

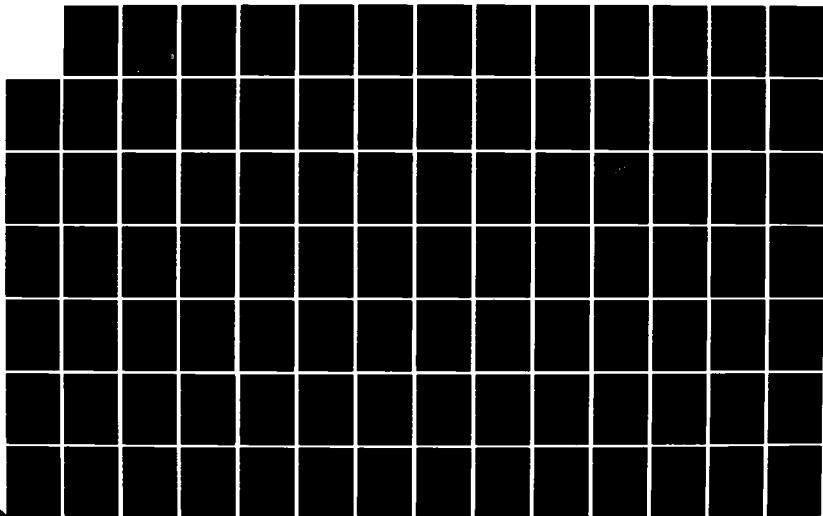
AD-A152 008

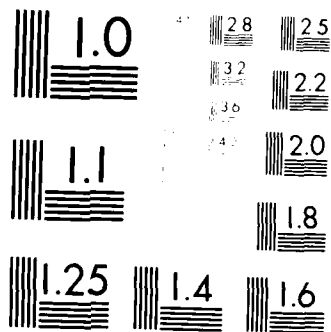
AN EXTENSION OF A MICROCOMPUTER BASED SYSTEM FOR
ANALYSIS OF LINE DRAWING. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI... T A MORRIS
DEC 84 AFIT/GE/ENG/84D-48 F/G 9/2

1/3

UNCLASSIFIED

NL





MICROCOPY RESOLUTION TEST CHART
 NATIONAL BUREAU OF STANDARDS-1963-A

DTIC 101

AD-A152 008



AN EXTENSION OF A MICROCOMPUTER BASED
SYSTEM FOR ANALYSIS OF LINE DRAWING
QUANTIZATION SCHEMES

THESIS

Thomas A. Morris
Captain, USAF

AFIT/GE/ENG/84D-48

DTIC FILE COPY

DTIC
ELECTE
MAR 28 1985
S D E

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

This document has been approved
for public release and sale by the
AFIT/GE/ENG/84D-48

85 03 13 079

AFIT/GE/ENG/84D-48

AN EXTENSION OF A MICROCOMPUTER BASED
SYSTEM FOR ANALYSIS OF LINE DRAWING
QUANTIZATION SCHEMES

THESIS

Thomas A. Morris
Captain, USAF

AFIT/GE/ENG/84D-48

Approved for public release; distribution unlimited

AFIT/GE/ENG/84D-48

AN EXTENSION OF A MICROCOMPUTER BASED SYSTEM FOR
ANALYSIS OF LINE DRAWING QUANTIZATION SCHEMES

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Thomas A. Morris, B.S.

Captain, USAF

December 1984



Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
Pvt.	
Distribution/	
Avail. and/or	
Special	
Dist	
A-1	

Acknowledgments

I would like to thank my advisor, Major Ken Castor of the Air Force Institute of Technology, for proposing this thesis topic. I am grateful for his assistance throughout the project. I would also like to thank my family for their constant support and encouragement.

Table of Contents

	Page
Acknowledgments	ii
List of Figures	v
List of Tables	viii
Abstract	x
I. Introduction	I-1
II. Background of Generalized Chain Codes . . .	II-1
The Single Ring Chain Code	-2
Extensions of the Single Ring Code . . .	-4
Higher Order Codes	-5
Quantization Procedure	-6
Generalized Chain Code Performance	
Measures	-9
Summary	-11
III. Overview of Existing System	III-1
Existing Hardware	-1
Hardware Modifications	-2
Existing Software	-2
Disk File Format	-3
Assembly Language Interface Routines .	-4
Pascal Routines	-6
Software Modifications	-11
Summary	-20
IV. Implementation of Multi-ring Chain Coding	
Algorithm	IV-1
Overall Operation of CHNCODE Program .	-2
Subroutines	-5
Testing of CHNCODE	-9
PLOTCODE	-10
Effect of Digitizer Resolution on	
CHNCODE and PLOTCODE	-12
Summary	-14
V. Analysis of Various Line Drawings	V-1
Circle	-2

Squares	-10
Sine Wave	-26
Written Text	-30
Summary	-34
VI. Conclusions and Recommendations	VI-1
Conclusions	-1
Recommendations for Future Study	-5
Appendix A: Program Listings	A-1
Appendix B: Line Drawing Data and Figures	B-1
Appendix C: User's Manual	C-1
Bibliography	BIB-1

List of Figures

Figure		Page
II-1	Octal Encoding Assignment for Single Ring Code	II-2
II-2	Example of Single Ring Code	II-3
II-3	Nodes of Level 2 Single Ring Code . . .	II-4
II-4	Nodes of a (1,3) Chain Code	II-5
II-5	Triangular Quantization Scheme Templates	II-8
II-6	Example of a Ring 3 LGS for a Line Drawing	II-9
III-1	Overlay of Coded Line onto Input Line .	III-10
IV-1	Nodes for a (1,2) Chain Code	IV-3
IV-2	Link Gate Sets for a (1,4) Code	IV-14
B-1	Digitized Circle Drawing	B-32
B-2	CIRCLE: (1,2,3,4) Code With 0.25 Inch Gridsize	B-33
B-3	CIRCLE: (1,3) and (1,4) Codes With 0.25 Inch Gridsize	B-34
B-4	CIRCLE: (1) Code With 0.25 Inch Gridsize	B-35
B-5	CIRCLE: (2,4) Code With 0.25 Inch Gridsize	B-36
B-6	CIRCLE: (1), (1,2), (1,3) and (1,4) Codes With 0.2 Inch Gridsize	B-37
B-7	CIRCLE: (1) Code With 0.15 Inch Gridsize	B-38
B-8	CIRCLE: (1,4) and (3,4) Codes With 0.1 Inch Gridsize	B-39
B-9	CIRCLE: (4) Code With 0.05 Inch Gridsize	B-40

B-10	Digitized Square Drawings	B-41
B-11	Example of Error for Non-rotated Square	B-42
B-12	SQUARE-30: (1) Code With 0.25 Inch Gridsize	B-43
B-13	SQUARE-30: (3,4) Code With 0.25 Inch Gridsize	B-43
B-14	SQUARE-60: (3,4) Code With 0.25 Inch Gridsize	B-44
B-15	SQUARE-60: (1), (1,3), and (1,4) Codes With 0.25 Inch Gridsize	B-45
B-16	SQUARE-30: (2,3), (2,4), (2,3,4), and (1) Codes With 0.2 Inch Gridsize	B-46
B-17	SQUARE-45: (1) Code With 0.2 Inch Gridsize	B-47
B-18	SQUARE-60: (1) Code With 0.2 Inch Gridsize	B-48
B-19	SQUARE-30: (2,4) and (3,4) Codes With 0.15 Inch Gridsize	B-49
B-20	SQUARE-60: (3,4) and (1) Codes With 0.15 Inch Gridsize	B-50
B-21	SQUARE-0: (4) Code With 0.1 Inch Gridsize	B-51
B-22	SQUARE-30: (2,4), (3,4), and (1,3) Codes With 0.1 Inch Gridsize	B-52
B-23	SQUARE-0: (1), (1,2), and (3) Codes With 0.05 Inch Gridsize	B-53
B-24	SQUARE-30: (1) Code With 0.05 Inch Gridsize	B-54
B-25	SQUARE-45: (1,2), (2,3), and (2,3,4) Codes With 0.05 Inch Gridsize	B-55
B-26	SQUARE-60: (1,2), (2,4), and (3,4) Codes With 0.05 Inch Gridsize	B-56
B-27	SQUARE-60: (1) and (2,3,4) Codes With 0.05 Inch Gridsize	B-57
B-28	Digitized Sine Wave Drawing	B-58

B-29	SINE WAVE: (1,2,3,4) Code With 0.25 Inch Gridsize	B-59
B-30	SINE WAVE: (1) Code With 0.2 Inch Gridsize	B-59
B-31	SINE WAVE: (1,4) Code With 0.15 Inch Gridsize	B-60
B-32	SINE WAVE: (1) Code With 0.1 Inch Gridsize	B-60
B-33	SINE WAVE: (3,4) Code With 0.1 Inch Gridsize	B-61
B-34	SINE WAVE: (1) Code With 0.05 Inch Gridsize	B-61
B-35	SINE WAVE: (2,3,4) Code With 0.05 Inch Gridsize	B-62
B-36	Digitized Written Text Drawing	B-63
B-37	TEXT: (1,3) Code With 0.25 Inch Gridsize	B-64
B-38	TEXT: (1,3) and (1,2,3,4) Codes With 0.2 Inch Gridsize	B-64
B-39	TEXT: (1) and (1,2,3,4) Codes With 0.15 Inch Gridsize	B-65
B-40	TEXT: (1,2,4) Code With 0.1 Inch Gridsize	B-65
B-41	TEXT: (1), (4), (2,4), and (2,3,4) Codes With 0.05 Inch Gridsize	B-66

