# COMPUTATIONAL STRUCTURES FOR EXTRACTING EDGE FEATURES FROM DIGITAL IMAGES FOR REAL-TIME CONTROL APPLICATIONS

Thesis by

Vincent Sydney Wong

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

1979

(Submitted March 14, 1979)

# ACKNOWLEDGEMENTS

Firstly, I am indebted to my advisor, Professor Carver A. Mead, for his patience and guidance in allowing me to finish this work. I would also like to thank Dr. William M. Whitney of the Jet Propulsion Labs for his encouragement and support while this work was in progress.

There are many other individuals who have helped me in my graduate career, and I am grateful to them also. Special thanks go to Ray Eskenazi of JPL for his assistance in various aspects of my work. Among the others are John Gray, Bill Heller, Floyd Humphrey, Mike Hyson, Rod Masumoto, Virginia Matthews, Jim McCaldin, Dave Middlebrook, Scott Roth, and Ivan Sutherland, from Caltech; Marge Brandenberg, Bob Cunningham, Len Friedman, Tom Griswold, Marc Raibert, and Don Williams from JPL; and Dr. Graham R. Nudd of Hughes Research Labs for discussions and data on his work in image processing.

Financial assistance from Caltech and the JPL Director's Discretionary Fund is gratefully acknowledged. Also, the use of facilities at Caltech and at the JPL Robotics Labs were crucial in developing this work.

# ABSTRACT

This thesis investigates extracting edge features from digital images for real-time control applications. A sequence of image operations is presented for extracting edge features from a digital image. This sequence emphasizes intensity edges, isolates them from the rest of the image and then converts them into line data structures for further analysis by a higher-level vision processor. This way, an image of 500 x 500 pixels is reduced to a few hundred data elements.

In order to extract edge features from digital images fast enough for real-time use, some consideration has been given to the computational structures for performing the edge-extraction operations. Past attempts at constructing parallel processing arrays to speed up image processing are summarized. From examining these past efforts, it is concluded that array processing is unsuitable for real-time image processing. One of the reasons for rejecting the use of array processors is the difficulty in constructing arrays large enough to cover reasonable-size images. A study reveals that this difficulty will remain even when ultimate LSI gate densities are used in

constructing the arrays.

Specialized processors are found to be more suitable for real-time image processing. From examining the computational characteristics of the edge-extraction algorithms, it was found that all but one of them could be suitably executed by pipeline processors. The exception is the chain-coding process, which is more suitably performed by a random-accessed structure. To demonstrate methodology and feasibility, a pipeline unit to perform the edge-enhancement operation in real-time has been designed and constructed using MOS LSI technology.

A vision system for real-time control purposes must be able to track moving objects. A brief scenario for using the edge features extracted from a sequence of images to track a moving object is presented in the last chapter of this thesis.

CONTENTS

1. Introduction ...1

1.1 Thesis Overview ...5

1.2 Comparing the Performance of a Machine Vision System with that of the Human Eye ...7

1.2.1 Geometrical Resolution and Image Size ...8

1.2.2 Frame Rate ...10

1.2.3 Rate of Feature Extraction from Images ...12

1.2.4 The Human Eye as a Computational Structure ...13

1.3 Parallel Array Structures for Image Processing ...15

1.4 General Comments on Parallel Array Computer Systems for Image Processing ...28

1.5 Summary ...34

2. Extracting Edge Features From Digital Images for Real-Time Control Applications ...36

2.1 Extracting Intensity Edges from Digital Images as Features for Real-Time Control Applications ...37

2.2 The Image Processing Operations for Extracting Intensity Edge Features from Digital Images ...39

2.3 Description and Simulation Results for the Image Processing Operations for Extracting Intensity Edges

from Digital Images ...41

2.3.1 Smoothing ...45

2.3.2 Edge Enhancement ...48

2.3.3 Dynamic Thresholding ...51

2.3.4 Thinning ...55

2.3.5 Identifying Node and End Points ...62

2.3.6 Chain Coding ...64

2.3.7 Generating Line Data Structures ...68

2.4 Extracting Edge Features from Other Scenes ...75

2.5 Summary ...86

3. Computational Structures for Performing the Edge-Extraction Operations in Real-Time ...88

3.1 Computational Characteristics of the Edge-Extraction Operations that Make Them Suited for Execution by Pipeline Processors ...89

3.2 Implementing a Pipeline Processor for the Edge-Enhancement Operation Using n-MOS LSI technology ...94

3.3 A Pipeline Approach to Chain-Coding ...107

3.4 A Random-Access Structure for Performing Chain-Coding Rapidly ...113

3.5 Summary ...121

4. Impact of Future Technological Advances on Image

Processing ...123

4.1 Advances in Fabrication of Micro-Electronic Circuitry ...124

4.2 Effect of Increasing Chip Complexity on Computational Structures for Image Processing ...127

4.3 Comparing CCD Analog Techniques for Image Processing with Digital Methods ...128

5. Conclusions and Directions for Further Research ...131

5.1 A Scenario for Tracking a Moving Object Using Edge Features Extracted from a Sequence of Digital Images ...133

Appendix A - Performing Edge-Detection on the Log of an Image Function ...139

Bibliography ...142

# CHAPTER 1
# INTRODUCTION

There are many applications where it is desirable to use a computer-based vision system to provide corrective feedback for real-time control of effectors such as manipulators and guidance systems. To accomplish this, the vision system must be able to track in real-time, moving objects in a scene. This work will address the problem of extracting features from a sequence of digital images of a dynamically changing scene so as to make possible the tracking of a moving object.

The problem to be solved can be illustrated in the following manner: suppose one were to use an electronic camera to photograph the motion of a box rotating and moving through space, how would one use a machine to analyze the series of images fast enough to provide the information needed to guide a mechanical arm to grasp the box? It may be assumed that the machine understands what a box is (i.e., it has a stored representation of a box) so that it can recognize the box in the scene.

Among other things, this work will treat the following topics:

- Summarize previous efforts at building computing machines for speeding up image analysis. Comments will be made on the strong and weak points of these attempts.
- Present a sequence of image operations for extracting edge features from a digital image for tracking moving objects.
- Examine the computational structures that will suitably perform the edge-extraction operations in real-time.
- Present the design of a pipeline unit, using n-MOS LSI technology, for one of the edge extraction operations - edge enhancement, to demonstrate methodology and feasibility.
- Predict the impact of some future advances in micro-electronics on image processing.

## EXAMPLES OF APPLICATIONS WHERE A REAL-TIME MACHINE VISION SYSTEM WOULD BE USEFUL

In recent years there has been considerable interest in using machine vision systems to process the voluminous amounts of data being taken in by imaging sensors of various kinds. Some areas where a machine vision system would be useful are:

### (A) AUTONOMOUS NAVIGATION AND GUIDANCE

In an environment that is hazardous to humans (such as in deep space where remote control of vehicles would also be difficult) it would be desirable to have a semi-autonomous vehicle that can locate obstacles in its path and navigate between specified locations using visual cues. The vision operations to be performed include tracking obstacles and other objects in the scene. Processing must be in "real-time" as determined by the speed of the vehicle.

### (B) AUTOMATED ASSEMBLY OF MACHINE PARTS

There are some industrial tasks that are hazardous or tedious for humans to perform (forming steel, for example, or bolting machine parts together). A

machine vision system could be used to guide mechanical arms to accomplish such tasks. Some of the vision operations to be performed here include identifying the machine parts and tracking them in real-time.

(C) INSPECTION OF FINISHED PRODUCTS

This task is routinely performed by humans to ensure that finished products meet specifications (e.g., ensuring that LSI chips are properly mounted in their packages). This job can be quite tedious and boring, and often leads to eye strain and errors. A machine vision system could be trained to locate and identify the finished products and measure their critical dimensions to ensure they satisfied specifications.

The above list is by no means exhaustive. Certain applications would require performing a combination of tasks. For example, a sighted robot should be able to locate and track moving objects for purposes of navigation and also identify common objects in its environment.

## 1.1 THESIS OVERVIEW

In the next section I will compare the performance of a machine vision system with that of the human eye. In Sec.1.3 a summary of past attempts at building machines to speed up image processing is given. Some comments and observations on the successes and failures of these attempts follow in Sec.1.4.

A sequence of image operations for extracting edge features from digital images is presented in Chapter 2. This sequence converts a grayscale digital image into a line drawing represention of intensity edges in the image. Sec.2.3 describes the sequence of operations, and presents some results of simulations performed on the JPL robot's vision system.

Chapter 3 examines the computational structures that will execute the edge-extraction operations in real-time. Pipeline processing is proposed as a suitable structure for performing some of the edge-extraction operations. For another of the operations - chain coding, a random-accessed structure is deemed more suitable. The design of a pipeline unit in n-MOS LSI technology for the edge-enhancement algorithm is presented in Sec.3.2. This is to demonstrate

methodology and feasibility.

Chapter 4 offers some predictions on the impact of advances in micro-electronics on machine image processing. The question of whether increasing LSI gate densities will permit the construction of sufficiently large parallel arrays for image processing will be examined here. This chapter also offers a comparison between digital techniques for image processing and a CCD analog approach being taken by researchers at Hughes Research Labs.

Chapter 5 summarizes the conclusions drawn from this thesis and points to further work that needs to be done to realize a fully operating real-time vision system. A scenario for using the edge features extracted from a sequence of images to track a moving object is presented in Sec.5.1.

The following list of keywords are defined here as they are used in the remainder of this work. The keywords pertain to visual tasks that are performed by a machine vision system with respect to the projection of an object in a scene.

-ACKNOWLEDGE: to recognize the presence of an object

without necessarily naming it.

-CHARACTERIZE: to form an internal representation of an object based on measured attributes of its projection.

-CLASSIFY: to place an object in a certain category based on some measured attributes of its projection.

-DESCRIBE: same as to characterize an object.

-IDENTIFY: to name an object based on comparing some measured attributes of the object's projection with those of stored models.

-ISOLATE: to separate an object's projection from the rest of the image.

-LOCATE: to recognize and determine the position of a named object from a number of other objects in a scene.

-RECOGNIZE: same as to identify an object.

-TRACK: to specify the position and orientation of an object in each of a sequence of images.

## 1.2 COMPARING THE PERFORMANCE OF A MACHINE VISION SYSTEM WITH THAT OF THE HUMAN EYE

This section provides a rough comparison of the

performance of a simple imaging system (Fig.1.2.1.1) with that of the human eye. The photo-array in the imaging system is assumed to have approximately 500x500 photo-elements, covering an area about 1 cm square.

## 1.2.1 GEOMETRICAL RESOLUTION AND IMAGE SIZE

The geometrical resolution of an imaging system may be important when texture processing is included in scene analysis. The resolution of present-day imaging systems is no where near that of the human retina, although the use of zoom lens for magnification could compensate for this.

The geometrical resolution of the simple imaging system shown in Fig.1.2.1.1 could be estimated from the following analysis. Distances L1 and L2 are related to the lens focal length f, by the relationship

$$1/f = (1/L1) + (1/L2)$$

If L2 is much greater than f, then L1 ~ f. The spacing between pixels on the photo-array is d, approximately 1cm/500 pixels, or 2 x 10**(-3) cm. The angular resolution, r, of the photo-array is given by

$$r = 2d/L1 \sim 2d/f, \qquad L2 >> f.$$

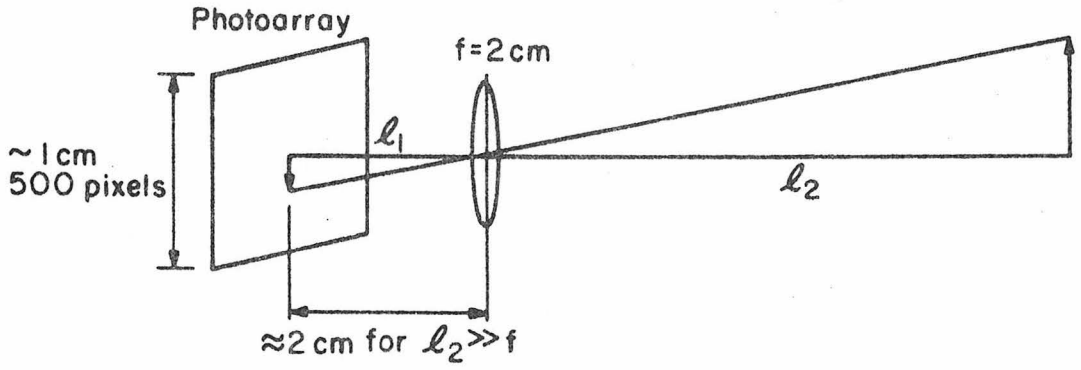



Fig.1.2.1.1 Simple imaging system

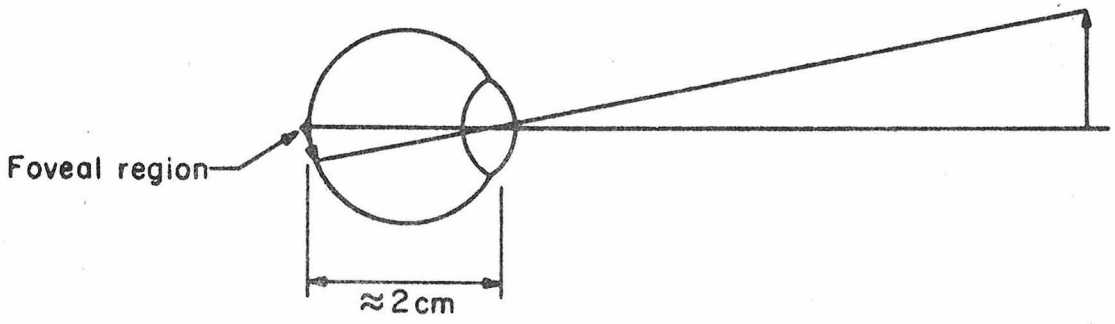



Fig.1.2.1.2 Imaging system of human eye

For a focal length f = 2 cm, r ~ 2 x 10**(-3) rads (or 6.8 arc-mins).

The foveal region of the human eye has a visual acuity of approximately 10 arc-secs, about 40 times better than the imaging system shown in Fig.1.2.1.1, and subtends about 1 arc-deg at the iris. This is equivalent to resolving 200 lines/cm [fn.1] in a field of view spanning 1.75 cm at a distance of 1 m. In contrast, the photo-array in Fig.1.2.1.1 is capable of resolving 6.8 arc-mins in an area subtending 28.0 arc-degs at the lens. This is equivalent to resolving 5 lines/cm in a field of view spanning 50 cm at distance 1 m. To achieve a resolution equivalent to that of the eye, the image on the photo-array would have to be magnified 40 times, with a subsequent reduction of light intensity at the image of 1600 times.

## 1.2.2 FRAME RATE

When using a machine to track a moving object, one may examine the extracted features from a series of images

[fn.1] A "line" consists of a black and white stripe of equal width.

taken of the scene in rapid succession, and then relate the position of the moving object to that of the extracted features. (Chapter 5 presents a scenario for tracking a moving object through a sequence of images.) This mode of tracking is possible only if the object's extracted features from one frame can be matched with those from the previous frame. If the object is moving fast enough so that the extracted features in one frame is substantially different from those in the previous frame, it will not be possible to match the extracted features, and the moving object will be momentarily lost. Scene analysis to locate the moving object will then have to be performed again. Scene analysis, however, is usually a time consuming process and difficult to accomplish in real-time. To avoid having to perform scene analysis repeatedly, the frame rate must be fast enough so that the extracted features from successive frames can be matched.

The factors that cause the extracted features of a moving object to change from 1 frame to the next include the changing viewing angle of the object and the change in angle of reflection of incident illumination to the viewer. Evidently, the faster the moving object, the greater the rate of change of the extracted features between frames. To increase the probability of successfully matching features

between frames, one may wish to limit the velocity of the moving object for a given frame rate.

The success of matching extracted features from successive frames also depends critically on the local uniqueness of the features. If the features are sufficiently unique, then the features may be displaced several pixel distances between frames, and yet it would be possible to match them.

### 1.2.3 RATE OF FEATURE EXTRACTION FROM IMAGES

When using a machine vision system for real-time control applications, the useful image features must be extracted from each of a sequence of images of the scene fast enough to enable an effector to interact effectively with the objects in the scene. A delay on the order of a frame interval in extracting features is probably tolerable in most real-time applications. If an object is moving at 30 cm/sec at 10 m and the frame rate is 30 frames/sec, then in 1 frame interval delay, the object will have moved about 1 cm.

The human visual system probably does not have a

similar discrete frame rate. Experimental facts obtained from selectively processing different attributes of a visual impression, such as color, contour, brightness, motion, and depth indicate that input stimulus is continuous, with some of the visual tasks being performed independently of the others. As stated by Kolers (p.34, [KOL.68]):

> "...It would be better still to say that there are not quanta at all, but that different perceptual processes take different amounts of time to occur and that some of them run off somewhat in parallel..."

Many of the visual processes are also dependent on the intensity and duration of the stimulus. For the performance of a particular task, there appears to be a reciprocal relationship between intensity and duration of the stimulus (IxT = constant) for durations less than 300 ms.

## 1.2.4 THE HUMAN EYE AS A COMPUTATIONAL STRUCTURE

In studying computational structures for machine vision processing, comparisons will be made with the human visual structure. The retina has about 130 million receptor cells intricately connected to a multi-layered structure.

Some processing for contrast enhancement occurs in this multi-layered structure before the image data is sent to the brain via the optic nerve. In the brain the process of constructing lines, shapes and form, and the perception of movement takes place. Apparently there are very specific cells specializing in performing each of these tasks present in the brain.

The physiology of the human vision system has prompted several researchers to examine the possibility of using a machine to analyze a scene along the same lines. Marr at MIT, for example, has proposed a scheme for representing a scene in terms of low-level symbols [MAR.75]. He extracts these symbols by convolving a set of line, bar and corner masks, in different orientations, with the image. In this way, a "primal sketch" is obtained of the scene, without the use of higher-level knowledge.

Other researchers have attempted to construct parallel processing machines after the structure of the human retina for vision processing. The next section presents a summary of these efforts.

## 1.3 PARALLEL ARRAY STRUCTURES FOR IMAGE PROCESSING

As early as 1958 it was realized that sequential processors were inefficient for problems involving spatial relations [UNG.58]. In the field of image processing, for example, each image to be processed may contain as many as 2.5 x 10**(5) pixels (a 500x500 image), and the transformation of each image often involves computations at every pixel position. The computation at each pixel position may take several machine instructions, involving the accessing of the local neighborhood pixel values, arithmetic and logical operations, and storing of the result. With the typical processing speeds encountered in today's computers it takes from seconds to minutes to transform an image.

It is desirable to speed up image processing for many applications, and especially for real-time control. A number of attempts have been made in the past to construct computational structures to speed up image processing. Most of these attempts involve parallel array computation, following the structure of the human retina. The remainder of this section will summarize previous efforts at constructing these computational structures and point to some of their deficiencies.

In 1958, Unger proposed an array computer for solving problems that involve spatial relations [UNG.58]. His idea was to have a central control providing instructions for the operation of a large rectangular array of identical modules (Fig.1.3.1). Each module would be connected with its 4 nearest neighbors to allow data exchange, and each would possess a 1-bit accumulator, a small amount of random access memory and some associated logic. Unger showed that many interesting image processing functions could be performed using this structure. This architecture, also referred to as Single Instruction Multiple Data (SIMD) organization, has influenced the efforts of many later researchers in constructing image processing machines. Some of the machines that have been designed since then are for processing binary images only, such as the Binary Image Processor [GRA.71] and the Golay Logic Processor [PRE.71]. Some of the other more sophisticated image processors are briefly described here.

ILLIAC III [MCC.63]

This Pattern Recognition Computer was designed at the University of Illinois for automated analysis of binary images such as bubble chamber negatives. It has 3

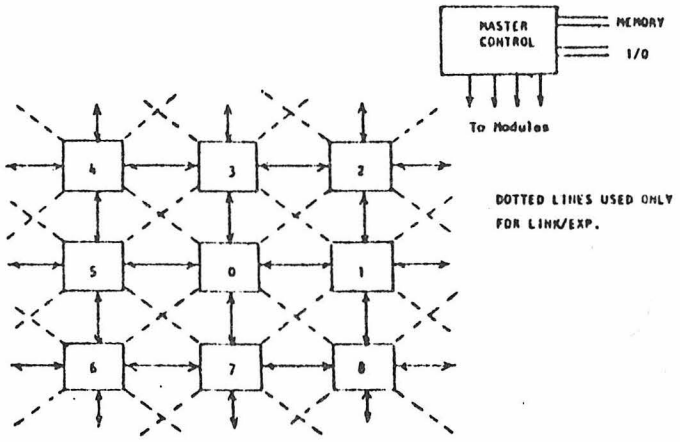



Fig.1.3.1 Module interconnections for Unger's machine

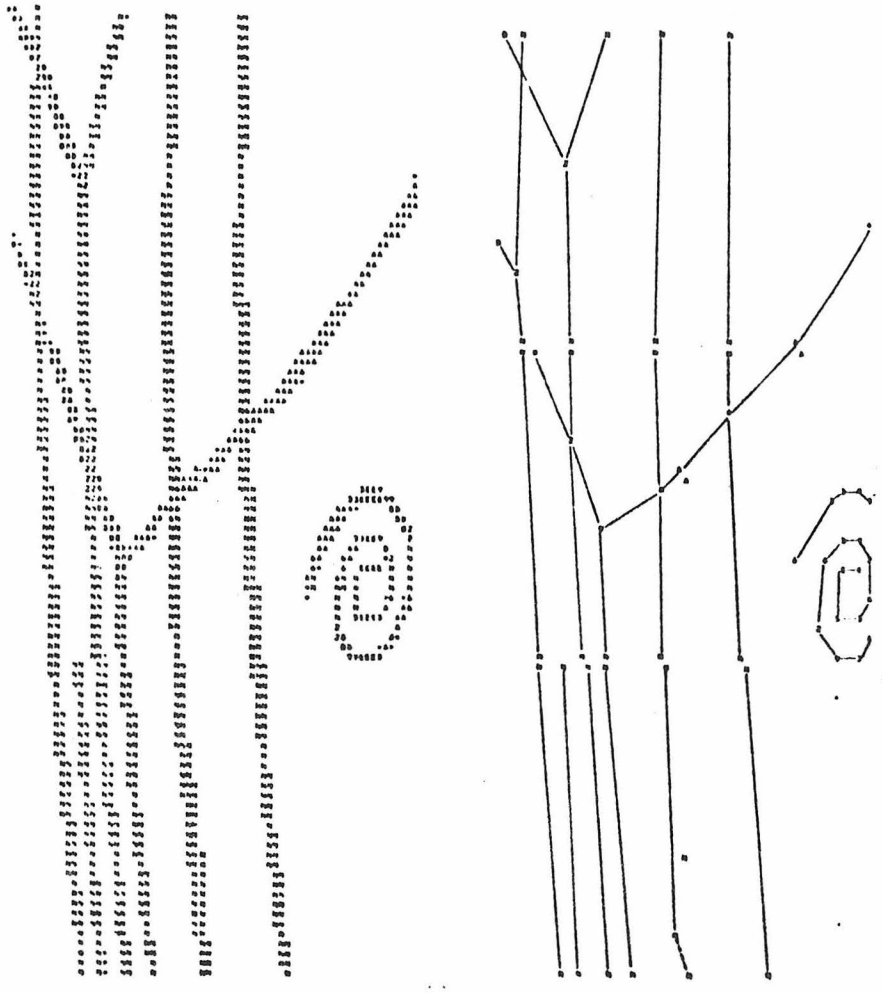


Fig.1.3.2 Constructing a linear graph (list) structure from binary pattern on Illiac III

computational structures for processing pictorial data - a Pattern Articulation Unit (PAU) for performing local preprocessing functions such as track thinning, gap filling, and line element recognition within a 32 x 32 window, a Taxicrinic Unit (TU) for manipulating and assembling the line elements generated by the PAU on a global scale, and an Arithmetic Unit (AU) for executing higher-level mathematical analysis (stereo reconstruction and statistical summary).

The PAU consists of a 32 x 32 element cellular array of processing modules (called stalactites), connected either as a rectangular or a hexagonal lattice. Each processing module is capable of performing logical functions, thresholding and signal path building. Each image to be processed is divided into 32 x 32 windows and these windows are processed sequentially. The operations within each window include edging, line thinning, gap filling and line smoothing. The objective of the PAU is to produce a line drawing of the binary pattern within the window (bubble chamber tracks for example). By labelling vertices, bends, crossovers and terminals this line drawing is then converted into a linear graph (a list) structure (Fig.1.3.2). This graph (or list) structure is then sent to the TU, whose function is to manipulate and organize the list structures from all the windows into descriptive categories, such as

tracks and electron spirals in bubble chamber exposures or alphanumeric characters of printed text. The tasks of the TU can be performed by a general-purpose sequential processor, though probably less efficiently, since the TU has been designed specifically to handle list structures. The AU handles higher-level statistical summary when necessary.

Little is known about the true capabilities of Illiac III because work on it stopped before it was completed (about 1967). One of the limitations of the system is the small processing capability of the PE's. It can handle only binary images and, moreover, even simple preprocessing tasks require numerous programming steps.

CLIP4 [DUF.76]

The CLIP series - CLIP being an acronym for Cellular Logic Image Processors - were designed by Duff and his colleagues at the University College, London, to study the use of parallel computer arrays for image processing. The latest version at the time of this writing is CLIP4, constructed using n-MOS LSI, incorporating 8 PE's on a chip. The PE's can be interconnected either as a rectangular or

hexagonal lattice and receive instructions from a master controller (a PDP 11/10). The computational hardware that is provided is suitable for processing binary images, although grayscale images may be processed at reduced speed in a bit-serial manner.

The block diagram of a PE is shown in Fig.1.3.3. A, B and C are single bits of storage while D is a 32 x 1 bit RAM. The Boolean Processor can form any of the 16 logical functions of A and P. The operations that can be performed by each PE include simple Boolean operations, propagating operations, and bit-plane arithmetic operations.

At the time of this writing, no results are available on the performance of the CLIP4 system. Simulations of CLIP4 operation for a 16 x 12 array size were carried out on CLIP3, an earlier version of CLIP4 built out of TTL logic. These simulations include thinning lines, finding edges in grayscale images, solving maze problems and producing histograms, perimeters and projections of particle areas. It is noted that these simulation runs involved lengthy execution times, partly because of the small array size (16 x 12) used to scan over the images. The designers expect that the overall processing speed for the CLIP4 system will increase considerably when a larger array

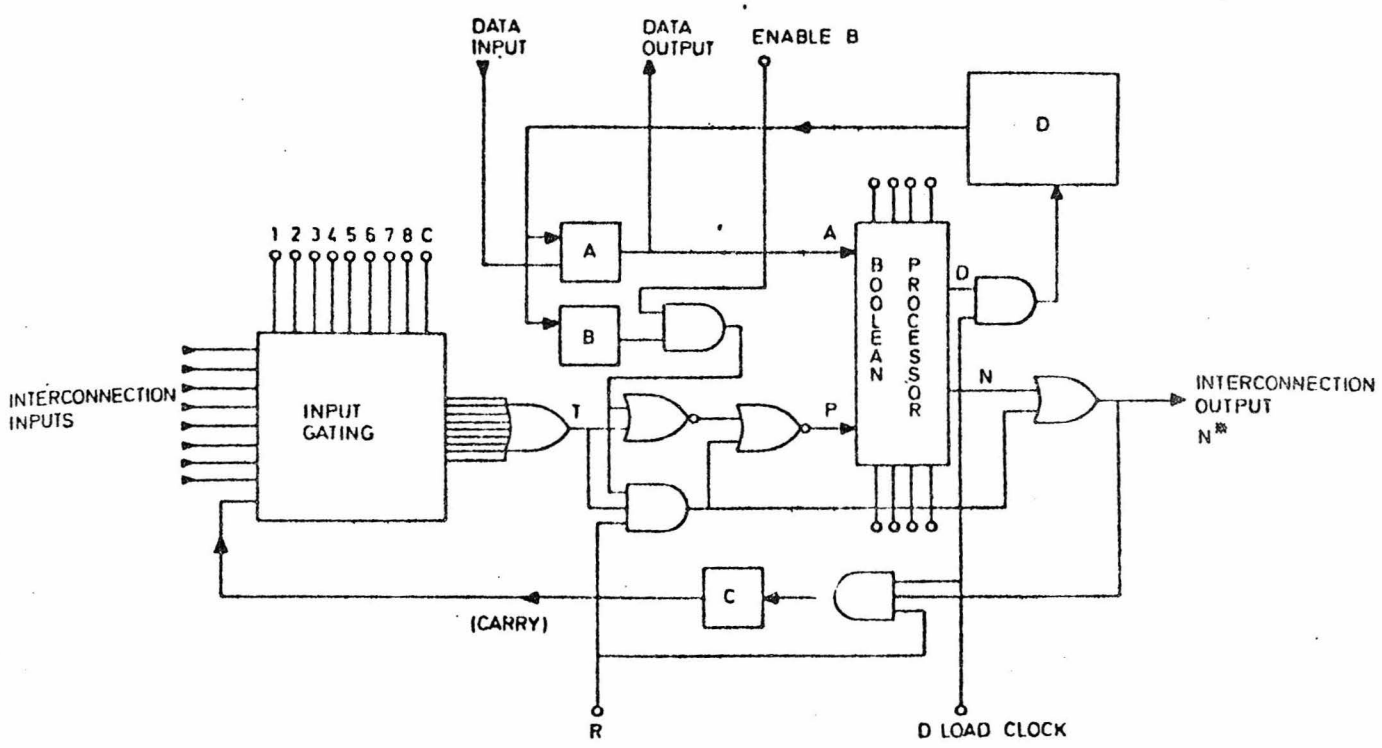



Fig.1.3.3 CLIP4 logic and memory structure

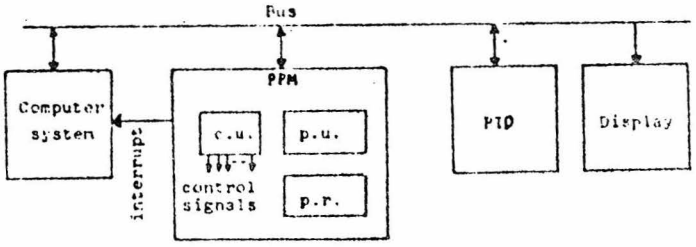



Fig.1.3.4 PPM image processing system

(perhaps as large as 96 x 96) is used. This is despite a five-fold reduction in operating speed of the individual CLIP4 modules (in going from TTL to n-MOS LSI technology).

