and Photogrammetry, Zurich.

Beis, J.S. and Lowe, D.G., 1997. Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces. IEEE trans. on Pami, 19(9).

Bentley, J.L., 1975. Multidimensional binary search trees used for associative searching. Communications of the ACM, 9(18): 509-517.

Blaser, A.D., 1998. Spatial-Query-by-Sketch: Fundamentals for a Sketch-Based User Interface in GIS, University of Maine, Orono, ME. USA.

Carson, C., Belongie, S., Greenspan, H. and Malik, J., 1997. Region-Based Image Querying, IEEE Workshop on Content-Based Access of Image and Video Libraries, San Juan, Puerto Rico, pp. 42-49.

Carswell, J., 1988. A Combined Digital Image Correlation Procedure for Establishing Relative Orientation. Technical Report, The Ohio State University, Columbus, Ohio, 55 pp.

Carswell, J. and Hasani, M., 1992. Transition to Digital Photogrammetry, EGIS, Munich, Germany.

Carswell, J. and Vanderlan, F., 1993. Automatic DTM Generation, International Symposium on Operationalization of Remote Sensing, ITC - Enschede, The Netherlands.

Chang, N.S. and Reuss, J., 1978. Design Considerations of a Pictorial Database System. Internationa Journal Policy Analysis Information Systems, 1: 49-70.

Chang, S.F., 1997. SaFe/VisualSEEk - Automatic Joint Spatial/Feature Based Image Search System. .

Chang, S.K., 1985. Image Information Systems. Proceedings of the IEEE, 73(4): 754-764.

Cohen, S.D. and Guibas, L.J., 1996. Shape-Based Indexing and Retrieval; Some First Steps. *Proceedings 1996 ARPA Image Understanding Workshop*, 2: 1209-1212.

Daoudi, M., Ghorbel, F., Mokadem, A. and Avaro, O., 1999. Shape Distances for Contour Tracking and Motion Estimation. to appear in Pattern Recognition.

Doorn, B., Agouris, P., Al-Tahir, R., Stefanidis, A. and Zilberstein, O., 1990. Digital Stereo Matching: In Perspective. 10, The Ohio State University, Columbus, Ohio.

Egenhofer, M. and Herring, J., 1990. A Mathematical Framework for the Definition of Topological Relationships. In: K.B.a.H. Kishimoto (Editor), Fourth International Symposium on Spatial Data Handling, Zurich, Switzerland, pp. 803-813.

Egenhofer, M. and Herring, J., 1991. Categorizing Binary Topological Relations Between Regions, Lines and Points in Geographic Databases, Department of Survey Engineering, University of Maine, Orono, ME.

FGDC, 1997. Content Standard for Digital Geospatial Metadata, Federal Geographic Data Committee, Washington, D.C.

Finkel, R.A. and Bentley, J.L., 1974. Quad trees: a data structure for retrieval on composite keys. Acta Informatica 4: 1-9.

Flickner, M. et al., 1995. Query by Image and Video Content: The QBIC System. IEEE Computer, 28(9): pp. 23-32.

Forstner, W., 1986. Digital Imatge Matching Techniques for Standard Photogrammetric Application, ACSM-ASPRS Annual Convention.

Forsyth, D.A. et al., 1996. Finding pictures of objects in larges collections of images, ECCV 96 Workshop on Object Representation.

Frankel, C., Swain, M. and Athitsos, W., 1996. WebSeer: An image search engine for the world wide web. TR-96-14, Department of Computer Science, University of Chicago.

Gadi, T., Benslimane, R., Daoudi, M. and Matusiak, S., 1999. Fuzzy Similarity Measure for Shape Retrieval, Vision Interface '99, Trois Rivieres, Canada, pp. 386-389.

Gonzalez, R.C. and Woods, R.E., 1992. Digital Imge Processing. Addison-Wesley Publishing Company, Inc., 716 pp.

Greenfeld, J.S., 1987. A Stereo Vision Approach to Automatic Stereo Martching in Photogrammetry. Ph.D. Dissertation, The Ohio State University, Columbus, Ohio.

