Euclidean Distance Transform and Its Applications: Algorithms and Cellular Architectures

Thesis
submitted by

N. Sudha

for the award of the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science & Engineering
Indian Institute of Technology Guwahati
July 2000

TH-2715_964102

Certificate

This is to certify that the thesis entitled "**Euclidean Distance Transform and Its Applications: Algorithms and Cellular Architectures**", submitted by N. Sudha, a research student in the *Department of Computer Science and Engineering, Indian Institute of Technology, Guwahati*, for the award of the degree of **Doctor of Philosophy**, is a record of an original research work carried out by her under my supervision and guidance. The thesis has fulfilled all requirements as per the regulations of the Institute and in my opinion has reached the standard needed for submission. The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

Dated: 10th Nov'99
Guwahati - 781001

Sukumar Nandi
Associate Professor
Dept. of Computer Science and Engg.
Indian Institute of Technology
Guwahati - 781001
India

TH-2715_964102

Acknowledgement

At the outset, I express my gratitude to my supervisor Dr. Sukumar Nandi for his suggestions, constant encouragement and support. I am also grateful to my other doctoral committee members, Prof. Gautam Barua, Dr. P.K. Bora and Dr. P.K. Das for their valuable comments on my work. I thank Dr. Indrajit Chakraborthy and Dr. P.K. Bora for their comments on Chapters 2 and 3 of my thesis. I sincerely thank Prof. Gautam Barua, HOD of Computer Science and Engineering, for providing computing facilities to carry out this work.

I thank the Director, IIT-Guwahati for sanctioning partial financial support for my travel to Detroit, USA to attend ICRA'99. I also thank CSIR and IEEE for supporting my visit to the conference. I sincerely thank Dr. Shamala Chickamenahalli of Wayne State University (WSU), Detroit for arranging my stay at Detroit. I express my thanks to the students of WSU, Aruna, Hima and Gowhitha, for accommodating me and for their hospitality during my stay.

I joined IIT-Guwahati when the PhD program was started here. I thank people at library, computer center and administration who are responsible for setting up facilities whenever needed to carry out my research. I thank also the staff at library of IIT-Madras for allowing me to access the periodicals there. I am thankful to the computer engineers at the computer center of IIT-Guwahati for solving system related problems and installing different software packages in the system.

I thank the faculty members of IIT-Guwahati who helped me in different ways. I thank all my fellow research students for their cooperation. I especially thank Rambabu for his help in installing VLSI packages. I am thankful to the families of faculty members for their friendship.

Finally, I express my gratitude to my husband Sridharan for his constant encouragement and support throughout my research work and my parents, sister Geetha, mother-in-law, father-in-law, brothers-in-law, Ganesh and Sivaraman, and co-sister Rama for their love and affection.

N. Sudha
10/11/99

TH-2715_964102

Abstract

The present thesis embodies an in-depth study of Euclidean Distance Transform (EDT) of a binary image and its applications in the areas of Image Processing and Computer Vision. Main emphasis has been given for designing fast and parallel algorithm, which ideally suits for VLSI implementation specifically in Cellular Architectures. Earlier work in the field of distance transform and its applications has been reviewed along with their proposed VLSI architectures. Subsequently, an $O(n)$ time parallel algorithm is developed for EDT of an $n \times n$ binary image and its implementation in a cellular architecture is presented. Based on sound EDT computation, a linear time complex parallel algorithm has been developed for computation of the skeleton of a binary image with VLSI implementation. A novel technique for computation of discrete Voronoi diagram for binary image using EDT technique is also presented with cellular architecture. One of the most recent and popular area of research is the computation of the Hausdorff distance between images. We have exploited the EDT computation technique to compute Hausdorff distance in parallel. It is also shown that the same is realizable in cellular architectures. Further, applications of Skeleton, Voronoi diagram and Hausdorff distance are also explored successfully in this thesis.

Keywords : Euclidean Distance Transform, Skeleton, Voronoi Diagram, Hausdorff Distance, VLSI, Cellular Architecture, FPGA, Image Processing, Computer Vision.

TH-2715_964102

Contents

1 Introduction **1**

1.1 Scope of the Thesis 2

1.2 Organization of the Thesis 4

2 Terminology **7**

2.1 Distance Transforms 8

2.1.1 City-block 8

2.1.2 Chessboard 9

2.1.3 Euclidean 9

2.1.4 Other Distance Transforms 10

2.2 Applications of Distance Transforms 13

2.2.1 Skeleton 13

2.2.2 Voronoi Diagram 14

2.2.3 Hausdorff Distance 16

2.3 VLSI Architectures for Image Processing 18

2.3.1 Systolic Architecture 19

TH-2715_964102

2.3.2 Wavefront Architecture 19
2.3.3 Cellular Architecture 20
2.3.4 Comparison of Different Architectures 22

3 Literature Review 23
3.1 Computation of Distance Transforms 23
3.2 Construction of the Skeleton 27
3.3 Construction of the Voronoi Diagram 28
3.4 Computation of the Hausdorff Distance 29
3.5 Conclusions 30

4 Cellular Architecture for EDT 31
4.1 Introduction 31
4.2 Mathematical Foundations for the Algorithm 32
4.3 Algorithm for Computing EDT and NNT 37
4.3.1 Working of Algorithm: An Illustration 39
4.3.2 Complexity Analysis 39
4.4 Simulation results of Algorithm 41
4.5 VLSI Architecture 43
4.5.1 Design of a Cell 45
4.6 Applications of EDT 56
4.7 Conclusions 57

5 Computation of the Skeleton 59
5.1 Introduction 59
5.2 Background 60
5.3 Proposed Method 61
5.3.1 Analysis of Neighborhood Window 64
5.3.2 Effect of Discretization 67
5.4 Algorithm 68
5.4.1 Complexity analysis 72
5.5 Simulation results and comparisons 73
5.6 VLSI Architecture 77
5.6.1 Design of a Cell 81
5.7 Conclusions 86

6 Construction of the Voronoi Diagram 87
6.1 Introduction 87
6.2 Method of Construction 88
6.3 Algorithm 93
6.3.1 Complexity Analysis 97
6.4 VLSI Architecture 98
6.4.1 Design of a cell 100
6.5 Simulation Results and Discussions 102
6.6 Comparison studies 105
6.7 Conclusions 109

TH-2715_964102

7 Computation of the Hausdorff Distance **111**

7.1 Introduction 111

7.2 Method of Computation 112

7.2.1 Illustration 113

7.3 Algorithm 115

7.3.1 Complexity Analysis 116

7.4 Image Matching 117

7.4.1 Partial Image Matching 117

7.5 Simulation Results 119

7.6 VLSI Architecture 122

7.6.1 Design of a Cell 122

7.7 Conclusions 126

8 Applications **127**

8.1 Introduction 127

8.2 Applications of the Skeleton 129

8.2.1 Shape Analysis 129

8.2.2 Image Compression 129

8.2.3 Skeleton based Processing 132

8.3 Applications of the Voronoi Diagram 134

8.3.1 Robot Path Planning 134

8.3.2 Nearest Neighbor Pattern Classification 135

8.3.3 Image Compression and Secure Transmission 138

TH-2715_964102

8.4 Applications of the Hausdorff Distance 142

8.4.1 Character Matching 143

8.4.2 Character Recognition 144

8.4.3 Partial Character Recognition 145

8.4.4 Character Recognition in the Presence of Noise 148

8.5 Conclusions 150

9 Conclusions and Future Work **153**

9.1 Contributions of the Thesis 153

9.2 Future Work 156

A VHDL design of basic cell **157**

B Functional Simulation in ModelSim **165**

C Xilinx FPGA **171**

References **173**

List of Figures

2.1 City-block and chessboard distance transforms of a 7×7 bitmap. 10

2.2 Euclidean distance transform of a 7×7 bitmap. 10

2.3 Chamfer distance transforms. 13

2.4 Skeletons of some objects. 14

2.5 Voronoi diagram of points 15

2.6 Voronoi diagram of objects 16

2.7 Illustration of directed Hausdorff distances. 17

2.8 Basic configuration of systolic arrays. 20

2.9 A 2-D cellular architecture. 21

3.1 Computation of EDT in [KK92]. In (c), the nearest neighbor $N_{ij} = (x, y)$ is printed as xy. 25

4.1 Nearest integer Euclidean distance and displacement vector of pixels . . . 33

4.2 Pixels in the 3 x 3 neighborhood of p_0. 34

4.3 Illustration of algorithm EDT_NNT. 40

4.4 Simulation results of algorithm EDT_NNT. 42

4.5 Neighborhood connectivity of a cell. 44

TH-2715_964102

4.6 Block diagram of a cell. 44

4.7 Adder-subtracter logic 47

4.8 MAX module. 48

4.9 CMP-MUX module. 49

4.10 Different modules of a cell. 51

4.11 Layout of chip for basic cell 55

5.1 (a) Skeleton computed using a 3×3 neighborhood. (b) Actual skeleton. . 60

5.2 Neighborhood (N_S) of a pixel for skeleton computation 61

5.3 Neighboring pixels in an $m \times m$ window. 65

5.4 Angle between successive directions in an $m \times m$ window. 66

5.5 Possibility of branch missing at the corner pixel p_0. 67

5.6 Effect of discretization 69

5.7 Skeletons of polygonal objects computed by Method1 and the reconstructed objects. 74

5.8 Skeletons of polygonal objects computed by Method2 and the reconstructed objects. 75

5.9 Skeletons computed by Method 1 and Method 2 of non-polygonal objects and the reconstructed objects. 76

5.10 Skeletons and reconstructed images. 78

5.11 Connectivity of a cell 79

5.12 Block diagram of a cell. 80

5.13 AND-OR logic for computing the input to s by Method 1. 82

5.14 Logic for computing the input to s by Method 2. 83

TH-2715_964102

5.15 Layout of chip for the design of cell for computing skeleton. 85

6.1 Discrete Voronoi diagram of three pixels 88

6.2 Illustration of our method for constructing Voronoi diagram. 90

6.3 Neighborhood $N_{vd}(p_C)$ of pixel p_C. 91

6.4 Conditions for setting connectivity flags of a pixel p_C. 94

6.5 Neighborhood connectivity of a cell. 98

6.6 Block diagram of a cell. 99

6.7 Logic circuit for generating the input to the flip-flop W. 101

6.8 Layout of the chip for the design of cell for computing Voronoi diagram. . 103

6.9 Voronoi diagrams of images. 104

6.10 Voronoi diagram of objects. 105

6.11 Voronoi diagram of objects in a real image 106

6.12 Voronoi diagram constructed in [TTT97]. 108

6.13 Voronoi diagram of objects. Results of [AdB86, TTT97]. 108

7.1 Illustration of directed Hausdorff distances between images. 114

7.2 Character images taken for simulation study. 120

7.3 Computation of $h_i(\mathbf{B},\mathbf{P})$. 121

7.4 Block diagram of a cell. 123

7.5 Layout of the chip for the design of cell for computing directed Hausdorff distance. 125

8.1 An example 5×5 skeleton image. 130

8.2 Images considered for skeleton based compression 131

8.3 A path for a robot from initial position to final position along the Voronoi diagram. 136

8.4 Voronoi regions of three classes of random points. 137

8.5 Test images for image compression using Voronoi tessellation. 141

8.6 Different forms of test characters **A** and **B** for character matching. 144

8.7 Character images considered for character recognition. 146

8.8 Skeleton of character images in Figure 8.7. 147

8.9 Character images considered for partial character recognition. 148

8.10 Images considered for noisy character recognition. 151

B.1 Results of functional testing of VHDL design of basic cell corresponding to configuration 1 in ModelSim. 166

B.2 Results of functional testing of VHDL design of basic cell corresponding to configuration 2 in ModelSim. 167

B.3 Results of functional testing of VHDL design of basic cell corresponding to configuration 3 in ModelSim. 168

B.4 Results of functional testing of VHDL design of basic cell corresponding to configuration 4 in ModelSim. 169

TH-2715_964102

List of Tables

4.1 Value of *sel* for different cases of inputs to CMP-MUX module. "-" indicates "don't care". 50

4.2 Number of different Xilinx FPGA logic components of chips obtained from the implementation of the design of basic cell for different sizes of images. 53

4.3 Sizes of registers for different sizes of images. 53

4.4 Results obtained after placement and routing of FPGA components listed in table 4.2 for different sizes of images. 54

5.1 Angle θ_b and thickness of skeleton arising from the corner of a polygonal object for different sizes of neighborhood window 68

5.2 Number of different Xilinx FPGA logic components of chips obtained from the implementation of the design for computation of skeleton by both methods. 84

5.3 Results obtained after placement and routing of FPGA components listed in Table 5.2 for different sizes of images. 84

6.1 Number of different Xilinx FPGA logic components of chips obtained from the implementation of the design of cell for computation of Voronoi diagram 102

6.2 Results of placement and routing of components in Table 6.1 102

7.1 The Hausdorff distances H_i and fractions f' computed between the partial character images and template character images in Figure 7.2. 120

7.2 Number of different Xilinx FPGA logic components of chips obtained from the implementation of the design of cell for computing $h(A, B)$. . . 124

7.3 Results of placement and routing of components in Table 7.3 124

8.1 Results of skeleton based compression 132

8.2 Results of compression. 142

8.3 Hausdorff distances between every pair of characters in Figure 8.6. 145

8.4 Hausdorff distances between the skeletons (Figure 8.8) of test characters and templates in Figure 8.7. 145

8.5 Hausdorff distances between templates. 149

8.6 Fractions f' computed between partial images in Figure 8.9 and templates in Figure 8.7. 149

8.7 Fractions f_{IT} computed between noisy character images in Figure 8.10 and templates in Figure 8.7. 150

TH-2715_964102

Nomenclature

EDT	Euclidean Distance Transform
NNT	Nearest Neighbor Transform
$d_e(p)$	Euclidean distance value of pixel p
$d(p)$	Nearest integer approximation to $d_e(p)$
$(\Delta x(p), \Delta y(p))$	Distance vector of pixel p
k	Iteration number
$d_f(p)$	$(k^2 + k) - d_e^2(p)$
F	Foreground
B	Background
$s(p)$	skeleton value of pixel p
$H(A, B)$	Hausdorff distance between images A and B
$h(A, B)$	directed Hausdorff distance from image A to image B
$H_i(A, B)$	Nearest integer approximation to $H(A, B)$
$h_i(A, B)$	Nearest integer approximation to $h(A, B)$
basic cell	logic cell designed for EDT

Chapter 1

Introduction

As we approach the new millennium, we can expect greater challenges in computing in terms of speed and cost-effectiveness for handling new applications. Image processing and computer vision would be no exception. Algorithms for various problems in these areas should be fast. The thrust in early years was on developing fast sequential algorithms (see for instance [KZ96]). Since sequential algorithms typically process an image by considering only one pixel at a time, their performance is inadequate when the image size is reasonably large. A 1024×1024 image frame with 256 gray levels requires 8M bits for storage and most applications require the storage and processing of multiple frames at very high speeds. For instance, multimedia and video applications require frame rates of 50 to 80 frames per second. Thus, the throughput requirements of these applications could run into giga operations per second.

The reduced cost of microprocessors arising from the advances in semiconductor technology has shifted the focus towards development of algorithms that use multiple processors simultaneously. Implementations of parallel algorithms have been mostly attempted in the past on general purpose computers. For a specific application, the use of a large number of general purpose machines would not be cost-effective. Fortunately, with the rapid advances in VLSI technology, we are now in a position to design high performance hardware while keeping the cost low.

VLSI technology has progressed rapidly in the past decade leading to high packing densities, decrease in gate delays, powerful CAD design automation tools, and reliable and fault-tolerant design strategies. Consequently, the VLSI circuit designers have the

TH-2715_964102

option of transferring complex functions from the software domain to hardware blocks on the silicon floor. But this has resulted in designing complex circuits, which is time consuming. The increasing demand to reduce the design turn around time has forced the designers to look for simple, regular, modular and cascadable building blocks. Also, with the semiconductor technology, circuit delay due to interconnections on a silicon floor has become a major concern. Further, in the next generation submicron technology, interconnections will behave more like a device on the silicon floor (rather than just a simple signal path between the functional modules), thereby contributing a lion's share to the circuit delay. This situation has invariably forced the designers to have local interconnections as far as possible, for reliable high-speed operations of the circuit. This has led to the design of massively parallel array processors like systolic arrays and cellular architectures [CCNC97].

1.1 Scope of the Thesis

In this thesis, we explore ways of obtaining parallel solutions to some image processing problems on cellular architectures. The cellular architecture is one of the recently developed architectures. A 2D cellular architecture is an appropriate one for image processing. Our efforts to obtain speed up while maintaining cost-effectiveness have culminated in algorithms that exploit local interconnection type of characteristics of cellular architectures.

Our work begins by considering a fundamental image processing problem: Computation of the Euclidean Distance Transform (EDT) of a binary image. The transform converts a binary image consisting of foreground and background pixels into a multi-valued image such that each pixel is assigned the Euclidean distance from the nearest background pixel.

In order for EDT to be truly useful for machine vision applications like object detection, shape identification, robot path planning etc, an efficient VLSI implementation is needed. There are however difficulties.

The nonlinear operations (squaring and square root) involved in the computation are expensive for VLSI implementation. Moreover, the straightforward computation of EDT is a global operation in which the entire image is scanned to compute the distance

TH-2715_964102

value of one pixel. Global operations are unsuitable for VLSI implementation in a locally connected array of processors.

It is known that morphological operations such as dilation and erosion in images are quite suitable for VLSI. We have taken advantage of this fact. We develop a fast parallel algorithm for the computation of Euclidean distance transform of a binary image through dilation of foreground pixels in the image. The algorithm performs only local neighborhood operations for each pixel of the image.

Based on the usefulness in many applications, we have then selected three problems for which we propose EDT based solutions. The problems are: (a) Construction of skeleton [AdB96], (b) Construction of Voronoi diagram [PS85] and (c) Computation of Hausdorff distance [HKR93] for binary images.

We now list a sample set of applications for the skeleton, Voronoi diagram and Hausdorff distance. The skeleton of objects in a binary image is used for analyzing the shapes of objects [Blu64]. The objects can be reconstructed from the skeleton and its distance values. An image can be compressed by storing only the skeleton and distance values.

The Voronoi diagram is a useful geometric structure in robot path planning [Lat91] and pattern classification [Moh92]. An image can be compressed by constructing the Voronoi regions of randomly selected pixels [RP88].

The Hausdorff distance between images is used for comparing images [HKR93]. It gives the degree of mismatch between images. A given model can be located in an image using Hausdorff distance. It can also be used for optical character recognition.

In brief, the following are addressed in this thesis.

1. Computation of Euclidean distance transform
2. Extraction of skeleton from a binary image.
3. Construction of Voronoi diagram of objects in a binary image.
4. Computation of Hausdorff distance between images.
5. Development of cellular architectures for the above four problems.

6. Discussion of the applications of skeleton, Voronoi diagram and Hausdorff distance. In particular, the usefulness of Hausdorff distance for character recognition is described.

1.2 Organization of the Thesis

In the next chapter, we review the definitions of different distance transforms and their applications. The distance transforms based on three distance measures, which are commonly adopted in image processing are discussed. These are city-block, chessboard and Euclidean. Other distance transforms, which are approximations to EDT, are also discussed. They are octagonal and chamfer distance transforms. We then define (i) the skeleton (ii) the Voronoi diagram and (iii) the Hausdorff distance. Some characteristics of different VLSI architectures for image processing are given.

In Chapter 3, we survey the literature on algorithms for the Euclidean distance transform, skeleton, Voronoi diagram and Hausdorff distance.

In Chapter 4, we propose a scheme for computing exact Euclidean distance transform of a binary image. An efficient parallel algorithm is then developed. We then present the VLSI implementation of the algorithm in a cellular architecture.

The next three chapters are devoted to the problems that can be solved using Euclidean distance transform. Chapter 5 describes the computation of skeleton of a binary image of objects. In Chapter 6, we propose a method for constructing Voronoi diagram of objects in a binary image. Chapter 7 presents a parallel method of computation of Hausdorff distance between two binary images. In each of these three chapters, a parallel algorithm for solving the problem is first developed. It is followed by VLSI implementation in a cellular architecture.

Finally, some applications of skeleton, Voronoi diagram and Hausdorff distance are outlined in Chapter 8. The applications include image compression, shape analysis, pattern classification, robot path planning and character recognition.

The summary of contributions and directions for future work are reported in Chapter 9.

TH-2715_964102

For ease of understanding, the VHDL code for the design of a cell used in the cellular architecture for computing Euclidean distance transform is provided in Appendix A. The functional testing of the design in ModelSim package is given in Appendix B. Appendix C gives information about Xilinx FPGA in which the design has been implemented.

Chapter 2

Terminology

This chapter defines various terms used in the thesis. We begin with the description of the various transforms and then proceed to define various structures that can be computed using the transforms. Finally, we briefly summarize the characteristics of some parallel architectures in vogue.

Distance transforms are taken only for binary images. A binary image can be obtained from a gray image, captured by a camera, using various binarization techniques. If G denotes a gray image in which the pixels have the values in the range [0,255] and B denotes the corresponding binary image whose pixels have the values either 0 or M, then binarization of G to obtain B can be done by simple thresholding as follows.

$$b(p) = \begin{cases} 0 & g(p) < T \\ M & \text{otherwise} \end{cases}$$

where p denotes a pixel at row x and column y, $b(p)$ denotes the gray value of p in B, $g(p)$ denotes the binary value of p in G and T is the preset threshold value between 0 and 255. More complex binarization methods are described in [KZ96, Sch89]. In the rest of this thesis, we refer to the pixels of the binary image with $b(p) = 0$ as 0-pixels and the ones with $b(p) = M$ as 1-pixels.

TH-2715_964102

2.1 Distance Transforms

Distance Transformation produces a mapping from a double-valued image, consisting of 0-pixels and 1-pixels, to a multi-valued image where every pixel has a value corresponding from the distance from the nearest 1-pixel. The *Nearest Neighbor Transformation* (NNT) assigns the identity of the nearest 1-pixel to each pixel of the image. NNT can be called as vector distance transformation if the identity is the distance vector pointing to the nearest 1-pixel. Let (x, y) refer to a pixel at row x and column y and its nearest 1-pixel be (x', y'). Then, the distance vector is given by $(\Delta x, \Delta y)$ where $\Delta x = x' - x$ and $\Delta y = y' - y$. Distance transforms are first introduced by Rosenfeld and Pfaltz [RP66, RP68] and have been the focus of extensive literature [Dan80, Bor84, Bor86]. Distance transforms and nearest neighbor transforms are widely used for machine vision applications.

Three types of distance metrics [GW92] are commonly used in image processing. They are city-block, chessboard and Euclidean. The distance transforms based on these metrics are discussed below.

2.1.1 City-block

The city-block distance between two pixels $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ is given by

$$d_4(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2|.$$

In this case, the pixels having a d_4 distance from (x, y) less than or equal to some value r form a diamond centered at (x, y). For example, the pixels with d_4 distance less than or equal to 2 from (x, y) (the center pixel) form the following contours of constant distance:

		2		
	2	1	2	
2	1	0	1	2
	2	1	2	
		2		

The pixels with $d_4 = 1$ are the 4-neighbors of (x, y). The city-block distance of a pixel from (x, y) can also be given by the minimum number of horizontal and vertical steps

TH-2715_964102

required to reach the pixel from (x, y).

2.1.2 Chessboard

The chessboard distance between two pixels $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ is given by

$$d_8(p_1, p_2) = \max(|x_1 - x_2|, |y_1 - y_2|).$$

In this case, the pixels with d_8 distance from (x, y) less than or equal to some value r form a square centered at (x, y). For example, the pixels with d_8 distance less than or equal to 2 from (x, y) (the center pixel) form the following contours of constant distance:

2	2	2	2	2
2	1	1	1	2
2	1	0	1	2
2	1	1	1	2
2	2	2	2	2

The pixels with $d_8 = 1$ are the 8-neighbors of (x, y). The chessboard distance of a pixel from (x, y) can also be given by the minimum number of horizontal, vertical and diagonal steps required to reach the pixel from (x, y). Since city-block and chessboard distances are integers, these two distance metrics are adopted for many image processing applications for convenience. The maximum error [Bor84] between these distances and the Euclidean distance are as follows. For city-block, it is 58.6% and for chessboard, it is 41.4%. The city-block and chessboard distance transforms of a 7×7 bitmap are shown in Figure 2.1.

2.1.3 Euclidean

The Euclidean distance between two pixels $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ is given by

$$d_e(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

For this distance measure, the pixels having a distance less than or equal to some value r from (x, y) are the points contained in a disk of radius r centered at (x, y). Unlike the

$$\begin{array}{ccccccc}1&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&1\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\1&0&0&0&0&0&0\end{array}$$

(a) Bitmap

$$\begin{array}{ccccccc}0&1&2&3&4&4&3\\1&2&3&4&4&3&2\\2&3&4&4&3&2&1\\3&4&4&3&2&1&0\\2&3&4&4&3&2&1\\1&2&3&4&4&3&2\\0&1&2&3&4&4&3\end{array}$$

(b) City-block

$$\begin{array}{ccccccc}0&1&2&3&3&3&3\\1&1&2&3&2&2&2\\2&2&2&3&2&1&1\\3&3&3&3&2&1&0\\2&2&2&3&2&1&1\\1&1&2&3&2&2&2\\0&1&2&3&3&3&3\end{array}$$

(c) Chessboard

Figure 2.1: City-block and chessboard distance transforms of a 7×7 bitmap.

$$\begin{array}{ccccccc}1&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&1\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\1&0&0&0&0&0&0\end{array}$$

(c) Bitmap

$$\begin{array}{ccccccc}0&1&2&3&\sqrt{13}&\sqrt{10}&3\\1&\sqrt{2}&\sqrt{5}&\sqrt{10}&\sqrt{8}&\sqrt{5}&2\\2&\sqrt{5}&\sqrt{8}&\sqrt{10}&\sqrt{5}&\sqrt{2}&2\\3&\sqrt{10}&\sqrt{13}&3&2&1&0\\2&\sqrt{5}&\sqrt{8}&\sqrt{10}&\sqrt{5}&\sqrt{2}&2\\1&\sqrt{2}&\sqrt{5}&\sqrt{10}&\sqrt{8}&\sqrt{5}&2\\0&1&2&3&\sqrt{13}&\sqrt{10}&3\end{array}$$

(d) Euclidean

Figure 2.2: Euclidean distance transform of a 7×7 bitmap.

previous distances, Euclidean distance is real valued. The Euclidean distance metric is attractive for the following reasons:

1. It is a natural measure of distance; the measure corresponds to the straight line distance between the pixels.
2. The Euclidean distance measure is rotation invariant. This property is useful in applications like shape analysis, object recognition etc.

The Euclidean Distance Transform (EDT) of a 7×7 bitmap (same as the one in Figure 2.1) is shown in Figure 2.2.

2.1.4 Other Distance Transforms

There are other distance transforms that are approximations to EDT. They are known as octagonal and chamfer distance transforms.

TH-2715_964102

Octagonal

It is the combination of city-block and chessboard distance. For example, the pixels with octagonal distance less than or equal to 3 from (x, y) (the center pixel) form the following contours of constant distance:

$$\begin{array}{ccccccc} & & 3&3&3& & \\ &3&2&2&2&3& \\3&2&2&1&2&2&3\\3&2&1&0&1&2&3\\3&2&2&1&2&2&3\\ &3&2&2&2&3& \\ & &3&3&3& & \end{array}$$

Here, the pixels with distance 1 are the 4-neighbors of center pixel. Those with distance 2 are the 8-neighbors of the pixels with distance 1 and the ones with distance 3 are again 4-neighbors of pixels with distance 2. In general, the pixels with distance n are either 4-neighbors or 8-neighbors of the pixels with distance $n-1$. If n is odd, then the pixels are 4-neighbors, otherwise they are 8-neighbors.

Chamfer

The term *chamfer* originally referred to a sequential two-pass distance transform developed by Rosenfeld and Pfaltz [RP66]. It was later improved and generalized by Borgefors [Bor86]. The chamfer distance transform approximates the global distance computation with repeated propagation of local distances within a small neighborhood mask. A 3×3 (chamfer 3-4) chamfer mask and a 5×5 (chamfer 5-7-11) chamfer mask are shown below.

4	3	4
3	0	3
4	3	4

Chamfer 3-4

14	11	10	11	14
11	7	5	7	11
10	5	0	5	10
11	7	5	7	11
14	11	10	11	14

Chamfer 5-7-11

One approach for computing the chamfer distance transform is via an iterative procedure with the following initialization.

$$d_c^0(p) = \begin{cases} 0 & p \text{ is 1-pixel} \\ \infty & p \text{ is 0-pixel} \end{cases}$$

The distance values of all 0-pixels are computed as follows.

$$d_c^k(p) = \begin{cases} d_c^0(p) & k = 0 \\ \min\limits_{p_n \in N_p} [d_c^{k-1}(p_n) + W_n] & k > 0 \end{cases}$$

where k stands for the iteration number, N_p is the neighborhood of p defined by the mask, p_n is a pixel in that neighborhood and W_n is the local distance given by the mask. The chamfer 3-4 and chamfer 5-7-11 distance transforms of a 4×4 bitmap are shown in Figures 2.3 (b) and (c). The distance values are integers and they are proportional to the Euclidean distance values but with some approximation error. In the case of the chamfer 3-4 mask, the proportionality constant is 3 and in the case of the chamfer 5-7-11, it is 5. The approximate Euclidean distance values computed from the chamfer distance transforms are given in Figures 2.3 (e) and (f). The approximation error depends upon the size of the neighborhood and the selection of the local distances. For chamfer 3-4 distance, the maximum error is 8.3% and for chamfer 5-7-11 distance, it is 8.1% [Bor84]. A geometric approach to find optimal local distances is given in [BM98].

The applications of distance transforms are discussed next.

TH-2715_964102

(a)

0	0	0	0
0	0	0	0
0	0	0	0
1	0	0	0

(b)

9	10	11	12
6	7	8	11
3	4	7	10
0	3	6	9

(c)

15	16	18	21
10	11	14	18
5	7	11	16
0	5	10	15

(d)

3	3.16	3.6	4.24
2	2.23	2.82	3.6
1	1.41	2.23	3.16
0	1	2	3

(e)

3	3.33	3.66	4
2	2.33	2.66	3.66
1	1.33	2.33	3.33
0	1	2	3

(f)

3	3.2	3.6	4.2
2	2.2	2.8	3.6
1	1.4	2.2	3.2
0	1	2	3

Figure 2.3: Chamfer distance transforms. (a) bitmap; (b) & (c) chamfer 3-4 & chamfer 5-7-11 distance transforms; (d) EDT; (e) distance values in (b) divided by 3; (f) distance values in (c) divided by 5.

2.2 Applications of Distance Transforms

Distance transforms are useful tools for image analysis [Rus95, KK87, Ver92], pattern recognition [LS90, Ahu80], robotics [Bou90, AdB86], machine vision [Pag92] and other related applications [Ye88, DC87]. Most of these applications involve computation of skeleton, Voronoi diagram or Hausdorff distance. These three terms are defined in this section.

2.2.1 Skeleton

The *skeleton* of a binary image is an important representation of the shape of objects in the image and is useful for many pattern recognition applications. The skeleton of a continuous binary image consisting of an object and a background, is defined as the set of points belonging to the object which are equidistant from more than one closest point on the boundary of the object. The object can be reconstructed from the skeleton points with their distance values. The reconstructed object consists of discs whose centers coincide with the skeleton points and radii are the distance values at these skeleton points. The skeletons of some objects are shown in Figure 2.4. The skeleton typically looks like a stem with branches arising from it. The stem of the skeleton is

medially located with respect to the boundary and each branch arises from a corner in the boundary of the object if any. Each corner corresponds to one and only one branch.

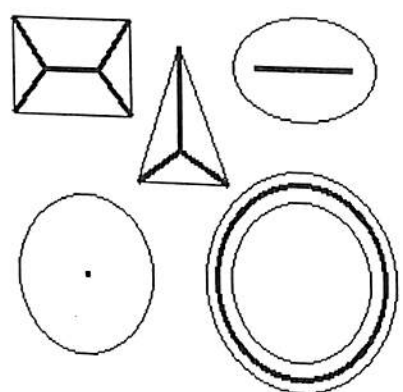


Figure 2.4: Skeletons of some objects. Object boundaries are shown by thin lines and the skeletons by thick lines.

In multimedia, world wide web and real-time applications, data is distributed over the network. Sometimes, binary images need to be transferred and it is required to analyze these images. The images are usually compressed and transferred. Compression of binary images can be achieved by storing the coordinates and distance values of skeleton points alone. Processing of these images can be done in the compressed form itself. This takes less memory and time.

Another application of EDT is discussed next.

2.2.2 Voronoi Diagram

The *Voronoi diagram* is an important geometric structure and it has applications in robot path planning [Lat91], pattern classification [Moh92] and image processing. In image processing, it is used for image compression [RP88, AAS85] and texture segmentation [TJ90, KSI98]. The Voronoi diagram of a finite set $P = \{p_1, p_2, ..., p_n\}$ of points in the Euclidean plane is a subdivision of the plane, associating to each p_i the region of the plane R_i consisting of points closer to p_i than to any other p_j, $j \neq i$ [PS85]. The region R_i is the *Voronoi region* of the point p_i. The Voronoi diagram consists of those points that do not have a unique nearest point in P. Figure 2.5 shows the Voronoi

TH-2715_964102

diagram of an example set of points. In the figure, the Voronoi regions are bounded or

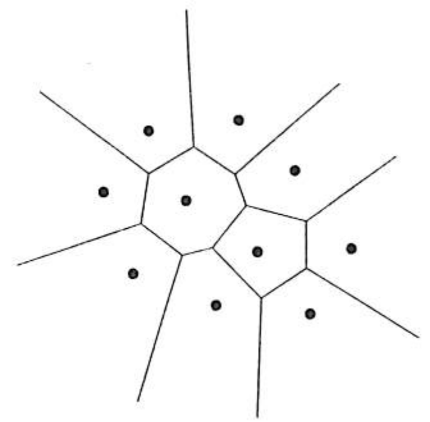


Figure 2.5: Voronoi diagram of points

unbounded convex [1] polygonal regions. The construction of Voronoi diagram of n points takes $O(n \log n)$ time in a sequential machine [PS85]. Some parallel implementations are available in [Roo94, DKF94, Guh94].

If each element of P is an object (a connected set of points) of arbitrary size and shape, the Voronoi diagram is still considered to be the subdivision of the plane obtained according to the proximity rule analogous to that of a point. In this case, the Voronoi regions are no longer convex and bounded by straight line segments. As an example, Figure 2.6 shows the Voronoi diagram of six objects namely A, B, C, D, E and F in a two dimensional space. The *Voronoi region* for object A in Figure 2.6 would correspond to the portion of the plane lying to the left of $a_1a_2a_3a_4$. If the objects are all convex, then the Voronoi diagram is simply the skeleton of the background. But if the objects are non-convex, the Voronoi diagram is different from the skeleton. However, before constructing the Voronoi diagram, we need to represent the objects. The geometric representation of any arbitrary object is not easy. One way of representation is modeling using parametric curves [FvDFH97].

This leads to the construction of a *discrete Voronoi diagram* [AdB86, SP97] in a binary image consisting of object and background pixels.

One more useful application of EDT is discussed next.

[1] A region is said to be convex if the line segment connecting any two points of the region lies within the region itself

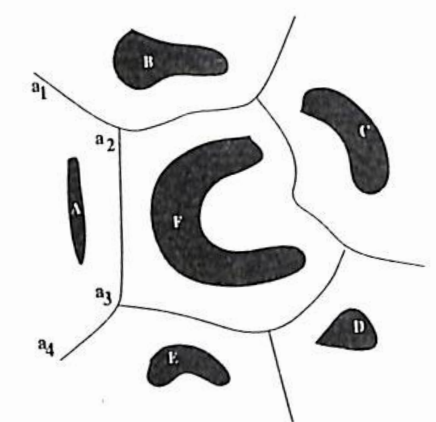


Figure 2.6: Voronoi diagram of objects

2.2.3 Hausdorff Distance

The Hausdorff distance between two images measures the degree of mismatch between the images. It can be used to determine the extent by which two shapes differ from one another. Hence it is applied to pattern recognition and computer vision [HKR93, Ruc96, YC95, Ruc95]. The *Hausdorff Distance* between two point sets $P = \{p_1, p_2, \ldots, p_m\}$ and $Q = \{q_1, q_2, \ldots, q_n\}$ is defined as $H(P,Q) = \max(h(P,Q), h(Q,P))$ where $h(P,Q) = \max_{p_i \in P} \min_{q_j \in Q} \rho(p_i, q_j)$ and $\rho(.,.)$ is some distance function for comparing two points p_i and q_j. The function $h(P,Q)$ is called the *directed Hausdorff distance* from P to Q. It identifies the point $p_i \in P$ that is farthest from the set Q and measures the distance from p_i to its nearest point in Q. The function $h(Q,P)$ is defined similarly. The directed Hausdorff distances are illustrated in Figure 2.7. The straightforward computation of $h(P,Q)$ takes $O(mn)$ time since for each point in P, the distances with all the points in Q are computed. Similarly, $h(Q,P)$ takes $O(mn)$ time and hence the time taken for computing $H(P,Q)$ is $O(mn)$. The Hausdorff distances have the following properties.

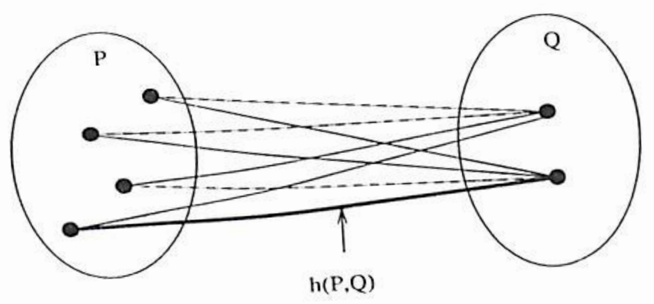
1. If $h(P,Q) = d$, then every point in P is at a distance less than or equal to d from some point in Q.

2. If $P \subset Q$ and $P \neq Q$, then $h(P,Q) = 0$ but $h(Q,P) \neq 0$.

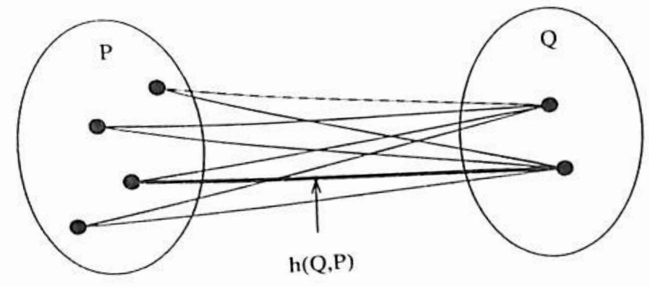
3. $H(P,Q) \geq 0$; $H(P,Q) = 0$ if $P = Q$.

4. $H(P,Q)$ and $H(Q,P)$ are equal but $h(P,Q)$ and $h(Q,P)$ need not be.

TH-2715_964102



For each point in P, the dashed line shows the distance to the closest point in Q, i.e., minimum distance to Q. Then, the bold line shows the longest of these minimum distances, i.e., maximum of the minimum distances.



For each point in Q, the dashed line shows the distance to the closest point in P, i.e., minimum distance to P. Then, the bold line shows the longest of these minimum distances, i.e., maximum of the minimum distances.

Figure 2.7: Illustration of directed Hausdorff distances.

The Hausdorff distance between images is defined between the set of foreground pixels. The Hausdorff distance between two binary images A and B (each with size $n \times n$) consisting of foregrounds F_A and F_B and backgrounds B_A and B_B is defined as $H(A,B) = \max(h(A,B), h(B,A))$ where $h(A,B)$ is $\max_{p_A \in F_A} \min_{p_B \in F_B} \rho(p_A, p_B) = \max_{p_A \in F_A} \rho(p_A, F_B)$ and similarly $h(B,A)$ is defined. p_A and p_B are foreground pixels of images A and B. For example, consider two 3×3 images A and B with their foreground pixels marked '1' and the background pixels marked '0' as shown below.

$$\begin{array}{ccc} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ & A & \end{array} \qquad \begin{array}{ccc} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ & B & \end{array}$$

F_A and F_B are given by $\{p_{A1}=(1,2), p_{A2}=(2,3)\}$ and $\{p_{B1}=(1,1), p_{B2}=(1,2)\}$. Let the distance metric be Euclidean. The different distance values are:

$$\left.\begin{array}{l}\rho(p_{A1},p_{B1})\\ \rho(p_{B1},p_{A1})\end{array}\right\} = 1; \quad \left.\begin{array}{l}\rho(p_{A1},p_{B2})\\ \rho(p_{B2},p_{A1})\end{array}\right\} = 0; \quad \left.\begin{array}{l}\rho(p_{A2},p_{B1})\\ \rho(p_{B1},p_{A2})\end{array}\right\} = \sqrt{5}; \quad \left.\begin{array}{l}\rho(p_{A2},p_{B2})\\ \rho(p_{B2},p_{A2})\end{array}\right\} = \sqrt{2};$$

$$\rho(p_{A1},F_B) = 0; \quad \rho(p_{A2},F_B) = \sqrt{2}; \quad \rho(p_{B1},F_A) = 1; \quad \rho(p_{B2},F_A) = 0;$$

$$h(A,B) = \sqrt{2}; \quad h(B,A) = 1;$$

$$H(A,B) = \sqrt{2};$$

Real-time operations are now integral part of image processing applications. Hardware implementation is essential for real-time applications. In the next section, different VLSI architectures for image processing are discussed.

2.3 VLSI Architectures for Image Processing

In real-time image processing, general-purpose parallel computers cannot offer satisfactory processing speed due to severe system overheads. The rapid advancement in VLSI technology leads to the design of VLSI architectures for real-time applications. In VLSI,

TH-2715_964102

memory and processing power are relatively cheap nowadays and the main emphasis of the design is shifted towards reducing the overall interconnection complexity and keeping the overall architecture highly regular, modular and cascadable. The architecture typically consists of an array of processors [Kun88]. Image processing algorithms are suitable for implementation in this type of VLSI architecture.

VLSI array processors can be classified into (1) Systolic architecture, (2) Wavefront architecture and (3) Cellular architecture. In the following, we discuss these three in detail.

2.3.1 Systolic Architecture

Systolic architecture is one of the architectures used for solving image processing problems. Examples of problems for which the systolic architecture has been designed are median/rank-order filtering for image enhancement [Ofl83, NFMM85], relaxation technique for image restoration [KHL86] and computing Hough transform [CL85, AF95], skeleton [RD95], distance transforms [CY96] and labeling connected components [RMS95] for image analysis. The architecture is a *pipelined architecture* in which input data is pipelined or queued and they are given to the systolic array in batches. The systolic architecture is a network of similar processors which synchronously compute and pass data through the network. The processors are locally interconnected and the data flows through the processors in a specific path. The systolic array of processors features properties such as modularity, regularity, local interconnection, a high degree of pipelining, highly synchronized multiprocessing and continuous flow of data between processors. Systolic arrays are especially appealing for a wide class of compute-bound computations, where multiple operations are performed on each data item in a repetitive manner. The basic configuration of a systolic array is illustrated in Figure 2.8. By replacing a single processor by a 1-D or 2-D array of processors, a higher computation throughput can be achieved without increasing memory bandwidth.

2.3.2 Wavefront Architecture

While systolic arrays are very amenable to VLSI implementation of computation-bound image processing algorithms, the activities are controlled by a global clock. From the

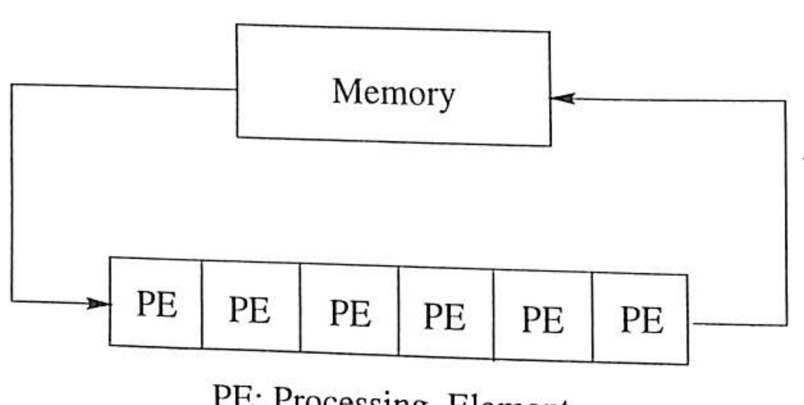


Figure 2.8: Basic configuration of systolic arrays. Memory stores input data and output results of systolic array.

hardware perspective, this global synchronization incurs problems of clock skew, fault tolerance and peak power. The clock skew can vary drastically depending on the size of the array. A simple solution to these problems is to adopt the principle of *data flow computing* in array processors. This leads to the design of wavefront array processors [KAGER82, Kun88].

In the data flow approach, the arrival of data from neighboring processors is interpreted as a signal to change state and to activate new actions. It envisages a distributed and asynchronous array processing system. The asynchronous data-driven model of wavefront array incurs a fixed time delay and hardware overhead due to handshaking. The wavefront array possesses most of the advantages of the systolic array, such as extensive pipelining and multiprocessing, regularity, modularity and cascadability.

2.3.3 Cellular Architecture

Both systolic and wavefront architectures have the characteristics of pipelining of inputs and flow of data in a predefined path. The architectures implement only those algorithms which have these characteristics. Morphological image processing algorithms are not suitable for implementation in pipelined architectures. Dilation and erosion are basic morphological operations and they are commonly performed in image analysis. Morphological algorithms are iterative procedures. At each iteration, all the pixels in the image are updated and for each pixel, operations are carried out within a small neighborhood of it. A cellular architecture is quite suitable for morphological processing. The cellular

TH-2715_964102

architecture is a parallel architecture which does not involve pipelining of inputs. Similar to the systolic and wavefront arrays, the cellular architecture is a network of locally connected identical cells (or processing elements) but the architecture does not incur any hardware overhead since there is no pipelining. Cellular architecture is also simple in design, modular and cascadable. Further, the architecture has high speed of operation due to local interconnection. Each cell is a sequential logic consisting of storage elements and a combinational circuit. The architecture is based on the concept of cellular automata [CCNC97, Wol83, Sip99]. The values stored in a cell represent the local state of the cell and the global state of the architecture is represented by the vector of local states. All the cells are synchronously operated and the cellular architecture evolves in discrete time steps. At any time step, the local state of a particular cell is a function of the local states of the neighboring cells in the previous time step. The function is implemented by the combinational circuit of the cell. Figure 2.9 shows a 2-D cellular architecture in which each cell is connected to all the eight neighboring cells except those cells on the boundary. 2-D cellular architectures are amenable to image processing [IO98].

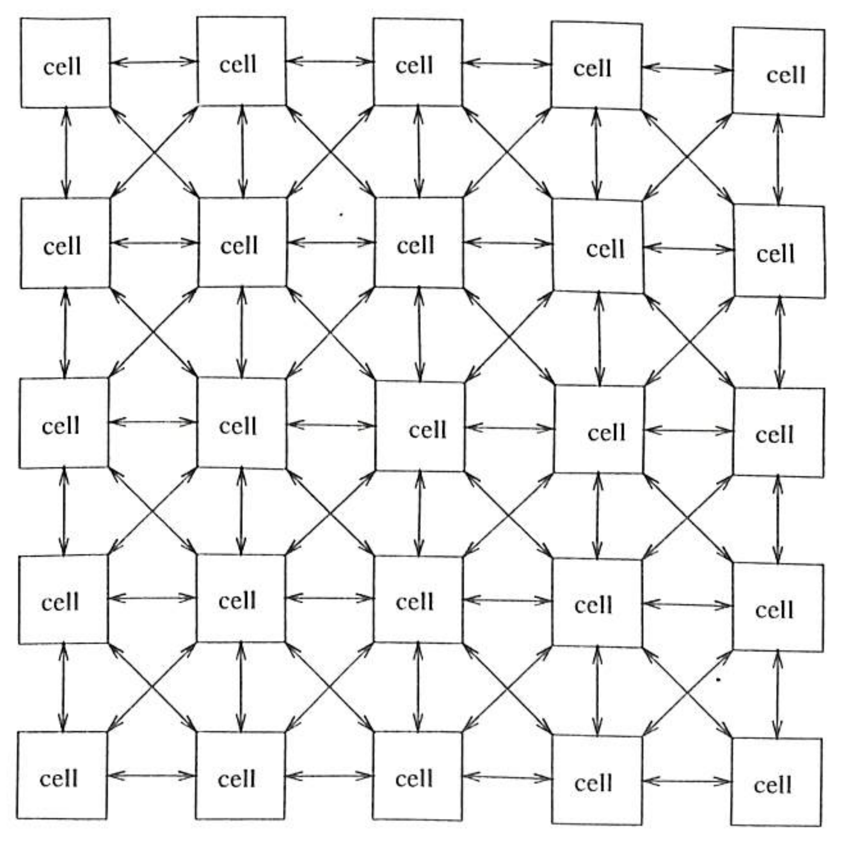


Figure 2.9: A 2-D cellular architecture.

The cellular architecture operates as follows. The architecture is first initialized

by loading the storage elements of the cells with input data. Then the architecture is allowed to run synchronously and the global state changes at each clock cycle until a final state is reached. The outputs are the contents of the storage elements of the cells at the final state.

2.3.4 Comparison of Different Architectures

All the three architectures described above share the important common feature of using a large number of modular and locally interconnected processors for massive parallel processing. The differences are in the hardware design such as clock and pipelining. The comparison of the architectures is given below.

Features	Systolic	Wavefront	Cellular
1. Modular	yes	yes	yes
2. Cascadable	yes	yes	yes
3. Local interconnection	yes	yes	yes
4. Pipelining	yes	yes	no
5. Operation	synchronous	asynchronous, data-driven	synchronous
6. Clock skew	yes	no	yes
7. Additional hardware	pipelining	pipelining, handshaking	no

Among the three architectures, the cellular architecture has been developed recently and many parallel algorithms can be directly mapped onto this architecture. Besides, the architecture does not involve pipelining.

In the next chapter, we review algorithms for computing various distance transforms and their applications.

TH-2715_964102

Chapter 3

Literature Review

In this chapter, we review prior work on distance transforms, skeleton, Voronoi diagram and Hausdorff distance. The survey is therefore divided into four sections. Our discussion focuses on sequential and parallel algorithms given by various researchers. Efficient sequential algorithms are cited and possibilities of parallelization of some existing algorithms are studied.

3.1 Computation of Distance Transforms

Computing the distances from a pixel to a set of 1-pixels is a global operation. The most straightforward approach to the construction of any distance transform involves, for each pixel, scanning of the entire image to find the nearest 1-pixel where "nearest" has different meanings depending on which transform (city-block/chessboard/Euclidean) we refer to. The time complexity of this approach is $O(n^4)$ for an $n \times n$ image. Global operations are often prohibitively costly and unsuitable for VLSI array processors. Therefore, a decomposition strategy using only local operations is useful.

City-block and chessboard distances can be computed by considering only a small neighborhood at a time. A simple solution is to initialize the distance values of 1-pixels to 0. The distance values of 0-pixels can be computed iteratively. In each iteration, for every pixel whose distance is to be computed, the neighboring pixels are checked to see if their distance values are known. In the case of city-block distance, it is adequate to

check the two vertical and two horizontal neighbors. In the case of chessboard distance, the four diagonal neighbors also have to be checked. The distance value of a pixel is then the minimum distance of the neighbors plus one. Efficient algorithms for computation of distance transforms based on these two distances are available in the literature [Ser82, Bor84, Bor86, YI86, Vos88, Pag92, Rag92, Hir96]. The optimal sequential algorithms run in $O(n^2)$ time. Since these two distances are integers and are easy to compute, they have been used for many image processing applications.

The Euclidean distance is however a more natural measure of distance since it corresponds to the straight line distance. Its computation is hard to decompose into local neighborhood operations because it involves a nonlinear square root operation. Hence, algorithms concerning the approximation of Euclidean distance have been extensively studied [Dan80, RK82, Bor84, Bor86, Vos88, Bor89, Bor91].