List of Tables

Table		Page
B-1	Circle Coded With 0.25 Inch Gridsize . . .	B-2
B-2	Circle Coded With 0.2 Inch Gridsize . . .	B-3
B-3	Circle Coded With 0.15 Inch Gridsize . . .	B-4
B-4	Circle Coded With 0.1 Inch Gridsize . . .	B-5
B-5	Circle Coded With 0.05 Inch Gridsize . . .	B-6
B-6	SQUARE-0 Coded With 0.25 Inch Gridsize . .	B-7
B-7	SQUARE-0 Coded With 0.2 Inch Gridsize . . .	B-8
B-8	SQUARE-0 Coded With 0.15 Inch Gridsize . .	B-9
B-9	SQUARE-0 Coded With 0.1 Inch Gridsize . . .	B-10
B-10	SQUARE-0 Coded With 0.05 Inch Gridsize . .	B-11
B-11	SQUARE-30 Coded With 0.25 Inch Gridsize . .	B-12
B-12	SQUARE-30 Coded With 0.2 Inch Gridsize . .	B-13
B-13	SQUARE-30 Coded With 0.15 Inch Gridsize . .	B-14
B-14	SQUARE-30 Coded With 0.1 Inch Gridsize . .	B-15
B-15	SQUARE-30 Coded With 0.05 Inch Gridsize . .	B-16
B-16	SQUARE-45 Coded With 0.25 Inch Gridsize . .	B-17
B-17	SQUARE-45 Coded With 0.2 Inch Gridsize . .	B-18
B-18	SQUARE-45 Coded With 0.15 Inch Gridsize . .	B-19
B-19	SQUARE-45 Coded With 0.1 Inch Gridsize . .	B-20
B-20	SQUARE-45 Coded With 0.05 Inch Gridsize . .	B-21
B-21	SQUARE-60 Coded With 0.25 Inch Gridsize . .	B-22
B-22	SQUARE-60 Coded With 0.2 Inch Gridsize . .	B-23
B-23	SQUARE-60 Coded With 0.15 Inch Gridsize . .	B-24

B-24	SQUARE-60 Coded With 0.1 Inch Gridsize . .	B-25
B-25	SQUARE-60 Coded With 0.05 Inch Gridsize . .	B-26
B-26	Sine Wave Coded With 0.25 Inch Gridsize . .	B-27
B-27	Sine Wave Coded With 0.2 Inch Gridsize . .	B-28
B-28	Sine Wave Coded With 0.15 Inch Gridsize . .	B-29
B-29	Sine Wave Coded With 0.1 Inch Gridsize . .	B-30
B-30	Sine Wave Coded With 0.05 Inch Gridsize . .	B-31

Abstract

This paper describes the extension of a micro-computer based system to analyze the performance of various grid based line drawing quantization schemes. The system is developed on a Heathkit H-89 microcomputer with a Hewlett-Packard 9874A digitizer and a Hiplot digital plotter. The capabilities of the existing system were expanded to allow real time digitizing/plotting operations and to provide for encoding and analysis using variable ring quantization schemes. Comparisons of specific drawings were then made for various quantization schemes based on a distortion metric (area between original and quantized image), a rate metric (number of bits in the quantization), and a subjective evaluation of the smoothness of the quantized image.

Results indicate that multi-ring codes usually produce less distortion in the quantized image than single ring codes while the single ring codes require less bits per arc-segment. Also, neither metric is a good indicator of the smoothness of the quantized image.

to analyze the line drawing. (Refer to references 7 and 8 for discussions of the types of analysis that can be performed on line drawings using the chain codes.) For applications where extensive analysis is to be performed, such as pattern recognition, this criteria can take precedence over others, such as the ease of encoding and decoding.

The performance measures used for this thesis are precision, compactness, and smoothness. The precision measurement will be an area error per unit length; this is defined as the total area enclosed between the coded representation of the line and original line drawing divided by the total length of the original line drawing, area error/unit length. The compactness measurement will be the number of bits required for encoding divided by the total length of the original line drawing, bits/unit length. The smoothness measure will be a visual examination of the coded line drawing using the digital plotter (this will be a very subjective measurement). A more detailed explanation of how these measures are implemented will be presented in the next chapter of this thesis, which is a description of an existing microcomputer based line drawing system.

CHAPTER

In this chapter, the single ring chain code was introduced and discussed. This led logically to a discussion of the generalized higher order and multi-ring

size used during the quantization process. Thus, using a small grid size in relation to the smallest radius of curvature and low order codes results in high precision. This comes, however, at the loss of compactness.

Compactness is the amount of storage space required to store the line drawing or amount of bandwidth required to transmit the data (4:11-15). This is of particular importance in drawings where large quantities of line drawing data are involved, such as with a geographic map (1:7). The tradeoff between compactness and precision can now easily be seen. For a given code level, the more precise the drawing the more nodes that are required, resulting in a loss of compactness. Also, to achieve compactness, precision must be sacrificed.

Smoothness is important only where the coded line drawing is to be visually displayed (3:9). In general, smoothness will depend heavily on the coding levels used in the quantization process, with the better angular resolution and fewer number of nodes of the higher order codes giving them the advantage. Obviously, there is a tradeoff with precision, which requires a large number of nodes and low order codes.

The ease of encoding and decoding depends on the coding algorithm used. It becomes important if large quantities of data are involved and if the coding needs to be done in real time. The ease of processing the data refers to the simplicity and speed of the algorithms used

An example of this process is shown in Figure II-6. Note that in Figure II-6a, the curve does not remain within the LGS and the lower order ring must be used; while in Figure II-6b, the higher order ring is used.

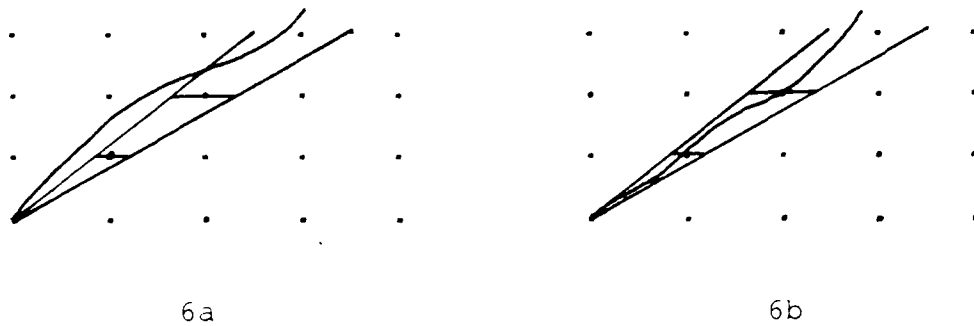
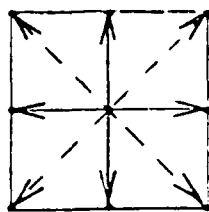


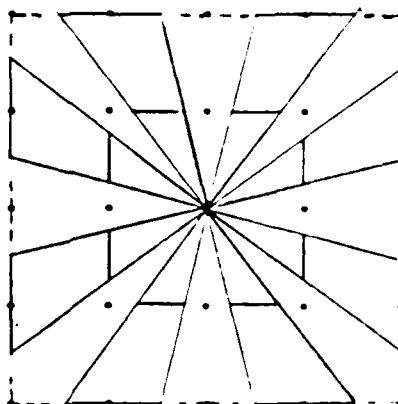
Figure II-6. Example of a Ring 3 LGS for a Line Drawing
Generalized Chain Code Performance Measures

There are five major criteria for the performance of the chain codes for line drawing data: These are: (1) precision, (2) compactness, (3) smoothness, (4) simplicity of encoding and decoding, (5) and facility for processing (1:7). Obviously, these criteria are very subjective and the importance of each depends heavily on the intended application of the drawing. Each of these criteria are explained below and the criteria used for this thesis is presented.

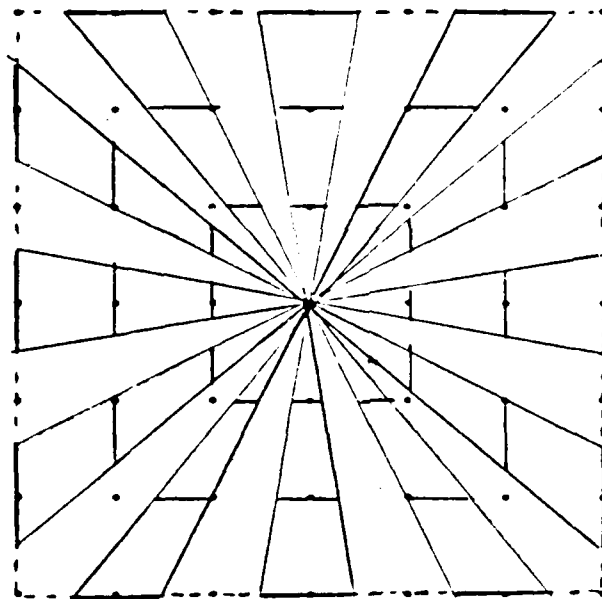
Precision, or accuracy, is a measure of how well the coding scheme correctly quantizes small changes of a function (6:II-9). This is extremely important if the data is used for quantitative analysis of the line drawing. The precision is heavily dependent upon the grid



(a)



(b)



(c)

Figure II-5. Triangular Quantization Scheme Templates
 (a) Ring 1, (b) Ring 2, (c) Ring 3

6. If chain does not contain ring i , repeat step 5, else go to step 2.

The set of all LGS of a ring is called a "template". Templates for rings 1, 2, and 3 are shown in Figure II-5 (1:1). Note that for ring 1 the generalized quantization scheme reduces to the one discussed for the single ring chain code.