Greenfeld, J.S. and Schenk, A.T., 1989. Experiments with Edge-Based Stereo Matching. Photogrammetric Engineering and Remote Sensing, 55(12): 1771-1777.

Grimson, W.E.L., 1985. Computational Experiments with a Feature-Based Stereo Algorithm. IEEE PAMI, 7(1): 17-34.

Gruen, A., Agouris, P. and Li, H., 1995. Linear Feature Extraction with Dynamic Programming and Globally Enforced Least Squares Matching. In: A. Gruen, K. O. and A. P. (Editors), *Automatic Extraction of Man-Made Objects from Aerial and Space Images*. Birkhaeuser Verlag, pp. 83-94.

Gruen, A.W. and Baltsavias, E.P., 1987. Geometrically Constrained Multiphoto Matching. Photogrammetric Engineering and Remote Sensing, 54(5): 633-641.

Gudivada, V.N. and Raghavan, V.V., 1995a. Content-Based Image Retrieval Systems. *IEEE Computer*: 18-22.

Gudivada, V.N. and Raghavan, V.V., 1995b. Design and Evaluation of Algorithms for Image Retrieval by Spatial Similarity. *ACM Transactions on Information Systems,* 13(2): 115-144.

Gupta, A., Weymouth, T. and Jain, R., 1991. Semantic Queries With Pictures: The VIMSYS Model, Seventeenth VLDB, Barcelona, Spain.

Güting, R.H., 1994. An Introduction to Spatial Database Systems. VLDB Journal, 3: 357-399.

Jagadish, H.V., 1991. A Retrieval Technique for Similar Shapes, ACM SIGMOD International Conference on Management of Data, pp. 208-217.

Kauppinen, H., Seppnaen, T. and Pietikaainen, M., 1995. An experimental comparison of auto-regressive and Fourier Descriptors in 2D shape classification. IEEE PAMI, 17(2): 201-207.

Kelly, P.M., Cannon, T.M. and Hush, D.R., 1995. Query by image example: the CANDID approach. SPIE Storage and Retrieval for Image and Video Databases III, Vol. 2420: pp.238-248.

Lindeberg, T., 1994. *Scale-Space Theory in Computer Vision*. Kluwer Academic Publishers Boston.

Maas, H.-G., Stefanidis, A. and Gruen, A., 1994. Feature Tracking in 3-D Fluid Tomography Sequences, *ICIP-94*, pp. 530-534.

Mehrotra, R. and Gray, J., 1993. Feature-Based Retrieval of Similar Shapes, Ninth International Conference on Data Engineering, Vienna, Austria, pp. 108-115.

Mehrotra, R. and Gray, J., 1995. Similar-Shape Retrieval in Shape Data Management. IEEE Computer, 28(9): 57-62.

Nishida, H., 1999. Shape Retrieval from Image Databases through Structural Feature Indexing, Vision Interface '99, Trois Rivieres,Canada, pp. 328-335.

Ogle, V.E., 1995. Chabot: Retrieval from a Relational Database of Images. IEEE Computer(September): 23-32.

Pentland, A., Picard, R.W. and Scarloff, S., 1996. Photobook: Content-based manipulation of image databases. International Journal of Computer Vision.

Persoon, E. and Fu, K.S., 1977. Shape discrimination using Fourier Descriptors. IEEE Trans. on SMC, 7(3): 170-179.

Pickard, R.W. and Minka, T.P., 1995. Vision Texture for Annotation. *Multimedia Systems*, 3(1): 3-14.

Ravela, S. and Manmatha, R., 1997. Retrieving Images by Similarity of Visual Appearance, IEEE Workshop on Content-Based Access of Image and Video Libraries, San Juan, Puerto Rico, pp. 67-74.

Samet, H., 1990. Application of Spatial Data Structures. Addison-Wesley Series in Computer Science.

Sclaroff, S., Taycher, L. and La Cascia, M., 1997. ImageRover: A Content-Based Image Browser for the World Wide Web, IEEE Workshop on Content-Based Access of Image and Video Libraries, San Juan, Peurto Rico, pp. 2-9.

