Combinatorial Techniques for Digital Image Characterization and Retrieval: Algorithms, Architectures, and Applications

Doctoral dissertation submitted by

Arijit Bishnu

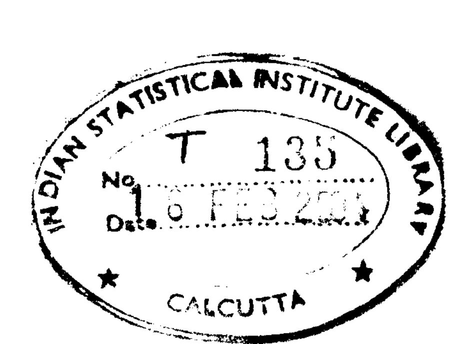
for award of the Ph.D. degree of the
Indian Statistical Institute, Kolkata

Advisor :

Professor Bhargab B. Bhattacharya

Indian Statistical Institute
Kolkata 700 108

May 2002



Acknowledgment

It is time to pause and reflect back at the end of those years during which I had been working on my thesis. Time it is to thank the people I had been working with.

First and foremost, I would thank my supervisor Professor Bhargab Bikram Bhattacharya. It was a pleasure working with him. I am indebted to him for his help during a critical phase of my student life.

I would keep fond memories of the Advanced Computing and Microelectronics Unit, where I have worked towards my thesis. The atmosphere in the department was always cordial. I could walk up to any one with my problems, and people were more than willing to help. I take this opportunity to thank them - one and all - for what they have been to me.

I do recall the help and words of encouragement Dr. Palash Sarkar had for me.

There have been many people with whom I worked with during the preparation of my thesis. Of special mention are Piyush Kanti Bhunre, Arindam Biswas and Jayanta Dey. I would like to acknowledge here Partha Bhowmick with whom I have shared a good academic and a personal relation as well.

Lastly, about Dr. Subhas C. Nandy and Dr. Sandip Das, whose relationships, I have cherished the most during this period. We three enjoyed good times, had lots of fun. Hope our relationship continues with the same fervor.

(Arijit Bishnu)

May 24, 2002

Advanced Computing and Microelectronics Unit

Indian Statistical Institute

203 B. T. Road, Kolkata - 700 108, India

e-mail: bishnu_t@isical.ac.in

Contents

1 Introduction 1

2 Euler number 5

2.1 Introduction 5

2.2 Proposed Sequential Algorithm 8

2.2.1 Theme 8

2.2.2 Space and time complexity 11

2.2.3 Results 13

2.3 Parallel Algorithm and VLSI Implementation 14

2.3.1 VLSI architecture 15

2.3.2 Truth Table and Combinational Circuit 17

2.3.3 Adder Design and Time Complexity 18

2.3.4 Circuit cost 20

2.3.5 Handling large images 20

2.4 Conclusions 22

3 Euler vector 29

3.1 Introduction 29
3.1.1 Geometric Features 29
3.1.2 Luminance Features 30
3.2 Euler Vector Computation 32
3.2.1 Bit-planes and Euler vector 32
3.2.2 Implementation details 35
3.3 Mahalanobis distance as a measure of similarity 37
3.3.1 Theme 38
3.3.2 Application of Mahalanobis distance to image ranking 39
3.4 Results 39
3.4.1 Euler Vector values 40
3.4.2 Mahalanobis Distance and Ranking 40
3.4.3 K-Nearest Neighbor Classifier 42
3.5 VLSI implementation 44
3.6 Conclusions 45

4 Image Retrieval using Euler Vector **52**
4.1 Introduction 52
4.2 Review of Multidimensional Access Methods 55
4.3 A Brief Review of Kd Tree 58
4.3.1 Algorithm for Kd Tree Construction 58
4.3.2 Query in a Kd Tree 60
4.4 Proposed Modification to Kd Tree 62
4.4.1 Split Operations for Partitioning 63

4.4.2 Merge Operations on the Partitions . . . 65
4.5 Experimental Results . . . 68
4.6 Conclusions . . . 71

5 Fingerprint Processing and Related Architecture **78**

5.1 Introduction . . . 78
5.2 Mathemetical Preliminaries . . . 83
5.3 Proposed Sequential Algorithm . . . 86
5.3.1 Theme . . . 86
5.3.2 Implementation Details . . . 90
5.4 Classification of a pixel . . . 93
5.4.1 Preliminary classification . . . 93
5.4.2 Final classification . . . 95
5.4.3 Thinning . . . 96
5.4.4 Algorithm . . . 97
5.5 Space and Time Complexity . . . 99
5.6 Evaluation and Results . . . 100
5.6.1 Evaluation Criteria for Ridge/Valley finding Algorithms . . . 100
5.6.2 Results . . . 100
5.7 Parallel Algorithm and VLSI Architecture . . . 102
5.7.1 Parallel Algorithm . . . 102
5.7.2 VLSI Architecture . . . 102
5.7.3 Time Complexity . . . 108
5.7.4 Circuit Cost . . . 108

5.8 Conclusions and Discussions 109

6 Point Set Pattern Matching 117

6.1 Introduction 117

6.2 An Existing Algorithm 119

6.2.1 Circular Sorting 119

6.2.2 Point Set Pattern Matching on a 2-D Plane 121

6.2.3 Space and Time Complexity 123

6.3 Proposed Algorithm 124

6.3.1 Preprocessing 125

6.3.2 Query 127

6.4 Experimental Results and Applications 130

6.4.1 Experiments on Randomly Generated Point Sets 130

6.4.2 Experiment with Real Fingerprint minutiae 131

7 Conclusions 133

Chapter 1

Introduction

Interest in digital images stems mostly from its application to various areas of computer vision [33, 57] and pattern recognition [145]. Problems include robotic vision and control, geographic and topographic map matching, target recognition, space applications, character recognition, scene analysis, fingerprint and face recognition, etc. Lately, with the advent of content-based image retrieval (CBIR) and proliferation of the Internet, digital imaging applications are in vogue now than ever before. In almost all the cases, the data size is enormously large, and at the same time, fast on-line as well as real-time computation is needed. For example, in fingerprint processing, each image frame is of size 480×512 with 256 gray levels; 256 gray levels requiring 8 bits. Thus, the total image size is 245760 bytes $\approx$ 0.25MB. Multimedia and video applications require frame rates of 50-80 frames per second. The throughput requirements of storage and processing for these applications can run into giga operations per second (*gops*) [116]. However, most of the operations are locally repetitive and modular, with a high degree of parallelism among these tasks. Parallel processing and VLSI technology provide a suitable paradigm for designing high performance hardware, systems-on-chip for real time solutions to such applications. VLSI technology has made rapid strides leading to efficient implementation of chips. This creates the scope for designing new architectures for application specific needs giving rise to application specific integrated circuits (ASIC).

The essence of developing specific architectures for image processing applications is to exploit the special forms of parallelism inherent in those algorithms. Most of the architectures used for computer vision and imaging applications can be classified into Single Instruction Multiple Data (SIMD) and Multiple Instruction Multiple Data (MIMD) architectures. Although SIMD and MIMD architectures are powerful, still another form of parallelism in image data can be found in the local neighborhood around each pixel. The neighborhood pixels can be processed simultaneously for local overlapped operations to exploit a temporal parallelism. Pipelining can be employed to perform such operations. Based on two types of parallelisms, temporal and spatial, parallel computers can be classified into *pipeline computers* and *array processors*.

For on-chip design of image processing hardware, the concerned algorithm should be tailored for a VLSI architecture. Many such earlier works can be found in [2, 26, 32, 34, 48, 60, 61, 72, 79, 81, 87, 99, 107], [114]-[117], [132, 133].

Apart from architectural aspects of digital imaging, many tasks in computer vision and CBIR require new and efficient algorithms for faster operation. Techniques borrowed from the inter-disciplinary areas like computational geometry, multidimensional data structures and database can be fused together to provide good solutions to these problems. CBIR involves indexing and searching in higher-dimensional feature spaces. Images are indexed using their features. Determination of compact, topologically invariant, easily computable features for image characterization is a fundamental problem. In this thesis, we have addressed some of the pertinent issues on combinatorial techniques for digital image characterization, and devoloped new algorithms and VLSI architectures for on-chip implementation. We further demonstrate applications of these techniques to image searching, CBIR, fingerprint analysis and matching.

In Chapter 2, we report new results on Euler number and an architecture for VLSI implementation. Euler number is a well known combinatorial feature and is defined as the difference of the number of connected components (objects), and the number of holes in a binary image. This is an important topological feature that remains invariant under arbitrary rubber-sheet transformations. Euler number has numerous applications in image processing, e.g., medical diagnosis, optical character recognition, document image processing, reflectance-based object recognition, analysis of sandstone for geological

applications, shadow detection etc. We revisit an existing run-based method of Euler number computation from a new angle. Certain combinatorial properties of 0-1 runs in the image matrix are used to compute the Euler number and they form the basis of the proposed on-chip design. Based on several underlying properties of this technique, an adder tree using a pipeline architecture is designed that can easily be implemented as a VLSI chip. The same circuit can be used to handle arbitrarily sized pixel matrices with minor modifications.

In Chapter 3, for a gray-tone image, we define a new feature resembling Euler number called Euler vector. We consider gray code representation of the intensity values in the pixel matrix, and observe the first 4 most significant bit planes. Each bit plane consists of only 0's and 1's and hence forms a binary (two-tone) image. We compute the Euler number of the partial image formed by each of the 4 bit planes to obtain a 4-tuple of integers, called Euler vector of the original gray-tone image. Euler vector thus, as a feature of a gray-tone image, uses both geometric and intensity level characterization. Empirical evidence shows that the Euler vector remains almost invariant under inclusion of salt and pepper or gaussian noise followed by filtering, and also under JPEG compression. On-chip implementation of the hardware for computing the Euler vector is also straightforward. We believe Euler vector would turn out to be a potential tool with numerous applications.

We demonstrate that Euler vector can be used in CBIR as it provides a robust feature for indexing and retrieval. *In Chapter 4*, we report the results of our investigation on multidimensional access methods for retrieval. For feature-based similarity searching, an image is mapped as a point in a multidimensional feature space that is indexed using some multidimensional data structure. We use a modified K-d-tree to support efficient clustering of data points into buckets for indexing.

As an application specific problem, we further extend the combinatorial idea of connectedness to a gray-tone image and develop a pixel classification algorithm. *In Chapter 5*, a new procedure for fingerprint analysis is reported based on the proposed pixel classification algorithm. Thinning of binary images is well studied in the literature. Gray-scale images are thresholded to obtain binary images which are thinned thereafter. Direct gray scale thinning has also received attention. *In Chapter 5*, we treat gray-level thin-

ning of fingerprint images as a two-fold process, where a pixel classification scheme is followed by a binary thinning. A two-pass algorithm is developed to classify a pixel into three classes, namely crest (CR), valley (VA) or plateau (PL). The crest pixels are then used to define the ridge lines of the image. The operations for classification being inherently parallel and highly modular, can be easily implemented on-chip. We propose a simple VLSI implementation of the proposed algorithm using D flip-flops, comparator circuits, combinational logic and adders. The classification scheme uses a Look-Up-Table (LUT).

In Chapter 6, we revisit the *point set pattern matching* (PSPM) problem, and show how this combinatorial technique can be directly applied to fingerprint matching. A set of characteristic points called minutiae is extracted from a thinned version of a fingerprint image, and then the PSPM algorithm is run. In a 2-dimensional Euclidean space, an earlier algorithm for PSPM takes $O(kn^2)$ time, where k (n) denotes the number of points in the query subset (the given set). In this chapter, we propose a new algorithm to check whether or not a query set of points matches with a subset of a given set of points. We pre-process the given set of n points and build a data structure based on relative distances and orientations among them. The worst-case query time complexity of the proposed algorithm is $O(kn^{4/3} \log n)$. Experimental results show that the maximum number of equidistant pairs of points (d_{eq}) among a set points on a $2D$-plane in real life problems is very small. Thus, the expected running time of our algorithm is improved to $O(kd_{eq} \log n)$.

Finally in *Chapter 7*, we summarize the work reported in this thesis, and point out a few research problems.

Chapter 2

Euler number

2.1 Introduction

Topological properties remain invariant under any arbitrary *rubber-sheet* transformation and hence, are very useful in image characterization for matching shapes, recognizing objects, image database retrieval, and many other image processing and computer vision applications. An important topological feature of an image is the *Euler number* (or genus), which is the difference between the number of connected components (objects), and the number of holes [45, 108, 124]. Many critical image processing applications involve large amount of data, and at the same time demand quick real-time response. Euler number provides a simple and fast method of screening in such cases. Euler number of cell images is widely used in medical diagnosis, e.g., detection of malaria infected cells, as the Euler number of an infected cell is often different from that of a good one. It has recently been observed that Euler number is the most clinically useful feature that discriminates many cervical disorders [105]. Being a fundamental topological feature, it has numerous applications in image processing, e.g., optical character recognition, document image processing [139], reflectance-based object recognition [96], analysis of sandstone for geological applications [149], shadow detection [126]. In the next chapter, based on Euler number, the concept of Euler vector is introduced to characterize a gray-tone image.

The classical algorithm for computing the Euler number of a binary image is based on counting certain (2×2) pixel patterns called bit quads [48, 108] over the entire image. Gray [48] used the fact that the Euler number of a region of space is locally countable. The classical graph-theoretic definition of Euler number relating vertices, edges and faces [54] is applied to an image followed by triangulation and then its Euler number is computed as the difference of the number of connected components and that of holes. Alternatively, Euler number is shown to be the difference of left-facing convexities and concavities in the image. Computation of these parameters needs counting of specific types of bit-quads in the image namely, $Q_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$, $Q_2 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ and $Q_D = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ and their rotational equivalents. Let v, t and d be the number of patterns of Q_1, Q_2 and Q_D in an image. Then the Euler number can be expressed as $= (v - t - 2 \times d)/4$ under the 8-connectivity definition [108]. The commercial image processing toolbox MATLAB uses this quad counting algorithm for Euler number computation [156]. Dyer [35] proposed an algorithm to compute the Euler number of an image represented by a quadtree. The local property counting techniques used with an array representation are generalized to counting local node configuration in a quadtree. The root of the quadtree is associated with the entire image. If the block of the image associated with an arbitrary node A, at any level k, does not consist of either all 1's or all 0's, the block is subdivided into four sub-blocks of equal size. These sub-blocks are designated as the four siblings of the given node A, and are at level *k-1*. This definition is applied recursively until uniform blocks of either 1's or 0's are obtained at the leaf. Associated with each node is the content of the sub-image it represents - WHITE, if the image is all 0's, BLACK, if the image is all 1's, and GRAY otherwise. It may be observed that all leaf nodes are either BLACK or WHITE and nonterminal nodes are GRAY. Dyer proves that given such a quadtree with x BLACK nodes, y pairs of adjacent BLACK nodes, and z triples or quadruples of BLACK nodes which surround a point, the Euler number of the binary image which the quadtree represents is equal to $x - y + z$. Samet and Tamminen [129] improved the algorithm further by using a new staircase type of data structure to represent the blocks that have already been processed. They show the technique of using linear quadtrees in conjunction with algorithms for operations that also require inspection of neighbors of leaf nodes. To examine a particular neighboring node, its key is calculated and the linear quadtree is searched to determine the BLACK

leaf node with the given key. If such a leaf is not found, the color(GRAY or WHITE) of the neighbor is inferred from the next higher value in the quadtree stored as a list. The algorithm for calculating geometric features like Euler number assumes that the tree can be traversed in a preorder fashion and the use of internal storage is minimal. However, VLSI implementation of quadtree-based algorithms are complicated as the sizes of the blocks represented by the leaf nodes may be unequal. Further, the number of leaf nodes may vary widely for different image samples.

Rosenfeld and Kak [124] observed that Euler number can be computed from the run length representation of an image. If for each run r, $k(r)$ be the number of runs on the preceding row to which r is adjacent, then Euler number can be expressed as: $E = \sum_r (1 - k(r))$. Further results on 2D and 3D images are reported in [86]. Zenzo et al. [159] suggested a planar graph representation of an image from run descriptions. A binary image is a set $S = (x_1, y_1, z_1), ..., (x_N, y_N, z_N)$ of triples of integers, where x_i is the column number, y_i and z_i are the starting row number and ending row number of a particular run in the x_ith column. A directed graph is defined where, runs are denoted as nodes and there is an edge between two runs (nodes) if they are adjacent. Two runs are adjacent if they lie in adjacent columns and touch one another. The graph G, thus formed is planar. Next, the number of connected components E in the image is computed by applying a standard graph algorithm [29]. The number of holes H is then computed from the Euler's formula of a planar graph [54] as $H = 1 + m - v$, where m = number of edges, and v = number of nodes in the graph. Finally, Euler number is calculated as $E = C - H$. Dey et al. [31] have reported a divide-and-conquer algorithm for computing the Euler number of a binary image and its VLSI implementation. In this approach, the image is partitioned recursively by horizontal and/or vertical cut lines into a number of small disjoint images, whose Euler numbers are computed by using any of the conventional methods [48, 108]. A simple arithmetic rule is then used to compute the Euler number of the original image. For VLSI implementation, a tree-like architecture is designed for calculating the Euler number.

In this thesis, we revisit the run-based method of Rosenfeld and Kak [124] from a new angle. Based on certain properties of 0-1 runs of the pixel matrix, we show that Euler number of a binary image can be computed very fast if the computation is performed

in a specific fashion. This run-based algorithm provides the basis of our VLSI implementation for on-chip computation. Although the theory behind computation of Euler number using runs was observed earlier in [124], their applications to designing commercial tools for computing Euler numbers have not been fully explored, and efficient hardware implementation of such run-based techniques does not seem to be available to date. An analytic expression for the time complexity of the proposed algorithm is derived and the rationale behind its performance is demonstrated with experimental and statistical evidence. Although the worst-case complexity of the algorithm is still $O(N^2)$ for an $(N \times N)$ image, its average-case behavior outperforms significantly the widely used bit-quad counting algorithm [48, 108]. Next, a simple parallel version of the algorithm is described that can be executed in $O(N \log N)$ time using $O(N)$ processing elements. We exploit the underlying properties of this technique to design an adder tree using a pipeline architecture that can easily be implemented as a VLSI chip. The same circuit can handle arbitrarily sized pixel matrices with minor modifications. The algorithm was run on a database of 1039 logo images for computing the Euler number of each of them. Experimental results are found to be very encouraging, and demonstrate the superiority of this method to earlier approaches.

2.2 Proposed Sequential Algorithm

2.2.1 Theme

Let the binary image be represented by a 0-1 pixel matrix of size $(N \times M)$, in which an object (background) pixel is denoted as 1 (0). In a binary image, a *connected component* is a set of object pixels such that any object pixel in the set is in the 8 (or 4) neighborhood of at least one object pixel of the same set. A *hole* is a set of background pixels such that any background pixel in the set is in the 4 (or 8) neighborhood of at least one background pixel of the same set and this entire set of background pixels is enclosed by a connected component. The sets referred to in the definition are sets contained in the image. A *run* in any column (or row) of the pixel matrix is defined to be a maximal sequence of consecutive 1's in that column (or row). Let *R(i)* denote

the number of such runs in the i-th column (row). In fact, $R(i)$ can be counted as the number of 0-1 transitions in that row with a 0 padded at the start.

Fact 1 *If the image I consists of a single row or a single column i, the Euler number $E(I) = R(i)$.*

Fact 2 *Euler number satisfies the additive set property. Given two images I_1 and I_2 with Euler numbers $E(I_1)$ and $E(I_2)$ respectively, the Euler number of the image $I = I_1 \cup I_2$ is given by: $E(I) = E(I_1 \cup I_2) = E(I_1) + E(I_2) - E(I_1 \cap I_2)$ (see [48]).*

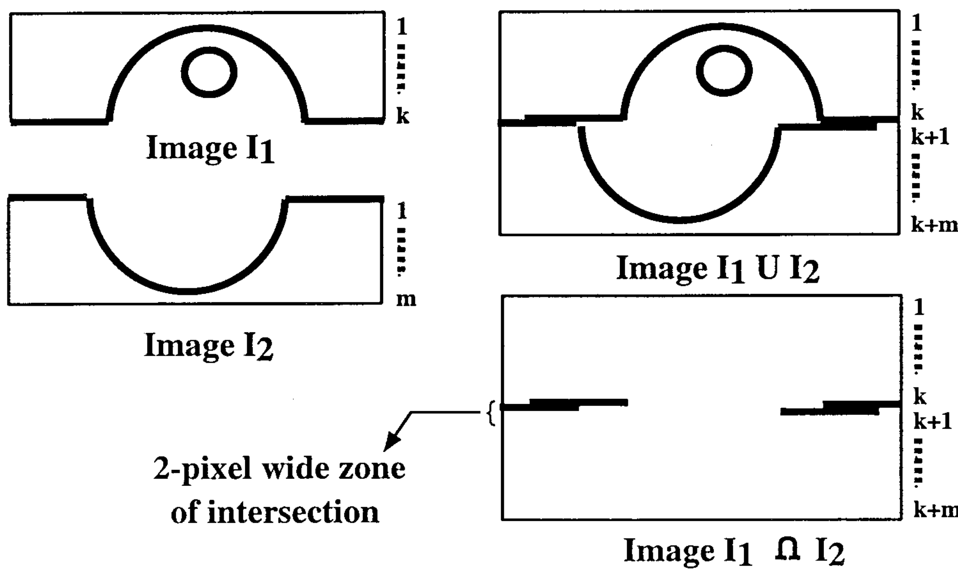


Figure 2.1: Union and intersection of images

The *union* ($\cup$) of two images is defined as simple juxtaposition of I_1 and I_2 either vertically or horizontally, without any overlap. The *intersection* ($\cap$) of I_1 and I_2 is the image formed by the last row (or column) of I_1, and the first row (or column) of I_2, if the images I_1 and I_2 are joined horizontally (or vertically). The intersection image is always two pixel row (or column) wide. Without any loss of generality, let I_1 and I_2 be joined horizontally. Thus, the last row of I_1 will lie above the first row of I_2. See Figure 2.1 for an example. Two runs appearing in two adjacent rows each, are said to be *neighboring* if at least one pixel of a run is in the 8 (or 4) neighborhood of a pixel

of the other run; we follow 8 neighborhood convention throughout. Clearly, $(I_1 \cap I_2)$ will denote the image containing the last row of I_1 and the first row of I_2 (see Figure 2.1), and as no holes can be present in a two-row wide image, we have the following observation:

Fact 3 $E(I_1 \cap I_2)$ = *the number of neighboring runs between* I_1 *and* I_2.

We now use Facts 1, 2 and 3 iteratively to compute the Euler number of the entire image, as follows.

Let I_{i-1} be the partial image consisting of rows $1, 2, \ldots, (i-1)$ of the pixel matrix. Let $E(I_{i-1})$ be the Euler number of I_{i-1}. The Euler number of the image consisting of only row $i = R(i)$ (by Fact 1). The row i is now added to I_{i-1} to form the union image I_i. The intersection image is formed by the $(i-1)^{th}$ and the ith row. Let the number of neighboring runs between them be O_i. Hence,

$E(I_1) = R(1)$

$E(I_2) = E(I_1) + E(2) - O_2 = R(1) + R(2) - O_2$

$E(I_3) = E(I_2) + E(3) - O_3 = R(1) + R(2) + R(3) - (O_2 + O_3)$

...

$E(I_N) = E(I_{N-1}) + E(N) - O_i$

$= (R(1) + R(2) + R(3) + \ldots + R(N)) - (O_2 + O_3 + \ldots + O_N)$

$= \sum_{i=1}^{N} R(i) - \sum_{i=2}^{N} O_i$

where, I_N denotes the entire image.

The above analysis proves the following known result [124], and provides the basis of our VLSI implementation.

Theorem 1 *The Euler number of a given binary image is the difference between the sum of the number of runs for all rows (or columns), and the sum of the neighboring runs between all consecutive pairs of rows (or columns).* □

Algorithm Compute_Euler_Sequential

Input: An $(N \times M)$ binary pixel matrix of an image I;

Output: Euler number $E(I)$.

Compute the number of runs $R(1)$ present in the first row;

$E(I) = R(1)$;

for ($i \leftarrow$ row number 2 to N)

Calculate the number of runs $R(i)$ in the ith row;

Calculate the number of neighboring runs O_i between $(i-1)^{th}$ and i^{th} rows;

$E(I) = E(I) + R(i) - O_i$;

endfor

return E(I) as the Euler number.

2.2.2 Space and time complexity

We require $N \times M$ space to store all the pixels of the given image. To calculate the number of neighboring runs between two consecutive rows, we need to store for each run, the column numbers where a run starts or terminates. The worst case arises when a row has the maximum number of runs, i.e., when every alternating pixel is an object pixel. For M columns, the maximum number of runs in a row is $\lceil (M/2) \rceil$. We calculate the number of neighborhood runs between two consecutive rows at a time. Since a run can be designated by its two ends, the total space required $= (N \times M) + 4 \times \lceil (M/2) \rceil \approx O(N \times M)$.

The time complexity can be measured in terms of the number of pixel accesses. Computation of runs for all the rows needs $(N \times M)$ pixel accesses. The number of pixel accesses required to find the two end points of all runs present in the matrix $= 2 \times \sum_{i=1}^{N} R(i) \leq (N \times M)$. Checking whether a run in row $(i-1)$ is in the neighborhood of a run of row i, requires 2 pixel accesses. Determination of neighboring runs for all consecutive pairs of rows needs $= 2 \times \sum_{i=2}^{N} O_i \leq (N \times M)$ pixel accesses. Therefore, the total number of pixel accesses is, $P = (N \times M) + 2 \times \{\sum_{i=1}^{N} R(i) - \sum_{i=2}^{N} O_i\}$ which is $O(N \times M)$ in the worst case. Thus, it has the same order as that of the earlier method [48]. However, almost for all the cases, the actual performance of the algorithm based on run and neighboring run is found to be superior. For a given image, the number of pixel accesses becomes equal to $2 \times \{(N \times M) + M\}$, if the bit quad counting algorithm [48] is used to compute Euler number, with the assumption that out of the four pixels

STATISTICAL INSTITUTE T 135

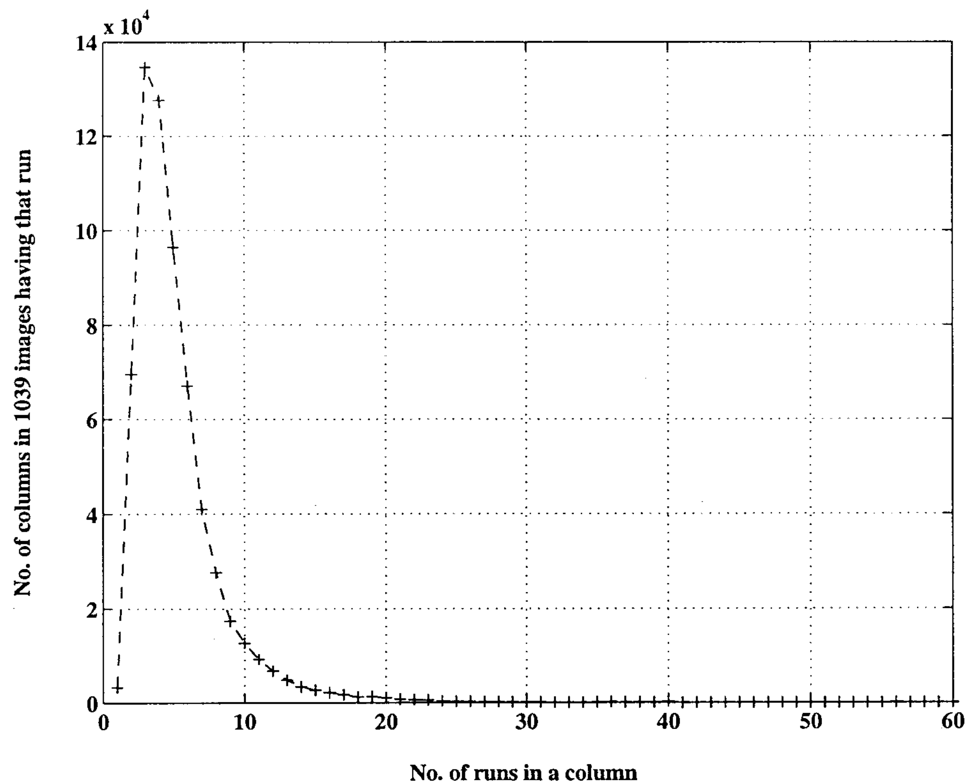


Figure 2.2: Run Statistics

forming the quad, two are retained from the last consecutive quad and two are read from the memory. In the run-based method, the number of pixel accesses depends on the distribution of the runs in the pixel matrix. It has been observed that for most of the images, $R(i) \ll M$ and also $O_i \ll M$, and typically both $R(i)$ and O_i have a value around 4. Thus, on the average, the number of pixel accesses will be much less compared to those of the other methods [31, 48]. Empirical evidence justifies the rationale behind savings. We have considered a database of 1039 logo images, and normalized each of them to the same size. From the experimental results, the expected value of the number of runs $R(i)$ present in a column is observed to be 4.252741 and that of the neighboring runs between two consecutive columns, O_i is found to be 4.266170 (see Figures 2.2 and 2.3).

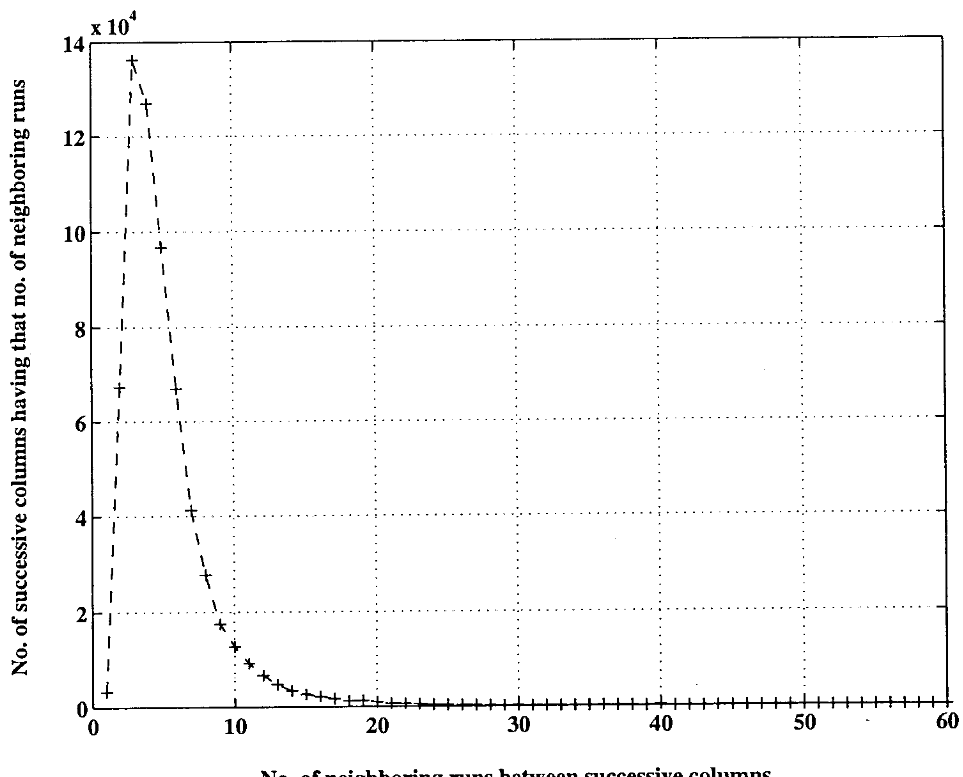


Figure 2.3: Neighboring Run Statistics

2.2.3 Results

We implemented both the proposed run-based and the earlier quad counting algorithm [48] in C, and studied its performance in terms of the number of pixel accesses as well as CPU time for each of the 1039 images in the logo database. We have observed significant improvement in most of these cases. Because of space limitation, only a few of them (104 images) are shown in Table 2.3. All the images in the database were

Table 2.1: Distribution of Euler numbers

Range of Euler Numbers	-3796 to -10	-9 to 10	11 to 2425
Number of images	85	829	125

grouped into bunches of ten, and from there one image was selected at random, thus selecting 104 images. The CPU time is for a 233 MHz Sun Ultra-5_10, Sparc; the

OS is SunOS Release 5.7 Generic. The values of Euler number of the images in the database vary widely from −3796 to 2425. There are 85 images having Euler numbers in the range −3796 to −10; 829 images in the range, −9 to 10; and 125 images in the range, 11 to 2425. There are 76 images each with a distinct Euler number; there are 18 cases where only two images have the same Euler number. The distribution and frequency of the images having different Euler numbers are shown in Tables 2.1 and 2.4. This observation justifies that Euler number can be used as a potential tool for image discrimination, search, and retrieval. Few sample images from the logo database with their Euler numbers are shown in Figure 2.12. Figure 2.4 depicts a bar chart showing the CPU time comparison between the proposed and bit-quad counting algorithm for a few logo images. Figure 2.5 shows the distribution of the ranges of the Euler numbers.

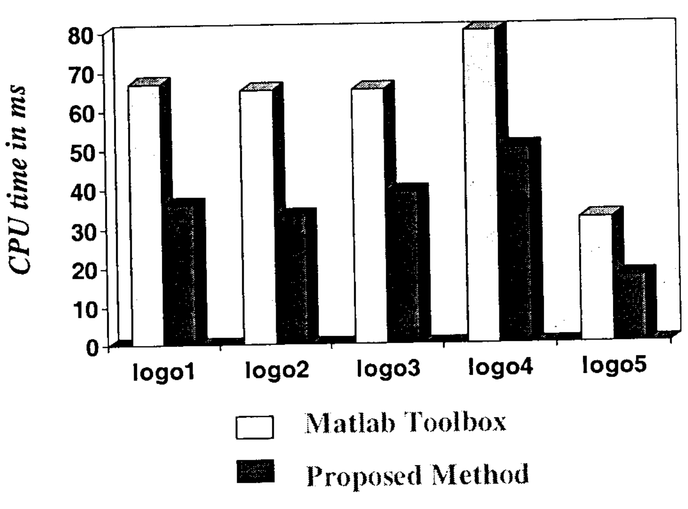


Figure 2.4: Bar chart showing comparison of CPU time for some logo images

2.3 Parallel Algorithm and VLSI Implementation

The algorithm proposed in the previous section can be easily parallelized so as to make it suitable for on-chip implementation.

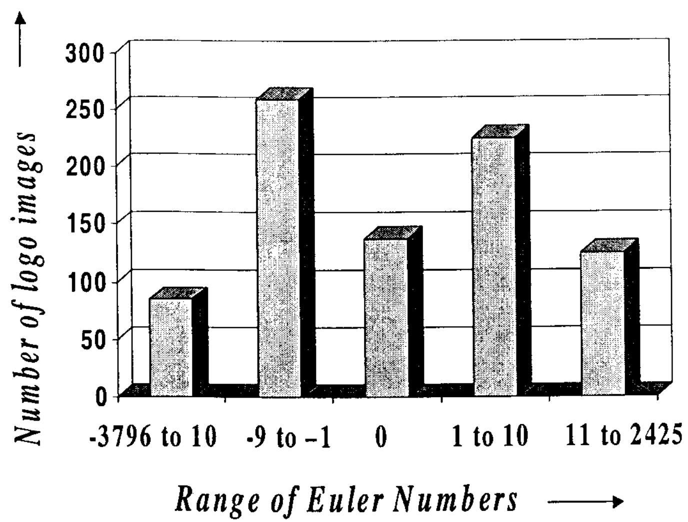


Figure 2.5: Bar chart showing distribution of ranges of Euler number

Algorithm Compute_Euler_Parallel

Input: An $(N \times M)$ binary pixel matrix of an image I;

Output: Euler number $E(I)$.

for ($i \leftarrow$ row number 1 till end) **do in parallel**

  Calculate the number of runs $R(i)$ in the ith row;

  Calculate the number of neighboring runs O_i between $(i-1)^{th}$ and i^{th} rows;

endfor

$E(I) = \sum_{i=1}^{N} R(i) - \sum_{i=2}^{N} O_i$

return E(I) as the Euler number.

2.3.1 VLSI architecture

We need two types of processing elements (PE), P_1 for identifying the start of a run in a row, and P_2, for identifying the start of a neighboring run.

The PE P_1 (shown in Figure 2.6) is used to detect the transition from 0 to 1 and the number of runs is equal to the number of such detections. A delay (D) flip-flop is initialized to 0 at the start of processing each row. It holds the value of the previous pixel for the purpose of checking a transition. The pixels in a row are pipelined into P_1.

At any instant of time t_i, the ith pixel and $(i-1)^{th}$ pixel of a row are checked for a 0 to 1 transition. The maximum number of runs in a row having M columns is $\lceil (M/2) \rceil$.

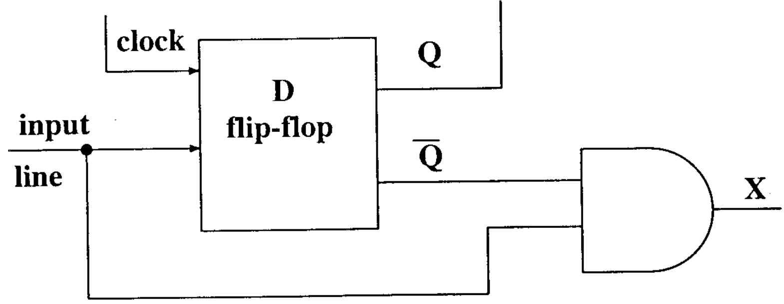


Figure 2.6: Processing element P_1

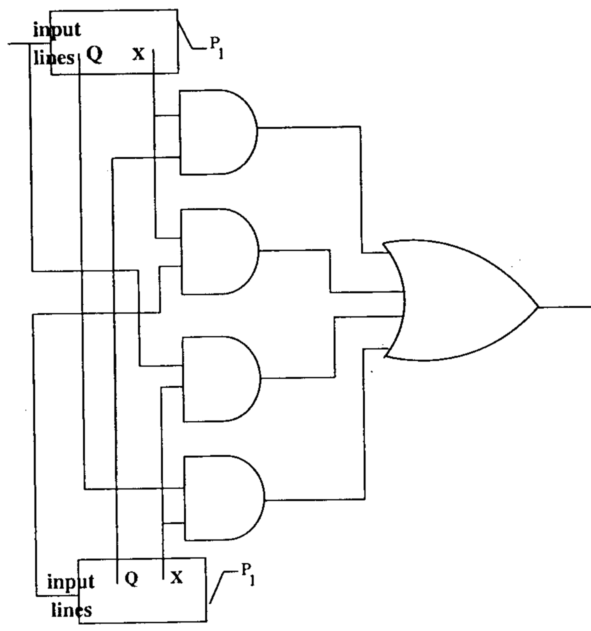


Figure 2.7: Processing element P_2

In Figure 2.7, P_1 is shown within a box, and the remaining portion of the circuit constitutes the processing element P_2. It is used to detect the start of a neighboring run between two consecutive rows. The PE P_2 checks for the condition when a run in a row begins, and whether it is in the neighborhood of another run in its adjacent row. The pixels corresponding to the columns of two adjacent rows are fed to P_2 in a pipeline. In addition, it receives data from the outputs of the D flip-flops of these two rows. At

any instant of time t_i, i^{th} and $(i-1)^{th}$ pixels of two consecutive rows are checked for a neighboring run. The maximum number of neighboring runs is $2 \times \lceil (M/2) \rceil$. To process an $(N \times M)$ image in parallel, we require N pieces of P_1, and $(N-1)$ pieces of P_2.

To compute the Euler number of the entire image, we add the number of runs (calculated by summing up all 0-1 transitions in all rows) and deduct from it the number of neighboring runs (calculated by adding up the number of all neighboring runs between consecutive rows). At any instant of time t_i, the output from P_1 or P_2 goes high indicating the start of a run or neighboring run respectively. A neighboring run implies the presence of a run in either or both of the corresponding row(s). Thus, a high (1) output from P_2 is accompanied by a high output from either one or both of the P_1's connected to it. The interrelation of runs and neighboring runs is depicted as a truth table in Table 2.2, where $\times$ represents a don't care entry.