One approximation to EDT known as the octagonal distance transform is computed by performing the neighborhood operations of city-block and chessboard distance transforms alternatively. The chamfer distance transform, another approximation to EDT, is obtained by performing morphological operation on the given image using the chamfer mask. Since the distance values are integers and their computation is using a small neighborhood mask, the chamfer distance transformation is quite suitable for VLSI implementation. A linear systolic array architecture for chamfer distance transformation is given in [CY96].

Some attempts at exact computation of EDT have also been made in the literature. Most of them are sequential algorithms and some of them have optimal computational complexity [CC94, BGKW95, Hir96, Egg98]. However, it is impossible for a sequential algorithm to have a time complexity better than $O(n^2)$ because each of the n^2 pixels has to be scanned at least once. It is desirable to compute EDT in parallel for time-critical applications.

Kolountzakis and Kutsulakos [KK92] have given an $O(n^2 \log n)$ time sequential algorithm for computing exact EDT. They have parallelized their algorithm on the EREW PRAM (Exclusive Read Exclusive Write Parallel Random Access Machine) model. Their algorithm runs in $O(n^2 \log n/p)$ time using p processors. The algorithm uses a divide and conquer strategy. A summary of the algorithm is given below.

For each pixel (i, j), the algorithm computes the distance r_{ij} from the nearest 1-pixel

TH-2715_964102

	1	2	3	4	5
1	0	0	0	0	0
2	0	1	0	0	0
3	0	0	0	1	0
4	0	1	0	0	0
5	0	0	0	0	0

(a) Image

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	2	2	∞	∞	∞
3	4	4	4	4	∞
4	2	2	∞	∞	∞
5	∞	∞	∞	∞	∞

(b) V_{ij}

	1	2	3	4	5
1	22	22	34	34	$\infty\infty$
2	22	22	34	34	$\infty\infty$
3	22	22	34	34	$\infty\infty$
4	42	42	34	34	$\infty\infty$
5	42	42	34	34	$\infty\infty$

(c) N_{ij}

Figure 3.1: Computation of EDT in [KK92]. In (c), the nearest neighbor $N_{ij} = (x, y)$ is printed as xy.

in the right side (columns greater than j) of (i, j). Computing the distance from the right and from the left and taking the minimum of the two gives the actual distance $d_e(i, j)$. The computation of r_{ij} involves computing V_{ij} for each (i, j). V_{ij} is the column index of the nearest 1-pixel in the right side of (i, j) at the ith row. If there is no 1-pixel, then V_{ij} is ∞. The computation of V_{ij} for all pixels in the ith row requires only a single scan of these pixels and hence the computation of V_{ij} for all i and j takes only $O(n^2)$ time. For example, V_{ij} of all pixels of an image in Figure 3.1(a) is given in Figure 3.1(b). Using the array of elements V_{ij}, the nearest 1-pixel N_{ij} in the right side of (i, j) is found by scanning only the jth column of the array. This involves computation of Euclidean distance between (i, j) and (k, V_{kj}) for each k. N_{ij} for all pixels of the example image (Figure 3.1(a)) is given in Figure 3.1(c). The authors have proposed a divide and conquer approach for finding N_{ij} and it takes $O(n^2 \log n)$ for the entire image. The parallel version of this algorithm with p processors computes all V_{ij} by assigning n/p consecutive rows to each processor. This computation takes $O(n^2/p)$ time. Once V_{ij} is computed for all i and j, n/p consecutive columns are assigned to each processor to compute N_{ij}. This computation takes $O(n^2 \log n/p)$. The algorithm described above does not have local neighborhood operations for each pixel. Besides, finding N_{ij} involves computing Euclidean distance which is a nonlinear operation. Hence it is not cost-effective to implement the algorithm in VLSI.

We now study another sequential algorithm for which also a parallel version is known. The algorithm has been proposed by Hirata [Hir96] and it runs in $O(n^2)$ time. The parallel version of this algorithm runs in $O(n^2/p)$ time with p $(1 \leq p \leq n)$ processors. Similar to [KK92], the algorithm finds EDT in two phases. For each pixel (i, j), the algorithm first computes g_{ij} by finding the nearest 1-pixel in the ith column. If (r, j) is the nearest 1-pixel, then g_{ij} is assigned the value $|i - r|$. g_{ij} is ∞ if there is no 1-pixel.

The computation of g_{ij} for all pixels in the jth column requires only a single scan of the column. Hence, its computation for the entire image takes $O(n^2)$ time. The g_{ij} values of pixels in Figure 3.1 (a) is given below.

	1	2	3	4	5
1	∞	1	∞	2	∞
2	∞	0	∞	1	∞
3	∞	1	∞	0	∞
4	∞	0	∞	1	∞
5	∞	1	∞	2	∞

g_{ij}

Once g_{ij} is computed for all i and j, the Euclidean distance value $d_e(i,j)$ is computed by scanning the pixels (i,q), $1 \leq q \leq n$, in the ith row. $d_e(i,j)$ is given by the minimum of $(j-q)^2 + g_{iq}$ for each (i,q). The author has proposed an algorithm using stack to compute $d_e(i,j)$ from g_{ij}. It takes $O(n^2)$ time for the entire image. The columns can be scanned independently for computing g_{ij} and similarly the rows for $d_e(i,j)$. In the parallel version of the algorithm using p processors, n/p columns are assigned to one processor while computing g_{ij} and n/p rows are assigned while computing $d_e(i,j)$. However, this parallelism involve only global operations and hence this algorithm too cannot be implemented in a cellular architecture.

A mathematical morphology approach to exact computation of EDT is presented in [SM92], where the Euclidean distance transform is decomposed into a set of 3×3 neighborhood local operations. The operations involve real-value arithmetic and hence the approach cannot be efficiently implemented in VLSI. The authors in [SM92] propose a parallel pipelined computer architecture to compute the exact EDT. This requires $n(n+1)/2$ computers for an image of size $(2n+1) \times (2n+1)$. While they achieve time complexity of $O(n)$ for this parallel configuration, the large requirements on the number of computers (each needing a memory unit of size $O(n^2)$ to store the entire image) makes it somewhat less attractive in practice.

TH-2715_964102

3.2 Construction of the Skeleton

Getting a connected skeleton in a discrete (or digital) image is not generally straightforward. Many sequential algorithms have been discussed in the literature [GF96, AdB85, AdB93, LL92, LW93, SP95, NGC92]. The skeletons generated by most of these algorithms are influenced by the noise in the boundary caused due to the discretization of the image. The algorithms find the skeleton either through iterative thinning [AdB85, LL92, LW93] or from the distance transform of the given image [GF96, AdB93, SP95, NGC92, JC90]. The algorithms based on iterative thinning remove the boundary pixels of objects until each object reduces to a thin structure. Although these algorithms are suitable for VLSI implementation in a two dimensional array of processors, the skeleton obtained does not correspond to the one defined in the beginning of this section. Moreover, the reconstruction of objects may not be possible.

The skeletons computed through distance transforms of images differ with respect to the distance metric. The skeleton based on Euclidean distance, termed Euclidean skeleton, is invariant to the orientation of the objects in the image. Different methods for obtaining the skeleton using distance transforms other than the Euclidean distance transform are available in [AdB89, NGC92, dB94, dBT94, CCB96, dBT96, AdB96]. A systolic algorithm [RD95] for computing the skeleton from city-block distance transform is available. The algorithm computes the skeleton by making two scans over the distance transform of the image. In each scan, the distance values within a 3×3 neighborhood are compared for each pixel to identify the skeleton pixels. The algorithm is specific to the city-block distance metric only. The architecture consists of n processing elements and takes $2(6n+n^2)$ clock cycles to compute the skeleton of an $n \times n$ image. The skeletons based on distance transforms other than EDT are not unique for a particular shape of an object.

The methods for generating Euclidean skeleton are as follows. Generally, they are based on extracting the centers of maximal discs that can be fitted within the object [KK87, BA91, AdB93, GF96] or by finding the pixels with locally maximal distance values [ACL81, Dor86, AdB88, SP90, SP91, AdB92, SP95]. Though the methods based on the latter are amenable to VLSI implementation, it is not attractive to use a hardware for extracting skeleton from EDT. One would like to extract skeleton directly from the binary image of objects. A hardware implementation for computing Euclidean skeleton of a binary image is not available in the literature to our knowledge. In Chapter 5

of this thesis, we give a parallel algorithm for computing Euclidean skeleton and its implementation in a cellular architecture.

3.3 Construction of the Voronoi Diagram

The construction of discrete Voronoi diagram of objects in a binary image can be done conveniently using image processing techniques. The Voronoi region of an object consists of those pixels that are closer to that object than any other object. The Voronoi diagram differs greatly from the expected one, if the same definition as that adopted in the continuous case is employed. Some of the points in the Voronoi diagram may be missing. For example, when the shortest path connecting the two interacting objects contains an even number of pixels, then the Voronoi diagram contains the mid point of the path which will be missing. Hence obtaining a connected discrete Voronoi diagram is not trivial.

One method of constructing discrete Voronoi diagram is by labeling different objects and propagating the labels into the background with constant speed. Here, each object is considered to be a connected component. A Voronoi region comprises of all pixels with same label and the Voronoi diagram can be found out by definition. In this method, the labeling and propagating labels are performed one after another and they are time consuming. Also labels need to be stored.

Another approach to construct discrete Voronoi diagram is using distance transforms. A sequential algorithm to construct the discrete Voronoi diagram of objects in a binary image based on city-block distance is given in [AdB85]. Here the distance transform of the background is first employed to find the exoskeleton (skeleton of background) and the Voronoi diagram is successively obtained after removal of certain pixels. This method is specific to city-block distance metric only. A dynamic algorithm for constructing discrete Voronoi diagram, based on Euclidean distance, of feature pixels in a binary image is given in [SP97]. The algorithm updates the diagram whenever a new feature pixel is added or removed. It is not suitable for constructing Voronoi diagram of objects.

As far as VLSI implementation for computing Voronoi diagram is concerned, Tzionas *et al* [TTT94, TTT97] have proposed a cellular architecture for constructing Voronoi diagram. The diagram is based on city-block distance. They applied the diagram for

nearest neighbor pattern classification [TTT94] and robot path planning [TTT97]. In the case of pattern classification, the Voronoi diagram is constructed for points and in the case of robot path planning, it is constructed for objects. The Voronoi diagram of objects is constructed as follows. Initially, the boundaries of the objects are identified and coded with respect to their orientation. These codes are then loaded as the initial source values onto the cellular architecture. The cellular architecture starts its evolution in time and each cell propagates its code to the four neighboring cells (two horizontal and two vertical neighbors) at each time step. The Voronoi diagram consists of those cells which receive different codes from their neighbors. The algorithm actually constructs Voronoi diagram of edges (horizontal, vertical and diagonal segments) of the boundaries of objects instead of objects itself. In Chapter 6, we propose a method for constructing Voronoi diagram of objects, based on Euclidean distance, on a binary image. The method involves only local neighborhood operations for each pixel and is suitable for VLSI implementation.

3.4 Computation of the Hausdorff Distance

Consider two binary images A and B. If n_A and n_B are the number of foreground pixels of A and B, then the straightforward method of computing $h(A,B)$ or $h(A,B)$ takes $O(n_A n_B)$ time. In the worst case, n_A and n_B equal n^2. Better algorithms are available in literature. A linear-time sequential algorithm for Hausdorff distance between images based on city-block metric is presented in [Sho89]. The algorithm computes the distance transforms of images and Hausdorff distance is computed from these transform values. The author of [Sho89] has extended the computation of Hausdorff distance to any L_p[1] distance metric and a linear-time algorithm for the computation is given in [Sho91]. The algorithm involves complex arithmetic operations and the possibility of parallelization of the algorithm on shared memory machines is briefly discussed. The usefulness of Hausdorff distance for image matching is discussed in [HKR93]. Here, the authors define partial Hausdorff distance to compare portions of patterns in images. This is used to locate a given model in an image at least partially, by computing the partial Hausdorff distance between all relative positions of a model and the test image.

[1]The L_p distance between two points (x_1, y_1) and (x_2, y_2) is given by $[|x_1 - x_2|^p + |y_1 - y_2|^p]^{1/p}$. City-block distance is L_1 metric and Euclidean distance is L_2 metric.

This computation is time-consuming. Even though some speed up techniques have been developed in [HKR93] which uses EDT, further improvement is desirable. The speed can be increased by means of parallel computation methods. Some parallel algorithms and their implementations are available in literature [MT95, ADP96a, ADP96b] for the computation of Hausdorff distance between images. These implementations are carried out using massively parallel computers. In some other work [YPC95, YCZP96], parallelization is achieved by means of distributed systems. To the best of our knowledge, no VLSI implementation is available in literature. In Chapter 7 of this thesis, we propose a method which is suitable for VLSI implementation.

3.5 Conclusions

We have surveyed the work on the computation of various distance transforms and their approximations, construction of skeleton and Voronoi diagram and computation of the Hausdorff distance. Many sequential algorithms have been cited for solving all these problems. However, when image size is large (say $n \geq 100$ for an $n \times n$ image), efficient sequential algorithms are insufficient for time critical applications.

Parallel solutions on general purpose architectures have been proposed for some problems but much less is known about VLSI based solutions. Our efforts to obtain speed up while maintaining cost-effectiveness have culminated in algorithms that exploit local interconnection type of characteristics of cellular structures. The details are given from the next chapter onwards.

TH-2715_964102

Chapter 4

Cellular Architecture for Euclidean Distance Transformation

4.1 Introduction

Earlier chapters present the definition of the Euclidean distance transformation and a review of prior work. The prior algorithms for the transformation have difficulties in VLSI implementation.

In this chapter, we propose an algorithm for the computation of Euclidean Distance Transform (EDT). The algorithm is amenable to VLSI implementation in a cellular architecture. Besides, the algorithm computes the Nearest Neighbor Transform (NNT). The algorithm involves simple arithmetic operations performed in a 3×3 neighborhood of each pixel. The cellular architecture consists of identical cells each having a simple combinational logic of a few adders, comparators and registers. The algorithm has been reported in [SNBS98].

In the next section, we derive the main results that are helpful to describe our algorithm for computing EDT and NNT.

4.2 Mathematical Foundations for the Algorithm

We begin with a review of the definitions. The Euclidean distance transformation of an image I, consisting of foreground F (0-pixels) and background B (1-pixels), converts the binary image to a multi-valued image in which each pixel p_0 is assigned the Euclidean distance $d_e(p_0)$ between p_0 and the nearest background pixel in I.

The nearest neighbor transformation of a binary image is an assignment, to each pixel p_0, of a two component displacement vector $(\Delta x(p_0), \Delta y(p_0))$ which points to the nearest background pixel. It is also called as signed or vector Euclidean distance transformation [Ye88]. The Euclidean distance $d_e(p_0)$ of the pixel p_0 is given by the magnitude, $\sqrt{\Delta x^2(p_0) + \Delta y^2(p_0)}$. If the pixel p_0 is at the location (x, y), then its nearest neighbor is at the location $(x + \Delta x(p_0), y + \Delta y(p_0))$. Even the city-block and chessboard distances can be obtained from the displacement vector. The city-block distance is given by $|\Delta x(p_0)| + |\Delta y(p_0)|$ and chessboard distance is given by $\max(|\Delta x(p_0)|, |\Delta y(p_0)|)$.

Let $D(p_0)$ be the square of the Euclidean distance $d_e(p_0)$ between the pixel p_0 and its nearest background pixel. It can also be given by $\Delta x^2(p_0) + \Delta y^2(p_0)$. Note that $D(p_0)$, $\Delta x(p_0)$ and $\Delta y(p_0)$ are all integers since the coordinates of a pixel are integers. Further, let $d(p_0)$ be the nearest integer approximation of $d_e(p_0)$: that is, it satisfies the inequality $(d(p_0) - 0.5)^2 < D(p_0) \leq (d(p_0) + 0.5)^2$. In other words, $d^2(p_0) - d(p_0) < D(p_0)$. Also, $D(p_0) \leq d^2(p_0) + d(p_0)$ since $D(p_0)$ is an integer. The error in the integer approximation is $\pm$ 0.5 pixel unit. Figure 4.1 shows the d values of pixels within 4 pixel units and their $(\Delta x, \Delta y)$ values from a pixel at row i and column j of a binary image.

In the proposed method for computing EDT and NNT, we iteratively compute $d(p_0)$ and $(\Delta x(p_0), \Delta y(p_0))$ for every pixel p_0 in the binary image. Whenever necessary, $d_e(p_0)$ is computed using $\Delta x(p_0)$ and $\Delta y(p_0)$. The iterative procedure involves initializing $\Delta x(p_0)$, $\Delta y(p_0)$, $D(p_0)$ and $d(p_0)$ of all pixels p_0 with respect to the given image. The initialization is as follows.

$$\Delta x^{(0)}(p_0) = \Delta y^{(0)}(p_0) = D^{(0)}(p_0) = d^{(0)}(p_0)$$
$$= \begin{cases} 0 & p_0 \in B \\ \infty & p_0 \in F \end{cases} \tag{4.1}$$

TH-2715_964102

columns

rows	j-4	j-3	j-2	j-1	j	j+1	j+2	j+3	j+4
i-4			4 (4,2)	4 (4,1)	4 (4,0)	4 (4,-1)	4 (4,-2)		
i-3		4 (3,3)	4 (3,2)	3 (3,1)	3 (3,0)	3 (3,-1)	4 (3,-2)	4 (3,-3)	
i-2	4 (2,4)	4 (2,3)	3 (2,2)	2 (2,1)	2 (2,0)	2 (2,-1)	3 (2,-2)	4 (2,-3)	4 (2,-4)
i-1	4 (1,4)	3 (1,3)	2 (1,2)	1 (1,1)	1 (1,0)	1 (1,-1)	2 (1,-2)	3 (1,-3)	4 (1,-4)
i	4 (0,4)	3 (0,3)	2 (0,2)	1 (0,1)	0 (0,0)	1 (0,-1)	2 (0,-2)	3 (0,-3)	4 (0,-4)
i+1	4 (-1,4)	3 (-1,3)	2 (-1,2)	1 (-1,1)	1 (-1,0)	1 (-1,-1)	2 (-1,-2)	3 (-1,-3)	4 (-1,-4)
i+2	4 (-2,4)	4 (-2,3)	3 (-2,2)	2 (-2,1)	2 (-2,0)	2 (-2,-1)	3 (-2,-2)	4 (-2,-3)	4 (-2,-4)
i+3		4 (-3,3)	4 (-3,2)	3 (-3,1)	3 (-3,0)	3 (-3,-1)	4 (-3,-2)	4 (-3,-3)	
i+4			4 (-4,2)	4 (-4,1)	4 (-4,0)	4 (-4,-1)	4 (-4,-2)		

Figure 4.1: Nearest integer Euclidean distance d and displacement vector $(\Delta x, \Delta y)$ of pixels from a pixel at row i and column j. The center of each cell of the grid represents a pixel.

The next iterates (values for $D^{(k+1)}(p_0), \Delta x^{(k+1)}(p_0), \Delta y^{(k+1)}(p_0), k = 0, 1, 2, \ldots$) are obtained as follows. The expressions use the pictorial representation of the 3 x 3 neighborhood of pixel p_0 denoted by $N_E(p_0)$ shown in Figure 4.2. If p_0 is a corner pixel in the image, then its neighbors inside the image are only of interest.

$$D^{(k+1)}(p_0) = \min_{p_i \in N_E(p_0)} [\Delta x_i^2 + \Delta y_i^2] \tag{4.2}$$

where

$$\Delta x_i = \begin{cases} |\Delta x^{(k)}(p_i)| & \text{for } i = 0, 1 \& 5 \\ |\Delta x^{(k)}(p_i)| + 1 & \text{for } i = 2, 3, 4, 6, 7 \& 8 \end{cases} \tag{4.3}$$

$$\Delta y_i = \begin{cases} |\Delta y^{(k)}(p_i)| & \text{for } i = 0, 3 \& 7 \\ |\Delta y^{(k)}(p_i)| + 1 & \text{for } i = 1, 2, 4, 5, 6 \& 8 \end{cases} \tag{4.4}$$

	$y-1$	y	$y+1$
$x-1$	p_4	p_3	p_2
x	p_5	p_0	p_1
$x+1$	p_6	p_7	p_8

Figure 4.2: Pixels in the 3 x 3 neighborhood of p_0.

The quantities Δx_i and Δy_i are positive and gives the minimum number of horizontal and vertical steps required to reach the nearest background pixel. In the expression for $D^{(k+1)}(p_0)$ given by Equation (4.2), the nonlinear squaring operation makes implementation using combinational logic expensive. We therefore rewrite $D^{(k+1)}(p_0)$ as follows.

$$D^{(k+1)}(p_0) = \min_{p_i \in N_E(p_0)} [D^{(k)}(p_i) + \Delta X_i + \Delta Y_i] \tag{4.5}$$

where

$$\Delta X_i = \begin{cases} 0 & \text{for } i = 0, 1 \& 5 \\ 2|\Delta x^{(k)}(p_i)| + 1 & \text{for } i = 2, 3, 4, 6, 7 \& 8 \end{cases} \tag{4.6}$$

$$\Delta Y_i = \begin{cases} 0 & \text{for } i = 0, 3 \& 7 \\ 2|\Delta y^{(k)}(p_i)| + 1 & \text{for } i = 1, 2, 4, 5, 6 \& 8 \end{cases} \tag{4.7}$$

The remaining quantities, namely the components of the displacement vector and the distance value of pixel p_0, are updated as follows. $|\Delta x^{(k+1)}(p_0)|$ and $|\Delta y^{(k+1)}(p_0)|$ take the corresponding values of Δx_i and Δy_i that yield $D^{(k+1)}(p_0)$ in Equation (4.5). If more than one neighbor p_i of p_0 give rise to $D^{(k+1)}(p_0)$, then $|\Delta x^{(k+1)}(p_0)|$ and $|\Delta y^{(k+1)}(p_0)|$ take Δx_i and Δy_i corresponding to minimum i. The signs, $S_{\Delta x}(p_0)$ and $S_{\Delta y}(p_0)$, of $\Delta x^{k+1}(p_0)$ and $\Delta y^{k+1}(p_0)$ are given by

$$S_{\Delta x}(p_0) = \begin{cases} '+' & \text{if } p_i \in \{p_2, p_3, p_4\} \\ '-' & \text{if } p_i \in \{p_6, p_7, p_8\} \\ S_{\Delta x}(p_i) & \text{if } p_i \in \{p_0, p_1, p_5\} \end{cases} \tag{4.8}$$

$S_{\Delta x}(p_0)$ is '+' if p_i is in $(x-1)$th row, '-' if p_i is in $(x+1)$th row and it is the sign of $\Delta x(p_i)$ if p_i is in row x. Similarly,

TH-2715_964102

$$S_{\Delta y}(p_0) = \begin{cases} '+' & \text{if } p_i \in \{p_4, p_5, p_6\} \\ '-' & \text{if } p_i \in \{p_1, p_2, p_8\} \\ S_{\Delta y}(p_i) & \text{if } p_i \in \{p_0, p_3, p_7\} \end{cases} \tag{4.9}$$

If the pixel p_0 is at a distance k from the nearest background pixel, then $d^{(k)}(p_0) = k$ and $D^{(k)}(p_0) \leq k^2 + k$. $d(p_0)$ can be given iteratively as follows.

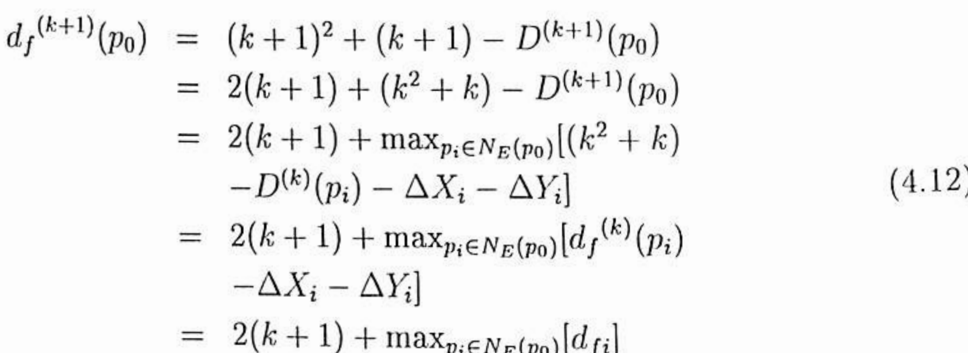
$$d^{(k+1)}(p_0) = \min \left[d^{(k)}(p_0), k+1 : D^{(k+1)}(p_0) \leq (k+1)^2 + (k+1) \right] \tag{4.10}$$

As far as hardware implementation is concerned, the values of $\Delta x(p_0), \Delta y(p_0)$ and $D(p_0)$ at each iteration have to be stored in memory elements. $D(p_0)$ is however quite large in magnitude. Hence, its computation needs a complex digital circuit and large storage space. To reduce the storage and the hardware complexity, we define a quantity $d_f(p_0)$ in terms of k and $D(p_0)$. However, once $d_f(p_0)$ is initialized, $d_f^{(k+1)}(p_0)$ can be expressed in terms of $d_f^{(k)}(p_0)$ and other quantities without involving $D(p_0)$. The iterative computation of $d_f(p_0)$ has the following initialization.

$$d_f{}^{(0)}(p_0) = \begin{cases} 0 & p_0 \in B \\ -\infty & p_0 \in F \end{cases} \tag{4.11}$$

At iteration $k+1$, $d_f(p_0)$ is given by [1]

[1] Note: The computation of $d_f(p_0)$ depends on k. Even though the displacement vector of p_0 is found out at some iteration, $d_f(p_0)$ will get updated for successive iterations because the updated values are used by the neighbors of p_0 for computing their displacement vectors. The updation is nothing but adding $2k$ at iteration k. This is because, once the distance values are computed, the term $\max_{p_i \in N_E(p_0)}[d_{fi}]$ in Equation (4.12) becomes $d_f^k(p_0)$. Since the neighbors are at a distance less than 2 pixel units from p_0, they always receive the transform values in the next two iterations.



$$\begin{aligned} d_f^{(k+1)}(p_0) &= (k+1)^2 + (k+1) - D^{(k+1)}(p_0) \\ &= 2(k+1) + (k^2+k) - D^{(k+1)}(p_0) \\ &= 2(k+1) + \max_{p_i \in N_E(p_0)}[(k^2+k) \\ &\quad -D^{(k)}(p_i) - \Delta X_i - \Delta Y_i] \\ &= 2(k+1) + \max_{p_i \in N_E(p_0)}[d_f^{(k)}(p_i) \\ &\quad -\Delta X_i - \Delta Y_i] \\ &= 2(k+1) + \max_{p_i \in N_E(p_0)}[d_{fi}] \end{aligned} \quad (4.12)$$

where $d_{fi} = d_f^{(k)}(p_i) - \Delta X_i - \Delta Y_i$.

$d^{(k+1)}(p_0)$ can now be rewritten as

$$d^{(k+1)}(p_0) = \min[d^{(k)}(p_0), k+1 : d_f^{(k+1)}(p_0) \geq 0] \quad (4.13)$$

The proposed algorithm is based on the following results.

$$d_f^{(k+1)}(p_0) = 2(k+1) + \max_{p_i \in N_E(p_0)}[d_{fi}] \quad (4.14)$$

where $d_{fi} = d_f^{(k)}(p_i) - \Delta X_i - \Delta Y_i$.

$$(|\Delta x^{(k+1)}(p_0)|, |\Delta y^{(k+1)}(p_0)|) = (\Delta x_i, \Delta y_i) \quad (4.15)$$

where i corresponds to pixel p_i that satisfies Equation (4.14). The quantities $\Delta x_i, \Delta y_i, \Delta X_i$ and ΔY_i are given by Equations (4.3),(4.4), (4.6) and (4.7).

$$S_{\Delta x}(p_0) = \begin{cases} '+' & \text{if } p_i \in \{p_2, p_3, p_4\} \\ '-' & \text{if } p_i \in \{p_6, p_7, p_8\} \\ S_{\Delta x}(p_i) & \text{if } p_i \in \{p_0, p_1, p_5\} \end{cases} \quad (4.16)$$

$$S_{\Delta y}(p_0) = \begin{cases} '+' & \text{if } p_i \in \{p_4, p_5, p_6\} \\ '-' & \text{if } p_i \in \{p_1, p_2, p_8\} \\ S_{\Delta y}(p_i) & \text{if } p_i \in \{p_0, p_3, p_7\} \end{cases} \quad (4.17)$$

The following section describes the algorithm.

TH-2715_964102

4.3 Algorithm for Computing EDT and NNT

In this section, the pseudocode of the proposed algorithm for computing EDT and NNT of a given binary image is presented based on the results obtained in the previous section. Given a binary image of size $n \times n$, the Euclidean distance values d_e and the nearest neighbor transform values (displacement vectors $(\Delta x, \Delta y)$) of the background pixels are set to zero. The corresponding values for the foreground pixels are computed iteratively from the previously computed $\Delta x, \Delta y$ and d_f values of the neighboring pixels. The algorithm builds the contour consisting of equidistant pixels at each iteration whose d values (nearest integer approximation to d_e) equal the iteration number. The transform values of these pixels are computed.

ALGORITHM: *EDT_NNT*

Inputs: Given binary image, consisting of foreground and background pixels.

Outputs: The integer approximation $d(p_0)$ to Euclidean distance and NNT value $(\Delta x(p_0), \Delta y(p_0))$ of each pixel p_0 in the given image; $d_e(p_0)$ is obtained using values of $\Delta x(p_0)$ and $\Delta y(p_0)$.

Step 1: Initialization

The initialization of $\Delta x(p_0), \Delta y(p_0)$ and $d_f(p_0)$ is given by Equations (4.1) and (4.11). To avoid initializing with ∞, we assign a flag *done* to each pixel p_0 whose value is set to 1 when the transform values of the pixel are computed at any iteration. The details are as follows.

$$|\Delta x^{(0)}(p_0)| = |\Delta y^{(0)}(p_0)| = d_f^{(0)}(p_0) = 0$$

$$S_{\Delta x}(p_0) = S_{\Delta y}(p_0) = '+'$$

$$done(p_0) = \begin{cases} 1 & p_0 \in B \\ 0 & p_0 \in F \end{cases}$$

Step 2: Iterative process

At each iteration $k > 0$, the vector $(\Delta x, \Delta y)$ of pixels whose d values equal k are computed. The pseudocode for the iterative process is as follows.

```
k = 0
while there exists pixel p such that done(p) = 0
  for all pixels p0 do in parallel

    switch (done(p0))
    1: {Transform values have been computed}
      Update d_f(p0) by adding 2(k + 1)
      break {switch done = 1}

    0: {Transform values have not been computed}
      if there exists p_i ∈ N_E(p0) such that done(p_i) = 1
        Compute d_f^(k+1)(p0) as in Equation (4.14)
          by considering only the neighbors whose done(p_i) = 1
        if (d_f^(k+1)(p0) ≥ 0) do
          d(p0) = k + 1
          Compute (|Δx^(k+1)(p0)|, |Δy^(k+1)(p0)|) as in Equation (4.15)
          Assign the signs S_Δx(p0) and S_Δy(p0) as in Equations (4.16) and (4.17)
          done(p0) = 1
        end if
      end if
      break {switch done = 0}

  end for
  k = k + 1
end while
```

TH-2715_964102

The above procedure runs until the *done* flags of all the pixels are set to 1. This means that the transform values of all pixels are computed. In each iteration, all the pixels are processed in parallel. The computation of transform values in each pixel is carried out depending on *done* flag. If done is 1, then the transform values have been already computed. Hence, d_f is only updated. The updation is done by adding $2(k+1)$ to the existing value of d_f. On the other hand, if *done* is 0, then we check the *done* flags of neighbors. If there exists some *done* set to 1, then d_f is computed using the values of neighbors whose *done* are set. If $d_f \geq 0$, then $(\Delta x, \Delta y)$ is computed. d is assigned the iteration number and *done* is set.

4.3.1 Working of Algorithm: An Illustration

Let us illustrate the working of the algorithm for a 5×10 image with two background pixels at the image coordinates (3,3) and (3,8). The values of *done*, d_f, d, d_e and $(\Delta x, \Delta y)$ for different pixels at different iterations are shown in Figure 4.3. The propagation of transform values is shown by arrows. The arrow from a pixel p' to a pixel p'' indicates that the neighbor p' of p'' satisfies the equation for computing $d_f(p'')$.

Consider the pixel p_0 at row 3 and column 5 at iteration $k = 2$. Only the neighbors p_i, $i = 4, 5\&6$ of p_0 (refer Figure 4.2) have $done(p_i) = 1$. Their d_{fi} values are -6,-2 and -6. $\max[d_{fi}]$ is -2 which corresponds to p_5, the pixel at row 3 and column 4, and $d_f^{(2)}(p_0) = 2$. $(\Delta x(p_0), \Delta y(p_0))$ is computed from $(\Delta x(p_5), \Delta y(p_5))$. The propagation of information (transform values) starting from the background pixels gives rise to 8-ary trees[2] with the roots being the background pixels. The d values correspond to the levels at which the pixels are in the trees. The maximum value of d in an image gives the maximum of heights of all trees.

4.3.2 Complexity Analysis

The *time complexity* of the proposed algorithm depends mainly on the **while** loop since the statements within **for do-done** are executed in parallel for all pixels. The loop runs until the transform values of all the pixels have been computed. This means the number

[2] An m-ary tree is a tree in which each node has at most 8 children

Figure 4.3: Illustration of Algorithm EDT_NNT. Each rectangle representing a pixel contains the values of $done$,d_f,d,d_e and $(\Delta x, \Delta y)$.

TH-2715_964102

of iterations equals the maximum distance value d that a pixel takes in the given image. The worst case scenario is when there is only one corner background pixel. This is the length of diagonal of rectangular image grid. For an image of size $m \times n$, it cannot be greater than $\sqrt{m^2+n^2}$. Let $n = \max(n, m)$. Then the maximum length would be $\sqrt{2}n$ which implies a time complexity of $O(n)$. The *space complexity* of the algorithm is $O(n^2)$, since the algorithm needs constant space to store $\Delta x, \Delta y, d, d_f$ and *done* for each pixel.

Remark 1 *The algorithm runs until the number of iterations equals the maximum d value that a pixel takes in the given image. Since the image has fixed number of rows and columns, the maximum d value is also finite. Hence, the algorithm converges in finite number of iterations.*

The identical local operations performed in a local neighborhood of each pixel make the algorithm feasible for VLSI implementation in a 2-D array of locally connected identical cells. Before describing VLSI implementation, we provide some simulation results of the algorithm.

4.4 Simulation results of Algorithm

The algorithm has been implemented in C and simulated on a sequential computer (HP 9000 series 700 J Model J200 workstation with 64 MB RAM). It has been tested on some binary images containing different types of objects as foregrounds. The test images are shown in Figure 4.4. The integer distance value, $d(p)$, computed by the algorithm for every pixel p has been transformed into intensity levels and plotted in Figure 4.4. For larger distance values, the pixels are assigned darker intensity values. It can be observed from the figure that the pixels are less dark along the boundaries of objects and they are darker while moving deep inside the objects. The equidistant contours in the foregrounds of images are also shown in Figure 4.4. The contours correspond to the distance values which are multiples of 4. We can observe from the plots that the contours are parallel to the boundary of the corresponding object.

(a) (e) (i)

(b) (f) (j)

(c) (g) (k)

(d) (h) (l)

Figure 4.4: Simulation results of algorithm EDT_NNT. (a),(b),(c) and (d) Test images. (e),(f),(g) and (h) Intensity plots of integer Euclidean distance values computed by the algorithm. (i),(j),(k) and (l) Contour plots of integer Euclidean distance values.

TH-2715_964102

4.5 VLSI Architecture

The proposed algorithm can be implemented in hardware with an $n \times n$ array of cells. Each cell is a sequential logic consisting of storage elements for Δx, Δy, d_f, d and *done*. These are stored in binary form. The computation of these values are implemented by pure combinational logic circuits or using data processing elements such as adder/subtracter and comparator. The size of the storage elements Δx, Δy, d_f and d depends on the size of the image and *done* is one bit in size. Δx and Δy are represented in sign magnitude form. The sign bit is 0 for $S_{\Delta x}$ and $S_{\Delta y}$ being '+' or 1 otherwise. The magnitudes $|\Delta x|$ and $|\Delta y|$ require $\lceil \log_2 m \rceil$ and $\lceil \log_2 n \rceil$ bits to store them. d can be stored in $\lceil \log_2 d_{max} \rceil$ bits where $d_{max} = \lceil \sqrt{m^2 + n^2} \rceil$. The size of d_f can be derived as follows.

If a pixel p receives its distance values at some iteration k, then $d_f^{(k)}(p)$ is bounded by $2k$ as per its definition. Also, once p's transform values have been computed, $d_f(p_0)$ need to be updated in the next two iterations, *i.e.*, $k+1$ and $k+2$. Updation is nothing but adding $2(k+1)$ and $2(k+2)$ to $d_f^{(k)}(p)$. The value of $d_f(p)$ upto $2k + 2(k+1) + 2(k+2)$ is useful for computations. Since the maximum number of iterations for an $m \times n$ image is d_{max}, where $d_{max} = \lceil \sqrt{m^2 + n^2} \rceil$, the size of storage element d_f of any cell can be $\lceil \log_2 6d_{max} \rceil$ bits.

Each cell is connected to its neighboring eight cells through which it receives the values stored in the neighbors. The neighborhood connectivity is shown in Figure 4.5. The iteration number k is generated by an external counter or by a counter kept inside the cell itself and the cells are updated synchronously with respect to an external clock. The block diagram of a cell with its inputs and outputs is shown in Figure 4.6.

The different components in the combinational logic (CL) for $(\Delta x, \Delta y)$ and d_f are as follows.

CL for $(\Delta x, \Delta y)$

It implements the computation of $|\Delta x|$ and $|\Delta y|$ as in Equation (4.15). This involves the computation of Δx_i and Δy_i for i=1 to 8. The values of Δx_i and Δy_i are given by Equations (4.3) and (4.4). This requires an addition of 1 to the values $|\Delta x(p_i)|$ and $|\Delta y(p_i)|$ received from the neighbors p_i. Hence the computation of $(|\Delta x|, |\Delta y|)$ needs

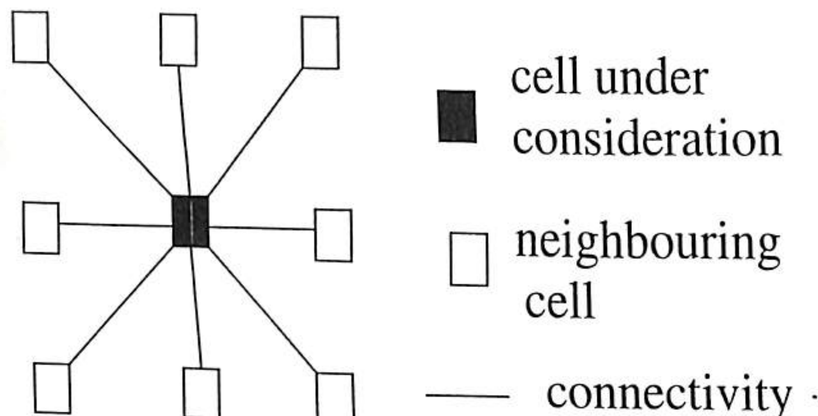


Figure 4.5: Neighborhood connectivity of a cell.

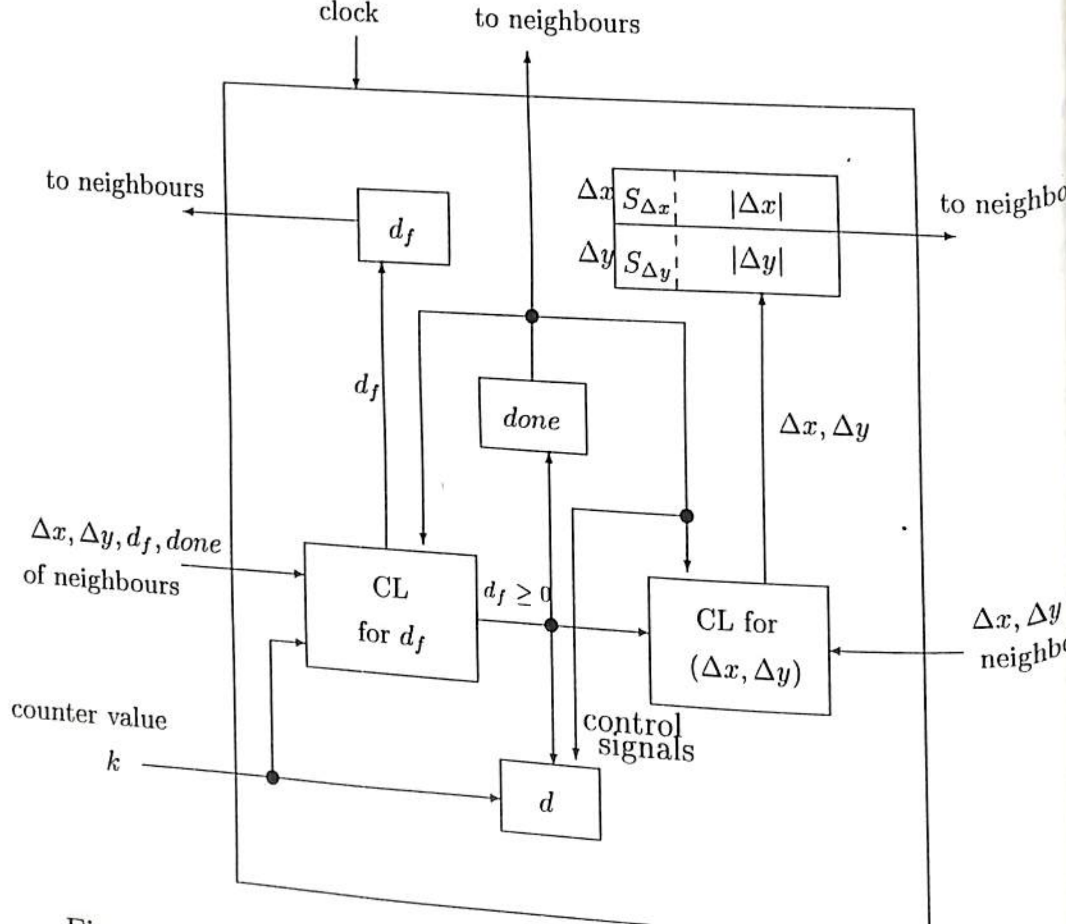


Figure 4.6: Block diagram of a cell. CL: Combination Logic.

TH-2715_964102

12 incrementers, 6 for computing Δx_i values and another 6 for computing Δy_i values. Assigning the signs of Δx and Δy do not need specific combinational logic.

CL for d_f

The computation of d_f is given by Equation (4.14). The computation of d_{fi} involves an addition and a subtraction for $i = 2, 4, 6\&8$ and only a subtraction for $i = 1, 3, 5\&7$. Finding the maximum of all values of d_{fi}, i=1 to 8, requires a comparison logic for its implementation. The computation of ΔX_i, ΔY_i and $2(k+1)$ do not need logic gates or data processing elements. To realize the computation of d_f in a combinational logic with data processing elements, 8 subtracters and 4 adders are required for computing all the values of d_{fi} and 7 comparators are needed for the comparison logic. Altogether, the computation of d_f requires 8 subtracters, 5 adders (one more for adding $2(k+1)$ to the maximum of all d_{fi}) and 7 comparators.

Operation of Hardware

The cellular array is first initialized with the image by loading *done* of background pixels' cells to 1 and object pixels' cells to 0. All other storage elements are loaded with 0. Then the cellular array and the external counter are allowed to run synchronously with the external clock. Each clock pulse corresponds to one iteration and all the cells are updated simultaneously at each clock pulse. The updation is stopped after d_{max} iterations. The final contents of d and $(\Delta x, \Delta y)$ of all cells give the transform values (EDT and NNT) of the given image.

The speed of operation of the cellular architecture depends primarily on the delay due to a cell. The delay due to interconnection between cells are negligible because the cells are locally connected. We have designed a cell in order to estimate the delay in terms of the maximum clock frequency using VHDL targeted to Xilinx FPGA.

4.5.1 Design of a Cell

The cell has registers for storing $|\Delta x|$, $|\Delta y|$ and d_f and a D flip-flop for keeping the flag *done*. We have not considered storing d and sign bits $S_{\Delta x}$ and $S_{\Delta y}$ since $(|\Delta x|, |\Delta y|)$

is sufficient to compute actual Euclidean distance and to solve the chosen problems, *viz*, computation of skeleton, Voronoi diagram and Hausdorff distance. The design of combinational logic is carried out hierarchically as follows. First, the computation of $d_{fi} = d_f(p_i) - (\Delta X_i + \Delta Y_i)$, i=1 to 8, in Equation (4.14) is performed. The computation of $\Delta X_i + \Delta Y_i$, i=2,4,6 & 8, requires four adders, one for each i. Two's complement addition is performed. The value of ΔX_i is either 0 or $2|\Delta x(p_i)| + 1$. Similarly, ΔY_i is. In binary representation, $2|\Delta x(p_i)| + 1$ is nothing but shifting $|\Delta x(p_i)|$ left by one bit and setting LSB to 1. For example, if $|\Delta x(p_i)|$="0101", then $2|\Delta x(p_i)| + 1$ is "1011". A logic circuit is not required to get the value of $2|\Delta x(p_i)| + 1$. A two's complement subtracter is used to subtract $\Delta X_i + \Delta Y_i$ from $d_f(p_i)$ for each i. The size of input and output ports of adders and subtracters is taken as the size of d_f. The construction of an N-bit adder/subtracter can be done with N full adders [Man96]. This adder-subtracter logic (ADD-SUB module) is shown in Figure 4.7.

The INC module computes Δx_i and Δy_i given by Equations (4.3) and (4.4). The computation requires twelve incrementers, 6 for implementing $\Delta x_i = |\Delta x(p_i)|+1$, i=2,3,4,6,7 & 8, and another six for implementing $\Delta y_i = |\Delta y(p_i)| + 1$, i=1,2,4,5,6 & 8. An N-bit incrementer can be constructed using N half-adders [Man96].

Once d_{fi}, Δx_i and Δy_i for i=1 to 8 have been computed, these values are given to a MAX module along with $done(p_i)$ and borrow bits (b_i) from the subtracters of the ADD-SUB module. The MAX module has seven CMP-MUX (comparator-multiplexer) modules arranged in three levels as shown in Figure 4.8 to compute $\max[d_{fi}]$, $i = 1$ to 8, in Equation (4.14). In the figure, $\max[d_{fi}]$ is denoted by d_{fM}. Further, the MAX module allows the values of Δx_i and Δy_i, corresponding to the i that yields d_{fM}. These values are denoted by Δx_M and Δy_M.

The CMP-MUX module takes two sets of inputs, $set_j = \{d_{fj}, \Delta x_j, \Delta y_j, done(p_j), b_j\}$ and $set_k = \{d_{fk}, \Delta x_k, \Delta y_k, done(p_k), b_k\}$. It has a comparator to compare $d_{fj} > d_{fk}$ of multiplexer. set_j is output when sel=0 while set_k is output when sel=1. The sel is generated using the output cmp of comparator and the values of $done$ and borrow bits. The comparator is designed for the simple case of comparing two unsigned binary numbers. $done(p_j)$=1 means the value of d_{fj} is valid. b_j=1 means d_{fj} is negative and cmp=1 means $d_{fj} > d_{fk}$. The value of sel for different cases is tabulated in Table 4.1. The components of the CMP-MUX module is shown in Figure 4.9

TH-2715_964102

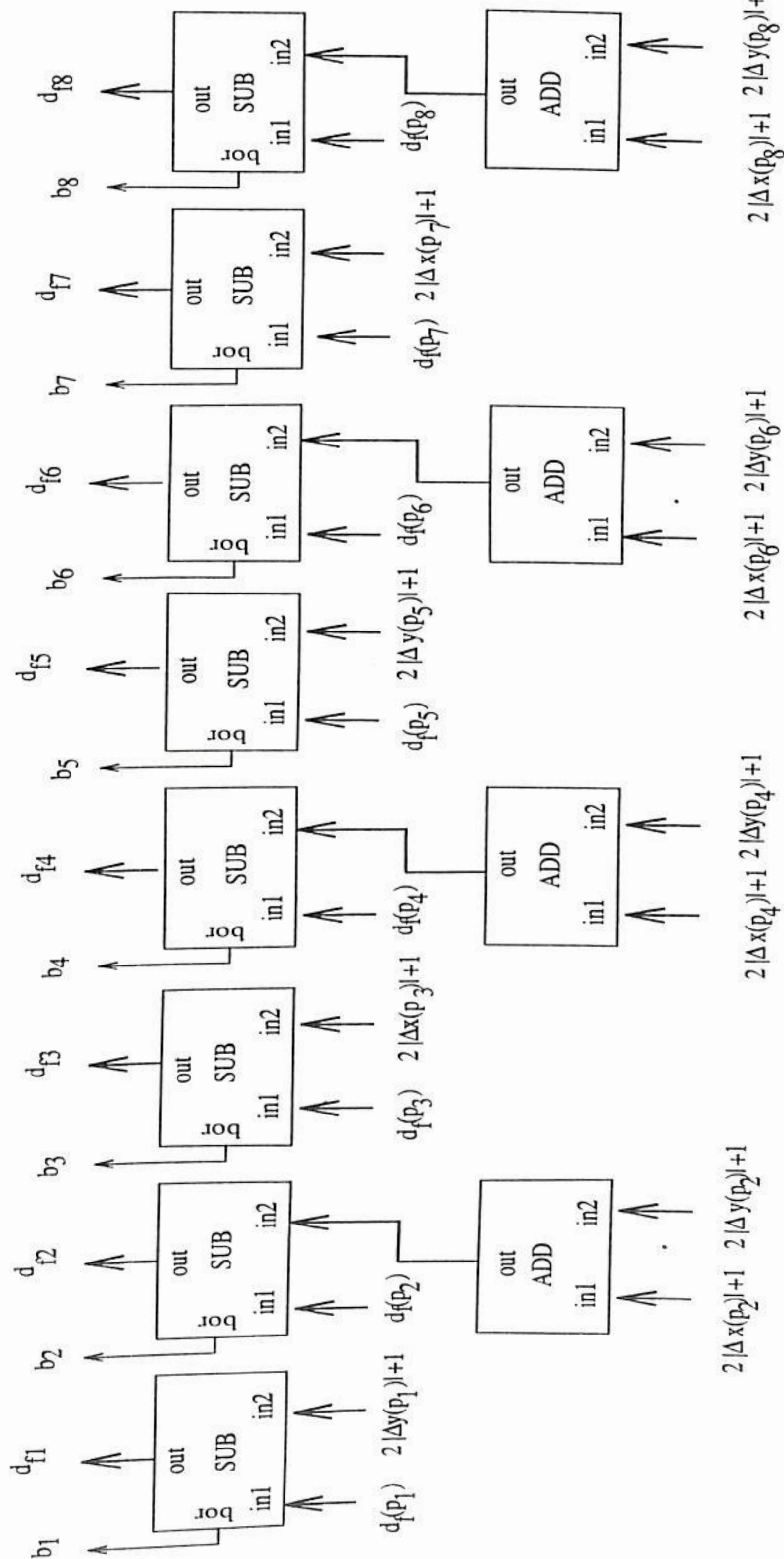


Figure 4.7: Adder-subtracter logic generating $d_{fi} = d_f(p_i) - (\Delta X_i + \Delta Y_i)$. The block ADD indicates an adder and SUB indicates a subtracter.

is sufficient to compute actual Euclidean distance and to solve the chosen problems, *viz*, computation of skeleton, Voronoi diagram and Hausdorff distance. The design of combinational logic is carried out hierarchically as follows. First, the computation of $d_{fi} = d_f(p_i) - (\Delta X_i + \Delta Y_i)$, i=1 to 8, in Equation (4.14) is performed. The computation of $\Delta X_i + \Delta Y_i$, i=2,4,6 & 8, requires four adders, one for each i. Two's complement addition is performed. The value of ΔX_i is either 0 or $2|\Delta x(p_i)| + 1$. Similarly, ΔY_i is. In binary representation, $2|\Delta x(p_i)| + 1$ is nothing but shifting $|\Delta x(p_i)|$ left by one bit and setting LSB to 1. For example, if $|\Delta x(p_i)|$="0101", then $2|\Delta x(p_i)| + 1$ is "1011". A logic circuit is not required to get the value of $2|\Delta x(p_i)| + 1$. A two's complement subtracter is used to subtract $\Delta X_i + \Delta Y_i$ from $d_f(p_i)$ for each i. The size of input and output ports of adders and subtracters is taken as the size of d_f. The construction of an N-bit adder/subtracter can be done with N full adders [Man96]. This adder-subtracter logic (ADD-SUB module) is shown in Figure 4.7.