Smith, J.R. and Chang, S.-F., 1996. VisualSEEk: a fully automated content-based image query system. ACM Multimedia '96, November.

Srihari, R.K., 1995. Automatic Indexing and Content-Based Retrieval of Captioned Images. IEEE Computer(September): 49-56.

Stonebraker, M., 1990. The Implementation of Postgres. IEEE Trans. Knowledge and Data Engineering, March.

Zloof, M.M., 1975. Query by Example, AFIPS 1975 NCC, pp. 431-438.

Appendix

Feature House Cleaner

```
//
//    feature_house_cleaner - a recursive cprogram to order a TFI
//    (a square matrix of matching percentages between features)
//    into a structured feature library with all unnecessary
//    duplicates removed.
//

#include "cips.h"

typedef char string30[31];

typedef struct{ string30 fn;
                        short level;
                        string30 path[11];
                        short numkids;
                        string30 kidnames[101];}FEATURE;


void  FINDKIDS(FEATURE  features[101],short  index,short  *totalkids,short
kids[101],short totalfeatures){

      short i,j,k;



            for(i=1;i<=features[index].numkids;i++){

                  for(j=1;j<=totalfeatures;j++){

      if(strcmp(features[index].kidnames[i],features[j].fn)==0){
                              *totalkids=*totalkids+1;
                              kids[*totalkids]=j;
                              j=totalfeatures+1;
                        }
                  }

                  for(k=1;k<=features[kids[*totalkids]].numkids;k++)

      FINDKIDS(features,kids[*totalkids],totalkids,kids,totalfeatures);


            }



}


void FINDMAX(FEATURE features[101], short index, float tfi[101][101],
                  short newfeature, float similar, float same,
                  float *max, short *position,short *found){

      short i,j,h,p,index2;
      float max2;
```

```
	if(tfi[newfeature][index]>=same){
		*found=1;
		*max=tfi[newfeature][index];
		*position=index;
	}


	else if(tfi[newfeature][index]>=similar || index==0){
		if(tfi[newfeature][index]>=*max){
			max2=0; h=0; index2=0; *found=1;
			for (p=1;p<=features[index].numkids;p++){

					for(j=1;j<newfeature;j++){

	if(strcmp(features[j].fn,features[index].kidnames[p])==0){
							h=j;
							j=newfeature;
						}
					}

					if     (tfi[newfeature][h]>=similar     &&
tfi[newfeature][h]>= max2){
						max2=tfi[newfeature][h];
						index2=h;
					}

			}
			if (max2==0){
				*max=tfi[newfeature][index];
				*position=index;
			}
			else {
				*max=max2;
				*position=index2;
			}
		}
		for(i=1;i<=features[index].numkids;i++){
				for(j=1;j<newfeature;j++){

	if(strcmp(features[j].fn,features[index].kidnames[i])==0){


	FINDMAX(features,j,tfi,newfeature,similar,same,max,position,found)
;

					}
				}
		}

	}
	else //case of different
	{
		if (tfi[newfeature][index]>=*max){
			*max=tfi[newfeature][index];
			*position=index;
			*found=1;
		}
	}

}
```

```
void FINDMAX2(FEATURE features[101], short index, float tfi[101][101],
                  short newfeature, float similar, float same,
                  float   *max,   short   *position,short   *found,   short
totalfeatures){

      short i,j,h,p,index2;
      float max2;



      if(tfi[newfeature][index]>=same){
            *found=1;
            *max=tfi[newfeature][index];
            *position=index;
      }


      else if(tfi[newfeature][index]>=similar || index==0){
            if(tfi[newfeature][index]>=*max){
                  max2=0; h=0; index2=0; *found=1;
                  for (p=1;p<=features[index].numkids;p++){

      if(strcmp(features[index].kidnames[p],features[newfeature].fn)!=0)
{

      //for(j=newfeature+1;j<=totalfeatures;j++){
                              for(j=1;j<=totalfeatures;j++){

      if(strcmp(features[j].fn,features[index].kidnames[p])==0){
                                          h=j;
                                          j=totalfeatures+1;
                                    }
                              }

                              if     (tfi[newfeature][h]>=similar     &&
tfi[newfeature][h]>= max2){
                                    max2=tfi[newfeature][h];
                                    index2=h;
                              }
                        }
                  }
                  if (max2==0){
                        *max=tfi[newfeature][index];
                        *position=index;
                  }
                  else {
                        *max=max2;
                        *position=index2;
                  }
            }