PICAP and PPM [KRU.73,76]

PICAP is a modified version of the Parallel Picture Processing Machine (PPM) first proposed by Kruse in 1973. The system consists of a mini-computer controlling the PPM (Fig.1.3.4), and image input and display devices. The PPM has a processing and control unit that decodes instructions sent by the mini-computer over a bus. It returns an interrupt to the mini-computer when it is ready for the next instruction. There are 9 picture registers within the PPM, each capable of storing a 64 x 64 image of 4-bit pixels. Two line buffers are used to obtain 3 x 3 local neighborhoods from any of the 9 stored images for processing. The actual processing is performed by the neighborhood matching logic (NML) which may accept inputs from the line buffers or directly from the picture registers, depending on whether it is a local operation or a pointwise comparison of multiple pictures. Templates are stored in the control unit for matching operations.

The PICAP picture processing system has been in operation since 1975 and some of the applications that it has been used for are fingerprint coding, malaria parasite detection and printed circuit board inspection. The execution time per operation for a 64 x 64 pixel image is typically 2.5 ms. The total processing time for the applications just mentioned varies from several seconds to minutes, depending on the complexity of the operations to be performed.

TOSPICS [MOR.78]

TOSPICS (Toshiba Pattern Information Cognitive System) is a distributed image processor controlled by a TOSBAC-40C mini-computer (Fig.1.3.5). It has 7 hardwired modules for performing the basic image processing functions listed in Table 1.3.1. These hardwired modules receive instructions from the TOSBAC-40C over a bus system and generate interrupts to the mini-computer when the current operations are complete. By building on the basic image processing functions, more global and complex functions such as edge detection, texture analysis and shape identification can be performed. Processing of neighborhoods in the image array is sequential, but data access and computation at each

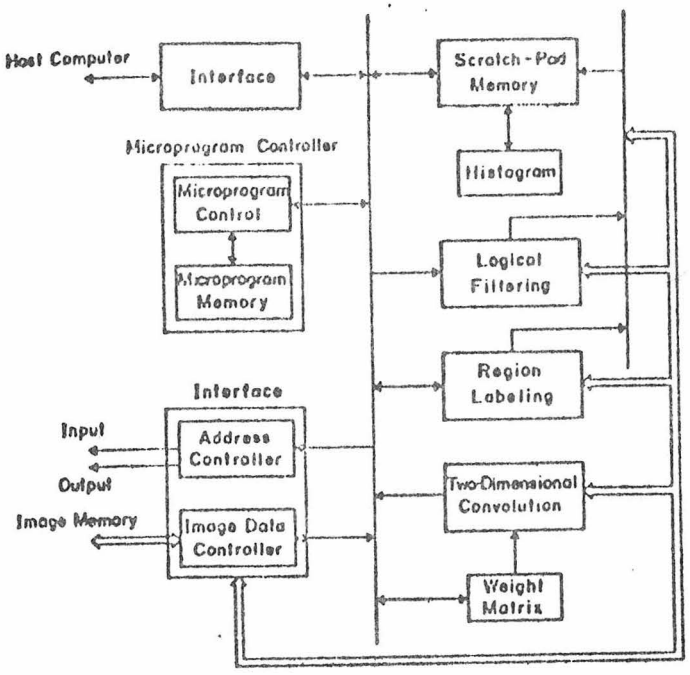



Fig.1.3.5 Structure of TOSPICS image processor

| Function | Applications | Execution Time |
|---|---|---|
| Two-Dimensional Convolution | Laplacian, Smoothing, etc | 262mS |
| Logical Filter | Feature Detection<br>Thinning | 262mS<br>262×W[1] mS |
| Region Labeling | Region Separation[2]<br>Particle Measurement[3] | 524mS<br>786mS |
| Data Conversion | $\gamma$-Correction<br>Thresholding | 262mS<br>〃 |
| Histogram Generation | Frequency Counting | 262mS |
| Affine Coordinate Transform | Enlargement with Rotation<br>Axis Skewing | 262mS<br>〃 |
| Pixel Operation | Logical Operation<br>Arithmetic Operation | 524mS<br>〃 |
| Gradient Operation | Edge Detection | 2.36S |

1) W is maximum line width
2) No of separated regions is less than 254
3) Region separation and area counting

Table 1.3.1 The basic image processing functions performed by TOSPICS and execution times

pixel position are performed in parallel.

Table 1.3.1 lists the execution times for the basic image processing functions. The execution times of more complex functions will be longer, depending on which of the basic functions are performed. Edge detection for a 512x512 image, for example, takes 2.36 seconds.

## OTHER PARALLEL COMPUTER ARRAYS

There are other parallel computer array structures that, although not constructed for image processing per se, have been programmed to do that. These include Illiac IV, the STARAN E processor and PEPE. Although these SIMD machines are not strictly image processors, they are mentioned here for completeness.

The ILLIAC IV array processor [BAR.68] was designed for solving scientific problems involving large matrices and systems of partial differential equations. The present configuration has 64 PE's connected as a linear array, with data routing of each PE to the 2 nearest neighbors and 2 others at a distance of 8 away (Fig.1.3.6). The PE's receive instructions from a master Control Unit, which in

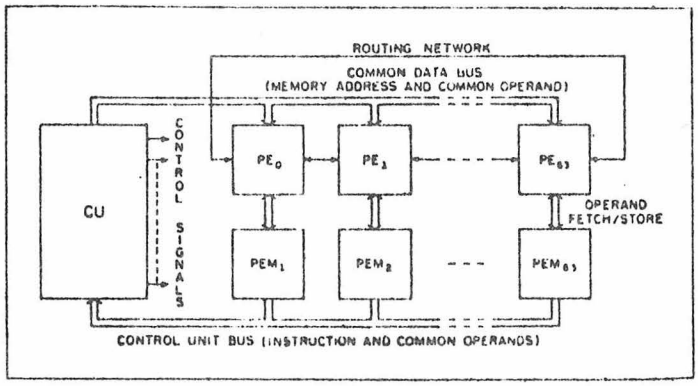



Fig.1.3.6 The ILLIAC IV structure

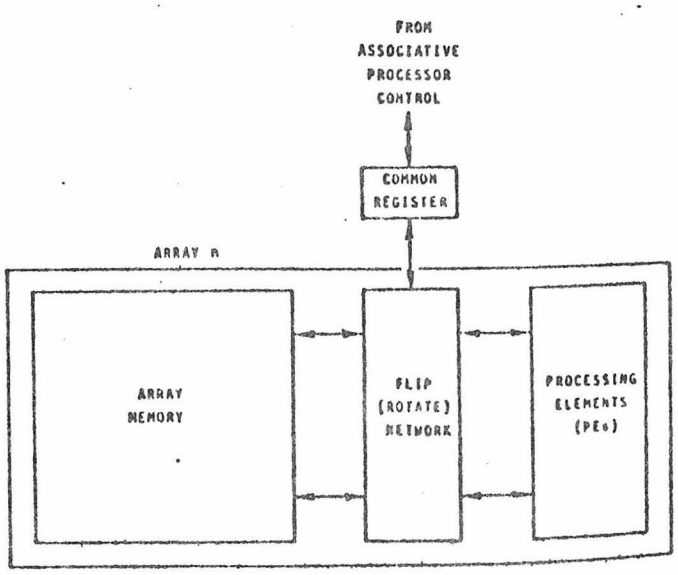



Fig.1.3.7 Structure of the STARAN E processor

turn is controlled by a B-6500 computer. Each PE has 2K x 64 bits of memory and an ENABLE flip-flop which determines if the current instruction is to be executed. The PE's have a powerful instruction set and are capable of performing floating point operations. The mismatch between the word length of this machine (64 bits) and the common 8-bit pixel representation in images causes some inefficiency when this machine is used for image processing.

The STARAN E processor [POT.78] is a SIMD machine with a host computer controlling from 1 to 32 arrays of PE's. Each array of PE's has 256 PE's, a FLIP (permutation) network and 256 x 9K of memory (Fig.1.3.7). The purpose of the FLIP network is to allow power of 2 data shifts between the array memory and the PE's. This is useful when performing Fourier analysis on images. Each PE has a bit register that is associatively addressed to determine if the PE is to execute the current instruction. Arithmetic is performed by the PE's in a bit-serial manner. The execution time for a 3 x 3 convolution operation on a 512x512 8-bit pixel image is 700 ms.

PEPE (Parallel Element Processing Ensemble) [ENS.74] is a large-scale associative processor developed in the early 70's by Bell Labs for the U.S. Army Advanced Ballistic

Missile Defense Agency to track objects in returned radar images. The current configuration has 288 PE's organized in 8 element bays (Fig.1.3.8). Each PE has an Arithmetic Unit (AU), a Correlation Unit (CU), an Associative Output Unit (AOU) and 1K x 32 bits of memory (Fig.1.3.9). The operation of the PE's are coordinated by a global Arithmetic Control Unit, a Correlation Control Unit and an Associative Control Unit, and these are in turn controlled by a CDC 7600 host computer. Each AU, CU and AOU of a PE has an associatively addressed register which determines if the unit is to execute the current instruction.

## 1.4 GENERAL COMMENTS ON PARALLEL ARRAY STRUCTURES FOR IMAGE PROCESSING

Here are some comments and observations drawn from studying parallel array structures for image processing.

The main reason for utilizing a parallel array structure is to achieve an increase in processing speed. This increase, however, is compromised by the fact that a multitude of programming steps is frequently required to execute even simple algorithms. There are two reasons explaining the large number of programming steps. In the

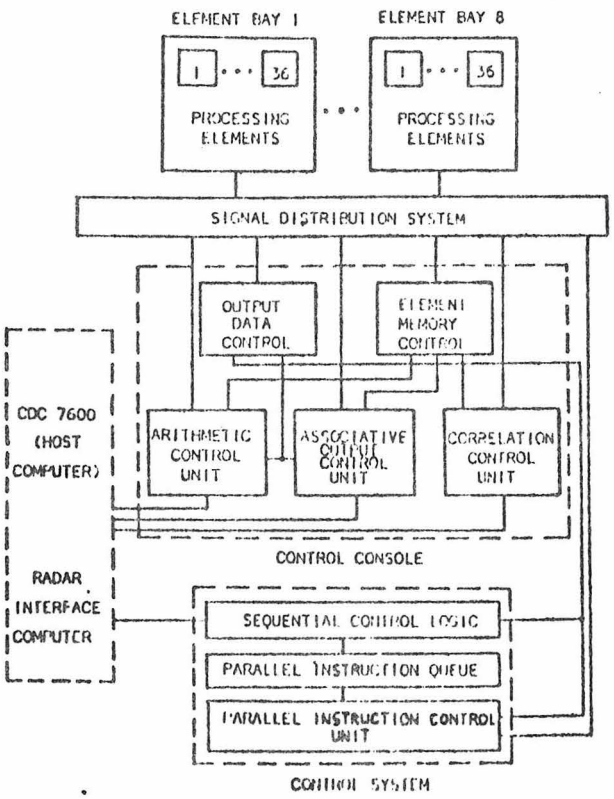



Fig.1.3.8 The PEPE associative processor

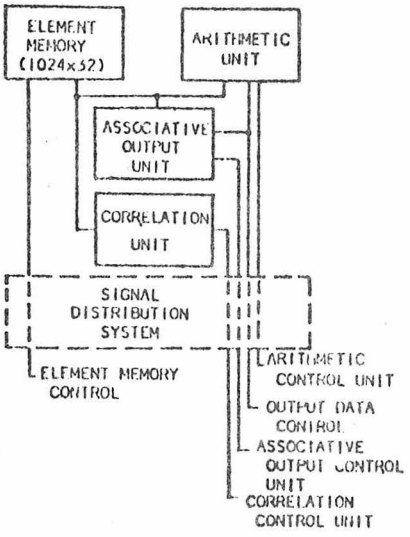



Fig.1.3.9 A processing element of PEPE

first place, to achieve a reasonably large array size, the arithmetic and logical capabilities of each PE are kept conveniently small, so that even relatively simple operations, such as addition, require several steps. The ADD operation in CLIP4 and STARAN E, for example, is performed in bit-serial manner. Another operation that increases programming steps is data access from the surrounding neighbors. Though most of the parallel structures considered are connected to the 4 nearest neighbors, data access from the nearest neighbors is performed serially to reduce the number of interconnections between neighbors. As stated by Kruse (p.1024, [KRU.78]):

> "...In processors where parallelism on the picture level is used, corresponding measurements have to be extracted by a cumbersome programmed procedure. The number of program steps for such an extraction may be several orders of magnitude larger than the initial number of processing steps that produced the feature to be extracted. Therefore the high inherent speed of truly parallel machines in picture transformations is very much impaired in pattern recognition applications..."

The suitability of using parallel array machines to

perform image processing operations for feature extraction can also be questioned. In extracting features from a digital image, some operations, such as smoothing and edge enhancment, involve computations that can be performed independently and concurrently at every pixel location. Others are sequential in nature. Chapter 2 presents a sequence of operations for extracting edge features from an image. Of these, chain coding is of a sequential nature. A parallel array structure may be used for computations that can be performed concurrently, but is poor at handling computations of a sequential nature. The designers of Illiac III recognized this distinction, and as a solution constructed a parallel array unit (the PAU) to handle concurrent operations (line thinning, gap filling) and devised two other units (the Taxicrinic Unit and the Arithmetic Unit) to handle operations of a sequential nature (assembling list structures, statistical summary).

Another factor that detracts from using a parallel array structure for real-time control applications is the overhead from loading and unloading the array with data. Loading and unloading of data from such arrays is often done in a serial manner because of practical limitations on the number of interconnects to these arrays. A 500x500 PE array would require 500x500x8 interconnections to allow fully

parallel data transfer. This many interconnections would require 3-D micro-structures to implement, a technology that is not here yet. Hence serial transfer of as many as 30 frames of image data per second is necessary. The PE's in the array are idle during data transfer periods, and as a result, the efficiency of the parallel array system goes down dramatically. This I/O overhead problem could be solved if the data transfer function and the computation function were separate, and could be performed concurrently.

Another practical problem encountered in using parallel array structures for processing pictures is the difficulty in constructing sufficiently large arrays. The number of modules and total number of interconnects between modules grow rapidly with increase in array size (the growth is n**(2), where n x n is the array size). Even when using LSI technology at today's gate densities, it is difficult to construct a sufficiently large array that will process reasonable-size images. In fact, a projection in Chapter 4 indicates that it would still be difficult to construct a sufficiently large array on a chip to cover reasonable-size images at ultimate LSI gate densities. Currently, an attempt is being made to use modules fabricated using n-MOS LSI technology to construct a 96 x 96 parallel array (CLIP4, with 8 PE's on a chip). The authors of CLIP4, Duff, et.al.,

have this to say (p.728, [DUF.76]):

> "...The obvious limitations of parallel processors is the need for very large numbers of processors in order to obtain adequate resolution over usefully large image areas..."

and (p.730, [DUF.76]),

> "...At a certain point...it no longer becomes sensible to consider building an array to cover the entire picture area to be processed..."

An alternative to building an array to cover the entire image to be processed is to build a smaller array and process the image in blocks as was done with Illiac III. This requires extra processing time, however, and the necessity to coordinate data access at the edges of the blocks.

One other concern with using a large array of PE's is the reliability of such a system. It is difficult to ensure that a large number of PE's will be working correctly at the same time without being able to monitor conditions arising from each PE. The correct programming and monitoring of a large array of PE's is non-trivial and a large amount of software support is usually necessary.

General programmability may be a factor in developing a machine structure for image processing. This feature may be emphasized if the image processor is to be used as a research tool and fast processing speeds are of secondary importance. This is not the case, however, in processing images for real-time control applications. In real-time applications, processing speed is of primary importance. Presumably, the same features are extracted from every image that is acquired. Because speed is of primary importance, and for the other reasons cited in preceding paragraphs, a parallel array structure is considered a poorer choice than specialized computational structures for real-time image processing. In later chapters, the image operations to be performed for feature extraction will be examined and special computational structures suitable for executing them in real-time will be proposed.

## 1.5 SUMMARY

Much of the work in artificial vision processing in the past have been directed at scene analysis for understanding. Researchers have wrestled with techniques for identifying and classifying objects and representing

objects in 3-D. In support of this work, there have been some attempts at constructing parallal array machines to speed up image processing. This chapter has summarized some of the efforts at building these machines. None of these efforts have, however, resulted in a machine capable of processing images for real-time control use.

When using a machine vision system for real-time control purposes, an important function of the vision system is to track moving objects in the scene. It is desirable, however, to perform scene analysis initially to identify and locate objects in the scene. Since scene analysis for object identification is usually a complex process, it is difficult to perform this rapidly enough for object tracking. Tracking moving objects in the scene, subsequently, involves extracting features from each of a sequence of images of the scene rapidly and relating the position of the object to those of its extracted features.

The next chapter presents a sequence of image processing operations for extracting edge features from digital images. A moving object may be tracked by following the positions of the extracted edge features from frame to frame. Chapter 5 presents a scenario for using the extracted edge features for object tracking.

# CHAPTER 2

# EXTRACTING EDGE FEATURES FROM DIGITAL IMAGES FOR REAL-TIME CONTROL APPLICATIONS

In this chapter I will present a scheme for extracting features from digital images for real-time control purposes. The features to be extracted are intensity edges in an image, and the image processing operations for extracting them will be described. Some of the image processing operations have resulted from the work of other researchers; I have taken the liberty to modify and adapt them to this work, while two of the operations, a dynamic thresholding process and the reduction of chain codes to straight lines representing intensity edges, are new techniques developed for this work.

In order to perform the edge-extraction operations fast enough for real-time control purposes, some consideration has been given to the algorithms and computational structures that will execute the operations. All but one of the operations can be performed sequentially by a raster scan of the image data from the previous operation. Pipelined units can be used to execute those operations that can be performed in raster order. The only

exception to raster order is the chain coding operation, where it is desirable to be able to access the image data randomly to form chain codes. More will be said about the computational structures for performing these operations in the next chapter.

## 2.1 EXTRACTING INTENSITY EDGES FROM DIGITAL IMAGES AS FEATURES FOR REAL-TIME CONTROL APPLICATIONS

We return to the problem of describing the motion of a rotating box by analyzing a sequence of digital images. To describe the motion of the box, some features that are related to the position of the box must be extracted from each of the images. Then by examining the positions of the extracted features from frame to frame, one may hope to describe the motion of the box.

What features could we hope to extract from a sequence of images that would enable us to follow the motion of the rotating box? The features that are used must be extracted rapidly enough for real-time use, must be easily manipulated by a higher-level vision processor, and must represent a considerable reduction of data from that contained in the original image.

With the preceding considerations in mind I have chosen to extract intensity edge features from a digital image for real-time control applications. Intensity edges are chosen as the low-level features because rapid spatial intensity changes in an image often correspond to edges or surface discontinuities in the scene. The notable exceptions are shadows and highlights. Hence intensity edges give a reasonable definition of the outlines of objects and surface features (changes in color or texture) in a scene. The extracted intensity edges are then used for object tracking. This task is made easier if a 3-D model of the object is available to the vision system.

Another reason for using intensity edges as low-level features is that they are easily extracted without the use of heuristics or world knowledge.

Not all intensity edges in an image are detected, because the illumination level in a scene will vary, and intensity edges in areas of low contrast will be lost. However this may not be a serious problem in tracking a moving object. If a stored model of the object is available, it is only necessary to track isolated features of the object in order to describe its motion.

## 2.2 THE IMAGE PROCESSING OPERATIONS FOR EXTRACTING INTENSITY EDGE FEATURES FROM DIGITAL IMAGES

A sequence of operations for extracting intensity edge information from digital images is presented here. The sequence of operations emphasizes rapid spatial intensity changes in a digital image and converts them to line data structures. This line structure is then passed on to a higher-level vision processer for further analysis (such as extraction of motion and shape).

The sequence of operations for extracting intensity edges is as follows:

1) Smoothing: This operation helps eliminate "salt-and-pepper" type noise in a digital image. Such noise may arise from a number of sources - the photosensors and A-to-D round-off errors are two of them. The presence of salt-and-pepper noise, if not eliminated, gives rise to spurious edge points at a later stage of processing.

2) Edge enhancement: A 2-dimensional gradient-intensity computation is performed on the digital image to emphasize rapid spatial intensity changes. The result is an edge-enhanced image composed of

gradient-intensity magnitude values at every point.

3) Dynamic thresholding: To locate the areas of rapid spatial intensity change, the edge-enhanced image is converted into a binary-valued image by thresholding. Since the contrast level may vary over an image, a single threshold value used over the entire image would produce a heavy concentration of edges (and noise) in areas of high contrast and would miss edges in areas of low contrast. A scheme using a threshold that depends on the local contrast would be more reliable and is described in Sec.2.3.3.

4) Thinning: This operation produces a single pixel-wide contour of an intensity edge from the cluster of edge points found from the thresholding operation. This thinning of the edge clusters is in preparation for a later operation - chain coding, which will collect together connected edge points.

5) Identifying node and end points: Node and end points (as defined in Sec.2.3.4) are logical starting and ending points for the chain-coding process. In order to identify these points, a 3 x 3 window is used to scan the thinned image. Whenever a node or end point is found, its location in the thinned image is stored in a list to be used

by the chain-coding process.

6) Chain-coding: The edge points resulting from the thinning process are isolated entities, and by themselves give no information as to the shape or size of objects. In order to form logical representations of intensity edges in the original image, the edge points that are connected are linked together in this process.

7) Generating line data structures: Intensity edges in the original image are now represented by chains. In order to reduce this collection of data further, and to form more global representations for the intensity edges, straight line segments are fit to the chain codes. The result is a collection of vectors representing intensity edges in the original image.

Fig.2.2.1 depicts the above sequence of operations.

## 2.3 DESCRIPTION AND SIMULATION RESULTS FOR THE IMAGE PROCESSING OPERATIONS FOR EXTRACTING INTENSITY EDGES FROM DIGITAL IMAGES.

The image processing operations for extracting

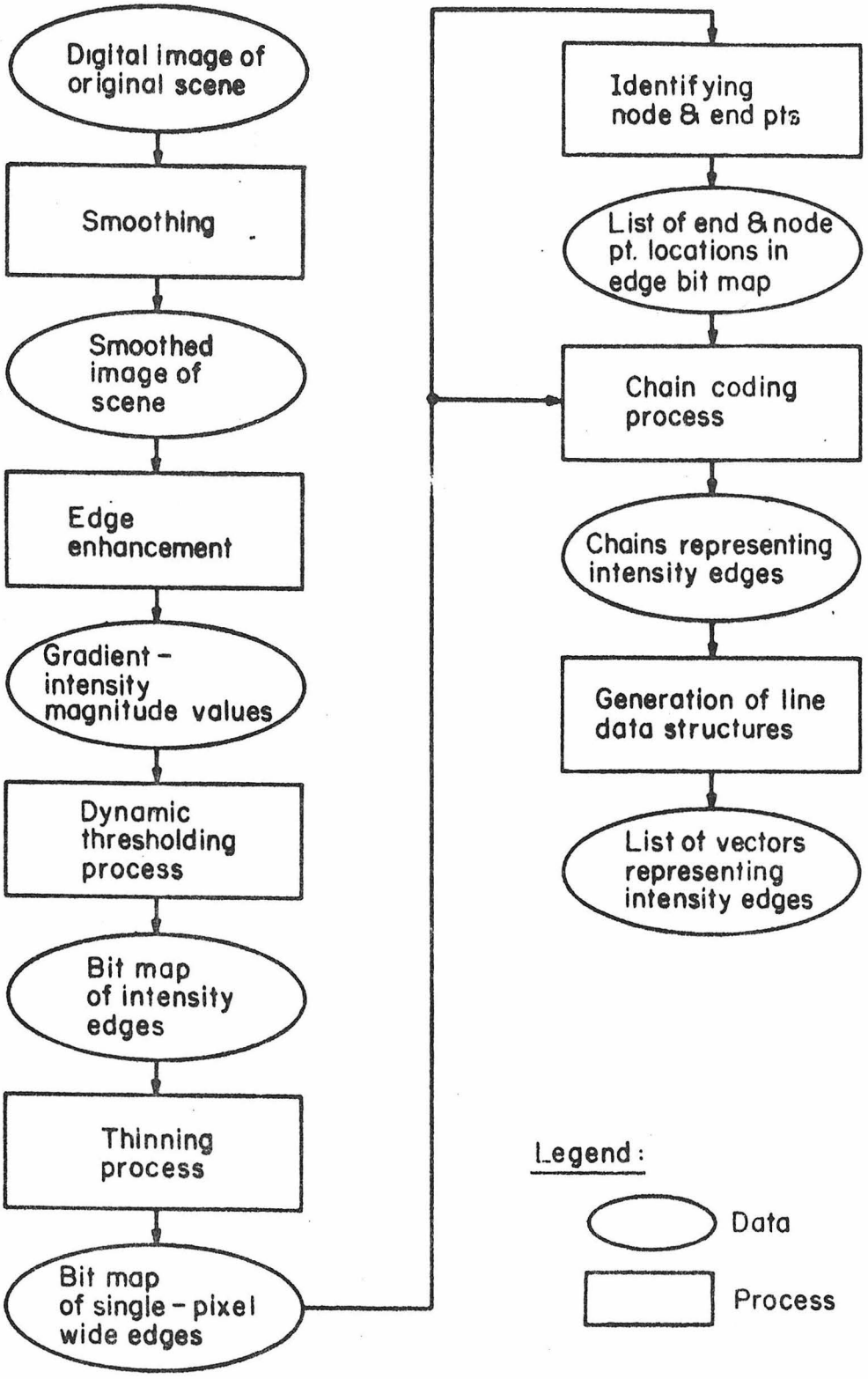



Fig.2.2.1 Image processing operations for extracting intensity edges from a digital image.

intensity edges from a digital image is presented in this section. These algorithms were investigated on the JPL robot's vision system. A brief description of this system follows.

The images are obtained from a GE TN-2000 CID (charge injection device) camera, with 188 x 244 photo-elements. The pixels from the camera are then digitized to 8 bits at TV frame rate. Appendix B shows the photoresponse of the CID imager. When the images were obtained, care was taken to adjust the iris of the camera so that the pixel values were in the non-saturated range of the imager as indicated in Fig.B.1. This adjustment was accomplished by examining the pixel histogram of the image, and ensuring that the pixel values were in the linear range of the imaging system, as indicated in Fig.B.2.

The digital images are stored on disk packs and transferred to a random-accessed memory array (RAPID) whenever they are to be processed. The image processor is a General Automation SPC-16 minicomputer, and depending on the complexity of the algorithm, it takes from 15 secs to 1 min to perform an operation on an image. The images being worked on are displayed on a CONRAC TV monitor and the illustrations in this chapter were obtained by photographing

the images on this monitor.

In each of the sub-sections that follow, a description is first given of the image operation, and then the algorithm used to perform it. The result of the operation on a test image, a HARDWARE scene shown in Fig.2.3.1, is presented. The test image is of a hacksaw, a magnetic tape and a carpenter's rule. The results of applying the edge-extraction operations to 3 other scenes will be presented in Sec.2.4.

### 2.3.1 SMOOTHING

Smoothing reduces "salt-and-pepper" type noise in a digital image. As mentioned previously, such noise can arise from a number of sources, such as defective photosensor elements and A-to-D approximation errors. Smoothing by replacing each pixel value with the average pixel value in a local neighborhood helps eliminate this noise.

In spatial filter theory, smoothing corresponds to low-pass filtering. The weights for a 1-dimensional digital filter that approximate a low-pass gaussian function is

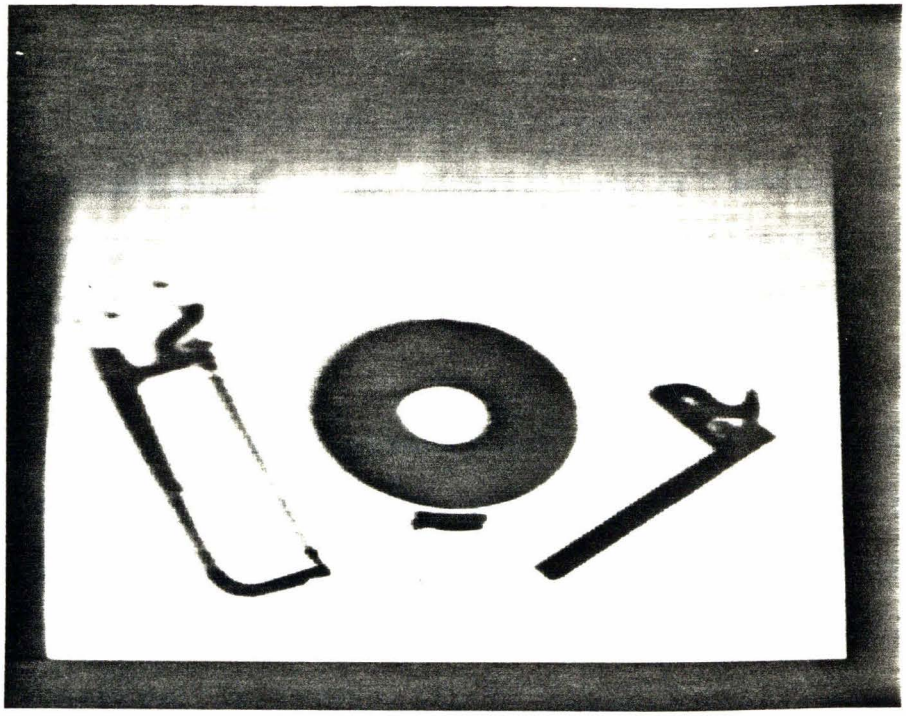


Fig.2.3.1 The HARDWARE scene

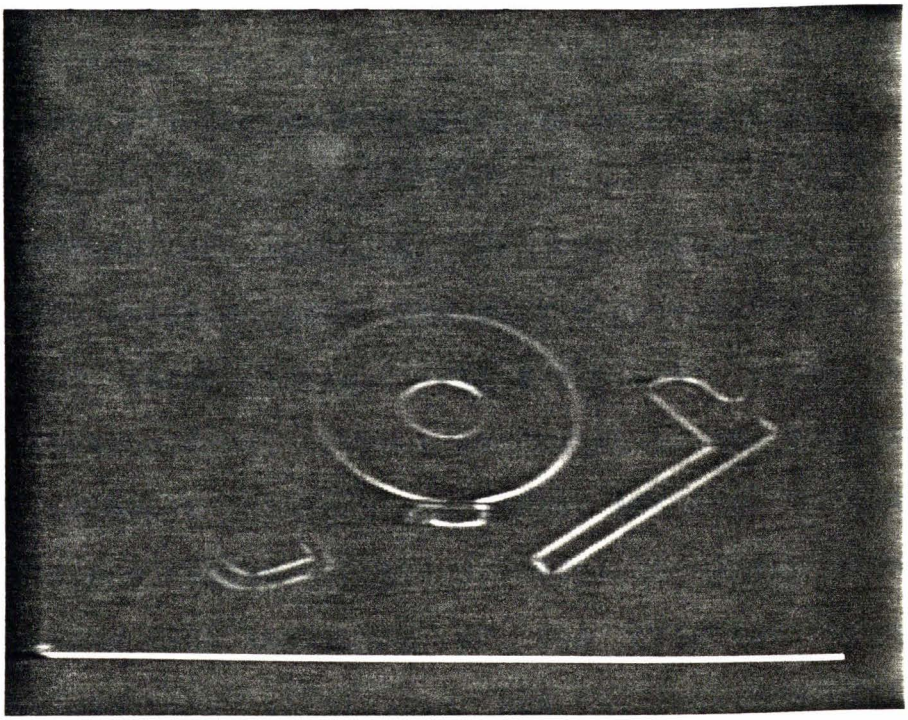


Fig.2.3.2.3 Edge-enhanced image of HARDWARE scene

shown in Fig.2.3.1.1(a), and its Fourier Transform shown in Fig.2.3.1.1(b). The weights for the low-pass filter have been chosen to attenuate the spatial noise around 0.5 cycles/pixel-distance (abbr. cy/pid) introduced by defective photosensors and round-off errors.