The INC module computes Δx_i and Δy_i given by Equations (4.3) and (4.4). The computation requires twelve incrementers, 6 for implementing $\Delta x_i = |\Delta x(p_i)|+1$, i=2,3,4,6,7 & 8, and another six for implementing $\Delta y_i = |\Delta y(p_i)| + 1$, i=1,2,4,5,6 & 8. An N-bit incrementer can be constructed using N half-adders [Man96].

Once d_{fi}, Δx_i and Δy_i for i=1 to 8 have been computed, these values are given to a MAX module along with $done(p_i)$ and borrow bits (b_i) from the subtracters of the ADD-SUB module. The MAX module has seven CMP-MUX (comparator-multiplexer) modules arranged in three levels as shown in Figure 4.8 to compute max$[d_{fi}]$, i = 1 to 8, in Equation (4.14). In the figure, max$[d_{fi}]$ is denoted by d_{fM}. Further, the MAX module allows the values of Δx_i and Δy_i, corresponding to the i that yields d_{fM}. These values are denoted by Δx_M and Δy_M.

The CMP-MUX module takes two sets of inputs, $set_j = \{d_{fj}, \Delta x_j, \Delta y_j, done(p_j), b_j\}$ and $set_k = \{d_{fk}, \Delta x_k, \Delta y_k, done(p_k), b_k\}$. It has a comparator to compare $d_{fj} > d_{fk}$ of multiplexer. set_j is output when sel=0 while set_k is output when sel=1. The sel is generated using the output cmp of comparator and the values of $done$ and borrow bits. The comparator is designed for the simple case of comparing two unsigned binary numbers. $done(p_j)$=1 means the value of d_{fj} is valid. b_j=1 means d_{fj} is negative and cmp=1 means $d_{fj} > d_{fk}$. The value of sel for different cases is tabulated in Table 4.1. The components of the CMP-MUX module is shown in Figure 4.9

TH-2715_964102

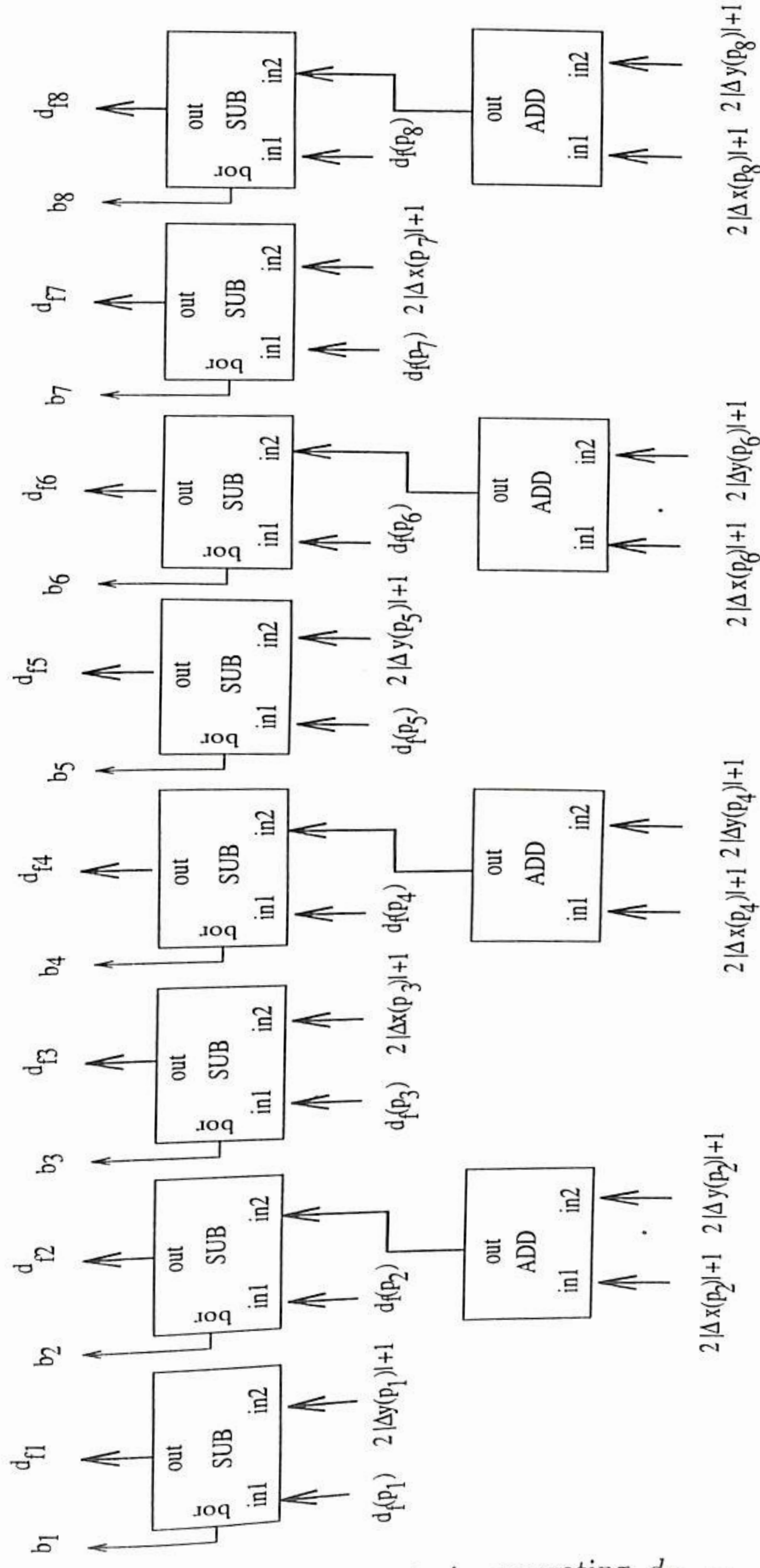


Figure 4.7: Adder-subtracter logic generating $d_{fi} = d_f(p_i) - (\Delta X_i + \Delta Y_i)$. The block ADD indicates an adder and SUB indicates a subtracter.

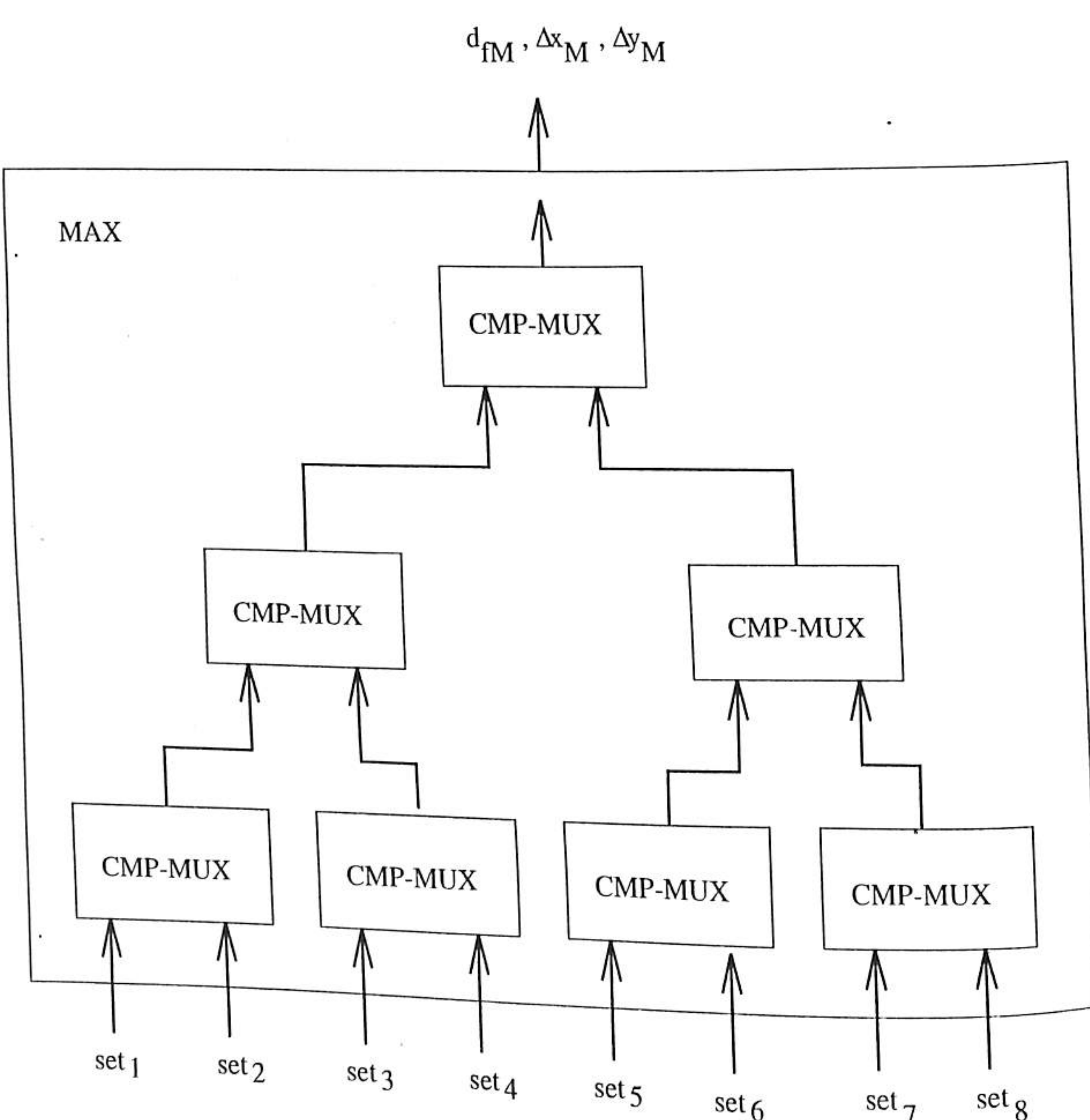


Figure 4.8: MAX module. set_i indicates the set of inputs $d_{fi}, \Delta x_i, \Delta y_i, done(p_i)$ and b_i.

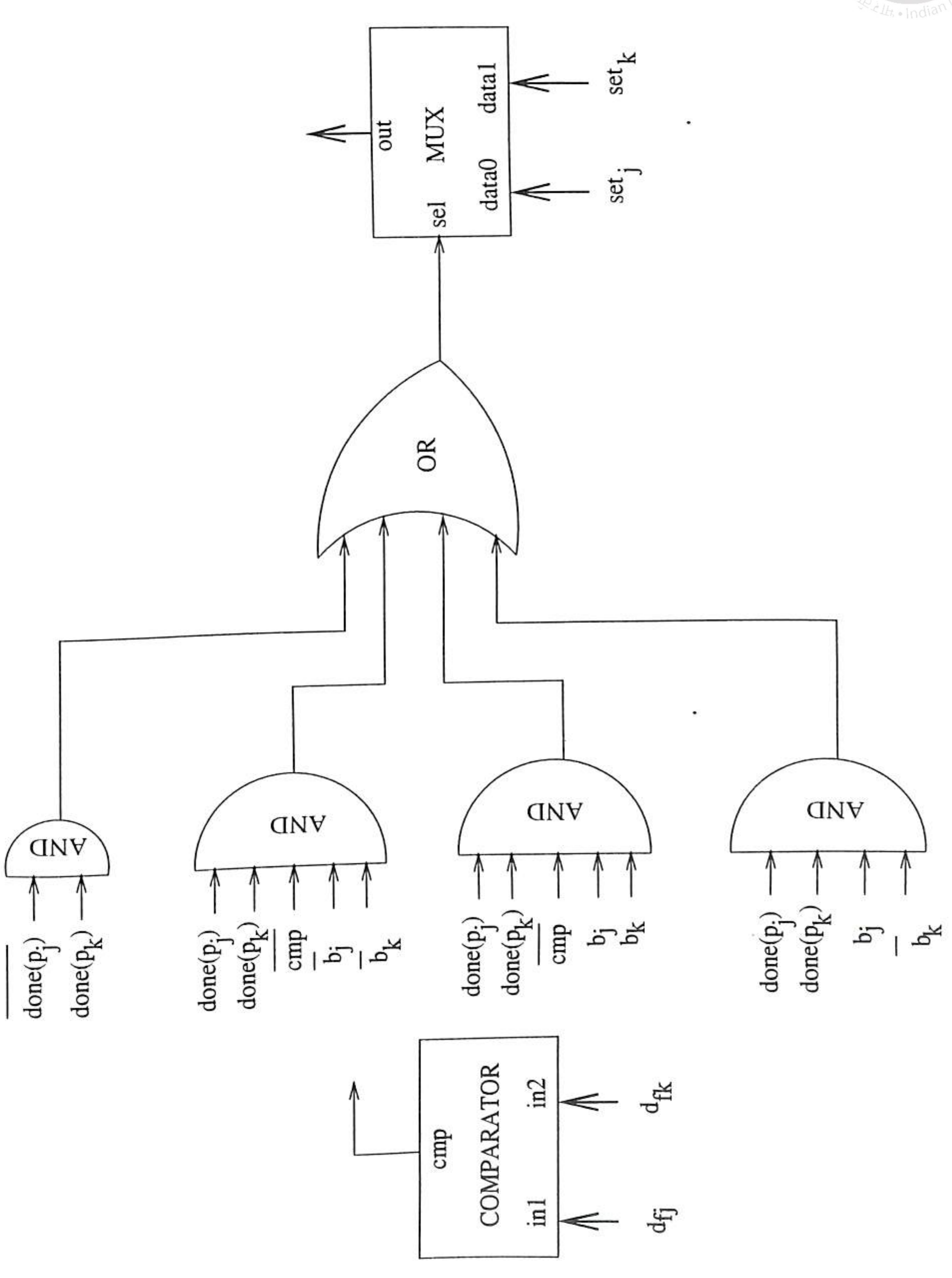


Figure 4.9: CMP-MUX module. The module has a comparator, a multiplexer and an AND-OR logic for generating *sel*. The set_j and set_k indicate the sets $\{d_{fj}, \Delta x_j, \Delta y_j, done(p_j), b_j\}$ and $\{d_{fk}, \Delta x_k, \Delta y_k, done(p_k), b_k\}$.

TH-2715_964102

Table 4.1: Value of *sel* for different cases of inputs to CMP-MUX module. "-" indicates "don't care".

$done(p_j)$	$done(p_k)$	b_j	b_k	*cmp*	*sel*	comments
0	0	-	-	-	-	Both d_{fj} & d_{fk} are invalid
1	0	-	-	-	0	d_{fj} is only valid
0	1	-	-	-	1	d_{fk} is only valid
1	1	0	0	1	0	Both are valid and positive. $d_{fj} > d_{fk}$
1	1	0	0	0	1	Both are valid and positive. $d_{fk} > d_{fj}$
1	1	1	1	1	0	Both are valid and negative. $d_{fj} > d_{fk}$
1	1	1	1	0	1	Both are valid and negative. $d_{fk} > d_{fj}$
1	1	0	1	-	0	d_{fj} is positive. d_{fk} is negative
1	1	1	0	-	1	d_{fj} is negative. d_{fk} is positive

The d_{fM}, output by the MAX module, is added to $2k$ where k is the iteration number generated by an external counter. Generating $2k$ does not need a logic circuit. The output $d_f(p_0)$ of the adder is given as input to the register d_f. The outputs, Δx_M and Δy_M, of the MAX module are given as inputs to the registers Δx and Δy. The *done* flip-flop is input with logic 1. In the design, the registers and flip-flop are loaded with the available inputs during the rising edge of the clock. From the **switch** and **if** statements of the algorithm in Section 4.3, it is clear that the clock is activated only when the following conditions are satisfied.

1. *done* of cell is not set.

2. At least a value of *done* of neighbours is logic 1.

3. $d_f(p_0)$ is positive. In two's complement representation of $d_f(p_0)$, this means that the MSB is 0.

In the case of register d_f, the clock is activated when the conditions 2 and 3 are satisfied. Let *cndn* be the signal whose value is 1 if the above three conditions are satisfied. *cndn* can be generated using a simple AND-OR logic circuit. The overall cell has the modules shown in Figure 4.10.

We call the cell designed above as *basic cell*. The implementation of the design of *basic cell* is given below.

TH-2715_964102

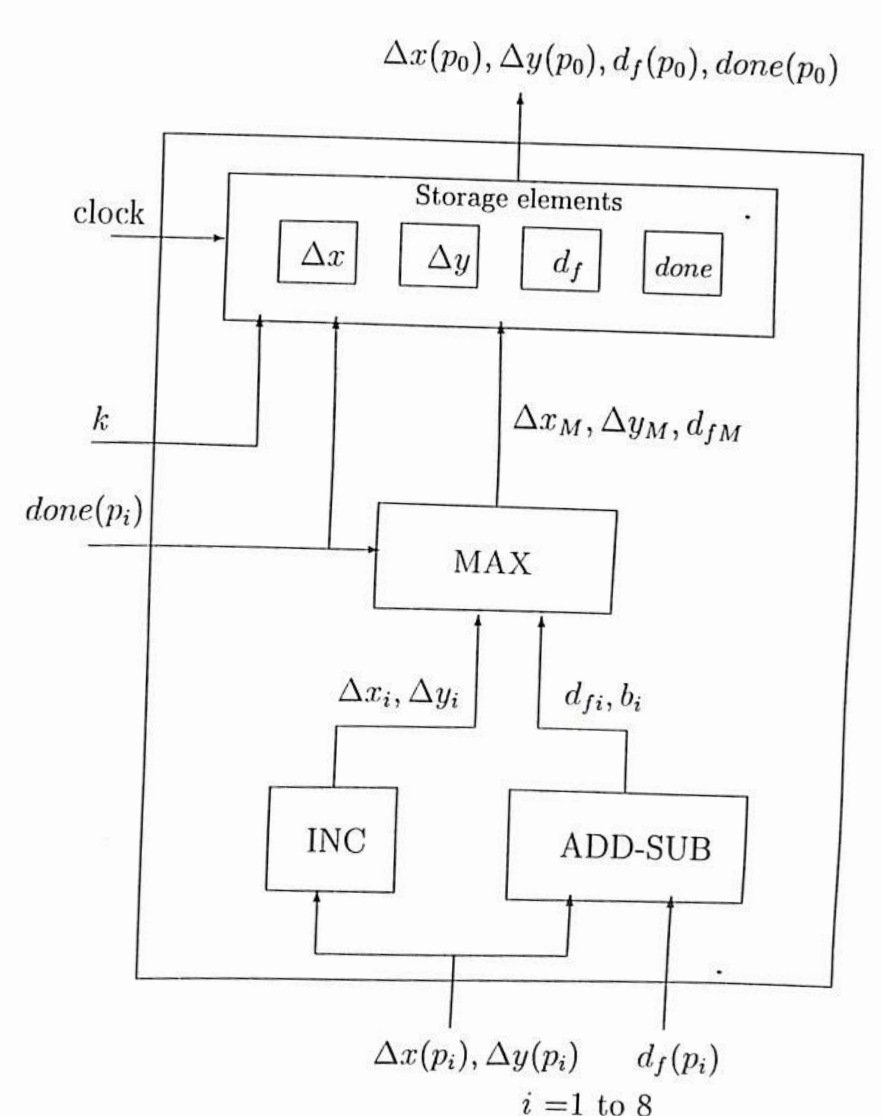


Figure 4.10: Different modules of a cell.

Implementation

The design of the basic cell has been coded in VHDL'93 (see Appendix A) and its functional behavior has been tested in ModelSim, a package for functional simulation of VLSI design. A set of input/output signal configurations, derived from the simulation examples presented in Section 4.4 of this chapter, has been used to test the design. No discrepancies between the outputs of the designed circuit and expected outputs have been detected. In Appendix B, some results of the functional simulation in ModelSim are provided. After the functional testing, implementation of the design has been carried out.

The design has been implemented in Xilinx FPGA (see Appendix C) for different sizes of the image. The design has been mapped onto one of the target devices of Xilinx. The appropriate device has been chosen taking into consideration the number of logic blocks and pins in the device. The specifications of the target device are as follows - family:XC4000; device:4036XL; package:BG432; speed grade:-2. Further details of the device are given in Appendix C. The chip obtained from the implementation has been optimized for speed. The area in terms of number of flip-flops and different Xilinx FPGA logic components of the chips obtained for different sizes of images are presented in Table 4.2.

The interpretation of results is as follows. The number of different flip-flops and buffers in the table are consistently increasing as image size increases. This is because the size of registers in the cell increases as image size increases (see Table 4.3). In the design of cell, the outputs of registers Δx and Δy are directly taken as the outputs of cell and hence OUTFF is used for these registers. DFFs are used for registers d_f and *done* since these outputs are fed back to the components of the cell. There are two derived clock signals in the design. One is given to register d_f and another to other registers. Hence two global buffers are needed. The number of input buffers increases as the size of the image increases. This is because the size of input lines for the quantities Δx, Δy and d_f of neighbors increases as image size increases. The number of output buffers is same as that of DFF because the outputs of the cell are the outputs of memory elements in the cell. FMAP increases as image size increases but HMAP does not. However, their total numbers are 498, 614, 696, 765 and 780 for different image sizes. This is because of the increasing complexity of combinational logic. The sizes of the components of combinational logic such as adders and comparators depend on the size of the image.

TH-2715_964102

Table 4.2: Number of different Xilinx FPGA logic components of chips obtained from the implementation of the design of basic cell for different sizes of images.

Image size $N \times N$	BUFG	DFF	FMAP	HMAP	IBUF	OBUF	OUTFF
16×16	2	9	424	74	142	9	8
32×32	2	10	516	98	167	10	10
64×64	2	11	616	80	192	11	12
128×128	2	12	644	121	217	12	14
256×256	2	13	658	122	242	13	16

BUFG: Global Buffer; DFF: D type flip-flop; FMAP,HMAP: Mapping design using F and H look-up tables; IBUF: Input Buffer; OBUF: Output Buffer; OUTFF: Output flip-flop

Table 4.3: Sizes of registers for different sizes of images.

Image size	Size in bits		
$N \times N$	Δx	Δy	d_f
16×16	4	4	8
32×32	5	5	9
64×64	6	6	10
128×128	7	7	11
256×256	8	8	12
512×512	9	9	13

Table 4.4: Results obtained after placement and routing of FPGA components listed in table 4.2 for different sizes of images.

Image size $N \times N$	Clock (MHz)	C path delay (ns)	Net delay (ns)	CLB	IOB	Gate count
16×16	6.67	165.41	12.43	246	159	2979
32×32	5.55	180.79	19.96	293	187	3711
64×64	5.23	213.22	22.81	340	215	4227
128×128	4.66	230.32	22.86	375	243	4671
256×256	4.66	243.26	24.82	384	271	4714

C path: Combinational path; CLB: Complex Logic Block; IOB: I/O Block

After mapping the design onto a Xilinx device, placement and routing of FPGA components have been carried out. The results obtained after placement and routing are presented in Table 4.4. Since there is a limit on the number of I/O blocks in the selected device, more than one device is needed for implementing the cell corresponding to images whose size exceed 256×256. As the size of image increases, the combinational logic blocks and I/O blocks of FPGA increases owing to increase in the size of registers and combinational logic. The clock frequency is neither consistently increasing nor decreasing. This is because only the bus width increases as the image size increases. The random change seen in the clock frequency and delays is expected since the placement and routing schemes implemented in Xilinx FPGA randomly picks a point to start and tries to get the best out of this starting point. The average clock frequency as seen from Table 4.4 is about 5 MHz. The layout of the chip for the design of basic cell corresponding to a 16×16 image is shown in Figure 4.11.

Once the clock frequency has been estimated, the overall cellular architecture of an $n \times n$ image in VHDL has been designed by replicating n^2 cells and interconnecting them. The other components of the architecture are a clock and a control logic having a counter to count upto $\lceil \sqrt{2}n \rceil$. The functional behavior of the VHDL design of the architecture has been tested in ModelSim by giving images of size $n \times n$ as input, allowing the design to run for $\lceil \sqrt{2}n \rceil$ clock pulses and then verifying the outputs of the registers Δx and Δy of cells.

Since a cellular architecture consists of locally connected identical cells, the delay due to interconnections is negligible. Hence, the time taken (T_{EDT}) to compute the EDT of an image of size $n \times n$ can be estimated from the clock rate (f_c) obtained from the

TH-2715_964102

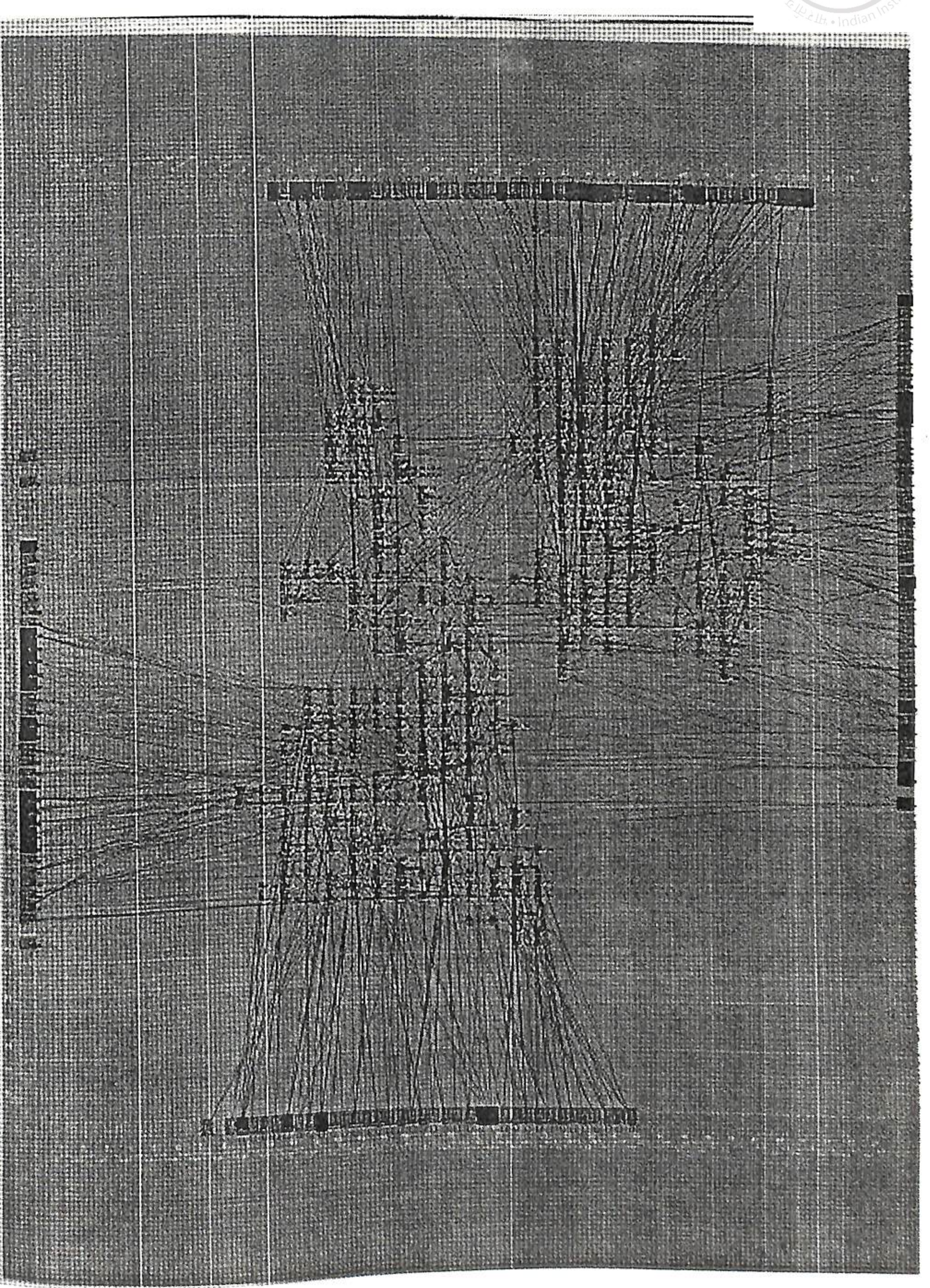


Figure 4.11: Layout of the chip obtained from the implementation of the design of basic cell corresponding to a 16×16 image.

implementation of a cell. The number of clock pulses needed to compute EDT equals $\lceil\sqrt{2}n\rceil$. The period (T_c) of a clock pulse is $1/f_c$ seconds. Hence, $T_{EDT} = \lceil\sqrt{2}n\rceil * T_c$. The number of images ($N_I$) per second whose EDT can be computed on a cellular architecture is greater than or equal to $\lceil 1/T_{EDT}\rceil$.

Some results and comments for an image size 256×256 are given now. $\lceil\sqrt{2}n\rceil$ for this case is 361. The frequency of operation of cell corresponding to this image is 4.66 MHz. T_{EDT} is approximately $77 * 10^{-6}$ seconds and N_I is approximately 12987. N_I is much greater than the video rate, which is about 30 frames per second and hence the computation of EDT on a cellular architecture is quite suitable for real-time applications.

The implementation results (speed of operation and gate count) are determined by the characteristics of the FPGA device chosen. Further improvements can be expected by an Application Specific Integrated Circuit (ASIC) implementation of the architecture presented.

4.6 Applications of EDT

EDT has potential applications in machine vision and pattern recognition. It is a useful tool for finding skeleton of objects, constructing Voronoi diagram and computing Hausdorff distance between images. In the chapters to follow, we consider these applications in detail and develop algorithms for VLSI implementation.

The NNT is useful for nearest neighbor pattern classification. If each pattern is represented as a point in a 2-D integer space Z^2, then this space can be treated as an image and a point can be mapped to a pixel. A given set of patterns with known classes can be represented as background pixels of an image. The NNT of this image is used to identify the class of a new pattern, which is nothing but the class of the nearest neighbor of the pixel corresponding to the new pattern.

Other applications of NNT include detection of dominant points in digital curves, curve smoothing and computing convex hulls [Ye88].

TH-2715_964102

4.7 Conclusions

The Euclidean distance transform and nearest neighbor transform are important tools in image processing. In this chapter, we have proposed a parallel algorithm for their computation for a binary image. The algorithm computes the integer Euclidean distance and the vector Euclidean distance. The exact Euclidean distance value and the nearest neighbor of each pixel can be obtained from the vector Euclidean distance. The algorithm takes $O(n)$ time $O(n^2)$ space. Due to simple, identical and local neighborhood operations, the proposed method of computation is suitable for VLSI implementation in a cellular architecture. Such an architecture is presented. The implementation of a cell of the architecture has been carried out in Xilinx FPGA. From the implementation results, it is observed that the architecture can process an image much faster than the video rate. The architecture is hence suitable for real-time applications.

In the next three chapters, the applications of EDT are given. The next chapter gives the computation of skeleton of binary images.

Chapter 5

Computation of the Skeleton

5.1 Introduction

In the previous chapter, we have proposed an iterative procedure to compute EDT of a binary image and designed a cellular architecture for the computation. One of the problems that can be solved using distance transform is the computation of skeleton of a binary image. As defined in in Section 2.2.1 of Chapter 2, the skeleton consists of points in an object that are closest to more than one boundary point. It is a linear structure representing the shape of an object. The skeleton based on Euclidean distance, *i.e.*, Euclidean skeleton, is invariant to the orientation of the object. In a discrete image, extraction of a connected skeleton is a non-trivial problem. Different methods of extraction given by various researchers have been discussed in Chapter 3 but a VLSI architecture for extracting the Euclidean skeleton is not available (to the best of our knowledge).

In this chapter, we propose a method to find the Euclidean skeleton of a binary image. The method finds the skeleton pixels while iteratively computing the Euclidean distance values. For an image of polygonal objects, the method gives an exact, connected skeleton which is insensitive to discretization of the boundary. A parallel algorithm is designed for the proposed method, which processes all the pixels in parallel and computes EDT and skeleton of an image simultaneously. The algorithm is amenable to VLSI implementation in a cellular architecture. The time and space complexities are same as the computation

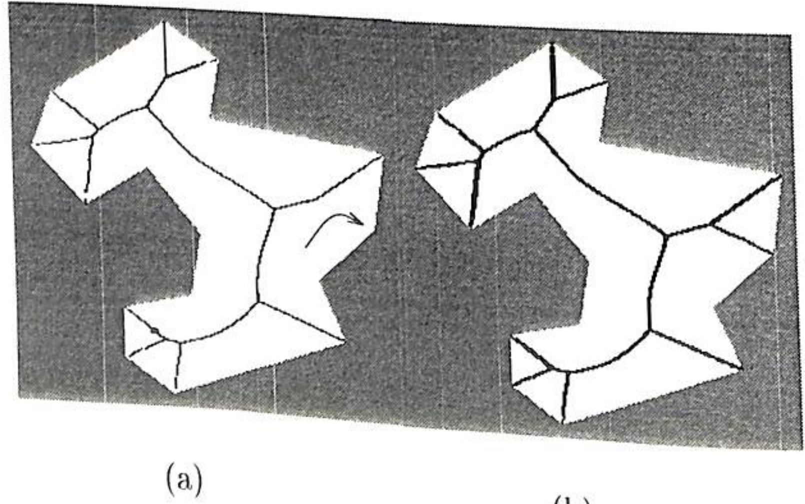
TH-2715_964102

of EDT. The time complexity is $O(n)$ and space complexity is $O(n^2)$. A preliminary version of this work is reported in [SN98].

The background for the proposed method is given in the next section.

5.2 Background

Consider a binary image consisting of objects O (0-pixels) and background B (1-pixels). The Euclidean distance transform can be computed by the algorithm given in Section 4.3. The objects are treated as foreground of the image. When visualizing the distance value as the altitude on a surface, the ridges of this distance surface, together with the corresponding distances, constitute the skeleton. The authors of [SP90] propose a method to find the ridges from the EDT of the image. They consider a 3×3 neighborhood for each pixel of the transformed image where the pixel being the center of the neighborhood. The ridges are those pixels which satisfy one of the following two conditions: (1) their distance values are greater than or equal to the distance values of their eight neighbors; (2) their distance values are greater than their neighboring two pixels in horizontal, vertical or diagonal direction. From experiments, we found that, this method gives a skeleton with missing branches as shown in Figure 5.1. This is due to the limitation in the number of directions to four. This leads us to propose a method



(a) (b)

Figure 5.1: (a) Skeleton computed using a 3×3 neighborhood. (b) Actual skeleton. Arrow points to the corner where a branch is missing.

TH-2715_964102

to extract the skeleton (based on Euclidean distance) with a larger neighborhood and includes more directions as shown in Figure 5.2. The method finds the skeleton directly from the image and simultaneously computes the EDT of the image. Also it can be efficiently implemented in hardware.

5.3 Proposed Method: Extraction of Skeleton by Detecting Ridge Points

In the proposed method for computing skeleton, a larger neighborhood, N_S (consisting of seventeen pixels in a 5×5 window as shown in Figure 5.2) is considered for each pixel. There are totally eight directions formed by each pair of neighbors lying along

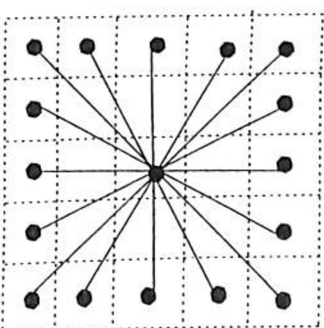


Figure 5.2: Neighborhood (N_S) of a pixel for skeleton computation

a line connecting the middle pixel. The number of directions is twice that of a 3×3 neighborhood. This increase in the number of directions helps in extracting the exact skeleton. The skeleton is computed by comparing the Euclidean distance value of each pixel with the distance values of its neighbors. It consists of all those pixels with distance values greater than their neighboring two pixels in any one of the eight directions. In the proposed method, as soon as the distance value of a pixel is computed, we determine whether it is a skeleton pixel or not. The derivations for the computation of EDT have been already given in Section 4.2. Out of which, the computation of $(|\Delta x|, |\Delta y|)$, d_f and d are useful for computing skeleton. The iterative equations for their computation are reproduced below.

$$|\Delta x^{(0)}(p_0)| = |\Delta y^{(0)}(p_0)| = D^{(0)}(p_0) = d^{(0)}(p_0) = \begin{cases} 0 & p_0 \in B \\ \infty & p_0 \in O \end{cases} \tag{5.1}$$

$$d_f^{(0)}(p_0) = \begin{cases} 0 & p_0 \in B \\ -\infty & p_0 \in O \end{cases} \tag{5.2}$$

$$d_f^{(k+1)}(p_0) = 2(k+1) + \max_{p_i \in N_E(p_0)} [d_{fi}] \quad \text{where } d_{fi} = d_f^{(k)}(p_i) - (\Delta X_i + \Delta Y_i) \tag{5.3}$$

where $N_E(p_0)$ is the neighborhood in Figure 4.2 and

$$\Delta X_i = \begin{cases} 0 & \text{for } i = 0, 1\&5 \\ 2|\Delta x^{(k)}(p_i)| + 1 & \text{for } i = 2,3,4,6,7\&8 \end{cases} \tag{5.4}$$

$$\Delta Y_i = \begin{cases} 0 & \text{for } i = 0, 3\&7 \\ 2|\Delta y^{(k)}(p_i)| + 1 & \text{for } i = 1,2,4,5,6\&8 \end{cases} \tag{5.5}$$

$$(|\Delta x^{(k+1)}(p_0)|, |\Delta y^{(k+1)}(p_0)|) = (\Delta x_i, \Delta y_i) \tag{5.6}$$

where

$$\Delta x_i = \begin{cases} |\Delta x^{(k)}(p_i)| & \text{for } i = 0, 1\&5 \\ |\Delta x^{(k)}(p_i) + 1| & \text{for } i = 2,3,4,6,7\&8 \end{cases} \tag{5.7}$$

$$\Delta y_i = \begin{cases} |\Delta y^{(k)}(p_i)| & \text{for } i = 0, 3\&7 \\ |\Delta y^{(k)}(p_i) + 1| & \text{for } i = 1,2,4,5,6\&8 \end{cases} \tag{5.8}$$

and i corresponds to pixel p_i that satisfies Equation (5.3).

TH-2715_964102

$$d^{(k+1)}(p_0) = \min[d^{(k)}(p_0), k+1 : d_f^{(k+1)}(p_0) \geq 0] \tag{5.9}$$

The computation of skeleton value of a pixel is done as follows. Let $s(p_0)$ indicate whether the pixel p_0 belongs to skeleton or not. $s(p_0)$ equals 1 if p_0 is in skeleton and it is 0 otherwise. $s(p_0)$ is initialized to zero. Whenever the distance value of a pixel is found at some iteration, the skeleton value of that pixel will be found in the next iteration. We propose two methods for finding the skeleton value of a pixel using the integer distance d (Equation (5.9)) or actual Euclidean distance d_e $(=\sqrt{|\Delta x|^2 + |\Delta y|^2})$ of neighbors in the neighborhood, N_S, given in Figure 5.2. Method 1 uses d values and Method 2 uses d_e values. Let p_0 be the middle pixel in N_S and (p', p'') represent a pixel pair along a direction. There are totally eight such pairs. Let P be the set containing all the eight pairs. If the distance vector of p_0 has been found at iteration k, *i.e.*, $d(p_0) = k$, then at iteration $k+1$, $s(p_0)$ is assigned either 0 or 1 based on the following methods.

<u>Method 1:</u> Skeleton based on d values

Method 1 assigns one to $s(p_0)$ if there exists a pixel pair $(p', p'') \in P$ such that $d(p_0)$ is greater than $d(p')$ and $d(p'')$.

$$s^{(k+1)}(p_0) = \begin{cases} 1 & d^{(k)}(p_0) = k \text{ and } \exists (p', p'') \in P[d^{(k)}(p_0) > d^{(k)}(p'), d^{(k)}(p_0) > d^{(k)}(p'')] \\ 0 & \text{otherwise} \end{cases} \tag{5.10}$$

That is,

$$s^{(k+1)}(p_0) = \begin{cases} 1 & d^{(k)}(p_0) = k \text{ and } \exists (p', p'') \in P[d^{(k)}(p') < k, d^{(k)}(p'') < k] \\ 0 & \text{otherwise} \end{cases} \tag{5.11}$$

<u>Method 2:</u> Skeleton based on d_e values

The computation of $s(p_0)$ by Method 2 is similar to Method 1, but instead of d, d_e

is used.

$$s^{(k+1)}(p_0) = \begin{cases} 1 & d^{(k)}(p_0) = k \text{ and } \exists(p', p'') \in P[d_e{}^{(k)}(p_0) > d_e{}^{(k)}(p'), d_e{}^{(k)}(p_0) > d_e{}^{(k)}(p'')] \\ 0 & \text{otherwise} \end{cases} \tag{5.12}$$

The above can be rewritten in terms of d_f as follows:

$$s^{(k+1)}(p_0) = \begin{cases} 1 & d^{(k)}(p_0) = k \text{ and } \exists(p', p'') \in P[d_f{}^{(k)}(p_0) < d_f{}^{(k)}(p'), d_f{}^{(k)}(p_0) < d_f{}^{(k)}(p'')] \\ 0 & \text{otherwise} \end{cases} \tag{5.13}$$

In the computation of $s(p_0)$, note that the actual values of d and d_f of some of the neighbors of p_0 may not be computed in previous iterations. Even if it is so, comparing with the present values (∞ and $-\infty$) is still valid because the actual d and d_e values of these neighbors are greater than that of p_0 and the actual d_f values of the neighbors are less than that of p_0.

The iterative procedures for the computation of distance vector and skeleton described above can be carried out simultaneously. At each iteration, their computation depends only on the values computed at previous iterations. This leads us to design a parallel algorithm in which the computation for all pixels is carried out in parallel. The algorithm is based on the Equations (5.3), (5.6),(5.11) and (5.13). Before giving the algorithm, we provide some analysis of the neighborhood window and the effect of discretization in computing skeleton.

5.3.1 Analysis of Neighborhood Window

The method described for the computation of skeleton uses the pixels along the border of a 5×5 neighborhood window. Let us consider a general $m \times m$ window, where m is a positive odd integer. The pixels along the border of the window are neighbors for the center pixel. The number of these neighboring pixels is $4(m-1)$, which is illustrated in Figure 5.3. There are totally $2(m-1)$ directions formed by each pair of neighbors lying along a line connecting the middle pixel. If the pixels are numbered as shown in Figure 5.4, then the angle between successive directions made by pixel numbered i and $i-1$ is given by $\theta_i = \theta(i) - \theta(i-1)$. That is,

$$\theta_i = \tan^{-1}[i/m'] - \tan^{-1}[(i-1)/m']$$

Indian Institute of Technology Guwahati

m
m-2
m-2
m
Number of border pixels
=2m +2 (m-2)
= 4 (m-1)

Figure 5.3: Neighboring pixels in an $m \times m$ window.

where $m' = (m-1)/2$. The value of θ_i decreases as i increases. Assume that the function $\tan^{-1}[.]$ gives the output in degrees. The minimum angle is $\theta_{m'}$, where $\theta_{m'} = 45° - \tan^{-1}[(m'-1)/m']$ and maximum angle is θ_1, where $\theta_1 = \tan^{-1}[1/m']$. The computation of skeleton by comparing the distance values described in the previous section may lead to missing branches at the corners of an object whose interior angle is greater than or equal to $180° - \theta_1$. The possibility of missing branch at a corner with interior angle $180° - \theta_1$ is shown in Figure 5.5 for $m = 5$. The pixel p_0 is a corner pixel of an object and the pixels p_i, i=1 to 16, are neighbors of p_0. The pixels p_j, j=0,1,10, have $d_e(p_j) = 1$. The pixels p_k, k=11 to 16, have $d_e(p_k) > d_e(p_0)$ and the pixels p_l, l=2 to 9, have $d_e(p_l) = 0$. So, no pair of pixels along any direction have their distance values less than that of p_0 and hence p_0 does not belong to skeleton by the proposed method. Hence, an $m \times m$ window is used to compute skeleton of an object whose interior angles at the corners are less than $\theta_b = 180° - \theta_1$.

As m increases, θ_1 decreases and hence θ_b increases. The following are some comments on increasing m. (1) As m increases, the thickness of skeleton also increases and it is no longer single pixel width. In application like skeleton based compression, storing thicker skeleton requires more space. (2) The connectivity of the skeleton is guaranteed for a thicker skeleton. (3) The corner of an object with an interior angle close to 180° will not be sharp in a discrete image. The branch of skeleton arising from this corner need not be detected using a larger window which gives θ_b close to 180°. (4) The computational

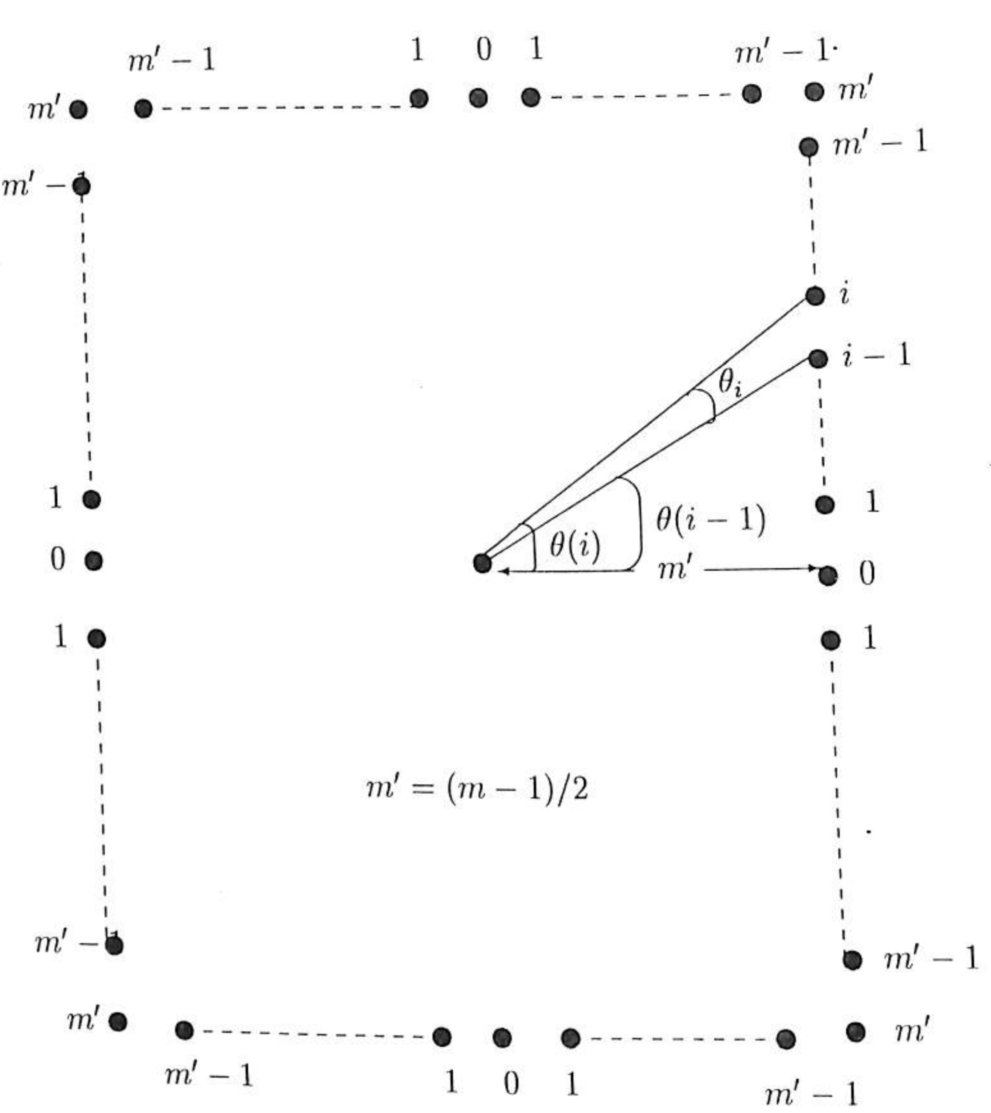


Figure 5.4: Angle between successive directions in an $m \times m$ window.

complexity increases as m increases.

Table 5.1 gives θ_b and shows the width of a branch of skeleton arising from a corner of a polygonal object for different sizes of neighborhood window. The skeleton for a 3×3 window is single pixel in width but the angle θ_b is very small. For a 5×5 window, this angle is 18.43° more and the thickness of skeleton is only slightly greater than that of the 3×3 window. In the case of 7×7 window, θ_b is only 8.13° more and the thickness is greater than that of the 5×5 window. From experimental studies, it is observed that a 5×5 window size gives best results for different images. This is due to following reasons.

TH-2715_964102

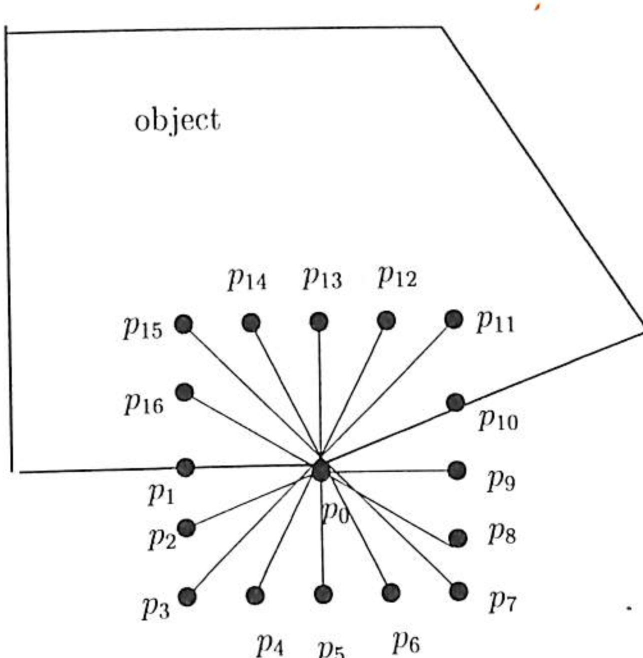


Figure 5.5: Possibility of branch missing at the corner pixel p_0.

1. θ_b is considerably large to detect almost all the branches arising from corners.
2. Though the skeleton is slightly thicker than that of a 3×3 window, 5 × 5 window leads to a better connected skeleton.
3. The complexity of computation is reasonable.

5.3.2 Effect of Discretization

The skeleton computed by the above method is insensitive to the discretization noise in the boundary of polygonal objects. In a discrete image, the boundary of an object is described by the object pixels whose any one of the eight neighboring pixels at the chessboard distance of 1 is a background pixel. Hence, the edge of a polygonal object in a discrete image is no longer straight except for the cases when the edges are horizontal or vertical. We refer to this as discretization noise. For example, see Figure 5.6(a). Due to this noise, the discrete boundary appears to have more horizontal and vertical edges and hence more vertices. One would like to compute a skeleton which is insensitive to these spurious vertices. In the figure, we can notice that no two neighbors along

Table 5.1: Angle θ_b and thickness of skeleton arising from the corner of a polygonal object for different sizes of neighborhood window

Neighborhood Window	θ_b in degrees	Skeleton Width
3×3	$135°$	
5×5	$153.43°$	
7×7	$161.56°$	

any direction in the neighborhoods of the spurious vertex pixels belong to background. Hence, by the proposed method, the spurious vertices do not belong to skeleton and no spurious skeleton branch occurs in the case of polygonal objects.

In the case of non-polygonal objects, spurious skeleton pixels may occur. For example, consider a circular disc shown in Figure 5.6 (b). The discrete boundary of the disc consists of horizontal and vertical edges which is shown in the figure. Consider the pixel marked 'X'. By the proposed method, this pixel belongs to skeleton because at this pixel, there exists two neighbors along a direction in its neighborhood which belong to background. The actual skeleton of a disc is its center. But the proposed method leads to spurious skeleton pixels for a discretized disc. Hence, the method is sensitive to discretization noise in the boundary of non-polygonal objects. The following section gives a step by step procedure to compute skeleton.