2.3.2 Truth Table and Combinational Circuit

We take the outputs from the two modules generating P_1 and two modules producing P_2 for consecutive rows. To distinguish them, we use upper and lower case symbols (P_1, p_1, P_2, p_2) in Table 2.2. The sum of the two P_1 outputs is computed, and the sum of the two P_2 outputs is then subtracted from it. It is easy to prove that the result is always either -1, or 0, or 1. This follows from the properties of runs and neighboring runs as mentioned in the previous subsection. The entries corresponding to rows 5 and 6 in the truth table are not feasible as $P_2 = 1$ implies that either or both of P_1 and p_1 must be 1. The case, as in row 10, does not appear as $p_2 = 1$ and $p_1 = 0$ implies the PE P_1' (and not included in this group of 4 P_1's and P_2's, and as such not shown in the truth table) associated with p_2 should be high. If P_1' and $p_2 = 1$, it implies the presence of a continuing run (not a run start) in the row associated with p_1; otherwise p_2 cannot become high. If there exists a continuing run associated with p_1 and also $P_1 = 1$, then P_2 should be high, but that does not happen. Hence, this input combination never arises. Rows 11 and 12 are also inadmissible as in both the cases, $P_1 = 1$ and $p_1 = 1$ imply that $P_2 = 1$, which is not true. To represent $-1, 0, 1$ in 2's complement, we require 2 bits (the sign bit and data bit). A combinational circuit C as shown in Figure

Table 2.2: Truth Table

	P_1	P_2	p_1	p_2	$(P_1 + p_1) - (P_2 + p_2)$	Sign-bit(s)	Data-bit(d)
1	0	0	0	0	0	0	0
2	0	0	0	1	-1	1	1
3	0	0	1	0	1	0	1
4	0	0	1	1	0	0	0
5	0	1	0	0	x	x	x
6	0	1	0	1	x	x	x
7	0	1	1	0	0	0	0
8	0	1	1	1	-1	1	1
9	1	0	0	0	1	0	1
10	1	0	0	1	x	x	x
11	1	0	1	0	x	x	x
12	1	0	1	1	x	x	x
13	1	1	0	0	0	0	0
14	1	1	0	1	-1	1	1
15	1	1	1	0	1	0	1
16	1	1	1	1	0	0	0

2.8, is designed following the truth table (Table 2.2) to produce the sign and data bits from different input combinations of the two P_1's and P_2's.

2.3.3 Adder Design and Time Complexity

The complete hardware design for a (16×16) image is shown in Figure 2.10. The output of C represents a 2-bit 2's complement number generated by two P_1's and two P_2's. An adder circuit is needed to sum up the outputs of all these C-modules to obtain the final result. We use a binary adder tree in pipeline [61] to accelerate the addition process. Addition of two 2's complement numbers may produce a 3-bit number. To implement such a scheme, we use at the leaf level of the adder tree, a set of 3-bit adders each with a sign bit extension. For each subsequent level in the tree, the

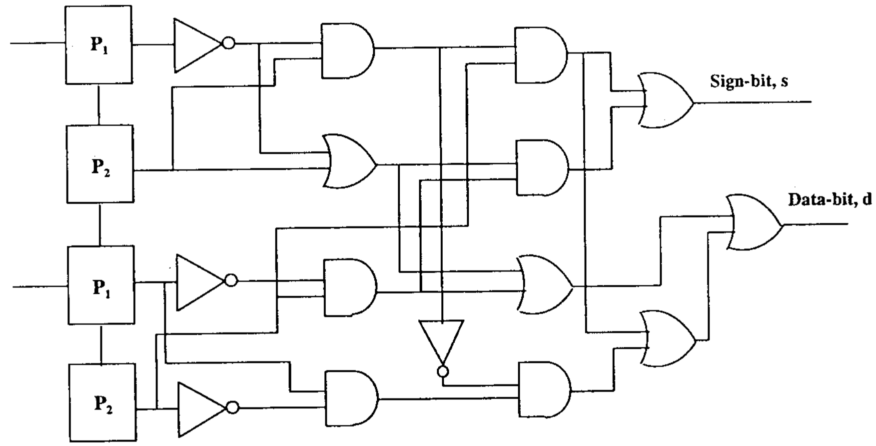


Figure 2.8: Combinational circuit for finding the sign and data bit

adder size (width) is increased by one bit. For an $N \times M$ image, we require $\lceil (N/2) \rceil$ pieces of combinational module C. Each C-module produces an output in the range $[-1, 1]$. Thus, each column can have a maximum sum of $\lceil (N/2) \rceil$ and a minimum sum of $-\lceil (N/2) \rceil$. The depth of the adder tree would be $\lceil \log_2 N \rceil - 1$. As the width of the adder in the tree increases by 1 at each level, the width of the adder A_r at the root, would be $= (3 - 1) + \lceil \log_2 N \rceil - 1 = \lceil \log_2 N \rceil + 1$. Finally a sequential adder FA (see Figure 2.10) is used to accumulate the sum. Since a run and a neighboring run for a row cannot begin in two consecutive columns, the range of numbers that FA should be able to handle is $[-M/2 \times \lceil (N/2) \rceil, M/2 \times \lceil (N/2) \rceil]$. Thus, the number of bits T required for the adder FA would be $= \lceil \log_2 \{M \times \lceil (N/2) \rceil + 1\} \rceil$. With the assumption that $M \approx N$, T is $O(\log_2 N)$. The number of the stages of the pipeline is one more (for the adder FA) than the depth of the adder tree. Therefore, the stages of the pipeline equals $\lceil \log_2 N \rceil$. The clock period of the linear pipeline is determined by the longest pipeline stage, and hence is equal to T, the time taken by the adder FA. The number of clock cycles required by the linear pipeline to perform the entire addition is $\lceil \log_2 N \rceil + (M - 1)$. Therefore, total time needed to produce the final output in 2's complement form is $T \times \{(M - 1) + \lceil \log_2 N \rceil\}$ and M being approximately equal to N, the time taken is $O(N \log N + 2 \times \log_2^2 N) \approx O(N \log N)$. If the final adder FA performs a carry look ahead addition [87] taking $O(\log T)$ time, then the clock period of the pipeline can further be reduced and would be determined by the delay of A_r.

2.3.4 Circuit cost

We require the following components to implement the VLSI architecture for processing an $(N \times M)$ binary image:

1) N pieces of processing element P_1, each having one edge triggered D-flip-flop and one 2-input AND gate;

2) $N-1$ pieces of processing element P_2, each having four 2-input AND gates and one 4-input OR gate;

3) $\lceil (N/2) \rceil$ pieces of the combinational circuit C, each having four inverters, six 2-input AND gates and five 2-input OR gates;

4) $\lceil (N/4) \rceil$ pieces of 3-bit full adder, $\lceil (N/8) \rceil$ pieces of 4-bit full adder, ..., one piece of $\lceil \log_2 N \rceil$-bit full adder;

5) one piece of $\lceil \log_2 \{M \times \lceil (N/2) \rceil + 1\} \rceil$-bit full adder.

A complete circuit for a (16×16) image with appropriate adder blocks is shown in Figure 2.10.

2.3.5 Handling large images

Given an architecture for computing the Euler number of an image matrix of size $N \times N$, the Euler number of a larger image of size $K \times K$, where $K = x \times N$ can be easily determined. The matrix is partitioned into several $N \times K$ sub-blocks as $B_1, B_2, ..., B_x$, where each B_i is an $N \times K$ matrix, as shown in Figure 2.9. The Euler number of each such block of size $N \times K$ is computed using our proposed architecture for an $N \times N$ block by changing the size of the adder FA. Let E_i be the Euler number of a block B_i. The Euler number of the overall image can now be easily determined by using the following algorithm based on Fact 2 and Fact 3 as mentioned in Section 2.2.1.

Algorithm Compute_Euler_General

Input: A $(K \times K)$ binary pixel matrix of an image I, where $K = x \times N$;

Output: Euler number $E(I)$.

$E(I) = 0;$

$O_i = 0;$

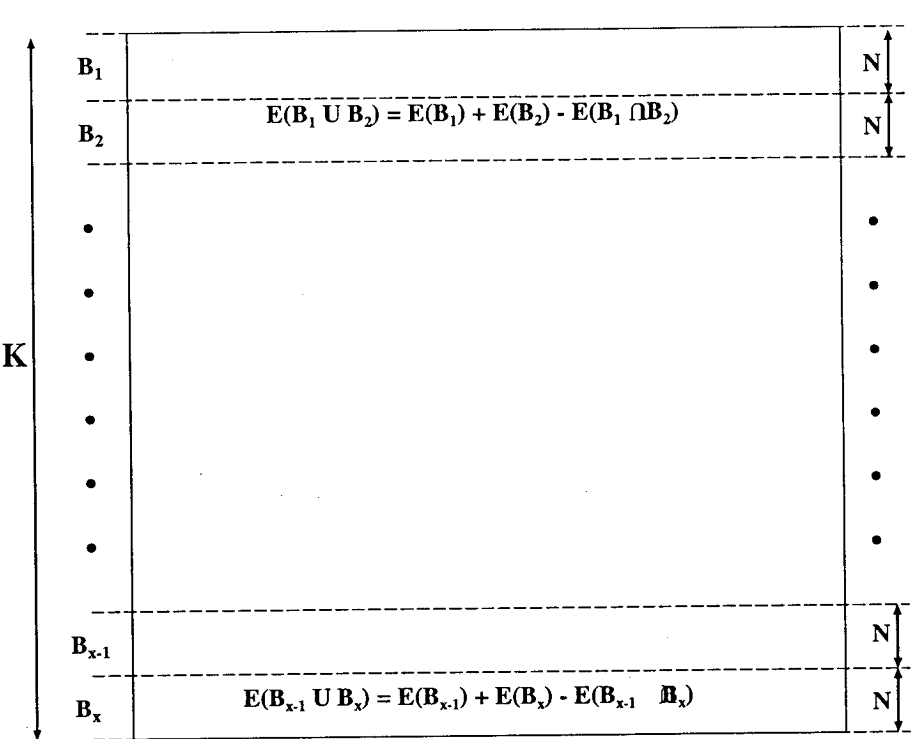


Figure 2.9: The scheme for handling large images

for ($i \leftarrow$ block number 1 to $x - 1$)

Calculate the Euler number, $E(B_i)$ of the block B_i using *Compute_Euler_Parallel*;

Calculate the number of neighboring runs O_i between last row of block B_i and first row of block B_{i+1};

$E(I) = E(I) + E(B_i) - O_i$;

endfor

Compute the Euler number, $E(B_x)$ of the last block B_x using *Compute_Euler_Parallel*;

$E(I) = E(I) + E(B_x)$;

return $E(I)$ as the Euler number.

The adder tree can handle an image matrix with N rows, as it deals with only one column at a time. The adder FA adds up values from all columns. When the number of columns changes from N to K, the width W of the adder should be made equal to $\lceil \log_2\{K \times \lceil (N/2) \rceil + 1\} \rceil$. To calculate the number of neighboring runs (O_i) between

the last row of block B_i and the first row of block B_{i+1}, we require processing elements P_1 and P_2. The output of P_2 is fed as an input of the last C-module in the column (Figure 2.11). The outputs of FA are stored in parallel-in, parallel-out registers. We need x such registers. The width of the registers should be the same as that of FA. The register corresponding to an $N \times K$ block of the given matrix is selected using a selector circuit, and the corresponding memory block to be pipelined is also selected. Final addition of all the values stored in these registers can be performed in $O(\log x + \log W)$ time using carry save addition [87]. An example of a circuit for computing the Euler number of an image of size (256×256) using the circuit module for a (16×16) image, is shown in Figure 2.11.

2.4 Conclusions

A run-based algorithm for computing the Euler number of a binary image is analyzed and a new VLSI implementation based on pipeline architecture for fast on-chip computation, is reported. The algorithm is based on certain combinatorial and statistical properties of the runs present in the pixel matrix of the image. Experimental results on a logo database show very encouraging results in terms of CPU time. The VLSI implementation is simple and a basic module can be used to handle arbitrarily large sized pixel matrices. Exploring such algorithms for applicability to higher dimensions needs further investigation.

Acknowledgments: We would like to thank Prof. Anil K. Jain and Aditya Vailya of the Michigan State Univ., USA, for sending us the logo trademark database.

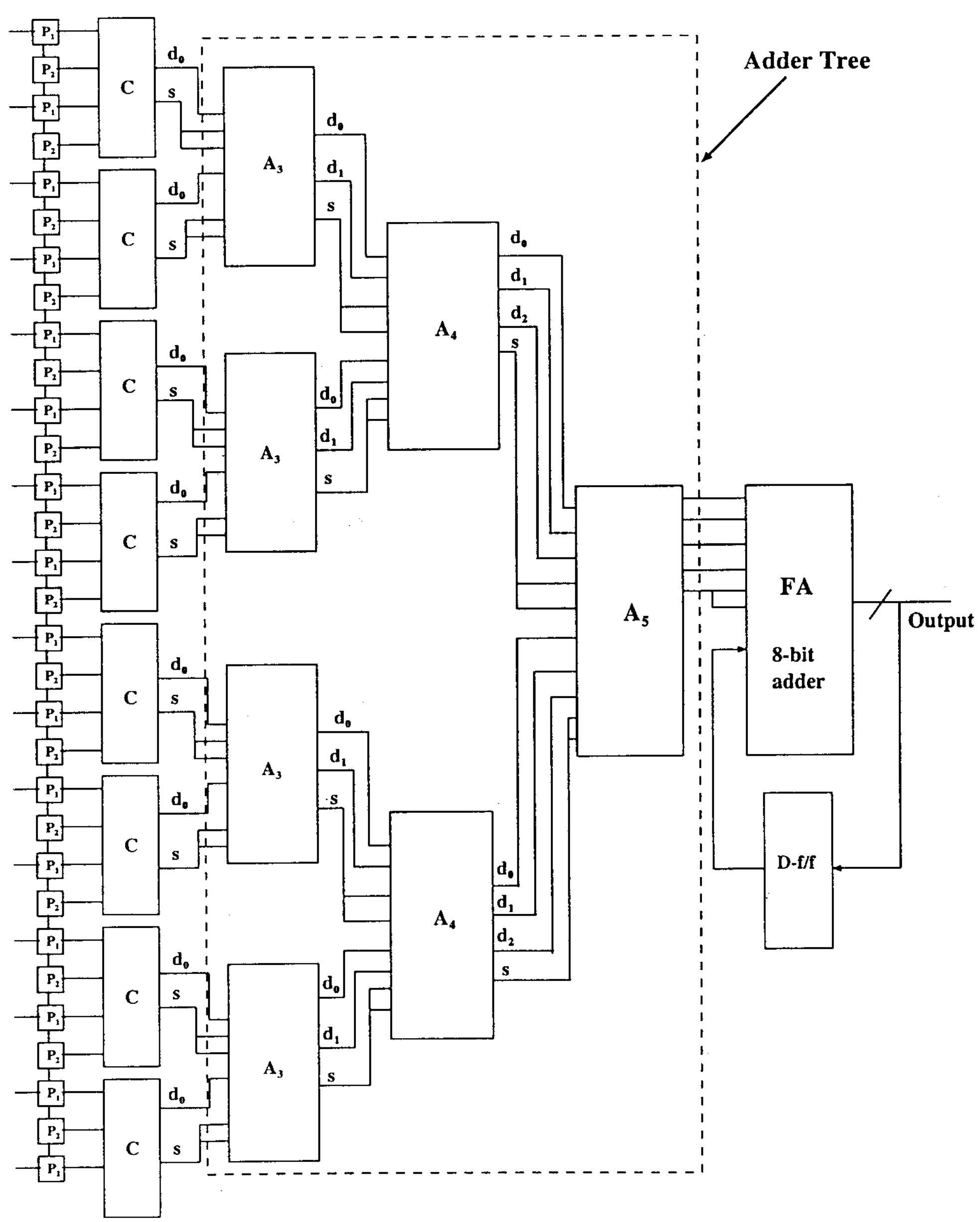


Figure 2.10: A sample circuit for calculating the Euler number of a (16×16) image

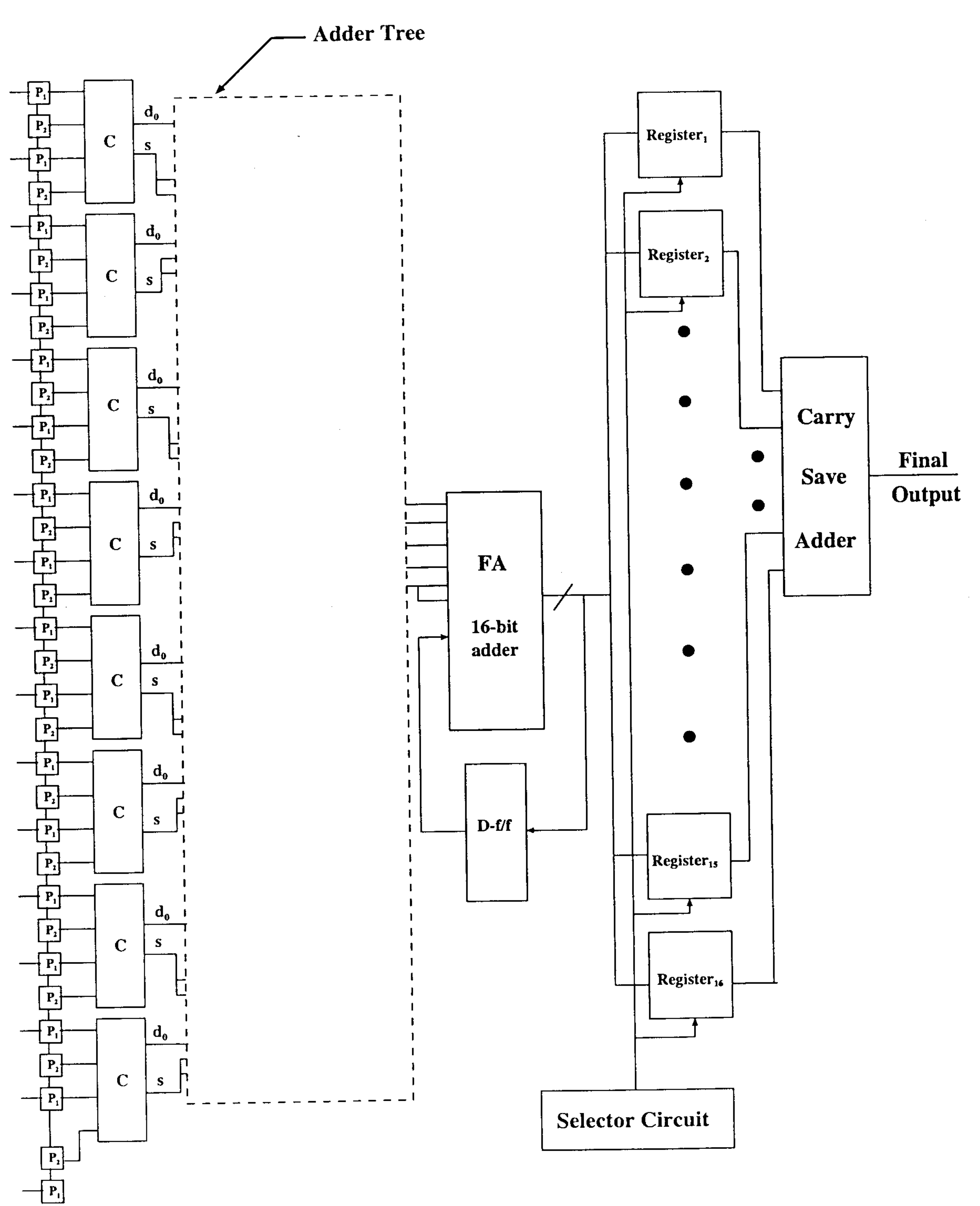


Figure 2.11: A scalable circuit for calculating the Euler number of a (256×256) image using the circuit for a (16×16) image

Table 2.3: Comparative results

Image number	Image size	Euler number	Pixel access			CPU time (in microseconds)		
			Proposed method	Gray's method	% saving	Proposed method	Gray's method	% saving
1	200x290	-2	94604	168780	43.9	14761	25919	43.0
2	274x274	-1	80800	150700	46.4	12251	23175	47.1
3	282x282	0	87406	159612	45.2	13903	24120	42.4
4	226x226	7	56108	102604	45.3	8737	17849	51.1
5	128x128	-2	18346	33024	44.4	2822	4948	43.0
6	128x128	0	18382	33024	44.3	2862	4946	42.1
7	256x256	1	69338	131584	47.3	10806	20413	47.1
8	138x138	0	26644	38364	30.5	4279	5726	25.3
9	170x170	9	31988	58140	45.0	5126	8647	40.7
10	160x160	-1	28698	51520	44.3	4409	7707	42.8
11	220x220	-4	54422	97240	44.0	9046	17112	47.1
12	158x158	2	30428	50244	39.4	5048	7487	32.6
13	186x186	-7	39598	69564	43.1	6020	10334	41.7
14	176x176	-3	39644	62304	36.4	7191	9324	22.9
15	176x176	1	34144	62304	45.2	5511	9259	40.5
16	162x162	1	29504	52812	44.1	4715	7880	40.2
17	128x128	-1	19758	33024	40.2	3234	4965	34.9
18	256x256	0	72426	131584	45.0	11857	19459	39.1
19	128x128	-8	18026	33024	45.4	2777	4999	44.4
20	128x128	-2	26490	33024	19.8	4397	4994	12.0
21	128x256	2	37302	65792	43.3	6814	9776	30.3
22	128x256	-4	40434	65792	38.5	6794	9871	31.2
23	128x128	0	17948	33024	45.7	2796	4944	43.4
24	256x256	0	68086	131584	48.3	10890	19638	44.5
25	256x128	-14	45086	66048	31.7	7140	9839	27.4
26	256x256	-5	76098	131584	42.2	12944	19459	33.5
27	256x256	24	82084	131584	37.6	13486	22095	39.0
28	64x256	-3	20072	32896	39.0	3256	4925	33.9
29	128x256	0	35778	65792	45.6	5611	9785	42.7
30	128x256	0	35746	65792	45.7	5527	9786	43.5
31	128x256	-1	35916	65792	45.4	6112	10078	39.4
32	64x256	-12	21042	32896	36.0	3497	4928	29.0
33	258x258	4	75094	133644	43.8	11834	19899	40.5
34	540x540	-3	304540	584280	47.9	48621	89588	45.7
35	400x400	5	172772	320800	46.1	25781	50950	49.4
36	350x280	-15	108734	196700	44.7	17006	29507	42.4
37	412x424	46	211260	350200	39.7	33249	54874	39.4
38	425x432	-1	199446	368050	45.8	31888	57469	44.5
39	433x432	-17	270730	374978	27.8	42252	58556	27.8
40	512x312	-12	173848	320512	45.8	28749	50531	43.1
41	462x408	2	207334	377916	45.1	33160	59370	44.1
42	425x416	0	219360	354450	38.1	35193	55563	36.7
43	455x376	37	204088	343070	40.5	34852	53909	35.4
44	563x488	-4	294732	550614	46.5	49084	85565	42.6
45	387x448	1	183272	347526	47.3	28112	54464	48.4
46	475x400	-1	199684	380950	47.6	30359	59364	48.9
47	375x424	0	170784	318750	46.4	26977	51384	47.5
48	356x488	2	186470	348168	46.4	31732	54722	42.0
49	442x424	13	442138	375700	-17.7	65924	58591	-12.5
50	425x320	-2	142428	272850	47.8	23337	40170	41.9

Image number	Image size	Euler number	Pixel access			CPU time (in microseconds)		
			Proposed method	Gray's method	% saving	Proposed method	Gray's method	% saving
51	406x408	-2	177498	332108	46.6	28686	48889	41.3
52	459x456	0	214512	419526	48.9	33606	65073	48.4
53	512x424	1	229170	435200	47.3	36652	67726	45.9
54	561x328	-3	199888	369138	45.9	31546	57768	45.4
55	499x240	1	148960	240518	38.1	23017	35924	35.9
56	469x344	1	171520	323610	47.0	28155	51538	45.4
57	437x584	1	266522	511290	47.9	43635	78633	44.5
58	375x384	-2	159834	288750	44.6	24812	42546	41.7
59	462x456	-6	248972	422268	41.0	38567	65675	41.3
60	337x464	-404	203392	313410	35.1	33477	50538	33.8
61	437x480	-8	247042	420394	41.2	39986	65203	38.7
62	412x424	2425	287160	350200	18.0	45772	55939	18.2
63	442x440	-1	263096	389844	32.5	44045	60809	27.6
64	242x272	2	73126	132132	44.7	11763	19538	39.8
65	475x280	1	144596	266950	45.8	22766	39373	42.2
66	387x408	-7	171440	316566	45.8	28024	46600	39.9
67	254x432	-49	126732	219964	42.4	19729	32425	39.2
68	392x264	-1	108864	207760	47.6	16774	31243	46.3
69	337x344	0	124986	232530	46.2	20325	34295	40.7
70	337x336	-3	130358	227138	42.6	21508	33590	36.0
71	337x336	2	125874	227138	44.6	20793	33557	38.0
72	358x296	1	116894	212652	45.0	17781	32103	44.6
73	512x432	-1	239682	443392	45.9	39593	68566	42.3
74	357x576	5	293048	411978	28.9	47449	63956	25.8
75	425x448	60	235146	381650	38.4	37824	59463	36.4
76	362x472	3	228736	342452	33.2	37191	53763	30.8
77	512x512	5	283878	525312	46.0	43663	80683	45.9
78	512x512	0	279436	525312	46.8	43466	81896	46.9
79	256x512	7	142814	262656	45.6	23614	38671	38.9
80	512x512	15	284272	525312	45.9	49445	80554	38.6
81	512x256	41	164742	263168	37.4	27866	38805	28.2
82	512x256	13	141948	263168	46.1	22889	38742	40.9
83	256x512	8	144682	262656	44.9	23343	38715	39.7
84	512x256	0	137132	263168	47.9	22096	38767	43.0
85	512x512	10	282702	525312	46.2	47087	80572	41.6
86	128x512	12	116642	131328	11.2	20167	19426	-3.8
87	512x512	3	318120	525312	39.4	51924	80885	35.8
88	512x256	-2	140828	263168	46.5	23449	39264	40.3
89	256x512	-15	158006	262656	39.8	26671	38668	31.0
90	128x512	-45	95388	131328	27.4	16189	19399	16.5
91	512x256	1	138928	263168	47.2	21890	38721	43.5
92	256x256	8	76656	131584	41.7	12530	19446	35.6
93	512x512	5	311916	525312	40.6	51615	80729	36.1
94	512x256	0	135784	263168	48.4	21206	38717	45.2
95	512x256	7	151640	263168	42.4	24346	39235	37.9
96	512x512	34	350744	525312	33.2	57685	80492	28.3
97	512x256	-153	166786	263168	36.6	28598	38851	26.4
98	256x512	-5	150124	262656	42.8	24978	40049	37.6
99	425x432	1	206530	368050	43.9	33882	57382	41.0
100	437x432	-2	219934	378442	41.9	36277	58744	38.2
101	300x320	0	110468	192600	42.6	18209	28519	36.2
102	256x256	-3	67426	131584	48.8	9810	19526	49.8
103	256x256	-3	81198	131584	38.3	13117	20071	34.6
104	512x512	9	274460	525312	47.8	41480	80620	48.5

Table 2.4: Distribution of Euler Number for 1039 logo images

Euler Number	Number of images	Euler Number	Number of images	Euler Number	Number of images
-3796	1	-18	3	27	2
-2624	1	-17	3	29	1
-1698	1	-16	1	30	1
-1059	1	-15	2	33	1
-653	1	-14	6	34	2
-650	1	-13	3	35	1
-404	1	-12	6	36	1
-378	1	-11	1	37	4
-302	1	-10	6	38	2
-256	1	-9	12	39	1
-160	1	-8	15	40	1
-154	1	-7	17	41	4
-153	1	-6	24	43	1
-147	1	-5	31	44	1
-142	1	-4	31	46	2
-112	1	-3	49	48	1
-110	1	-2	69	49	2
-95	1	-1	80	50	2
-92	1	0	137	53	1
-87	1	1	121	55	1
-56	1	2	60	56	1
-54	2	3	44	57	2
-52	2	4	46	58	1
-51	1	5	25	60	1
-49	1	6	14	62	1
-48	1	7	16	65	1
-45	1	8	14	66	1
-43	1	9	11	73	2
-42	1	10	13	75	1
-40	1	11	9	80	1
-36	1	12	6	82	1
-35	1	13	7	112	1
-33	1	14	4	114	1
-32	3	15	7	163	1
-31	1	16	3	168	1
-29	1	17	7	204	1
-28	1	18	8	233	1
-26	1	19	1	254	1
-25	2	20	2	276	1
-24	1	21	3	293	1
-23	2	22	3	534	1
-22	2	23	1	861	1
-21	2	24	2	1567	1
-20	2	25	1	2425	1
-19	1	26	3		

Figure 2.12: Some Logo Images and their Euler numbers (EN) from Table 2.3

(3) EN = 0

(39) EN = -17

(43) EN = 37

(46) EN = -1

(49) EN = 13

(58) EN = -2

(62) EN = 2425

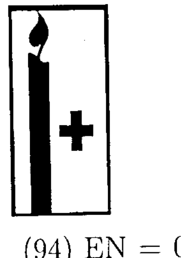
(68) EN = -1

(73) EN = -1

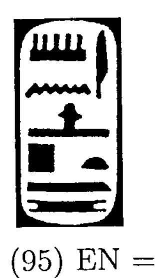
(76) EN = 3

(80) EN = 15

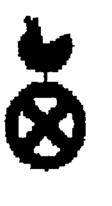
(84) EN = 0



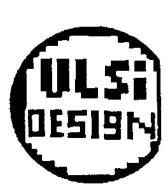
(94) EN = 0



(95) EN = 7



(102) EN = -3



(104) EN = 9

Chapter 3

Euler vector

3.1 Introduction

A compact representation of image features is always needed for efficient management of image database, search and retrieval. Defining a good numerical characterization of an image is a fundamental problem in image processing. Further, the characteristic parameters of the image should preferably remain invariant in the presence of various perturbations or transformations, such as translation, rotation, scaling, rubber-sheet shearing, compression, degradation by compression etc.

Existing feature extraction methodologies for image analysis can be broadly classified into two types - i) geometric features of an object and ii) luminance signature of an object [130, 147].

3.1.1 Geometric Features

Various geometric shapes such as lines, vertices, ellipses etc. are used to provide characteristic features for recognizing an object [20, 27]. Geometric parameters such as edge, corner, line, curve, hole, and boundary curvature define individual features of an image. Several transformations e.g., medial axis transformation, morphological trans-

forms, may be applied for analyzing the structure of the shape patterns. These features and their spatial relations are then combined to generate object descriptions. The geometric features necessarily represent specific higher level 3-D primitives that correspond to physically meaningful properties of the scene. These features are less sensitive to variations than the original noisy gray-level values. Choice of features is obviously application specific. These geometric features can be roughly categorized into three types [27]:

(a) *Global parameters* are features extracted from the image as a whole. Different geometric and topological parameters like perimeter, area, centroid, curvature, Euler number, distribution of contour points from centroid are used. Geometric features calculated from image moments, like center of mass, orientation, bounding rectangle etc., are also used.

(b) *Structural parameters* are the features which are local in nature, each describing a portion of the object. Features like line segment, arc segment with constant curvature, corner specifications etc., that define pieces of an object's boundary are widely in use.

(c) *Relational parameters* represent geometrical relations among local features using graph representations. Distance and relative orientation of substructures and regions of an object are interrelated using mainly graph-based methods.

3.1.2 Luminance Features

In the second category, feature extraction from luminance information of an object is done with respect to its intensity values. Different sorts of features are used based on the intensity values [62].

(a) *Spatial features* are characterized using the gray-levels or colors and their distributions like amplitude and histograms based on different properties of the intensity levels [103, 142].

(b) *Transform features* provide the frequency domain information of the image, obtained by zonal filtering in the selected transform space e.g. Fourier descriptor, DCT. Fourier descriptors [158] for a simple closed curve is used for shape discrimination [104]. Recently, affine invariant functions based on the dyadic wavelet transform of the object boundary has been used for shape discrimination [70].

(c) *Edges and boundaries* characterize object boundaries and shape. These are extracted using different gradient operators. Boundaries characterize shape of an object, and are extracted by linking edges by techniques like contour following, edge linking and heuristic graph searching [45, 62].

(d) *Invariant moments* are certain functions of moments [45, 62] and are invariant under linear coordinate transforms, contrast [59, 118, 119, 144] and have been used for shapes and scene matching applications. Zernike moments of an image are a set of orthogonal complex moments whose magnitudes have been used as features for rotation invariant image recognition [71].

(e) *Texture features* are mostly based on structural, statistical, or spectral properties. Any feature that characterizes texture efficiently must include both statistical and structural information present in the image. This information is implicitly embedded in the local intensity variation patterns of the texture region. There are several methods for texture extraction using gray-level co-occurrence statistics [51], Gabor filters [63], windowed Fourier filters [15], association rules [127].

Content-based image retrieval (CBIR) has become one of the most important research topics spanning disciplines like image processing, computer vision, information retrieval, multidimensional access methods, databases. Also, with the explosive growth of the internet technology, CBIR has become a very important area of focus. Feature extraction pertaining to CBIR has received wide attention. A survey of feature uses in CBIR can be found in [137]. In [148], a good survey of the technical aspects of 39 existing CBIR systems is presented. The features used by the systems are mostly low level features. Determination of a compact set of parameters for a gray-tone image is now highly needed in the emerging domain of the Internet technology, such that it is easy to compute, suitable for efficient database search, and admits robustness against transformations.

In this chapter, we define a new parameter called Euler vector of a gray-tone image. For a binary image, Euler number (genus) is a well-known geometric feature, which is defined as the difference of the number of connected components (objects) and the number of holes [45, 48, 108, 124, 159]. Efficient techniques of computing Euler number of a binary image are also well known [35, 48, 108]. In an attempt to generalize the concept for a gray-tone image, we consider gray code representation of the intensity values in the pixel matrix, and observe the first 4 most significant bit planes. Each bit plane consists of only 0's and 1's and hence forms a binary (two-tone) image. We compute Euler number of the partial image formed by each of the 4 bit planes to obtain a 4-tuple of integers, called Euler vector of the original gray-tone image. Euler vector thus, as a feature of a gray-tone image, uses both geometric and intensity level characterization [130, 147]. Bit planes based on gray codes were also used earlier for other bio-medical applications [80]. The Euler vector of an image is found to remain near invariant under inclusion of salt and pepper or gaussian noise followed by filtering, and also under JPEG compression. We show statistically the need for filtering the image to make the Euler vector robust. It has a strong discriminatory power and can thus be used to augment other features to facilitate image searching and retrieval. The JPEG2000 image compression standard uses a wavelet transform and a bit-plane entropy coder to provide state-of-the-art compression [112, 113]. Thus, the JPEG2000 standard supports multiresolution in terms of scale and embedded quantization by bit-plane coding. Euler vector has the bit-plane representation inherent to its definition and being topologically invariant can support multiresolution in terms of scale. VLSI implementation of Euler number has been studied in the earlier chapter. We extend the run-based method of Euler number computation [124, 159] for the VLSI implementation of Euler vector.

3.2 Euler Vector Computation

3.2.1 Bit-planes and Euler vector

We assume that a gray-tone image be represented as an $(N \times M)$ matrix, where each element is an integer lying between $[0, 255]$ denoting the intensity of the corresponding

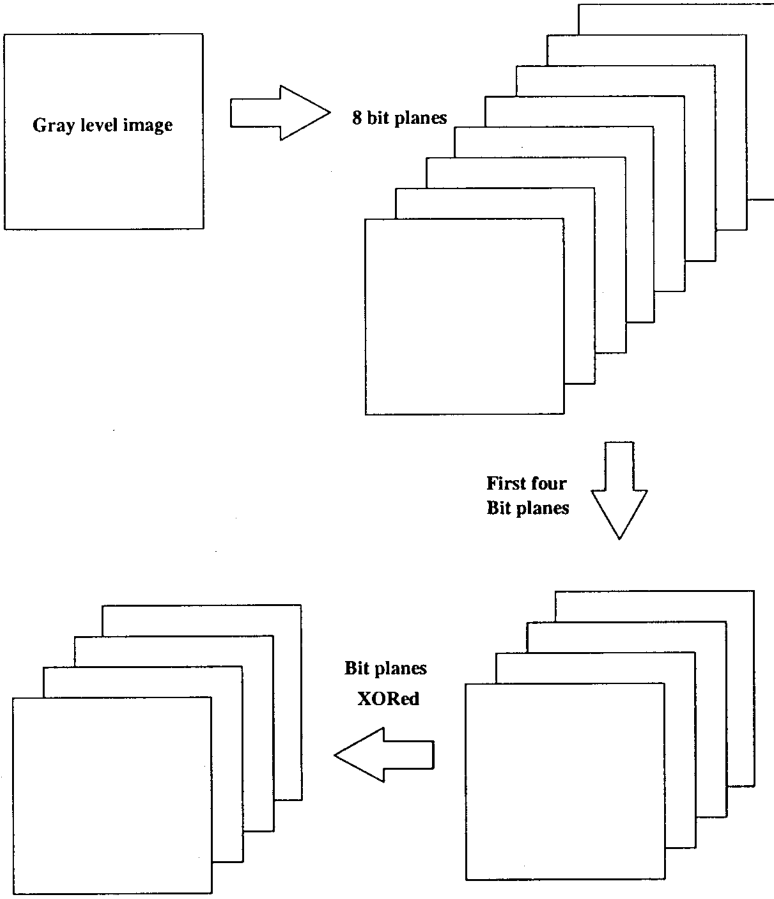
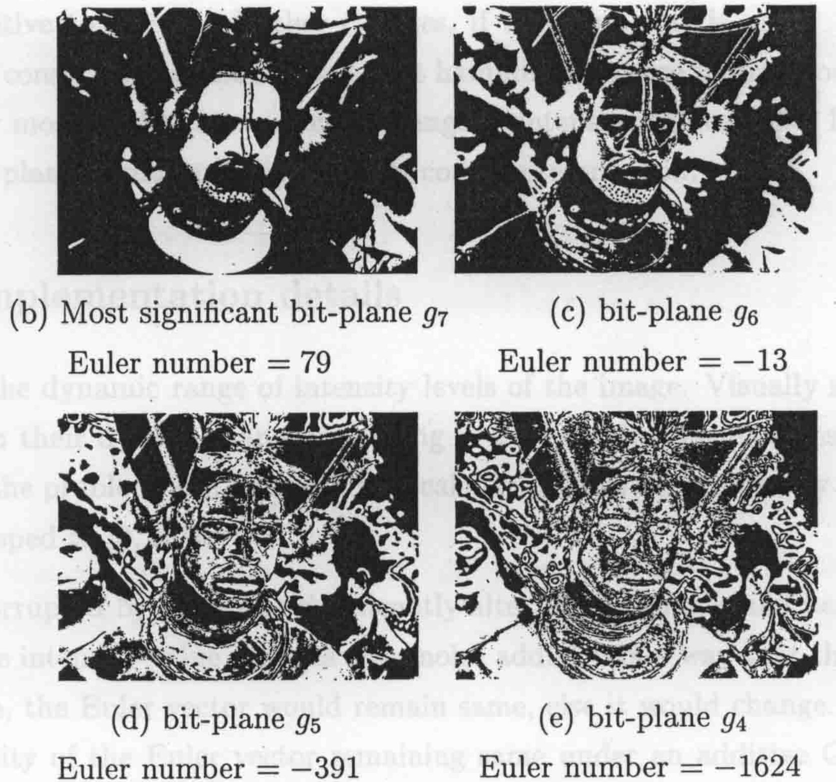


Figure 3.1: Bit planes and Euler vector

pixel. Thus, a 8-bit binary vector, denoted by $(b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)$ can represent the intensity value of each pixel, where each b_i is either '0' or '1'. Thus, the image may be considered as an overlay of 8 bit-planes. See Figure 3.1 for an illustration. Each bit-plane can be thought of as a two-tone image and can be represented by a binary matrix of size $(N \times M)$.

To characterize a gray-tone image, we now define a 4-tuple called Euler vector. We retain the first *4 most significant bit planes* (corresponding to (b_7, b_6, b_5, b_4)) as they contain most of the information of the image, and ignore the remaining planes. However, each of these 4-bit binary vectors is converted to its corresponding reflected gray code (g_7, g_6, g_5, g_4) [77], where: $g_7 = b_7; g_6 = b_7 \oplus b_6; g_5 = b_6 \oplus b_5; g_4 = b_5 \oplus b_4$. Here, $\oplus$ denotes XOR (modulo-2) operation. For any binary vector, the corresponding reflected gray code is unique and vice-versa.

(a) Gray-tone image



(b) Most significant bit-plane g_7 Euler number = 79

(c) bit-plane g_6 Euler number = −13

(d) bit-plane g_5 Euler number = −391

(e) bit-plane g_4 Euler number = −1624

Figure 3.2: Gray-code bit planes and Euler vector = {79, -13, -391, -1624}

Definition: The Euler vector of a gray-tone image is a 4-tuple E_7, E_6, E_5, E_4 where E_i is the Euler number of the partial two-tone image formed by the ith bit-plane, $7 \leq i \leq 4$, corresponding to the reflected gray code representation of intensity values.

For the gray-tone image (africa.g) shown in Figure 3.2, the Euler vector is found to be $\{79, -13, -391, -1624\}$.

Euler vector serves as a fundamental topological feature of a gray-tone image. Like Euler number of a binary image, it remains invariant under translation, rotation, scaling, and rubber-sheet transformation of the image. Since Euler number depends on the

combinatorial properties of 0-1 runs in the binary pixel matrix [124, 159] and is easily computable, the Euler vector also provides a quick combinatorial signature of a gray-tone image.

Reflected gray code representation of intensity values offers a distinct advantage over standard binary representation in this particular context. Euler vector is found to be more insensitive to noise and other changes, if the gray code is used. This happens because two consecutive numbers have unit hamming distance in gray-code representation, and for most of the cases, a small change in intensity values is not likely to affect all the 4 bit planes simultaneously in gray code representation.