The weights for a 2-dimensional low-pass digital filter are chosen to approximate a 2-dimensional gaussian function, scaled to the smallest integer values (Fig.2.3.1.2). The smoothing algorithm, with a normalizing factor, may be written as (referring to Fig.2.3.1.3):

$$z := [4z + 2(b+d+f+h) + (a+c+e+g)]/16$$

The interested reader is referred to [ROS.76], [DUD.73], [GOO.68] and [AND.77] for more discussion on smoothing.

The effect of smoothing on the HARDWARE scene (Fig.2.3.1) is evident after the next 2 operations - edge enhancement and thresholding. Fig.2.3.3.4(a) shows the result after thresholding, without smoothing of the original image. Spurious noise points are present, and the contours of the objects are ragged. Fig.2.3.3.4(b) shows the result after thresholding with smoothing of the original scene. The noise points are eliminated and the contours of the objects are more rounded.

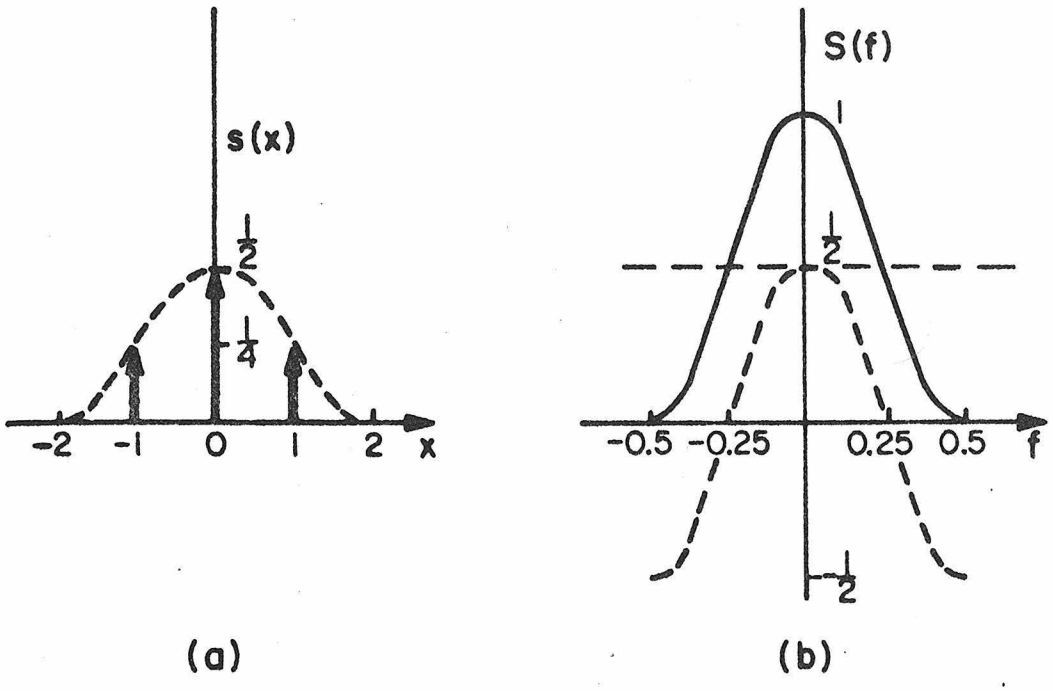



Fig.2.3.1.1 (a) The discrete low-pass filter (1-D), $s(x) = \frac{1}{2}\delta(x) + \frac{1}{4}[\delta(x-1) + \delta(x+1)]$. (b) The discrete Fourier Transform, $S(f) = \frac{1}{2}[1 + \cos(2\pi f)]$.

| | | |
|---|---|---|
| 1 | 2 | 1 |
| 2 | 4 | 2 |
| 1 | 2 | 1 |

Fig.2.3.1.2 The 3x3 window weights for smoothing.

| | | |
|---|---|---|
| a | b | c |
| h | z | d |
| g | f | e |

Fig.2.3.1.3 The lettering convention for pixel values in a 3x3 window.

## 2.3.2 EDGE ENHANCEMENT

The next step in extracting intensity edges from a digital image is to enhance the areas of rapid spatial intensity changes. For this purpose, high-pass filtering is applied to the image.

Fig.2.3.2.1(a) shows the weights for a 1-dimensional high-pass digital filter that approximate a sine function. The Fourier Transform of this filter is shown in Fig.2.3.2.1(b). The weights for the filter have been chosen so that its maximum response frequency is at 0.25 cycles/pid.

The 2-dimensional high-pass digital filter is obtained by measuring the partial derivative in 2 orthogonal directions, taking the magnitudes, and adding them together. This "poor man's" 2-D gradient operator is easy to implement in hardware, which makes it attractive to use. The algorithm for the 2-dimensional high-pass digital filter may be written as (referring to Fig.2.3.1.3):

$$HI.PASS(z) = [|(c+d+e)-(a+h+g)| + |(g+f+e)-(a+b+c)|]/4$$

The frequency analysis shown in Fig.2.3.2.1(b) for the 1-dimensional high-pass filter does not account for the

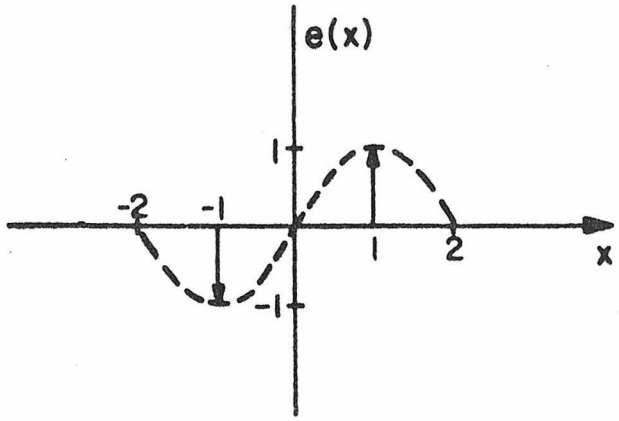



Fig.2.3.2.1(a) The high-pass filter function (1-D), $e(x) = \delta(x-1) + \delta(x+1)$.

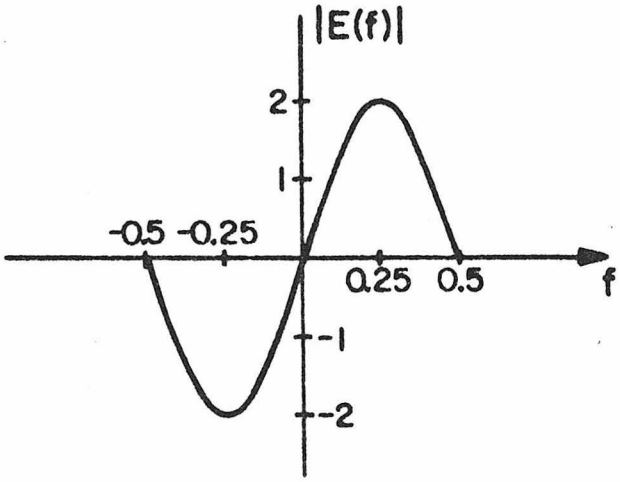



Fig.2.3.2.1(b) Magnitude plot of Fourier Transform of high-pass filter, $E(f) = 2\sin(2\pi f)\exp(i\pi/2)$.

magnitude operation, and provides a crude idea of the performance of the 2-dimensional high-pass filter in the frequency domain.

The high-pass filter is quite similar to an edge-enhancement operator commonly used in pattern recognition, called the Sobel operator (p.6, [BUL.74]):

$$SOBEL(z) = |(c+2d+e)-(a+2h+g)| - |(g+2f+e)-(a+2b+c)|$$

The performance of the high-pass filter and the Sobel operator are quite similar. Many other edge operators are known for detecting intensity edges in digital images. Among these are the Heuckel operator [HEU.71], Kirsch operator, and the Roberts cross operator (see p.275f of [ROS.76], p.267f of [DUD.73], and [BUL.74] for a discussion of these edge operators). The window sizes used by these operators range from a 5 to 6-pixel radius disk for the Hueckel operator to a 2 x 2 window for the Roberts cross operator. In choosing an edge operator for real-time image analysis, the following factors were taken into consideration - sensitivity to spurious noise points, performance in enhancing edges and ease of implementation in hardware. The Heuckel operator, for example, is reliable, but complex to realize in hardware, while the Roberts cross is simple to implement, but is sensitive to noise points,

and does not emphasize edges as well as the other edge operators. As a compromise between performance and ease of implementation, the high-pass filter was chosen for further investigation.

The result of applying the high-pass filter to the image in Fig.2.3.1 is shown in Fig.2.3.2.3. The pixels in this image have been normalized to 8 bits.

### 2.3.3 DYNAMIC THRESHOLDING

The next step in extracting intensity edge features from the image is to separate edge points from the rest of the image. This can be done by thresholding the gradient-intensity image to produce a binary-valued image indicating the location of the edge points.

It is desirable to use a dynamic thresholding process because the illumination level in different parts of the scene (and hence the contrast in the image) will vary. A constant threshold value for the entire image would produce a heavy concentration of edges and noise in regions of high contrast and miss edges in regions of low contrast. Dynamic thresholding, where the threshold value depends on

local contrast values will produce edge points more reliably.

Dynamic thresholding can be accomplished in effect, by using a lateral inhibition filter (such as shown for one dimension in Fig.2.3.3.1(a)) to emphasize peaks in the gradient image, and then thresholding the resulting image. If a pixel value in the gradient image exceeds the surrounding average gradient value by a significant amount, then an edge point will be produced by this operation. The Fourier Transform of the 1-dimensional lateral inhibition digital filter is shown in Fig.2.3.3.1(b).

The weights for a 2-dimensional digital filter that approximate a 2-dimensional lateral inhibition function are shown in Fig.2.3.3.2. The dynamic thresholding algorithm may be written as

```
IF [LATERAL.INHIBITION.VAL(I,J) > THRESHOLD]
  THEN "EDGE.POINT(I,J):= 1" ELSE "EDGE.POINT(I,J):= 0"
```

THRESHOLD is chosen as some small value ($\approx 4$) that separates the spatial noise that have not been completely filtered out in previous operations and weak edges in the image.

The composite frequency response of the smoothing,

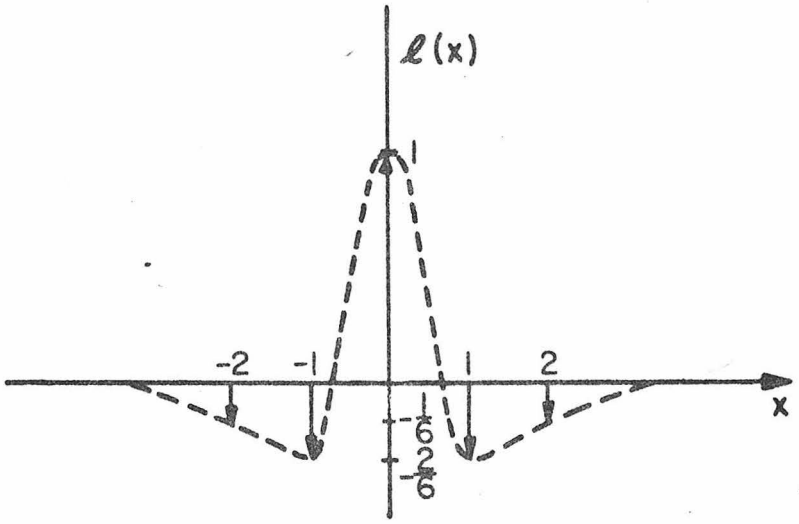



Fig.2.3.3.1(a) The lateral inhibition function (1-D) used in dynamic thresholding to emphasize peaks in the gradient image, $\ell(x) = \delta(x) - \frac{1}{3}[\delta(x-1) + \delta(x+1)] - \frac{1}{6}[\delta(x-2) + \delta(x+2)]$.

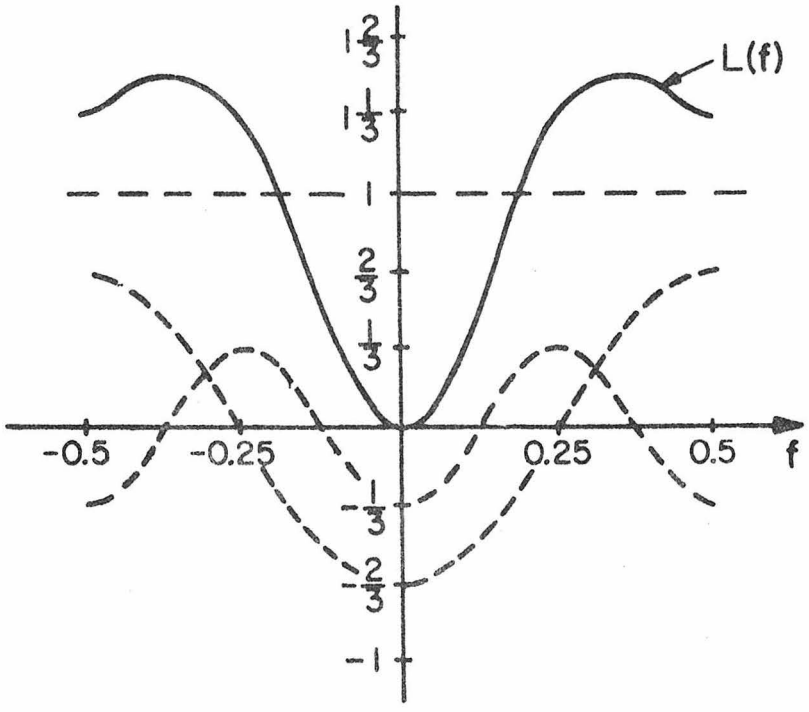



Fig.2.3.3.1(b) The Fourier Transform of the lateral inhibition function, $L(f) = 1 - \frac{2}{3}\cos(2\pi f) - \frac{1}{3}\cos(4\pi f)$.

| | | | | |
|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 |
| -1 | -2 | -2 | -2 | -1 |
| -1 | -2 | +32 | -2 | -1 |
| -1 | -2 | -2 | -2 | -1 |
| -1 | -1 | -1 | -1 | -1 |

Fig.2.3.3.2 The weights for a 2-dimensional digital filter that approximate a 2-dimensional lateral inhibition function.

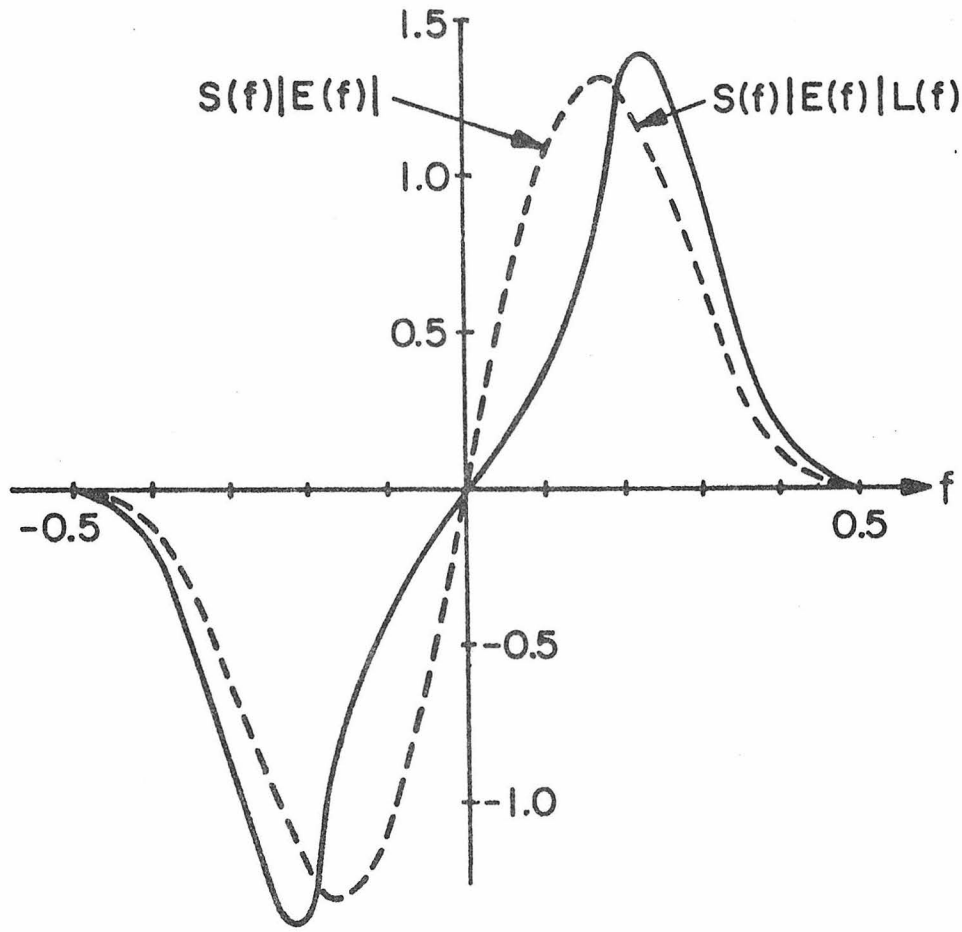



Fig.2.3.3.3 The composite frequency response of the smoothing, edge-enhancement and lateral inhibition filters in 1-dimension.

edge-enhancement and lateral inhibition filters is shown in Fig.2.3.3.3. As is noted in the figure, the effect of the lateral inhibition filter is to narrow the band-pass characteristics, and shift the maximum response frequency upward slightly. Fig.2.3.3.4(b) shows the result of performing dynamic thresholding on Fig.2.3.2.3.

If log digitizers are available, an alternative to using dynamic thresholding to compensate for variations in incident illumination in the scene is described in Appendix A.

Up to this point it may be argued that the three preceding image operators may be combined into a single, larger operator. What are the trade-offs involved? In the first place, combining the three operators together is not as straightforward as it first appears, because the magnitude operation in the edge-enhancement process is a non-linear operation. Hence it would not be possible to combine the three operators in a linear manner to form a single operator. If a linear 3x3 edge-enhancement operator had been used, however, the three preceding operators could be linearly combined by convolving the 3x3 mask for smoothing, the 3x3 mask for edge-enhancement, and the 5x5 mask for dynamic thresholding to produce a 9x9 mask. It

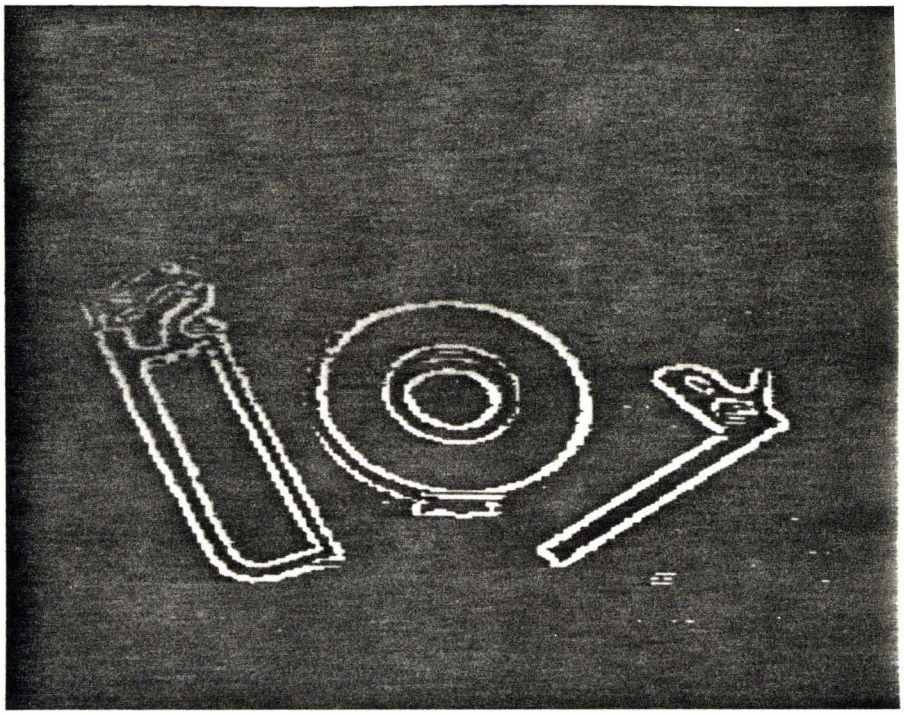


Fig.2.3.3.4(a) Result of thresholding Fig.2.3.2.3, without smoothing of original image

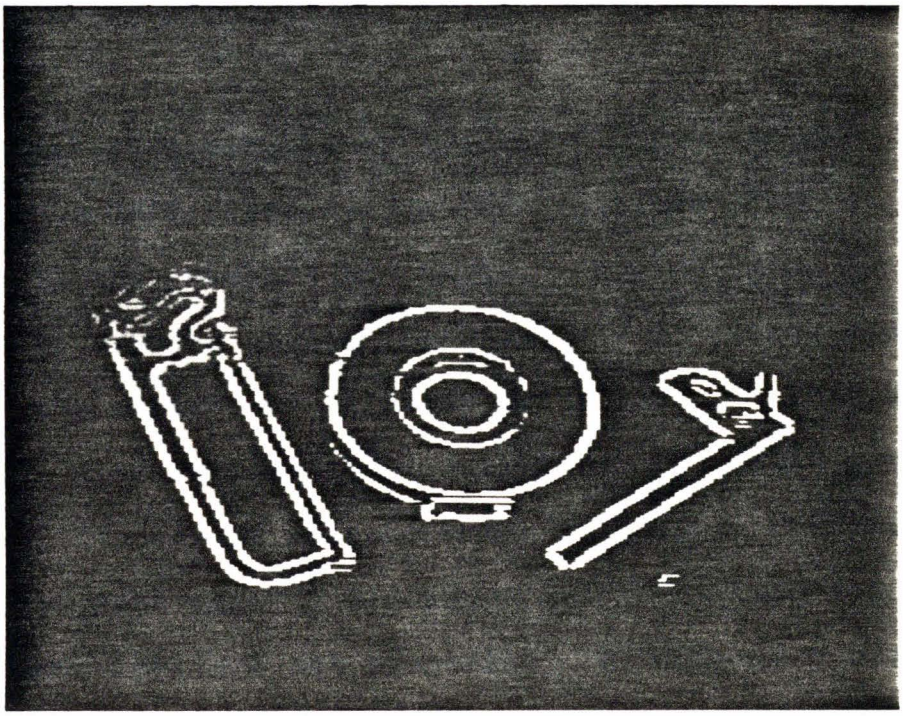


Fig.2.3.3.4(b) Result of thresholding Fig.2.3.2.3, with smoothing of original image

turns out that the pixel storage required for obtaining a 9x9 window of pixel values for computation (as described in Sec.3.1) is about the same as when the three image operations are performed separately in sequence, so there is no advantage to combining the three operations as far as pixel storage is concerned.

## 2.3.4 THINNING

The edge clusters produced by the thresholding operation vary from 1 to 3 pixels in width. Thinning reduces these edge clusters to single pixel-wide contours in preparation for the chain-coding process later on.

In order to describe the algorithm for thinning, it is necessary to define the binary pattern types for a 3 x 3 window found in the thresholded image. Fig.2.3.4.3 shows the 6 binary pattern types for a 3 x 3 neighborhood. Referring to the figure, an INTERIOR point is a '1' that has '1's in its 4 nearest-neighbor positions. A BORDER point has 2 or more '1's in its neighborhood that are connected (see [fn.1] on next page). A CONTOUR point has two '1's in its neighborhood that are not connected. A NODE point has 3 or more '1's in its neighborhood that are not connected. An

END point has a single '1' in its neighborhood and an ISOLATED point has no '1's. The objective of the thinning operation is to remove all the border and isolated points and leave only contour, node, and end points in the thinned image.

A basic algorithm for thinning might be:

```
For every edge point in the image:
    BEGIN
      IF "edge point is a border or isolated point"
        THEN "eliminate edge point"
    END
```

[fn.1] In a square lattice, the 8 neighbors in the 3 x 3 neighborhood of a point form two groups (i) the 4 nearest-neighbors in the north, south, east, and west positions, and (ii) the 8 nearest-neighbors comprising all the members in the 3 x 3 neighborhood (see Fig.2.3.4.1). Two points are considered to be 4-connected when they are in each other's 4-nearest neighbor positions. Similarly, two points are 8-connected when they are in each other's 8 nearest-neighbor positions. In this work, 8-connectedness is used for regions of '1's whereas 4-connectedness is used for regions of '0's. This definition surmounts the connectivity dilemma presented by the pattern in Fig.2.3.4.2. Here if the 8-connectedness definition is used for both black and white squares, the 4 black squares are connected to each other (forming a ring), but the central white square is at the same time connected to the white squares in the background. If 4-connectedness is used for both colored squares, however, the 4 black squares are not connected to each other, but the central white square is not connected to the white background either. Hence using either 8-connectedness or 4-connectedness for both colored squares presents a connectivity dilemma in chain-coding, when a decision has to be made whether a region is bounded by a chain or whether it is part of the background.




Fig.2.3.4.1 (a) The 4 nearest neighbors of a point.
(b) The 8 nearest neighbors of a point.

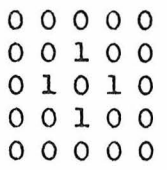



Fig.2.3.4.2 The connectivity dilemma posed by a '0' surrounded by 4 '1's (see [fn.1]).

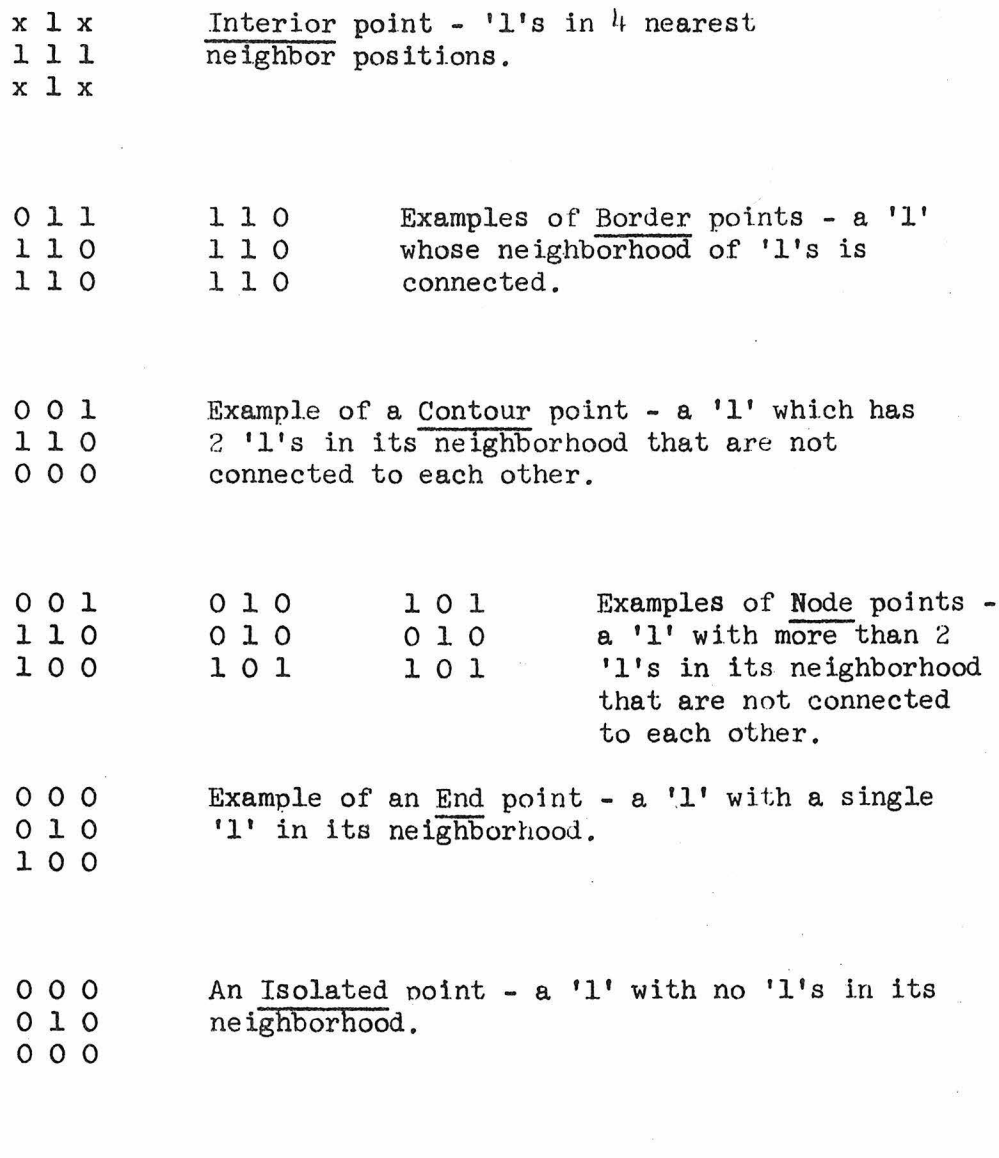



Fig. 2.3.4.3 The binary pattern types for a 3x3 window.

This algorithm is applied in raster fashion to the thresholded image.

Some additional constraints are necessary on the basic algorithm for thinning. Fig.2.3.4.4 shows 3 sets of 2 pixel-wide edge clusters in different orientations. If the basic algorithm is applied to these edge clusters in raster fashion, each edge point will be found as a border point and thus eliminated, leaving no edge points at all. This problem may be avoided if results of previous thinning decisions are made available to later thinning decisions. The algorithm would then be working on a dynamically changing image, where border points may turn into contour points. Then, if one side of the edge clusters in Fig.2.3.4.4 is eliminated, the other side will not.

