USC-SIPI REPORT #133

A Digital Optical Cellular Image Processor (DOCIP): Theory, Architecture and Implementation

by

Kung-Shiuh Huang

Signal and Image Processing Institute
UNIVERSITY OF SOUTHERN CALIFORNIA
Department of Electrical Engineering-Systems
3740 McClintock Avenue, Room 404
Los Angeles, CA 90089-2564 U.S.A.

Dedication

To

MY FAMILY and MY TEACHERS,

who teach me to love people and to solve problems

as simply as possible.

Acknowledgements

I am deeply indebted to my thesis supervisors, Dr. B. Keith Jenkins and Dr. Alexander A. Sawchuk, for their invaluable guidance, help, knowledge and moral support throughout the course of this work. I owe special thanks to Dr. Ramalingam Chellappa, who also served on my thesis committee, for his constant encouragement and useful comments. Special thanks also to Dr. Edward K. Blum, who served as an outside member on my thesis committee, for his enthusiasm and helpful ideas, especially in the weekly discussion at the early stage of this thesis. Other USC faculty, Dr. Cauligi S. Raghavendra, Dr. Richard Leahy, Dr. Prasanna Kumar, Dr. Kai Hwang, Dr. Elsa Garmire and Dr. Armand Tanguay, Jr. have been sources of inspiration.

I gratefully acknowledge Dr. Pierre Chavel, Mr. Jimmy Wang, Dr. Allan G. Weber, Mr. Chein-Hsun Wang, Dr. Isaia Glaser, Mr. Hung-Jen Chang, Mr. Toy Mayeda, Mr. Ken Prager, Mr. William Kressen, Mr. Chih-Lin Wang, Ms. Clare Stassen and Mr. Wai-Chi Fang for their ideas, discussions and technical support. I also owe thanks to Dr. Timothy C. Strand, Dr. Hua-Kuang Liu, Dr. Demetri Psaltis, Mr. Ken Hsu, Dr. Richard P. Feynman, Dr. Ahmed Louri, Dr. Hungwen Li, Dr. Pochi Yeh, Dr. Arthur Chiou, Dr. George Eichmann, Dr. Yao Li, Dr. Sing H. Lee, Dr. Toyohiko Yatagai, Dr. Shing-Fong Lin and Dr. Chiung-Sheng Wu for their encouragement and interesting discussions. I would also like to thank

Mr. Yih-Yuh Chen for his helpful discussions in numerous problems of physics and mathematics.

Thanks are extended to all other members of the Signal and Image Processing Institute, especially Gloria Bullock, Linda Varilla, Ray Schmidt, Yih-Herng Chuang, Aseem Vaid, Robert Frankot, Herb Barad, Dan Antzoulatos, Chun-Chau Lin, Zhou Yitong, Shiaw-Shiang Jiang, Qin-Feng Zheng, Zeév Lichtenstein, Yu-Hsiung Hsiao, Hao Jinchi, Shankar Chatterjee, Xiao-hong Yang, Ted Broida, Jason Young, Lily Chen, Charlie Kuznia, and Dong Wiley for their help, friendship and many interesting discussions; also to the members of the USC Family Student Housing, especially Rong-Feng & Yu-Hua Chang, Min-Hsiu & Mei-Chu Chiang, Hwang-Cheng & Yuh-Chyng Wang and Changhaw & Huiling Lin.

This research was supported by the the Air Force Office of Scientific Research under grant AFOSR-84-0181, by the Office of Naval Research under contract N00014-86-K-0602, by an IBM graduate fellowship, and by an SPIE scholarship. Their financial support is greatly appreciated.

The greatest thanks of all go to my family, my parents Ching-Ming and Chien-Yueh, my brothers Kuan-Tase and Kung-Hui, my sister Tase-Mei, my parents-in-law Chin-Fuang and Shiu-Ken, and especially my wife Shin-Yuan. Dr. Kuan-Tase Huang is the one sparking my interest in the field of computing and having the greatest influence on my study.

Contents

List of Figures

List of Tables

Abstract

Is there a simple unified complete theory of parallel binary digital image processing? Can this theory suggest new parallel algorithms and architectures for image processing and numerical computation? Can these algorithms and architectures be implemented by optical computing techniques? This thesis attempts to answer these questions.

A binary image algebra (BIA), built from five elementary images and three fundamental operations, serves as a unified theory of parallel binary image processing and a spatial logic of parallel optical computing. It also leads to a formal parallel language approach to the design of parallel binary image processing and parallel binary arithmetic algorithms. Digital optical cellular image processors (DOCIPs), based on cellular automata and cellular logic architectures, implement parallel algorithms of BIA efficiently. An algebraic structure provides a link between the algorithms of BIA and architectures of DOCIP.

Optical computing suggests an efficient and high-speed implementation of the DOCIP architectures because of its inherent parallelism and 3-D global interconnection capabilities. The use of optical interconnections permits a two-dimensional cellular hypercube (DOCIP-hypercube) topology to be implemented without paying a large penalty in chip area (the cellular hypercube interconnections are space-invariant which implies a low hologram complexity); it also enables images to be input to and output from the machine in parallel. A computer-controlled system has been constructed to fabricate multi-facet interconnection holograms for 3-D optical circuits. A prototype DOCIP system has been implemented to demonstrate the concept of the DOCIP architecture. Experimental results are presented.

Chapter 1

Introduction

1.1 Motivation

This research is stimulated by the search for a simple parallel digital optical architecture for image processing. However, image processing has no standard unified theory, so many image processing algorithms and architectures exist in a state of chaos. Hence, we first develop a simple unified consistent theory of parallel binary image processing (covering both algorithms and architectures), and then consider its implementation on digital optical processors.

1.1.1 Toward A Unified Theory of Image Processing

Image processing (including image analysis) is concerned with the manipulation and analysis of images (or pictures) by processors (or computers) [Chellappa85a, Chellappa85b, Rosenfeld82, Pratt78, Stucki79]. It cuts across various application areas, such as medical diagnosis, machine vision, planetary physics, and industrial inspection. But as useful as these applications are, the concepts that give rise to them are deep and the tools are complicated. Furthermore, more image processing algorithms and architectures are proposed in the literature each year. In English alone, thousands of technical papers are being published each year in more than 10 technical journals and numerous conferences and workshops. Unfortunately, little use is made of any unified standard systematic mathematical structure in both algorithms and architectures [Duff81]. This results in the following drawbacks:

1. Each research group has its own notation and tools.
2. Algorithms and architectures for image processing are not well matched.
3. It is very difficult to evaluate and compare so many different image processing algorithms and architectures.
4. It is very difficult for a user to choose the best algorithm and the best architecture for his own purpose.
5. Overall, the cost of development and the complexity of research in both algorithms and architectures of image processing are increasing.

To improve on this chaotic situation in image processing, we propose

- To develop a unified consistent systematic mathematical structure for image processing algorithms and architectures.

To accomplish the above goal, we first pose the question:

- What kind of mathematical structure is appropriate to serve as a theoretical framework of image processing (covering algorithms and architectures)?

Algebra is a foundation of mathematics and successfully provides fundamental tools for many disciplines of science and technology. For example, the theory of computing machines is primarily based on modern (or abstract) algebra. Thus, it is possible that an appropriate algebraic structure should provide a framework of theoretical image processing (covering both algorithms and architectures).

Definition of Algebraic Structure

An algebraic structure (or algebra) [Birkhoff65, Birkhoff70, Gilbert76] is a pair (or system) $A = (S, F)$ where

- S is a set, and
- F is a family of operations which are functions:

$$f : S^k \rightarrow S,$$

  and k is a finite non-negative integer.

Remark: For any finite non-negative integer k, we define a k-ary operation on S as an operation which is a function $f : S^k \rightarrow S$. Thus, a unary (or 1-ary) operation on S is simply a function on S to S. A binary (or 2-ary) operation on S is a function on S^2 to S. For completeness, we define a nullary (or 0-ary) operation on S to be a particular element of S.

Therefore, the problem to be solved is essentially to find an "appropriate" algebraic structure (S, F) for parallel binary image processing, i.e. to search for S and F, and its "efficient" hardware implementation.

1.1.2 Toward An Era of Digital Parallel Optical Computing

Optical computing is the use of optical systems to perform computations on one-dimensional or multidimensional data [Sawchuk84, ProcIEEE84]. Due to its parallel processing ability and large data capacity, analog optical computing has received a great deal of research interest since the 1960's [Goodman68]. While many analog optical systems can perform specific operations (e.g. convolution and correlation) with extremely high throughput rates, they face several severe limitations in accuracy, flexibility, and programmability. In order to eliminate or reduce these limitations, an obvious approach is to utilize discrete, instead of analog, signal levels [Sawchuk84, Jenkins84a, Jenkins84b]. To achieve such digital optical computing, we have at least 3 possible logic systems: residue logic [Huang79, Psaltis79, Horrigan79, Tai79], multilevel logic [Abraham86, Tao86, Arrathoon86, Hurst86], and binary logic. Because it is much easier to make reliable 2-level devices for binary logic and only $log_2 k$ of them are needed to to represent k levels, optical technology may be able to implement binary computing as electronics does. Thus, in this thesis we will only consider parallel binary optical computing. The basic purpose of this proposed research essentially follows from the goal proposed in [Sawchuk84]: "to perform binary digital computing with optical systems having photons as the primary information-carrying medium, avoiding electronic logic, and having as few photon-electron and electron-photon conversions as possible." The potential advantages of binary parallel optical computing architectures have been summarized in [Jenkins86]:

- They offer flexibility of operations — numerical, symbolic or logical, compared to analog or discrete multilevel processors;
- They have binary digital accuracy and dynamic range;
- They offer computing architectures very different from electronic very large scale integration (VLSI): they permit global interconnections and parallel input-output compared to the local interconnections, clock-skew problems, pin-in/pin-out and bus limitations of very large scale integration (VLSI);
- They utilize the 2-D parallel nature of optical device arrays and low interaction of optical signals for interconnections in 3-D; and
- They offer the possibility of high throughput and processing speed with arrays of fast optical switches that are being developed.

The reasons that we use optics to complement electronics are highlighted on Table 1.1.

Technology	Electronics	Optics
Information carrier	Electron	Photon (no mass, no charge)
Interconnections	Primarily 2-D	3-D
Bandwidth	Low	High
Crosstalk	High	Low
Interconnection Connnectivity	Local	Global/local
Clock-skew Problems	Yes (or serious)	No (or little)
Pin-in/pin-out limitations	Yes (or serious)	No (or little)
Switching	Easy	Difficult

Table 1.1: A comparison between electronic and optical computing technologies.

To take advantage of binary parallel optical computing, we must find out:

- What kinds of problems are appropriate for optical systems?

These problems have to utilize some of the above advantages. One most significant example is the field of image processing. This is due to the following reasons:

- The computational requirements in image processing are extreme;
- Large amounts of parallelism are involved in image processing;
- Images are inherently two-dimensional;
- Global information extraction from images require global communication;
- The communication and interconnection requirements between cells and memory, and input/output of data to the cells are extremely high;
- Images are inherently formed by optical waves; and
- Cellular logic architectures which put a cell (or processing element) behind each pixel of an image are appropriate architectures of parallel image processors and suited to implementation on optics.

Philosophically, the parallel processing ability and free interconnection capabilities of optics are well-matched to parallel image processing tasks and can dramatically reduce the computation time compared with today's serial computers. Thus, digital cellular logic image processors may be well suited to digital optical computing techniques. The problem then becomes how to design a simple and efficient optical cellular logic processor for image processing.

1.2 Objectives

This research cuts across the fields of image processing, parallel processing and optical computing. It includes both theoretical and experimental work, and covers both algorithmic and architectural designs. The objectives of this work are:

- To develop a consistent systematic complete algebraic theory, called binary image algebra (BIA), to serve as
  - A unified theory of parallel binary image processing, and
  - A spatial logic of parallel digital optical computing.
- To devise highly parallel optical computing architectures, namely digital optical cellular image processor (DOCIP) architectures, for
  - Implementing BIA parallel algorithms effectively, and
  - Utilizing the capabilities of optics.
- To experimentally implement a prototype DOCIP system for
  - Demonstrating the concept of the DOCIP architecture, and
  - Demonstrating the feasibility of optical holographic interconnections and digital optical computers.

1.3 Thesis Organization

Chapter 2 discusses past work related to this research: Section 2.1 reviews previous work on image algebra; Section 2.2 reviews previous work on cellular logic architectures; Section 2.3 reviews previus work on digital optical cellular logic processors.

Chapter 3 gives the framework of BIA: Section 3.1 discusses the philosophy of BIA; Section 3.2 gives the basic definitions of BIA; Section 3.3 presents two fundamental principles which prove the completeness of BIA. The tedious proof of a lemma and theorems for the two fundamental principles is presented in Appendix A.

Chapter 4 describes some image processing applications of BIA: Section 4.1 reviews basic properties of images and image transformations, and derives from them some standard image operations; Section 4.2 gives some examples of special cases; Section 4.3 gives some useful theorems and examples for low level vision operations, including morphological filtering, shape recognition, "salt" and "pepper" noise removal, size and location verification; Section 4.4 describes a shift and scale invariat pattern recognition algorithm for a certain class of problems. The basic properties of BIA fundamental operations and standard image operations are contained in Appendix B. Proofs of the theroems for low level vision are presented in Appendix C.

Chapter 5 discusses the relationship of BIA and other computing theories: Section 5.1 formalizes symbolic substitution as compact BIA expressions and suggests some BIA algebraic techniques for describing and comparing symbolic substitution algorithms; Section 5.2 describes cellular logic as a speical case of BIA; Section 5.3 describes Boolean logic as a serial version of BIA; Section 5.4 discusses the relationship with linear shift invariant system theory, convolution and correlation; Section 5.5 describes some standard algebraic structures supported by BIA.

Chapter 6 presents the implementation of BIA on digital optical cellular image processor (DOCIP) architectures, including the DOCIP-array and DOCIP-hypercube: Section 6.1 discusses the design principles of the DOCIP machines; Section 6.2 describes the general organization of the DOCIPs, including an algebraic description and a general description; Section 6.3 describes the array of cells, discusses the implementation of the three fundamental operations of BIA and the cell structure of the DOCIPs; Section 6.4 discusses and compares the interconnection networks, including a cellular array (DOCIP-array interconnection network), conventional hypercube, and cellular hypercube (DOCIP-hypercube interconnection network); Section 6.5 gives their conceptual optical implementation techniques.

Chapter 7 describes the control and the programming of the DOCIPs: Section 7.1 determines the DOCIPs as "level 1 state machines"; Section 7.2 discusses the control structure of the DOCIPs; Section 7.3 gives the instruction set of the DOCIPs, which consists of only one instruction, pipelines the three fundamental operations of BIA; it also shows that the DOCIPs are "1-step instruction machines" and are general purpose in that, given expandable memory, a DOCIP can compute any computable function. Section 7.4 considers the programming of the DOCIPs, and presents decomposition algorithms of dilation and some programming examples.

Chapter 8 uses BIA to develop parallel binary arithmetic algorithms and describes the execution of these algorithms on the DOCIPs: Section 8.1 presents binary row(or column)-coded arithmetic: binary addition, binary subtraction, and binary multiplication (including a matrix-constant multiplication and an element-element multiplication); Section 8.2 presents binary stack-coded arithmetic; Section 8.3 discusses binary symbol-coded arithmetic (symbolic substitution arithmetic). Section 8.4 gives a comparison for the above different number representations. For the reader who is interested only in the image processing algorithms and architectures, Chapter 8 can be bypassed without discontinuity.

Chapter 9 discusses the implementation of a prototype DOCIP system: Section 9.1 describes the implementation of the optical gate array; Section 9.2 describes the fabrication of the multi-exposure multi-facet interconnection hologram for 3-D optical circuits; Section 9.3 presents the experimental DOCIP system and the circuit design to demonstrate the concept of the DOCIP architecture; Section 9.4 gives the experimental results.

Chapter 10 concludes the thesis: Section 10.1 gives a conclusion; Section 10.2 discuses future research on this work.

1.4 Contributions

The contributions of this work are summarized as follows:

- An algebraic foundation called binary image algebra (BIA) for parallel binary image processing [Huang88a] is given, including:
  - An image space and a family of fundamental operations and elementary images for parallel binary image prcessing;
  - Two fundamental principles for generating images and image transformations to demonstrate its completeness; and

  - A structure which suggests a mapping between parallel algorithms and parallel architectures.

- A spatial logic and a parallel language for parallel digital optical computing is described in which:

  - Algorithms for optical symbolic substitution, cellular logic, and binary logic processors are formalized as compact BIA expressions [Huang87d];

  - An inherently parallel (SIMD) language for parallel computing architectures is suggested [Huang87e, Huang88h]; and

  - A standard notation and a unified approach for parallel binary optical computing and image processing is provided [Huang88a, Huang88b, Huang87d, Huang87e].

- Highly parallel algorithms for image processing, numerical computation and symbolic substitution are given, including:

  - Fast low level vision algorithms [Huang88a], and pattern recognition algorithms to reduce the complexity for a certain class of image processing and machine vision problems [Huang88c];

  - A compact representation and a comparison of different kinds of parallel binary arithmetic [Huang88b];

  - Symbolic substitution rules for image processing and numerical computation applications [Huang88c, Huang88e]; and

  - A technique for the comparison of algorithms on various architectures [Huang88e].

- Highly parallel architectures for image processing and optical computing are described; specifically,
  - A two-dimensional cellular hupercube architecture for image processing that combines the features of a cellular array and conventional hypercube while preserving a relatively low interconnection hologram complexity [Huang87b, Huang88d, Huang88g]; and
  - Digital optical cellular image processor (DOCIP) architectures for parallel image processing and parallel numerical array computation that make uses of the capabilites of optics and implement BIA effectively and naturally [Huang87c, Huang88a, Huang88g].
- Optical holographic interconnections for 3-D optical circuits are experimentally demonstrated, including:
  - An optical multiplexing technique that improves the flexibility and programability of optical holographic interconnections and reduces the hardware complexity of optical processors [Huang87a]; and
  - A computer-controlled optical system for automatically fabricating multi-exposure multi-facet interconnection holograms for 3-D optical circuits [Huang88f, Huang88i].
- An experimental demonstration of a prototype DOCIP system is performed [Huang88d, Huang88i], which:
  - Experimentally demonstrates a 54-gate optical processor, an instruction decoder and electronic input/output interfaces; and

- Shows the functioning of the most sophisticated and complicated experimental digital optical computing system to date (to the best of our knowledge).

Chapter 2

Previous Work — A Review

Through parallel studies of architectures, algorithms, mathematical structures, and optics we have that: 1) an image algebra extending from mathematical morphology [Serra82, Lougheed80, Matheron75, Haralick87] can lead to a formal parallel language approach to the design of image processing algorithms; 2) cellular automata are appropriate models for parallel image processing machines [Preston84, Burks70]; 3) an algebraic structure serves as a framework for both algorithms and architectures of parallel image processing; and 4) the parallel processing and global interconnection advantages of optical computing may be useful in efficiently implementing image algebra with cellular logic architectures. This chapter will first discuss the previous work on image algebra, cellular logic architectures and then digital optical cellular logic processors.

2.1 Previous Work on Image Algebra

The idea of binary image algebra (BIA) comes from several sources — mathematical morphology, image processing, parallel processing, modern algebra, and fundamental concepts of modern physics. This section will concentrate only on reviewing previous work relating to image algebra and its relationship to BIA. During the past few years, numerous papers have used an algebraic approach to aid in image processing [Serra82, Lougheed80, Ritter87, Giardina84, Agui82]. Among of them, morphological image algebra has the closest relation to BIA.

Mathematical morphology was born in 1964 when G. Matheron was asked to investigate the relationships between the geometry of porous media and their permeabilities, and at the same time J. Serra was asked to quantify the petrography of iron ores for predicting their milling properties at the Paris School of Mines in France [Serra82, Matheron75]. For a more historically accurate viewpoint, it originates from research in integral geometry by H. Minkowski who formulated the system of integral equations which permit the extraction of global measurements from a set of projections [Minkowski03]. The structure of mathematical morphology is derived from the four principles of quantification which must be satisfied by every morphological transformation and measure in geology, and then becomes a physical theory (not just a mathematical one). In this approach, objects are modeled as subsets of the space of their definition. The purpose of mathematical morphology is to determine the structure of the objects by transforming the sets that model them. Structuring elements, predefined shapes, are employed to probe or to test out the spatial nature of the object undergoing analysis.

S. R. Sternberg et al. used morphological methods for a variety of industrial machine vision applications [Lougheed80, Sternberg82, Sternberg85]. Sternberg's image algebra is primarily based on Matheron and Serra's mathematical morphology. His major contribution is to introduce the umbra transform for gray-level images and to develop the cytocomputer for rapid morphological image processing.

This morphological image processing approach has been applied to a wide variety of applications including cytology, genetics, petrology, microscopy, and industrial machine vision. Many papers describe either specific theoretical aspects of mathematical morphology or application-specific morphological algorithms [Klein72, Mandeville83, CAPAIDM85a, CAPAIDM85b, Sternberg86]. The applications of mathematical morphology have been fruitful. In this thesis we adapt it to provide the following features:

1. A simplified mathematical structure. Mathematical morphology comprises two branches, integral geometry and geometrical probability, plus a few collateral ancestors (harmonic analysis, stochastic processes, algebraic topology) [Serra82]. The mathematical details and formal proofs in morphology are often intricate and involve advanced set theoretic and topological concepts which are not always necessary for engineering applications.

2. A complete algebraic theory. Mathematical morphology defines some algebraic operators and utilizes some algebra. With our adaptation, we would like to answer the following questions:

   - What is the algebraic definition of this mathematical morphology?
   - How powerful is this mathematical morphology?
   - What is the definition of a transformation? Morphological transformations are constrained by four principles [Serra82], here we introduce a complete definition of image transformations.

3. Clarification of its relationship to other areas. We define its relationship to linear system theory, image processing, and common computing techniques including boolean logic, cellular logic, and algebraic structures.

There are other image algebras, each with its own characteristics [Ritter87, Giardina84]. Because of our intended application to a highly parallel computing machine with simple processing elements and a reduced instruction set, we utilize a BIA with only three fundamental operations that can implement any binary image transformation. For example, the counting function, which gives the number of pixels having a certain level, is considered a mapping from a picture type of operand to a number type of operand [Ritter87, Giardina84]; in BIA numbers are also represented as images [Huang88b]. BIA suggests several simple but fast parallel image algorithms and a parallel image processing architecture with a very low cell complexity.

2.2 Previous Work on Cellular Logic Architectures

To match BIA parallel algorithms by cellular logic architectures in a transparent way, we characterize a cellular automaton by an algebraic structure as BIA does. The cellular logic computer was first inspired by the writings of von Neumann [Neumann51, Neumann66] on cellular automata. The first highly parallel cellular image processor was suggested by Unger [Unger58, Unger59]. Unger proposed and, later, simulated a two-dimensional array of modules (or processing elements or cells) as a natural spatial computer architecture for image processing and recognition. In this approach, each computational cell is responsible for one pixel (or one element of an image) with its neighboring pixels.

A cellular logic (or neighboring logic) operation is then referred to as a transform of an array of data $X(i,j)$ into a new array of data $X'(i,j)$ where each element in the new array has a value determined only by the corresponding element in the original array along with the values of its neighbors. Figure 2.1 describes a sequential process of cellular logic operations.

A cellular logic processor specialized for applications in image processing is then called a cellular image processor. Some review of cellular image processors can be found in Ref. [Preston79, Rosenfeld83, Preston83, Preston84]. Over the last two decades, many cellular computers have been constructed for implementing cellular logic operations, and some idea extending the original nearest neighborhood connected cellular logic computer for improving speed and flexibility are proposed [Rosenfeld83]. Most existing or proposed architectures can be classified as:

1. the cellular string (Fig. 2.2),
2. the cellular array (Fig. 2.3), and
3. the cellular hypercube (Fig. 2.4) and the cellular pyramid (Fig. 2.5).

These three types of architectures share a common feature in the simplicity and regularity of interconnecting simple processing elements, and represent interconnection geometries from 1-D to 2-D then 3-D. The 3-D case leads to further improvements in speed and flexibility, but can be very difficult to implement on a planar VLSI chip [Rosenfeld83, Sawchuk85, Jenkins85]. However,

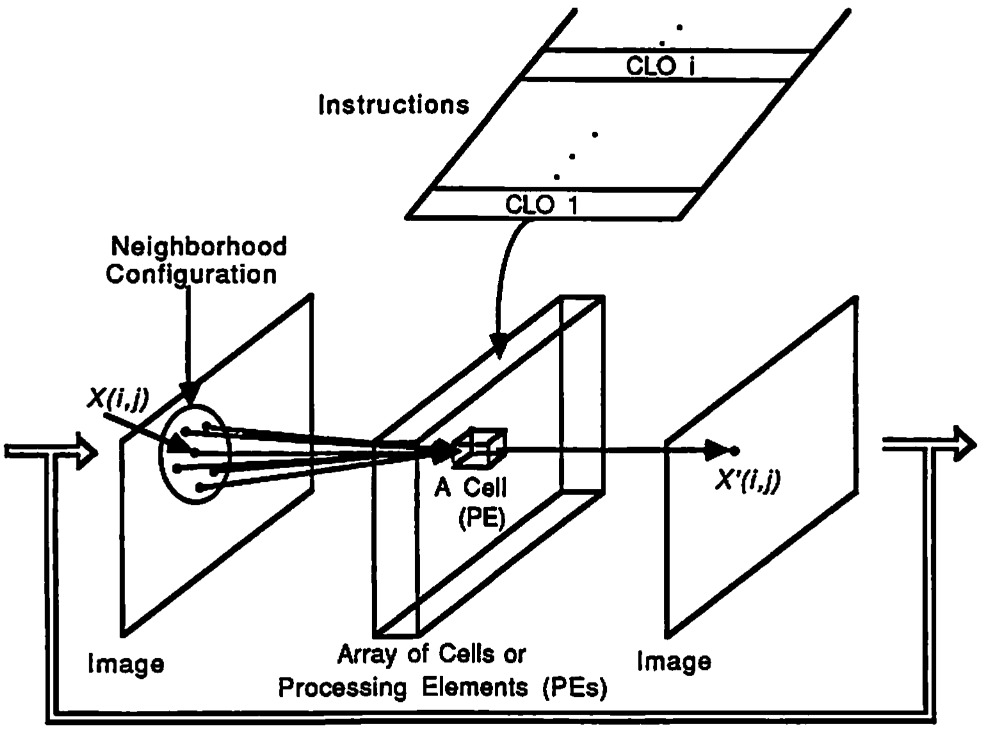

CLO i
Instructions
CLO 1
Neighborhood Configuration
X(i,j)
A Cell (PE)
X'(i,j)
Image
Array of Cells or Processing Elements (PEs)
Image

Figure 2.1: A sequential process of cellular logic operations (CLOs). The value $X'(i,j)$ is determined by the corresponding $X(i,j)$ in the original image along with the values of its neighbors.

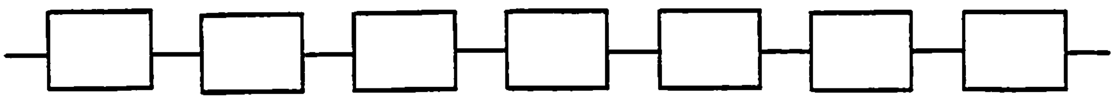


Figure 2.2: A cellular string. It requires only a 1-D interconnection geometry. Each cell only connects with its two nearest cells.

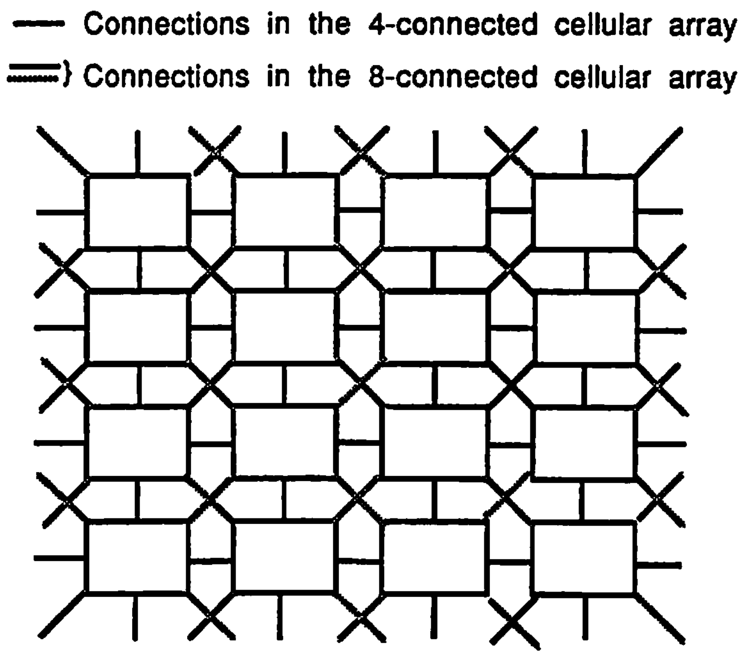

Connections in the 4-connected cellular array
Connections in the 8-connected cellular array

Figure 2.3: A cellular array. It requires a 2-D interconnection geometry. Each cell connects with its 4 or 8 nearest cells.

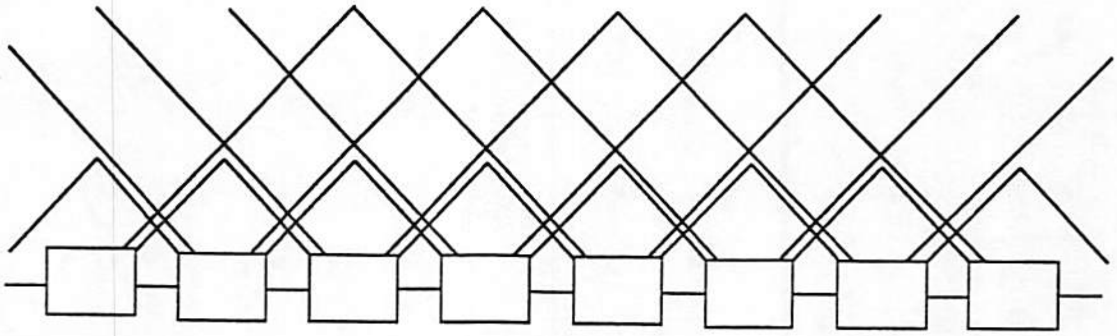


Figure 2.4: A one-dimensional cellular hypercube [Rosenfeld83]. It requires a 3-D interconnection geometry. Each cell connects with cells at distances $1, 2, 4, 8, ..., 2^k$ from it. Here, only the connections with distances 1, 2, and 4 are shown.

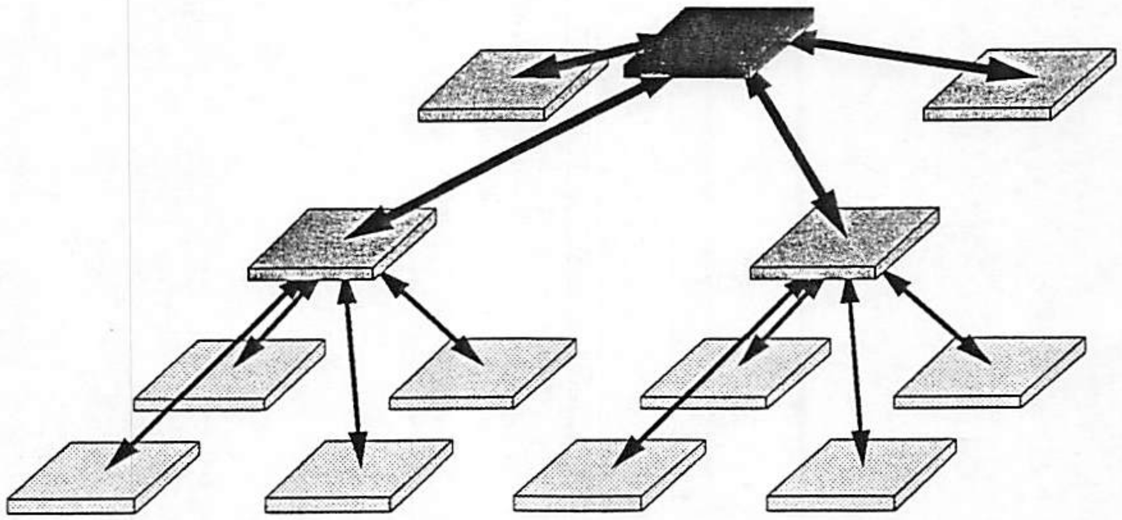


Figure 2.5: A two-dimensional cellular pyramid. It consists of stages of arrays with extra connection between two adjacent stages and is most efficiently implemented with a 3-D interconnection geometry.

in principle, the 3-D interconnection geometry is realizable by a digital optical system without further difficulty, because the general architectural structure of a digital computer as shown in Fig. 2.6 is inherently 3-dimensional [Jenkins84a, Jenkins84b, Sawchuk84, Chavel83]. Digital optical architectures will be discussed in the following section.

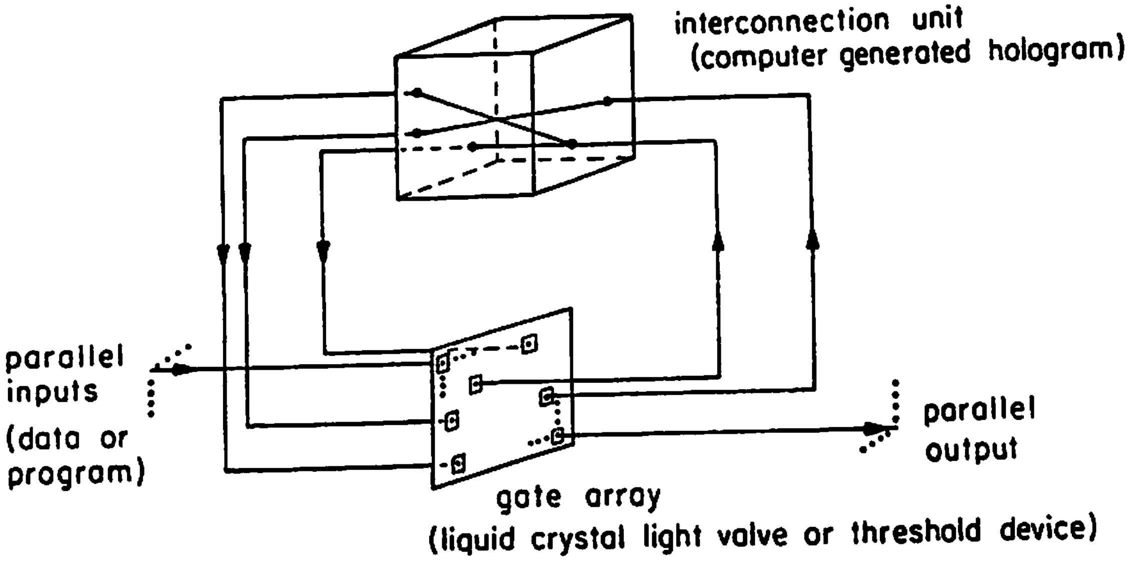

interconnection unit
(computer generated hologram)
parallel inputs
(data or program)
parallel output
gate array
(liquid crystal light valve or threshold device)

Figure 2.6: A 3-D free-interconnection digital optical sequential logic system [Sawchuk84, Jenkins84b].

2.3 Previous Work on Digital Optical Cellular Logic Processors

Recently, several techniques for constructing optical cellular logic processors for image processing have been proposed [Jenkins85, Sawchuk85, Yatagai86, Tanida85, Caulfield86]. However, most of them are only conceptual considerations and do not have any detailed design. The work by Sawchuk and Jenkins, et al, [Sawchuk84, Jenkins84a, Jenkins84b, Jenkins85, Sawchuk85, Chavel83] leads to a very promising direction of digital optical computing and offers possible implementation techniques for digital optical cellular logic processors. They have described and implemented an optical binary sequential logic system that offers a free-interconnection capability very different from that of electronic VLSI.

The system (shown schematically in Fig. 2.6) consists of a spatially parallel array of optically implemented independent binary gates, and an interconnection unit that optically directs some of signals on the output side of the gate array back to the input side of the gate array. Its inherent 3-D structure provides for a high degree of interconnection flexibility. The holographic interconnection system connects the output of each gate to inputs of other gates, effectively wiring up a circuit. For ease of manufacture, the holograms can be generated by (electronic) computer and written out using a computer plotting device. The concept of this system with a 16-gate circuit has been experimentally demonstrated [Jenkins84a]. The limitations of this system are primarily due to the space-bandwidth product (resolution) of optical gate array (spatial light modulator), interconnection unit (computer-generated hologram) and optics. The following discussion of the gate array and interconnection system is primarily a digest from [Jenkins85, Sawchuk85] (with a minor modification).

2-D arrays of optical gates demonstrated to date have one drawback or another that preclude their use in a practical, competitive optical logic system. While current devices can implement 10^5 to 10^6 gates in one array, in most cases the major drawback is the extremely slow speed of the devices. (Typical response times are > 1 ms.) Recent progress in the area of optical bistability, however, provides hope for fast optical logic systems. To date their demonstrations have been primarily on individual (single gate) devices, but in principle they can be used for 2-D arrays as well. Gate switching times on the order of ns-ps have been demonstrated [Jewell84a, Jewell84b, Jewell85, Jewell86, Jewell87, Miller85, Miller86, Smith87, Sharfin86a, Sharfin86b, West87, Lee86a, Lee86b, Wheatley87, Gibbs87], and there is potential for even much faster gates [Fork82, Smith81] (although other considerations such as power may limit the usable response time in a system to $\sim$ 100 ps). Many of these devices are all optical (intrinsic) in that the signal is not converted to electrons and then back to photons again in order to obtain the nonlinearity. This is one of the reasons for their speed advantage. Table 2.1 summarizes some recent results of optical gates. For a review of optical bistability, the reader is referred to [Gibbs85, Gibbs80, Miller82].

Three different optical interconnection systems for interconnecting the gates have been previously described in [Jenkins84b, Chavel83]. All of them use holograms in conjunction with free-space propagation. Their characteristics differ and this manifests itself in the kinds of circuits and processors that can be implemented most efficiently with each system. The gate requirement, hologram space-bandwidth product, and possible optical interconnection systems for cellular logic have been discussed in [Jenkins85, Sawchuk85].

We predict that the number of the elements of the optical gate array and the interconnection subhologram array in a sequental optical logic system (Fig. 2.6) can be $\sim 10^6/cm^2$, each loop feedback time can be $\sim$ 100 ps (the feedack path $\sim$ 3 cm), we can achieve the performance of 10^{16} operations per second. Current optical technology in conjunction with anticipated progress in research may make the construction of such a processor feasible. It is also very possible that the communication and interconnection capabilities of optics for optical cellular logic

Device	Major Developer	Material	Swiching Time	Swiching Energy	Fabricated Array	Gate Spacing	Gate Diameter
Optical Logic Etalon	J.K. Jewell et al. (AT&T)	GaAs/AlAs	On: 1ps Rec: 40ps *150ps	1.5pJ	$10^7/cm^2$ (uniformity problem)	3 µm	1.5 µm
Self Electro-optic Effect Device (SEED)	D.A.B. Miller et al. (AT&T)	GaAs/AlGaAs MQW	On: 30ns Rec: 30ns *2µs	100pJ ($4fJ/\mu m^2$) (optical)	2x2 (6x6?)	400µm	120x 50µm
Nonlinear Interference Filters (NLIF)	S.D. Smith et al. (Edinburgh)	ZnSe	On: 100ps Rec: 10µs	10µJ	$10^4/cm^2$	30µm	
Laser Amplifier	W.F. Sharfin M. Dagenais (GTE)	InGaAs/InP	0.5ns	0.5fJ (opt.) 20pJ (tot.)			
Quantum Well Envelope State Transition Device (QWEST)	L.C. West (AT&T)	GaAs/AlGaAs MQW	2ps ?	30fJ?			

On: switch-on time
Rec: recovery time
* indicates the recovery time for the whole array of gates
Note: SEED time and energy correspond to a single pixel device (except where otherwise noted).

Table 2.1: A summary of recent results of optical gates.

processors could provide for substantially reduced computation time for image processing tasks.

Obviously, the work of Jenkins and Sawchuk, et al, in optical computing and optical cellular logic processors guides the progress to the detailed design of a digital optical cellular logic processor. The goal of this thesis is to have a complete optical circuit design of a cellular machine, the DOCIP, and to demonstrate its concept experimentally.

Chapter 3

Binary Image Algebra (BIA): Fundamentals

Binary image algebra (BIA) is intended to be a unifying theory based on a set of three specific fundamental operations. Another basic purpose of BIA is for the development of a programming language for a specific parallel architecture, namely a digital optical cellular image processor (DOCIP) (Chapter 6 and Chapter 7). These fundamental operations are the key operations in the instruction set of the DOCIP machine. The BIA provides a decomposition of general operations, including low-level image processing operations, into the three fundamental operations of the instruction set. This decomposition is inherently parallel and provides a direct mapping to the machine architecture.

In this chapter, we first discuss the philosophy of BIA, give the definition of BIA, and provide the fundamental principles of BIA for proving its completeness.

3.1 Underlying Philosophy

The overall philosophy of BIA is:

- *An image, but not a pixel, is an object.* For parallel languages and machines for image processing, images can be considered as primitive variables for simplifying the design.
- *Complex image processing operations can be reduced to simple instructions.* Although image processing operations appear complex, the fundamental interactions and the elementary components in a system are very simple.

Thus, BIA begins by:

1. Defining the universal image as the working space for images and their image transformations.
2. Defining elementary images which can be combined to generate any image.
3. Defining fundamental operations which can be cascaded to form complex operations.
4. Defining image processing/analysis algorithm design as the choice of "good" (or "appropriate") reference images and transformations.

A reference image can be any image and is a generalization of structuring elements in mathematical morphology [Serra82]. Reference images contain some predefined image property (or information); image transformations (or operations) are used for measuring the image property from an input image. Image description, image information extraction or image property measurement is done by using reference images to model or transform the original image to a final state which reveals the desired information or is used to detect the desired properties easily.

Here we give the algebraic structure of BIA first, and then provide definitions and present two fundamental principles which allow us to generate any reference image and implement any image transformation. Ideally, BIA may be further generalized to GIA (General Image Algebra) which deals with grey-level and complex-valued images.

3.2 Definitions

Definition of Binary Image Algebra (BIA)

Binary image algebra is an algebra with an image space S, which is the power set of a predefined universal image $P(W)$, and a family F of operations including 3 fundamental operations $(\oplus, \cup, ^{-})$, which are non 0-ary operations, and 5 elementary images $(I, A, A^{-1}, B, B^{-1})$, which are 0-ary operations. Symbolically,

$$BIA = (P(W); \oplus, \cup, ^{-}, I, A, A^{-1}, B, B^{-1}) \qquad (3.1)$$

i.e. $S = P(W)$ and $F = (\oplus, \cup, ^{-}, I, A, A^{-1}, B, B^{-1})$. The image space S and the family F of operations will be derived in the following.

Basic Definitions

In general, a binary digital image is defined as a function f that maps each spatially sampled grid point (x, y) of the picture on an orthogonal coordinate system onto the set composed of two elements: 1 (i.e. white, foreground point or image point) and 0 (i.e black or background point). However, it will be more convenient for our algebra, if we use a set of the coordinates of image points ('1's) to specify an image. In this paper, an image is treated as the set of coordinates of image points (i.e. foreground points or pixels that have value 1). We begin the description of BIA by defining our artificial universe:

Definition 3.1 The Universal Image.

The universal image is the set $W = \{(x,y) \mid x \in Z_n, y \in Z_n\}$, where $Z_n = \{0, \pm1, \pm2, ..., \pm n\}$ and n is a positive integer (Fig. 3.1).

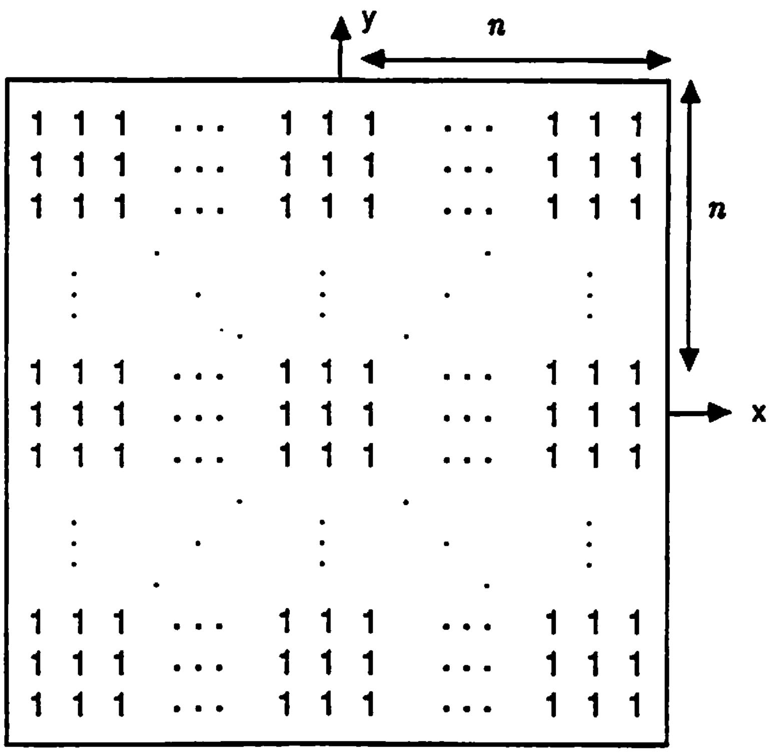


Figure 3.1: The universal image W. It has $(2n+1) \times (2n+1)$ image points and n is a positive integer.

Remark: "$\in$" means "belongs to". Notice that given n, the universal image defines the domain of our images. In fact, for an image with size larger than $(2n+1) \times (2n+1)$ (the size of the universal image), we need to increase the size of the universal image or decompose the tested image into subimages whose sizes are smaller than the size of the universal image. For the reason of simple practice, we only consider the square tessellation of images. To deal with non-square (e.g. hexagonal) tessellations, we can simply replace the universal image to be the set of grid points corresponding to the new tessellation pattern.

Definition 3.2 Image Space.

The image space is the power set (the set of all subsets) of the universal image, i.e. $S = P(W)$.

Definition 3.3 Image.

A set X is an image if and only if X is an element of the image space S, i.e. X is a subimage (subset) of the universal image W. Symbolically,

$$X \text{ is an image} \leftrightarrow X \in S \leftrightarrow X \subset W.$$

Remark: "$\subset$" means "is included in". There exist $2^{(2n+1)\times(2n+1)}$ different images. Three terms related to images are defined:

1. Size (or area) of an image X, denoted as $\#(X)$, is the cardinality (i.e. the number of elements) of the image X.

2. Foreground of an image X, simply denoted as X, is referred to those pixels with value 1.

3. Background of an image X, denoted as the complement $\overline{X}$ (Definition 3.6), is referred to those pixels with value 0.

Once we know the foreground of an image, the background of this image is well defined (since the universal image is given first). Thus, the foreground is sufficient to specify an image.

Definition 3.4 Image Point (Foreground Point).

A point (x, y) is an image point of an image X if and only if (x, y) is an element of the set X.

Remark: The largest image is the universal image W and consists of $(2n + 1) \times (2n + 1)$ image points, i.e. $\#(W) = (2n + 1) \times (2n + 1)$; the smallest image is the null image ϕ (defined as the complement $\phi = \overline{W}$) and has no image points, i.e $\#(\phi) = 0$.

Definition 3.5 Image Transformation.

A transformation T is an image transformation if and only if T is a function mapping from the image space S to the image space S.

Remark: There exist $(2^{(2n+1)\times(2n+1)})^{(2^{(2n+1)\times(2n+1)})}$ image transformations.

Definition 3.6 Three Fundamental Operations.

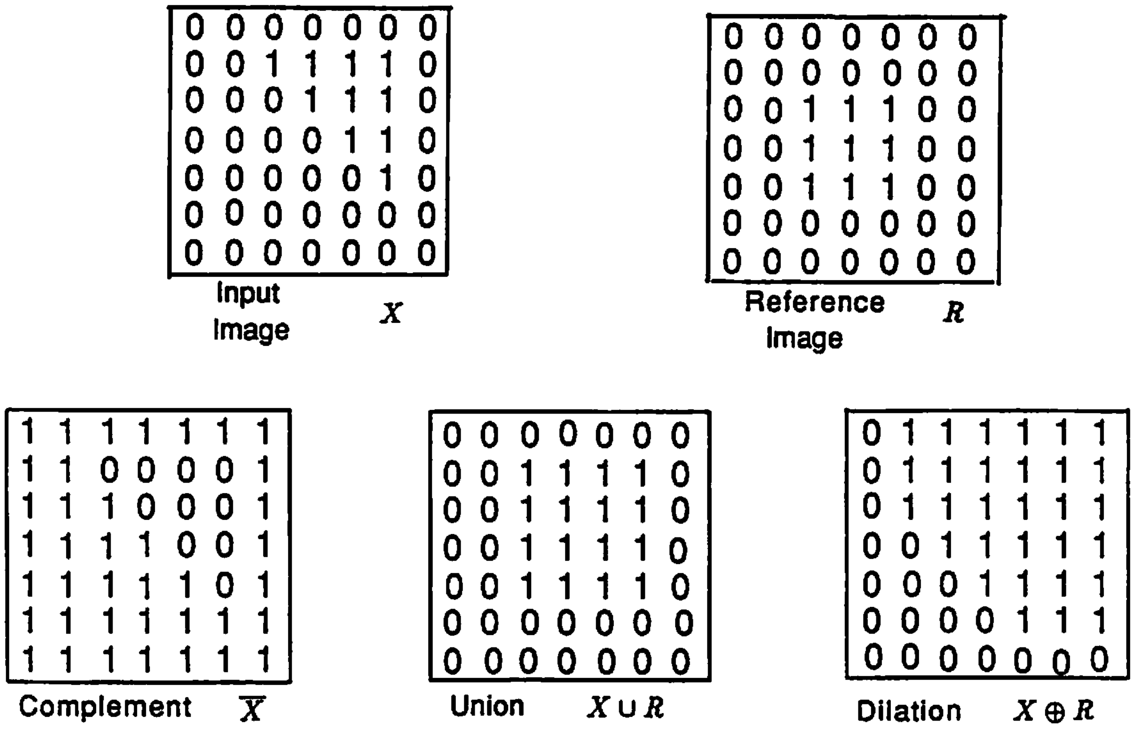
There are three fundamental operations (Fig. 3.2):

1. Complement of an image X:

$$\overline{X} = \{(x, y) \mid (x, y) \in W \wedge (x, y) \notin X\} \tag{3.2}$$

2. Union of two images X and R:

$$X \cup R = \{(x, y) \mid (x, y) \in X \vee (x, y) \in R\} \tag{3.3}$$



$$X = \begin{array}{|ccccccc|}\hline 0&0&0&0&0&0&0\\ 0&0&1&1&1&1&0\\ 0&0&0&1&1&1&0\\ 0&0&0&0&1&1&0\\ 0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ \hline\end{array} \qquad R = \begin{array}{|ccccccc|}\hline 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ 0&0&1&1&1&0&0\\ 0&0&1&1&1&0&0\\ 0&0&1&1&1&0&0\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ \hline\end{array}$$

Input Image X Reference Image R

$$\overline{X} = \begin{array}{|ccccccc|}\hline 1&1&1&1&1&1&1\\ 1&1&0&0&0&0&1\\ 1&1&1&0&0&0&1\\ 1&1&1&1&0&0&1\\ 1&1&1&1&1&0&1\\ 1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1\\ \hline\end{array} \qquad X \cup R = \begin{array}{|ccccccc|}\hline 0&0&0&0&0&0&0\\ 0&0&1&1&1&1&0\\ 0&0&1&1&1&1&0\\ 0&0&1&1&1&1&0\\ 0&0&1&1&1&1&0\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ \hline\end{array} \qquad X \oplus R = \begin{array}{|ccccccc|}\hline 0&1&1&1&1&1&1\\ 0&1&1&1&1&1&1\\ 0&1&1&1&1&1&1\\ 0&0&1&1&1&1&1\\ 0&0&0&1&1&1&1\\ 0&0&0&0&1&1&1\\ 0&0&0&0&0&0&0\\ \hline\end{array}$$

Complement $\overline{X}$ Union $X \cup R$ Dilation $X \oplus R$

Figure 3.2: An example of fundamental operations: complement, union and dilation.

3. Dilation of two images X and R:

$$X \oplus R = \begin{cases} \{(x_1 + x_2, y_1 + y_2) \in W \mid (x_1, y_1) \in X, (x_2, y_2) \in R\} & (X \neq \phi) \wedge (R \neq \phi) \\ \phi & \mathit{otherwise} \end{cases} \tag{3.4}$$

Remark: "$\wedge$" means "and", and "$\vee$" means "or". Note that X usually represents an input or data image and R is a reference image. The consideration of null image in the dilation operation is missing in mathematical morphology (where the dilation is defined the union of all translations of X by all image points in R); with this generalization we have a complete theory which is not found in other image algebras because of a lack of demonstration of their capabilities for implementing any image transformation. We can also define other image operations as fundamental operations instead of these three operations. The reason for choosing these three operations is because of their simplicity, and resulting simple software design and hardware implementation. As shown later, these three operations may be implemented by a 2-D optical gate array with 3-D interconnections.

Definition 3.7 Elementary Images.

These elementary images are constant images, i.e. 0-ary operations. Each elementary image has only one image point. There are 5 elementary images:

1. $I = \{(0,0)\}$ — consisting of an image point at the origin

2. $A = \{(1,0)\}$ — consisting of an image point right of the origin

3. $A^{-1} = \{(-1,0)\}$ — consisting of an image point left of the origin

4. $B = \{(0,1)\}$ — consisting of an image point above the origin

5. $B^{-1} = \{(0,-1)\}$ consisting of an image point below the origin

Remark: In fact, these 5 elementary images could be reduced to 4 elementary images, because $I = A^0 \equiv A \oplus A^{-1} = B^0 \equiv B \oplus B^{-1}$.

Definition 3.8 Reflected Reference Image.

Given a reference image R which is a predefined image for containing some desired image property or image information, its reflected image is defined as

$$\check{R} = \{(-x,-y) \mid (x,y) \in R\}. \tag{3.5}$$

Remark: In many useful cases the reference image R is symmetric, then $\check{R} = R$.

3.3 Two Fundamental Principles

Two fundamental principles basically define the binary image algebra (BIA). Before stating these two principles, we give some preliminary results.

Lemma 3.1.

$$\overline{(X \oplus \overline{R}) \cup (\overline{X} \oplus R) \cup \overline{I}} = \begin{cases} I & if X = \check{R} \\ \phi & otherwise \end{cases} \quad \forall X, R \in P(W) \tag{3.6}$$

where $I = \{(0,0)\}$ is an elementary image, $\check{R}$ is the reflected reference image of R, and "$\forall$" means "for all".

Proof: Appendix A.1.

Remark: This lemma says that if the image X matches the image $\check{R}$, then the origin (central pixel) of the above output image has value '1', otherwise always '0'.

Theorem 3.1.

Any image transformation $T : P(W) \rightarrow P(W)$ can be expressed as

$$T(X) = \bigcup_{i=1}^{k} \{ \overline{(\overline{X} \oplus R_i) \cup (X \oplus \overline{R_i}) \cup \overline{I}} \oplus Q_i \} \tag{3.7}$$

where $k \leq \#(P(W))$, R_i and Q_i are the reference images used to form any desired image transformation, and

$$\bigcup_{i=1}^{k} R_i \equiv R_1 \cup R_2 \cup \ldots \cup R_k.$$

Proof: Appendix A.2.

Theorem 3.2.

Any image can be represented as

$$X = \bigcup_{(i,j) \in X} A^i B^j \tag{3.8}$$

where $A^i B^j \equiv A^i \oplus B^j$,

$A^i \equiv \underbrace{A \oplus A \oplus \ldots \oplus A}_{i} = \{(i, 0)\}$ if $i > 0$,

$A^i \equiv \underbrace{A^{-1} \oplus A^{-1} \oplus \ldots \oplus A^{-1}}_{-i} = \{(i, 0)\}$ if $i < 0$,

and A, B, A^{-1}, B^{-1} are the elementary images defined in Definition 3.7.

Proof: Appendix A.3.

Principle 1. Fundamental Principle of Image Transformations

Any image transformation T can be implemented by using appropriate reference images R and the three fundamental operations: 1. Complement $\overline{X}$ of an image X, 2. Union $\cup$ of two images, 3. Dilation $\oplus$ of two images.

Proof: It follows from Theorem 3.1.

In order to use principle 1 efficiently in practice, we invoke principle 2 for the generation of reference images.

Principle 2. Fundamental Principle of Reference Images

Any reference image R can be generated from elementary images $(I, A, A^{-1}, B, B^{-1})$ by using the three fundamental operations.

Proof: It follows from Theorem 3.2.