            for(i=1;i<=features[index].numkids;i++){

      if(strcmp(features[index].kidnames[i],features[newfeature].fn)!=0)
{
                        //for(j=newfeature+1;j<=totalfeatures;j++){
                        for(j=1;j<=totalfeatures;j++){

      if(strcmp(features[j].fn,features[index].kidnames[i])==0){


      FINDMAX2(features,j,tfi,newfeature,similar,same,max,position,found
,totalfeatures);
                                    j=totalfeatures+1;
                              }
```

```
                        }
                  }
            }

      }
      else //case of different
      {
            if (tfi[newfeature][index]>=*max){
                  *max=tfi[newfeature][index];
                  *position=index;
                  *found=1;
            }
      }

}


int INSERT(FILE *outFilep, FEATURE features[101], short index, float
tfi[101][101],
                  short totalfeatures, short newfeature, float same,
float similar,
                  short *found){

      short i,p,m,position;
      float max;

      char  parent[31];


                  position=0;
                  max=0;
                  index=0;

      for(i=1;i<=features[index].numkids;i++){



      FINDMAX(features,index,tfi,newfeature,similar,same,&max,&position,
found);

            if (max>=same){
                  features[newfeature].level=0;
                  return 0;
            }
            else if (max>=similar){




                  features[newfeature].level=features[position].level+1;

      features[position].numkids=features[position].numkids+1;

      strncpy(features[position].kidnames[features[position].numkids],fe
atures[newfeature].fn,31);
                  for(p=1;p<features[position].level;p++){

      strncpy(features[newfeature].path[p],features[position].path[p],31
);
                  }

      strncpy(features[newfeature].path[p],features[position].fn,31);
```

```
                  return 0;
            }

            else{
                  features[newfeature].level=features[position].level;
                  if(features[newfeature].level==1){
                        strncpy(parent,"ROOT",31);
                        strncpy(features[newfeature].path[1],parent,31);
                  }
                  else{

      strncpy(parent,features[position].path[features[position].level-
1],31);
                  }
                  for(m=0;m<=totalfeatures;m++){
                        if(strcmp(features[m].fn,parent)==0){
                              features[m].numkids=features[m].numkids+1;

      strncpy(features[m].kidnames[features[m].numkids],features[newfeat
ure].fn,31);
                              m=totalfeatures+1;
                        }
                  }
                  for(p=1;p<features[position].level;p++){

      strncpy(features[newfeature].path[p],features[position].path[p],31
);
                  //    if (newfeature==5 && p==1)
                     //
      printf("%15s",features[newfeature].path[p]);

                  }

                  return 0;

            }
      }
}


int INSERTKIDS(FILE *outFilep, FEATURE features[101], short index, float
tfi[101][101],
                  short  totalfeatures,  short  newfeature,  float  same,
float similar,short *found){

      short i,p,m,position;
      float max;

      char  parent[31];


                  position=0;
                  max=0;
                  index=0;

      for(i=1;i<=features[index].numkids;i++){



      FINDMAX2(features,index,tfi,newfeature,similar,same,&max,&position
,found,totalfeatures);

            if (max>=same){
```

```
                features[newfeature].level=0;


                return 0;
          }
          else if (max>=similar){

                features[newfeature].level=features[position].level+1;

      features[position].numkids=features[position].numkids+1;

      strncpy(features[position].kidnames[features[position].numkids],fe
atures[newfeature].fn,31);


                for(p=1;p<features[position].level;p++)

      strncpy(features[newfeature].path[p],features[position].path[p],31
);


      strncpy(features[newfeature].path[p],features[position].fn,31);



                return 0;
          }

          else{


                features[newfeature].level=features[position].level;
                if(features[newfeature].level==1){
                      strncpy(parent,"ROOT",31);
                      strncpy(features[newfeature].path[1],parent,31);
                }
                else{

      strncpy(parent,features[position].path[features[position].level-