5.4 Algorithm

In this section, a pseudocode of the algorithm for finding skeleton pixels and computing EDT of a given binary image is presented. The algorithm can be summarized as follows. Given a binary image of size $n \times n$ containing object (O) and background (B) pixels, the

TH-2715_964102

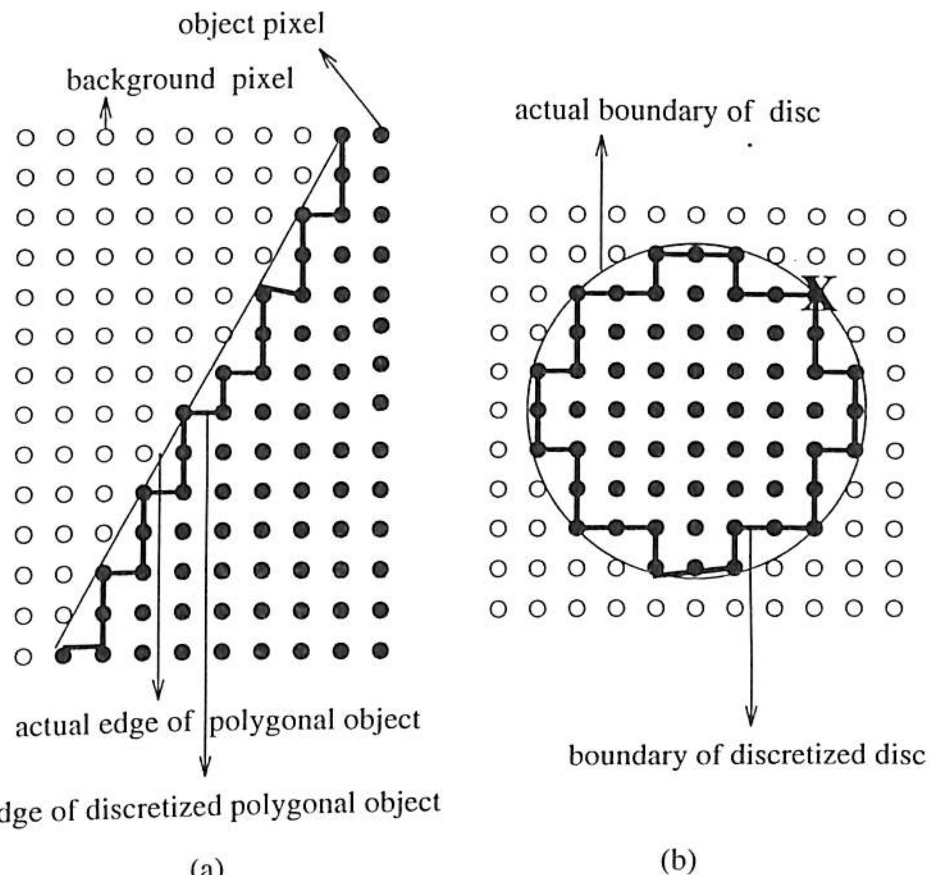


Figure 5.6: Effect of discretization. (a) Discretization noise at the edge of polygonal object. (b) Discretization noise at the boundary of circular disc. The pixel marked 'X' belongs to skeleton by the proposed method.

skeleton value s and the Euclidean distance vector $(|\Delta x|, |\Delta y|)$ of background pixels are set to zero. The corresponding values of the object pixels are computed iteratively from the previously computed $|\Delta x|, |\Delta y|$ and d_f values of the neighboring pixels. At each iteration, the algorithm computes $(|\Delta x|, |\Delta y|)$ of those pixels whose d values equal the iteration number and the s values of those pixels whose $(|\Delta x|, |\Delta y|)$ have been computed in the previous iteration.

<u>ALGORITHM:</u> *SKEL*

<u>Inputs:</u> The input to the algorithm is the given binary image consisting of object and background pixels.

Outputs: The outputs are the skeleton value $s(p_0)$ and Euclidean distance vector $(\Delta x(p_0), \Delta y(p_0))$ of each pixel p_0 in the given image.

Step 1: Initialization

Similar to Algorithm *EDT_NNT* in Section 4.3, $|\Delta x|$, $|\Delta y|$ and d_f are initialized and a flag *done* is assigned to each pixel in the image. s is initialized to 0. The value of *done* is set to 1 when s and $(|\Delta x|, |\Delta y|)$ of a pixel are computed at any iteration. The initialization is carried out as follows.

$$|\Delta x^{(0)}(p_0)| = |\Delta y^{(0)}(p_0)| = d_f^{(0)}(p_0) = s^{(0)}(p_0) = 0$$

$$done(p_0) = \begin{cases} 1 & p_0 \in B \\ 0 & p_0 \in O \end{cases}$$

Step 2: Iterative process

At each iteration $k > 0$, the Euclidean distance vectors $(|\Delta x|, |\Delta y|)$ of pixels whose d values equal k are computed. The skeleton values s of those pixels, whose $(|\Delta x|, |\Delta y|)$ values have been recently computed are found out. These pixels are kept track of using a flag rc for each pixel p_0 of the image, instead of checking $d(p_0) = k$. rc is initialized to zero. Whenever a pixel receives its $(|\Delta x|, |\Delta y|)$ value, its rc value is set to 1. In the next iteration, the skeleton value s of this pixel is computed, its rc is reset to 0 and *done* is set to 1. The pseudocode of the iterative process is as follows.

```
k = 0
while there exists pixel p such that done(p) = 0
  for all pixels p0 do in parallel

    switch (done(p0))
      1: {Both (|Δx|, |Δy|) and s have been computed}
        Update d_f(p0) by adding 2(k + 1)
        break {switch done = 1}
```



```
      0: switch (rc^(k)(p0))
        0: {Both (|Δx|, |Δy|) and s have not been computed yet.
          Do Euclidean distance vector computation}
          if there exists pi ∈ N_E(p0) such that (done(pi) = 1) ∨ (rc(pi) = 1) do
            Compute d_f^(k+1)(p0) as in Equation (5.3)
            if (d_f^(k+1)(p0) ≥ 0) do
              Compute (|Δx^(k+1)(p0)|, |Δy^(k+1)(p0)|) as in Equation (5.6)
              rc^(k+1)(p0) = 1
            end if
          end if
          break {switch rc = 0}

        1: {(|Δx|, |Δy|) has been computed in previous iteration.
          Do Skeleton computation}
          ComputeSkeleton()
          rc^(k+1)(p0) = 0
          done(p0) = 1
          Update d_f^(k+1)(p0) as in Equation (5.3)
          break {switch rc = 1}
        break {switch done = 0}

  end for
  k = k + 1
end while
```

The function *ComputeSkeleton()* for computing skeleton pixels by different methods is as follows.

Method 1 [Equation (5.11)]: *ComputeSkeleton()*

In Equation (5.11), at $(k+1)$th iteration, if d values of p' and p'' are less than k, then their $(|\Delta x|, |\Delta y|)$ and s must have been computed already. Hence, $s(p_0)$ can be computed by testing $done(p')$ and $done(p'')$ as follows.

if $(\exists(p', p'')[(done(p') = 1) \wedge (done(p'') = 1)])$
$s(p_0) = 1$

Method 2 [Equation (5.13)]: *ComputeSkeleton()*

if $(\exists(p', p'')[(d_f{}^{(k)}(p') > d_f{}^{(k)}(p_0)) \wedge (d_f{}^{(k)}(p'') > d_f{}^{(k)}(p_0))])$
$s(p_0) = 1$

The above procedure runs until the *done* flags of all the pixels are set to 1. In each iteration, all the pixels are processed in parallel. At any iteration, one of the following is carried out in each pixel: Euclidean distance computation, skeleton computation and no computation. The choice depends on rc and *done* flags of the pixel. If *done* is 1, then both $(|\Delta x|, |\Delta y|)$ and s have been computed already. Only the value of d_f is updated since it depends on k, the iteration number. If *done* is 0 and rc is also 0, both $(|\Delta x|, |\Delta y|)$ and s have not been computed yet. Hence, we perform the Euclidean distance computation. First, d_f is computed if there exists some neighbor whose $(|\Delta x|, |\Delta y|)$ has been computed already. If $d_f \geq 0$, then $(|\Delta x|, |\Delta y|)$ of the pixel is computed and rc is set to 1. In the case when $rc = 1$, computation of s is carried out, *done* is set to 1 and rc is reset to 0. Also d_f is updated. The time and space complexities of the algorithm are as follows.

5.4.1 Complexity analysis

Similar to the Algorithm *EDT_NNT*, the *time complexity* of the proposed algorithm for computing skeleton is determined by the **while** loop. The **for** loop runs in parallel and hence takes constant time. The **while** loop runs until the skeleton values of all the pixels have been computed. This means the total number of iterations equals the number of iterations for computing EDT plus one. The number of iterations for computing EDT depends on the maximum distance value that a pixel takes in the given image and hence the time complexity is $O(n)$ for an $n \times n$ image. The *space complexity* of the algorithm is $O(n^2)$, since the algorithm needs constant space to store the values $|\Delta x|, |\Delta y|, d$ and d_f and the flags *done*, rc and s for each pixel.

TH-2715_964102

The identical local operations performed in a local neighborhood of each pixel make the algorithm feasible for VLSI implementation in a cellular architecture. Before describing VLSI implementation, we provide some simulation results of the algorithm.

5.5 Simulation results and comparisons

The algorithm has been simulated on a computer to test its performance. It has been implemented in ANSI-C on an HP 9000 Series 700 J Model J200 workstation. The skeletons computed by the algorithm for different objects are shown in Figures 5.7,5.8 and 5.9. The reconstructed objects from the computed skeleton and distance values are also shown in the figures. The distance values are given by $\sqrt{\Delta x^2 + \Delta y^2}$. The reconstructed object is given by those pixels within the circular discs whose centers are skeleton pixels and radii are the distance values at the skeleton pixels. Figure 5.7 and Figure 5.8 show the skeletons of different types of polygonal objects [O'R94] computed by Method 1 and Method 2. These skeletons match with the expected actual ones. The skeletons computed by Method 2 have better connectivity than the ones computed by Method 1. For instance, see Figure 5.7(c) and Figure 5.8(c). But, in both methods, the skeletons give exact reconstruction of objects. The rotational invariance property of skeletons based on Euclidean distance can be seen in Figures 5.7 (a) and (b). Figure 5.9 shows the skeletons of the non-polygonal objects. They are different from their expected actual ones. For instance, the actual skeleton of an elliptical object is a medial line segment, while the computed one has extra branches at the ends of the medial line segment. The reconstruction of non-polygonal objects (Figures 5.9 (d)-(f),(j)-(l)) is exact and it is unaffected by the extra branches.

We compare the skeleton, based on Euclidean distance, computed by the proposed algorithm with the skeletons based on other distances that are commonly used in image processing. A test image, taken from [KK87] and shown in Figure 5.10(a), has been considered. The object is a polygon and the size of the image is 240×240. We have generated skeletons from different distance transforms. Since our algorithm for computing skeleton is based on the neighborhood N_S, shown in Figure 5.2, we generate all the skeletons using the same neighborhood. Figure 5.10(b) shows the skeleton obtained from the Euclidean distance transform computed by the brute force method. The brute force method computes the Euclidean distance value of each pixel by finding the min-

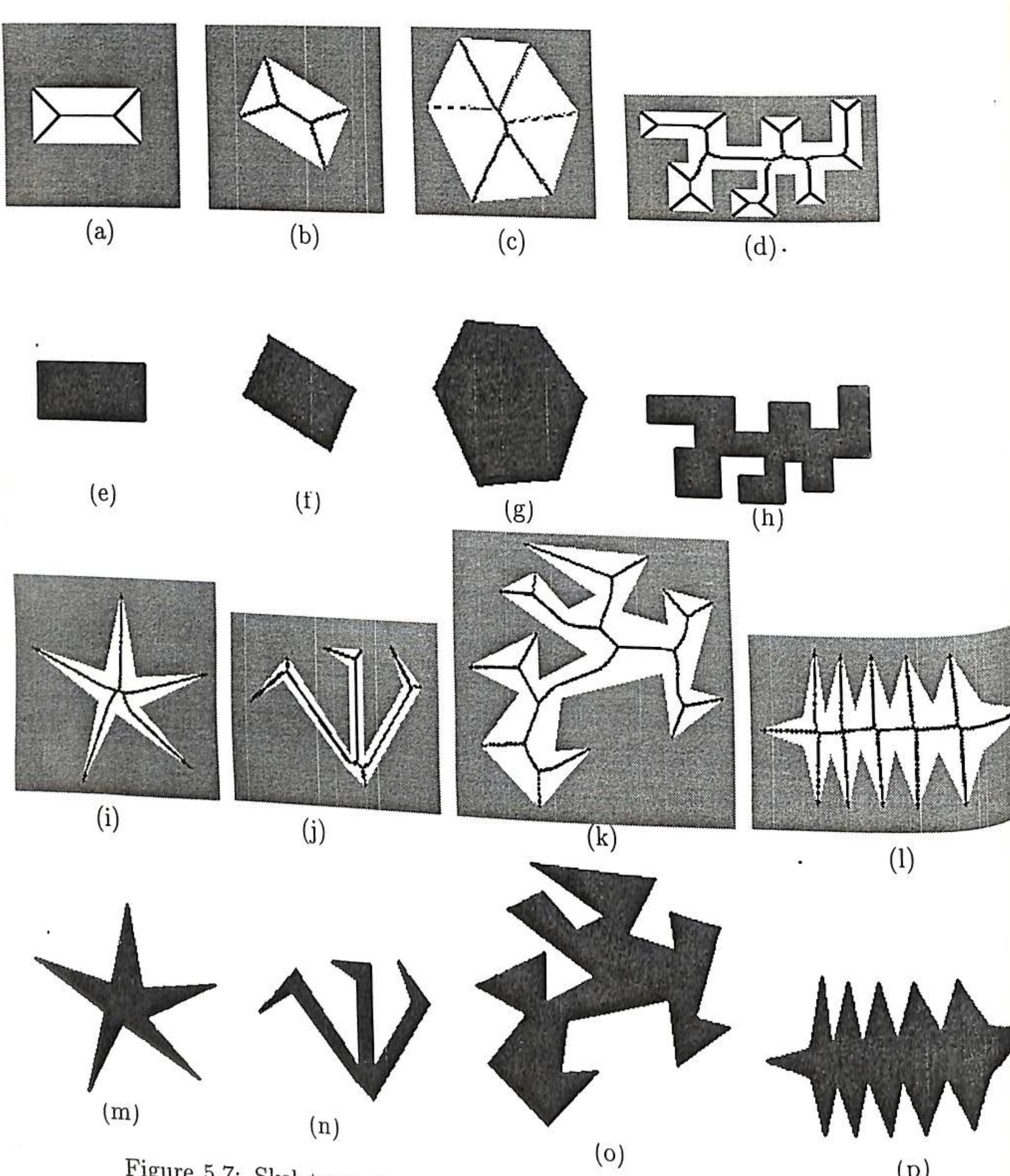


Figure 5.7: Skeletons of polygonal objects computed by Method1 and the reconstructed objects.

TH-2715_964102

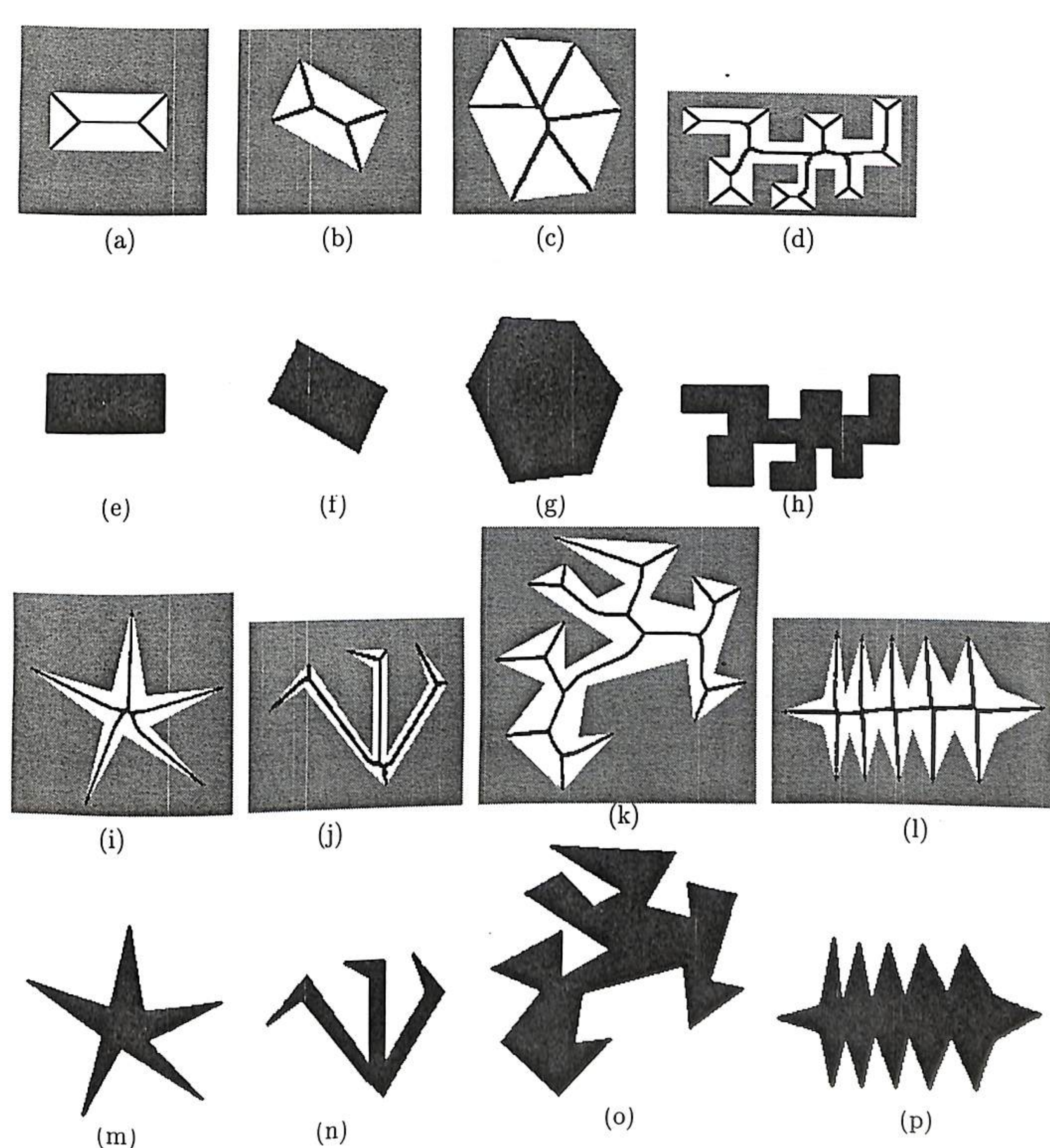


Figure 5.8: Skeletons of polygonal objects computed by Method2 and the reconstructed objects.

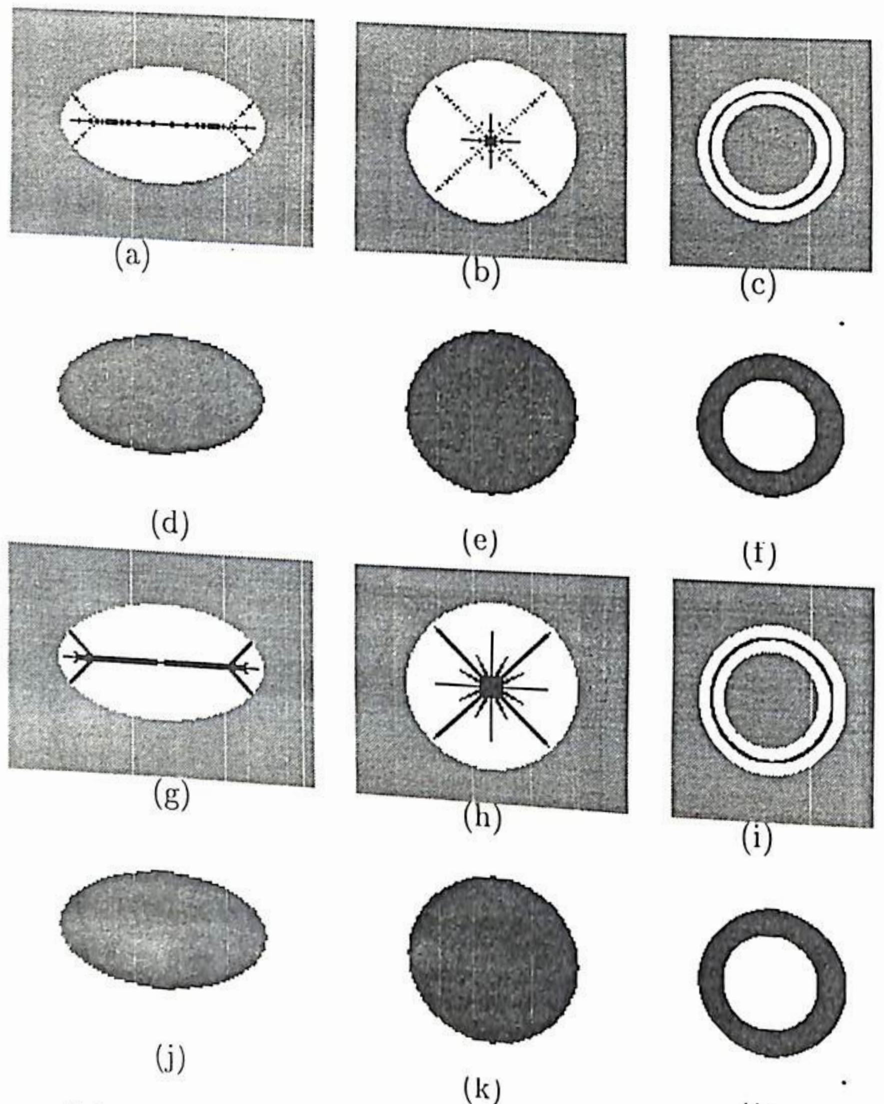


Figure 5.9: Skeletons computed by Method 1 and Method 2 of non-polygonal objects and the reconstructed objects.

imum of the Euclidean distances between the pixel and every background pixel. We have taken this skeleton as a template for comparison. The skeletons obtained from city-block distance and chessboard distance transforms are shown in Figures 5.10(c) and 5.10(d). These skeletons differ from the Euclidean skeleton in Figure 5.10(b) at those branches pointed to by arrows. Figure 5.10(e) shows the skeleton obtained from the chamfer 3-4 distance transform of the image. The chamfer distance is considered to be a good approximation to the Euclidean distance and can be computed by doing local neighborhood operations with the chamfer mask. The skeleton is reasonably good, but the reconstructed object shown in Figure 5.10(h) has deformities in its shape because the distance values do not match the exact Euclidean distance values. The arrows in the figure point to the deformities. The skeletons generated by the proposed algorithm are shown in Figures 5.10(f) and 5.10(g) and their reconstructed objects using the magnitude of the computed distance vector are given in Figures 5.10(i) and 5.10(j). The skeleton generated by Method 1 is shown in Figure 5.10(f) and the one by Method 2 is in Figure 5.10(g). The skeletons are similar to the one in 5.10(b) and give exact reconstruction of objects.

The computation of chamfer, city-block and chessboard distance transforms is done through local neighborhood operations and can therefore be implemented in a cellular array. But the skeletons from city-block and chessboard distance transforms differ from the Euclidean skeleton. The skeleton based on chamfer distance is reasonably good but it reconstructs the object poorly. The proposed algorithm generates the skeletons which match the Euclidean skeleton and reconstruct the objects exactly. From the simulation results, it is found that the skeleton using Method 2 is better connected than the one using Method 1 (see Figures 5.10(f) and 5.10(g)). However, from hardware implementation point of view, Method 1 is better than Method 2. From the results, it is also observed that, for polygonal objects, the generated skeletons have no branches missing and are insensitive to the discretization noise in the boundary.

The next section describes the VLSI architecture of the proposed scheme.

5.6 VLSI Architecture

Similar to the VLSI implementation of Algorithm *EDT_NNT*, Algorithm *SKEL* can also be implemented in hardware by an $n \times n$ cellular array where each cell represents one

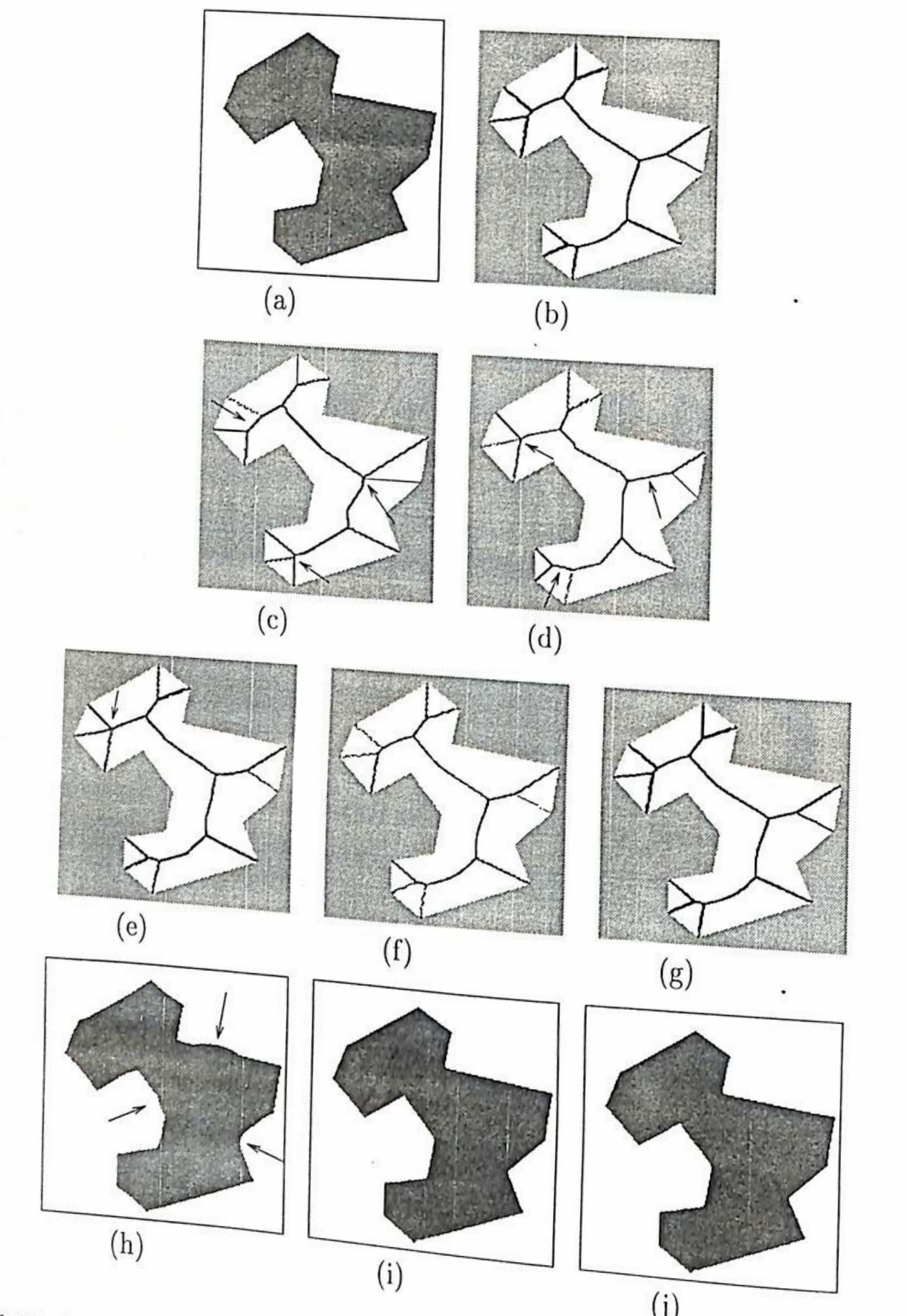


Figure 5.10: Skeletons and reconstructed images. (a) Test image taken from [KK87]. (b) Skeleton from Euclidean distance transform computed by brute force method. (c) City-block skeleton of test image. (d) Chessboard skeleton. (e) Chamfer 3-4 skeleton of test image. (f) Euclidean skeleton computed by Method 1. (g) Euclidean skeleton computed by Method 2. (h), (i) and (j) Reconstructed images of (e), (f) and (g).

TH-2715_964102

pixel of the binary image. A cell consists of storage elements $|\Delta x|, |\Delta y|, d_f, s, done$ and rc. The size of storage elements $|\Delta x|, |\Delta y|$ and d_f depends on the size of the image and $s, done$ and rc are one bit in size. Each cell is connected to its neighboring cells through which it receives the required values stored in the neighbors. The neighborhood connectivities required for both the Euclidean distance computation (Figure 4.2) and skeleton computation (Figure 5.2) are provided. The overall neighborhood connectivity of a cell is shown in Figure 5.11. The dashed lines show the connectivity for skeleton computation and they carry either *done* or d_f value of neighbors depending on the method implemented. The thick lines show the connectivity for EDT computation and they carry all the stored values except the s of the neighbors. The computation

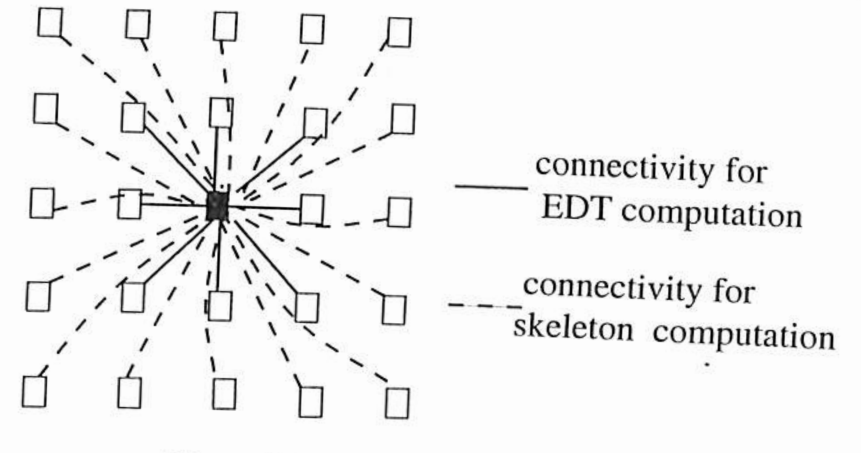


Figure 5.11: Connectivity of a cell

of $d_f, |\Delta x|, |\Delta y|$ and s is implemented by using data processing elements like adders, subtracters and comparators. The iteration number k is generated by an external counter and the cells are updated synchronously with respect to an external clock. The block diagram of a typical cell is shown in Figure 5.12.

The combinational logic (CL) for computing $(|\Delta x|, |\Delta y|)$ and d_f has been described in Section 4.5 and the CL for computing s is as follows. The computation of s is performed by the function *ComputeSkeleton()* if rc of a pixel is set to 1. The function *ComputeSkeleton()* of the algorithm by Method 1 needs testing rc and *done* flags of neighbors. This is easily implemented using few logic gates. The function *ComputeSkeleton()* by Method 2 compares the d_f values of neighbors with that of p_0. Since there are 16 neighbors, 16 comparators are needed. The complexity of the circuit needed for Method 1 is very less when compared to the one needed for Method 2. As Method 1 tests only single bit flags, the combinational logic of Method 1 is less complex than that of Method 2.

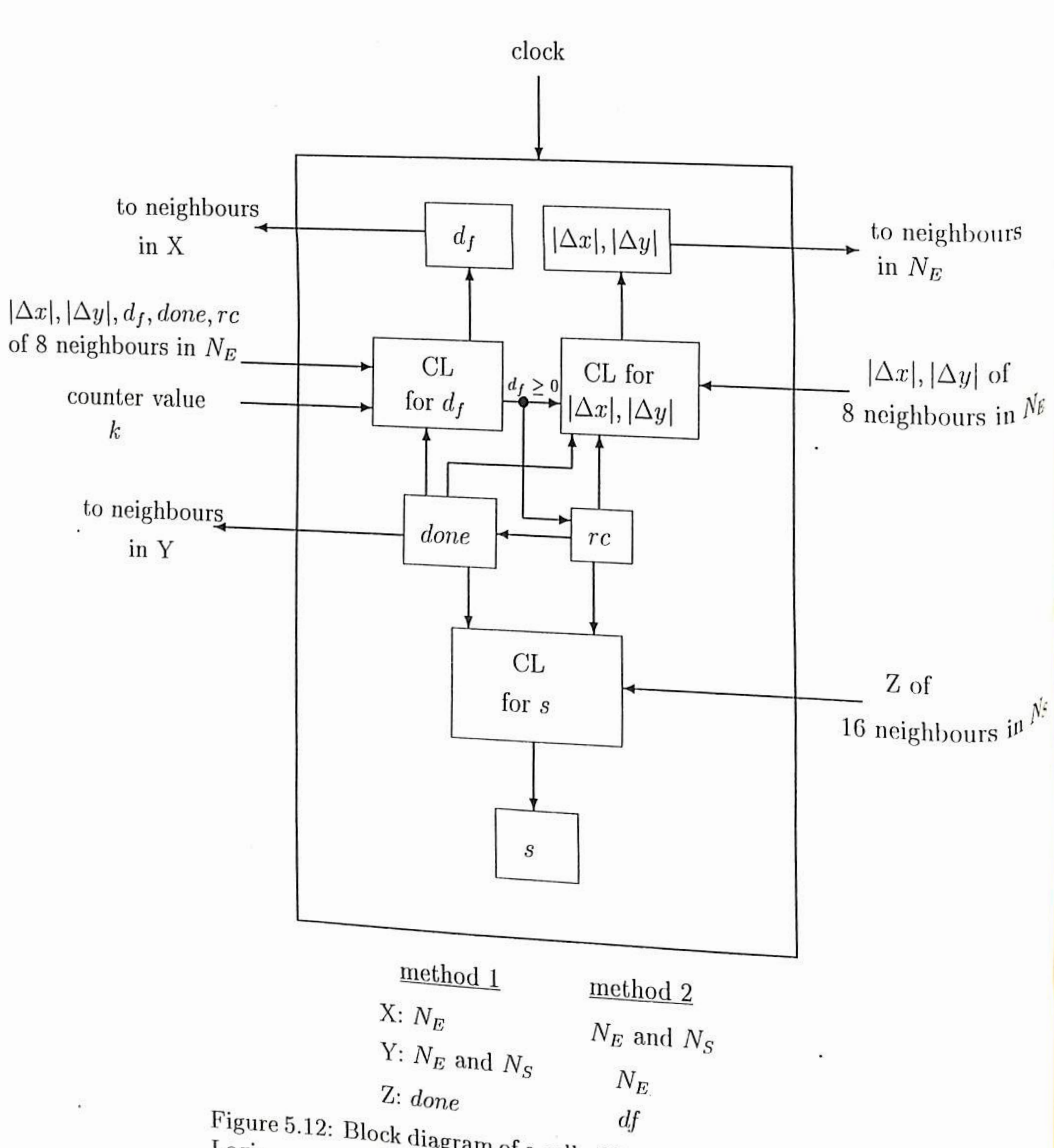


Figure 5.12: Block diagram of a cell. CL: Combinational Logic.

TH-2715_964102

Operation of Hardware

The cellular array is first initialized with the image by loading *done* of background pixels' cells to 1 and those of object pixels' cells to 0. All other storage elements are loaded with 0. Then the cellular array and the external counter are allowed to run synchronously with the external clock. Each clock pulse corresponds to one iteration and all the cells are updated simultaneously at each clock pulse. The updation is stopped after $\lceil\sqrt{2}n\rceil+1$ iterations; $\lceil\sqrt{2}n\rceil$ iterations are needed for computing EDT. The final contents of s and $(|\Delta x|, |\Delta y|)$ of all cells give the skeleton and vector Euclidean distance transform of the given image.

5.6.1 Design of a Cell

The cell comprises of components of the basic cell for distance computation (refer 4.5.1) with some modification in the clock to *done* flip-flop and additional components such as two flip-flops for s and rc and their corresponding combinational logic. The clocks to flip-flops *done* and s are activated only when rc is set. The clock to rc is same as *done* of basic cell because rc is set when the cell receives its distance information. But rc resets in the next clock cycle and hence the input to the flip-flop is given the complement of its output.

The logic for the input to s is based on the method adopted for computing skeleton. Consider a 5×5 neighborhood for skeleton computation. Let the neighboring pixels are numbered as in Figure 5.5. The logic circuits for computing skeleton by Methods 1 and 2 are shown in Figure 5.13 and 5.14.

Implementation

The VLSI designs of cells for both methods of computation of skeleton have been coded in VHDL and the designs have been mapped onto a target device of Xilinx FPGA, which is same as the one chosen for implementing the basic cell. The area in terms of number of different logic components of the chips obtained for different sizes of images are tabulated in Table 5.2. For both methods of skeleton computation, these values are same as in the table. The interpretation of these results is similar to the interpretation

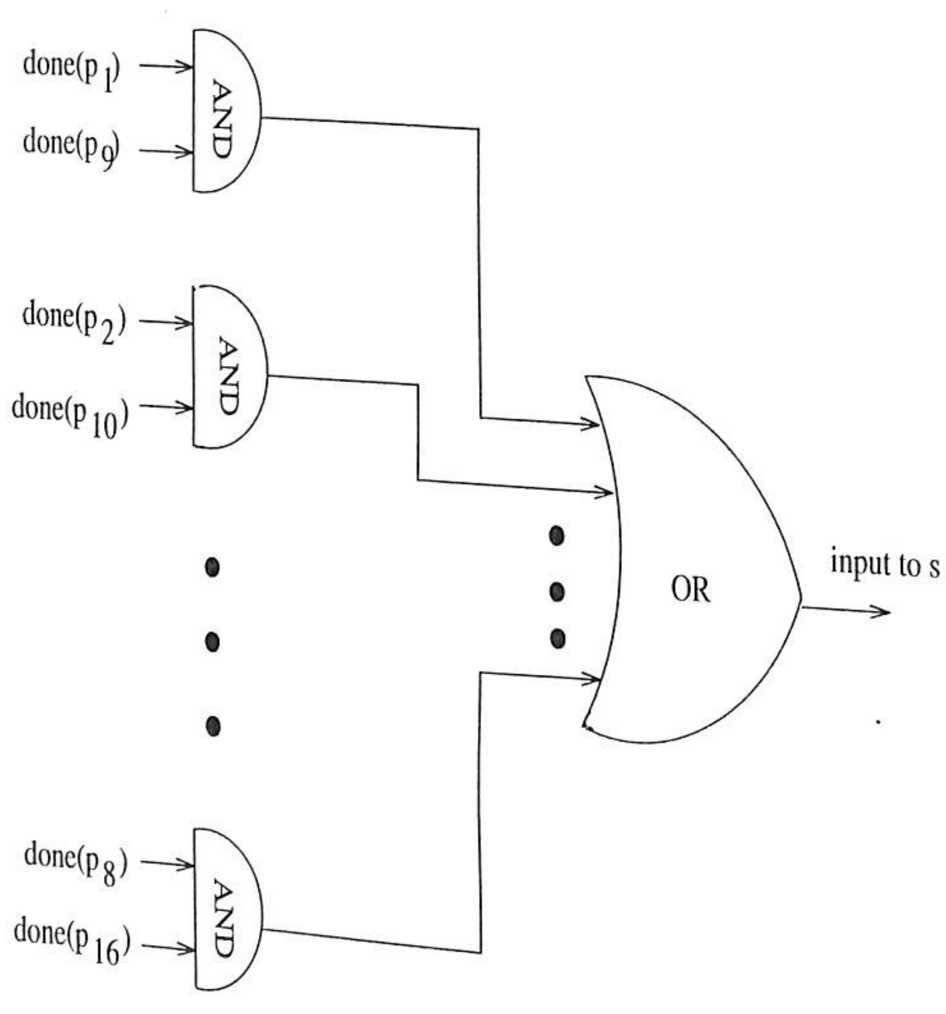


Figure 5.13: AND-OR logic for computing the input to s by Method 1.

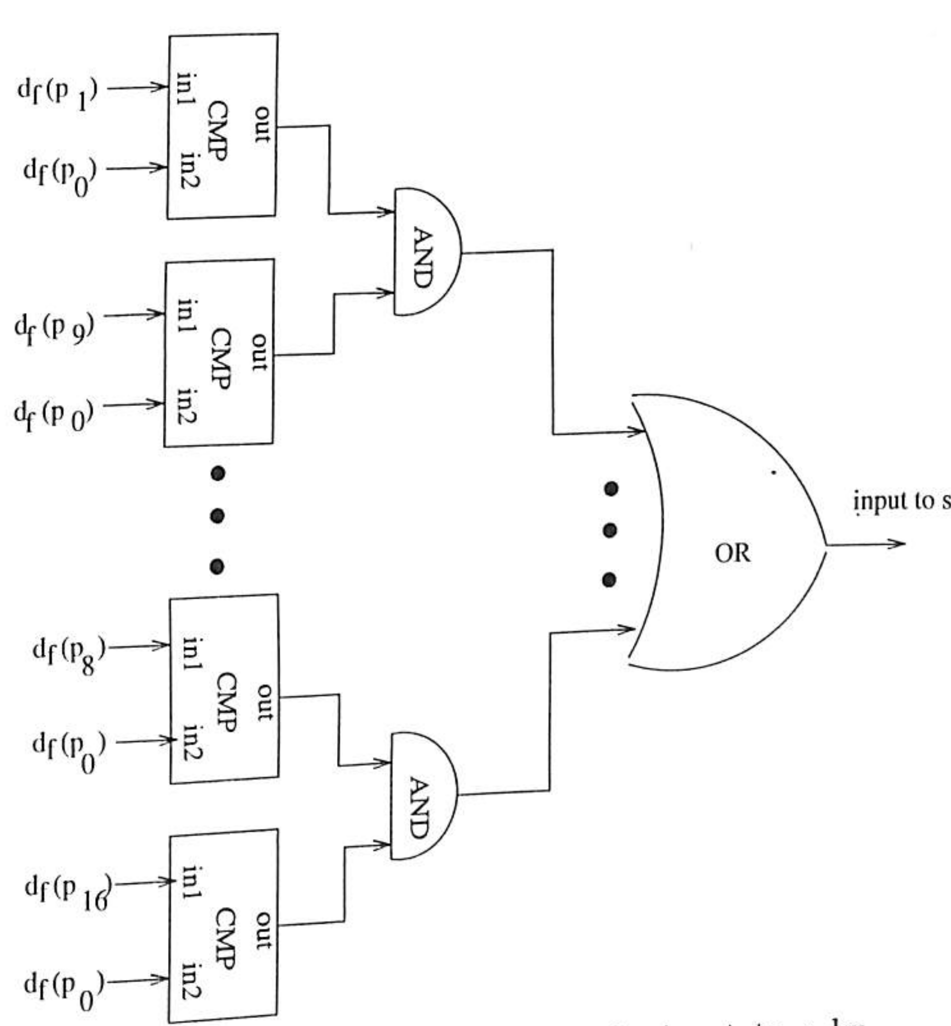


Figure 5.14: Logic for computing the input to s by Method 2. CMP indicates a comparator performing the comparison $in1 > in2$.

TH-2715_964102

Table 5.2: Number of different Xilinx FPGA logic components of chips obtained from the implementation of the design for computation of skeleton by both methods.

Image size $N \times N$	BUFG	DFF	FMAP	HMAP	IBUF	OBUF	OUTFF
16×16	4	10	433	71	142	9	8
32×32	4	11	518	97	167	10	10
64×64	4	12	607	84	192	11	12
128×128	4	13	660	119	217	12	14
256×256	4	14	667	123	242	13	16

BUFG: Global Buffer; DFF: D type flip-flop; FMAP,HMAP: Mapping design using F and H look-up tables; IBUF: Input Buffer; OBUF: Output Buffer; OUTFF: Output flip-flop

Table 5.3: Results obtained after placement and routing of FPGA components listed in Table 5.2 for different sizes of images.

Image size $N \times N$	Clock (MHz)	C path delay (ns)	Net delay (ns)	CLB	IOB	Gate count
16×16	7.2	159.66	19.43	248	159	3025
32×32	5.9	193.41	17	297	187	3670
64×64	5.3	201.30	19.28	336	215	4164
128×128	4.99	220.05	20.07	372	243	4693
256×256	5.2	215.42	18.94	383	271	4699

C path: Combination path; CLB: Complex Logic Block; IOB: I/O Block

of the results in Table 4.2 for the basic cell. The four global buffers are needed for four different clock inputs to memory elements (1) Δx and Δy (2) d_f (3) rc and (4) *done* and s. Similar to basic cell, OUTFFs are Δx and Δy whose outputs are directly taken as the outputs of cell and DFFs are used for d_f, rc and *done*. The number of OBUF is less than the number of DFF by one because the output of rc is not taken as the output of cell.

After mapping the design onto a Xilinx device, placement and routing of FPGA components have been carried out and the results are reported in Table 5.3. The layout of the chip corresponding to a 16 × 16 image size is shown in Figure 5.15. The cellular architectures for both methods have been then designed in VHDL and tested functionally in ModelSim package for different input images.

TH-2715_964102

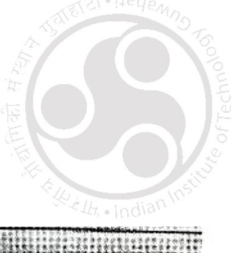


Figure 5.15: Layout of chip for the design of cell for computing skeleton of a 16 × 16 binary image by both methods.

Comparison with the Existing Architecture

N. Ranganathan and K.B. Doreswamy [RD95] have designed a systolic architecture for computing skeleton based on city-block metric. The skeleton is not rotation invariant and the architecture takes $O(n^2)$ clock cycles to compute the skeleton of an $n \times n$ image from its city-block distance transform.

The proposed architecture extracts the Euclidean skeleton which is rotation invariant and it takes only $O(n)$ clock cycles. The architecture also computes the skeleton directly from the city-block distance transform of the image.

5.7 Conclusions

In this chapter, we have presented a parallel algorithm to extract the skeleton of objects in a binary image. The algorithm simultaneously computes the Euclidean distance values of pixels. We have also presented the implementation of the algorithm in a cellular architecture. We have proposed two methods for computing the skeleton. The skeletons obtained by both the methods reconstruct the objects well. For polygonal objects, the extracted skeletons have no branches missing and are not influenced by the discretization of the boundary.

Some applications of skeleton are reported in Chapter 8. In the next chapter, another problem that can be solved using EDT is given.

TH-2715_964102

Chapter 6

Construction of the Voronoi Diagram

6.1 Introduction

In Chapter 4, we have designed a cellular architecture for the Euclidean distance transformation. The construction of Voronoi diagram using Euclidean distance transform is the theme of this chapter. The Voronoi diagram is a useful geometric structure in robotics, pattern classification and image representation. As discussed in the Section 2.2.2 of Chapter 6, the Voronoi diagram of a set of points in a plane partitions the plane into different regions called Voronoi regions where each region is associated with one and only one point. Algorithms are available for computing Voronoi diagram of points [PS85]. If a set of objects is considered instead of a point set, then the construction of Voronoi diagram needs first an appropriate representation of objects. Representation of real objects in a computer is difficult. However, an image of the objects in an environment can be obtained using a digital camera. One can attempt to construct an equivalent Voronoi diagram on a binary image of objects. Such a Voronoi diagram is called a discrete Voronoi diagram and it is defined as follows: *The discrete Voronoi diagram is given by the set of all boundary pixels of different Voronoi regions. The Voronoi region of an object consists of pixels which are closest to that object.*

The discrete Voronoi diagram is illustrated in Figure 6.1 for the case of three

Figure 6.1: Discrete Voronoi diagram (based on Euclidean metric) of three black pixels a, b and c. The pixels belonging to Voronoi regions of these black pixels are also labeled appropriately. The boundary of the regions is shown by thick lines. The pixels belonging to the discrete Voronoi diagram are the shaded ones close to the boundary.

black pixels. The diagram uses the Euclidean distance metric. Few algorithms [AdB86, TTT97] are available in literature for constructing discrete Voronoi diagram of objects (discussed in Chapter 3). However, they are based on city-block distance metric.

In this chapter, we propose a parallel algorithm to construct the discrete Voronoi diagram of objects based on Euclidean distance. The algorithm uses the method proposed in Chapter 4 for computing distance values. It constructs the discrete Voronoi diagram on a binary image by performing computations within a small neighborhood of every pixel and hence it is suitable for VLSI implementation in a cellular architecture. A preliminary version of this work is reported in [SNS99].

6.2 Method of Construction

We construct the Voronoi diagram of objects in a binary image consisting of objects O (1-pixels) and background B (0-pixels). Each object is considered to be a connected component in which the pixels are connected to their neighbors belonging to the compo-

TH-2715_964102

nent. As an example, the connected components of three objects are shown in Figure 6.2 (a). The construction of Voronoi diagram involves dilating each connected component of the binary image uniformly by one pixel unit and maintaining the connectivity between the neighboring pixels belonging to the same connected component. The dilation can be carried out iteratively based on the method proposed for EDT (Section 4.2) by constructing equidistant contours at each iteration. The iterative dilation of connected components in Figure 6.2(a) is shown in Figures 6.2(b)-(d). The dilation process is stopped when the entire image is occupied by the dilated objects. We call this *total dilation*. The boundary pixels of the dilated objects (similar to the boundary pixels of Voronoi regions in Figure 6.1) under total dilation give the discrete Voronoi diagram (shown in Figure 6.2). Since we establish the connectivity between the neighboring pixels belonging to the same dilated object, the boundary pixels are those pixels that are not connected to all their four neighbors, *viz*, North, South, East and West.

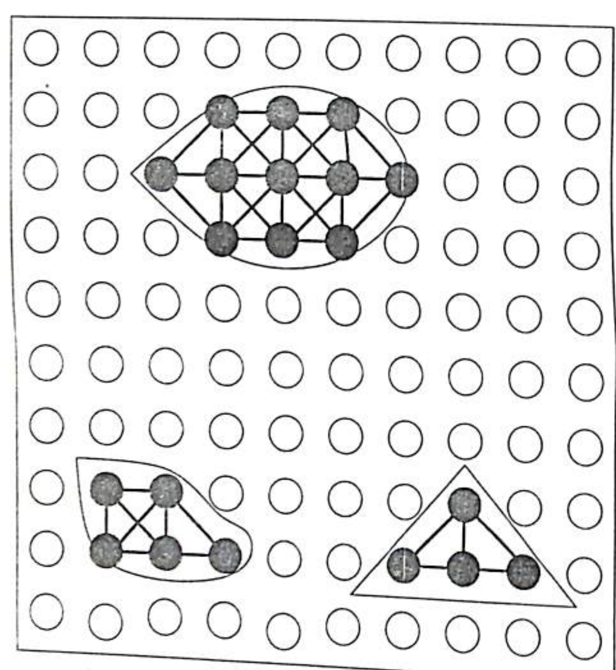
Remark 2 *Suppose we assume all pixels have appropriate labels. Then boundary pixels are those that do not have same label for all the four neighbors. Interior pixels on the other hand have same label for north, south, east and west neighbors. See Figure 6.1. Our algorithm however does not need explicit labeling.*

The discrete Voronoi diagram of objects in Figure 6.2(a) is shown in Figure 6.2(d) by the pixels which are not shaded black. The following lemma establishes that the discrete Voronoi diagram constructed by the above method is indeed the one defined.