Therefore, by the above principles, we can represent BIA as:

$$\text{BIA} = (P(W); \oplus, \cup, {}^{-}, I, A, A^{-1}, B, B^{-1}).$$

Chapter 4

BIA: Development

BIA can have many applications in character recognition, industrial inspection, medical image processing, and scientific computation. In this chapter we first review the basic properties of images and image transformations, define 11 standard operations, and give some special cases of dilation [Serra82, Lougheed80, Matheron75, Haralick87, Rosenfeld70, Rosenfeld82, Duda73, Levialdi72]. Then we summarize four theorems and some examples for binary image processing and suggest a pattern recognition algorithm from the observation of the growth of patterns.

Sections 4.1-4.3 primarily give a survey of binary image processing algorithms with implementation using BIA fundamental operations. These fundamental operations are so chosen because they form an efficient basis for the instruction set of an optically-based cellular image processor. This survey serves as a description of a parallel language for controlling the processor and how it is compiled into low level instructions. In Section 4.4, BIA formulates a class of pattern recognition problems. The use of BIA for parallel numerical computation is described in Chapter 8 [Huang88b].

4.1 Basic Properties of Images and Image Transformations

Definition 4.1 Connectivity in Images

1. 4-neighbor and 8-neighbor:

   An image point (x, y) in an image X can have two types of neighors:

   (a) An image point (i, j) is a 4-neighbor of (x, y)
   $\leftrightarrow (i, j) \in \{(x \pm 1, y), (x, y \pm 1)\}$.

   Remark: $\{(x, y), (x \pm 1, y), (x, y \pm 1)\}$ is called the 4-neighborhood of (x, y) and $N_4 \equiv \{(0,0), (0, \pm 1), (\pm 1, 0)\} = I \cup A \cup A^{-1} \cup B \cup B^{-1}$ (Fig. 4.1(a)).

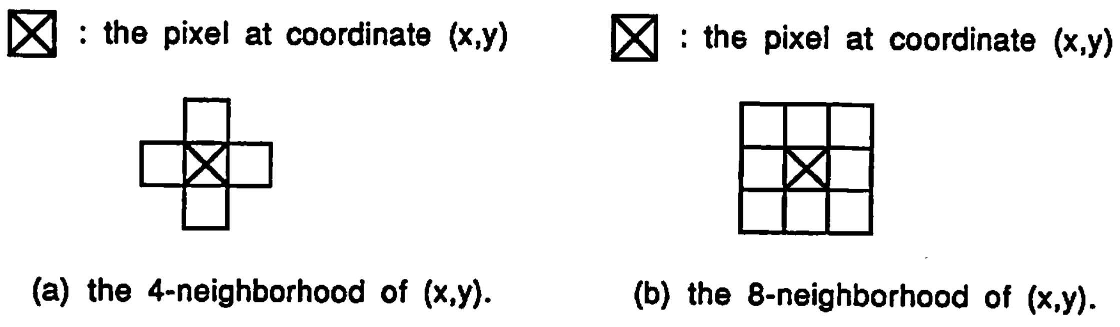

: the pixel at coordinate (x,y)
: the pixel at coordinate (x,y)
(a) the 4-neighborhood of (x,y).
(b) the 8-neighborhood of (x,y).

Figure 4.1: The 4-neighborhood and 8-neighborhood of an image point (x, y).

(b) An image point (i,j) is a 8-neighbor of (x,y)

$\leftrightarrow (i,j) \in \{(x \pm 1, y), (x, y \pm 1), (x \pm 1, y \pm 1)\}$.

Remark: $\{(x,y), (x \pm 1, y), (x, y \pm 1), (x \pm 1, y \pm 1)\}$ is called the 8-neighborhood of (x,y) and $N_8 \equiv \{(0,0), (0,\pm 1), (\pm 1, 0), (\pm 1, \pm 1)\}$ (Fig. 4.1(b)).

2. 4-connected and 8-connected:

(a) Two image points (x,y) and (i,j) of an image X are 4-connected

$\leftrightarrow$ there exists a sequence of image points $(x,y) = (x_0,y_0), (x_1,y_1), \ldots,$ $(x_m,y_m) = (i,j)$, where (x_k,y_k) is a 4-neighbor of (x_{k-1},y_{k-1}) and $(x_k,y_k) \in X$, $1 \leq k \leq m$.

(b) Two image points (x,y) and (i,j) of an image X are 8-connected

$\leftrightarrow$ there exists a sequence of image points $(x,y) = (x_0,y_0), (x_1,y_1), \ldots,$ $(x_m,y_m) = (i,j)$, where (x_k,y_k) is a 8-neighbor of (x_{k-1},y_{k-1}) and $(x_k,y_k) \in X$, $1 \leq k \leq m$.

Remark 1: "4-connected in X" and "8-connected in X" are equivalence relations (reflexive, symmetric and transitive).

Remark 2: For any image point (x,y) in a non-null image X, the set of (i,j) such that (x,y) and (i,j) are 4-connected (or 8-connected) is called a 4-connected (or 8-connected) component of X. A 4-connected (or 8-connected) component of X is just an equivalence class in X under the equivalence relation — "4-connected (or 8-connected) in X". Thus, a collection of 4-connected (or 8-connected) components of X forms a partition of X, i.e. the set of all 4-connected (or 8-connected) components $\{X_i\}_{i \in I}$

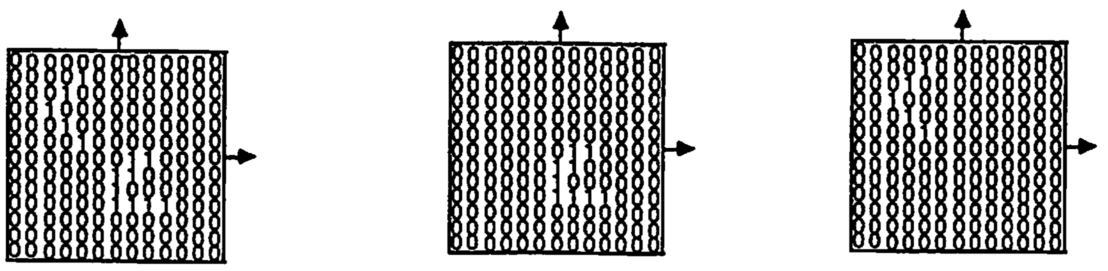
(where I is the index set of connected components) is a family of non-null subimages of X and has the following properties:

(a) $X_i \neq \phi$ for all $i \in I$.

(b) $X_i \cap X_j = \phi$ for all $i \neq j$, $i, j \in I$. ($X_i \cap X_j = \overline{\overline{X_i} \cup \overline{X_j}}$ as defined in Definition 4.3)

(c) $X = \cup_{i \in I} X_i$.

Fig. 4.2(a) shows a 4-connected component in an image X and Fig. 4.2(b) shows an 8-connected component in X.



(a) A 4-connected (8-connected too) component of X.

(b) An 8-connected component of X

Figure 4.2: The 4-connected component and 8-connected component of an image.

Remark 3: If an image X has l 4-connected (or 8-connected) components, there are l distinct equivalence classes in X. Each equivalence class X_i can be represented by an image point in X_i. Thus, we may use l distinct image points which belong to l different 4-connected (or 8-connected) components to represent the classes of the image X.

Remark 4: In dealing with connectedness in both X and $\overline{X}$, to avoid the "connectivity paradox" [Rosenfeld70], it is preferable to use opposite types of connectedness for X and $\overline{X}$, i.e. if we use "4-connected" for X, then we use "8-connected" for $\overline{X}$, and vice versa.

Remark 5: If any image X is surrounded by a border of 0's, the component of $\overline{X}$ consisting of the points connected to (any one of) these 0's is called the *outside* of X (Fig. 4.3(a)). If $\overline{X}$ has any other components, they are called *holes* in X (Fig. 4.3(b)).

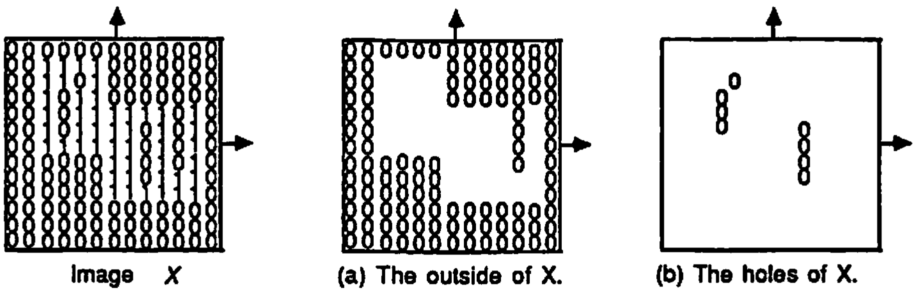


Figure 4.3: The outside and holes of an image.

For more detailed discussion of geometric properties of images, the reader is referred to [Rosenfeld70, Rosenfeld82, Duda73]. For equivalence relations, equivalence classes and partitions, please refer to [Birkhoff70, Birkhoff65, Gilbert76].

Definition 4.2 Basic Properties in Image Transformations

The key properties of image transformations are the following ten basic properties

1. Increasing: An image transformation $T(X)$ is increasing
   $\leftrightarrow (X \subset Y \rightarrow T(X) \subset T(Y))$ for all $X, Y \in P(W)$.

2. Decreasing: An image transformation $T(X)$ is decreasing
   $\leftrightarrow (X \subset Y \rightarrow T(Y) \subset T(X))$ for all $X, Y \in P(W)$.

3. Extensive: An image transformation $T(X)$ is extensive

   $\leftrightarrow X \subset T(X)$ for all $X \in P(W)$.

4. Antiextensive: An image transformation $T(X)$ is antiextensive

   $\leftrightarrow T(X) \subset X$ for all $X \in P(W)$.

5. Idempotent: An image transformation $T(X)$ is idempotent

   $\leftrightarrow T(T(X)) = T(X)$ for all $X \in P(W)$.

6. Shift invariant: An image transformation $T(X)$ is shift invariant

   $\leftrightarrow T(X \oplus P) = T(X) \oplus P$ for all $X, P \in P(W)$ and P is a point image which consists of one and only one image point.

   If an image transformation is not shift invariant, then it is shift variant: $T(X \oplus P) \neq T(X) \oplus P$ (in general).

7. Homotopic: An image transformation $T(X)$ is homotopic

   $\leftrightarrow$ there exists a one-to-one and onto correspondence between the connected components of X and those of $T(X)$, for all $X \in P(W)$. The same is then true for the holes.

8. Commutative: A binary image operation $\cdot$ is commutative

   $\leftrightarrow X \cdot R = R \cdot X$ for all $X, R \in P(W)$.

9. Associative: A binary image operation $\cdot$ is associative

   $\leftrightarrow (X \cdot R) \cdot Q = X \cdot (R \cdot Q)$ for all $X, R, Q \in P(W)$.

10. Distributive: A binary image operation $\cdot$ is distributive over a binary image operation $+$

    $\leftrightarrow X \cdot (R + Q) = (X \cdot R) + (X \cdot Q)$ for all $X, R, Q \in P(W)$.

Definition 4.3 Standard Operations

Most standard operations can be derived from the three fundamental operations; eleven common ones follow:

1. Difference of X by R (Fig. 4.4):

$$X/R = \{(x,y) \in X \mid (x,y) \notin R\} = X \cap \overline{R} = \overline{\overline{X} \cup R} \qquad (4.1)$$

Remark: $\overline{X} = W/X$ where W is the universal image. The difference is

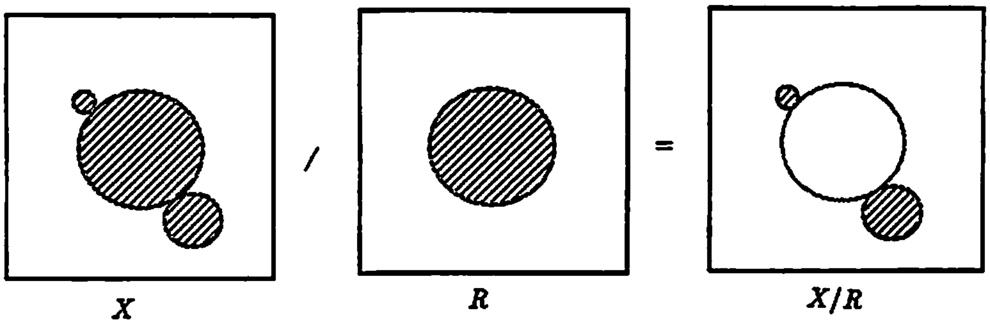


Figure 4.4: Difference.

an obvious approach to detect defects in the foreground of a tested image.

2. Intersection of two images X and R (Fig. 4.5):

$$X \cap R = \{(x,y) \mid (x,y) \in X \wedge (x,y) \in R\} = \overline{\overline{X} \cup \overline{R}} \qquad (4.2)$$

Remark: $X \cup R = \overline{\overline{X} \cap \overline{R}}$. If $X \cap R \neq \phi$, then we say that an image X hits (or is joint with) an image R. If $X \cap R = \phi$, then we say that an image X misses (or is disjoint with) an image R

3. Erosion of an image X by a reference image R or foreground template matching of X by R (Fig. 4.6):

$$X \ominus R = \overline{\overline{X} \oplus \check{R}} \qquad (4.3)$$

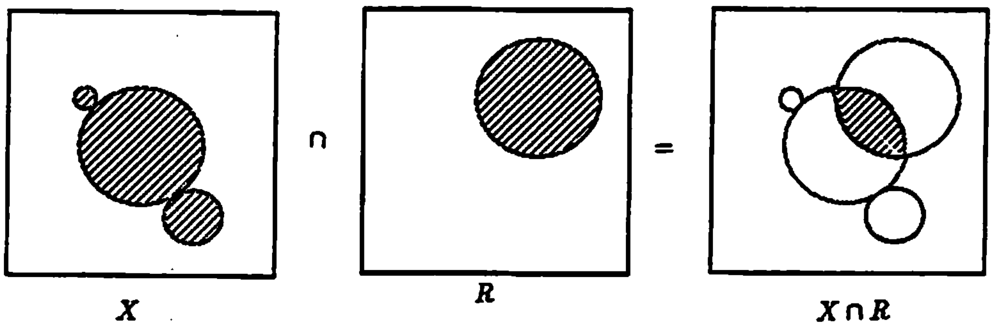


Figure 4.5: Intersection.

Remark: $X \oplus R = \overline{\overline{X} \ominus \check{R}}$, and $R = \check{R}$ when R is symmetic. The erosion

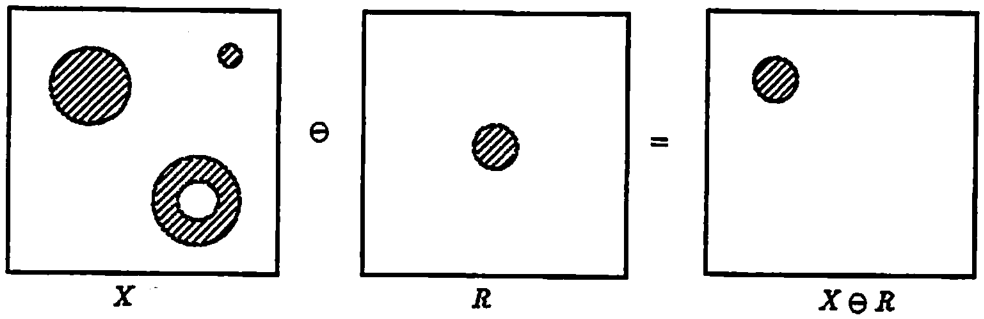


Figure 4.6: Erosion.

of an image X by a reference image R can be thought of as the complement of the dilation of the backgound by the reflection of the reference image R. In general, the erosion of a non-null image X by a non-null reference image R can be used to decrease the size of regions, increase the size of holes, eliminate regions, and break bridges in X; on the contrary, the dilation of a non-null image X by a non-null reference image R can increase the size of regions, decrease or fill in holes and cavities, and bridge gaps in X. Furthermore, the erosion can be interpreted as a foreground template matching where the foreground points of $X \ominus R$ indicates the ocurrences of the foreground template R in X (for this purpose, the size of R usually is much smaller than the size of X).

4. Symmetric Difference of two images (mod 2 image addition or subtraction) (Fig. 4.7):

$$X \triangle R = (X/R) \cup (R/X) = \overline{\overline{X} \cup R} \cup \overline{\overline{R} \cup X} \tag{4.4}$$

Remark: The symmetric difference is a commutative operation, and its

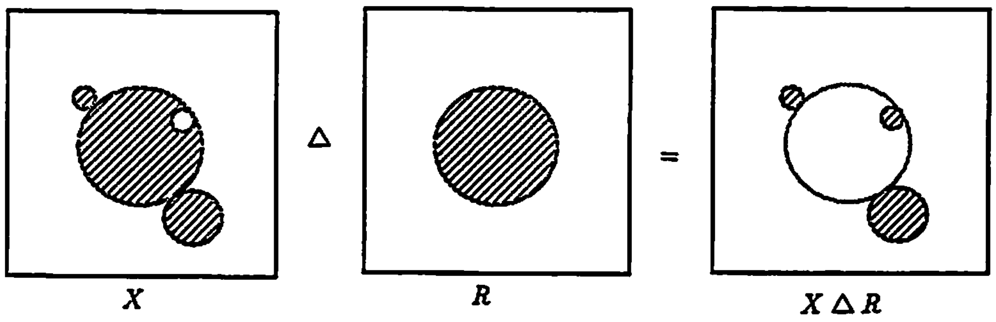


Figure 4.7: Symmetric difference.

inverse operation can be defined as itself. In Chapter 5 we show that this operation is the parallel form of boolean EXCLUSIVE-OR. It is an obvious approach to detect defects (including the foreground or background defects) of a tested image.

5. Opening of an image X by a reference image R (Fig. 4.8):

$$X \circ R = (X \ominus R) \oplus R = \overline{\overline{X} \oplus \check{R}} \oplus R \tag{4.5}$$

Remark: The opening operation is an erosion followed by a dilaton with the same reference image R. In general, the opening $X \circ R$ with a non-null

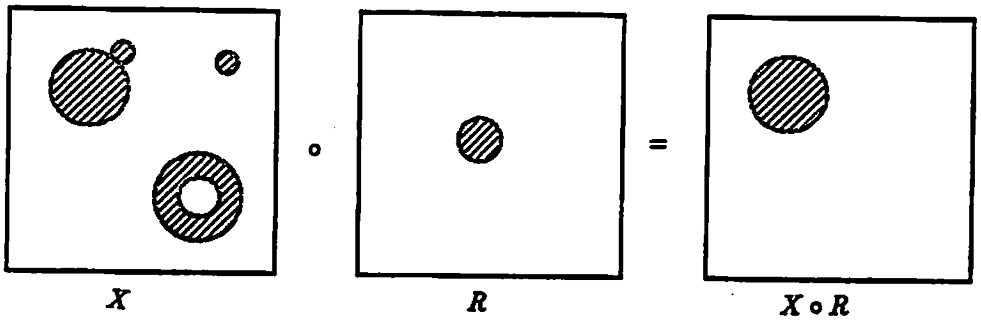


Figure 4.8: Opening.

reference image R reduces the size of regions and eliminates some image points by removing all features in X which can not contain the reference image R.

6. Closing of an image X by a reference image R (Fig. 4.9):

$$X \bullet R = (X \oplus R) \ominus R = \overline{\overline{(X \oplus R)} \oplus \check{R}} \tag{4.6}$$

Remark: The closing operation is a dilation followed by an erosion with

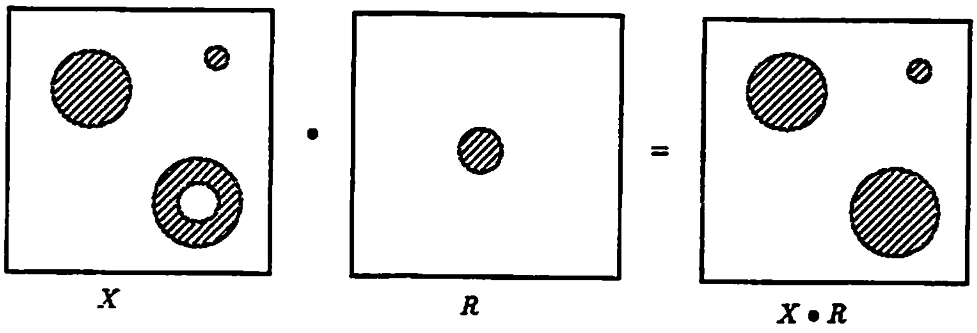


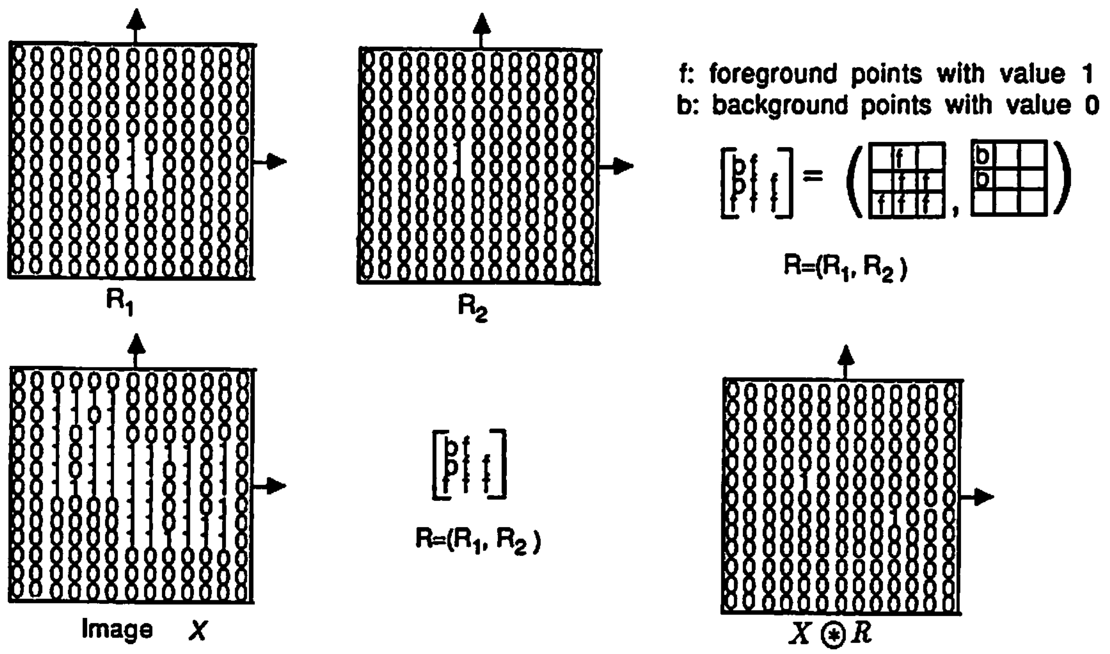
Figure 4.9: Closing.

the same reference image R. In general, the closing $X \bullet R$ with a non-null reference image R increases the size of regions and eliminates some background points by filling in all background areas that can not contain the reference image R, such as holes and concavities in the image X.

7. Hit or miss transform $\circledast$ of an image X by an image pair $R = (R_1, R_2)$ or template matching of X by R (Fig. 4.10):

$$X \circledast R = (X \ominus R_1) \cap (\overline{X} \ominus R_2) = \overline{(\overline{X} \oplus \check{R}_1) \cup (X \oplus \check{R}_2)} \qquad (4.7)$$

Remark: The hit or miss transform of an image X by a reference



f: foreground points with value 1
b: background points with value 0
R=(R1, R2)
R1
R2
Image X
R=(R1, R2)
X ⊛ R

Figure 4.10: Hit or miss transform (template matching).

image pair $R = (R_1, R_2)$ is used to match the shape (or template) defined by the reference image pair R where R_1 defines the foreground of the shape and R_2 defines the background of the shape. The key conditions are that the foreground X must match R_1 (i.e. $X \ominus R_1$), while simutaneously the background $\overline{X}$ matches R_2 (i.e. $\overline{X} \ominus R_2$). In order to better define the hit or miss transform and its relationship with conventional boolean logic operations, we start from a pixel-wise boolean comparison to derive the hit or miss transform in shape recognition (Theorem 4.2). Note the similarity of the symmetric difference and the hit or miss transform.

8. Thinning $\circledcirc$ an image X by an image pair $R = (R_1, R_2)$ (Fig. 4.11):

$$X \circledcirc R = X/(X \circledast R) = \overline{\overline{X} \cup \overline{(\overline{X} \oplus \check{R}_1) \cup (X \oplus \check{R}_2)}} \qquad (4.8)$$

Remark: The thinning operation is antiextensive and decreases the size

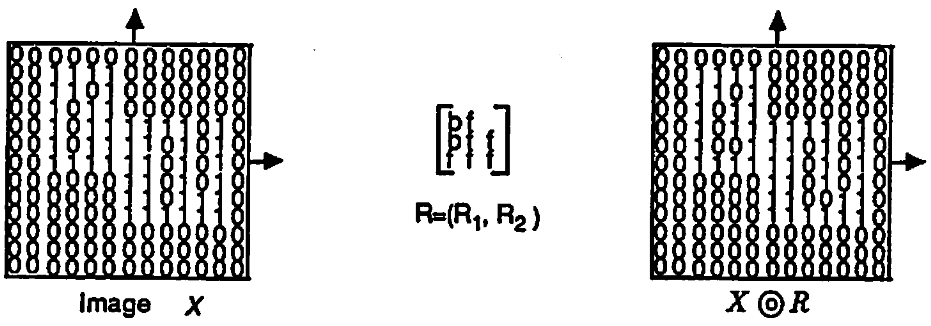


Figure 4.11: Thinning.

by removing the central points of the regions which match the reference image pair $R = (R_1, R_2)$.

9. Thickening $\odot$ an image X by an image pair $R = (R_1, R_2)$ (Fig. 4.12):

$$X \odot R = X \cup (X \circledast R) = X \cup \overline{(\overline{X} \oplus \check{R}_1) \cup (X \oplus \check{R}_2)} \qquad (4.9)$$

Remark: The thickening operation is extensive and increases the size by filling the image points where the regions match the reference image pair $R = (R_1, R_2)$.

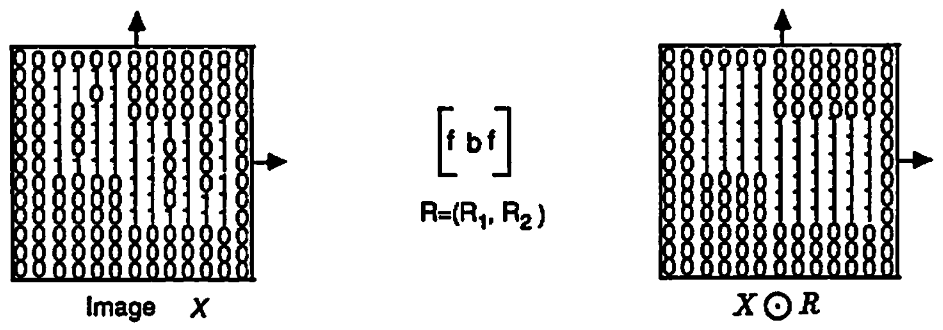


Figure 4.12: Thickening.

10. Sequential operations (e.g. sequential dilation, sequential erosion, sequential thinning etc.):

    If an image operation $\cdot$ is successively performed with each reference image (or image pairs) in a sequence $(R_\theta) \equiv (R_a, R_b, ..., R_z)$, then we define a sequential image operation

    $$X \cdot (R_\theta) = (...((X \cdot R_a) \cdot R_b) ... \cdot R_z). \tag{4.10}$$

    Two examples are:

    (a) Sequential thinning of an image X by a sequence of image pairs $(R_\theta) \equiv (R_a, R_b, ..., R_z)$:

    $$X \circledcirc (R_\theta) = (...((X \circledcirc R_a) \circledcirc R_b) ... \circledcirc R_z). \tag{4.11}$$

    Remark: The sequential thinning is powerful in many applications, such as constructing a digital homotopic skeleton of an image X. Skeletonization of an image is an operation that transforms the image to a simplified image, called skeleton, which emphasizes its connectivity.

However, a homotopic skeleton cannot be obtained by digitizing an analog skeletonization algorithm; instead, a sequential thinning with a sequence of reference image pairs should be used. Several different algorithms employing different reference image pairs (called masks) have been proposed by several authors [Preston84, Levialdi72]. Fig. 4.13 shows an example of the skeletonization by a sequential thinning with a sequence of eight reference image pairs proposed by Levialdi et al [Levialdi72].

(b) Sequential dilation of an image X and a sequence of reference images $(R_\theta) \equiv (R_a, R_b, ..., R_z)$:

$$X \oplus (R_\theta) = (...((X \oplus R_a) \oplus R_b)... \oplus R_z). \tag{4.12}$$

Remark: Since the dilation is commutative and associative, in practice the dilation $X \oplus R$ with a *large* reference image R is usually implemented as a sequential dilation with a sequence of *small* reference images. For example, if $R = E_1 \oplus E_2 \oplus ... \oplus E_k$, then

$$X \oplus R = (...((X \oplus E_1) \oplus E_2) \oplus ... \oplus E_k); \tag{4.13}$$

and if $E = E_1 = E_2 = ... = E_k$, then

$$R = E^k \equiv \underbrace{E \oplus E \oplus ... \oplus E}_{k}. \tag{4.14}$$

11. Conditional operations (e.g. conditional dilation, conditional erosion, conditional thinning etc.):

An image operation $\cdot$ between an image X and a reference image (or image pairs) R performed within a limiting set Y is called a conditional operation and is denoted

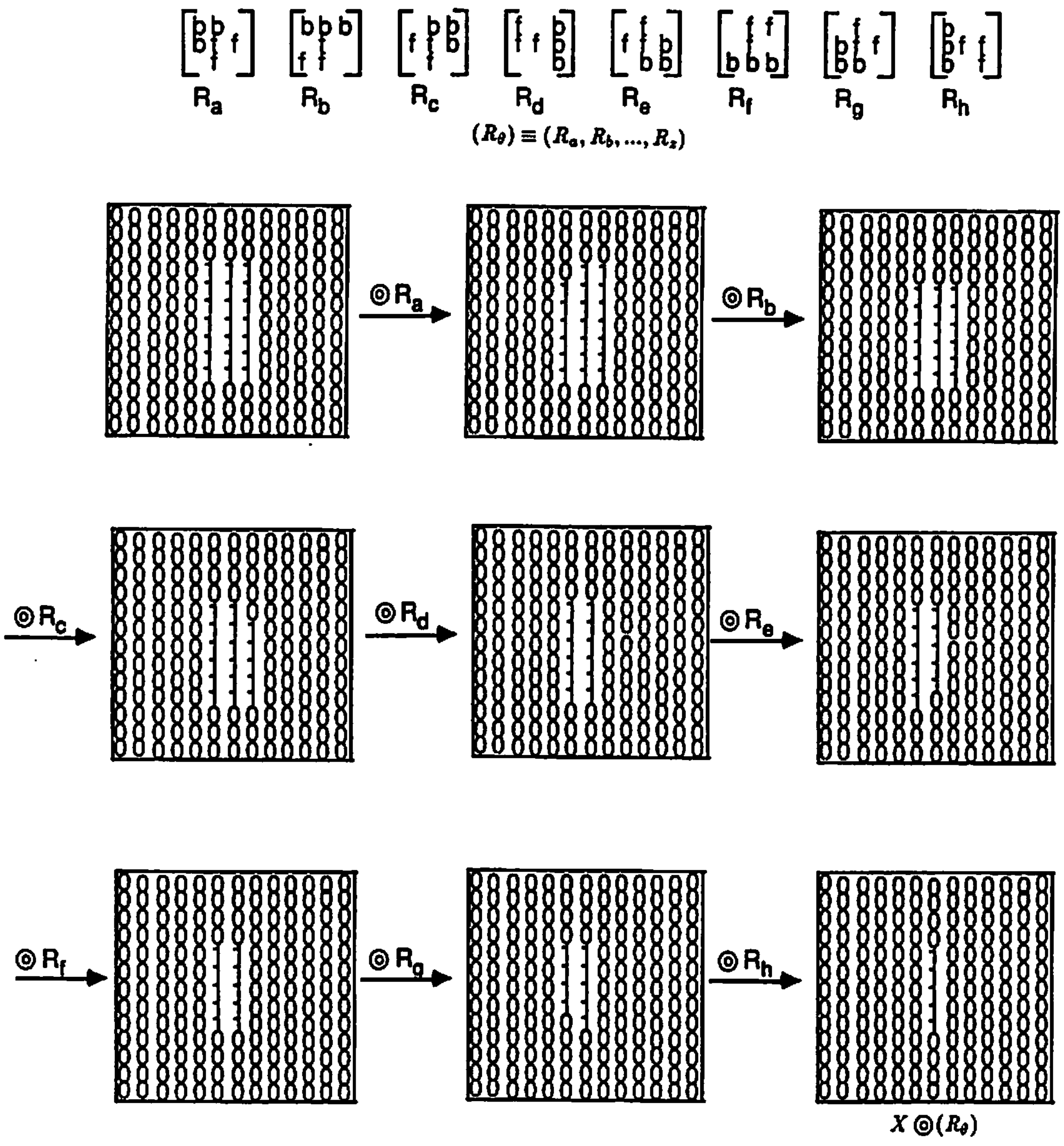


Figure 4.13: A sequential thinning (used for homotopics skeletonization) [Levialdi72].

$$X \cdot R \mid Y = (X \cdot R) \cap Y = \overline{\overline{X \cdot R} \cup \overline{Y}} \tag{4.15}$$

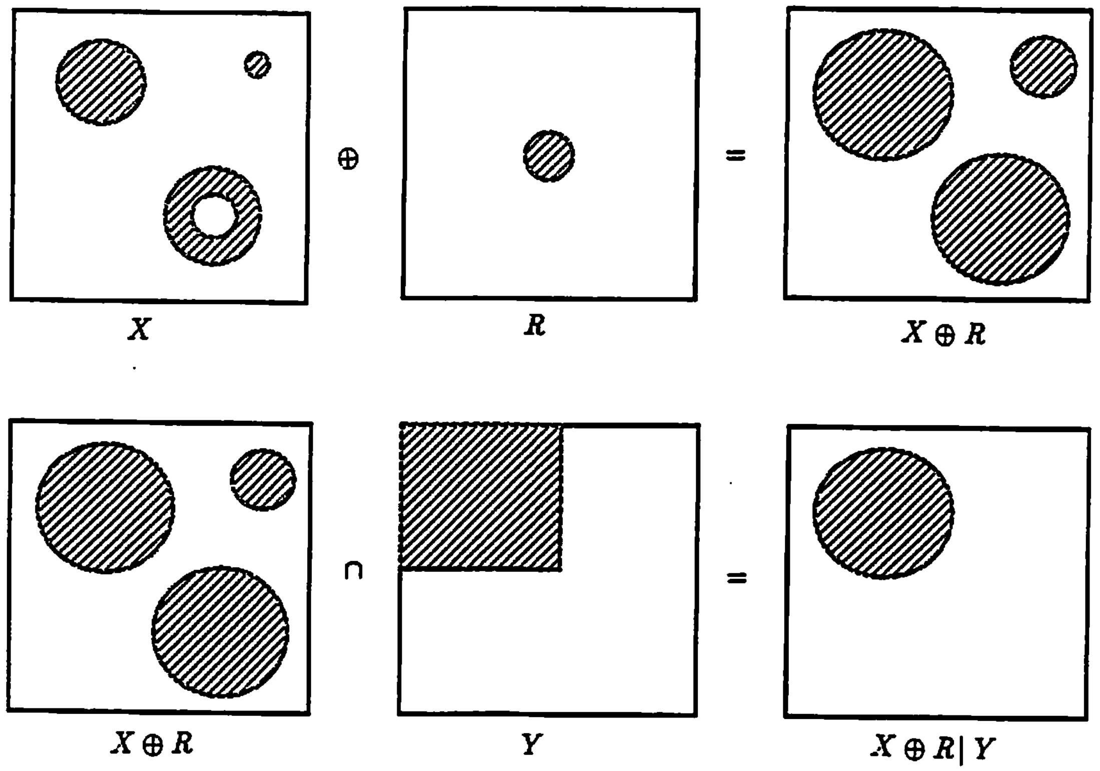


Figure 4.14: A conditional dilation.

Remark: Fig. 4.14 gives an example of the conditional dilation.

4.2 Examples of Special Cases: Translation (Shifting), Expansion, Shrinking, and Projection

Translation (shifting), expansion, shrinking and projection in a direction can by achieved by the dilation (or erosion) in a direct way.

1. Shifting an image X from coordinate (x, y) to coordinate $(x+i, y+j)$ is done by

$$X \oplus \{(i,j)\} = X \ominus \{(-i,-j)\}. \tag{4.16}$$

Remark: A point image $\{(i,j)\}$ corresponds to a discrete delta function at $\delta(x-i, y-j)$. Thus, an image function $X(x,y)$ (which corresponds to the image X) convolved with the delta function $\delta(x-i, y-j)$ or correlated with $\delta(x+i, y+j)$ is the same as $X \oplus \{(i,j)\} = X \ominus \{(-i,-j)\}$.

2. Adding a new 8-connected or 4-connected boundary to an image X (i.e. expansion) is done by

$$X \oplus N_4 \tag{4.17}$$

or

$$X \oplus N_8 \tag{4.18}$$

where $N_4 \equiv I \cup A \cup A^{-1} \cup B \cup B^{-1}$ and $N_8 \equiv \bigcup_{i,j=-1}^{1} A^i B^j$.

3. Removing the 8-connected or 4-connected boundary of an image X (i.e. shrinking) is done by

$$X \ominus N_4 = \overline{\overline{X} \oplus N_4} \tag{4.19}$$

or

$$X \ominus N_8 = \overline{\overline{X} \oplus N_8} \tag{4.20}$$

where $N_4 \equiv I \cup A \cup A^{-1} \cup B \cup B^{-1}$ and $N_8 \equiv \bigcup_{i,j=-1}^{1} A^i B^j$.

4. Projecting an image X to distance k in a direction θ, i.e. producing a shadow of X where the furthest image point in the shadow in the direction θ is at distance k from the furthest image point in X in the direction θ, this

can be achieved by

$$X \oplus \Theta^k \tag{4.21}$$

where Θ can be any one of the following:

- East: $E = I \cup A$, $E^k = \bigcup_{i=0}^{k} A^i$
- South: $S = I \cup B^{-1}$, $S^k = \bigcup_{i=0}^{k} B^{-i}$
- West: $W = I \cup A^{-1}$, $W^k = \bigcup_{i=0}^{k} A^{-i}$
- North: $N = I \cup B$, $N^k = \bigcup_{i=0}^{k} B^i$
- Southeast: $S_E = I \cup AB^{-1}$, $S_E^k = \bigcup_{i=0}^{k} A^i B^{-i}$
- Southwest: $S_W = I \cup A^{-1}B^{-1}$, $S_W^k = \bigcup_{i=0}^{k} A^{-i} B^{-i}$
- Northwest: $N_W = I \cup A^{-1}B$, $N_W^k = \bigcup_{i=0}^{k} A^{-i} B^i$
- Northeast: $N_E = I \cup AB$, $N_E^k = \bigcup_{i=0}^{k} A^i B^i$
- Horizontal: $H = \bigcup_{i=-1}^{1} A^i$, $H^k = \bigcup_{i=-k}^{k} A^i$
- Vertical: $V = \bigcup_{i=-1}^{1} B^i$, $V^k = \bigcup_{i=-k}^{k} B^i$
- Left-diagonal: $L_D = \bigcup_{i=-1}^{1} A^{-i} B^i$, $L_D^k = \bigcup_{i=-k}^{k} A^{-i} B^i$
- Right-diagonal: $R_D = \bigcup_{i=-1}^{1} A^i B^i$, $R_D^k = \bigcup_{i=-k}^{k} A^i B^i$

4.3 Theorems for Low Level Vision

Here we summarize four theorems and some examples for binary image processing applications. We first give basic properties of the BIA fundamental operations and standard operations. Then we describe the implementation of morphological filters, shape recognition algorithms, "salt" and "pepper" noise removal, size and location verifications, convex hull and connected component labeling. Those more obvious proofs are omitted for brevity.

Theorem 4.1 Properties of Image Operations

The BIA fundamental operations and standard operations have the properties shown in Table 4.1 and Table 4.2.

Proof: Appendix B gives some of their mathematical expressions which follow form the definitions.

Examples of Morphological Filters

Many image transformations are interpreted as morphological filtering [Serra82] or cellular filtering [Preston84]. Some major mophological filters are listed in the following:

1. One kind of morphological low pass filter (Fig. 4.15): to remove high frequencies in the foreground of an image X can be achieved by opening, i.e.

$$\begin{aligned} X \circ R &= (X \ominus R) \oplus R \\ &= \overline{\overline{X} \oplus \check{R}} \oplus R. \end{aligned} \qquad (4.22)$$

2. A second kind of morphological low pass filter (Fig. 4.16): to remove high frequencies in the background of an image X can be achieved by closing, i.e.

$$\begin{aligned} X \bullet R &= (X \oplus R) \ominus R \\ &= \overline{\overline{(X \oplus R)} \oplus \check{R}}. \end{aligned} \qquad (4.23)$$

Operations / Properties	Complement $\overline{X}$	Union $X \cup R$	Dilation $X \oplus R$	Difference X/R	Intersection $X \cap R$	Erosion $X \ominus R$
Increasing	No	Yes	Yes	Yes	Yes	Yes
Decreasing	Yes	No	No	No	No	No
Extensive	No	Yes	Yes (if $R \supset I$)	No	No	No
Antiextensive	No	No	No	Yes	Yes	Yes (if $R \supset I$)
Idempotent	No	Yes	No	Yes	Yes	No
Shift invariant	No	No	Yes	No	No	Yes
Homotopic	No	No	No	No	No	No
Commutative	No	Yes	Yes	No	Yes	No
Associative	No	Yes	Yes	No	Yes	No
Distributive (with some oper.)	No	Yes (with $\cap$)	Yes (with $\cup$)	No	Yes (with $\cup$, Δ)	No

Table 4.1: Basic properties of three fundamental operations and three derived operations (alternative fundamental operations).

Operations / Properties	Symmetric Difference $X \triangle R$	Opening $X \circ R$	Closing $X \bullet R$	Thinning $X \circledcirc R$	Thickening $X \odot R$	Homotopic skeletonization $X \circledcirc (R_\theta)$
Increasing	No	Yes	Yes	Yes	Yes	No
Decreasing	No	No	No	No	No	No
Extensive	No	No	Yes	No	Yes	No
Antiextensive	No	Yes	No	Yes	No	Yes
Idempotent	No	Yes	Yes	No	No	Yes
Shift invariant	No	Yes	Yes	Yes	Yes	Yes
Homotopic	No	No	No	No	No	Yes
Commutative	Yes	No	No	No	No	No
Associative	Yes	No	No	No	No	No
Distributive (with some oper.)	No	No	No	No	No	No

Table 4.2: Basic properties of some standard derived operations.

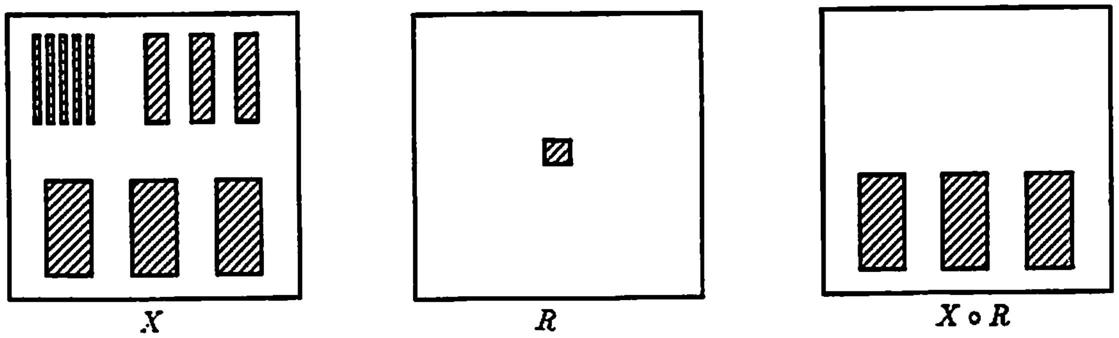


Figure 4.15: One kind of morphological low pass filter (opening).

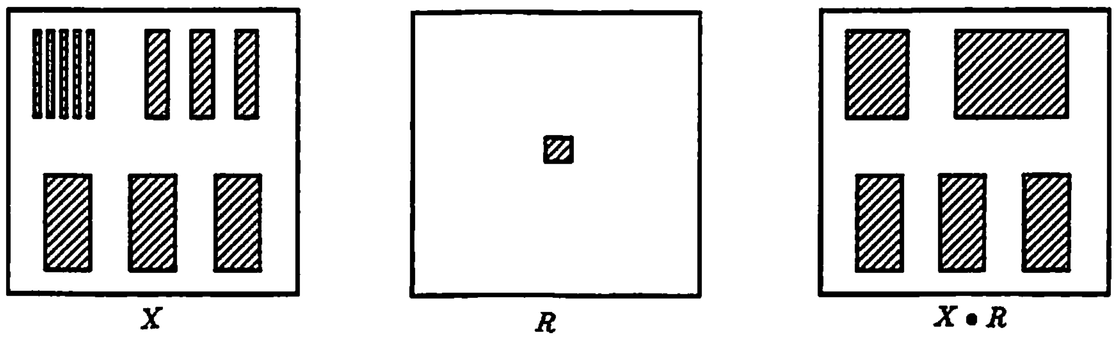


Figure 4.16: A second kind of morphological low pass filter (closing).

3. A morphological high pass filter (as shown in Fig. 4.17) which removes low frequencies in the foreground of an image X can be achieved by the difference of X and its opening, i.e.

$$\begin{aligned} X/(X \circ R) &= X/((X \ominus R) \oplus R) \\ &= X/(\overline{\overline{X} \oplus \check{R}} \oplus R) \\ &= \overline{\overline{X} \cup (\overline{\overline{X} \oplus \check{R}} \oplus R)}. \end{aligned} \tag{4.24}$$

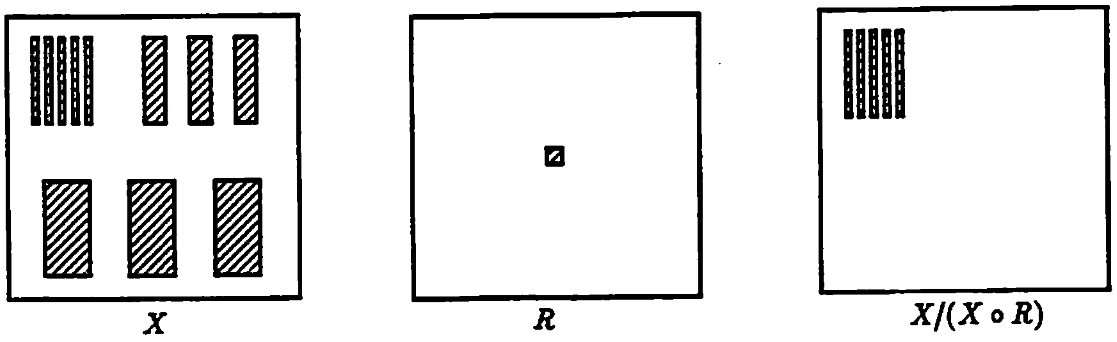


Figure 4.17: A morphological high pass filter.

4. A morphological band pass filter (as shown in Fig. 4.18) which removes low frequencies and high frequencies in the foreground of an image X can be achieved by the difference of its opening with a smaller reference image R and its opening with a larger reference image Q, where $R \subset Q$, i.e.

$$\begin{aligned} (X \circ R)/(X \circ Q) &= ((X \ominus R) \oplus R)/((X \ominus Q) \oplus Q) \\ &= ((\overline{\overline{X} \oplus \check{R}}) \oplus R)/((\overline{\overline{X} \oplus \check{Q}}) \oplus Q) \\ &= \overline{(\overline{\overline{\overline{X} \oplus \check{R}} \oplus R}) \cup (\overline{\overline{X} \oplus \check{Q}} \oplus Q)}. \end{aligned} \tag{4.25}$$

Theorem 4.2 Shape Recognition (Template Matching)

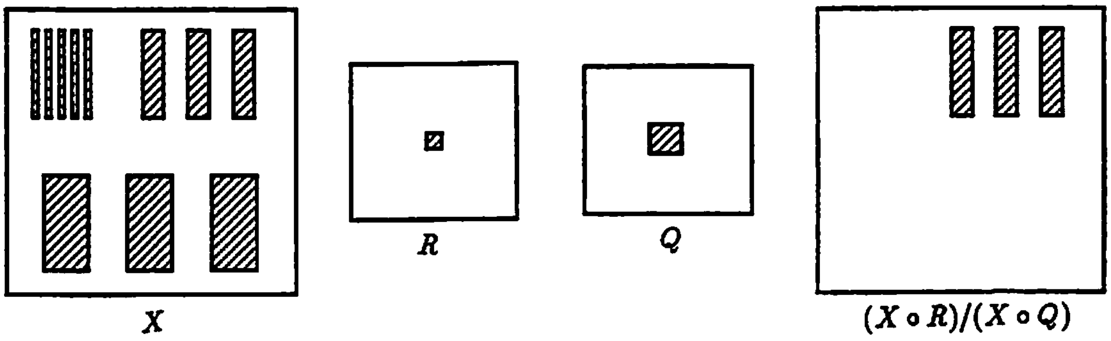


Figure 4.18: A morphological band pass filter.

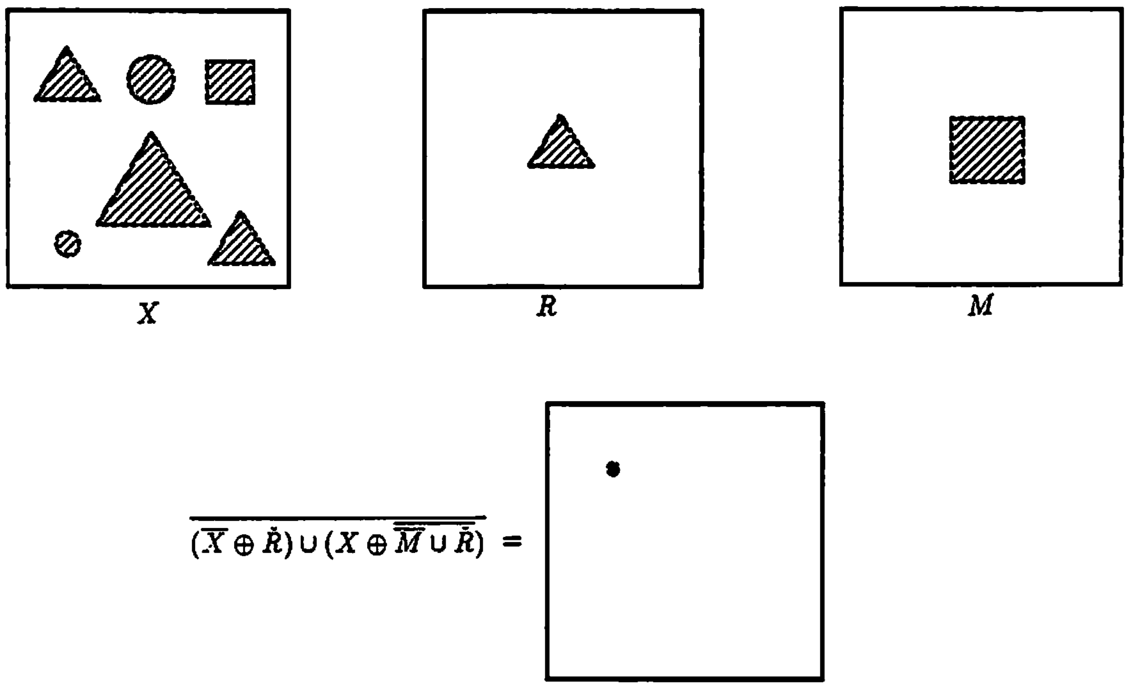
1. The locations of a shape, that is defined by a non-null reference image R and a non-null reference image (called mask) M (Fig. 4.19), with $R \subset M \subset W$ (W is the universal image), can be detected by

$$
\begin{aligned}
(X \ominus R) \cap (\overline{X} \ominus (M/R)) &= \overline{(\overline{X} \oplus \check{R}) \cup (X \oplus (\check{M}/\check{R}))} \\
&= \overline{(\overline{X} \oplus \check{R}) \cup (X \oplus \overline{\overline{\check{M}} \cup \check{R}})}.
\end{aligned} \tag{4.26}
$$

Equivalently, setting $R_1 = R$, $R_2 = M/R$ and redefining a non-null reference image pair $R = (R_1, R_2)$ (Fig. 4.20) yields the hit or miss transform of X by R:

$$X \circledast R = (X \ominus R_1) \cap (\overline{X} \ominus R_2) = \overline{(\overline{X} \oplus \check{R}_1) \cup (X \oplus \check{R}_2)}.$$

2. The locations of a shape, that is defined by a family of non-null reference image pairs $\{R(\theta)\}$ with $\theta \in \Theta$ (Θ is the index set of the family of non-null reference image pairs and $R(\theta) = (R_1(\theta), R_2(\theta))$, can be detected by the



$$\overline{(\overline{X} \oplus \check{R}) \cup (X \oplus \overline{\overline{M} \cup \check{R}})} =$$

Figure 4.19: One kind of shape recognition. R represents the shape to be identified, and must lie entirely and exclusively in the mask defined by M.

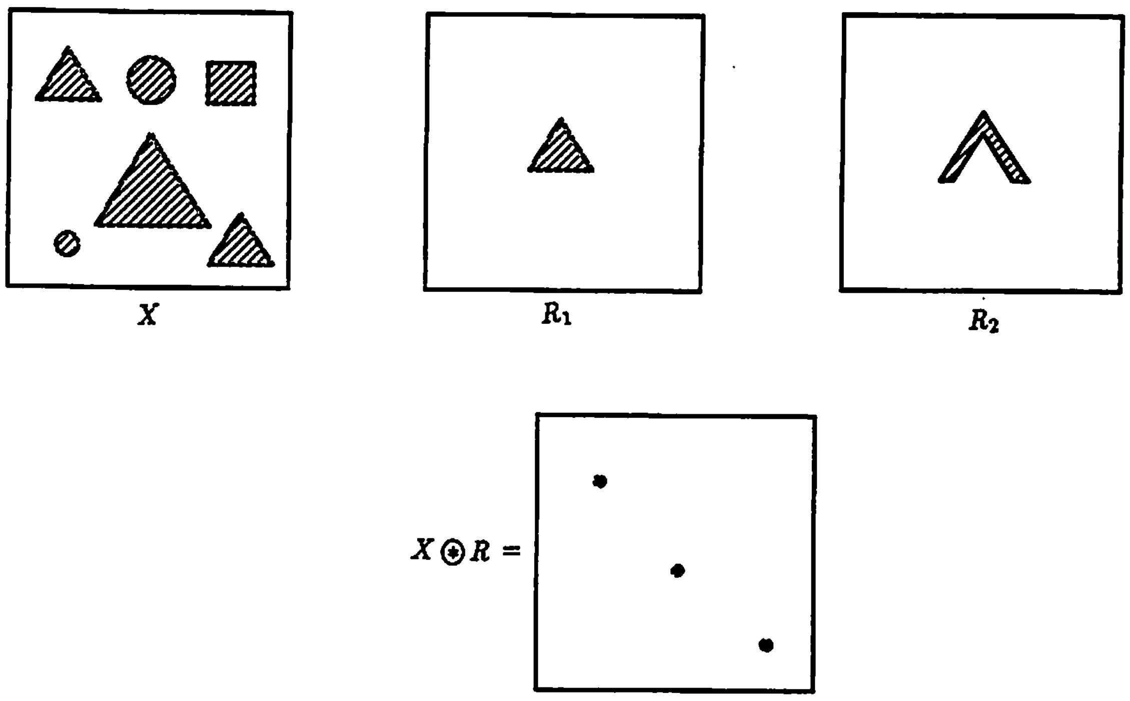


Figure 4.20: Hit or miss transform, which recognizes locations of foreground points given by R_1 in conjunction with background points given by R_2.

union of the hit or miss transform of X by $R(\theta)$:

$$\begin{aligned} \cup_{\theta\in\Theta} X \circledast R(\theta) &= \cup_{\theta\in\Theta}(X \ominus R_1(\theta)) \cap (\overline{X} \ominus R_2(\theta)) \\ &= \cup_{\theta\in\Theta} \overline{(\overline{X} \oplus \check{R}_1(\theta)) \cup (X \oplus \check{R}_2(\theta))}. \end{aligned} \tag{4.27}$$

Proof: Appendix C.1.

Theorem 4.3 "Salt" and "Pepper" Noise Removal

1. "Salt" noise removal (isolated image point removal) (Fig. 4.21(a)): removal of an image point if and only if all its 4-connected or 8-connected neighbors are background points (0's) can be achieved by

$$X \odot Q_4 = X \cup \overline{\overline{X} \oplus M_4} \tag{4.28}$$

or

$$X \odot Q_8 = X \cup \overline{\overline{X} \oplus M_8} \tag{4.29}$$

where $Q_4 = (M_4, I)$, $Q_8 = (M_8, I)$, $M_4 = A \cup A^{-1} \cup B \cup B^{-1} = N_4/I$ and $M_8 = N_8/I$.