1],31);
                }

                for(m=0;m<=totalfeatures;m++){
                      if(strcmp(features[m].fn,parent)==0){
                            features[m].numkids=features[m].numkids+1;

      strncpy(features[m].kidnames[features[m].numkids],features[newfeat
ure].fn,31);
                            m=totalfeatures+1;
                      }
                }

                for(p=1;p<features[position].level;p++)

      strncpy(features[newfeature].path[p],features[position].path[p],31
);
```

```
                return 0;

            }
      }
}




int INSERT2(FILE *outFilep, FEATURE features[101], short index, float
tfi[101][101],
                short totalfeatures, short newfeature, float same,
float similar,
                short *found){

      short i,j,jj,p,m,position,kidposition,
                kids[101],totalkids=0;

      float max;

      char  parent[31];


                kidposition=0;
                position=0;
                max=0;
                index=0;

      for(i=1;i<=100;i++)
            kids[i]=0;


      for(i=1;i<=features[index].numkids;i++){



      FINDMAX2(features,index,tfi,newfeature,similar,same,&max,&position
,found,totalfeatures);

            if (max>=same){

                  features[newfeature].level=0;


      FINDKIDS(features,newfeature,&totalkids,kids,totalfeatures);


                  for(j=1;j<=features[newfeature].numkids;j++)
                        *features[newfeature].kidnames[j]='\0';

                  features[newfeature].numkids=0;

                  for(jj=1;jj<=totalkids;jj++){
                        kidposition=kids[jj];

                        found=0;

                        for(j=1;j<=10;j++)
                              *features[kidposition].path[j]='\0';

                        for(j=1;j<=features[kidposition].numkids;j++)
```

```
                              *features[kidposition].kidnames[j]='\0';

                        features[kidposition].numkids=0;


      INSERTKIDS(outFilep,features,index,tfi,totalfeatures,kidposition,s
ame,similar,&found);
                  }



                  totalkids=0;
                  for(j=1;j<=100;j++)
                        kids[j]=0;

                  return 0;
            }

            else if (max>=similar){

                  features[newfeature].level=features[position].level+1;

      features[position].numkids=features[position].numkids+1;

      strncpy(features[position].kidnames[features[position].numkids],fe
atures[newfeature].fn,31);
                  for(p=1;p<features[position].level;p++){

      strncpy(features[newfeature].path[p],features[position].path[p],31
);
                  }

      strncpy(features[newfeature].path[p],features[position].fn,31);



      FINDKIDS(features,newfeature,&totalkids,kids,totalfeatures);


                  for(j=1;j<=features[newfeature].numkids;j++)
                        *features[newfeature].kidnames[j]='\0';

                  features[newfeature].numkids=0;

                  for(jj=1;jj<=totalkids;jj++){
                        kidposition=kids[jj];

                        found=0;

                        for(j=1;j<=10;j++)
                              *features[kidposition].path[j]='\0';

                        for(j=1;j<=features[kidposition].numkids;j++)
                              *features[kidposition].kidnames[j]='\0';

                        features[kidposition].numkids=0;


      INSERTKIDS(outFilep,features,index,tfi,totalfeatures,kidposition,s
ame,similar,&found);
                  }

                  totalkids=0;
                  for(j=1;j<=100;j++)
```

```
                        kids[j]=0;

                  return 0;

            }
            else{

                  features[newfeature].level=features[position].level;
                  if(features[newfeature].level==1){
                        strncpy(parent,"ROOT",31);
                        strncpy(features[newfeature].path[1],parent,31);
                  }
                  else{

      strncpy(parent,features[position].path[features[position].level-
1],31);
                  }
                  for(m=0;m<=totalfeatures;m++){
                        if(strcmp(features[m].fn,parent)==0){
                              features[m].numkids=features[m].numkids+1;

      strncpy(features[m].kidnames[features[m].numkids],features[newfeat
ure].fn,31);
                              m=totalfeatures+1;
                        }
                  }
                  for(p=1;p<features[position].level;p++){