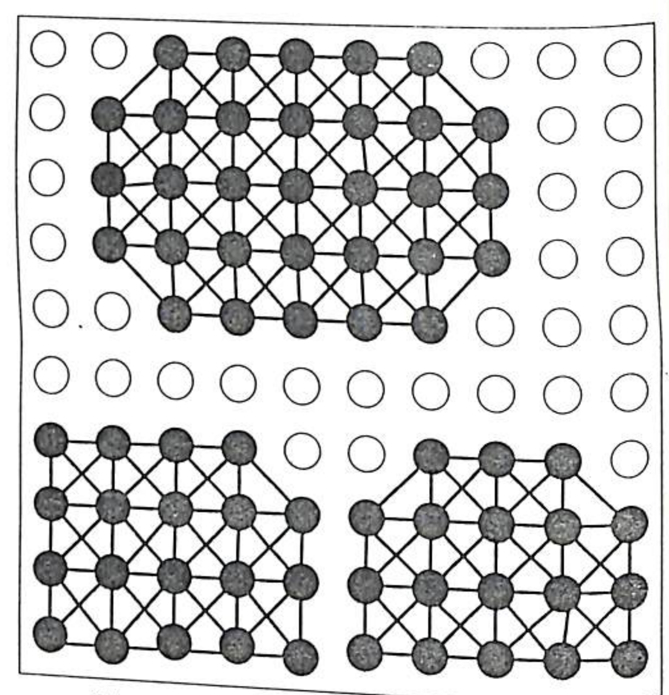
Lemma 1 *The boundary pixels of all dilated objects under total dilation give discrete Voronoi diagram of the objects.*

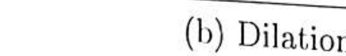
Proof The connected components of objects in a given image are dilated uniformly by the above method. If a background pixel p in the image is closest to a connected component C, then p belongs to the dilated component of C, *i.e.*, C_{dil}, after total dilation. C_{dil} consists of all those pixels closest to C and refers to the Voronoi region of C. Hence, the boundary pixels of the dilated objects form the Voronoi diagram. Q.E.D

Let p_C be a pixel at row x and column y of the image. Let $b(p_C)$ denote the binary value of the pixel p_C. We assign connectivity flags $N(p_C)$, $S(p_C)$, $E(p_C)$, $W(p_C)$, $NE(p_C)$, $NW(p_C)$, $SE(p_C)$ and $SW(p_C)$ to each pixel p_C in order to establish connectivities with north (p_N), south (p_S), east (p_E), west (p_W), north-east (p_{NE}), north west

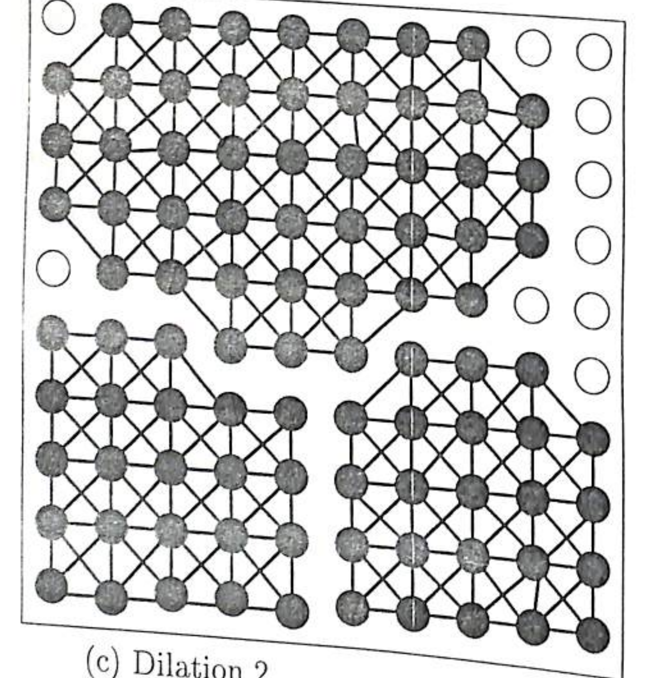


(a) Connected components

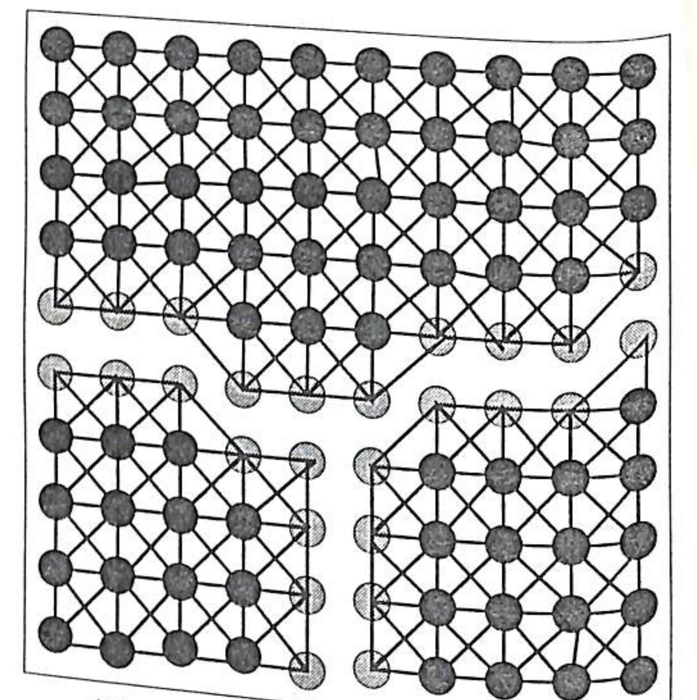




(b) Dilation 1



(c) Dilation 2

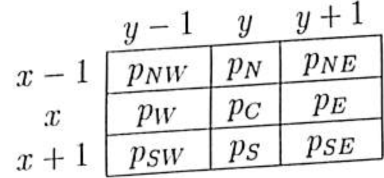


(d) Dilation 3 and Voronoi diagram

Figure 6.2: Illustration of our method for constructing Voronoi diagram. (a) Connected components of three objects. (b), (c) and (d) Dilated components after every iteration. Voronoi diagram is shown in (d).

TH-2715_964102

(p_{NW}), south east (p_{SE}) and south west (p_{SW}) neighbors respectively. The neighborhood of p_C, $N_{vd}(p_C)$, is shown in Figure 6.3. If p_C is a corner pixel in the image, then its neighbors inside the rectangular grid (representing the entire work space) are only of interest.



	$y-1$	y	$y+1$
$x-1$	p_{NW}	p_N	p_{NE}
x	p_W	p_C	p_E
$x+1$	p_{SW}	p_S	p_{SE}

Figure 6.3: Neighborhood $N_{vd}(p_C)$ of pixel p_C.

We iteratively dilate the objects by one pixel unit by setting $b(p_C)$. For a given binary image, the initialization of the iterative process is carried out first. The binary values b of object pixels are set to 1 while those of background pixels are set to 0. The connectivity flags of object pixels are set to 1 if the corresponding neighbors are also object pixels. The iterative process of dilation based on Euclidean distance involves the iterative computation of $(|\Delta x(p_C)|, |\Delta y(p_C)|)$ and $d_f(p_C)$ given in Section 4.2. We rewrite those iterative equations as follows. The $|\Delta x|, |\Delta y|$ and d_f of all pixels are initialized as below.

$$|\Delta x^{(0)}(p_C)| = |\Delta y^{(0)}(p_C)| = \begin{cases} 0 & p_C \in O \\ \infty & p_C \in B \end{cases} \tag{6.1}$$

$$d_f{}^{(0)}(p_C) = \begin{cases} 0 & p_C \in O \\ -\infty & p_C \in B \end{cases} \tag{6.2}$$

Here, the object pixels' values are initialized to 0. After this initialization, the objects are dilated iteratively by one pixel unit. At each iteration, the $|\Delta x|, |\Delta y|$ and d_f of background pixels are computed as below. The neighborhood $N_{vd}(p)$ is similar to $N_E(p_0)$ considered for EDT computation in Figure 4.2 and hence the pixels $p_C, p_E, p_{NE}, p_N, p_{NW}, p_W, p_{SW}, p_S$ and p_{SE} can be denoted by $p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7$ and p_8.

$$d_f{}^{(k+1)}(p_0) = 2(k+1) + \max_{p_i \in N_E(p_0)} [d_f{}^{(k)}(p_i) - (\Delta X_i + \Delta Y_i)] \tag{6.3}$$

where

$$\Delta X_i = \begin{cases} 0 & i = 0, 1\&5 \\ 2|\Delta x^{(k)}(p_i)| + 1 & i = 2, 3, 4, 6, 7\&8 \end{cases}$$

$$\Delta Y_i = \begin{cases} 0 & i = 0, 3\&7 \\ 2|\Delta y^{(k)}(p_i)| + 1 & i = 1, 2, 4, 5, 6\&8 \end{cases}$$

and

$$(|\Delta x^{(k+1)}(p_0)|, |\Delta y^{(k+1)}(p_0)|) = (\Delta x_i, \Delta y_i) \quad (6.4)$$

where

$$\Delta x_i = \begin{cases} |\Delta x^{(k)}(p_i)| & i = 0, 1\&5 \\ |\Delta x^{(k)}(p_i)| + 1 & i = 2, 3, 4, 6, 7\&8 \end{cases}$$

$$\Delta y_i = \begin{cases} |\Delta y^{(k)}(p_i)| & i = 0, 3\&7 \\ |\Delta y^{(k)}(p_i)| + 1 & i = 1, 2, 4, 5, 6\&8 \end{cases}$$

and i in Equation (6.4) corresponds to pixel p_i that satisfies Equation (6.3).

For any background pixel p_C, upto a certain iteration $d_f(p_C)$ is less than 0 and it is greater than or equal to 0 in the successive iterations. At any iteration k, the new boundary pixels of a dilated object are those background pixels p_C whose $d_f(p_C)$ crosses 0 at iteration k. The $b(p_C)$ of these pixels p_C are set to 1. The connectivity of each of these pixels are established to the neighbors from which the Euclidean distance information, $|\Delta x(p_C)|, |\Delta(p_C)|$ and $d_f(p_C)$ are obtained. This is done by setting the corresponding connectivity flag. In the successive iterations, the connectivities are established totally with the neighbors belonging to the same dilated object to which p_C belongs. This is done as follows. For example, the connectivity of a pixel p_C is established with its west neighbor p_W by setting $W(p_C)$ to 1 if one of the following conditions is satisfied.

1. p_W is connected to p_C.
2. p_C is connected to p_N and p_N is connected to p_W. p_W is the south west neighbor of p_N.
3. p_C is connected to p_{NW} and p_{NW} is connected to p_W. p_W is the south neighbor of p_{NW}.
4. p_C is connected to p_S and p_S is connected to p_W. p_W is the north west neighbor of p_S.

Guwahati · Indian Institute of Technology

5. p_C is connected to p_{SW} and p_{SW} is connected to p_W. p_W is the north neighbor of p_{SW}.

This is illustrated in Figure 6.4(1). The conditions for establishing the connectivities with other neighbors are illustrated in Figures 6.4(2)-(8).

The iterative procedure is stopped when the connectivity flags are established fully after total dilation. The Voronoi diagram is represented by the boundaries of the dilated objects. It consists of those pixels for which any one of the four flags, N, S, E and W, is not set.

We have proposed a parallel algorithm which performs the dilation of all objects simultaneously.

6.3 Algorithm

Given a binary image of size $n \times n$, we initialize $b, |\Delta x|, |\Delta y|, d_f$ and connectivity flags $(N, S, E, W, NE, NW, SE$ and $SW)$ of object pixels to their corresponding values. Those of background pixels are computed iteratively by dilating the objects. The iteration is continued until the connectivity flags are established fully. The pixels belonging to Voronoi diagram are then identified using the connectivity flags.

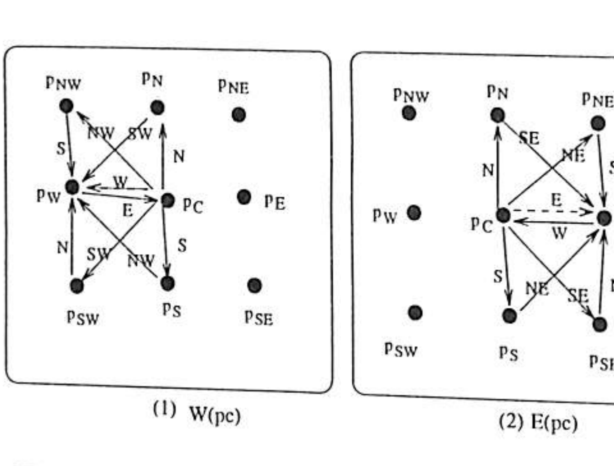
<u>ALGORITHM:</u> *VD*

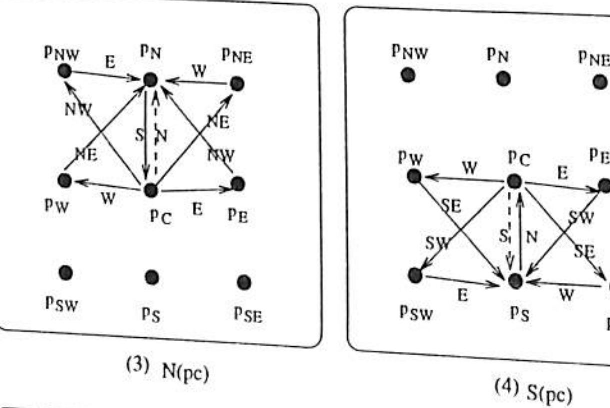
<u>Inputs:</u> Given binary image, consisting of object and background pixels.

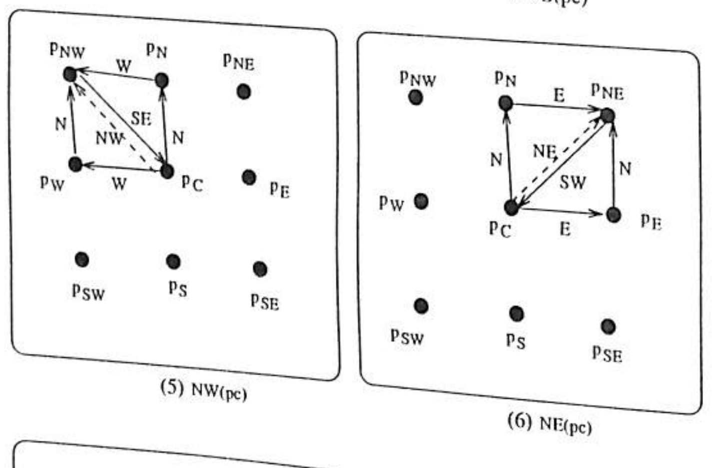
<u>Outputs:</u> The Voronoi diagram.

<u>Step 1:</u> Initialization

The initialization of $|\Delta x|, |\Delta y|$ and d_f is given by Equations (6.1) and (6.2). To avoid initializing with ∞ and $-\infty$, the value b of all pixels are initialized with respect to the binary image and the $|\Delta x|, |\Delta y|$ and d_f of all pixels are initialized to 0.


(1) W(pc) (2) E(pc)


(3) N(pc) (4) S(pc)


(5) NW(pc) (6) NE(pc)

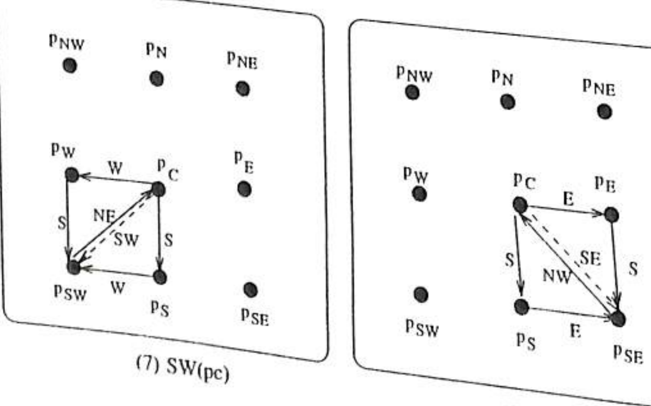

(7) SW(pc) (8) SE(pc)

Figure 6.4: Conditions for setting connectivity flags of a pixel p_C.

TH-2715_964102

$$|\Delta x^{(0)}(p_C)| = |\Delta y^{(0)}(p_C)| = d_f^{(0)}(p_C) = 0$$

$$b(p_C) = \begin{cases} 1 & p_C \in O \\ 0 & p_C \in F \end{cases}$$

The connectivity flags of object pixels are initialized as follows. The flag $N(p_C)$ is set to 1 if the binary value $b(p_N)$ of the north neighbor p_N is 1. Similarly other connectivity flags are set.

$$N(p_C) = b(p_N); \ \cdots$$

$$NE(p_C) = b(p_{NE}); \ \cdots$$

Step 2: Iterative process of dilation

At each iteration $k \geq 0$, the distance vector $(|\Delta x|, |\Delta y|)$ and d_f of pixels whose integer approximation of Euclidean distance value equals k are computed. The binary value b of these pixels are set to 1. One of the eight connectivity flags of these pixels is also set. This flag corresponds to the neighbor from which the Euclidean distance information is received to compute the pixel's $(|\Delta x|, |\Delta y|)$ and d_f. For the pixels whose Euclidean distance values have already been computed in any of the previous $k-1$ iterations, the remaining seven connectivity flags are set according to the conditions in Figure 6.4. The pseudocode of the iterative process is as follows.

$k = 0$
repeat iteration
 for all pixels p_C **do in parallel**

 switch $(b(p_C))$
 0: $\{(|\Delta x|, |\Delta y|)$ has not been computed yet$\}$
 Perform $(|\Delta x|, |\Delta y|)$ computation$\}$

if there exists $p_i \in N_{vd}(p_C)$ such that $b(p_i)$=1 **do**
Compute $d_f^{(k+1)}(p_C)$ as in Equation (6.3)

if $(d_f^{(k+1)}(p_C) \geq 0)$ **do**
Compute $(|\Delta x^{(k+1)}(p_C)|, (\Delta y^{(k+1)}(p_C)|)$
as in Equation (6.4)
$b(p_C)$=1 {binary value}
Set the connectivity flag
{flag corresponds to the pixel from which
the Euclidean distance information is received}
end if
end if
break {switch $b = 0$}

1: $\{(|\Delta x|, |\Delta y|)$ has been computed}
Update $d_f(p)$ by adding $2(k+1)$

{Establish connectivity of p by computing
the remaining seven flags}
if $(E(p_W) \vee [N(p_C) \wedge SW(p_N)] \vee [NW(p_C) \wedge S(p_{SW})]$
$\vee [S(p_C) \wedge NW(p_S)] \vee [SW(p_C) \wedge N(p_{SW})])$
$W(p_C) = 1$
{with p_W according to conditions in Figure 6.4(1)}
...
...
{similarly with p_E,p_N and p_S according to
conditions in Figure 6.4(2)-(4)}

if $(SE(p_{NW}) \vee (N(p_C) \wedge W(p_N)) \vee (W(p_C) \wedge N(p_W))$
$NW(p_C) = 1$
{with p_{NW} according to conditions in Figure 6.4(5)}
...
...
{similarly with p_{NE},p_{SW} and p_{SE} according to
conditions in Figure 6.4(6)-(8)}

TH-2715_964102

break {switch $b = 1$}

end for
k = k+1
until N, S, E, W, NE, NW, SE and SW flags remain unchanged for all pixels

In the above procedure, at each iteration, all the pixels are processed in parallel. Depending on the value of $b(p_C)$ of each pixel p_C, different operations are carried out in p_C. When $b(p_C) = 0$, displacement vector has not been computed already for p_C. If $d_f(p_C) \geq 0$, then $(|\Delta x(p_C)|, |\Delta y(p_C)|)$ is computed, $b(p_C)$ is set and the connectivity flag corresponding to the neighbor which gives $d_f(p_C)$ is also set. When $b(p_C) = 1$, p_C belongs to the dilated object and hence $d_f(p_C)$ is only updated and the connectivity establishment is carried out. The iterative process runs until the connectivities are fully established after the b values of all the pixels are set to 1. The connectivities are fully established when the connectivity flags of all pixels remain unchanged for successive iterations.

Step 3: Identification of Voronoi pixels

Let $vor(p_C)$=1 denote p_C belongs to Voronoi diagram. The pseudocode for identification of Voronoi pixels is as follows.

for all pixels p_C **do in parallel**
if $(\neg N(p_C) \vee \neg S(p_C) \vee \neg E(p_C) \vee \neg W(p_C))$
$vor(p_C)$=1
else $vor(p_C)$=0

The time and space complexities of the algorithm are as follows.

6.3.1 Complexity Analysis

The time complexity of the proposed algorithm is determined by the **repeat-until** loop. The time taken by the loop includes the time required for total dilation and an additional time t_a for establishing the connectivities between the pixels of the dilated objects. The time for total dilation is equal to the maximum integer distance number of iterations needed for total dilation is equal to the maximum integer distance

if there exists $p_i \in N_{vd}(p_C)$ such that $b(p_i)$=1 **do**
  Compute $d_f^{(k+1)}(p_C)$ as in Equation (6.3)

  if $(d_f^{(k+1)}(p_C) \geq 0)$ **do**
    Compute $(|\Delta x^{(k+1)}(p_C)|, |\Delta y^{(k+1)}(p_C)|)$
     as in Equation (6.4)
    $b(p_C)$=1 {binary value}
    Set the connectivity flag
    {flag corresponds to the pixel from which
    the Euclidean distance information is received}
  end if
end if
break {switch $b = 0$}

1: $\{(|\Delta x|, |\Delta y|)$ has been computed}
Update $d_f(p)$ by adding $2(k+1)$

{Establish connectivity of p by computing
  the remaining seven flags}
if $(E(p_W) \vee [N(p_C) \wedge SW(p_N)] \vee [NW(p_C) \wedge S(p_{SW})]$
  $\vee [S(p_C) \wedge NW(p_S)] \vee [SW(p_C) \wedge N(p_{SW})])$
  $W(p_C) = 1$
  {with p_W according to conditions in Figure 6.4(1)}
...
...
{similarly with p_E,p_N and p_S according to
  conditions in Figure 6.4(2)-(4)}

if $(SE(p_{NW}) \vee (N(p_C) \wedge W(p_N)) \vee (W(p_C) \wedge N(p_W))$
  $NW(p_C) = 1$
  {with p_{NW} according to conditions in Figure 6.4(5)}
...
...
{similarly with p_{NE},p_{SW} and p_{SE} according to
  conditions in Figure 6.4(6)-(8)}

    break {switch $b = 1$}

  end for
  k = k+1
until N, S, E, W, NE, NW, SE and SW flags remain unchanged for all pixels

In the above procedure, at each iteration, all the pixels are processed in parallel. Depending on the value of $b(p_C)$ of each pixel p_C, different operations are carried out in p_C. When $b(p_C) = 0$, displacement vector has not been computed already for p_C. If $d_f(p_C) \geq 0$, then $(|\Delta x(p_C)|, |\Delta y(p_C)|)$ is computed, $b(p_C)$ is set and the connectivity flag corresponding to the neighbor which gives $d_f(p_C)$ is also set. When $b(p_C) = 1$, p_C belongs to the dilated object and hence $d_f(p_C)$ is only updated and the connectivity establishment is carried out. The iterative process runs until the connectivities are fully established after the b values of all the pixels are set to 1. The connectivities are fully established when the connectivity flags of all pixels remain unchanged for successive iterations.

Step 3: Identification of Voronoi pixels

Let $vor(p_C)$=1 denote p_C belongs to Voronoi diagram. The pseudocode for identification of Voronoi pixels is as follows.

for all pixels p_C **do in parallel**
  if $(\neg N(p_C) \vee \neg S(p_C) \vee \neg E(p_C) \vee \neg W(p_C))$
    $vor(p_C)$=1
  else $vor(p_C)$=0

The time and space complexities of the algorithm are as follows.

6.3.1 Complexity Analysis

The time complexity of the proposed algorithm is determined by the **repeat-until** loop. The time taken by the loop includes the time required for total dilation and an additional time t_a for establishing the connectivities between the pixels of the dilated objects. The number of iterations needed for total dilation is equal to the maximum integer distance

value that a pixel takes in the given image. For an image of size $n \times n$, it cannot be greater than $\sqrt{2}n$ and hence the time for total dilation is $O(n)$. t_a depends on the shape of the objects in the image. In the worst case (for instance, spiral shaped objects), t_a is $O(n^2)$. The space complexity of the algorithm is $O(n^2)$, since the algorithm needs constant space for storing $|\Delta x|, |\Delta y|, d_f, b$ and the eight connectivity flags for each pixel.

The small local neighborhood operations of the algorithm make it suitable for VLSI implementation in a cellular architecture.

6.4 VLSI Architecture

The proposed algorithm can be implemented in an $n \times n$ array of identical cells where each cell stores $|\Delta x|, |\Delta y|, d_f, b$ and flags N, S, E, W, NE, NW, SE and SW. The computation of these values is implemented using data processing elements such as adders, subtracters and comparators. The size of the storage elements $|\Delta x|, |\Delta y|, d_f$ depends on the size of the image while the other storage elements ($N, S, E, W, NE, NW, SE, SW$ and b) are one bit in size. Each cell is connected to all the eight neighboring cells through which it receives the values stored in the neighbors and the neighborhood connectivity is shown in Figure 6.5. The cell has two modules, *viz.*, dilation module and connectivity establishment module. The dilation module has components for distance computation and the connectivity establishment module has components for establishing connectivity between neighboring pixels.

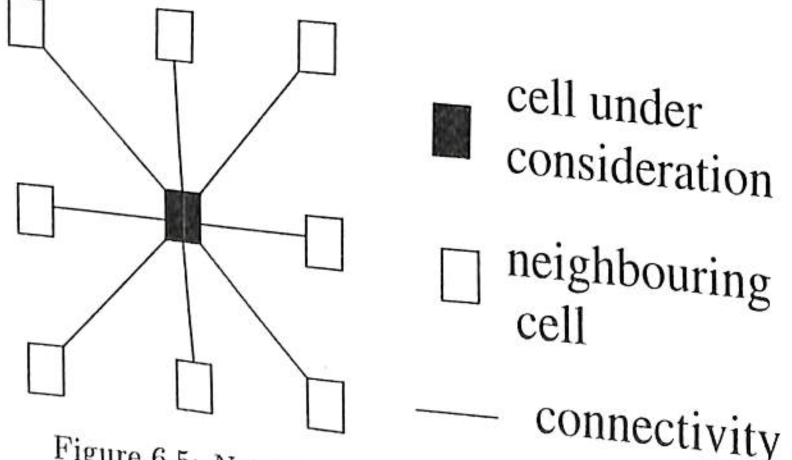


Figure 6.5: Neighborhood connectivity of a cell.

The iteration number k is generated by an external counter and the cells are updated synchronously with respect to an external clock. The block diagram of a typical cell is shown in Figure 6.6. The combinational logic for $|\Delta x|$, $|\Delta y|$ and d_f have been

TH-2715_964102

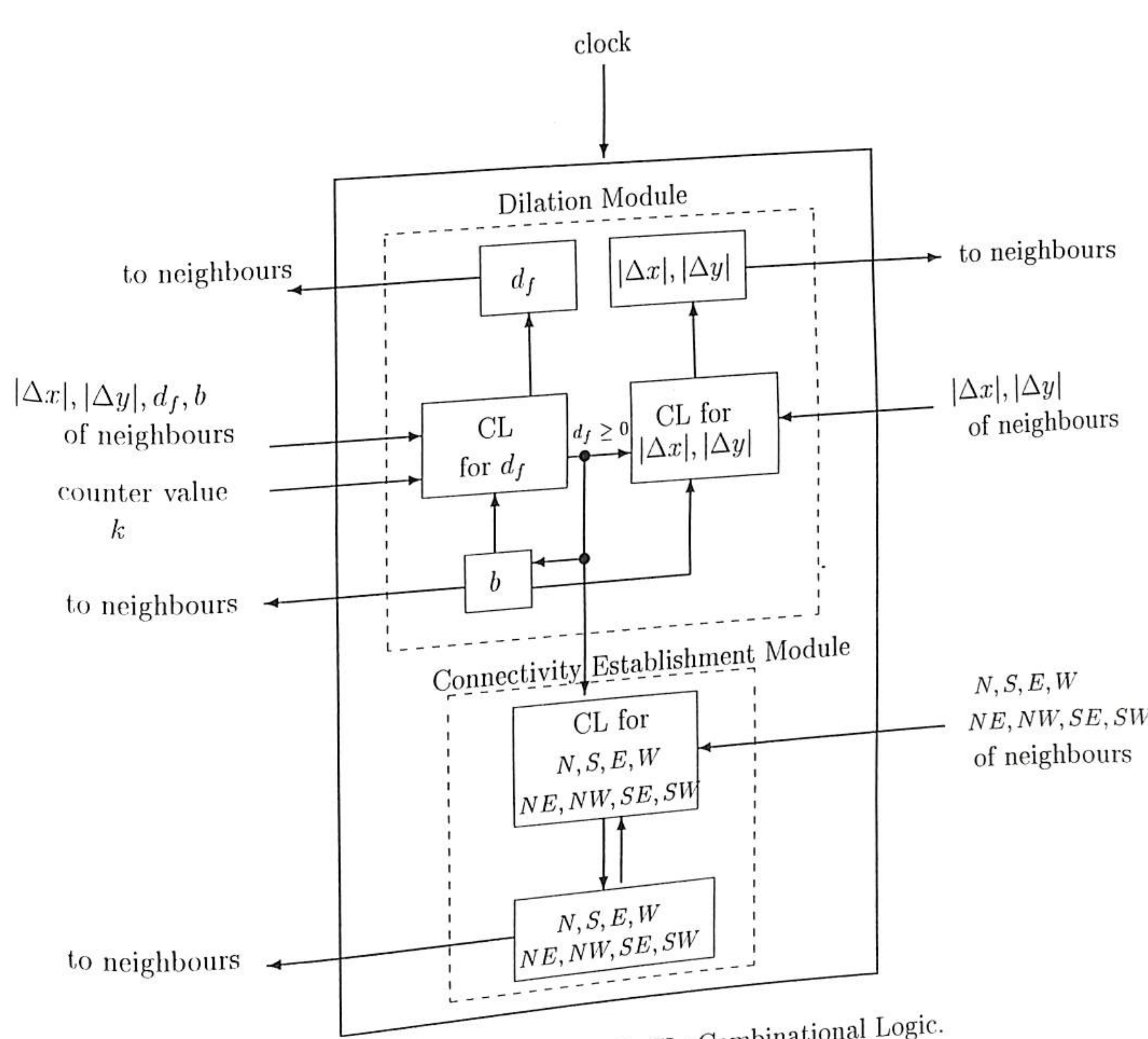


Figure 6.6: Block diagram of a cell. CL: Combinational Logic.

described already in Section 4.5. The logic for connectivity flags involves simple testing of connectivity flags of neighbors.

Operation of Hardware

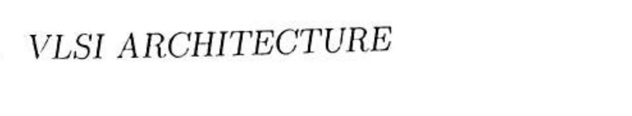
The storage elements of the cellular array are first initialized as in the algorithm given in Section 6.3. Then the cellular array and the external counter are allowed to run synchronously for n^2 iterations and the cells are updated simultaneously at each clock pulse. After updation is stopped, the Voronoi diagram is given by those cells for which any one of the four flags N, S, E and W is 0,*i.e.*, $N \wedge S \wedge E \wedge W$ is 0.

6.4.1 Design of a cell

The dilation module consists of the components of basic cell (Figure 4.6) and the flip-flop that stores the flag *done* in basic cell is used to store b. Each of the eight input sets, corresponding to eight neighbors, to the MAX module (Figure 4.8) also includes an eight bit code vector. The code vectors are as follows. Assume that the flags are arranged in the following order: N,S,E,W,NE,NW,SE,SW. Then "10000000" is included in the set of inputs corresponding to north neighbor and "00001000" is included in the set corresponding to north-east neighbor. Similarly, for other sets, the code vector with the corresponding bit being set to 1 is included. Let "$f_N f_S f_E f_W f_{NE} f_{NW} f_{SE} f_{SW}$" be the output bit vector of the MAX module. If the cell receives the distance values from a neighbor at some clock cycle, then this bit vector corresponds to that neighbor.

The connectivity establishment module has eight flip-flops for storing the flags and a combinational logic to implement the conditions (Figure 6.4) for setting the flip-flops. For example, the input to the flip-flop W is generated by the logic circuit shown in Figure 6.7. At any clock cycle, this depends on f_W if the cell is receiving its Euclidean distance information (given by *cndn* described in Section 4.5.1) or on the conditions in Figure 6.4(1) if the cell has already received its distance information (given by b).

The design of a cell has been coded in VHDL and the implementation of the design in Xilinx FPGA has been carried out. The components of the chips obtained for different image sizes are tabulated in Table 6.1. The interpretation of these results is similar to that of basic cell. The eight additional DFFs, when compared to DFF entries in Table 4.2 corresponding to EDT, are due to connectivity flags. An additional BUFG is meant for the clock input to flip-flops used for connectivity flags. The increase in IBUF and OBUF are also due to the inputs and outputs of connectivity establishment module of



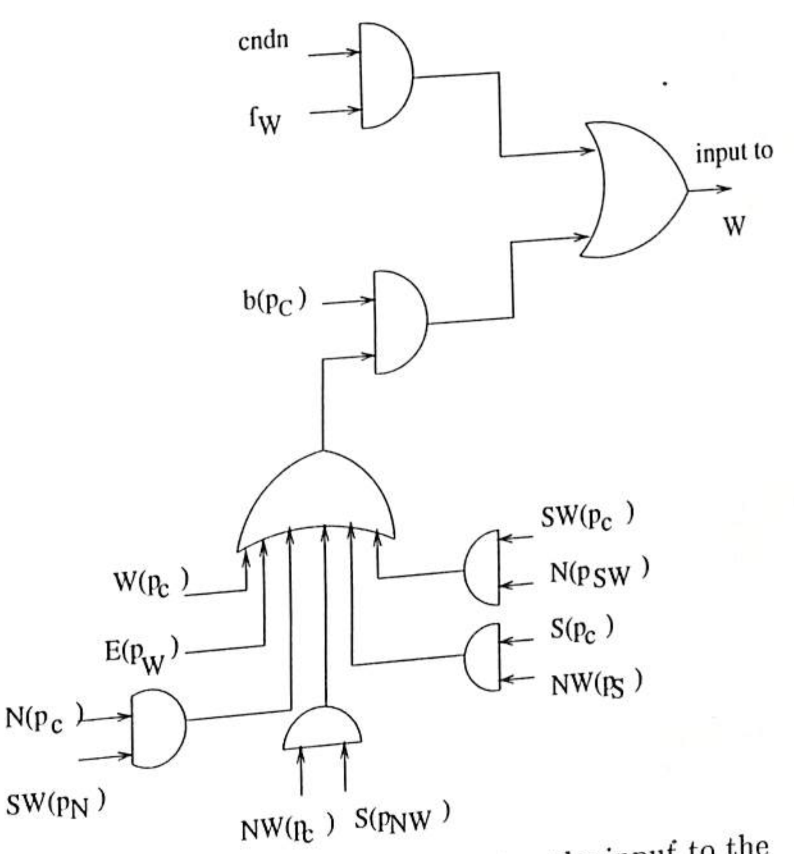


Figure 6.7: Logic circuit for generating the input to the flip-flop W.

Table 6.1: Number of different Xilinx FPGA logic components of chips obtained from the implementation of the design of cell for computation of Voronoi diagram

Image size $N \times N$	BUFG	DFF	FMAP	HMAP	IBUF	OBUF	OUTFF
16×16	3	17	459	95	174	17	8
32×32	3	18	541	118	199	18	10
64×64	3	19	585	92	224	19	12
128×128	3	20	666	109	249	20	14

BUFG: Global Buffer; DFF: D type flip-flop; FMAP,HMAP: Mapping design using F and H look-up tables; IBUF: Input Buffer; OBUF: Output Buffer; OUTFF: Output flip-flop

Table 6.2: Results of placement and routing of components in Table 6.1

Image size $N \times N$	Clock (MHz)	C path delay (ns)	Net delay (ns)	CLB	IOB	Gate count
16×16	6.28	165.66	18.65	267	199	3363
32×32	5.62	194.53	22.8	300	227	3772
64×64	6.01	178.06	16.33	360	255	4441
128×128	5.15	200.4	21.34	390	283	4860

C path: Combinational path; CLB: Complex Logic Block; IOB: I/O Block

the cell.

The results of placement and routing are reported in Table 6.2 and the layout of the chip corresponding to 16 × 16 size of an image is shown in Figure 6.8. The entire cellular architecture has also been designed and functionally tested in ModelSim.

In the next section, the performance of the algorithm proposed in the Section 6.3 is tested by simulating it on a sequential computer.

6.5 Simulation Results and Discussions

The proposed algorithm has been simulated on a sequential computer to test its performance. It has been implemented in ANSI-C on an HP 9000 series 700 J Model J200 workstation with 64 MB RAM. We have generated some test images using a graphics

Figure 6.8: Layout of the chip for the design of cell for computing Voronoi diagram of a 16 × 16 binary image.

package. The discrete Voronoi diagrams computed for these images are shown in Figure 6.9. The number of iterations needed to construct the Voronoi diagram of the image (Figure 6.9(a)) with size 117 × 217 containing random points is 143 and that of the image (Figure 6.9(b)) with size 285 × 225 containing objects is 113. Total dilation occurs after 77 iterations for the former image and 35 iterations for the latter.

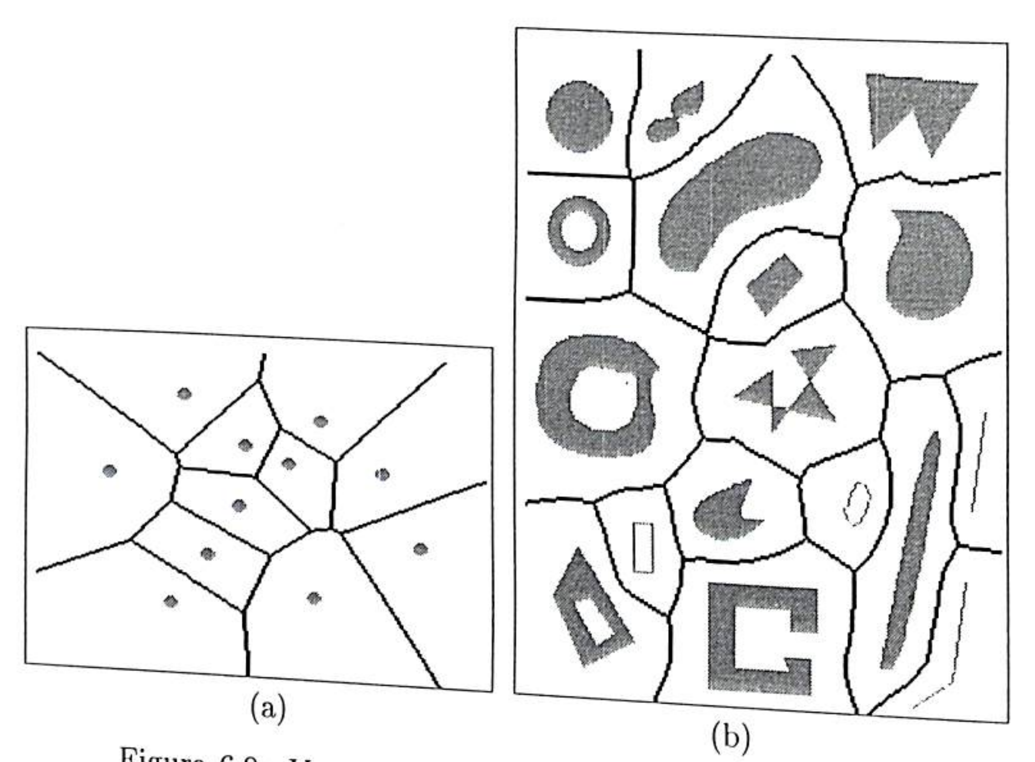


Figure 6.9: Voronoi diagrams of images. (a) Voronoi diagram of random points. (b) Voronoi diagram of well separated objects.

The algorithm constructs the exact Voronoi diagram for most of the cases except for a special case shown in Figure 6.10. The Voronoi diagram has a false branch which is pointed to by an arrow. This branch occurs whenever an object 1 is not enclosed completely by object 2, but during dilation, the dilated version of object 2 encloses the object 1. This is due to the failure in establishing the connectivity between some neighbors of the dilated version of object 2, because of the presence of object 1. For applications such as robot path planning, this false branch is advantageous. This will be discussed in Chapter 8. In the case of images consisting of well separated objects with arbitrary shape, the Voronoi diagram is exact.

We have also tested the performance of the algorithm on a real image. The image

TH-2715_964102

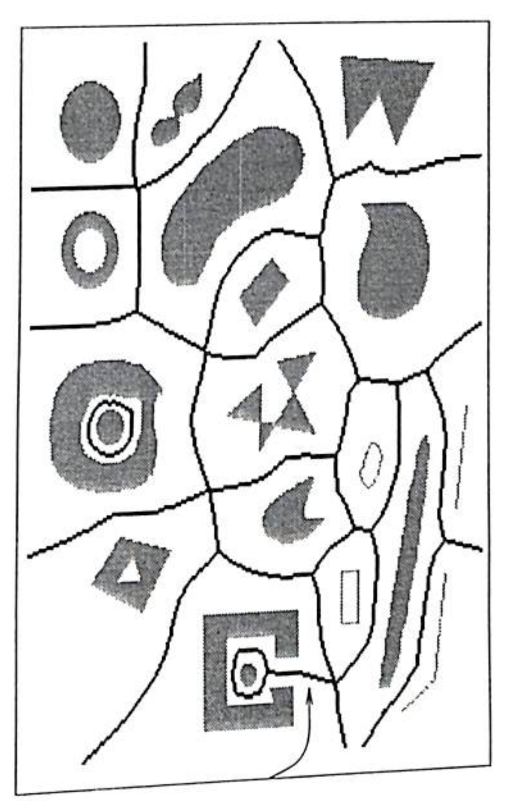
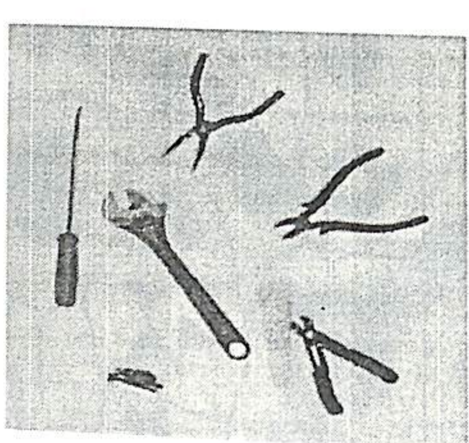


Figure 6.10: Voronoi diagram of objects. The arrow points to the false branch.

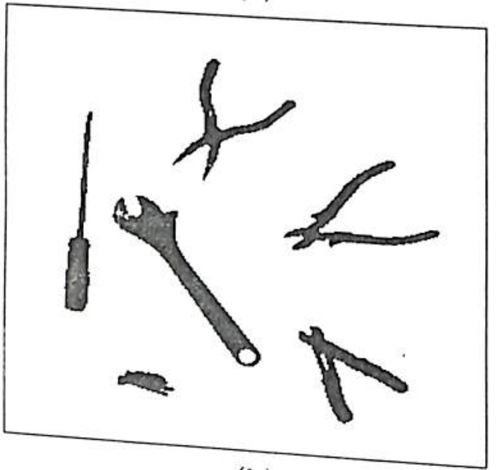
was obtained by taking a black and white photograph of some workshop tools and subsequently scanning the photo using an HP Scanjet 4C scanner. Figure 6.11(a) shows the image of size 260 × 300. This gray image is first sharpened and then binarized with the threshold set at a gray value of 155. The threshold is chosen in such a way that the objects in the image can be distinguished from the background. The binarized image is shown in Figure 6.11(b). The computed Voronoi diagram of objects is shown in Figure 6.11(c). The Voronoi diagram has been constructed in 180 iterations while total dilation has occurred after 95 iterations. The comparison of the proposed method with the existing methods in the literature is studied next.

6.6 Comparison studies

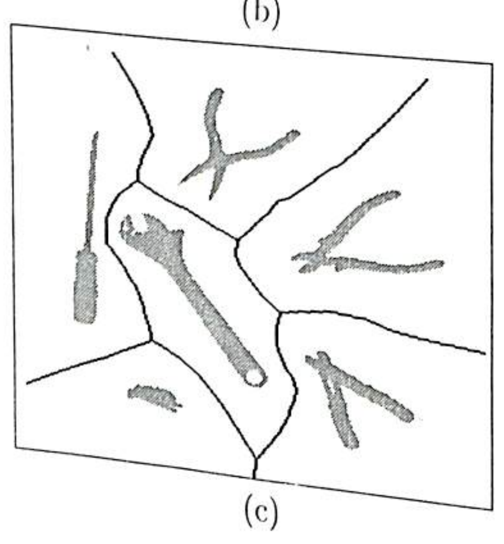
In this section, the proposed method of construction of discrete Voronoi diagram of objects and its implementation in cellular architecture are compared with existing methods and VLSI architectures.



(a)



(b)



(c)

Figure 6.11: (a) Real image. (b) Binarized image. (c) Voronoi diagram of (b) constructed by our method is given by the thick segments.

TH-2715_964102

A sequential algorithm for computing discrete Voronoi diagram of arbitrary shaped objects is given in [AdB86]. The diagram is based on city-block distance metric. The algorithm first finds the exoskeleton (skeleton of the background) from the city-block distance transform of the background and successively obtains the Voronoi diagram by removing certain pixels from the exoskeleton. This takes only a constant number of scans over the image. The Voronoi diagram obtained by this algorithm is shown in the Figure 6.13(a). When compared to Figure 6.9(b), the Voronoi diagram is not smooth due to the characteristics of city-block distance. The same algorithm is not suitable to compute Voronoi diagram from the Euclidean distance transform. The algorithm performs only local neighborhood operations. But these operations can not be performed in parallel for all pixels in order to make the algorithm suitable for efficient VLSI implementation on a cellular architecture.

A parallel algorithm for the construction of discrete Voronoi diagram and its VLSI implementation in a 2-D cellular architecture is presented in [TTT97] for path planning of a translating diamond shaped robot. The cellular architecture is simple in design, but the discrete Voronoi diagram constructed here is different from the one defined in Section 6.1. Also it is based on city-block distance. The authors in [TTT97] have constructed the Voronoi diagram as follows. Initially, the boundaries of the objects are identified and coded with respect to their orientation. These codes are then loaded as the initial source values onto the cellular architecture. The cellular architecture starts its evolution in time and each cell propagates its code to the four neighboring cells (two horizontal and two vertical neighbors) at each time step. The Voronoi diagram consists of those cells which receive different codes from their neighbors. The algorithm actually constructs Voronoi diagram of edges (horizontal, vertical and diagonal segments) of the boundaries of objects instead of objects itself. The Voronoi diagram has additional branches depending on the number of edges. For objects, whose boundary has fewer edges, the number of additional branches are tolerable. The authors in [TTT97] considered only simple images of size 32×32 to demonstrate the operation of their algorithm. Figure 6.12 reproduces the Voronoi diagram of an image considered in [TTT97]. But in general, the images are larger in size and the shape of objects in the images is complex. For example, the Voronoi diagram constructed by the algorithm described in [TTT97] of a larger image is shown in Figure 6.13(b). It has intolerable number of additional branches and finding a path for a robot in this Voronoi diagram is in general difficult.

Our method constructs the discrete Voronoi diagram of objects based on Euclidean

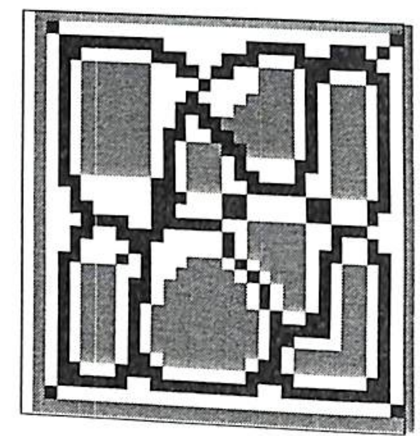


Figure 6.12: Voronoi diagram constructed in [TTT97] for a 32×32 image. Each pixel in the image has a square shape due to the magnification of the image. Black squares are Voronoi pixels.

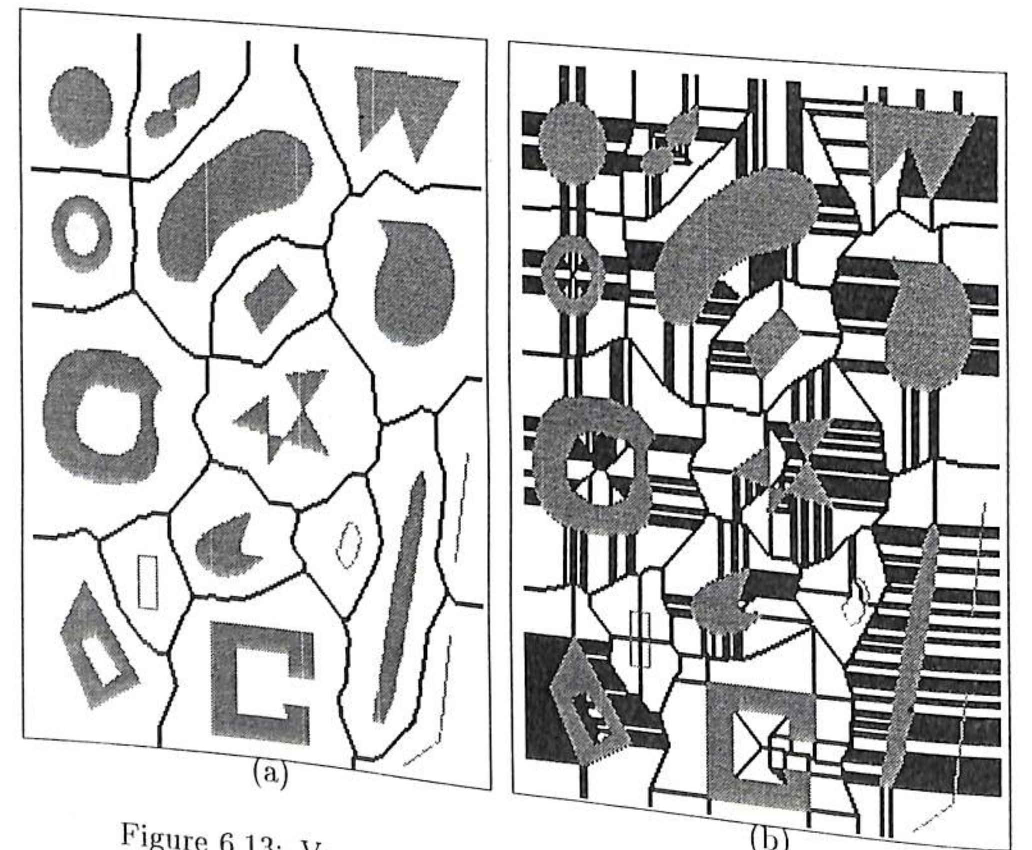


Figure 6.13: Voronoi diagram of objects. (a) Result of the algorithm in [AdB86]. (b) Result of the algorithm in [TTT97].

distance. The Voronoi diagram with its Euclidean distance values is useful for collision-free path planning of an arbitrary shaped robot among arbitrary shaped obstacles on a two dimensional rectangular grid. The motion of the robot can be both translation and rotation. The Voronoi diagram has no additional branches except for a special case as in Figure 6.10. In robot path planning, the additional branch occurring in the special case is advantageous. This is discussed in detail in Chapter 8. The method is also efficient to implement in a 2-D cellular architecture.

6.7 Conclusions

In this chapter, we have given a parallel method for the construction of Voronoi diagram of objects on a binary image. The diagram is based on the Euclidean distance metric and the objects can have arbitrary shape. The proposed method constructs the Voronoi diagram exactly for most images except for a special configuration of objects (an object is enclosed by another but not completely) in which an additional branch arises. A VLSI implementation of the algorithm for the construction in a cellular architecture has been described. A comparison of the proposed method with some existing methods has also been given.

Some applications of Voronoi diagram are discussed in Chapter 8. The next chapter considers the computation of Hausdorff distance.

Figure 6.12: Voronoi diagram constructed in [TTT97] for a 32×32 image. Each pixel in the image has a square shape due to the magnification of the image. Black squares are Voronoi pixels.