2. "Pepper" noise removal (interior fill) (Fig. 4.21(b)): creation an image point at a coordinate if and only if all of its 4-connected or 8-connected neighbors are image points (1's) can be achieved by

$$X \circledcirc R_4 = \overline{\overline{X} \cup \overline{X \oplus M_4}} \tag{4.30}$$

or

$$X \circledcirc R_8 = \overline{\overline{X} \cup \overline{X \oplus M_8}} \tag{4.31}$$

where $R_4 = (I, M_4)$, $R_8 = (I, M_8)$.

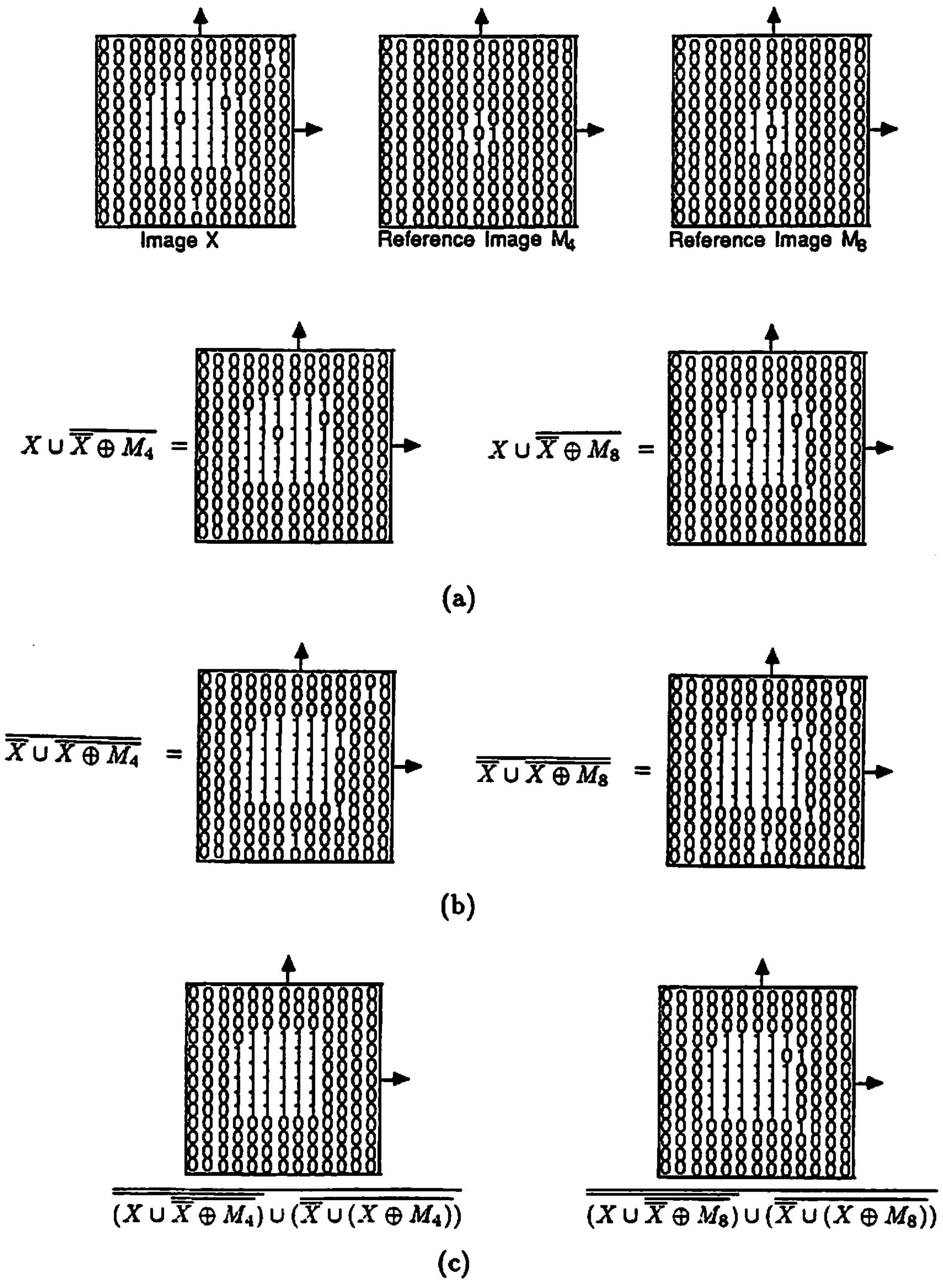


Image X
Reference Image M_4
Reference Image M_8
$X \cup \overline{\overline{X} \oplus M_4} =$
$X \cup \overline{\overline{X} \oplus M_8} =$
(a)
$\overline{\overline{X} \cup \overline{X \oplus M_4}} =$
$\overline{\overline{X} \cup \overline{X \oplus M_8}} =$
(b)
$\overline{(X \cup \overline{\overline{X} \oplus M_4}) \cup (\overline{\overline{X} \cup (X \oplus M_4)})}$
$\overline{(X \cup \overline{\overline{X} \oplus M_8}) \cup (\overline{\overline{X} \cup (X \oplus M_8)})}$
(c)

Figure 4.21: "Salt" and "pepper" noise removal. (a): "Salt" noise removal. (b): "Pepper" noise removal. (c) "Salt" and "pepper" noise removal.

3. "Salt and pepper" noise removal (Fig. 4.21(c)): removal of noise points, that are completely surrounded with 4-connected neighbors or 8-connected neighbors of the opposite value, can be achieved by

$$(X \odot Q_4)/(X \circledast R_4) = \overline{\overline{(X \cup \overline{\overline{X} \oplus M_4})} \cup (\overline{\overline{X} \cup (X \oplus M_4)})} \quad (4.32)$$

or

$$(X \odot Q_8)/(X \circledast R_8) = \overline{\overline{(X \cup \overline{\overline{X} \oplus M_8})} \cup (\overline{\overline{X} \cup (X \oplus M_8)})}. \quad (4.33)$$

Proof: Appendix C.2.

Remark: This theorem demonstrates the fact that many higher level operations (e.g. involving thinning and thickening) can be efficiently implemented by the three fundamental operations. Using the same design methodology as the "salt and pepper" noise removal, we can design many similar algorithms, such as spur removal, bridge break, and edge detection (perimeter) etc. For example, the detection of the 4-connected or 8-connected edge of an image X (Fig. 4.22) can be achieved by

$$X/(X \ominus N_8) = \overline{\overline{X} \cup (\overline{X} \oplus N_8)} \quad (4.34)$$

or

$$X/(X \ominus N_4) = \overline{\overline{X} \cup (\overline{X} \oplus N_4)}. \quad (4.35)$$

Theorem 4.4 Size and Location Verification

The locations in an image X of the regions including the reference image R and included in the reference image Q, where $R \subset Q$, can be detected by

$$S((X \ominus R)/((X \ominus Q) \oplus Q))) = S(\overline{(\overline{X} \oplus \check{R}) \cup (\overline{\overline{X} \oplus \check{Q}} \oplus Q)}). \quad (4.36)$$

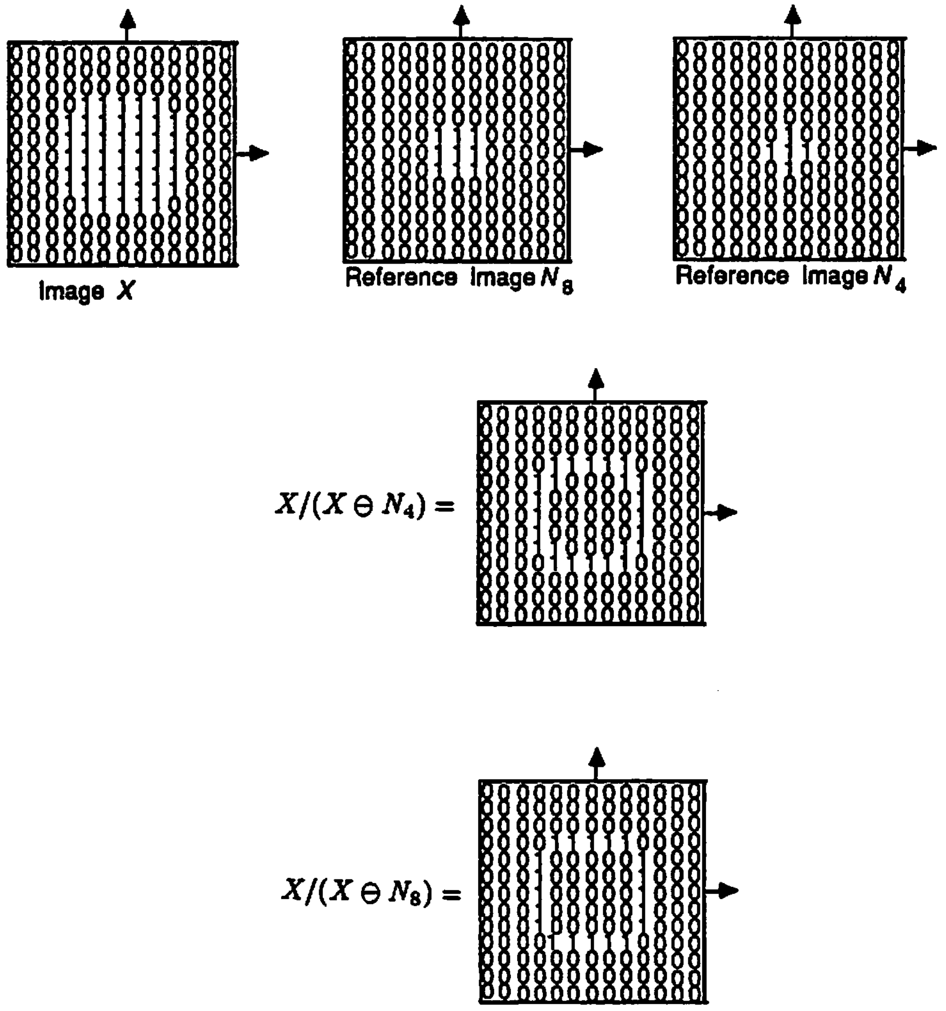


Figure 4.22: Edge detection.

where $S(\cdot)$ means the homotopic skeletonization. (An example is given in Fig. 4.23.)

Proof: Appendix C.3.

The above theorems serve as the typical rules for morphological image processing. In fact, there are many ways to analyze the shapes and sizes of an image by using only the three fundamental operations. As another example: comparing an image X with its convex hull $C(X)$ [Rosenfeld82] is a useful technique to analyze shape. If there is only one object or objects separated by distances greater than their own diameters in the image X, then its convex hull is the intersection of projections (Fig. 4.24):

$$C(X) = \bigcap_{i=1}^{4}(X \oplus \Theta_i^k) \tag{4.37}$$

where Θ_i, $i = 1,2,3,4$, are H, V, R_D, L_D (defined in Definition 4.4), and k should be greater than the longest radius of objects in X. Then the difference of the convex hull and the image $C(X)/X$ indicates how many concavities the image X has and what their individual shapes and sizes are. Fig. 4.25 illustrates an example.

One more observation is to apply the convex hull and the hit or miss transform for a particular class of connected component labeling problems: given an $N \times N$ image X which consists of k connected components (each with size smaller than $M \times M$, separated by distances larger than M, and $M \ll N$), label each connected component by a single image point (the upper left image point of its convex hull). This can be achieved by

$$\bigcup_{\theta\in\Theta} C(X) \circledast R(\theta) \tag{4.38}$$

where $C(X)$ is obtained from Eq. (4.37), the set of reference image pairs $\{R(\theta) \mid \theta \in \Theta\}$ contains three reference image pairs (Fig. 4.26), each corresponds to a

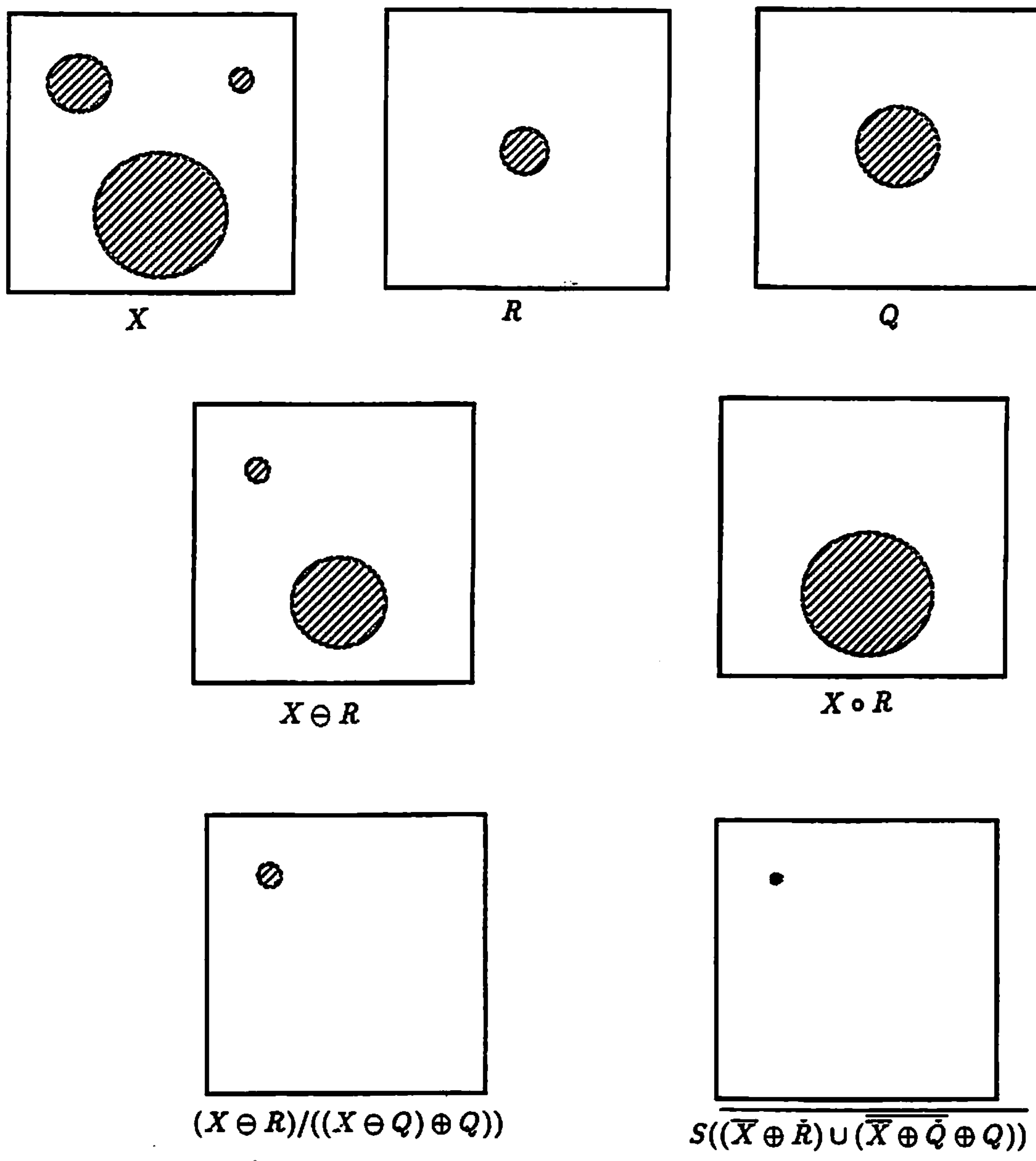

X
R
Q
$X \ominus R$
$X \circ R$
$(X \ominus R)/((X \ominus Q) \oplus Q))$
$S(\overline{(\overline{X} \oplus \check{R}) \cup (\overline{\overline{X} \oplus \check{Q}} \oplus Q))}$

Figure 4.23: A size verification (for holes).

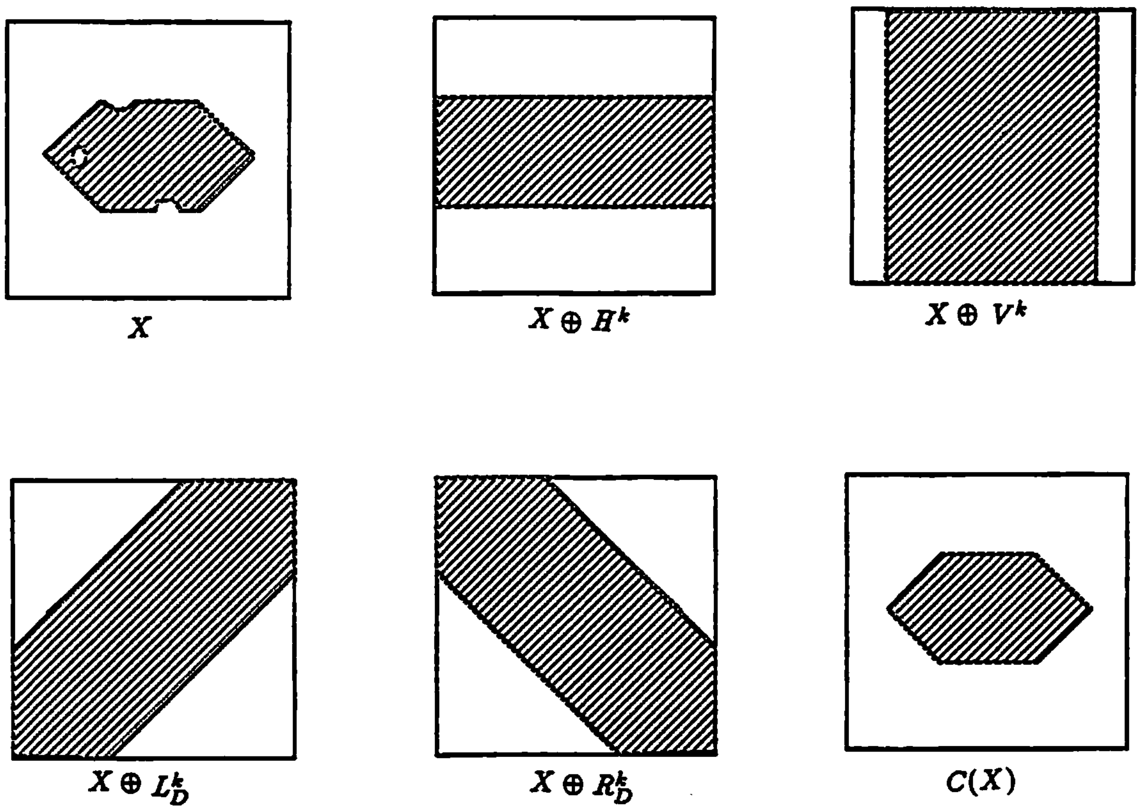


Figure 4.24: An example of the convex hull of an image X (implemented by the intersection of projections).

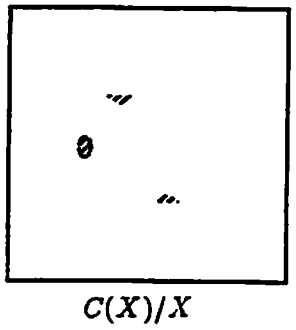


Figure 4.25: The difference of the convex hull $C(X)$ by X ($C(X)$ and X are shown in Fig. 4.24).

possible neighborhood of the upper left image point of convex hulls. This parallel algorithm can be executed on the DOCIP-array and the DOCIP-hypercube (Chapter 6) in $O(M)$ and $O(log_2 M)$ time respectively. This algorithm requires only simple cell hardware complexity and is executed much faster than other connected component labeling algorithms [Stout88]. The only problem is its strong restriction on the input image.

$$\left\{ \begin{bmatrix} b & b & b \\ b & f & b \\ b & b & b \end{bmatrix}, \begin{bmatrix} b & b & b \\ b & f & \\ & f & \end{bmatrix}, \begin{bmatrix} b & b & b \\ b & f & f \\ & & \end{bmatrix} \right\}$$

Figure 4.26: A set of reference image pairs $\{R(\theta) \mid \theta \in \Theta\}$ for labeling a particular class of connected components. Each array represents an image pair consisting of foreground points (value 1, represented by "f") and background points (value 0, represented by "b"). The origin is located at the center.

4.4 Parallel Pattern Recognition Based on BIA

A shape recognition algorithm has been summarized in Theorem 4.2 in Section 4.3. The practical difficulty with this theorem is that it is efficient only for shift invariant recognition and requires a large number of intricate reference image pairs to perform the recognition step in the presence of changes in scale, rotation or both. To solve this kind of invariance problem, in this section we recognize all the desired patterns by reversing the growing procedure of a family of patterns. This family defines all patterns in presence of such changes as scale and transforms all the desired patterns into their original seeds, which are usually chosen as isolated single image points.

4.4.1 Life, Pattern Growth and Pattern Recognition

These pattern recognition algorithms are motivated from the "games of life" and patterns of growth in cellular automata [Berlekamp82, Gardner70, Gardner71, Gardner83, Ulam62, Poundstone85, Wolfram84, Wolfram86, Farmer84, Demongeot85, Packard85, Preston84]. Cellular automata are mathematical systems constructed from many identical components, each simple, but together capable of complex behaviour. Growth from simple "seeds" in two-dimensional cellular automata can produce patterns with complicated structures. Several examples of cellular automata, such as Conway's game "Life" [Berlekamp82, Gardner70], illustrate that how extremely simple rules can be used to characterize very complex behavior. Binary image algebra (BIA) can efficiently describe these games or patterns of growth of two-dimensional cellular automata also. Some examples are:

- Ulam's transition rule [Ulam62]: A foreground image point (with value 1) never becomes a background point (with value 0) while a background point with a single foreground point in its 4-connected neighborhood becomes a foreground point (Fig. 4.27). BIA describes this as

$$X(t_{k+1}) = X(t_k) \cup \big(\bigcup_{\theta \in \Theta} X(t_k) \circledast R(\theta) \big) \qquad (4.39)$$

  where the set of $\{R(\theta) \mid \theta \in \Theta\}$ contains four reference image pairs (Fig. 4.28), each corresponding to a 4-connected neighborhood with a single foreground point; $X(t_{k+1})$ (at time t_{k+1}) is the image corresponding to the state subsequent to $X(t_k)$ in each iteration, and Θ is the index set of these four reference image pairs. The rule leads to infinite growth which is in general not self-reproducing.

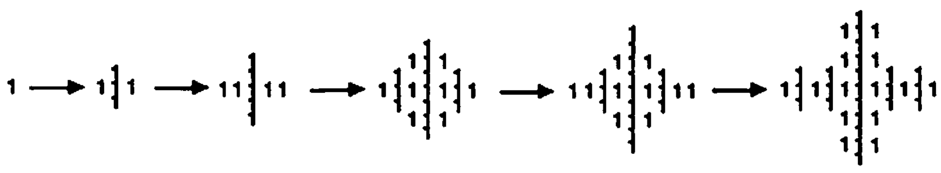


Figure 4.27: Pattern growth using Ulam's transition rule.

$$\left\{ \begin{bmatrix} & f & \\ b & b & b \\ & b & \end{bmatrix}, \begin{bmatrix} & b & \\ b & b & f \\ & b & \end{bmatrix}, \begin{bmatrix} & b & \\ b & b & b \\ & f & \end{bmatrix}, \begin{bmatrix} & b & \\ f & b & b \\ & b & \end{bmatrix} \right\}$$

Figure 4.28: A set of reference image pairs $\{R(\theta) \mid \theta \in \Theta\}$ for Ulam's transition rule. Each array represents an image pair consisting of foreground points (value 1, represented by "f") and background points (value 0, represented by "b"). The origin is located at the center.

- Fredkin's transition rule [Gardner71]: Any pixel (or point) with an even number of foreground points in its 4-connected neighborhood becomes a background point while a pixel with an odd number of foreground points in its 4-connected neighborhood becomes a foreground point. BIA describes this as

$$X(t_{k+1}) = \bigcup_{\theta \in \Theta} X(t_k) \circledast R(\theta) \tag{4.40}$$

where the set of $\{R(\theta) \mid \theta \in \Theta\}$ contains eight reference image pairs, each corresponding to a 4-connected neighborhood with an odd number of foreground points. This rule can lead to a self-reproducing pattern of growth. The same equation with a defferent set of $\{R(\theta) \mid \theta \in \Theta\}$ can be used to perform the Conway's game "Life" and other patterns of growth.

These games of life and patterns of growth remind us that an orignal "seed" with a simple rule may characterize a set of many extremely complex patterns. Now, in the reverse direction, if this set of complex patterns is a class of patterns to be recognized, then we apply a rule for growth reversal to transform those complex patterns into their orignial "seeds". By examing the "seeds", we can easily classify or recognize different patterns and determine their locations. For a simple example, we assume all the patterns generated from an isolated singe image point using Ulam's transition rule are considered as a class of patterns. To recognize the patterns of this class, we then apply the reversal of Ulam's transition rule (Fig. 4.29) as

$$X(t_{k+1}) = X(t_k)/(\bigcup_{\theta\in\Theta} X(t_k) \circledast R(\theta)) \qquad (4.41)$$

where the set of $\{R(\theta) \mid \theta \in \Theta\}$ contains eight reference image pairs (Fig. 4.30). For a pattern with diameter m, it will be transformed to the "seed" (an isolated single image point) after $m - 1$ iterations of the above equation.

The difficulty is in determining the "seed" and the rule of growth reversal for a class of patterns. In the next subsection, the BIA formulations for this idea will be described.

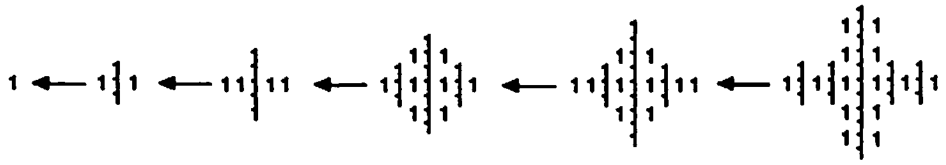


Figure 4.29: Pattern recognition using the reversal of Ulam's transition rule.

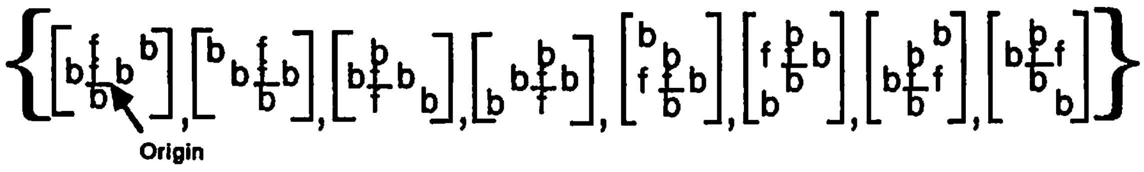


Figure 4.30: A set of reference image pairs $\{R(\theta) \mid \theta \in \Theta\}$ for the reversal of Ulam's transition rule.

4.4.2 BIA Formulations for Pattern Recognition

We have developed the BIA formulation for a class of pattern recognition problems. The algorithm involves two phases (Fig. 4.31): (1) the training phase chooses or calculates a small set of *acceptable* reference image pairs for characterizing sequences that present all the patterns of a class; and (2) the recognition phase recognizes all the desired patterns in an input image by transforming them into their original "seeds" (each seed represents the end of a sequence) and then picking up those desired "seeds" for indicating the locations of desired patterns. We will discuss two cases, distinguished by restrictions on the allowable sequences: (1) the recognition sequences of patterns that are decreasing in size ("shrinking" sequences), and (2) general sequences that have no such restriction (generalized "reduction" sequences).

Case 1. Recognition by A Shrinking Sequence

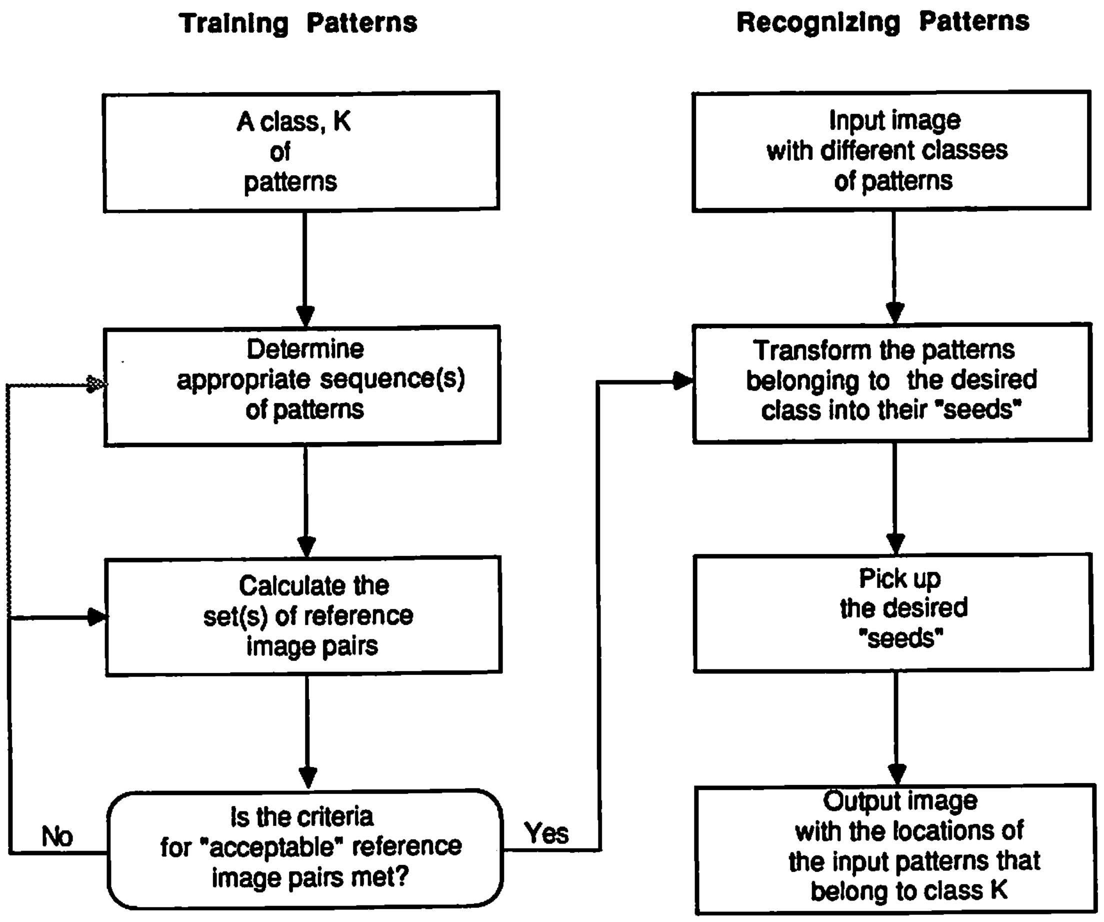

Training Patterns
Recognizing Patterns
A class, K of patterns
Determine appropriate sequence(s) of patterns
Calculate the set(s) of reference image pairs
Is the criteria for "acceptable" reference image pairs met?
No
Yes
Input image with different classes of patterns
Transform the patterns belonging to the desired class into their "seeds"
Pick up the desired "seeds"
Output image with the locations of the input patterns that belong to class K

Figure 4.31: Algorithm structure for BIA pattern recognition.

Suppose we want to recognize all patterns of a class (e.g. all solid square patterns) with different scale sizes and locations in an input image X (e.g. Fig. 4.32(a)) and produce the output image Y (e.g. Fig. 4.32(b)) with the recognition results. The procedure is:

1. Training the patterns of a class:

   - Step 1. Determine a growing sequence of the desired patterns T_i (e.g. Fig. 4.32(c)) which are increasing in size, i.e.

     $$T_0 \subset T_1 \subset ...T_i... \subset T_{m-1} \subset T_m \tag{4.42}$$

     where $0 \leq i \leq m$, T_0 determines the "seed" for the patterns of this growing sequence, and T_m is the largest pattern (having diameter m). The reversal of Eq. (4.42) represents the shrinking sequence. For simplicity we generally assume the same transition rule will be applied for each step $T_i \rightarrow T_{i-1}$ of the shrinking sequence.

   - Step 2. Calculate a small set of reference image pairs $\{R(\theta) \mid \theta \in \Theta\}$ (e.g. Fig. 4.32(d)) and an auxiliary set of reference image pairs $\{R(\lambda) \mid \lambda \in \Lambda\}$ (e.g. Fig. 4.32(e)) with the property:

     – Each reference image pair $R(\theta)$ corresponds to a possible neighborhood subimage of a given *foreground* image point of a pattern T_i, $1 \leq i \leq m$, whose previous state in the previous pattern T_{i-1} is a *background* point. Symbolically,

     $$\begin{aligned} & \{R(\theta) \mid \theta \in \Theta\} \qquad\qquad (4.43) \\ = \; & \{((P \oplus N) \cap T_i) \oplus \check{P} \mid P = \{(x,y)\} \subset T_i/T_{i-1}, 1 \leq i \leq m\} \end{aligned}$$

where $T_i/T_{i-1} = T_i \cap \overline{T_{i-1}} = \overline{\overline{T_i} \cup T_{i-1}}$, P is a point reference image containing only single foreground point which is also a foreground point in the difference image T_i/T_{i-1}, and N is the chosen neighborhood configuration (e.g. 3×3 neighborhood). This set will be used in the shrinking sequence for recognizing the desired patterns below.

- Each reference image pair $R(\lambda)$ corresponds to a possible neighborhood of a given *foreground* image point of a pattern T_i, $1 \leq i \leq m$, whose previous state in the previous pattern T_{i-1} is a *foreground* point. Symbolically,

$$\begin{aligned} & \{R(\lambda) \mid \lambda \in \Lambda\} \qquad (4.44) \\ = \ & \{((P \oplus N) \cap T_i) \oplus \check{P} \mid P = \{(x,y)\} \subset T_i \cap T_{i-1}, 1 \leq i \leq m\} \end{aligned}$$

where P is a point reference image containing only one single foreground point which is also a foreground point in the intersection of T_i and T_{i-1}. This set will be used in the next step to help to determine whether the growing sequence is "acceptable", i.e. whether $\{R(\theta) \mid \theta \in \Theta\}$ meets the criterion below.

- Step 3. Check the criteria of *acceptable* reference image pairs for $\{R(\theta) \mid \theta \in \Theta\}$:

  - The above two sets of reference image pairs $\{R(\theta) \mid \theta \in \Theta\}$ and $\{R(\lambda) \mid \lambda \in \Lambda\}$ must be disjoint, i.e.

$$\{R(\theta) \mid \theta \in \Theta\} \cap \{R(\lambda) \mid \lambda \in \Lambda\} = \phi \qquad (4.45)$$

This condition means that we can not expect to change a fore-gound image point into both foreground and background points simultaneously at a coordinate during the growth reversal.

– For multiple classes of patterns in an input image, we must also require that the patterns of a class never become the seeds of other classes when applying the transition rules of other classes. One way of ensuring this is to require the sets of reference image pairs $\{R(\theta) \mid \theta \in \Theta\}$ for different classes to be disjoint. This avoids errors in the recognition process.

If the above criteria are not satisfied, then we have to modify the growing sequence or increase the neighborhood configuration (i.e. the size of reference image pairs) until they are satisfied.

2. Recognizing the desired patterns:

- Step 1. Transform the desired patterns $T_i, i = 1, 2, ..., m$, in the 2-D input image X into their original seeds T_0 (usually consisting of one and only one foreground image point):

  This is to reduce patterns of the same class (e.g. having different scales and locations) in an input image $X = X(t_0)$ to a set of seeds (i.e. isolated single foreground image points usually) by the recursive relation:

$$X(t_{k+1}) = X(t_k) / \bigcup_{\theta \in \Theta} X(t_k) \circledast R(\theta) \tag{4.46}$$

  where $0 \leq k \leq m$.

- Step 2. Pick up the original seeds by the equation

$$Y = X(t_m) \circledast Q \tag{4.47}$$

where Q (e.g. Fig. 4.32(f)) is a reference image pair associated with the seed, (i.e. with one and only one foreground image point at the center usually), and Y is the final recognition output.

This procedure is shift invariant because of the parallel BIA formalism. Scale invariance can be incorporated by using an appropriate sequence of patterns that represent scaled versions of the same basic pattern. In addition, rotation invariance can be incorporated by using multiple sequences, one for each different orientation (since the input space is discrete, the number of different orientations is finite and in some cases is small, e.g. 4 or 8 orientations). The main restriction (Eq. (4.42)) can be removed in the more general procedure described below.

Case 2. Recognition by A Generalized Reduction Sequence

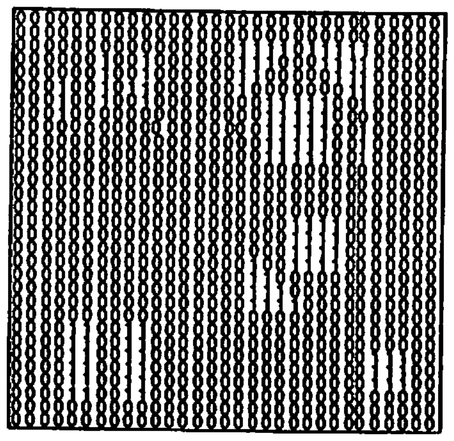
In this case, the patterns in the growing sequence need not satisfy the increasing size condition (Eq. (4.42)). We can modify the above procedure as follows:

1. Training the patterns of a class:

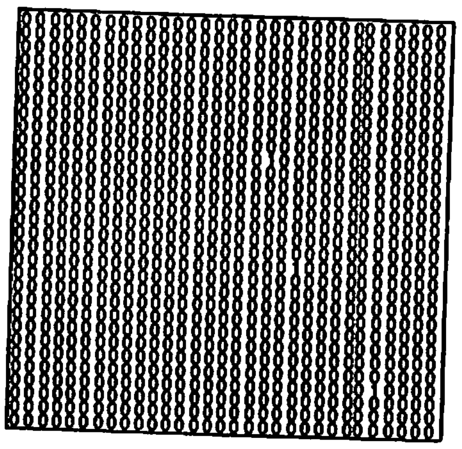
    - In addition to calculating $\{R(\theta) \mid \theta \in \Theta\}$ and $\{R(\lambda) \mid \lambda \in \Lambda\}$, we also calculate one more set of reference image pairs $\{R(\gamma) \mid \gamma \in \Gamma\}$ and one more auxiliary set of reference image pairs $\{R(\xi) \mid \xi \in \Xi\}$:

        – Each reference image pair $R(\xi)$ corresponds to a possible neighborhood of a given *background* image point of a pattern T_i, $1 \leq i \leq m$, whose previous state in the previous pattern T_{i-1} is a *background* point. Symbolically,

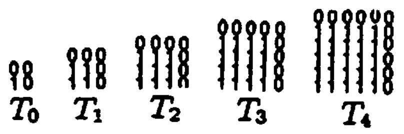
$$\begin{aligned} & \{R(\gamma) \mid \gamma \in \Gamma\} & (4.48) \\ = \; & \{((P \oplus N) \cap T_i) \oplus \check{P} \mid P = \{(x,y)\} \subset \overline{T_i} \cap \overline{T_{i-1}}, 1 \leq i \leq m\} \end{aligned}$$



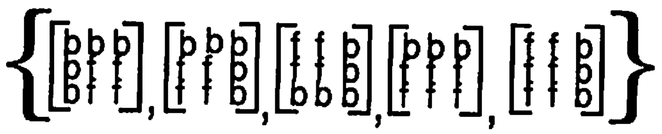
(a)



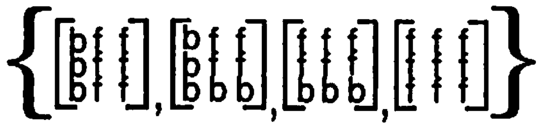
(b)



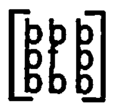
(c)



(d)



(e)



(f)

Figure 4.32: A shift and scale invariant pattern recognition of solid square patterns, as an example of recognition by a shrinking sequence. (a): An input image X. (b) The output image Y. (c): The growing sequence of solid square patterns T_i, $0 \leq i \leq 4$. (d): A set of *acceptable* reference image pairs $\{R(\theta) \mid \theta \in \Theta\}$ for solid square patterns with different scales. (e): An auxiliary set of reference image pairs $\{R(\lambda) \mid \lambda \in \Lambda\}$ for solid square patterns with different scales. (f): The reference image pair Q.

where P is a point reference image containing only one single foreground point which is also a foreground point in the intersection of $\overline{T_i}$ by $\overline{T_{i-1}}$. The sets $\{R(\theta) \mid \theta \in \Theta\}$ and $\{R(\gamma) \mid \gamma \in \Gamma\}$ will be used to recognize the desired patterns.

– Each reference image pair $R(\xi)$ corresponds to a possible neighborhood subimage of a given *background* image point of a pattern T_i, $1 \leq i \leq m$, whose previous state in the previous pattern T_{i-1} is a *foreground* point. Symbolically,

$$\begin{aligned} & \{R(\xi) \mid \xi \in \Xi\} \\ = \; & \{((P \oplus N) \cap T_i) \oplus \check{P} \mid P = \{(x,y)\} \subset T_{i-1}/T_i, 1 \leq i \leq m\} \end{aligned} \tag{4.49}$$

where P is a point reference image containing only one single foreground point which is also a foreground point in the difference of T_{i-1} by T_i. This set will help to determine whether $\{R(\gamma) \mid \gamma \in \Gamma\}$ is "acceptable" or not.

- Because we can not expect to change a background image point into both foreground and background points simutaneously, we have to add the criterion that the sets $\{R(\gamma) \mid \gamma \in \Gamma\}$ and $\{R(\xi) \mid \xi \in \Xi\}$ must be disjoint. Similarly, the sets of reference image pairs $\{R(\gamma) \mid \gamma \in \Gamma\}$ for different classes must be disjoint to ensure that the patterns of a class never become the seeds of other classes when applying the transition rules of other classes.

2. Recognizing the desired patterns:

- Instead of the Eq. (4.46), we transform the desired patterns $T_i, i = 1, 2, ..., m$, in the 2-D input image X into their original seeds T_0 by the recursive relation:

$$X(t_{k+1}) = (\bigcup_{\gamma \in \Gamma} X(t_k) \circledast R(\gamma)) \cup (X(t_k) / \bigcup_{\theta \in \Theta} X(t_k) \circledast R(\theta)) \quad (4.50)$$

  where $0 \leq k \leq m$. Finally, we pick up the original seeds by $Y = X(t_m) \circledast Q$ as desbribed in Case 1.

This case allows more general pattern sequences such as hollow objects (e.g. sequences of successively larger line objects such as squares or rectangles). In both cases, this algorithm can reduce the computation complexity for a class of pattern recognition and symbolic substitution problems because of its inherent parallelism. In addition, the concept of using sequence(s) of patterns to describe a class may allow a simple description of classes that are not easily represented by a set of features, such as different natural pattern textures. How easily the BIA technique described here can be applied to such problems is not known at present.

Chapter 5

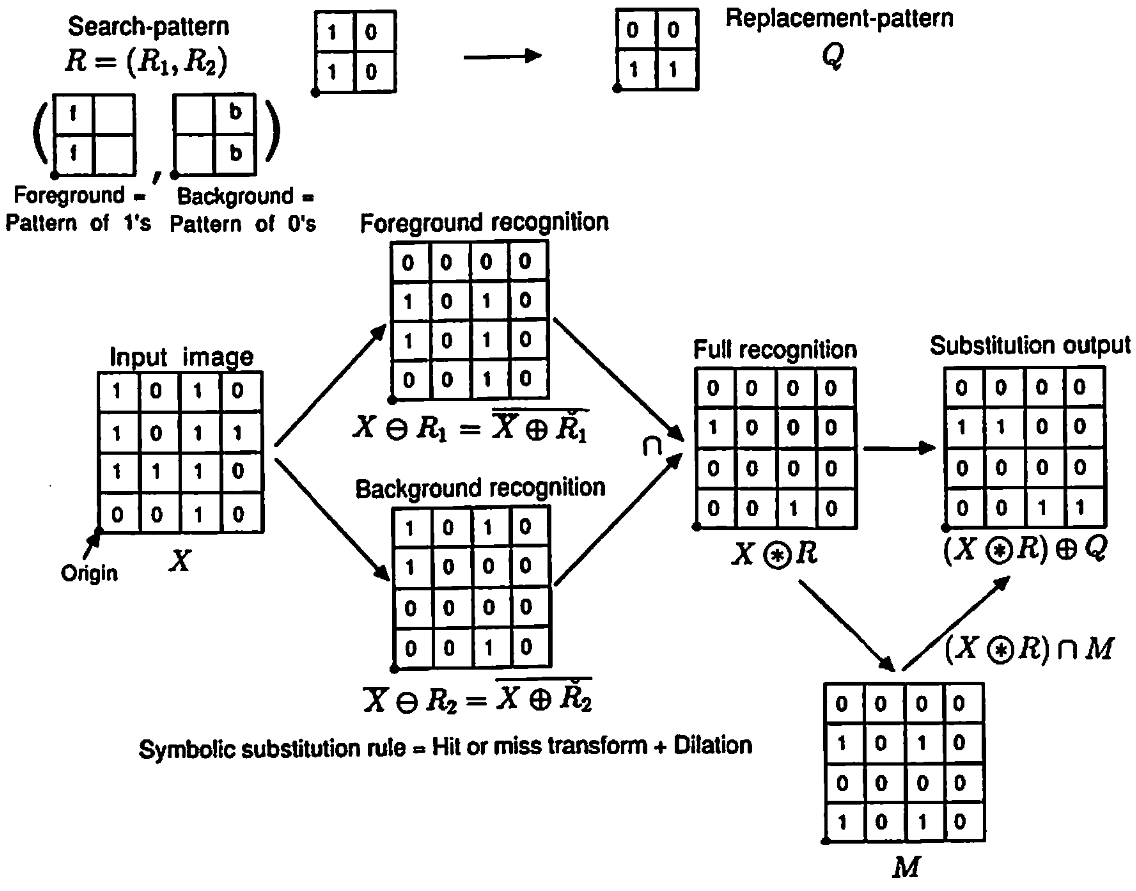
BIA: Relationship to Other Computing Theories

5.1 Relationship to Symbolic Substitution

Symbolic substitution was first considered as a means of utilizing the parallelism of optics by Huang [Huang83]. Recently, the use of symbolic substitution as a basis for digital optical computing has been reported in [Huang83, Brenner85, Brenner86a, Brenner86b, Brenner87, Mait87, Cloonan87, Capps87, Ramamoorthy87, Jeon87, Taso87]. Special symbolic substitution rules can be applied to perform arithmetic operations and simulate a Turing machine [Brenner86a]. Symbolic substitution can solve any computable problem and performs many operations. Here we formalize symbolic substitution by BIA algebraic symbols, demonstrate that symbolic substitution rules are particular BIA image transformations, and give some BIA algebraic techniques for deriving and comparing symbolic substitution algorithms.

5.1.1 BIA Representation of Symbolic Substitution

In this subsection we derive the BIA equation for symbolic substitution. A symbolic substitution rule involves two steps: 1) recognizing the locations of a certain search-pattern within the 2-D binary input data, and 2) substituting a replacement-pattern wherever the search-pattern is recognized. We derive it by BIA in the following steps (illustrated in Fig. 5.1):



Search-pattern
$R = (R_1, R_2)$
Foreground = Pattern of 1's
Background = Pattern of 0's
Replacement-pattern
Q
Foreground recognition
$X \ominus R_1 = \overline{\overline{X} \oplus \check{R}_1}$
Background recognition
$\overline{X} \ominus R_2 = \overline{X \oplus \check{R}_2}$
Input image
Origin
X
Full recognition
$X \circledast R$
Substitution output
$(X \circledast R) \oplus Q$
$(X \circledast R) \cap M$
M
Symbolic substitution rule = Hit or miss transform + Dilation

Figure 5.1: BIA representation of symbolic substitution. The optional mask M is used for controlling the block search region.

1. BIA Notations for Symbolic Substitution:

   - 2-D binary input data = image (bit plane) X
   - Symbol to be recognized (search-pattern) = reference image (or image pairs) R
   - Symbol to be replaced (replacement-pattern) = reference image Q

2. A Symbolic Substitution Rule:

   - Step 1. recognition of the search-pattern:

     (a) Foreground recognizer: the locations of a spatial search-pattern R_1 (a set of 1's called the *foreground*) within the 2-D input data X can be recognized by the erosion operation of X and R_1:

     $$X \ominus R_1 = \overline{\overline{X} \oplus \check{R}_1}. \tag{5.1}$$

     (b) Background recognizer: the locations of a spatial search-pattern R_2 (a set of points that correspond to 0's which is called the *background*) within the 2-D input data X can be recognized by the erosion of $\overline{X}$ and R_2:

     $$\overline{X} \ominus R_2 = \overline{X \oplus \check{R}_2}. \tag{5.2}$$

     (c) Full recognizer: by combining the two above steps, the locations of a certain spatial search-pattern $R = (R_1, R_2)$ (R_1 defines the foreground, and R_2 defines the background) within the 2-D input data X can be recognized by the hit or miss transform of X and R:

     $$\begin{aligned} X \circledast R &= (X \ominus R_1) \cap (\overline{X} \ominus R_2) \\ &= \overline{(\overline{X} \oplus \check{R}_1) \cup (X \oplus \check{R}_2)}. \end{aligned} \tag{5.3}$$

- Step 2. substitution of the replacement-pattern:
  - Substituter: a new replacement-pattern Q can be substituted for R wherever the search-pattern R is recognized by the dilation of $X \circledast R$ by Q.
- Synthesis:
  - A complete symbolic substitution rule is implemented by the hit or miss transform of X by R followed by the dilation by Q:

$$\begin{aligned}(X \circledast R) \oplus Q &= ((X \ominus R_1) \cap (\overline{X} \ominus R_2)) \oplus Q \\ &= \overline{(\overline{X} \oplus \check{R}_1) \cup (X \oplus \check{R}_2)} \oplus Q.\end{aligned} \tag{5.4}$$

- Optional masking:
  - An optional mask M can be used for controlling the block search region. A symbolic substitution rule can be modified as:

$$((X \circledast R) \cap M) \oplus Q. \tag{5.5}$$

    By proper choice of M, the search can be made in overlapping, disjoint or non-contiguous blocks.

3. A symbolic substitution system (Fig. 5.2):

- This idea can be extended to more than one rule (say p substitution rules as shown in Fig. 5.2). Herre a symbolic substitution processor produces several copies of the input X, provides p different recognizer-substituter units, and then combines the outputs of various units to form a new output. Thus, a symbolic substitution system is implemented by

$$\bigcup_{i=1}^{p} (X \circledast R^{(i)}) \oplus Q^{(i)} \tag{5.6}$$

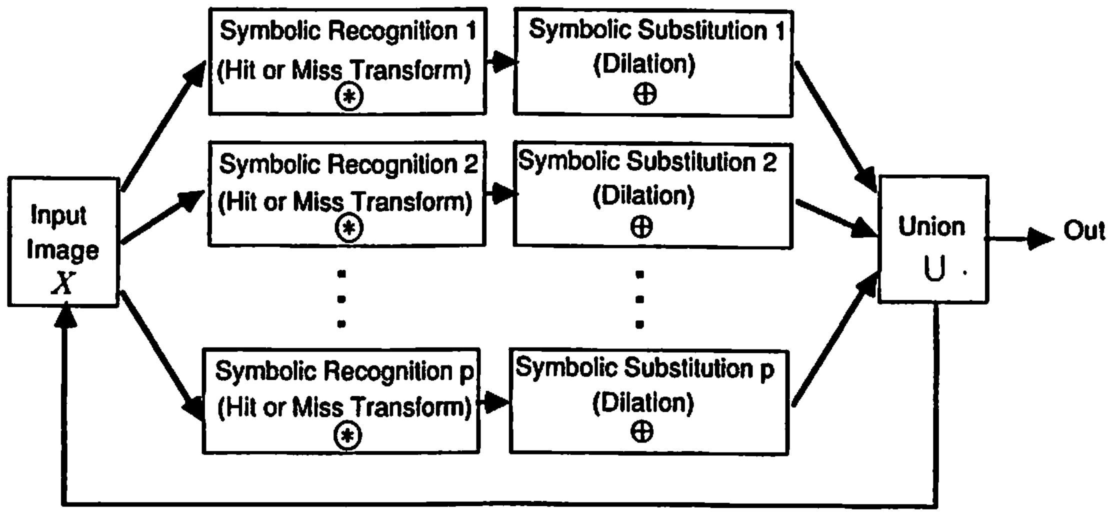


Figure 5.2: A symbolic substitution system with p symbolic substitution rules.

or

$$\bigcup_{i=1}^{p}((X \circledast R^{(i)}) \cap M) \oplus Q^{(i)} \tag{5.7}$$

where $R^{(i)}$ and $Q^{(i)}$, $i = 1, 2, ..., p$, are the reference image pairs and replacement patterns in the i^{th} symbolic substitution rule. This, then, is the BIA formula for general symbolic substitution.

Hence, a general mathematical formalism of symbolic substitution has been developed. BIA represents a general complete systematic mathematical tool for formalizing the symbolic substitution algorithms.

5.1.2 Examples of Symbolic Substitution Using BIA

In many special cases the above form for symbolic subsitution (Eq. (5.6) or Eq. (5.7)) is inefficient and can be reduced to a relatively simpler form or implemented in a more efficient way by using some BIA algebraic techniques. Here are some examples:

- Case 1. $R_1^{(i)} = \phi$ (null image) or $R_2^{(i)} = \phi$:

  This implies
  $$(X \circledast R^{(i)}) \oplus Q^{(i)} = (\overline{X} \ominus R_2^{(i)}) \oplus Q^{(i)} \tag{5.8}$$
  or
  $$(X \circledast R^{(i)}) \oplus Q^{(i)} = (X \ominus R_1^{(i)}) \oplus Q^{(i)}, \tag{5.9}$$
  so that we can implement the full recognition of the i^{th} symbolic substitution rule by implementing only the background recognition or foreground recognition.

- Case 2. $Q^{(i)} = \phi$:

  This implies
  $$(X \circledast R^{(i)}) \oplus Q^{(i)} = \phi, \tag{5.10}$$
  so that the i^{th} symbolic substitution rule is not needed and has no effect at all. For example, the subtraction of two binary numbers stored in a 2-D bit array can be described symbolically as a sequence of BIA operations. The four general rules of recognition and substitution needed to implement subtraction can be reduced to two rules using Eq. (5.10) [Huang88b].

- Case 3. $X \ominus R_1^{(i)} = \overline{X} \ominus R_2^{(i)}$:

  This implies
  $$X \circledast R^{(i)} = \overline{X} \ominus R_2^{(i)} = X \ominus R_1^{(i)}, \tag{5.11}$$
  so that we can implement the full recognition by either the background recognition or foreground recognition, alone. This always happens in the

case of the dual-rail coding, so that dual-rail coding implements symbolic substitution according to

$$\bigcup_{i=1}^{p}(\overline{X} \ominus R_2^{(i)}) \oplus Q^{(i)} = \bigcup_{i=1}^{p} \overline{X \oplus \check{R}_2^{(i)}} \oplus Q^{(i)}; \tag{5.12}$$

or

$$\bigcup_{i=1}^{p}((\overline{X} \ominus R_2^{(i)}) \cap M) \oplus Q^{(i)} = \bigcup_{i=1}^{p}(\overline{X \oplus \check{R}_2^{(i)}} \cap M) \oplus Q^{(i)}. \tag{5.13}$$

However, dual-rail coding requires more complicated coding and doubles the device area.

- Case 4. $R_1^{(i)} = Q^{(i)}$, $i = 1, 2, ..., k$, $k \leq p$:

For $k = p$, we simply replace the input image X with the same image X, and we do not need to process the image at all. For $k < p$, we can keep the portion of the input image which contains the patterns $R_1^{(i)}$, $i = 1, 2, ..., k$, and process only the other portion of the input image as follows

$$\begin{aligned} & \cup_{i=1}^{p}(X \circledast R^{(i)}) \oplus Q^{(i)} \\ = \ & (X / \cup_{i=k+1}^{p}(X \circledast R^{(i)}) \oplus R_1^{(i)}) \cup (\cup_{i=k+1}^{p}(X \circledast R^{(i)}) \oplus Q^{(i)}) \\ = \ & (X / \cup_{i=k+1}^{p}(X \circledast R^{(i)}) \oplus N_c^{(i)}) \cup (\cup_{i=k+1}^{p}(X \circledast R^{(i)}) \oplus Q^{(i)}) \end{aligned} \tag{5.14}$$

where $N_c^{(i)} = R_1^{(i)} \cup R_2^{(i)}$ specifies the neighborhood configuration. In symbolic subtitution, we usually choose $N_c^{(i)}$ as a constant image, i.e. $N_c^{(i)} = R_1^{(i)} \cup R_2^{(i)} = N_c$ for all i, then Eq. (5.14) can also be implemented as

$$\begin{aligned} & (X / (\cup_{i=k+1}^{p}(X \circledast R^{(i)}) \oplus N_c)) \cup (\cup_{i=k+1}^{p}(X \circledast R^{(i)}) \oplus Q^{(i)}) \\ = \ & (X / ((\cup_{i=k+1}^{p}(X \circledast R^{(i)})) \oplus N_c)) \cup (\cup_{i=k+1}^{p}(X \circledast R^{(i)}) \oplus Q^{(i)}) \end{aligned} \tag{5.15}$$

where the first term produces the portion of image containing the patterns which will be substituted with the same patterns. Obviously, the efficiency of this algorithm is increased as the number k is increased or the size of reference images (or image pairs) is increased. This algorithm can reduce the number of required BIA fundamental operations, compared to the implementation with Eq. (5.6). If we have a special control structure to update only a portion of the input image, where the encoded symbols are replaced with new symbols (the other portion of the image remains the same), without extra time. An example with $k = p/2$ occurs in the parallel binary addition of symbol-coded data (Chapter 8), in which we do not assume the existence of such special control structure in Chapter 8 for reserving the hardware simplicity. Instead of using this algorithm, we can see that one rule of binary addition and two rules of binary subtraction in single-pixel coding also satisfy the condition of Case 2, which algorithm is used in Chapter 8.