3.2.2 Implementation details

We rescale the dynamic range of intensity levels of the image. Visually similar images may differ in their dynamic ranges resulting in different bit-plane representations. To circumvent the problem, the images are rescaled such that their intensity level dynamic range is mapped to $[0, 255]$.

An image corrupted by noise can significantly alter the bit-planes and hence, the Euler vector. If the intensity value changes after noise addition in a way that the bit-patterns remain same, the Euler vector would remain same, else it would change. We calculate the probability of the Euler vector remaining same under an additive Gaussian noise model. The additive Gaussian noise model is

$$g(i,j) = f(i,j) + \eta(i,j); \text{where } 0 \leq f(i,j) \leq 255, 0 \leq g(i,j) \leq 255$$

where $f(i,j)$ and $g(i,j)$ are respectively the original and degraded gray-level intensity of the pixel at $(i,j)^{th}$ location, and $\eta(i,j)$ is the additive noise at that location. Note that, the noise η follows Gaussian distribution; i.e. $\eta \sim N(\mu, \sigma)$ where μ and σ are the mean and standard deviation of the noise distribution. For calculating the Euler vector, we consider the first *4 most significant bit planes.* The binary and its equivalent Gray-code representation has consecutive unchanged bit patterns for the 4 most significant bits from 16 to 255 at intervals of 2^4. The bit-patterns for binary values in the ranges of $r_1 = [16, 31]$ (bit-pattern=0001); $r_2 = [32, 47]$, (bit-pattern=0010), ..., $r_{15} = [240, 255]$,

(bit-pattern=1111) remain same. The bit-pattern of the degraded value $g(i,j)$ would remain same as that of the original $f(i,j)$ if the value of $g(i,j)$ lies in the same range as $f(i,j)$. For a given $f(i,j)$, we first find out the range r_k ($k = 1, 2, \ldots, 15$) in which it lies, by computing k as follows:

$$k = \lfloor f(i,j)/16 \rfloor.$$

The intensity value $f(i,j)$ has the same bit-pattern as any other intensity value in the following range:

$$\begin{aligned} & k \times 16 \leq f(i,j) \leq k \times 16 + 15 \\ or \quad & m \leq f(i,j) \leq m + 15, \text{ where } m = k \times 16. \end{aligned}$$

Now, the bit-pattern of the degraded pixel would be same as $f(i,j)$ if

$$\begin{aligned} & m \leq g(i,j) \leq m + 15 \\ or \quad & m \leq f(i,j) + \eta(i,j) \leq m + 15 \\ or \quad & m - f(i,j) \leq \eta(i,j) \leq m - f(i,j) + 15. \end{aligned}$$

Now, $\eta \sim N(\mu, \sigma)$. Probability, p_{ij} that $g(i,j)$ has the same bit-pattern for the 4 most significant bits as $f(i,j)$ is same as the probability that $\eta(i,j)$ lies in the range $[m - f(i,j), m - f(i,j) + 15]$:

$$\begin{aligned} &= \int_{m-f(i,j)}^{m-f(i,j)+15} \frac{1}{\sqrt{2\pi}\sigma} \exp^{\frac{-1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} dx \\ &= \int_{\frac{m-f(i,j)-\mu}{\sigma}}^{\frac{m-f(i,j)-\mu+15}{\sigma}} \frac{1}{\sqrt{2\pi}} \exp^{\frac{-z^2}{2}} dz \\ &= \Phi\left(\frac{m - f(i,j) - \mu + 15}{\sigma}\right) - \Phi\left(\frac{m - f(i,j) - \mu}{\sigma}\right) \end{aligned}$$

We assume that the pixels are independently degraded i.e., pixels are degraded by noise independent of their positions. Therefore, the probability of the event that all pixels in the image-pixel matrix have no alteration in their first 4 most significant bit planes is

$$P = \prod_{i=1}^{N} \prod_{i=1}^{M} p_{ij}.$$

So, the probability $\bar{P}$ that the 4 most significant bit-patterns change is the complement of the above event and is

$$1 - P = 1 - \prod_{i=1}^{N} \prod_{i=1}^{M} p_{ij}.$$

Note that $P \ll 1$, so $\bar{P} \approx 1$. Similar deduction and observation are true for other types of noise distributions. Therefore, the probability that the bit-planes undergo change in presence of noise is significantly high. Euler vector derives its definition from bit-planes, and with change in the pattern of bit-planes, Euler vector might undergo significant change. So, to make the Euler vector more robust, the given image is cleaned successively by median and mean filters. Other sophisticated filters may also be used.

We now describe the proposed algorithm.

Algorithm EulerVector(I)
Input: A pixel matrix for a gray-tone image I
Output: Euler vector.

1. Apply median filtering followed by mean filtering on I;
2. Linearly rescale the dynamic range of the image to [0-255];
3. Consider the first 4 most significant binary bit planes of I;
4. Convert each 4-bit vector to its corresponding reflected gray code;
5. For each bit plane, compute Euler number;
6. Output the Euler Vector;

3.3 Mahalanobis distance as a measure of similarity

While characterizing gray-tone images by its Euler vector, we have observed that the ranges and the variances of various elements of the vector widely differ. In such an environment, the Mahalanobis distance [10, 89] can be adopted to provide a very good measure that captures similarity/dissimilarity properties among the members of a given set of images.

3.3.1 Theme

Let $\tilde{x}' = (x_1, x_2, x_3, \ldots, x_d)$ and $\tilde{y}' = (y_1, y_2, y_3, \ldots, y_d)$ be two vectors in $I\!R^d$. Suppose, our goal is to define a distance measure between $\tilde{x}$ and $\tilde{y}$ that captures the 'closeness' of $\tilde{x}$ and $\tilde{y}$. Conventionally, we define distance measures between $\tilde{x}$ and $\tilde{y}$ as:

$$L_p(\tilde{x}, \tilde{y}) = (\sum_{i=1}^{d} |x_i - y_i|^p)^{\frac{1}{p}}$$

In particular, when $p = 2$, L_p denotes the usual Euclidean distance between $\tilde{x}$ and $\tilde{y}$. Let a phenomenon or an instance be represented by d features namely $x_1, x_2, \ldots, x_d$. Each feature x_i takes a value between a_i and b_i, i.e., $a_i < x_i < b_i$, $\forall i = 1, 2, \ldots, d$, where a_i and b_i are real numbers. Let $\tilde{x}$ and $\tilde{y}$ be two realizations of the said phenomenon, the distance of which is to be measured. Note that the feature with a larger variance generally contributes more to $L_p(x, y)$ than the feature with a smaller variance. This property is desirable in many situations but not in all the cases. In some problems, we may want to give more importance to a feature with smaller variance than a feature with larger variance. Thus, for the case of Euclidean distance, the following distance measure may be appropriate:

$$d_1(\tilde{x}, \tilde{y}) = \sqrt{\sum_{i=1}^{d} w_i (x_i - y_i)^2}, \text{where } w_i > 0.$$

Secondly, the features need not always be uncorrelated. In fact, in many real life situations, the features under consideration are in general correlated. Thus, a desirable distance measure should give weights according to the correlation between the concerned variables. In other words, a suitable distance measure would be given by:

$$d_2(\tilde{x}, \tilde{y}) = \sqrt{\sum_{i=1}^{d} \sum_{j=1}^{d} w_{ij} (x_i - y_i)(x_j - y_j)}, \text{where } w_{ij} > 0.$$

One way of obtaining such w_{ij}'s is to consider a $(d \times d)$ positive definite matrix Γ and taking $d_2(\tilde{x}, \tilde{y})$ as $\sqrt{\{(\tilde{x} - \tilde{y})'\Gamma(\tilde{x} - \tilde{y})\}}$. Let $\tilde{x}' = (x_1, x_2, \ldots, x_d)$ be a vector in $I\!R^d$ denoting d features (variables). Corresponding to each feature, x_i, we have its variance and corresponding covariances with other members in $\tilde{x}$. The variance-covariance matrix

of $\tilde{x}$, denoted as Λ, is a $(d \times d)$ matrix given by:

$$\begin{aligned} \Lambda(i,j) &= \text{variance of } x_i, && \text{if } i = j; \\ &= \text{covariance of } x_i \text{ and } x_j, && \text{if } i \neq j. \end{aligned}$$

We may use the matrix Λ as the weight matrix Γ and compute the distance as follows:

$$\Delta = \sqrt{(\tilde{x} - \tilde{y})'\Lambda^{-1}(\tilde{x} - \tilde{y})}$$

Λ is also known as dispersion matrix. If $\tilde{x}$ and $\tilde{y}$ denote the mean vector of two populations then Δ is called the Mahalanobis distance between those two populations [10, 89].

3.3.2 Application of Mahalanobis distance to image ranking

In a large database of images, the variance-covariance matrix defined by feature vectors of the images may be taken as the population dispersion matrix. If the number of images in the database is large, removal of a few images will not have a significant impact on the entries of the variance-covariance matrix. The variance-covariance matrix can also be incrementally updated when new images are added to the database. So, the suggested distance measure for our purpose is:

$$D = \sqrt{\{(\tilde{x} - \tilde{y})'\Lambda^{-1}(\tilde{x} - \tilde{y})\}}$$

where Λ is the variance covariance matrix formed by the image feature vectors. Since, Λ is positive definite, it is easy to show that, for a fixed Λ, D is indeed a metric. Such a distance measure satisfies the following: (i) a feature with large variance does not contribute more to the distance value, and (ii) the correlation among variables affects the distance measure. Therefore, it fits very well in the proposed environment of image retrieval based on Euler vector as the feature.

3.4 Results

We have coded the algorithm for Euler vector computation in C and run on Sun Ultra-5_10, Sparc; the OS is SunOS Release 5.7 Generic. A database consisting of several

gray-tone images is considered. Samples are shown in Figure 3.6. For each of these images, we computed the Euler vector after cleaning them by median and mean filters. The effects of adding salt and pepper or gaussian noise, and JPEG compression have been studied. It has been observed that Euler vector provides a quick signature with robust behavior in the presence of noise and compression.

3.4.1 Euler Vector values

Each image is of size (480×640). For each image shown in Figure 3.6, the *Euler vector* $\{E_7, E_6, E_5, E_4\}$ is shown in Table 3.1, the top one corresponds to E_7. The images were filtered successively by median and mean filters each of size (3×3). The *salt & pepper* noise was a random noise which affected 5% of the image (= 15360 pixels approximately). The gaussian noise is a white noise with mean = 0 and variance = 0.05.

3.4.2 Mahalanobis Distance and Ranking

The Euler vectors for the original, noisy and compressed images are given in Table 3.1 for the 31 images stored in the database. Here, we have $d = 4$, and for each feature vector, x_1 corresponds to E_7, x_2 corresponds to E_6, x_3 corresponds to E_5, and x_4 to E_4. The variance covariance matrix Λ, for the Euler vector would be a 4×4 matrix computed from the values of E_7, E_6, E_5 and E_4 of the original images as given in Table 3.1, and is as follows:

$$10^6 \times \begin{pmatrix} 0.0182 & 0.0085 & 0.1602 & 0.0351 \\ 0.0085 & 0.0436 & 0.2966 & 0.2139 \\ 0.1602 & 0.2996 & 3.8035 & 1.3227 \\ 0.0351 & 0.2139 & 1.3227 & 2.0762 \end{pmatrix}.$$

The inverse of the above matrix, Λ^{-1} is:

$$10^{-4} \times \begin{pmatrix} 0.9483 & 0.2066 & -0.0556 & -0.0019 \\ 0.2066 & 0.8421 & -0.0561 & -0.0545 \\ -0.0556 & -0.0561 & 0.0091 & 0.0009 \\ -0.0019 & -0.0545 & 0.0009 & 0.0099 \end{pmatrix}.$$

Consider the Euler vector $\{79, -13, -391, -1624\}$ for the original image *africa.g* (see Figure 3.6). The Euler vector for the *salt & pepper* corrupted version of *africa.g* is $\{78, -13, -406, -1624\}$. The Mahalanobis distance between these two vectors according to the proposed measure is:

$$\begin{aligned} D_1 &= [(79-78), (-13-(-13)), (-391-(-406)), (-1624-(-1624))] \cdot \Lambda^{-1} \cdot [(79-78), \\ &\quad (-13-(-13)), (-391-(-406)), (-1624-(-1624))]' \\ &= 1.3195 \times 10^{-4} \end{aligned}$$

The Euler vector of the original image *army.g* (see Figure 3.6) is $\{82, -150, -953, -1543\}$. The distance between the Euler vectors of the original *africa.g* and *army.g*, according to the proposed distance measure is:

$$\begin{aligned} D_2 &= [(79-82), (-13-(-150)), (-391-(-953)), (-1624-(-1543))] \cdot \Lambda^{-1} \cdot [(79-82), \\ &\quad (-13-(-150)), (-391-(-953)), (-1624-(-1543))]' \\ &= 1.1245 \end{aligned}$$

It may be observed that $D_2 >> D_1$, which demonstrates that the distance between two similar images is much less than that between two dissimilar images.

We performed an experiment with the Mahalanobis distance measure for ranking the images with respect to a query image. For, a database of 31 gray-tone images, we have computed the Euler vectors for the original, noise corrupted and jpeg compressed images. Thus, we have $31 \times 4 = 124$ images in our database. We computed the variance covariance matrix Λ using the Euler vectors of the original images. Given any query image (the query image is also from the image database), we have calculated the Mahalanobis distance between it and each of the 124 images and ranked them according to the descending order of the Mahalanobis distance. The *closest* image i.e. the one

whose distance is the smallest from the query image is given the first rank, and so on. Using Euler vector as a representative image feature and Mahalanobis distance as a closeness measure, the desired image is always found to lie at minimum distance. Noisy and compressed versions of the query image have next higher ranks in most of the cases. We report experimental results in support in Table 3.2 where e.g., the images *africa.gauss*, *africa.salt*, and *africa.jpg* are obtained respectively by adding gaussian noise, salt and pepper noise and by applying jpeg compression. The corresponding Mahalanobis distance of the degraded image from the query image is shown in Table 3.2 in paranthesis below the name of the retrieved image. The query image itself always has rank-1, and is not shown.

The various features used to characterize an image may not be equally important for retrieval. In such a case, the elements of the variance-covariance matrix may be multiplied by a suitable weight factor according to the importance for the corresponding feature. For a large database, computing the distance from a query image to all other images in the database may take substantial time. Thus, ranking and in turn, retrieval will become slow. To obviate the problem, multidimensional search techniques are employed to retrieve a subset of images lying in the neighborhood of the query image in the feature space. Ranking according to the proposed measure can then be performed on this subset only.

3.4.3 K-Nearest Neighbor Classifier

Database Used for K-NN Classifier

The image database we have used is a standard image database known as Columbia Object Image Library (COIL-20) [97]. It is a database of gray-scale images of 20 objects. The objects were placed on a motorized turntable against a black background. The turntable was rotated through 360° to vary object pose with respect to a fixed camera. Images of the objects were taken at pose intervals of 5°. This corresponds to 72 images per object. The objects have a wide variety of complex geometric and reflectance characteristics. The experimental setup used for image acquisition includes a CCD

camera (SONY XC77) with a 25mm lens which was fixed to a rigid stand about 1 ft. from its base. A motorized turntable was placed about 2 ft. from the base of the stand. The camera was tilted down at about 25° to point towards the turntable. This way most objects appeared at the center of the image when placed at the center of the turntable. To avoid strong shadows, only ambient(fluorescent) room lighting was used. A black background was provided by covering the turntable and visible background surfaces with black cloth. Each object was placed in a stable configuration at approximately the center of the turntable. The turntable was then rotated through 360° and 72 images were taken per object; one at every 5° of rotation. The images were digitized using an Analogics grayscale frame grabber. The database we work with in this and the following chapter contains 1,440 grayscale images of all the 20 objects and have been size normalized as follows. The object is clipped out from the black background using a rectangular bounding box. The bounding box is resized to 128×128 using interpolation-decimation filters to minimize aliasing. In the Figure 3.3, the frontal poses of each of the 20 objects are shown. We performed a classification of the above data set using K-Nearest Neighbor classifier [145]. The experiment was performed for different sets of training data set size T, with varying K for a 20 class problem (as because there are 20 different objects). The graph showing misclassification percentage against K is given in Figure 3.4 for different values of T. For each object in the COIL database [97], the Euler vectors for different orientations were computed. Now, to form the training set we picked up objects having orientations with regular intervals. By varying the intervals, different training sets are obtained. As an example, for the graph with $T = 240$ in Figure 3.4, for each object we calculated the Euler vector at an interval of 30°. So for each object, we have 12 such Euler vectors ($360/30 = 12$), thus making the training set size 240 for 20 such objects ($20 \times 12 = 240$). Note that, with increasing T, we include in the training data set Euler vectors at shorter intervals of angles. Thus, classification error reduces at higher values of T. The distance between two Euler vectors is measured using Mahalanobis distance measure. The variance-covariance matrix used for this purpose is calculated from the entire data set.

Figure 3.3: COIL objects

3.5 VLSI implementation

Having designed the on-chip implementation of Euler number, we can simply extend it for Euler vector calculation. A sample circuit is shown in Figure 3.5 for a $N \times M$ image, whose intensity value of each pixel lies in the range $[0, 255]$ and hence represented by 8-bits. For each gray-code bit-plane we use the circuit block of Euler number calculation which is discussed in the previous chapter. Thus, we would need 4 such blocks as shown in Figure 3.5 with some XOR gates for reflected gray-code computation. It may be noted that the run-based computation proposed in the previous chapter can be used straight way. The circuit blocks C_1, C_2, C_3 and C_4 perform the Euler number computation of bit-planes in parallel. The time complexity for finding the Euler Vector would be same as that of Euler number implementation save except a delay due to the XOR-gates used. The circuit cost would be four times that of Euler number implementation (see Section 2.3.4) with extra $3 \times N$ 2-input XOR gates.

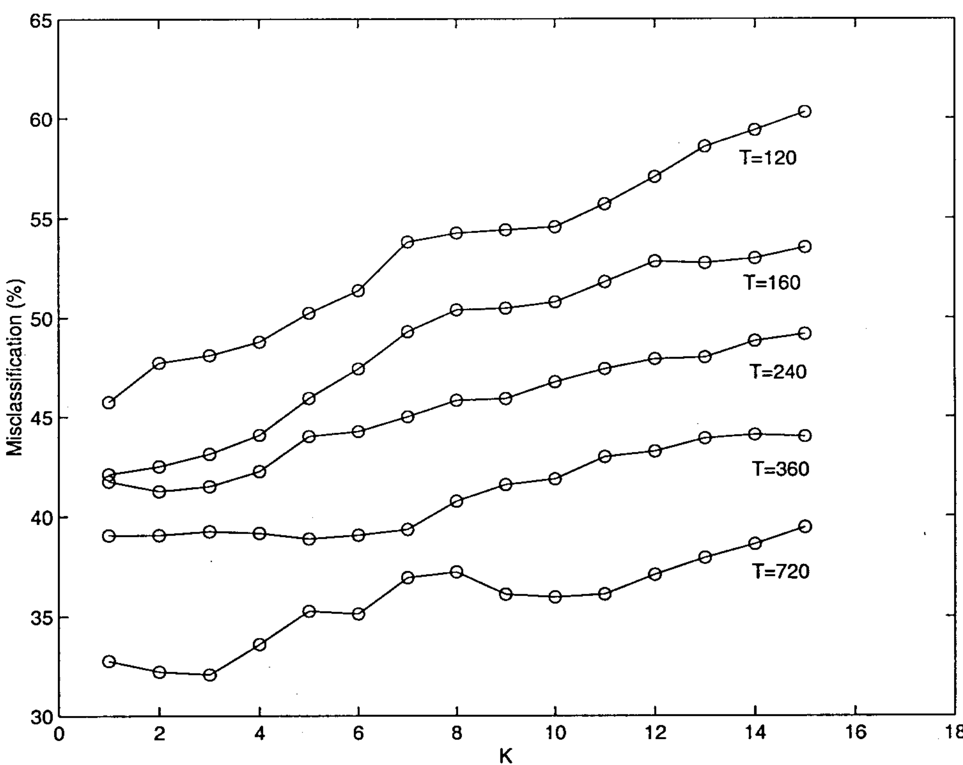


Figure 3.4: K vs. % Misclassification for K-NN classifier

3.6 Conclusions

The upcoming JPEG2000 image compression standard [113] uses Discrete Wavelet Transform (DWT) followed by uniform deadzone quantizer. The multi-resolution image representation is inherent to DWT. JPEG2000 supports embedded quantization by bit-plane coding. Euler vector being topologically invariant can support multiresolution scale and the bit-plane representation is inherent to its definition. For feature extraction of images in the compressed domain, bit-plane representation and feature extraction in those planes can be attempted. Further investigation is needed to ascertain whether the same feature extracted in different frequency subbands can be collated to formulate the feature for the original uncompressed image e.g., if the Euler vector of the original image can be deduced from the subband values (LL, LH, HL, HH). In this chapter, we have used simple reflected gray-codes to define Euler vector. Use of other sorts of symmetric balanced gray-codes for noise immunity may be considered.

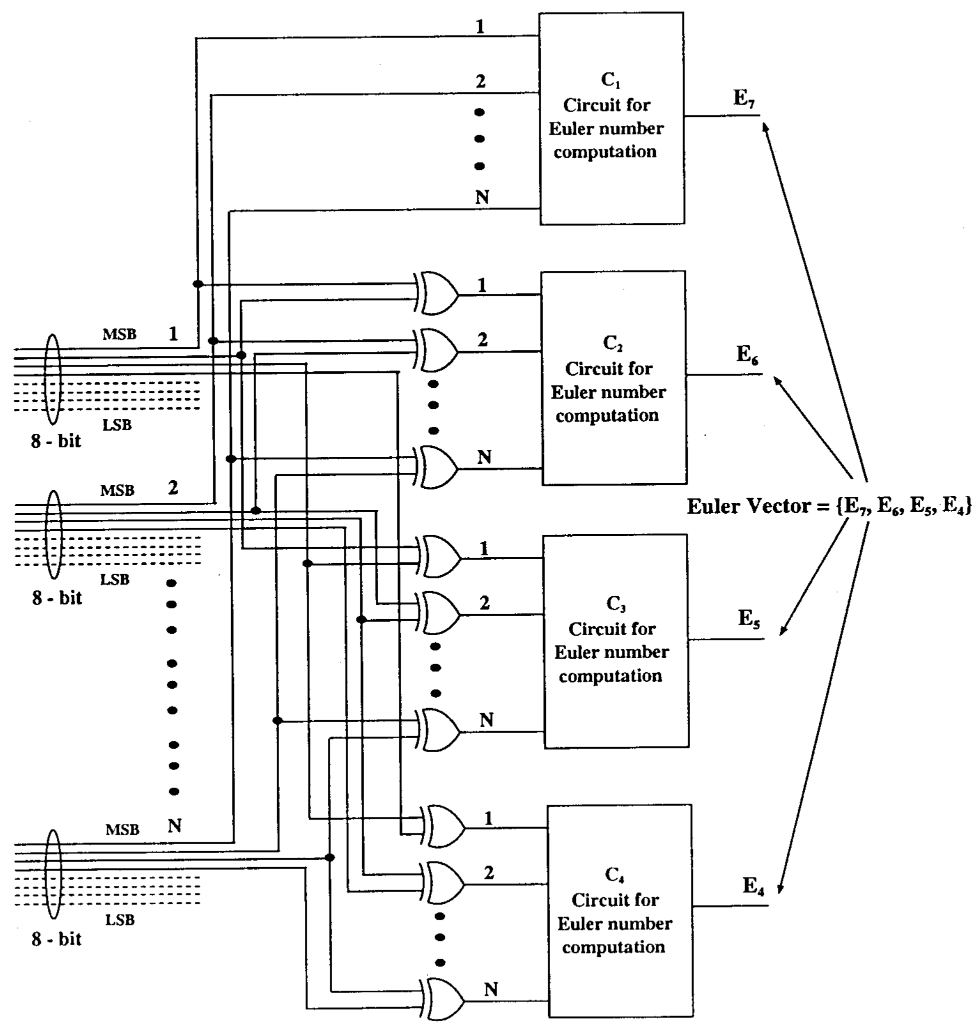


Figure 3.5: Sample Circuit for Euler Vector Computation

Table 3.1: Computation of Euler vector

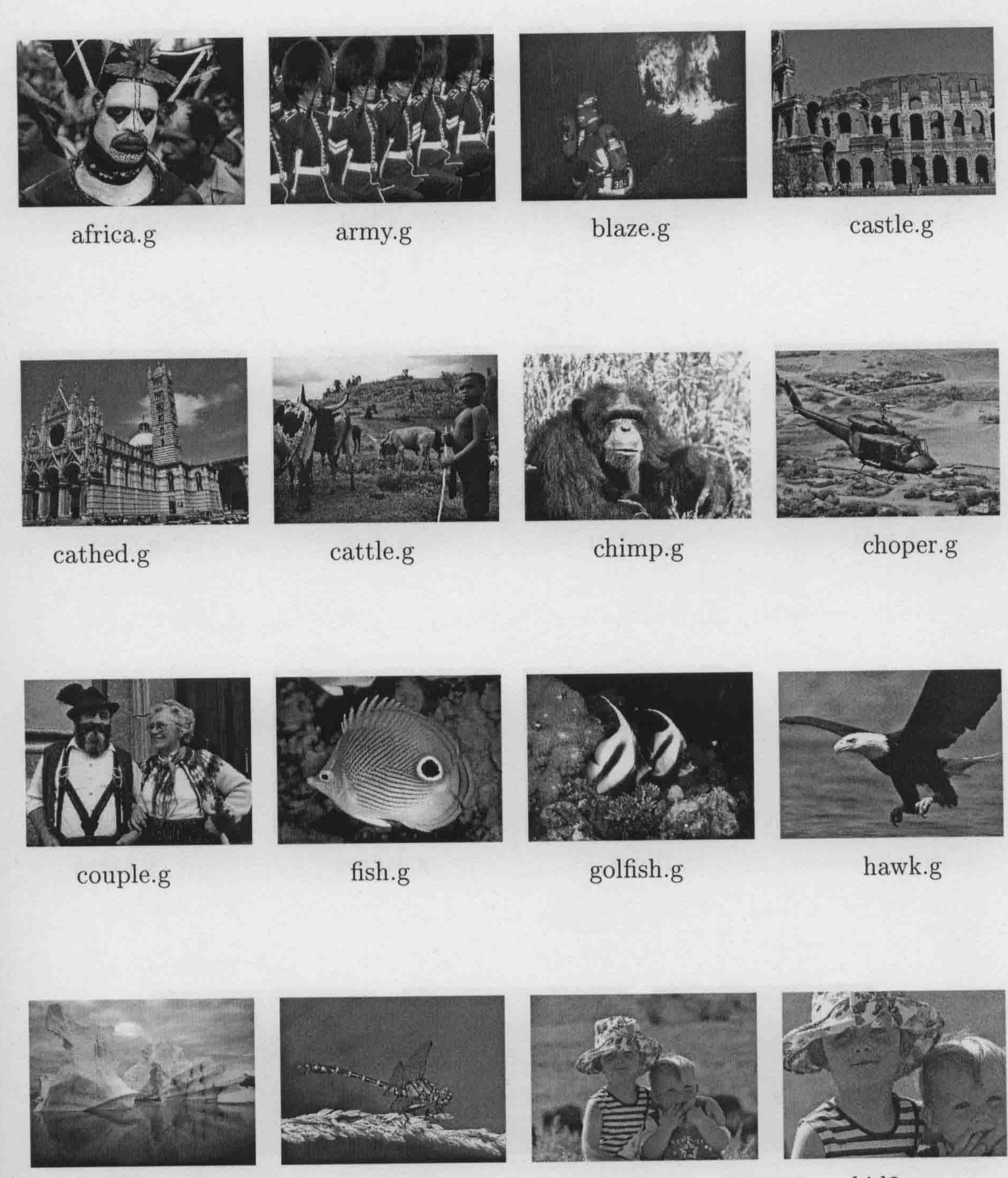
Image name	Original image	Image corrupted by salt & pepper noise	Image corrupted by gaussian noise	JPEG compressed image
africa.g	79 -13 -391 -1624	78 -13 -406 -1624	83 3 -364 -1484	74 -10 -403 -1608
army.g	82 -150 -953 -1543	83 -145 -923 -1562	88 -85 -976 -1856	86 -167 -833 -1644
blaze.g	21 -44 -250 -880	20 -50 -245 -857	16 55 -1089 -1576	22 -41 -201 -974
castle.g	135 -297 -259 -1748	115 -341 -206 -1765	392 -360 177 -1657	121 -315 -172 -1768
cathed.g	401 -393 -429 -1840	407 -358 -376 -2097	441 -760 -174 -1858	405 -367 -368 -1753
cattle.g	116 -91 -579 -1119	116 -97 -579 -1103	155 -192 -672 -1388	120 -121 -528 -1051
chimp.g	-67 -82 -530 -1737	-72 -79 -557 -1550	-85 -113 -589 -1564	-73 -93 -526 -1711
choper.g	53 -201 141 -1155	58 -205 117 -1207	12 -279 222 -1303	37 -194 129 -1123
couple.g	76 -75 -480 -1882	80 -76 -479 -1894	46 -122 -78 -2016	84 -65 -443 -1727
fish.g	20 -122 -117 -972	19 -123 -111 -978	29 -89 -182 -773	20 -118 -106 -970
goldfish.g	135 -2 -331 -842	140 -4 -312 -885	150 -51 -306 -854	137 -2 -294 -852
hawk.g	42 0 -49 -247	36 4 -46 -268	277 -20 225 -469	33 -2 -43 -230
ice.g	-10 -77 7 -10	-12 -74 6 -26	-57 -223 -163 -187	-8 -66 6 -4
insect.g	13 -185 -287 -1431	11 -186 -284 -1442	12 -402 -731 -1736	16 -161 -268 -1342
kid1.g	42 -80 -102 -539	50 -75 -104 -550	72 -112 -205 -35	23 -81 -106 -474

Image name	Original image	Image corrupted by salt & pepper noise	Image corrupted by gaussian noise	JPEG compressed image
kid2.g	13 -86 -31 -734	11 -78 -47 -746	-316 -181 -274 -433	9 -84 -56 -586
kid3.g	11 -62 -143 -281	7 -57 -154 -278	-2 -274 -194 -184	8 -50 -140 -295
leaf.g	-44 -554 -337 -6735	-38 -591 -317 -6363	-94 -1174 145 -4284	-47 -564 -279 -6568
neweng.g	201 2 -376 -471	216 14 -412 -507	207 33 -514 -412	206 21 -384 -305
photogra.g	143 31 -480 -2510	140 40 -496 -2483	126 84 -410 -2684	150 51 -470 -2396
rgb3.g	206 -176 -779 -3138	202 -164 -800 -3079	186 -265 -777 -2844	183 -162 -749 -3063
rock.g	-15 1 -45 -371	-7 3 -129 -392	-66 49 -389 -533	-10 3 -124 -359
rodeo.g	130 -229 -330 -1548	127 -233 -336 -1531	179 -306 -370 -1673	138 -231 -346 -1542
rose.g	1 138 -254 -42	1 176 -255 -45	4 1581 -2238 317	1 140 -294 -45
santa.g	-2 -7 -127 -646	0 -13 -132 -614	26 -36 1069 -2135	1 -15 -127 -641
seafish.g	81 -20 -197 -373	82 -21 -210 -369	108 -26 -536 -580	81 -14 -221 -458
soldir.g	316 -302 35 -1038	309 -305 24 -1174	337 -326 119 -1147	319 -291 10 -1025
star.g	-400 -979 -11056 -4675	-436 -1144 -10783 -3784	-440 -1252 -11018 -5509	-399 -993 -11065 -5574
town.g	154 -330 -347 -2004	157 -349 -331 -2091	258 -376 -521 -2027	170 -334 -317 -1896
wave.g	42 -166 -8 -34	38 -168 0 -24	29 -295 29 107	54 -156 14 32
zebra.g	205 -168 -931 -3371	216 -177 -927 -3401	213 -245 -900 -3335	205 -186 -898 -3350

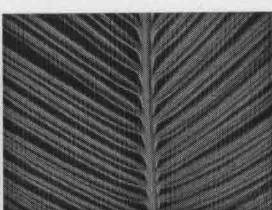
Table 3.2: Ranking of images using Mahalanobis distance, for retrieval

Query Image	Rank-2	Rank-3	Rank-4
africa.g	africa.salt (0.000132)	africa.jpg (0.002100)	africa.gauss (0.002109)
army.g	army.gauss (0.000457)	army.salt (0.002498)	army.jpg (0.042833)
blaze.g	blaze.gauss (0.001931)	blaze.salt (0.005846)	blaze.jpg (0.011915)
castle.g	castle.gauss (0.003506)	town.g (0.071703)	town.gauss (0.089862)
cathed.g	cathed.jpg (0.029282)	cathed.salt (0.254711)	cathed.gauss (0.288829)
cattle.g	cattle.salt (0.004331)	cattle.gauss (0.030318)	cattle.jpg (0.116950)
chimp.g	chimp.gauss (0.002323)	chimp.jpg (0.020971)	chimp.salt (0.030358)
choper.g	choper.salt (0.004405)	choper.gauss (0.004451)	choper.jpg (0.021405)
couple.g	couple.salt (0.001431)	couple.gauss (0.011573)	couple.jpg (0.019001)
fish.g	fish.salt (0.000348)	fish.jpg (0.000884)	fish.gauss (0.006540)
goldfish.g	goldfish.jpg (0.000833)	goldfish.gauss (0.001094)	goldfish.salt (0.002806)
hawk.g	hawk.salt (0.005134)	hawk.gauss (0.006204)	hawk.jpg (0.010263)
ice.g	ice.salt (0.001668)	ice.jpg (0.010934)	ice.gauss (0.020496)
insect.g	insect.salt (0.000639)	insect.gauss (0.006914)	insect.jpg (0.031650)
kid1.g	kid1.salt (0.010878)	kid1.gauss (0.020136)	kid1.jpg (0.039532)
kid2.g	kid2.salt (0.007635)	kid2.gauss (0.011871)	kid2.jpg (0.019446)
kid3.g	kid3.salt (0.002877)	kid3.gauss (0.005815)	kid3.jpg (0.013197)
leaf.g	leaf.jpg (0.069744)	leaf.gauss (0.195873)	leaf.salt (0.403987)
neweng.g	neweng.jpg (0.031120)	neweng.salt (0.059364)	neweng.gauss (0.072748)
photogra.g	photogra.salt (0.005891)	photogra.gauss (0.006318)	photogra.jpg (0.029050)
rgb3.g	rgb3.salt (0.009521)	rgb3.gauss (0.011049)	zebra.jpg (0.032991)
rock.g	rock.gauss (0.002218)	rock.jpg (0.014613)	rock.salt (0.024101)
rodeo.g	rodeo.salt (0.003287)	rodeo.gauss (0.005596)	rodeo.jpg (0.007170)
rose.g	rose.jpg (0.002780)	rose.salt (0.123275)	rose.gauss (0.388863)
santa.g	santa.gauss (0.005051)	santa.jpg (0.005706)	santa.salt (0.005761)
seafish.g	seafish.salt (0.000337)	seafish.gauss (0.001456)	seafish.jpg (0.018242)
soldir.g	soldir.jpg (0.015421)	soldir.salt (0.018868)	soldir.gauss (0.023998)
star.g	star.gauss (0.039673)	star.jpg (0.676613)	star.salt (5.787396)
town.g	town.gauss (0.004213)	town.salt (0.021282)	town.jpg (0.035970)
wave.g	wave.salt (0.003125)	wave.jpg (0.019143)	wave.gauss (0.034353)
zebra.g	zebra.gauss (0.006244)	zebra.salt (0.012181)	zebra.jpg (0.039619)

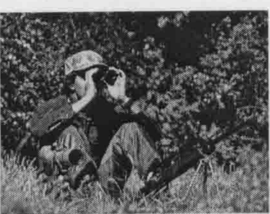
Figure 3.6: Some Sample Images

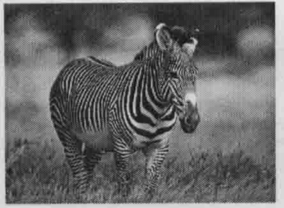


kid3.g


leaf.g

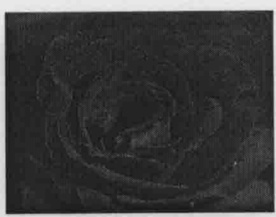
neweng.g


photogra.g

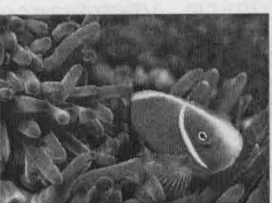

rgb3.g

rock.g

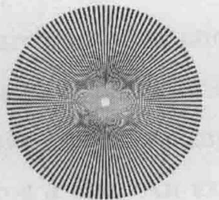
rodeo.g


rose.g

santa.g


seafish.g

soldir.g


star.g

town.g

wave.g

zebra.g

Chapter 4

Image Retrieval using Euler Vector

4.1 Introduction

With the emergence of multimedia systems, images are being processed, stored, transmitted and searched like never before. The data volume for such applications is enormously large. Typical application domains include geographic information systems, art galleries and museum management, scientific database management, trademark and copyright databases, fingerprint and face images for access control, architectural and engineering design etc. [49]. Such wide range of applications of storage and retrieval has fueled the development of content based image retrieval (CBIR) systems. Existing systems implementing CBIR include QBIC [98], VIRAGE [16], to name a few. An extensive overview and up-to-date survey of different aspects of CBIR using still images, appeared in [137]. In this survey, around 200 papers related to this field have been reviewed by Smeulders et al. Several image processing methods used for content or feature extraction in terms of shape, color and texture have been discussed. Interpretation of images by their content, similarity between pair of images, indexing are also reviewed. Further, query definition, display and user interaction with the retrieval system have also been discussed. Veltkamp and Tanase [148] summarized 39 existing retrieval systems in terms of features, index structures, query formulation, matching of a query image in a database, delivering the retrieved data to the user, and relevance feedback.

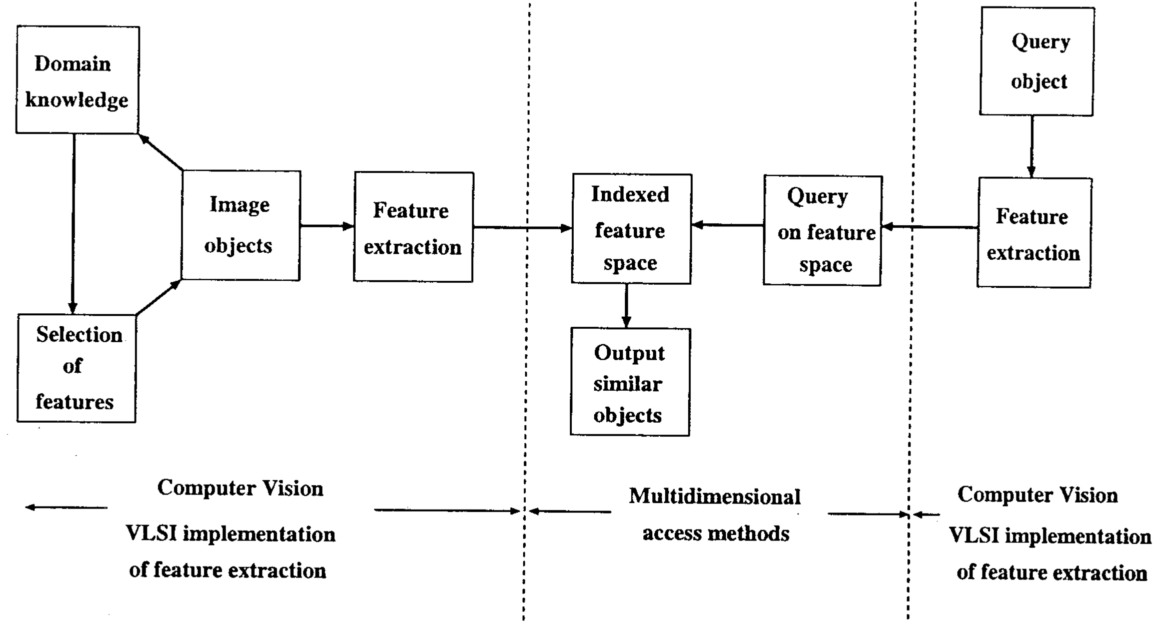


Figure 4.1: A simple illustration of basic components of a retrieval system

Relevance feedback is a tool to refine the search of the query image. The user provides positive or negative feedback about the retrieved results that is again used for refining the search. Cox et al. [30] developed an image retrieval system namely *PicHunter* using Bayesian rules for relevance feedback. For combined feature usage of color and shape invariants, Gevers and Smeulders have developed a system called *PicToSeek* [44]. Shape based retrieval with trademark databases have been studied in [66]. For other surveys on image retrieval issues, refer [37, 147].