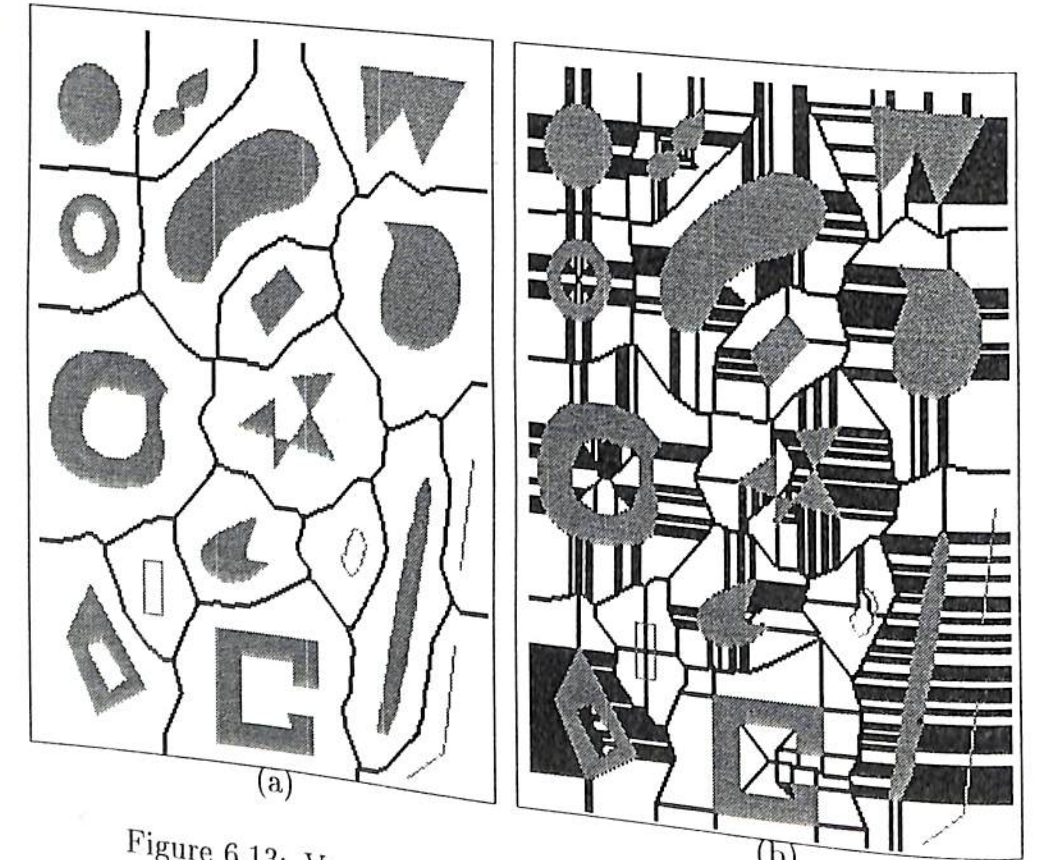


Figure 6.13: Voronoi diagram of objects. (a) Result of the algorithm in [AdB86]. (b) Result of the algorithm in [TTT97].

TH-2715_964102

distance. The Voronoi diagram with its Euclidean distance values is useful for collision-free path planning of an arbitrary shaped robot among arbitrary shaped obstacles on a two dimensional rectangular grid. The motion of the robot can be both translation and rotation. The Voronoi diagram has no additional branches except for a special case as in Figure 6.10. In robot path planning, the additional branch occurring in the special case is advantageous. This is discussed in detail in Chapter 8. The method is also efficient to implement in a 2-D cellular architecture.

6.7 Conclusions

In this chapter, we have given a parallel method for the construction of Voronoi diagram of objects on a binary image. The diagram is based on the Euclidean distance metric and the objects can have arbitrary shape. The proposed method constructs the Voronoi diagram exactly for most images except for a special configuration of objects (an object is enclosed by another but not completely) in which an additional branch arises. A VLSI implementation of the algorithm for the construction in a cellular architecture has been described. A comparison of the proposed method with some existing methods has also been given.

Some applications of Voronoi diagram are discussed in Chapter 8. The next chapter considers the computation of Hausdorff distance.

Chapter 7

Computation of the Hausdorff Distance

7.1 Introduction

There are many practical task domains such as automated industrial assembly, document processing applications, landmark recognition for navigation etc., where one encounters the problem of *model-based recognition*. The problem involves locating an object, for which the computer has a model, in an image. The Hausdorff distance is very useful to solve problems such as these. Its computation is the focus of this chapter.

The Hausdorff distance between two images is computed by considering the foreground pixels of two images as point sets in the integer space. Consider binary images consisting of foreground (1-pixels) and background (0-pixels). As mentioned earlier, the Hausdorff distance between two images A and B of same size with foregrounds F_A and F_B is:

$$H(A,B) = \max\{h(A,B), h(B,A)\}$$

where

$$h(A,B) = \max_{p_A \in F_A} d_e(p_A, F_B); \quad h(B,A) = \max_{p_B \in F_B} d_e(p_B, F_A);$$

$h(A,B)$ is the directed Hausdorff distance from A to B and $d_e(p_A, F_B)$ is the Euclidean distance between pixel p_A of A and the foreground F_B of B. The Hausdorff distance gives

TH-2715_964102

the degree of mismatch between images. Prior work on the computation of Hausdorff distance has been discussed in Chapter 3. The Hausdorff distance has been applied for image matching [HKR93, Ruc95]. A VLSI architecture for computing Hausdorff distance is useful for real-time image matching.

In this chapter, we propose a method suitable for VLSI implementation for finding the Hausdorff distance between two binary images. The method is based on dilating the images and uses the Euclidean distance metric. The usefulness of Hausdorff distance for image matching is also explored. The following section describes the method.

7.2 Method of Computation

The iterative method of finding equidistant contours for the computation of EDT (Section 4.2) using local neighborhood operations can be applied to compute the nearest integer approximations to $h(A,B)$ and $h(B,A)$, say $h_i(A,B)$ and $h_i(B,A)$. To compute $h_i(A,B)$, we iteratively dilate F_B based on Euclidean distance, by one pixel unit at each iteration. The boundary of the dilated F_B at some iteration k consists of those pixels in image B whose integer Euclidean distance is k. Let us assume that image B is placed over image A. The boundary then covers those pixels $p_A \in F_A$ whose integer approximation to $d_e(p_A, F_B)$, say $d(p_A, F_B)$, is k. In other words, we identify those pixels in F_A whose integer approximation to $d_e(p_A, F_B)$ is k. The dilation of F_B is stopped when the dilated F_B fully covers F_A. The number of iterations taken for dilation gives $h_i(A,B)$. The iterative process of dilation based on Euclidean distance requires the computation of distance related quantities $\Delta x, \Delta y$ and d_f for each pixel p_0 in an image using the neighborhood $N_E(p_0)$ as in Figure 4.2. The equations for their computation are reproduced below.

$$d_f^{(k+1)}(p_0) = 2(k+1) + \max_{p_i \in N_E(p_0)} [d_f^{(k)}(p_i) - (\Delta X_i + \Delta Y_i)] \quad (7.1)$$

where

$$\Delta X_i = \begin{cases} 0 & i = 0, 1\&5 \\ 2|\Delta x^{(k)}(p_i)| + 1 & i = 2,3,4,6,7\&8 \end{cases}$$

TH-2715_964102

$$\Delta Y_i = \begin{cases} 0 & i = 0, 3\&7 \\ 2|\Delta y^{(k)}(p_i)| + 1 & i = 1,2,4,5,6\&8 \end{cases}$$

and

$$(|\Delta x^{(k+1)}(p_0)|, |\Delta y^{(k+1)}(p_0)|) = (\Delta x_i, \Delta y_i) \quad (7.2)$$

where

$$\Delta x_i = \begin{cases} |\Delta x^{(k)}(p_i)| & i = 0, 1\&5 \\ |\Delta x^{(k)}(p_i)| + 1 & i = 2,3,4,6,7\&8 \end{cases}$$

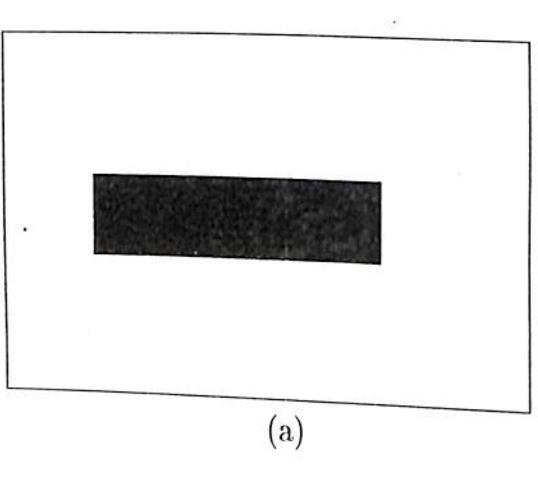
$$\Delta y_i = \begin{cases} |\Delta y^{(k)}(p_i)| & i = 0, 3\&7 \\ |\Delta y^{(k)}(p_i)| + 1 & i = 1,2,4,5,6\&8 \end{cases}$$

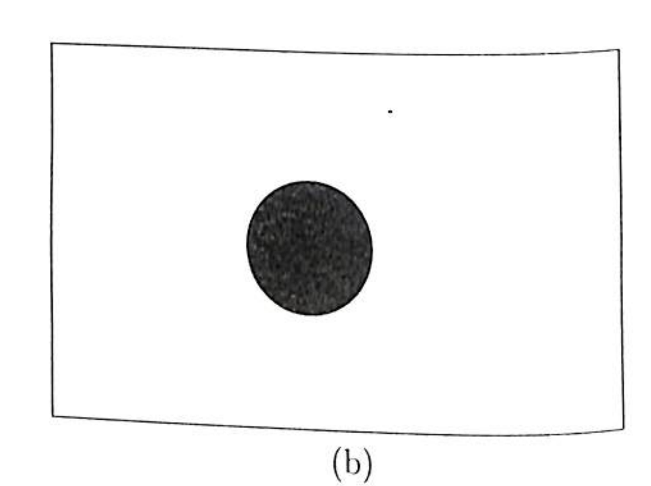
and i in Equation 7.2 corresponds to pixel p_i that satisfies Equation 7.1. Similar to the above procedure, $h_i(B,A)$ is obtained by dilating F_A. The maximum of these two integers gives the nearest integer approximation to $H(A,B)$, say $H_i(A,B)$. The error in the approximation is $\pm$ 0.5 pixel unit. Let us illustrate the method with an example.

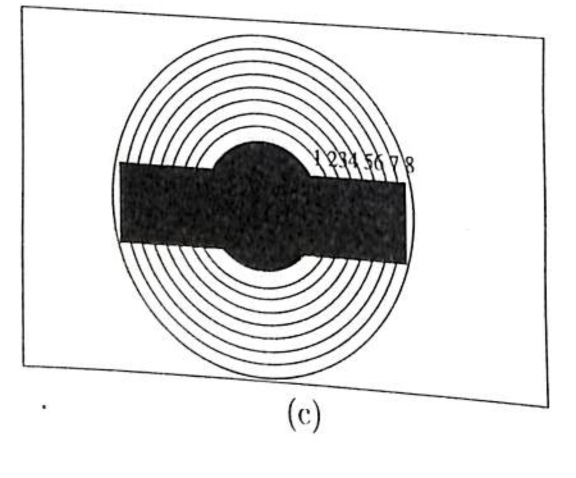
7.2.1 Illustration

Consider two images A and B shown in Figures 7.1 (a) and (b). The image A has a rectangular object and the image B has a circular object. The objects belong to the foregrounds of images. The computation of directed Hausdorff distances $h_i(A,B)$ and $h_i(B,A)$ is illustrated in Figures 7.1 (c) and (d). To compute $h_i(A,B)$, the circular object in image B is dilated iteratively until the dilated object covers the rectangular object when B is placed over A. Figure 7.1 (c) shows the superposition of images and the boundaries (concentric circles) of the dilated circular object at different instants. If we assume that these instants correspond to iterations $k = 1, 2, 3, \ldots$, then it takes eight iterations for the dilated circular object to cover the rectangular object and $h_i(A,B) = 8$. However, the actual number of iterations depends on the size of the objects. Similarly $h_i(B,A)$ is computed by dilating the rectangular object until the dilated object covers the circular object. This is shown in Figure 7.1 (d). $h_i(B,A) = 3$ since dilation has been carried out in three iterations. $H_i(A,B)$ is the maximum of $h_i(A,B)$ and $h_i(B,A)$.

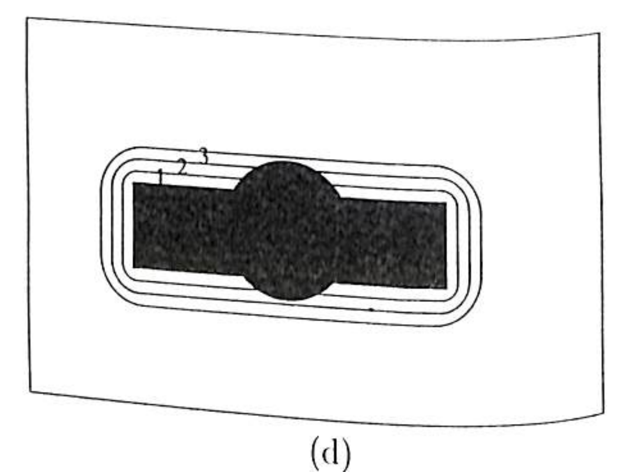
The above method of computation can be carried out in parallel. The method is formulated as an algorithm discussed next.


(a)


(b)



(c)



(d)

Figure 7.1: Illustration of directed Hausdorff distances between images. (a) Image A. (b) Image B. (c) Computation of $h_i(A, B)$. (d) Computation of $h_i(B, A)$.

TH-2715_964102

7.3 Algorithm

Given two binary images A and B, the algorithm first finds the integer approximation to the directed Hausdorff distance $h(A, B)$ from A to B, *i.e.*, $h_i(A, B)$, by dilating B. Let p represent a pixel at row x and column y. Let $b_A(p)$ and $b_B(p)$ represent the binary values of pixels at row x and column y in images A and B. The pseudocode of the algorithm for the computation of $h_i(A, B)$ is as follows.

ALGORITHM *HD_AtoB*

Inputs: Images A and B.
Output: $h_i(A, B)$.

Step 1: Initialization
The initialization of $b_A(p), b_B(p), \Delta x(p), \Delta y(p)$ and $d_f(p)$ are carried out with respect to images A and B.

$$\Delta x^{(0)}(p) = \Delta y^{(0)}(p) = d_f^{(0)}(p) = 0$$

$$b_A(p) = \begin{cases} 1 & p \in F_A \\ 0 & p \in B_A \end{cases}$$

$$b_B(p) = \begin{cases} 1 & p \in F_B \\ 0 & p \in B_B \end{cases}$$

Step 2: Iterative process
The dilation of image B is carried out iteratively by setting $b_B(p)$ to 1 for all pixels whose integer Euclidean distance equals the iteration number k. The $b_A(p)$ of these pixels are simultaneously set to 0. The dilation is stopped when $b_A(p) = 0$ for all pixels.

$k = 0$
repeat iteration
  for all pixels p **do in parallel**

switch $(b_B(p))$
1: { pixel p is in dilated F_B }
  Update $d_f(p)$ by adding $2(k+1)$
  $b_A(p)$=0 { $b_A(p)$ is set to 0 since $p \in F_B$ }
  break { switch $b_B(p) = 1$ }

0: { check whether p belongs to dilated boundary of F_B }
  if there exists $p_I \in N_E(p)$ such that $b_B(p_I) = 1$ **do**
    Compute $d_f^{(k+1)}(p)$ as in Equation 7.1

    if $(d_f^{(k+1)}(p) \geq 0)$ **do**
      Compute $(\Delta x^{(k+1)}(p), \Delta y^{(k+1)}(p))$ as in Equation 7.2
      $b_B(p) = 1$
      $b_A(p) = 0$
    end if
  end if
  break { switch $b_B(p) = 0$ }

end for
$k = k + 1$
until $b_A(p) = 0$ for all pixels

In the above iterative process, when $b_A(p) = 0$ for all pixels, the dilated F_B fully covers F_A. The value of k gives $h_i(A, B)$. $h_i(B, A)$ is computed similarly. $H_i(A, B)$ is the maximum of these two. The time and space complexities of the algorithm are as follows.

7.3.1 Complexity Analysis

The time and space complexities are similar to the computation of EDT. Since the dilation is based on Euclidean distance, for an $n \times n$ image, the dilation distance cannot be greater than $\sqrt{2}n$. Hence $O(n)$ iterations are required for dilation. The space complexity of the algorithm is $O(n^2)$ since the algorithm needs constant space for storing $\Delta x, \Delta y, d_f, b_A$ and b_B for every pixel. In the following section, we discuss the usefulness of Hausdorff distance for image matching.

TH-2715_964102

7.4 Image Matching

The Hausdorff distance between two images measures the degree of mismatch between images. Two images have a good match if the Hausdorff distance between them is within a tolerable limit. This limit allows some perturbation between images. If the Hausdorff distance is 0, then the two images coincide. The following are some applications of image matching.

In industrial robotics, an object to be picked by a robot has to be first located in the environment [Sch90]. The directed Hausdorff distance is used to locate a given model in a test image. The directed Hausdorff distance from the model to a portion of the test image, same as the model size, gives how much the model matches with the pattern in that portion.

In many machine vision and pattern recognition applications, one needs to identify instances of a model that are only partly visible (either due to occlusion or due to failure of the sensing device to detect the entire object). The Hausdorff distance can be extended for the comparison of portions of two shapes. Also, in the case of two point sets which differ by just one point, the classical Hausdorff distance will move away from 0. In these cases, partial matching is quite meaningful.

7.4.1 Partial Image Matching

Partial image matching using Hausdorff distance has been attempted in [HKR93]. The authors of [HKR93] have done image matching by computing partial Hausdorff distance between images A and B, $H_p(A, B)$, and comparing it with a distance limit. The details are as follows. Let f_A and f_B be two fractional values between 0 and 1. These values are given as input. Let N_A and N_B be the number of foreground pixels in images A and B. For the given fractions f_A and f_B, find $K = \lfloor f_A N_A \rfloor$ and $L = \lfloor f_B N_B \rfloor$. Let the foreground pixels of A be ranked by their distance to their nearest foreground pixel in B and similarly the foreground pixels of B be ranked. The partial Hausdorff distance from A to B, $h_p(A, B)$, is the distance of the Kth ranked pixel in A. Similarly, $h_p(B, A)$ is the distance of the Lth ranked pixel in B. The partial Hausdorff distance $H_p(A, B)$ is $\max[h_p(A, B), h_p(B, A)]$. For a given distance limit d_L, if $H_p(A, B) \leq d_L$, then the images match each other partially.

The above procedure of partial image matching needs the computation of distance values for all pixels. We have modified this procedure in such a way that it is suitable for parallel implementation. The modified procedure uses the computational method proposed in Section 7.2. It does not require the computation of distance values for all pixels. The inputs to the procedure are same as above. The inputs are images A and B, fractions f_A and f_B and a distance limit d_L. The foregrounds of the images are dilated to a distance d_L. That is, dilation is carried out for d_L iterations. Let n'_A be the number of pixels in F_A that are covered by the dilated F_B and similarly n'_B be. We compute two fractions f'_A and f'_B as follows. They are given by n'_A/N_A and n'_B/N_B respectively. These fractions are then compared with the input fractions f_A and f_B. If $f'_A \geq f_A$ and $f'_B \geq f_B$, then the images are partially matched. In our studies, we input only one fraction f instead of f_A and f_B. We compute $f'(A,B) = \min\{f'_A, f'_B\}$. From the computation of $f'(A,B)$, it can be seen that $f'(A,B)$ gives the amount of matching between images. If $f'(A,B) \geq f$, then the images are partially matched. Larger the value of $f'(A,B)$, better would be the matching.

Remark 3 *$H_i(A,B)$ gives the degree of mismatch between images A and B while $f'(A,B)$ gives the degree of matching between images.*

In the case of partial image recognition, one would like to find the best match for a given image I with a set of templates $T_1, T_2, \ldots, T_N$. Here, we find the fractions $f'(.,.)$ giving the extent of match between each template and the image I, *i.e.*, $f'(I,T_i)$, $1 \leq i \leq N$. The best matching template T_b is the one whose $f'(I,T_b)$ is the maximum. For partial image recognition, we need not input the fraction f.

The procedure for partial image matching and recognition is given below.

Partial Image Matching

Step 1: Input images A and B, fraction f and dilation limit d_L.

Step 2: Dilate images for d_L iterations.

Step 3: Find n'_A and n'_B.
n'_A: number of pixels in F_A that are covered by dilated F_B.
n'_B: number of pixels in F_B that are covered by dilated F_A.

Step 4: Compute f'_A and f'_B.

TH-2715_964102

$f'_A = n'_A/N_A; \qquad f'_B = n'_B/N_B;$

Step 5: Compute $f'(A,B)$. $f'(A,B) = \min\{f'_A, f'_B\}$

Step 6: if $f'(A,B) \geq f$, images are partially matched.

Partial Image Recognition

Step 1: Input image I and templates T_i, $1 \leq i \leq N$.

Step 2: Compute $f'(I,T_i)$, $1 \leq i \leq N$.

Step 3: Find $\max_{1 \leq i \leq N} f'(I,T_i)$. The best matching template is the one corresponding to this maximum.

7.5 Simulation Results

For the simulation studies, we have taken character images of size 150×90. First, the images containing the characters **P** and **B** as shown in Figures 7.2(a) and (b) have been considered. The directed Hausdorff distances computed by the algorithm are h_i(**P**,**B**) = 4 and h_i(**B**,**P**) = 33 respectively. h_i(**P**,**B**) is very less when compared to h_i(**B**,**P**) because character **P** is almost a part of character **B**. The computation of h_i(**B**,**P**) for example is demonstrated in Figure 7.3. The results of dilation of image **P** at different iterations are shown in the figure. Also, the superposition of dilated images and image **P** are shown.

The performance of the partial image matching scheme has been evaluated using the images shown in Figure 7.2. We have first computed the classical Hausdorff distances by the proposed algorithm. The Hausdorff distances $H_i(.,.)$ between the partial character images and template character images in Figure 7.2 are given in Table 7.1 (a). In the case of $\mathbf{P}_{par}$, though minimum Hausdorff distance of 45 is attained with template **P**, the correct match, the Hausdorff distances $H(\mathbf{P}_{par}$,**B**) and $H(\mathbf{P}_{par}$,**A**) are very close to $H(\mathbf{P}_{par}$,**P**). Also the minimum distance value is quite large. In the case of $\mathbf{A}_{par}$, minimum is attained with **P**, which is not the correct match. Hence, the classical Hausdorff distances are not suitable for finding the exact match for partial images.

We have then considered the partial image matching described in the previous section. The distance limit d_L has been set to 10 and the fractions $f'(.,.)$ have been computed

Table 7.1: The Hausdorff distances H_i and fractions f' computed between the partial character images and template character images in Figure 7.2.

H	P	B	A
$\mathbf{P}_{par}$	45	47	48
$\mathbf{A}_{par}$	23	39	39

(a)

f'	P	B	A
$\mathbf{P}_{par}$	0.85	0.67	0.59
$\mathbf{A}_{par}$	0.72	0.60	0.87

(b)

between images. They are tabulated in Table 7.1 (b). In the case of $\mathbf{P}_{par}$, $f'(\mathbf{P}_{par},\mathbf{P})$ is maximum and in the case of $\mathbf{A}_{par}$, $f'(\mathbf{A}_{par},\mathbf{A})$ is maximum which implies that the partial characters match well with their corresponding templates.

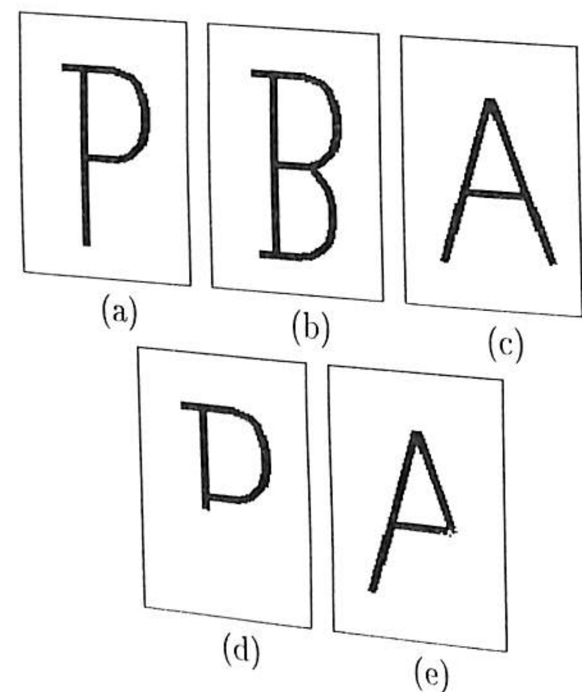


Figure 7.2: Character images. (a),(b),(c) Template characters. (d),(e) Partial characters $\mathbf{P}_{par}$ and $\mathbf{A}_{par}$ of **P** and **A**.

More discussions and details of experimental studies in character recognition are given in Chapter 8.

TH-2715_964102

7.6 VLSI Architecture

Similar to the architecture for computing EDT, the proposed algorithm can be implemented in an $n \times n$ array of cells to compute the directed Hausdorff distance $h_i(A, B)$ from an image A with size $n \times n$ to an image B with the same size. Each cell represents one pixel of the binary image. A cell is a sequential circuit consisting of storage elements b_A and b_B, besides $\Delta x, \Delta y$ and d_f. The size of $\Delta x, \Delta y$ and d_f depends on the size of the image and b_A and b_B are one bit in size. Each cell is connected to its eight neighboring cells and this neighborhood connectivity is similar to the one for Euclidean distance computation. The computation of $\Delta x, \Delta y$ and d_f can be implemented by combinational logic circuits and the setting of one bit storage elements b_A and b_B to 0 and 1 depends on d_f. The block diagram of a typical cell is shown in Figure 7.4.

Operation of Hardware

The b_A and b_B of cells are initialized first with respect to images A and B. That is, the b_A of a cell is loaded with 1 if the corresponding pixel belongs to F_A or with 0 if the pixel belongs to B_A. Similarly, b_B is initialized. All other storage elements are loaded with 0. Then the cellular array and the external counter are allowed to run synchronously with the external clock and the cells are updated at each clock pulse. The updation is stopped when the b_A of all cells are set to 0. This is checked by an external control logic using OR gates. The value of external counter gives the integer approximation of $h(A, B)$.

The above operation is repeated to compute $h_i(B, A)$ by initializing b_A with image B and b_B with image A.

7.6.1 Design of a Cell

The design of a cell for computing $h_i(A, B)$ is similar to the basic cell (described in Section 4.5.1) with an additional flip-flop for storing b_A. The flip-flop that stores the flag *done* in the basic cell is used to store b_B. The input to the flip-flop b_A is logic 0 and the clock to the flip-flop is activated when $b_B = 1$ or when $cndn = 1$ ($cndn$ is defined in Section 4.5.1). The clock is same as the one given to register d_f.

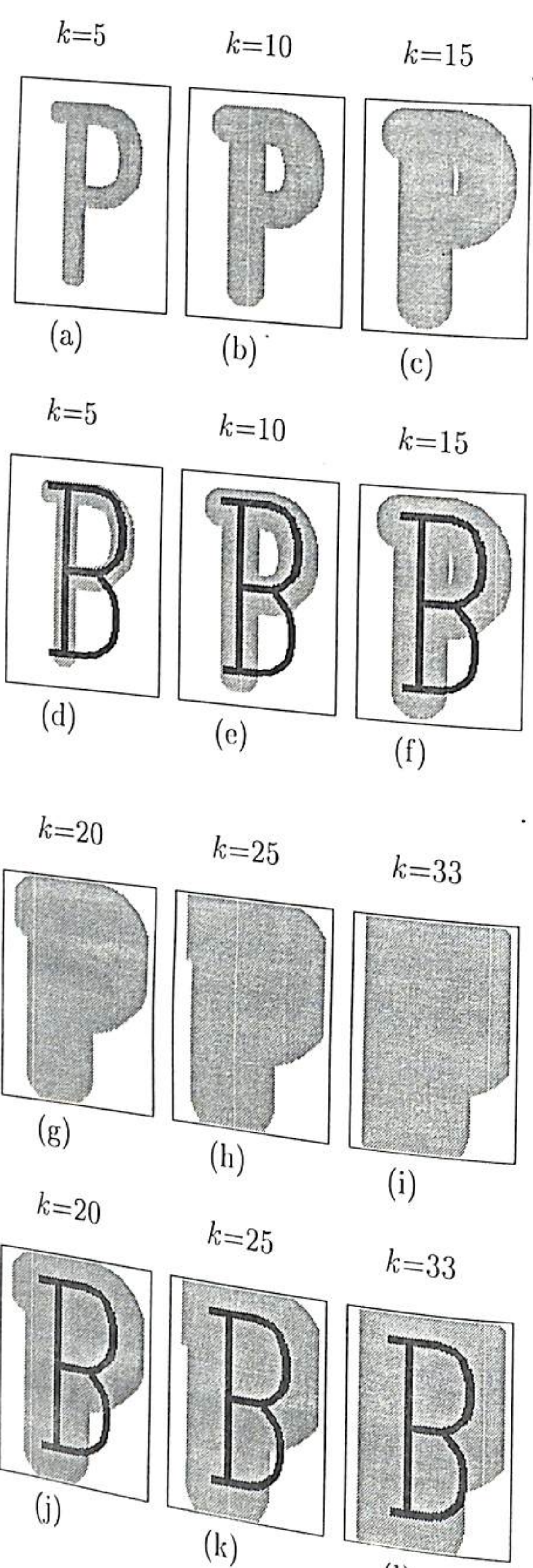


Figure 7.3: Computation of $h_i(\mathbf{B},\mathbf{P})$. (a),(b),(c),(g),(h) and (i) Dilated versions of image P at different iterations k. (d),(e),(f),(j),(k) and (l) Superposition of dilated versions on image B.

TH-2715_964102

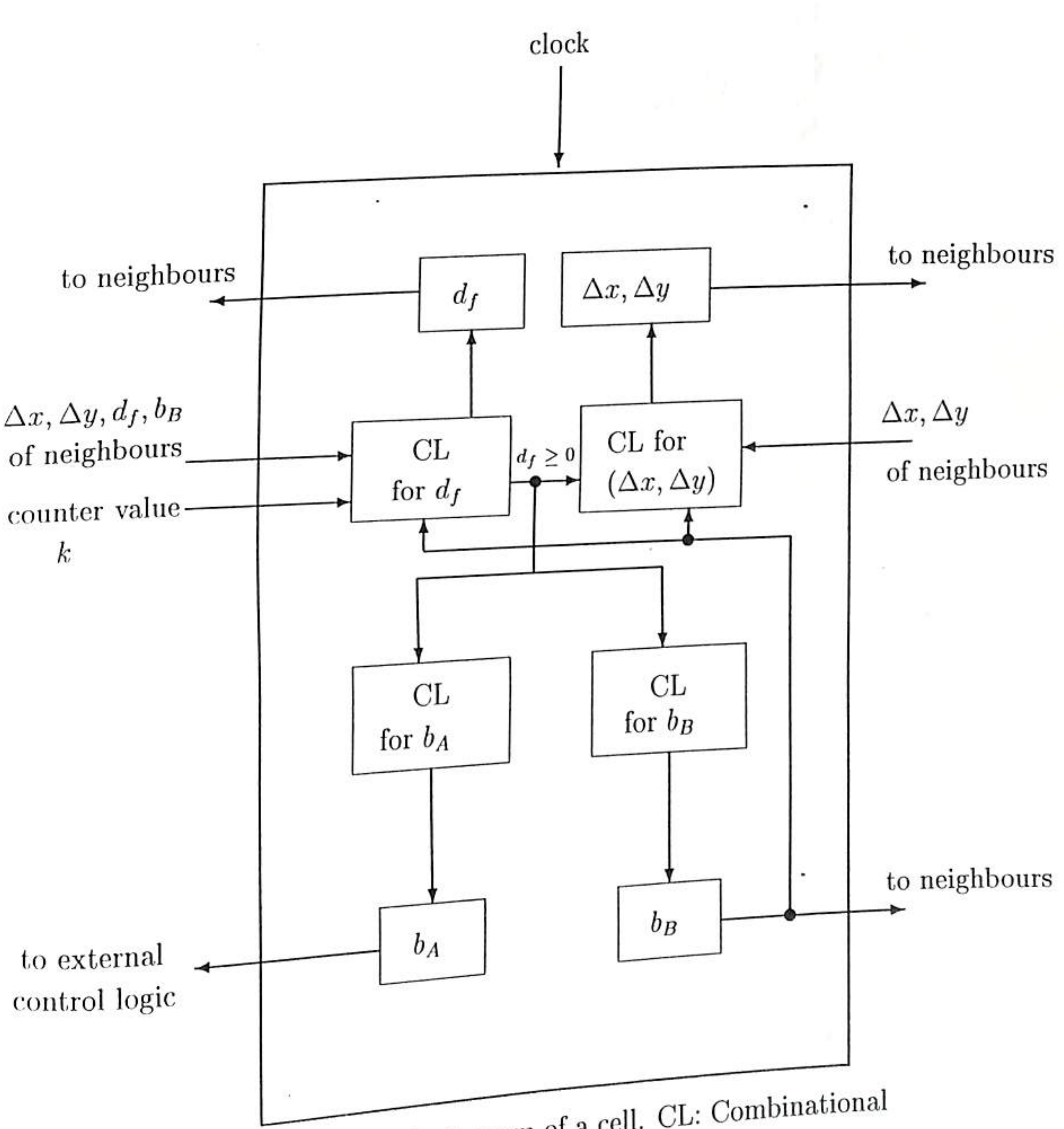


Figure 7.4: Block diagram of a cell. CL: Combinational Logic.

Table 7.2: Number of different Xilinx FPGA logic components of chips obtained from the implementation of the design of cell for computing $h(A, B)$.

Image size $N \times N$	BUFG	DFF	FMAP	HMAP	IBUF	OBUF	OUTFF
16×16	2	9	424	74	142	9	9
32×32	2	10	516	98	167	10	11
64×64	2	11	616	80	192	11	13
128×128	2	12	658	122	217	12	15
256×256	2	13	664	121	242	13	17

BUFG: Global Buffer; DFF: D type flip-flop; FMAP,HMAP: Mapping design using F and H look-up tables; IBUF: Input Buffer; OBUF: Output Buffer; OUTFF: Output flip-flop

Table 7.3: Results of placement and routing of components in Table 7.3

Image size $N \times N$	Clock (MHz)	CL delay (ns)	Net delay (ns)	CLB	IOB	Gate count
16×16	6.8	155.85	13.22	246	160	2979
32×32	5.9	190.77	15.55	301	188	3657
64×64	5.78	181.52	14.18	344	216	4194
128×128	4.81	228.04	18.35	380	244	4671
256×256	5.09	208.67	21.5	384	272	4684

C path: Combinational path; CLB: Complex Logic Block; IOB: I/O Block

The design has been implemented in Xilinx FPGA. The number of different logic components of the chips obtained from implementation for different image sizes are tabulated in Table 7.2. The interpretation of results is similar to the interpretation provided for Table 4.2. An additional OUTFF is due to b_A. The output of b_A is taken directly as the output of the cell without giving to combinational logic of cell (see Figure 7.4) and hence OUTFF is used for b_A. After mapping, placement and routing have been carried out and the results of placement and routing are given in Table 7.3. The layout of the chip corresponding to 16 × 16 size of an image is shown in Figure 7.5. Finally, a cellular architecture has been designed in VHDL and tested with different images in ModelSim.

TH-2715_964102

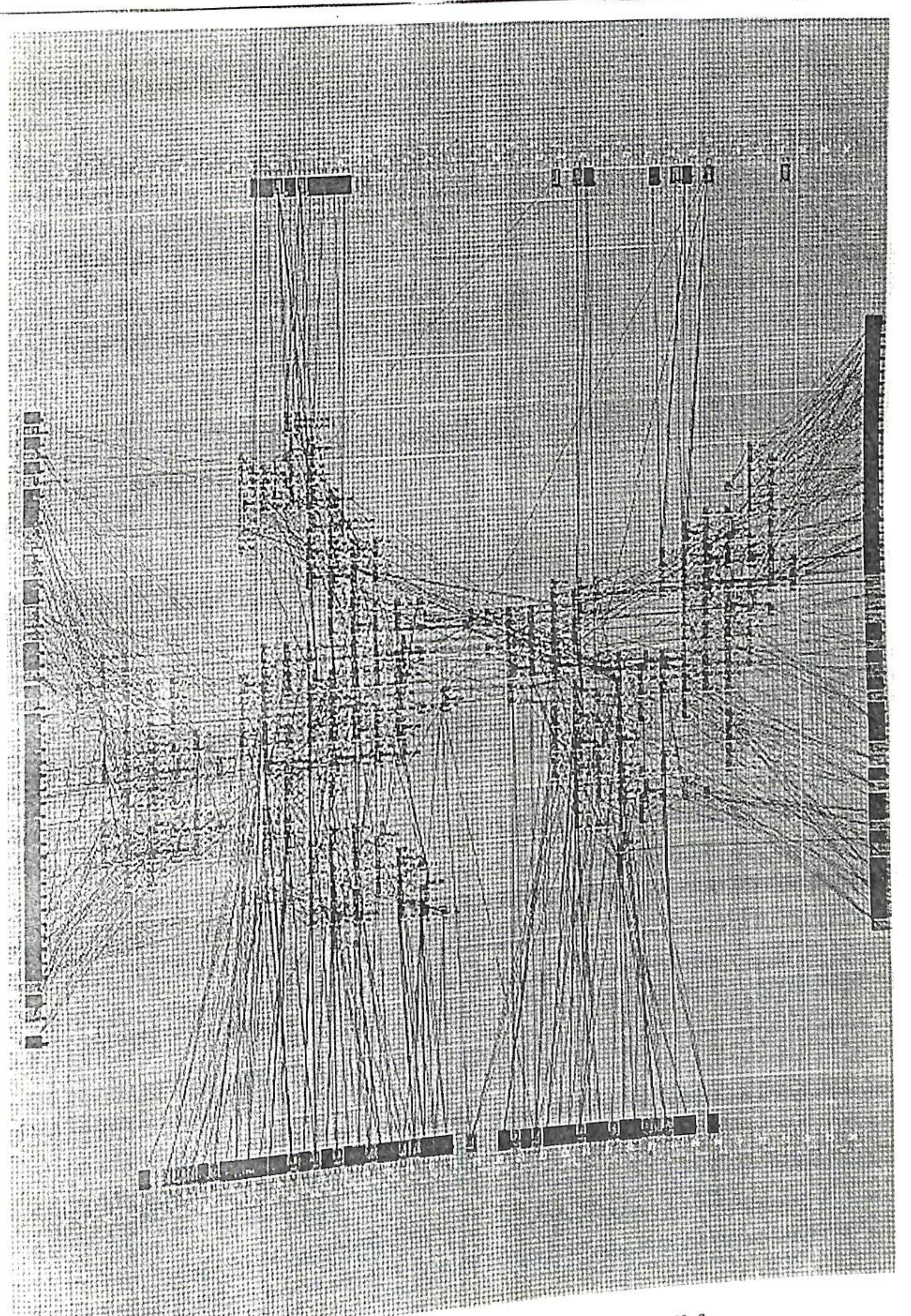


Figure 7.5: Layout of the chip for the design of cell for computing directed Hausdorff distance between two 16 × 16 binary images.

7.7 Conclusions

In this chapter, we have given a parallel algorithm to compute an integer approximation to Hausdorff distance between two binary images. The approximation error is between -0.5 to +0.5. Besides, we have explored the use of Hausdorff distance for partial image matching. The performance of Hausdorff distance for image matching is demonstrated on character images. The proposed algorithm can be implemented in VLSI on a locally connected cellular architecture and the VLSI architecture has been described. The applications of Hausdorff distance for optical character recognition are discussed in detail in the following chapter along with the applications of skeleton and Voronoi diagram.

TH-2715_964102

Chapter 8

Applications of Skeleton, Voronoi Diagram and Hausdorff Distance

8.1 Introduction

The skeleton, Voronoi diagram and Hausdorff distance are important tools in machine vision. In *machine vision*, images are grabbed and then processed. Machine vision applications do not involve a human being in the visual loop. In other words, the images are examined and acted upon by the machines. Although humans are involved in the development of the system, the final application requires a machine to use the visual information directly. Machine vision applications involve tasks that either are tedious for people to perform, require work in a hostile environment, require a high rate of processing, or require access and use of a large database of information. Machine vision helps in industrial automation. With a pressing need for increased productivity and delivery of end products of uniform quality, manufacturing industry is turning more and more towards computer-based automation using robots. An industrial robot in an assembly unit is generally a computer-controlled manipulator consisting of an arm and a wrist plus a tool. It is designed to reach a workpiece located within its workspace. The workspace is the space of influence of a robot whose arm can take the wrist with the tool to any point in the space. To reach a particular workpiece, it is necessary to control the manipulator to follow a desired path. Before moving the robot arm, it is of interest to know whether there are any obstacles (can be other workpieces) present in

the path that the robot arm has to traverse and whether the manipulator hand needs to traverse along a specific path. Machine vision can be applied to find a collision-free path for the robot. One of the major topics within the field of machine vision is image analysis.

Image analysis involves the examination of the image data to facilitate solving a vision problem. The image analysis process involves two other topics: feature extraction and pattern classification. *Feature extraction* is the process of acquiring higher level image information, such as shape or color information, and *pattern classification* is the act of taking this higher level information and identifying objects within the image.

The skeleton of an object represents its shape and can be used to identify the object in an image. The Voronoi diagram of objects can be used to find a collision-free path for a robot. The discrete Voronoi diagram, constructed for different classes of feature points of patterns distributed in a 2-D integer space, is useful for classifying a new pattern. The Hausdorff distance measures the degree of mismatch between images and hence it can be used to locate a given model in an image. In an assembly unit, the workpiece to be picked by the robot is first identified in the workspace by using skeleton or Hausdorff distance for images of workpiece and workspace and a path for the robot is then planned using the Voronoi diagram for the image of workspace. When performing image matching for identifying objects, feature level matching is preferable compared to pixel level matching to reduce the time for matching process. The skeleton is one such feature where matching can be done in compressed form. While exploring the machine vision applications, we also observed another potential application of Voronoi diagram, *i.e.*, image compression and secure transmission. The use of Hausdorff distance for image matching can be extended to optical character recognition. This chapter explores all possible applications in which the proposed methods for the computation of skeleton, Voronoi diagram and Hausdorff distance are useful. Due to parallelism of the methods, high speed of operation can be achieved and the problems can be solved in real-time.

The applications of skeleton are discussed first.

TH-2715_964102

8.2 Applications of the Skeleton

The skeleton is a linear structure. It describes the shape of objects in an image. Skeleton is useful for identifying an object of a particular shape in an image. Since the skeleton of an image consists of very few pixels and the image can be reconstructed from the skeleton, storing skeleton achieves compression. Further, some image processing operations such as geometric transformations and morphological operations can be performed in the compressed form. Such a processing takes less time and space.

8.2.1 Shape Analysis

The shape descriptors such as number of end pixels, branches and loops in the skeleton give the topology of the objects and can be measured to obtain the shape information. The end pixels with distance value of 1 correspond to the corners in the boundary of an object. The skeleton of polygonal object has all its end pixels with distance value equal to 1. The loops in the skeleton are due to holes in the object. The width of the object can be determined by the maximum distance value and the length is determined by the number of skeleton pixels along the longest skeleton branch. The skeleton of linear object is also linear without branches. In applications like character recognition and pattern matching, the shape descriptors are quite useful. An object of a given shape can be identified (invariant to size and orientation) in an image by measuring the shape descriptors of each connected component in the image. The linear structure of skeleton makes it useful for image compression.

8.2.2 Image Compression

The skeleton has the following properties which are useful for compressing images. (1) The skeleton of an image consists of fewer foreground pixels than the image. (2) The skeleton with its distance values can reconstruct the image. Hence, a given binary image can be compressed by storing the skeleton pixels' coordinates and distance values. The coordinate of a pixel is given by the row x and column y of the pixel's location. The distance value d is the integer approximation to Euclidean distance. We can avoid the repetition in storing x for skeleton pixels belonging to the same row by storing x

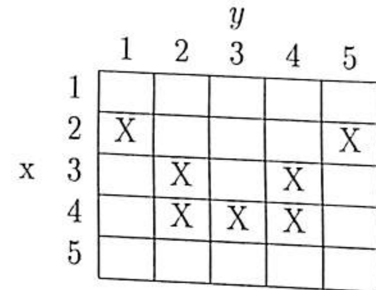


Figure 8.1: An example 5 × 5 skeleton image. Skeleton pixels are marked by 'X'.

first and then only the y and d of skeleton pixels of row x. Each row is ended with a delimiter 0. For example, the skeleton image shown in Figure 8.1 can be represented by the sequence of numbers 2,1,$d_{2,1}$,5,$d_{2,5}$,0,3,2,$d_{3,2}$,4,$d_{3,4}$,0,4,2, $d_{4,2}$,3,$d_{4,3}$,4,$d_{4,4}$,0 where $d_{x,y}$ is the distance value of the pixel at row x and column y. For an image of size $M \times N$, the number of bits needed to store x,y and $d_{x,y}$ are as follows. x is stored in $\lceil \log_2 M \rceil$ bits, y and 0 are stored in $\lceil \log_2 N \rceil$ bits and $d_{x,y}$ is stored in $\lceil \log_2 d_{max} \rceil$ bits where d_{max} is the maximum possible distance value of skeleton pixels. which is $\lceil \sqrt{M^2 + N^2} \rceil$. If M,N and d_{max} are powers of 2, then the number of bits needed to store x,y,$d_{x,y}$ and 0 are increased by 1. For the above example, the number of bits needed to store x,y and 0 are $\lceil \log_2 5 \rceil$ bits each and $d_{i,j}$ is stored in $\lceil \log_2 7 \rceil$ bits. The size of compressed file depends on the dimension of the image, number of skeleton pixels and the maximum distance value. The compression ratio (CR) is given by

$$CR = \frac{\text{Image size } (= M * N)}{\text{Compressed file size}}$$

We have conducted experiments to show the performance of skeleton based compression.

Experimental Studies

For experimentation, we have considered the images shown in Figure 8.2. The skeletons for these images are computed by Methods 1 and 2 described in Section 5.3 using 3 × 3 and 5 × 5 neighborhoods. The number of bits required to store these skeletons and the compression obtained are tabulated in Table 8.1.

The compressed file size for storing skeleton computed using 5×5 neighborhood is more than using 3×3 neighborhood. This is due to the increase in the thickness of the

TH-2715_964102

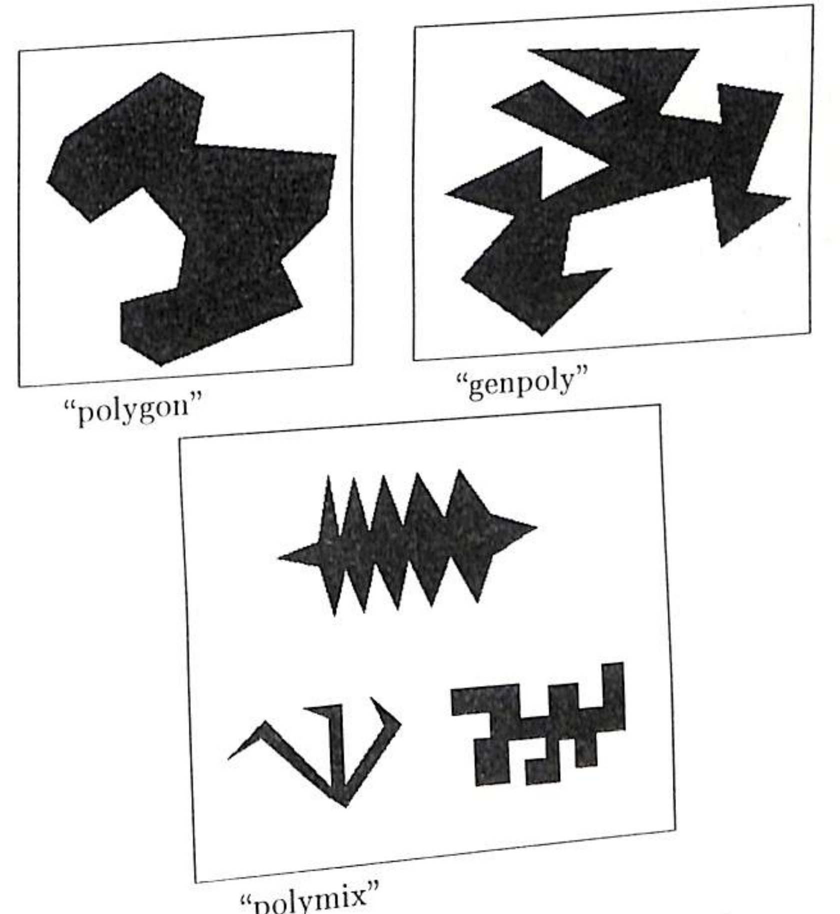


Figure 8.2: Images considered for skeleton based compression

Table 8.1: Results of skeleton based compression

Image	dimension	size in bits	3×3				5×5			
			Method 1		Method 2		Method 1		Method 2	
			size	CR	size	CR	size	CR	size	CR
polygon	243×220	53460	6720	7.95	12292	4.35	18606	2.87	27006	1.97
genpoly	211×228	48108	10387	4.6	15470	3.1	23179	2.07	28691	1.67
polymix	297×330	98010	21070	4.65	26152	3.74	40302	2.43	47446	2.06

skeleton with the increase in the neighborhood size. Also the compression achieved using Method 1 is better than using Method 2.

8.2.3 Skeleton based Processing

Storing skeleton not only achieves compression of binary image but also helps in performing some image processing operations in the compressed form. Processing in this compressed form takes less memory space and less time. The processed binary image can be reconstructed from the processed skeleton. We identified some of the image processing operations that can be carried out in the compressed form. They are as follows.

Geometric Transformations

The transformations like translation, rotation, size reduction and mirroring which do not distort the shape of objects in an image can be performed in the skeleton. We say the shape of objects is not distorted if the Euclidean distance between any two pixels p_1 and p_2 in the original binary image and the distance between the corresponding pixels $p_1{}^*$ and $p_2{}^*$ in the transformed image are proportional to each other, *i.e.*, dist(p_1,p_2)=c^*dist($p_1{}^*$,$p_2{}^*$) and c^* is a constant. Let (x,y) and d denote the coordinates and distance value of a skeleton pixel. Let $M \times N$ be the size of original image. Then the transformed skeleton pixel has coordinates (x^*,y^*) and distance value d^* as follows.

Translation

$$(x^*, y^*) = (x + d_x, y + d_y), \quad d^* = d$$

TH-2715_964102

where (d_x, d_y) is the translation vector.

Rotation

$$(x^*, y^*) = (x\cos\theta - y\sin\theta, x\sin\theta + y\cos\theta), \quad d^* = d$$

where θ is the angle of rotation. x^*,y^* and d^* can be approximated to the nearest integer.

Size reduction

$$(x^*, y^*) = (sx, sy), \quad d^* = sd$$

where s is the reduction factor. The values of x^*,y^* and d^* can be approximated to the nearest integer. Unlike reduction of size, magnification of size usually needs interpolation.

Mirroring

1. vertical mirroring

$$(x^*, y^*) = (x, N - y + 1), \quad d^* = d$$

2. horizontal mirroring

$$(x^*, y^*) = (M - x + 1, y), \quad d^* = d$$

Morphological Operations

The morphological operations like *erosion* and *dilation* (based on Euclidean distance) are possible in the skeleton itself. The erosion by d' pixel units is performed by decrementing the distance values of skeleton pixels by d' and eliminating those skeleton pixels whose resulting distance values are less than 0. Similarly dilation by d' pixel units is performed by incrementing the distance values by d'.

Other morphological operations like *opening* and *closing* are also possible in the skeleton itself. These operations are concatenation of both dilation and erosion. Opening is erosion followed by dilation while closing is dilation followed by erosion.