- Case 5. $Q^{(i)} = Q$ for all $1 \leq i \leq p$:

  This implies that we should combine the results of the hit or miss transforms first and then replace them by the same replacement-pattern Q instead of implementing p substitution units for realizing the same substitution step, i.e.

$$(\bigcup_{i=1}^{p} X \circledast R^{(i)}) \oplus Q. \tag{5.16}$$

  This happens in cases when a type of pattern is defined by a set of reference image pairs $R^{(i)}$, $i = 1, 2, ..., p$.

As we will see later, BIA also suggests a digital optical cellular image processor (DOCIP) architecture which can implement all the above algorithms of symbolic substitution in an efficient, programmable, and highly parallel way.

5.1.3 Symbolic Substitution Representation of BIA

BIA provides useful parallel algorithms for image processing and numerical array computation (Chapter 4 and Chapter 8). To systematically develop parallel algorithms for symbolic substitution processors (Fig. 5.2), we first list symbolic substitution rules for the three fundamental operations of BIA (Chapter 3). Then we can derive symbolic substituion rules for all BIA parallel algorithms. Choosing some particular search-patterns and replacement-patterns, symbolic substitution rules represent the three fundamental operations as follows

1. Complement can be implemented by a symbolic substitution rule as

$$\overline{X} = (X \circledast R_c) \oplus I \tag{5.17}$$

   where the search-pattern is the reference image pair $R_c = (\phi, I)$, ϕ is a null image and I is an elementary image (also the identity of dilation) defined in Chapter 3, the replacement-pattern is also the elemetnary image I.

$$\begin{aligned} Proof: \ (X \circledast R_c) \oplus I &= ((X \ominus \phi) \cap (\overline{X} \ominus I)) \oplus I \\ &= (W \cap \overline{X}) \oplus I \\ &= \overline{X} \oplus I \\ &= \overline{X}. \end{aligned}$$

2. Union can be implemented by two symbolic substitution rules as

$$X \cup R = ((X \circledast R_u) \oplus I) \cup ((R \circledast R_u) \oplus I) \tag{5.18}$$

   where $R_u = (I, \phi)$ is the right identiy of the hit or miss transform $\circledast$.

$$Proof: \qquad ((X \circledast R_u) \oplus I) \cup ((Y \circledast R_u) \oplus I)$$

$$= ((X \ominus I) \cap (\overline{X} \ominus \phi)) \cup ((Y \ominus I) \cap (\overline{X} \ominus \phi))$$
$$= (X \cap W) \cup (Y \cap W)$$
$$= X \cup Y.$$

3. Dilation can be implemented by a symbolic substitution rule as

$$X \oplus R = (X \circledast R_u) \oplus R \tag{5.19}$$

where the search-pattern is $R_u = (I, \phi)$ as used in the union and the replacement-pattern is R.

$$\begin{aligned} Proof: \ (X \circledast R_u) \oplus R &= ((X \ominus I) \cap (\overline{X} \ominus \phi)) \oplus r \\ &= (X \cap W) \oplus R \\ &= X \oplus R. \end{aligned}$$

Erosion, morphological low pass and high pass filters are useful image operations (Chapter 4). We list their symbolic substitution rules here as examples for deriving symbolic substitution algorithms based on BIA:

- Erosion can be implemented by a symbolic substitution rule as

$$X \ominus R = (X \circledast R_e) \oplus I \tag{5.20}$$

where the search-pattern is $R_e = (R, \phi)$ and the replacement is I.

$$\begin{aligned} Proof: \ (X \circledast R_e) \oplus I &= (X \ominus R) \cap (\overline{X} \ominus \phi) \\ &= (X \ominus R) \cap W \\ &= X \ominus R. \end{aligned}$$

- Opening (one kind of morphological low pass filter) can be implemented by a symbolic substitution rule as

$$X \circ R = (X \circledast R_e) \oplus R \tag{5.21}$$

where the search-pattern is $R_e = (R, \phi)$ as used in the erosion and the replacement pattern is R.

$$\begin{aligned} \mathit{Proof}: \ (X \circledast R_e) \oplus R &= ((X \ominus R) \cap (\overline{X} \ominus \phi)) \oplus R \\ &= ((X \ominus R) \cap W) \oplus R \\ &= (X \ominus R) \oplus R \\ &= X \circ R. \end{aligned}$$

- Morphological high pass filter can be implemented by three symbolic substitution rules as

$$X/(X \circ R) = [([(X \circledast R_c) \oplus I] \cup [(X \circledast R_e) \oplus R]) \circledast R_c] \oplus I \tag{5.22}$$

where $R_c = (\phi, I)$ as used in the complement and $R_e = (R, \phi)$ as used in the erosion.

$$\begin{aligned} \mathit{Proof}: \quad & [([(X \circledast R_c) \oplus I] \cup [(X \circledast R_e) \oplus R]) \circledast R_c] \oplus I \\ = \ & [(\overline{X} \cup (X \circ R)) \circledast R_c] \oplus I \\ = \ & \overline{\overline{X} \cup (X \circ R)} \\ = \ & X/(X \circ R). \end{aligned}$$

Thus, BIA offers a systematic mathematic structure for developing parallel algorithms for symbolic substitution processors. In general, BIA also provides a universal description of algorithms on parallel single instruction multiple data (SIMD) machines. Its description of an algorithm on a given architecture simplifies comparison of the number of operations required.

5.2 Relationship to Cellular Logic

Cellular logic architectures have been briefly reviewed in Section 2.2. A cellular logic operation transforms an array of data into a new array of data where each element in the new array has a value determined only by the corresponding element in the original array along with the values of its neighbors (Fig. 2.1). In BIA, an image transformation can be written as a polynominal of reference images (Theorem 3.1) where the reference images can have arbitrary large size. In terms of cellular logic, the reference image essentially defines the neighborhood of a cell where the neighborhood can be very large and not just nearest 4- or 8-neighborhood as in conventional cellular logic. Thus, cellular logic operations are also particular cases of image transformations with small local reference images, and BIA also serves as a systematic mathematical tool for formalizing cellular logic.

Because of existing hardware interconnection limitations, it is difficult and costly to implement an image transformation with a large reference image in one clock cycle. In addition, the conventional nearest-neighbor connected cellular arrays have poor communication capabilities. To improve this, we develop the DOCIP-hypercube architecture in Chapter 6, which combines features of conventional nearest-neighbor connected cellular logic architectures and conventional hypercube architectures for implementing BIA effectively.

In summary, BIA provides a systematic mathematical formalism for both symbolic substitution and cellular logic. The applications of symbolic substitution and cellular logic can be accomplished by BIA; on the other hand, generalized cellular logic architectures are good candidates for implementing BIA (Chapter 6 and Chapter 7).

5.3 Relationship to Boolean Logic

BIA can implement any boolean logic operation on binary images. It is also obvious that BIA fundamental operations can be implemented by a boolean logic gate array with interconnections. The following straight-forward correspondence can be drawn between the BIA operations and boolean logic operations:

BIA Operations	*Boolean Logic Operations*
1. Complement	NOT
2. Union	OR
3. Dilation	Multiple-input OR
4. Intersection	AND
5. Erosion	Multiple-input AND
6. Symmetric Difference	EXCLUSIVE-OR

Note that the inputs of OR and AND (corresponding to union and intersection) come from two different images. The multiple inputs of OR and AND (corresponding to dilation and union) come from the same image while the other operand image R only determines the number and location of input pixel values. A complete logical set is able to implement any boolean logic function; it consists of at least one of the following sets: NOT and OR; NOT and AND; NAND;

NOR. In BIA, in order to implement any image transformation, we need a complete system of pixel-wise logic operations and we also need a translational type of operation (such as translation, dilation, erosion, convolution and correlation etc.) to allow the global information extraction in an image or the information exchange between pixels of the same images. In order to have a 2-D compact parallel form of image processing algorithms whose variables are whole images, we define the parallel form of those corresponding boolean logic operations as BIA operations. In fact, there are two boolean algebras, $(P(W); \cup, \cap, ^{-}, \phi, W)$ and $(P(W); \triangle, \cap, ^{-}, \phi, W)$, supported by BIA also (Section 5.5). We can define several possible sets of fundamental operations for implementing any image transformation, such as a parallel form of NOR (or NAND or (NOT and OR) or (NOT and AND)) and a translational-type operation (e.g. translation, dilation, erosion, convolution, and correlation etc). The reasons that we choose complement, union and dilation as the three fundamental operations are:

- Nice mathematical properties: The dilation is commutative, associative, and distributive over the union; the erosion has no such properties.
- Simple hardware implementation: These three operations are easily implemented by a 2-D gate array and 3-D interconnection techniques.
- Simple software design: These three operations are inherently parallel and frequently used operations. Algorithms can be written as compact formulas which easily become very efficient fast parallel algorithms by simply applying the fundamental operations and removing the data dependencies.

Comparing BIA with the conventional boolean expressions for logic functions, the major advantages of BIA are summarized in the following:

- BIA operations are inherently parallel, but boolean logic operations are serial.
- BIA operations include parallel information transferring capabilities which are missing in boolean logic operations.
- Algorithms in BIA are written as compact algebraic formulas whose variables are whole images, while a typical image processing algorithm is very difficult to write in a compact precise boolean logic expression.
- BIA has pictorial physical meaning, while boolean expressions provide little physical feeling for parallel image processing algorithms.

5.4 Relationship to Linear Shift Invariant Systems, Convolution, and Correlation

It is well known that the theory of linear shift-invariant (LSI) systems plays a key role in conventional signal (including image) and system analysis [Oppenheim75, Pratt78]. It is very natural that we like to ask what the relation between BIA and LSI system theory is. A system is defined as a transformation or mapping from a set of input functions into a set of output functions, and a two dimensional discrete LSI system is defined as a system which obeys two properties:

- Linearity: $T[ax(i,j)+bz(i,j)] = aT[x(i,j)]+bT[z(i,j)]$ for arbitrary constants a and b;
- Shift-invariance: $y(i,j)=T[x(i,j)] \rightarrow y(i-k,j-l)=T[x(i-k,j-l)]$.

A linear system can be completely characterized by its unit-impulse response $r(i,j;k,l) = T[\delta(i-k, j-l)]$. In an LSI system the unit-impulse response is simply $r(i,j;k,l) = r(i-k, j-l)$, and the output of an LSI system with input $x(i,j)$ and unit-impulse response $r(i,j)$ is the convolution of $x(i,j)$ and $r(i,j)$, denoted by

$$x(i,j) * r(i,j) = \sum_{k,l=-\infty}^{\infty} x(k,l)r(i-k, j-l). \tag{5.23}$$

Now, let us consider only binary images. In terms of the set notation, an image $X = \{(i,j) \mid x(i,j) = 1\}$ corresponds to function $x(i,j)$. If we assume $r(i,j) = 1$ at and only at n points which correspond to an image R with n image points, then the convolution of $x(i,j)$ and $r(i,j)$ with a threshold $t = 0$ is

$$\begin{aligned} X * R \mid_{t=0} &= \{(i,j) \mid \textstyle\sum_{k,l} x(k,l)r(i-k, j-l) > 0\} \\ &= \{(i+k, j+l) \mid \textstyle\sum_{k,l} x(k,l)r(i,j) > 0\} \\ &= \{(i+k, j+l) \mid x(k,l)r(i,j) > 0\} \\ &= \{(i+k, j+l) \mid (i,j) \in X, (k,l) \in R\} \\ &= X \oplus R \end{aligned} \tag{5.24}$$

where the ouput of the threshold is defined as 1 if $x(i,j) * r(i,j) > 0$, and is 0 otherwise; and the universal image, as before, contains all image points (i,j), (k,l), and $(i+k, j+l)$. This means that the dilation $X \oplus R$ is the same as adding a threshold $t = 0$ to the convolution sum. The reference image plays a role similar to that of the unit impluse response in the binary image system. Similarly the erosion $X \ominus R$ is the same as the convolution $x(i,j) * r(-i,-j)$ followed by the threshold $t = n - 1$.

Correlators have been used in pattern recognition for a long time [Goodman68]. Correlation is strongly related to convolution: convolution involves folding, shifting and summing; correlation involves shifting and summing without folding. Therefore,

$$X \oplus R = X * R \mid_{t=0} = X \diamond \check{R} \mid_{t=0} \tag{5.25}$$

$$X \ominus R = X * \check{R} \mid_{t=n-1} = X \diamond R \mid_{t=n-1} \tag{5.26}$$

where $*$ means convolution, $\diamond$ means correlation, and $\check{R}$ means the reflected image of R.

Furthermore, although the three fundamental operations of BIA are nonlinear, with appropriate number representations they are able to implement parallel numerical and linear operations also. Also, BIA can implement both shift invariant and shift variant image transformations.

5.5 Some Standard Algebraic Structures

Some algebraic structures supported by BIA are in the following:

1. $(P(W); \oplus)$ is a semigroup.
2. $(P(W); \oplus, I)$ is a monoid.
3. $(P(W); \triangle, \phi, \triangle)$ is an abelian group.
4. $(P(W); \cup, \cap, ^{-}, \phi, W)$ and $(P(W); \triangle, \cap, ^{-}, \phi, W)$ are Boolean algebras.
5. $(P(W); \subset)$ is a poset (partially ordered set).
6. $(P(W); \cup, \cap, \subset)$ is a complete lattice.

Proof: (1) A semigroup is a set with an associative binary operation [Birkhoff70, Birkhoff65, Gilbert76]. By Theorem 3.1, the dilation $\oplus$ is associative for all images in $P(W)$.

(2) A monoid is a semigroup with an identity [Birkhoff70, Birkhoff65, Gilbert76]. By Appendix B, the dilation has an identity $I = \{(0,0)\}$. Note that $(P(W); \ominus)$ is neither a semigroup nor a monoid.

(3) A group is a monoid in which every element has an inverse. An abelian group is a group in which the operation is commutative [Birkhoff70, Birkhoff65, Gilbert76]. By the definition of symmetric difference (mod 2 image addition), it can be easily verified that its identity is ϕ and its inverse operation (mod 2 image subtraction) is itself.

(4) A boolean algebra is a set with operations $\vee, \wedge, ^{-}, 0$ and 1 satisfying: 1. $a \vee b = b \vee a$, $a \wedge b = b \wedge a$ (commutativity); 2. $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$, $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$ (associativity); 3. $a \vee 0 = a$ (universal bound); 4. $a \wedge 1 = a$ (universal bound); 5, $a \vee \overline{a} = 1$, $a \wedge \overline{a} = 0$ (complementarity) [Birkhoff70, Birkhoff65, Gilbert76]. By Appendix B, $(P(W); \cup, \cap, ^{-}, \phi, W)$ and $(P(W); \triangle, \cap, ^{-}, \phi, W)$ are Boolean algebras.

(5) A poset is a set with a relation satisfying: 1. the reflexivity; 2. the antisymmetry; and 3. the transitivity [Birkhoff70, Birkhoff65, Gilbert76]. The relation $\subset$ satisfies these three conditions: 1. $X \subset X$ for all $X \in P(W)$; 2. if $X \subset R$ and $R \subset X$, then $X = R$; and 3. if $X \subset R$ and $R \subset Q$, then $X \subset Q$.

(6) A complete lattice is a poset $(S; \leq)$ in which every subset of S has a *sup* (the least upper bound) and an *inf* (the greatest lower bound) [Birkhoff70, Birkhoff65, Gilbert76]. In the algebra $(P(W); \cup, \cap, \subset)$, given any subset of $P(W)$, say $\{X(\theta) \mid \theta \in \Theta\}$ (Θ is the index set of the elements in this subset of $P(W)$),

we have

$$sup = \bigcup_{\theta \in \Theta} X(\theta) \tag{5.27}$$

$$inf = \bigcap_{\theta \in \Theta} X(\theta). \tag{5.28}$$

Thus, several standard algebraic structures and their properties can be directly implemented and used in BIA.

Chapter 6

Digital Optical Cellular Image Processor (DOCIP): Architectures

This chapter describes the digital optical cellular image processor (DOCIP) architectures:

- The DOCIP-array, a cellular array processor, which uses optical parallelism to map an inherently 2-D parallel data structure to a 2-D nearest-neighbor connected cellular computer in a simple and direct way; its performance is primarily limited by its $O(1)$ interconnectivity, and
- The DOCIP-hypercube, a two-dimensional cellular hypercube processor, which uses optical parallelism and 3-D global interconnection capabilities to implement a hypercube interconnection.

Here, the term "DOCIP" will be used to refer to both of the DOCIP-array and the DOCIP-hypercube. The major characteristic of the DOCIP is to match parallel image processing tasks by using inherent optical parallelism and 3-D free interconnection capabilities. The complexity of cell structure is reduced and the conventional communication bottleneck problem is released. Furthermore, the DOCIP-hypercube offers further improvements in speed and flexibility for global operations. We will first discuss the design principles, then the array of cells, the interconnection networks, and finally optical conceptual implementations.

6.1 Design Principles

The philosophy behind the DOCIP is to provide very flexible and fast 3-D parallel optical processing to efficiently implement BIA algorithms in a simple and natural way. In general, to construct an image processor one should address the problems listed below.

1. Choice of algorithms
   - Solving problems in image processing is primarily to search for algorithms, and then implement them. Here the algorithms supported by BIA are our algorithmic choice for the DOCIPs, because of their simplicity and parallelism.
2. Choice of a language
   - Language is used for conceiving and writing algorithms and for communication between people and machines. It is a window between simplicity and complexity, and should be able to represent complex

algorithms in a simple, clear, and clean way. The algebraic language comprising those algebraic expressions in BIA is a good candidate for the language of the DOCIPs. This algebraic language makes a complex parallel algorithm appear as a compact formula where a symbol represents a whole image or a parallel operator, and has no requirement of knowing English.

3. Choice of architecture

   - Image processing and analysis tasks have the following properties:
     - Large data processing requirement,
     - Mostly local parallel operations,
     - Momogeneous processing of all pixels, and
     - Inherent two-dimensional parallelism of images.
   - The fundamental characteristics (such as discrete space, discrete time, finite state variables, local transition function, parallel bits, homogeneous space and time) of cellular automata [Wolfram86, Preston84] correspond to the above properties of digital image processing in a direct way. Thus, two dimensional cellular automata are appropriate models of image processors, and two dimensional cellular logic architectures are good candidates for our choice of architecture.
   - The neighborhood configuration in a cellular imagee processor usually defines the interconnection network and strongly depends on the type of tessellation (digitization pattern) of those images allowed to be processed. To be able to completely cover an area, there exist three possible regular forms — the equilateral triangle, the square and the

hexagon. Their relative merits can be found in [Rosenfeld70, Gray71, Golay69, Deutsch72, McCormick65, Duff73]. Considering pattern connectivity properties, the hexagonal pattern has only one type of nearest connected neighborhood and is favored in this sense. However, the hexagonal tessellation has its own drawbacks:

  - The orthogonal coordinates of the array points with the hexagonal tessellation cannot be expressed as integer multiples of the length unit.
  - The natural shifting directions of the hexagonal array do not coincide with the coordinate axes.
  - For practical applications, it is inconvenient and unconventional to digitize an image by the hexagonal tessellation.

Hence, for the reason of simple practice and direct mapping BIA algorithms, the square tesseliation of image and the square array of cells is used in the DOCIP.

4. Choice of hardware implementation

   - The main difficultiess in implementing an image processing machine are in the following:
     - Solving the image parallel input/output problems,
     - Effectively implementing parallel operations,
     - Effectively handling huge amounts of homogeneous image data with inherently two-dimensional structure,
     - Effectively extracting global information in parallel for making decisions fast,

  - Efficiently reducing the hardware complexity, and
  - Fully utilizing avaliable parallelism.
- Ideally, the optical implementation of cellular logic architectures, especially the optical two-dimensional cellular hypercube architectures, will solve the above problems naturally, because it combines two desirable features of the universe:
  - Fully using the natural 3-D space: two spatial dimensions for storing the inherently two-dimensional image data and for implementing the nonlinear logic units (planar gate-array), and the third dimension used for interconnection; and
  - Using the fastest speed (propagation speed of light) and the lightest (no mass, no charge, and infinite life time) particles (photons) to propagate and to carry information.

Overall, the design principles of the DOCIP architectures are:

- Parallelism should be fully explored by using parallel algorithm, parallel language, parallel architecture, and parallel hardware implementation;
- Each cell, responsible for a pixel, need not be "general-purpose". On the contrary, each cell should be made as simple as possible to keep a low hardware complexity.

6.2 General Organization of the DOCIPs

Based on the above design principles, we want to implement a simple cellular processor with the ability to implement any image transformation. We first answer this algebraically, then give an algebraic description and a general description of the DOCIP design.

6.2.1 Algebraic Description

To map BIA into the DOCIP architecture in a transparent way, we first define the DOCIP algebraically:

> *Definition of Cellular Automata*
>
> A cellular automaton is an algebra $(S; F, N_c)$ where S is the state space which is a set of states, F is a family of transition functions, and N_c is the neighborhood configuration.
>
> *Constraints on a cellular automaton for Implementing BIA:*
>
> 1. $S \supset P(W)$
> 2. $F \supset \{\oplus, \cup, ^{-}\}$
> 3. $N_c \supset I \cup A \cup A^{-1} \cup B \cup B^{-1}$ or $N_c \supset A \cup A^{-1} \cup B \cup B^{-1}$
>
> where "$\supset$" means "contains", the definitions of the image space, three fundamental operations and five elementary images are referred to Chapter 3.

Thus, in terms of cellular automata, the DOCIPs have to satisfy the above constraints for realizing BIA. For storing input images and temporary results in a more flexible way, the DOCIPs utilize three memory modules and all share the same algebraic structure (except the neighborhood configuration):

$$DOCIP = (P(W \times W \times W); \oplus, \cup, ^{-}, N_c) \tag{6.1}$$

where "$\times$" denotes cross product and N_c can be one of the following 4 types:

1. DOCIP-array4: each cell connects with its four nearest neighbors and itself, i.e.

$$N_{array4} = I \cup A \cup A^{-1} \cup B \cup B^{-1}. \tag{6.2}$$

2. DOCIP-array8: each cell connects with its eight nearest neighbors and itself, i.e.

$$N_{array8} = \bigcup_{i,j=-1}^{1} A^i B^j. \tag{6.3}$$

3. DOCIP-hypercube4: each cell connects with itself and those cells in the 4 directions at distances $1, 2, 4, 8, ..., 2^k$ from itself, i.e.

$$N_{hypercube4} = \bigcup_{i=o,\pm 1,\pm 2,\ldots,\pm 2^k} (A^i \cup B^i) \tag{6.4}$$

   where k is sufficiently large for the connections to traverse the entire array of cells.

4. DOCIP-hypercube8: each cell connects with itself and those cells in the 8 directions at distances $1, 2, 4, 8, ..., 2^k$ from itself, i.e.

$$N_{hypercube8} = \bigcup_{i=o,\pm 1,\pm 2,\ldots,\pm 2^k} (A^i \cup B^i \cup A^i B^i \cup A^i B^{-i}). \tag{6.5}$$

The reason for choosing these specific structures is: 1) the DOCIP-array has extremely simple organization and low hardware complexity for implementing BIA; and 2) the DOCIP-hypercube will improve the computation time for many global operations from $O(N)$ of the DOCIP-array with $N \times N$ cells to only $O(log_2 N)$ while reserving a reasonable number of interconnections and simple optical hardware requirement.

6.2.2 General Description

From the above algebraic definitions, the DOCIPs have the same algebraic structure except the neighborhood configuration N_c. Thus, they share the same architecture as shown in Fig. 6.1 except the allowed configuration of the reference images E_i at a cycle which defines the neighborhood configuration and the corresponding interconnection network.

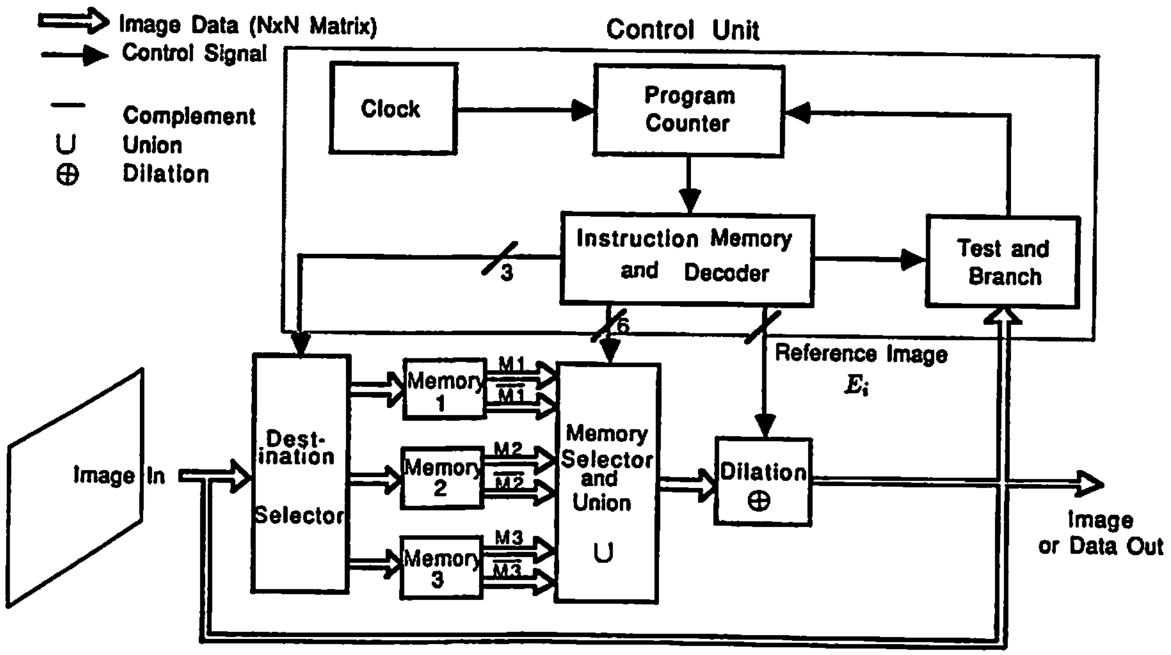


Figure 6.1: A digital optical cellular image processor (DOCIP) architecture — one implementation of binary image algebra (BIA). The DOCIP-array requires 9 (or 5) control bits for reference image E_i. The DOCIP-hypercube requires $O(logN)$ control bits for reference image E_i.

Basically, the proposed DOCIP as shown in Fig. 6.1 is a cellular SIMD machine and consists of an array of cells or processing elements (PEs) under the supervision of a control unit. The control unit includes a clock, a program counter, a test and branch module for feedback control, and an instruction decoder for storing instructions and decoding them to supervise cells. The array of cells includes a destination selector, three memory elements for storing images, a memory selector, and a dilation unit.

The DOCIP machine operates as follows: (1) a binary image ($N \times N$ matrix) is selected by the destination selector and then stored in any memory as the instruction specifies; (2) after storing the images (1 to 3 $N \times N$ matrices), these images and their complemented versions are piped into the next stage, which forms the union of any combination of images; (3) the result is sent to a dilation where the reference image specified by the instruction is used to control the type of dilation; (4) finally, the dilated image can be output, tested for program control, or fed back to step (1) by the address field of the instruction.

6.3 Array of Cells

6.3.1 Implementation of Fundamental Operations

The most complicated fundamental operation in BIA is the dilation operation $\oplus$. The interesting fact is that dilation $\oplus$ operation in general is a typical sequential cellular logic operation. The cellular logic operations can be implemented in the following ways: (1) the neighborhood table look-up (or template matching) approach. (2) the processor array and (3) the run-list processing approach [Verbeek84]. The table look-up approach is most widely used in cellular logic computers; the reason may come from its flexibility and conceptual simplicity.

The most straight forward implementation of a truth table may be achieved by the storage of the entire truth table in a direct, or location-addressable memory (LAM). These systems require a memory size (in bits) of

$$S = 2^p q$$

where p is the number of input bits and q is the number of output bits.

However, to set up a large number of different cellular logic operations, it requires large memory size. For example, we have $(2^{(2n+1)\times(2n+1)})^{(2^{(2n+1)\times(2n+1)})}$ image transformations and each transformation requires $2^{(2n+1)\times(2n+1)}$ bits. Thus, to implement any image transformation by only the take look-up approach it, we will require a huge memory size (in bits) of

$$S = (2^{(2n+1)\times(2n+1)})^{(2^{(2n+1)\times(2n+1)})} \times 2^{(2n+1)\times(2n+1)}.$$

The run-list processing approaches were original designed as software methods for use in general purpose computer. Here, we propose the implementation of 3 fundamental operations by the gate array and 3-D interconnection approach. This approach is similar as the processor array approach. Its major requirement is that the interconnection mechanism has to assure rapid access to the required image memory from the neighboring (or the extended cellular hypercube neighboring) pixels. A digital optical system offers parallel and global accessing ability, and, it is one of the best candidates to implement cellular logic operations in this approach.

Implementation of Complement Operation

We can use an array of memory with inverting and noninverting outputs, or use an array of inverter gates. In this approach, we can always let the boundary of the binary image X be white (0 or transparent); and let the universal binary image W include X in a plane of pixels. The complement of a binary image then can be considered as the output of a full inverter gate array which places an inverter behind every pixel.

Implementation of Union (or Intersection) Operation

The union (or intersection) of two binary images A and B is the same as the output of a full 2-input OR (or AND) gate array which places a 2-input OR (or AND) gate for every corresponding two pixels, one pixel from X and the other from R. Further more, the action of superimposing two optical signals into a gate is equivalent to the OR operation of these two optical signals into the gate. Thus, by appropriate arrangement of gates, we may get the union operation for free.

Implementation of Dilation (or Erosion) Operation

The dilation (or erosion) of the binary image X by a reference image R which has n foregfround image points is the same as the output of a full n-input OR (or AND) gate array which places an n-input OR (or AND) gate behind each pixel; and those n inputs for each gate come from n pixels in X corresponding to the reference image R.

One important feature of parallel binary image processing is that there are many possible complex image transformations which conceptually employ large or intricate reference images. Thus, the most difficult problems are: 1) how to implement the desired image transformation quickly and economically; and 2) how to supply the required arbitrary large reference image in a rapid way. Must we utilize many complicated hardware devices to implement complex image transformations? Do we really need a large number of memory bits to store a lot of large reference images? It is not necessary; two fundamental principles in BIA suggest an economic approach. The DOCIP array of cells with an inter-cell commnuication interconnection network will implement these three BIA fundamental operations in a pipeline and has the ability to realize any image transformation and to generate any image.

6.3.2 Cell Structure

Since each cell is identical, the cell structure actually defines the whole array of cells. Each cell of the DOCIPs is a synchronous sequential logic circuit that takes its input from the neighboring cells (nearest neighbors or extended hypercube neighbors) and delivers its output to the same cell.

To be mathematically precise, a cell of the DOCIP is a finite state machine and can be described as $M = (Q, I, T)$ where

- Q is a finite set of states,
- I is a finite set of inputs, and
- T is a finite set of state transition functions mapping from $Q \times I$ into Q.

Here, we emphasize the analysis of the state transition function. Detailed discussions of finite state machines can be found in [Hartmanis70, Minsky67]. In our case, the cell structure is shown in Fig. 6.2, and the descriptions of this state machine (Q, I, T) are given below.

- A state of the cell of the DOCIP is

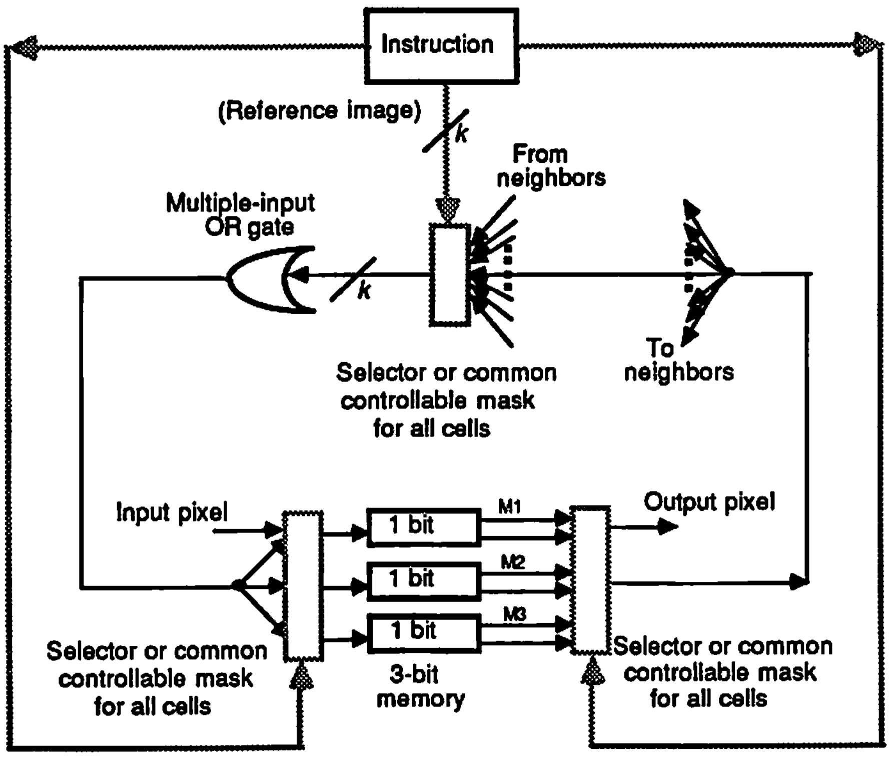

Instruction
(Reference image)
k
From neighbors
Multiple-input OR gate
k
Selector or common controllable mask for all cells
To neighbors
Input pixel
1 bit
1 bit
1 bit
M1
M2
M3
Output pixel
Selector or common controllable mask for all cells
3-bit memory
Selector or common controllable mask for all cells

Figure 6.2: The cell structure of the DOCIP machine.

$$q = (m_1, m_2, m_3) \in Q \tag{6.6}$$

where m_1, m_2 and m_3 are three binary memory bits;

- An input of the cell of the DOCIPs is

$$i = (x, c, d_1, d_2, d_3, s_1, s_2, ...s_6, n_1, n_2, ..., n_k, r_2, ..., r_k) \in I \tag{6.7}$$

where i consists of $2k+10$ binary bits, $r_j, n_j, j = 2, ..., k$, are the values from its neighbors (corresponding to a reference image of the dilation operation) and their control bits; n_1 controls the input from its previous own state. The DOCIP-array4 has the value $k = 5$, the DOCIP-array8 has the value $k = 9$, the DOCIP-hypercube4 has the value of $k = 4log((N+1)/2)) + 1$, and the DOCIP-hypercube8 has the value of $k = 8log((N+1)/2)+1$ for an $N \times N$ array, where $(N+1)/2$ is a power of 2. x is the input pixel from an input image X; $c, s_1, s_2, ..., s_6, d_1, d_2$, and d_3 are common control bits used for all cells.

- The state transition function $f \in T$ is

$$\begin{aligned} q(t+1) &= f(q(t), i(t)) \\ &= (n_1(t)(x(t) + c(t)(s_1(t)m_1(t) + s_2(t)m_2(t) + s_3(t)m_3(t) + \\ &\quad s_4(t)\overline{m_1}(t) + s_5(t)\overline{m_2}(t) + s_6(t)\overline{m_3}(t))) \\ &\quad + n_2(t)r_2(t) + ... + n_k(t)r_k(t)) \cdot (d_1(t), d_2(t), d_3(t)) \\ &= (m_1(t+1), m_2(t+1), m_3(t+1)) \end{aligned} \tag{6.8}$$

where $+$ is an OR operation, $\cdot$ is an AND operation, and $\overline{m_1}$ is a NOT operation of m_1. A cell circuit diagram with 3-input NOR gates for the experimental DOCIP system can be found in Chapter 9 (Fig. 9.5).

6.4 Interconnection Networks

Similar to other SIMD array processors [Hwang84], the topological structure of the DOCIP machines are mainly characterized by their interconnection networks used in interconnecting the cells. Formally, an inter-cell communication network can be specified by a set of data routing functions. If we index all the cells by a set S, each routing function f is a bijection (a one-to-one and onto mapping) from S to S. Algebraic definitions of the neighborhood configurations of the DOCIPs, which also define their interconnection networks, have been given in Section 6.2.1. Here we will further discuss their properties and characteristics.

6.4.1 Cellular Array, Conventional Hypercube, and Cellular Hypercube

The DOCIP-array uses the conventional nearest-neighbor connected cellular array (Fig. 2.3) to process 2-D image data in parallel and to easily realize its simple interconnection. However, conventional nearest-neighbor connected cellular arrays have poor communication capabilities; their performance is primarily limited by their $O(1)$ interconnectivity. To improve this while preserving a reasonable number of interconnections, ideally a conventional hypercube increases the interconnectivity to $O(log_2 M)$ for M processing elements (PEs). (We refer to a PE in a cellular computer as a cell which usually has no address and index registers.) A conventional SIMD hypercube computer is comprised of $M = 2^l$ PEs, where l is a non-negative integer. All the PEs are synchronized and operated under the control of a single instruction stream. They are indexed 0 through $M-1$ and the p^{th} PE is referred to as $PE(p)$ for $p \in [0, M-1]$. A hypercube is denoted as an l-cube where $l = log_2 M$ represents the number of directly connected PEs. Let $p_{l-1}p_{l-2}...p_0$ be the binary representation of p, and let $p^{(b)}$ be the number whose binary representation is $p_{l-1}...p_{b+1}\overline{p_b}p_{b-1}...p_0$, where $\overline{p_b}$ is the complement of p_b and $0 \leq b < l$. In the hypercube model, $PE(p)$ is connected to those $PE(p^{(b)})$ for $0 \leq b < l$ (i.e. a direct connection exists only between processors whose binary indices differ by one bit position), and data can be transmitted from one PE to another in one step only via this interconnection pattern [Hwang84]. The worst case for an inter-PE communication requires $log_2 M$ routes.

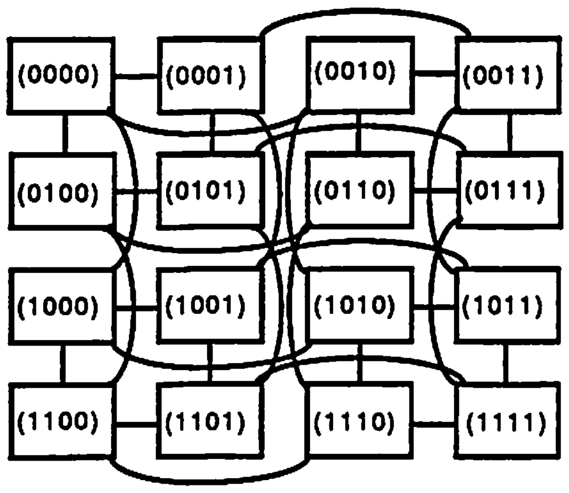


Figure 6.3: A conventional hypercube (4-cube) laid out in two dimensional space. Its interconnections have no spatial invariance.

However, when a conventional hypercube is laid out in two-dimensional space (e.g. Fig. 6.3 gives a 4-cube), its interconnection patterns are not space invariant; such spatial invariance is desirable for image processing and for simple hardware implementation. To include this, we increase the interconnections to make a two dimensional cellular hypercube (Fig. 6.4).

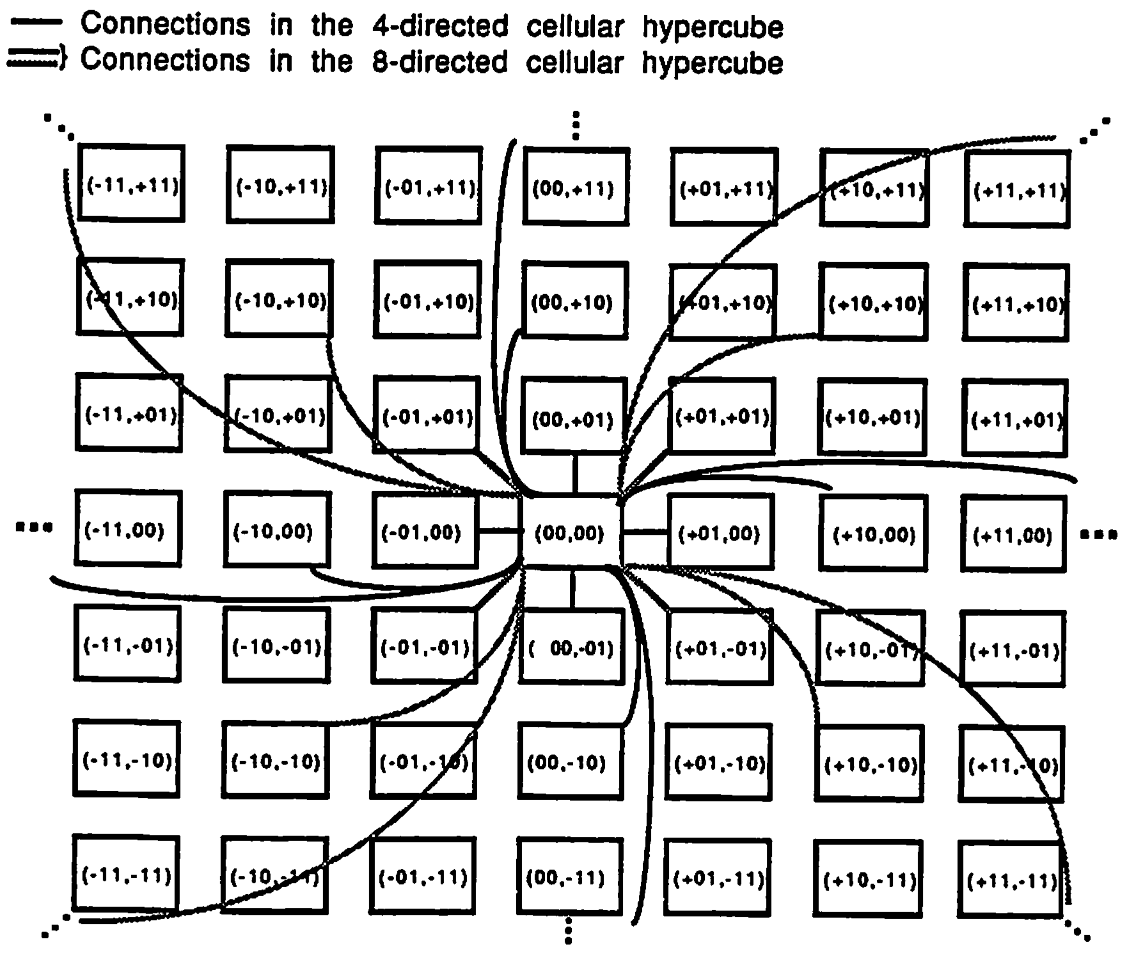


Figure 6.4: A two-dimensional cellular hypercube — DOCIP-hypercube. Each cell is interconnected with other cells having a *relative* one bit difference in coordinate label in positive or negative x and y directions to achieve a spatially symmetric and invariant interconnection pattern. Only connections from the central cell are shown; all cells are connected identically so the resulting interconnections are space invariant.

The cellular hypercube introduces a symmetrical positive and negative index so that each cell is connected with cells having a *relative* one bit difference in coordinate label in positive or negative x and y directions; the numerical difference of addresses of connected cells is nonzero in at most 1 bit. A two-dimensional SIMD cellular hypercube computer consists of $M = N^2 = (2n-1)^2$ cells and $n = 2^k$, k is a non-negative integer. They are indexed $(-n+1, -n+1)$ through $(n-1, n-1)$ and the $(q,r)^{th}$ cell is refered as $CELL(q,r)$ for $q, r \in [-n+1, n-1]$. In the 4-directed cellular hypercube (cellular hypercube4) model, $CELL(q,r)$ is connected to those $CELL(q \pm 2^d, r)$ and $CELL(q, r \pm 2^d)$ for $0 \leq d < k$; and in the 8-directed cellular hypercube (cellular hypercube8) model, $CELL(q,r)$ is connected to those $CELL(q \pm 2^d, r)$, $CELL(q, r \pm 2^d)$ and $CELL(q \pm 2^d, r \pm 2^d)$ for $0 \leq d < k$. Data can be transmitted from one cell to another in one step only via this interconnection pattern, although it occurs in parallel for each pixel. For $N^2 = (2n-1)^2$ cells, the worst case for an inter-cell communication requires $2log_2n$ or $4log_2n$ (they are $O(log_2N)$) routes for the 8-directed or 4-directed cellular hypercube repectively.

Both the conventional hypercube and cellular hypercube require a 3-D global interconnection mechanism which is difficult to implement on a planar VLSI chip [Rosenfeld83, Sawchuk85, Jenkins85]; they pay a large penalty in increasing chip area from $O(N^2)$ of celluar array to $O(N^4)$ [Ullman84]. However, in principle, the 3-D interconnection mechanism is realizable by digital optical systems: the conventional hypercube requires the interconnection hologram space-bandwidth product of $O(N^2 log_2 N)$ while the cellular hypercube still preserves a low hologram complexity $O(N^2)$. (We use a hybrid interconnection system ([Jenkins84b, Sawchuk84]) as a model for measuring the complexity of optical holographic interconnections.) Table 6.1 gives a comparision between these three different interconnection networks: cellular array, conventional hypercube and cellular hupercube. Thus, the cellular array has the simplest hardware requirement and the cellular hypercube has the overall desired features.

Interconnection Networks	Cellular Array	Conventional Hypercube	Cellular Hypercube
Connectivity (Interconnections per PE)	$O(1)$	$O(log_2 N)$	$O(log_2 N)$
2-D Spatial Invariance	Yes	No	Yes
Inter-PE Communication Time Complexity	$O(N)$	$O(log_2 N)$	$O(log_2 N)$
VLSI Chip Area	$O(N^2)$	$O(N^4)$	$O(N^4)$
Hologram Space-bandwidth Product	$O(N^2)$	$O(N^2 log_2 N)$	$O(N^2)$

Table 6.1: A comparison between three different interconnection networks of $N \times N$ processing elements (PEs): cellular array, conventional hypercube and cellular hypercube.

6.4.2 Characteristics of DOCIP Interconnection Networks

To directly match the universal image W of BIA into the array of cells and to simplify the cell structure, the DOCIP interconnection networks, the cellular array and cellular hypercube, requires some special characteristics which usually do not exist in the general interconnection networks of parallel processors. These characteristics are summarized below.

1. **Symmetric index of cells**

   The two dimensional array of cells range from $(-n,-n)$ up to the index of (n,n) (Fig. 6.4). For an $N \times N$ array, N is then equal to $2n+1$. The purpose is to preserve the symmetry of the array space, i.e. the symmetry of the universal image W. Thus, if an image is accepted by the DOCIPs, then its reflected image is guaranteed to be accepted too.

2. **No address and index register in a cell**

   In fact, the natural spatial positions of cells have distinguished the cells. Furthermore, the output of a cell at a cycle is determined by only the values of its neighbors corresponding to the neighborhood configuration N_c and all cells have the same behavior through the whole space. Thus, to reduce the hardware requirement of a cell, we discard the address registers. This means that each cell does not record the addresses of those cells which send the information to it, does not record the addresses of those cells which will be sent message from it, and does not record the address of itself. On the other hand, the control signals of a reference image at a cycle and the spatial position of a cell have replaced the function of those address registers in a natural way.

3. **Two-dimensional spatial symmetry and spatial invariance**

   Each cell only interconnects with its neighbors(or extended hypercube neighbors). The neighborhood configuration is spatially symmetric and the interconnection pattern is spatially invariant.

To be mathematically precise, the interconnection network of a DOCIP with a given neighborhood configuration

$$N = \{(i_1, j_1), (i_2, j_2), ..., (i_k, j_k)\} \tag{6.9}$$

is characterized by the following data routing functions:

$$R(i,j) = (i,j) + (i_l, j_l) \tag{6.10}$$

where $l = 1, 2, ..., k$. This means that a cell at (i,j) is connected with cells at $(i+i_1, j+j_1), (i+i_2, j+j_2), ..., (i+i_k, j+j_k)$. If some neighboring cells do not exist, then the corresponding signals are connected to free space. Thus, the interconnection networks of the DOCIPs are also completely defined by the neighborhood configuration N_c.

6.5 Optical Conceptual Implementation

The DOCIP entire system can be realized by an optical gate array with optical 3-D interconnections as we discussed in Section 2.3; Figures 6.5 and 6.6 show the conceptual optical implementation for the DOCIP-array and the DOCIP-hypercube respectively. It embeds an array of cells in an array of optical binary gates and performs interconnections of these gates by an optical hologram.

It should be noted that current optical technology has implemented only arrays of moderately large numbers of gates (500 × 500) at very slow (~ms) switching speeds, and alternatively, arrays of small numbers of gates (2 × 2 to 6 × 6) at fast switching speeds (0.1μs - 2ps) (Section 2.3). Current ongoing research in a number of laboratories looks promising in eventually providing the needed arrays of large numbers of gates with reasonably fast switching speeds.

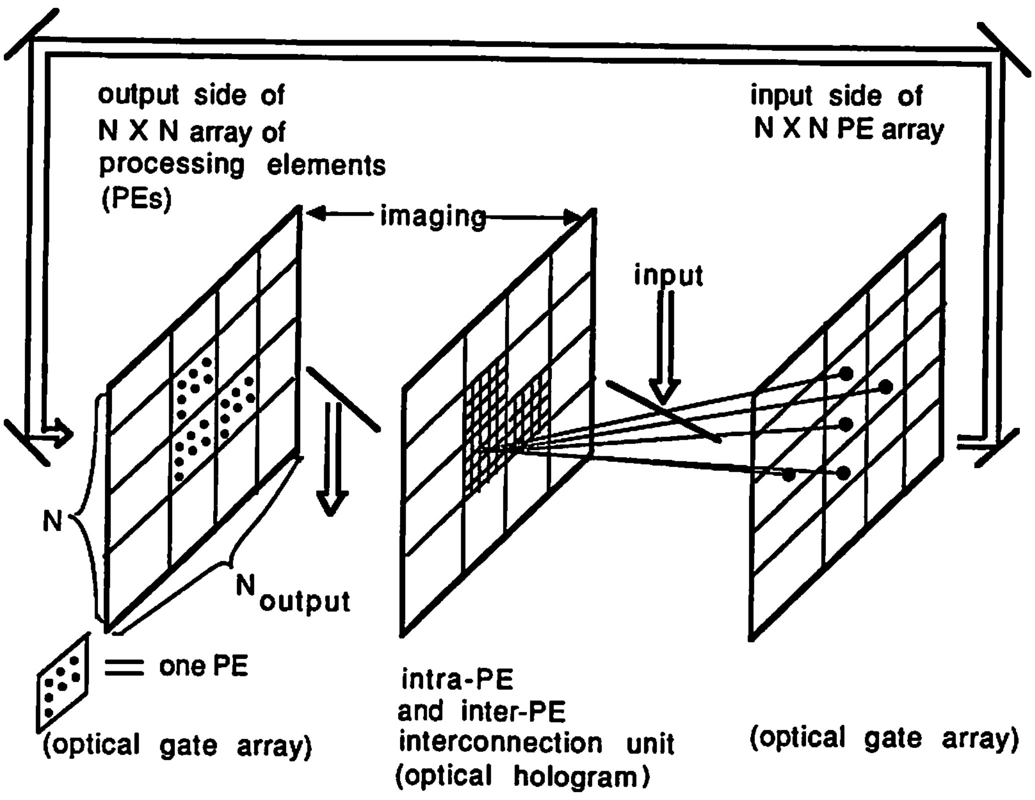
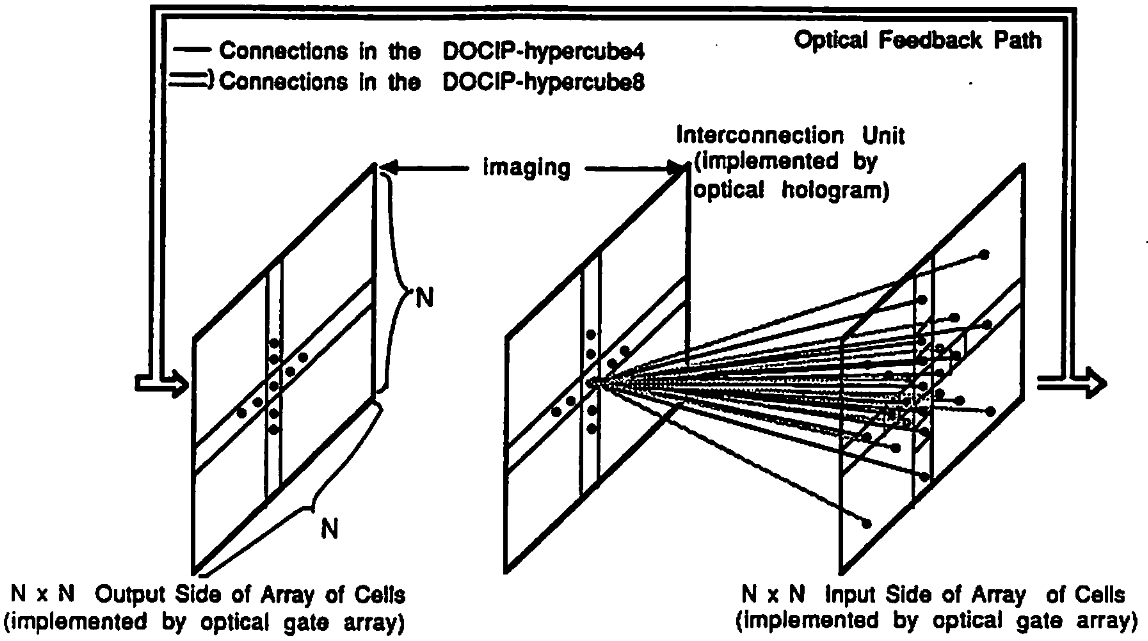


Figure 6.5: An optical 4-connected or 8-connected cellular array (DOCIP-array4 or DOCIP-array8). Each cell connects with its four nearest cells and itself by optical 3-D free interconnection. The optical hologram provides both intra-cell and inter-cell interconnections. Intra-cell interconnections and imaging optics are omitted for clarity. The input and output sides of the optical gate array are interconnected by an optical feedback path and are shown separately for clarity.



— Connections in the DOCIP-hypercube4
⟹ Connections in the DOCIP-hypercube8
Optical Feedback Path
Interconnection Unit (implemented by optical hologram)
Imaging
N
N
N x N Output Side of Array of Cells (implemented by optical gate array)
N x N Input Side of Array of Cells (implemented by optical gate array)

Figure 6.6: An optical 4-directed or 8-directed cellular hypercube (DOCIP-hypercube4 or DOCIP-hypercube8). Each cell connects with cells in the 4 directions or 8 directions at distances $1, 2, 4, 8, ..., 2^k$ from it by optical 3-D free interconnection.

Control of the DOCIP can be easily realized by using an electronic host instead of an all-optical control unit, since control of SIMD systems is primarily a serial process. The tradeoff is a possible inefficiency in the interfaces between electronic and optical units. Because of this, the all-optical approach may be preferable in the long term.

To efficiently utilize optical gates, they can be interconnected with a 2-D optical multiplexing technique (Fig. 6.7) in which a common controllable mask is used for all cells. This is essentially to improve the flexibility of the space invariant interconnection system in [Jenkins84b, Sawchuk84, Chavel83]. In [Huang87a] and [Taboury87], a similar holographic interconnection technique is discussed in detail.

The optical multiplexing technique has the following advantages: 1) the DOCIP will no longer require the broadcasting of instructions from the control unit — instead all cells fan their outputs into a common controlling mask pixel; 2) it will reduce the number of gates; and 3) each cell has a simple structure — essentially containing only a 3-bit memory with inverting and non-inverting outputs, and a multiple-input OR gate for dilation.