A second constraint is necessary to prevent a vertically oriented edge cluster from being successively eliminated if the basic algorithm is applied in a raster scan of the image (Fig.2.3.4.5). The constraint is to treat border points with '1's in the bottom row only as end points, and not eliminate them.

Fig.2.3.4.6 shows the result of thinning Fig.2.3.3.4(b). There are approximately 1300 edge points in

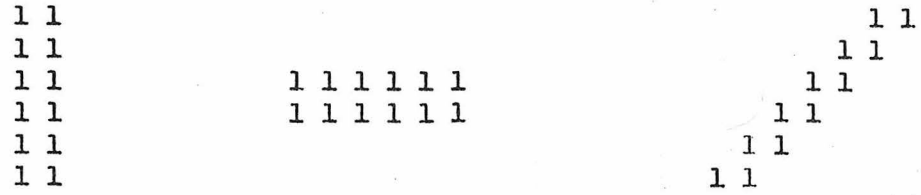


Fig.2.3.4.4 These 2-pixel wide edge clusters would be completely eliminated if the basic thinning algorithm were applied to them without additional constraints.

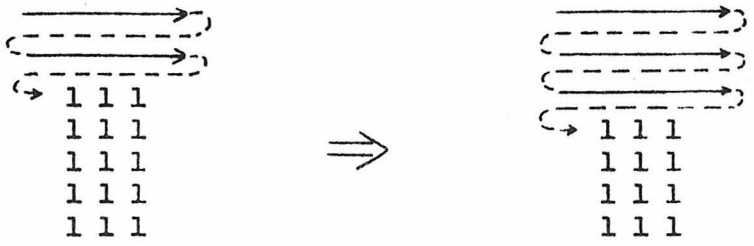


Fig.2.3.4.5 A vertically oriented edge cluster would be completely eliminated in successive passes of the basic thinning algorithm without additional constraints.

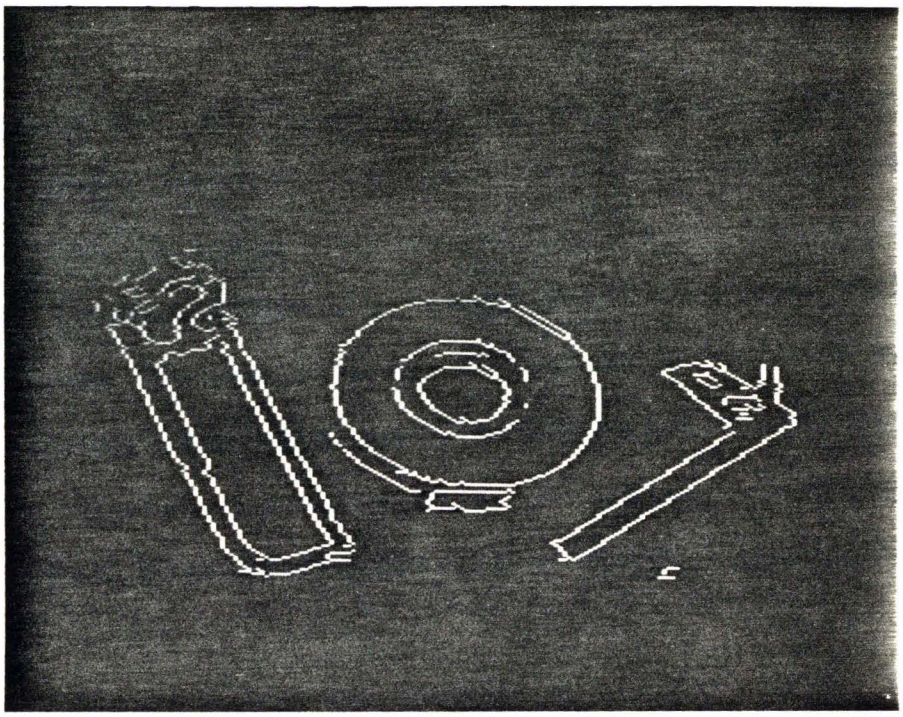


Fig.2.3.4.6 Result of thinning Fig.2.3.3.4(b)

the thinned image.

### 2.3.5 IDENTIFYING NODE AND END POINTS

To prepare for the next stage of processing. chain coding, it is useful to identify node and end points in the thinned image. Chain coding is the process that collects together edge points that are connected. Because node points may indicate the presence of occlusions or corners of objects, one would like edge chain codes to start and stop at nodes or end points. By doing this, a logical ordering in the formation of edge chain codes is produced.

To identify node and end points, a raster scan is made of the thinned image. A 3 x 3 window is used in this scan to determine if the edge point is a node or end point as defined in Sec.2.3.4. If it is a node or end point, its location in the thinned image is stored in a list for use in the chain-coding process.

165 nodes and end points were found for the thinned image in Fig.2.3.4.6.

## 2.3.6 CHAIN CODING

Chain coding is the process whereby edge points that are connected are linked together. The edge points remaining after the thinning process are isolated entities, and by themselves, give no information as to the size or shape of objects. In order to obtain a representation of the edges in the thinned image, the edge points that are connected are linked together to form chain codes.

The possible orientations and the assigned codes for chain elements are shown in Fig.2.3.6.1. Using this coding scheme, the "edge" shown in Fig.2.3.6.2 may be represented by the linked-list (777,666,770,000,101).

To describe the chain-coding algorithm, let us suppose we are collecting the edge points for the edge shown in Fig.2.3.6.2. We have generated the chain elements up to edge point j, and we wish to find the next chain element member. To do so, we examine the 8 neighbors of j, the current edge point, in counter-clockwise fashion, starting with the previous edge point collected, i. The first '1' found in this manner (the first '1' that is not i) becomes the next member of the chain (in this case, k). The process then repeats; the 8 neighbors of k are examined to find the




Fig.2.3.6.1 Assigned codes for the 8 orientations of chain elements.

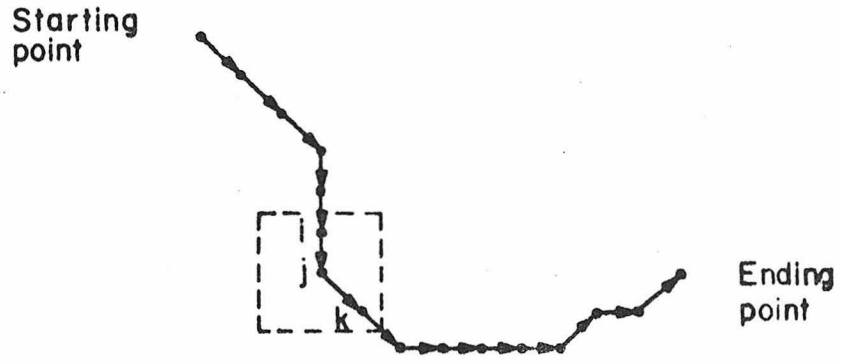



Fig.2.3.6.2 Example of chain code - 777,666,770,000,101.

next element in the chain.

As the edges in the thinned image are being chain coded, edge points that have been collected are converted to '0's, so that no edge point is collected twice. The exceptions are node points, since 3 or 4 branches may grow from such points.

The starting locations of the chain-coding process are provided by the list of node and end points generated by the previous process. The chain-coding process starts with fetching the location of a node or end point from the list. The chain elements for a branch of the starting point are then formed. When another node or end point is encountered in collecting the edge points, the process stops and returns to the starting point. If the starting point is a node, it will have 3 or 4 branches; if it is an end point, it will have only 1 branch. In the case of a node, after 1 branch has been chain coded, the process returns to the same node and collects edge points for another branch. This procedure is repeated until all the branches emanating from that node have been chain coded. The process then moves on to the next entry in the node and end point list and proceeds to chain code the branches for that entry. The chain-coding process for an image continues until all the entries in the

list have been considered.

In rare instances, edges in the form of perfect rings, without node or end points, will occur in the thinned image. These rings will not be chain coded in the normal process. These edge points can be collected, however, if a scan is made of the thinned image after all other edge points have been collected and converted to '0's.

53 separate chains and 1282 chain elements were found for the thinned image in Fig.2.3.4.6. Each chain requires a starting address of 16 bits, and each chain element is represented by 3 bits. Hence it takes about 4,694 bits (587 8-bit bytes) to store the edge chain codes, as compared with 244x188x8 bits (45.6K bytes) for the original image. The data reduction up to this stage is by a factor of about 78.

In Secs.3.3 and 3.4 I will compare the use of a pipeline structure with a randomly accessed structure for chain coding. Because it is desirable to be able to follow along an edge while collecting edge points, the random access structure appears to be more suitable for chain coding. The interested reader is referred to [FRE.74] for a more detailed discussion of chain coding.

## 2.3.7 GENERATING LINE DATA STRUCTURES

This process fits straight line segments to the edge chain codes found in the previous operation. The objective is to find global representations for the intensity edges and to compress the chain-code data as much as possible before passing them on to a higher-level vision processor. Only straight lines are used in fitting to the chain codes because they are reasonably effective at representing the edges and the technique is easy to implement in hardware. Other researchers have tried fitting splines (also circular and elliptic arcs) to edge clusters ([SHI.75] and [PER.78]), but these techniques are more complex to implement in hardware.

The basic algorithm for this process examines the orientation of chain elements one at a time, and determines if it differs significantly (to be defined later) from the orientation of the first chain element in the current chain segment being considered (ie., the reference orientation). If the orientation difference is determined to be significant, a breakpoint is set and the process begins a new line segment, with the orientation of the first chain element in this new segment as the new reference

Fig.2.3.7.1(a) shows the 8 discrete orientations of chain elements. Because the orientations are quantized, a straight line in a digital image may contain chain elements that exhibit discrete orientation jumps, as shown in Fig.2.3.7.2(a). Hence an orientation difference of 1 unit from the reference orientation is allowed when fitting line segments to chain codes. However an orientation difference of 1 unit from the reference twice in a row is not allowed; a breakpoint would be set in this case (Fig.2.3.7.3(a)).

In allowing for discrete changes of chain element orientations in a straight line, it is possible that an error will arise when portions of a line change directions slowly. The worst error possible occurs when a portion of a line is followed by another whose orientation differs from that of the initial portion by the amount allowed for discrete jumps in a straight line. Fig.2.3.7.4(a) shows the worst error possible when fitting lines to chain elements in 8-orientation quantized space. The line starts off in the '0' orientation and then rises at 26.5 arc-degrees from the horizontal. Because the rise is in an alternate series of '0' and '1' orientations, no breakpoints are set. As a result, a single line segment is formed across the endpoints of the chain code.

| | | |
|---|---|---|
| 3 | 2 | 1 |
| 4 | + | 0 |
| 5 | 6 | 7 |

(a)

| | | | | |
|---|---|---|---|---|
| | | 6 | 5 | 4 |
| 7 | | | | 1 |
| 8 | | + | | 0 |
| 9 | | | | 15 |
| 10 | 11 | 12 | 13 | 14 |

(b)

Fig.2.3.7.1 (a) Chain element orientations in 8-space.
(b) Chain element orientations in 16-space.



(a) 8-orientation space (b) 16-orientation space

Fig.2.3.7.2 A straight line in a digital image may be composed of chain elements which exhibit discrete orientation jumps.

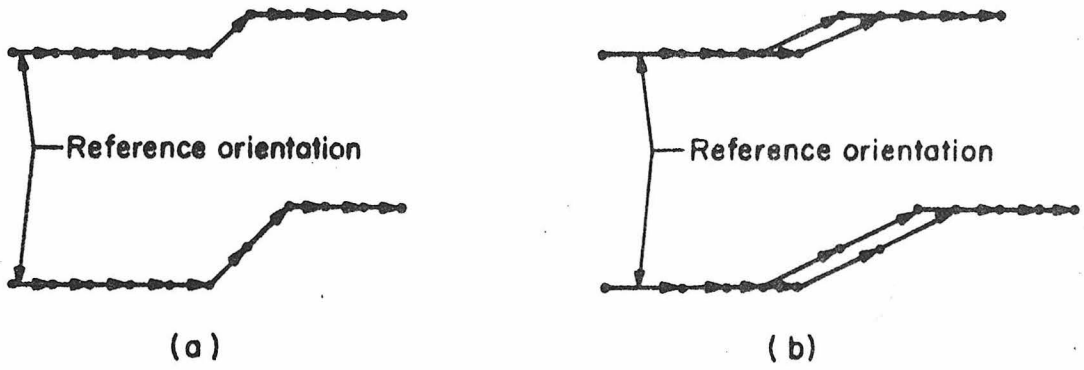
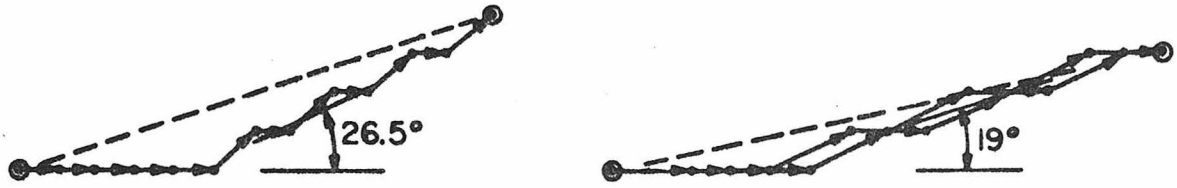



Fig.2.3.7.3 (a) 8-orientation space: An orientation difference of 1 unit from the reference orientation is allowed once, but not twice in a row. (b) 16-orientation space: An orientation difference of 1 unit from the reference orientation twice in a row is allowed, but not 3 times in a row.




(a) 8-orientation space (b) 16-orientation space

Fig.2.3.7.4 Worst error possible in fitting line segments to chain codes. No breakpoints are set, and a single line segment is formed between the end points of the chain code.

In an attempt to minimize the worst error possible, a finer quantized orientation space was investigated. By considering the orientations of chain elements 2 at a time, 16 discrete orientations are possible (Fig.2.3.7.1(b)). The reference orientation is now the orientation of the first 2 chain elements in the current chain segment. Fig.2.3.7.2(b) shows the most-frequent-orientation-change case for a straight line. A 1 unit orientation difference from the reference orientation twice in a row is now allowed, but not 3 times in a row (Fig.2.3.7.3(b)). Fig.2.3.7.4(b) shows the worst error possible for the 16-orientation space case. The chain code starts at '0' orientation and then rises at 19.0 arc-degrees. Because a 1 unit orientation difference is present no more than twice in a row, no breakpoints are set, and as a result, a line segment is formed between the endpoints of the chain code. Fig.2.3.7.5 shows the chain element orientations for a semi-circle plotted in 8- and 16-orientation space. The orientation transitions for the 16-orientation case are smoother and the breakpoints generated are more evenly distributed around the semi-circle.

As the breakpoints are set on the chain codes from examining chain element orientations, the locations of breakpoints in the thinned image are being stored away in

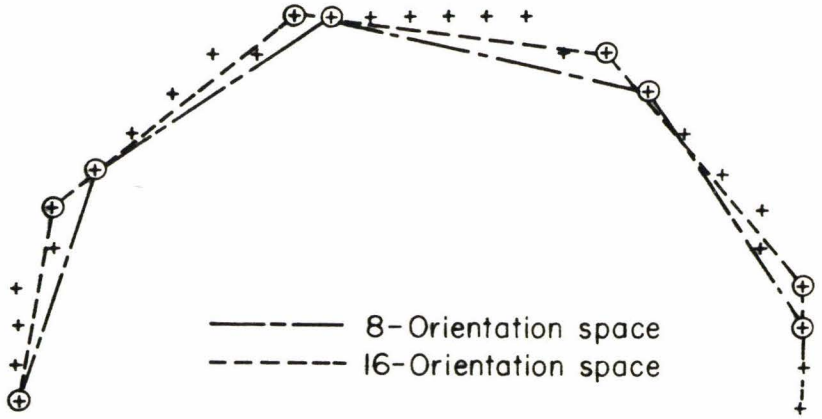



Fig.2.3.7.5 Breakpoints generated for a semi-circle from considering chain element orientations in 8- and 16-orientation space.



Fig.2.3.7.6 Line segments generated from Fig.2.3.4.6

another list. Each entry in this list will contain the starting location of a chain, the breakpoint locations along the chain, and the endpoint location.

Fig.2.3.7.6 shows the line segments generated from the thinned image in Fig.2.3.4.6, using 16-orientation space. The bright points indicate start, end and breakpoints; line segments have been filled in with a lower intensity. 166 line segments (represented by 424 bytes; since there are on the average 3 vectors requiring 4 addresses per chain, and there are 53 chains) result from the 1282 chain elements in Fig.2.3.4.6.

As is evident from comparing the 8- and 16-orientation space cases, the angle for worst error could be reduced by using an even finer quantized orientation space. A finer orientation space may be obtained by examining the orientation of more chain elements at a time (i.e., using larger templates). The price to be paid is increased computation and more heuristics for allowing discrete jumps in straight lines.

## 2.4 EXTRACTING EDGE FEATURES FROM OTHER SCENES

The sequence of operations for extracting edge features was applied to 3 other scenes - the JPL robot's ARM scene (Fig.2.4.1(a)), a PERSON scene (Fig.2.4.2(a)) and a CAR scene (Fig.2.4.3(a)).

Figs.2.4.1(a)-(f) show the results of each of the image processing operations on the scene of the robot's ARM holding a wrench. The edge-enhanced image is shown in Fig.2.4.1(b). Fig.2.4.1(c) shows the thresholded image without smoothing of the original scene. The presence of edge points caused by the shadow of the arm and other noise points can be seen in the background. In Fig.2.4.1(d), the thresholded image with smoothing, most of the noise points in the background have been eliminated. The resulting thinned image (Fig.2.4.1(e)) has 377 nodes, 104 chains and 1251 chain elements. After reducing this image to line segments, the number of vectors resulting is 257; almost a 5:1 data reduction from the 1251 chain elements, and a 178:1 data reduction from the original image.

Fig.2.4.2(a) shows a PERSON scene and Fig.2.4.2(b) the edge-enhanced image of it. Figs.2.4.2(c) and (d) show the thresholded images without and with smoothing. Many of