The quantization procedure simply consists of a search for the highest order ring link for which all LGS intersect the curve. This procedure is as follows (1:1):

1. Set $i=k$ where k is the order of the highest order ring in the code.
2. Position the template i so that its center lies on the last encoded node and its sides lie parallel to the grid.
3. Find the intersection points of the curve with rings $i, i-1, i-2, \dots, 1$.
4. If the chain does not contain any ring lower than ring i , delete from the above set all but the intersection points which lie on ring i .
5. If any LGS of ring i contains all the intersection points found in step 3 then the associated link is selected and we return to step 1. Else set $i=i-1$ and go to step 3.

Quantization Procedure

The quantization procedure is more complex for the generalized higher order codes than for the single ring codes. There are various quantization techniques used for the generalized codes (5:13), each with different properties. The one described here and used for this thesis is the triangular quantizing scheme. Two definitions will aid in the explanation of the quantizing scheme. First, a "link" is defined as a vector from the center of a ring to any node on the ring. Now, given a particular chain code, "sets of link gates" can be defined which provide a means of selecting the links that best approximate the curve to be encoded. The "link gate sets" (LGS) are constructed as follows:

1. Set $i=k$, where k is the order of the largest ring in the selected chain code.
2. Find the midpoints of all pairs of adjacent nodes on this ring.
3. Connect the midpoints to the center of the ring with straight lines called "midpoint lines".
4. The parallel line segments cut out of each ring, 1 through k , by a pair of adjacent midpoint lines, form the (LGS) for the link of ring k lying between the two midpoint lines.
5. Set $i=i-1$. If $i=0$, stop.

times the distance from the current node as it is in the level 1 code, there are fewer nodes to quantize for any given line drawing. This decreases the amount of storage space required. The disadvantage is the loss of ability to accurately follow curves with a radius of curvature which is small relative to the size of the ring.

Higher Order Codes

Generalized higher order chain codes are formed by using a combination of two or more single ring codes of different levels. The advantage of this algorithm is that where the radius of curvature of the line drawing is small, the lower level ring is used; and where the curvature is large, the higher level ring is used. In this way, the accuracy of the drawing is maintained while the number of nodes for a nearly straight portion of the drawing is reduced (2:IV-6). Chain codes of this type are identified by the rings used in the code. For example, a chain code using rings 1 and 3 would be called a (1,3) code. A (1,3) code is shown in Figure II-4.

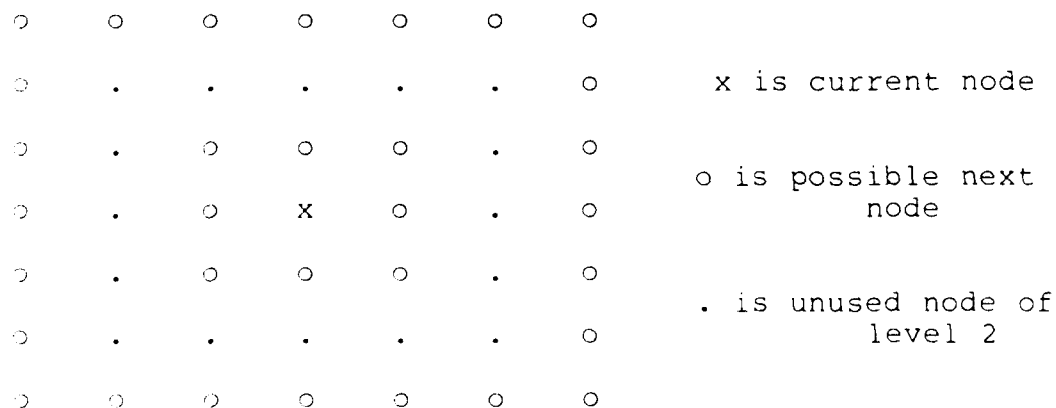


Figure II-4. Nodes of a (1,3) Chain Code

to become 7012. Now, the drawing moves to the left, crossing the grid near node 4, causing the chain code to become 70124. Finally, the drawing moves to the left corner, crossing the grid nearest node 3, and then ends. Therefore, the final chain code representation is 701243 octal.

Extensions of the Single Ring Code

The single ring coding algorithm is easily extended to levels greater than one. A single ring code of level 2 has 16 possible next nodes and is shown in Figure II-3.

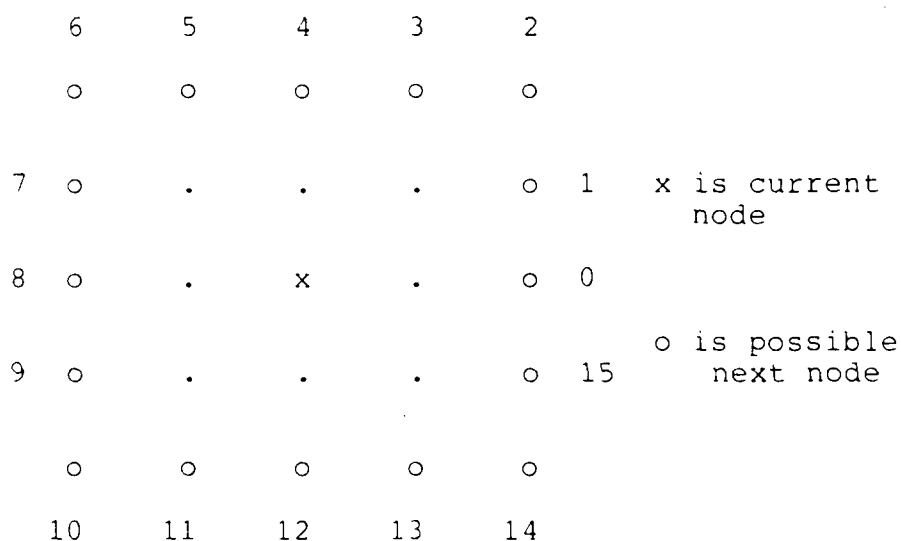


Figure II-3. Nodes of Level 2 Single Ring Code

This idea can be extended to a single ring code of level n . An n -level code would have $8n$ possible next nodes. An obvious advantage to increasing the level of the code is improved angular resolution. Also, since the ring is n

and added to the chain. A binary encoding procedure is to associate a three bit binary number with each node of the ring. For example, node 0 would be encoded as 000, node 3 as 011, etc. An octal representation of this binary encoding is shown in Figure II-1. The assignment of a particular integer to each node is arbitrary, but must be consistent for both encoding and decoding (4:II-7). An example of the encoding process is illustrated in Figure II-2.

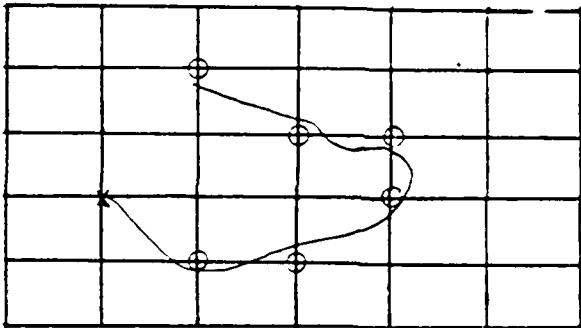
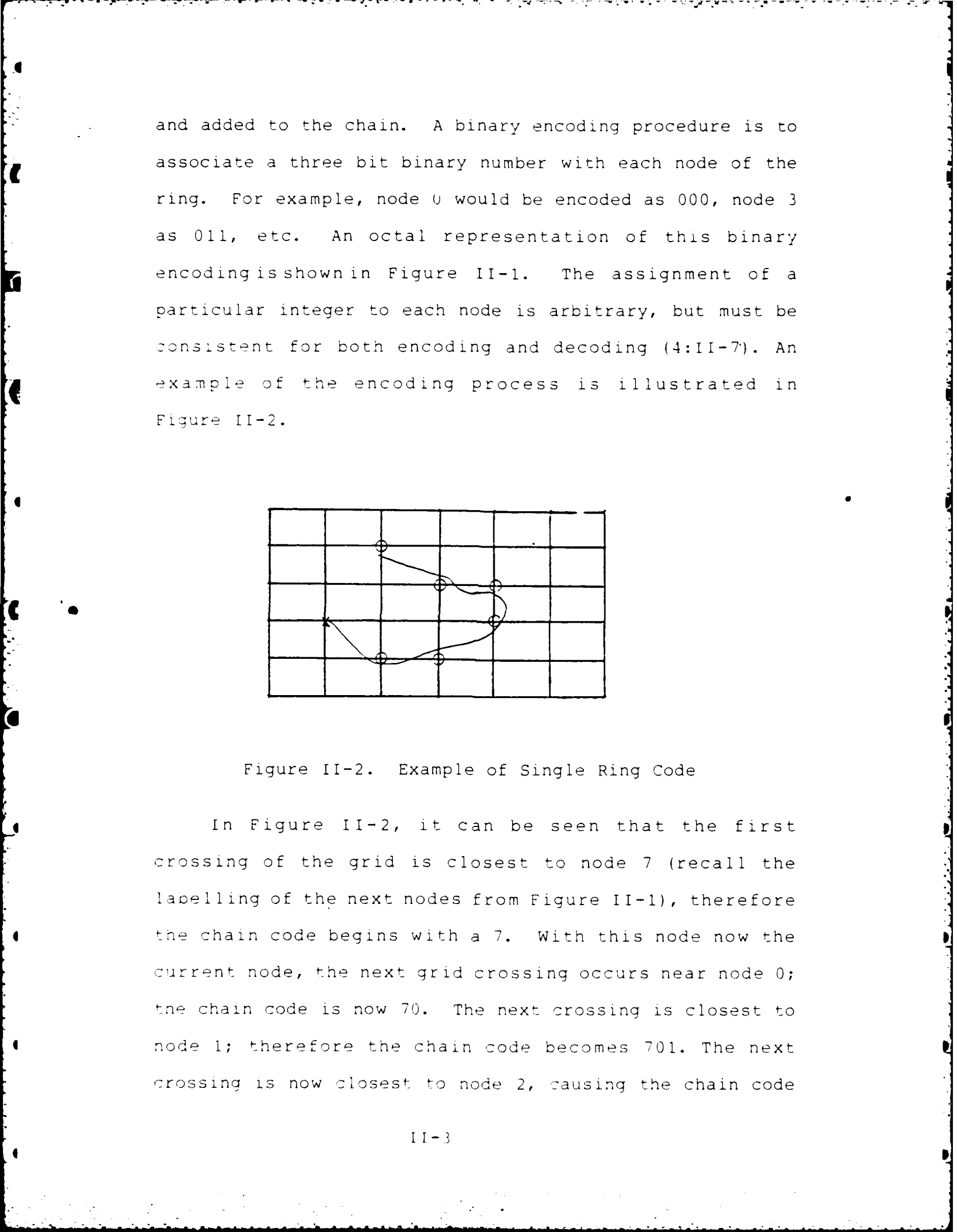