In a CBIR system, given a query image and a database, the objective is to return images that are similar to the query image. The problem is easier said than done. Since, images in general require huge memory space, matching it with another image at the pixel level is infeasible. An effective procedure is based on comparing feature representations. Some finite number (say d) of features (e.g. Euler number, moments, texture, shape etc.) are extracted from the image. The image is then mapped as a point in a d-dimensional feature space. Given a query image, the same features are extracted to represent it as a point in the same space. If the feature set is chosen correctly using domain knowledge, the neighbors lying within a certain range of the

query image in the multidimensional space are likeley to be similar to the query image. CBIR systems involve a fusion of computer vision with multidimensional access methods [137]. Computer vision techniques use domain knowledge of image databases for selecting features. The process can be expedited using on-chip implementation of feature extraction (Figure 4.1). A naive method for retrieval involves calculating the distance of all images from the query image and then selecting the closest ones. The number of images employed in the early generation retrieval systems was in the range of 1000-10,000 [98]. A simple linear or optimized linear search to perform a query was adequate. With n images and each being represented by d dimensions (features), the search time is $O(nd)$. However, with the advent of the Internet and digital libraries [93], searching has become a challenging task because $O(nd)$ would be too high. One way to circumvent this is to index the multidimensional feature space with suitable data structures that support similarity-based query of multidimensional feature vectors [155]. Similarity queries correspond to nearest neighbor queries and range queries [98]. Nearest neighbor queries, or their variants, the approximate nearest neighbor queries [13] report the feature vectors that are closest to a query based on some distance measure. The most commonly used metric is the Euclidean distance or weighted Euclidean distance [155]. Multidimensional access methods, have received attention in visual information systems. A good survey of such access methods, including multidimensional hashing can be found in [41]. There exists a vast literature on multidimensional data structures [17, 18, 19, 50, 69, 122, 131, 138, 155]. Index structures supporting similarity queries are observed to perform poorly if the structure does not fit in the main memory. So, there have been efforts towards developing secondary memory based index structures.

In this chapter, we use a modification of Kd-Tree to build an index structure that supports efficient clustering of data points into buckets. The data set is partitioned into several small non-overlapping isothetic hyperboxes. Two hyperboxes are merged sharing a common face (called a separating plane) if the distance between two hyperboxes is less than a similarity distance threshold. At the end of the process, a set of non-overlapping hyperboxes is obtained, indexed by the same Kd-Tree, containing data points, and each of them represents a bucket containing a cluster of points. These buckets can be accessed from the secondary storage with the indexing structure residing in the main

memory. It has been observed that the ranges and variances of the various elements of the Euler vector widely differ. In such an environment, the Mahalanobis distance can be adopted to provide a very good measure that captures the similarity/dissimilarity properties among the members of a given set of images, as discussed in the previous chapter.

4.2 Review of Multidimensional Access Methods

Interest in multidimensional access methods stems mainly from its manifold applications to diverse areas like VLSI CAD, geosciences applications, CBIR, etc. On the other hand, feature based similarity searching has received wide attention for retrieval of images .

Gaede and Günther in a survey of multidimensional access methods [41], first discuss main memory based multidimensional data structures, including the overview of a K-d-tree [18, 19]. The existing multidimensional access methods are classified into *point access methods* and *spatial access methods*. For *point access methods*, several schemes are known: (a) multidimensional hashing schemes, (b) hierarchical access methods and (c) space filling curves. In particular, the hierarchical access methods are of interest to our work, where the data points are organised into a number of data buckets. Each bucket usually corresponds to a leaf node of the tree. These buckets are usually assigned to disk pages. The internal nodes of the tree reside in the main memory and guides the search path to the leaf nodes. The K-D-B tree [122], a variant of K-d-tree, adapts the K-d-tree for secondary memory structure by combining the properties of a B-tree [28] and the adaptive K-d-tree [19]. Other variants of K-d-tree for nearest neighbor searching have also been reported [138]. There exist several spatial access methods based on the types of bounding regions like R tree [50], R^+ tree [131], R^* tree [17]. Others techniques use hyperspheres as the bounding regions like in SS tree [155]. Some index structures also use as bounding regions intersection of hyperspheres and hyperboxes like SR tree [69].

Multidimensional index structures can also be classified as (i) *space partitioning* based techniques and (ii) *data partitioning* based techniques [22]. The former techniques recursively partition the entire space into mutually disjoint sub-spaces. Each node of

the tree defines a sub-space which is mutually disjoint with the space represented by all other nodes at the same level of the tree. K-d-tree and K-D-B tree [122] are examples of such *space partitioning* techniques. Other methods include range tree and fractional cascading, that use a primary structure based on a single dimension and each node is attached with an associated structure formed on the other dimensions [102]. *Space partitioning* based index structures are almost similar to the hierarchical access of *point access methods* [41]. In *data partitioning* based index structures, the data points at the lowest level are clustered using different bounding regions like rectangular hyperbox, sphere, etc. At any subsequent higher level, the bounding regions of the next lower level are clustered to form bigger bounding regions. This allows overlaps of bounding regions at the same level. The clustered bounding regions thus formed at different levels define the hierarchical index structure. Depending on the nature of bounding regions, there are various index structures like R tree, R^+ tree, R^* tree, SS tree, and SR tree. *Data partitioning* based index structures are similar to overlapping and clipping techniques as described in [41].

There are merits and demerits of both the techniques. In *space partitioning* techniques, including the most widely used K-d-tree, a single dimension is used for splitting a sub-space to create nodes of the tree at the next lower level. Here, the index structure stores only the particular splitting dimension. On the other hand, the *data partitioning* techniques use all the dimensions for further levels of split. Thus, each node of a *data partitioning* index structure stores the ranges for all the dimensions. Space partitioning techniques generate and store dead spaces (sub-spaces of multidimensional space containing no data point), but data partitioning techniques provide better space utilization as the regions are bounded by their containment of data points at the corresponding levels of the tree. A range query on the multidimensional space, using a *space partitioning* index structure like K-d-tree, requires only a single check at each node corresponding to the splitting dimension. But for the same query on any *data partitioning* index structure, the entire bounding range box corresponding to a node is to be checked for its containment in the query range box. Thus, in terms of accessing time, the methods degrade drastically with the increase in the dimensionality of the data. Most of the access methods suffer from this problem and are not known to scale beyond 10-15 di-

mensions [22]. It has been observed by Kleinberg [73], that if $d \geq \log n$ (where d is the number of dimensions and n is the number of data points), 'brute-force' works better than indexing structures. Chakrabarti and Mehrotra [22] proposed a new type of access method known as *hybrid* tree. The *hybrid* tree combines the positive aspects of *space partitioning* and *data partitioning* techniques to achieve better search performance. In a *hybrid* tree, a node is always split using a single dimension as in a K-d-tree. But unlike a K-d-tree, where the split sub-spaces are always mutually disjoint, the *hybrid* tree allows for an overlap of indexed sub-spaces.

Index structures supporting similarity queries perform poorly if they do not fit in the main memory. A portion of the index structure needs to be fetched from the disk, making the search algorithm inefficient. Further, neighboring subregions can grow exponentially in terms of the dimensionality of the data and may require near exponential I/O access. Also, increased dimensionality is a potent problem as mentioned earlier. As a possible solution to these problems, there have been efforts towards indexing meaningful clusters of the data. Chang et al. [23] have used i) a clustering technique to cluster similar data on disk to minimize disk latency for retrieving similar objects and ii) a hashing technique to index clusters. Data clustering [65, 145], in itself, is a wide area of active research and is beyond the scope of this thesis. Clustering techniques have received wide attention in the pattern recognition community.

In this work, we study retrieval techniques using Euler vector. Euler vector is a 4-dimensional feature. The queries are limited to point or range queries. As mentioned in Section 3.3, the dispersion of each dimension of the Euler vector may vary widely. The query range for each dimension will also be different. In such an environment of widely differing dispersions, the partitioning would be better if we can choose the splits of sub-spaces guided by the dispersions of their dimensions. The *space partitioning* techniques allow us to choose such splits, as already discussed. Thus, we use a basic K-d-tree, for supporting space partitioning and tailor it to suit our purpose. In the next section, we would briefly review the construction and query on a K-d-tree. In the Section 4.4, we will discuss proposed modifications on K-d-tree.

4.3 A Brief Review of Kd Tree

The Kd Tree [18] is a multidimensional data structure based on space partitioning techniques. Let P be a set of points, with $|P| = n$, where each point $p_i \in P$ has d dimensions, i.e. $p_i = \{x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(d)}\}$. An image with d features can be represented as a point in the d-dimensional feature space. A K-d-tree defines a recursive partitioning of the space where the partitioning at each level of recursion is done using a single dimension out of the d dimensions. The K-d-tree supports range queries. A d-dimensional rectangular range query on P is a search for points in P that lie inside a query rectangle R: $[l_1, r_1] \times [l_2, r_2] \times \ldots \times [l_d, r_d]$. A point p_i lies inside R if and only if $x_i^{(1)} \in [l_1, r_1]$, $x_i^{(2)} \in [l_2, r_2]$, ..., $x_i^{(d)} \in [l_d, r_d]$.

4.3.1 Algorithm for Kd Tree Construction

The construction of the data structure is done in the following way. The root of the tree corresponds to the whole data set P. At the first step, the set P is split into two subsets P_1 and P_2 of roughly equal size by a hyperplane perpendicular to the first dimension drawn at the median of $\{x_1^1, x_2^1, \ldots, x_n^1\}$, (i.e., median of the coordinate values of the first dimension of the points). At the root of the tree, this median value is stored. In the next level, P_1 is split by the median of the coordinate values of the second dimension of the points in it, into two subsets. Similarly, P_2 is split into two parts. These two median values, corresponding to the second dimension of the points in P_1 and P_2, are stored as the two children of the root. The left and right subtree correspond to P_1 and P_2 respectively. In the next level, the partition is on the third dimension on the subsets generated at this level. This process continues until depth d-1, where we partition on the dth dimension. At depth d, we start partitioning again on the first dimension and the same procedure continues on the subsets generated. This recursive partitioning stops when each partition contains exactly one point, which is then stored as a leaf. At any level l of the binary tree, the splitting dimension is $(l \ mod \ d + 1)$. Associated with each node of the tree there is a region containing the corresponding data points of the d-dimensional space. The node at the root corresponds to the entire bounding

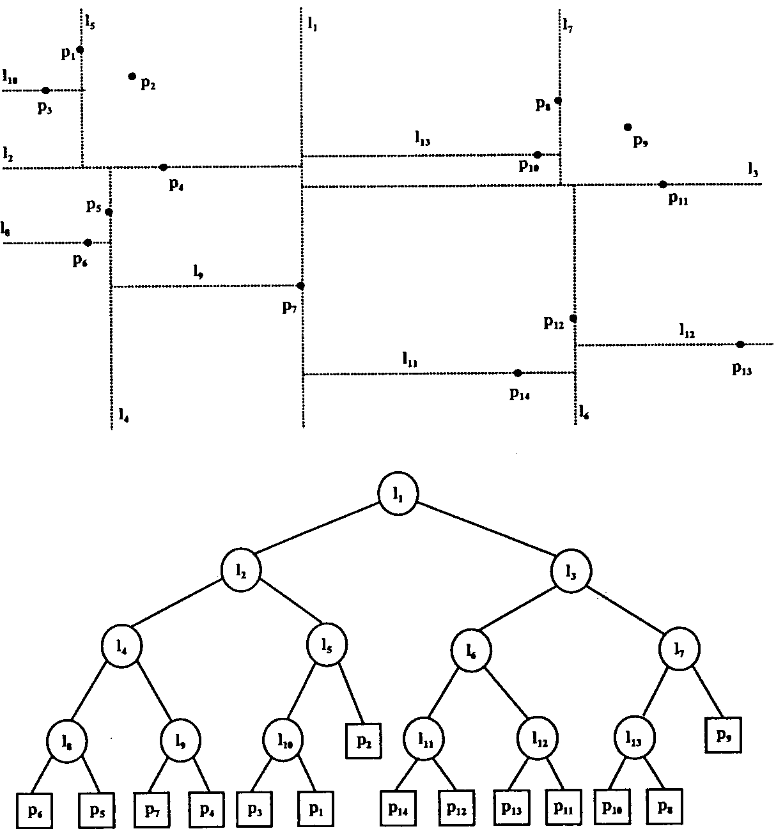


Figure 4.2: An illustration of a K-d-tree in 2D

hyperbox of the data. A node at any level of the tree represents a sub-space of the region containing the points it splits for the next level of partition. See Figure 4.2 for an illustration of the K-d-tree in 2-D.

Algorithm BuildKdTree(P, l)

Input: A set of d-dimensional points P, and the current depth l of the tree

Output: The root of the K-d-tree

1. **if** (P contains only a single point) **then**
   Store the point in the leaf;
   return a pointer to the leaf;
2. **else**
   $k \leftarrow (l \ mod \ d + 1)$; /* note that $k \leq d$ */
   Find the median m_k of the coordinate values of the kth dimension of the points in P;

Split P into P_1 and P_2 such that P_1 be the set of points whose k^{th} dimension coordinate values are less than or equal to m_k and P_2 be the set of points whose k^{th} dimension coordinate values are greater than m_k;

3. left_node $\leftarrow$ **BuildKdTree**(P_1, $l+1$);
4. right_node $\leftarrow$ **BuildKdTree**(P_2, $l+1$);
5. Create a node v storing the value m_k;
   Make left_node the left child of v;
   Make right_node the right child of v;
6. Return v;

endif

All the points are stored at the leaf. For a binary tree, the storage space is bounded by the number of leafs, and for each leaf, the data corresponding to d dimensions are to be stored. Thus, the storage space requirement for the K-d-tree is precisely $O(dn)$. □

The time complexity for constructing the K-d-tree is dominated by the median finding for each partition of the point sets. Median of a set of real numbers can be found in linear time [29]. The K-d-tree construction is a recursive procedure and therefore, the time complexity is $O(dn \log n)$ [18, 19, 102, 109]. □

4.3.2 Query in a Kd Tree

Each node of a K-d-tree represents a region of the entire space [41, 102] as mentioned in the previous subsection. Now, given a query range R like: $[l_1, r_1] \times [l_2, r_2] \times \ldots \times [l_d, r_d]$, the K-d-tree is searched recursively down for finding intersection of the query range with the regions represented by the nodes. The internal nodes storing the splitting hyperplanes act as discriminators to guide the search path from the root level down to the leaf level. The query algorithm, guided by the internal nodes, visits those nodes whose regions are properly intersected by the query range R. If a node is found such that the entire space it represents is fully contained in R, the entire subtree rooted at that node is reported. For some other nodes whose regions partially intersect with R, the search continues in the left or right or both subtrees (as the case may be depending

on which side of the median, R intersects). If such a node is not found and the query traversal reaches a leaf node, the point stored at the leaf node is checked for its presence in R.

Algorithm QueryKdTree(v, R)

Input: v, the root of the tree; R, the query range

Output: The k points that lie in R

```
1. if (v is a leaf) then
       Check if the point stored in v lies inside R;
       Report the point if it lies in R;
2. else
3.     if (the region represented by the left_node of v lies entirely in R) then
           Report the entire subtree rooted at left_node(v);
4.     else
5.         if (the region represented by the left_node of v intersects R) then
               QueryKdTree(left_node(v), R);
           endif
       endif
6.     if (the region represented by the right_node of v lies entirely in R) then
           Report the entire subtree rooted at right_node(v);
7.     else
8.         if (the region represented by the right_node of v intersects R) then
               QueryKdTree(right_node(v), R);
           endif
       endif
   endif
```

The worst case time complexity of the algorithm is determined by the number of regions (nodes) represented in the K-d-tree that can intersect with each of the bounding hyperplanes of R in the worst case. The query time in the worst case is $O(n^{1-1/d} + k)$, where k is the number of points in S that lies inside R. For a detailed analysis that uses

a recurrence formulation see, [18, 109, 102]. If the query is made with respect to a point instead of a range, the K-d-tree can check the presence of the point in time complexity of $O(\log n + d)$ in the worst case [102]. □

4.4 Proposed Modification to Kd Tree

In any binary tree, with the increase of data points, the depth as well as the number of nodes increases. It has already been mentioned that the nodes of the K-d-tree store the discriminators which guide the search path of a query. To answer a query, internal nodes are to be traversed towards the leaf nodes that store the data points. Therefore, the time for answering a query increases with the number of nodes checked for guiding the search path. A solution to this problem is to reduce the depth of the K-d-tree by assigning to each leaf node a set of data points instead of a single data point. Now, to assign data points to a node v, such that the queries can be handled in a manner similar to the original K-d-tree, the best way is to merge the nodes that belong to the subtree rooted at node v to a set of data points, and assign that set to the node v. In a K-d-tree, one does not consider distances between data points while partitioning. Thus, if the set of data points to be merged is to represent a meaningful ensemble of points, the merge should be based on some distance measure. The techniques that try to adapt main memory based index structures with the secondary memory containing the clusters, split the data so that the leaf nodes correspond to buckets storing a set of points. These buckets correspond to disk pages and are stored in the secondary memory. The main memory stores the index structure guiding the search path to the corresponding bucket which is retrieved from the secondary memory. Usually, the similarity queries (where the objective is to retrieve a set of images similar to the query image) are mapped to range queries on points. Thus, it is logical to store a set of similar points in a bucket. The size of each bucket lies within a fixed range, say $MaxSize$ and $MinSize$. The space partitioning techniques should reduce the height of the index structure for storing the buckets at the leaf level by choosing appropriate values of $Maxsize$ and $Minsize$ depending on the application. To address the above issues, we propose a modification of the K-d-tree by performing split and merge operations on the nodes of the K-d-tree.

4.4.1 Split Operations for Partitioning

Let P denote a set of points representing images as feature vectors, where each point p_i is a d-dimensonal vector. At each node v of the K-d-tree, representing a sub-space of the data space, the partition is done as follows. First, we choose the dimension along which the split is to be done. We calculate the range of each dimension of the space P_v represented by the node v. Let k ($\leq d$) be the dimension whose range is maximum (see Figure 4.3 for an illustration).

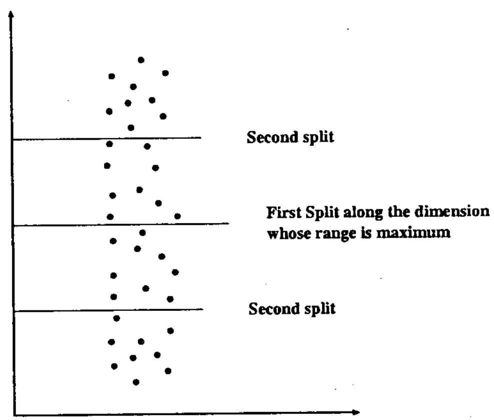


Figure 4.3: Split along the dimension having maximum range

Next, the splitting hyperplane along the kth dimension is found. The median m_{k_v} of the kth coordinates of the points belonging to the space represented by node v is calculated. The splitting hyperplane is perpendicular to the kth dimension and is drawn at $\alpha_k = m_{k_v} + \epsilon$, where ϵ is a small positive constant. We select α_k as follows. Let m'_k be the next higher k^{th} dimension coordinate to m_{k_v} of the points belonging to the space represented by the node v. Then, α_k is the average of m_{k_v} and m'_k. This is done to ensure that the splitting hyperplane does not pass through any point in P. Next, we split P_v into two parts P_{v_1} and P_{v_2} such that the kth dimension coordinates of all points in P_{v_1} are less than α_k and the kth dimension coordinates of all points in P_{v_2} are greater than α_k. Two new nodes v_1 and v_2 are created and v_1 (v_2) is attached as the left (right) child of v. This recursive partitioning scheme continues till the cardinality of the sub-space to be split is greater than $MinSize$. The partition, thus stops at those nodes that have points less than $MinSize$. Note that, these nodes may not be at the

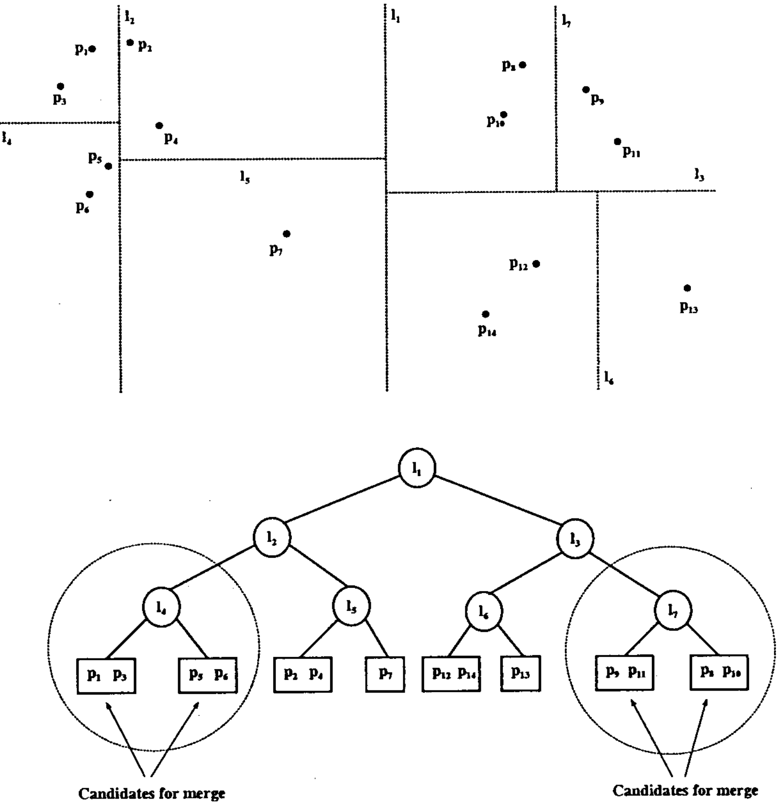


Figure 4.4: An illustration of a modified K-d-tree in 2D

same level. At each node, we store (i) the dimension with respect to which the split has been performed, and (ii) the range of the space represented by it. A global bounding hyperbox is stored at the root. The bounding hyperbox of the nodes at the same level are non-overlapping, and each leaf node of the tree gives a bounding hyperbox containing points less than $MinSize$.

Algorithm SplitOperation(P, $MinSize$)

Input: A set of d-dimensional points P and
$MinSize$, the minimum size of a bucket

Output: The root of the K-d-tree, with buckets of size less
than $MinSize$ stored at the root

1. **if** ($|P| \leq MinSize$) **then**
   Store the set P in the leaf and keep a marker indicating the node is

a leaf node storing a bucket;

return a pointer to the leaf;

2. **else**

Determine the dimension k having the maximum range among points $\in P$;

Find median m_{k_v} of the coordinate values of the

k^{th} dimension of points in P;

Find m'_k, the next higher value to m_{k_v} among the

k^{th} dimension coordinates of P;

$\alpha_k \leftarrow (m_{k_v} + m'_k)/2$;

Split P into P_{v_1} and P_{v_2} such that P_{v_1} be the set of points

whose k^{th} dimension coordinate is less than α_k,

and P_{v_2} be the set of points whose k^{th} dimension coordinate

is greater than α_k;

3. left_node $\leftarrow$ **SplitOperation**(P_{v_1}, $MinSize$);
4. right_node $\leftarrow$ **SplitOperation**(P_{v_2}, $MinSize$);
5. Create a node v storing the value of α_k and k;

Make left_node the left child of v;

Make right_node the right child of v;

6. **return** v;

endif

4.4.2 Merge Operations on the Partitions

To obtain optimal buckets, we now perform a merge operation between two neighboring hyperboxes, represented by sibling nodes, according to their data size and the distance based on the spatial information. Each of the leaf nodes contains one hyperbox containing some of the data points and any two pairs of boxes are non overlapping. In a d-dimensional space any hyperbox has $2d$ neighboring hyperboxes sharing $(d-1)$ dimensions with it. For merging operation, we shall consider only these neighboring hyperboxes. As the time of finding all such hyperboxes would depend on the dimensionality of the data, we choose only hyperboxes that are represented by sibling nodes. We implement the merging in a bottom-up fashion. Two sibling leaf nodes are merged

if the sum of the number of points in both of them is less than $MaxSize$ and the point sets of the two nodes are 'close' relative to a *similarity measure*. The similarity measure should be a measure between two sets. It should also be robust to outlier points in the sets. Let nodes v_1 and v_2 be the candidates for a merge and P_{v_1} and P_{v_2} be the point sets stored in the nodes v_1 and v_2. In such a scenario, we take the similarity measure between sets P_{v_1} and P_{v_2} as follows:
$SimilarityMeasure(P_{v_1}, P_{v_2}) =$
$Min\{Min_{p_1 \in P_{v_1}} Median\{||(p_1, p_2)|| : p_2 \in P_{v_2}\}, Min_{p_2 \in P_{v_2}} Median\{||(p_1, p_2)|| : p_1 \in P_{v_1}\}\}$
where $||(p_1, p_2)|| = MahalanobisDistance(p_1, p_2)$. The $MahalanobisDistance(p_1, p_2)$ is calculated as discussed in Section 3.3. We calculate the variance-covariance matrix from the feature vectors prior to the construction of the modified K-d-tree. The two point sets P_{v_1} and P_{v_2} are merged if $|P_{v_1}| + |P_{v_2}| \leq MaxSize$, and $SimilarityMeasure(P_{v_1}, P_{v_2}) \leq DistThreshold$, where $DistThreshold$ is a threshold on the $SimilarityMeasure$ for merging. The algorithm of merge opertation starts from the root and traverses the Kd-tree in post-order manner [58, 74]. At each internal node (v) that has as its sibling, leaf nodes (containing buckets), it checks for merging between the two siblings. If a merging is possible, a new bucket is prepared with the merged set of points corresponding to the two siblings of v. The new bucket replaces the old two buckets in the secondary storage. The siblings of v are deleted from the data structure, and the node v will now be considered as a leaf node. Finally, the new bucket is attached to v. The sequence of the merge operations is basically predefined by the partition of the data. To get better bucketing, different partitioning techniques can be used for the best merging sequence during the merge operation. At the end, the algorithm returns a tree whose leaf nodes correspond to a bucket saved on the disk. Further, the depth of the tree is also reduced.

Algorithm MergeOperation(v, $MaxSize$, $DistThreshold$)
Input: The root v of the K-d-tree that has the buckets at leaf nodes, $MaxSize$, the maximum size of a bucket, $DistThreshold$, a threshold on the similarity measure between buckets
Output: The root of the same K-d-tree, with bounded buckets

1. **if** (both left_node(v) and right_node(v) are not leaf nodes) **then**
2. **MergeOperation**(left_node(v), $MaxSize$, $DistThreshold$);

```
3.      MergeOperation(right_node(v), MaxSize, DistThreshold);
4.      if (left_node(v) and right_node(v) are leaf nodes) then
        /*though left_node(v) and right_node(v) were not leaf nodes initially,
        after function calls to MergeOperation(), they may be leaf nodes */
5.          Find the SimilarityMeasure D between the two sets of points at
            left_node(v) and right_node(v);
6.          if (D ≤ DistThreshold and the sum of points in
            left_node(v) and right_node(v) does not exceed MaxSize) then
7.              Combine the points attached to left_node(v) and
                right_node(v) in a single bucket;
                Delete the buckets attached to left_node(v) and right_node(v) from disk;
                Write the new bucket in the disk; attach the new bucket with node v;
                Delete left_node(v) and right_node(v) from the Kd-tree;
                Mark v as a leaf node;
                return v;
            endif
        endif
8.  else
9.      Find the SimilarityMeasure D between the two sets of points at
        left_node(v) and right_node(v);
10.     if (D ≤ DistThreshold and the sum of points in
        left_node(v) and right_node(v) does not exceed MaxSize) then
11.         Combine the points attached to left_node(v) and
            right_node(v) in a single bucket;
            Delete the buckets attached to left_node(v) and right_node(v) from disk;
            Write the new bucket in the disk; attach the new bucket with node v;
            Delete left_node(v) and right_node(v) from the Kd-tree;
            Mark v as a leaf node;
            return v;
    endif
```

The entire algorithm for construction of the modified K-d-tree consists of the *SplitOperation* and *MergeOperation*, and is as follows:

Algorithm BuildModifiedKdtree(S, $MinSize$, $MaxSize$, $DistThreshold$)

Input: A set of d-dimensional points S and $MinSize$, the minimum size of a bucket $MaxSize$, the maximum size of a bucket, $DistThreshold$, a threshold on the similarity measure between buckets

Output: v, The root of the K-d-tree, with bounded buckets at the leaf nodes

1. $v \leftarrow$ **SplitOperation**(S, $MinSize$);
2. $v \leftarrow$ **MergeOperation**(v, $MaxSize$, $DistThreshold$);

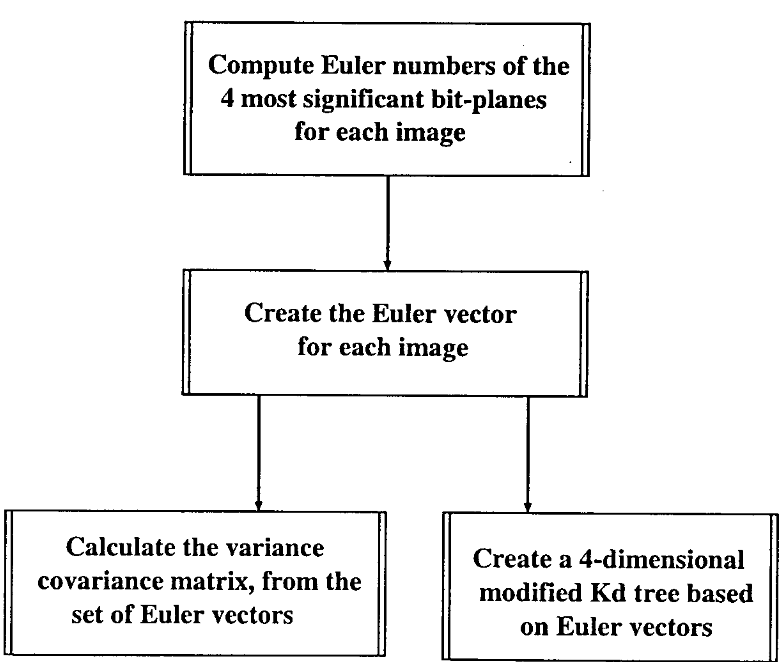


Figure 4.5: The scheme for construction of image database

4.5 Experimental Results

The database used in our experiment is the COIL image database [97], as discussed in the previous chapter and shown in Figure 3.3.

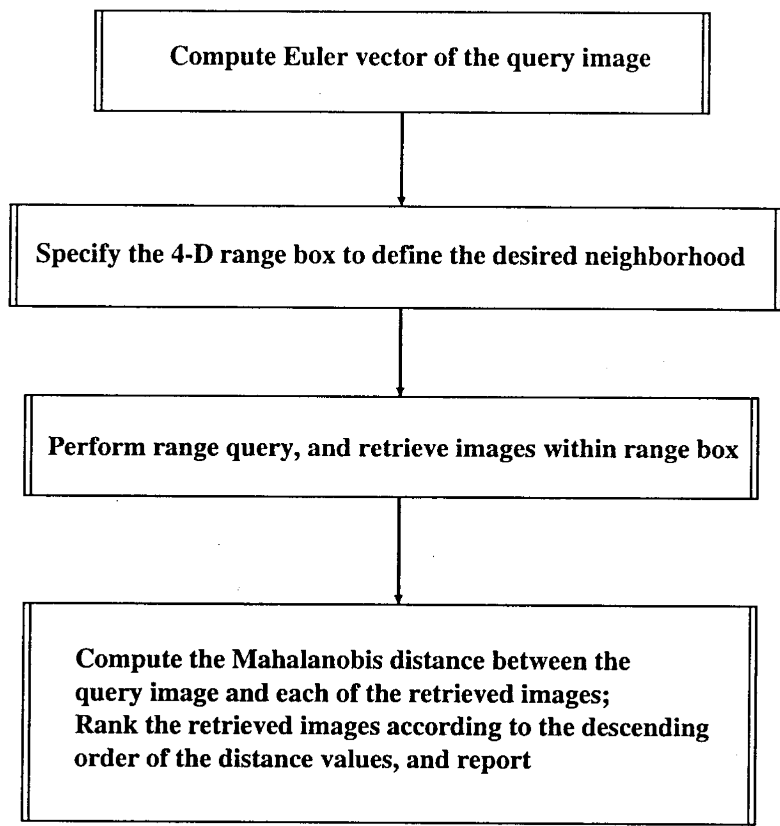


Figure 4.6: The scheme for query processing and retrieval

Schematic diagrams for the entire process of image database construction, query, retrieval, and ranking according to Mahalanobis distance, are shown in Figures 4.5 - 4.8. Each bucket corresponds to a leaf node, and keeps a pointer to disk block where the bucket is stored. The internal nodes guide the search path to the desired leaf node. If only a point query is made for exact search, the bucket at the leaf node containing the point is returned. For a range query, the search algorithm returns the buckets at the leaf nodes whose bounding boxes overlap with the range hyperbox. If the query point is outside the global bounding box, no images are returned. After the corresponding bucket is read, a linear or sublinear search can be performed on it in the main memory for an exact match. We conducted the experiment with different values of $Minsize$, $Maxsize$ and $DistThreshold$, and for each of their combinations with different query images (pertaining to the frontal pose as shown in Figure 3.3) for all objects of the database. The buckets whose bounding boxes overlap with the query range, are de-

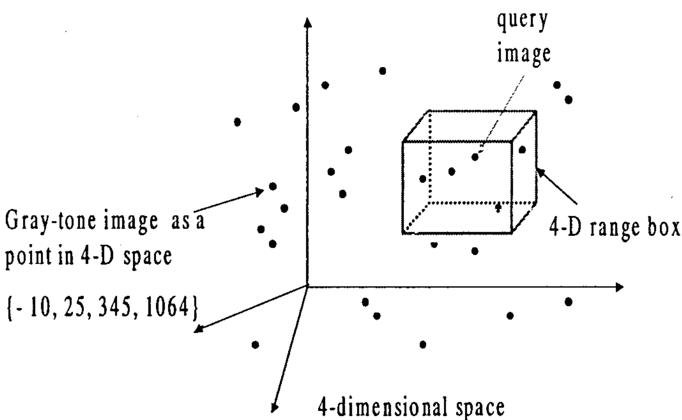


Figure 4.7: A schematic diagram for images as points in feature space

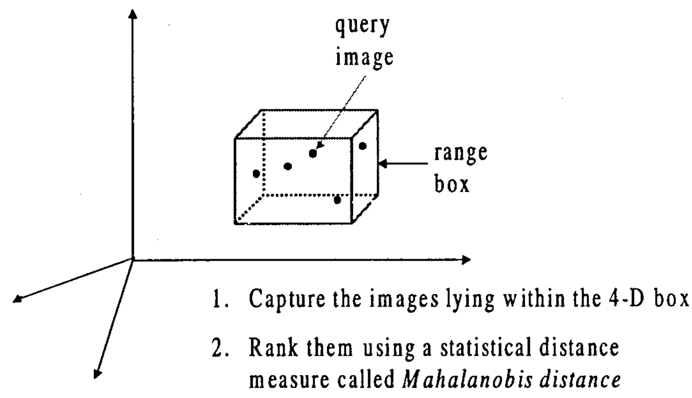


Figure 4.8: A schematic diagram for range query

termined from the modified K-d-Tree; the Mahalanobis distances from all the images are then computed. At most 10 best samples having minimum distance from the query point are reported. We also measured the average depth of the modified K-d-tree (the height of the original K-d-tree is $\lfloor \log_2(1440) \rfloor + 1 = 11$). Two sets of results (for particular values of $MaxSize$, $MinSize$ and $DistThreshold$) are given in Table 4.1 and Table 4.2. Each row in the Tables correspond to a query for a particular image, e.g., *obj8* is the third image in the second row of Figure 3.3. The retrieved results for exhaustive search have also been given. Accepted cases are those where the retrieved image is of the same object, but may be of different poses. Figures 4.9 - 4.12 show the ranked retrieved images corresponding to some cases given in Table 4.1. Figures 4.9 and 4.10 show two cases, where all the reported images belong to the same object. Figure 4.11

show a case where not all the reported images correspond to the same object. Figure 4.12 shows an example of a worst case retrieval, where excepting the query image, no other reported image corresponds to the same object. From the results shown in Figures 4.11 and 4.12, it may be observed that the images of the other retrieved objects, show a striking similarity in either shape or luminance properties with the query object.

4.6 Conclusions

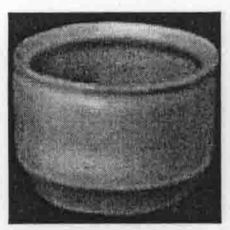
In this chapter, we have addressed the retrieval related issues pertaining to multidimensional access methods, using a feature based on Euler vector alone. Some other known features can be used to augment Euler vector for improving performance. For the queries reported in the tables, the ranges have been calculated using simple methods based on the dispersion of each element of the vector. A better way of finding the ranges would be to estimate it from the distribution pattern of the multidimensional data. As an example, a user may give an image as a query, and ask the system to retrieve a fraction α of the images in the database, that are closest to it. In these cases, the ranges should be determined from the image feature distribution in the multidimensional space.