Order of magnitude execution times for image processing on the DOCIP machines and on the conventional electronic array processors are compared in Table 6.2. In contrast with the DOCIP-array, the DOCIP-hypercube improves the execution times of many global operations from $O(N)$ to $O(log_2 N)$ time which is difficult to achieved by planar VLSI technology. Comparing with the conventional-array processors having serial or N-parallel input/output, the DOCIP-array will have the same order of performance in local and global operations but will be improved in input/output performance, and in principle could be as low as $O(1)$ in

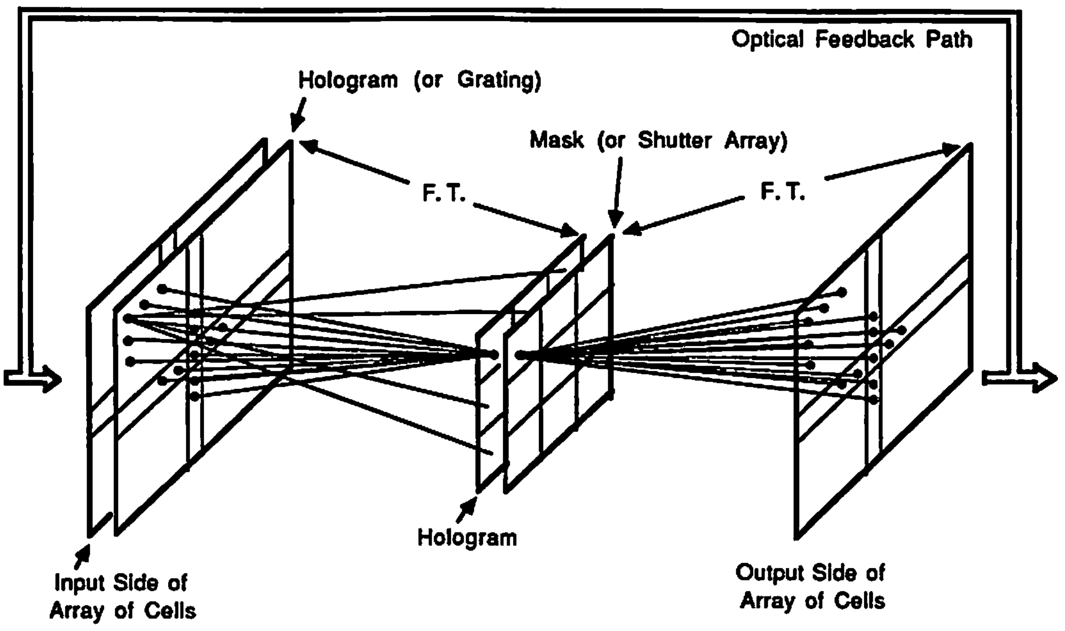


Figure 6.7: An optical multiplexing technique for interconnections. The hologram (or grating) attached to (or imaged from) the output side of the array of cells is to group all the same type of interconnections with the same direction and length (i.e. all the routing functions of all cells defined with the paticular element of the neighborhood configuration N_c; there exists k different types of interconnections for k elements in N_c) from all cells. Each subhologram of the second hologram is to provide a common type of interconnections for all cells (e.g. all cells connect to their east nearest neighbors). The common controllable mask (or shutter array) (e.g. 3×3 mask for DOCIP-array8) attached to the second hologram is to provide all 2^k (e.g. $k = 9$ for DOCIP-array8) possible spatial invariant interconnection patterns for all cells.

I/O operations. The DOCIP-hypercube will not only be improved in input/output performance but also in global operations. One important feature in the design of the DOCIP-array and DOCIP-hypercube is that optical 3-D free interconnection capabilities can be used to reduce the cell hardware requirements as well as solve the global connection and parallel I/O problems.

TECHNOLOGY / OPERATION	Conventional Array (Electronics)	DOCIP-array (Optics)	DOCIP-hypercube (Optics)
Local Operations	O(1)	O(1)	O(1)
Global Operations	O(N) or $O(N^2)$	O(N)	O(logN)
Communication PE↔Main Memory	O(N) or $O(N^2)$	O(1)	O(1)
Input/Output	O(N) or $O(N^2)$	O(1)	O(1)

Table 6.2: Cellular image processor execution times for $N \times N$ image data. It roughly compares the execution time for the conventional electronic array processor, the DOCIP-array and the DOCIP-hypercube.

Another intersting question is, "Can we also build an analog optical computer to do morphological image processing?" The answer is "yes", because the dilation and erosion are exactly the same as the convolution and correlation by adding some particular thresholding (Section 5.4) which are possibly implemented by

Fourier optics. But, an analog optical morphological processor may face drawbacks such as dynamic range, accuracy limitations, and flexibility limitations as other analog systems do.

In Chapter 9, a prototype DOCIP experimental system with a 54-gate optical processor, an instruction decoder and electronic input/output interfaces will be implemented for demonstrating the concept of the DOCIPs.

Chapter 7

DOCIP: Control and Programming

The DOCIP is directed particularly toward spatial problems — implementing BIA algorithms, but is alos general purpose in the sense that it can simulate any Turing or Random Access Machine (RAM) machine. Both the DOCIP-array and the DOCIP-hypercube have a simple control: 1) a single-control-level that has only one higher control level for interpretation between the program and the data; and 2) a single instruction that includes 'fetch', 'execute', 'store', and pipelines the 3 fundamental operations of BIA.

A single-instruction machine is generally considered to be impractical in electronics because it may lead to excessively complex programs. In the DOCIPs, the single-instruction machine is well suited to the system architecture. The reason for this is that the optical parallelism and global interconnection capabilities of DOCIP also contribute to parallel optical control signal flow. We will first discuss the control structure, then the instruction set and the programming.

7.1 Control Structure and Level 1 State Machine

In addition to having a simple cell structure, the DOCIP also has an extremely simple control structure; it includes only a clock, a program counter, a test/branch module, and an instruction memory/decoder with simple codes as described in Section 6.2. We will discuss the reason for this simple design through the study of state machines.

A state machine is a sequential machine with a finite number of states. A state provides the memory of past history to determine future behavior. We define a level 0 state machine as a machine with no other control unit other than a state transition function. A level n state machine is a machine with n control levels. A somewhat similar classification of state machines is given by C. R. Clare in [Clare73]. Some examples:

1. A Turing machine is a level 0 state machine (Fig. 7.1). It has no separate control unit. The control is provided by the instruction memory which memory serves to store the state transition function. There is no program counter and the instructions are not accessed through an address unit.

2. A von Neumann machine is a level 1 machine (Fig. 7.2). Its control unit includes a program counter, address register, and instruction register. Instructions are retrieved by their address via the address register.

3. A microprogrammed machine is a level 2 machine (Fig. 7.3), since it is considered as a von Neumann machine with one extra level of interpretation between the stored program and the ALU.

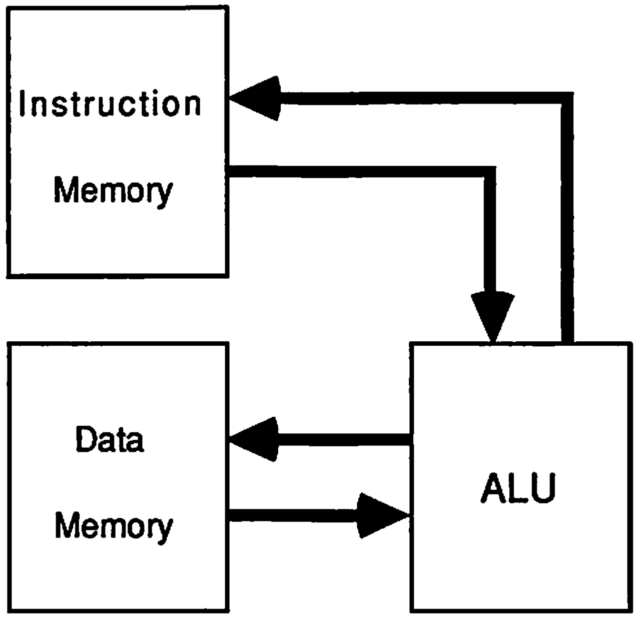


Figure 7.1: A level 0 state machine. ALU: Arithmetic Logic Unit.

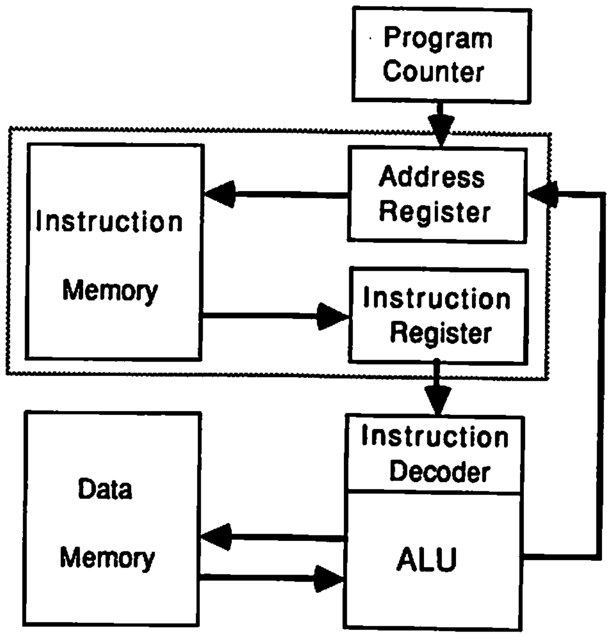


Figure 7.2: A level 1 state machine.

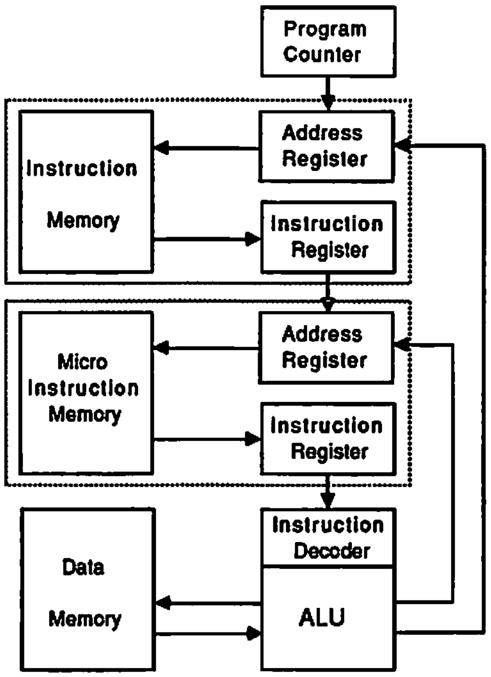


Figure 7.3: A level 2 state machine.

If the machine has more control levels, it usually has the following positive features:

- the length of program is shorter;
- the requirement of parallel communication is reduced; and
- the complexity of parallel interconnection is simpler.

However, the higher level machine also has the following negative features:

- the execution time is longer; and
- the gate requirement of the control is higher.

At present, optical parallel and global interconnections are generally inexpensive, but gates are expensive. Compared to electronics (VLSI), the major constraint in a digital optical system is the availability of gates. Hence, at this stage, the DOCIP is designed as a level 1 state machine as shown in Fig. 7.1. Then, the DOCIP has the following nice properties:

- it makes extensive use of optical parallelism and 3-D free interconnection capabilities;
- it reduces the optical gate count of the control unit;
- it has a simple control structure (i.e. the control of the DOCIP has one and only one interpreting state machine); and
- the control unit also utilizes the advantages of parallel processing.

To avoid using the DOCIP for the possible complex programs from high level vision or other complex sequential problems, which are essentially serial processes instead of 2-D parallelism, we may attach the DOCIP to a general purpose host machine.

7.2 Instruction Set

7.2.1 1-step Instruction Machine

The program of a conventional computer usually is a sequence of 3-step instructions which includes these three steps: 'fetch', 'execute', and 'store' (Fig. 7.4). The basic reason is that operations often require two or more operands from memory modules. But in a conventional machine it is difficult to access several memory modules in parallel. The central processing unit (CPU) of a conventional computer is then designed to operate on the contents of an accumulator rather than accessing memory words directly. For example, the operation $C = A + B$ is performed by the three instructions:

1. LOAD A — Transfer A to accumulator S,
2. ADD B — $S \leftarrow S + B$,
3. STORE C — Transfer result to memory location C.

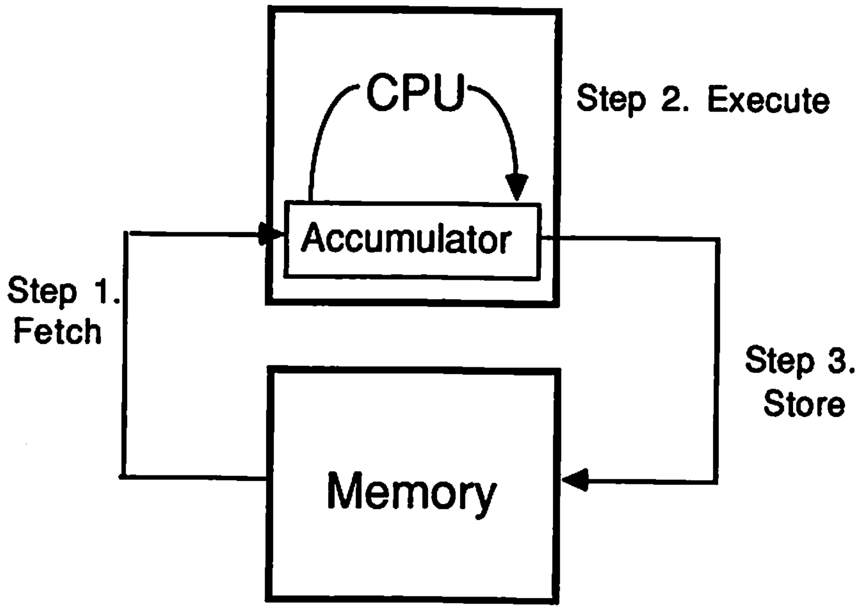


Figure 7.4: A 3-step instruction machine. Its instructions include three steps: 'fetech ', 'execute', and 'store'.

A better match to the capabilities of optics is obtained with the DOCIPdesigned as a 1-step instruction machine which combines the 'fetch', 'execute', and 'store' into one step (executed in one instruction cycle time) (Fig. 7.5).

7.2.2 Single Instruction Machine

Recently, it has been argued that "complex instructions are rarely used by actual programs, their inclusion into the processor's instruction set has more negative

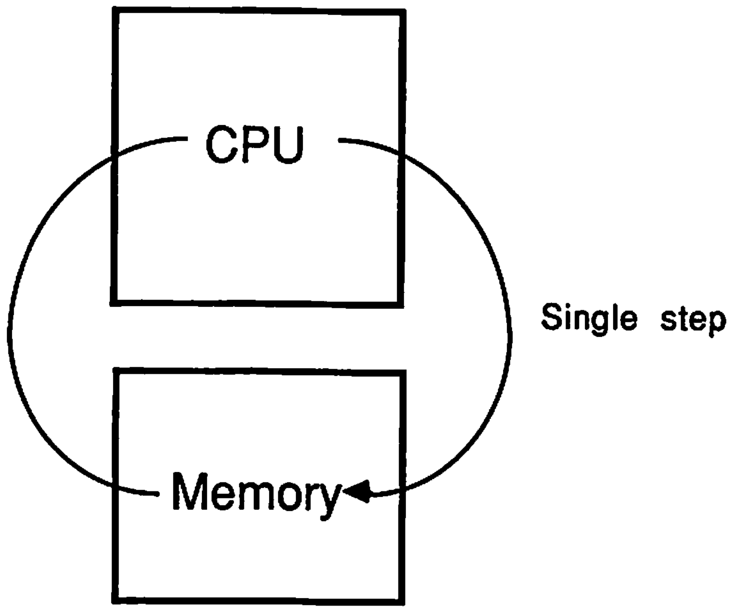


Figure 7.5: A 1-step instruction machine. Its instructions combines the 'fetech', 'execute', and 'store' into one step.

effects on overall performance than it has positive ones" [Katevenis85]. Obviously, the simplest case for this consideration is to reduce the instruction set into only one instruction. Van der Poel has designed a simple one-instruction machine which has only one instruction [Poel56]. However, it is criticized as being quite impractical [Hayes78] . The primary objection to a simple instruction set is that it may lead to excessively complex programs.

The DOCIP is a specialized machine designed for parallel binary image processing. The three fundamental operations are generally the most frequently used operations and are able to implement any other image transformation. Thus, our instruction set should include these three fundamental operations. Having just one instruction leads extremely simple processing elements made up of minimal logic circuits. This allows a highly parallel machine with a moderate total number of gates, appropriate for optics since gates are expensive and interconnections are plentiful. Obviously, there is a tradeoff between processor simplicity and programming complexity. The DOCIP is designed as a one-instruction machine in order to:

- simplify the instruction set,
- maximize the utilization of optical parallelism and 3-D interconnection,
- reduce the execution time, and
- minimize the hardware cost and gate requirement.

This one and only one instruction has the following format:

$$(c, d_1, d_2, d_3, s_1, s_2, ..., s_6, n_1, n_2, ..., n_k, j_1, j_2, a_1, a_2, ..., a_l, b_1, b_2, ..., b_l)$$

where k is determined by the chosen neighborhood configuration N and l defines the maximum length of a program, 2^l. The functions of these $12+k+2l$ instruction codes are given below.

- c is used to select the image from the input or from the feedback, $c = 0$ for input and $c = 1$ for feedback.
- d_1, d_2, and d_3 are used to select the destination memory for storing the image; $d_i = 1$ implies the image is stored into memory M_i.
- $s_1, s_2, ..., s_6$ are used to select the output from the memory elements; $s_{2i-1} = 1$ implies select the non-inverted image from memory M_i, s_{2i} implies select the complement of the image in memory M_i.
- $n_1, n_2, ..., n_k$ are used to control the neighborhood mask, i.e. to supply the reference image; a 1 corresponds to a foreground point (value 1) and a 0 corresponds to a background point (value 0) of the reference image.

- j_1 and j_2 are used to control the absolute jump or conditional jump; $j_1 = 1$ implies an absolute jump and $j_2 = 1$ implies a conditional jump. $j_1 = j_2 = 1$ represents a 'Halt' (i.e. do not continue to the next instruction).
- $a_1, a_2, ..., a_l$ are the address for the jump.
- $b_1, b_2, ..., b_l$ are the address of the instruction.

7.2.3 General Purpose Machine

A Turing Machine defines what is computable. A Random Access Machine (RAM) is as powerful as a Turing Machine, since they can simulate each other. A general purpose machine means that it has the capability of computing any algorithm that is computable. Here, we will show that the DOCIP is a general purpose machine by simulating any RAM program. The concept of the Turing Machine and the RAM can be found in [Hopcroft79, Aho74].

Simulation of A RAM Machine

The instruction set of the RAM consists of the following operations:

Operation code	**Address**
1. LOAD	operand
2. STORE	operand
3. ADD	operand
4. SUB	operand
5. MULT	operand
6. DIV	operand
7. READ	operand
8. WRITE	operand
9. JUMP	label
10. JGTZ	label
11. JZERO	label
12. HALT	

Before going to the simulation, please note that the instruction of the DOCIP essentially performs image transformations, i.e. transforms an array of data, while an instruction of the RAM performs a one-word operation. Thus, we first make an assumption: the word length is w in the RAM's memory and we can choose any w bits in the image memory in the DOCIP to represent a word. Then, we can simulate the RAM instructions on the DOCIP in the following straightforward way:

1. The data transfer instructions (LOAD, STORE, READ, and WRITE) are simulated by the first 10 instruction codes of the instruction of the DOCIP which are $c, d_1, ..., d_3$, and $s_1, ..., s_6$.

2. The data manipulation instructions (ADD, SUB, MULT, and DIV) are simulated by the following $10 + k$ instruction codes: $d_0, ..., d_3$, $s_1, ..., s_6$, and $n_1, ..., n_k$ in $O(w^2)$ steps for the DOCIP; because we can implement the ADD and the SUB in $O(w)$ steps and the MULT and the DIV in $O(w^2)$ steps as we will see in Chapter 8.

3. The program control instructions (JUMP,JGTZ, JZERO, and HALT) are simulated by the following:

   - j_1 and j_2 control the type of jump (absolute or conditional);
   - $a_1, ..., a_l$ gives the jump address;
   - The above $l + 2$ bits with the other bits of the single instruction can be used to simulate JGTZ and JZERO; because the 3 fundamental operation can be used to implement the comparison circuit in $O(w)$ steps..
   - The HALT can be done by JUMP to the address which is the address of itself or JUMP to an instruction which has $j_1 = j_2 = 1$.

Thus, any RAM instruction can be simulated by the DOCIP-array and the DOCIP-hypercube in at most $O(w^2)$ steps. Therefore, the DOCIP-array and the DOCIP-hypercube are "general purpose" and are able to compute any computable algorithm.

Simulation of A Symbolic Subsitution Processor

Another approach to demonstrate that it is general purpose is: 1) the DOCIP can simulate any symbolic substitution system; and 2) a symbolic substitution system has simulated a Turing machine in [Brenner86a].

A general mathematical formalism of symbolic substitution has been developed in Section 5.1. For a local search-pattern and replacement-pattern (i.e. $R_1, R_2, Q \subset N_{array}$ or $N_{hypercube}$), the DOCIP-array or DOCIP-hypercube can implement a symbolic substitution rule in four (or five with the optional mask) steps:

Assume start with X in M_1.

1. $\overline{M_1} \oplus \check{R}_1 \rightarrow M_2$
2. $M_1 \oplus \check{R}_2 \rightarrow M_3$
3. $M_2 \cup M_3 \rightarrow M_3$
4. $\overline{M_3} \oplus Q \rightarrow$ Out $(= (X \circledast R) \oplus Q)$

Let the pixels used in the substitution rule(s) of a symbolic substitution processor be the *neighborhood*, N_{ss}, of the processor. We see from the above steps that the DOCIP can simulate the symbolic substitution processor in constant time if the two machines have the same neighborhood. If N_{ss} is not a subset of the DOCIP neighborhood, then the simulation will take longer. In either case, it is not presently known how many steps it takes the symbolic substitution processor to simulate the DOCIP.

Simulation of the Game "Life"

One more obvious way for demonstrating the general purpose capability is: 1) the DOCIP implements BIA and can implement any game of life or pattern of growth of binary cellular automata as described in Subsection 4.4.1; and 2) with suitable encoding of variables and processes, the Conway's game "Life" have been used to simulate a general purpose machine [Berlekamp82].

7.3 Programming

BIA and DOCIP architectures can have many applications in character recognition, industrial inspection, medical and scientific research. Since BIA is able to implement morphological operations efficiently, the DOCIP machines can efficiently analyze the shape and connectivity of regions as well as measure their size; they also have the potential to accomplish any image transformation. Their appications for numerical array computation will be contained in Chapter 8. In this section, we discuss a decomposition algorthm for the DOCIP programming, and then illustrate the programming with a size verification algorithm.

7.3.1 Decomposition of Dilation

From the applications of binary image algebra (Chapter 4), we see that the performances of most algorithms are primarily limited by the complexity of dilation. The reseason is that the other two fundamental operations (complement and union) are executed in $O(1)$ in a cellular computer, but a dilation with a *large* reference image is very difficult to implement at a clock cycle (unless we have a fully interconnected network which is too costly). However, the dilation of an image with size $N \times N$ by a reference image $M \times M$ takes long time for input/output and $O(N^2 \times M^2)$ computation time for a uniprocessor. To comprimise the hardware implementation cost and the time complexity, we have to consider the decomposition of a dilation with a *larger* reference image into a sequence of dilations with *small* reference images where each dilation with *small* reference image can be implemented in a DOCIP-array or DOCIP-hypercube at a clock cycle. If it is possible to decompose R into a sequence $R = E_1 \oplus E_2 \oplus ... \oplus E_k$, then

$$X \oplus R = (...((X \oplus E_1) \oplus E_2) \oplus ... \oplus E_k). \tag{7.1}$$

This decomposition may not exist, in which case R can always be decomposed as $R = R_1 \cup R_2 \cup ... \cup R_k$, and then

$$X \oplus R = (X \oplus R_1) \cup (X \oplus R_2) \cup ... \cup (X \oplus R_k) \tag{7.2}$$

where each R_j can be composed from the smaller reference images E_i.

The following is the performance analysis of DOCIP-array and DOCIP-hypercube for computing the dilation with different type of reference images. The time complexity of the dilation is the same as generating the reference image from the elementary image $I = \{(0,0)\}$ by unions or dilations with *small* reference images where each small reference image must be a subset of nearest neighborhood or cellular hypercube neighborhood (i.e. N_{array4}, N_{array8}, $N_{hypercube4}$, or $N_{hypercube8}$).

Now let us consider different kinds of reference image:

1. Point reference image: the reference image contains only one image point at coordinate (x, y) (without loss of generality, consider only $x, y \geq 0$):

$$\begin{aligned} P &= \{(x,y)\} \\ &= \{(a_{k-1}a_{k-2}...a_0, b_{l-1}b_{l-2}...b_0)_2\} \\ &= \Pi_{j=0}^{k-1} A^{a_j 2^j} \Pi_{j=0}^{l-1} B^{b_j 2^j} \end{aligned} \tag{7.3}$$

   where $(a_{k-1}a_{k-2}...a_0, b_{l-1}b_{l-2}...b_0)_2$ is the binary representation of x, y, $k = \lfloor log_2(x) \rfloor, l = \lfloor log_2(y) \rfloor$, and $\Pi_{j=0}^{k-1} A^{a_j} \equiv A^{a_0} \oplus A^{a_2} \oplus ... \oplus A^{a_{k-1}}$. Thus, it requires $O(log_2 L)$ for DOCIP-hypercube and $O(L)$ for DOCIP-array where L is the distance between the image point and the origin.

2. Horizontal or vertical line reference image:

   - Consider the horizontal line reference image centered at the origin (0,0): $H = \{(-l,0),(-l+1,0),...,(0,0),(1,0),...,(m,0)\}$ (where $m-1 \leq l \leq m$). With the properties of spatial symmetrical and invariant interconnection patterns in cellular architectures, the time complexity of generating H is the same as that of generating $H' = \{(0,0),(1,0),...,(m,0)\}$. If $m = (a_{k-1}a_{k-2}...a_0)_2$ (binary representation) and $k = \lfloor log_2(m) \rfloor$, then
     $$\begin{aligned} H' &= \cup_{i=0}^{m} A^i \\ &= \Pi_{j=0}^{k-1}(\cup_{i=0,1,2^1,...,2^j} A^{a_j \cdot i}). \end{aligned} \tag{7.4}$$
     Thus, it requires $O(log_2 L)$ for DOCIP-hypercube and $O(L)$ for DOCIP-array where L is the distance between the farest image point and the origin.

   - Consider the horizontal line H'' is located at (x,y), i.e.
     $$\begin{aligned} H'' &= \{(x-l,y),(x-l+1,y),...,(x,y),(x+1,y),...,(x+m,y)\} \\ &= \{(-l,0),(-l+1,0),...,(0,0),(1,0),...,(m,0)\} \oplus \{(x,y)\} \\ &= H \oplus P \end{aligned} \tag{7.5}$$
     where H is a horizontal line reference image centered at origin and P is a point reference image. Since both H and P take $O(log_2 L)$ and $O(L)$ for DOCIP-hypercube and DOCIP-array respectively. Thus, for the shifted horizontal or vertical reference image, it requires $O(log_2 L)$ for DOCIP-hypercube and $O(L)$ for DOCIP-array where L is the distance between the farest image point and the origin.

3. Rotated line reference image:

- Rotated multiple 45 degrees: let us consider $D = \{(0,0),(1,1),...,(m,m)\}$. If $m = (a_{k-1}a_{k-2}...a_0)_2$ (binary representation) and $k = \lfloor log_2(m) \rfloor$, then

$$\begin{aligned} D &= \bigcup_{i=0}^{m}(AB)^i \\ &= \Pi_{j=0}^{k-1}(\bigcup_{i=0,1,2^1,...,2^j}(AB)^{a_j \cdot i}) \qquad (7.6) \\ &= \Pi_{j=0}^{k-1}(\bigcup_{i=0,1,2^1,...,2^j}(A^{-1}B)^{a_j \cdot i})\Pi_{i=0}^{k-1}A^{a_i \cdot 2^i}. \end{aligned}$$

  Thus, it requires $O(log_2L)$ for DOCIP-hypercube and $O(L)$ for DOCIP-array where L is the distance between the farest image point and the origin.

- Arbitrary rotated line reference image: By a similar argument as the above, for generating arbitrary rotated line reference image, it still takes $O(log_2L)$ for DOCIP-hypercube and $O(L)$ for DOCIP-array where L is the distance between the farest image point and the origin. In general, DOCIP-hypercube8 (or DOCIP-array8) requires about twice hardware complexity of DOCIP-hypercube4 (or DOCIP-array4) and at best have twice speedup for generating arbitrary rotated line.

4. Parallelogram (filled) reference image: A parallelogram reference image P can be generated by a dilation of two line reference image. So, it still takes $O(log_2L)$ for DOCIP-hypercube and $O(L)$ for DOCIP-array where L is the distance between the farest image point and the origin.

Obviously, the hexagon and the polygon (e.g. Fig. 7.6) (filled) can be generated by a dilation with 3 line reference images and a dilation with 4 line reference images respectively, and so forth. Thus, a simple algorithm for implementing the dilation is to consider a *large* reference image as a number of connected sets of parallelograms (filled) R_i. Each connected set of parallelograms is generated by a sequential dilation of some line reference images and the union of all connected sets of parallelograms (filled) is the desired reference image. Thus, for a reference image with $M \times M$ image points it takes the time between $O(M \times M \times log_2 M)$ (assume that any pair of two or more arbitrary image points can not be formed a connected set of parallelograms, i.e. each image point is a set of parallelograms) and $O(log_2 M)$ for DOCIP-hypercube; and it takes the time between $O(M \times M \times M)$ and $O(M)$ for DOCIP-array.

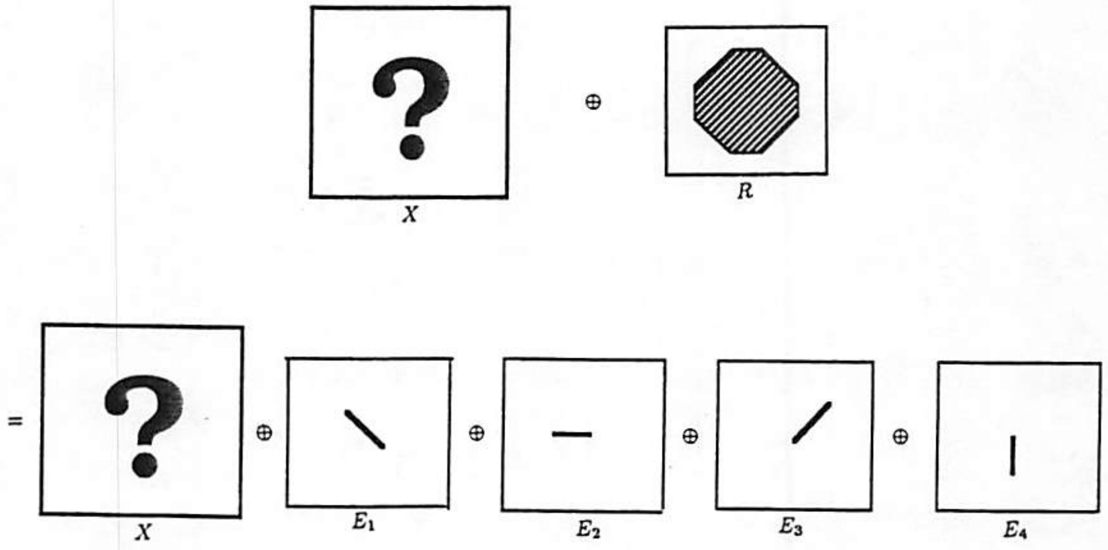


Figure 7.6: An example of decomposing a dilation with a *larger* reference image R into a sequential dilation with some *smaller* reference image E_i. It requires $O(N)$ and $O(logN)$ time for DOCIP-array and DOCIP-hypercube respectively.

A sorting algorithm of decomposition for a nearest-neighbor connected cellular processor can be also found in [Zhuang86].

7.3.2 A Programming Example— Size Verification

Here we illustrate the programming of the DOCIP machines by a simple size verification algorithm:

- **Problem:** Given an input image X with 31×31 pixels (Fig. 7.7) which contains some square objects X_i, we want to preserve those square objects X_i which satisfy the following condition:

$$\text{size of } R \leq \text{size of } X_i < \text{size of } Q$$

where R and Q are reference images as shown in Fig. 7.8. Other objects will be eliminated in the output image Y. The expected output image Y is shown in Fig. 7.9.

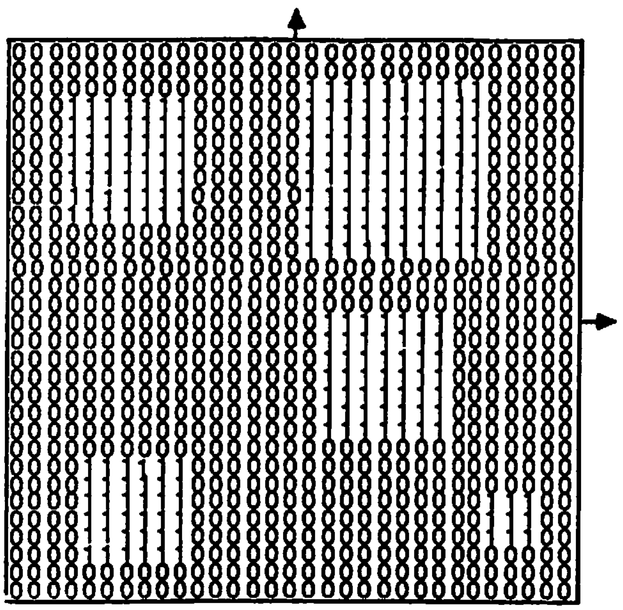
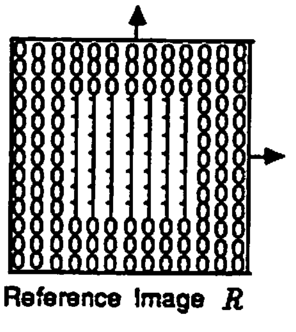


Figure 7.7: The input image X.



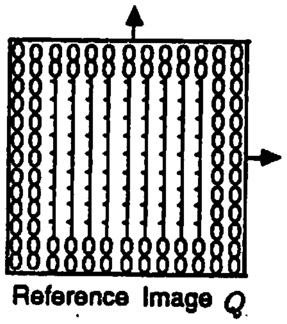


Figure 7.8: The reference images R and Q.

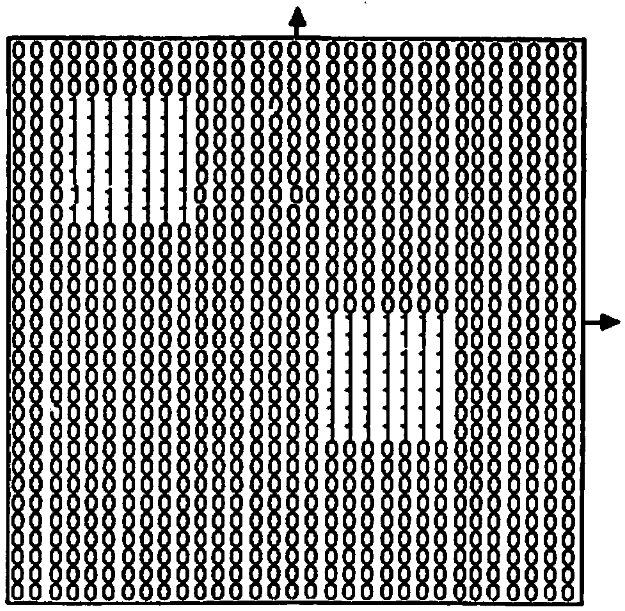


Figure 7.9: The expected output image Y.

- **Algebraic expression for the size verification using band pass morphological filtering:**

$$\overline{(\overline{\overline{\overline{X}\oplus R}}\oplus R)\cup(\overline{\overline{X}\oplus Q}\oplus Q)}$$

where $R = \check{R}$ and $Q = \check{Q}$ in this special example.

- **Algorithm for the DOCIP-array8:**

$$\overline{(\overline{\overline{\overline{X}\oplus E^3}}\oplus E^3)\cup(\overline{\overline{X}\oplus E^4}\oplus E^4)}$$

where E (Fig. 7.10) is the allowed reference image with the maximum size at a clock cycle in the DOCIP-array8, the reference images $R = E^3 = E\oplus E\oplus E$ and $Q = E^4 = E\oplus E\oplus E\oplus E = R\oplus E$.

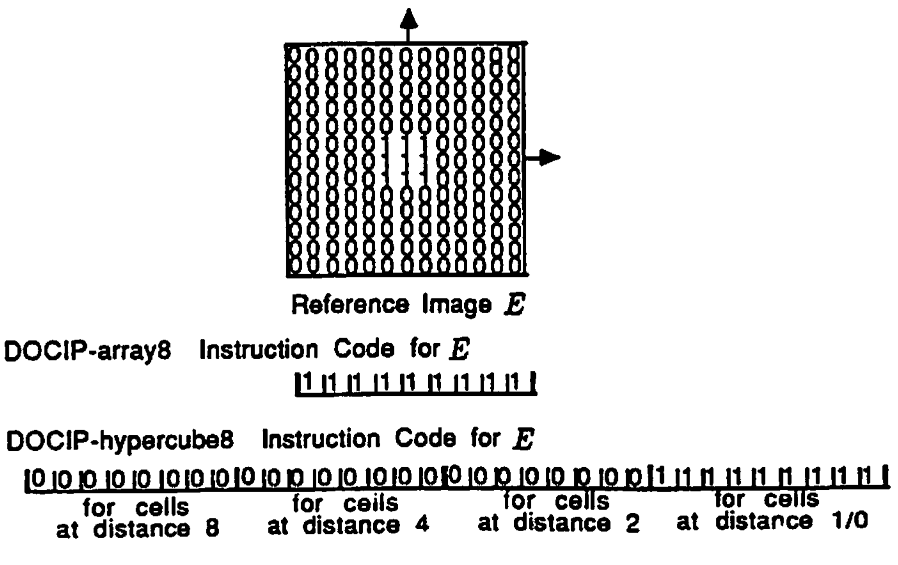


Figure 7.10: An allowed reference image E at a clock cycle in the DOCIP-array8 (also allowed in DOCIP-hypercube8) and its corresponding 9 (or 33) bits in instruction $(n_1n_2...n_k)$ for controlling the neighborhood mask (i.e. the reference image for the dilation).

The DOCIP-array8 requires 13 steps to complete this algorithm, its program (instructions) is in the following:

Assume start with $X \rightarrow M_1$ (X stored in Memory1)

1. $\overline{M_1} \oplus E \rightarrow M_2$ $(= \overline{X} \oplus E)$
2. $M_2 \oplus E \rightarrow M_2$ $(= \overline{X} \oplus E^2)$
3. $M_2 \oplus E \rightarrow M_2$ $(= \overline{X} \oplus E^3)$
4. $M_2 \oplus E \rightarrow M_3$ $(= \overline{X} \oplus E^4)$
5. $\overline{M_2} \oplus E \rightarrow M_2$ $(= \overline{\overline{X} \oplus E^3} \oplus E)$
6. $M_2 \oplus E \rightarrow M_2$ $(= \overline{\overline{X} \oplus E^3} \oplus E^2)$
7. $M_2 \oplus E \rightarrow M_2$ $(= \overline{\overline{X} \oplus E^3} \oplus E^3)$
8. $\overline{M_3} \oplus E \rightarrow M_3$ $(= \overline{\overline{X} \oplus E^4} \oplus E)$
9. $M_3 \oplus E \rightarrow M_3$ $(= \overline{\overline{X} \oplus E^4} \oplus E^2)$
10. $M_3 \oplus E \rightarrow M_3$ $(= \overline{\overline{X} \oplus E^4} \oplus E^3)$
11. $M_3 \oplus E \rightarrow M_3$ $(= \overline{\overline{X} \oplus E^4} \oplus E^4)$
12. $\overline{M_2} \cup M_3 \rightarrow M_3$ $(= (\overline{\overline{\overline{X} \oplus E^3} \oplus E^3}) \cup (\overline{\overline{X} \oplus E^4} \oplus E^4))$
13. End with $\overline{M_3} \rightarrow Y$ $(= \overline{(\overline{\overline{\overline{X} \oplus E^3} \oplus E^3}) \cup (\overline{\overline{X} \oplus E^4} \oplus E^4)})$

- **Algorithm for the DOCIP-hypercube8:**

$$\overline{(\overline{\overline{\overline{X} \oplus P \oplus E} \oplus P \oplus E}) \cup (\overline{\overline{X} \oplus P \oplus E^2} \oplus P \oplus E^2)}$$

where P (Fig. 7.11) and E (Fig. 7.10) are allowed reference images at a clock cycle in the DOCIP-hypercube8, the reference images $R = E^3 = P \oplus E$ and $Q = E^4 = P \oplus E^2 = R \oplus E$.

The DOCIP-hypercube8 requires 10 steps to complete this algorithm, its program (instructions) is shown in the following:

Assume start with $X \rightarrow M_1$ (X stored in Memory1)

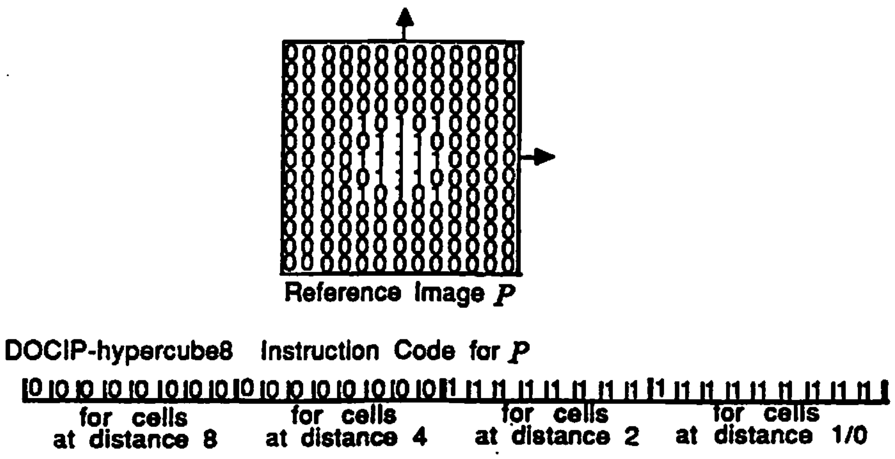


Figure 7.11: An allowed reference image P at a clock cycle in the DOCIP-hypercube8 (not allowed in the DOCIP-array8) and its corresponding 33 bits (assume 31×31 cells) in instruction $(n_1 n_2 ... n_{33})$ for controlling the neighborhood mask (i.e. the reference image for the dilation).

1. $\overline{M_1} \oplus P \rightarrow M_2\ (= \overline{X} \oplus P)$
2. $M_2 \oplus E \rightarrow M_2\ (= \overline{X} \oplus P \oplus E)$
3. $M_2 \oplus E \rightarrow M_3\ (= \overline{X} \oplus P \oplus E^2)$
4. $\overline{M_2} \oplus P \rightarrow M_2\ (= \overline{\overline{X} \oplus P \oplus E^2} \oplus P)$
5. $M_2 \oplus E \rightarrow M_2\ (= \overline{\overline{X} \oplus P \oplus E^2} \oplus P \oplus E)$
6. $\overline{M_3} \oplus P \rightarrow M_3\ (= \overline{\overline{X} \oplus P \oplus E^2} \oplus P)$
7. $M_3 \oplus E \rightarrow M_3\ (= \overline{\overline{X} \oplus P \oplus E^2} \oplus P \oplus E)$
8. $M_3 \oplus E \rightarrow M_3\ (= \overline{\overline{X} \oplus P \oplus E^2} \oplus P \oplus E^2)$
9. $\overline{M_2} \cup M_3 \rightarrow M_3\ (= (\overline{\overline{\overline{X} \oplus P \oplus E} \oplus P \oplus E}) \cup (\overline{\overline{X} \oplus P \oplus E^2} \oplus P \oplus E^2))$
10. End with $\overline{M_3} \rightarrow Y\ (= \overline{(\overline{\overline{\overline{X} \oplus P \oplus E} \oplus E}) \cup (\overline{\overline{X} \oplus P \oplus E^2} \oplus P \oplus E^2)})$

The above programs can be translated into the machine instruction codes directly.

If we want to detect the geometric centers (locations) of the desired objects, then we can use a sequential thinning to achieve the homotopic skeleton (Theorem 4.5) (Fig. 7.12).

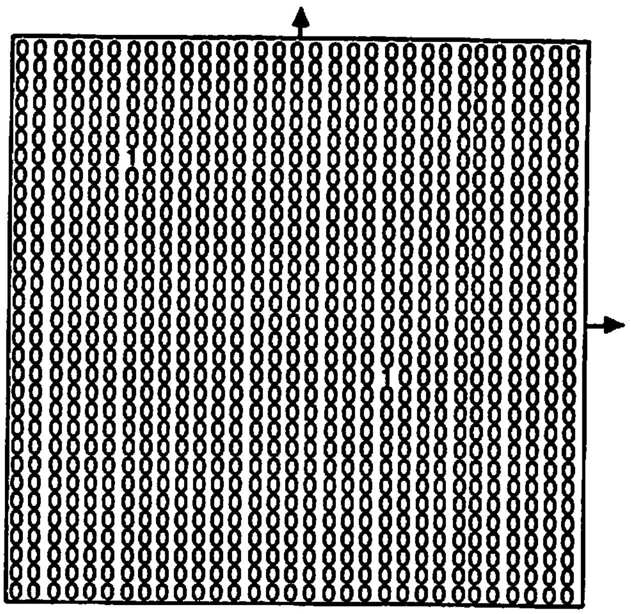


Figure 7.12: The locations of the desired objects in the output image Y.

Chapter 8

Parallel Optical Binary Arithmetic

Optical computers can operate on 2-D planes of data in parallel. Boolean logic equations do not provide a complete description of such parallel operations for binary arithmetic. An optical system that operates on planes of data should employ an inherently parallel mathematical description for its arithmetic. The purposes of this chapter are: to use binary image algebra to develop parallel numerical computation algorithms, and to describe the execution of these algorithms on a digital optical cellular image processor (DOCIP) architecture. We discuss three basic binary number representations: 1) binary row-coding; 2) binary stack-coding; and 3) binary symbol-coding for symbolic substitution arithmetic.

Parallel operations of binary addition, subtraction and multiplication are derived by BIA and illustrated as examples. Parallelism is achieved by performing arithmetic operations on many pairs of operands simultaneously. The carries for each pair of operands are essentially propagated serially to keep hardware complexity low [Psaltis86]. This enables speed-ups close to, and in some cases equal to, linear to be obtained. In this chapter we will consider only positive numbers. A suitable digital number representation will easily provide for negative numbers also. For example, two's complement arithmetic can be performed with only minor modifications to the algorithms and programs given in this chapter, and with the addition of one more bit (the sign bit) to each operand and result.

8.1 Binary Row-coded Arithmetic

Binary addition of two k-bit numbers yields at most $k+1$ bits, and binary multiplication of two k-bit numbers yields at most $2k$ bits. In this paper, we assume that all input numbers are padded with enough zeroes to avoid the possibility of overflow. This also guarantees that the different operands in the image will be treated separately. A binary row-coded number is encoded in a part of a row of an image. Although the word lengths of numbers do not need to be equal, we assume in this discussion that an image (bit plane) with $N \times N$ bits contains N^2/k numbers of k-bit length as a simple illustration (Fig. 8.1).

In this section, we describe parallel addition, subtraction and multiplication by BIA expressions and their programs on the DOCIP machine.

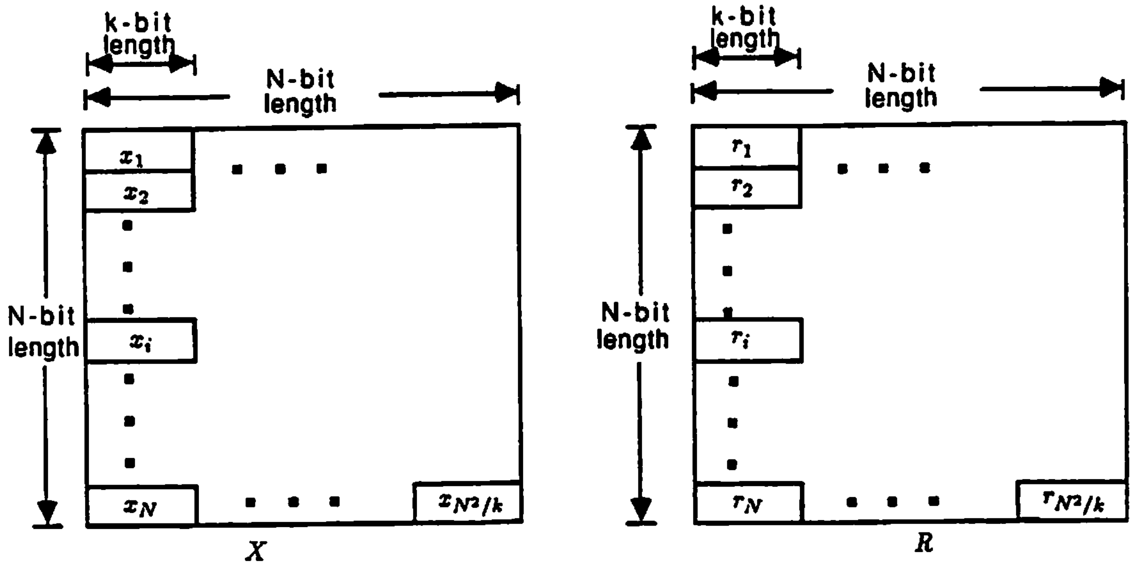


Figure 8.1: Binary row-coded numbers.

8.1.1 Addition of Binary Row-coded Numbers

Consider an image X (e.g. Fig. 8.2(a)) composed of N^2/k numbers $x_i, i = 1, 2, ..., N^2/k$, an image R (e.g. Fig. 8.2(b)) composed of N^2/k numbers $r_i, i = 1, 2, ..., N^2/k$, and the output of the addition $S = X + R$ (Fig. 8.2(c)). To realize this addition in parallel by means of BIA, we first consider the serial (carray-propagate) addition of 2 binary numbers $s_i = x_i + r_i$. The first step of serial addition is to add the least significant bits, say $x_{i(o)}$ and $r_{i(o)}$. The boolean logic equations for adding the two least significant bits (half-adder) are

- sum bit: $s_{i(o)} = x_{i(o)}$ XOR $r_{i(o)}$,
- carry bit: $c_{i(o)} = x_{i(o)}$ AND $r_{i(o)}$.

Now, applying the corresponding parallel operations of XOR and AND, i.e. the symmetrical difference Δ and intersection $\cap$, and shifting the set of carry bits by a dilation $\oplus$, we can implement parallel addition by the following recursive equations:

1. Define the initial states of images of sum bits and carry bits (called sum-bit image and carry-bit image) at time t_o as :

$$S(t_0) = X, \;\; C(t_0) = R. \tag{8.1}$$

2. The recursive relation between the states of the sum-bit image and carry-bit image at two adjacent time intervals is then:

$$S(t_{i+1}) = S(t_i) \triangle C(t_i) = \overline{\overline{S(t_i)} \cup C(t_i)} \cup \overline{S(t_i) \cup \overline{C(t_i)}} \tag{8.2}$$

$$C(t_{i+1}) = (S(t_i) \cap C(t_i)) \oplus A^{-1} = \overline{\overline{S(t_i)} \cup \overline{C(t_i)}} \oplus A^{-1} \tag{8.3}$$

   where $i = 0, 1, 2, ..., k+1$, and the elementary image A^{-1} is used to shift the carry-bit image one bit to the left for the next iteration.

3. After a maximum of $k + 1$ iterations, the sum-bit image is the result and the carry-bit image is the null image ϕ:

$$S(t_{k+1}) = X + R, \;\; C(t_{k+1}) = \phi. \tag{8.4}$$

This procedure is illustrated in Fig. 8.3.

The result of parallel addition of binary numbers with a maximum k-bit word size is obtained after $k + 1$ iterations. This algorithm can be implemented in the DOCIP architecture by the program (instructions) given below. M_1, M_2, and M_3 represent the three $N \times N$-bit memories. "$X \rightarrow M_1$" denotes "store X into memory M_1". Each numbered line represents a single DOCIP machine instruction for one value of i. Comments are in parentheses.

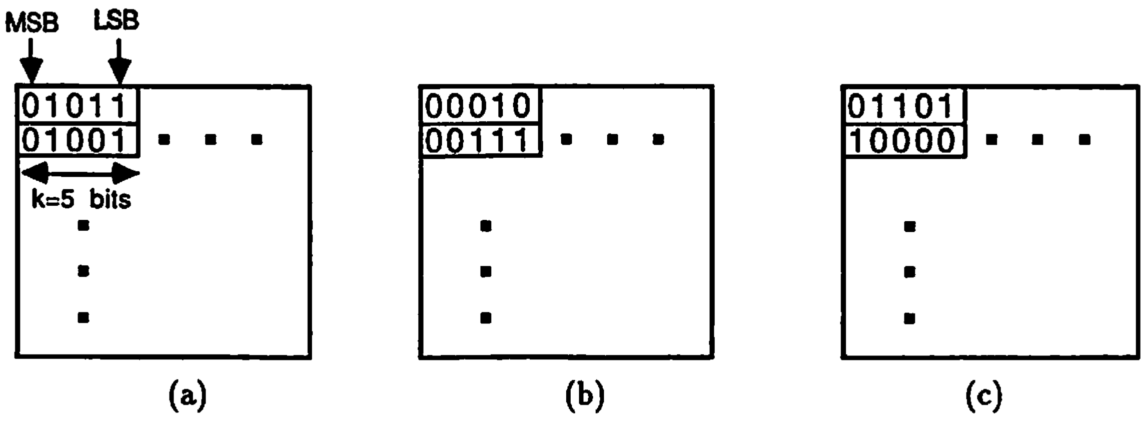


Figure 8.2: Parallel addition of binary row-coded numbers (I). (a): An image X of operands. (b): An image R of other operands. (c): The output $X + R$.

- Assume start with X in M_1 $(= S(t_0))$ and R in M_2 $(= C(t_0))$.

- First to k^{th} iterations:

  1. $\overline{M_1} \cup \overline{M_2} \rightarrow M_3$ $(= \overline{S(t_i)} \cup \overline{C(t_i)})$
  2. $\overline{M_1} \cup M_2 \rightarrow M_1$ $(= \overline{S(t_i)} \cup C(t_i))$
  3. $\overline{M_1} \cup \overline{M_2} \cup \overline{M_3} \rightarrow M_2$ $(= S(t_i) \cup \overline{C(t_i)})$
  4. $\overline{M_1} \cup \overline{M_2} \rightarrow M_1$ $(= S(t_{i+1}))$
  5. $\overline{M_3} \oplus A^{-1} \rightarrow M_2$ $(= C(t_{i+1}))$

  where $i = 0, 1, 2, ..., k - 1$.

- $(k + 1)^{th}$ iteration:

  1. $\overline{M_1} \cup M_2 \rightarrow M_3$ $(= \overline{S(t_k)} \cup C(t_k))$
  2. $M_1 \cup \overline{M_2} \rightarrow M_1$ $(= S(t_k) \cup \overline{C(t_k)})$

  3 $\overline{M_1} \cup \overline{M_3} \rightarrow$ Out $(= S(t_{k+1}) = X + R)$.

The total number of clock cycles for the execution of this program on the DOCIP machine is

$$t(k) \leq 5k + 3 = O(k)$$

which is independent of the number of words being added.

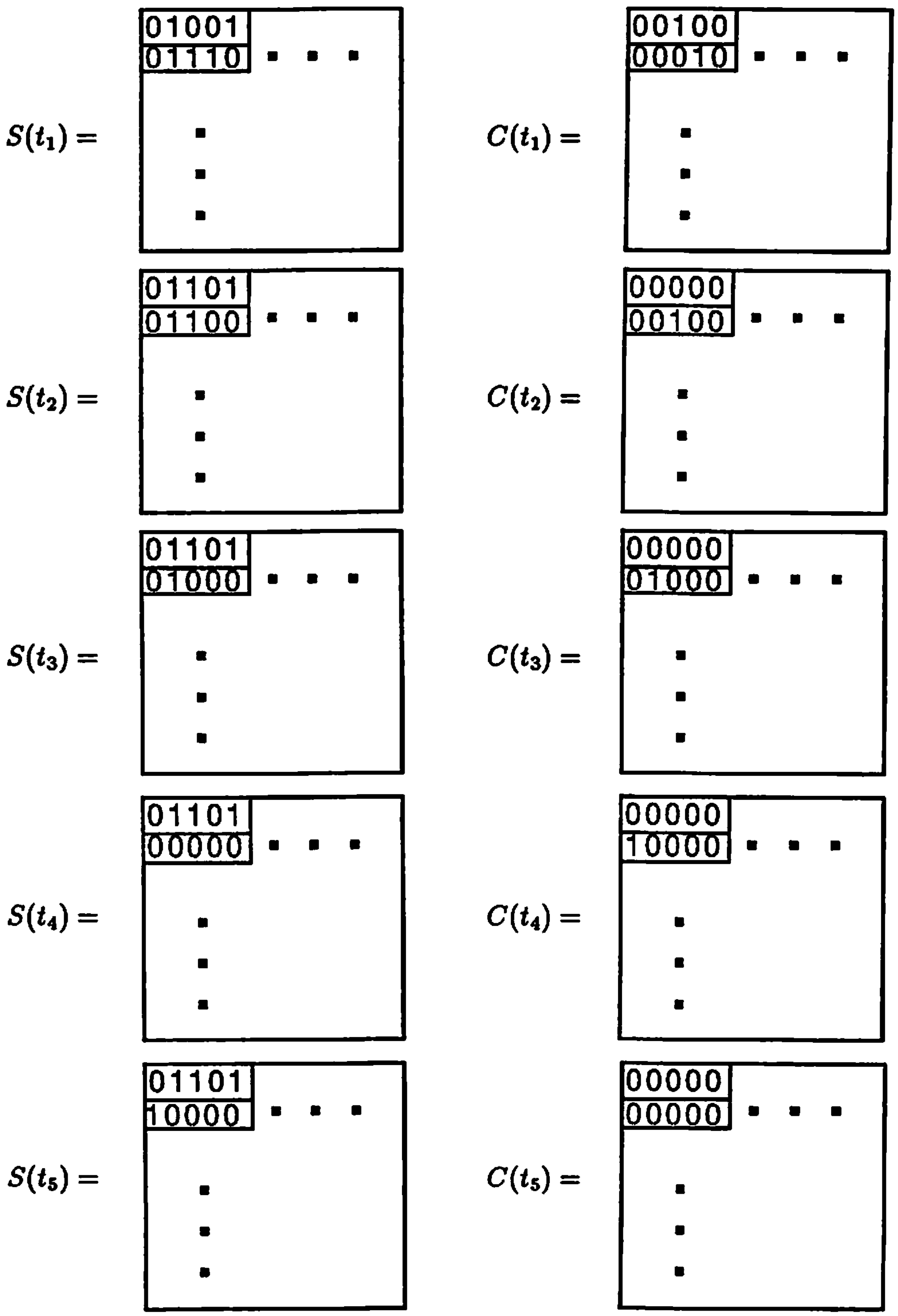


Figure 8.3: Parallel addition of binary row-coded numbers (II): The procedure for parallel addition $X+R$ where X and R are shown in Fig. 8.2, $S(t_5) = S = X+R$ and $C(t_5) = \phi$.