Figure II-2. Example of Single Ring Code

In Figure II-2, it can be seen that the first crossing of the grid is closest to node 7 (recall the labelling of the next nodes from Figure II-1), therefore the chain code begins with a 7. With this node now the current node, the next grid crossing occurs near node 0; the chain code is now 70. The next crossing is closest to node 1; therefore the chain code becomes 701. The next crossing is now closest to node 2, causing the chain code

II-3



and added to the chain. A binary encoding procedure is to associate a three bit binary number with each node of the ring. For example, node 0 would be encoded as 000, node 3 as 011, etc. An octal representation of this binary encoding is shown in Figure II-1. The assignment of a particular integer to each node is arbitrary, but must be consistent for both encoding and decoding (4:II-7). An example of the encoding process is illustrated in Figure II-2.

Figure II-2. Example of Single Ring Code

In Figure II-2, it can be seen that the first crossing of the grid is closest to node 7 (recall the labelling of the next nodes from Figure II-1), therefore the chain code begins with a 7. With this node now the current node, the next grid crossing occurs near node 0; the chain code is now 70. The next crossing is closest to node 1; therefore the chain code becomes 701. The next crossing is now closest to node 2, causing the chain code

II-3

and added to the chain. A binary encoding procedure is to associate a three bit binary number with each node of the ring. For example, node 0 would be encoded as 000, node 3 as 011, etc. An octal representation of this binary encoding is shown in Figure II-1. The assignment of a particular integer to each node is arbitrary, but must be consistent for both encoding and decoding (4:II-7). An example of the encoding process is illustrated in Figure II-2.

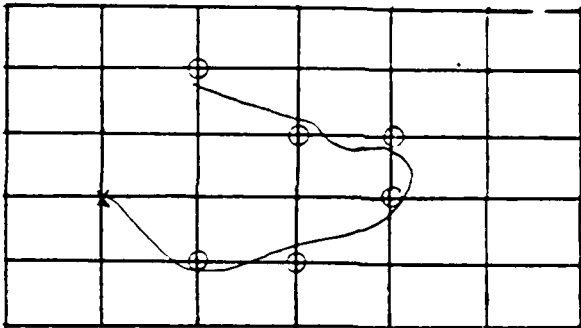


Figure II-2. Example of Single Ring Code

In Figure II-2, it can be seen that the first crossing of the grid is closest to node 7 (recall the labelling of the next nodes from Figure II-1), therefore the chain code begins with a 7. With this node now the current node, the next grid crossing occurs near node 0; the chain code is now 70. The next crossing is closest to node 1; therefore the chain code becomes 701. The next crossing is now closest to node 2, causing the chain code

II-3

The Single Ring Chain Code

One of the simplest chain codes is the single ring chain code. The encoding and quantizing algorithm is best described by visualizing a grid overlaying the line drawing. For any chain code, a starting point must be given or assumed, and this starting point must be a node on the grid. Figure II-1 demonstrates this concept. With the starting point of the line drawing located at point x, the grid overlays the drawing and surrounds the starting point (referred to in Figure II-1 as the current node) with eight nodes. No matter which direction the line drawing takes, it must cross the grid near one of these nodes; this node is encoded and stored as the beginning of the chain, and becomes the new current node. This process repeats until the end of the drawing is reached.

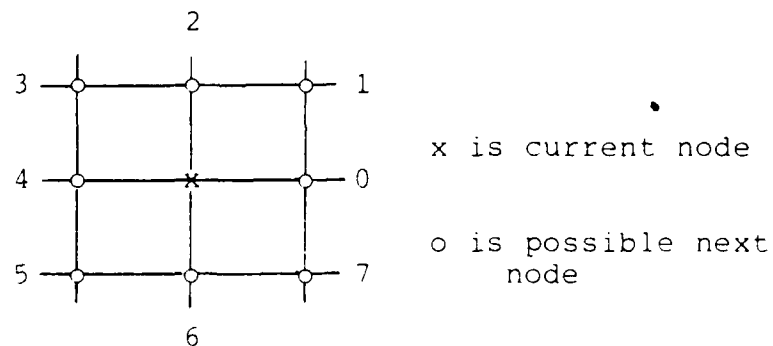


Figure II-1. Octal Encoding Assignment for Single Ring Code

Each time the drawing crosses the grid, the node nearest the crossing then becomes the current node and is encoded

II. Background of Generalized Chain Codes

The generalized chain codes are a family of methods for quantizing and encoding line drawing images. The quantization is done by superimposing a grid of some specified gridsize onto the line drawing and selecting a set of nodes to represent the drawing. A node is defined as the intersection of the horizontal and vertical grid lines. The quantization of the line drawing is accomplished by selecting the node that is nearest to the intersection of the line drawing and the grid lines; not allowing a single node to be selected twice in succession. A line drawing is then described by a sequence of nodes connected by straight lines with the first node located at the first point of the line drawing. For encoding, a node is only identified relative to the node which immediately preceded it in the sequence, hence the name chain code (2:IV-1). The nodes are encoded by any method in which each node is assigned a binary number which represents its relative position to the previously selected node (3:3). In this chapter, the quantizing and encoding processes are explained more fully. The chain codes themselves are explained starting with the simple single ring codes and continuing with the more complex higher order generalized chain codes. Also, the performance evaluation criteria for the chain codes are discussed.

the analysis of several specific line drawings, and Chapter VI details the conclusions drawn from that analysis and proposes areas for future study.

developed to efficiently encode line drawings using a set of schemes known as generalized chain codes (1).

A microcomputer based system for the analysis of line drawing quantizations schemes was developed at AFIT by Lt Joseph E. Rock in 1983 (2). This system utilizes disk files for intermediate storage of the image data, and implements a small subset of the generalized chain codes.

The objective of this thesis was to expand the system developed by Lt Rock to allow real time digitizing, encoding, and plotting operations; and, to provide for encoding line drawings using variable ring chain code quantization schemes. These new capabilities were used to provide quantitative comparisons of quantizations of specific drawings. Comparisons between chain codes were made on the basis of a rate (number of bits in the quantization), a distortion (area between original and quantized image), and a subjective evaluation of the smoothness of the quantized image.

The organization of this thesis parallels the approach to the problem. Chapter II discusses the background and development of generalized chain codes. In Chapter III, the existing system developed by Lt Rock is described as well as the modifications that were made to the system for this thesis. Chapter IV describes the software that implements the variable ring chain coding algorithm and the software written to allow real time operation. Chapter V is a discussion of the results of

AN EXTENSION OF A MICROCOMPUTER BASED SYSTEM FOR ANALYSIS OF LINE DRAWING QUANTIZATION SCHEMES

I. Introduction

In many Air Force and engineering applications, the ability to store two-dimensional image data in a digital format is very important. The classical method of processing this data is two-dimensional sampling. This technique is computationally intensive and requires a large amount of computer memory and storage media. For some types of images, such as photographs, these problems are tolerated because of the need to completely reproduce the original image from the stored version. However, for other types of images the two-dimensional sampling method is not required. An example of such an image is the line drawing. A line drawing can be defined as an image consisting entirely of thin lines on a contrasting background. Examples of line drawings include maps, printed or written text, graphs, engineering drawings, and temperature charts. Since a line drawing can be seen to be a very limited type of two-dimensional image, it seems logical that there would be a way to sample and store a line drawing much more efficiently than more complex images, such as photographs. Indeed, methods have been

chain codes. The major performance measures for the chain codes and their various tradeoffs were discussed as well as the performance measures used for this thesis effort. In the next chapter, the existing microcomputer system developed by Lt Joseph E. Rock (2) will be described.

III. Overview of Existing System

This chapter presents a brief overview of an existing microcomputer based system for analysis of line drawing quantization techniques. Also, the modifications to the system accomplished for this thesis are discussed. This system was developed by 1st Lt Joseph E. Rock, Jr. at the Air Force Institute of Technology for his master's thesis (2). This overview consists of a description of the system hardware and of the software algorithms developed by Lt Rock.

Existing Hardware

The system as developed by Lt Rock utilizes the hardware listed below:

1. Heathkit H-89 microcomputer with 64 kilobytes of RAM, 3 serial RS-232 I/O ports, one 90 kilobyte 5 inch disk drive, two 594 kilobyte 8 inch disk drives, and CP/M disk operating system.
2. Heathkit H-25 printer.
3. Hewlett-Packard model 9874A digitizer and an ICS Electronics Corporation model 4885A IEEE-488 to RS-232 bus controller.
4. Houston Instruments model DMP7 x-y plotter with a RS-232 serial interface.

Hardware Modifications

The only major hardware modification accomplished for this thesis increases the computer clock rate from 2.048 MHz to 4.096 Mhz. This increased speed is necessary to implement real time operation of the digitizer and coding algorithms.