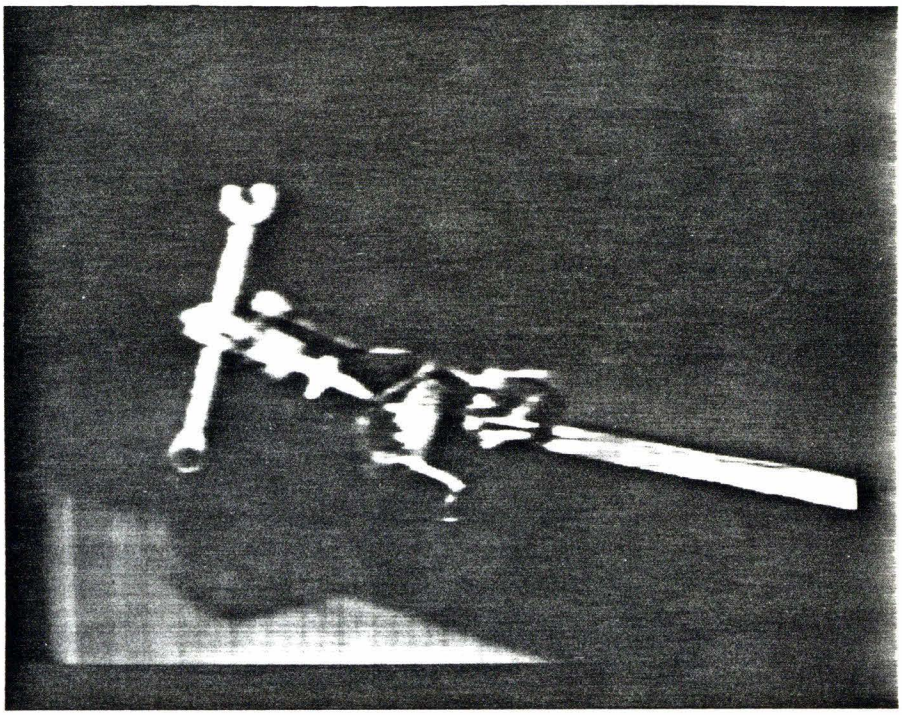


Fig. 2.4.1(a) The robot ARM scene

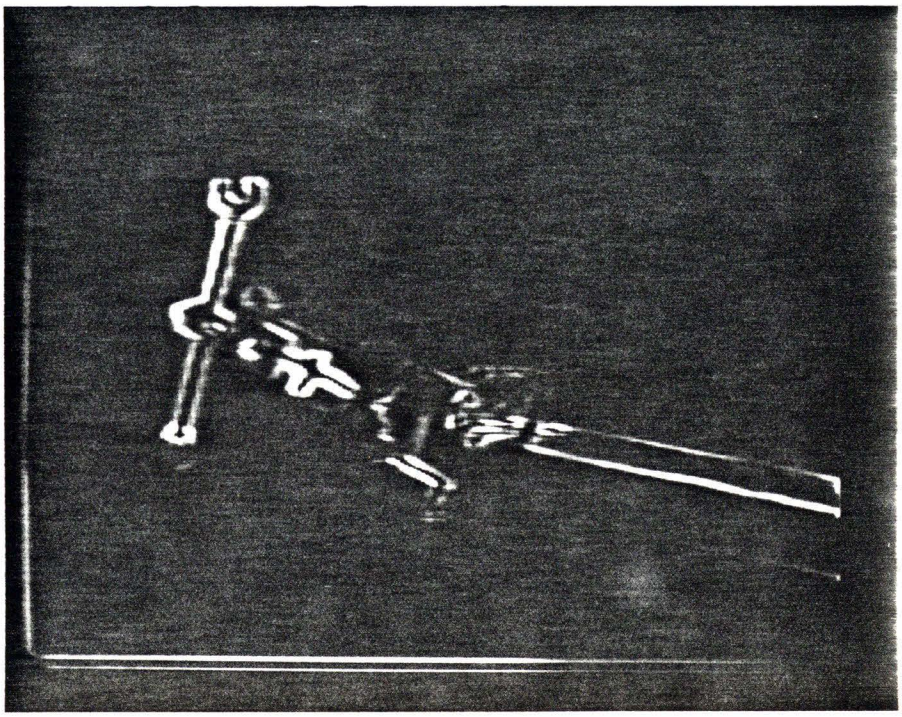


Fig.2.4.1(b) Edge-enhanced image of ARM scene

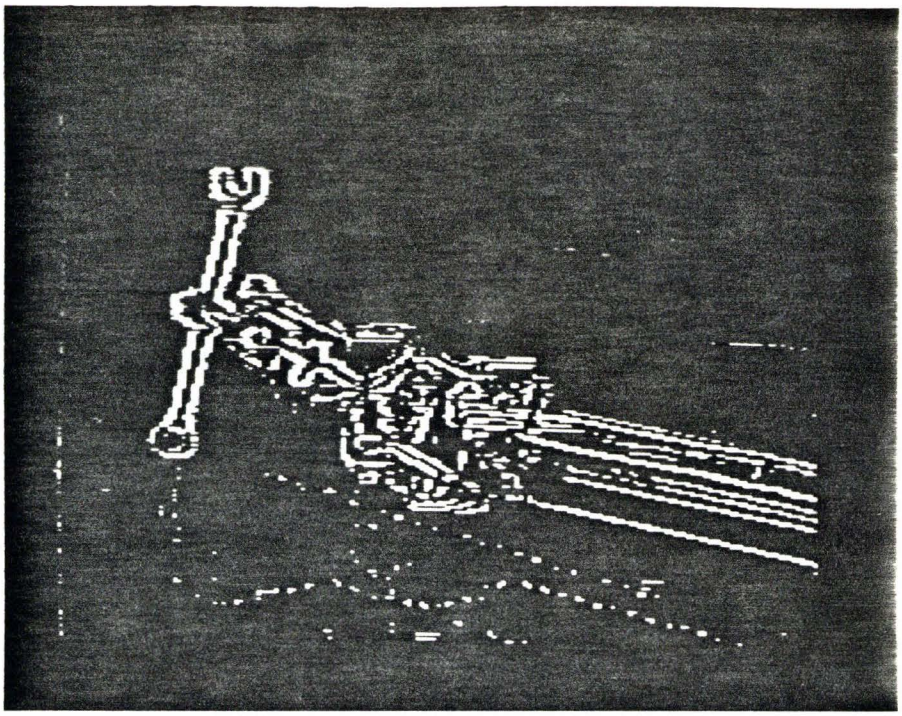


Fig.2.4.1(c) Thresholded image of ARM scene, without smoothing of original image

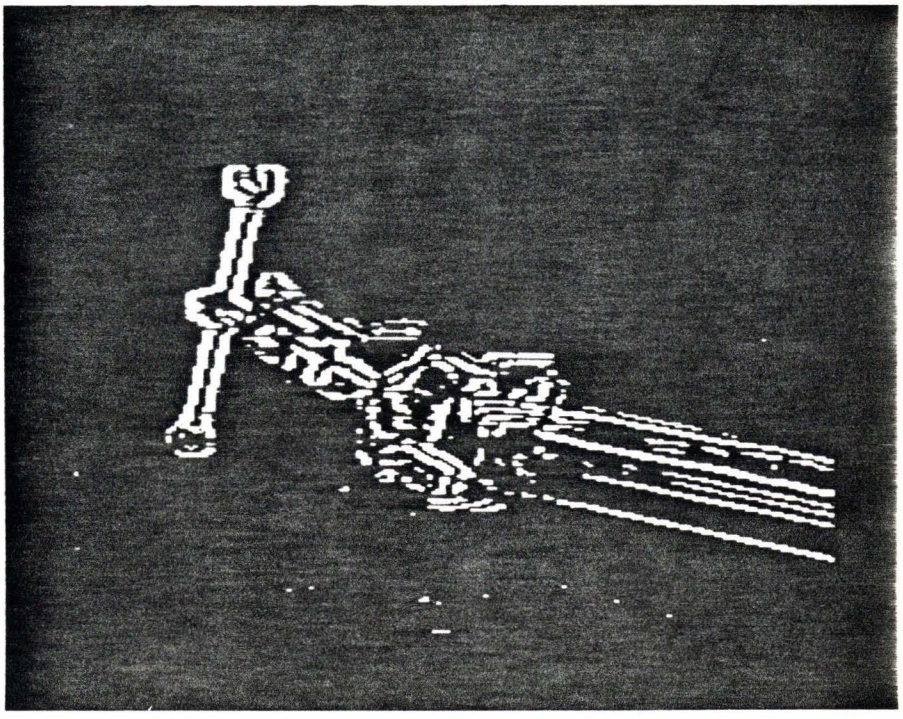


Fig.2.4.1(d) Thresholded image of ARM scene, with smoothing of original image

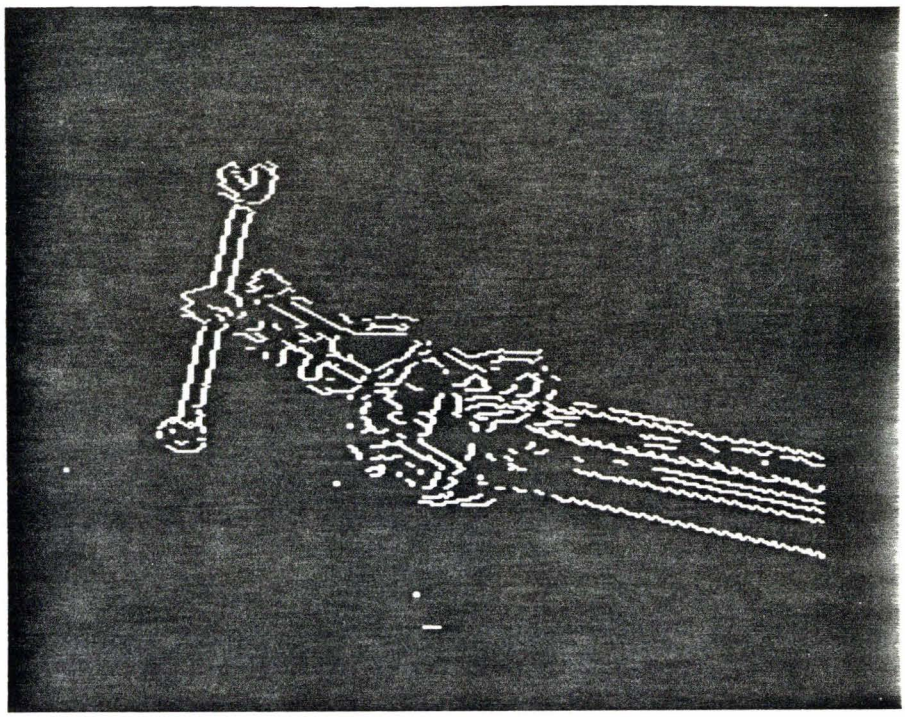


Fig.2.4.1(e) Thinned image of ARM scene

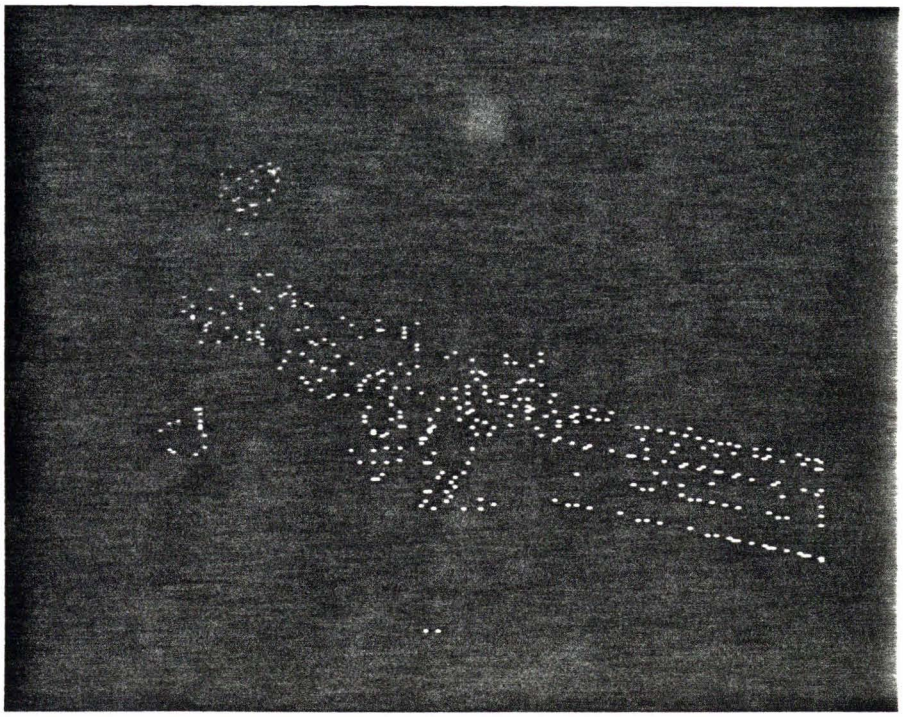


Fig.2.4.1(f) Reconstructed image of ARM scene

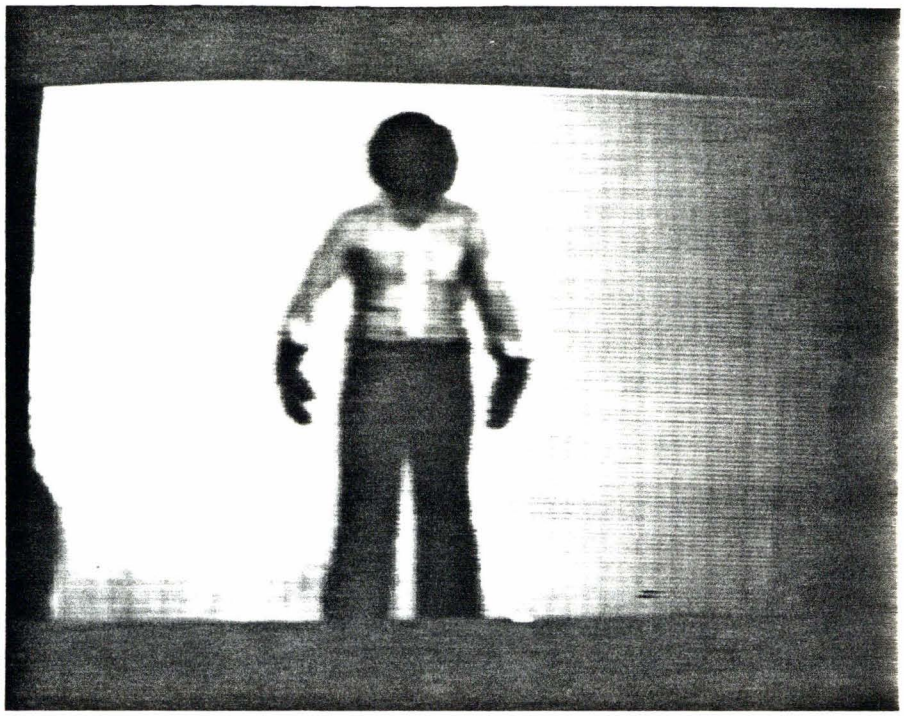


Fig.2.4.2(a) PERSON scene

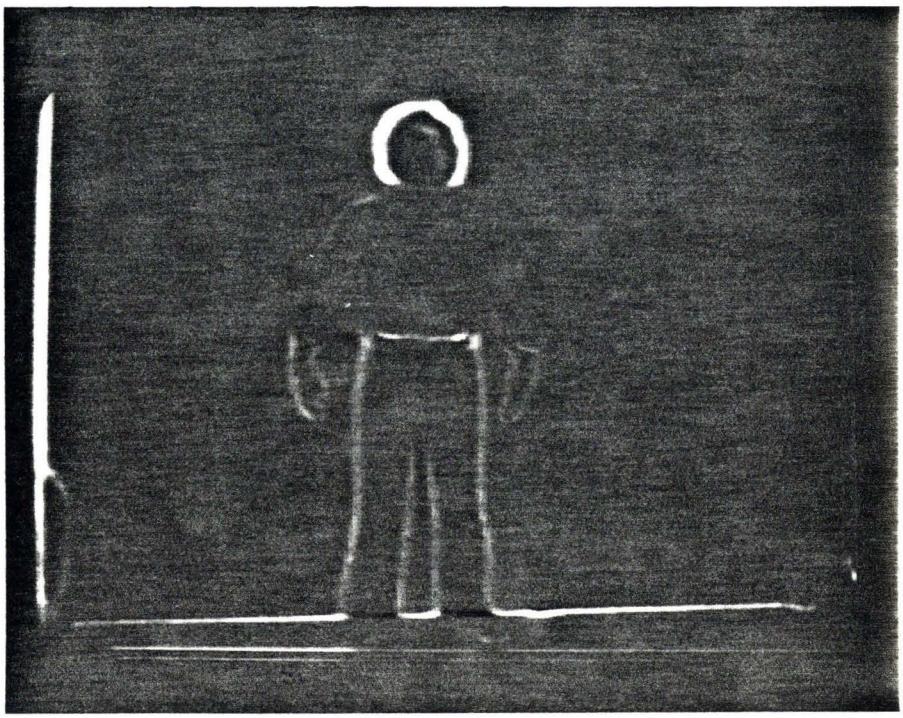


Fig.2.4.2(b) Edge-enhanced image of PERSON scene

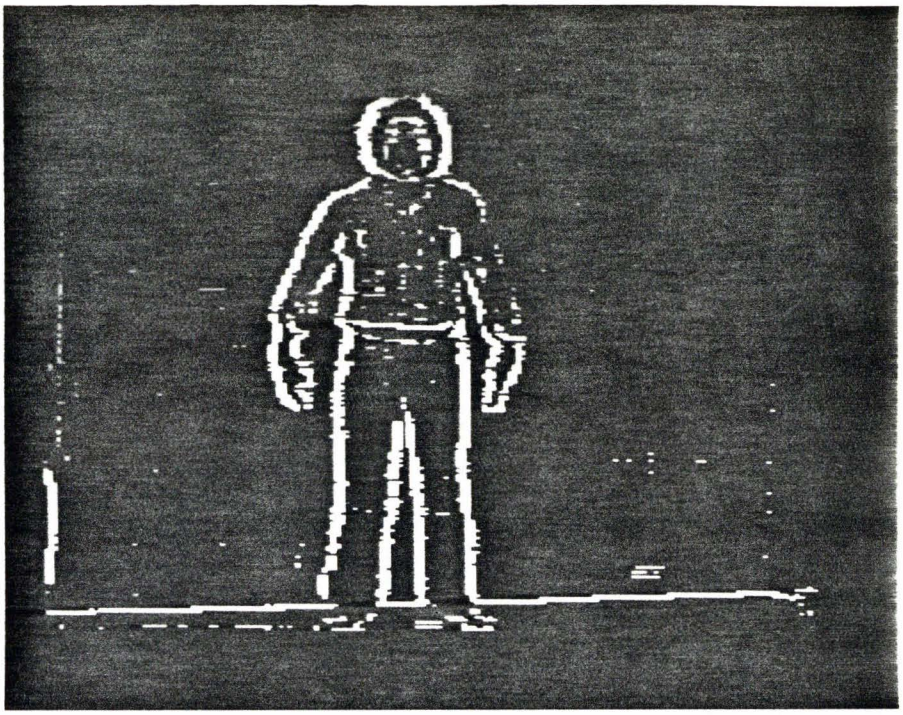


Fig.2.4.2(c) Thresholded image of PERSON scene, without smoothing of original image

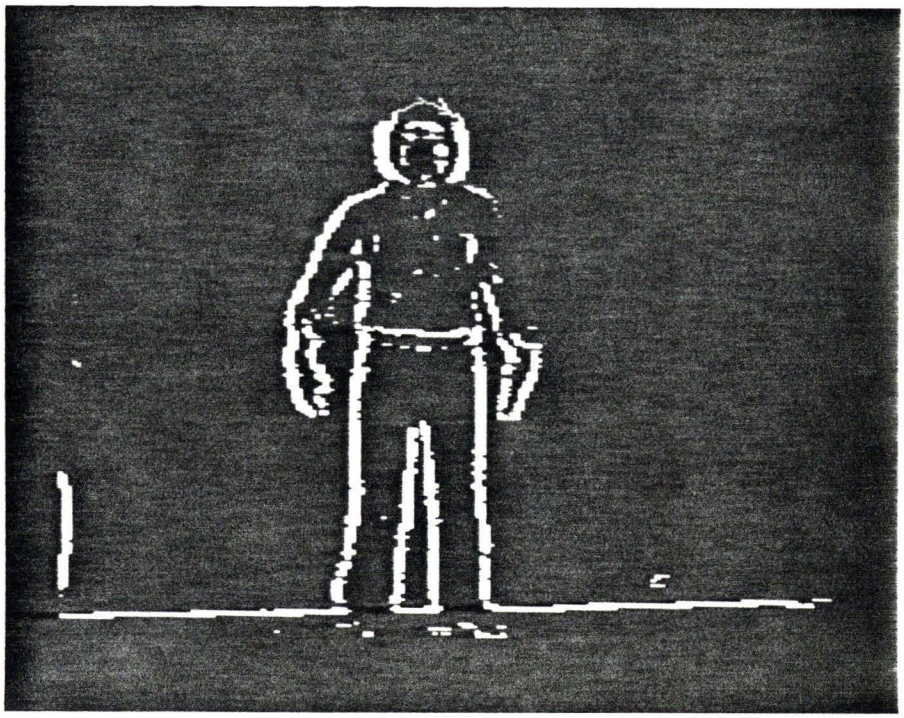


Fig.2.4.2(d) Thresholded image of PERSON scene, with smoothing of original image

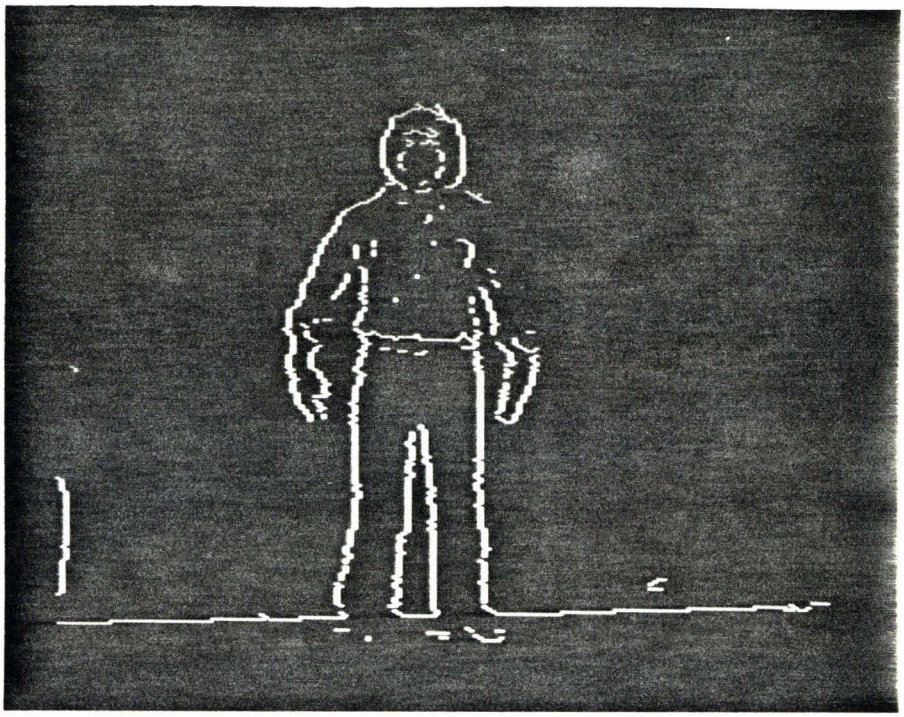


Fig.2.4.2(e) Thinned image of PERSON scene



Fig.2.4.2(f) Reconstructed image of PERSON scene



Fig.2.4.3(a) CAR scene



Fig.2.4.3(b) Edge-enhanced image of CAR scene

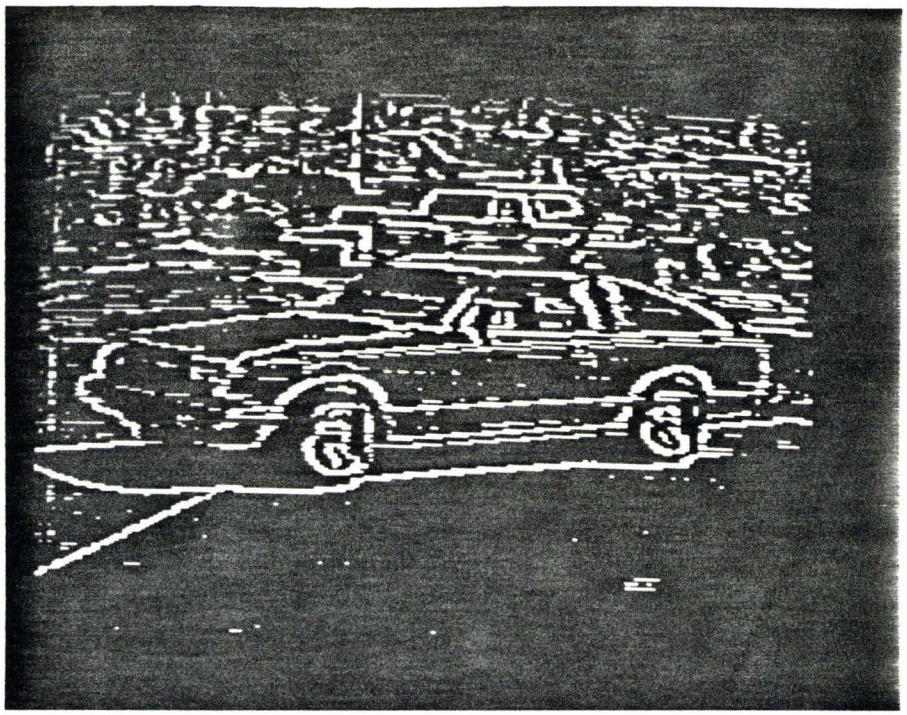


Fig.2.4.3(c) Thresholded image of CAR scene, without smoothing of original image

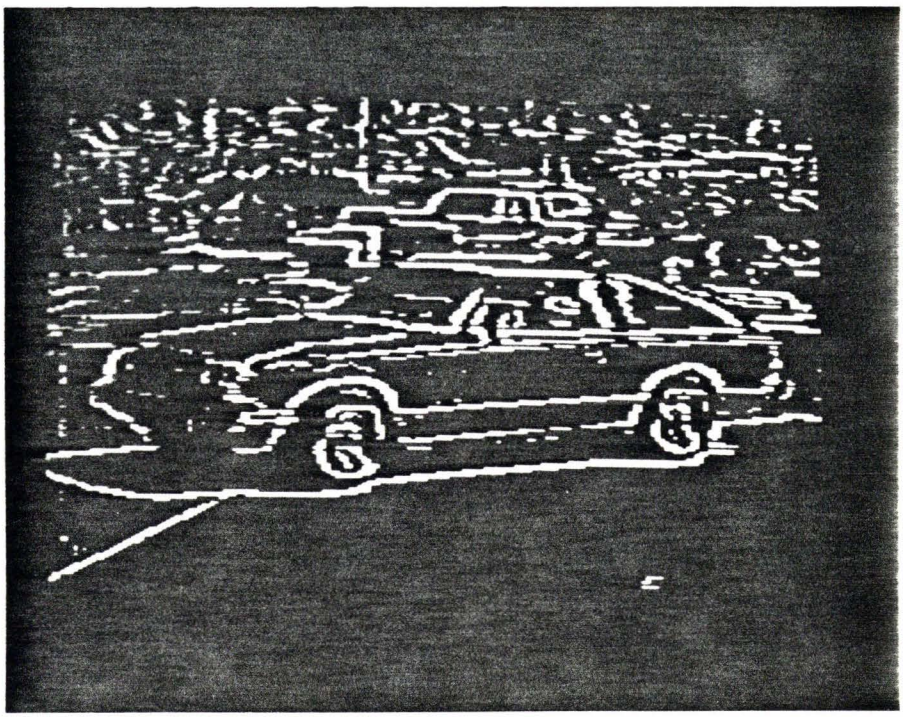


Fig.2.4.3(d) Thresholded image of CAR scene, with smoothing of original image

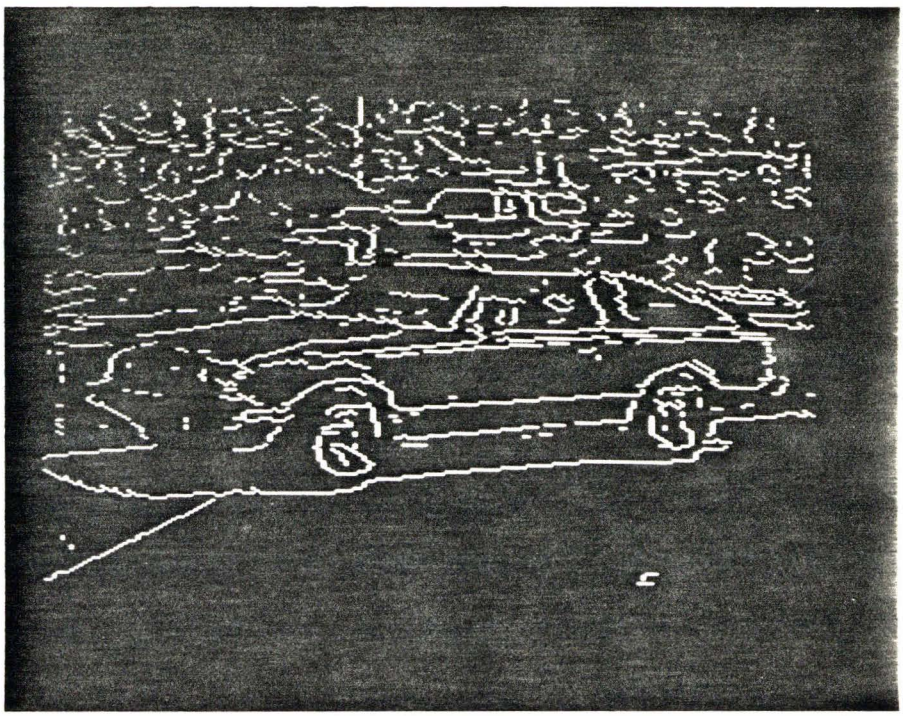


Fig.2.4.3(e) Thinned image of CAR scene

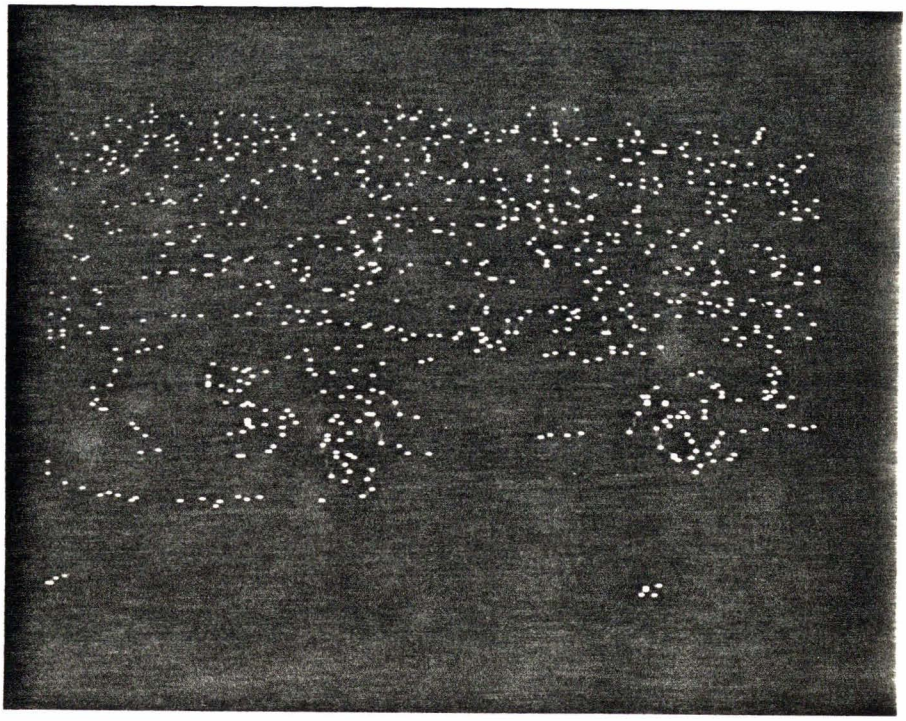


Fig.2.4.3(f) Reconstructed image of CAR scene

the noise points have been eliminated in the thresholded image with smoothing. Part of the outline of the left arm is not detected, because of poor contrast in that area in the original scene. The resulting thinned image has 166 nodes, 49 chains and 1069 chain elements. The reconstructed image, with 106 line segments, represents a 10-fold data reduction from the 1069 chain elements, and a 430-fold data reduction from the original image.

The sequence of operations for extracting edge features was tried on a more complex outdoor scene. Fig.2.4.3(a) shows a CAR scene, with considerable texture (bushes, trees and rocks) in the background. The edge-enhanced image is shown in Fig.2.4.3(b). Again, there are fewer noise points in the thresholded image with smoothing of the original scene (Fig.2.4.3(d)) than without (Fig.2.4.3(c)). The thinned image (Fig.2.4.3(e)) has 1001 nodes, 282 chains and 3011 chain elements. The number of vectors resulting in the reconstructed image (Fig.2.4.3(f)) is 538, almost a 6-fold reduction from 3011 chain elements, and an 85-fold data reduction from the original image.

## 2.5 SUMMARY

In this chapter I have presented a sequence of operations for extracting edge features from digital images for real-time control purposes. The data reduction in proceeding through the sequence is considerable - from a 188 x 244 pixel image to the reconstructed edge image of a few hundred vectors. The objective, of course, is to represent the relevant edge features for object tracking by as few data elements as possible so that they can be processed in real-time by a higher-level vision processor.

Another objective of this work is to show that the sequence of operations for edge-extraction can be performed in real-time. In the next chapter I will examine the computational structures for performing the edge-extraction operations in real-time and present the design of a pipeline unit for the edge-enhancement operation.

# CHAPTER 3
# COMPUTATIONAL STRUCTURES FOR PERFORMING THE EDGE-EXTRACTION OPERATIONS IN REAL-TIME

In this chapter I will examine the computational structures for performing the edge-extraction operations in real-time. It turns out that the majority of the edge-extraction operations presented in Chapter 2 have similar computational characteristics, and may be performed by pipeline processors. The exception is the chain-coding process. Computational structures for performing the chain-coding process rapidly is discussed in Secs.3.3 and 3.4.

In Sec.3.1 I will discuss the use of pipeline processors for executing in real-time, the algorithms for smoothing, edge enhancement, thresholding, thinning, identifying node and end points and generating line data structures. To demonstrate methodology and feasibility, the design of a pipeline unit in n-MOS LSI for the edge-enhancement process is presented in Sec.3.2.

Sec.3.3 investigates the pipelined approach to chain-coding. The edges are tracked in a raster-scan mode,

and as edge points are encountered, they are sorted into "bins" or "stacks". To make maximum use of the stacks, a dynamic allocation scheme for assigning stacks to edges to be tracked is desirable, and this is a drawback to using pipelining, since all operations are to be performed in real-time.

Sec.3.4 describes a simpler technique for chain-coding, using a random-access structure to collect edge points in the thinned edge map. At worst, an extra frame interval is required when using this structure for chain-coding than when using a pipeline structure.

## 3.1 COMPUTATIONAL CHARACTERISTICS OF THE EDGE-EXTRACTION OPERATIONS THAT MAKE THEM SUITED FOR EXECUTION BY PIPELINE PROCESSORS

Upon examining the computational characteristics of the algorithms presented in Chapter 2 for smoothing, edge enhancement, thresholding, thinning, identifying node and end points, and generating line data structures, one notices 2 things common to these algorithms. Firstly, each operation is local in nature, that is, only data elements in a small surrounding neighborhood is involved in the transformation of any datum. The local nature of the

computation may allow parallel access of the data elements and parallel computation. Secondly, for each of the algorithms, the same computation is performed for each data element. Therefore the transformation of each datum can take place simultaneously and independently of the others, allowing some form of parallelism to be utilized in executing the algorithms. Because of the factors listed in Sec.1.6, however, array processing of the SIMD kind is not considered suitable for processing reasonable-size images (of about 500 x 500 pixels).

Another form of parallel processing exists, however, that circumvents the problems arising from array processing. This is overlap or pipeline processing. Fig.3.1.1 shows the pipeline concept. The pipeline unit accepts data elements serially for computation. Since the pixels from most solid-state imagers (such as the CID imager used for obtaining images in this thesis) are shifted out serially, the pipeline structure is ideally suited for this application.

The pipeline processor is divided into several sections, each of which performs part of the transformation on the incoming data stream, so that collectively they achieve the desired computation. Each of the sections contain clocked latches, which store the data being

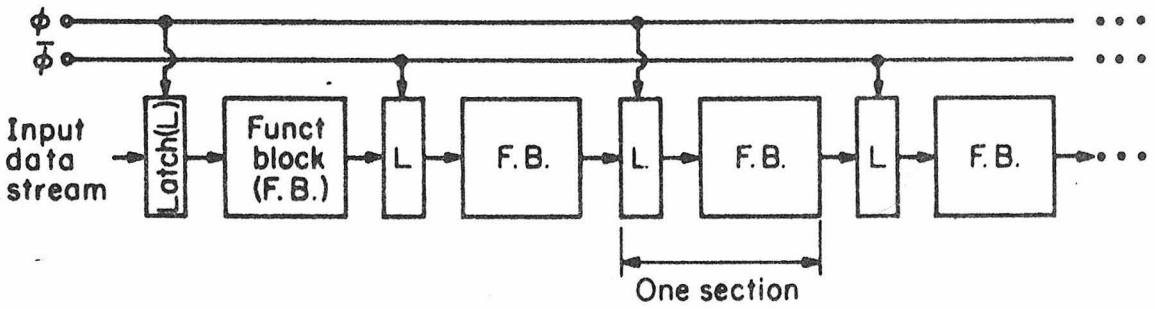



Fig.3.1.1 Configuration of a pipeline processor.

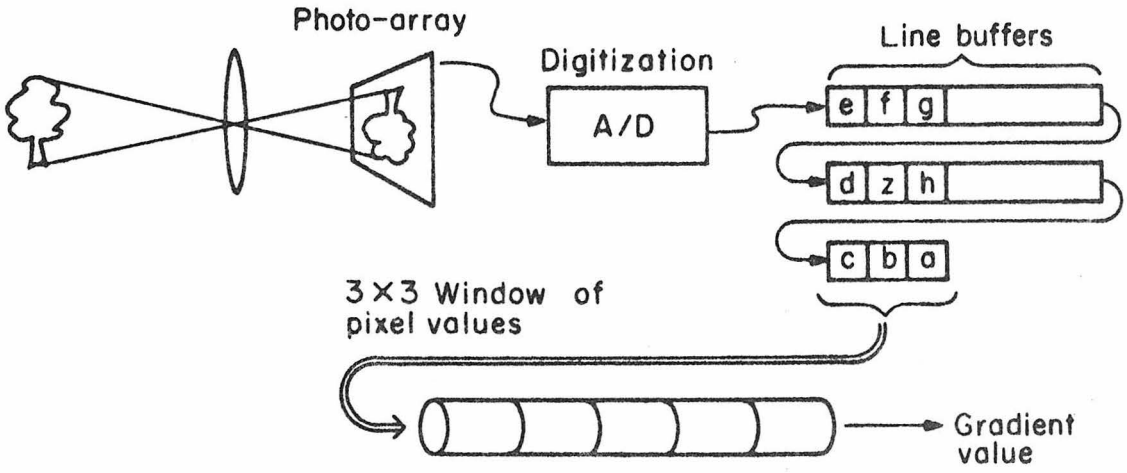



Fig.3.1.2 Front end of imaging system for extracting edge features.

transformed in their respective sections at any given moment. Hence, at any instant, the pipeline contains data at different stages of being transformed in its many sections. The data elements are moved from section to section of the pipeline by application of clock pulses to the latches. The latches are alternately controlled by opposite phases of the pipeline clock. Hence in 1 clock cycle a datum will traverse 2 sections of the pipeline. The transformations on the data in every section of the pipeline must be complete at the end of each clock pulse. The computational efficiency of a pipelined unit can be very high because there is little overhead arising from input/output transfers, and the same structure is utilized in the transformation of all the data elements.

Consider the speed of operation of a pipeline processor. Suppose it has m sections and the cycle period of the pipeline clock is t seconds. The latency of the pipeline, defined as the time taken to fill the pipeline with data, is mt/2 secs. Once the pipeline is full, the rate at which processed data emerges from the pipeline, called the thruput rate, is 1/t per sec. Hence if there are n data elements, it would take nt secs to process them, after an initial delay of mt/2 secs to fill the pipeline. If n is large (say, 1 frame of 500x500 pixels) compared with the number of sections in the pipeline (m), then the latency

of the pipeline will be small compared with the frame time (the time taken for a frame of data to emerge from the pipeline).

Before the details of the pipeline unit design are discussed, a description of a simple imaging system is given here (Fig.3.1.2). The scene to be analyzed is imaged onto a photo-array, which (assuming a solid-state device) converts the light energy at each pixel point to a packet of charge representing the light intensity at that point. These charge packets are then sent serially to an analog-to-digital converter which produces 8-bit binary numbers proportional to the size of the charge packets (or perhaps proportional to the log, see Appendix A).

Assuming a serial transfer of pixel values from the photo-array imaging device, the shift rate for a 512 x 512 pixel image at 30 frames/sec would be

SR = (512x512x30) pixels/sec

~ 7.86 x 10**(6) pixels/sec.

This would also be the rate at which a pipeline unit accepts pixels from the photo-array. The cycle period of the pipeline unit would then be

t = 1/(7.86 x 10**(6)) sec = 127 ns.

The frame time would then be 33.3 ms, and the latency of the pipeline would be 63.5m x 10**(-9) secs.

## 3.2 IMPLEMENTING A PIPELINE PROCESSOR FOR THE EDGE-ENHANCEMENT OPERATION USING N-MOS LSI TECHNOLOGY

The algorithm for the edge-enhancement operator described in Sec.2.3.2 is presented here again:

$$EDGE.ENH(z) = |(c+d+e)-(a+h+g)| + |(g+f+e)-(a+b+c)|$$

This algorithm requires a 3x3 neighborhood of pixel values for computation. Parallel access of the pixel values in the 3x3 neighborhood is desirable so that the transformation of each datum can proceed as rapidly as possible. In order to obtain the 3x3 neighborhood of pixel values simultaneously, the digitized pixel values from the A-to-D converters are sent to line buffers, which store enough pixel values to produce the neighborhood required for computation (Fig.3.1.2). As the pixels in the digitized picture are shifted through the line buffers, the 3x3 computational window effectively moves through the digital picture. The pixel values from this 3x3 window are fed in parallel into the edge-enhancement pipeline. Blanking is required at the end of each image line and at the end of each frame, when the 3x3 computational window becomes 'discontinuous'.

Fig.3.2.1 shows the clocking scheme for a pipeline processor constructed using MOS LSI technology. Pass transistors are used to isolate different sections of the pipeline and data are stored in the form of charge packets at the gates of input transistors. The pass transistors isolating different sections of the pipeline are controlled by opposite phases of the clock. The clocking waveform is symmetrical, with the requirement that the skew time, T<s>, between the rise of CLKPOS and the fall of CLKNEG be less than the propagation time, T<p>, through any logic block. This ensures that data in one section of the pipeline will not race through and destroy data in the next section because of excessive skew time.