In fact, BIA can be used to devise a parallel form of a conditional-sum adder or carry-lookahead adder for further extracting additional parallelism, and the execution time of this addition can be reduced to $O(log_2 k)$. Obviously, there exists a tradeoff between execution time and hardware complexity. This paper concentrates only on some simple algorithms .

8.1.2 Subtraction of Binary Row-coded Numbers

Let the output of the parallel subtraction be $D = X - R$ (e.g. Fig. 8.4(a)-(c)). To realize it, we first consider the serial binary subtraction of 2 binary numbers $d_i = x_i - r_i$. The procedure in the least significant bits $x_{i(o)}$ and $r_{i(o)}$ of binary subtraction generates a difference bit $d_{i(o)}$ and a borrow bit $b_{i(o)}$. The boolean logic equations for subtracting the two least significant bits (half-subtractor) are

- difference bit: $s_{i(o)} = x_{i(o)}$ XOR $r_{i(o)}$,
- borrow bit: $c_{i(o)} = \overline{x}_{i(o)}$ AND $r_{i(o)}$.

Now, applying the corresponding parallel operations, and shifting the set of borrow bits by a dilation $\oplus$, we can implement the parallel subtraction as follows:

1. Define the initial states of images of difference bits and borrow bits (called difference-bit image and borrow-bit image) at time t_o as :

$$D(t_0) = X, \quad B(t_0) = R. \tag{8.5}$$

2. The recursive relation between the states of the difference-bit image and borrow-bit image at two adjacent time intervals is:

$$D(t_{i+1}) = D(t_i) \bigtriangleup B(t_i) = \overline{\overline{D(t_i)} \cup B(t_i)} \cup \overline{D(t_i) \cup \overline{B(t_i)}} \tag{8.6}$$

$$B(t_{i+1}) = (\overline{D(t_i)} \cap B(t_i)) \oplus A^{-1} = \overline{D(t_i) \cup \overline{B(t_i)}} \oplus A^{-1} \tag{8.7}$$

where $i = 0, 1, 2, ..., k+1$, and the elementary image A^{-1} is used to shift the borrow-bit image one bit to the left for the next iteration.

3. After a maximum of $k+1$ iterations, the difference-bit image is the result and the borrow-bit image becomes the null image ϕ:

$$D(t_{k+1}) = X - R, \quad B(t_{k+1}) = \phi. \tag{8.8}$$

This procedure is illustrated in Fig. 8.4(d).

The result of parallel subtraction of binary numbers with a maximum k-bit word size is obtained after $k+1$ iterations. The DOCIP architecture can realize this by the following program (instructions):

- Assume start with X in M_1 $(= D(t_0))$ and R in M_2 $(= B(t_0))$.

- First to k^{th} iterations:

  1. $M_1 \cup \overline{M_2} \rightarrow M_3$ $(= D(t_i) \cup \overline{B(t_i)})$
  2. $\overline{M_1} \cup M_2 \rightarrow M_1$ $(= \overline{D(t_i)} \cup B(t_i))$
  3. $\overline{M_1} \cup \overline{M_2} \rightarrow M_1$ $(= D(t_{i+1}))$
  4. $\overline{M_3} \oplus A^{-1} \rightarrow M_2$ $(= B(t_{i+1}))$

  where $i = 0, 1, 2, ..., k-1$.

- $(k+1)^{th}$ iteration:

  1. $M_1 \cup \overline{M_2} \rightarrow M_3$ $(= D(t_k) \cup \overline{B(t_k)})$
  2. $\overline{M_1} \cup M_2 \rightarrow M_1$ $(= \overline{D(t_k)} \cup B(t_k))$
  3. $\overline{M_1} \cup \overline{M_2} \rightarrow M_1$ $(= D(t_{k+1}) = X - R)$

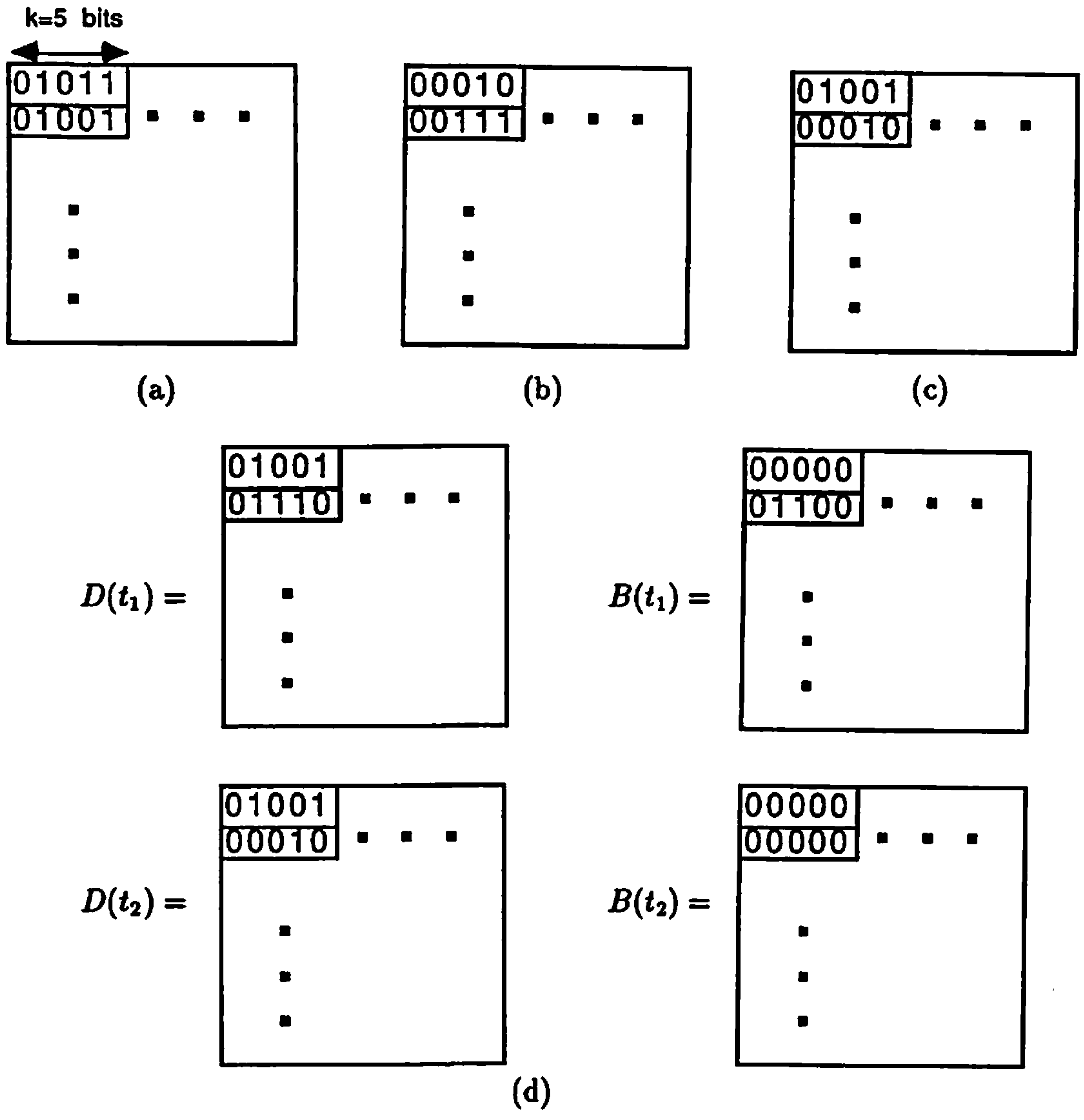


Figure 8.4: Parallel subtraction of binary row-coded numbers. (a): An image X of operands. (b): An image R of other operands. (c): The output $X - R$. (d): The procedure for parallel subtraction $X - R$, $D(t_2) = X - R$ and $B(t_2) = \phi$.

The total number of clock cycles in the DOCIP to complete this subtraction process is

$$t(k) \leq 4k + 3 = O(k).$$

8.1.3 Multiplication of Binary Row-coded Numbers

Using the representation illustrated in Fig. 8.1, we define a parallel (matrix-constant) multiplication of an image set of binary numbers and one single binary number $X \cdot R_r$, and parallel (element-element) multiplication of two image sets of binary numbers $X \times R$.

I. Matrix-Constant Multiplication $X \cdot R_r$

Consider an image X (e.g. Fig. 8.5(a)) comprising N^2/k numbers $x_i, i = 1, 2, ..., N^2/k$, and a reference image R_r (e.g. Fig. 8.5(b)) comprising only one single k-bit binary number $r = (r_{(k-1)}r_{(k-2)}...r_{(0)})_2$. The output of the parallel multiplication is $X \cdot R_r$ (Fig. 8.5(c)).

To realize it, we first consider the serial multiplication of two binary numbers that is the sum of the shifted versions of the multiplier or the multiplicand. Then, by applying the corresponding parallel operations and parallel shifting by a dilation $\oplus$, we can implement this parallel multiplication by the equation

$$X \cdot R_r = \sum_{l, \forall r_{(l)}=1} X \oplus A^{-l} \tag{8.9}$$

where the sum notation $\sum$ refers to a sequence of parallel additions and the parallel addition $+$ is defined in Subsection 8.1.1.

The DOCIP takes $O(k^2)$ clock cycles for implementing this matrix-constant multiplication. Its procedure involves:

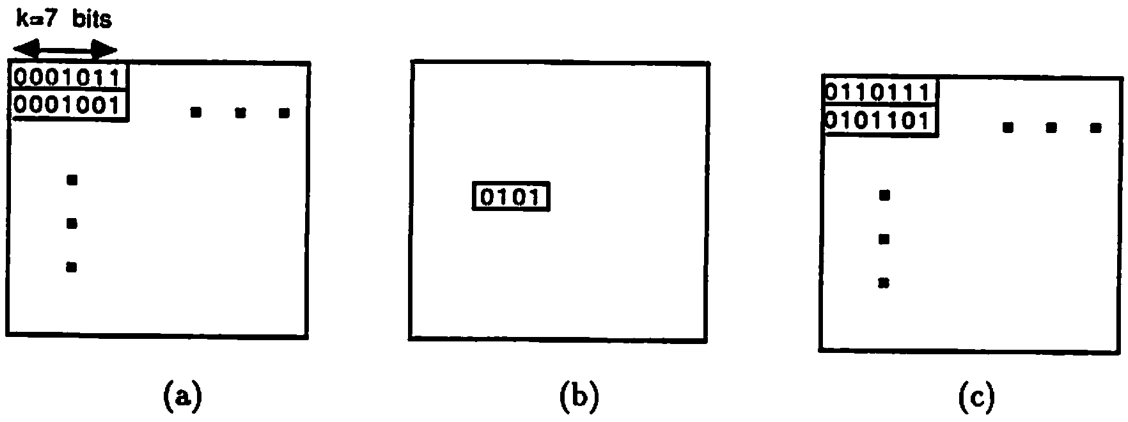


Figure 8.5: Parallel (matrix-constant) multiplication of binary row-coded numbers. (a): An image X of operands. (b): An image R_r containing only a single number. (c): The output $X \cdot R_r$.

1. Generating the term $X \oplus A^{-l}$:

   - The DOCIP-array requires at most $l \leq k-1 = O(k)$ clock cycles, because

$$\begin{aligned} A^{-l} &= (A^{-1})^l \\ &\equiv \underbrace{A^{-1} \oplus A^{-1} \oplus ... \oplus A^{-1}}_{l} \\ X \oplus A^{-l} &= (...((X \oplus \underbrace{A^{-1}) \oplus A^{-1}) \oplus ... \oplus A^{-1}}_{l}). \end{aligned} \tag{8.10}$$

   - The DOCIP-hypercube requires at most $log_2 l \leq log_2(k-1) = O(log_2 k)$ clock cycles, because we can rewrite l as a binary number $l = (a_{(\lfloor log_2 l \rfloor)} ... a_{(1)} a_{(0)})_2$, and we have

$$\begin{aligned} A^{-l} &= \prod_{j=0}^{\lfloor log_2 l \rfloor} A^{-a_{(j)} \cdot 2^j} \\ &\equiv A^{-a_{(0)}} \oplus A^{-a_{(1)} \cdot 2^1} \oplus ... \oplus A^{-a_{(\lfloor log_2 l \rfloor)} \cdot 2^{\lfloor log_2 l \rfloor}} \\ X \oplus A^{-l} &= (...((X \oplus A^{-a_{(0)}}) \oplus A^{-a_{(1)} \cdot 2^1}) \oplus ... \\ &\quad \oplus A^{-a_{(\lfloor log_2 l \rfloor)} \cdot 2^{\lfloor log_2 l \rfloor}}) \end{aligned} \tag{8.11}$$

where $\lfloor log_2 l \rfloor$ is the greatest integer less than or equal to $log_2 l$, and each dilation with $A^{-a(j)\cdot 2^j}$ can be implemented in the DOCIP-hypercube in one single clock cycle.

- The total time delay for generating all required $X \oplus A^{-l}$, $0 \leq l \leq k-1$, is bounded by $O(k)$ for both the DOCIP-array and the DOCIP-hypercube. Since

$$X \oplus A^{-l} = (X \oplus A^{-(l-1)}) \oplus A^{-1}, \tag{8.12}$$

we can generate the new term $X \oplus A^{-l}$ by simply deriving it from the previous term $X \oplus A^{-(l-1)}$ without starting from the original X. The total generating time is then dominated by the number of terms $X \oplus A^{-l}$ which is at most $O(k)$.

2. Implementing the summation $\sum_{l, \forall r_{(l)}=1} X \oplus A^{-l}$:

- The DOCIPs require at most $k-1 = O(k)$ parallel additions to implement this summation, and each parallel addition requires at most $k+1 = O(k)$ iterations (as shown in subsection 3.1). Since it takes $O(k)$ time for generating all the terms $X \oplus A^{-l}$, the total execution time of the DOCIPs for this matrix-constant multiplication of k-bit binary numbers is

$$O(k) \times O(k) + O(k) = O(k^2).$$

From the example shown in Fig. 8.5, $R_r = I \cup A^{-2}$ contains only a single number $r = (0101)_2 = 5$, and the DOCIP can implement this matrix-constant multiplication $X \cdot R_r$ as follows:

Assume start with X in M_1 $(= X \oplus I)$.

1. $M_1 \oplus A^{-2} \rightarrow M_2$ $(= X \oplus A^{-2})$

2. The instructions of the parallel addition are performed as shown in subsection 3.1:

$M_1 + M_2 \rightarrow$ Out $(= X \cdot R_r)$.

II. Element-Element Multiplication $X \times R$

Consider an image X (e.g. Fig. 8.6(a)) comprising N^2/k numbers $x_i, i = 1, 2, ..., N^2/k$, and an image R (e.g. Fig. 8.6(b)) comprising N^2/k numbers $r_i, i = 1, 2, ..., N^2/k$. The output of the element-element parallel multiplication is $X \times R$ (Fig. 8.6(c)).

Because the multiplication of two binary numbers is the sum of the shifted versions of the multiplier or the multiplicand, applying the corresponding parallel operations, we can implement this parallel multiplication by the equation

$$\begin{aligned} X \times R &= \textstyle\sum_{l=0}^{k-1} (X \oplus A^{-l}) \cap ((R \cap (M \oplus A^{-l})) \oplus \cup_{j=0}^{k-l-1} A^{-j}) \\ &= \textstyle\sum_{l=0}^{k-1} \overline{\overline{\overline{X \oplus A^{-l}} \cup \overline{\overline{\overline{R} \cup \overline{M \oplus A^{-l}}}} \oplus \cup_{j=0}^{k-l-1} A^{-j}}} \end{aligned} \tag{8.13}$$

where the mask M (Fig. 8.6(d)) is used to extract the l^{th} bit (where the 0^{th} bit is least significant and the $(k-1)^{th}$ bit is most significant). The DOCIPs can implement this element-element multiplication by the procedure

1. Generate $X \oplus A^{-l}$ and $\overline{\overline{R} \cup \overline{M \oplus A^{-l}}}$:

   - Using an argument similar to that in subsection I above, the DOCIP-array takes $O(k)$ time and the DOCIP-hypercube takes $O(log_2 k)$ time.

2. Generate $\overline{\overline{\overline{R} \cup \overline{M \oplus A^{-l}}}} \oplus \cup_{j=0}^{k-l-1} A^{-j}$:

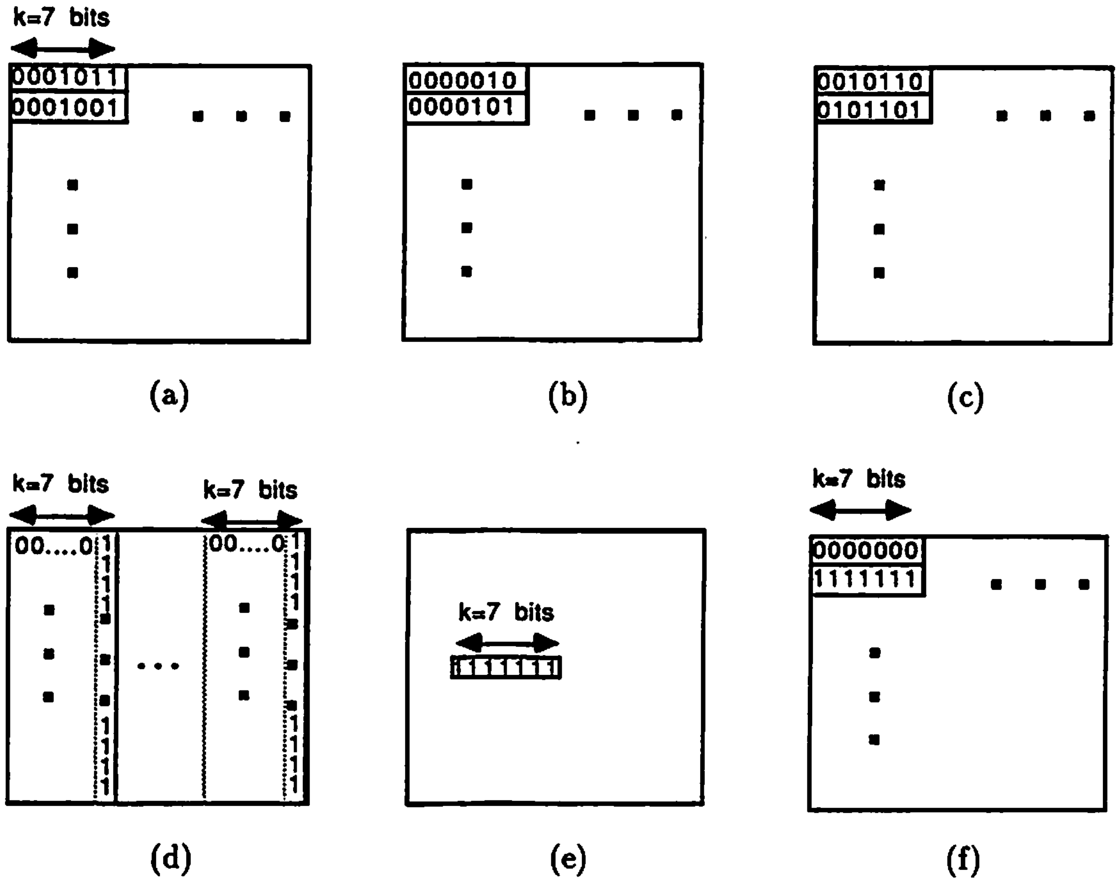


Figure 8.6: Parallel (element-element) multiplication of binary row-coded numbers. (a): An image X of operands. (b): An image R of other operands. (c): The output of the parallel (matrix-matrix) multiplication $X \times R$. (c): The output $X \times R$. (d): The mask M. (e): The image $\bigcup_{j=0}^{k-1} A^{-j}$. (f): The image $(R \cap M) \oplus \bigcup_{j=0}^{k-1} A^{-j}$.

- The DOCIP-array takes $O(k)$ time, because

$$\bigcup_{j=0}^{k-l-1} A^{-j} = (\bigcup_{j=0}^{1} A^{-j})^{k-l-1} \equiv \underbrace{(\bigcup_{j=0}^{1} A^{-j}) \oplus (\bigcup_{j=0}^{1} A^{-j}) \oplus \ldots \oplus (\bigcup_{j=0}^{1} A^{-j})}_{k-l-1}, \tag{8.14}$$

$l \geq 0$, and each dilation by a term in parentheses executes in one clock cycle.

- The DOCIP-hypercube takes $O(log_2 k)$ time, since

$$\bigcup_{j=0}^{k-l-1} A^{-j} = \prod_{n=0}^{\lfloor log_2(k-l-1) \rfloor} (\bigcup_{j=0}^{n} A^{-a_{(j)} \cdot 2^j}) \tag{8.15}$$

where $k - l - 1 = (a_{(\lfloor log_2(k-l-1) \rfloor)} \ldots a_{(1)} a_{(0)})_2$, and again each dilation by the term in parentheses executes in one clock cycle.

- It takes $O(k)$ time for the DOCIP-array and $O(log_2 k)$ for the DOCIP-hypercube to generate the term

$$\overline{\overline{(X \oplus A^{-l})} \cup \overline{(\overline{(\overline{\overline{R}} \cup \overline{(M \oplus A^{-l})})} \oplus \cup_{j=0}^{k-l-1} A^{-j})}}.$$

3. Implementing the summation $\sum_{l=0}^{k-1} \overline{\overline{X \oplus A^{-l}} \cup \overline{\overline{\overline{\overline{R}} \cup \overline{M \oplus A^{-l}}} \oplus \cup_{j=0}^{k-l-1} A^{-j}}}$:

   - The summation requires at most $(k-1)$ addition operations, and each addition operation takes $O(k)$ time on the DOCIP system. We also require $O(k)$ time for the DOCIP-array and $O(log_2 k)$ time for the DOCIP-hypercube to generate each operand of the addition. Thus, for this element-element multiplication of k-bit binary numbers, the total computation time is $O(k^3)$ for the DOCIP-array and $O(k^2 log_2 k)$ for the DOCIP-hypercube.

Multiplication requires more than three memories. This can be accommodated by either building more memory into the DOCIP machine or by swapping intermediate results into and out of an external memory. In the latter case we assume the external memory can be loaded and unloaded with one image in a single time step. In section 4, binary stack-coded arithmetic also requires more than three memories; we'll make the same assumptions on the use of an external memory.

For binary column-coded arithmetic, a number is encoded in a part of a column of an image as in Fig. 8.7. All the algorithms derived in this section can be also applied to binary column-coded numbers except that we replace the elementary image A^{-1} by a different elementary image B for shifting the carry-bit image or borrow-bit image in the vertical direction.

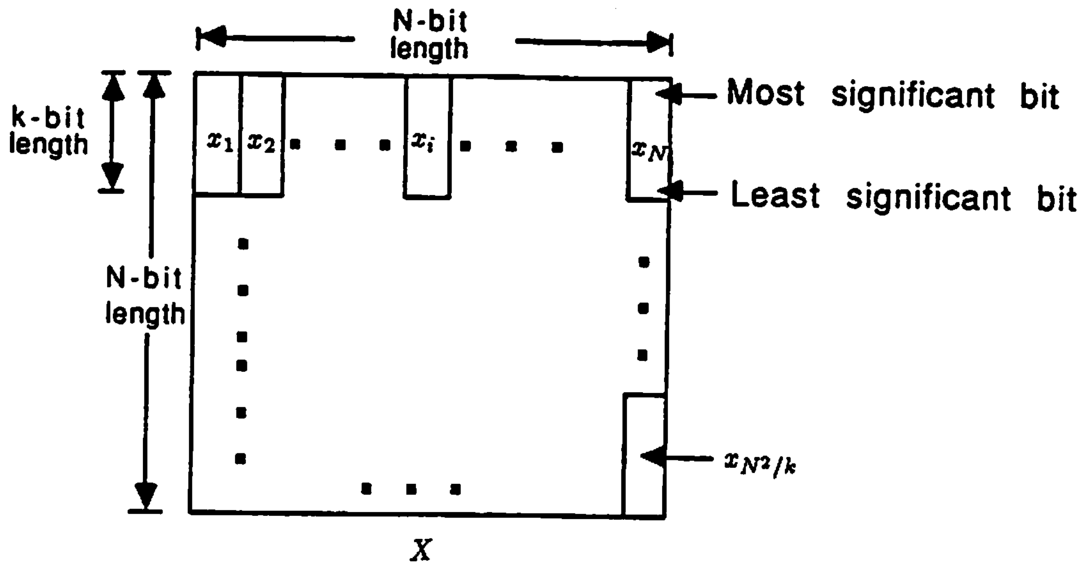


Figure 8.7: Binary column-coded numbers.

8.2 Binary Stack-coded Arithmetic

In this case, a number is encoded in a stack of k image planes with the least significant bit in the first plane, next least significant bit in the second plane, etc. (Fig. 8.8).

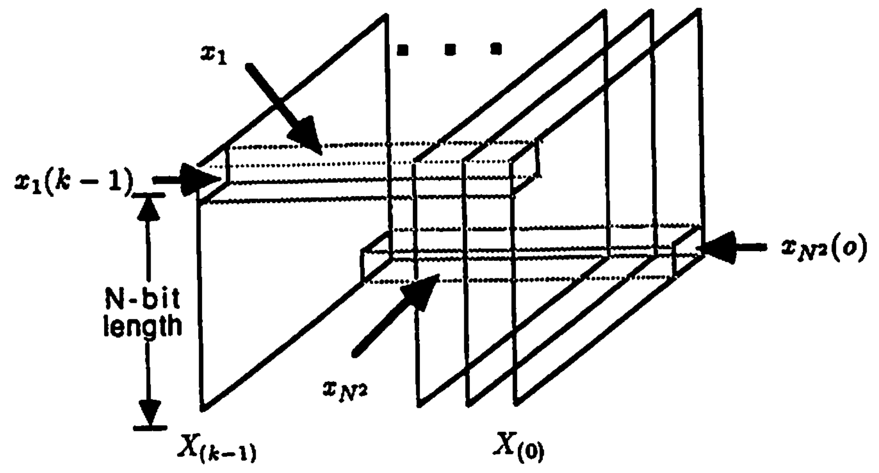


Figure 8.8: Binary stack-coded numbers. $x_i(m)$ represents the m^{th} bit of the i^{th} number in the image plane. $X_{(0)}$ represents the image plane of least significant bits and $X_{(k-1)}$ represents the image plane of most significant bits.

We assume all numbers including the results of arithmetic operations can be represented in k bits, so that k images, each with $N \times N$ bits, contain N^2 binary numbers. Here, we describe parallel addition, subtraction and multiplication by BIA expressions.

8.2.1 Addition of Binary Stack-coded Numbers

Using the representation illustrated in Fig. 8.8, we consider the parallel addition of two sequences of images of binary numbers. Assume a sequence of images $X = (X_{(k-1)}, X_{(k-2)}, ..., X_{(0)})$ (e.g. Fig. 8.9(a)) storing N^2 binary numbers $x_i, i = 1, 2, ..., N^2$, and a sequence of images $R = (R_{(k-1)}, R_{(k-2)}, ..., R_{(0)})$ (e.g. Fig. 8.9(b)) storing N^2 numbers $r_i, i = 1, 2, ..., N^2$. Then the output of the parallel addition is $X + R = S = (S_{(k)}, S_{(k-1)}, ..., S_{(0)})$ as shown in Fig. 8.9(c).

To realize this addition using our three fundamental operations, we implement an array of full adders as described by the equations

1. The least significant bit planes of sum bits and carry bits are given by:

$$S_{(0)} = X_{(0)} \bigtriangleup R_{(0)} = \overline{\overline{X_{(0)}} \cup R_{(0)}} \cup \overline{X_{(0)} \cup \overline{R_{(0)}}} \tag{8.16}$$

$$C_{(1)} = X_{(0)} \cap R_{(0)} = \overline{\overline{X_{(0)}} \cup \overline{R_{(0)}}} \tag{8.17}$$

2. The recursive relations:

$$\begin{aligned} S_{(i)} &= X_{(i)} \bigtriangleup R_{(i)} \bigtriangleup C_{(i)} \\ &= \overline{(\overline{X_{(i)}} \cap \overline{R_{(i)}} \cap \overline{C_{(i)}}) \cup (\overline{X_{(i)}} \cap R_{(i)} \cap C_{(i)}) \cup} \\ &\quad \overline{(X_{(i)} \cap \overline{R_{(i)}} \cap C_{(i)}) \cup (X_{(i)} \cap R_{(i)} \cap \overline{C_{(i)}})} \\ &= \overline{\overline{(X_{(i)} \cup R_{(i)} \cup C_{(i)})} \cup \overline{(X_{(i)} \cup \overline{R_{(i)}} \cup \overline{C_{(i)}})} \cup} \\ &\quad \overline{\overline{\overline{(\overline{X_{(i)}} \cup R_{(i)} \cup \overline{C_{(i)}})} \cup \overline{(\overline{X_{(i)}} \cup \overline{R_{(i)}} \cup C_{(i)})}}} \end{aligned} \tag{8.18}$$

$$\begin{aligned} C_{(i+1)} &= (X_{(i)} \cap R_{(i)}) \cup (X_{(i)} \cap C_{(i)}) \cup (R_{(i)} \cap C_{(i)}) \\ &= \overline{(\overline{X_{(i)}} \cup \overline{R_{(i)}})} \cup \overline{(\overline{X_{(i)}} \cup \overline{C_{(i)}})} \cup \overline{(\overline{R_{(i)}} \cup \overline{C_{(i)}})} \end{aligned} \tag{8.19}$$

where $i = 0, 1, 2, ..., k - 1$.

3. The final solution is:

$$X + R = S = (S_{(k)}, S_{(k-1)}, ..., S_{(0)}). \tag{8.20}$$

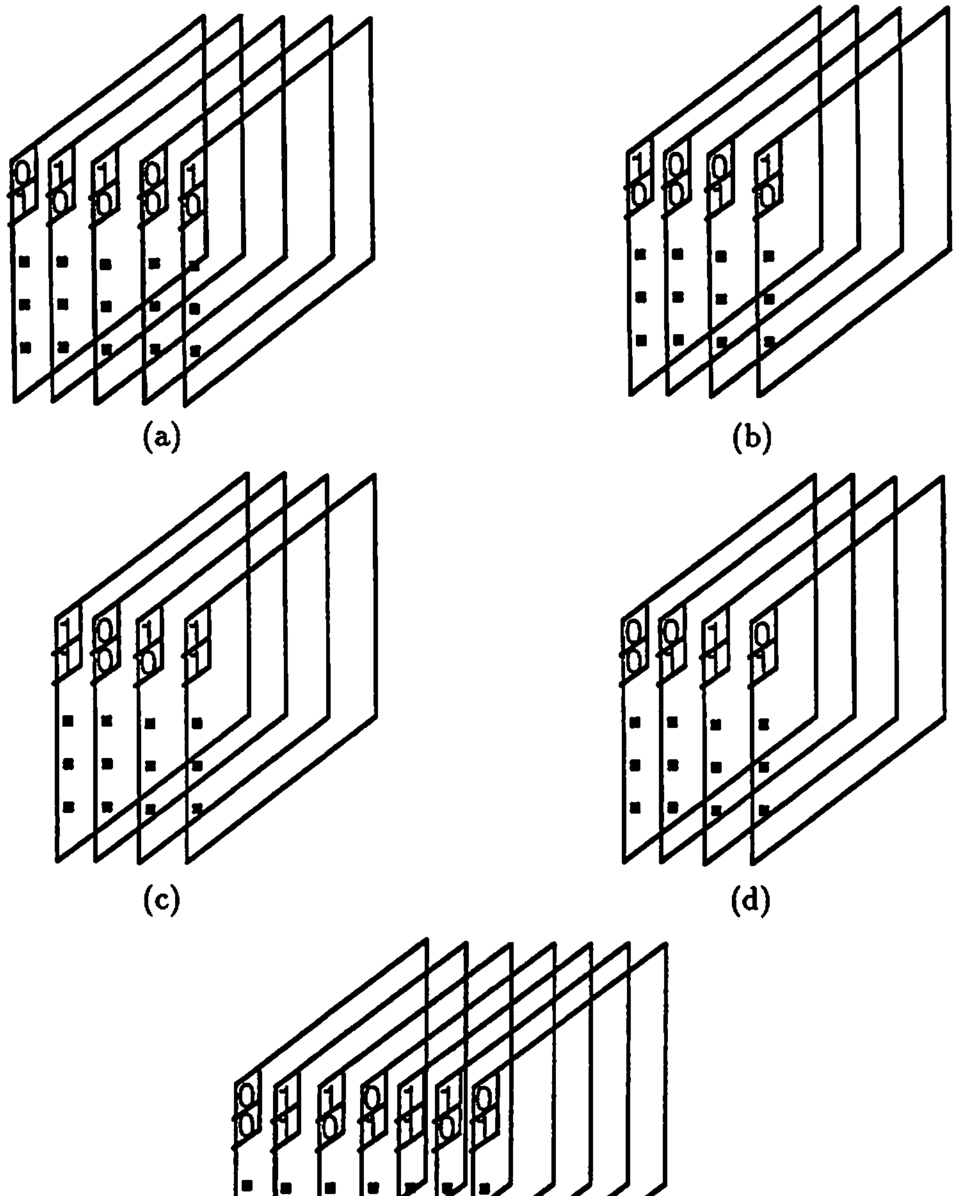


Figure 8.9: Parallel arithmetic with binary stack-coded numbers. (a): A sequence of images $X = (X_{(3)}, X_{(2)}, X_{(1)}, X_{(0)})$. (b): A sequence of images $R = (R_{(3)}, R_{(2)}, R_{(1)}, R_{(0)})$. (c): The sum $X + R = (S_{(4)}, S_{(3)}, S_{(2)}, S_{(1)}, S_{(0)})$. (d): The difference $D = X - R = (D_{(3)}, D_{(2)}, D_{(1)}, D_{(0)})$. (e): The product $M = X \times R = (M_{(7)}, M_{(6)}, ..., M_{(0)})$.

where $S_{(k)} = C_{(k)}$ because $X_{(k)} = R_{(k)} = \phi$.

This algorithm can be implemented in the DOCIP architecture by the program (DOCIP instructions):

- Assume start with $X_{(0)}$ stored in M_1 and $R_{(0)}$ stored in M_2.

- Calculate $S_{(0)}$ and $\overline{C_{(1)}}$:

  1. $\overline{M_1} \cup \overline{M_2} \rightarrow M_3$ & Out $(= \overline{C_{(1)}})$
  2. $\overline{M_1} \cup M_2 \rightarrow M_1$ $(= \overline{X_{(0)}} \cup R_{(0)})$
  3. $\overline{M_1} \cup \overline{M_2} \cup \overline{M_3} \rightarrow M_2$ $(= X_{(0)} \cup \overline{R_{(0)}})$
  4. $\overline{M_1} \cup \overline{M_2} \rightarrow$ Out $(= S_{(0)})$

- Calculate $S_{(1)}$ and $C_{(2)}$:

  1. $X_{(1)} \rightarrow M_1$
  2. $\overline{M_1} \cup \overline{M_3} \rightarrow M_2$ $(= \overline{X_{(1)}} \cup C_{(1)})$
  3. $M_1 \cup M_3 \rightarrow M_1$ $(= X_{(1)} \cup \overline{C_{(1)}})$
  4. $\overline{M_1} \cup \overline{M_2} \rightarrow M_1$ $(= X_{(1)} \triangle C_{(1)})$
  5. $R_{(1)} \rightarrow M_2$
  6. $\overline{M_1} \cup M_2 \rightarrow M_3$
  7. $M_1 \cup \overline{M_2} \rightarrow M_2$
  8. $\overline{M_2} \cup \overline{M_3} \rightarrow$ Out $(= S_{(1)})$
  9. $X_{(1)} \rightarrow M_1$
  10. $R_{(1)} \rightarrow M_2$
  11. $\overline{M_1} \cup \overline{M_2} \rightarrow M_3$
  12. $\overline{C_{(1)}} \rightarrow M_1$
  13. $M_1 \cup \overline{M_2} \rightarrow M_2$
  14. $\overline{M_2} \cup \overline{M_3} \rightarrow M_3$

15. $X_{(1)} \rightarrow M_2$

16. $M_1 \cup \overline{M_2} \rightarrow M_2$

17. $\overline{M_2} \cup M_3 \rightarrow M_3$ & Out $(= C_{(2)})$

- Calculate $S_{(2)}$ to $S_{(k-1)}$ and $C_{(3)}$ to $C_{(k)}$:

  Use the same instructions for calculating $S_{(1)}$ and $C_{(2)}$ except that $X_{(1)}$ and $R_{(1)}$ (and $S_{(1)}$ and $C_{(2)}$) are replaced by $X_{(i)}$ and $R_{(i)}$ (and $S_{(i)}$ and $C_{(i+1)}$) in each iteration, and in the beginning of an iteration the memory M_3 stores $C_{(i)}$ instead of $\overline{C_{(1)}}$, $i = 2, 3, .., k$.

The complete execution of this operation in the DOCIP requires

$$t(k) \leq 17(k-1) + 4 = 17k - 13 = O(k).$$

clock cycles. Additional parallelism could be extracted to further reduce the execution time by utilizing carry-lookahead techniques or by optimizing the above program.

8.2.2 Subtraction of Binary Stack-coded Numbers

Let the result of the parallel subtraction be $X - R = D = (D_{(k-1)}, D_{(k-2)}, ..., D_{(0)})$ (e.g. Fig. 8.9(d)). To realize it using the 3 fundamental operations, we consider a serial full-subtractor. Applying the corresponding parallel operations, we can implement this parallel subtraction by the equations

1. The least significant bit planes of difference bits and borrow bits:

$$D_{(0)} = X_{(0)} \triangle R_{(0)} = \overline{\overline{X_{(0)}} \cup R_{(0)}} \cup \overline{\overline{R_{(0)}} \cup X_{(0)}} \qquad (8.21)$$

$$B_{(1)} = \overline{X_{(0)}} \cap R_{(0)} = \overline{X_{(0)} \cup \overline{R_{(0)}}} \qquad (8.22)$$

2. The recursive relations:

$$\begin{aligned} D_{(i)} &= (\overline{X_{(i)}} \cap \overline{R_{(i)}} \cap B_{(i)}) \cup (\overline{X_{(i)}} \cap R_{(i)} \cap \overline{B_{(i)}}) \cup \\ &\quad (X_{(i)} \cap \overline{R_{(i)}} \cap \overline{B_{(i)}}) \cup (X_{(i)} \cap R_{(i)} \cap B_{(i)}) \\ &= \overline{(X_{(i)} \cup R_{(i)} \cup \overline{B_{(i)}})} \cup \overline{(X_{(i)} \cup \overline{R_{(i)}} \cup B_{(i)})} \cup \\ &\quad \overline{(\overline{X_{(i)}} \cup R_{(i)} \cup B_{(i)})} \cup \overline{(\overline{X_{(i)}} \cup \overline{R_{(i)}} \cup \overline{B_{(i)}})} \end{aligned} \tag{8.23}$$

$$\begin{aligned} B_{(i+1)} &= (\overline{X_{(i)}} \cap \overline{R_{(i)}} \cap B_{(i)}) \cup (\overline{X_{(i)}} \cap R_{(i)} \cap \overline{B_{(i)}}) \cup \\ &\quad (\overline{X_{(i)}} \cap R_{(i)} \cap B_{(i)}) \cup (X_{(i)} \cap R_{(i)} \cap B_{(i)}) \\ &= \overline{(X_{(i)} \cup R_{(i)} \cup \overline{B_{(i)}})} \cup \overline{(X_{(i)} \cup \overline{R_{(i)}} \cup B_{(i)})} \cup \\ &\quad \overline{(X_{(i)} \cup \overline{R_{(i)}} \cup \overline{B_{(i)}})} \cup \overline{(\overline{X_{(i)}} \cup \overline{R_{(i)}} \cup \overline{B_{(i)}})} \end{aligned} \tag{8.24}$$

where $i = 0, 1, 2, ..., k-1$.

3. The final solution:

$$X - R = D = (D_{(k-1)}, D_{(k-2)}, ..., D_{(0)}). \tag{8.25}$$

This algorithm can be implemented in the DOCIP architecture by the program (instructions):

- Assume start with $X_{(0)}$ in M_1 and $R_{(0)}$ in M_2.
- Calculate $D_{(0)}$ and $\overline{B_{(1)}}$:
  1. $M_1 \cup \overline{M_2} \rightarrow M_3$ & Out $(= \overline{B_{(1)}})$
  2. $\overline{M_1} \cup M_2 \rightarrow M_1$ $(= \overline{X_{(0)}} \cup R_{(0)})$
  3. $\overline{M_2} \cup \overline{M_3} \rightarrow$ Out $(= D_{(0)})$
- Calculate $D_{(1)}$ and $B_{(2)}$:
  1. $X_{(1)} \rightarrow M_1$
  2. $M_1 \cup M_3 \rightarrow M_2$

3. $\overline{M_1} \cup M_3 \rightarrow M_1$

4. $R_{(1)} \rightarrow M_3$

5. $M_2 \cup M_3 \rightarrow M_2$

6. $M_1 \cup M_3 \rightarrow M_3$

7. $\overline{M_2} \cup \overline{M_3} \rightarrow M_2$

8. $R_{(1)} \rightarrow M_3$

9. $M_1 \cup \overline{M_3} \rightarrow M_1$

10. $\overline{M_1} \cup M_2 \rightarrow M_2$

11. $X_{(1)} \rightarrow M_2$

12. $\overline{M_1} \cup M_3 \rightarrow M_3$

13. $B_{(1)} \rightarrow M_1$

14. $M_1 \cup M_3 \rightarrow M_3$

15. $M_2 \cup \overline{M_3} \rightarrow$ Out $(= D_{(1)})$

16. $X_{(1)} \rightarrow M_3$

17. $\overline{M_1} \cup M_3 \rightarrow M_1$

18. $R_{(1)} \rightarrow M_3$

19. $M_1 \cup \overline{M_3} \rightarrow M_1$

20. $\overline{M_1} \cup M_2 \rightarrow M_3$ & Out $(= B_{(2)})$

- Calculate $D_{(2)}$ to $D_{(k-1)}$ and $B_{(3)}$ to $B_{(k)}$:

  Use the same instructions for calculating $D_{(1)}$ and $B_{(2)}$ except that $X_{(1)}$ and $R_{(1)}$ (and $D_{(1)}$ and $B_{(2)}$) are replaced by $X_{(i)}$ and $R_{(i)}$ (and $D_{(i)}$ and $B_{(i+1)}$) in each iteration, and in the beginning of an iteration the memory M_3 stores $B_{(i)}$ instead of $\overline{B_{(1)}}$, $i = 2, 3, .., k$.

Therefore, the total execution time in the DOCIP to complete this parallel subtraction is

$$t(k) \leq 20(k-1) + 3 = 20k - 17 = O(k).$$

8.2.3 Multiplication of Binary Stack-coded Numbers

Let the result of the parallel multiplication be $X \times R = M = (M_{(2k-1)}, M_{(2k-2)}, ..., M_{(0)})$ (e.g. Fig. 8.9(e)). Since binary multiplication is equivalent to the addition of shifted versions of the multiplicand, applying the corresponding parallel operations, we can implement the parallel multiplication by the equations

$$P^{(0)} = (\underbrace{0, 0, ..., 0,}_{k} X_{(k-1)} \cap R_{(0)}, X_{(k-2)} \cap R_{(0)}, ..., X_{(0)} \cap R_{(0)}) \tag{8.26}$$

$$P^{(i)} = (\underbrace{0, 0, ..., 0,}_{k-i} X_{(k-1)} \cap R_{(i)}, X_{(k-2)} \cap R_{(i)}, ..., X_{(0)} \cap R_{(i)}, \underbrace{0, 0, ..., 0}_{i}) \tag{8.27}$$

$$X \times R = M = \sum_{i=0}^{k-1} P^{(i)} \equiv P^{(0)} + P^{(1)} + ... + P^{(k-1)} \tag{8.28}$$

where $i = 0, 1, ..., k-1$, and the addition $+$ is defined in subsection 4.1. Since this parallel multiplication requires at most $k-1$ additions, each addition takes $O(k)$ time for the DOCIP, and each $P^{(i)}$ can be generated in $O(k)$ time, the total execution time is $O(k^3)$.

8.3 Binary Symbol-coded Arithmetic

Symbolic substitution has the ability to solve any computable problem and performs many operations [Huang83, Brenner86a]. As shown in Section 5.1, we formalized symbolic substitution by BIA algebraic symbols and demonstrated that symbolic substitution rules are particular BIA image transformations and can be improved by BIA algebraic techniques and pattern recognition algorithms in many cases. Here we will give the BIA formal notations of binary symbol-coded (symbolic substitution) arithmetic. We show that the symbolic substitution implementation of some operations is relatively complicated to other implementations.

A bit in a binary number is encoded in symbolically as pixels of an image (Fig. 8.10). In this section, we primarily concentrate on single-pixel coding: a logic value (0 or 1) is represented by a single pixel (dark or bright) (Fig. 8.10(a)), as in the binary row and stack-coded number representations, but the operands of binary numbers x_i and r_i are stored in the same input image X as shown in Fig. 8.11(a). The expected output images of symbolic substitution for binary addition and binary subtraction are shown in Fig. 8.11(b)-(c).

To achieve these desired operations, the symbols associated with the operands are recognized and then replaced by new symbols associated with the results of the operation. Systems for implementing binary addition and subtraction are formalized and illustrated as examples of binary symbol-coded arithmetic below.

8.3.1 Addition of Binary Symbol-coded Numbers

This parallel binary addition (Fig. 8.12) can be implemented with four symbolic substitution rules [Huang83, Brenner86a] (Fig. 8.12(a)) . In the case of single-pixel coding, as we will show, Rule 1 is not necessary. The symbolic substitution system for single-pixel coding can be realized as

$$Y(t_0) = X \tag{8.29}$$

$$Y(t_{j+1}) = \bigcup_{i=1}^{4}((Y(t_j) \circledast R^{(i)}) \cap M) \oplus Q^{(i)} \tag{8.30}$$

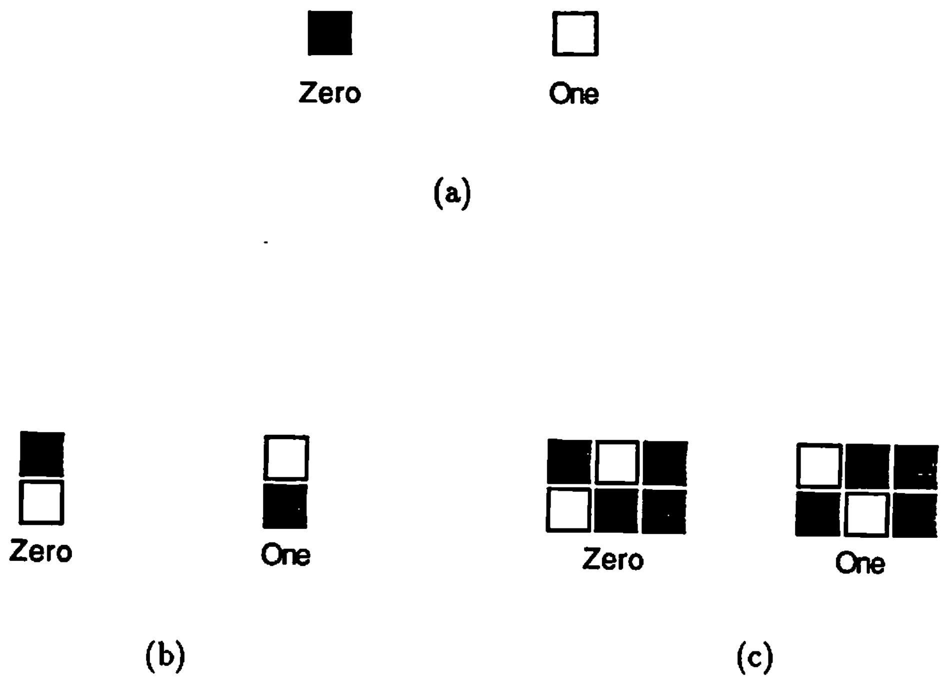
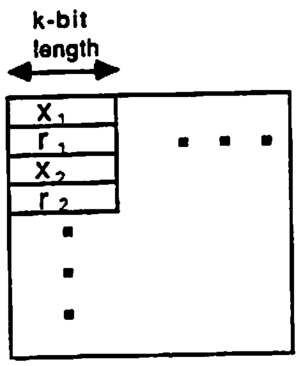
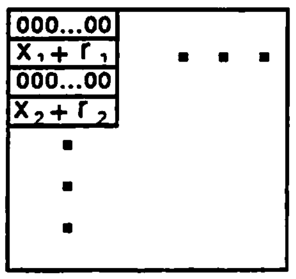


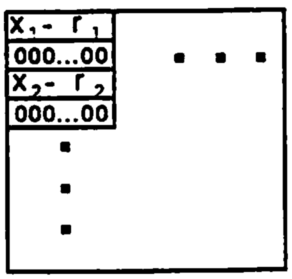
Figure 8.10: A bit encoded as a symbol. (a): The single-pixel coding of zero and one (a bit is a pixel). (b): The two-pixel (i.e. dual-rail) coding of zero and one (a bit is encoded as two pixels) [Huang83, Brenner86a]. (c): The six-pixel coding of zero and one (a bit with value zero or one is encoded as six pixels) [Jeon87].



(a)



(b)



(c)

Figure 8.11: Binary symbol-coded (symbolic substitution) arithmetic. (a): The input image X contains the operands x_i and r_i. (b): The output of parallel addition. (c): The output of parallel subtraction.

where $Y(t_{k+1})$ is the result, $j = 0, 1, 2, ..., k+1$, k is word size (i.e. the number of bits in a operand); $R^{(i)} = (R_1^{(i)}, R_2^{(i)})$ and $Q^{(i)}$ are shown in Fig. 8.12(b) and represented as

1. $R_1^{(1)} = \phi$, $R_2^{(1)} = \bigcup_{i=0}^{1} B^i$, $Q^{(1)} = \phi$,
2. $R_1^{(2)} = I$, $R_2^{(2)} = B$, $Q^{(2)} = I$,
3. $R_1^{(3)} = B$, $R_2^{(3)} = I$, $Q^{(3)} = I$,
4. $R_1^{(4)} = \bigcup_{i=0}^{1} B^i$, $R_2^{(4)} = \phi$, $Q^{(4)} = A^{-1}B$.

Here the null image ϕ and the elementary images are as defined in Chapter 3; the mask M (Fig. 8.12(c)), used for controlling the block search region, is the image corresponding to the coordinates of the origins (lower-lefter pixels) of the input symbols in the input image X. An example is given in Fig. 8.12(d).

Note that $Q^{(1)} = \phi$ implies

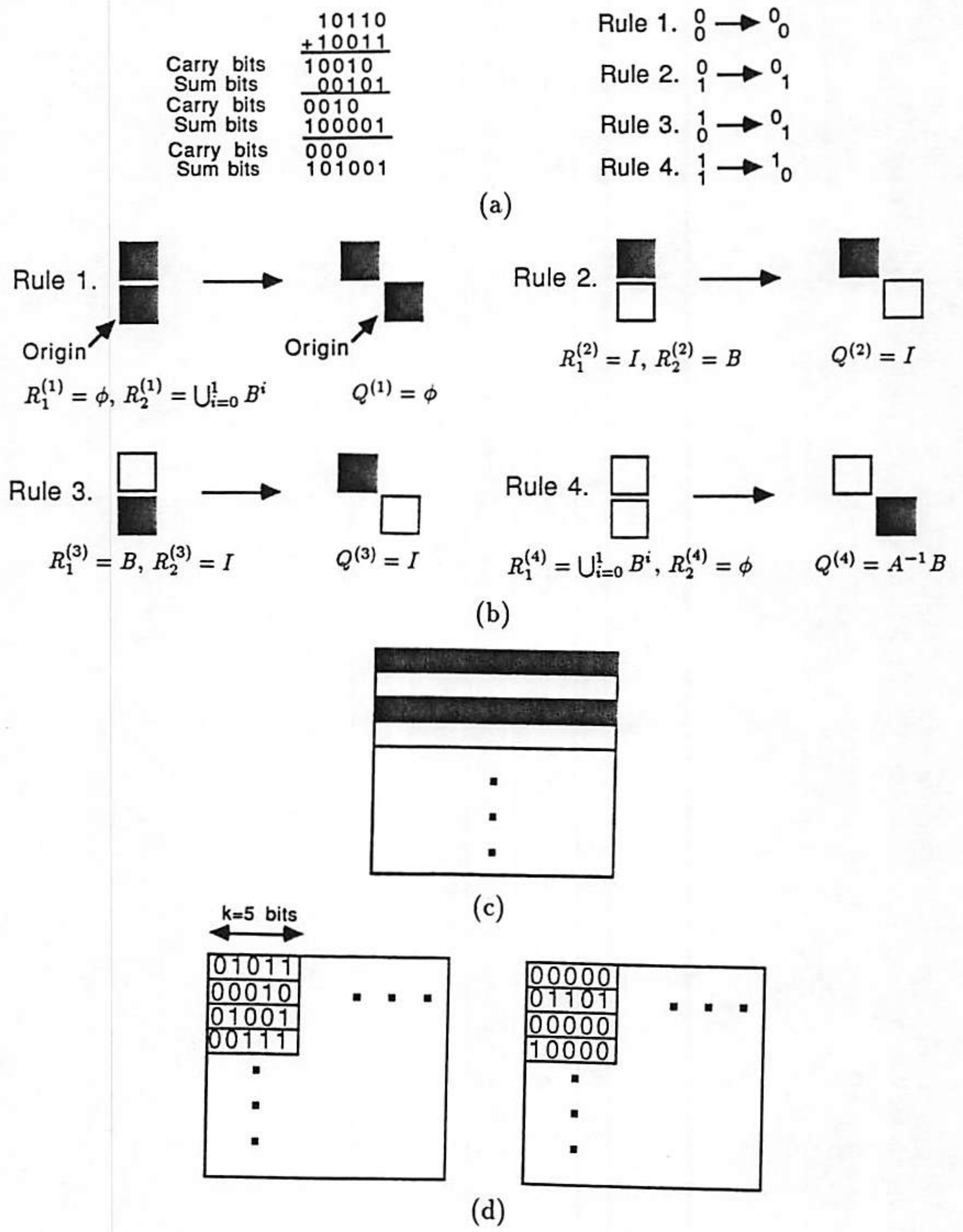
$$((Y(t_j) \circledast R^{(1)}) \cap M) \oplus Q^{(1)} = \phi, \tag{8.31}$$

so that

$$\begin{aligned} Y(t_{j+1}) &= \bigcup_{i=1}^{4}((Y(t_j) \circledast R^{(i)}) \cap M) \oplus Q^{(i)} \\ &= \bigcup_{i=2}^{4}((Y(t_j) \circledast R^{(i)}) \cap M) \oplus Q^{(i)} \\ &= \bigcup_{i=2}^{4}(\overline{(\overline{Y}(t_j) \oplus \check{R}_1{}^{(i)}) \cup (Y(t_j) \oplus \check{R}_2{}^{(i)})} \cap M) \oplus Q^{(i)}. \end{aligned} \tag{8.32}$$

Thus, for single-pixel coding of symbolic substitution, we can reduce the four rules of binary addition to only three rules. However, this reduction of complexity cannot be applied to two-pixel (i.e. dual-rail) or six-pixel coding.