Existing Software

The software developed by Lt Rock allows the user to trace a line drawing on the digitizer, store this digitized version on a disk file, and use it as if it were the actual line drawing. A single ring chain coding algorithm is then employed using the stored digitized version of the line drawing; this coded version is also stored as a disk file. A performance measurement routine is available that provides a measurement of the precision of the coded line drawing. This measurement is the area error divided by the unit length; this is defined as the total area enclosed between the coded representation of the line and the original line drawing divided by the length of the original line. The routine also provides the length of both the digitized and the coded versions of the line. The plotter routines allow the user to plot both the digitized and coded versions of the line drawing, allowing a visual inspection.

The overview of the algorithms developed by Lt Rock will consist of a discussion of the disk file format and of the input and output parameters and the functions of

each program module. First, the disk file format is discussed. Next, the assembly language routines to interface the digitizer and plotter to the computer are discussed. Finally, the higher level digitizing, coding, and performance measurement routines are discussed; these routines are written in the Pascal language.

Disk File Format

The disk file format created by Lt Rock was designed so that only one plotting program would be needed to serve both the digitized and coded files and with ease of processing in the performance measurement routine in mind. For both types of files, digitized and coded, each point in the drawing is described by a line in the disk file. The format of the line for both types of files is standard for the first three items. These three items are the pen up/down indicator (either a 'U' or a 'D'), the x coordinate value, and the y coordinate value (x and y values are integers from 1 to 32,765 where each unit represents 0.001 inches). Having these three items first in each line of the file made compatibility with the file plotting program and the performance analysis algorithm a simple matter; both these files need read only the first three items from each line to gain all the pertinent information they need to perform their function.

After the first three items of a line, the format for the two types of disk files differ. For the digitized file, each line of the file also contains the pen position

indicator of the digitizer and the annotation number entered from the digitizer keypad. For a file of points generated by the coding program, the first line of each line segment contains the gridsize and ring level of the chain code along with a code value of -1 (Lt Rock used -1 to indicate the first and last points of a line segment). Every other line in the coded file contains as a fourth element the value of the chain code for that point. For both types of files, there must be a space between line elements and each line must be terminated by a carriage return.

Assembly Language Interface Routines

The assembly language interface routines allow the user to control the interfaces between the digitizer, plotter and the computer from a Pascal program. Each of these routines is discussed in the following paragraphs.

The BUSINT routine initializes the serial port to establish communications with the IEEE-488 controller and then initializes the controller and the digitizer. It is called as an external function from the higher order program as "BUSINT:CHAR" and returns a value of 'E' if an error was encountered during execution or a value of 'O' otherwise.

The BUSIN routine is used to input a line from the digitizer to the computer via the IEEE-488 controller. It is called as an external procedure by the Pascal program as "BUSIN (DEVICE:INTEGER; VAR ERFLAG:CHAR; VAR

LINE:ARRAY[1,40] OF CHAR)". DEVICE is the device number (06 for the digitizer), ERFLAG returns an 'E' if a problem was encountered or a 'O' otherwise, and LINE is the array of characters from the digitizer (the last character will be a carriage return, CR).

The BUSOUT routine is used to output a line of characters from the computer to the digitizer via the IEEE-488 bus controller. It is called as an external procedure by the Pascal program as "BUSOUT (DEVICE:INTEGER; VAR ERFLAG:CHAR; VAR LINE:ARRAY[1,40] OF CHAR)". The labels used for this routine have the same meaning for BUSOUT as they did for BUSIN in the previous paragraph.

The PORTIN routine is used to initialize the RS-232 serial port connected to the plotter. The routine is called as an external procedure from the main higher order program as "PORTIN". There are no input/output parameters.

The CHARIN routine is used to input single characters from the plotter to the computer. This routine is called from the main program as an external procedure as "CHARIN (VAR INPUT:CHAR; VAR ERFLAG:CHAR)". INPUT is the character received from the plotter and ERFLAG is an error flag to indicate whether or not an error is encountered. If an error is encountered, an 'E' is returned to the main program, and a 'O' otherwise.

The CHAROT routine is used to output a single

character from the computer to the plotter. This routine is called as an external procedure from the main program as "CHAROT (VAR OUTPUT:CHAR; VAR ERFLAG:CHAR)". OUTPUT is the address of the character to be sent to the plotter and ERFLAG is as described above.

The LINOUT routine is used to output an array of characters from the computer to the plotter. This routine is called as an external procedure from the main program as "LINOUT (VAR LINE:ARRAY[1..40] OF CHAR; VAR ERFLAG:CHAR); where LINE is the address of the array of characters to be sent (the array must end with a } character) and ERFLAG is as described above.

This completes the description of the assembly language interface routines. These routines are hardware dependent and will require modification if different hardware is used. For a complete listing of these routines, refer to reference 2, pages A-1 through A-10.

Pascal Routines

The major Pascal programs written by Lt Rock include DIGITIZE, PLOTFILE, CODER, and, ERROR. These programs as well as some of the more important subroutines (procedures in Pascal) within these programs are explained in this section.

DIGITIZE. The purpose of the DIGITIZE program is to take points from the digitizer and place them in a user specified file. This file of points then serves as the original line drawing in all further coding and

performance analysis programs. The DIGITIZE program is fully interactive with the user entering the required digitizer instructions from the computer and operating the pen up/down and close file controls from the digitizer keypad.

In order to properly operate the digitizer under program control, Lt Rock developed a set of high level routines which communicate with the digitizer via the IEEE-488 controller and the assembly language routines discussed earlier. This set of high level routines was developed as a module called DIGRTNS. The major procedure within this module is called GETPOINT. When called by the DIGITIZE program, GETPOINT returns with the coordinates of the digitized point, the pen up/down indicator, and the annotation number entered from the digitizer keypad. Error or out-of-bound conditions are also handled by the GETPOINT procedure.

When the DIGITIZE program is finished, a file has been created with a separate line for each digitized point. Each line contains a pen up/down indicator (U or D), the x-y coordinates of the point, and the annotation number entered from the digitizer keypad. This file is ready for either plotting with PLOTFILE or coding with CODER.

CODER. The function of the CODER program is to convert a set of x-y coordinates into a single ring chain code with gridsize and level specified by the user. The

program reads the points of a digitized line drawing from a file and generates a file containing the chain codes and the coordinates of the nodes. This file can now be plotted by the PLOTFILE program for comparison between the original digitized version and the coded version of the line drawing.

To use CODER, the user interactively supplies the input and output file names and the gridsize and level of the chain code. The program then examines the points from the input file and calls a subroutine to determine where the drawing crosses the grid and the node that is closest to this intersection. It then outputs the coordinates of this new current node and the corresponding chain code to the output file.

The major procedure used by the CODER program is called WHERE. This is the routine that, when called, returns with the point where the digitized line intersects the grid. It also returns with the pen up/down information or if the end of the file was found. The CODER program then finds the closest node and computes the chain code for this node and adds this information as well as the x-y coordinate of the node to the output file.

PLOTFILE. The purpose of the PLOTFILE program is to allow the user to plot files generated by the DIGITIZE or CODER routines via the assembly language routines discussed earlier. In addition, the program will plot any file that adheres to the same format as files created by

DIGITIZE or CODER. The program is fully interactive, prompting the user for all necessary inputs and allowing the user the capability to scale and translate the drawing relative to its size and position on the digitizer. This program can provide a visual display of both the digitized line drawing and the coded version. An effective way of showing the difference between the digitized and the coded versions is to plot the two drawings directly on top of each other but using a different pen color.

ERROR. The purpose of the ERROR program is to provide a measurement of the precision, or accuracy, of the coded version of the line drawing. The program accomplishes this by finding the area difference between the digitized drawing and the coded drawing and dividing by the length of the digitized version. ERROR calculates the length of the digitized line by taking a summation of the square root of the sum of the squares of the x and y deltas for each two consecutive points. The program calculates the area error by, in effect, overlaying the two drawings and calculating the area of each of the resulting connecting closed figures and summing these areas up for the total area. An example of the connecting closed figures that result from the overlay are shown in Figure III-1.

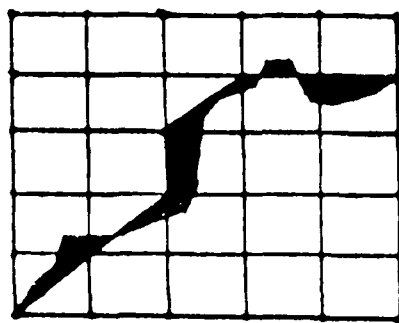


Figure III-1. Overlay of Coded Line onto
Input Line (2:IV-13)

To use ERROR, the user interactively supplies the name of the file containing the digitized points and the name of the file containing the coded version of the digitized drawing. The program then utilizes two major subroutines to find the area error; these are CLOSELOOP and GROUND. The CLOSELOOP procedure finds the intersections between the two versions of the drawing and returns to ERROR with a set of coordinates that describe a closed figure like those shown in Figure III-1. These coordinates are then sent to the GROUND subroutine which calculates the enclosed area and returns to the ERROR program. ERROR then sums the areas of all the closed figures until the end of the drawing is reached. This total area is then divided by the length of the digitized portion of the line drawing and displayed on the CRT along with the length of the digitized and coded lines.

This completes a brief description of the Pascal programs developed by Lt Rock that are pertinent to this thesis effort. A complete listing of these programs can be found in reference 2, pages A-11 through C-13. Also, a user's manual is included in pages D-1 through D-6 of reference 2. (Note: An updated user's manual is included in Appendix C of this thesis.)

Software Modifications.

Two of the software routines described above have been modified for this thesis effort. These are the PLOTFILE and ERROR routines. These modifications are each discussed below.