      strncpy(features[newfeature].path[p],features[position].path[p],31
);
                  //    if (newfeature==5 && p==1)
                     //
      printf("%15s",features[newfeature].path[p]);

                  }


      FINDKIDS(features,newfeature,&totalkids,kids,totalfeatures);




                  for(j=1;j<=features[newfeature].numkids;j++)
                              *features[newfeature].kidnames[j]='\0';

                  features[newfeature].numkids=0;

                  for(jj=1;jj<=totalkids;jj++){
                        kidposition=kids[jj];

                        found=0;

                        for(j=1;j<=10;j++)
                              *features[kidposition].path[j]='\0';

                        for(j=1;j<=features[kidposition].numkids;j++)
                              *features[kidposition].kidnames[j]='\0';

                        features[kidposition].numkids=0;


      INSERTKIDS(outFilep,features,index,tfi,totalfeatures,kidposition,s
ame,similar,&found);
                  }
```

```
                totalkids=0;
                for(j=1;j<=100;j++)
                      kids[j]=0;

                return 0;
          }
      }

}


//
//beginning of main program
//

int main(){


      FEATURE features[101];

      char  junk[31],parent[31],tfiname[80],result1[80],result2[80];


      short numfiles=0, found=0,index=0,
                i,j,status,k,p,parentposition,
                totalfeatures;// total number of features in feature
library

      short
      zero=0,one=0,two=0,three=0,four=0,five=0,six=0,seven=0,eight=0,nin
e=0,ten=0;

      float same,
                similar,
                tfi[101][101];


      FILE *inFilep,*outFilep, *library;


      ////////////////////////////////////


inptfi:
      printf("\n\nEnter file name of tfi");//i.e. tfi66
      printf("  --> ");
      gets(tfiname);
      strncpy(result1,"d:\\tfi\\",80);
      strncat(result1,tfiname,80);
      strncat(result1,".dat",80);

      if (strcmp(tfiname,"")==0) goto inptfi;

      //inFilep=fopen(tfiname,"rb");

      //strncpy(result1,"d:\\tfi\\tfi66.dat",80);
      inFilep=fopen(result1,"r");



      if (!(inFilep))   {
```

```
            printf("\nInput    file    open    error:%s    does    not
exist\n\n",tfiname);
            goto inptfi;
      }

inpt2:
      printf("\n\nEnter percentage for SAME (between 0 and 100 percent)-
->");//i.e. 80

      scanf("%f",&same);
      if(same>100 || same<0){
            printf("\nInvalid  entry,  SAME  must  be  between  0  and  100
percent\n\n");
            goto inpt2;
      }


inpt3:
      printf("\n\nEnter percentage for SIMILAR (must be less than SAME)-
->");//i.e. 80

      scanf("%f",&similar);
      if(similar>=same || similar<0){
            printf("\nInvalid   entry,   SIMILAR   must   be   less   than
SAME\n\n");
            goto inpt3;
      }

      printf("\n  press enter to start timer--> ");
      system("time >> d:\\tfi\\time.dat");

      for(i=1;i<=100;i++){
            for(j=1;j<=100;j++)
                  tfi[i][j]=0.;
      }

      for(i=0;i<=100;i++){
            features[i].level=0;
            features[i].numkids=0;

            for(j=0;j<=10;j++)
                  *features[i].path[j]='\0';
            for(k=0;k<=100;k++){
                  *features[i].kidnames[k]='\0';
            }
      }



      outFilep=fopen("d:\\tfi\\featurelibrary.dat","w");

      if (!(outFilep))
      {
            printf("Output file open error\n\n");
            return 1;
      }


      //inFilep=fopen("d:\\tfi\\tfi66.dat","r");
      //inFilep=fopen("d:\\tfi\\test2.dat","r");

      i=1;
      status=fscanf(inFilep,"%s",features[i].fn);
```

```
	while(status != '\n')
		status=getc(inFilep);

	while(status!=EOF){
		i++;
		status=fscanf(inFilep,"%s",features[i].fn);
		while(status != '\n' && status!=EOF)
			status=getc(inFilep);
	}
	totalfeatures=i;

	fclose(inFilep);

	//inFilep=fopen("d:\\tfi\\tfi66.dat","r");
	//inFilep=fopen("d:\\tfi\\test2.dat","r");

	inFilep=fopen(result1,"r");

	for(i=1;i<=totalfeatures;i++){
		status=fscanf(inFilep,"%s",junk);

		for(j=1;j<=totalfeatures;j++)
			status=fscanf(inFilep,"%f",&tfi[i][j]);

			while(status != '\n' && status!=EOF)
			status=getc(inFilep);

	}

	fclose(inFilep);

	for(i=1;i<=totalfeatures;i++){
		fprintf(outFilep,"\n%15s",features[i].fn);
		for(j=1;j<=totalfeatures;j++)
			fprintf(outFilep,"%10.1f",tfi[i][j]);
	}

	//here is where the insertion starts

	features[1].level=1;
	strncpy(features[index].fn,"ROOT",31);
	strncpy(features[1].path[1],features[index].fn,31);