The edge-enhancement algorithm may be depicted as shown in Fig.3.2.2. The front section consists of four 3-input adders, followed by 2 subtractors and a logic block that forms the absolute value of the previous subtraction. The resulting absolute values are then added to produce the gradient value, R. A comparator has been included in the last section to perform constant thresholding. It compares the gradient value, R, with a constant threshold, T, and indicates at the output if the gradient value is greater or less than the threshold (that is, whether an edge is present).

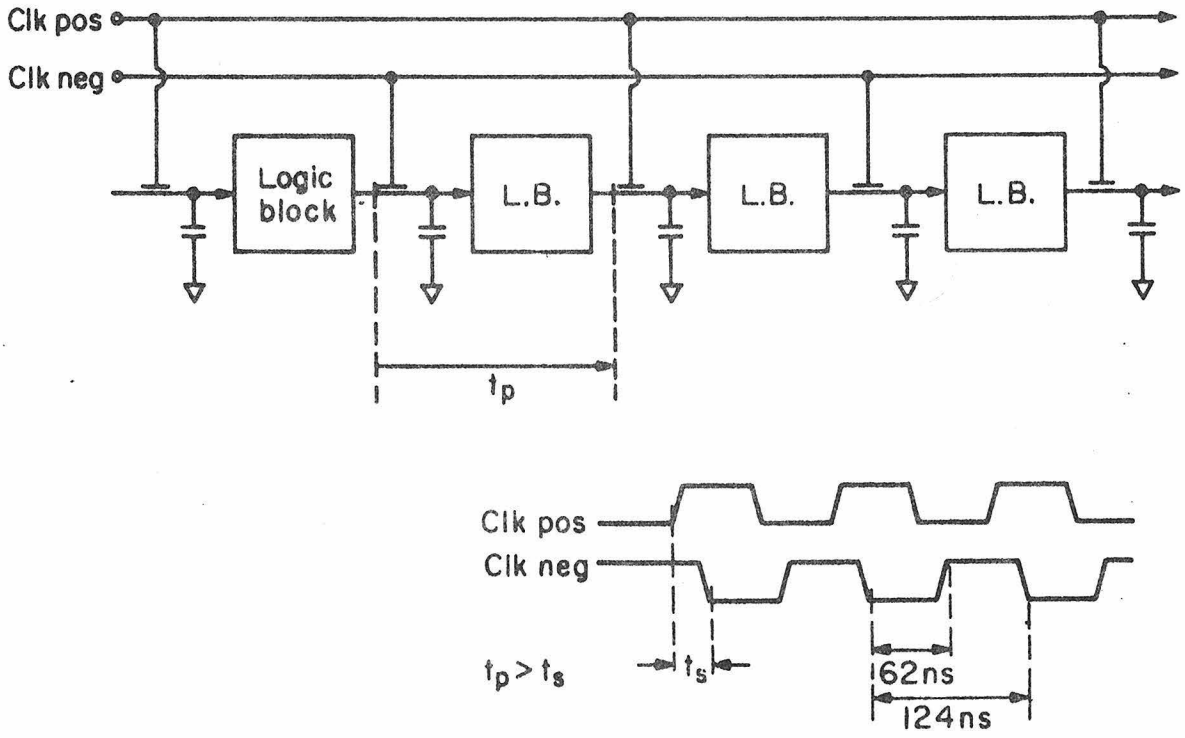



Fig.3.2.1 A MOS pipeline clocking scheme.

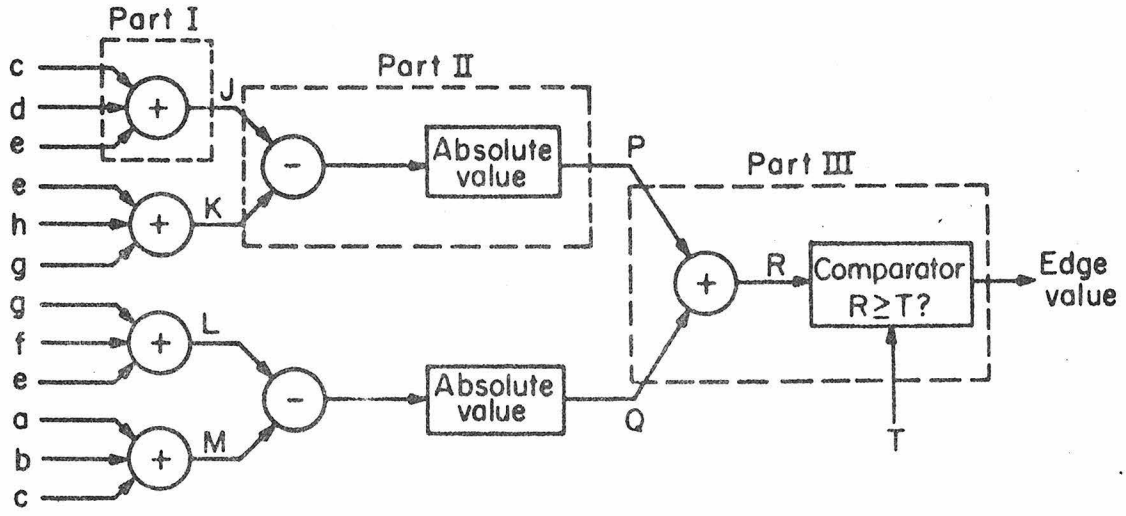



Fig.3.2.2 Configuration of edge-enhancement algorithm.

The actual implementation of the edge-enhancement operator is partitioned into 3 parts (Figs.3.2.3, 3.2.4, 3.2.5), on the same chip. The 3-input adder is in the first part, the difference and absolute value cells in the second part, and an adder and comparator in the third part. The advantages to this partitioning are (a) it makes possible characterization of each part independently of the others (b) the number of pin interconnections per package is reduced and (c) there is a savings in chip area utilized.

The central logic cell in the design of the pipeline unit for edge enhancement is the Manchester full adder (see [SUT.75] for the characteristics of this adder). The unique feature of this adder is that the carry chain is of a transmission-gate type, which results in a faster addition process for a small number of stages than the conventional inverter carry chain. Fig.3.2.6(a) shows the logic table for the Manchester Adder while Fig.3.2.6(b) shows the circuit configuration of the adder implemented in MOS technology.

Implementation of Part I of the edge-enhancement pipeline unit (a 3-input 8-bit adder) is shown in Fig.3.2.3. The inputs go to clocked latches, one of which is shown in Fig.3.2.7. At the output of the latch, the variable is available in true and complemented form. The first column

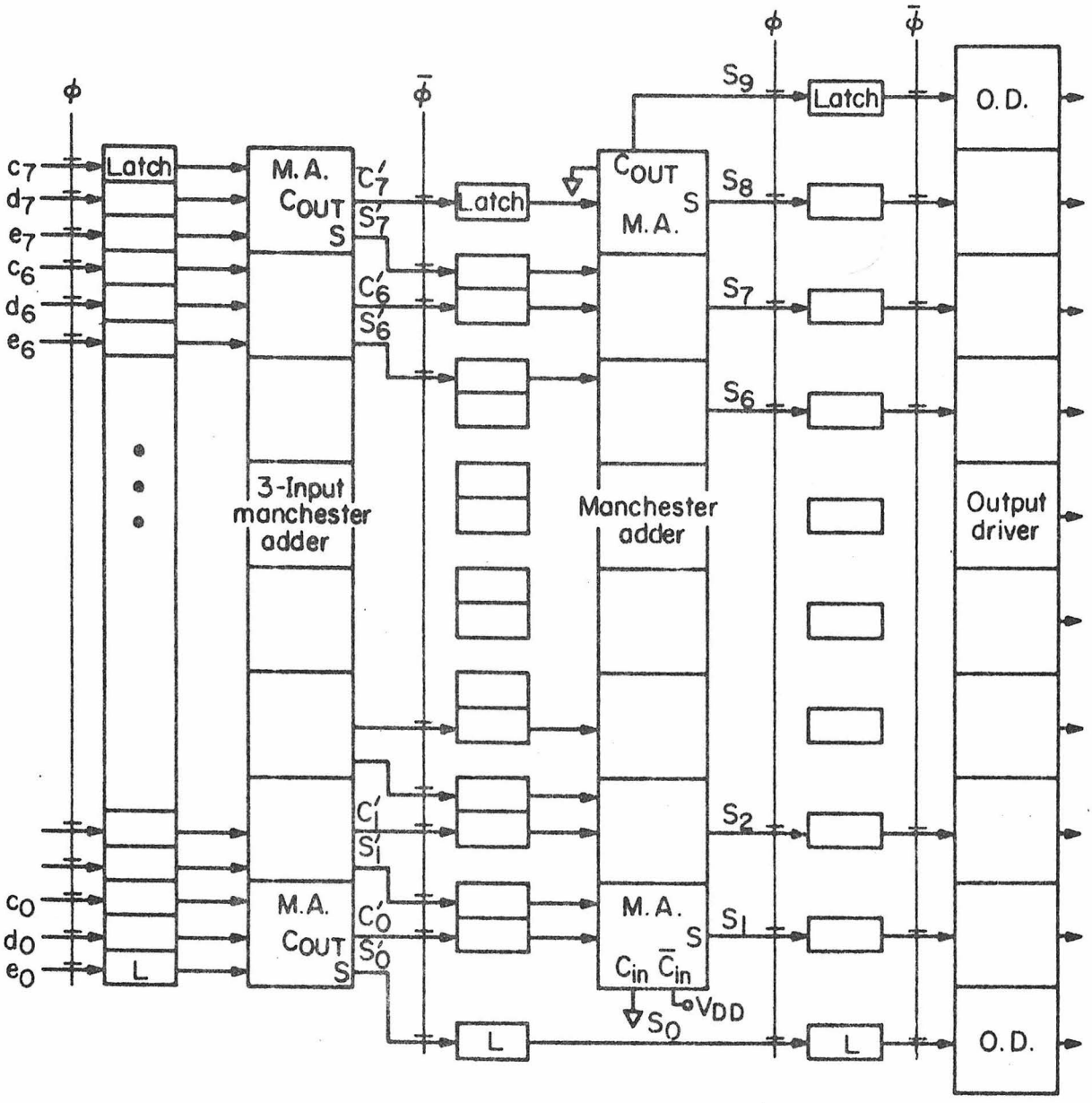



Fig.3.2.3 Part I of edge-enhancement pipeline processor - a 3-input 8-bit adder.

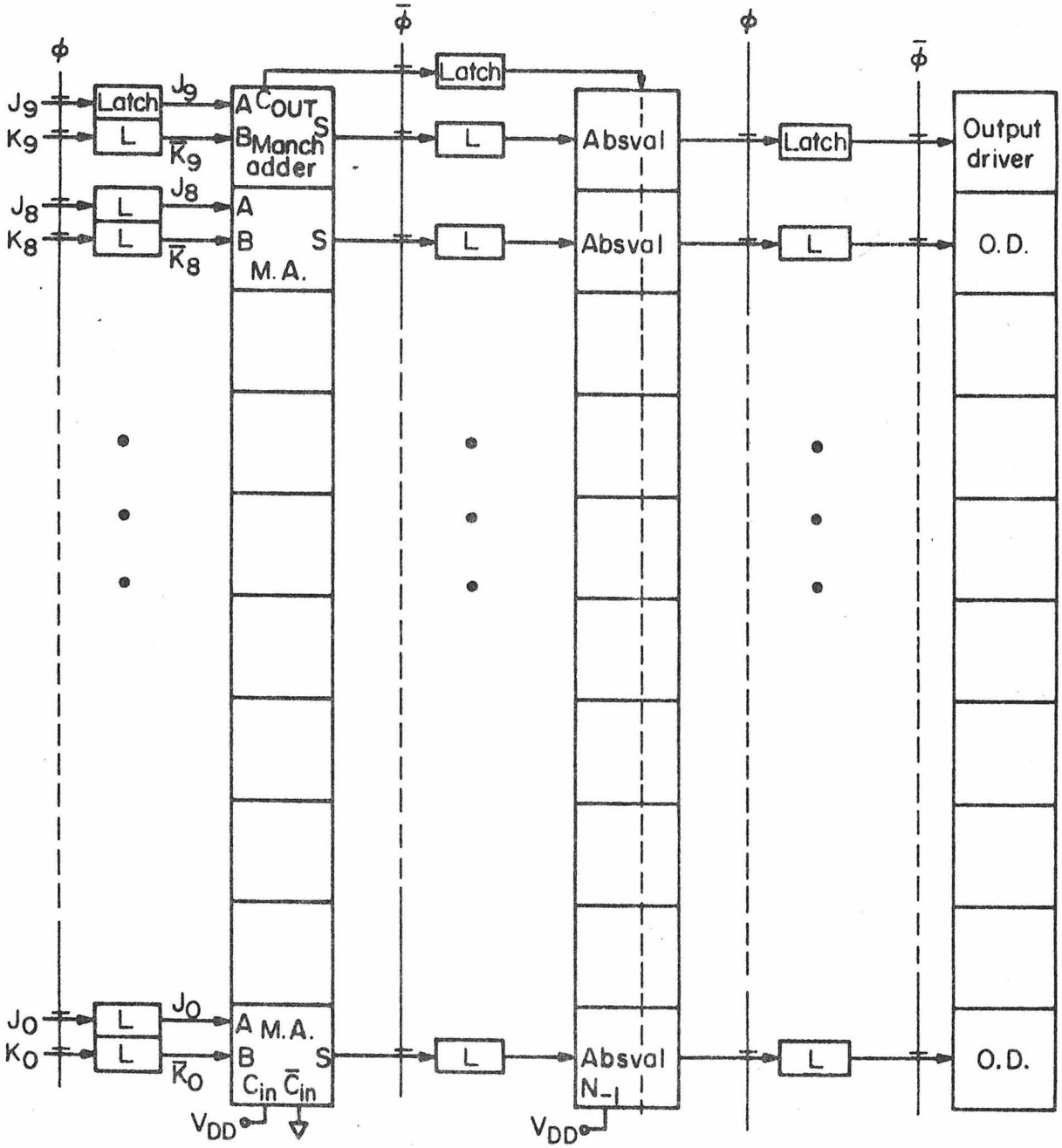



Fig.3.2.4 Part II of edge-enhancement pipeline - the absolute difference of 2 numbers.

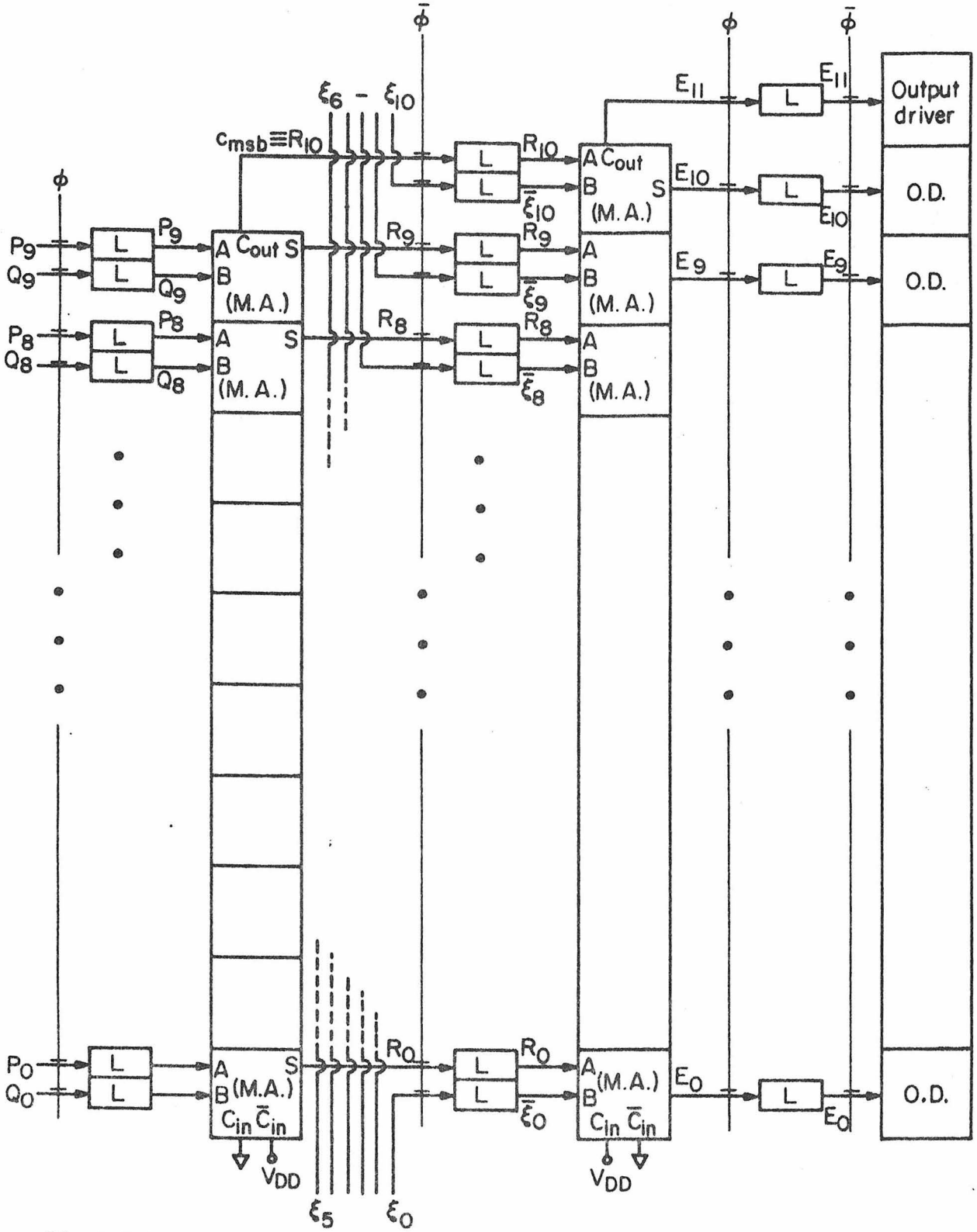



Fig.3.2.5 Part III of edge-enhancement pipeline - sum of column and row gradient magnitudes, and comparison with threshold value.

| $C_{in}$ | A | B | $P=A\oplus B$ | $S=P\oplus C_{in}$ | $C_{out}=P\,C_{in}+\overline{P}A$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 |

Fig.3.2.6(a) Logic table for Manchester Adder.

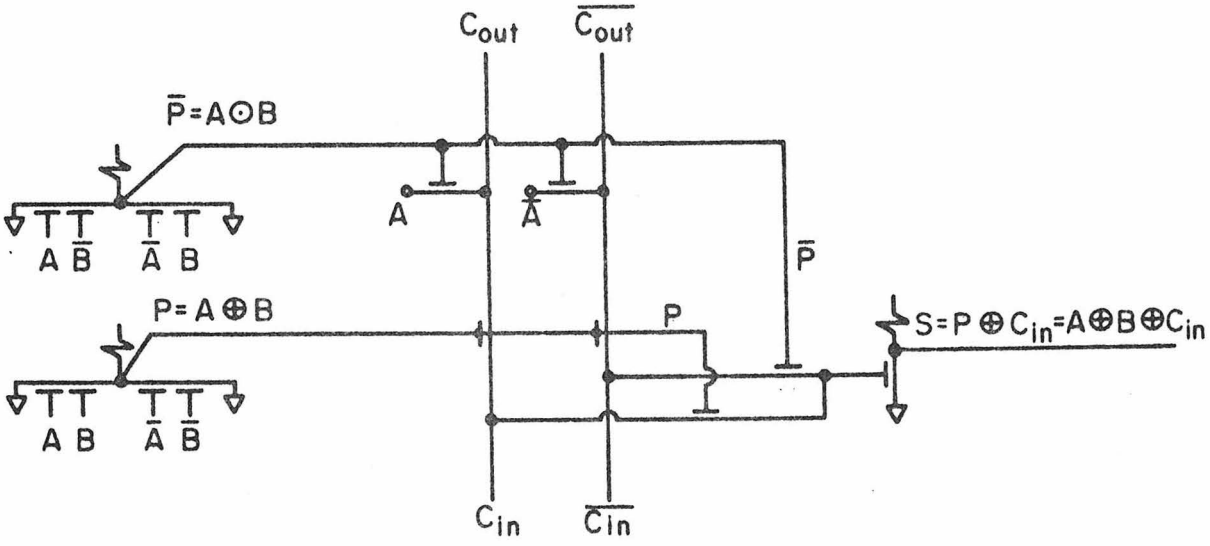



Fig.3.2.6(b) Manchester Adder implemented in MOS technology.

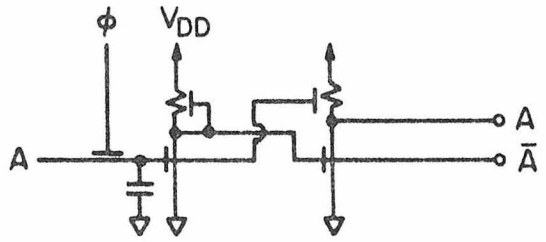



Fig.3.2.7 Clocked latch for pipelined processor.

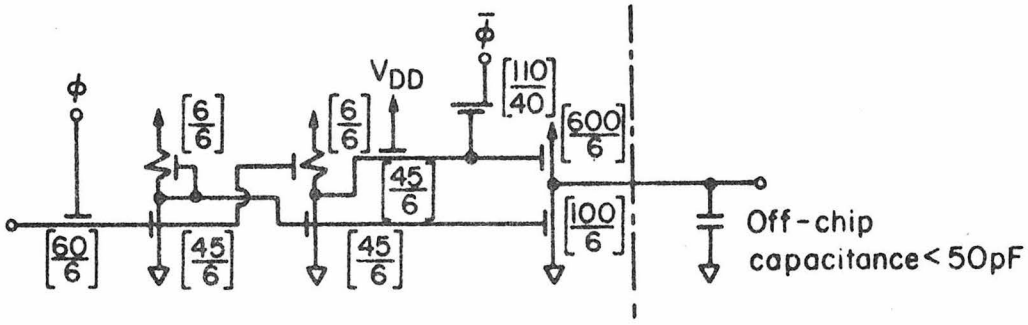



Fig.3.2.8 A bootstrapped output driver designed to handle an off-chip load of 50 pF in 60 ns.

of Manchester Adders adds the three 8-bit inputs in carry-save form, the carry and partial sum being stored in a second column of latches. The carry and partial sum are then added together in the second column of Manchester Adders. The resulting 10-bit number is stored in the next column of latches and is then output off-chip by a column of output drivers. A circuit diagram of the bootstrapped output driver is shown in Fig.3.2.8. It is designed to drive an off-chip load of 50 pF in less than 60 ns.

The longest propagation delay limiting the operating speed of the pipeline is in the sections where addition is occurring. The time taken for completion of a carry through all adder stages is responsible for the major portion of the propagation delay in these sections. Using a MOS simulation program, MSINC, the maximum propagation delay of a section with 10 adder stages was estimated to be 120 ns.

If speed of operation is of essence, it is possible to divide the addition process up into 2 or more sections to reduce the longest propagation delay in any 1 section (Fig.3.2.9). This is particularly advantageous in the case of a large number of adder stages. The price to be paid is the use of extra latches to store the intermediate data. Fig.3.2.9 shows that dividing an 8-stage addition process into 2 sections requires an extra 13 latches. In the

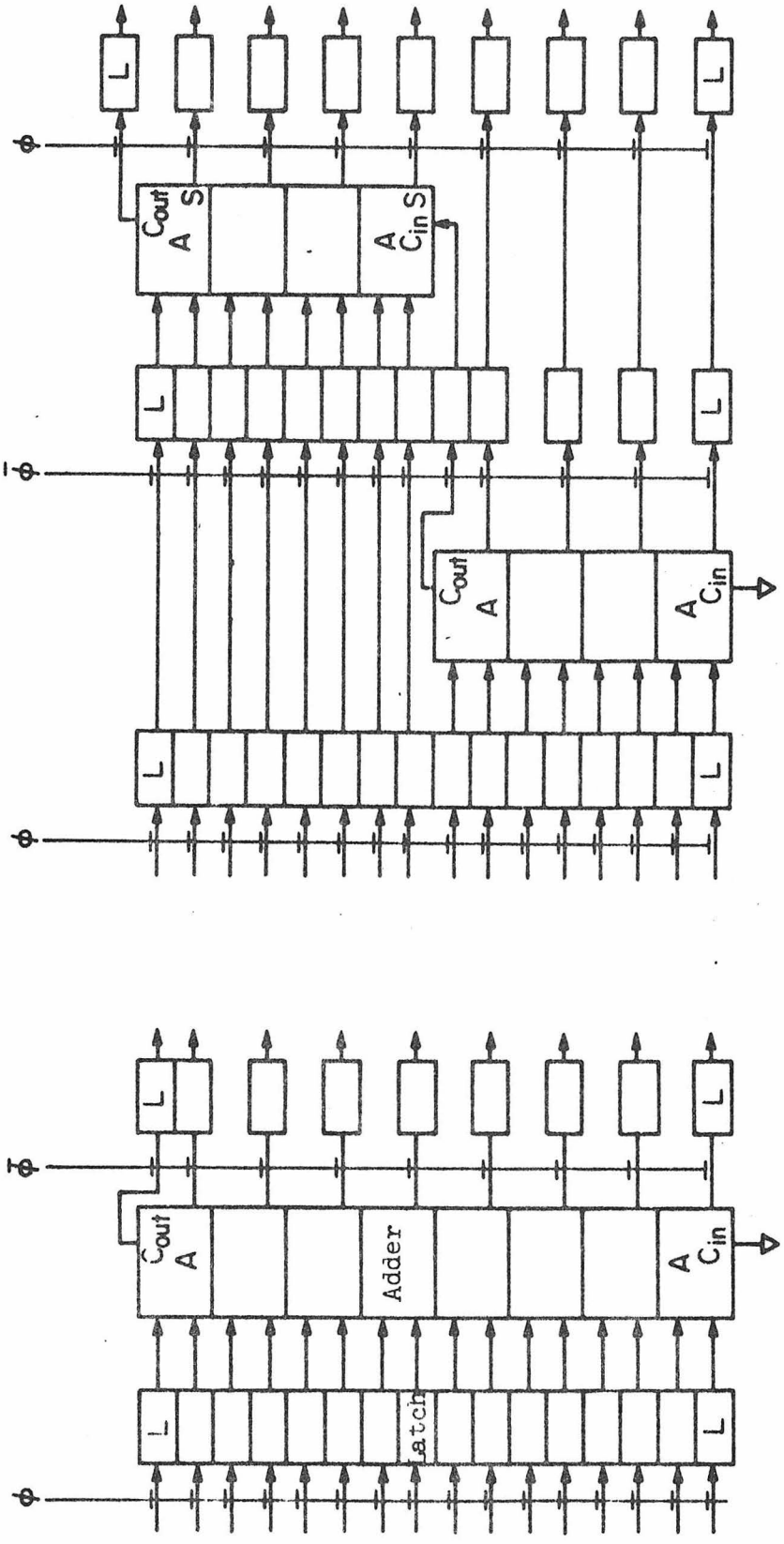



Fig.3.2.9 Dividing an addition section into 2 sections to reduce the propagation delay. Thirteen extra latches are required to store the intermediate data.

pipeline implementation of the edge-enhancement operator, the maximum propagation delay in the addition sections was not considered large enough to impede real-time operation of the device. Hence no attempt was made in this case to divide up the addition sections.

Part II of the pipeline consists of a subtractor and an absolute-value logic block (Fig.3.2.4). Subtraction is performed in 2's complement fashion. The result is sent to the absolute-value logic section, which examines the most significant bit to determine if it is positive or negative. If the result is negative, the 2's complement is formed and stored in the next column of latches. If the result is positive, it is stored in the next column of latches directly, and then sent to the output drivers. Figs.3.2.10(a), (b), (c) show the logic design for the absolute-value cell.

In Part III of the edge-enhancement pipeline, the P and Q outputs from Part II are added together, and the result, R, is sent to a comparator (Fig.3.2.5). In the comparator section, the difference of R and the edge threshold, T, is obtained in 2's complement form and the result stored in the next column of latches to be output by the drivers. The most significant bit indicates if an edge is present (msb = 1) or not (msb = 0). The maximum

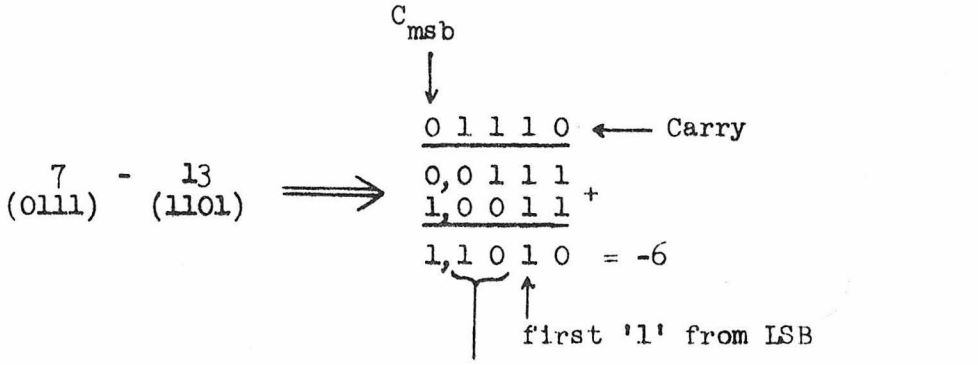



Fig.3.2.10(a) Forming the absolute difference of 2 numbers in 2's complement fashion.

| $C_{msb}$ | $N_{i-1}$ | $S_i$ | $N_i$ | $A_i$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | x | 0 |
| 1 | 0 | 1 | x | 1 |
| 1 | 1 | 0 | x | 0 |
| 1 | 1 | 1 | x | 1 |

$$N_i = \bar{S}_i N_{i-1}$$

$$A_i = C_{msb} S_i + \bar{C}_{msb} N_{i-1} S_i + \bar{C}_{msb} \bar{N}_{i-1} \bar{S}_i$$

Fig.3.2.10(b) Logic table for absolute value Boolean function.

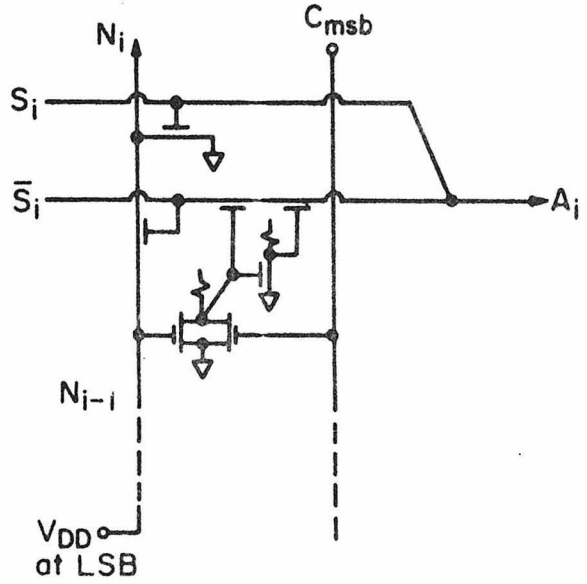



Fig.3.2.10(c) Absolute value cell implemented in MOS technology.

propagation delay of the edge-enhancement pipeline is in the comparator section. Here, a carry from the least significant bit adder may propagate through 11 adder stages.