Modifications to PLOTFILE. The modification to PLOTFILE is minor; the only change is to allow the user to plot more than one file without having to call PLOTFILE from the disk for each file. This is accomplished by interactively asking the user how many files are to be plotted at the beginning of the program. A "FOR" statement is then used to loop through the original PLOTFILE program as many times as there are files to be plotted. The change in the original program listing is shown below (the original program listing can be found in Appendix B of reference 2).

```
BEGIN (* MAIN PROGRAM *)
  WRITELN('PROGRAM TO PLOT DIGITIZED POINTS FROM A FILE');
  WRITELN;
  WRITE('ENTER THE NUMBER OF FILES TO BE PLOTTED : ');
  READLN(NUMFILES);
  FOR A := 1 TO NUMFILES DO
    BEGIN (* FOR STATEMENT *)
```

```

        ERROR_FLAG := ' ';
        .
        .
        CHARIN(SIGNAL,ERROR_FLAG);
    END; (* FOR STATEMENT *)
    WRITELN;
    WRITELN('ALL DONE.....')
END. (* MAIN PROGRAM *)

```

Of course, NUMFILES and the FOR loop index variable, A, must be declared as integers in the variable declaration section of the program.

It can be seen from the listing that the user enters the number of files, then the program prompts for a file name, plotter line type, scale factor, and x-y translation. The program then plots this file and then interactively prompts for the information for the next file to be plotted. This repeats until the last file is plotted. This modification results in a more convenient and efficient program if the user has many files to be plotted.

Modifications to ERROR. The first modification to the ERROR program accomplishes the same purpose as that described for the PLOTFILE program. The change was implemented in a similar manner; however, the program was made even more automatic than the PLOTFILE routine. The difference is that the user does not have to wait until a file is analyzed to enter the data for the next file; this is all done at the beginning of program operation. This modification, along with the CONTROL-P CP/M toggle (sends all CRT output to the printer), allows the user to enter

the files to be coded and then leave the computer (the program can run for several hours completely unattended if there are many files to be analyzed). The changes in the Pascal code are shown below.

```
BEGIN (* MAIN PROGRAM *)
  REPEAT
    WRITE(' ENTER DIGITIZED DATA FILENAME : ');
    READLN(FILENAME);
    ASSIGN(D,FILENAME);
    RESET(D);
  UNTIL IORESULT <> 255;
  WRITELN;
  WRITE('ENTER NUMBER OF CODED FILES TO BE ANALYZED : ');
  READLN(NUMFILES);
  (* THE FOLLOWING SECTION OF CODE PROMPTS THE USER FOR ALL
    THE FILENAMES OF THE CODED FILES *)
  FOR J := 1 TO NUMFILES DO
    BEGIN
      WRITE('ENTER CODED DATA FILENAME ',J,' : ');
      READLN(FILENAME);
      CODEFILE[J] := FILENAME;
    END;
  (* THE ORIGINAL PROGRAM IS NOW EMBEDDED IN A FOR LOOP *)
  FOR J := 1 TO NUMFILES DO
    BEGIN
      REPEAT
        WRITELN;
        WRITELN('ANALYZING ',CODEFILE[J]);
        FILENAME := CODEFILE[J];
        ASSIGN(C,FILENAME);
        RESET(C);
        RESET(D)
      UNTIL IORESULT <> 255;
      FINISH := FALSE;
      .
      .
      .
    END (* FOR STATEMENT *)
  END. (* MAIN PROGRAM *)
```

The new variables to be added to the program variable declaration list are shown below:

```
CODEFILE : ARRAY [1..25] OF STRING;
NUMFILES, J : INTEGER;
```

This modification greatly enhances the efficiency of the program from the standpoint of the amount of time the

the program begins reading points from the input file and using the INSERT routine to update the list.

After a node is encoded, a routine is needed to eliminate the points in the list that precede the point that intersected the ring. If these points were not eliminated, the list might grow large enough to use all the microcomputer's memory and program execution would end. The ALIGN subroutine performs this function. In addition to eliminating the points, it adjusts the pointer variables.

After the end of the line segment has been reached, the CLRLIST routine is called to eliminate all points from the list. This completely clears the computer's memory for the next line segment.

INTERSECT. The INTERSECT subroutine is the one that finds the intersection points of each ring level out to the outermost ring or the end of the line segment. It also returns with a boolean variable for each ring indicating whether or not the ring was intersected. In the case of a single ring code, only the intersection of the ring used in the code is found. The algorithm used to find these intersection points is the same as that used by DILLER in his CODER program (2).

COMPCODE. The COMPCODE subroutine is called after all the intersection points have been found. Its purpose is to compute the chain code for this particular link in the chain. This is a long procedure containing many

were not used and the encoded node lies on one of the inner rings, then the points read from the file that were between the encoded node and the outer ring would be lost because Pascal does not have the capability to read backwards through a text file. Since it is impossible to determine in advance how many points there are between rings and for ease in keeping track of a pointer variable, it was deemed appropriate to use a Pascal linked list instead of declaring an array large enough to handle the largest possible number of points. The data structure most appropriate for the list is the queue. This is a first in first out (FIFO) structure.

Immediately after a point is read in from the input file, the INSERT routine is called. This routine inserts the point into the rear of the queue and updates pointer variables to keep track of the front and back of the list (first and last points).

As soon as a node is encoded, the main program starts searching for ring intersection points; this search always begins with the points in the linked list. To read a point from the list, the READLIST subroutine is called. This subroutine reads in the point and increments a pointer variable that keeps track of the sequence of points in the list. The points are read from the list until all the points in the list are read or the outermost ring is again intersected. If all the points in the list are read before the outermost ring is intersected, then

and the chain code to the output file.

9). Eliminate from the linked list all points up to the point that determined the intersection coordinates of the ring that was selected for encoding.

10). If the end of the current line segment has not been reached, go to step 4.

11). If the end of the current line segment has been reached, go to step 4 and repeat until the linked list pointer is nil. When the end of the linked list is reached, output the coordinates of the end point and a chain code of -1.

12). If the end of the line drawing has not been reached, go to step 3.

13). If the end of the line drawing has been reached, close the output file, clear the linked list, and go to step 2.

14). Continue until the last coded file is created.

Subroutines

Although the program listing in Appendix A is thoroughly documented, a brief explanation of the purpose and operation of the most important subroutines used will aid in the understanding of the program.

INSERT, READLIST, ALIGN, and CLRLIST. These are the subroutines that do the housekeeping for the linked list. The list is needed to store the points read in from the input file until an intersection point for the outermost ring is found. If some method of storing the data points

make it the first node of that line segment, and write the coordinates to the output file along with a chain code of -1, the gridsize, and the rings used in the code. Also, make this point the first in a Pascal linked list.

4). Input points from the linked list, or from the input file if the list pointer is nil, until the innermost ring of the code has been intersected and calculate the coordinates of the intersection. As points are read from the input file, insert them into the linked list.

5). Repeat step 4 for the remaining rings of the code until the outermost ring is intersected or the end of the line segment is reached.

6). If the code is a single ring code, then find the node closest to the intersection point and compute the chain code for that node, then go to step 8. If the code is a multi-ring code, then set $i = k$, where k is the order of the highest order ring in the code.

7). Find the node closest to the intersection point on the ring i and determine whether the drawing passes the LGS test, described in Chapter II, for that node. If it does, then compute the chain code for the node. If not, then let $i = i - 1$. If i is the lowest order ring in the code, then find the closest node and compute the chain code; if i is not the lowest order code, repeat this step. Continue this process until a ring passes the LGS test or the innermost ring is reached.

8). Output the coordinates of the new current node

right of the current node. Numbering then proceeds sequentially in a counter-clockwise direction until the last node of the innermost ring is reached. The next number is then assigned to the node on the next ring which is directly to the right of the current node and the numbering proceeds as before. This continues until every node on the outermost ring used in the code is numbered. Figure IV-1 illustrates this concept for a (1,2) code.

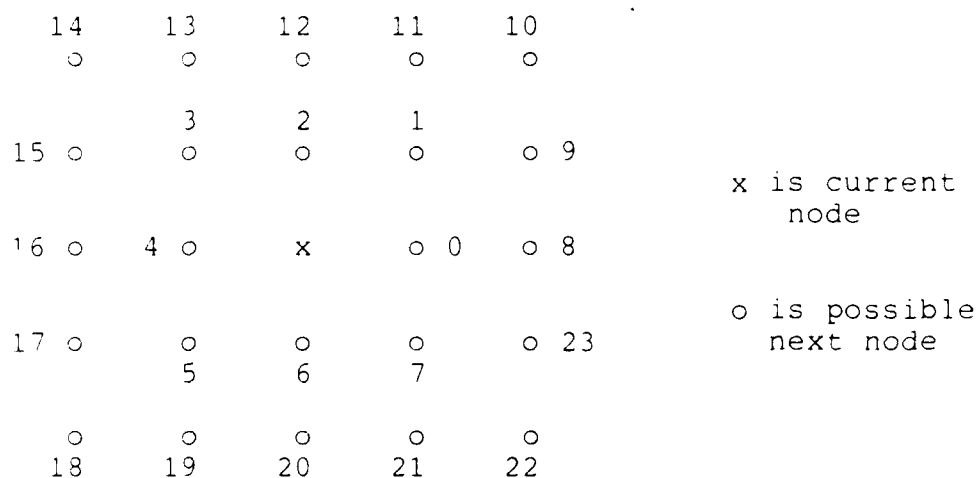


Figure IV-1. Nodes for a (1,2) Chain Code

The operation of the chain coding procedure itself is described by the following set of steps:

- 1). Interactively get parameters from the user such as filename of digitized data file, number of coded files to be created (maximum of 25), a name for each coded file, gridsize and rings used for each coded file.

- 2). Initialize variables.

- 3). Find the first point of the next line segment,

listing for these programs is contained in Appendix A and a user's manual is included in Appendix C.