Table 4.1: Retrieval result - 1

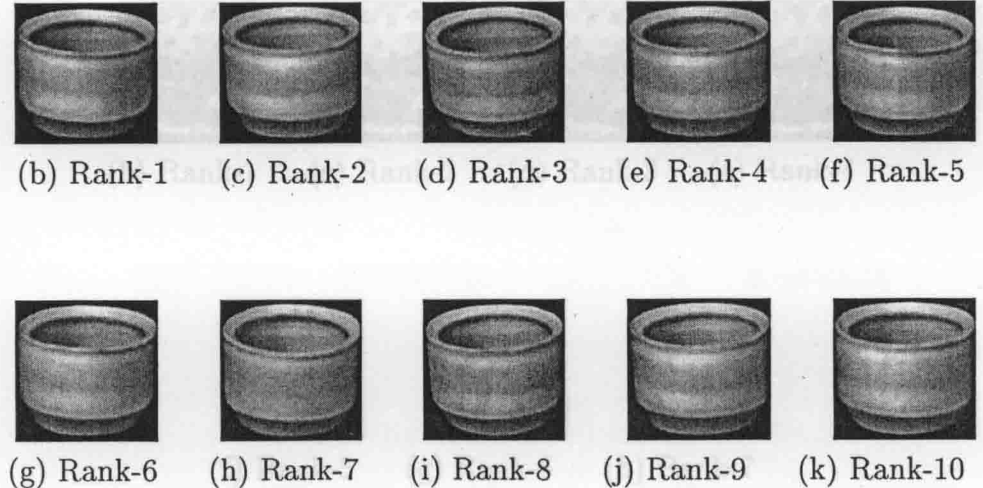
$MaxSize = 70$, $MinSize = 30$, $DistThreshold = 50.0$ Number of buckets = 31, Average Height of K-d-tree = 4.97				
	K-d-tree search		Exhaustive search	
Query Object	No. of images retrieved	No. of images accepted	No. of images retrieved	No. of images accepted
obj1	10	5	10	5
obj2	5	4	5	4
obj3	10	3	10	3
obj4	7	7	7	7
obj5	6	6	6	6
obj6	10	1	10	1
obj7	10	1	10	1
obj8	7	3	7	3
obj9	7	7	7	7
obj10	9	6	9	6
obj11	10	1	10	1
obj12	10	2	10	2
obj13	10	7	10	7
obj14	10	4	10	4
obj15	10	4	10	4
obj16	10	5	10	5
obj17	10	10	10	10
obj18	10	9	10	9
obj19	10	5	10	4
obj20	10	3	10	3

Table 4.2: Retrieval results - 2

$MaxSize = 40$, $MinSize = 20$, $DistThreshold = 10.0$ Number of buckets = 58, Average Height of K-d-tree = 5.90				
	K-d-tree search		Exhaustive search	
Query Object	No. of images retrieved	No. of images accepted	No. of images retrieved	No. of images accepted
obj1	10	5	10	5
obj2	4	3	4	3
obj3	10	4	10	4
obj4	5	5	5	5
obj5	4	4	4	4
obj6	10	1	10	1
obj7	10	1	10	1
obj8	1	1	1	1
obj9	3	3	3	3
obj10	6	4	6	4
obj11	10	1	10	1
obj12	10	3	10	3
obj13	10	6	10	6
obj14	10	5	10	5
obj15	10	4	10	4
obj16	10	5	10	5
obj17	10	9	10	9
obj18	10	8	10	8
obj19	10	3	10	3
obj20	5	4	5	4



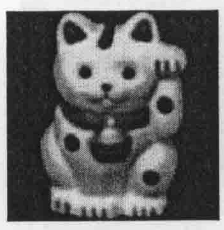
(a) Query image



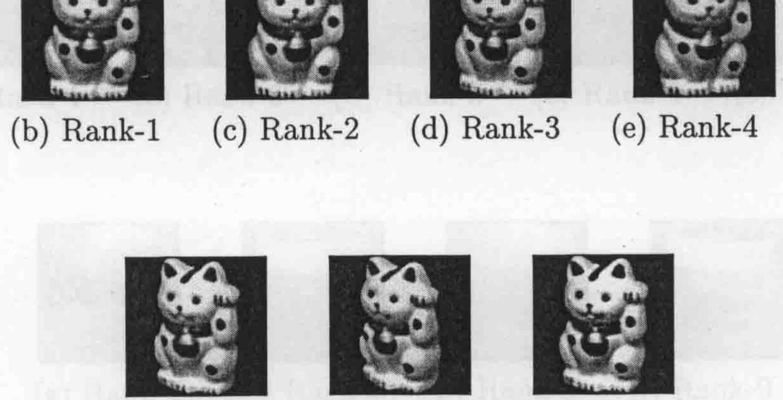
(b) Rank-1 (c) Rank-2 (d) Rank-3 (e) Rank-4 (f) Rank-5

(g) Rank-6 (h) Rank-7 (i) Rank-8 (j) Rank-9 (k) Rank-10

Figure 4.9: A sample result showing a good retrieval



(a) Query image

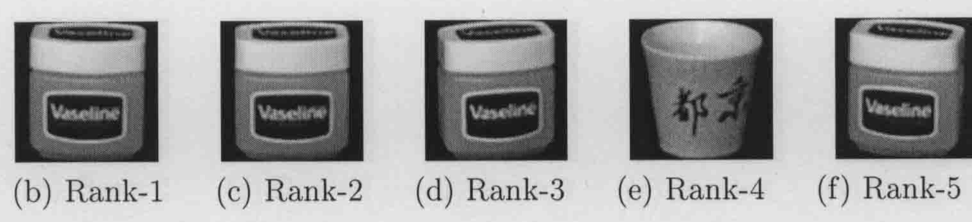


(b) Rank-1 (c) Rank-2 (d) Rank-3 (e) Rank-4

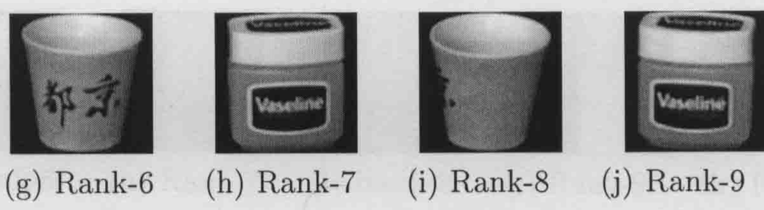
(f) Rank-5 (g) Rank-6 (h) Rank-7

Figure 4.10: A sample result showing a good retrieval

(a) Query image

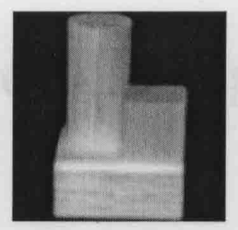


(b) Rank-1 (c) Rank-2 (d) Rank-3 (e) Rank-4 (f) Rank-5

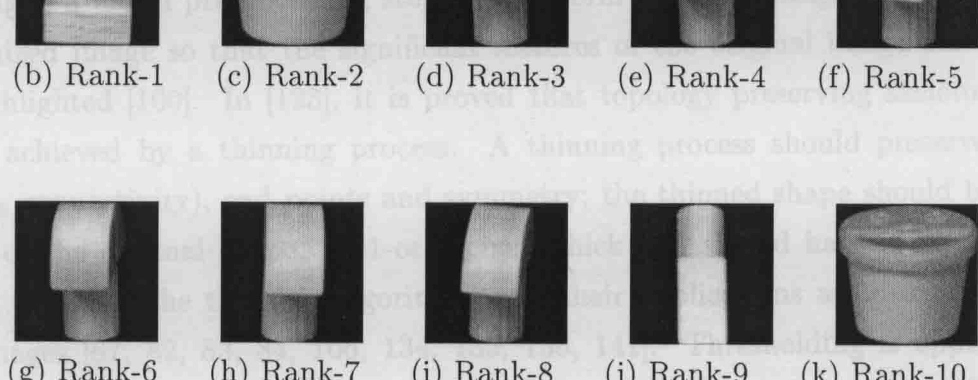


(g) Rank-6 (h) Rank-7 (i) Rank-8 (j) Rank-9

Figure 4.11: A sample result showing a not so good retrieval



(a) Query image



(b) Rank-1 (c) Rank-2 (d) Rank-3 (e) Rank-4 (f) Rank-5

(g) Rank-6 (h) Rank-7 (i) Rank-8 (j) Rank-9 (k) Rank-10

Figure 4.12: A sample result showing a very poor retrieval

Chapter 5

Fingerprint Processing and Related Architecture

5.1 Introduction

Thinning is a useful preprocessing step to transform a digital image to a 1-pixel wide skeletonized image so that the significant features of the original image are retained and highlighted [100]. In [123], it is proved that topology preserving skeletonization can be achieved by a thinning process. A thinning process should preserve homotopy (d_8 connectivity), end points and symmetry; the thinned shape should be in the middle of the original shape, be 1-or-2 pixels thick and should have some noise immunity. Most of the thinning algorithms and their applications are designed for binary images [67, 82, 83, 84, 106, 134, 135, 136, 141]. Thresholding is applied on a gray-scale image to obtain a binary image which is thinned to obtain the skeletonized version. The selection of threshold value affects the correctness of the resulting skeleton. For this reason, direct thinning of gray-scale image has also received attention [1, 11, 36, 110, 111, 141, 150, 157]. Dyer and Rosenfeld [36] generalized the standard binary thinning algorithms for gray-level images. Two pixels are said to be connected if and only if the gray-level of any element along the path that connects them is not less than the gray-levels of both of them. The boundary point is defined according to the

gray-level values of the neighboring pixels and the image is thinned in a parallel mode. The skeleton lies in a central place determined by the boundary of the image. In [157], each pixel is assigned to either a skeleton or a non-skeleton category so that relaxation technique can be used to perform thinning. To preserve straightness of lines, contextual information in the data is used. Four skeletal point classes corresponding to four directions of $0°, 45°, 90°, 135°$ and one non-skeleton category are created to classify each point. An initial probability is computed to assign a pixel to each class. Next, in the relaxation process, neighbor point probability values interact to refine the probability value. The relaxation terminates when points lying on skeleton achieve high skeleton probability values while other points have unity non-skeleton probability values. Experimental results are shown on Chinese characters. A polynomial in two variables with real coefficients can be used to represent a gray image [110]. The authors then use some algebraic operations on the polynomial representation of a gray image by a suitably chosen template polynomial to perform different image processing tasks. They perform a three step process for the skeletonization. A parallel computation on a cellular machine is also proposed. Abe et al. in [1] considered the object (a part of the image) whose skeleton is to be extracted with non-uniform gray values and the background as having value zero. The authors observed that Salari and Siys' algorithm [128] being purely sequential, might produce unsuitable results in some cases. They used a definition of local gray-scale connectivity and applied a combination of sequentially and parallely judged conditions for pixel removal. Their algorithm is a gray-image equivalent of Hilditch's binary image thinning [55]. Arcelli and Ramella [11] proposed a parallel thinning algorithm based on iterated erosion of patterns, proceeding from the lower gray-values towards higher ones until the gray skeleton is finally obtained. In a preliminary phase, significance of hollows and plateaux possibly existing in the pattern was investigated. Hollows with significant depth were regarded as topological constraints for the skeleton structure. Qian et al. [111] used the Gray Weighted Distance Transform [125] to take special care of the hollows and ridges in gray images.

Finding the cause of the spatial distribution of gray-level intensity values in an image is a key requirement of many computer vision algorithms. Therefore, a very important task of computer vision algorithms is ridge and valley classification of pixels. A ridge

(valley) occurs when there is a connected sequence of pixels having gray-tone intensity values that are significantly higher (lower) in the sequence than those pixels in the neighborhood of the sequence. In [52], ridges and valleys on digital images were found by looking for zero-crossings of the first directional derivative in a suitable direction. Ridge like structures in digital images can also be extracted by convolving the images with different derivatives of Gaussians. In [90], two ridgeness measuring differential operators were studied with respect to their usability in CT/MRI matching of human brain scans. López et al. in [88] discusses the use of some discrete multilocal measures for ridge finding.

Automatic fingerprint identification systems (AFIS) are a class of biometric techniques widely used for personal identification. AFIS's are usually based on minutiae matching [39, 56, 64, 78]. Minutiae, or Galton's characteristics [42], are local discontinuities in the ridge lines within the area of interest in the fingerprint pattern. Minutiae are essentially the terminations and bifurcations of the ridge lines that constitute the fingerprint pattern. These two types of minutiae are considered by Federal Bureau of Investigation for identification purposes [154]. AFIS based on minutiae matching involves different stages:

1. Fingerprint image acquisition,
2. Preprocessing of the fingerprint image,
3. Feature extraction (e.g. minutiae) from the image,
4. Matching of fingerprint images for identification.

In [94], Mehtre described the steps for handling noise in fingerprint images, enhancement and restoration of the image and a parallel thinning procedure. A more comprehensive and detailed discussion on all the aspects of personal identification using fingerprints as an important biometric technique can be found in Jain et al. [64]. Issues involving practical uses of biometric techniques, history of fingerprints and their uses, fingerprint image acquisition to matching, and a minutiae extraction algorithm were discussed. In their method, the orientation field of the fingerprint image is estimated and then based

on the local certainty level of the orientation field, a segmentation algorithm is used to locate the region of interest in the fingerprint image. Ridges in these zones are then detected by convolving the original image with two masks, that are used for increasing the local maximum gray value along a direction normal to the local ridge direction. Following these steps, ridges are thinned and then the minutiae are detected. Further, a procedure for an alignment based matching of minutiae is reported. Thinning of gray-scale fingerprint images for successive minutiae extraction had also been reported. In O'Gorman and Nickerson's [101] work, a $k \times k$ spatial filter mask, designed based on user inputs, is used with appropriate orientation for labeling the pixels as foreground (crest) or background. Thinning is done on the binary image obtained before minutiae extraction. In [95], Mehtre and Chatterjee deal with segmenting a fingerprint image into ridge zones and background based on some statistics of the directional image that represents the local orientations of the ridges of the original image. The gray-scale variance method is used in the image blocks of uniform gray-level where the directional method of segmentation fails. A comprehensive work on minutiae extraction from skeletonized binary images appears in Farina et al.'s work [39]. However, their work is solely based on processing of skeletonized one pixel thick binary images, and obtaining it from the original gray level image is a challenging as well as time consuming task. The problem of automatic minutiae extraction, though studied for quite a long time, still attracts attention as a complete and efficient solution is elusive. Reasons stem mainly from the fact that most of the fingerprint images are of low quality, and noise and different levels of contrast in the image may produce false minutiae or hide real minutiae. Lately, there have been efforts to avoid minutiae extraction from binary images and attempt the extraction directly from the gray scale domain. Maio and Maltoni [91] have developed a gray level ridge line tracing algorithm for minutiae extraction. The algorithm basically traverses the fingerprint directional image for discontinuities. The direction is computed as a tangent to the ridge line in such a way that the orientation becomes closest to the direction computed at the previous step on that ridge. The algorithm treats the fingerprint image as a gray level zone of ridges and background, whereas, such an image actually consists of three regions - ridges, valleys, and background (which we would refer to as plateau in our work). Based on the three regions of an image a statistical analysis of the gray level histogram is used for having the global information about

the range of ridges, valleys and background [24]. Minutiae detection from a gray-scale fingerprint image is extremely difficult when the image is of poor quality. Minutiae extraction methods involve preprocessing of the ridge detected image. Thus, a ridge detection procedure for reliable minutiae extraction should not miss any ridge. False ridges leading to spurs and bridges [39] may be taken care of by a preprocessing stage prior to minutiae extraction.

In this chapter, we study gray-level thinning with special application to fingerprint images. Essentially, we view gray-level thinning of fingerprint images as a twofold process: a pixel classification scheme for ridge extraction followed by a binary thinning on the ridges. Ridges or crests (henceforth the term ridge or crest will be used interchangeably) are extracted from a fingerprint image by classifying each pixel combinatorially. A two-pass algorithm is developed to classify a pixel into three classes, namely crest (CR), valley (VA) or plateau (PL). The crest pixels necessarily trace out the ridge lines in the image. The pixels are classified by their relative gray-scale topographical configuration in its neighborhood, viewed from certain directions. In the first pass of our algorithm, a pixel is classified as unambiguous (ridge, valley and plateau), or ambiguous. In the second pass, local neighborhoods are checked to further classify the ambiguous pixels as the unambiguous class. Finally, any binary thinning algorithm can be applied on the crest pixels as object and the rest as background to extract the one-pixel thick ridge line. The operations for classification being inherently parallel and highly modular can be easily implemented on-chip. We propose a simple VLSI implementation of the said algorithm using D flip-flops, comparator circuits, combinational logic, and adders. The classification scheme uses a Look-Up-Table (LUT). Finally, architectures for binary image thinning reported elsewhere [32, 133] can be used to extract one-pixel thick ridge lines. The proposed classification technique retains the wholesomeness and the information content in the image, and needs very low processing time. The algorithm is fast, robust, and use of thresholding is minimal. The continuity of a (possibly weakly) connected ridge line is also ensured. It has been tested on several fingerprint images in the NIST Special Database 4 and NIST Special Database 14 [21, 152, 153], and is observed to have produced good results both in terms of quality of solutions and CPU time.

5.2 Mathemetical Preliminaries

In this section, we review the mathematical characterizations of crests and in the next section we justify our combinatorial approach for finding a crest.

There has been a plethora of notions used for finding crests and valleys in a digital image. An intuitive notion to define a digital crest (valley) is to consider a connected sequence of pixels with gray-tone intensity values that are significantly higher (lower) than those neighboring the sequence [52]. The term 'significant' is a qualitative term and as such depends on the intensity values of the neighborhood of the sequence. It has been pointed out that a unique description of a ridge lacks convergence [90]. A ridge (valley) may be thought of as a locus of points on the image intensity landscape such that the point is always at a higher (lower) altitude compared to the intensity landscape to its left and right. The directions left and right are with respect to the tangent of the locus. Though the above notion might characterize a ridge to a certain extent, a problem still remains on how to reach a ridge on an intensity landscape. We consider the following definition [90]: keep walking uphill on the intensity landscape; the point where a sharp turn is made, is a ridge. This definition might fail if the ridge itself is uphill. On a walk along an uphill ridge, there would obviously be no sharp turn. This observation shows us that the direction of walk on such an intensity landscape is important for finding a ridge. This leads us to the concept of directional derivatives.

In [52], a coordinate system is set up whose origin runs through the center of the pixel $p_{i,j}$ (henceforth, $p_{i,j}$ will be represented as P) under consideration. To estimate a continuous surface around P, a fitting cubic polynomial $f(i,j)$ in two variables, row(i) and column(j), is chosen. An analytical form for all second partial directional derivatives $f''_\alpha(i,j)$ in terms of the direction α is deduced from $f(i,j)$. From the expression $f''_\alpha(i,j)$, two directions α_1 and α_2 are found that respectively minimizes and maximizes $f''_\alpha(i,j)$. Now, if along α_1, the first partial derivative $f'_{\alpha_1}(i,j)$ has a zero-crossing, P is a crest. Similarly, along α_2, if $f'_{\alpha_2}(i,j)$ has a zero-crossing, P is a valley. If along α_1, a crest, and along α_2, a valley is found, P is a saddle point. This characterization fails for some surfaces, e.g., a radially symmetric $f(i,j)$. To tackle such problems, a threshold is maintained on the gradient-curvature ratio and the angle between the gradient vectors

on each side of the ridge (valley). To prevent occurrence of artifacts, a threshold is maintained on the height (depth) of a crest (valley).

There are some other measures of ridge(ness) like L_{vv} and *isophote curvature*, κ. We discuss next some notations required for understanding these measures. We restrict ourselves strictly to two-dimensional geometry. The operator ∇ is defined as follows:

$$\nabla = (\frac{\partial}{\partial x} \ \frac{\partial}{\partial y}).$$

The operator ∇^2 is as follows:

$$\begin{aligned} \nabla^2 &= \nabla^t \cdot \nabla \\ &= \begin{pmatrix} \frac{\partial^2}{\partial x^2} & \frac{\partial^2}{\partial x \partial y} \\ \frac{\partial^2}{\partial x \partial y} & \frac{\partial^2}{\partial y^2} \end{pmatrix} \end{aligned}$$

L denotes a 2-D function, i.e.,

$$L : \{x, y\} \to z; x, y, z \in \mathbb{R}.$$

Any vector is denoted as

$$\mathbf{v} = (v_x \ \ v_y)^t.$$

The first order partial derivative of L along $\mathbf{v}$ is denoted as

$$L_v = \frac{1}{\| \mathbf{v} \|}(\nabla L \cdot \mathbf{v}).$$

The second order partial derivative of L along $\mathbf{v}$ is

$$L_{vv} = \frac{1}{\| \mathbf{v} \|^2}(\mathbf{v}^t \cdot \nabla^2 L \cdot \mathbf{v}). \tag{5.1}$$

Also, denote

$$\frac{\partial L}{\partial x} = L_x, \frac{\partial L}{\partial y} = L_y, \frac{\partial^2 L}{\partial x^2} = L_{xx}, \frac{\partial^2 L}{\partial y^2} = L_{yy}, \frac{\partial^2 L}{\partial x \partial y} = L_{xy}. \tag{5.2}$$

Let L be a scalar function having continuous first partial derivatives. Then ∇L exists and its length and direction are independent of the particular choice of the Cartesian

coordinates in the plane. We choose $\mathbf{v}$ as the right handed normal of the gradient $\mathbf{w}$, of L, i.e. ($\mathbf{w} = \nabla L$). Thus, $\mathbf{w} = (L_x \; L_y)^t$ and $\mathbf{v} = (L_y \; -L_x)^t$. Deducing L_{vv} using Equations 5.1 and 5.2, we get

$$\begin{aligned} L_{vv} &= \frac{1}{L_x^2 + L_y^2} \begin{pmatrix} L_y & -L_x \end{pmatrix} \cdot \begin{pmatrix} L_{xx} & L_{xy} \\ L_{xy} & L_{yy} \end{pmatrix} \cdot \begin{pmatrix} L_y \\ -L_x \end{pmatrix} \\ &= (L_y^2 L_{xx} - 2L_x L_y L_{xy} + L_x^2 L_{yy})/(L_x^2 + L_y^2). \end{aligned} \tag{5.3}$$

Let L be the image intensity landscape function. If L_{vv} has a local minimum (maximum) along $\mathbf{v}$ at a point, the point is a ridge (valley). The gradient $\mathbf{w}$ has opposing directions on the two sides of a ridge. The direction of the gradient $\mathbf{w}$ is

$$\theta = \tan^{-1}(L_y/L_x). \tag{5.4}$$

The directional derivative of θ along $\mathbf{v}$, known as isophote curvature and denoted as κ is:

$$\begin{aligned} \kappa &= \frac{1}{\| \mathbf{v} \|}(\nabla\theta \cdot \mathbf{v}) \\ &= -(L_y^2 L_{xx} - 2L_x L_y L_{xy} + L_x^2 L_{yy})/(L_x^2 + L_y^2)^{3/2} \\ &= -\frac{L_{vv}}{L_w} \; [\text{where } L_w = \frac{1}{\| \mathbf{w} \|}(\nabla L \cdot \mathbf{w}) = \frac{1}{(L_x^2 + L_y^2)^{1/2}}] \\ &= -L_{vv} L_w^{-1} \end{aligned}$$

At a ridge (valley), κ should have a positive maxima (negative minima). The isophote curvature κ is basically the curvature of the *level curves* of the landscape function. The *level curves* of L are the curves satisfying the equation $L(x, y) = c$, where c is a constant. In Maintz et al. [90], ridgeness measures L_{vv} and $L_{vv}L_w^{-\alpha}$ $\alpha \in [0, 1]$ on Gaussian smoothed CT/MRI images for matching, were used.

The different ridge/valley characterizations are classified by López et al. [88] as

(i) local: classification is based on a local test;

(ii) global: classification depends on image features arbitrarily far away from the point under consideration;

(iii) multilocal: classification depends on local tests in a predefined region or on the particular geometry of the image.

The ridges/valleys obtained by local tests are called creases. For a pixel P its boundary is defined by a set of discrete points $\mathcal{B} = \{p_{i,j-1}, p_{i+1,j}, p_{i,j+1}, p_{i-1,j}\}$ and a set of directions $\mathcal{N} = \{\mathbf{n_W}, \mathbf{n_S}, \mathbf{n_E}, \mathbf{n_N}\}$ are defined where each $\mathbf{n}$ is a unit normal vector at the points from $\mathcal{B}$. Also, $\tilde{\mathbf{w}} = (\tilde{w}^1, \tilde{w}^2)$ is determined from the gradient vector of the image intensity landscape function L at P, and the eigen vector corresponding to the largest eigen value of a symmetric semipositive definite matrix S, obtained by the convolution of the gradient function with a 2-D Gaussian centered at P. A multilocal creaseness measure at P discretized from a continuous form is defined as:

$$\tilde{\kappa}_2[i,j] = -\frac{1}{2}(\tilde{w}^1[i,j+1] - \tilde{w}^1[i,j-1] + \tilde{w}^2[i+1,j] - \tilde{w}^2[i-1,j]). \tag{5.5}$$

This measure basically is a weighted measure, with different weights taken along different normal directions at the boundary of a region centered around P. An important observation from this creaseness measure is that a crest/valley can be identified from the gradients along different possible directions in a region centered around that point. The same authors have shown the efficacy of this method on different sets of images.

5.3 Proposed Sequential Algorithm

5.3.1 Theme

In an effort to find crests in discrete points and implement it on-chip, we develop a new crest finding algorithm using a Look-Up-Table (LUT). Let $L(i,j)$ denote the image intensity level at a pixel P. Denote the boundary pixels $\mathcal{B}$ of P as $\{p_{i-1,j}$, $p_{i+1,j}$, $p_{i-1,j-1}$, $p_{i+1,j+1}$, $p_{i,j-1}$, $p_{i,j+1}$, $p_{i+1,j-1}$, $p_{i-1,j+1}\}$ respectively with corresponding directions {N, S, NW, SE, W, E, SW, NE} as shown in Figure 5.1. Fix the direction of the straight line walk from each point in $\mathcal{B}$ to another point in $\mathcal{B}$ through P. It is obvious that 4 such directions are possible (see Figure 5.1). The directions $\mathcal{D}$, are $(k,l) \in \{(N,S), (NW,SE), (W,E), (SW,NE)\}$, where (k,l) is a fixed direction of walk

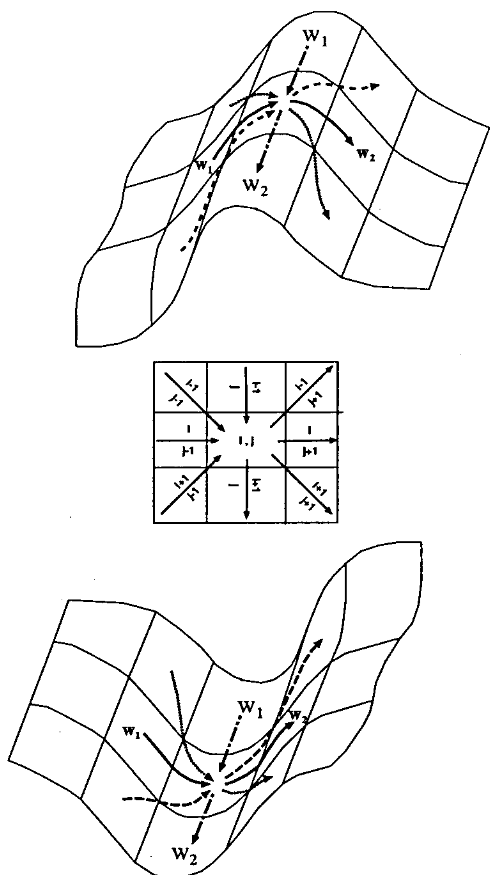


Figure 5.1: Intensity Profiles of Crest and Valley

from k to l. We define two types of elementary walks, w_1 and w_2. A movement from any point in $\mathcal{B}$ to P is defined as w_1; w_2 is defined as a movement from P to any point in $\mathcal{B}$. It is obvious that a walk along any of the directions of $\mathcal{D}$ consists of a walk w_1 followed by a walk w_2 in the same direction. (e.g., a walk along (N, S) consists of a walk w_1 from $p_{i-1,j}$ to P followed by a walk w_2 from P to $p_{i+1,j}$). Note that the sign of the gradient changes for walks w_1 and w_2 if there is a crest (valley) (See Figure 5.1) in a particular direction. The gradient along w_1 is measured as the difference between $L(i,j)$ and the the intensity level at any point on $\mathcal{B}$, where w_1 starts. Similarly, the gradient along w_2 is the difference between the intensity level at any point on $\mathcal{B}$ and $L(i,j)$ e.g., along $(k,l) = (N,S)$, the gradient for w_1 is measured as $L'_{w_1} = L(i,j) - L(i-1,j)$ and for w_2, it is measured as $L'_{w_2} = L(i+1,j) - L(i,j)$. We define first difference pairs, $\Delta_{(k,l)} = \{L'_{w_1,(k,l)}, L'_{w_2,(k,l)}\}$ along any direction $(k,l) \in \mathcal{D}$;

and $sign(\Delta_{(k,l)}) = \{sign(L'_{w_1,(k,l)}), sign(L'_{w_2,(k,l)})\}$, where,

$$\begin{aligned} sign(L'_{w_x,(k,l)}) &= +, \text{if } L'_{w_x,(k,l)} > 0, \\ &= -, \text{if } L'_{w_x,(k,l)} < 0, \\ &= 0, \text{if } L'_{w_x,(k,l)} = 0, \text{for } x = 1, 2 \end{aligned}$$

For a point to be a crest (CR) along any particular direction $\mathcal{D}_{(k,l)}$, there can be three cases (see Figure 5.3):

(i) walk w_1 reaches a crest and walk w_2 falls off signifying a change in the sign of gradient i.e. $sign(\Delta_{(k,l)}) = \{+, -\}$,

(ii) walk w_1 reaches a crest and walk w_2 is on the crest with no gradient change i.e. $sign(\Delta_{(k,l)}) = \{+, 0\}$,

(iii) walk w_1 is on the crest with no gradient change, and walk w_2 falls off the crest i.e. $sign(\Delta_{(k,l)}) = \{0, -\}$.

Similarly for a valley (VA), the three cases are (see Figure 5.3):

(i) walk w_1 reaches a valley and walk w_2 rises from the point signifying a change in the sign of gradient i.e. $sign(\Delta_{(k,l)}) = \{-, +\}$,

(ii) walk w_1 reaches a valley and walk w_2 is on the valley with no gradient change i.e. $sign(\Delta_{(k,l)}) = \{-, 0\}$,

(iii) walk w_1 is on the valley with no gradient change, and walk w_2 rises from the valley i.e. $sign(\Delta_{(k,l)}) = \{0, +\}$.

For a pixel P which is on the slope of the intensity landscape, we call P to be a plateau (PL) point. Obviously there would be no change in the sign of gradients (see Figure 5.3):

(i) $sign(\Delta_{(k,l)}) = \{+, +\}$,

(ii) $sign(\Delta_{(k,l)}) = \{-, -\}$.

Note that, considering pairs from the set $\{+, -, 0\}$, we have $3^2 = 9$ cases of which 8 (3 cases of CR, 3 cases of VA and 2 cases of PL) have been taken care of, and in the last case, i.e., where $sign(\Delta_{(k,l)}) = \{0, 0\}$, we label P as undecidable (UN).

So, along any direction $(k, l) \in \mathcal{D}$, we can label P from any one of the elements of set $C = \{CR, VA, PL, UN\}$, i.e. $C_{k,l}(P) = CR/VA/PL/UN$. Note that if along most of the directions $(k, l) \in \mathcal{D}$, P is a crest, then there is a high probability of P to be a crest. So, we define the label of $C(P)$ as:

$$C(P) = f_{(k,l)\in\mathcal{D}}\{C_{(k,l)}(P)\}, \text{ where } f \text{ is a function as follows:}$$

$$f : C_{(N,S)}(P) \times C_{(NW,SE)}(P) \times C_{(W,E)}(P) \times C_{(SW,NE)}(P) \rightarrow C(P)$$

As an example, we can take a majority vote among the different directions to finally assign P to any element from the set C, i.e.,

$$C(P) = max_{(k,l)\in\mathcal{D}}\{C_{(k,l)}(P)\}.$$

Now, as we want a VLSI implementation of this algorithm, we are interested in the number of combinatorial possibilities of the elements of C along directions $(k, l) \in \mathcal{D}$. There are 4 possibilities along 4 directions, so the number of combinatorial possibilities is intuitively $4^4 = 256$. But the definition of $C(P)$ does not take into account directions. Many combinations of the elements of C are the same (e.g., 2 CRs along *any* 2 directions are the same). To resolve this, we define $x_i, i \in C$, as the number of directions having the label 'i' ($CR/VA/PL/UN$). Now, x_i can take integral values in $[0, 4]$ i.e., minimum it can have label 'i' in none of the directions, and maximum it can have label 'i' along all 4 directions. But, the total number of directions is fixed at 4. Thus,

$$\sum_{i\in C} x_i = 4$$
$$or \quad x_{CR} + x_{VA} + x_{PL} + x_{UN} = 4.$$

Finding the number of possible integral solutions of the above equation is equivalent to finding the coefficient of x^4 in the *generating function* [121] $(x^0 + x^1 + x^2 + x^3 + x^4)^4$

$$\begin{aligned} \text{i.e., coefficient of } x^4 \text{ in} \quad & (1 + x + x^2 + x^3 + x^4)^4 \\ = \text{coefficient of } x^4 \text{ in} \quad & (1 - x)^{-4}(1 - x^5)^4 \\ = \quad & 35. \end{aligned} \tag{5.6}$$

Fact 1 The number of combinatorial possibilities, thus determined, depends on the cardinality of the set $\mathcal{D}$ $((k,l) \in \mathcal{D})$ of directions and not on the set $\mathcal{B}$ of boundary pixels. Generalizing, if n directions are chosen and along each direction the number of possibilities is 4, then the number of combinatorial possibilities is the coefficient of x^n in $(1-x)^{-n}(1-x^5)^n$. The size of the Look-Up-Table can thus be determined. □

5.3.2 Implementation Details

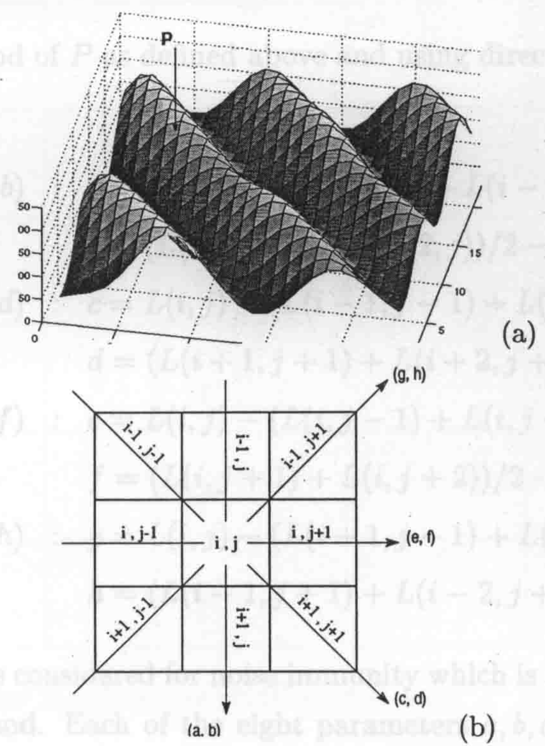


Figure 5.2: (a) Gray scale topology of I. (b) 4 pairs of gray-scale gradient vectors in 8-neighborhood of $L(i,j)$.

Let I be an $N \times M$ gray-scale image with g gray levels. Let $L(i,j)$ be the gray level of the $(i,j)^{th}$ pixel P of I. The discrete surface $z = L(i,j)$ corresponding to a part of such an image I is shown in Figure 5.2(a). Pixels with gray-levels close to g (or 1) are bright (or dark). The fingerprint ridge lines (appearing bright in image I) correspond to surface ridges or crests, and the space between the ridge lines (appearing dark in I) correspond to the ravines or valleys. The relative gray-scale topographical configuration

of L in its locality can be viewed from four possible directions as shown in Figure 5.2(b). To calculate the first difference pairs along the walks w_1 and w_2 defined in Section 5.3.1, we take directional averages along the directions $(k,l) \in \mathcal{D}$ for calculating L'_{w_1} and L'_{w_2}. The calculation of L'_{w_1} and L'_{w_2} at P using directional averages for a 5×5 neighborhood along a direction (NW, SE) is as follows:

$$\begin{aligned} L'_{w1,(NW,SE)} &= L(i,j) - (L(i-1,j-1) + L(i-2,j-2))/2, \\ L'_{w2,(NW,SE)} &= (L(i+1,j+1) + L(i+2,j+2))/2 - L(i,j). \end{aligned}$$

Consider a neighborhood of P as defined above and using directional averages, the first difference pairs are:

$$\begin{aligned} \Delta_{(N,S)} = (a,b) \quad &: \quad a = L(i,j) - (L(i-1,j) + L(i-2,j))/2, \\ & \quad\quad b = (L(i+1,j) + L(i+2,j))/2 - L(i,j), \\ \Delta_{(NW,SE)} = (c,d) \quad &: \quad c = L(i,j) - (L(i-1,j-1) + L(i-2,j-2))/2, \\ & \quad\quad d = (L(i+1,j+1) + L(i+2,j+2))/2 - L(i,j), \\ \Delta_{(W,E)} = (e,f) \quad &: \quad e = L(i,j) - (L(i,j-1) + L(i,j-2))/2, \\ & \quad\quad f = (L(i,j+1) + L(i,j+2))/2 - L(i,j), \\ \Delta_{(SW,NE)} = (g,h) \quad &: \quad g = L(i,j) - (L(i+1,j-1) + L(i+2,j-2))/2, \\ & \quad\quad h = (L(i-1,j+1) + L(i-2,j+2))/2 - L(i,j). \end{aligned} \tag{5.7}$$

Directional averages are considered for noise immunity which is a desirable property [88] of a ridge finding method. Each of the eight parameters $a, b, c, \ldots, h$, can be positive (increasing gray-scale gradient), or negative (decreasing gray-scale gradient), or zero (constant gray-scale gradient), and, therefore, can give rise to three cases. Now, based on the definition of walks w_1 and w_2 in the previous section, we consider the pairwise property of the eight parameters (i.e. a and b, c and d, e and f, g and h) and, therefore, each pair of parameters along the direction of the walks can have 9 cases as laid out in Figure 5.3.

The column, named 'Gray Levels', in Figure 5.3 exhibits the pictorial representation of the three points P, P', P'', where P' and P'' are the adjacent pixels of P in the concerned pair of parameters. The values of these four pairs indicate the topographical

CASE	a	b	GRAY LEVELS	CLASS OF P
1	+	+	P'' P P' ; P'' P' P	PL
2	+	-	P P' P''	CR
3	+	0	P P'' P'	CR
4	-	+	P'' P' P	VA
5	-	-	P' P P'' ; P' P P''	PL
6	-	0	P' P'' P	VA
7	0	+	P'' P' P	VA
8	0	-	P P' P''	CR
9	0	0	P'' P' P	UN

Figure 5.3: Relative position of P w.r.t. P' and P''

configuration of P in its locality. As shown in the right column of Figure 5.3, the *sign* (as defined in Section 5.3.1) of each of the four pairs of parameters $((a, b), (c, d), (e, f), (g, h))$ is used to assign one out of the following four classes to the respective pixel P:

CR: P is a crest pixel if either the gray-value of P is higher than the gray-values of both P' and P'' (see case 2 in Figure 5.3), or, the gray-value of P is same as that of one of P' and P'' but higher than that of the other (see cases 3 and 8 in Figure 5.3).

VA: P is a valley pixel if either the gray-value of P is lower than the gray-values of

both P' and P'' (see case 4 in Figure 5.3), or, the gray-value of P is same as that of one of P' and P'' but lower than that of the other (see cases 6 and 7 in Figure 5.3).

PL: P is a plateau pixel if the gray-value of P lies strictly within the gray-values of P' and P'' (see cases 1 and 5 in Figure 5.3).

UN: P is an undecidable pixel if the gray-values of P, P' and P'' are same (see case 9 in Figure 5.3).

Thus, each pixel P is assigned to any one of the four preliminary classes along a single direction. After the preliminary classification pass, P can be either strongly classified or weakly classified as discussed next.

5.4 Classification of a pixel

5.4.1 Preliminary classification

Let C_{ab}, C_{cd}, C_{ef} and C_{gh} denote the four classes preliminarily assigned to P by the four pair of parameters $(sign(a,b), sign(c,d), sign(e,f), sign(g,h))$. Note that, each of these four classes is one of the four possible preliminary classes (CR, VA, PL and UN). We implement the Look-Up-Table given in Table 5.1 with a bias towards crest (CR). This stems from the particular application to fingerprint images because the minutiae may be defined as the discontinuities on the crest lines. The cases that cannot be topographically resolved to any of the classes in C, are kept for further processing. We define four intermediate classes CV, CP, VP and XX into which the unresolvable pixels are classified for further processing. P is classified to one of the classes among CR, VA, PL, CV(can be crest or valley), CP(crest or plateau), VP(valley or plateau), XX(crest or valley or plateau) depending on the following exhaustive 35 cases as deduced in Section 5.3.1 (see Table 5.1) :

CR: (a): if 3 or 4 classes among C_{ab}, C_{cd}, C_{ef} and C_{gh} are equal to CR;

Table 5.1: Classification of a pixel

No. of cases	No. of classes in preliminary set				Preliminary Class
	CR	VA	PL	UN	
1	0	0	0	4	XX
2	0	0	1	3	XX
3	0	0	2	2	XX
4	0	0	3	1	PL
5	0	0	4	0	PL
6	0	1	0	3	XX
7	0	1	1	2	XX
8	0	1	2	1	XX
9	0	1	3	0	VP
10	0	2	0	2	VA
11	0	2	1	1	VA
12	0	2	2	0	VP
13	0	3	0	1	VA
14	0	3	1	0	VA
15	0	4	0	0	VA
16	1	0	0	3	XX
17	1	0	1	2	XX
18	1	0	2	1	CP
19	1	0	3	0	CP
20	1	1	0	2	CV
21	1	1	1	1	CV
22	1	1	2	0	CV
23	1	2	0	1	CV
24	1	2	1	0	CV
25	1	3	0	0	VA
26	2	0	0	2	CR
27	2	0	1	1	CR
28	2	0	2	0	CR
29	2	1	0	1	CR
30	2	1	1	0	CR
31	2	2	0	0	CV
32	3	0	0	1	CR
33	3	0	1	0	CR
34	3	1	0	0	CR
35	4	0	0	0	CR

(b): if 2 classes are CR and at most 1 of the other 2 classes is VA;

VA: (a): if 3 or 4 classes are VA;

(b): if 2 classes are VA and not any one class is CR;

PL: (a): if 4 classes are PL;

(b): if 3 classes are PL and 1 is UN;

CV: the pixel may be a crest or valley, if the number of CRs and VAs are non-zero and the difference between the number of CRs and VAs is at most one.

CP: the pixel may be a crest or plateau, if there is one CR and PL is in majority;

VP: the pixel may be a valley or plateau, if there are one or two VA and the rest PL;

XX: the pixel may be a crest or valley or plateau in all other cases.

It may be observed that even if along any direction at least one CR is present, the possibility of that pixel to be labeled CR is not ruled out; it may be labeled as either CV, CP or XX, thus keeping the option of CR at the next stage of classification.

5.4.2 Final classification

In Table 5.1, the classes CR, VA and PL denote the final classes. However, the criteria of being strongly classified, is not satisfied by some of the cases in Table 5.1 which are the ambiguous classes CV, VP, CP and XX. These pixels are finally classified in the second pass by inspecting the presence of the unambiguous classes in its neighborhood. This method can be viewed as the multilocal approach suggested in [88]. For the ambiguously classified pixels P belonging to CV, CP, VP or XX, we define a neighborhood $\mathcal{R}(P)$ whose size is determined by some criteria (as stated next under the heading Determination of $\mathcal{R}(P)$). In $\mathcal{R}(P)$, we find the average value of the pixels belonging to the unambiguous classes (CR, VA and PL). Let in $\mathcal{R}(P)$, the average gray value of the pixels belonging to CR be $Ave(CR)$. Similarly, $Ave(VA)$ and $Ave(PL)$ be the average gray values of pixels classified as valley and plateau respectively. For any pixel $P \in CV$, we classify it to either crest or valley based on the closeness of the gray value $L(i,j)$ of P to $Ave(CR)$ or $Ave(VA)$ i.e. if $|L(i,j) - Ave(CR)| \leq |L(i,j) - Ave(VA)|$, assign P to CR, else assign it to VA. Similarly, assign CP to CR or PL; and VP to VA or PL. The pixel $P \in XX$ is assigned to either CR, VA or PL for which its gray-value difference is minimum.