	strncpy(features[index].kidnames[1],features[1].fn,31);
	features[index].numkids=1;

	for(i=2;i<=totalfeatures;i++){
		found=0;

	INSERT(outFilep,features,index,tfi,totalfeatures,i,same,similar,&f
ound);
	}

//printout feature library before housecleaning

	sprintf(result2,"%.1f",same);
	strncpy(result1,tfiname,80);
	strncat(result1,"_",80);
	strncat(result1,result2,80);
	sprintf(result2,"%.1f",similar,80);
	strncat(result1,"_",80);
	strncat(result1,result2,80);
	strncat(result1,"_before",80);
	strncat(result1,".dat",80);
	strncpy(result2,"d:\\featurelibrary\\",80);
```

```
	strncat(result2,result1,80);


	library=fopen(result2,"w");
	for(i=1;i<=totalfeatures;i++){
		//if(features[i].level!=0){
			fprintf(library,"\n%15s                                    %3d
%3d",features[i].fn,features[i].level,features[i].numkids);
			for(j=1;j<features[i].level;j++)
				fprintf(library,"%15s",features[i].path[j]);
		//}
	}


	fclose(library);

/*
	for(i=0;i<=100;i++){
		for(j=0;j<=10;j++)
			*features[i].path[j]='\0';
	}
*/


	//housecleaning begins here


	for(i=1;i<=totalfeatures;i++){
		found=0;
		if(features[i].level>0){

			if(features[i].level==1){
					strncpy(parent,"ROOT",31);

			}
			else{

	strncpy(parent,features[i].path[features[i].level-1],31);
			}

			for(p=0;p<=totalfeatures;p++){
				if(strcmp(parent,features[p].fn)==0){
					parentposition=p;
					p=totalfeatures+1;
				}
			}

			for(j=1;j<=features[parentposition].numkids;j++){

	if(strcmp(features[parentposition].kidnames[j],features[i].fn)==0)
{

	*features[parentposition].kidnames[j]='\0';
					k=j;

				}
			}

			for(j=1;j<=10;j++)
				*features[i].path[j]='\0';


			for(j=1;j<features[parentposition].numkids;j++){
```

```
      if(strcmp(features[parentposition].kidnames[j],"")==0){

      strncpy(features[parentposition].kidnames[j],features[parentpositi
on].kidnames[j+1],31);
                              k=j+1;
                              j=features[parentposition].numkids;
                        }
                  }
                  if(features[parentposition].numkids>1){

      for(j=k;j<features[parentposition].numkids;j++)

      strncpy(features[parentposition].kidnames[j],features[parentpositi
on].kidnames[j+1],31);
                  }

                  *features[parentposition].kidnames[j]='\0';


                  if(features[parentposition].numkids>0)

      features[parentposition].numkids=features[parentposition].numkids-
1;



      INSERT2(outFilep,features,index,tfi,totalfeatures,i,same,similar,&
found);
            }
      }


//printout feature library after housecleaning

      sprintf(result2,"%.1f",same);
      strncpy(result1,tfiname,80);
      strncat(result1,"_",80);
      strncat(result1,result2,80);
      sprintf(result2,"%.1f",similar,80);
      strncat(result1,"_",80);
      strncat(result1,result2,80);
      strncat(result1,"_after",80);
      strncat(result1,".dat",80);
      strncpy(result2,"d:\\featurelibrary\\",80);
      strncat(result2,result1,80);

      library=fopen(result2,"w");
      for(i=1;i<=totalfeatures;i++){
            //if(features[i].level!=0){
            if(features[i].level==0)
                  zero=zero+1;
            if(features[i].level==1)
                  one=one+1;
            if(features[i].level==2)
                  two=two+1;
            if(features[i].level==3)
                  three=three+1;
            if(features[i].level==4)
                  four=four+1;
            if(features[i].level==5)
                  five=five+1;
            if(features[i].level==6)
                  six=six+1;
```

```
            if(features[i].level==7)
                  seven=seven+1;
            if(features[i].level==8)
                  eight=eight+1;
            if(features[i].level==9)
                  nine=nine+1;
            if(features[i].level==10)
                  ten=ten+1;

                  fprintf(library,"\n%15s                                %3d
%3d",features[i].fn,features[i].level,features[i].numkids);
                  for(j=1;j<features[i].level;j++)
                        fprintf(library,"%15s",features[i].path[j]);
            //}
      }

      fprintf(library,"\n\nzero=%d\none=%d\ntwo=%d\nthree=%d\nfour=%d\nf
ive=%d\nsix=%d\nseven=%d\neight=%d\nnine=%d\nten=%d",

      zero,one,two,three,four,five,six,seven,eight,nine,ten);

fclose(library);

      fclose(outFilep);
      printf("\n\nALL DONE\n\n  press enter to stop timer-->\n\n");
      //system("time >> d:\\tfi\\time.dat");

      return 0;
}              /* ends main                */
```