Similar to the skeleton, the Voronoi diagram is also a linear structure. But the Voronoi diagram is constructed on the background instead of objects.

8.3 Applications of the Voronoi Diagram

The Voronoi diagram is useful for robot path planning, pattern classification and compression of images. In robot path planning, Voronoi diagram of objects in a workspace of a robot can be used for planning a collision-free path for robot. In the problem of pattern classification, constructing the Voronoi regions of different known classes of patterns helps in identifying the class of new pattern. Image compression with security can be achieved by constructing Voronoi regions of randomly selected pixels in a given image.

8.3.1 Robot Path Planning

The problem of finding a collision-free path for a robot from its initial location to its final location in the presence of obstacles in an workspace is called robot path planning. One approach to this problem is using Voronoi diagrams [Lat91].

The traditional approach to path planning using Voronoi diagrams [Lat91, RSI91] involves reducing the robot to a point and suitably expanding all the obstacles. The problem is then transformed to moving the point robot from some initial configuration q_i to a final configuration q_f. q_i and q_f are retracted onto the nearest point in the Voronoi diagram and a path is obtained by searching the diagram. In this approach, we generally assume that a model of the obstacles and robot (with the required geometric information) is available a priori.

Modeling arbitrary shaped obstacles and robot is in general difficult. The discrete Voronoi diagram finds its use here. Assuming the obstacles are static and the robot may be translating and rotating, our approach is based on obtaining an image of the complete workspace using a digital camera, representing it as a two dimensional grid and binarizing the image so that the pixels corresponding to the obstacles have a value of 1 while those of the free space have a value of 0. No reduction of the robot to a point is performed. The Voronoi diagram of the obstacles is directly computed without

TH-2715_964102

expanding them with respect to the robot.

The path planning of an arbitrary shaped robot undergoing translation and rotation, can be done using the Voronoi diagram and $(\Delta x, \Delta y)$ of every pixel computed by our algorithm. $\sqrt{\Delta x^2 + \Delta y^2}$ gives the distance of a pixel from the nearest obstacle. Let d_{far} be the distance between the center of rotation of the robot and the farthest point in the robot from the center. Let the initial location of the center of rotation be given by c_i while the final location be given by c_f. First c_i and c_f have to be connected to the Voronoi diagram. This can be done as follows. c_i and c_f are in general located in the Voronoi regions of two different obstacles O_1 and O_2 respectively. (The case where c_i and c_f are located in the Voronoi region of the same obstacle is handled much more easily.) We can find a set of pixels (but not always) connecting c_i to a pixel on the Voronoi diagram by following pixels from c_i with increasing Euclidean distance from O_1. In a similar manner, a path connecting c_f to boundary of Voronoi region of O_2 can be found. We note that a collision-free path for the entire robot can be found if there is a set of pixels $p_1, p_2, \dots, p_k$ connecting c_i and c_f such that $d_{far} < \sqrt{\Delta x^2(p_i) + \Delta y^2(p_i)}$, $\forall i = 1, \dots, k$. The pixels $p_1, p_2, \dots, p_k$ are those on the path connecting c_i and c_f to the Voronoi diagram and those along the diagram itself. The center of rotation is aligned with these pixels.

One such path for the robot from initial to final positions of a robot is shown in Figure 8.3. In this application of robot path planning, the false Voronoi branch in Figure 6.10 has an advantage of having a path for the robot to access the object inside the other object.

The next potential application of Voronoi diagram is pattern classification.

8.3.2 Nearest Neighbor Pattern Classification

A *pattern classification* is a decision-making process which assigns an arbitrary pattern x of unknown class to any N possible classes $C_1, C_2, \dots, C_N$. Generally pattern classification involves two phases:

1. Training phase in which for a given set of patterns with known classes, feature vectors are extracted and a classifier is constructed using these feature vectors and their classes.

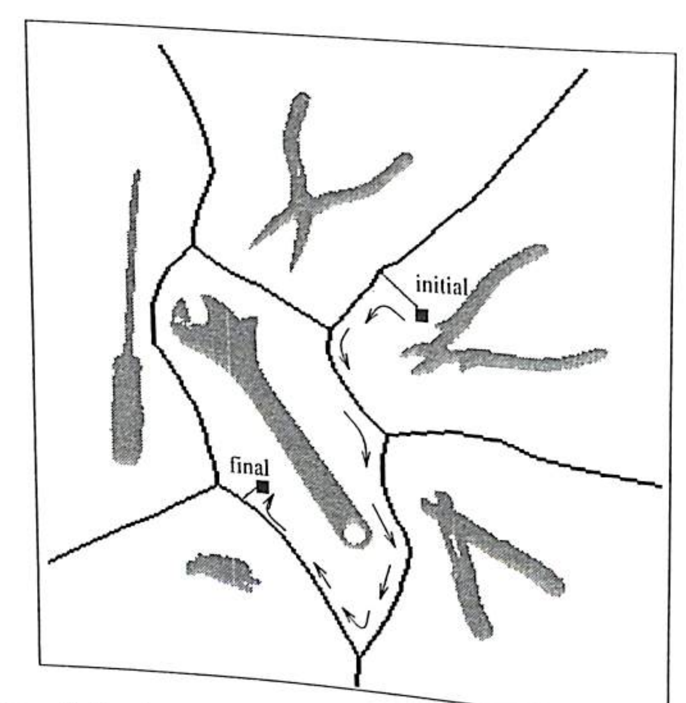


Figure 8.3: A path (shown by arrows) for a robot from initial position to final position along the Voronoi diagram.

2. Testing phase in which a new pattern with unknown class is assigned its class after giving it to the classifier.

The *nearest neighbor pattern classification* is an approach in which the decision for the classification of pattern x is based on the distance measure in the feature space of patterns. The class of x is assigned the known class of nearest neighboring pattern in the feature space. This classification is well suitable when the feature vectors of the patterns belonging to same class are clustered together in the feature space and the clusters of different classes do not overlap.

Our proposed nearest neighbor pattern classifier is meant for the 2-D integer feature space. Given a set of 2-D feature vectors whose elements are integers and the classes to which the features belong, the classifier is constructed by constructing the Voronoi regions of different classes. Voronoi region of class C consists of those points closer to the given feature points belonging to class C than to other points belonging to other classes. Any new pattern will lie in some Voronoi region. The class of the pattern is assigned the class of that Voronoi region. The 2-D integer space consisting of feature points can be treated as a binary image with feature and non-feature pixels. The Voronoi regions

TH-2715_964102

of different classes are constructed by dilating the feature pixels and p[ropagating the] class labels. The dilation process is stopped when total dilation occurs. That is, all the pixels are labeled and the Voronoi region of a class C consists of those pixels labeled with class C. For a new feature vector, the label of the pixel corresponding to that feature vector is the class to which it belongs. For an image of size $n \times n$, the construction of Voronoi diagram takes $O(n)$ time and identifying the class of new pattern takes constant time. The cellular automaton architecture for this problem consists of an $n \times n$ array of cells. Each cell has Euclidean distance module as in Figure 6.6. Instead of connectivity establishment module, there is only a storage for class label. A simulation result of construction of Voronoi regions for three classes of point sets is given below.

Simulation Study

We have generated three classes of random points with 1000 points in each class clustered within three non-overlapping circles of radii 50 pixel units and centers at (60,60),(60,200) and (200,125) in an image of size 256 × 256. The random points are generated by means of Gaussian distributions with their means at the centers of the circles. The Voronoi regions of three classes of random points are shown in the Figure 8.4 with different gray scales. It took 94 iterations to construct these Voronoi regions. The Voronoi diagram consists of the boundary pixels of different Voronoi regions.

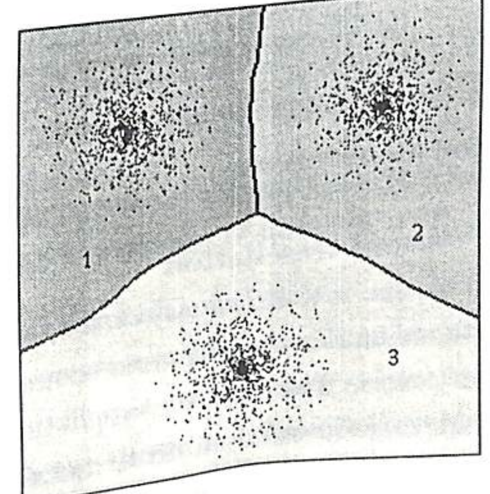


Figure 8.4: Voronoi regions of three classes of random points.

Another useful application of Voronoi diagram is image compression.

8.3.3 Image Compression and Secure Transmission

The Voronoi regions or cells of Voronoi tessellation can be used as primitives to represent regions [AAS85] of a piecewise uniform gray scale image. The Voronoi tessellation of the image grid is constructed for random pixels generated by a discrete planar random point process. The color of each random pixel is assigned the mean gray value of all the pixels in its Voronoi region. In the case of binary images, it is the color of the majority of pixels in the Voronoi region. The set of random pixels along with the assigned colors constitute the representation of the image. This representation achieves compression and secure transmission. If the sender and receiver use the same pseudo-random number generator to generate the random pixels, only the ordered list of colors need to be transmitted. The errors in the Voronoi representation result from the pixels in the vicinity of borders of the regions with same gray values. An adaptive Voronoi representation to minimize the errors is discussed in [RP88]. Here, the density of the random pixels can be adaptive to image properties: smoother regions will have fewer random pixels than more detailed regions. The adaptation property results in better quality of reconstructed image.

The above adaptive Voronoi representation does repeated construction of Voronoi tessellation over the image and in each construction, the density of random pixels is increased. In a sequential machine, the repeated construction is time consuming. In case of transmitting sequence of moving frames of images over a communication channel, a fast compression technique is required. We propose a parallel algorithm for the adaptive Voronoi representation of binary images. The algorithm provides fast compression and exact reconstruction and it is suitable to implement in a cellular architecture. The various steps of our method are as follows.

1. **Construction of Voronoi tessellation**

   For a binary image of size $n \times m$, we select $nm/125$ random pixels initially. A discrete version of planar uniform point process on a grid of size $n \times m$ is used to generate the random pixels. The Voronoi tessellation of the image is constructed for these pixels by dilating the random pixels based on Euclidean distance. During dilation, each pixel p is assigned a pointer value $ptr(p)$ which points to the pixel from which the Euclidean distance information $(\Delta x(p), \Delta y(p)$ and $d_f(p))$ is received. After total dilation, the pixels in the Voronoi region of a random pixel p form a tree structure with the root being p and every other pixel points to its parent pixel.

TH-2715_964102

2. **Identification of homogeneous and heterogeneous Voronoi r**

   The homogeneous region consists of all pixels with same values, either 1 or 0, and the heterogeneous region consists of pixels with both values 1 and 0. Let $b(p)$ be the binary value of pixel p and $h(p)$ is defined as follows.

   $$h(p) = \begin{cases} 1 & subtree(p) \text{ is homogeneous} \\ 0 & subtree(p) \text{ is heterogeneous} \end{cases}$$

   "$subtree(p)$ is homogeneous" means the subtree rooted at p have pixels with same binary value $b(p)$. A $subtree(p)$ is homogeneous if it satisfies the following conditions.

   - For every child p_c of p, $subtree(p_c)$ is homogeneous. That is, $h(p_c) = 1$.
   - The binary values $b(.)$ of p and its children p_c are same.

   Now, $h(p)$ can be defined in terms of children as follows.

   $$h(p) = \begin{cases} 1 & \forall p_c[h(p_c) = 1] \land \Big[\big[b(p) = 1 \land \forall p_c[b(p_c) = 1]\big] \\ & \quad \lor \big[b(p) = 0 \land \forall p_c[b(p_c) = 0]\big]\Big] \\ 0 & \text{otherwise} \end{cases}$$

   Once $h(.)$ of leaf pixels of a tree are initialized with 1, the corresponding values of other pixels in the tree can be iteratively computed from the level above the leaves to the root of the tree. This computation is performed in the same number of iterations taken to construct Voronoi tessellation in the previous step. The homogeneous and heterogeneous Voronoi regions are given by the values of $h(.)$ of root pixels (chosen random pixels) of different trees.

3. **Compression and secure transmission**

   The homogeneous regions are represented by the binary values of random pixels associated with them. Compression and secure transmission is achieved by transmitting $h(.)$ of all random pixels and $b(.)$ of those random pixels with $h(.) = 1$.

4. **Repeated construction of Voronoi tessellation**

   (a) Now the density of random pixels is increased to get the detailed representation of heterogeneous regions. We generate $nm/25$ random pixels in the

image. Only the pixels in the heterogeneous regions are considered and the Voronoi tessellation is constructed for these regions as in step 1. Steps 2 and 3 are repeated for this construction.

(b) The density of random pixels is further increased by generating $nm/5$ pixels and the Voronoi tessellation is constructed for the heterogeneous regions obtained in the previous step. Steps 2 and 3 are again repeated.

5. **Final step**

   Finally, $b(p)$ of all pixels p in the heterogeneous regions obtained in step 4(b) are transmitted

The data sent in all the above steps represent the image. The receiver of this data, generates the same $nm/125$ random pixels first and constructs Voronoi tessellation. He then identifies the homogeneous and heterogeneous regions using the received values, $h(.)$ of random pixels and assigns the corresponding binary values $b(.)$ to all the pixels in the homogeneous regions. Then he generates $nm/25$ pixels and constructs Voronoi tessellation of the heterogeneous regions of previous construction. The identified homogeneous regions of this construction are assigned the received binary values (transmitted in step 4(a)). The process is repeated for $nm/5$ pixels using the received values, transmitted in step 4(b). The pixels in the final heterogeneous regions are directly assigned the received values, transmitted in step 5. The reconstructed image is exactly like the original. The construction of Voronoi tessellation and identification of homogeneous and heterogeneous regions are the time consuming steps in the above described method when performed sequentially. But these steps involve identical local neighborhood operations and can be performed by all pixels in parallel and hence suitable to implement in a cellular architecture.

Experimental studies

The image compression algorithm has been tested for some binary images shown in Figure 8.5. The results of compression are tabulated in Table 8.2.

The step by step execution of the algorithm for an example image, "mixed1" is as follows. The algorithm first selects 71136/125=566 pixels randomly from the image and constructs the Voronoi tessellation. The Voronoi regions correspond to 352 pixels are

TH-2715_964102

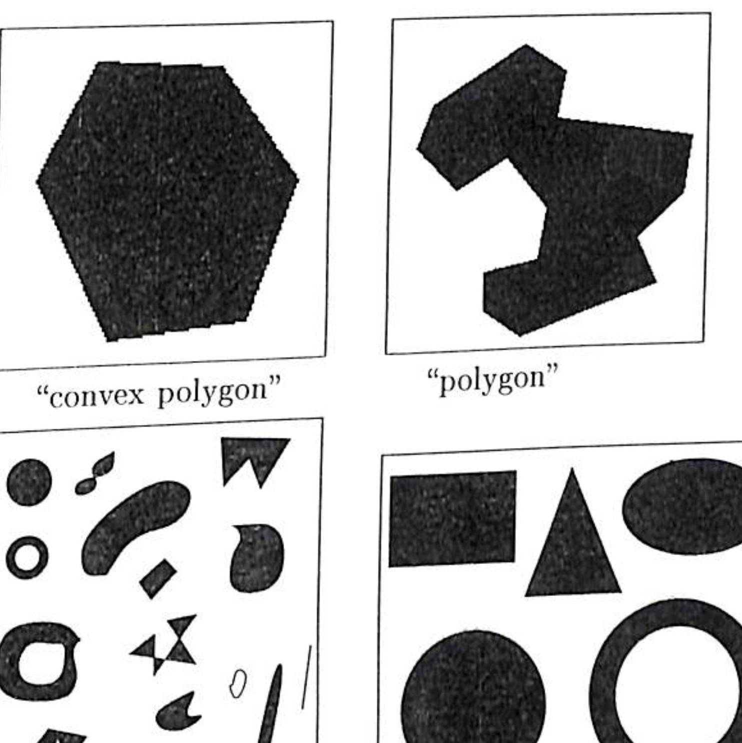


Figure 8.5: Test images for image compression using Voronoi tessellation.

identified to be homogeneous and the remaining 566-352=214 Voronoi regions are heterogeneous. The first phase of compression results in (2*352)+214= 918 bits. Secondly, the algorithm selects 71136/25=2845 random pixels. Only 1149 pixels lie in the heterogeneous regions of the previous construction. The Voronoi tessellation of these regions are constructed using the 1149 pixels. This construction leads to 662 homogeneous and 487 heterogeneous regions. The number of bits resulted in this phase is (2*662)+487=1811 bits. Next, 71136/5=14227 random pixels are generated, out of which, only 2422 lie in the 487 heterogeneous regions obtained. The Voronoi tessellation of these regions are constructed. 1670 homogeneous and 752 heterogeneous regions are obtained and (2*1670)+752=4092 bits are resulted. The number of pixels in the final 752 heterogeneous regions is 5462. The compressed image is composed of the bits resulting in all the three phases of compression and the bits that carry the binary values of pixels in the final heterogeneous regions. Hence, the total number of bits in the compressed image is 918+1811+4092+5462=12283.

Table 8.2: Results of compression.

Image and its dimension	size in bits	bits sent	compression factor
"convex polygon" 125×130	16250	1951	8.33
"polygon" 243×220	53460	4447	12.02
"mixed1" 288×247	71136	12283	5.79
"mixed2" 194×236	45784	7031	6.51

8.4 Applications of the Hausdorff Distance in Matching Images

In Chapter 7, we indicate how Hausdorff distance is useful to determine the degree of mismatch between two images (Section 7.4). Most of the previous work on Hausdorff distance for image matching, available in the literature, attempt to solve the problem of locating a given model M in an image I where both M and I are edge images. In this problem, the distance $h(M, I)$ is more significant than $h(I, M)$. We have taken the problem of Optical Character Recognition (OCR) and shown how to solve it using Hausdorff distance between character images. We have observed that both the directed Hausdorff distances are important in the problem of character recognition.

OCR is an important task in document analysis [SBB+92, MaKY92, Nag92, IOO91]. Given a document containing text, document analysis involves acquiring the image of the document, segmenting the text regions from drawings and pictures [SW89, RS86], isolating each character in the text [FNK92], normalizing these character images with respect to rotation and size and then classifying these into equivalent symbols so that a computer understands them. The classification step usually extracts features from the character images and these features are given to a classifier which outputs the desired class labels. Some feature extraction techniques [MaKY92] are computing Fourier series [Gra72], structure analysis approaches like thinning line analysis [Tam78], vectorization [Pav86], contour following analysis [FD77, PA75] and background analysis [MMY74].

The use of Hausdorff distance for comparing images [HKR93] in recent times helps

TH-2715_964102

us to propose a method for recognizing character images. The computat distance between images is described in Section 7.2. Hausdorff distance based recognition is tolerable to slight modifications in the character and it does not involve feature extraction and classifier design. A fast recognition system can be built in hardware. In this section, experimental studies are carried out to demonstrate the performance of the Hausdorff distance in solving the problem of recognizing different types of characters, *viz*, handwritten and printed. The characters can be partial, *i.e.*, some parts of characters are missing, and the characters can appear on a noisy background. We have taken character images of size 150×90 for all experiments.

8.4.1 Character Matching

Two character images match well if the Hausdorff distance H between them is small. In the case of character matching, both the directed Hausdorff distances from one character to the other are important. This can be easily seen by considering the characters **P** and **B**. Since character **P** is almost contained in character **B**, $h(\mathbf{P},\mathbf{B})$ is nearly 0 and hence **P** matches well with **B**. In order to distinguish these characters, $h(\mathbf{B},\mathbf{P})$ is needed, which is quite large. Hence bidirectional Hausdorff distance, $H(\mathbf{P},\mathbf{B}) = \max(h(\mathbf{P},\mathbf{B}), h(\mathbf{B},\mathbf{P}))$, has to be taken to distinguish characters **P** and **B**.

The Hausdorff distance is quite tolerable to slight modifications in a character. The Hausdorff distance between a character and its slightly perturbed one is less. Figure 8.6 shows different forms of characters **A** and **B**. The Hausdorff distance $H_i(.,.)$ (defined in Section 7.2 of Chapter 7) between every pair of these characters is given in Table 8.3. From the table, we can observe that the Hausdorff distance between different forms of same character is less when compared to Hausdorff distance between different characters. That is, $H_i(\mathbf{A}^m, \mathbf{A}^n)$ and $H_i(\mathbf{B}^m, \mathbf{B}^n)$ are less than 20 and $H_i(\mathbf{A}^m, \mathbf{B}^n) > 20$ for different values of m and n. By setting a distance limit $d_L = 20$, it is possible for us to do character matching. Two characters match each other if the Hausdorff distance between them is less than d_L. The character matching is a part of character recognition.

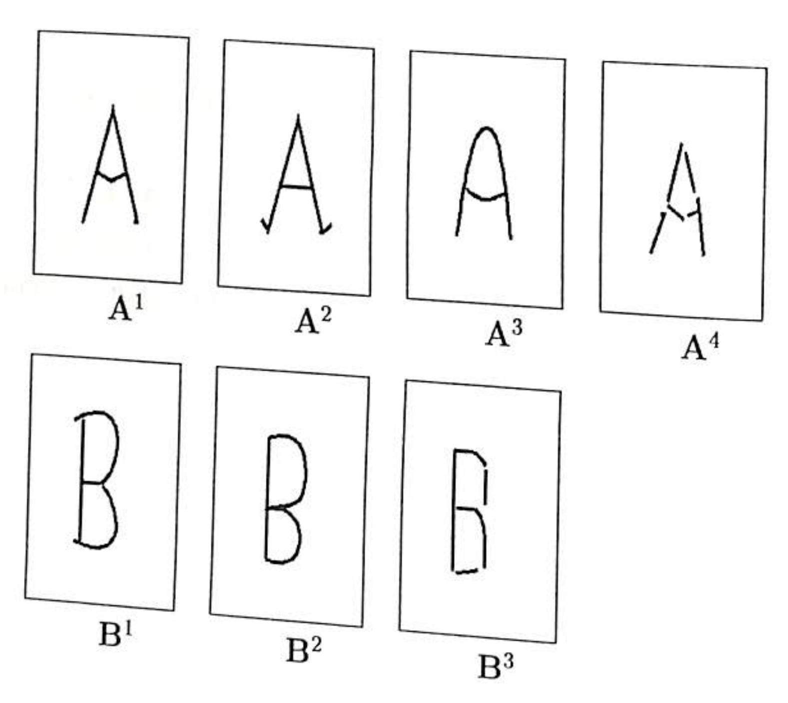


Figure 8.6: Different forms of test characters **A** and **B** for character matching.

8.4.2 Character Recognition

Given a character image, character recognition deals with recognizing the character present in the image. Character recognition can be performed by keeping the templates of different characters and then finding the best match of the given character with these templates. The Hausdorff distance can be used to find the best match. The distances between the given character and templates are first computed and the best matching template is one for which the Hausdorff distance between the given image and the template is minimum.

Experiments have been conducted to study the performance of Hausdorff distance for character recognition. For the experiments, we have taken the templates **A**,**B** and **P** shown in Figure 8.7 (first three characters). The remaining characters in the figure are considered for testing. These characters are handwritten and they differ in thickness. In order to nullify the effect of thickness in Hausdorff distance computation, the skeletons of these characters have been computed. The skeletons are shown in Figure 8.8. The Hausdorff distance between the skeletons of the test characters and templates are computed and the distances are given in Table 8.4. The best matching template for each test character is the one for which the Hausdorff distance is minimum. These

TH-2715_964102

Table 8.3: Hausdorff distances between every pair of characters in Figure 8.6.

H_i	$\mathbf{A}^1$	$\mathbf{A}^2$	$\mathbf{A}^3$	$\mathbf{A}^4$	$\mathbf{B}^1$	$\mathbf{B}^2$	$\mathbf{B}^3$
$\mathbf{A}^1$	0	9	9	8	31	24	26
$\mathbf{A}^2$	9	0	11	15	30	25	30
$\mathbf{A}^3$	9	11	0	9	26	25	24
$\mathbf{A}^4$	8	15	9	0	30	23	25
$\mathbf{B}^1$	31	30	26	30	0	8	10
$\mathbf{B}^2$	24	24	25	23	8	0	11
$\mathbf{B}^3$	26	30	24	25	10	11	0

minimum Hausdorff distances are shown in the table in boldface. Correct matching has been obtained for all the chosen test characters.

Table 8.4: Hausdorff distances between the skeletons (Figure 8.8) of test characters and templates in Figure 8.7.

H_i	$\mathbf{A}_1^s$	$\mathbf{A}_2^s$	$\mathbf{A}_3^s$	$\mathbf{B}_1^s$	$\mathbf{B}_2^s$	$\mathbf{B}_3^s$	$\mathbf{P}_1^s$	$\mathbf{P}_2^s$	$\mathbf{P}_3^s$
$\mathbf{A}^s$	**4**	**10**	**16**	30	29	30	46	49	42
$\mathbf{B}^s$	29	30	33	**8**	**10**	**16**	30	34	31
$\mathbf{P}^s$	45	49	52	38	34	35	**5**	**15**	**16**

8.4.3 Partial Character Recognition

In the case of an old document, the characters in the document may be erased partially. Partial image matching, described in Section 7.4, is useful in this case. Here we set a limit, d_L, to Hausdorff distance and find the fraction f' between two images. The images match well if f' is high. In the case of partial character recognition, a given partial character image is recognized by finding the best match with the existing templates.

For experimentation, we have taken the templates **A**,**B** and **P** in Figure 8.7 and the test characters in partial form as shown in Figure 8.9. d_L is chosen in such a way that it should be much less when compared to the minimum of the Hausdorff distances between different templates. This avoids mismatch. Also d_L should be large enough to tolerate slight perturbations and varying thickness between characters. The Hausdorff distance between templates is shown in Table 8.5. The minimum is 29 and we have chosen $d_L = 10$. The fractions f' computed between partial characters and templates

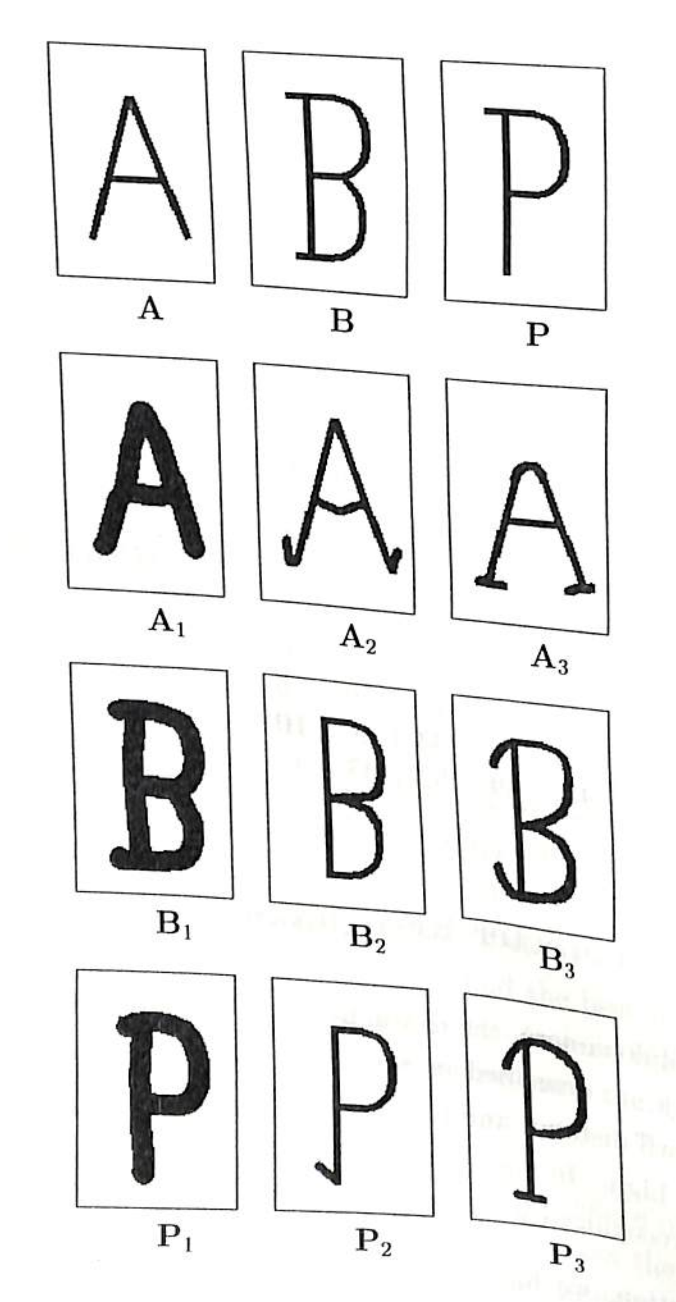


Figure 8.7: Character images considered for character recognition. **A**,**B** and **P** are templates. Remaining are test characters.

TH-2715_964102

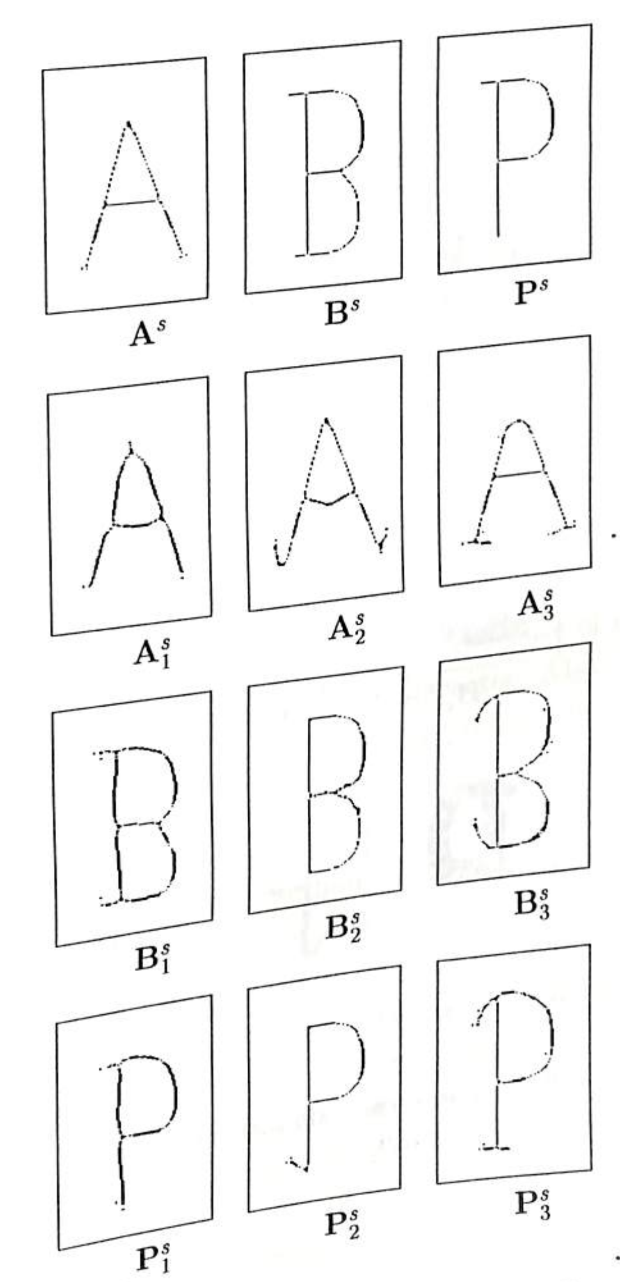


Figure 8.8: Skeleton of character images in Figure 8.7.

are given in Table 8.6. For a partial character, the template which gives maximum fractional value is the best match. This maximum is shown in boldface. The partial characters match correctly with their corresponding templates.

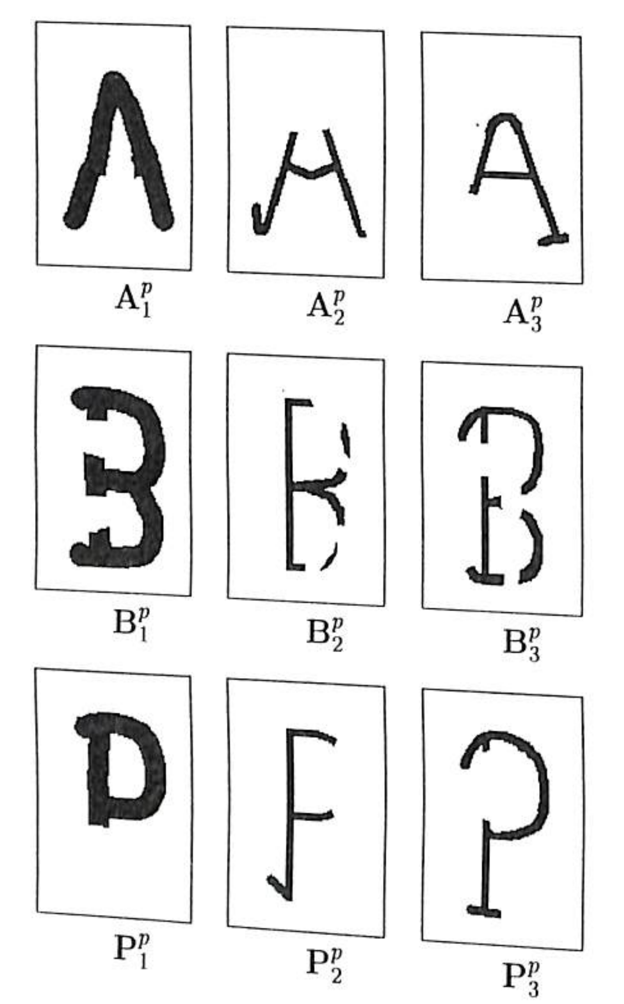


Figure 8.9: Character images considered for partial character recognition.

8.4.4 Character Recognition in the Presence of Noise

In real situation, images are in general noisy. If the document is old, then the background may not be clear. In some cases, the text may be printed on a patterned background. In these cases, an image will have unwanted components other than the character. Also, the characters in the text may be partial. Recognizing the characters present in a noisy (unclear) background using Hausdorff distance cannot be done in the way described in

TH-2715_964102

Table 8.5: Hausdorff distances between templates.

H_i	A	B	P
A	0	29	49
B	29	0	34
P	49	34	0

Table 8.6: Fractions f' computed between partial images in Figure 8.9 and templates in Figure 8.7.

f'	A_1^p	A_2^p	A_3^p	B_1^p	B_2^p	B_3^p	P_1^p	P_2^p	P_3^p
A	**1**	**0.82**	**0.84**	0.49	0.61	0.49	0.54	0.73	0.38
B	0.54	0.41	0.47	**0.92**	**0.98**	**0.96**	0.72	0.68	0.79
P	0.52	0.42	0.44	0.7	0.8	0.76	**0.86**	**0.77**	**0.89**

Section 8.4.2. Preprocessing by taking skeleton is not appropriate because the skeleton may have unwanted loops due to missing pixels (because of noise) in the character. The skeleton will not preserve the shape of the character. Also, the directed Hausdorff distance $h_i(I,T)$ from image I to a template T is highly influenced by the unwanted components in the background. The partial character recognition described in the previous section is also not appropriate because the fractions computed are influenced by the unwanted components. Hence a method is suggested to remove these unwanted components, which involves setting a distance limit d_L and computing fractions as in partial character recognition. First, we find the region of interest in the given image with respect to templates. This consists of portions covered by the dilated characters in the templates. The dilation is performed up to distance d_L. The components outside these regions are considered as noise and they are removed. If the templates are large in number, then even the unwanted components present in the region of interest will be predominant. So, we consider a smaller region of interest by selecting only those templates which match well with the image. This selection is done by finding the fraction f_T which is equal to the number of foreground pixels that are covered by the dilated character of the template T in the image divided by the number of pixels in the dilated character. The region of interest now consists of the portions covered by the dilated characters of templates whose $f_T \ge f_{th}$, where f_{th} is a threshold. We take $f_{th} = 0.9 \max_T f_T$. The selection of f_{th} helps us to consider only those templates which closely match with the character in the noisy image. For example, consider the noisy character $\mathbf{B}_3^n$ in Figure 8.10 and the templates **A**, **B** and **P** in Figure 8.7. With $d_L = 5$, the computed fractions

$f_{\mathbf{A}}$,$f_{\mathbf{B}}$ and $f_{\mathbf{P}}$ are 0.29,0.48 and 0.46. So $f_{th} = 0.432$ and the templates that match well with $\mathbf{B}_3^n$ are **B** and **P**. Considering the portions of the dilated characters **B** and **P** as region of interest, the noise removed image $\mathbf{B}_3^{nr}$ is shown in Figure 8.10. Once the noise is removed, the character is recognized by finding fraction f_{IT} for each template T. f_{IT} is given by the number of foreground pixels that are common to the dilated character of template T and the noise removed image divided by the number of foreground pixels of the noise removed image. f_{IT} gives the degree of matching of noise removed image with the template. The template that gives $\max_T f_{IT}$ corresponds to the best match.

Experiments have been conducted for recognizing noisy characters. The noisy character images in Figure 8.10 have been considered for testing and the templates are **A**,**B** and **P** in Figure 8.7. The fraction f_{IT} has been computed for every test image I and template T. Table 8.7 shows these computed fractions with $d_L = 5$. For each test image, the maximum of three fractional values with the three templates is given in boldface. In Table 8.7, the maximum values correspond to the correct match. Correct matching depends on the values of d_L and f_{th}. For a given training set of noisy characters for each template, we can tune d_L and f_{th} for correct matching. Once d_L and f_{th} have been selected, any new noisy image can be recognized by finding the best match with the templates.

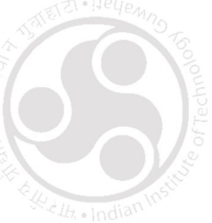
Table 8.7: Fractions f_{IT} computed between noisy character images in Figure 8.10 and templates in Figure 8.7.

f_{IT}	$\mathbf{A}_1^n$	$\mathbf{A}_2^n$	$\mathbf{A}_3^n$	$\mathbf{B}_1^n$	$\mathbf{B}_2^n$	$\mathbf{B}_3^n$	$\mathbf{P}_1^n$	$\mathbf{P}_2^n$	$\mathbf{P}_3^n$
A	**1**	**1**	**1**	0.36	0.35	0.31	0.41	0.37	0.35
B	0.47	0.46	0.46	**0.96**	**0.95**	**0.96**	0.97	0.93	0.93
P	0.39	0.37	0.33	0.74	0.72	0.68	**1**	**1**	**1**

8.5 Conclusions

This chapter gives some applications of skeleton, Voronoi diagram and Hausdorff distance. Most of the applications are related to the field of machine vision. The applications include shape analysis using skeleton, robot path planning and pattern classification using Voronoi diagram and image matching using Hausdorff distance. We have proposed a new application of Hausdorff distance for character recognition.

TH-2715_964102



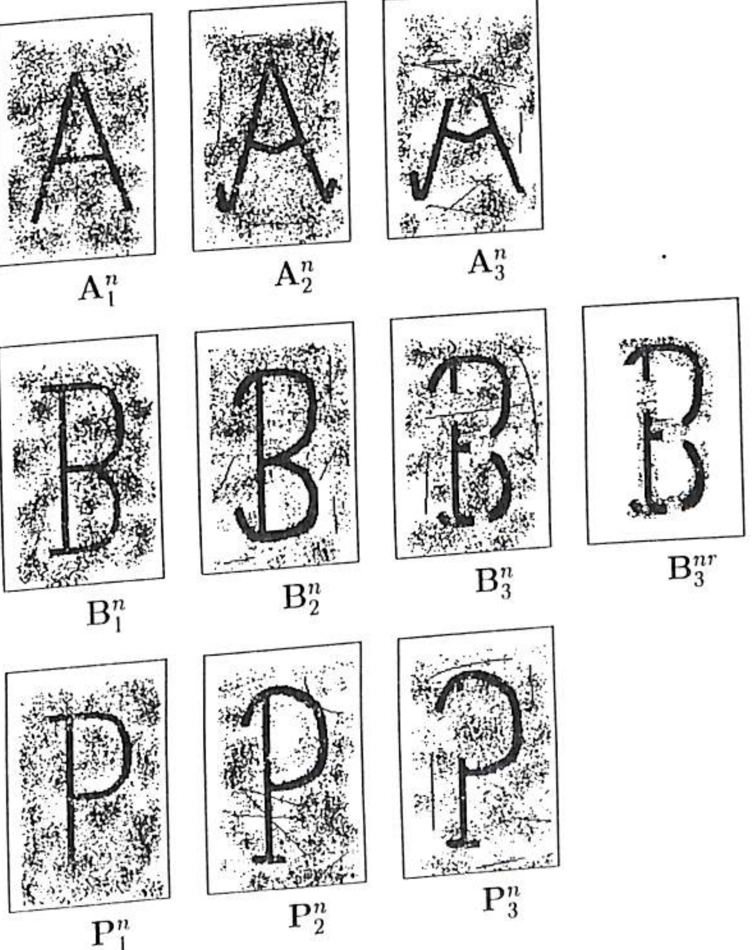


Figure 8.10: Images considered for noisy character recognition. $\mathbf{B}_3^{nr}$ is the noise removed image of $\mathbf{B}_3^n$.

Chapter 9

Conclusions and Future Work

We have addressed some important problems in machine vision in this thesis. It is shown how one can obtain a fast and cost-effective solution using cellular architectures. The simple design of the architecture and its evolution in time based on the concept of cellular automata are exploited.

9.1 Contributions of the Thesis

The main contributions of the thesis are listed below.

1. We have considered the problem of computing Euclidean distance transform (EDT) of a binary image. EDT is crucial to solving many problems in machine vision. Difficulties in directly implementing algorithm for EDT in VLSI have been discussed. A parallel algorithm has been developed for the computation of distance vector $(\Delta x(p), \Delta y(p))$ for each pixel p. The computation is an iterative procedure and it takes $O(n)$ time to compute EDT of an $n \times n$ image. The procedure is based on dilation. A cellular architecture consisting of an $n \times n$ array of cells has been designed to implement the algorithm. The design has been implemented in Xilinx FPGA and the maximum frequency of operation of the clock is about 5 MHz. The implementation results show that the architecture can process an image much faster than the video rate.

TH-2715_964102

2. We have then identified three important problems that can be solved using the approach for EDT. The problems are computation of skeleton, Voronoi diagram and Hausdorff distance for binary images. We have developed algorithms and designed cellular architectures for these problems.

   Skeleton An approach for the computation of skeleton based on comparing the distance values of neighboring pixels for each pixel has been proposed. Two methods have been developed, one based on integer Euclidean distance $d(p)$ and the other based on actual distance $d_e(p)$. Simulation of the method in a sequential computer has been carried out for images with different types of objects. From the results, we have observed that the proposed method extracts exact skeleton for polygonal objects in the image. However, for arbitrarily shaped objects, the reconstruction of objects from the computed skeleton and the distance values is perfect.

   An efficient cellular architecture has been designed. The cell is similar to the one for EDT with some extra logic for computing the skeleton value of a pixel. One of the findings is that a cell in the architecture does not require storage for $d(p)$ and $d_e(p)$. It exploits the existing storage elements for computing the skeleton value.

   Voronoi diagram A method has been proposed for computing Voronoi diagram of real objects in an image. The method considers each object as a connected component and dilates all the connected components uniformly to construct the diagram. While dilating, the connectivity between neighboring pixels belonging to the same connected component is established using connectivity flags. Performance of the method has been studied with real and synthetic images containing different types of objects at various locations. Except for one special configuration of two objects (one object is partially enclosed by another; in this case the diagram has an additional branch), the performance is quite good. The additional branch produced for this special configuration seems advantageous for robot path planning type of applications.

   A cell designed for this problem consists of two modules, *viz*, dilation and connectivity establishment. The dilation module has components of the cell for EDT and the other module stores connectivity flags and has logic for computing them.

   Hausdorff distance A method has been proposed for computing the integer ap-

TH-2715_964102

   proximation to the Hausdorff distance $H_i(A, B)$ between two images A and B. The integer approximation is adequate for image matching. The integer approximation to directed Hausdorff distance $h_i(A, B)$ from A to B is computed by dilating the foreground of B until it covers the foreground of A. The use of Hausdorff distance for image matching has been explored. The method of computation of Hausdorff distance has been extended to partial image matching. Experimental studies have been conducted with character images to evaluate the performance.

   A cellular architecture has been designed to compute $h_i(A, B)$. The same architecture can be used to compute $h_i(B, A)$ and $H_i(A, B)$ is the maximum of these two. A cell is similar to the one designed for EDT but with two flip-flops to store the binary values of pixels of both images. The architecture has an external control logic to implement the stopping condition for the updation of cells.

3. Finally, we have explored the applications of skeleton, Voronoi diagram and Hausdorff distance.

   Skeleton It is used for image compression and shape analysis. Compression amounts to storing the coordinates of skeleton pixels and their distance values. Some image processing operations which can be performed in this compressed image have been identified. The number of branches and loops in the skeleton of an object gives the topology of the object.

   Voronoi diagram The applications of Voronoi diagram include robot path planning, nearest neighbor pattern classification and image compression. A collision-free path for a translating and rotating robot can be found out using the Voronoi diagram of obstacles in the robot work space and the distance values. Constructing Voronoi regions for different classes of feature vectors of known patterns helps in identifying the class of an unknown pattern. In the case of image compression, the Voronoi regions can be used as primitives to represent regions of a piecewise uniform gray scale image.

   Hausdorff distance We have applied Hausdorff distance for character recognition. Methods have been developed to recognize noisy and partial character images. Performance of the methods has been evaluated on different test images.

9.2 Future Work

In the following, we outline some of the possible directions of future research.

Similar to EDT for a binary image, the gray weighted distance transform can be taken for a gray image. Gray weighted distances are useful to get depth information from shading. A cellular architecture may be designed for computing the gray weighted distance transform.

The method proposed for the computation of skeleton extracts exact skeleton for polygonal objects. It is of interest to improve the method for non-polygonal case.

It is generally difficult to segment objects and background in a gray image to get a binary image. Finding skeletons for a gray image is given in [KZ96]. A cellular architecture may be designed for this problem.

The application of discrete Voronoi diagram for robot path planning has been discussed in Section 8.3.1 of previous chapter. We have given an approach to find whether a collision-free path exists from the initial to final location of a robot. A complete path planning involves finding an arrangement of pixels. It must be investigated.

The Hausdorff distance is used for measuring fractal dimensions. Its use for fractal images can be explored.

EDT can be applied to solve many other problems. They include watershed segmentation, fractal dimension measurement and cluster analysis [Rus95]. We can obtain cellular architecture based solutions for solving them

Real-time imaging involves many image processing operations. In literature, VLSI based solutions are available only for some operations. Finding VLSI based solutions to other real-time image processing operations is useful.

TH-2715_964102

Appendix A

VHDL design of basic cell

VHDL is an acronym for VHSIC Hardware Description Language (VHSIC is an acronym for Very High Speed Integrated Circuits). It is a hardware description language that can be used to model a digital system at many levels of abstraction, ranging from the architecture level to the gate level. The digital system can be described hierarchically. Timing can also be explicitly modeled in the same description.

A hardware abstraction of a digital system is called an *entity* in VHDL. An entity X, when used in another entity Y, becomes a component for the entity Y. An entity is modeled using an entity declaration and at least one architecture body. The entity declaration describes the external view of the entity; for example, the input and output signal names. The architecture body contains the internal description of the entity; for example, as a set of interconnected components that represents the structure of the entity, or as a set of concurrent or sequential statements that represents the behavior of the entity. Each style of representation can be specified in a different architecture body or mixed within a single architecture body.

A *process* statement contains sequential statements that describe the functionality of a portion of an entity in sequential terms. A set of signals to which the process is sensitive is defined in the process statement after the keyword **process**. A *variable* assignment statement assigns a value to a variable using operator ":=" and a *signal* assignment statement assigns a value to a signal using " <= ".

A *package* declaration is used to store a set of common declarations, such as compo-

nents, types, procedures and functions. These declarations can then be imported into other design units (or entities) using a **use** clause. For more details, the reader may consult [Bha95].

The different VLSI architectures presented in Chapters 4-7 have been designed in VHDL (based on IEEE std 1076-1993). A sample code follows.