Determination of $\mathcal{R}(P)$

$\mathcal{R}(P)$ is a square region of size $w \times w$ centered at P. We estimate w from the domain knowledge of fingerprints. We know the approximate range of the number of ridges in a typical fingerprint. Let the number of ridges be approximately k. The size of the fingerprint image is $N \times M$. Let l be the width of the ridges in terms of pixels. So, the image dimension can be estimated in terms of k, l and the inter-ridge distance

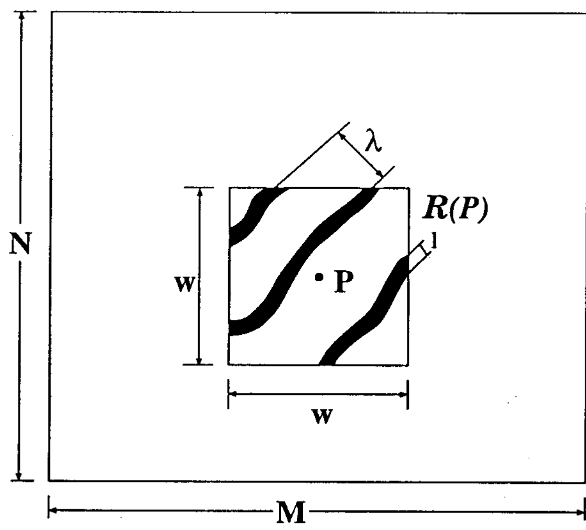


Figure 5.4: Determination of $\mathcal{R}(P)$

λ. Therefore, the approximate inter-ridge distance λ can be estimated roughly as (see Figure 5.4)

$$
\begin{aligned}
(\lambda + l)k &= \sqrt{N^2 + M^2} \\
\text{or } \lambda &= \frac{\sqrt{N^2 + M^2}}{k} - l
\end{aligned}
$$

The size of $\mathcal{R}(P)$ should be such that it includes at least some of the crest lines. To include, say, r crest lines, w should be equal to $(r\lambda)/\sqrt{2}$.

5.4.3 Thinning

After the final classification, the image contains 2/3-pixels thick crest lines, valley lines, and the rest plateau regions, (see Figure 5.5(a)). The valley and plateau pixels are treated as background and the crest pixels are treated as foreground object. The resulting image is a binary image with only crests as objects. Any of the standard thinning techniques, widely reported in literature [82, 124], can now be used on the binary image to obtain one-pixel thick crest lines (see Figure 5.5(b)). Figures 5.5(a) and (b) are perspective projections of a part of the original pixel matrix.

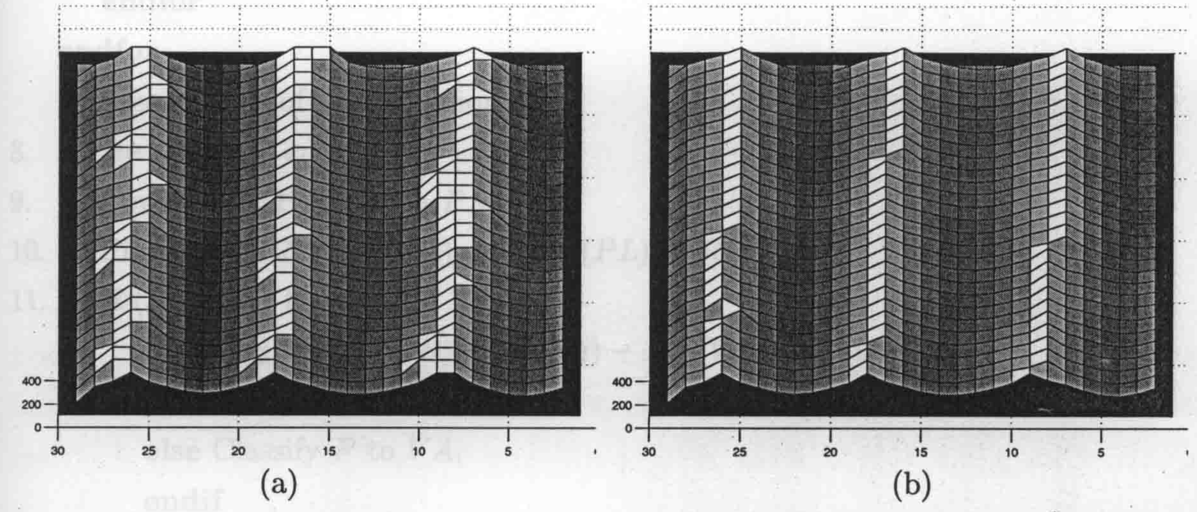


Figure 5.5: (a) Detected Crest after pixel classification, (b) same after thinning

5.4.4 Algorithm

Finally, we present the entire algorithm for detecting a crest, valley or plateau in a gray-level fingerprint image.

Algorithm Crest_Detect

Input: A Gray level fingerprint image I of size $N \times M$

Output: Identification of crest/valley

/* First Pass of Classification */

1. **for** $(i \leftarrow 2; i \leq N$, i++)
2. **for** $(j \leftarrow 1; j \leq M$, j++)
3. Calculate a, b, c, d, e, f, g, h following equation 5.7;
4. Calculate $sign(a, b)$, $sign(c, d)$, $sign(e, f)$ and $sign(g, h)$ and classify P along each of the directions to $CR/VA/PL/UN$;
5. Use Look-Up-Table (LUT) given in Table 5.1 to classify P to $CR/VA/PL/CV/CP/VP/XX$.
6. **if** $(P \in \{CV, CP, VP$ or $XX\})$ **then**

```
7.            Put the pixel location and gray value of P in a list 𝓛;
          endif
       endfor
    endfor
    /* Second Pass of Classification */
8.  while (𝓛 is not empty) do
9.     Consider 𝓡(P) for each P in 𝓛;
10.    Find Ave(CR), Ave(VA) and Ave(PL) in 𝓡(P);
11.    if (P ∈ CV) then
          if (|L(i,j) − Ave(CR)| ≤ |L(i,j) − Ave(VA)|) then
             Classify P to CR;
          else Classify P to VA;
          endif
       endif
12.    else if (P ∈ CP) then
          if (|L(i,j) − Ave(CR)| ≤ |L(i,j) − Ave(PL)|) then
             Classify P to CR;
          else Classify P to PL;
          endif
       endif
13.    else if (P ∈ VP) then
          if (|L(i,j) − Ave(VA)| ≤ |L(i,j) − Ave(PL)|) then
             Classify P to VA;
          else Classify P to PL;
          endif
       endif
14.    else if (P ∈ XX) then
          if (|L(i,j) − Ave(CR)| ≤ |L(i,j) − Ave(VA)| and
          |L(i,j) − Ave(CR)| ≤ |L(i,j) − Ave(PL)|) then
             Classify P to CR;
          else if (|L(i,j) − Ave(VA)| ≤ |L(i,j) − Ave(PL)| then
             Classify P to VA;
```

```
        else Classify P to PL;
        endif
    endif
endwhile
```

5.5 Space and Time Complexity

Space is required to store the image pixel matrix I, the Look-Up-Table, which takes constant space, and the list $\mathcal{L}$. The list $\mathcal{L}$ cannot grow beyond the size of the pixel matrix I. So, total space required is $O(N \times M)$, i.e. when $M \approx N$, the space is $O(N^2)$.

Step 3 of the algorithm in the previous section takes time proportional to the neighborhood size along which the directional averaging is done, and therefore, is constant. Step 4 computes the first difference pairs and their *signs* for each pixel, and takes constant amount of time. Step 5 uses the Look-Up-Table and takes unit time for finding one of the 35 exclusive and exhaustive possibilities. Steps 6-7 involve adding an ambiguous pixel to the list $\mathcal{L}$, taking constant time. Thus, for each pixel, a constant amount of time is needed. Hence, for the entire pixel matrix the first pass for classification takes time $O(N \times M)$, i.e., $O(N^2)$, if $M \approx N$. Steps 8-14 are needed for final classification given in Section 5.4.2, where the ambiguous pixels are classified. For each pixel in $\mathcal{L}$, we calculate the average of the gray values of pixels in the classes CR, VA and PL in a predefined fixed neighborhood $\mathcal{R}(P)$ that takes constant amount of time. All other computations take constant time. Thus, the worst case time complexity of the final classification algorithm is $O(N^2)$.

5.6 Evaluation and Results

5.6.1 Evaluation Criteria for Ridge/Valley finding Algorithms

López et al. [88] devised a set of desirable properties for clean and robust extraction of ridges. These are:

- P1. Irrelevant ridges should not be detected.
- P2. Salient ridges should always be detected.
- P3. Ridges extracted from a continuous object must be continuous.
- P4. A creaseness measure should have a similar value along the crease and a much higher value than the region around it.
- P5. Small perturbations of the image should not alter greatly the shape or the location of the ridges.
- P6. Ridges should run as closely as possible through the center of anisotropic regions.

For fingerprint images, any minutiae extraction method involves a lot of preprocessing even on the one-pixel thick binary image [39]. In case of fingerprint images, retention of crest lines are more important. Crest lines should not be missed, if irrelevant ridges occur, they are taken care of during minutiae extraction. So, the property P2 is more important to our case than P1. Property P4 does not hold in our case, as our method is a classification scheme and not a creaseness measure. The properties are subjective in nature and cannot be quantified. We provide experimental results in support.

5.6.2 Results

The fingerprint images used in our experiment were taken from the NIST Special Database 14 and NIST Special Database 4 [21, 152, 153] that consist of various samples of size 480×512 with 500 dpi resolution and 256 gray levels. Figure 5.6 is a

flow diagram that depicts where our classification scheme can be used during thinning fingerprint images for minutiae extraction from a gray-level fingerprint image. The proposed algorithm was implemented in C on a Sun_Ultra 5_10, Sparc, 233 MHz, the OS is SunOS Release 5.7 Generic. The average CPU time for classification was found to be 0.41 secs. The original image, classified image and the thinned ridge lines for a particular image are shown in Figure 5.12. Figures 5.13(a) to 5.20(a) (in the left column) show the original image and Figures 5.13(b) to 5.20(b) (in the right column) show the thinned ridge line superimposed on the original image. A complete sequence of images at different stages of minutiae extraction following the steps in Figure 5.6, is shown in Figure 5.21.

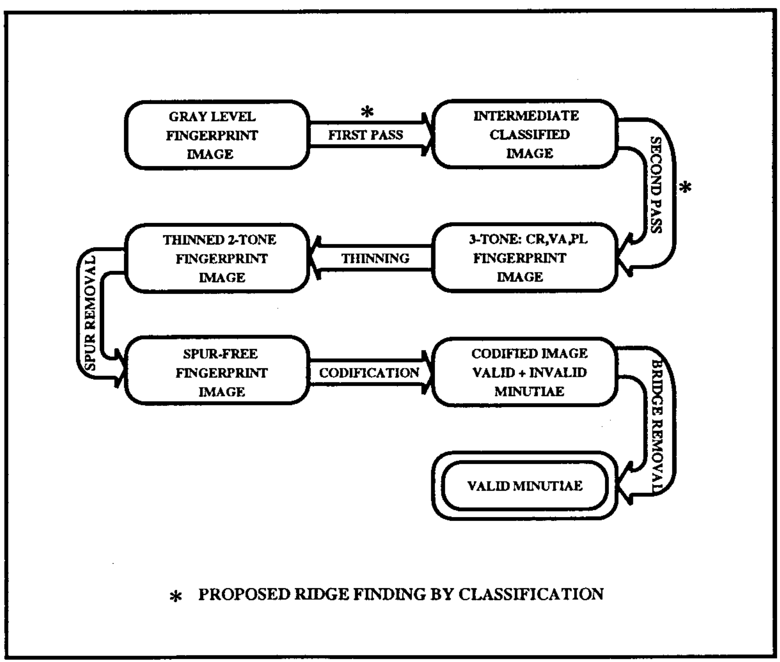


Figure 5.6: A flow chart showing the steps of minutiae extraction

5.7 Parallel Algorithm and VLSI Architecture

5.7.1 Parallel Algorithm

We describe below a parallel algorithm for classifying each pixel in the image to any of the classes $\{CR, VA, PL, CV, CP, VP, XX\} \in C$.

Algorithm Classify_Parallel
Input: A Gray Level Fingerprint Image I of size $N \times M$
Output: A Classified Image into the seven classes as in Section 5.4.2.

1. **for** ($j \leftarrow$ column number 1 to M) **do in parallel**
2. Calculate directional averages;
3. Calculate a, b, c, d, e, f, g and h for each pixel in the jth column;
4. Pair up corresponding first difference pairs (a, b), (c, d), (e, f) and (g, h);
5. Find the number of CR, VA, PL and UN along each of the directions;
6. Use Look-Up-Table (LUT) given in Table 5.1 to find out the element in C to which the pixel belongs;

endfor

5.7.2 VLSI Architecture

We present an on-chip implementation of the first pass of our pixel classification scheme proposed in Sections 5.3.1 and 5.3.2. The first pass of the algorithm over the image-pixel matrix resolves pixels to any of the CR, VA, PL, CV, VP, CP and XX as shown in Table 5.1.

The algorithm given in Section 5.4.4 being highly modular has an inherent parallelism to it. We exploit those properties to design a pipeline architecture for implementing the first pass of classification including the directional averaging. At each clock pulse, a column of the pixel matrix is fed into the pipeline. The simple directional averaging

Table 5.2: Availabilty of pixels at different clock pulses

	Pixels Needed	Input line at which it is available	Time at which which it is available	Pixels Needed	Input lines at which they are available	Time at which they are available
a	(i,j)	R_i	t_j	$(i-1,j)$, $(i-2,j)$	R_{i-1}, R_{i-2}	t_j
b	(i,j)	R_i	t_j	$(i+1,j)$, $(i+2,j)$	R_{i+1}, R_{i+2}	t_j
c	(i,j)	R_i	t_j	$(i-1,j-1)$, $(i-2,j-2)$	R_{i-1}, R_{i-2}	t_j-1, t_j-2
d	(i,j)	R_i	t_j	$(i+1,j+1)$, $(i+2,j+2)$	R_{i+1}, R_{i+2}	t_j+1, t_j+2
e	(i,j)	R_i	t_j	$(i,j-1)$, $(i,j-2)$	R_i	t_j-1, t_j-2
f	(i,j)	R_i	t_j	$(i,j+1)$, $(i,j+2)$	R_i	t_j+1, t_j+2
g	(i,j)	R_i	t_j	$(i+1,j-1)$, $(i+2,j-2)$	R_{i+1}, R_{i+2}	t_j-1, t_j-2
h	(i,j)	R_i	t_j	$(i-1,j+1)$, $(i-2,j+2)$	R_{i-1}, R_{i-2}	t_j+1, t_j+2

requires holding back data in columns that were fed in previous clocks. The circuit element Ave_d is shown in Figure 5.7. The processing elements for calculating each of the first difference pairs are shown in Figure 5.8. It adds two 8-bit numbers, representing the gray values of the pixels, and uses a serial shift right register to perform a division by two for generating the average. The size of the register should be equal to 9 bits to accommodate the added value of two 8-bit numbers. Table 5.2 shows the input lines and corresponding clock pulses at which the respective pixel values would be available for calculating the first differences a, b, c, d, e, f, g and h for the pixel at location (i,j). The column indices of the pixels match with the clocks at which they would be available. As an example, $c = L(i,j) - (L(i-1,j-1) + L(i-2,j-2))/2$, is calculated

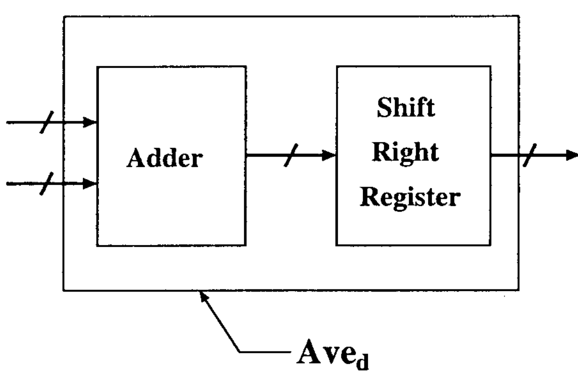


Figure 5.7: Circuit for finding the directional average

as follows. The pixels corresponding to $L(i-2, j-2)$ and $L(i-1, j-1)$ are available at the input lines R_{i-2} and R_{i-1} at the clocks $j-2$ and $j-1$ respectively. To hold them back, we use a bank of D flip-flops. Obviously, line R_{i-2} is to be retained for two clock pulses, and line R_{i-1} should be held for one clock pulse. The outputs from the D flip-flops corresponding to lines R_{i-2} and R_{i-1} are fed to the circuit block Ave_d for computing the directional average. The pixel corresponding to $L(i, j)$, available on the line R_i at clock t_j is to be held during which the circuit block Ave_d computes the directional average. The circuits given in Figure 5.8 show separately the requirement for calculating each first difference pair. To pair up the first difference pairs and calculate $sign(a, b)$, $sign(c, d)$, $sign(e, f)$ and $sign(g, h)$, we need comparator circuits [92] (see Figure 5.9). It may be noted that, a, b, c, e and g are available at the comparator input after the t_j^{th} clock pulse, while d, f and h are available at the comparator input after the $(t_j + 2)^{th}$ clock pulse. Thus, the outputs of the comparator circuits are to be held for two clock pulses for further processing.

The comparator block C_j (see Figure 5.9) has five 8-bit magnitude comparators, one each for a, b, c, e and g. After the clock pulse t_j, C_j calculates *signs* of a, b, c, e and g for the pixel at location (i, j) of the image pixel matrix. Comparator block C_{j+2} (see Figure 5.9) has three 8-bit magnitude comparators, one each for d, h and f. C_{j+2} calculates *signs* of d, h and f for the pixel at location (i, j) after the $(t_j + 2)^{th}$ clock pulse. After clock pulse $(t_j + 2)$, C_j calculates *signs* of a, b, c, e and g corresponding to the pixel at location $(i, j+2)$ of the pixel matrix; C_{j+2} calculates *signs* of d, h and f

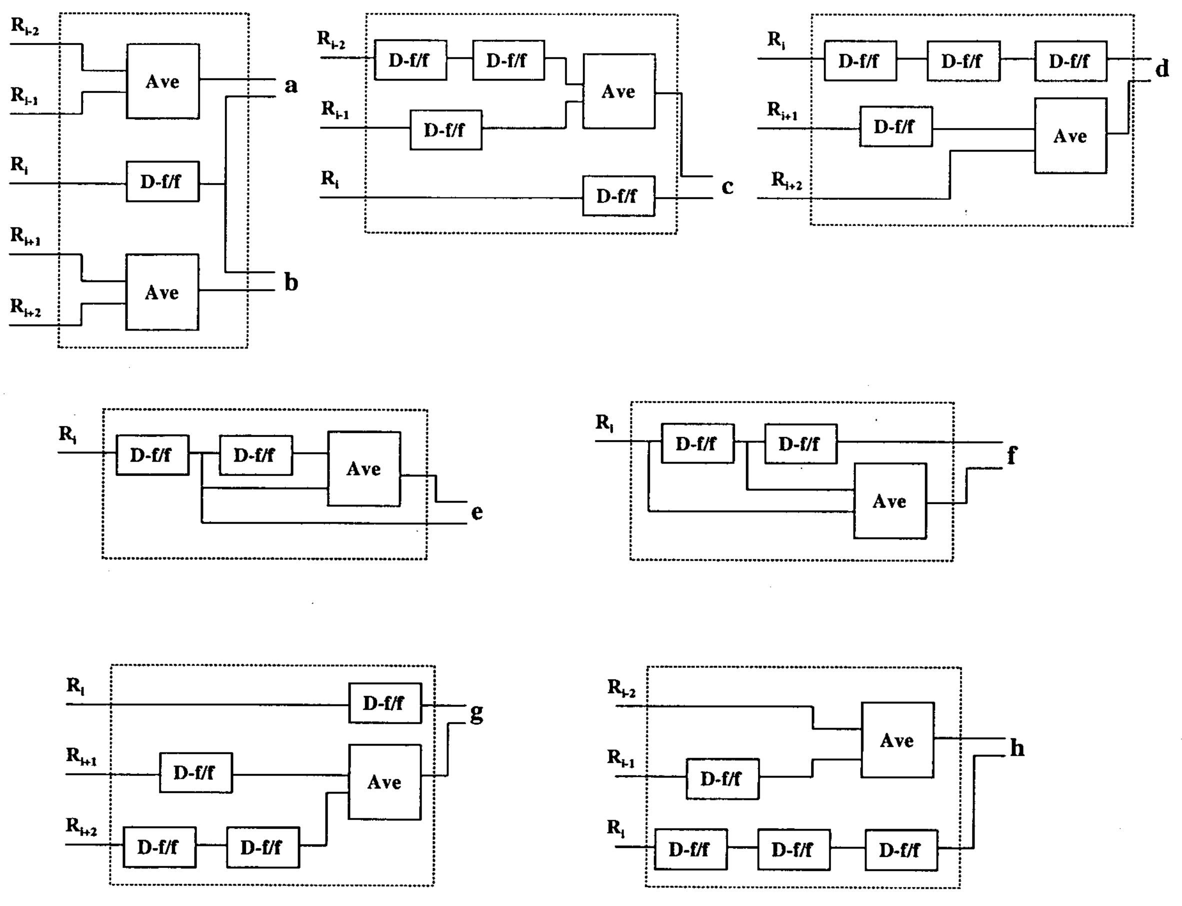


Figure 5.8: Circuits for computing the first difference pairs

corresponding to the pixel at location (i, j). The *signs* of h, d and f are available two clock pulses after *signs* of a, b, c, e and g are available. So, the outputs from C_j are held for two clock pulses by the two banks of D flip-flops, called D_4 (see Figure 5.9). The bus-width of the outputs from the comparators C_j and C_{j+2} is 3, since the outputs of the magnitude comparator can be either $+$, or $-$, or 0.

The outputs from the comparators are fed into four combinational circuits, P_1, P_2, P_3 and P_4 for finding $sign(a, b)$, $sign(c, d)$, $sign(e, f)$ and $sign(g, h)$ respectively. The circuit diagram of $P_i, i = 1, 2, 3, 4$ is shown in Figure 5.10. The combinational circuits $P_i, i = 1 \ldots 4$ implement the table given in Figure 5.3. Along any of the directions (a, b), (c, d), (e, f) or (g, h), we can have either a crest, or valley, or plateau, or undecided. Thus, out of the four output lines (each line being 1 bit wide) from each of P_1, P_2, P_3

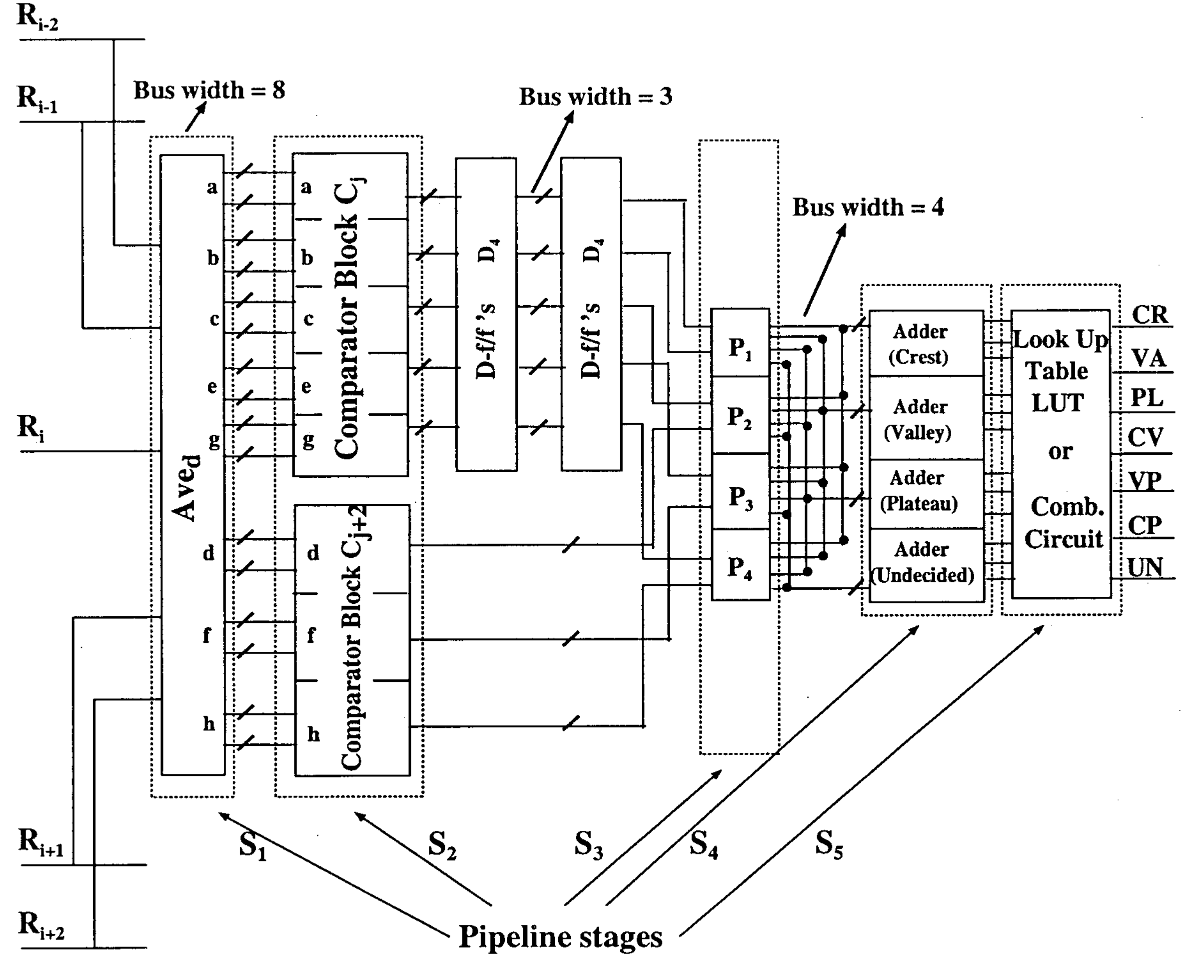


Figure 5.9: A sample circuit for the first pass classification of a row R_i

and P_4, only one line can go high indicating the presence of crest or valley or plateau or undecided pixel.

Next, we need four adders. The CR, VA, PL, and UN (all 1-bit) output lines shown in Figure 5.9 are fed respectively to Crest-adder, Valley-adder, Plateau-adder and Undecided-adder. As we have four directions, the maximum number of occurrences of any of crest, valley, plateau or undecided can be four. Therefore, in the adders, we require three bits to represent the sum. As we have four adders, the bus-width for the Look-Up-Table LUT is twelve. The outputs from the adders are fed into the LUT implementing the classification scheme of Table 5.1. The LUT outputs any one of the seven classes (CR, VA, PL, CV, CP, VP or XX) and the corresponding output line

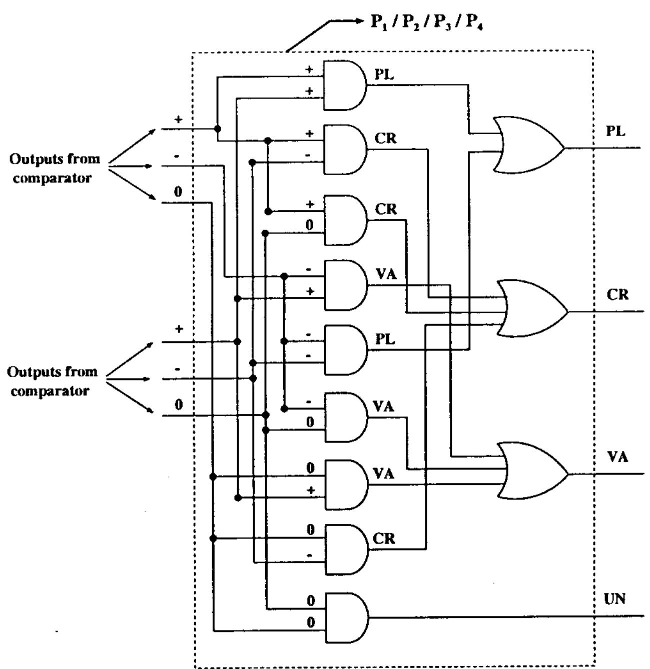


Figure 5.10: Circuit for each P_i, $i = 1, 2, 3, 4$

is enabled. The *LUT* (Figure 5.9) decodes the 12-bit input to one of the 35 entries in it. Thus, a location of the *LUT* is addressed, and any one of the corresponding seven lines is enabled.

To summarise, the computation of the first pass of classification resembles a linear pipeline [61] with five stages, namely S_1, S_2, S_3, S_4 and S_5 as shown in Figure 5.9. An entire column is fed into the pipeline at the same time. Stage S_1 computes the directional average. Stage S_2 consists of magnitude comparators. The two banks of flip-flops, shown as D_4 in Figure 5.9, are used to insert the required delay. Stage S_3 is used to form the pairing of the *signs* of the first difference pairs; the outputs are added in the stage S_4. A *LUT* is used in stage S_5 to decode the input to any one of the outputs, namely CR, VA, PL, CV, CP, VP, or XX. Figure 5.11 shows a schematic circuit diagram for classifying an image having 8 rows.

After the first pass, each of the $N \times M$ pixels is classified into one of the seven classes. These pixels are to be stored in a temporary storage for further processing in the second pass.

5.7.3 Time Complexity

The number of stages in the linear pipeline is five. The clock period of the linear pipeline is determined by the longest pipeline stage. Here, stage S_1 is the longest one requiring 9 clock cycles (8 clock cycles for addition followed by a single clock for serial shift right register). Now, the number of steps in the linear pipeline is $M+5$. Thus, the number of clock cycles required to perform the classification is $9(M+5) = 9M + 45$, i.e., $O(M) \approx O(N)$. The number of processing elements is clearly $O(N)$. The time complexity of the first pass of the sequential classification is $O(N^2)$ (see Section 5.5). Using $O(N)$ processing elements, the first pass of the classification can be performed in $O(N)$ parallel time of execution. Thus, the speed up [87] achieved is $O(N)$.

5.7.4 Circuit Cost

The components used to implement the VLSI architecture for processing of each row of an $N \times M$ image are given below. The requirements for each stage S_i are shown separately:

S_1: (a) 3 banks of D flip-flops each having 8 flip-flops making a total of 24 flip-flops. Note that, each line R_i requires a maximum of 3 flip-flops (see Figure 5.8).

(b) 8 pieces of 9-bit adder and 8 pieces of 9-bit shift registers.

S_2: 8 pieces of 8-bit magnitude comparators.

S_3: 36 pieces of 2-input AND gates (9 each for P_1, P_2, P_3 and P_4) and 12 pieces of 3-input OR gates (3 each for P_1, P_2, P_3 and P_4).

S_4: 4 pieces of 3-bit adders.

S_4: one piece of Look-Up-Table, LUT.

Apart from these, for each row, we require two delay stages for synchronization. The bus width at this particular stage is three. Since each line requires 3 flip-flops, the total number of D flip-flops required for five lines and two stages, is 30.

5.8 Conclusions and Discussions

In this chapter, the procedure for ridge finding for a gray level fingerprint image is viewed as a combinatorial problem. We enumerated the exhaustive combinatorial possibilities that a particular pixel can have in a digital image landscape in terms of the first difference pairs. Based on the possibilities, the pixels are classified into three different classes using a two-pass algorithm. The combinatorial possibilities stored as a look-up-table provide us a tool to implement the first pass of the classification scheme on-chip. The construction of the LUT is based on some heuristics justified by the experimental evidence on the NIST14 and NIST4 databases. A better design of the LUT including magnitudes in addition to signs needs further investigation. The generating function has to be suitably changed for calculating the combinatorial possibilities accordingly.

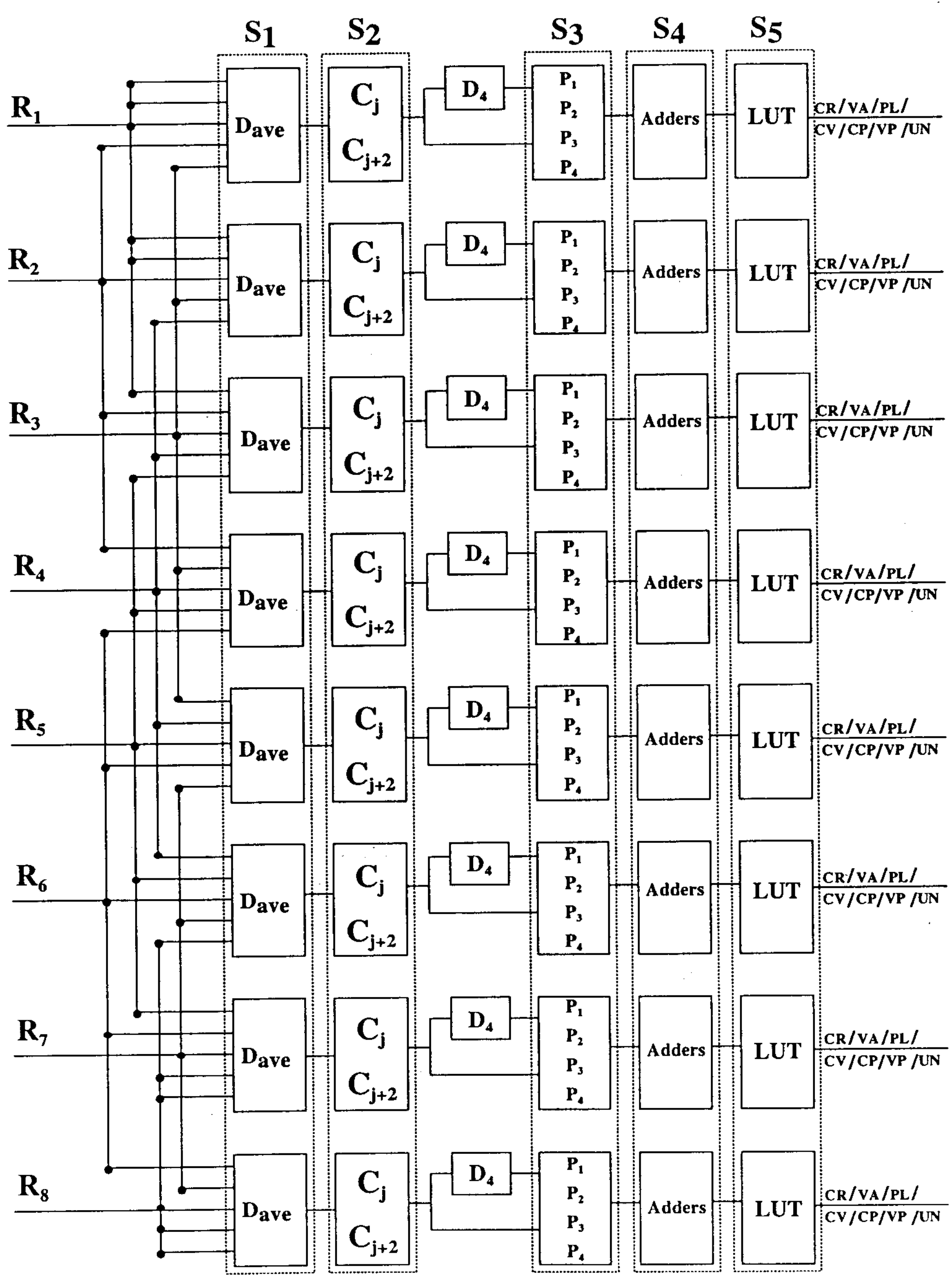
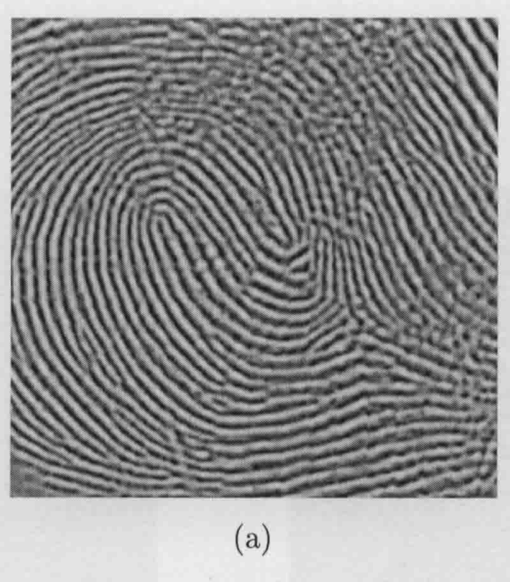
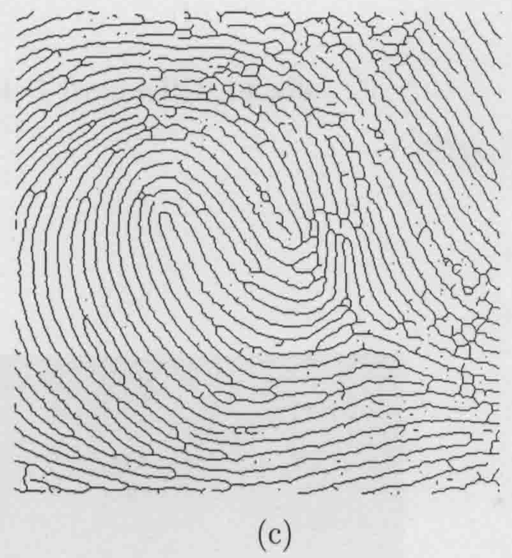


Figure 5.11: A schematic circuit for classifying pixels in an image having 8 rows



(a)

(b)



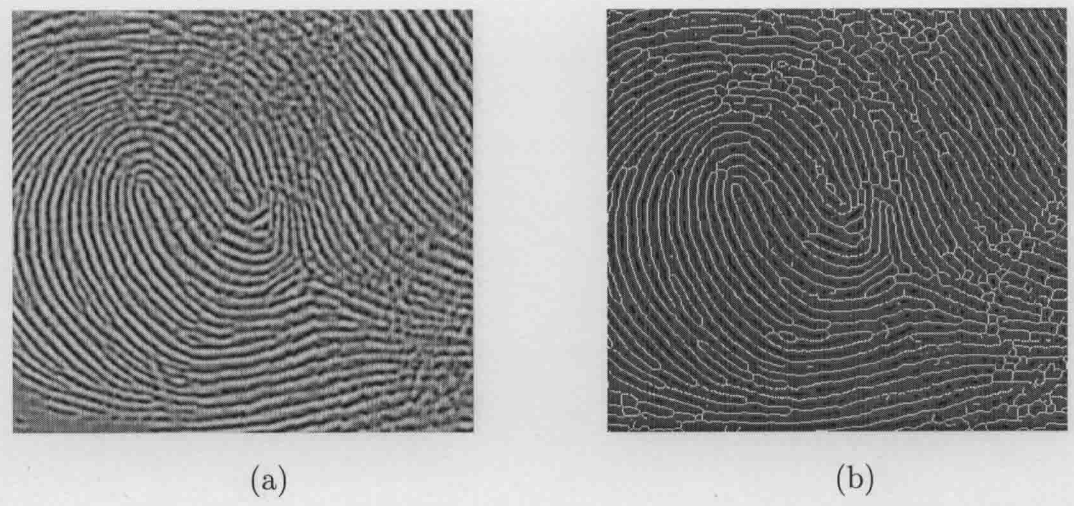
(c)

Figure 5.12:

(a) Original image

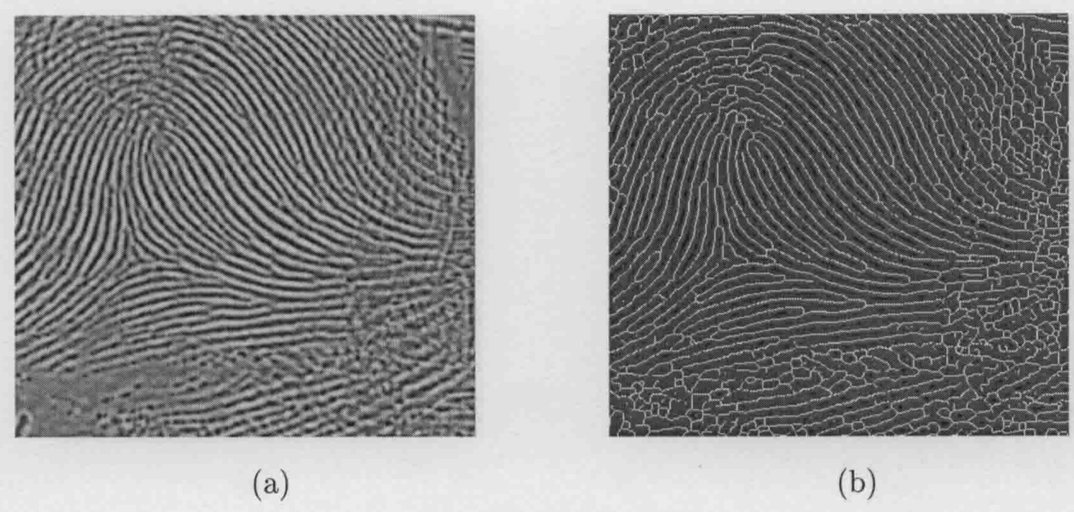
(b) Ternary image after pixel classification (Crest, Valley, Plateau)

(c) Binary image after thinning (Crest, Background)