Fig.3.2.11 shows the layout of the edge-enhancement pipeline using n-MOS LSI technology. The design for the 3 parts measures approximately 3000μ x 5000μ. Minimum transistor channel length is about 5.0μ. At the time of this writing, the chip is being fabricated, and no results of its performance are as yet available. A breadboard of the design, however, has been constructed using discrete TTL logic components and interfaced with the JPL robot's vision system. The pipeline is being operated at a cycle period of about 727 ns (244x188 pixels x 30 frames/sec ≈ 1.38 x 10**(6) pixels/sec) to enable the edge-enhancement operation to be performed in real-time (video rates).

## 3.3 A PIPELINE APPROACH TO CHAIN-CODING

Sec.2.3.6 described the chain-coding process as collecting connected edge points in the thinned image to form a linked-list representation of intensity edges in the original image. This section examines the use of a pipelined structure for chain-coding.

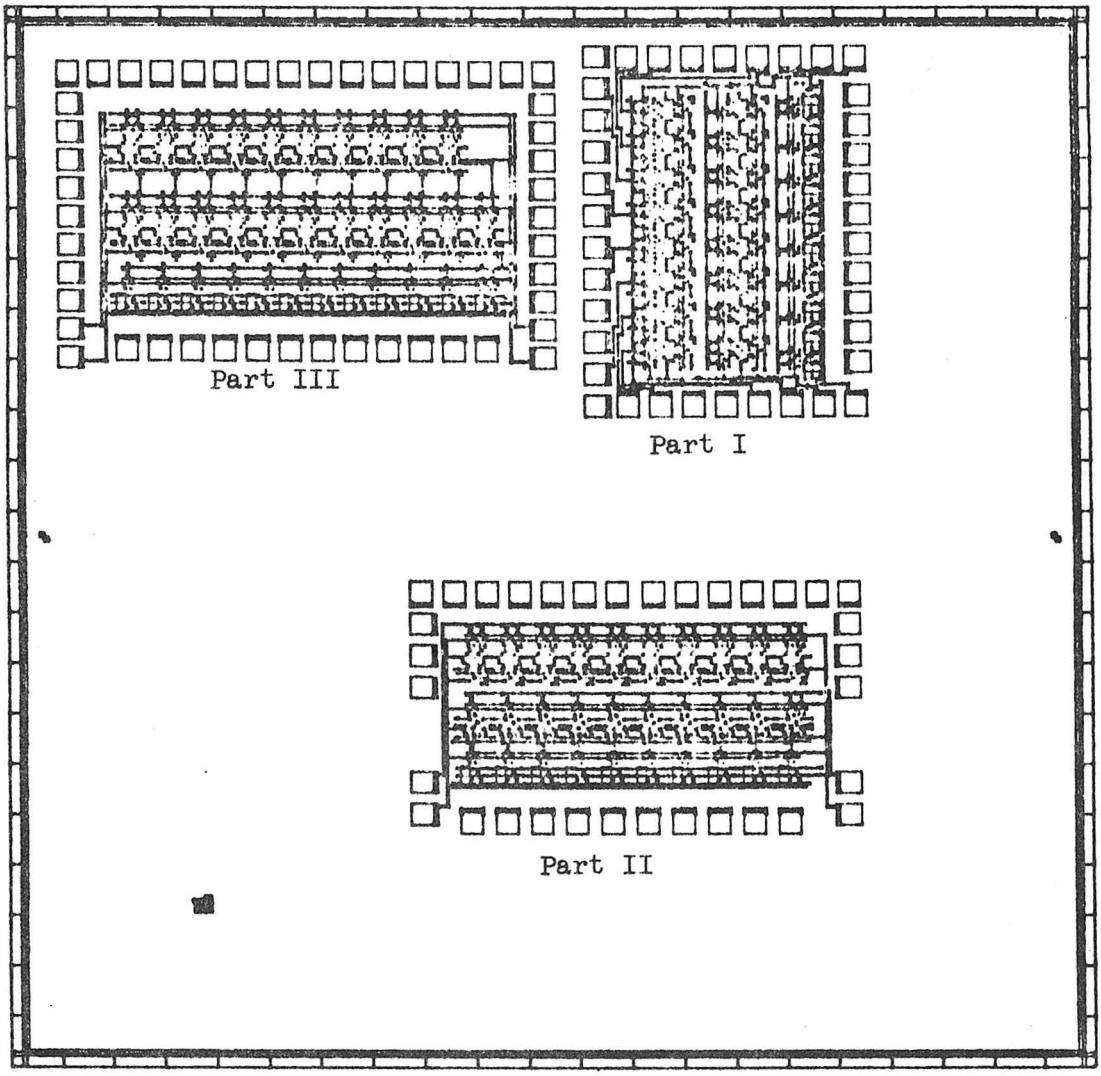



Fig.3.2.11 Metal interconnect layer of edge-enhancement pipeline chip.

When using a pipeline structure for chain-coding the edge points are collected in raster-scan fashion. A sliding 3x3 window may be used for linking an edge point to its neighbors (Fig.3.3.1). Because the edge points are being collected in raster-scan mode, it is only necessary to link the neighbors in a "forward-looking' direction, i.e., in the d, e, f and g directions. Links to neighbors in the h, a, b and c directions have already been formed previously.

An example of tracking an edge in raster-scan mode is shown in Fig.3.3.2. The chain-coding structure scans the edge map until it encounters the edge point in position (1,6), whereupon it places that coordinate in S1, and also the link to the next neighbor, e. The location of the edge in the present scan line (in this case, column 6) is also entered in CR1, to keep track of the edge. In the next scan line another link (f) is entered in S1, and the location of the edge in the present scan line (column 7) in CR1. In the same scan line, another chain code is started in S2, with (2,13) as the initial coordinate. As the chain-coding structure continues to scan the edge bit map, the chain codes in S1 and S2 are built up. When the edge point in location (6,10) is encountered, it is recognized as a node. The chain codes in S1 and S2 are terminated and a new chain started in S1. The links for this chain are collected in

| | | |
|---|---|---|
| a | b | c |
| h | z | d |
| g | f | e |

Fig.3.3.1 3x3 window for linking an edge point with its neighbors.

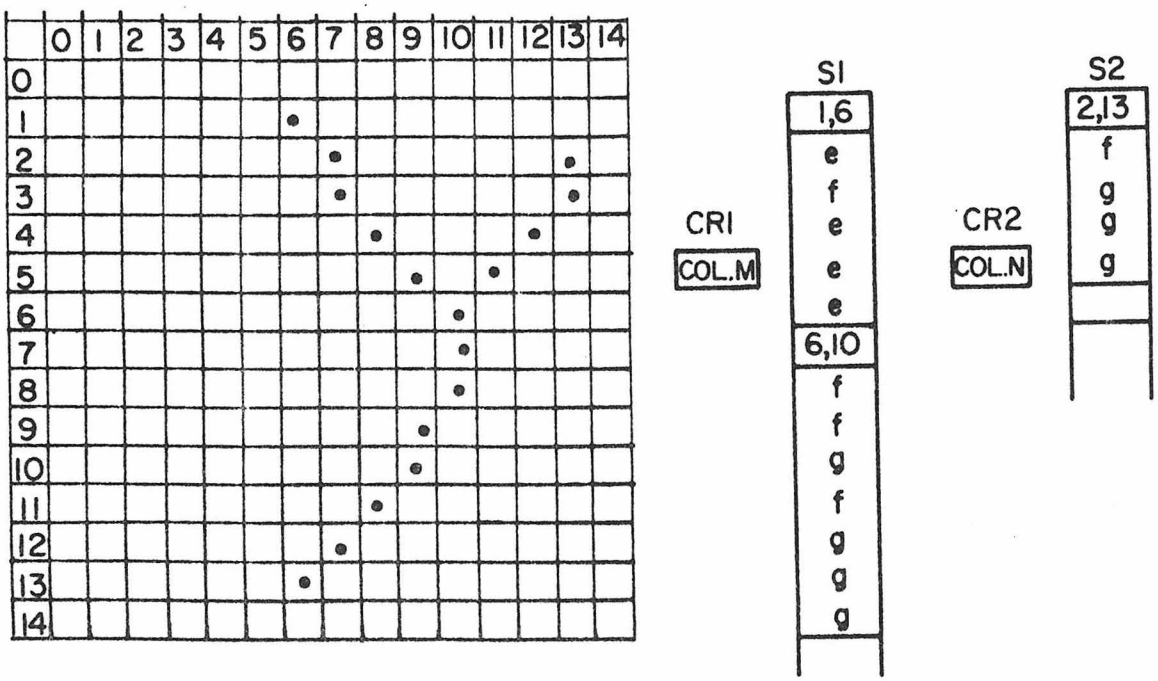



Fig.3.3.2 Example of tracking an edge by raster scan method.

the same manner until the end point is encountered in location (13,6).

There are some special cases to be considered. Since the edge points are being collected in raster-scan mode, formation of the chain codes is dependent on the orientation of the edges. Because of this, even simple, smoothly changing edges may be fragmented into a number of chain codes. Three examples are given in Fig.3.3.3. In (a), when a hump-shaped edge occurs, the left and right sides will have to be tracked separately. Similarly, when a U-shaped edge occurs as in (b), two pieces of chain codes result. In (c), when a horizontal edge (S2) is encountered such as at (5,5), it is not known if S2 is connected to S1 until the edge point at (5,10) is collected. Hence 3 chain codes result in this case.

It is possible and desirable to unite fragmented edges together after the chain-coding process is completed, by noting the chain codes that have false terminating or starting points (i.e., not node or end points), such as those found in Fig.3.3.3. The fragmented chain codes could be joined by noting which of the false terminating or starting points have coinciding coordinates, and reversing the direction of the links in one of the chain codes.

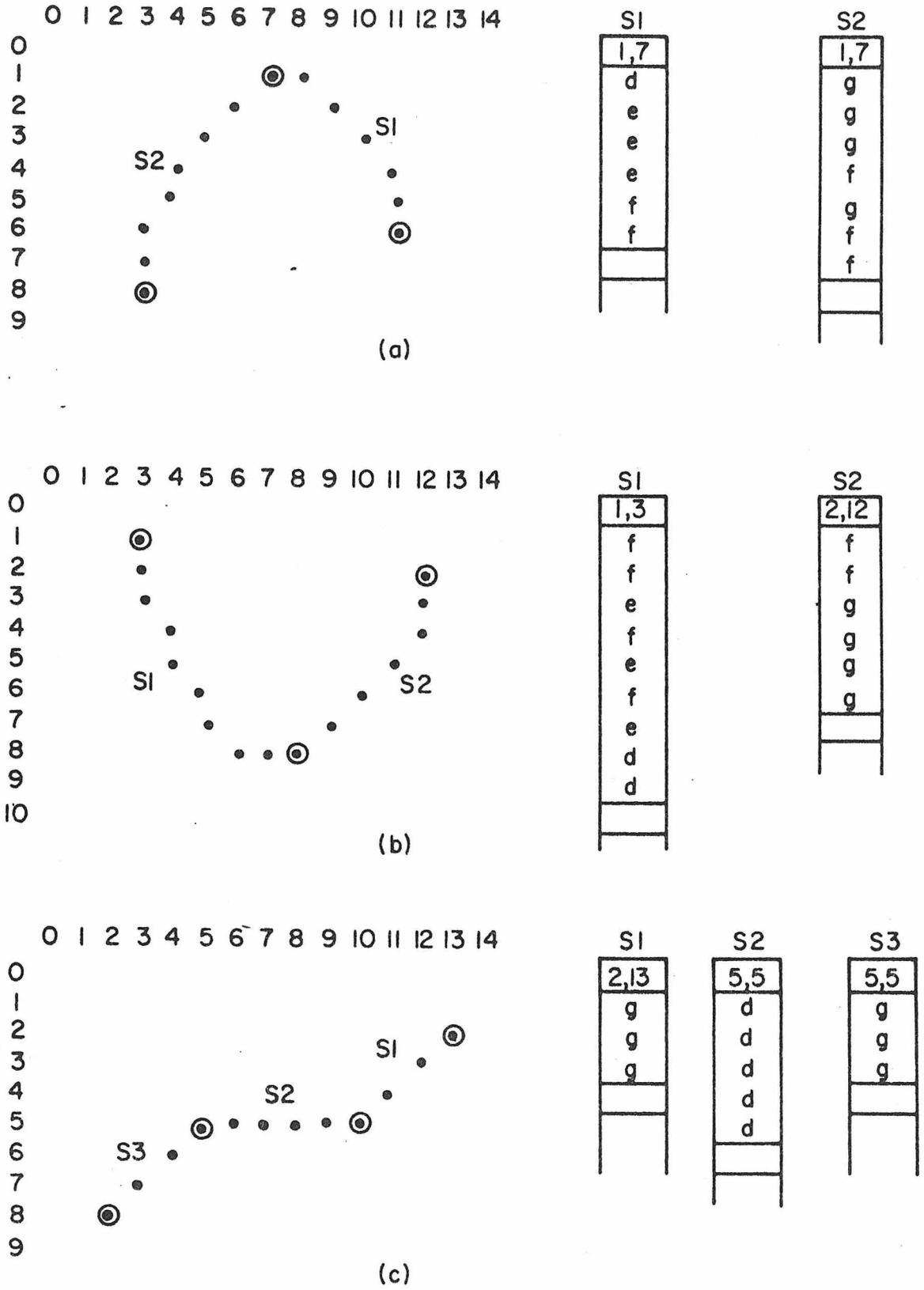



Fig.3.3.3 Three examples of fragmenting smoothly changing edges when using raster-scan method to collect edge points.

How many stacks are required for collecting the edge points? The worst case for a 512x512 image is when there are 256 vertical edges to be tracked in a scan line. Then 256 stacks would be needed to follow the edges. Also, since there may be more than 256 pieces of chain codes in an image, it is desirable to have a stack management scheme whereby stacks containing chain codes that have terminated are returned to a pool for re-use. Otherwise the number of separate stacks required to store all the chain codes may be quite large. The management scheme would maintain a list of unused stacks, and assign stacks to track new edges as they are encountered in the image. Since dynamic allocation of the stacks to the edges to be tracked must be accomplished in real-time, quite stringent demands are made on the hardware for performing the allocation.

It is possible to perform the chain-coding process while avoiding the problems associated with the real-time management of stacks to track the edges, and also the fragmentation of edges. The next section describes a random-access structure for collecting edge points in a buffered edge map. Using this structure, an extra frame delay is introduced in forming the edge chain codes for a frame.

## 3.4 A RANDOM-ACCESS STRUCTURE FOR PERFORMING CHAIN-CODING RAPIDLY

To avoid fragmenting smoothly changing edges when forming chain codes in the thinned image, it is desirable to be able to track along an edge. To have this capability, it is necessary to buffer the edge map in random-access storage. Furthermore, the chain-coding process could be speeded up if it were possible to access and examine 3x3 neighborhoods simultaneously. To accomplish this, the thinned image is divided up and stored in 9 memory banks so that no 2 pixels in a 3x3 neighborhood are stored in the same memory bank (Fig.3.4.1). Any 3x3 neighborhood of pixel values could then be accessed simultaneously by properly indexing the location of the 3x3 neighborhood for each memory bank. As the node and end point list is being generated in the preceding process, the thinned image is written into the 9 random-access memory banks. Each datum in this buffer contains 2 bits (z[1:0]), which encodes whether an edge point is present (z[1:0]=01), and if so, whether it is a node or end point (z[1:0]=11). Two such buffers are necessary; while the preceding pipeline process is filling one buffer, data in the other buffer can be chain coded.

The overall chain-coding structure is shown in

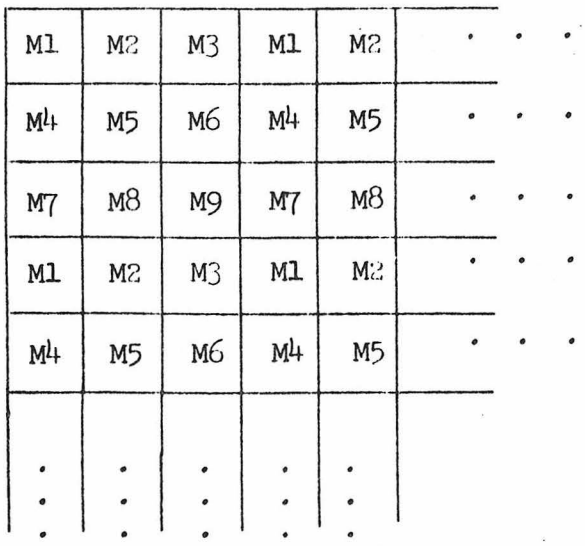



Fig.3.4.1 Pixels in the thinned image are stored in 9 memory banks as shown, to permit simultaneous access of pixel values in a 3x3 neighborhood.

Fig.3.4.2. The location of the 3x3 neighborhood to be accessed is sent to the neighborhood location decoder, which translates this location into addresses for fetching the neighborhood of pixel values (a:h and z[1]) from the 9 memory banks. For the memory bank containing the central pixel value in the neighborhood (the z-th pixel value), z[1] is read to see if this is a node or end point (if z[1]=1), while a '0' is written into z[0] to indicate that this edge point has been collected. The 9 pixel values are then sent to a multiplexor, which arranges the pixels in the correct order for sending to the chain-coding units (CCU's). Starting with pixel 'a', in clockwise fashion, the 8 CCU's examine the neighborhood of pixel values to determine the next member in the chain. CCU1 is first activated, and then the succeeding CCU's, until an edge point is found, which becomes the next member of the chain. Since edge points are erased as they are collected, and contour points have only two neighbors which are not connected, an edge point found in the neighborhood automatically becomes the next member of the chain. This CCU's code number is output as the next link in the chain. This code number is also sent to a chain code decoder to increment or decrement the address in the X and Y registers to provide the correct address of the next 3x3 neighborhood to be fetched.

When a node or end point is encountered in forming a

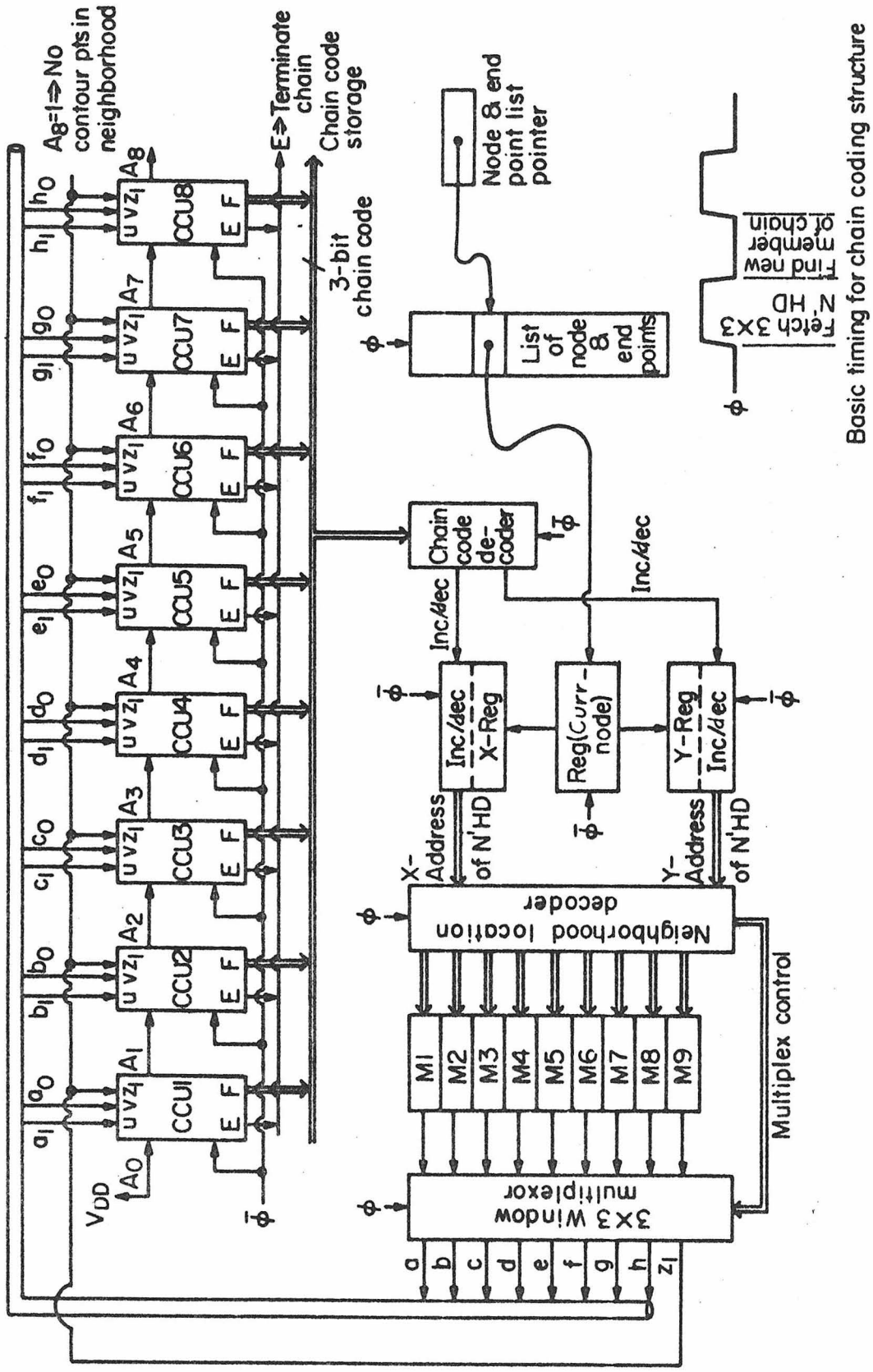
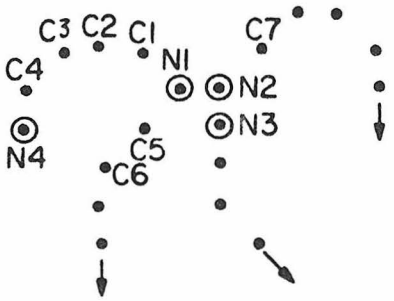


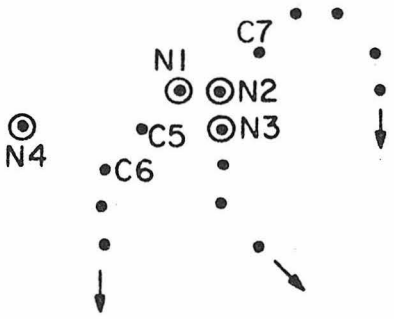

Fig.3.4.3 Overall chain-coding structure.

chain (E=1), the chain is terminted, and the process returns to the current starting node to see if there are other branches emanating from that node. If there are, the branches are chain coded, if not (A(8)=1), a new node address is transferred from the node and end point stack to the X, Y registers, and its branches chain coded.

Fig.3.4.3 shows an example of chain-coding, which covers instances of starting a chain from a node, collecting edge points for the chain, terminating the chain on encountering another node, returning to the current starting node, chaining another branch, and then moving on to a new starting node when all the branches for the current starting node have been chain coded. The timing for the instructions to be executed in each cycle is indicated in the figure, and corresponds with the clocking of components in Fig.3.4.2. To enable the formation of chains at real-time rates it is desirable to be able to collect 1 edge point every pipeline cycle. The actual requirement is less stringent, when one considers that edge points constitute less than 10% of points in an image, even for reasonably textured scenes (such as the CAR scene, about 6.6% are edge points). The implication is that chains could be formed in real-time even if 1 edge point is collected every 2 or 3 pipeline cycles.

The truth table and a logic diagram for a CCU is



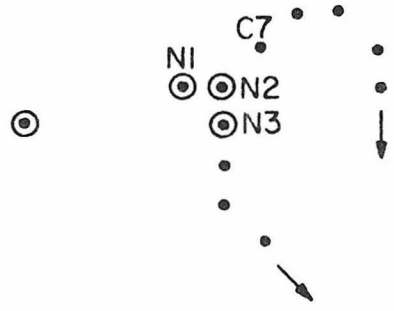




Fig.3.4.3 Example of chain-coding using the random-access structure shown in Fig.3.4.2.

| | |
|---|---|
| $\not\emptyset$ | Address(N1) in Reg(Curr_Node) → X, Y-registers; |
| $\emptyset$ | N'hd(N1) → CCUs; N1:= 10; |
| $\not\emptyset$ | Examine n'hd(N1); First_link:= C1; Inc/dec X, Y-registers; |
| $\emptyset$ | N'hd(C1) → CCUs; C1:= 00; |
| $\not\emptyset$ | Examine n'hd(C1); Next_link:= C2; Inc/dec X, Y-registers; |
| ⋮ | ⋮ |
| $\emptyset$ | N'hd(C4) → CCUs; C4:= 00; |
| $\not\emptyset$ | Examine n'hd(C4); Next_link:=N4; E:= 1; |
| $\emptyset$ | Terminate chain (E = 1); |
| $\not\emptyset$ | Address(N1) in Reg(Curr_Node) → X, Y-registers; |
| $\emptyset$ | N'hd(N1) → CCUs; |
| $\not\emptyset$ | Examine n'hd(N1); First_link:= C5; Inc/dec X, Y-registers; |
| $\emptyset$ | N'hd(C5) → CCUs; C5:= 00; |
| ⋮ | ⋮ |
| $\emptyset$ | Terminate chain; |
| $\not\emptyset$ | Address(N1) in Reg(Curr_Node) → X, Y-registers; |
| $\emptyset$ | N'hd(N1) → CCU's; |
| $\not\emptyset$ | Examine n'hd(N1); $A_8$:= 1 (No contour points in n'hd); |
| $\emptyset$ | NEP_list(Next_Node=Address(N2)) → Reg(Curr_Node); |
| $\not\emptyset$ | Inc. NEP_list pointer; Address(N2) in Reg(Curr_Node) → X, Y-registers; |
| $\emptyset$ | N'hd(N2) → CCUs; N2:= 10; |
| $\not\emptyset$ | Examine n'hd(N2); First_link:= C7; |
| ⋮ | ⋮ |

shown in Figs.3.4.4(a) and (b). Inputs u, v and z[1] are clocked on CLKNEG. An activate line (A(i)) serves to signal succeeding CCU's whether the search for an edge point is still active. Once an edge point is found, A(i) is deactivated, and F activated so that the CCU's code number is sent to the CCU code bus. When a node with no contour points in its neighborhood is encountered, A(8) is activated to indicate that a new node address should be transferred into the X, Y registers from the node and end point stack. Similarly, when a node or end point is encountered in collecting edge points for a chain, the E line is activated to indicate termination of the chain and a return to the current starting node.

## 3.5 SUMMARY

In this chapter I have described the use of pipeline processors for performing some of the edge-extraction operations in real-time. A pipeline processor for the edge-enhancement operation has been designed using n-MOS LSI technology to demonstrate methodology and feasibility. A breadboard version of the design has also been constructed using discrete TTL logic and interfaced with the JPL robot's vision system to demonstrate the design concept. I have also compared the use of a pipeline structure and a

| $z_1$ | $A_{-i}$ | u | v | $A_i$ | F | E |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | X | X | X |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 |

Fig.3.4.4(a) Truth table for a CCU.

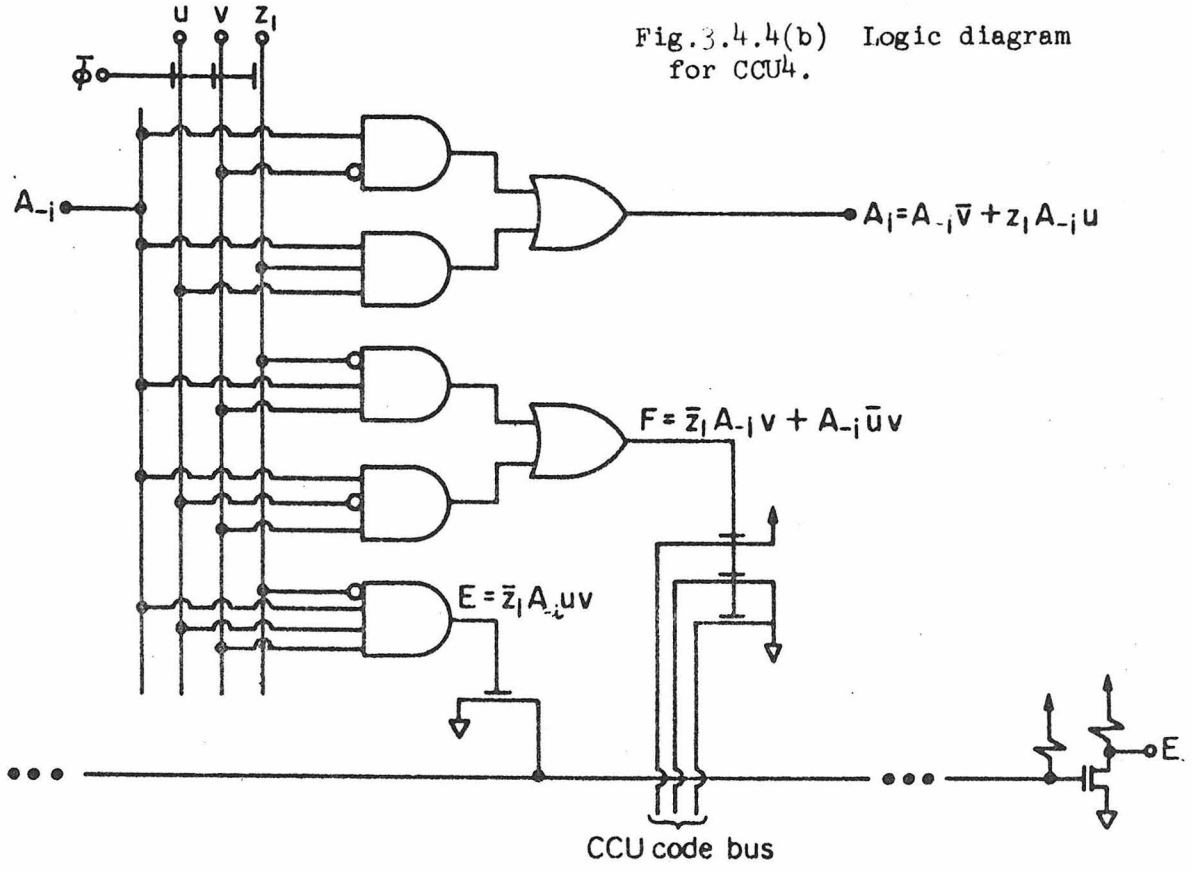



Fig.3.4.4(b) Logic diagram for CCU4.

random-accessed structure for the chain-coding process. From this comparison, I have arrived at the conclusion that the latter is more suitable for chain-coding.