Overall Operation of CHNCODE Program

The method of chain coding implemented in this thesis is very similar to the method used by Lt Rock for the single ring CODER program described in Chapter III (2). The first point of each line segment in a line drawing is used as the origin of the grid and is the first point in the chain code. The algorithm then follows the line drawing and constructs the chain code until the end of the line segment is reached, with the last point on the segment being the last point in the chain code. Following Lt Rock's convention, the chain code for the first and last points of each line segment is defined to be -1. Also, as mentioned earlier, the file format of pen up/down indicator, x-y coordinate of the node, and then the chain code, is adhered to in this program. The first line of each new line segment also contains the gridsize and ring levels used by the chain code. The end of line drawing code is the end of file indicator. The program as written allows a chain code using any combination of the first five rings (or levels) described in Chapter II. The program is written so that it would be a simple matter to modify the Pascal code to allow more rings.

In this algorithm, the numbering system for the chain code assigned to each node begins with a zero assigned to the node on the innermost ring which is directly to the

IV. Implementation of Multi-ring Chain Coding Algorithm

This chapter describes the design, implementation, and testing of a multi-ring chain coding algorithm for use on the system described in Chapter III. The program, called CHNCODE, allows the user to create chain coded versions of a digitized line drawing using any combination of ring levels one through five. It interactively prompts the user for the filename, gridsize, and rings used and allows the user to create up to 25 coded versions of the digitized input file. This program uses the same file format as that described in Chapter III and is fully compatible with the ERROR and PLOTFILE programs. Also described in this chapter is a program that combines the functions of the DIGITIZE, CHNCODE, and PLOTFILE programs while eliminating the use of disk files. This program calculates the chain code as the drawing is being traced on the digitizer and immediately outputs the chain coded drawing to the plotter. This process takes place in as close to real time as the computer processing speed allows. This program is called PLOTCODE.

First, this chapter presents an overall explanation of the chain coding algorithm and then briefly describes the purpose and operation of each subroutine (procedure in Pascal) in the program. Then, the method used to test and verify the program is discussed. Lastly, the development of the PLOTCODE program is discussed. The complete source

procedure. This procedure is called when the end of the digitized file is reached and there are points remaining in the coded file; its purpose is to read in the rest of the points in the coded file. The error in the routine was that these points were being placed in the digitized point array instead of the coded point array. This problem was easily corrected and the modified procedure listing is shown below.

```
PROCEDURE LASTDIG;

BEGIN (* LASTDIG *)
  REPEAT
    READLN(C,PEN,XVAL,YVAL);
    LENGTHC;
    NUMNODES := NUMNODES + 1;
    NUMCODE := NUMCODE + 1;
    CODED[NUMCODE,1] := XVAL;
    CODED[NUMCODE,2] := YVAL;
  UNTIL EOF(C) OR (PEN <> 'D');
  FINISH := TRUE
END; (* LASTDIG *)
```

This completes the modifications to the original system software.

Summary

This chapter has provided a brief overview of the programs and algorithms developed by Lt Rock that are used or modified for this thesis. Each of the major subroutines and programs of the existing system were described and the modifications required for this thesis were explained.

procedure when the slopes of the coded and digitized segments being investigated for an intersection did not match. This problem was easily corrected by creating new variables that provide the direction of both of these lines, instead of using only the coded line. The Pascal code to correct the procedure is shown below:

```

PROCEDURE CHECKPNT;

VAR SIGNXC, SIGNYC, SIGNXD, SIGNYD: INTEGER;

BEGIN (* CHECKPNT *)
  SIGNXC := CODED[NOCODE,1] - CODED[NOCODE - 1,1];
  IF SIGNXC <> 0 THEN IF SIGNXC > 0
    THEN SIGNXC := 1 ELSE SIGNXC := -1;
  IF SIGNXC = 0 THEN SIGNXC := 1;
  SIGNYC := CODED[NOCODE,2] - CODED[NOCODE - 1,2];
  IF SIGNYC <> 0 THEN IF SIGNYC > 0
    THEN SIGNYC := 1 ELSE SIGNYC := -1;
  IF SIGNYC = 0 THEN SIGNYC := 1;
  SIGNXD := DIG[NOPTS,1] - DIG[NOPTS - 1,1];
  IF SIGNXD <> 0 THEN IF SIGNXD > 0
    THEN SIGNXD := 1 ELSE SIGNXD := -1;
  IF SIGNXD = 0 THEN SIGNXD := 1;
  SIGNYD := DIG[NOPTS,2] - DIG[NOPTS - 1,2];
  IF SIGNYD <> 0 THEN IF SIGNYD > 0
    THEN SIGNYD := 1 ELSE SIGNYD := -1;
  IF SIGNYD = 0 THEN SIGNYD := 1;
  IF (SIGNXC * ROUND(SEGINTR[1] - CODED[NOCODE - 1,1])
    >= 0) AND
    (SIGNXC * (CODED[NOCODE,1] - ROUND(SEGINTR[1])))
    >= 0) AND
    (SIGNYC * (CODED[NOCODE,2] - ROUND(SEGINTR[2])))
    >= 0) AND
    (SIGNYC * (ROUND(SEGINTR[2] - CODED[NOCODE - 1,2]))
    >= 0) AND
    (SIGNXD * (ROUND(SEGINTR[1] - DIG[NOPTS - 1,1])
    >= 0) AND
    (SIGNXD * (DIG[NOPTS,1] - ROUND(SEGINTR[1])))
    >= 0) AND
    (SIGNYD * (DIG[NOPTS,2] - ROUND(SEGINTR[2])))
    >= 0) AND
    (SIGNYD * (ROUND(SEGINTR[2] - DIG[NOPTS - 1,2])
    >= 0)
    THEN CROSS := TRUE ELSE CROSS := FALSE
END; (* CHECKPNT *)

```

The last problem was discovered in the LASTDIG

```

XD := XVAL / 1000;
YD := YVAL / 1000;
LX := LASTXD / 1000;
LY := LASTYD / 1000;
LONGDIG := LONGDIG + SQRT(SQR(XD - LX) + SQR(YD - LY));
LASTXD := XVAL;
LASTYD := YVAL
END; (* LENGTHD *)

PROCEDURE LENGTHC; (* THIS PROCEDURE CALCULATES THE LENGTH
                     OF THE CODED LINE *)

VAR XC, YC, LX, LY : REAL;

BEGIN (* LENGTHC *)
  XC := XVAL / 1000;
  YC := YVAL / 1000;
  LX := LASTXC / 1000;
  LY := LASTYC / 1000;
  LONGCODE := LONGCODE + SQRT(SQR(XC - LX) + SQR(YC - LY));
  LASTXC := XVAL;
  LASTYC := YVAL
END; (* LENGTHC *)

```

This modification brought the variables down to a size the compiler could handle without errors.

Another problem was found in several of the procedures of the program in the decision (IF) statements. This problem occurred because the variables used in these decision statements were real variables, and were subject to the normal truncation error common in compilers. The problem was solved by declaring the variables, the CODED and DIG arrays, to be integers. This solution made it necessary to use the ROUND instruction whenever these variables were assigned the value of a real variable, such as SEGINTR. This rounding process has no effect on the accuracy of the program because the values read into these arrays are restricted to integer values anyway.

Another problem was revealed in the CHECKPNT

The other modifications to the ERROR routine were added when it was discovered that it was not properly analyzing certain drawings. The first such modification proved to be caused by a deficiency in the MTPLUS Pascal compiler. The problem surfaced when it was noted that the length of the drawings being returned by the program could not possibly be correct. Lt Rock's method of calculating these lengths was straight forward and obviously correct. He used a separate subroutine (procedure) to calculate each length. These procedures were exactly alike except LENGTHC calculated the length of the coded line and LENGTHD calculated the length of the digitized line. The key statement in each procedure was (shown only for the LENGTHC procedure):

```
LONGCODE := LONGCODE + SQR(SQR(XVAL - LASTXC) + SQR(YVAL -
                        LASTYC));
```

After some testing, it was discovered that the double transcendental operation " SQR(SQR(...)) could not be handled by the compiler if the SQR(...) variable was large (on the order of 10^3). This problem was corrected by eliminating the factor of 1000 at this point instead of waiting to the OUTDATA procedure as was done for the area error metric. The new code listings for both procedures are shown below.

```
PROCEDURE LENGTHD; (* THIS PROCEDURE CALCULATES THE LENGTH
                    OF THE DIGITIZED LINE *)
```

```
VAR XD, YD, LX, LY : REAL;
```

```
BEGIN (* LENGTHD *)
```

```

IF (NODES > 32) AND (NODES <= 64) THEN BITSNODE := 6;
IF NODES > 64 THEN BITSNODE := 7;
NUMBITS := NUMNODES * BITSNODE
END; (* NOBITS *)

```

Note that the procedure uses the fact that there are $8n$ nodes to a ring as its basic principal of operation.