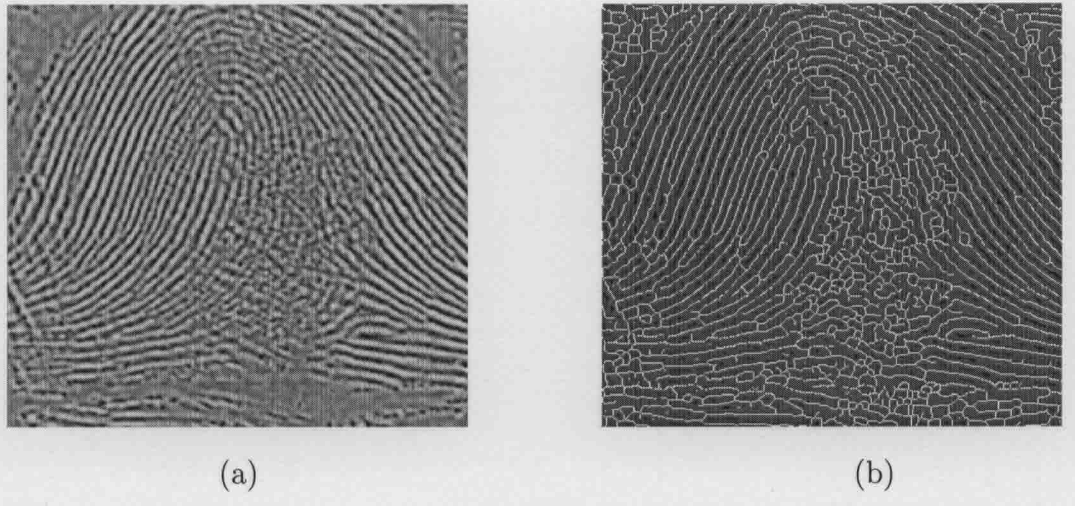
(a) (b)

Figure 5.13: Fingerprint image sample 1 from NIST 14 sdb



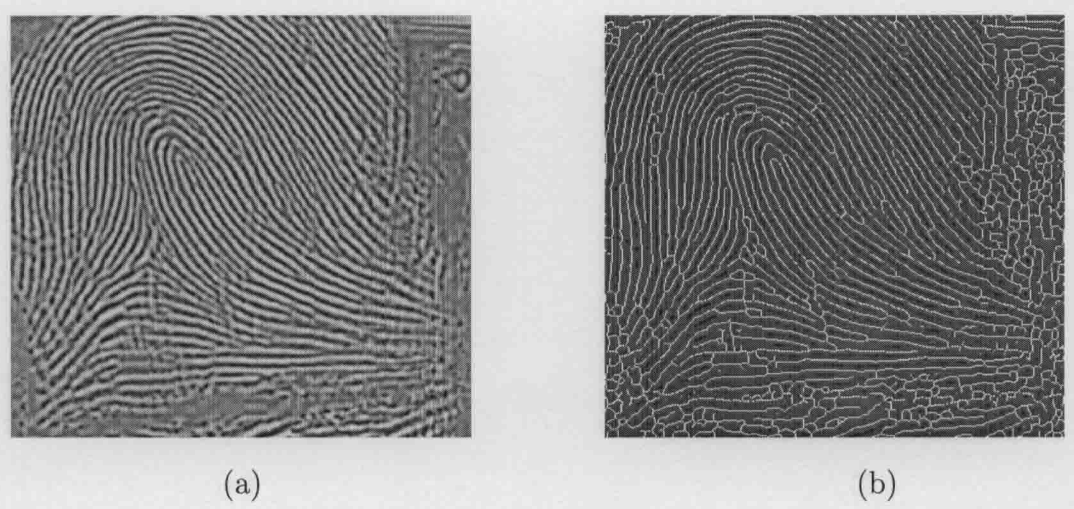
(a) (b)

Figure 5.14: Fingerprint image sample 2 from NIST 14 sdb



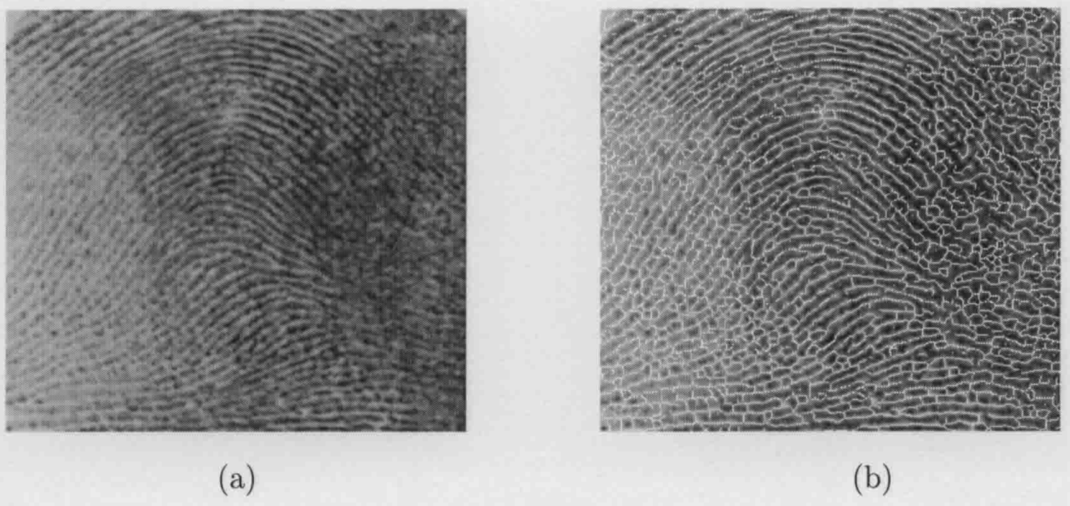
(a) (b)

Figure 5.15: Fingerprint image sample 3 from NIST 14 sdb



(a) (b)

Figure 5.16: Fingerprint image sample 4 from NIST 14 sdb

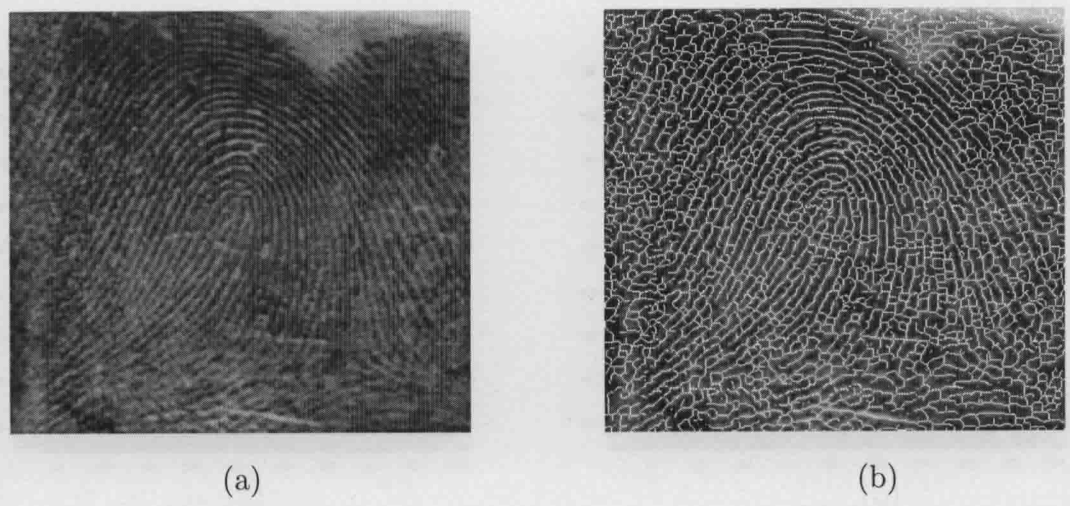


(a) (b)

Figure 5.17: Fingerprint image sample 1 from NIST 4 sdb

(a) (b)

Figure 5.18: Fingerprint image sample 2 from NIST 4 sdb

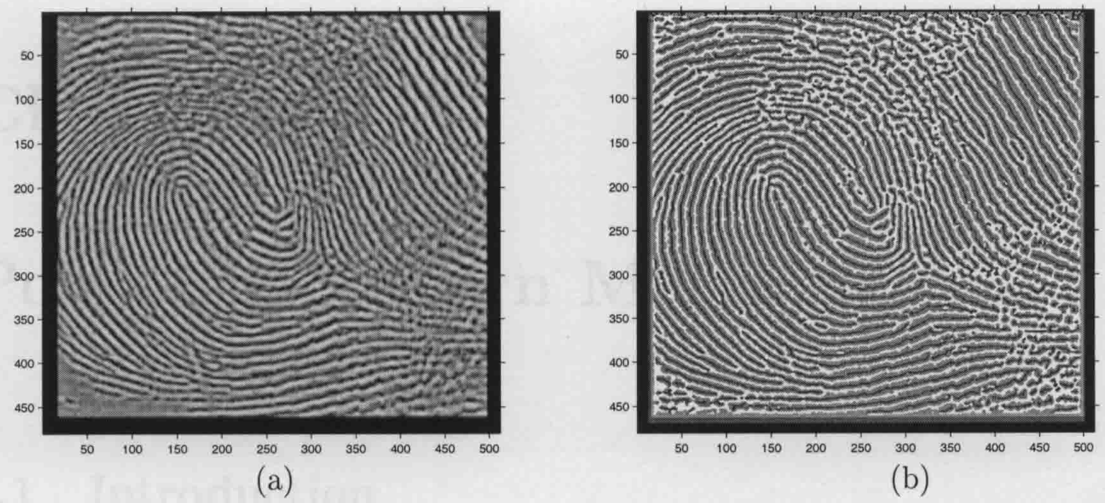


(a) (b)

Figure 5.19: Fingerprint image sample 3 from NIST 4 sdb

(a) (b)

Figure 5.20: Fingerprint image sample 4 from NIST 4 sdb



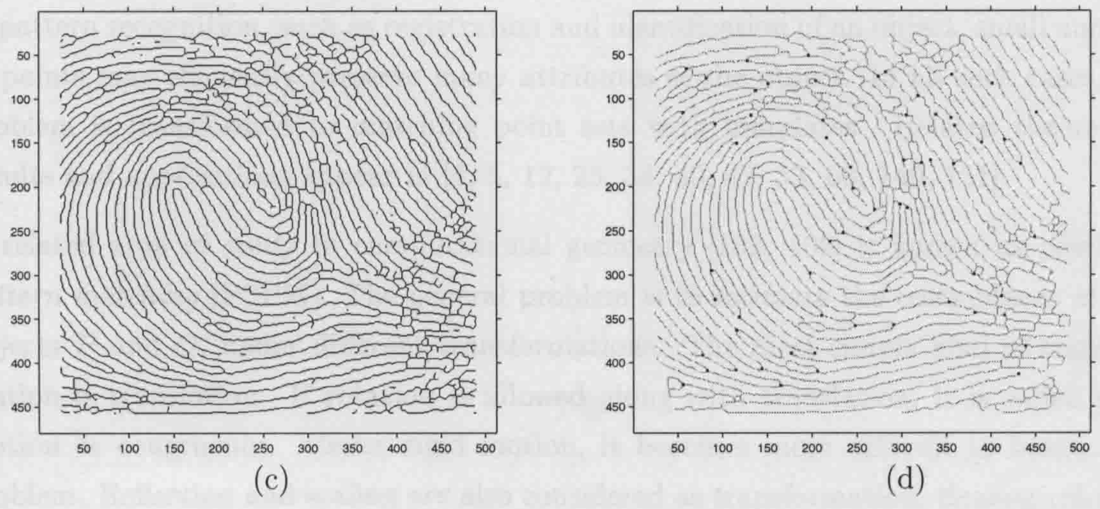


Figure 5.21: Minutiae extraction from fingerprint image after classification and thinning

Chapter 6

Point Set Pattern Matching

6.1 Introduction

In computer vision and related applications, point sets represent some spatial features like spots, corners, lines, curves in various images pertaining to fingerprints, natural scenery, maps, air traffic, astronomical maps, etc. Pose estimation involves estimation of object position and orientation relative to a model reference frame. In many problems of pattern recognition, such as registration and identification of an object, small number of points may efficiently preserve many attributes of the object. In all such cases, the problem is transformed to matching point sets with templates. Related theoretical results and applications appear in [4, 5, 12, 25, 38, 40, 47, 53, 68, 140, 151].

A related area of study in computational geometry [102, 109] is known as *point set pattern matching* (PSPM). The general problem is to estimate the resemblance of two objects P and Q, under different transformations. The most simple kind of transformation is translation. If rotation is allowed along with translation, it is called rigid motion or congruence. Under rigid motion, it becomes more difficult to handle the problem. Reflection and scaling are also considered as transformation. Scaling refers to stretching the object by a certain factor λ about the origin. Combination of translation and scaling is called homothetics, and combination of rotation, translation and scaling is called similarity. An isometric mapping is a combination of translation, rotation and

reflection. Under these transformations, given a set P of n points and a set Q of k points, P may be:

(i) equal to Q,
(ii) a translation of Q,
(iii) a rotation of Q,
(iv) a reflection of Q,
(v) an isometric mapping of Q,
(vi) a scaled copy of Q (may be combined with an isometry).

The problem of PSPM can be classified into three classes:

(a) Exact point set pattern matching: Alt et al. in [8] reported methods for finding congruences between two point sets P and $Q \subset \mathbb{R}^d$ of n points under rigid motions. In [14], Atkinson showed that exact point pattern matching can be easily reduced to string matching [76].

(b) Approximate point set pattern matching [6, 8, 13]: The approximate point pattern matching is more realistic in the actual application. Given two finite sets of points P and Q, the problem is to find an approximate matching under some transformation, i.e., for each point $q_i \in Q$, find its match, $p_i \in P$ such that q_i lies in the ϵ-neighbourhood of p_i (a circle of radius ϵ centered at p_i). Variants of this problem are:

  (i) cardinality of P and Q are equal and the objective is to check the existence of a one-to-one matching.

  (ii) more than one point of one set may be matched with the same point of the other set.

(c) Partial point set pattern matching: Let P be the larger set of points (called the sample set) and Q be a small set of points (called a query set). The problem is to ascertain whether Q has a match in P, i.e., whether a subset of P matches with Q. In one dimension, the problem is solved using the following procedure: the points are sorted with respect to their coordinates and a sequence of distances is

found among the pair of consecutive members. The same thing is done for the query point set Q also. Using this idea, Rezende and Lee [120] showed that given a set Q of k points and a set P of $n \geq k$ points in the d-dimensional Eucledian space, determining whether a k-subset of P matches with Q under translation, rotation and scaling can be done in $O(kn^d)$ time. The notion of circular sorting [85] is utilized to implement the algorithm.

Detailed surveys of PSPM problems in computational geometry appear in [7, 43].

6.2 An Existing Algorithm

Let $P = \{p_1, p_2, \ldots, p_n\}$ be the sample set and $Q = \{q_1, q_2, \ldots, q_k\}$ be the query set, with $k \leq n$. Rezende and Lee [120] have used computational geometric techniques to design an algorithm for finding a k-subset of P, that matches with Q, under translation, rotation and scaling. The problem can be solved naively by checking all possible $\binom{n}{k}$ subsets of P for a match with Q. However, this approach has an exponential complexity. To improve the search time complexity, an extension of *circular sorting* in higher dimensions [85, 120] is used to achieve an orderly traversal of the point sets. It may be noted that the points in d-dimensional space, $d \geq 2$, lack a total order. Goodman and Pollack [46] have reported a characterization in terms of a matrix formed from the order type of the set of points in d-dimensional space. It is a canonical ordering of the set of points. In [120], the authors pointed out that the method given in [46], though elegant, may not be of much use in subset matching. Rezende and Lee extended the idea of canonical ordering in [46], to the concept of circular sorting, that basically imposes a partial order on the points.

6.2.1 Circular Sorting

Linear sorting is one of the most widely studied problems in computer science [3, 75]. Unfortunately, the lack of total ordering for points in dimensions higher than one, invalidates linear sorting e.g. sorting a set of n points $P = \{p_1, p_2, \ldots, p_n\}$ in the plane.

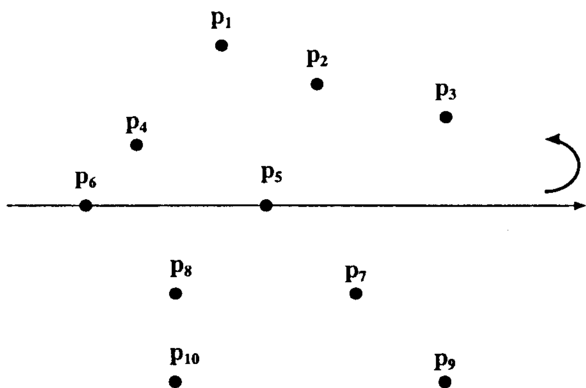


Figure 6.1: Circular sorting around $p_5 = \{p_5, p_3, p_2, p_1, p_4, p_6, p_8, p_{10}, p_7, p_9\}$

Let $p_i \in P$. Then the points in $P \setminus \{p_i\}$ can be circularly sorted around p_i by lexicographically sorting the points in $P \setminus \{p_i\}$ with respect to two keys:

(i) polar angles around p_i in a counter-clockwise direction and

(ii) their distances from p_i (see Figure 6.1).

Needless to say, there are different sorted orders for each p_i, where $i = 1, 2, \ldots, n$.

The simplest algorithm for circular sorting of a planar set P of n points considers each point $p_i \in P$ separately and finds the ordering of the other points by calculating their polar angle θ_i (with origin at p_i) and distance r_i from p_i, thus forming the tuple (θ_i, r_i) and then applying the lexicographic sorting algorithm on the tuples (θ_i, r_i). The entire process is repeated for all members in P. The lexicographic sorting centered at p_i takes time $O(n \log n)$. So, this naive algorithm needs $O(n^2 \log n)$ time.

The best known algorithm for this problem uses geometric duality and takes $\Theta(n^2)$ time [25, 85]. Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n points on a 2-D plane, and $P^* = \{p_1^*, p_2^*, \ldots, p_n^*\}$ be the dual lines [102] corresponding to the points $p_i \in P$. Let O be the center of the coordinate system. We consider the line Op_i, and the line at p_i perpendicular to Op_i. This divides the plane into four quadrants. If a point $p_j \in P \setminus \{p_i\}$ is in the first or third quadrant, the gradient of the line joining p_i, p_j is positive; otherwise it is negative. In the dual plane, the gradient of the line p_ip_j is the

x-coordinate of the point of intersection of p_i^* and p_j^*. In order to get the points sorted in angular order around the point p_i, we traverse along the line p_i^* in the dual plane from $-\infty$ to ∞, and observe the point of intersection of the members in $P^* \setminus \{p_i^*\}$ with p_i^*. If for a line p_j^*, its x-coordinate is observed to be negative (or positive), we check whether the point p_j belongs to first or third (or second or fourth) quadrant. Finally, it is added at the end of the list of that quadrant attched to p_i. If more than one member of $P^* \setminus \{p_i^*\}$ intersects at same point on p_i^*, and all of them lie in the same quadrant then the angle of the wedges of those lines with p_i^* in the dual plane indicates the distance of those points from p_i in the primal plane.

After processing all the intersection points on p_i^*, four lists are concatenated to get the sorted order of the points in $P \setminus \{p_i\}$ around p_i.

In an arrangement of lines in P^*, we can use topological line sweep to process the vertices of the arrangement. At each intersection point of p_i^* and p_j^*, p_i is inserted in the list of p_j and vice versa. Thus the whole process can be completed in $O(n^2)$ time. The arrangement of the dual lines for all the points has to be considered for the sorting process. Thus, the space complexity of this method is $O(n^2)$ in the worst case [85].

6.2.2 Point Set Pattern Matching on a 2-D Plane

Given a sample set $P = \{p_1, p_2, \ldots, p_n\}$ consisting of n points, and a query set $Q = \{q_1, q_2, \ldots, q_k\}$ of k points in the 2-D plane, where $k \leq n$, the objective is to find a k-subset M of P matching Q under translation and/or rotation provided such a match exists. The output of this algorithm is the set of points $M = \{p_{\alpha_1}, p_{\alpha_2}, \ldots, p_{\alpha_k}\}$, where $p_{\alpha_i} \in P \; \forall i = 1(1)k$. If no match exists, then $M = \phi$.

As the first step, we sort the set P circularly and generate $\mathcal{P}$ having n lists corresponding to ordering at each p_j, $j = 1(1)n$. Let $\mathcal{P}_j = \{(p_j, 0, 0), (p_j^{(1)}, \theta_j^{(1)}, r_j^{(1)}), (p_j^{(2)}, \theta_j^{(2)}, r_j^{(2)}), \ldots, (p_j^{(n-1)}, \theta_j^{(n-1)}, r_j^{(n-1)})\}$ be the list of points in P ordered circularly around p_j. The first term $(p_j, 0, 0)$ denotes the point p_j itself. Each $(p_j^{(i)}, \theta_j^{(i)}, r_j^{(i)})$ denotes the point $p^{(i)}$, its polar angle $\theta^{(i)}$ centered at p_j and distance $r_j^{(i)}$ from p_j. The query set Q is circularly sorted around q_1, generating $Q_1 = \{(q_1, 0, 0), (q^{(2)}, \theta_q^{(2)}, r_q^{(2)}), \ldots, (q^{(k-1)}, \theta_q^{(k-1)}, r_q^{(k-1)})\}$.

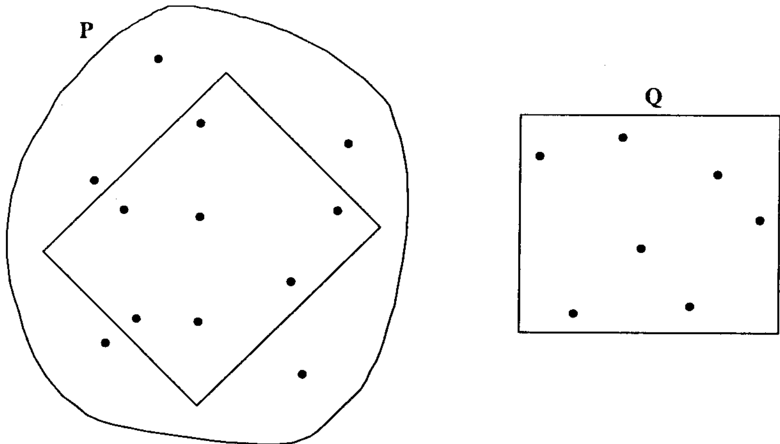


Figure 6.2: Pattern and query sets matched under translation and rotation

Then, we start the matching process. Anchor q_1 at a p_j of a row $\mathcal{P}_j$ of $\mathcal{P}$ and try for a match. Two pointers *p_ptr* and *q_ptr* are used. Initially *p_ptr* is fixed at $(p_j, 0, 0)$ of $\mathcal{P}_j$ and *q_ptr* is fixed at $(q_1, 0, 0)$ of Q_1. Now, *q_ptr* is moved one step ahead. *p_ptr* is moved over $\mathcal{P}_j$ till a distance $r_j^{(\alpha_1)} = r_q^{(2)}$ is found. Let $\theta_{diff} = \theta_j^{(\alpha_1)} - \theta_q^{(2)}$ (if such an $r_j^{(\alpha_1)}$ is not found in the entire list of $\mathcal{P}_j$, try for a match at another anchor point from the set P). Move *q_ptr* one step ahead and also start moving *p_ptr* over $\mathcal{P}_j$ till a distance $r_j^{(\alpha_2)}$ $= r_q^{(3)}$ and an angle $\theta_j^{\alpha_2}$ is found such that $\theta_j^{\alpha_2} - \theta_q^{(3)} = \theta_{diff}$. By proceeding thus, if *q_ptr* reaches the last element of Q_1 i.e. $(q^{(k-1)}, \theta_q^{(k-1)}, r_q^{(k-1)})$ and finds a distance $r_j^{(\alpha_k)}$ $= r_q^{(k-1)}$ and a polar angle $\theta_q^{(k-1)}$ such that $\theta_j^{\alpha_k} - \theta_q^{(k-1)} = \theta_{diff}$, then we say a match has been found and we report points $\{p_j^{\alpha_1}, p_j^{\alpha_2}, \ldots, p_j^{\alpha_k}\}$ as the k-subset of P matching Q. The rotation angle of the match is θ_{diff} and the translation is the difference of the coordinates of $p_j^{\alpha_1}$ and q_1.

Algorithm PSPM

Input: $P = \{p_1, p_2, \ldots, p_n\}$, $Q = \{q_1, q_2, \ldots, q_k\}$

Output: $M = \{p_{\alpha_1}, p_{\alpha_2}, \ldots, p_{\alpha_k}\}$.

1. Circularly sort P; it generates a $n \times n$ array $\mathcal{P}$, whose each element $\mathcal{P}[j][i]$ has
   (i) $\mathcal{P}[j][i].p$: the ith point when the circular sort is done centered at p_j;
   (ii) $\mathcal{P}[j][i].\theta$: $\theta_j^{(i)}$, the angle of the point $p_j^{(i)}$ centered at p_j;
   (iii) $\mathcal{P}[j][i].r$: $r_j^{(i)}$, the distance between the point $p_j^{(i)}$ and p_j.
2. Similarly, circularly sort Q around q_1 to generate a $1 \times n$ array

```
3.  for (j ← 1; j ≤ n; j ← j+1) /* loop for checking all rows of 𝒫 */
        p_temp ← 1;
4.      while (p_temp ≤ n − 1) do
            q_ptr ← 2; p_ptr ← p_temp + 1; p_temp ← p_ptr; flag←FALSE
5.          while (p_ptr ≤ n) do
6.              if (𝒫[j][p_ptr].r = Q1[q_ptr].r) then
7.                  if (flag=TRUE) then /* successive matching after the first one */
                        θdiff ← (𝒫[j][p_ptr].θ − Q1[q_ptr].θ;
8.                      if (θdiff=𝒫[j][p_ptr].θ - Q1[q_ptr].θ) then
9.                          if (q_ptr = k) then Report a match;
10.                         else                q_ptr ← q_ptr + 1; p_ptr ← p_ptr + 1;
                            endif
11.                     else   p_ptr ← p_ptr + 1;
                        endif
12.                 else /* flag = FALSE indicating the first match */
                        θdiff ← 𝒫[j][p_ptr].θ - Q1[q_ptr].θ;
                        q_ptr ← q_ptr + 1; p_ptr ← p_ptr + 1; flag ← TRUE;
                        p_temp ← p_ptr;
                        /* next match starts ahead of the failure position here */
                    endif
13.             else /* no match, advance p_ptr by one keeping q_ptr fixed */
                    p_ptr ← p_ptr + 1;
                endif
            endwhile
        endwhile
    endfor
```

6.2.3 Space and Time Complexity

Step 1 uses the arrangement of the dual lines of all the points for sorting. Further, the circularly sorted list around all the points in P is stored in $\mathcal{P}$ thus requiring an $O(n^2)$ space. □

Step 1 can be done in $O(n^2)$ [85]. Circular sorting of Q around q_1 can be done in time $O(k \log k)$. For Steps 3 to 13, the anchoring procedures are done to check for a match. The time complexity is bounded by the product of the number of anchors and the corresponding number of times the pointer p_ptr moves, which is $O(k(n-1))$. In the worst case, this process might be repeated for all the rows of $\mathcal{P}$. So, the total time taken by Steps 3 to 13 is $O(nk(n-1))$. Therefore, the total time complexity of the partial point set pattern matching algorithm proposed by Rezende and Lee [120] in 2-D is $O(n^2 + k \log k + nk(n-1)) = O(kn^2)$. □

6.3 Proposed Algorithm

In the algorithm proposed by Rezende and Lee [120], the anchoring of a query point has to be done with all other points of the sample set in the worst case; this process dominates the overall time complexity. An improvement of the above algorithm can be achieved if the number of anchors of a query point with the points in the sample set is reduced. Many applications in pattern recognition and computer vision (e.g., fingerprint minutiae matching for access control mechanisms) demand fast real time solutions. A reasonable preprocessing on the data set can be performed so that the actual matching can be expedited significantly. The above two observations lead to the following two facts that hold the key to the development of a faster algorithm.

Fact 1: For a sample set P of n points, the number of possible distances amongst the n points is $\binom{n}{2}$. Similarly, for a query set Q of k points, the number of possible distances amongst the k points is $\binom{k}{2}$. Note that, distances are preserved in translational or rotational transformations. If a k-subset of P matches with Q under translation and/or rotation, all the $\binom{k}{2}$ distances amongst the points of Q should occur among the $\binom{n}{2}$ distances of points in P. This condition is obviously a sufficient condition. Even if a single distance amongst the $\binom{k}{2}$ distances in Q does not have a similar distance in the $\binom{n}{2}$ distances of P, then there cannot be a k-subset of P matching Q. □

Fact 2: For a sample set P of n points, the number of possible distances amongst the n points is $\binom{n}{2}$. Of these $\binom{n}{2}$ distances, the number of equidistant pairs is $O(n^{4/3})$ [143] in the worst case. □

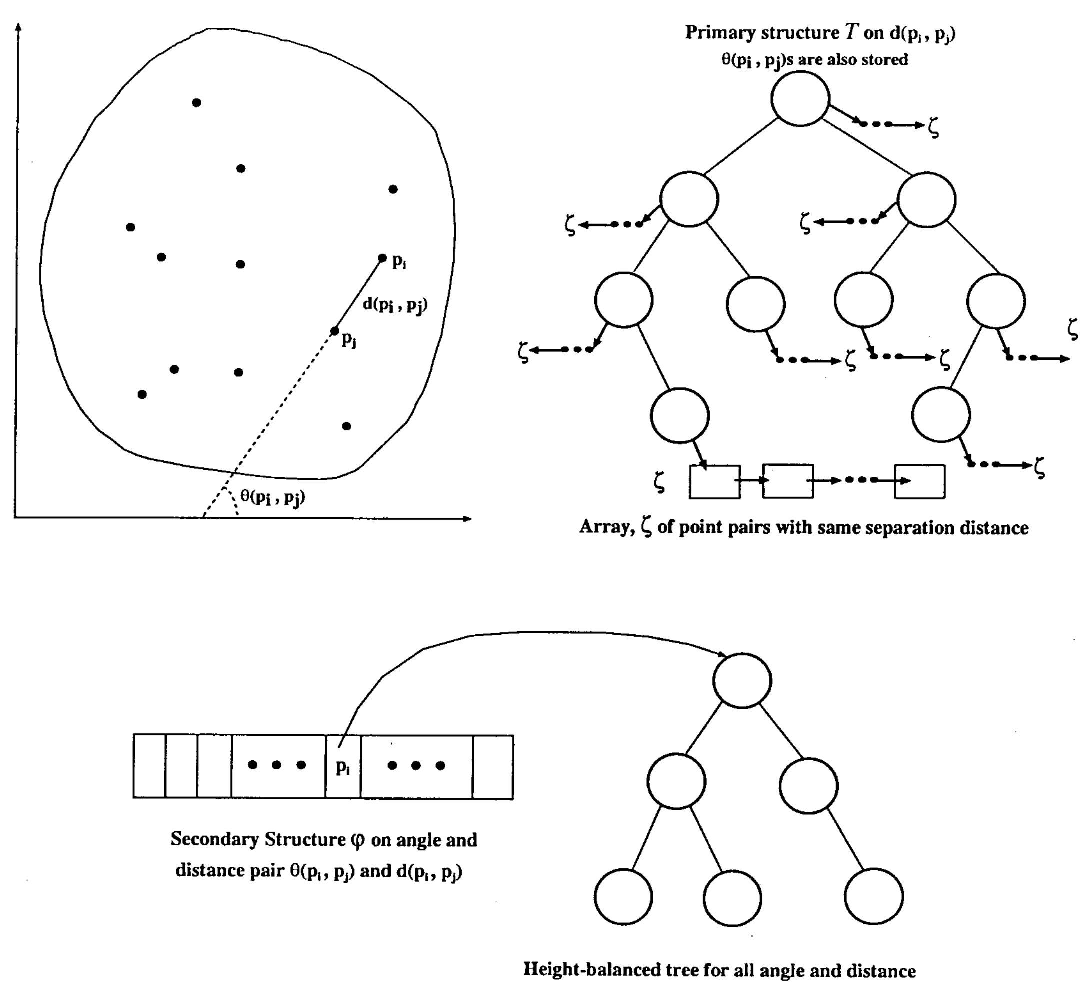


Figure 6.3: *Primary* and *secondary* data structures after preprocessing

6.3.1 Preprocessing

In actual applications related to fingerprint matching, we have a number of sample sets. We preprocess each sample set such that, given an arbitrary pattern set, we try to identify the sample sets such that a subset of points in each of these sample sets matches with the points in the given pattern set, if it exists. Let m sample sets be given, namely P_1, P_2, ..., P_m. In this step, we prepare data structures for each set separately that will be used for matching with a query pattern set. The pre-processing procedure for one set of points, say P, amongst the m sample sets, is given below.

Suppose that the set P has n points inside a rectangular box. Each point is arbitrarily labeled. The distances of all $\binom{n}{2}$ pairs of points are stored in a height-balanced binary tree, $\mathcal{T}$ [58]. This is the *primary* data structure for our algorithm (see Figure 6.3). The nodes of $\mathcal{T}$, correspond to the distinct distances among the members in the point set. Each node of $\mathcal{T}$ is attached with a pointer to an array ζ (see Figure 6.3), whose elements correspond to all the pairs of points separated by the same distance. The size of the array ζ, attached to a particular node of $\mathcal{T}$ storing a distance, say *dist*, is equal to the number of distances among the points in P having the same distance *dist*. Each element of this array contains:

(i) identification of the pair of points contributing this distance, and

(ii) angle of the line joining these points with the x-axis.

Apart from the *primary* data structure $\mathcal{T}$, we need another *secondary* data structure $\varphi[i]$, for each element p_i of P (see Figure 6.3). It is a height-balanced binary tree where each node corresponds to a distinct pair of angle and distance. The angle is taken as the angle made by the line segment between p_i and points in $P \setminus p_i$ with the axis, and the distance is the length of the line segment between p_i and points in $P \setminus p_i$. We assume that there can be no two instances of the same point, so the angle and distance pair corresponding to a point is unique. The size of the *primary* data structure $\mathcal{T}$ of all the points in P is $O(n^2)$. Each *secondary* structure $\varphi[i]$ takes space linear in the number of points and n such $\varphi[i]$'s are stored. Thus the *secondary* structure also requires $O(n^2)$ space.

Algorithm Preprocessing

Input: $P = \{p_1, p_2, \ldots, p_n\}$.

Output: A height-balanced tree, $\mathcal{T}$ corresponding to all distances;
n height-balanced trees, $\varphi[i]$ for each p_i, $i = 1, 2, \ldots, n$.

1. Label the points p_i arbitrarily, if they aren't labeled already;
2. **for**$(i \leftarrow 1;\ i \leq n;\ i \leftarrow i + 1)$ /* process each $p_i \in P$ */
3.   **for**$(j \leftarrow i + 1;\ j \leq n;\ j \leftarrow j + 1)$
4. $d_{ij} \leftarrow$ distance(p_i, p_j); $\theta_{ij} \leftarrow$ angle of line segment $\overline{p_i p_j}$ with x-axis;

```
5.        create a record β with a 4-tuple (d_ij, θ_ij, p_i, p_j);
6.        search in 𝒯 for d_ij;
7.        if(d_ij is already present in 𝒯) then
8.            attach β at the end of the array ζ rooted at the node corr. to d_ij;
9.        else
10.           create a new node in 𝒯 for β, the attached array ζ contains β
              as a single element; perform rebalancing operations on 𝒯, if necessary;
          endif
11.     insert β into φ[i] and φ[j]; perform rebalancing operations, if necessary;
        endfor
    endfor
```

Space and time complexity of preprocessing

Space required to store $\mathcal{T}$ is clearly $O(n^2)$. Each $\varphi[i]$ needs $O(n)$ space. There are n such $\varphi[i]$'s, so the total space complexity for storing the secondary structure is $O(n^2)$. Thus, the entire space complexity is $O(n^2)$. □

Steps 2 to 11 are concerned with the construction of height-balanced tree $\mathcal{T}$ with $\binom{n}{2}$($\approx O(n^2)$) distance values. So, these steps take time $O(n^2 \log n)$. Step 11 takes $O(n \log n)$ time for each $\varphi[i]$, and as there are n such lists, total time needed for creating the secondary structures is $O(n^2 \log n)$. Thus, the total time complexity is $O(n^2 \log n)$. □

6.3.2 Query

Given the set $Q=\{q_1, q_2, \ldots, q_k\}$ with k points, the query is to ascertain whether a k-subset of P matches with Q under translation and/or rotation. We select any two points, q_1 and q_2 from the set Q, and calculate the distance $d(q_1, q_2)$. Now, we search $d(q_1, q_2)$ in $\mathcal{T}$. If $d(q_1, q_2)$ is not found in $\mathcal{T}$, we can infer that Q cannot have a match in P (see Fact 1). If such an entry is found in $\mathcal{T}$, the following steps are performed:

Let the labels of the two end points of the matched distance in $\mathcal{T}$ be p_{α_1} and p_{α_2}. We anchor the line segment $\overline{q_1 q_2}$ of Q with the line segment $\overline{p_{\alpha_1}, p_{\alpha_2}}$ of P with q_1 anchored at p_{α_1}, (if it fails we would anchor q_2 with p_{α_1}) and execute the following step.

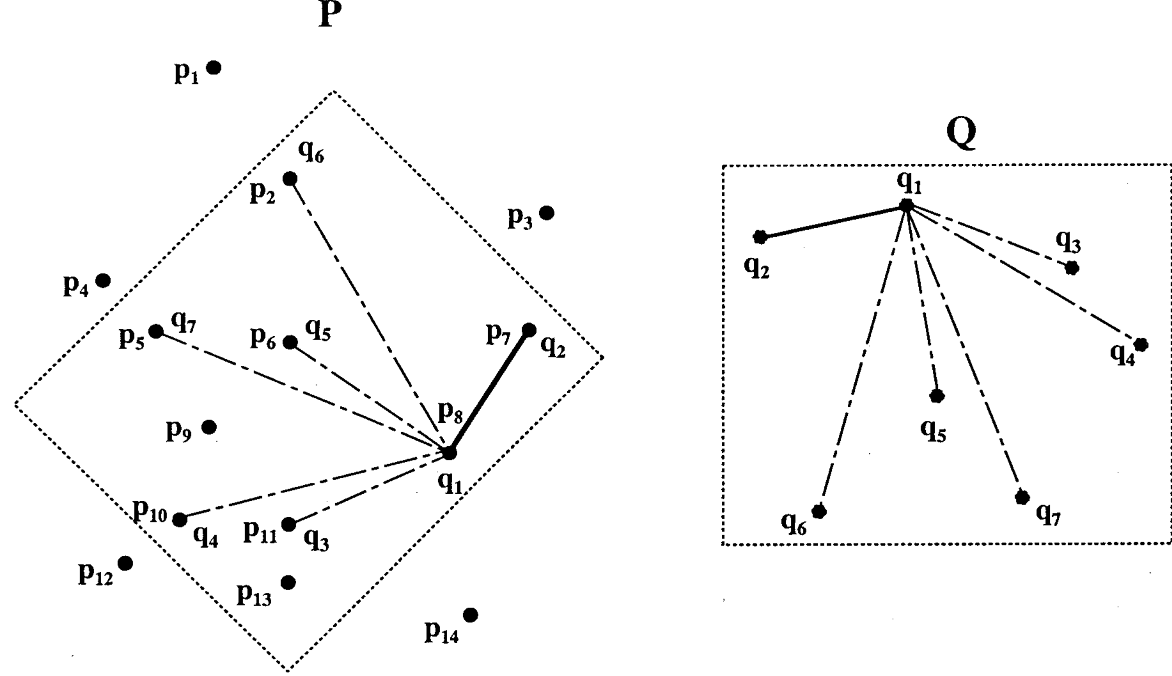


Figure 6.4: Illustration of the match with the bold lines showing anchors

We search the *secondary* structure attached with p_{α_1} (if it fails, then with p_{α_2}) to inspect whether there exists a match. For each point $q_i \in Q$, we compute the angle $\angle q_i q_1 q_2$ and the distance $d(q_1, q_i)$. The angle and distance pair $(\angle q_i q_1 q_2,\ d(q_1, q_i))$ is searched in the secondary structure $\varphi[p_{\alpha_1}]$ attached to p_{α_1}. If such a pair, $(\angle p_{\alpha_j}, p_{\alpha_1}, p_{\alpha_2},\ d(p_{\alpha_1}, p_{\alpha_j})$ is found, p_{α_j} matches with q_i. If all the points in Q are matched with k points of P, we say Q matches with P (see Figure 6.4 for an illustration). The angle of rotation is determined as the difference between line segments $\overline{q_1 q_2}$ and $\overline{p_{\alpha_1}, p_{\alpha_2}}$.