VHDL Code

```
-------------------------------------------------------
----- Design of basic cell for computing EDT in VHDL -----
-------------------------------------------------------

library STD,IEEE;
use STD.all,IEEE.all;

package declarations is
  -- constant declarations
  constant dx_size:integer:=9;  -- ceil log M
  constant dy_size:integer:=9;  -- ceil log N
  constant diff_size:integer:=13; -- ceil log 6*sqrt(M^2+N^2)
  constant count_size:integer:=10; -- ceil log sqrt(M^2+N^2)
end declarations;


------------ CMP-MUX block: the component of the MAX block ------------
use work.declarations.all;
entity cmp_mux is
  port (diff0,diff1:in BIT_VECTOR(diff_size downto 1);
        done0,done1,borrow0,borrow1:in BIT; dx0,dx1:in
BIT_VECTOR(dx_size downto 1); dy0,dy1:in
BIT_VECTOR(dy_size downto 1); diff:out
BIT_VECTOR(diff_size downto 1); done,borrow:out BIT;
dx:out BIT_VECTOR(dx_size downto 1);
dy:out BIT_VECTOR(dy_size downto 1));
end cmp_mux;

architecture CmpMux_struct of cmp_mux is

  signal A1out,A2out,A3out,A4out,O1out,cmp:BIT;
  signal A1in,A2in:BIT_VECTOR(1 to 5);
  signal A3in,O1in:BIT_VECTOR(1 to 4);
  signal A4in:BIT_VECTOR(1 to 2);
  signal MUX1in0,MUX1in1,MUX1out:BIT_VECTOR(1 to
         diff_size+dx_size+dy_size+2);
```

```
begin
  -- assigning the ports of AND gates
  A1in(1)<=done0; A1in(2)<=done1; A1in(3)<=not cmp; A1in(4)<=not borrow0; A1in(5)<=not borrow1;
  A1:AND_GATE generic map(5) port map(A=>A1in,Z=>A1out);

  A2in(1)<=done0; A2in(2)<=done1; A2in(3)<=not cmp; A2in(4)<=borrow0;  A2in(5)<=borrow1;
  A2:AND_GATE generic map(5) port map(A2in,A2out);

  A3in(1)<=done0; A3in(2)<=done1; A3in(3)<=not borrow0; A3in(4)<=borrow1;
  A3:AND_GATE generic map(4) port map(A3in,A3out);

  A4in(1)<=not done0; A4in(2)<=done1;
  A4:AND_GATE generic map(2) port map(A4in,A4out);

  -- assigning the ports of an OR gate
  O1in(1)<=A1out; O1in(2)<=A2out; O1in(3)<=A3out; O1in(4)<=A4out;
  O1:OR_GATE port map(O1in,O1out);

  -- assigning the ports of a multiplexer
  MUX1in0(1 to diff_size)<=diff0;
  MUX1in0(diff_size+1 to diff_size+dx_size)<=dx0;
  MUX1in0(diff_size+dx_size+1 to diff_size+dx_size+dy_size)<=dy0;
  MUX1in0(diff_size+dx_size+dy_size+1)<=borrow0;
  MUX1in0(diff_size+dx_size+dy_size+2)<=done0;

  -- similar code for MUX1in1

  diff<=MUX1out(1 to diff_size);
  dx<=MUX1out(diff_size+1 to diff_size+dx_size);
  dy<=MUX1out(diff_size+dx_size+1 to diff_size+dx_size+dy_size);
  borrow<=MUX1out(diff_size+dx_size+dy_size+1);
  done<=MUX1out(diff_size+dx_size+dy_size+2);

  MUX1:MUX port map(MUX1in0,MUX1in1,MUX1out,O1out);

  CMP1:comp_gt port map(num1=>diff0,num2=>diff1,out_bit=>cmp);
end CmpMux_struct;

----------------- MAX block -----------------------------------
use work.declarations.all;
entity MAX is
  port(diff_in1,diff_in2,diff_in3,diff_in4,diff_in5,diff_in6,
    diff_in7,diff_in8:in BIT_VECTOR(diff_size downto 1);
    done_in1,done_in2,done_in3,done_in4,done_in5,done_in6,
    done_in7,done_in8,borrow_in1,borrow_in2,borrow_in3,borrow_in4,
    borrow_in5,borrow_in6,borrow_in7,borrow_in8:in BIT;
    dx_in1,dx_in2,dx_in3,dx_in4,dx_in5,dx_in6,dx_in7,dx_in8: in
    BIT_VECTOR(dx_size downto 1); dy_in1,dy_in2,dy_in3,dy_in4,dy_in5,
    dy_in6,dy_in7,dy_in8: in BIT_VECTOR(dy_size downto 1); diff_out: out
    BIT_VECTOR(diff_size downto 1); done_out,borrow_out: out BIT;
    dx_out: out BIT_VECTOR(dx_size downto 1);
    dy_out: out BIT_VECTOR(dy_size downto 1));
```

TH-2715_964102

```
end MAX;

architecture  MAX_struct of MAX is

  component cmp_mux
    port(diff0,diff1:in BIT_VECTOR(diff_size downto 1);
         done0,done1,borrow0,borrow1:in BIT;
         dx0,dx1: in BIT_VECTOR(dx_size downto 1);
         dy0,dy1: in BIT_VECTOR(dy_size downto 1);
         diff: out BIT_VECTOR(diff_size downto 1);
         done,borrow: out BIT;
         dx: out BIT_VECTOR(dx_size downto 1);
         dy: out BIT_VECTOR(dy_size downto 1));
  end component;

  signal diff_tmp1,diff_tmp2,diff_tmp3,diff_tmp4,diff_tmp5,
   diff_tmp6,diff_tmp7,diff_tmp8: BIT_VECTOR(diff_size downto 1);
  signal dx_tmp1,dx_tmp2,dx_tmp3,dx_tmp4,dx_tmp5,dx_tmp6,
   dx_tmp7,dx_tmp8: BIT_VECTOR(dx_size downto 1);
  signal dy_tmp1,dy_tmp2,dy_tmp3,dy_tmp4,dy_tmp5,dy_tmp6,
   dy_tmp7,dy_tmp8: BIT_VECTOR(dy_size downto 1);
  signal done_tmp1,done_tmp2,done_tmp3,done_tmp4,done_tmp5,done_tmp6,borrow_tmp1,
   borrow_tmp2,borrow_tmp3,borrow_tmp4,borrow_tmp5,borrow_tmp6: BIT;

  begin
    cmp_mux1: cmp_mux port map(diff_in1,diff_in2,done_in1,done_in2,borrow_in1,borrow_in2,
      dx_in1,dx_in2,dy_in1,dy_in2,diff_tmp1,done_tmp1,borrow_tmp1,dx_tmp1,dy_tmp1);
    cmp_mux2: cmp_mux port map(diff_in3,diff_in4,done_in3,done_in4,borrow_in3,borrow_in4,
      dx_in3,dx_in4,dy_in3,dy_in4,diff_tmp2,done_tmp2,borrow_tmp2,dx_tmp2,dy_tmp2);
    cmp_mux3: cmp_mux port map(diff_in5,diff_in6,done_in5,done_in6,borrow_in5,borrow_in6,
      dx_in5,dx_in6,dy_in5,dy_in6,diff_tmp3,done_tmp3,borrow_tmp3,dx_tmp3,dy_tmp3);
    cmp_mux4: cmp_mux port map(diff_in7,diff_in8,done_in7,done_in8,borrow_in7,borrow_in8,
      dx_in7,dx_in8,dy_in7,dy_in8,diff_tmp4,done_tmp4,borrow_tmp4,dx_tmp4,dy_tmp4);
    cmp_mux5: cmp_mux port map(diff_tmp1,diff_tmp2,done_tmp1,done_tmp2,borrow_tmp1,borrow_tmp2,
      dx_tmp1,dx_tmp2,dy_tmp1,dy_tmp2,diff_tmp5,done_tmp5,borrow_tmp5,dx_tmp5,dy_tmp5);
    cmp_mux6: cmp_mux port map(diff_tmp3,diff_tmp4,done_tmp3,done_tmp4,borrow_tmp3,borrow_tmp4,
      dx_tmp3,dx_tmp4,dy_tmp3,dy_tmp4,diff_tmp6,done_tmp6,borrow_tmp6,dx_tmp6,dy_tmp6);
    cmp_mux7: cmp_mux port map(diff_tmp5,diff_tmp6,done_tmp5,done_tmp6,borrow_tmp5,borrow_tmp6,
      dx_tmp5,dx_tmp6,dy_tmp5,dy_tmp6,diff_out,done_out,borrow_out,dx_out,dy_out);
end MAX_struct;

--------------------- EDT cell ---------------------------------
use work.declarations.all;
entity CELL is
  port(clk,done1,done2,done3,done4,done5,done6,done7,done8:in BIT;
    dx1,dx2,dx3,dx4,dx5,dx6,dx7,dx8:in BIT_VECTOR(dx_size downto 1);
    dy1,dy2,dy3,dy4,dy5,dy6,dy7,dy8:in BIT_VECTOR(dy_size downto 1);
    diff1,diff2,diff3,diff4,diff5,diff6,diff7,diff8:in
    BIT_VECTOR(diff_size downto 1); k:in BIT_VECTOR (count_size downto 1);
    dx:out BIT_VECTOR(dx_size downto 1); dy:out BIT_VECTOR(dy_size downto 1);
    diff:out BIT_VECTOR(diff_size downto 1); done:out BIT);
end CELL;

architecture CELL_struct of CELL is

 function shlft(din:BIT_VECTOR;size:integer) return BIT_VECTOR is
   variable dout:BIT_VECTOR(diff_size downto 1);
   begin
     dout(1):='1';
     for I in 2 to size+1 loop
       dout(I):=din(I-1);
     end loop;
     for I in size+2 to diff_size loop
       dout(I):='0';
     end loop;
     return dout;
   end shlft;

  signal df1,df2,df3,df4,df5,df6,df7,df8,df_out,df:
    BIT_VECTOR(diff_size downto 1);
  signal done_out,bor1,bor2,bor3,bor4,bor5,bor6,bor7,bor8,clk_donedxdy,
    cndn,clk_diff,or_out,DFF_in:BIT;
  signal dx2pl1,dx3pl1,dx4pl1,dx6pl1,dx7pl1,dx8pl1,dx_out:
    BIT_VECTOr(dx_size downto 1);
  signal dy1pl1,dy2pl1,dy4pl1,dy5pl1,dy6pl1,dy8pl1,dy_out:
    BIT_VECTOR(dy_size downto 1);
  signal sub1in,add2in1,add2in2,add2out,sub3in,add4in1,add4in2,add4out,
    sub5in,add6in1,add6in2,add6out,sub7in,add8in1,add8in2,add8out,
    addoutin1,addoutin2,addout:BIT_VECTOR(diff_size downto 1);

 begin
  -- 4 subtracters and 4 adder-subtracters in the input side
  sub1in <= shlft(dy1,dy_size);
  sub1: subN port map(diff1,sub1in,df1,bor1);

  add2in1 <= shlft(dx2,dx_size);
  add2in2 <= shlft(dy2,dy_size);
  add2: adderN port map(add2in1,add2in2,open,add2out);
  sub2: subN port map(diff2,add2out,df2,bor2);

  sub3in <= shlft(dx3,dx_size);
  sub3: subN port map(diff3,sub3in,df3,bor3);

  add4in1 <= shlft(dx4,dx_size);
  add4in2 <= shlft(dy4,dy_size);
  add4: adderN port map(add4in1,add4in2,open,add4out);
  sub4: subN port map(diff4,add4out,df4,bor4);

  sub5in <= shlft(dy5,dy_size);
  sub5: subN port map(diff5,sub5in,df5,bor5);

  add6in1 <= shlft(dx6,dx_size);
  add6in2 <= shlft(dy6,dy_size);
  add6: adderN port map(add6in1,add6in2,open,add6out);
  sub6: subN port map(diff6,add6out,df6,bor6);
```

TH-2715_964102

```
sub7in <= shlft(dx7,dx_size);
sub7: subN port map(diff7,sub7in,df7,bor7);

add8in1 <= shlft(dx8,dx_size);
add8in2 <= shlft(dx8,dy_size);
add8: adderN port map(add8in1,add8in2,open,add8out);
sub8: subN port map(diff8,add8out,df8,bor8);

-- .incrementing dx and dy of appropriate neighbours
inc1y: incrementer generic map(dy_size) port map(dy1,open,dy1pl1);
inc2y: incrementer generic map(dy_size) port map(dy2,open,dy2pl1);
 -- similar component instantiation for inc4y, inc5y, inc6y, inc8y

inc2x: incrementer generic map(dx_size) port map(dx2,open,dx2pl1);
inc3x: incrementer generic map(dx_size) port map(dx3,open,dx3pl1);
 -- similar component instantiation for inc4x, inc5x, inc7x, inc8x

-- MAX logic
MAXdf: MAX port map(df1,df2,df3,df4,df5,df6,df7,df8,done1,done2,
  done3,done4,done5,done6,done7,done8,bor1,bor2,bor3,bor4,bor5,bor6,
  bor7,bor8,dx1,dx3pl1,dx3pl1,dx4pl1,dx5,dx6pl1,dx7pl1,dx8pl1,dy1pl1,
  dy2pl1,dy3,dy4pl1,dy5pl1,dy6pl1,dy7,dy8pl1,df_out,open,open,dx_out,dy_out);

-- MUX in the output side of MAX
MUXout: MUX port map(df_out,df,addoutin1,done_out);

-- adder in the output side of MAX
process (addoutin1)
 variable temp:BIT_VECTOR(diff_size downto 1);
begin
 temp(1):='0';
 for I in 2 to count_size+1 loop
  temp(I):= k(I-1);
 end loop;
 if (diff_size > (count_size+1)) then
  for I in count_size+2 to diff_size loop
   temp(I):= '0';
  end loop;
 end if;
 addoutin2 <= temp;
end process;

ADDERout: adderN port map(addoutin1,addoutin2,open,addout);
             -- addout carries df value
-- regs
DFF_in <= '1';
REGdf: reg generic map(diff_size) port map(addout,clk_diff,df);
REGdx: reg generic map(dx_size) port map(dx_out,clk_donedxdy,dx);
REGdy: reg generic map(dy_size) port map(dy_out,clk_donedxdy,dy);
DFFdone: DFF port map(DFF_in,clk_donedxdy,done_out);

--  clock inhibition
or_out<= done1 or done2 or done3 or done4 or done5 or done6 or
  done7 or done8;
cndn<= (not done_out) and or_out and (not addout(diff_size));
                      -- addout(diff_size) is the MSB of df
clk_donedxdy <= clk and cndn; clk_diff <= clk and (cndn or done_out);

-- assign done_out to done and df to diff
done <= done_out; diff <= df;

end CELL_struct;
```

Appendix B

Functional Simulation in ModelSim

ModelSim is a software package used to check the syntax and the functional behaviour of the VHDL code of a VLSI design before implementing the design in hardware. The functional simulation allows the designers to trace design problems and to fix them by giving input signals to the design and checking the outputs of different components of the design.

The functional simulation of VHDL designs of different cells in the thesis has been carried out in ModelSim. The VHDL design of each cellular architecture has also been tested functionally with different images. For example, the functional testing of VHDL code (Appendix A) of basic cell in ModelSim is given below.

The basic cell corresponding to a 16 × 16 image has been considered. The sizes of the storage elements Δx, Δy and d_f are therefore 4, 4 and 8 bits respectively. The external counter is a 5-bit counter. The functional testing has been carried out by giving different input test signals. The test inputs correspond to pixels at different configurations. The configurations are as follows.

Configuration 1

The cell corresponds to a pixel whose distance values have been computed. The flip-flop *done* has the value 1 and the registers Δx and Δy have the components of distance vector. In this case, only d_f is incremented by $2k$. For iteration number $k = (01011)_2$,

TH-2715_964102

the design has been run for 100 ms and a clock trigger has been given at 50 ms. The waveforms at different signal points at different time instants are shown in Figure B.1. The signals *done_old* and $diff_old$ represent the feedback inputs to the combinational logic of the basic cell. These are the values stored in *done* and d_f before clock trigger. In the figure, we can observe that the clock *clk_donedxdy* to Δx, Δy and *done* is not activated and only the clock clk_diff to d_f is activated. The output of register d_f is given by $diff$. The value of $diff$ is verified for different values of k and $diff_old$.

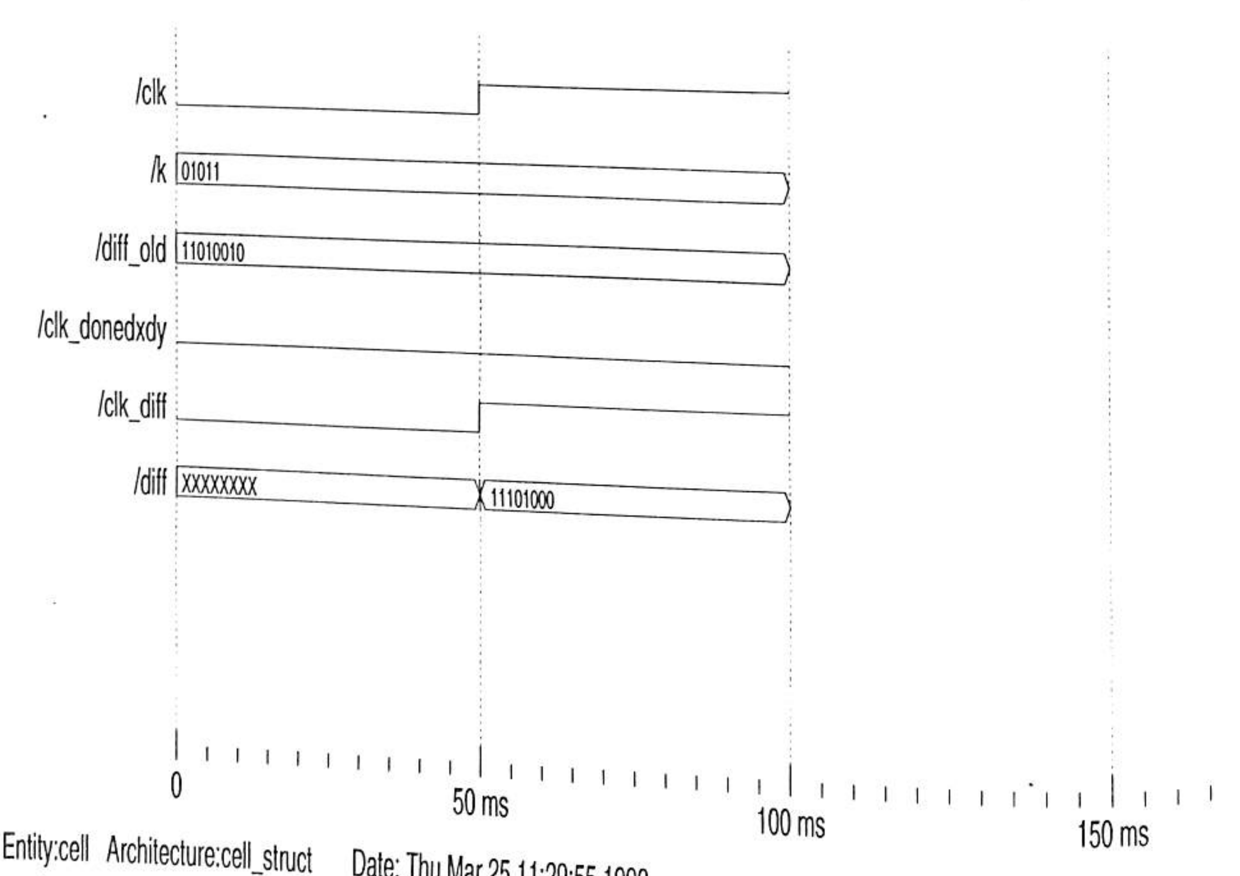


Figure B.1: Results of functional testing of VHDL design of basic cell corresponding to configuration 1 in ModelSim. The design has been run for 100 ms and the clock trigger has been given at 50 ms. Change of state occurs after clock trigger. *clk*, *k*, $diff_old$, *clk_donedxdy*, clk_diff and $diff$ are the signals declared in VHDL design. *clk*: clock to cell; *k*: external counter output; $diff_old$: contents of d_f before change of state of cell; *clk_donedxdy*: clock to *done*, *dx* and *dy*; clk_diff: clock to d_f; $diff$: contents of d_f after change of state of cell.

Configuration 2

We assume here that the cell corresponds to a pixel whose distance values have not yet been computed and the distance values of neighbors have also not been computed. Hence, all storage elements of the cell have the value 0. This means that $diff_old$ and *done_old* are applied the value 0. All the inputs from the neighbors are also 0. For any value of k and a clock trigger as input, the clocks to the storage elements are displayed in Figure B.2. These clocks are not activated which shows that the contents of storage elements do not change.

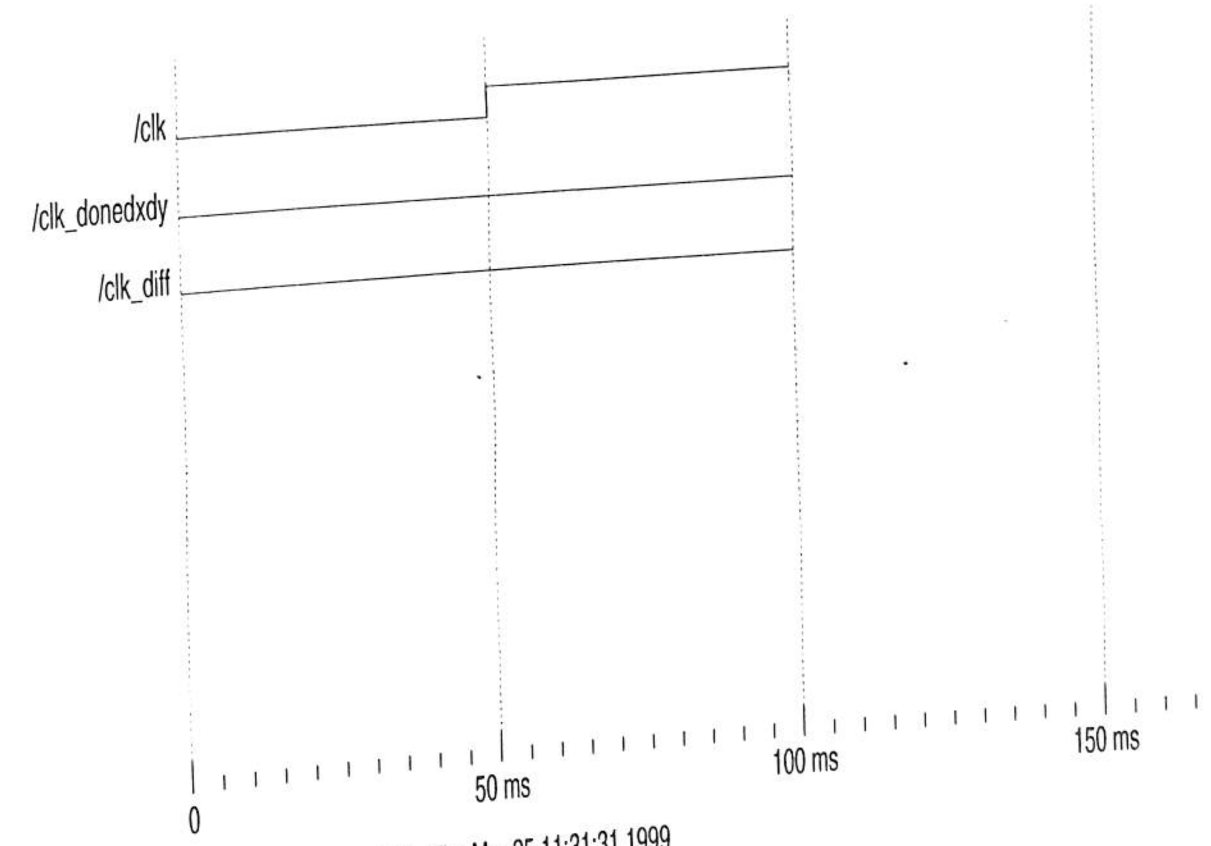


Figure B.2: Results of functional testing of VHDL design of basic cell corresponding to configuration 2 in ModelSim. *clk*, *clk_donedxdy* and clk_diff are the signals declared in VHDL design and they are defined in the caption of Figure B.1.

Configuration 3

Here the cell corresponds to a pixel whose distance values have not been computed and the distance values of some of the neighbors have been computed. But the pixel does not receive its distance values at the next iteration. An instance of this configuration corresponds to the pixel at row 1 and column 5 at iteration 1 in Figure 4.3. The distance values of left bottom neighbor of this pixel have already been computed. But in the next iteration, the pixel does not receive its distance values. This instance was tested in ModelSim and the waveforms are shown in Figure B.3. The waveforms after the clock trigger correspond to next iteration.

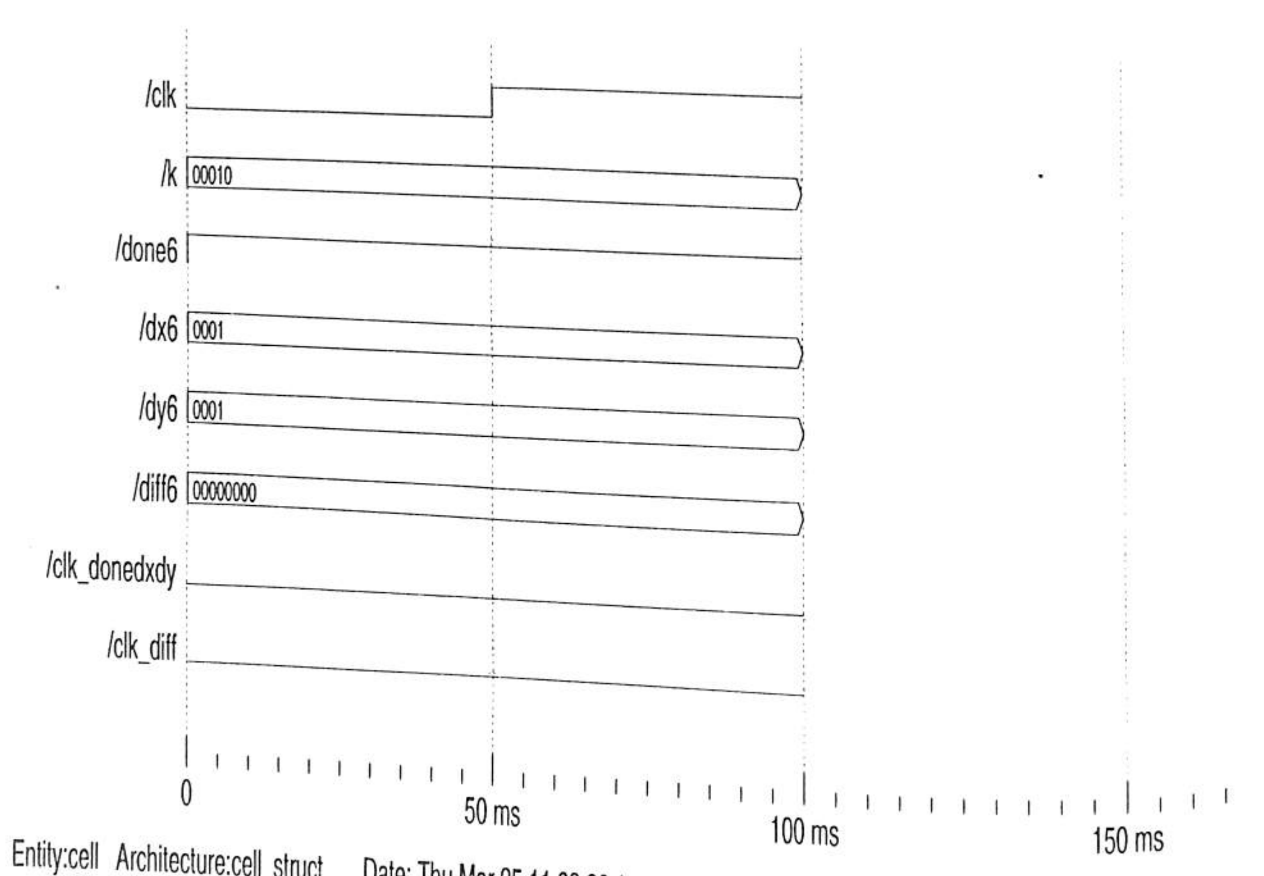


Figure B.3: Results of functional testing of VHDL design of basic cell corresponding to configuration 3 in ModelSim. clk, k, $done6$ etc are signals declared in the VHDL design. clk, $clk_donedxdy$ and clk_diff are defined in the caption of Figure B.1 and the rest are the outputs of storage elements of a neighbor.

Configuration 4

This is similar to the previous configuration but the pixel under consideration receives the distance values at the next iteration. An instance of this configuration corresponds to the pixel at row 1 and column 5 at iteration 2 in Figure 4.3. In the next iteration, the pixel receives its distance values from its left neighbor. The inputs corresponding to this instance are given to the design and the computed values at the output ports of the cell are shown in Figure B.4.

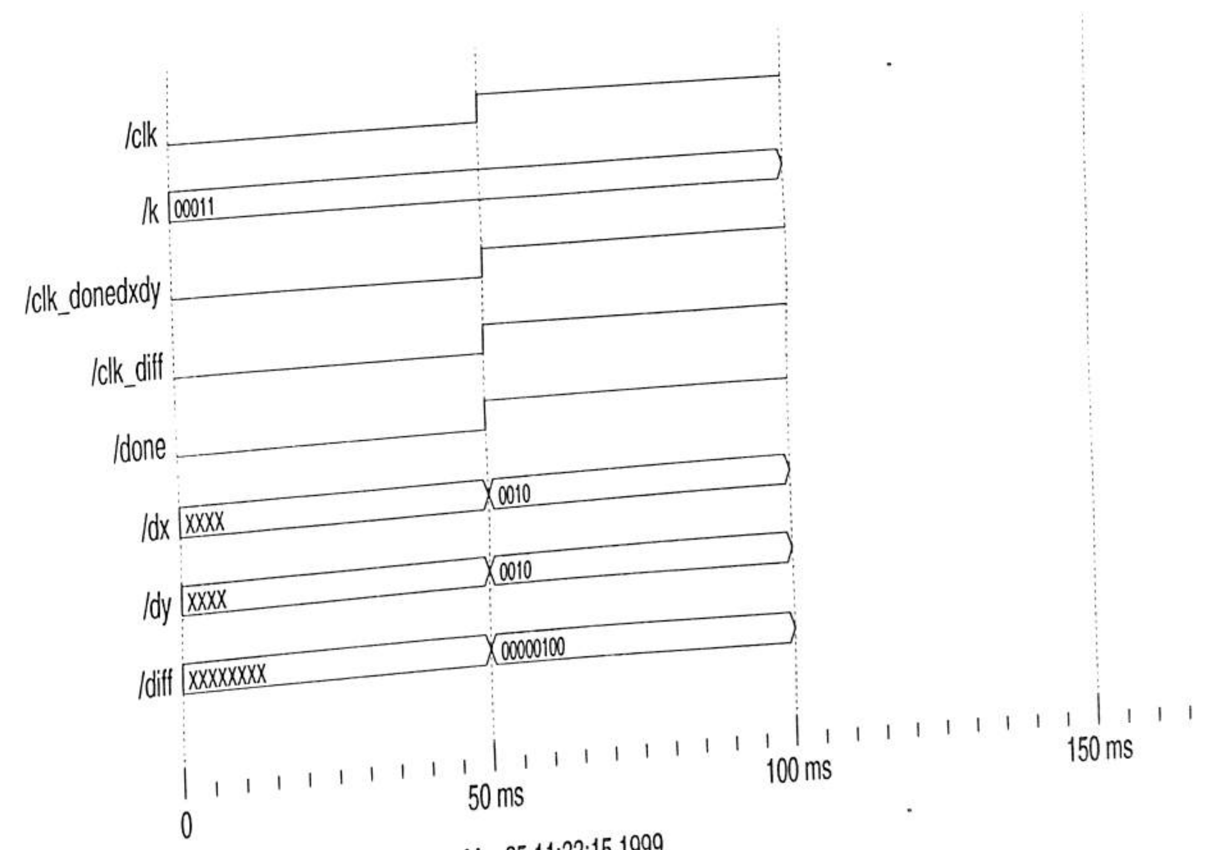


Figure B.4: Results of functional testing of VHDL design of basic cell corresponding to configuration 4 in ModelSim. clk, k etc are the signals declared in the VHDL design. clk, k, $clk_donedxdy$ and clk_diff are defined in the caption of Figure B.1 and the rest are the outputs of the storage elements of the cell.

TH-2715_964102

Appendix C

Xilinx FPGA

An FPGA (Field Programmable Gate Array) consists of rows of gates and programmable connections. Any arbitrary design can be implemented by electrically programming the connections within hours at low cost. FPGA is even reprogrammable and also has a large number of pins/contacts.

Xilinx is a vendor for FPGA devices. The Xilinx FPGA devices are organized as an array of logic blocks surrounded by I/O blocks. The blocks are connected by programmable interconnects. The Xilinx FPGAs use SRAM (Static RAM) technology to control the interconnections, so that the hardware may be reused as well as dynamically reconfigured. The logic block contains both combinational logic and registers. The combinational logic is implemented in small look-up tables. It can be configured to implement any function of five variables. For wider inputs, the logic blocks are easily concatenated. The interconnection scheme includes fixed connections to adjacent blocks, general purpose connections to connect the nearby blocks, and potential connections to long lines running through the array.

Xilinx sells standard devices in multiple families which include XC3000, XC4000, XC5200 and XC9500 series. The first three families consist of SRAM based FPGAs and XC9500 has complex PLDs. All Xilinx devices are general purpose. Any family can implement any type of logic. The maximum frequency of operation of a logic circuit and the FPGA components used for implementing the circuit depend on the chosen Xilinx device. Generally, smaller devices are faster than the larger ones. There are, however, some features that make certain families more appropriate than others. In

TH-2715_964102

XC4000 family, the look-up-table architecture and the dedicated carry structure are very efficient for distributed arithmetic. For digital signal processing applications, this family is appropriate. It also has high-performance, high-capacity FPGAs which provide the benefits of custom CMOS VLSI, while avoiding the initial cost, long development cycle, and inherent risk of a conventional masked gate array. These FPGAs combine architectural versatility, on-chip select-RAM memory with edge-triggered and dual-port modes, increased speed, abundant routing resources and new, sophisticated software to achieve fully automated implementation of complex, high-density, high-performance designs.

BUFG, DFF, FMAP, HMAP, OUTFF, IBUF, and OBUF are some of the components of a device in XC4000 family. BUFGs are global buffers which are used for clock signals. DFF, FMAP and HMAP are kept inside the complex logic blocks of device. DFFs are D type flip-flops which are used for registers and flip-flops in a given design whose outputs are fed to the combinational logic of the design. FMAP and HMAP corresponds to F and H look-up-tables. The F look-up-table has 4 inputs and 1 output and H look-up-table has 3 inputs and 1 output. OUTFF, IBUF and OBUF are kept inside the I/O blocks of the device. The combinational logic of the given design is mapped onto these tables. OUTFFs are output flip-flops which ensure minimum delay of signals to the output pins. These are assigned to those memory elements in the design whose outputs are directly taken as the outputs of the design. IBUF is input buffer which is placed between the input pin and the internal FPGA logic. An IBUF is needed for every input signal. Similarly, OBUF is output buffer which is placed before an output pin. OBUF is needed for every signal that leaves the device but it is not needed for an output signal that is taken from OUTFF.

The details of the device XC4036XL are as follows:

Maximum logic gates	:	36000
Complex logic blocks	:	1296
I/O blocks	:	288
Flip-flops	:	3168
Fastest speed grade	:	-2
Clock to pad (ns)	:	6.5

Package BG432 has 225 pins.

TH-2715_964102

The Xilinx development system software reads the schematic or HD[illegible] circuit. The software first partitions the design into logic blocks, then finds a near-optimal placement for each block, and finally selects the interconnect routing. Once the design is complete, a serial bit stream is produced which can be downloaded into the hardware. This is generally called as programming the selected Xilinx FPGA device. Inside the device, these configuration bits control or define the combinatorial circuitry, flip-flops, interconnect structure and the I/O buffers. They also define the registers, input threshold and output slew rate.

References

[AAS85] N. Ahuja, B. An, and B. Schachter. Image representation using Voronoi tessellation. *Computer Vision, Graphics and Image Processing*, 29:286–295, 1985.

[ACL81] C. Arcelli, L.P. Cordella, and S. Levialdi. From local maxima to connected skeleton. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3(2):134–143, March 1981.

[AdB85] C. Arcelli and G. Sanniti di Baja. A width-independent fast thinning algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(4):463–474, July 1985.

[AdB86] C. Arcelli and G. Sanniti di Baja. Computing Voronoi diagrams in digital pictures. *Pattern Recognition Letters*, 4:383–389, October 1986.

[AdB88] C. Arcelli and G. Sanniti di Baja. Finding local maxima in a pseudo-Euclidean distance transform. *Computer Vision, Graphics and Image Processing*, 43:361–367, 1988.

[AdB89] C. Arcelli and G. Sanniti di Baja. A one-pass two operation process to detect the skeletal pixels on the 4-distance transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(4):411–414, 1989.

[AdB92] C. Arcelli and G. Sanniti di Baja. Ridge points in Euclidean distance maps. *Pattern Recognition Letters*, 13:237–243, 1992.

[AdB93] C. Arcelli and G. Sanniti di Baja. Euclidean skeleton via centre-of-maximal-disc extraction. *Image and Vision Computing*, 11(3):163–173, April 1993.

TH-2715_964102

[AdB96] C. Arcelli and G. Sanniti di Baja. Skeletons of planar patterns. In T.Y. Kong and A. Rosenfeld, editors, *Topological Algorithms for Digital Image Processing*, 99–143. Elsevier, 1996.

[ADP96a] R. Azencott, F. Durbin, and J. Paumard. Multiscale identification of buildings in compressed large aerial scenes. *Proceedings of IEEE International Conference on Pattern Recognition*, 3:974–978, 1996.

[ADP96b] R. Azencott, F. Durbin, and J. Paumard. Robust recognition of buildings in compressed large aerial scenes. *Proceedings of IEEE International Conference on Image Processing*, 2:617–620, 1996.

[AF95] M. Albanesi and M. Ferretti. Systolic merging and ranking of votes for the generalized Hough transform. In N. Ranganathan, editor, *VLSI and Parallel computing for Pattern Recognition and Artificial Intelligence*, 145–172. 1995.

[Ahu80] N. Ahuja. Dot pattern processing using Voronoi polygons as neighborhoods. *Proceedings of Fifth International Conference on Pattern Recognition*, 1122–1127, 1980.

[BA91] J.W. Brandt and V.R. Algazi. Computing a stable, connected skeleton from discrete data. *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1991.

[BGKW95] H. Breu, J. Gil, D. Kirkpatrik, and M. Werman. Linear time Euclidean distance transform algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(5):529–533, May 1995.

[Bha95] J. Bhasker. *A VHDL primer*. Prentice Hall, 1995.

[Blu64] H. Blum. A transformation for extracting new descriptors of shape. *Proceedings of Symposium on Models for the Perception of Speech and Visual Forms*, 1964.

[BM98] M.K. Butt and P. Maragos. Optimum design of chamfer distance transforms. *IEEE Transactions on Image Processing*, 7(10):1477–1484, 1998.

[Bor84] G. Borgefors. Distance transformations in arbitrary dimensions. *Computer Vision, Graphics and Image Processing*, 27:321–345, 1984.

TH-2715_964102

[Bor86] G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing*, 34:344–371, 1986.

[Bor89] G. Borgefors. A comment on "A note on 'Distance transformations in digital images'". *Computer Vision, Graphics and Image Processing*, 47:89–91, 1989.

[Bor91] G. Borgefors. Another comment on "A note on 'Distance transformations in digital images'". *CVGIP: Image Understanding*, 54(2):301–306, September 1991.

[Bou90] T.E. Boult. Dynamic digital distance maps in two dimensions. *IEEE Transactions on Robotics and Automation*, 6(5):590–597, October 1990.

[CC94] L. Chen and H.Y.H. Chuang. A fast algorithm for Euclidean distance maps of a 2-D binary image. *Information Processing Letters*, 25–29, 1994.

[CCB96] Y. Chehadeh, D. Coquin, and Ph. Bolon. A skeletonization algorithm using chamfer distance transformation adapted to rectangular grids. *Proceedings of IEEE International Conference on Pattern Recognition*, 1996.

[CCNC97] P.P. Chaudhuri, D.R. Chowdhury, S. Nandi, and S. Chattopadhyay. *Additive Cellular Automata - Theory and Applications*, volume 1. IEEE Computer Society Press, California, 1997.

[CL85] H.Y.H. Chuang and C.C. Li. A systolic array processor for straight line detection by modified Hough transform. *Proceedings of CAPAIDM*, 300–304, 1985.

[CY96] C.H. Chen and D.L. Yang. Fast algorithm and its systolic realisation for distance transformation. *IEE Proceedings - Computers and Digital Techniques*, 143(3):168–173, May 1996.

[Dan80] P.E. Danielsson. Euclidean distance mapping. *Computer Vision, Graphics and Image Processing*, 14:227–248, 1980.

[dB94] G. Sanniti di Baja. Well-shaped, stable and reversible skeletons from the (3,4)-distance transform. *Journal of Visual Communication and Image Representation*, 5(1):107–115, March 1994.

[dBT94] G. Sanniti di Baja and E. Thiel. (3,4)-weighted skeleton decomposition for pattern representation and description. *Pattern Recognition*, 27(8):1037–1049, 1994.

[dBT96] G. Sanniti di Baja and E. Thiel. Skeletonization algorithm running on path-based distance maps. *Image and Vision Computing*, 14:47–57, 1996.

[DC87] P.P. Das and P.P. Chakrabarti. Distance functions in digital geometry. *Information Sciences*, 42:113–136, 1987.

[DKF94] F. Dehne, C. Kenyon, and A. Fabri. Scalable and architecture independent geometric algorithms with high probability optimal time. *Proceedings of IEEE Symposium on Parallel and Distributed Processing*, 586–593, 1994.

[Dor86] L. Dorst. Pseudo-Euclidean skeletons. *Proceedings of IEEE International Conference on Pattern Recognition*, 286–288, 1986.

[Egg98] H. Eggers. Two fast Euclidean distance transformations in z^2 based on sufficient propagation. *Computer Vision and Image Understanding*, 69(1):106–116, Jan. 1998.

[FD77] H. Freeman and L.S. Davis. A corner finding algorithm for chain coded curves. *IEEE Transactions on Computers*, C-26:297–303, 1977.

[FNK92] H. Fujisawa, Y. Nakano, and K. Kurino. Segmentation methods for character recognition: From segmentation to document structure analysis. *Proceedings of the IEEE*, 80(7):1079–1092, July 1992.

[FvDFH97] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics - Principles and Practice*. Addison-Wesley, second edition, 1997.

[GF96] Y. Ge and J.M. Fitzpatrick. On the generation of skeletons from discrete Euclidean distance maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(11):1055–1066, November 1996.

[Gra72] G.H. Granlund. Fourier preprocessing for hand printed character recognition. *IEEE Transactions on Computers*, C-21:195–201, Feb. 1972.

[Guh94] S. Guha. An optimal mesh computer algorithm for constrained Delaunay triangulation. *Proceedings of IEEE International Symposium on Parallel Processing*, 102–109, 1994.

[GW92] R.C. Gonzalez and R.E. Woods. *Digital Image Processing*. Addison-Wesley Publishing Company, 1992.

[Hir96] T. Hirata. A unified linear-time algorithm for computing distance maps. *Information Processing Letters*, 58:129–133, 1996.

[HKR93] D.P. Huttenlocher, G.A. Klanderman, and W.J. Rucklidge. Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:850–863, September 1993.

[IO98] T. Ikenaga and T. Ogura. CAM^2: A highly parallel two-dimensional cellular automaton architecture. *IEEE Transactions on Computers*, 47(7), July 1998.

[IOO91] S. Impedevo, L. Ottaviano, and S. Occhingro. Optical character recognition - A survey. *Int. J. Pattern Recognition and Artificial Intelligence*, 1991.

[JC90] B.K. Jang and R.T. Chin. Analysis of thinning algorithms using mathematical morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6):541–551, June 1990.

[KAGER82] S.Y. Kung, K.S. Arun, R.J. Gal-Ezer, and D.V. Bhaskar Rao. Wavefront array processor: Language, architecture, and applications. *IEEE Transactions on Computers, Special Issue on Parallel and Distributed Computers*, 11(C-31):1054–1066, November 1982.

[KHL86] S.Y. Kung, J.N. Hwang, and S.C. Lo. Mapping digital signal processing algorithms onto VLSI systolic/wavefront arrays. *Proceedings of 12th Annual Asilomar Conference on Signals, Systems and Computers*, 6–12, November 1986.

[KK87] F. Klein and O. Kubler. Euclidean distance transformations and model-guided image interpretation. *Pattern Recognition*, 5:19–29, 1987.

TH-2715_964102

REFERENCES

[KK92] M.N. Kolountzakis and K.N. Kutulakos. Fast computation of the Euclidean distance maps for binary images. *Information Processing Letters*, 43:181–184, 1992.

[KSI98] K. Kise, A. Sato, and M. Iwata. Segmentation of page images using the area Voronoi diagram. *Computer Vision and Image Understanding*, 70(3):370–382, June 1998.

[Kun88] S.Y. Kung. *VLSI Array Processors*. Prentice Hall, New Jersey, 1988.

[KZ96] R. Klette and P. Zamperoni. *Handbook of Image Processing Operators*. John Wiley and Sons, 1996.

[Lat91] J.C. Latombe. *Robot Motion Planning*. Kluwer, Boston, MA, 1991.

[LL92] F. Leymarie and M.D. Levine. Simulating the grassfire transform using an active contour model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(1):56–75, Jan. 1992.

[LS90] H.C. Liu and M.D. Srinath. Partial shape classification using contour matching in distance transformation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(11), November 1990.

[LW93] E.H. Liang and E.K. Wong. An efficient method for obtaining morphological skeleton. *Pattern Recognition Letters*, 14:689–695, 1993.

[Man96] M. Morris Mano. *Digital Design*. Prentice Hall of India, 1996.

[MMY74] T. Mori, S. Mori, and K. Yamamoto. Feature extraction method based on field effect method. *Trans. IECE Japan*, 57-D:308–315, 1974.

[Moh92] P.S. Moharir. *Pattern Recognition Transforms*. Research Studies Press Ltd., England, 1992.

[MSY92] S. Mori, C.Y. Suen, and K. Yamamoto. Historical review of OCR research and development. *Proceedings of the IEEE*, 80(7):1029–1058, July 1992.

[MT95] J.L. Moigne and J.C. Tilton. Refining image segmentation by integration of edge and region data. *IEEE Transactions on Geoscience and Remote Sensing*, 33(3):605–615, May 1995.

[Nag92] G. Nagy. At the frontiers of OCR. *Proceedings of the IEEE*, 80(7):1093–1100, July 1992.

[NFMM85] D. Nicolas, J. Francis, S. Marc, and D. Michel. VLSI architecture for a one chip video median filter. *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, 26.7.1–26.7.4, 1985.

[NGC92] C.W. Niblack, P.B. Gibbons, and D.W. Capson. Generating skeletons and centerlines from the distance transform. *CVGIP:Graphical Models and Image Processing*, 54(5):420–437, September 1992.

[Ofl83] K. Oflazer. Design and implementation of a single chip 1-D median filter. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 31(5):1164–1168, October 1983.

[O'R94] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1994.

[PA75] T. Pavlidis and F. Ali. Computer recognition of handwritten numerals by polynomial approximations. *IEEE Transactions on Systems, Man Cybernetics*, SMC-5:610–614, 1975.

[Pag92] D.W. Paglieroni. Distance transforms: Properties and machine vision applications. *CVGIP: Graphical Models and Image Processing*, 54(1):56–74, January 1992.

[Pav86] T. Pavlidis. A vectorizer and feature extractor for document recognition. *Computer Vision, Graphics and Image Processing*, 35:111–127, 1986.

[PS85] F.P. Preparata and M.I. Shamos. *Computational Geometry - An Introduction*. Springer-Verlag, 1985.

[Rag92] I. Ragnemalm. Neighborhoods for distance transformations using ordered propagation. *CVGIP: Image Understanding*, 56(3):399–409, November 1992.

[RD95] N. Ranganathan and K.B. Doreswamy. A VLSI chip for computing the medial axis transform of an image. *Proceedings of IEEE International Conference on Computer Architecture and Machine Perception*, 36–43, 1995.

TH-2715_964102

[RK82] A. Rosenfeld and A.C. Kak. *Digital Picture Processing.* Academic Press, New York, 1982.

[RMS95] N. Ranganathan, R. Mehrotra, and S. Subramanian. A high speed systolic architecture for labeling connected components in an image. *IEEE Transactions on Systems, Man Cybernetics*, 25:415–423, March 1995.

[Roo94] Thomas Roos. Maintaining Voronoi diagrams in parallel. *Proceedings of IEEE International Conference on System Sciences*, 179–186, 1994.

[RP66] A. Rosenfeld and J.L. Pfaltz. Sequential operations in digital picture processing. *Journal of Association of Computing Machines*, 13:471–494, 1966.

[RP68] A. Rosenfeld and J.L. Pfaltz. Distance functions on digital pictures. *Pattern Recognition*, 1:33–61, 1968.

[RP88] Hillel Rom and Shmuel Peleg. Image representation using Voronoi tesselation: Adaptive and secure. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 282–285, 1988.

[RS86] A. Rastogi and S.N. Srihari. Recognizing textual blocks in document images using the Hough transform. Technical Report TR 86-01, Dept. of Computer Science, SUNY at Buffalo, 1986.

[RSI91] N.S.V. Rao, N. Stoltzfus, and S.S. Iyengar. A retraction method for learned navigation in unknown terrains for a circular robot. *IEEE Transactions on Robotics and Automation*, 7:699–707, 1991.

[Ruc95] W.J. Rucklidge. Locating objects using the Hausdorff distance. *Proc. of the IEEE 5th Int. Conf. on Computer Vision*, 457–464, 1995.

[Ruc96] William Rucklidge. *Efficient Visual Recognition Using the Hausdorff Distance.* LNCS 1173, Springer Verlag, 1996.

[Rus95] John C. Russ. *The Image Processing Handbook.* CRC Press, second edition, 1995.

[SBB+92] J. Schurmann, N. Bartneck, T. Bayer, J. Franke, E. Mandler, and M. Oberlander. Document analysis - From pixels to contents. *Proceedings of the IEEE*, 80(7):1101–1119, July 1992.

TH-2715_964102

ch89] Robert J. Schalkoff. *Digital Image Processing and Computer Vision.* John Wiley and Sons, Inc., 1989.

ch90] Robert J. Schilling. *Fundamentals of Robotics - Analysis and Control.* Prentice Hall of India, 1990.

er82] J. Serra. *Image Analysis and Mathematical Morphology.* Academic Press, New York, 1982.

ho89] R. Shonkwiler. An image algorithm for computing the Hausdorff distance efficiently in linear time. *Information Processing Letters*, 30:87–89, January 1989.

ho91] R. Shonkwiler. Computing the Hausdorff set distance in linear time for any l_p point distance. *Information Processing Letters*, 38:201–207, May 1991.

ip99] Moshe Sipper. The emergence of cellular computing. *IEEE Computer*, July 1999.

M92] F.Y. Shih and O.R. Mitchell. A mathematical morphology approach to Euclidean distance transformation. *IEEE Transactions on Image Processing*, 1(2):197–204, 1992.

N98] N. Sudha and S. Nandi. A parallel skeletonization algorithm and its VLSI architecture. *Proceedings of the 5th International Conference on High Performance Computing (HiPC'98)*, 65–72, Dec. 1998.

NBS98] N. Sudha, S. Nandi, P.K. Bora, and K. Sridharan. Efficient computation of Euclidean distance transform for applications in image processing. *Proceedings of IEEE Region Ten Conference on Global Connectivity in Energy, Computer, Communication and Control (TENCON'98)*, 1:49–52, Dec. 1998.

NS99] N. Sudha, S. Nandi, and K. Sridharan. A parallel algorithm for the construction of Voronoi diagram and its VLSI architecture. *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, 1683–1688, May 10-15 1999.

[SP90] F.Y. Shih and C.C. Pu. Medial axis transformation with single-pixel and connectivity preservation using Euclidean distance computation. *Proceedings of IEEE International Conference on Pattern Recognition*, 1990.

[SP91] F.Y. Shih and C.C. Pu. A mixima-tracking method for skeletonization from Euclidean distance function. *Proceedings of International Conference on Tools for AI*, 1991.

[SP95] F.Y. Shih and C.C. Pu. A skeletonization algorithm by maxima tracking on Euclidean distance transform. *Pattern Recognition*, 28(3):331–341, 1995.

[SP97] Raul E. Sequeira and Francoise J. Preteux. Discrete Voronoi diagrams and the SKIZ operator - A dynamic algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(10):1165–1170, October 1997.

[SW89] S. Srihari and D. Wang. Classification of newspaper image blocks using texture analysis. *Computer Vision, Graphics and Image Processing*, 42:327-352, 1989.

[Tam78] H. Tamura. A comparison of line thinning algorithms from digital computer view point. *Proc. 4th Int. Joint Conf. Patt. Recog.*, 715–719, 1978.

[TJ90] Mihran Tuceryan and Anil K. Jain. Texture segmentation using Voronoi polygons. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):211–216, February 1990.

[TTT94] P.G. Tzionas, P.G. Tsalides, and A. Thanailakis. A new, cellular automaton-based, nearest neighbor pattern classifier and its VLSI implementation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2(3):343–354, September 1994.

[TTT97] P.G. Tzionas, A. Thanailakis, and P.G. Tsalides. Collision-free path planning for a diamond-shaped robot using two-dimensional cellular automata. *IEEE Transactions on Robotics and Automation*, 13(2):237–250, April 1997.

[Ver92] F.J. Verbeek. Deformation correction using Euclidean contour distance maps. *Proceedings of IEEE International Conference on Pattern Recognition*, 347–350, 1992.

TH-2715_964102

[Vos88] A.M. Vossepoel. A note on distance transformations in digital images. *Computer Vision, Graphics and Image Processing*, 43:88–97, 1988.

[Wol83] S. Wolfram. Statistical mechanics of cellular automata. *Rev. Mod. Phys.*, 55:601–644, July 1983.

[YC95] X. Yi and O.I. Camps. Line feature-based recognition using Hausdorff distance. *Proceedings of IEEE International symposium on Computer Vision*, 79–84, 1995.

[YCZP96] J. You, H.A. Cohen, W.P. Zhu, and E. Pissaloux. A robust and real-time texture analysis system using a distributed workstation cluster. *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, 4:2207–2210, 1996.

[Ye88] Qin-Zhong Ye. The signed Euclidean distance transform and its applications. *Proceedings of IEEE International Conference on Pattern Recognition*, 495–499, 1988.

[YI86] M. Yamashita and T. Ibaraki. Distances defined by neighborhood sequences. *Pattern Recognition*, 19(3):237–246, 1986.

[YPC95] J. You, E. Pissaloux, and H.A. Cohen. A hierarchical image matching scheme based on the dynamic detection of interesting points. *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, 4:2467–2470, 1995.

List of Publications

1. N. Sudha, S. Nandi, P.K. Bora and K. Sridharan, "Efficient computation of Euclidean distance transform for applications in image processing", *Proceedings of IEEE Region Ten Conference on Global Connectivity in Energy, Computer, Communication and Control (TENCON'98)*, volume 1, pages 49-52, December, 1998.

2. N. Sudha and S. Nandi, "A parallel skeletonization algorithm and its VLSI architecture", *Proceedings of the 5th International Conference on High Performance Computing (HiPC'98)*, pages 65-72, December, 1998.

3. N. Sudha, S. Nandi and K. Sridharan, "A parallel algorithm for the construction of Voronoi diagram and its VLSI architecture", *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1683-1688, May, 1999.

4. N. Sudha, S. Nandi and K. Sridharan, "A Cellular Architecture for Euclidean Distance Transformation", to appear in *IEE Proceedings - Computers and Digital Techniques*, 2000.

TH-2715_964102