Biography of the Author

James Duncan Carswell was born in Renfrew, Ontario, Canada on March 15, 1962. He began his formal schooling in Sault Ste. Marie and from there moved on to Pembroke, Kingston, and back to Pembroke to finish his final years of secondary schooling at Fellowes High School, graduating in June 1981. During this time, his was a familiar presence amongst all the major varsity sports teams including track & field, football, basketball, volleyball, and soccer; the last three of which he captained to multiple conference championships.

He began his post secondary schooling at Ryerson Polytechnical Institute in Toronto, Ontario, earning a Bachelor of Technology Degree in Survey Engineering in 1986. Deciding to further strengthen his academic credentials, he obtained a Master of Science in Geodetic Science from The Ohio State University in 1988. While attending OSU, he held both teaching and research assistantship positions on a NASA sponsored digital photogrammetry project.

After graduation, he began his first "real" job as a GIS Analyst with Unisys' European Center for GIS in Amsterdam, The Netherlands. He remained there for three years presenting, installing and customizing their Oracle based GIS to suit customer requirements throughout Europe. For the next five years, he was the Digital Photogrammetry Product Marketing Manager within the European Headquarters of Intergraph, also located in The Netherlands. It was through both these professional appointments that allowed him to work with the major mapping institutions of Europe,

which encouraged him to return to school in pursuit of a doctoral degree in the mapping sciences.

In 1996, Mr. Carswell left Europe to accept a graduate research assistantship position with the Department of Spatial Information Science and Engineering at The University of Maine. He has authored or co-authored numerous research papers published in fully refereed journals and conference proceedings and was awarded the Altenhofen Memorial Scholarship for photogrammetry in 1998 by the American Society of Photogrammetry and Remote Sensing

He is a candidate for the Doctor of Philosophy degree in Spatial Information Science and Engineering from The University of Maine in May, 2000.