When implemented on the DOCIP, this addition requires at most $k+1$ iterations, each iteration requiring two union operations of three results of symbolic



10110
+10011
Carry bits 10010
Sum bits 00101
Carry bits 0010
Sum bits 100001
Carry bits 000
Sum bits 101001
Rule 1. $\begin{smallmatrix}0\\0\end{smallmatrix} \rightarrow {}^{0}{}_{0}$
Rule 2. $\begin{smallmatrix}0\\1\end{smallmatrix} \rightarrow {}^{0}{}_{1}$
Rule 3. $\begin{smallmatrix}1\\0\end{smallmatrix} \rightarrow {}^{0}{}_{1}$
Rule 4. $\begin{smallmatrix}1\\1\end{smallmatrix} \rightarrow {}^{1}{}_{0}$
(a)
Rule 1.
Origin
Origin
$R_1^{(1)} = \phi,\ R_2^{(1)} = \bigcup_{i=0}^{1} B^i$
$Q^{(1)} = \phi$
Rule 2.
$R_1^{(2)} = I,\ R_2^{(2)} = B$
$Q^{(2)} = I$
Rule 3.
$R_1^{(3)} = B,\ R_2^{(3)} = I$
$Q^{(3)} = I$
Rule 4.
$R_1^{(4)} = \bigcup_{i=0}^{1} B^i,\ R_2^{(4)} = \phi$
$Q^{(4)} = A^{-1}B$
(b)
(c)
k=5 bits
01011
00010
01001
00111
00000
01101
00000
10000
(d)

Figure 8.12: Parallel addition of binary symbol-coded numbers. (a): Four symbolic substitution rules for addition. (b): Reference image pairs $R^{(i)}$ and reference images $Q^{(i)}$, $i = 1, 2, 3, 4$, used for addition. $Q^{(1)}$ is a null image, Rule 1 is not needed for this single-pixel coding. (c): The mask M. (d): An example of parallel addition of binary symbol-coded numbers.

substitution rules, and each rule is realized within five steps as shown in Subsection 7.4. Thus, the total execution time in the DOCIP is

$$t(k) \leq (3 \times 5 + 2)(k+1) = 17(k+1) = O(k).$$

When using 2 or 6 pixels to represent a logic value (Fig. 8.10(b)-(c)), we can formalize symbolic substitution addition as

- Two-pixel coding (Fig. 8.10(b)) [Huang83, Brenner86a]: we can implement a full recognition with only a background recognizer (or foreground recognizer)

  $$\begin{aligned} Y(t_{j+1}) &= \bigcup_{i=1}^{4}((Y(t_j) \circledast R^{(i)}) \cap M) \oplus Q^{(i)} \\ &= \bigcup_{i=1}^{4}(\overline{(\overline{Y}(t_j) \oplus \check{R}_1{}^{(i)}) \cup (Y(t_j) \oplus \check{R}_2{}^{(i)})} \cap M) \oplus Q^{(i)} \qquad (8.33) \\ &= \bigcup_{i=1}^{4}(\overline{Y(t_j) \oplus \check{R}_2{}^{(i)}} \cap M) \oplus Q^{(i)} \end{aligned}$$

  where $j = 0, 1, 2, ..., k$; $R^{(i)} = (R_1^{(i)}, R_2^{(i)})$ and $Q^{(i)}$ are shown in Fig. 8.13(a) and represented by elementary images as

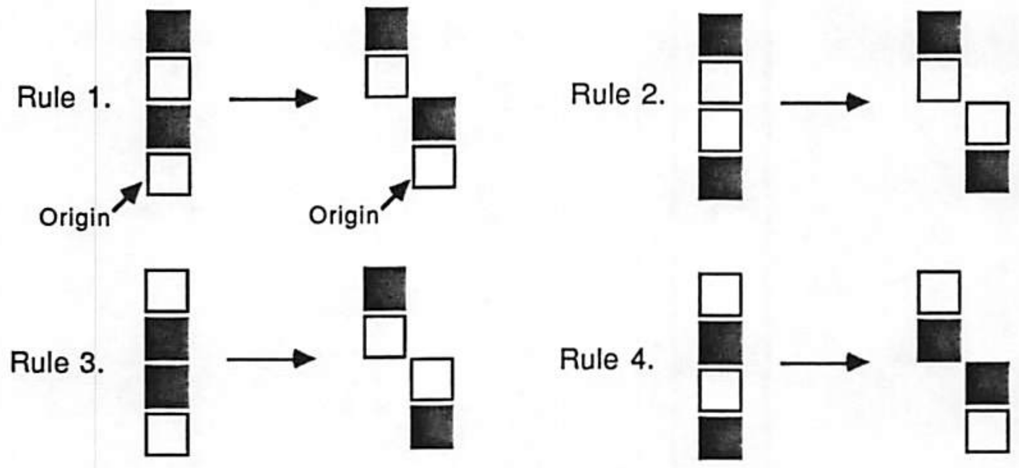
  1. $R_1^{(1)} = I \cup B^2$, $R_2^{(1)} = B \cup B^3$, $Q^{(1)} = I \cup A^{-1}B^2$,
  2. $R_1^{(2)} = \bigcup_{i=1}^{2} B^i$, $R_2^{(2)} = I \cup B^3$, $Q^{(2)} = B \cup A^{-1}B^2$,
  3. $R_1^{(3)} = I \cup B^3$, $R_2^{(3)} = \bigcup_{i=1}^{2} B^i$, $Q^{(3)} = B \cup A^{-1}B^2$,
  4. $R_1^{(4)} = B \cup B^3$, $R_2^{(4)} = I \cup B^2$, $Q^{(4)} = I \cup A^{-1}B^3$;

  and the mask M is shown in Fig. 8.13(b).

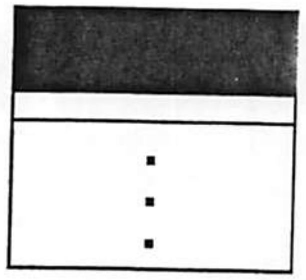
  Since

  $$(\overline{Y}(t_j) \oplus \check{R}_1{}^{(i)}) \cup (Y(t_j) \oplus \check{R}_2{}^{(i)}) = (\overline{Y}(t_j) \oplus \check{R}_1{}^{(i)}) = (Y(t_j) \oplus \check{R}_2{}^{(i)}), \quad (8.34)$$

  for the two-pixel coding, $R^{(i)}$ can be represented by only its foreground $R_1^{(i)}$ or background $R_2^{(i)}$. For implementation on the DOCIP, this algorithm



(a)



(b)

Figure 8.13: Symbolic substitution binary addition with two-pixel coding [Huang83, Brenner86a]. (a): Reference image pairs $R^{(i)}$ and reference images $Q^{(i)}$, $i = 1, 2, 3, 4$, used for addition (with two-pixel coding). (b): The mask M.

requires four rules, and each rule involves two dilations and one union or intersection. Because they may be not included in N_{array} or $N_{hypercube}$, each dilation of $R_2^{(i)}$ or $Q^{(i)}$ is implemented by 2-4 steps for the DOCIP-array8 and 1-2 steps for the DOCIP-hypercube8. The total execution time is bounded by $28(k+1)$ for the DOCIP-array8 and $18(k+1)$ for the DOCIP-hypercube8. Moreover, it requires more difficult two-pixel coding and doubles the device area.

- Six-pixel coding (Fig. 8.10(c)) [Jeon87]: the mask M is not needed and

$$Y(t_{j+1}) = \bigcup_{i=1}^{4}(Y(t_j) \circledast R^{(i)}) \oplus Q^{(i)} \tag{8.35}$$

where $j = 0, 1, 2, ..., k$, k is the word size; $R^{(i)} = (R_1^{(i)}, R_2^{(i)})$ and $Q^{(i)}$ are shown in Fig. 8.14 and are represented as

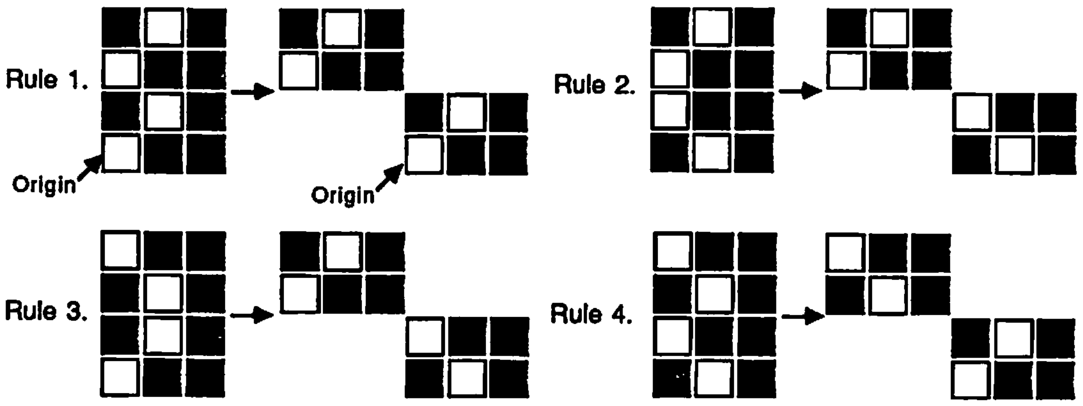


Figure 8.14: Symbolic substitution binary addition with encoding a bit as six pixels [Jeon87].

1. $R_1^{(1)} = I \cup AB \cup B^2 \cup AB^3$, $R_2^{(1)} = B \cup \cup B^3 \cup A \cup AB^2 \cup (\bigcup_{i=0}^{3} A^2B^i)$,
   $Q^{(1)} = A^{-3}B^2 \cup A^{-2}B^3 \cup I \cup AB$,

2. $R_1^{(2)} = (\bigcup_{i=1}^{2} B^i) \cup A \cup A^3$, $R_2^{(2)} = I \cup B^3 \cup (\bigcup_{i=1}^{2} AB^i) \cup (\bigcup_{i=0}^{3} A^2B^i)$, $Q^{(2)} = A^{-3}B^2 \cup A^{-2}B^3 \cup B \cup A$,
3. $R_1^{(3)} = I \cup B^3 \cup (\bigcup_{i=1}^{2} AB^i)$, $R_2^{(3)} = (\bigcup_{i=1}^{2} B^i) \cup A \cup A^3 \cup (\bigcup_{i=0}^{3} A^2B^i)$, $Q^{(3)} = A^{-3}B^2 \cup A^{-2}B^3 \cup B \cup A$,
4. $R_1^{(4)} = B \cup B^3 \cup A \cup AB^2$, $R_2^{(4)} = I \cup B^2 \cup AB \cup AB^3 \cup (\bigcup_{i=0}^{3} A^2B^i)$, $Q^{(4)} = A^{-3}B^3 \cup A^{-2}B^2 \cup I \cup AB$.

The six-pixel coding removes the need for the mask M, but requires more difficult encoding, more difficult implementation of the hit or miss transform by $R^{(i)}$ and dilation by $Q^{(i)}$, and six times the hardware area. Addition on the DOCIP-array or DOCIP-hypercube using six-pixel coding takes much more time (on the order of ten times) than single-pixel coding or two-pixel coding.

8.3.2 Subtraction of Binary Symbol-coded Numbers

Similar to addition, we generally use 4 symbolic substitution rules (Fig. 8.15(a)), but Rule 1 and Rule 4 are not necessary for single-pixel coding. The symbolic substitution system using single-pixel coding for binary subtraction can be realized as

$$Y(t_0) = X \tag{8.36}$$

$$\begin{aligned} Y(t_{j+1}) &= \bigcup_{i=1}^{4}((Y(t_j) \circledast R^{(i)}) \cap M) \oplus Q^{(i)} \\ &= \bigcup_{i=1}^{4}(\overline{(\overline{Y}(t_j) \oplus \check{R}_1{}^{(i)}) \cup (Y(t_j) \oplus \check{R}_2{}^{(i)})} \cap M) \oplus Q^{(i)} \qquad (8.37) \\ &= \bigcup_{i=2}^{3}(\overline{(\overline{Y}(t_j) \oplus \check{R}_1{}^{(i)}) \cup (Y(t_j) \oplus \check{R}_2{}^{(i)})} \cap M) \oplus Q^{(i)} \end{aligned}$$

where $Y(t_{k+1})$ is the result of the subtraction, $j = 0, 1, 2, ..., k$, k is word size (i.e. the number of bits in a operand); $R^{(i)} = (R_1^{(i)}, R_2^{(i)})$ and $Q^{(i)}$ are shown in Fig. 8.15(b) and represented as

1. $R_1^{(1)} = \phi$, $R_2^{(1)} = \bigcup_{i=0}^{1} B^{-i}$, $Q^{(1)} = \phi$,
2. $R_1^{(2)} = B^{-1}$, $R_2^{(2)} = I$, $Q^{(2)} = I \cup A^{-1}B^{-1}$,
3. $R_1^{(3)} = I$, $R_2^{(3)} = B^{-1}$, $Q^{(3)} = I$,
4. $R_1^{(4)} = \bigcup_{i=0}^{1} B^{-i}$, $R_2^{(4)} = \phi$, $Q^{(4)} = \phi$

where the null image ϕ and the elementary images are as defined in Chapter 3; and the mask M (Fig. 8.15(c)) is a shifting of the mask for binary addition. Because $Q^{(1)}$ and $Q^{(4)}$ are null images, and the dilation of a null image is a null image, Rule 1 and Rule 4 are not needed for single-pixel coding. Fig. 8.15(d) gives an example. The execution time for the DOCIP is

$$t(k) \leq 11(k+1) = O(k).$$

Similar to binary addition, we can develop symbolic substitution binary subtraction algorithms with BIA representations for coding a symbol with two or six pixels. However, four symbolic substitution rules are still required because $Q^{(1)}$ and $Q^{(4)}$ will not be equal to the null image. The DOCIPs take approximately the same execution time for binary subtraction using two-pixel or six-pixel coding as for binary addition.

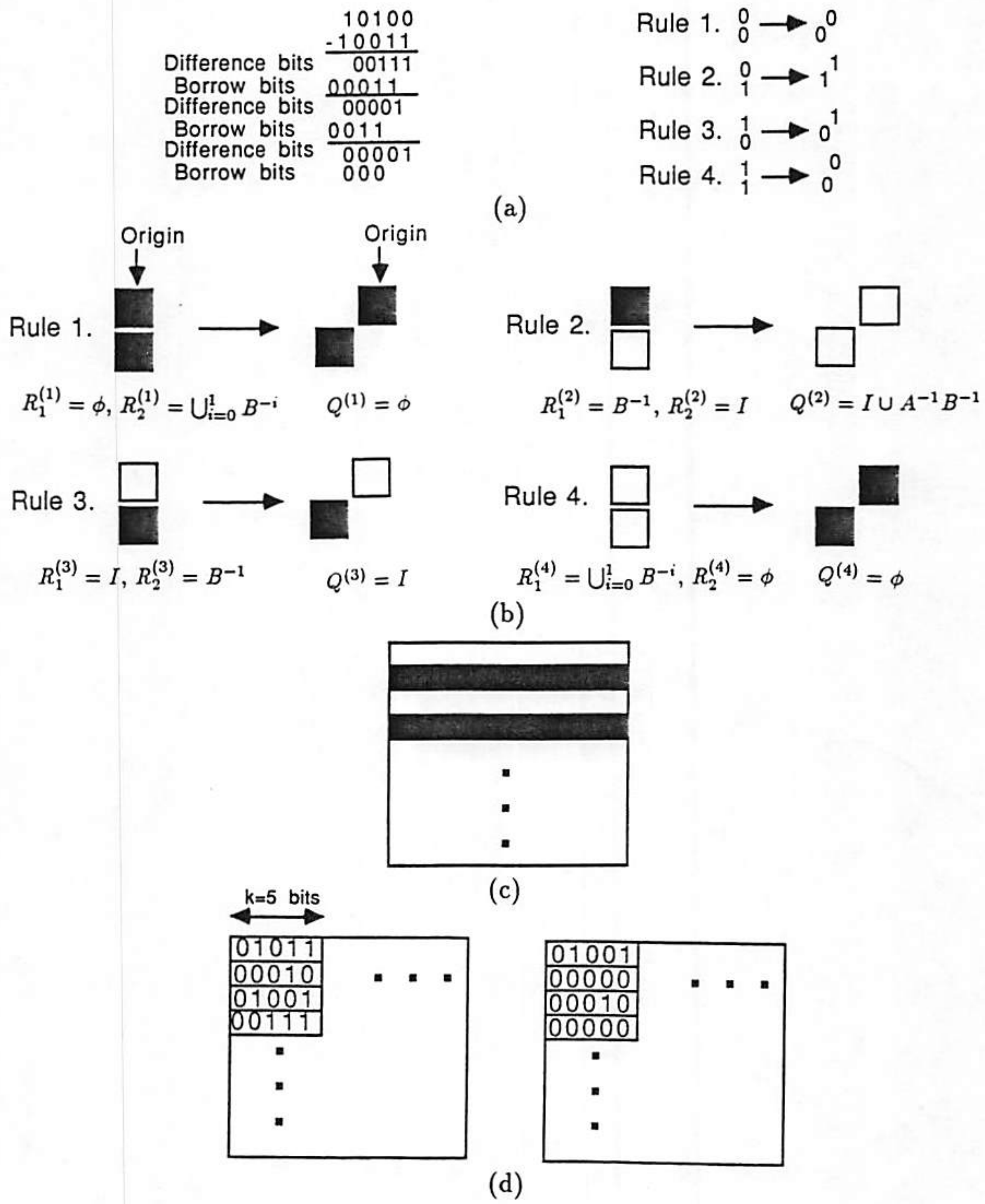

10100
-10011
Difference bits 00111
Borrow bits 00011
Difference bits 00001
Borrow bits 0011
Difference bits 00001
Borrow bits 000
Rule 1. $\genfrac{}{}{0pt}{}{0}{0} \rightarrow {}_0{}^0$
Rule 2. $\genfrac{}{}{0pt}{}{0}{1} \rightarrow {}_1{}^1$
Rule 3. $\genfrac{}{}{0pt}{}{1}{0} \rightarrow {}_0{}^1$
Rule 4. $\genfrac{}{}{0pt}{}{1}{1} \rightarrow {}_0{}^0$
(a)
Origin
Origin
Rule 1.
$R_1^{(1)} = \phi$, $R_2^{(1)} = \bigcup_{i=0}^{1} B^{-i}$
$Q^{(1)} = \phi$
Rule 2.
$R_1^{(2)} = B^{-1}$, $R_2^{(2)} = I$
$Q^{(2)} = I \cup A^{-1}B^{-1}$
Rule 3.
$R_1^{(3)} = I$, $R_2^{(3)} = B^{-1}$
$Q^{(3)} = I$
Rule 4.
$R_1^{(4)} = \bigcup_{i=0}^{1} B^{-i}$, $R_2^{(4)} = \phi$
$Q^{(4)} = \phi$
(b)
(c)
k=5 bits
01011
00010
01001
00111
01001
00000
00010
00000
(d)

Figure 8.15: Parallel subtraction of binary symbol-coded numbers. (a): Four symbolic substitution rules for subtraction. (b): Reference image pairs $R^{(i)}$ and reference images $Q^{(i)}$, $i = 1, 2, 3, 4$, used for subtraction. Because $Q^{(1)}$ and $Q^{(4)}$ are null images, Rules 1 and 4 are not needed for single-pixel coding. (c): The mask M. (d): An example of parallel subtraction of binary symbol-coded numbers.

8.4 Complexity of Parallel Optical Binary Arithmetic

We have shown that BIA offers a general tool for mapping serial binary arithmetic into different forms of parallel binary arithmetic (including binary row-coding, binary stack-coding, and three coding techniques for symbolic substitution arithmetic) in a precise and compact way. The complexity of parallel addition and subtraction of two $N \times N$ arrays of binary numbers (each number with k-bit length) for these different number representations are compared in Table 8.1 and Table 8.2.

Binary row-coded arithmetic requires the smallest number O of fundamental operations. Binary stack-coded arithmetic requires the lowest number of processing elements (or cells) P and the smallest overall $O \times P$ complexity (assume each parallel fundamental operation corresponds to P processing elements executing in parallel). For the normal case in which the word size is larger than one and much smaller than the image size ($1 < k \ll N$), binary row-coded arithmetic can be implemented in the DOCIP with the fastest computation speed (assume the DOCIP can input all operands in an image at a time). The complexity of binary symbol-coded (symbolic substitution) arithmetic in general is in all cases higher than that of binary row-coded and binary stack-coded arithmetic. For implementing symbolic substitution algorithms on the DOCIPs, the single-pixel coding is superior to the other symbol coding techniques.

Number Representation	Binary Row-coding	Binary Stack-coding	Symbolic Substitution (single-pixel coding)	Symbolic Substitution (two-pixel coding)
No. of Dilations (or Erosions)	k	0	9(k+1)	8(k+1)
No. of Unions (or Intersections)	4k+3	16k-12	15(k+1)	16(k+1)
No. of Complements	7k+4 *2k+2	20k-13 *7k-5	12(k+1) *3(k+1)	16(k+1) *4(k+1)
Total No. of Parallel Fundamental Operations O	12k+7 *7k+5	36k-25 *23k-17	36(k+1) *27(k+1)	40(k+1) *28(k+1)
No. of Processing Elements P	k N^2	N^2	2 k N^2	4 k N^2
Total No. of Computations OxP	(12k+7)kN^2 *(7k+5)kN^2	(36k-25)N^2 *(22k-16)N^2	76k(k+1)N^2 *54k(k+1)N^2	160k(k+1)N^2 *112k(k+1)N^2
DOCIP Execution Time T	5k+3	17k-13	17(k+1)	18(k+1) or 28(k+1)
PxT	(5k+3)k N^2	(17k-13)N^2	34k(k+1)N^2	72k(k+1)N^2 or 112k(k+1)N^2

* indicates the number of operations when erosion and intersection are also allowed.

Table 8.1: Complexity of parallel optical binary addition of two NxN arrays of k-bit binary numbers. Each parallel fundamental operation corresponds to P processing elements executing in parallel.

Number Representation	Binary Row-coding	Binary Stack-coding	Symbolic Substitution (single-pixel coding)	Symbolic Substitution (two-pixel coding)
No. of Dilations (or Erosions)	k	0	6(k+1)	8(k+1)
No. of Unions (or Intersections)	4k+3	16k-12	10(k+1)	16(k+1)
No. of Complements	6k+4 *3k+2	22k-18 *11k-8	8(k+1) *2(K+1)	16(k+1) *4(k+1)
Total No. of Parallel Fundamental Operations O	11k+7 *8k+5	43k-33 *33k-26	24(k+1) *18(k+1)	40(k+1) *28(k+1)
No. of Processing Elements P	kN^2	N^2	$2kN^2$	$4kN^2$
Total No. of Computations OxP	$(11k+7)kN^2$ *$(8k+5)kN^2$	$(43k-33)N^2$ *$(33k-26)N^2$	$48k(k+1)N^2$ *$36k(k+1)N^2$	$160k(k+1)N^2$ *$112k(k+1)N^2$
DOCIP Execution Time T	4k+3	20k-17	11(k+1)	18(k+1) or 28(k+1)
PxT	$(4k+3)kN^2$	$(20k-17)N^2$	$22k(k+1)N^2$	$72k(k+1)N^2$ or $112k(k+1)N^2$

* indicates the number of operations when erosion and intersection are also allowed.

Table 8.2: Complexity of parallel optical binary subtraction of two NxN arrays of k-bit binary numbers.

Chapter 9

Implementation of A Prototype DOCIP — Experimental Demonstration

In this chapter we experimentally demonstrate the concept of the DOCIP architecture by implementing one processing element of a prototype optical computer including a 54-gate processor, an instruction decoder, and electronic input/output interfaces. The 54-gate processor consists of a 2-D array of 54 optical logic gates and a 2-D array of 53 subholograms to provide interconnections between gates. The interconnection hologram used in this system is fabricated by a computer-controlled optical system to offer very flexible interconnections. To the best of our knowledge, this experimental system is the most complicated digital optical computing system that has been constructed to date.

A diagram of the main components of this experimental system is shown in Fig. 9.1. A multiple-exposure multi-facet interconnection hologram provides the fixed interconnections between the outputs and the inputs of an array of optical gates. The input data and the instructions are supplied from an LED array. The outputs of optical gates are detected by a video camera and compared with the results of a software simulation. Here we discuss the fabrication of the interconnection hologram, the measurement of the characteristics and uniformity of a Hughes liquid crystal light valve (LCLV) used for implementing the optical gate array, and finally the experimental prototype DOCIP system with its operating principles and experimental results.

9.1 Interconnection Hologram

A space-variant interconnection system [Jenkins84a, Jenkins84b, Sawchuk84] for within-processor interconnection is used in this experimental demonstration. A computer-controlled system is used to make an array of 53 interconnection subholograms. An optical point source S, whose position is controlled by the mirror M2 with two rotational stages (Fig. 9.1), is used to provide an object beam for determining an interconnection of a subhologram in the multi-facet hologram. A mask with a circular aperture, controlled by two translational stages, is used to determine the sizes and positions of subholograms on a holographic plate. The interconnection hologram for this 54-gate optical processing element consists of 53 subholograms (one for each gate except the output gate), which are laid out in a 2-D array. Each subhologram covers a circular area with a diameter of 1.5 mm. The spacing between the centers of two subholograms is 3.0 mm. Note that the path of the object beam and the mask for subholograms are only used for making

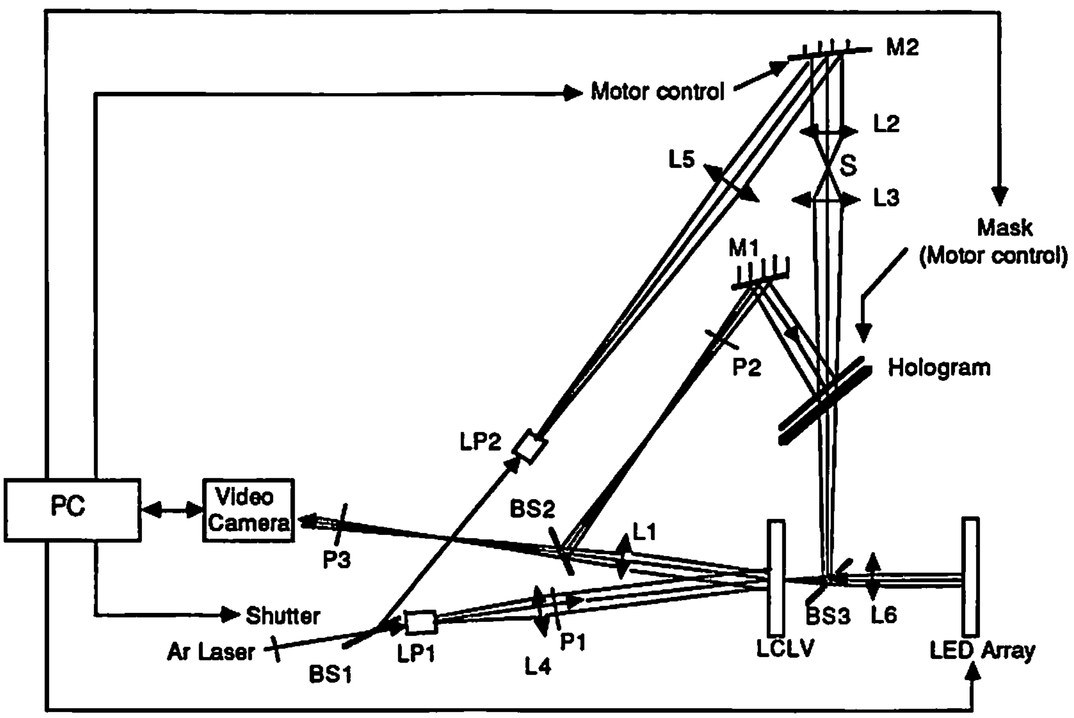


Figure 9.1: Experimental DOCIP system. P1 and P2 are crossed polarizers. Lens L1 images the LCLV gate output plane to the hologram plane. Beam splitter BS3 is in front of the LCLV gate input plane and combines the external input signals from the LED array and the feedback signals from the interconnection hologram. LP1 and LP2 are lens-pinhole assemblies. P1 and P2 are crossed polarizers. The hologram consists of an array of subholograms. Mirror M2 controls the position of point source S during the hologram exposure. After the hologram is made, the mask and all components in the path from the reflection off of BS1 to the hologram are not needed.

the interconnection hologram; they are blocked or moved when we reconstruct the hologram to implement the interconnections of the optical gates.

To successfully make the desired hologram, we have performed a sequence of experiments on two kinds of holograhic recording media: (1) dichromated gelatin, (2) Agfa-Gevaert 8E56HD silver halide photographic plate. We first discuss dichromated holograms and then silver halide holograms.

Dichromated gelatin holograms have many favorable characteristics [Chang80, Chang79, Collier71], for example: 1) the largest refractive index modulation capacity among holographic materials available today; 2) high resolution capacity (higher than 5000 lines/mm); and 3) reprocessing ability [Chang76]. Numerous possible procedures for making dichromated gelatin holograms and their possible formation principles have been reported in [Brandes69, Chang71, Chang79, Chang80, Shankoff68, Curran70, Meyerhofer72, Lin69]. Figure 9.2 shows the preparing and recording procedure of dichromated gelatin holograms used in this experiment, which is primarily a simplified version of that in [Chang79]. This procedure involves five steps:

1. Preparation of gelatin plates from Kodak 649F plates:

   (a) Soak in Kodak fixer F5 for 5 minutes in the dark room. The fixer F5 is prepared from 1) distilled water, 50°C, 600 milliliters; 2) Sodium Thiosulfate $Na_2S_2O_3 \cdot 5H_2O$ (Pentahydrated), 240 grams; 3) Sodium Sulfite (Anhydrous), 15 grams; 4) 28% Acetic Acid, 48 milliliters; 5) Boric Acid (Crystals), 7.5 grams; 6) Potassium Alum $AlK(SO_4)_2 \cdot 12H_2O$ (Fine Granular Dodecahydrated), 15 grams; and 7) stirring and adding cold water to make the total solution 1.0 liters.

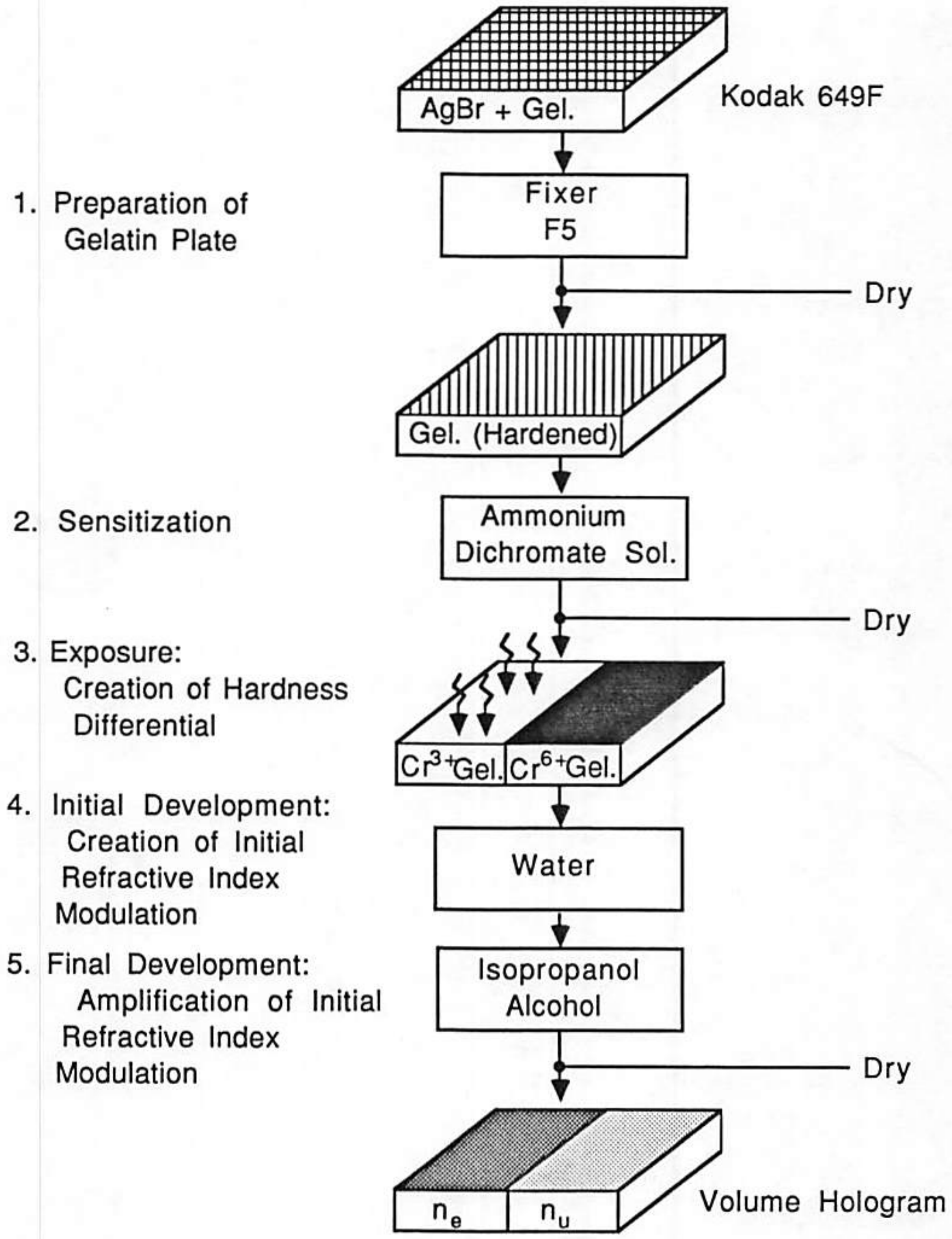

AgBr + Gel.
Kodak 649F
1. Preparation of Gelatin Plate
Fixer F5
Dry
Gel. (Hardened)
2. Sensitization
Ammonium Dichromate Sol.
Dry
3. Exposure: Creation of Hardness Differential
Cr3+Gel.
Cr6+Gel.
4. Initial Development: Creation of Initial Refractive Index Modulation
Water
5. Final Development: Amplification of Initial Refractive Index Modulation
Isopropanol Alcohol
Dry
n_e
n_u
Volume Hologram

Figure 9.2: Preparing and recording procedures for the interconnection hologram of dichromated gelatin.

(b) Wash in running tap water for 2 hours.

(c) Dry about one day or more.

2. Sensitization:

   (a) Sensitize with ammonium dichromate solution (7% ammonium dichromate and 0.5% photo-flo) for 5 minutes in the dark room. The plates are now light sensitive.

   (b) Take the plates from the solution, hang them for 2 or 3 minutes, and clean their glass surfaces in the dark room.

   (c) Dry for 2 days and store in a dry (dessicator) and dark place, and use it within 8 days. Control of humidity during this time and during exposure is important.

3. Exposure:

   - Sequentially expose the dichromated gelatin plate with different point sources of object beams and a common reference beam. Hardness differential between exposed and unexposed region of the plate is created. The sensitivity of the plate is about 200 mJ/cm^2. Each fan-out of a gate requires an exposure. The optical system for fabricating the interconnection hologram is described in in Fig. 9.1. (After the exposure, we develop the hologram; after the development, we block the object path and reconstruct the hologram the same place as the exposures for interconnecting the optical gate array from its output side to its input side.)

4. Initial water development:

- Wash in running tap water for 10 minutes in the dark room. The initial refractive index modulation is created by swelling and transferring gelatin. In the exposed regions much less water is absorbed than in the unexposed regions.

5. Final isopropanol alcohol development:

   (a) Plates are no longer light sensitive. Dehydrate in a 50%/50% solution of distilled water and isopropanol alcohol for 2 minutes.

   (b) Dehydrate in a 10%/90% solution of distilled water and isopropanol alcohol for 2 minutes.

   (c) Dehydrate in 100% isopropanol alcohol for 10 minutes.

   (d) Dry immediately with a blow dryer (until dry, approx. 5 minutes).

   (e) The solutions in (a), (b) and (c) can be saved and re-used. Plates are sensitive to humidity.

Though we can easily achieve very high efficiency (about 90%) by the dichromatic gelatin medium, the uniformity of all subholograms is difficult to control (we have 35 single-exposure subholograms and 18 double-exposure subholograms; totally 71 interconnections should be reconstructed about equal brightness). One possible reason is that the long exposure times (sensitivity is about 200 mJ/cm^2) make the system very sensitive to mechanical vibrations.

The final hologram used in the current system is fabricated with Agfa-Gevaert 8E56HD holographic plate. Its sensitivity is about 0.03 mJ/cm^2; it is much easier to control the uniformity of subholograms. To reconstruct the double-exposure subholograms to have equally bright interconnections, the exposure energy of the second object beam is about 1.11 times that of the first object beam (this value is similar that reported in previous work on Agfa-Gevaert 8E75HD in [Johnson85]). After the final exposure, the plate is developed using D-19 (3 minutes), Kodak Stop (45 seconds), Rapid Fixer (2 minutes) and water (30 minutes). The development procedure of silver halide holograms is much simpler than that of dichromated gelatin holograms and the sensitivity of silver halide holographic plates is also much higher; but silver halide holograms are amplititude holograms and the diffraction efficiency is only about 6%.

The above procedures for fabricating dichromated gelatin and silver halide holograms can be also applied to fabricate the interconnection holograms for general optical sequential logic circuits. The general principle of holography can be found in [Goodman68]; the coupled wave theory of volume holography is described by [Kogelnik69, Collier71].

9.2 Optical Gate Array

Besides the interconnection hologram, the other crucial component in the experimental DOCIP system is the optical gate array. Device requirements and techniques for implementing parallel two-dimensional arrays of independent optical logic gates have been studied in [Jenkins83,Jenkins84c,Sawchuk84]. We use a Hughes liquid-crystal light valve (LCLV) with liquid-crystal molecules in a 45° twisted nematic configuration, which is the same one used in [Jenkins84a, Sawchuk84], as a spatial light modulator to provide a nonlinear threshold response for implementing a two-dimensional optical NOR gate array. The AC voltage bias supply of the LCLV is 6.7 volts and the frequency is 400 Hz.

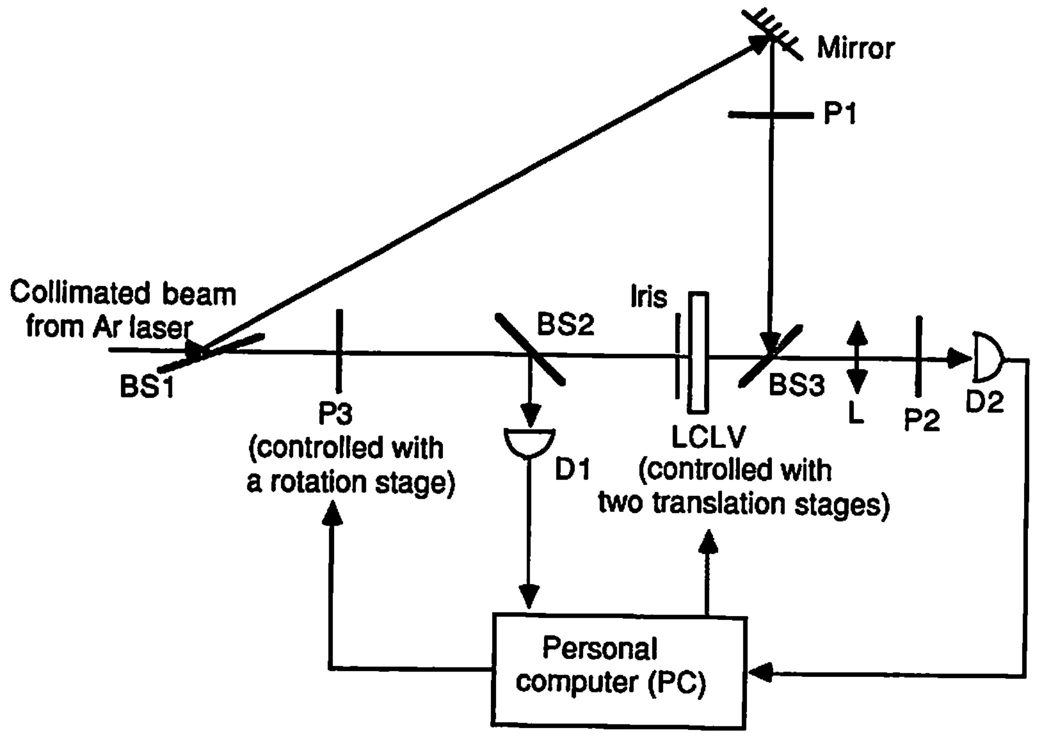

Mirror
P1
Collimated beam from Ar laser
BS1
P3 (controlled with a rotation stage)
BS2
D1
Iris
LCLV (controlled with two translation stages)
BS3
L
P2
D2
Personal computer (PC)

Figure 9.3: Experimental system for testing the LCLV uniformity. P1 and P2 are crossed polarizers. Lens L images the LCLV output to the detector D1. BS1, BS2, and BS3 are beam splitters. The polarizer P3, controlled by a rotation stage, is used to control the input light intensity of the LCLV. The LCLV is controlled by two translation stages for measuring the characteristics of different regions.

The Hughes LCLV characteristics have been also studied in [Jenkins84c,Sawchuk84,Michaelson79]. For the 54-gate optical processor of the experimental DOCIP system to function properly, the uniformity over the spatial region of the LCLV must be considered. A computer controlled system is used to measure the LCLV uniformity. It main components are described in Fig. 9.3. The LCLV is read out between crossed polarizers, P1 and P2, and is biased to implement a NOR operation. The polarizer P3, controlled by a rotation stage with a PC, is used to control the input light intensity of the LCLV. The detectors D1 and D2 are used to read the input and output intensity of the LCLV. The position of the LCLV is controlled by two translation stages for measuring the characteristics of different regions. The result of this uniformity test indicates that the central region of the LCLV is reasonably uniform and allows a fan-in of 3. The stead-state output vs. input relationship of 49 tested points in the center region of the LCLV is given in Fig. 9.4.

9.3 Experimental DOCIP System and Circuit Design

The major components of the experimental DOCIP system (9.1.) are (1) a 2-D array of optical logic gates, (2) a multi-facet interconnection hologram for connecting optical logic gates, (3) an LED array for supplying instructions and inputs into the machine, (4) a video camera for recording outputs, and (5) a host computer for providing the control of this binary digital optical computing system. The gate size in this experiment has a diameter of 0.3 *mm* and the spacing between the centers of two gates is 0.6 *mm*. The LCLV requires an illuminance of about

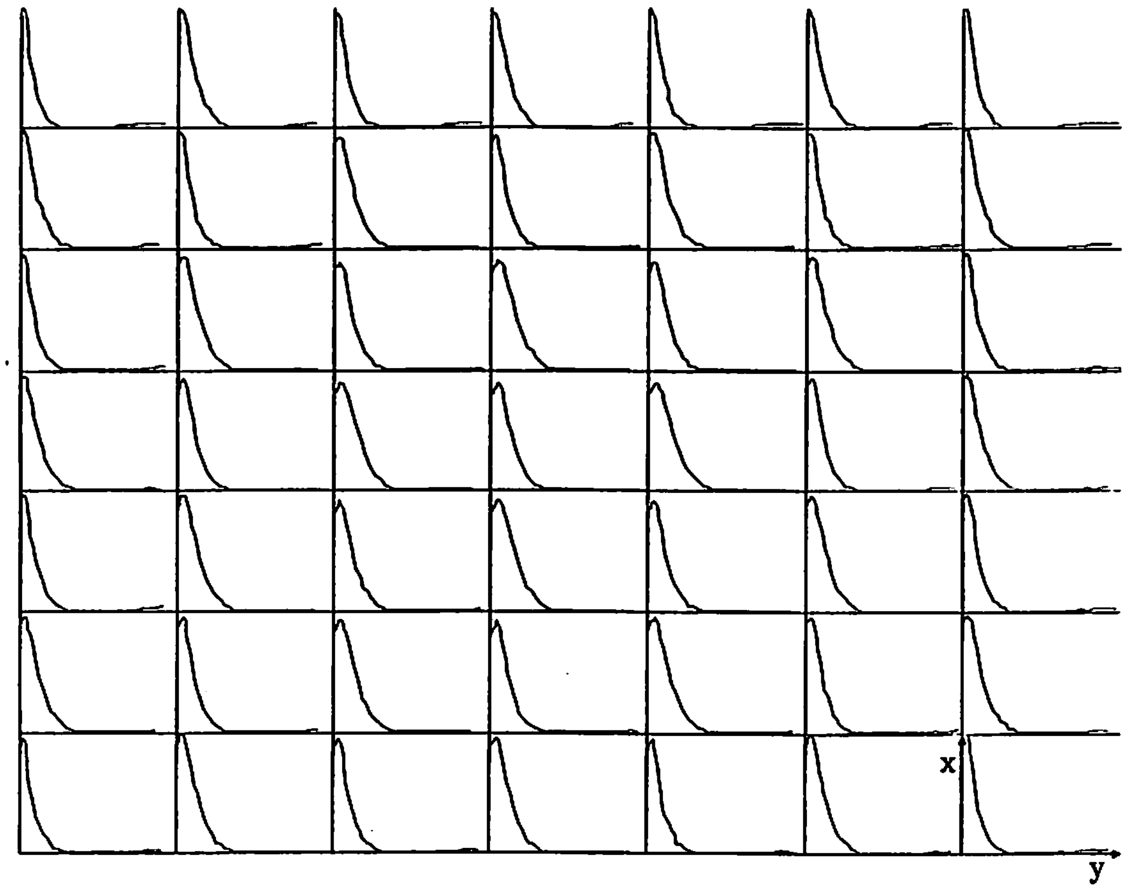


Figure 9.4: The stead-state output vs. input relationship of 49 tested points in the center region of the LCLV. Each characteristic curve corresponds to an area of 1 mm^2 in the LCLV. The x and y axes of each characteristic curve represent the input and output energy of the LCLV respectively.

25 $\mu W/cm^2$ on the input side. The spectral sensitivity of the LCLV requires us to use high-brightness green LEDs with spectrum centered at 565 nm. We used HP HLMP 1540 LEDs that produce 30 mcd at 565 nm in a cone of half angle 45°. An 8×8 square array of these LEDs with drivers, controller and PC interface was constructed. The center-to-center spacing is 3.59 mm in both the x and y axes.

The circuit diagram of the processing element, as shown in Fig. 9.5, consists of 54 NOR gates with maximum fan-in of 3 and fan-out of 2. Gate 54 is not needed because the output y (the output of gate 49) is connected to gate 54 and also detected by the camera, so there is no subhologram corresponding to gate 54. The direct holographic implementation of the interconnections for this circuit is illustrated in Fig. 9.6. There are 19 point-spread functions (PSFs) stored in 53 subholograms for these gates; each PSF determines an interconnection pattern that connects the output of a gate to the input(s) of other desired gate(s). A layout of the PSFs is shown in Fig. 9.6(a), with each number representing the gate output that is imaged onto that subhologram. The PSF stored in each subhologram is shown in Fig. 9.6(b), with the ordered pairs representing the locations of the gate inputs that each subhologram addresses (relative to the subhologram location).

The processing element includes a 3-bit destination selector, a 3-bit master-slave flip-flop memory, a 6-bit memory selector with a union module, and a 5-bit neighborhood selector (for DOCIP-array4) with a dilation module. Instructions for this experimental DOCIP system are supplied from an LED array and decoded by the optical hardware. These instructions have the format: $(c, d_1, d_2, d_3, s_1, s_2, ..., s_6, n_1, n_2, ..., n_5)$ where c selects the image from the input or from the feedback; d_1, d_2, and d_3 select the destination memory for storing the

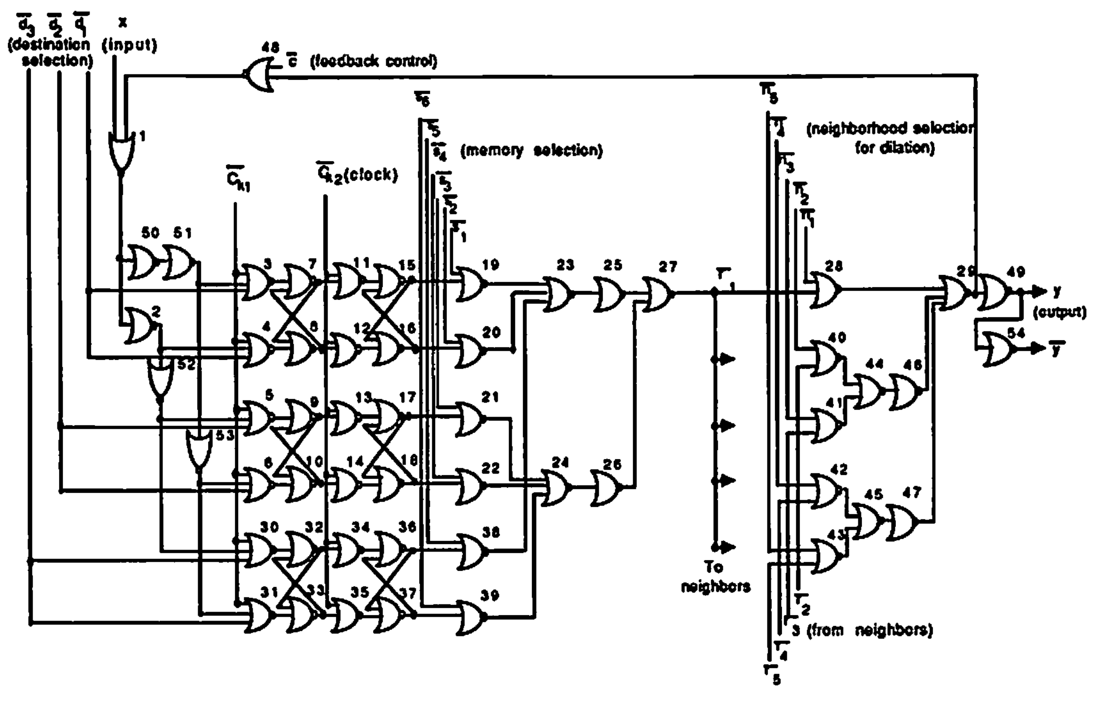


Figure 9.5: The circuit diagram of a 54-gate processing element of the DOCIP-array4.

	2	23	25	27	28	49	
3	7	11	15	19	29	48	50
4	8	12	16	20	24	26	51
5	9	13	17	21	40	41	52
6	10	14	18	22	44	46	53
30	32	34	36	38	42	43	1
31	33	35	37	39	45	47	

(a)

	(-1,-2) (6,-3)	(1,0)	(1,0)	(1,0)	(0,-1)	(1,0)	
(1,0)	(1,0) (0,-1)	(1,0)	(1,0) (0,-1)	(-2,1)	(1,0) (1,1)	(1,-4)	(0,-1)
(1,0)	(1,0) (0,1)	(1,0)	(1,0) (0,1)	(-2,2)	(1,0)	(-2,2)	(-7,1) (0,-2)
(1,0)	(1,0) (0,-1)	(1,0)	(1,0) (0,-1)	(1,1)	(0,-1)	(-1,-1)	(-7,0) (-7,-2)
(1,0)	(1,0) (0,1)	(1,0)	(1,0) (0,1)	(1,2)	(1,0)	(-1,3)	(-7,0) (-7,-2)
(1,0)	(1,0) (0,-1)	(1,0)	(1,0) (0,-1)	(-2,5)	(0,-1)	(-1,-1)	(0,4) (-6,5)
(1,0)	(1,0) (0,1)	(1,0)	(1,0) (0,1)	(1,4)	(1,0)	(-1,5)	

(b)

Figure 9.6: Direct implementation of the interconnections of Fig. 9.5. (a): Layout of the PSFs on the hologram. The numbers correspond to those of Fig. 9.5. (b): PSF stored in each subhologram. The ordered pairs represent the locations of the gate inputs (image points) addressed (relative to the subhologram location).

image; $s_1, s_2, ..., s_6$ select the output from the memory elements; and $n_1, n_2, ..., n_5$ control the neighborhood mask, i.e. supply the reference image.

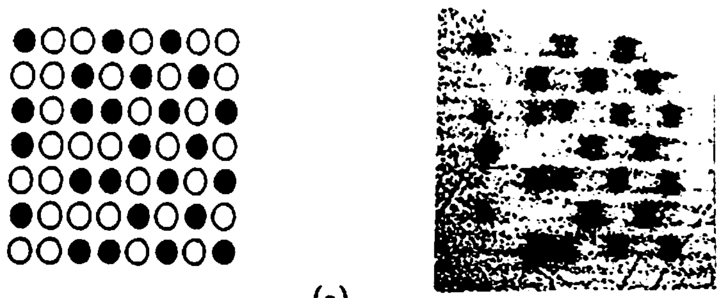
9.4 Experimental Results

When biased as described in Section 9.3, our LCLV had a response time of approximately 300 *msec*. Implementing any instruction requires a maximum of 16 gate delays, so our clock cycle of instructions is limited to few seconds. The layout of the outputs of the 54 gates at the monitor plane is shown in Fig. 9.7. We have provided a sequence of instructions to test the circuit and compare with the simulation in the host computer. Figures 9.8 and 9.9 shows the simulation results and the outputs of 54 gates of some instructions which include the functions of CLEAR, STORE, COMPLEMENT, UNION, and DILATION. A comparison with simulations performed electronically showed that the DOCIP system functioned properly.

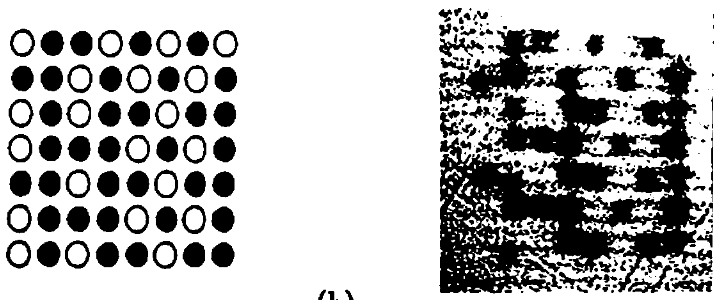
In summary, we have presented a digital optical computing system which consists a 54-gate processor, an instruction decoder, and electronic input/output interfaces to demonstrate the concept of the DOCIP architecture. The system functioned properly, producing the expected outputs. The operation speed fundamentally depends on the speed of the optical gate array. Recent progress in high speed optical gate arrays (Chapter 2) shows promise for fabricating a fast and large-scale DOCIP system.

54 $\bar{y}$	49 Output y	28	27	25	23	2	
50	48	29	19	15	11	7	3
51	26	24	20	16 Memory1 m1	12	8	4
52	41	40	21	17 $\overline{m2}$	13	9	5
53	46	44	22	18 Memory2 m2	14	10	6
1	43	42	38	36 $\overline{m3}$	34	32	30
	47	45	39	37 Memory3 m3	35	33	31

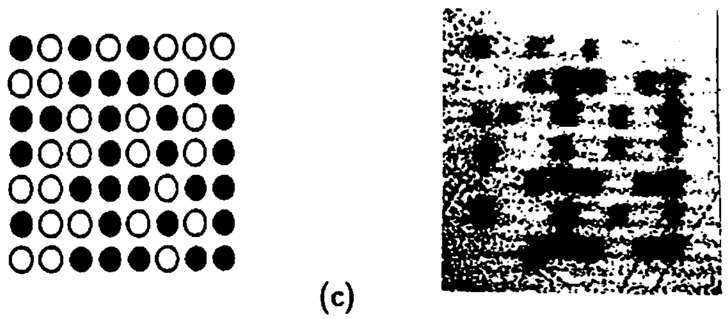
Figure 9.7: Layout of outputs of the 54 gates in the camera input plane (i.e. monitor plane). The numbers correspond to those of Fig. 9.5.



(a)

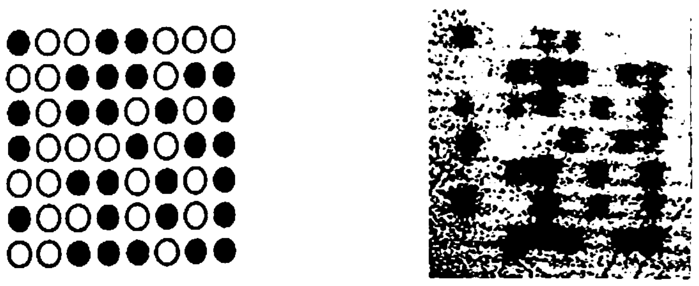


(b)

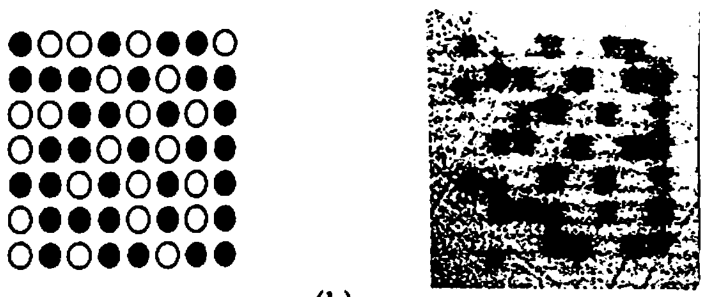


(c)

Figure 9.8: The simulation results and the outputs of 54 gates after the execution of some instructions (I). (a) The state before executing the instructions (all LEDs are off). (b) CLEAR (reset 3 master-slave flip-flop memories to 0; ouputs of gates 8, 10, 33, 16, 18,and 37 are dark). (c) STORE, COMPLEMENT and DILATION (store an external input $x_1 = 1$ in Memory 1 (outputs of gates 8 and 16 have value 1), choose its complement (output of gate 25 is 0 and output of gate 27 is 1), and dilate with a reference image corresponding to the 4-connected nearest-neighborhood (output of gate 49 is 1)).