Algorithm Query

Input: $Q = \{q_1, q_2, \ldots, q_k\}$, $\mathcal{T}$, φ

Output: $M = \{p_{\alpha_1}, p_{\alpha_2}, \ldots, p_{\alpha_k}\}$ or $M = \phi$

1. Label the points q_i arbitrarily, if they aren't labeled already; $M \leftarrow \phi$;
2. Choose any two points $q_1, q_2 \in Q$; $\delta \leftarrow$ distance between q_1 and q_2;
3. Search in $\mathcal{T}$ for δ .
4. **if** (δ is not present in $\mathcal{T}$) **then** no match between P and Q;

5. **else** /* need to perform a detailed search in P */
6. **while** (the array ζ is not empty) **do**
8. p_{α_1} and p_{α_2} be the labels of the points whose distance is δ;
   insert p_{α_1} and p_{α_2} in M; anchor q_1 with p_{α_1};
9. **for** (all points q_i in $Q \setminus \{q_1, q_2\}$)
10. Search for the angle distance pair $(\angle q_i q_1 q_2,\ d(q_1, q_i))$ in $\varphi[p_{\alpha_1}]$;
11. **if** (such an angle distance pair is found) **then**
12. Insert p_{α_j} in M; /* p_{α_j} matches with q_i */
13. **endif**
14. **endif**
15. **endfor**
16. **if** $(| M | = k)$ **then**
17. Match exists; output M;
18. else
19. Repeat Steps 9-15 by anchoring q_2 with p_{α_1} ;
20. **if** $(| M | = k)$ **then**
21. Match exists; output M;
22. else
23. No match exists; $M \leftarrow \phi$; **exit**;
24. **endif**
25. **endif**
26. **endwhile**
27. **endif**

Time complexity of query

Step 3 for searching a distance in $\mathcal{T}$ requires $O(\log n)$ time. Steps 9-15 deal with the search in the *secondary* structure φ for the angle and distance pair $(\angle q_i q_1 q_2,\ d(q_1, q_i))$, for all $i \neq \{1, 2\}$ (see Step 11). Step 10 searches for a particular node having the angle distance pair $(\angle q_i q_1 q_2,\ d(q_1, q_i))$ in $\varphi[p_{\alpha_1}]$. This, in the worst case, needs $O(\log n)$ time. So, $k - 2$ such angles are to be searched and then their distances are to be matched. Each search requires $O(\log n)$ time. Therefore, steps 10-15 needs $O(k \log n)$ time. If a

match with q_1 anchored at p_{α_1} is not found, then q_1 is anchored at p_{α_2}, and the same procedure is repeated. So, step 19 also takes $O(k \log n)$ time. In the worst case, the match has to be checked at all the pairs of points having equal distances. This is taken care of by the **while** loop of step 6, which in the worst case, will be executed in time equal to the number of equidistant pairs d_{eq} present in the point set P. So, the total time complexity of our algoritm is $O(\log n + kd_{eq} \log n) \approx O(kd_{eq} \log n)$. From Fact 2, we know that the maximum value of d_{eq} may be $O(n^{4/3})$ in the worst case. Thus, the worst case time complexity of the above algorithm is $O(kn^{4/3} \log n)$. □

6.4 Experimental Results and Applications

6.4.1 Experiments on Randomly Generated Point Sets

We have implemented the above algorithm as well as the best known algorithm for the same problem by Lee and Rezende [120] on a Sun Ultra-5_10, Sparc, 233 MHz; the OS is SunOS Release 5.7 Generic. Table 6.1 and Table 6.2 illustrate the theoretical and experimental performance of the two algorithms. Table 6.3 shows the number of equal distances for randomly generated point sets having n points. We generated different number of sample sets (K), and in each sample set P_i ($i = 1, 2, \ldots, K$), we randomly generated 50 to 80 points (that is in the range of the number of minutiae present in a typical fingerprint). Thereafter, we selected a particular point set P_i, at random; took a sub-set from that point set and arbitrarily translated and rotated it to form the query set Q. Then a subset matching was performed, where each of the K sample sets was checked for a possible match. As the query set Q is a subset of a particular sample set from one of the K sample sets, a match is always ensured. That is also an experimental test of the correctness of our algorithm. The time reported is averaged over a number of experiments for different values of K. It may also be observed that under rotation the coordinates of the points may be real numbers instead of integers. Hence, the matching is performed allowing some small tolerance on the distances and angles. Such an approximate matching is very helpful for minutiae matching. The improved performance of the proposed algorithm is due to the fact that the maximum

Table 6.1: Worst-case complexity analysis

	Preprocessing time	Query time	Space
Lee and Rezende [120]	——	$O(kn^2)$	$O(n^2)$
Proposed algorithm	$O(n^2)$	$O(kn^{4/3}logn)$	$O(n^2)$

number of equidistant pair of points is very small in practical examples. Thus, the number of anchors to be considered is reduced to a great extent.

Table 6.2: Comparative results with respect to CPU Time

Number of sample sets, K	CPU Time in μ sec.		% saving
	Rezende and Lee's method [120]	Proposed method	
10	39,696,066	1,280,111	96.70
20	85,016,140	3,457,594	95.90
50	228,242,586	11,784,255	94.83
100	548,033,187	19,754,415	96.39
200	1075,977,761	39,225,959	96.35

Table 6.3: Experimental distance values

n	$n^{4/3}$	Experimental maximum, d_{eq}
100	464	4
200	1169	6
500	3887	20
1000	10000	53

6.4.2 Experiment with Real Fingerprint minutiae

The proposed algorithm was also tested using real fingerprint minutiae, extracted from fingerprints in the NIST 14 sdb [21, 152]. The procedure adopted for minutiae extraction

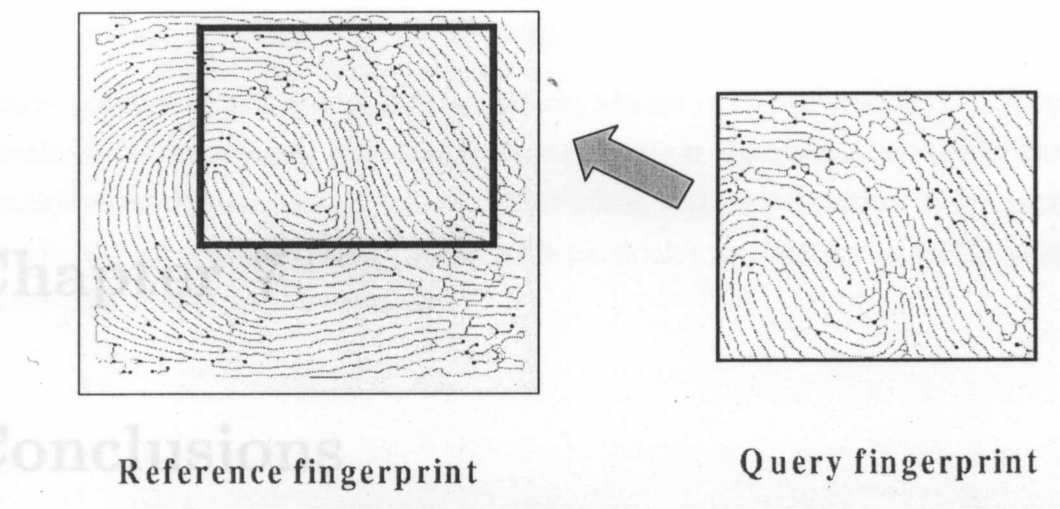


Figure 6.5: An example of a subset matching in a fingerprint

is described earlier as a flow chart in Figure 5.6. Ridges from the original gray-level fingerprints were extracted using the method proposed in the earlier chapter. The ridge lines were thinned to one-pixel thick lines. The resulting fingerprint image (as shown in Figure 5.21(c)) may have many spurious minutiae originating from spurs, bridges, short ridges, etc., [39] which can be eliminated by using the method of Farina et al. [39] (as shown in Figure 5.21(d)). We analyzed 100 images of the NIST14 sdb for extracting minutiae. A fingerprint image was then chosen at random; its minutiae are extracted and searched in the database using the proposed PSPM algorithm. A match was always found. Preprocessing stages for the 100-image database required on an average 2.3 seconds on a Sun Ultra-5_10, Sparc, 233 MHz machine; the OS being SunOS Release 5.7 Generic. The search for a particular minutiae point set in the database took on an average 0.7 seconds.

Chapter 7

Conclusions

In this thesis, we have developed new combinatorial techniques for image characterization, and reported related VLSI architectures to implement them. Most of the image processing operations have a parallelism inherent to it. The operations are locally repetitive and modular. Image processing applications demand high performance computing. On the other hand, VLSI microelectronic technology has made rapid strides to bring parallel computing structures in the realms of possibility. Interest in parallel processing and VLSI implementations of digital imaging applications has been an area of interest for quite some time. But now with the explosion of internet since early 1990s, images are being stored, processed and transmitted like never before. Performance of sequential computers to this end has been lacking as most of the applications require real time processing. Thus, the scope for design of new architectures for specific applications is on the rise. Coupled with the strides VLSI technology is making, application specific integrated circuits (ASIC) are in huge demand now.

Many applications in computer vision and pattern recognition problems require faster algorithms for processing on sequential computers. To this end, techniques from other areas of study like multidimensional index structure, computational geometry are adopted to provide fast real time solutions. Applications of these techniques to search and retrieval are studied in context to logo searching, multi-dimensional image query processing and fingerprint matching. Analytical and experimental results have been presented

to demonstrate the efficacy of our methods.

Images are now stored or transmitted almost always in compressed forms. Most of the operations on the images, including feature extraction algorithms, are performed in the uncompressed domain. To achieve real time goals, feature extraction in the compressed domain and its VLSI implementation with particular application to CBIR would be a challenging area of future research.

Bibliography

[1] Abe, K., Mizutani, F. and Wang, C., "Thinning of Gray-scale Images with Combined Sequential and Parallel Conditions for Pixel Removal", *IEEE Trans. SMC*, vol. 24, pp. 294-299, 1994.

[2] Acharya, T. and Mukherjee, A., "Parallel Butterfly Algorithm and VLSI Architecture for Image Decorrelation", *in Proc. SPIE, Digital Video Compression on Personal Computers: Algorithms and Technologies*, SPIE vol. 2187, pp. 322-332, May, 1994.

[3] Aho, A. V., Hopcroft, J. E. and Ullman, J. D., *The Design and Analysis of Computer Algorithms,* Addison–Wesley.

[4] Ahuja, N., "Dot Pattern Processing Using Voronoi Neighborhoods", *IEEE Trans. PAMI*, vol. 4, no. 3, pp. 336-343, May 1982.

[5] Akutsu, T., Tamaki, H. and Tokuyama, T., "Distribution of distances and triangles in a plane set and algorithms for computing the largest common point sets", *Discrete Computational Geometry,* vol. 20, pp. 307–331, 1998.

[6] Alt, H., Behrends, B. and Blömer, J., "Approximate Matching of Polygonal Shapes", *in Proc. ACM Symposium on Computational Geometry*, pp. 186-193, 1991.

[7] Alt, H. and Guibas, L. J., "Discrete Geometric Shapes: Matching, Interpolation, and Approximation -A Survey", *Technical Report B 96-11*, Freie Universität Berlin, 1996.

[8] Alt, H., Mehlhorn, K., Wagener, H. and Welzl, E., "Congruence, Similarity and Symmetries of Geometric Objects" *Discrete Computational Geometry*, vol. 3, pp. 237-256, 1988.

[9] American National Standards Institute, *Fingerprint Identification- Data Format for Information Interchange*, New York, 1986.

[10] Anderson, T. W., *An Introduction to Multivariate Statistical Analysis*, Wiley Eastern Pvt. Ltd., 1972.

[11] Arcelli, C. and Ramella, G., " Finding Gray Skeletons by Iterated Pixel Removal", *Image Vision Computing*, vol. 13, pp. 159-167, 1995.

[12] Arun, K.S., Huang, T.S. and Bolstein, S.D., "Least Squares Fitting of Two 3-(D) Point Sets", *IEEE Trans. PAMI*, vol. 9, pp. 698-700, 1987.

[13] Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R. and Wu, A., "An Optimal Algorithm for Approximate Nearest Neighbor Searching" *in Proc. ACM-SIAM Symposium on Discrete Algorithms*, pp. 573-582, 1994.

[14] Atkinson, M. D., "An Optimal Algorithm for Geometrical Congruence", *J. Algorithms,* vol. 8, pp. 159−172, 1987.

[15] Azencott, R., Wang, J. and Younes, L., "Texture Classification Using Windowed Fourier Filters", *IEEE Trans. PAMI*, vol. 19, no. 2, pp. 148-153, 1997.

[16] Bach, J., Fuller, C., Gupta, A., Hampapur, A., Horowitz, B., Humphrey, R., Jain, R. and Shu, C., "Virage Image Search Engine: An Open Framework for Image Management", *in Proc. SPIE, Storage and Retrieval for Still Image and Video Databases IV*, SPIE vol. 2670, pp. 76-87, Feb., 1996.

[17] Beckmann, N., Kriegel, H. -P., Schneider, R. and Seeger, B., "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles", *in Proc. ACM SIGMOD Intl. Conf. on Management of Data*, Atlantic City, USA, pp.322-331, 1990.

[18] Bentley, J. L., "Multidimensional Binary Search Trees Used for Associative Searching" *Commun. ACM*, vol. 18, no. 9, pp. 509-517, 1975.

[19] Bentley, J. L. and Friedman, J. H., "Data Structures for Range Searching" *ACM Computing Surveys*, vol. 11, no. 4, pp. 397-409, 1979.

[20] Besl, P. J. and Jain, R. C., "Three-Dimensional Object Recognition", *ACM Computing Surveys*, vol. 17, no. 1, pp. 75-145, 1985.

[21] Candela, G. T., Grother, P. J., Watson, C. I., Wilkinson, R. A. and Wilson, C. L., "PCASYS - A Pattern-Level Classification Automation System for Fingerprints", NISTIR 5647, *National Institute of Standards and Technology*, August 1995.

[22] Chakrabarti, K., Mehrotra, S., "The Hybrid Tree: An Index Structure for Higher Dimensional Feature Space", *Proceedings of the* 15th *International Conference on Data Engineering*, 1999.

[23] Chang, E., Li, C., Wang, J. Z., Mork, P., and Wiederhold, G, "Searching Near-Replicas of Images via Clustering", *in Proc. of SPIE Symposium of Voice, Video, and Data Communications*, pp.281-292, Boston, September 1999.

[24] Chang, J. -H., Fan, K. -C., "Fingerprint Ridge Allocation in Direct Gray-Scale Domain", *Pattern Recognition*, vol. 34, no. 10, pp. 1907-1925, Oct., 2001.

[25] Chazelle, B. M., Guibas, L. J. and Lee, D. T., "The Power of Geometric Duality", *BIT*, vol. 25, pp. 76-90, 1985.

[26] Cheng, H. D., Wu, C. Y. and Hung, D. L., "VLSI for Moment Computation and its Applications to Breast Cancer Detection", *Pattern Recognition*, vol. 31, no. 9, pp. 1391-1406, Sep., 1998.

[27] Chin, R. T. and Dyer, C. R., "Model-based Recognition in Robot Vision", *ACM Computing Surveys*, vol. 18, no. 1, pp. 67-108, 1986.

[28] Comer, D., "The Ubiquitous B-tree", *ACM Computing Surveys*, vol. 11, no. 2, pp. 121-138, 1979.

[29] Cormen, T. H., Leiserson, C. E. and Rivest, R. L., *Introduction to Algorithm*, Prentice Hall of India, 1998.

[30] Cox, I. J., Miller, M. L., Minka, T. P. and Papathomas, T. V., "The Bayesian Image Retrieval System, PicHunter: Theory, Implementation, and Psychophysical Experiments", *IEEE Trans. Image Processing*, vol. 9, no. 1, pp. 20-37, 2000.

[31] Dey, S., Bhattacharya, B. B., Kundu, M. K., and Acharya, T., "A Fast Algorithm for Computing the Euler Number of an Image and its VLSI Implementation", in *Proc.* 13th *Intl. Conf. on VLSI Design*, 2000, pp. 330-335.

[32] Doreswamy, K. B. and Ranganathan, N.,"A Systolic Algorithm and Architecture for Image Thinning", in Proc. of Great Lakes Symp. on VLSI(GLSVLSI'95), Mar 16-18, 1995.

[33] Duda, R. O. and Hart, P. E., *Pattern Classification and Scene Analysis*, John Wiley and Sons, New York, 1973.

[34] Duff, M. J. B. ed., *Computing Structures for Image Processing*, Academic Press Inc., London, 1983.

[35] Dyer, C. R., "Computing the Euler Number of an Image from its Quadtree", *Computer Graphics Image Processing*, vol. 13, no. 3, pp. 270-276, July 1980.

[36] Dyer, C. R. and Rosenfeld, A., "Thinning Algorithms for Gray Pictures", *IEEE Trans. PAMI*, vol. 1, pp. 88-89, 1979.

[37] Eakins, J. P. and Graham, M. E., "Content Based Image Retrieval: A Report to the JISC Technology Application Programme", *Technical Report*, Institute for Image Data Research, University of Northumbria at Newcastle, UK, Jan., 1999.

[38] Edelsbrunner, H., Rourke, J., O', and Seidel, R., "Constructing Arrangement of Lines and Hyperplanes with Applications", *SIAM Journal of Comput.*, vol 15, pp. 341-363, 1986.

[39] Farina, A., Zs. M. Kovacs-Vajna, Zs., M. and Leone, A., "Fingerprint Minutiae Extraction from Skeletonized Binary Images", *Pattern Recognition*, vol. 32, pp. 877-889, 1999.

[40] Forsyth, D., Mundy, J.L., Zisserman, A., Coelho, C., Heller, A. and Rothwell, C., "Invariant Descriptors for 3-D Object Recognition and Pose", *IEEE Trans. PAMI*, vol. 13, no. 10, pp. 971-991, Oct. 1991.

[41] Gaede, V. and Günther, O. "Multidimensional Access Methods", *ACM Computing Surveys*, pp. 170-231, Vol. 30, No. 2, June 1998.

[42] Galton, F., "Fingerprints", *London: Macmillan*, 1892.

[43] Gavrilov, M., Indyk, P., Motwani, R. and Venkatasubramanian, S., "Geometric Pattern Matching: A Performance Study" *in Proc. ACM Symposium on Computational Geometry*, 1999.

[44] Gevers, T. and Smeulders, A. W. M., "PicToSeek: Combining Color and Shape Invariant Features for Image Retrieval" *IEEE Trans. Image Processing*, vol. 9, no. 1, pp.102-119, 2000.

[45] Gonzalez, R.C., and Woods, R.E., *Digital Image Processing*, Addison-Wesley, Reading, Massachusetts, 1993.

[46] Goodman, J. E. and Pollack, R., "Multidimensional Sorting", *SIAM Journal on Computing*, vol. 12, pp. 484-507, 1983.

[47] Goodrich, M. T., Mitchell, J. S. B. and Orletsky, M. W., "Approximate Geometric Pattern Matching Under Rigid Motions", *IEEE Trans. PAMI*, vol. 21, no. 4, pp. 371-379, April, 1999.

[48] Gray, S. B., "Local Properties of Binary Images in Two Dimensions", *IEEE Trans. Computers*, no. 5, pp. 551-561, May 1971.

[49] Gudivada, V. N. and Raghavan, V. V., "Content-Based Image Retrieval Systems", *IEEE Computer*, vol. 28, no. 9, pp. 18-22, 1995.

[50] Guttman, A., "R-tree: a Dynamic Index Structure for Spatial Searching", *in Proc. ACM SIGMOD Intl. Conf. on Management of Data*, Boston, USA, pp.47-54, 1984.

[51] Haralick R. M., "Statistical and Structural Approaches to Texture", *Proc. IEEE*, vol. 67, no. 5, pp. 786-804, 1979.

[52] Haralick, R., "Ridges and Valleys on Digital Images", *Computer Vision Graphics Image Processing*, vol. 22, pp. 28-38, 1983.

[53] Haralick, R.M., Lee, C.N., Zhuang, X., Vaidya, V.G. and Kim, M.B., "Pose Estimation from Corresponding Point Data", *IEEE Trans. SMC*, vol. 19, no. 6, pp. 1426-1446, Nov. 1989.

[54] Harary, F., *Graph Theory*, Addison-Wesley, Reading, Mass., 1972.

[55] Hilditch, C. J., "Linear Skeletons from Square Cupboards", in *Machine Intelligence*, vol. 4, pp. 403-420, Edinburgh Univ. Press, 1969.

[56] Hollingum, J., "Automated Fingerprint Analysis Offers Fast Verification", *Sensor Review*, vol. 12, no. 13, pp. 12-15, 1992.

[57] Horn, B. K. P., *Robot Vision*, The MIT Press, Mass., 1991.

[58] Horowitz, E. and Sahni, S., *Fundamentals of Data Structures*, Computer Science Press Inc., 1981.

[59] Hu, M.K., "Visual Pattern Recognition by Moment Invariants", *IRE Trans. Information Theory 8*, pp 179-187, Feb. 1962.

[60] Hung, D. L., Cheng, H. D. and Sengkhamyong, S., "Design of a Configurable Accelerator for Moment Computation", *IEEE Trans. on VLSI Systems*, vol. 8, no. 6, pp. 741-746, Dec., 2000.

[61] Hwang, K., and Briggs, F. A., *Computer Architecture and Parallel Processing*, McGraw-Hill International Edition, Singapore, 1985.

[62] Jain, A. K., *Fundamentals of Digital Image Processing*, Prentice Hall of India, 1989.

[63] Jain, A. K. and Karu, K., "Learning Texture Discrimination Masks", *IEEE Trans. PAMI*, vol. 18, no. 2, pp. 195-205, Feb., 1996.

[64] Jain, A. K., Hong, L., Pankanti, S. and Bolle, R., "An Identity-Authentication System Using Fingerprints" , *Proc. of IEEE*, vol. 85, no. 9, pp. 1365-1388, Sep., 1997.

[65] Jain, A. K., Murty, M. N. and Flynn, P. J., "Data Clustering: A Review", *ACM Computing Surveys*, vol. 31, no. 3, pp. 264-323, 1999.

[66] Jain, A. K. and Vailaya, A., "Shape-Based Retrieval: A Case Study with Trademark Databases", *Pattern Recognition*, vol. 31, no. 9, pp. 1369-1390, 1998.

[67] Jang, B. K. and Chin, R. T., "One-pass Parallel Thinning; Analysis, Properties and Quantitative Evaluation", *IEEE Trans. PAMI*, vol. 14, pp. 1129-1140, Nov., 1992.

[68] Kahl, D. J., Rosenfeld, A. and Danker, A., "Some Experiments in Point Pattern Matching", *IEEE Trans. SMC*, vol. 10, no. 2, pp. 105-116, Nov. 1980.

[69] Katayama, N. and Satoh, S., "The SR-tree: An Index structure for Higher-Dimensional Nearest Neighbor Queries", *in Proc. of the International Conference on Management of Data, ACM SIGMOD*, pp.13-15, May 1997.

[70] Khalil, M. I. and Bayoumi, M. M., "Affine Invariants for Object Recognition Using the Wavelet Transform", *Pattern Recognition Letters*, vol. 23, no. 1-3, pp. 57-72, 2002.

[71] Khotanzad, A. and Hong Y. H., "Rotation Invariant Image Recognition Using Features Selected by a Systematic Method", *Pattern Recognition*, vol. 23, no. 10, pp. 1089-1101, 1990.

[72] Kittler, J. and Duff, M. J. B. ed., *Image Processing System Architectures*, Research Studies Press Ltd., Hertfordshire, England, 1985.

[73] Klinberg, J. M., "Two Algorithms for Nearest Neighbour Search in Higher Dimensions", *in Proc.* 29th *ACM Symposium on Theory of Computing*, 1997.

[74] Knuth, D. E., *The Art of Computer Programming, Vol. 1: Fundamental Algorithms* Addison-Wesley Publishing Company, Inc., 1973.

[75] Knuth, D. E., *The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison-Wesley Publishing Company, Inc., 1973.

[76] Knuth, D. E., Morris Jr., J. H. and Pratt, V. R., "Fast Pattern Matching in Strings", *SIAM J. on Computing*, vol. 6, no. 2, pp. 240-267, 1977.

[77] Kohavi, Z., *Switching And Finite Automata Theory*, McGraw Hill, New York, 1977.

[78] Kovacs-Vajna, Z.M., "A Fingerprint Verification System Based on Triangular Matching and Dynamic Time Warping", *IEEE Trans. PAMI*, vol. 22, no. 11, pp. 1266-1276, 2000.

[79] Kung, S. Y., *VLSI Array Processors*, Englewood Cliffs, Prentice Hall, NJ, 1988.

[80] Kunt, M.,"Source Coding of X-Ray pictures", *IEEE Trans. Biomedical Engineering*, vol. BME-25, no. 2, March 1978.

[81] Kushner, T., Wu, A. Y., Rosenfeld, A., "Image Processing on ZMOB", *IEEE Trans. on Computers*, vol. 31, no. 10, pp. 943-951, 1982.

[82] Kwok, P. C. K., "A Thinning Algorithm for Contour Generation", *Communications of the ACM*, vol. 31, no. 11, pp. 1314-1324, 1988.

[83] Lam, L., Lee, S. -W. and Suen, C. Y., "Thinning Methodologies - A Comprehensive Survey", *IEEE Trans. PAMI*, vol. 14, no. 9, pp. 869-885, Sep., 1992.

[84] Lam, L. and Suen, C. Y., "An Evaluation of Parallel Thinning Algorithms for Character Recognition", *IEEE Trans. PAMI*, vol. 17, no. 9, pp 914-919, Sept., 1995.

[85] Lee, D. T. and Ching, Y. T., "The Power of Geometric Duality Revisited", *Inform. Process. Lett.*, vol 21, pp. 117–122, 1985.

[86] Lee., C. N., Poston, T., and Rosenfeld, A., "Winding and Euler Numbers for 2D and 3D Digital Images", *Computer Vision, Graphics and Image Processing*, vol. 53, pp. 522-537, 1991.

[87] Leighton, F. T., *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann Publishers, San Mateo, California, 1992.

[88] López, A. M., Lumbreras, F., Serrat, J. and Villanueva, J. J., "Evaluation of Methods for Ridge and Valley Detection", *IEEE Trans. PAMI*, vol. 21, no. 4, pp 327-335, April, 1999.

[89] Mahalanobis, P. C., "On Tests and Measures of Group Divergences", *J. and Proc. Asiat. Soc. Beng.*, pp. 541-548, 1930.

[90] Maintz, J. B. A., Elsen, van den P. A. and Viergever, M. A., "Evaluation of Ridge Seeking Operators for Multimodality Medical Image Matching", *IEEE Trans. PAMI*, vol. 18, no. 4, pp. 353-365, April, 1996.

[91] Maio, D. and Maltoni, D., "Direct Gray-Scale Minutiae Detection In Fingerprints", *IEEE Trans. PAMI*, vol. 19, no. 1, pp. 27-39, 1997.

[92] Mano, M. M., *Computer System Architectures*, Prentice Hall India Ltd., New Delhi, 1992.

[93] Manjunath, B. S. and Ma, W. Y., "Texture Features for Browsing and Retrieval of Image Data", *Technical Report*, CIPR-TR-95-06, July, 1995.
To be found at: http://www.dlib.org/dlib/august95/contents.html

[94] Mehtre, B. M., "Fingerprint Image Analysis for Automatic Identification", *Machine Vision and Applications*, vol. 6, no. 2, pp. 124-139, 1993.

[95] Mehtre, B. M. and Chatterjee, B., "Segmentation of Fingerprint Images - a composite method", *Pattern Recognition*, vol. 22, pp. 381-385, 1989.

[96] Nayar, S. K., and Bolle, R. M., "Reflectance-based Object Recognition", *International Journal of Computer Vision*, vol. 17, no. 3. pp. 219-240, 1996.

[97] Nene, S.A., Nayar, S.K. and Murase H. "Columbia Object Image Library: COIL-100", *Technical Report CUCS-006-96*, Department of Computer Science, Columbia University, February 1996.

[98] Niblack, W., Barber, R., Equitz, W., Flickner, M., Glasman, E., Petkovic, D., Yanker, P., Faloutsos, C. and Taubin, G., "The QBIC Project: Querying Images by Content Using Color, Texture and Shape", *Storage and Retrieval for Image and Video Databases (SPIE)*, vol. 1908, pp. 173-187, 1993.

[99] Offen, R. J. ed., *VLSI Image Processing*, Collins, London, 1985.

[100] O'Gorman, L., "$k \times k$ Thinning", *Computer Vision, Graphics and Image Processing*, vol. 51, pp. 195-215, 1990.

[101] O'Gorman, L. and Nickerson, J. V., "An Approach to Fingerprint Filter Design", *Pattern Recognition*, vol. 22, pp. 29-38, 1989.

[102] Overmars, M., Berg, M. D., Kreveld, M. V. and Schwarzkopf, O., *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin, 1997.

[103] Pass, G. and Zabith, R., "Comparing Images Using Joint Histograms", *Multimedia Systems*, vol. 7, pp. 234-240, 1999.

[104] Persoon, E. and Fu, K. S., "Shape Discrimination using Fourier Descriptors", *IEEE Trans. SMC*, vol. SMC-7, pp. 170-179, 1977.

[105] Pogue, B. W., Mycek, M. A., and Harper, D., "Image Analysis for Discrimination of Cervical Neoplasia", *Journal of Biomedical Optics*, vol. 05(01), pp. 72-82, January 2000.

[106] Poty, V. and Ubeda, S., "A Parallel Thinning Algorithm Using $k \times k$ Masks", *Intl. J. of Pattern Recognition and Artificial Intelligence* vol. 7, no. 5, pp. 1183-1202, 1993.

[107] Chung, Y. and Prasanna, V. K., "Parallelizing Image Feature Extraction on Coarse-Grain Machines", *IEEE Trans. PAMI*, vol. 20, no. 12, pp. 1389-1394, Dec. 1998.

[108] Pratt, W. K., *Digital Image Processing.* John Wiley & Sons., 1978.

[109] Preparata, F. P. and Shamos, M. I., *Computational Geometry,* Springer–Verlag, New York, 1985.

[110] Qian, K. and Bhattacharya, P., "A Template Polynomial Approach for Image Processing and Visual Recognition", *Pattern Recognition*, vol. 25, pp. 1505-1515, 1992.

[111] Qian, K., Cao, S. and Bhattacharya, P., "Gray Image Skeletonization with Hollow Preprocessing Using Distance Transformation", *Intl. J. of Pattern Recognition and Artificial Intelligence*, vol. 13, pp. 881-892, 1999.

[112] Rabbani, M. and Joshi, R.,"An Overview of the JPEG2000 Still Image Compression Standard", *Signal Processing: Image Communications Journal*, vol. 17, no. 1, Oct., 2001.

[113] Rabbani, M. and Cruz, D. S., "The JPEG2000 Still-Image Compression Standard", *Course given at Intl. Conf. on Image Processing (ICIP)*, Oct., 2001, Thessaloniki, Greece . Also available at *http://jj2000.epfl.ch/jj_publications/index.html.*

[114] Ranganathan, N. ed., *VLSI Algorithms and Architectures: Fundamentals*, IEEE Computer Society Press, 1993.

[115] Ranganathan, N. ed., *VLSI Algorithms and Architectures: Advanced Concepts*, IEEE Computer Society Press, Los Alamitos, California, 1993.

[116] Ranganathan, N. ed., *VLSI & Parallel Computing for Pattern Recognition & Artificial Intelligence*, Machine Perception & Artificial Intelligence Series, vol.18, World Scientific, Singapore, 1995.

[117] Ranganathan, N., Vijaykrishnan, N. and Bhavanishankar, N., "A Linear Array Processor with Dynamic Frequency Clocking for Image Processing Applications", *IEEE Trans. CSVT*, vol. 8, no. 4, pp. 435-445, Aug., 1998.

[118] Reiss, T.H.,"The Revised Fundamental Theorem of Moment Invariants", *IEEE Trans. PAMI*, vol. 13, no. 8, pp. 830-834, 1991.

[119] Reiss, T.H., "Recognizing Planar Objects Using Invariant Image Features", *Lecture Notes in Computer Science*, vol. 676, Springer Verlag, Berlin, 1993.

[120] Rezende, P. J. and Lee, D. T., "Point Set Pttern Matching in *d*-dimensions", *Algorithmica*, vol. 13, pp. 387−404, 1995.

[121] Roberts, F. S., *Applied Combinatorics*, Prentice Hall Inc., Englewood Cliffs, NJ, 1984.

[122] Robinson, J. T., "The K-D-B-tree: A Search Structure for Large Multidimensional Dynamic Indexes", *in Proc. ACM SIGMOD Intl. Conf. on Management of Data*, Ann Arbor, USA, pp.10-18, 1981.

[123] Ronse, C., "A Topological Characterization of Thinning", *Theoretical Computer Science* vol. 43, pp. 31-41, 1986.

[124] Rosenfeld, A., and Kak, A. C., *Digital Picture Processing*, Academic Press Inc., New York, 1982.

[125] Rosenfeld, A. and Pfaltz, J. L., "Distance Function on Digital Pictures", *Pattern Recognition*, vol. 1, pp. 33-61, 1968.

[126] Rosin, P. L., and Ellis, T., "Image Difference Threshold Strategies and Shadow Detection", *in Proc., British Machine Vision Conference*, pp. 347-356, 1995.

[127] Rushing, J. A., Ranganath, H. S., Hinke, T. H. and Graves, S. J., "Using Association Rules as Texture Features", *IEEE Trans. PAMI*, vol. 23, no. 8, pp. 845-858, Aug., 2001.

[128] Salari, E. and Siy, P., "The Ridge Seeking Method for Obtaining the Skeleton of Digital Images", *IEEE Trans. SMC*, vol. SMC-14, no. 3, pp. 524-528, 1984.

[129] Samet, H., and Tamminen, H., "Computing Geometric Properties of Images Represented by Linear Quadtrees", *IEEE Trans. PAMI*, vol. PAMI-7, no. 2, March 1985.

[130] Schmid, C. and Mohr, R., "Local Grayvalue Invariants for Image Retrieval", *IEEE Trans. PAMI*, vol. 19, no. 5, pp. 530-535, Feb., 1997.

[131] Sellis, T., Roussopoulos, N. and Faloutsos, C., "The R^+-tree: A Dynamic Index for Multidimensional Objects", *in Proc. Intl. Conf. on Very Large Databases*, pp. 507-518, 1987.

[132] Shi, H., Ritter, G. X. and Wilson, J. N., "A Fast General Algorithm for Extracting Image Features on SIMD Mesh-Connected Computers", *Pattern Recognition*, vol. 30, no. 7, pp. 1205-1211, 1997.

[133] Shih, F. Y., King, C.T. and Pu, C. C., "Pipeline Architectures for Recursive Morphological Operations", *IEEE Trans. Image Processing*, vol. 4, no. 1, pp. 11-18, Jan. 1995.

[134] Shih, F. Y. and Pu, C. C.,"A Skeletonization Algorithm by Maxima Tracking on Euclidean Distance Transform", *Pattern Recognition*, vol. 28, no. 3, pp. 331-341, March 1995.

[135] Shih, F. Y. and Wong, W., -T., "Fully Parallel Thinning with Tolerance to Boundary Noise", *Pattern Recognition*, vol.27, no. 12, pp. 1677-1695, Dec. 1994.

[136] Shih, F. Y. and Wong, W., "A New Safe-Point Thinning Algorithm Based on the Mid-Crack Code Tracing", *IEEE Trans. SMC.*, vol. 25, no. 2, pp. 370-378, Feb. 1995.

[137] Smeulders, A. W. M., Worring, M., Santini, S., Gupta, A. and Jain, R., "Image Retrieval at the End of Early Years", *IEEE Trans. PAMI*, vol. 22, no. 12, pp. 1349-1380, Dec., 2000.

[138] Sproull, R., "Refinements to Nearest Neighbor Searching in k-dimensional Trees", *Algorithmica*, vol. 6, no. 4, pp. 579-589, 1991.

[139] Srihari, S. N., "Document Image Understanding", in *Proc. ACM/IEEE Joint Fall Computer Conference*, 1986.

[140] Stockman, G., Kopstein, S. and Benett, S., "Matching Images to Models for Registration and Object Detection via Clustering", *IEEE Trans. Pattern Analysis and Mach. Intell.*, vol. 4, no. 3, pp. 229-241, May, 1982.

[141] Suen, C. Y. and Wang, P. S. P., eds., "Thinning Methodologies for Pattern Recognition", *Intl. J. of Pattern Recognition and Artificial Intelligence*, vol. 7, no. 5, Oct., 1993.

[142] Swain, M. J. and Ballard, B. H., "Color Indexing", *Int'l J. Computer Vision*, vol. 7, no. 1, pp. 11-32, 1991.

[143] Szekely, L., "Crossing numbers and Hard Erdös problems in Discrete Geometry", *Combinatorics, Probability and Computing,* vol. 6, pp. 353–358, 1997.

[144] Teh, C. H. and Chin, R. T., "On Image Analysis by the Method of Moments", *IEEE Trans. PAMI*, vol. 10, no. 4, pp. 496-513, 1988.

[145] Tou, T.J. and Gonzalez, C.R., *Pattern Recognition Principles.* Addition-Wesley Publishing company, 1974.

[146] Trier, O. D., Jain, A. K., and Taxt, T., "Feature Extraction Methods for Character Recognition - A Survey", Pattern Recognition, Vol. 29, pp. 641-662, 1996.

[147] Veltkamp, R. C., Burkhardt, H. and Kriegel, H.-P., *State-of-the-Art in Content-Based Image and Video Retrieval*, Kluwer Academic Publishers, Dordrecht, Oct. 2001.

[148] Veltkamp, R. C. and Tanase, M., "Content-based Image Retrieval Systems: A Survey", *Technical Report*, UU-CS-2000-34, Universiteit Utrecht. Also available at *http://ftp.cs.ruu.nl/research/techreps/aut/remcov.html.*

[149] Venkatarangan, A. B., *Geometric and Statistical Analysis of Porous Media*, Ph.D. Dissertation, Dept. of Applied Math. and Statistics, SUNY at Stony Brook, NY, USA, 2000.

[150] Verwer, B. J. H., Vliet, L. J. V. and Verbeek, P. W., "Binary and Grey-Value Skeletons: Metrics and Algorithms", *Intl. J. of Pattern Recognition and Artificial Intelligence*, vol. 7, no. 5, pp. 1287-1308, Oct., 1993.

[151] Wang, C., Sun, H., Yada, S. and Rosenfeld, A., "Some Experiments in Relaxation Image Matching Using Corner Features", *Pattern Recognition* vol. 16, no. 2, pp. 167-182, 1983.

[152] Watson, C. I., Mated Fingerprint Card Pairs 2, Technical Report Special Database 14, MFCP2, National Institute of Standards and Technology, September 1993.

[153] Watson, C. I., Wilson, C. L., Fingerprint Database, National Institute of Standards and Technology, Special Database 4, FPDB, April, 1992.

[154] Wegstein, J. H., "An Automated Fingerprint Identification System", *US Government Publication*, Washington, 1982.

[155] White, D. A. and Jain, R., "Similarity Indexing with the SS-tree", *Proc. of the* 12th *Int. Conf. on Data Engineering*, New Orleans, USA, pp.516-523, Feb. 1996.

[156] www.mathworks.com/access/helpdesk/help/toolbox/images/bweuler.shtml

[157] Yu, S. -S. and Tsai, W., -H., "A New Thinning Algorithm for Gray-Scale Images by the Relaxation Technique", *Pattern Recognition*, vol. 23, no. 10, pp. 1067-1076, 1990.

[158] Zahn, C. T. and Roskies, R. Z., "Fourier Descriptors for Plane Closed Curves", *IEEE Trans. Computers*, vol. C-21, pp. 269-281, 1972.

[159] Zenzo, S. D., Cinque, L., and Levialdi, S.,"Run-Based Algorithms for Binary Image Analysis and Processing", *IEEE Trans. PAMI*, vol. PAMI-18, no. 1, January 1996.

List of publications related to the thesis

1 Bishnu, A., Bhattacharya, B. B., Kundu, M. K., Murthy, C. A., and Acharya, T., "On-Chip Computation of Euler Number of a Binary Image for Efficient Database Search", *in Proc. Intl. Conf. on Image Processing(ICIP), IEEE CS Press*, vol. III, pp. 310-313, Greece, 2001.

2 Bishnu, A., Bhattacharya, B. B., Kundu, M. K., Murthy, C. A., and Acharya, T., "On-Chip Computation of Euler Number of a Binary Image", *Communicated as Transaction Paper to IEEE Trans. on Circuits and Systems for Video Technology.*

3 Bishnu, A., Bhattacharya, B. B., Kundu, M. K., Murthy, C. A., and Acharya, T., "Euler Vector: A Combinatorial Signature for Gray-Tone Images", in *in Proc.* 3rd *Intl. Conf. on Information Technology: Coding and Computing (ITCC), IEEE CS Press*, pp. 121-126, Las Vegas, April 2002.

4 Bishnu, A., Bhunre, P. K., Bhattacharya, B. B., Kundu, M. K., Murthy, C. A., and Acharya, T., "Content-Based Image Retrieval: Related Issues Using Euler Vector", *Accepted for publication in Intl. Conf. on Image Processing(ICIP), IEEE CS Press*, 2002.

List of patents related to the thesis

1 Acharya, T., Bhattacharya, B. B., Bishnu, A., Kundu, M. K. and Murthy, C. A., "Computing the Euler Number of a Binary Image", US Patent pending, 2000.

2 Acharya, T., Bhattacharya, B. B., Bishnu, A., Kundu, M. K. and Murthy, C. A., "Developing an Euler Vector for Images", US Patent pending, 2000.

3 Acharya, T., Bhattacharya, B. B., Bishnu, A., Kundu, M. K. and Murthy, C. A., "Image Retrieval Using Distance Measure", US Patent pending, 2001.

4 Acharya, T., Bhattacharya, B. B., Bhowmick, P., Bishnu, A., Dey, J., Kundu, M. K. and Murthy, C. A., "Method and Apparatus for Providing a Binary Fingerprint Image", US Patent pending, 2001.

5 Acharya, T., Bhattacharya, B. B., Bhowmick, P., Bishnu, A., Dey, J., Kundu, M. K. and Murthy, C. A., "Architecture for Processing Fingerprint Images", US Patent pending, 2001.