In the next chapter I will examine how advances in micro-electronics may affect the computational structures for image processing. Also I will look at CCD technology, and how this may be used for image processing.

# CHAPTER 4
# IMPACT OF FUTURE TECHNOLOGICAL ADVANCES ON IMAGE PROCESSING

In this chapter, projections will be made on how some advances in electronics would affect machine image processing in the next 10-15 years. The advances are categorized as follows:

(1) INCREASING GATE DENSITY OF LSI TECHNOLOGIES

As VLSI systems and larger chip sizes are introduced, the following question will be raised - will this permit large parallel arrays of PE's to be constructed so that machine vision processing could be performed by such structures? This question will be dealt with in Sec.4.1.

(2) USE OF CCD ANALOG CIRCUITRY FOR PERFORMING IMAGE PROCESSING

Using CCD's for peforming image processing has certain advantages over digital techniques. Researchers at Hughes Research Labs are investigating this approach for image processing. A comparison will be made between the CCD approach and the digital pipeline technique in Sec.4.3.

## 4.1 ADVANCES IN FABRICATION OF MICRO-ELECTRONIC CIRCUITRY

The past decade has seen a remarkable reduction in the size of electronic components, allowing more and more functions to be integrated on a chip. So far we have seen the development of the microprocessor and the 16K-bit RAM. The increasing circuit complexity of LSI chips is discussed under (a) ultimate dimensions of MOS devices that is achievable. It is expected that the ultimate gate densities for other LSI processes will not be significantly better than that for MOS. (b) Improved resolution of the lithographic process and (c) fabrication of larger chips resulting from reduction in defect densities in the Silicon crystal wafers used for fabrication of the chips.

### (A) LIMITS ON FEATURE SIZE OF MOS TRANSISTORS

The ultimate gate density of the MOS technology will be examined here. Honeisen and Mead, in studying the limits imposed by solid-state physics, have estimated that the minimum channel length for a MOS transistor, L, is about 0.2u [HON.72]. The minimum channel length that can be reliably achieved, however, is probably about 0.5u.

As device dimensions scale down, the device density

will scale as 1/(L**2). Table 4.1 shows the device densities that can be expected from decreasing device sizes. The projected gate densities for L=1.0u and 0.5u assume that other problems such as electromigration of metal connections caused by increasing current density, increased number of interconnections between circuits on-chip, and increased power dissipation requirements will be solved. The ultimate gate density for MOS LSI may reach 10**6 gates/cm**2. The consequence of this as far as constructing large arrays for image processing is presented in Sec.4.2.

(B) LITHOGRAPHIC IMPROVEMENTS AND (C) FABRICATION OF LARGER CHIPS RESULTING FROM REDUCTION IN DEFECT DENSITIES IN SILICON CRYSTAL WAFERS.

Presently the minimum line width of LSI processes is limited by the wavelength of light (about 0.5u) to about 1-2u because it is difficult to perform multiple alignments of masks more accurately than 2-3 wavelengths of light. Development is underway to use electron-beam pattern generation and X-ray imaging systems to reduce the minimum line widths further. With such systems it is possible to reproduce line widths 0.1u across [KEY.77]. This will enable the fabrication of devices approaching the dimensional limits imposed by solid-state physics.

| Time scale | MOS device channel length L (u) | Device density for random logic (devices/$cm^2$) | Equivalent gate density (gates/$cm^2$) |
|---|---|---|---|
| Now (1979) | 5.0 | $5.0 \times 10^4$ | $10^4$ |
| 5 years | 1.0 | $1.25 \times 10^6$ | $2.5 \times 10^5$ |
| ? | 0.5 | $5.0 \times 10^6$ | $10^6$ |

Table 4.1 Projected gate densities for MOS LSI technology

| Feature to be compared | CCD processor | Digital Pipeline processor |
|---|---|---|
| Frequency of operation (for 512x512 images at 30 frames/sec requires ~ 7.5 MHz) | $\lesssim$ 10 MHz, $t_c$ = 100 ns | ~5 MHz, $t_c$ = 200 ns (this could be increased by using a greater degree of pipelining) |
| Silicon area utilized (not including pads, interconnections, or off-chip drivers) | ~ 0.09 $cm^2$ | ~ 0.133 $cm^2$ |
| Computational precision | ~ 6 bits | Present version uses 8 bits |

Table 4.3 Comparison of CCD processor and digital pipeline processor for edge-enhancement operator.

As device sizes are being reduced, so are the defect densities in Silicon crystal structures. This permits the fabrication of larger chips with acceptable yields. Noyce predicts that larger chips, coupled with increasing gate densities will result in an increase of chip functional complexity by over 2000 in the next 15 years [NOY.77].

As line widths become smaller and chips larger, however, a limit may be reached in the precision of aligning mask layers. A study by Sutherland, et.al., indicates that the alignment precision may be limited by the coefficient of expansion of Silicon and common metals to about 1 part in 10**5 [SUT.76]. Hence if a minimum line width of 1u is used, the alignment accuracy (about 0.2u) may limit the maximum chip size to about 2 cm (assuming single alignment per mask level). Multiple alignments per mask level may be possible, and would permit fabrication of larger chips, but may be much more costly [SUT.76].

## 4.2 EFFECT OF INCREASING CHIP COMPLEXITY ON COMPUTATIONAL STRUCTURES FOR IMAGE PROCESSING

One of the main obstacles to using array processors for performing image processing is the difficulty in implementing sufficiently large arrays to cover

reasonable-size images (of about 500x500 pixels). Will increasing chip complexity enable sufficiently large arrays of PE's to be constructed for image processing? Specifically, would it be possible to integrate an entire 500x500 array of PE's on a chip? To discover if this is feasible it is necessary to estimate the number of gates it would take to construct the array. From the work done by Duff, et.al. [DUF.76], it is estimated that there are 625 gates per CLIP4 PE. Hence 250,000 PE's would require 156 x 10**6 gates. This is more than 2 orders of magnitude greater than the ultimate number of gates per chip without multiple alignments per mask level. Even with multiple alignments, an array of 500x500 PE's would cover a chip area 156 cm**2 (12.5 cm square) at ultimate gate density of 10**6 gates/cm**2. Hence, fabricating such an array on a chip would be quite a formidable task.

## 4.3 COMPARING CCD ANALOG TECHNIQUES FOR IMAGE PROCESSING WITH DIGITAL METHODS

An alternative to using digital methods for performing image processing is provided by CCD analog techniques. The CCD approach has been investigated by researchers at Hughes Research Labs. They have successfully implemented a number of image operations using CCD

techniques. Among these are the Sobel edge operator, smoothing, gap filling, adaptive stretching and unsharp masking [NUD.77,78]. This section will provide a comparison of processing bandwidth, LSI area utilized and computational accuracy between the CCD Sobel processor and the digital pipeline implementation of it.

The CCD computational structure for the Sobel operator is similar to a digital pipeline in operation. Charge packets representing data are shifted sequentially through the different stages of the CCD processor, where different transformations are applied to the data at each stage.

Table 4.3 summarizes the performance figures for the CCD Sobel processor and the digital pipeline implementation of it. The CCD processor can be operated up to 10 MHz, whereas the present design for the digital pipeline device has an estimated maximum operating frequency of 5 MHz. This is not a true limitation on the speed of pipeline designs, however, since it is always possible to reduce propagation delays in any section by dividing the work in that section into 2 or more sections. The thruput of the pipeline would thus be increased, at the expense of greater latency. This is true whether digital or CCD analog techniques are used, if the design is of a pipeline nature.

The silicon area utilized by the digital pipeline (not including pads or off-chip drivers) is about 50 % more than for its CCD counterpart. The number of stage delays is approximately the same in both cases. The computational precision for the CCD processor is estimated to be equivalent to 6 bits of digitization, as opposed to 8 bits with associated round-off errors for the digital pipeline processor. 6-bit precision is probably adequate for most image processing applications.

Summarizing, the advantages of the CCD approach for image processing are (i) the circuitry utilizes less chip area, (ii) fewer interconnections are required for data paths, and (iii) the pixels do not require digitizing. For the digital method, the advantages are (i) pixel values can be represented to a greater precision, (ii) the circuitry for implementing the image functions is less dependent on variations in chip fabrication, (iii) computation and data transfer are more immune to noise, and (iv) the circuitry is more easily interfaced with a digital computer.

# CHAPTER 5

# CONCLUSIONS AND DIRECTIONS FOR FURTHER RESEARCH

In this thesis I have addressed the problem of using a machine vision system for real-time control purposes. An important capability of such a vision system is the ability to track moving objects. To track a moving object, it would be helpful to perform scene analysis initially to identify the objects in the scene. Since scene analysis is usually a complex process, however, it would be difficult to perform analysis on each of a sequence of images of the scene fast enough to permit object tracking. Instead, edge features are extracted from each of the sequence of images, and by relating these features to edge projections from stored 3-D models (assuming the object to be tracked is familiar to the vision system), the moving object's position may be specified. Sec.5.1 presents a scenario for using extracted edge features to track a moving object.

A sequence of image operations has been presented for extracting edge features from digital images. In this sequence, an image of about 500 x 500 pixels is reduced to a few hundred line data elements representing intensity edges

in the image. In order to extract the edges in real-time, computational structures for performing the edge-extraction operations rapidly have been studied. From this study it is concluded that parallel array structures are not suitable for executing the algorithms. One of the main obstacles to using parallel array structures is the difficulty in constructing sufficiently large processing arrays to cover reasonable-size images. This problem will remain even when ultimate LSI gate densities are used in constructing the arrays.

Pipeline processing is more suitable for performing the edge-extraction operations in real-time. Most of the edge-extraction operations can be efficiently performed using pipeline processors. The exception is the chain-coding process. From a study comparing the use of a pipelined and a random-accessed structure for chain-coding, it appears that the latter is more suitable for performing chain-coding.

To demonstrate methodology and feasibility, a pipeline unit to perform edge-enhancement on digital images in real-time has been designed using n-MOS LSI technology. A breadboard version of the pipeline design has also been constructed, using discrete TTL components, and interfaced

with the JPL robot's vision system.

Further work needs to be done to realize a real-time object-tracking vision system. Computational structures for the other edge-extraction operations are yet to be designed and built. It is desirable to integrate as many of these operations on an LSI chip as possible, thereby making the use of this object-tracking system cost-effective in many applications.

More work is also necessary to implement a scenario for using the edge features extracted from each of a sequence of images to track a moving object. The next section describes this scenario in greater detail.

## 5.1 A SCENARIO FOR TRACKING A MOVING OBJECT USING EDGE FEATURES EXTRACTED FROM A SEQUENCE OF DIGITAL IMAGES

When using a machine vision system for real-time control purposes, the objects in the scene may or may not be familiar to the vision system. If the object to be tracked is unfamiliar to the vision system, or has an undefined shape (such as a rock), then tracking the object may be accomplished by following a set of edge features presumed to

belong to the object, from frame to frame. Perhaps all that could be said in tracking the object is this case might be

> "...the object producing the edge contour, C (a particular set of edge features), is presently located in the top-right hand corner of the image, and appears to be moving towards the left at angular velocity, 'v' arc-deg/sec..."

Little else could be said about the orientation of the object, or its exact position. Also, when occlusions occur it would be difficult to resolve the edge features belonging to each object if the objects are unfamiliar to the vision system.

More interesting possibilities arise when it is assumed that the objects to be tracked are familiar to the vision system. In this case it would be desirable for the vision system to identify and locate the objects in its working environment before proceeding to track their movements. Scene analysis for identifying objects, however, is a non-trivial problem and may be quite time consuming. Much work has been done in the area of scene analysis for understanding (see [BAR.78], [OHL.75], [PER.78] and [MAR.75]). The capabilities of a successful vision system for scene analysis must include an understanding of viewing

perspective, the ability to compute 2-D projections of features from 3-D representations, an understanding of the relationship between size of the object and viewing distance, the ability to handle variations in illumination (highlights and shadows) in a scene, and the occurrence of occlusions. Sufficient time must be allowed the vision system to identify the objects in its environment before proceeding to track their movements. If the vision system is working in a constrained environment (in an industrial setting, for example, where machine parts are assembled), the time taken for the initial phase of object identification can be reduced by providing a model of the working environment to the vision system.

In order to facilitate object tracking, 3-D models of common objects to be tracked may be stored in computer memory. This 3-D model may be in the form of polyhedral planes, generalized cones [NEV.74], or a wire frame representation of the object's surface (see [BAU.74] for some ways of modeling 3-D objects). In this work, the wire representation is appropriate, since the features I propose to extract are intensity edges. The reason for choosing the wire representation will become clear shortly.

When the object to be tracked has been identified,

its edge features extracted from the image are matched with the edge projections from its 3-D model. The scenario for matching is as follows. From scene analysis, the approximate location and orientation of the object with respect to the camera coordinate system is known. A set of linear transformations exists that will map edge features in the xyz coordinate system of the 3-D model to the uvw coordinate system of the camera (see illustration in Fig.5.1.1). The positions of these edge projections are then matched with the positions of the edge features extracted from the image. The matching process may be iterative, and continues until as few conflicts arise as possible.

The coordinate transformations mentioned here are similar to the ones used in computer graphics to obtain perspective views of 3-D models (see Ch.12 of [NEW.73] and and Ch.10 of [DUD.73] on "Perspective Transformations ). Essentially, the transformations perform a rotation and translation of points in the model coordinate system to the camera coordinate system. It may be possible to solve for the rotation (R<ij>'s) and translation coefficients (T<ij>'s) in the transformation matrix (Fig.5.1.2) explicitly in the matching process. The coefficients in the transformation matrix completely determine the location and

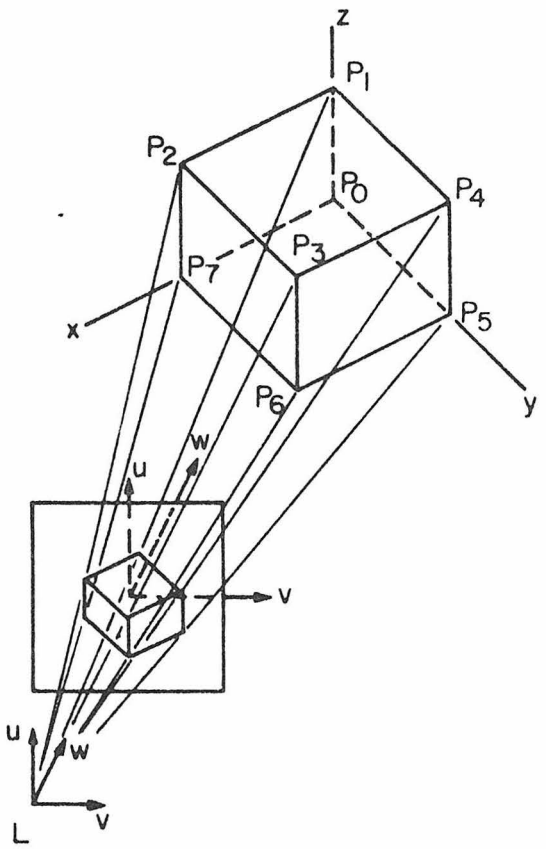



Fig.5.1.1 Obtaining edge projections in the camera coordinate system from a 3-D edge model of the object.

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} R_{ux} & R_{uy} & R_{uz} & T_{ux} \\ R_{vx} & R_{vy} & R_{vz} & T_{vy} \\ R_{wx} & R_{wy} & R_{wz} & T_{wz} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Fig.5.1.2 The transformation matrix for obtaining the edge projections from a 3-D edge model.

orientation of the moving object in the camera coordinate system, and thus may be used to specify the position of the object. The interested reader is referred to the above-mentioned references for a more detailed description of perspective coordinate transformations.

Success in matching the edge projections from an object's 3-D model with the edge features extracted from each of a sequence of images depends on whether the extracted edge features change substantially between frames. If the velocity of the object being tracked produces edge features that are displaced only a little between frames. then matching the extracted edge features with edge projections from the object's 3-D model would be highly successful. If the velocity of the object results in its extracted edge features being displaced substantially between frames, however, then matching these features with edge projections from its 3-D model might be performed incorrectly. In this event, scene analysis would have to be performed again to locate the objects in the scene. To avoid having to do this, one should ensure that the frame rate is fast enough so that the extracted edge features of moving objects in the scene are displaced incrementally between frames.

# APPENDIX A

# PERFORMING EDGE-DETECTION ON THE LOG OF AN IMAGE FUNCTION

If log digitizers are available, another way to compensate for variations of illumination in the scene is to digitize the pixels as the log of brightness (Fig.A.1). The pixel values would then be given by

$$P = \log(Ir)$$

where I is the incident illumination on the object (assumed constant over local regions of the scene), and r is the reflectance of the object (defined as ratio of reflected flux to incident flux). The edge step between 2 adjacent regions of different reflectances is given by

$$\begin{aligned} E &= P\langle 2\rangle - P\langle 1\rangle \\ &= \log(Ir\langle 2\rangle) - \log(Ir\langle 1\rangle) \\ &= \log(r\langle 2\rangle / r\langle 1\rangle) \end{aligned}$$

The edge step is dependent only on the ratio of reflectances accross the step, assuming that the illumination remains constant across the step. A single edge threshold value could then be used to separate edge pixels from the rest of the image. This method of finding edge points is potentially better than dynamic thresholding, since the

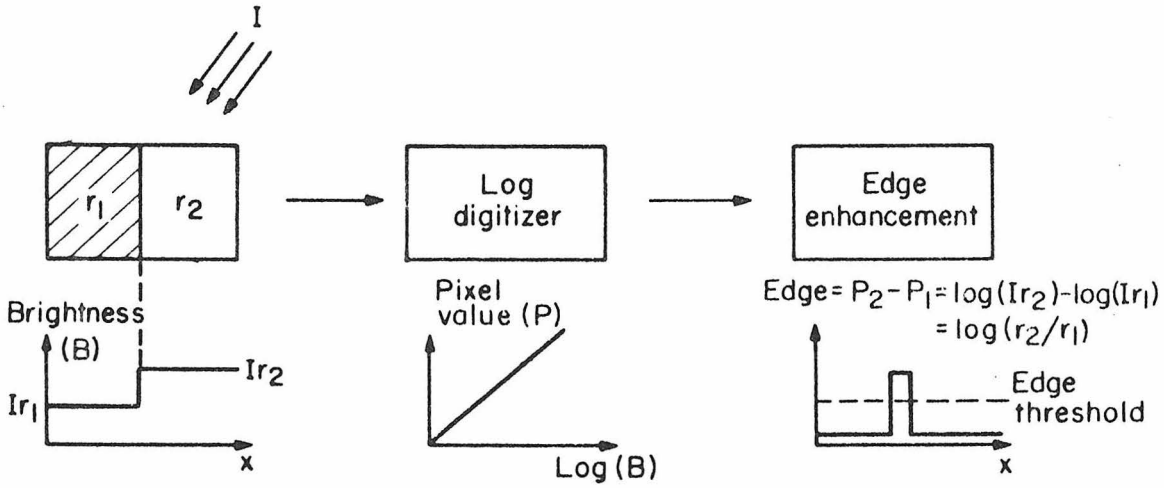



Fig.A.1 Using log digitizers for computing the edge step.

computed edge step is entirely independent of the incident illumination.

Appendix B

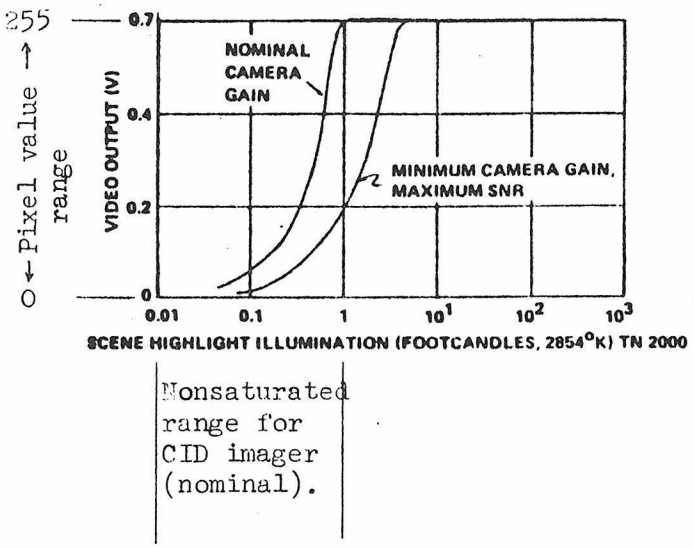



Fig.B.1 The photoresponse curve for the GE TN2000 CID imager.

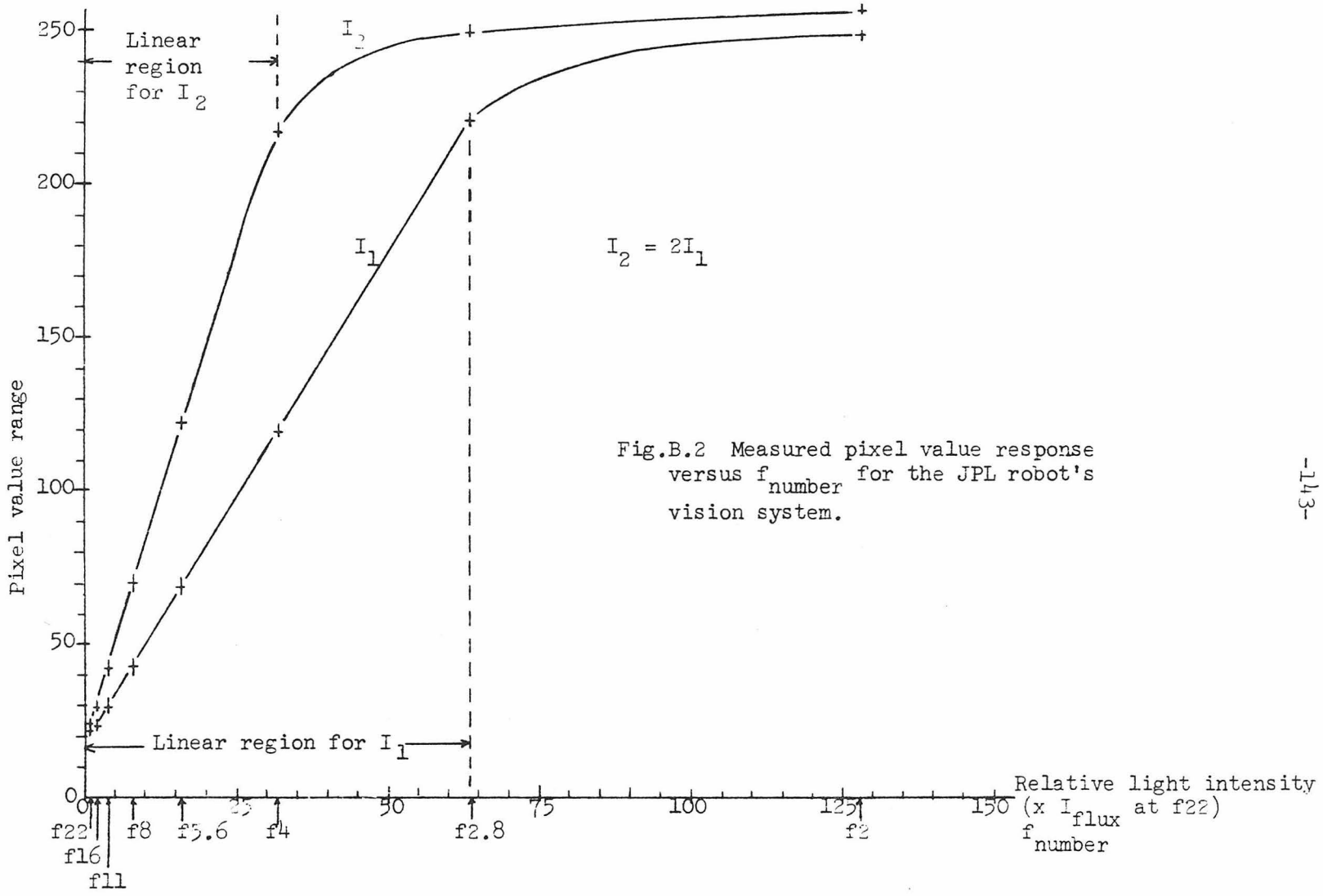



Fig.B.2 Measured pixel value response versus $f_{number}$ for the JPL robot's vision system.

# BIBLIOGRAPHY

This bibliography is arranged by year, and then by first 3 letters of author's name.

## REFERENCES ON COMPUTATIONAL STRUCTURES FOR IMAGE PROCESSING

[UNG.58] Unger S.H., "A Computer Oriented Toward Spatial Problems", Proc. of the IRE, Oct. '58, p.1744.

[MCC.63] McCormick B.H., "The Illinois Pattern Recognition Computer - ILLIAC III", IEEE Trans. Electronic Computers, Dec. 1963, p.791.

[BAR.68] Barnes G.H., et.al., "The Illiac IV Computer", IEEE Trans., c-17, v.8, Aug.68, p.746.

[GRA.71] Gray S.B., "Local Properties of Binary Images in Two Dimensions", IEEE Trans. Computers, v.c-20, no.5, May 1971.

[PRE.71] Preston K., Jr., "Feature Extraction by Golay Hexagonal Pattern Transforms", IEEE Trans. Computers, v.c-20, no.9, Sept. 1971.

[KRU.73] Kruse B., "A Parallel Picture Processing Machine", IEEE Trans. Computers, v.c-22, No.12, Dec. 1973, p.1075.

[ENS.74] Enslow P.H., "Multiprocessors and Parallel Processing", John Wiley and Sons, 1974, p.139.

[DUF.76] Duff M.J.B., "CLIP 4: A Large Scale Integrated

Circuit Array Parallel Processor", Proc. of the Int'l Conf. on Patt. Recog., Coronado, Nov. 1976.

[KRU.76] Kruse B., "The PICAP Picture Processing Laboratory", Proceedings of the International Joint Conference on Pattern Recognition, Coronado, Nov. 1976.

[NUD.77] Nudd G.R., "CCD Image Processing Circuitry", USC Semi-Annual Tech. Rep., Mar.'77, p.142.

[KRU.78] Kruse B., "Experience with a Picture Processor in Pattern Recognition Processing", Nat'l Comp. Conf. Proc., 1978, p.1015.

[MOR.78] Mori K., et.al., "Design of Local Parallel Pattern Processor for Image Processing", Nat'l Comp. Conf. Proc., 1978, p.1025.

[NUD.78] Nudd G.R. and Nygaard P.A., "Demonstration of a CCD Image Processor for 2-Dimensional Edge Detection", Electronic Letters, v.14, No.4, Feb. '78, p.83.

[POT.78] Potter J.L., "The STARAN architecture and its Application to Image Processing and Pattern Recognition Algorithms", Nat'l Comp. Conf. Proc., 1978, p.1041.

REFERENCES ON IMAGE PROCESSING TECHNIQUES, AND PSYCHOLOGY OF MACHINE AND NATURAL VISION

[GOO.68] Goodman J.W., "Introduction to Fourier Optics", McGraw-Hill, 1968.

[KOL.68] Kolers P.A. and Eden M., "Recognizing Patterns , The MIT Press, 1968.

[HEU.71] Heuckel M., "An Operator which Locates Edges in Digitized Pictures", Stanford University, AI Memo 105, Oct.69. Also J. of ACM, No.1, Jan.71, p.113.

[DUD.73] Duda R.O. and Hart P.E., "Pattern Classification and Scene Analysis", John Wiley and Sons, 1973.

[NEW.73] Newman W.M. and Sproull R.F., "Principles of Interactive Computer Graphics", McGraw-Hill, 1973.

[BAU.74] Baumgart B.G., "Geometric Modeling for Computer Vision", Stanford University, AI Memo 249,CS 463, ADA-002-261, Ph.D. in CS, Oct.74.

[BUL.74] Bullock B., "The Performance of Edge Operators on Images with Texture", Hughes Research Labs Tech. Rep., Oct.74.

[FRE.74] Freeman H., "Computer Processing of Line Drawing Images", Computing Surveys, v.6, No.1, Mar.74.

[NEV.74] Nevatia R., "Structured Descriptions of Complex Curved Objects for Recognition of Visual Memory", Stanford University, AI Memo 250, CS 464, ADA-003-486, Ph.D. in EE, Oct.74.

[AGI.75] Agin G.J. and Duda R.O., "SRI Vision Research for Advanced Industrial Automation", Proc. 2nd USA-Japan Computer Conf., 1975.

[MAR.75] Marr D., "Analyzing Natural Images: a Computational Theory of Texture Vision", MIT AIM 334, AD-A013-257, June '75.

[OHL.75] Ohlander R.B., "Analysis of Natural Scenes", Ph.D. thesis, Carnegie-Mellon University, April '75.

[SHI.75] Shirai Y., "Edge Finding, Segmentation of Edges and Recognition of Complex Objects", Proc. of 4th Int'l Joint Conf. on Artificial Intelligence, Sept. '75, p.674.

[WIN.75] Winston P.H., "The Psychology of Computer Vision", McGraw-Hill, 1975.

[ROS.76] Rosenfeld A. and Kak A.C., "Digital Picture Processing", Academic Press, 1976.

[AND.77] Andrews H.C. and Hunt B.R., "Digital Image Restoration", Prentice-Hall, 1977.

[BAR.78] Barrow H.G. and Tenenbaum J.M., "Recovering Intrinsic Scene Characteristics from Images", SRI TN 157, April '78.

[PER.78] Perkins W.A., "A Model Based Vision System for Industrial Parts", IEEE Trans. Comp., v.c-27, Feb.78, p.126.

REFERENCES ON MICRO-ELECTRONIC TECHNOLOGY

[HON.72] Honeisen B. and Mead C.A., Fundamental Limitations in Micro-electronics - I. MOS Technology', Solid State Elect., 1972, v.15, p.819.

[SUT.75] Sutherland I.E., "The Manchester Adder", TM 75.4

[SUT.76] Sutherland I.E., Mead C.A. and Everhart T.E., "Basic Limitations in Microcircuit Fabrication Technology", Rand report, R-1956-ARPA, Nov.76.

[KEY.77] Keyes R.W., "Physical Limits in Semiconductor Electronics", Science, Mar. 18, 1977, v.195, No.4283, p.1230.

[NOY.77] Noyce R.N., "Large-Scale Integration: What is Yet to Come?", Science, Mar. 18, 1977, v.195, No.4283, p.1102.