(a)



(b)

Figure 9.9: The simulation results and the outputs of 54 gates after the execution of some instructions (II). (a) Store the dilated result in Memory 2 (outputs of gates 16 and 18 are 1). (b) STORE and UNION (store an external input $x_2 = 0$ in Memory 3 (outputs of gates 33 and 37 are 0), and perform the union of three memories (output of gate 27 is 0 and output of gate 49 is 1)).

Chapter 10

Conclusions and Future Extensions

10.1 Conclusions

Techniques for digital optical cellular image processing are presented in this thesis. A binary image algebra (BIA), which serves as its software, provides an unified theory of parallel binary image processing and a spatial logic of parallel digital optical computing for developing parallel algorithms/languages. The applications and relationships with other computing theories demonstrate that BIA is a powerful systematic tool for formalizing and analyzing parallel algorithms. Digital optical cellular image processors (DOCIPs), which serve as its hardware, utilize the parallel communication and 3-D free interconnection capabilities of optics for avoiding communication bottlenecks and matching BIA parallel algorithms efficiently. BIA parallel algorithms for low level vision and array numerical computation demonstrate the programming and power for these 1-instruction

DOCIP machines. Besides the software simulation, the concept of the DOCIP architectures is expermentally demonstrated by implementing a prototype optical computing system with a 54-gate optical processor, an instruction decoder and electronic input/output interfaces.

In conclusion, BIA is a simple, precise and complete algebraic theory of binary images and binary optical computing; the DOCIP machines are highly parallel, have simple organization, low cell complexity and potentially fast processing ability.

10.2 Future Extensions

Several promising research directions have become apparent during the course of this work. Following is a list of future research topics:

- The study of the possible extensions of binary image algebra (BIA) for providing the algebraic foundation to solve general image processing and computer vision problems.
- The analysis of the performance of BIA parallel algorithms (e.g. pattern recognition algorithms) on real data and comparing to other methods (e.g. perceptron, neural networks and conventional pattern classification systems for pattern recognition).
- The analysis of different DOCIP cell structures with larger grain sizes for developing fast sophisticated high level vision algorithms.

- The implementation, comparison and mapping techniques of different interconnection networks of processors for fully utilizing the capabilities of optical parallelism and 3-D global interconnections.
- The construction of a large scale DOCIP system (e.g. 512 $\times$ 512 cells).
- The investigation and implementation of a compact DOCIP system.
- The construction of a DOCIP with an optical multiplexing technique for cell interconnections.
- Refining the procedure for making multi-exposure multi-facet dichromated gelatin holograms.
- The development of fast 2-D optical gate arrays for optical computing systems.

Appendix A

Proof of Two Fundamental Principles

In this appendix, the proof of Lemma 3.1, Theorem 3.1 and Theorem 3.2 for two fundamental prociples of BIA is presented.

A.1 Proof of Lemma 3.1

We start with the case of $X = \check{R}$ and then the case of $X \neq \check{R}$.

Case 1: $X = \check{R}$, i.e. $R = \check{X}$.

We want to prove

$$\overline{(\overline{X} \oplus \check{X}) \cup (X \oplus \overline{\check{X}}) \cup \overline{I}} = I \leftrightarrow \overline{(\overline{X} \oplus \check{X}) \cup (X \oplus \overline{\check{X}})} \cap I = I.$$

1. Claim $I \subset \overline{(\overline{X} \oplus \check{X}) \cup (X \oplus \overline{\check{X}})} \cap I$

   $\leftrightarrow (0,0) \in \overline{(\overline{X} \oplus \check{X}) \cup (X \oplus \overline{\check{X}})}$

   $\leftrightarrow (0,0) \notin (\overline{X} \oplus \check{X}) \cup (X \oplus \overline{\check{X}})$

   $\leftrightarrow [(0,0) \notin (\overline{X} \oplus \check{X})] \wedge [(0,0) \notin (X \oplus \overline{\check{X}})]$:

(a) Claim $(0,0) \notin (\overline{X} \oplus \check{X})$:

Assume $(0,0) \in (\overline{X} \oplus \check{X})$

$\rightarrow (0,0) \in \{(a+(-x), b+(-y)) \in W \mid (a,b) \in \overline{X}, (-x,-y) \in \check{X}\}$

$\rightarrow (0,0) \in \{(a-x, b-y) \in W \mid (a,b) \notin X, (x,y) \in X\}$

$\rightarrow \exists(a-x, b-y) = (0,0)$ where $(a,b) \notin X, (x,y) \in X$

$\rightarrow \exists(x,y) = (a,b)$ where $(a,b) \notin X, (x,y) \in X$

which is impossible, since $(x,y) = (a,b) \notin X$ contradicts with $(x,y) = (a,b) \in X$. Therefore, the assumption is wrong, we have that $(0,0) \notin (\overline{X} \oplus \check{X})$.

(b) Claim $(0,0) \notin (X \oplus \overline{\check{X}})$:

Assume $(0,0) \in (X \oplus \overline{\check{X}})$

$\rightarrow (0,0) \in \{(x+(-a), y+(-b)) \in W \mid (x,y) \in X, (-a,-b) \in \overline{\check{X}}\}$

$\rightarrow (0,0) \in \{(x-a, y-b) \in W \mid (x,y) \in X, (-a,-b) \notin \check{X}\}$

$\rightarrow (0,0) \in \{(x-a, y-b) \in W \mid (x,y) \in X, (a,b) \notin X\}$

$\rightarrow \exists(x-a, y-b) = (0,0)$ where $(a,b) \notin X, (x,y) \in X$

$\rightarrow \exists(x,y) = (a,b)$ where $(a,b) \notin X, (x,y) \in X$

which is impossible, since $(x,y) = (a,b) \notin X$ contradicts with $(x,y) = (a,b) \in X$. Therefore, the assumption is wrong, we have that $(0,0) \notin (X \oplus \overline{\check{X}})$.

By (a) and (b), we have $[(0,0) \notin (\overline{X} \oplus \check{X})] \wedge [(0,0) \notin (X \oplus \overline{\check{X}})]$, i.e.

$$I \subset \overline{(\overline{X} \oplus \check{X}) \cup (X \oplus \overline{\check{X}})}.$$

We also know $I \subset I$, then we have

$$I \subset \overline{(\overline{X} \oplus \check{X}) \cup (X \oplus \overline{\check{X}})} \cap I.$$

2. Claim $\overline{(\overline{X}\oplus\check{X})\cup(X\oplus\overline{\check{X}})\cup\overline{I}}\subset I$:

   Since $I \subset I$, it implies

$$\overline{(\overline{X}\oplus\check{X})\cup(X\oplus\overline{\check{X}})\cup\overline{I}} = \overline{(\overline{X}\oplus\check{X})\cup(X\oplus\overline{\check{X}})}\cap I \subset I.$$

From (1) and (2), we have

$$I \subset \overline{(\overline{X}\oplus\check{X})\cup(X\oplus\overline{\check{X}})}\cap I$$

and

$$\overline{(\overline{X}\oplus\check{X})\cup(X\oplus\overline{\check{X}})\cup\overline{I}} = \overline{(\overline{X}\oplus\check{X})\cup(X\oplus\overline{\check{X}})}\cap I \subset I.$$

Thus, by the equivalence of sets, we have $\overline{(\overline{X}\oplus\check{X})\cup(X\oplus\overline{\check{X}})}\cap I = I$.

Case 2: $X \neq \check{R}$, i.e. $R \neq \check{X}$.

We want to prove

$$\overline{(\overline{X}\oplus R)\cup(X\oplus\overline{R})\cup\overline{I}} = \phi \leftrightarrow \overline{(\overline{X}\oplus R)\cup(X\oplus\overline{R})}\cap I = \phi$$

1. Claim $I \not\subset \overline{(\overline{X}\oplus R)\cup(X\oplus\overline{R})}$

   $\leftrightarrow (0,0) \notin \overline{(\overline{X}\oplus R)\cup(X\oplus\overline{R})}$

   $\leftrightarrow (0,0) \in (\overline{X}\oplus R)\cup(X\oplus\overline{R})$

   $\leftrightarrow (0,0) \in (\overline{X}\oplus R) \vee (0,0) \in (X\oplus\overline{R})$:

   Now we assume $(0,0) \notin (\overline{X}\oplus R) \wedge (0,0) \notin (X\oplus\overline{R})$:

   (a) If $(0,0) \notin (\overline{X}\oplus R)$

      $\rightarrow (0,0) \notin \{(a+k, b+l) \mid (a,b) \in \overline{X}, (k,l) \in R\}$

      $\rightarrow (a+k, b+l) \neq (0,0)$, $\forall (a,b) \in \overline{X}$, $\forall (k,l) \in R$

      $\rightarrow (a,b) \neq (-k,-l)$, $\forall (a,b) \notin X$, $\forall (k,l) \in R$

      $\rightarrow \forall (k,l) \in R$, $\exists (a,b) \in X$, $(a,b) = (-k,-l)$

$\rightarrow \forall(-k,-l) \in \check{R},\ \exists(a,b) \in X,\ (a,b) = (-k,-l)$

$\rightarrow \forall(i,j) \in \check{R},\ \exists(a,b) \in X,\ (a,b) = (i,j)$

$\rightarrow ((i,j) \in \check{R}) \rightarrow ((i,j) \in X)$

$\rightarrow \check{R} \subset X.$

(b) If $(0,0) \not\in (X \oplus \overline{R})$, then X $\subset \check{R}$. Since the dilation operation is commutative, by interchanging the variables X and $\check{R}$ and applying the same procedure as (a), we have $X \subset \check{R}$.

2. By the above (a) and (b), we have $X = \check{R}$ which contradicts with $X \neq \check{R}$. Thus, the assumption is wrong, and we get

$(0,0) \in (\overline{X} \oplus R) \vee (0,0) \in (X \oplus \overline{R})$

$\leftrightarrow I \not\subset \overline{(\overline{X} \oplus R) \cup (X \oplus \overline{R})}$

$\leftrightarrow \overline{(\overline{X} \oplus R) \cup (X \oplus \overline{R})} \cap I = \phi$

$\leftrightarrow \overline{(\overline{X} \oplus R) \cup (X \oplus \overline{R}) \cup \overline{I}} = \phi.$

Hence, by case 1. and case 2., we have shown that

$$\overline{(\overline{X} \oplus R) \cup (X \oplus \overline{R}) \cup \overline{I}} = \begin{cases} I & if X = \check{R} \\ \phi & otherwise. \end{cases}$$

A.2 Proof of Theorem 3.1

Consider any image transformation (general case):

$$T : \begin{cases} X_1 \longrightarrow A_1 \\ X_2 \longrightarrow A_2 \\ \quad\quad . \\ \quad\quad . \\ \quad\quad . \\ X_l \longrightarrow A_l \end{cases}$$

where $X_i \in P(W), A_i \in P(W), i = 1, 2, ..., l$.

If we choose $R_i = \check{X}_i$, $Q_i = \check{A}_i, i = 1, 2, ..., l$, use Lemma 3.1 and some properties of the dilation (i.e. $I \oplus X = X$ and $\phi \oplus X = \phi$), then we have

$$T(X) = \bigcup_{i=1}^{l} \{\overline{(\overline{X} \oplus R_i) \cup (X \oplus \overline{R_i}) \cup \overline{I}} \oplus Q_i\}.$$

Since some images X_i may map into the null image ϕ for a given image transformation; by Lemma 3.1, we have that

$$T(X) = \bigcup_{i=1}^{k} \{\overline{(\overline{X} \oplus R_i) \cup (X \oplus \overline{R_i}) \cup \overline{I}} \oplus Q_i\}$$

where $k \leq l$, $l = \#(P(W))$ is the cardinality of $P(W)$.

A.3 Proof of Theorem 3.2

This can be shown in a very straight forward way. Any image is a set of image points and is the union of point images (consisting one and only one image point). A point image $\{(i, j)\}$ can be written as

$$\{(i,j)\} = A^i B^j.$$

Hence, the union of all point images which are contained in X is the image X. For example, an image $X = \{(2,0),(1,-1),(-1,2)\}$ is denoted by

$$X = A^2 \cup AB^{-1} \cup A^{-1}B^2.$$

Appendix B

Properties of BIA Image Operations

B.1 Properties of Complement and Difference

The complement $^-$, a unary operation, is decreasing and shift variant (considering the outside of an image). The difference X/R, a binary operation, is increasing (but decreasing with respect to the reference image R), antiextensive with respect to X, and shift variant (the reference image R is fixed once it is given). Note that the difference operation is not commutative, not associative, and not distributive over other operations. Furthermore, the difference operation is more complicated than the complement. Hence, it is preferable to employ the complement as a fundamental operation, but not the difference. The major properties of the complement and the difference are listed in the following:

1. $\overline{X} = W/X$

2. $X/R = X \cap \overline{R}$

3. $\overline{\phi} = W$

4. $\overline{W} = \phi$

5. $\overline{\overline{X}} = X$ (idempotent for twice complements)

6. $X/\phi = X$ (idempotent for a given reference image $R = \phi$)

7. $X/X = \phi$

8. $X \subset Y \leftrightarrow \overline{Y} \subset \overline{X}$ (decreasing)

9. $X \subset Y \leftrightarrow X/R \subset Y/R$ (increasing)

10. $X/R \subset X$ (antiextensive)

11. $X \subset R \leftrightarrow X/R = \phi$

12. $\overline{X \cap R} = \overline{X} \cup \overline{R}$

13. $\overline{X \cup R} = \overline{X} \cap \overline{R}$

14. $X \cap \overline{X} = \phi$

15. $X \cup \overline{X} = W$

16. $\overline{X \oplus R} = \overline{X} \ominus \check{R}$ where $\check{R} = \{(-x, -y) \mid (x, y) \in R\}$

17. $\overline{X \oplus R} = \overline{X} \ominus \check{R}$ where $\check{R} = \{(-x, -y) \mid (x, y) \in R\}$

B.2 Properties of Union and Intersection

The union $\cup$, a binary operation, is increasing, extensive, shift variant, idempotent, commutative, associative, and distributive over intersection. The intersection $\cap$, a binary operation, is increasing, antiextensive, shift variant, idempotent, commutative, associative, and distibutive over union. The major properties of the union and the intersection are listed in the following:

1. $X \cup \phi = X$

   $X \cap \phi = \phi$

2. $X \cup X = X$

   $X \cap X = X$

3. $X \cup R = R \cup X$ (commutative)

   $X \cap R = R \cap X$ (commutative)

4. $X \cup (R \cup Q) = (X \cup R) \cup Q$ (associative)

   $X \cap (R \cap Q) = (X \cap R) \cap Q$ (associative)

5. $X \cup W = W$

   $X \cap W = X$ (idempotent for a given reference image $R = W$)

6. $X \cup (R \cap Q) = (X \cup R) \cap (X \cup Q)$ (distributive)

   $X \cap (R \cup Q) = (X \cap R) \cup (X \cap Q)$ (distributive)

7. $X \subset X \cup R$ (extensive)

   $X \cap R \subset X$ (antiextensive)

8. $X \subset Y \leftrightarrow X \cup R \subset Y \cup R$ (increasing)

   $X \subset Y \leftrightarrow X \cap R \subset Y \cap R$ (increasing)

9. $X \subset R \leftrightarrow X \cup R = R$

   $X \subset R \leftrightarrow X \cap R = X$

10. $R \subset X \wedge Q \subset X \rightarrow R \cup Q \subset X$

    $X \subset R \wedge X \subset Q \rightarrow X \subset R \cup Q$

11. $R \subset X \wedge Q \subset Y \rightarrow R \cup Q \subset X \cup Y$

    $R \subset X \wedge Q \subset Y \rightarrow R \cap Q \subset X \cap Y$

B.3 Properties of Dilation and Erosion

The dilation $\oplus$, a binary operation, is increasing, extensive for a given reference image R which contains the elementary image I, shift invariant, commutative, associative, distributive over union, and possesses an identity which is I. The erosion $\ominus$, a binary operation, is shift invariant, increasing (but decreasing with respect to the refernce image R), antiextensive for a given reference image R which contains the elementary image I. But, in general, the erosion is not commutative, not associative, not distibutive over other operations, and does not possesses a left identity. The major properties of the union and the intersection are listed in the following:

1. $X \oplus R = R \oplus X$ (commutative)

   $X \ominus R \neq R \ominus X$ (in general)

2. $(X \oplus R) \oplus Q = X \oplus (R \oplus Q)$ (associative)

   $(X \ominus R) \ominus Q \neq X \ominus (R \ominus Q)$ (in general)

   $(X \ominus R) \ominus Q = (X \ominus Q) \ominus R$

3. $X \oplus (R \cup Q) = (X \oplus R) \cup (X \oplus Q)$ (distributive)

   $X \ominus (R \cup Q) = (X \ominus R) \cap (X \ominus Q)$

   $X \ominus (R \oplus Q) = (X \ominus R) \ominus Q$

4. $X \oplus I = X = I \oplus X$ (identity)

   $X \ominus I = X \neq I \ominus X$ (in general)

5. $X \oplus \phi = \phi = \phi \oplus X$

   $X \ominus \phi = W \neq \phi \ominus X$ (in general)

6. $X \subset X \oplus R$ when $I \subset R$ (extensive)

   $X \ominus R \subset X$ when $I \subset R$ (antiextensive)

7. $X \subset Y \leftrightarrow X \oplus R \subset Y \oplus R$ (increasing)

   $X \subset Y \leftrightarrow X \ominus R \subset Y \ominus R$ (increasing)

8. $R \subset Q \leftrightarrow X \oplus R \subset X \oplus Q$

   $R \subset Q \leftrightarrow X \ominus Q \subset X \ominus R$

9. $X \oplus (R \cap Q) \subset (X \oplus R) \cap (X \oplus Q)$ (distributive inequality)

   $X \ominus (R \cap Q) \supset (X \ominus R) \cup (X \ominus Q)$

   $(X \cup Y) \ominus R \supset (X \ominus R) \cap (Y \ominus R)$

   $(X \ominus R) \oplus Q \subset (X \oplus R) \ominus Q$

   Remark: "$\supset$" means "contains".

B.4 Properties of Some Standard Operations

1. The symmetric difference is shift variant (with a fixed reference image R), commutative and associative. Symbolically,

(a) $X \triangle R = R \triangle X$

(b) $X \triangle (R \triangle Q) = (X \triangle R) \triangle Q$

(c) $X \triangle \phi = X$

(d) $X \triangle X = \phi$

(e) $X \triangle \overline{X} = W$

(f) $X \triangle W = \overline{X}$

(g) $X \cap (R \triangle Q) = (X \cap R) \triangle (X \cap Q)$

(h) $X \cup (R \triangle Q) \neq (X \cup R) \triangle (X \cup Q)$ (in general)

(i) $X \triangle R = Y \triangle R \rightarrow X = Y$

2. The opening $\circ$ is shift invariant, increasing, anti-extensive and idempotent. The closing $\bullet$ is shift invarinat, extensive, and idempotent. Symbolically,

(a) $X \circ R \subset X \subset X \bullet R$

(b) $X \subset Y \rightarrow X \circ R \subset Y \circ R$

(c) $X \subset Y \rightarrow X \bullet R \subset Y \bullet R$

(d) $(X \circ R) \circ R = X \circ R$

(e) $(X \bullet R) \bullet R = X \bullet R$

3. The thinning is shift invariant and antiextensive. The thickening is shift invariant and extensive. The major properties are in the following:

(a) $X \circledcirc R \subset X \subset X \odot R$

(b) $X \subset Y \rightarrow X \circledcirc R \subset Y \circledcirc R$

(c) $X \subset Y \rightarrow X \odot R \subset Y \odot R$

(d) If $R \subset Q$ (which means $R_1 \subset Q_1$ and $R_2 \subset Q_2$), then we have:

$$R \subset Q \to X \circledcirc R \subset X \circledcirc Q \subset X \subset X \odot Q \subset X \odot R.$$

(e) $\overline{X \odot R} = \overline{X} \circledcirc R^*$ where $R = (R_1, R_2)$ and $R^* = (R_2, R_1)$.

Appendix C

Proof of Theorems for Low Level Vision

C.1 Proof of Theorem 4.2

We can easily see that (2) in Theorem 4.2 is a generalization of (1) in Theorem 4.2. (1) is used for exactly matching shapes (or templates) with shift invariance; (2) is generalized to more general cases. For example, to consider noise and to have rotational invariance, we can choose the family $\{R(\theta)\}$ to incorporate all aspect reference image pairs. In the following, we prove (1) and then (2) will follow from it directly. The proof will demonstrate the mathematical correspondance between boolean logic and BIA. The notations $x(i,j)$ and $r(i,j)$ will be used to represent the binary values (0 or 1) of pixels at coordinate (i,j) of image functions which correspond to the images X and R in BIA notations.

First, let us use the boolean logic XOR (excusive or) operation, i.e.

$$x(i,j) \text{ XOR } r(i,j) = (\overline{x}(i,j) \wedge r(i,j)) \vee (x(i,j) \wedge \overline{r}(i,j)),$$

to achieve the pixel-wise comparison where the ouput value with '0' means that '$x(i,j)$' matches '$r(i,j)$' and the output value with '1' means that '$x(i,j)$' does not match '$r(i,j)$'.

Second, to check the occurence of the shape (defined by R with M) in the tested image X at coordinate (i,j), we have to shift the origin of the shape to the coordinate (i,j) in X. Then the process of the comparison of the shape and the subimage in X (limited in the mask M) and the indication of "match" (0) and "not match" (1) will be performed by

$$\bigvee_{(k,l)\in M} (\overline{x}(i+k,j+l) \wedge r(k,l)) \vee \bigvee_{(k,l)\in M} (x(i+k,j+l) \wedge \overline{r}(k,l)).$$

If the above equation is considered as a binary operation operating on two images $x(i,j)$ and $r(i,j)$, then this operation is not commutative; in order to achieve the commutativity, we change (k,l) with $(-k,-l)$ and denote $\check{r}(k,l) = r(-k,-l)$:

$$\bigvee_{(-k,-l)\in \check{M}} (\overline{x}(i-k,j-l) \wedge \check{r}(k,l)) \vee \bigvee_{(-k,-l)\in \check{M}} (x(i-k,j-l) \wedge \overline{\check{r}}(k,l)).$$

If the output value of the above equation is '0', then it means that the location (i,j) of the image X has the occurrence of the shape (defined by R and M); if '1', the shape is not occurred at (i,j).

Third, let us run over all coordinates (i,j) (i.e. for all $(i,j) \in W$ the universal image) and then the union of those coordinates with value '0' would be the answer. The value '0' at a coordinate (i,j) corresponds to the null image in set notation and the value '1' at a coordinate (i,j) corresponds to the point image $\{(i,j)\}$. For convenience, in the following we mix the notations of boolean logic functions and set notations; if the output of a boolean logic expression is '0', it represents the null image ϕ; if '1', it represents the point image $\{(i,j)\}$. Thus, we have

$$\bigcup_{(i,j)\in W} \Big(\bigvee_{(-k,-l)\in \check{M}} (\bar{x}(i-k,j-l) \wedge \check{r}(k,l)) \vee \bigvee_{(-k,-l)\in \check{M}} (x(i-k,j-l) \wedge \check{\bar{r}}(k,l))\Big)$$

which is the same as

$$\bigcup_{(i,j)\in W} \Big(\bigvee_{(-k,-l)\in \check{M}} (\bar{x}(i-k,j-l) \wedge \check{r}(k,l))\Big) \cup \bigcup_{(i,j)\in W} \Big(\bigvee_{(-k,-l)\in \check{M}} (x(i-k,j-l) \wedge \check{\bar{r}}(k,l))\Big).$$

Since $x(i,j) \neq 0$ only when $(i,j) \in X$ and $\check{r}(k,l) \neq 0$ only when $(k,l) \in \check{R}$, we have

$$\begin{aligned}
&\bigcup_{(i,j)\in W} (\textstyle\bigvee_{(-k,-l)\in \check{M}} (\bar{x}(i-k,j-l) \wedge \check{r}(k,l))) \\
&= \{(i,j) \mid (i-k,j-l) \in \overline{X}, (k,l) \in \check{R}\} \\
&= \{(i+k,j+l) \mid (i,j) \in \overline{X}, (k,l) \in \check{R}\} \\
&= \overline{X} \oplus \check{R}.
\end{aligned}$$

Similarly, we have

$$\bigcup_{(i,j)\in W} \Big(\bigvee_{(-k,-l)\in \check{M}} (x(i-k,j-l) \wedge \check{\bar{r}}(k,l))\Big) = X \oplus (\check{M}/\check{R}).$$

Hence, if we use '0' to indicate "match", we have

$$(\overline{X} \oplus \check{R}) \cup (X \oplus (\check{M}/\check{R}));$$

if we use '1' to indicate "match",then we have

$$\overline{(\overline{X} \oplus \check{R}) \cup (X \oplus (\check{M}/\check{R}))}.$$

Thus, the locations of a shape, which is defined by a non-null reference image R with a non-null reference image (called mask) M and $R \subset M \subset W$, are the image points in the following

$$\begin{aligned}
\overline{(\overline{X} \oplus \check{R}) \cup (X \oplus (\check{M}/\check{R}))} &= \overline{(\overline{X} \oplus \check{R}) \cup (X \oplus \overline{\overline{\check{M}}} \cup \check{R})} \\
&= (X \ominus R) \cap (\overline{X} \ominus (M/R)).
\end{aligned}$$

A more intuitive illustration is that the foreground X should match R by $X \ominus R$ (using multiple-input AND gates to examine the locations where the 1's should be), while the background $\overline{X}$ should match M/R by $\overline{X} \ominus (M/R)$. Combining both results by the intersection (AND), we then implement the shape recognition by $(X \ominus R) \cap (\overline{X} \ominus (M/R))$. Replacing R by R_1 and (M/R) by R_2, we obtain the hit or miss transform (template matching) for shape recognition.

C.2 Proof of Theorem 4.3

(1) The straight forward way for removing the "pepper" noise is the thinning operation $X \circledcirc R_4$ (or $X \circledcirc R_8$). Follow this, we have

$$\begin{aligned}
X \circledcirc R_4 &= \overline{\overline{X} \cup \overline{(\overline{X} \oplus I) \cup (X \oplus M_4)}} \\
&= \overline{\overline{X} \cup \overline{\overline{\overline{X}} \cup (X \oplus M_4)}} \\
&= \overline{\overline{X} \cup (X \cap \overline{\overline{X} \oplus M_4})} \\
&= \overline{(\overline{X} \cup X) \cap (\overline{X} \cup \overline{\overline{X} \oplus M_4})} \\
&= \overline{W \cap (\overline{X} \cup \overline{\overline{X} \oplus M_4})} \\
&= \overline{\overline{X} \cup \overline{\overline{X} \oplus M_4}}.
\end{aligned}$$

(2) The straight forward way for removing the "pepper" noise is the thickening operation $X \odot Q_4$ (or $X \odot Q_8$). Follow this, we have

$$\begin{aligned}
X \odot Q_4 &= X \cup \overline{(\overline{X} \oplus M_4) \cup (X \oplus I)} \\
&= X \cup \overline{(\overline{X} \oplus M_4) \cup X} \\
&= X \cup (\overline{\overline{X} \oplus M_4} \cap \overline{X}) \\
&= (X \cup \overline{\overline{X} \oplus M_4}) \cap (X \cup \overline{X})
\end{aligned}$$

$$\begin{aligned} &= (X \cup \overline{\overline{X} \oplus M_4}) \cap W \\ &= (X \cup \overline{\overline{X} \oplus M_4}). \end{aligned}$$

(3)The straight forward way for removing the "salt and pepper" noise is the difference of $X \odot Q_4$ by $X \circledast R_4$ (or the difference of $X \odot Q_8$ by $X \circledast R_8$). By a similar procedure as above we can achieve the desired result.

C.3 Proof of Theorem 4.4

To extract the region whose sizes are between two reference images R and Q, the straight forward way is to design a morphological band pass filter:

$$(X \circ R)/(X \circ Q) = ((X \ominus R) \oplus R)/((X \ominus Q) \oplus Q).$$

To obtain the locations of those desired regions, we then perform the skelotonization:

$$\begin{aligned} S(((X \ominus R) \oplus R)/((X \ominus Q) \oplus Q)) &= S((X \ominus R)/((X \ominus Q) \oplus Q))) \\ &= S(\overline{(\overline{X} \oplus \check{R}) \cup (\overline{\overline{X} \oplus \check{Q}} \oplus Q)}). \end{aligned}$$

References

[Abraham86] G. Abraham, "Multiple-valued Logic for Optoelectronics," *Optical Engineering*, Vol. 25, No. 1, p.3, 1986.

[Agui82] T. Agui, et al., "An Algebraic Approach to the Generation and Description of Binary Pictures", *IEEE Transaction on Pattern Analysis and Machine Intelligence*, Vol. PAMI-4, No. 6, pp. 635-641, 1982.

[Aho74] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

[Arrathoon86] R. Arrathoon and S. Kozaitis, " Shadow Casting for Multiple-valued Associative Logic," *Optical Engineering*, Vol. 25, No. 1, p. 29, 1986.

[Bell86] T. E. Bell, "Optical Computing: A Field in Flux," *IEEE Spectrum*, Vol. 23, No. 8, pp. 34-57, 1986.

[Berlekamp82] E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning Ways for Your Mathematical Plays*, Vol. 2, Chap. 25, Academic Press, New York, 1982.

[Bartelt82] H. Bartelt and S. K. Case, "High-efficiency Hybrid Computer-generated Hologram," *Applied Optics*, Vol. 21, No. 16, pp. 2886-2890, 1982.

[Birkhoff65] G. Birkhoff and S. MacLane, *A Brief Survey of Mordern Algebra*, Macmillan, New York, 1965.

[Birkhoff70] G. Birkhoff and T. C. Bartee, *Mordern Applied Algebra*, McGraw-Hill, New York, 1970.

[Brandes69] R. G. Brandes, E. E. Francois, T. A. Shankoff, "Preparation of Dichromated Gelatin Films for Holography", *Applied Optics*, Vol. 8, No. 11, pp. 2346-2348, 1969.

[Brenner85] K. Brenner and A. Huang, "An Optical Processor Based on Symbolic Substitution", *Conference Proceeding of the Topical Meeting on Optical Computing*, Optical Society of America, March 18, pp. WA4.1-WA4.3, 1985.

[Brenner86a] K.-H. Brenner, A. Huang, and N. Streibl, "Digital optical computing with symbolic substitution," *Applied Optics*, Vol. 25, pp. 3054-3060, 1986.

[Brenner86b] K.-H. Brenner, "New Implementation of Symbolic Substitution," *Applied Optics*, Vol. 25, pp. 3061-3064, 1986.

[Brenner87] K.-H. Brenner and G. Stucke "Programmable Optical Processor Based on Symbolic Substitution," *Topical Meeting on Optical Computing*, Technical Digest Series 1987, Vol. 11, Optical Society of America, Washington, D.C., pp. 6-8, 1987.

[Burks70] A. Burks ed., *Essays on Cellular Automata*, Univ. of Illinois Press, 1970.

[CAPAIDM85a] "Morphological Systems Software," A Session in *Computer Architecture for Pattern Analysis and Image Database Management*, 1985 IEEE Computer Society Workshop, pp. 429-468.

[CAPAIDM85b] "Morphological Systems Hardware," A Session in *Computer Architecture for Pattern Analysis and Image Database Management*, 1985 IEEE Computer Society Workshop, pp. 469-500.

[Capps87] C. D. Capps, R. A. Falk and T. L. Houk "Optical Arithmetic/Logic Unit Based on Residue Number Theory and Symbolic Substitution", *Topical Meeting on Optical Computing*, Technical Digest Series 1987, Vol. 11, Optical Society of America, Washington, D.C., pp. 62-65, 1987.

[Caulfield79] H. J. Caulfield, *Handbook of Optical Holography*, Academic Press, New York, 1979.

[Caulfield86] H. J. Caulfield, "Systolic Optical Cellular Array Processors," *Optical Engineering*, Vol. 25, No. 7, pp. 825-827, 1986.

[Chang71] M. Chang, "Dichromated Gelatin of Improved Optical Quality", *Applied Optics*, Vol. 10, No. 11, pp. 2550-2551, 1971.

[Chang76] B. J. Chang, "Post-Processing of Developed Dichromated Gelatin Holograms", *Optical Communication*, Vol. 17, p. 270, 1976.

[Chang79] B. J. Chang and C. D. Leonard, "Dichromated Gelatin for the Fabrication of Holographic Optical Elements", *Applied Optics*, Vol. 18, No. 14, pp. 2407-2417, 1979.

[Chang80] B. J. Chang, "Dichromated Gelatin Holograms and Their Applications", *Optical Engineering*, Vol. 19, No. 5, pp. 642-648, 1980.

[Chavel83] P. Chavel, et al, "Architectures for A Sequential Optical Logic Processor," *Proc. 10th Interconnectional Optical Computing Conference*, pp. 6-12, 1983.

[Chellappa85a] R. Chellappa and A. A. Sawchuk ed., *Digital Image Processing and Analysis: Volume 1: Digtial Image Processing*, IEEE Computer Society Press, 1985.

[Chellappa85b] R. Chellappa and A. A. Sawchuk ed., *Digital Image Processing and Analysis: Volume 2: Digtial Image Analysis*, IEEE Computer Society Press, 1985.

[Clare73] C. R. Clare, *Designing Logic Systems Using State Machines*, McGraw-Hill, New York, 1973.

[Cloonan87] T. J. Cloonan, "Strengths and Weaknesses of Optical Architectures Based on Symbolic Substitution," *Topical Meeting on Optical Computing*, Technical Digest Series 1987, Vol. 11, Optical Society of America, Washington, D.C., pp. 12-15, 1987.

[Collier71] R. J. Collier, C. B. Buckhardt, and L. H. Lin, *Optical Holography*, Academic Press, New York, 1971.

[Curran70] R. K. Curran and T. A. Shankoff, "The Mechanism of Hologram Formation in Dichromated Gelatin", *Applied Optics*, Vol. 9, No. 7, pp. 1651-1657, 1970.

[Demongeot85] J. Demongeot, E. Goles, and M. Tchuente ed., *Dynamical Systems and Cellular Automa*, Academic Press, New York, 1985.

[Deutsch72] E. S. Deutsch, "Thinning algorithms on rectangular, hexagonal, and triangular arrays," *Commun. Ass. Comput. Mach.*, Vol. 15, pp.827-837, 1972.

[Duda73] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, John Wiley & Sons, 1973.

[Duff73] M. J. B. Duff et al., "A cellular logic array for image processing," *Patt. Recog.*, Vol. 5, pp. 229-234, 1973.

[Duff81] M. J. B. Duff and S. Levialdi, *Languages and Architectures for Image Processing*, Academic Press, 1981.

[Farmer84] D. Farmer, T. Toffoli, and S. Wolfram ed., "Special Issue on Cellular Automata", *Physica D Nonlinear Phenomena*, Vol. 10D, 1984, pp. 273-383.

[Fork82] R.L. Fork, "Physics of Optical Switching", *Phys. Rev. A,* Vol. 26, p. 2049, 1982.

[Gardner70] M. Gardner, "Mathematical Games", *Scientific American*, Vol. 223, No. 4, pp. 120-123, 1970.

[Gardner71] M. Gardner, "Mathematical Games", *Scientific American*, Vol. 224, No. 2, pp. 112-117, 1971.

[Gardner83] M. Gardner, *Wheels, Life, and Other Mathematical Amusements*, Freeman, San Francisco, 1983.

[Giardina84] C. R. Giardina, "The Universal Imaging Algebra", *Pattern Recognition Letters*, Vol. 2, pp. 165-172, 1984.

[Gibbs80] H.M. Gibbs, S.L. McCall, and T.N.C. Venkatesan, "Optical Bistable Devices: The Basic Components of All-Optical Systems?", *Optical Engineering,* Vol. 19, p. 463, 1980.

[Gibbs85] H.M. Gibbs, *Optical Bistability: Controlling Light with Light*, Academic Press, Inc., New York, 1985.

[Gibbs87] H.M. Gibbs, "Two-Dimensional Arrays of Semiconductor Optical Gates for Optical Computing", *Topical Meeting on Optical Computing*, Technical Digest Series 1987, Vol. 11, Optical Society of America, Washington, D.C., pp. 168-171, March, 1987.

[Gilbert76] G. Gilbert, *Mordern Algebra with Applications*, John Wiley & Sons, New York, 1976.

[Golay69] J. E. Golay, "Hexagonal parallel pattern transformation", *IEEE Trans. Comput.*, Vol. C-18, pp.733-740, 1969.

[Goodman68] J. W. Goodman, *Introduction to Fourier Optics*, McGraw-Hill, 1968.

[Gray71] S. B. Gray, "Local properties of binary images in two dimensions", *IEEE Trans. Comput.*, Vol C-20, pp. 551-561, 1971.

[Haralick87] R. M. Haralick, S. R. Sternberg, and X. Zhuang, "Image Analysis Using Mathematical Morphology," *IEEE Transaction on Pattern Analysis and Machine Intelligence*, Vol. PAMI-9, No. 4, pp. 532-550, 1987.

[Hartmanis70] J. Hartmanis and R. E. Stearms, *Algebraic Structure Theory of Sequential Machines*, Prentice-Hall, Englewood Cliffs, N. J., 1970.

[Hayes78] J. P. Hayes, *Computer Architecture and Organization*, McGraw-Hill, Inc., New York, 1978.

[Hopcroft79] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, Mass., 1979.

[Horrigan79] F. A. Horrigan and W. W. Stoner, "Reside-Based Optical Prosessor," *Proc. SPIE*, 185, 19, 1979.

[Huang79] A. Huang, et al, "Optical Computation Using Residue Arithmetic," *Applied Optics*, 18, p. 149, 1979.

[Huang83] A. Huang, "Parallel Algorithms for Optical Digital Computers," in *Technical Digest, IEEE Tenth International Optical Computing Conference*, pp. 13-17, 1983.

[Huang87a] K. S. Huang, B. K. Jenkins, A. A. Sawchuk, "Binary Image Algebra and Digital Optical Cellular Image Processors", *Topical Meeting on Optical Computing*, Technical Digest Series 1987, Vol. 11, Optical Society of America, Washington, D.C., pp. 20-23, March, 1987.

[Huang87b] K. S. Huang, B. K. Jenkins, A. A. Sawchuk, "A Cellular Hypercube Architecture for Image Processing", *Applications of Digital Image Processing X*, Proc. Soc. Photo-Opt. Instr. Eng., Vol. 829, pp. 331-338, August, 1987.

[Huang87c] K. S. Huang, B. K. Jenkins, A. A. Sawchuk, "Optical Cellular Logic Architectures Based on Binary Image Algebra ", *Proc. IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence*, pp. 19-26, October, 1987.

[Huang87d] K. S. Huang, B. K. Jenkins, A. A. Sawchuk, "Binary Image Algebra Representations of Optical Cellular Logic and Symbolic Substitution", Annual Meeting, Paper ThA4, Optical Society of America, Rochester, October, 1987; *Journal of the Optical Society of America A*, Vol. 4, No. 13, p. 87, December, 1987.

[Huang87e] K. S. Huang, B. K. Jenkins, A. A. Sawchuk, "Programming A Digital Optical Cellular Image Processor", Annual Meeting, Paper ThA6, Optical Society of America, Rochester, October, 1987; *Journal of the Optical Society of America A*, Vol. 4, No. 13, pp. 87-88, December, 1987.

[Huang88a] K. S. Huang, B. K. Jenkins, A. A. Sawchuk, "Binary Image Algebra and Digital Optical Cellular Image Processor Design", accepted for publication in *Computer Vision, Graphics, and Image processing.*

[Huang88b] K. S. Huang, B. K. Jenkins, A. A. Sawchuk, "An Image Algebra Representation of Parallel Optical Binary Arithmetic", submitted to *Applied Optics.*

[Huang88c] K. S. Huang, B. K. Jenkins, A. A. Sawchuk, "Optical Symbolic Substitution and Pattern Recognition Algorithms Based on Binary Image Algebra", *ICO Topical Meeting On Optical Computing*, Toulon, France, August 29 - September 2, 1988; to be published in *Proc. Soc. Photo-Opt. Instr. Eng.*, Vol. 829, 1988.

[Huang88d] K. S. Huang, et al, "Implementation of A Prototype Digital Optical Cellular Image Processor (DOCIP)", *ICO Topical Meeting On Optical Computing*, Toulon, France, August 29 - September 2, 1988; to be published in *Proc. Soc. Photo-Opt. Instr. Eng.*, Vol. 829, 1988.

[Huang88e] K. S. Huang, B. K. Jenkins, A. A. Sawchuk, "Image Processing Applications of Binary Image Algebra", to be presented at Annual Meeting, Optical Society of America, Santa Clara, California, October 30 - November 4, 1988; to be published in *Journal of the Optical Society of America A*, 1988.

[Huang88f] K. S. Huang, et al, "A Computer-Controlled Optical System for Fabricating Multi-Facet Interconnection Holograms", to be presented at Annual Meeting, Optical Society of America, Santa Clara, California, October 30 - November 4, 1988; to be published in *Journal of the Optical Society of America A*, 1988.

[Huang88g] K. S. Huang, B. K. Jenkins, A. A. Sawchuk, "A Digital Optical Cellular Image Processor (DOCIP): I. Theory and Architecture", to be submitted to *Applied Optics.*

[Huang88h] K. S. Huang, B. K. Jenkins, A. A. Sawchuk, "A Digital Optical Cellular Image Processor (DOCIP): II. Control and Programming", to be submitted to *Applied Optics.*

[Huang88i] K. S. Huang, et al, "A Digital Optical Cellular Image Processor (DOCIP): III. Experiment", to be submitted to *Applied Optics.*

[Hurst86] S. L. Hurst, "Multiple-valued Threshold Logic: Its Status and Its Realization," *Optical Engineering*, Vol. 25, No. 1, p. 44, 1986.

[Hwang84] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill, New York, 1984.

[Jenkins83] B. K. Jenkins, A. A. Sawchuk and T. C. Strand, "Spatial Light Modulator Requirements for Sequential Optical Logic," Annual Meeting, Optical Society of America, New Orleans, October, 1983; *Journal of the Optical Society of America*, Vol. 73, p. 1950A, 1983.

[Jenkins84a] B. K. Jenkins, et al., "Sequential Optical Logic Implementation," *Applied Optics*, Vol. 23, No. 19, pp. 3455-3464, 1984.

[Jenkins84b] B. K. Jenkins, et al., "Architectural Implications of A Digital Optical Processor," *Applied Optics*, Vol. 23, No. 19, pp. 3465-3474, 1984.

[Jenkins84c] B. K. Jenkins, *Optical Logic Systems: Implementation and Architectural Implications*, Ph.D. Thesis, University of Southern California, Los Angeles, 1984.

[Jenkins85] B. K. Jenkins and A. A. Sawchuk, "Optical Cellular Logic Architectures for Image Processing," *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, Florida, , pp. 61-65, Nov. 1985.

[Jenkins86] B. K. Jenkins and A. A. Sawchuk, "Binary Optical Computing Architecture," *Optics News*, Vol. 12, No. 4, p. 25, 1986.

[Jeon87] Ho-In Jeon, "Digital Optical Processor Based on Symbolic Substitution Using Matched Filtering", *Topical Meeting on Optical Computing*, Technical Digest Series 1987, Vol. 11, Optical Society of America, Washington, D.C., pp. 115-118, 1987.

[Jewell84a] J.L. Jewell, M.C. Rushford, and H.M. Gibbs, "Use of a Single Nonlinear Fabry-Perot Etalon as Optical Logic Gates", *Appl. Phys. Lett.,* Vol. 44, p. 172, 1984.

[Jewell84b] J.L. Jewell, M.C. Rushford, H.M. Gibbs, and N. Peyghambarian, "Single-Etalon Optical Logic Gates", in *Technical Digest Conference on Lasers and Electrooptics,* Optical Society of Amera, Washington, D.C., paper THg2, 1984.

[Jewell85] J. L. Jewell, et al, *3-PJ, 82-MHz Optical Logic Gates in A Room-temperature GaAs-AlGaAs Multiple-quantum-well Etalon*, *Appl. Phys. Lett.,* Vol. 46, pp. 918-920, 1986.

[Jewell86] J. L. Jewell, et al, *Parallel Operation and Crosstalk Measurements in GaAs Etalon Optical Logic*, *Appl. Phys. Lett.,* Vol. 48, pp. 1342-1344, 1986.

[Jewell87] J. L. Jewell, et al, *GaAs-AlAs Monolithic Microresonator Arrays*, *Appl. Phys. Lett.,* Vol. 51, pp. 94-96, 1987.

[Johnson85] K. M. Johnson, et al, "Multiple Multiple-exposure Hologram", *Applied Optics*, Vol. 24, No. 24, pp. 4467-4472, 1985.

[Katevenis85] M. G. H. Katevenis, *Reduced Instruction Set Computer Architectures for VLSI*, The MIT Press, Cambridge, Massachusetts, 1985.

[Klein72] J. Klein and J. Serra, "The texture analyzer," *J. Microscopy*, Vol. 95, No. 2, 1972, pp.349-356.

[Kogelnik69] H. Kogelnik, "Coupled Wave Theory for Thick Hologram Gratings," *Bell Syst. Tech. J.*, Vol. 48, No. 9, pp. 2909-2947, 1969.

[Lee86a] Y. H. Lee, et al, *Streak-camera Observation of 200-ps recovery of an optical gate in a windowless GaAs Etalon Array, Appl. Phys. Lett.,* Vol. 48, pp. 754-756, 1986.

[Lee86b] Y. H. Lee, et al, *Speed and Effectiveness of Windowless GaAs Etalons as Optical Logic Gates, Appl. Phys. Lett.,* Vol. 49, pp. 486-488, 1986.

[Levialdi72] S. Levialdi, "On shrinking of binary patterns," *Commun. ACM*, Vol. 15, pp. 7-10, 1972.

[Lin69] L. H. Lin, "Hologram Formation in Hardened Dichromated Gelatin Films," *Applied Optics*, Vol. 8, No. 5, pp. 963-966, 1969.

[Lougheed80] R. M. Lougheed, D. L. McCubbrey, and S. R. Sternberg, " Cytocomputers: Architectures for Parallel Image Processing," *Proc. IEEE Workshop Picture Data Description and Management*, CA, pp. 281-286, 1980.

[Mait87] J. N. Mait and K.-H. Brenner, "Optical Systems for Symbolic Substitution," *Topical Meeting on Optical Computing*, Technical Digest Series 1987, Vol. 11, Optical Society of America, Washington, D.C., pp. 12-15, 1987.

[Mandeville83] J. Mandeville, "Novel Method for Automated Optical Inspection of Printed Circuits," *IBM Research Report RC-9900*, IBM T. J. Waston Research Center, Yorktown Heights, N. Y., 1983.

[Matheron75] G. Matheron, *Random Sets and Integral Geometry,* Wiley, New York, 1975.

[McCormick65] B. H. McCormick, "The Illinois pattern recognition computer", *Trans. Electron. Comput.*, Vol. EC-12, pp. 434-443, 1965.

[Meyerhofer72] D. Meyerhofer, "Phase Holograms in Dichromated Gelatin," *RCA Review*, Vol. 33, pp. 110-130, 1972.

[Michaelson79] J. D. Michaelson, *Characterization of Liquid Crystal Light Valves and Their Applications to Real-Time Nonlinear Optical Processing*, Ph.D. Thesis, University of Southern California, Los Angeles, 1979.

[Miller82] D. A. B. Miller, "Bistable Optical Devices: Physics and Operating Charactristics", *Laser Focus,* Vol. 18, No. 4, p. 79, 1982.

[Miller85] D. A. B. Miller, et al, *The Quantum Well Self-Electrooptic Effect Device: Optoelectronic Bistability and Oscillation, and Self-Linearized Modulation, IEEEE Journal of Quantum Electronics,* Vol. QE-21, No. 9, pp. 1462-1476, 1985.

[Miller86] D. A. B. Miller, et al, *Integrated Quantum Well Self-electro-optic Effect Device:* 2×2 *Array of Optically Bistable Switches, Appl. Phys. Lett.,* Vol. 49, pp. 821-823, 1986.

[Minkowski03] H. Minkowski, "Volumen and Oberflaeche," *Math. Ann.*, Vol. 57, pp. 447-495, 1903.

[Minsky67] M. L. Minsky, *Computation: Finite and Infinte Machines*, Prentice-Hall, Engelwood Cliffs, N. J., 1967.

[Neumann51] J. von Neumann, "The General Logical Theory of Automata," in *Cerebral Mechanisms in Behavior-The Hixon Symposium* , L. A. Jeffress, Ed. New York: Wiley, 1951.

[Neumann66] J. von Neumann, *Theory of Self-reproducing Automata,* edited by A. W. Burks, Univ. of Illinois Press, 1966.

[Oppenheim75] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing,* Prentice-Hall, 1975.

[Packard85] N. H. Packard and S. Wolfram, "Two-Dimensional Cellular Automata," *Journal of Statistical Physics*, Vol. 38, Nos. 5/6, pp. 901-946, 1985.

[Poundstone85] W. Poundstone, *The Recursive Universe*, William Morrow and Company, New York, 1985.

[Poel56] W. L. Van der Poel, "The Essential Types of Operations in an Automatic Computer,"*Nachrichtentechnishe Fachberichte*, Vol. 4, pp. 144-145, 1956.

[Pratt78] W. K. Pratt, *Digital Image Processing*, John Wiley & Sons, 1978.

[Preston79] K. Preston Jr. et al., "Basics of Cellular Logic with Some Application's in Medical Image Processing," *Proc. of IEEE*, Vol. 67, No. 5, 1979, pp. 826-856.

[Preston83] K. Preston Jr., "Cellular Logic Computers for Pattern Recognition," *IEEE Computer*, Jan. 1983, pp.36-47.

[Preston84] K. Preston and M. J. B. Duff, *Modern Cellular Automata — Theory and Applications,* Plenum Press, New York, 1984.

[ProcIEEE84] *Proc. IEEE*, Special Issue on Optical Computing, Vol. 72, No. 7, 1984.

[Psaltis79] D. Psaltis and D. Casasent, "Optical Residue Arithmetic: A Correlation Approach," *Applied Optics*, 18, p. 163, 1979.

[Psaltis86] D. Psaltis and R. A. Athale, "High Accuracy Computation with Linear Analog Optical Systems: A Critical Study", *Applied Optics*, Vol. 25, No. 18, pp. 3071-3077, 1986.

[Ramamoorthy87] P. A. Ramamoorthy and S. Antony "Optical MSD Adder Using Polarization Coded Symbolic Substitution", *Topical Meeting on Optical Computing*, Technical Digest Series 1987, Vol. 11, Optical Society of America, Washington, D.C., pp. 111-114, 1987.

[Ritter87] G. X. Ritter and P. D. Gader, "Image Algebra Techniques for Parallel Image Processing",*Journal of Parallel and Distributed Computing*, Vol. 4, No. 1, pp. 7-44, 1987.

[Rosenfeld70] A. Rosenfeld, "Connectivity in Digital Pictures," *J. Assoc. Comput. Mach.*, Vol. 17, pp. 146-160, 1970.

[Rosenfeld82] A. Rosenfeld, A. C. Kak, *Digital Picture Processing*, Academic Press, 1982.

[Rosenfeld83] A. Rosenfeld, "Parallel Image Processing Using Cellular Arrays," *IEEE Computer*, Jan. 1983, pp. 14-20.

[Sawchuk84] A. A. Sawchuk and T. C. Strand, "Digital Optical Computing," *Proc. IEEE*, Vol. 72, pp. 758-779,1984.

[Sawchuk85] A. A. Sawchuk and B. K. Jenkins, "Optical Cellular Logic Processors," Annual Meeting, Optical Society of America, Washington, D.C., October, 1985; *Journal of the Optical Society of America-A*, Vol. 2, p. P22, 1985.

[Serra82] J. Serra, *Image Analysis and Mathematical Morphology*, Academic Press, Inc., New York, 1982.

[Shankoff68] T. A. Shankoff, "Phase Holograms in Dichromated Gelatin", *Applied Optics*, Vol. 7, No. 10, pp. 2101-2150, 1968.

[Sharfin86a] W. F. Sharfin and M. Dagenais, "Femtojoule Optical Switching in Nonlinear Semiconductor Laser Amplifier", *Appl. Phys. Lett.,* Vol. 48, pp. 321-322, 1986.

[Sharfin86b] W. F. Sharfin and M. Dagenais, "High Contrast, 1.3 μm Optical AND Gate with Gain", *Appl. Phys. Lett.,* Vol. 48, pp. 1510-1512, 1986.

[Smith69] H. M. Smith, *Principles of Holography,* Wiley, New York, 1969.

[Smith81] P. W. Smith and W. J. Tomlinson, "Bistable Optical Devices Promise Subpicosecond Switching", *IEEE Spectrum,* Vol. 18, No. 6, pp. 26-33, June 1981.

[Smith87] S. D. Smith, et al, "Restoring Optical Logic: Demonstration of Extensible All-Optical Digital Systems", *Optical Engineering,* Vol. 26, No. 1, pp. 45-52, 1987.

[Sternberg82] S. Sternberg, "Biomedical Image Processing," *IEEE Computer*, Jan. 1982, pp. 22-34.

[Sternberg85] S. Sternberg, "An Overview of Image Algebra and Related Architectures", in *Integrated Technology for Parallel Image Processing*, Edited by S. Levialdi, Academic Press, New York, 1985, pp.79-100.

[Sternberg86] S. R. Sternberg and J. Serra ed., "Special Section on Mathematical Morphology", *Computer Vision, Graphics, and Image Processing*, Vol. 35, 1986, pp. 273-383.

[Stout88] Q. F. Stout, "Supporting Divide-and-Conquer Algorithms for Image Processing",*Journal of Parallel and Distributed Computing*, Vol. 4, No. 1, pp. 95-115, 1987.

[Stucki79] P. Stucki, *Advances in Digital Image Processing*, Plenum Press, 1979.

[Taboury87] J. Taboury, et al, "Cellular Optical Processor Architecture with Modulable Holographic Interconnections", *Topical Meeting on Optical Computing*, Technical Digest Series 1987, Vol. 11, Optical Society of America, Washington, D.C., pp. 31-34, March, 1987.

[Tai79] A. Tai, et al, "Optical Residue Arithmetic Computer with Programmable Computation Modules," *Applied Optics*, 18, p. 2812 , 1979.

[Tanida85] J. Tanida and Y. Ichioka, "Optical-logic-array Processor Using Shadowgrams. II. Optical Parallel Digital Image Processing," *J. Opt. Soc. Am. A*, Vol. 2, No. 8, pp. 1237-1244, 1985.

[Tao86] T. T. Tao and D. M. Campell, "Multiple-valued Logic: An implementation," *Optical Engineering*, Vol. 25, No. 1, p.14, 1986.

[Taso87] M. T. Taso, et al, "Symbolic Substitution Using ZnS Interference Filters", *Optical Engineering*, Vo. 26, No. 1, January, pp. 41-44, 1987.

[Ulam62] S. M. Ulam, "On Some Mathematical Problems Connected with Patterns of Growth of Figures", *Proc. Symposia Appl. Math.*, Amer. Math. Soc., Vol. 14, pp. 214-224, 1962.

[Ullman84] J. D. Ullman, *Computation Aspects of VLSI*, Computer Science Press, 1984.

[Unger58] S. H. Unger, "A Computer Oriented Toward Spatial Problems," *Proc. IRE*, Vol.46,1958, pp. 1744-1750.

[Unger59] S. H. Unger, "Pattern Detection and Recognition," *Proc. IRE*, Vol. 47, 1959, pp.1737-1752.

[Verbeek84] P. W. Verbeek, "Implementation of Cellular-Logic Operators Using 3*3 Convolution and Table Lookup Hardware," *Computer Vision, Graphics, and Image Processing*, Vol. 27, 1984, pp. 115-123.

[West87] L. C. West, "Picosecond Integrated Optical Logic," *Computer*, pp. 34-46, December 1987.

[Wheatley87] P. Wheatley, et al, "Three-terminal Noninverting Optoelectronic logic device", *Optics Letters*, Vol. 12, No. 10, pp. 784-786, 1987.

[Wolfram84] S. Wolfram, "Cellular Automata as Models of Complexity", *Nature*, Vol. 311, No. 4, 1984.

[Wolfram86] S. Wolfram, *Theory and Application of Cellular Automata*, World Scientific, Singapore, 1986.

[Yatagai86] T. Yatagai, "Cellular Logic Architectures for Optical Computers," *Applied Optics*, Vol. 25, pp. 1571-1577, 1986.

[Zhuang86] X. Zhuang and R. M. Haralick, "Morhological Structural Element Decomposition", *Computer Vision, Graphics and Image Processing*, Vol. 35, pp. 370-382, 1986.