Another modification to the ERROR program is the OUTDATA procedure. Before the modification, the output of the ERROR program gave the area error multiplied by a factor of 10^6 and the length of the lines multiplied by a factor of 10^3 . The reason for this is that the data points stored in the digitized and coded files are in the format 1 unit = 1/1000 inches. This modification simply eliminates the need for the user to perform the correction arithmetic. In addition, the OUTDATA procedure outputs the number of bits and (number of bits) / (length of digitized line). The Pascal code to accomplish the modification is shown below.

```

PROCEDURE OUTDATA;
BEGIN (* OUTDATA *)
  WRITELN;
  WRITELN(' AREA = ', AREA / 1E+06, ' SQUARE INCHES' );
  WRITELN(' AREA / LENGTH = ', (AREA / 1E+06) / LONGDIG, ' INCHES' );
  WRITELN;
  WRITELN(' LENGTH OF CODED LINE = ', LONGCODE, ' INCHES' );
  WRITELN(' LENGTH OF DIGITIZED LINE = ', LONGDIG, ' INCHES' );
  WRITELN;
  WRITELN(' THE TOTAL NUMBER OF BITS = ', NUMBITS, ' BITS' );
  WRITELN(' NUMBER OF BITS / INCH = ', NUMBITS / LONGDIG );
  WRITELN
END; (* OUTDATA *)

```

The point in the main program where OUTDATA is called was shown earlier in the discussion of the NOBITS modification.

```

        LOOP := LOOP + 1;
        READ(C,LEVEL[LOOP])
    END;
WHILE LOOP <> 5 DO
    BEGIN
        LOOP := LOOP + 1;
        LEVEL[LOOP] := 0
    END;
    .
    .
    .
UNTIL FINISH OR ERFLAG;
NOBITS; (* THIS COMPUTES THE NEW METRIC *)
OUTDATA (* ANOTHER MODIFICATION DESCRIBED LATER *)
END (* FOR STATEMENT *)
END. (* MAIN PROGRAM *)

```

The NOBITS procedure was added to calculate the number of bits used to encode the chain. The listing of the procedure is shown below. Note that the program is capable of handling up to five code rings (the CHNCODE program handles up to five rings).

```

PROCEDURE NOBITS; (* THIS PROCEDURE CALCULATES THE TOTAL
                    NUMBER OF BITS NEEDED TO REPRESENT
                    THE CHAIN CODED DRAWING *)

VAR
    LOOP, BITSNODE, NODES : INTEGER;

BEGIN (* NOBITS *)
    NODES := 0;
    FOR LOOP := 1 TO 5 DO
        BEGIN
            CASE LEVEL[LOOP] OF
                0 : NODES := NODES;
                1 : NODES := NODES + 8;
                2 : NODES := NODES + 16;
                3 : NODES := NODES + 24;
                4 : NODES := NODES + 32;
                5 : NODES := NODES + 40
            END (* CASE STATEMENT *)
        (* THE CASE STATEMENT CALCULATES THE NUMBER OF
           POSSIBLE NEXT NODES FROM ANY CURRENT NODE *)
        END; (* FOR STATEMENT *)
        (* THE FOLLOWING IF STATEMENT DETERMINES THE NUMBER
           OF BITS NEEDED TO ENCODE EACH NODE IN THE DRAWING *)
        IF NODES <= 8 THEN BITSNODE := 3;
        IF (NODES > 8) AND (NODES <= 16) THEN BITSNODE := 4;
        IF (NODES > 16) AND (NODES <= 32) THEN BITSNODE := 5;

```

user must spend monitoring the computer when there are many files to be analyzed.

Another modification to the ERROR program adds an output metric to the program. This metric is the number of bits and the number of bits divided by the length of the digitized line. This change allows the user to compare the compactness of the various chain codes for a particular drawing.

The modification required that the program read the first line of the coded file to obtain the ring levels used for the code (the CHNCODE program described in Chapter IV of this thesis is a multi-ring coding program). This is easily done by slightly modifying the main program to read these levels from the first line of the coded file. A variable, NUMNODES, is set to one when this first point is read and incremented by one every time a subsequent point is read from the coded file. This change to the main program listing is shown below beginning with the line where the first coded point is read.

```
READ(C,PEN,XVAL,YVAL,LEVEL[1],LEVEL[2]);
```

```
(*LEVEL IS THE VARIABLE NAME FOR THE RING LEVELS USED--  
IN THE ABOVE STATEMENT LEVEL[1] AND [2] TEMPORARILY HOLD  
THE VALUE OF THE -1 CHAIN CODE AND THE GRIDSIZE; HAD TO  
READ THEM BECAUSE THEY ARE IN THE FIRST LINE OF THE  
CODED FILE BEFORE THE ACTUAL VALUES FOR THE RING  
LEVELS *)
```

```
NUMNODES := 1;  
CODED[1,1] := XVAL;  
CODED[1,2] := YVAL;  
LOOP := 0;  
WHILE NOT EOLN(C) DO  
  BEGIN
```

internal subroutines; for this reason, it was written as a program module and compiled separately. The operation of the module is described below.

Beginning with the outermost ring, each ring is analyzed to determine if it can be used for encoding. The COMPCODE subroutine checks one ring at a time until a ring is found that passes the LGS test and is suitable for encoding, or the lowest ring is reached (in which case no LGS test is needed). The subroutine contains additional internal subroutines that find the closest node to the intersection point on the ring under test and determine if the LGS test for that node is successful. If it is, then the node is encoded and the COMPCODE subroutine is exited; if not, the process repeats with the next ring in the code as described in Chapter II of this thesis.

To perform these functions, COMPCODE contains three major internal subroutines: FINDNODE, RINGTEST, and ENCODE. The FINDNODE subroutine finds the node on the ring under investigation that is closest to the point where the line drawing intersected the ring. The RINGTEST procedure then determines where the intersection occurred on the ring (top, corner, or side) and calls subroutines that perform the LGS test for the node found in FINDNODE. The subroutine called by RINGTEST depends on the location of the node found in FINDNODE. If the LGS test is not passed, then the entire process begins again with FINDNODE for the next lowest ring in the code. The process repeats

until a ring that passes the LGS test is found or the lowest ring in the code is reached. In either case, the ENCODE routine is then called to determine the chain code for the node. The algorithm used for the ENCODE subroutine is the same one Lt Rock used in his CODER program. The only modification necessary to allow for multi-ring codes was to add $8n$ to the code for each ring lower than the ring selected for encoding, where n is the order of the lower ring.

DONE. The DONE subroutine is called when the end of a line segment or the end of the drawing is reached. This subroutine supervises reading through the remainder of the linked list until the last possible chain code is found. It then outputs the last point in the line segment to the code file with a chain code of -1.

This completes the discussion of the major subroutines contained within the CHNCODE program. This discussion, along with the program listing in Appendix A, should provides sufficient information for anyone wishing to use or modify this program.

Testing of CHNCODE.

The methodology used to test the CHNCODE program is discussed below. Each subroutine was tested individually before the complete program was compiled. After each of the subroutines was verified to be performing correctly, the complete program was compiled and tested in several

different ways. The first test was to compare the CHNCODE program results against Lt Rock's CODER program for single ring codes. Although this only tested the program for single ring codes, it was the only way to test long, complex input files with known results. After comparing results against the CODER program at all five ring levels and for five different drawings, both coding programs were found to produce exactly the same output for each file. Therefore, CHNCODE passed this portion of the testing.

The next step in the testing process was to test the program using simulated files (input files created with a text editor) that could simulate particular situations and determine if the coding program would handle them correctly. These files were always short (none were more than 20 points) and the correct results were calculated by hand. Situations were tested in which the drawing left the current node in all directions and for all possible ring combinations necessary to include every subroutine in the program. These tests insured that the correct node was encoded and that the search for ring intersections after the node was encoded started with the correct point. In short, every situation that was conceived of was tested via the simulated files. After a few minor corrections, the program passed all testing and is presumed to be operating correctly.

PLOTCODE

The PLOTCODE program operates using the same basic

algorithms as the CHNCODE program to calculate the chain codes. Since these two programs are so similar, a discussion of their differences will suffice to explain the PLOTCODE program. Since PLOTCODE must of necessity operate on one drawing at a time, one minor difference between the two programs is that the counting loop that allowed CHNCODE to code 25 input files has been eliminated. The filename and text file variables are also eliminated. The major difference between the two programs is the input of the line drawing and the output of the chain coded version of the drawing. Instead of reading in points from an input file representing the line drawing, a subroutine is called that reads the input points in directly from the digitizer. And, instead of writing the chain code information to an output file, a subroutine is called to output this information to the plotter. The additional subroutines required for the PLOTCODE program are discussed below.

PLOTIN. The PLOTIN subroutine is a simple routine that builds an array of characters that holds the initialization information needed to output information to the plotter. The array holds plotter "wake-up" characters (::), the communications mode, and line type. Every time a line is output to the plotter, the line consists of this array with the x y coordinates appended to it.

READ DIG. The READ_DIG subroutine is called when the main program is ready to read new input points. This

subroutine then calls subroutines from the DIGRTNS module (discussed in Chapter III) that read the input points from the digitizer. These points come from the DIGRTNS module in an ASCII format; READ_DIG translates this ASCII information into numerical data for the main program.

PLOT. The PLOT subroutine is called when the coordinates of the link in the chain code have been determined. The subroutine then transforms the numerical data representing the x y coordinates of the node into the ASCII representations needed to communicate with the plotter. This information is then appended to the array discussed in the PLOTIN subroutine and then sent to the plotter.

A program listing for the PLOTCODE program is included in Appendix A, except for the COMPCODE module. The only difference between the version of COMPCODE used for PLOTCODE and the version used for CHNCODE is the counting loop variable in PLOTCODE that allows the user to create up to 25 files at once. Therefore, the COMPCODE module need not be listed again. Note that the elimination of the counting variable reduces the NUMRINGS and GRIDSIZE arrays to integers and the LEVEL array from a two dimensional to a single dimensional array.

Effect of Digitizer Resolution on CHNCODE and PLOTCODE

A important factor to bear in mind when using the CHNCODE and PLOTCODE programs is the hardware limitations imposed by the digitizer. Specifically, this is the

resolution of the digitizer, 0.001 inches. To understand the limiting effect of this resolution, consider as an example the (1,4) chain code shown in Figure IV-2. In this example, the line drawing has passed close to the upper right corner node of the level 4 ring; therefore, the link gate sets to determine if the fourth ring can be used for encoding are as shown in Figure IV-2. It is a simple matter of geometry to show that the length of the LGS on ring 1 is 0.125 of the chosen gridsize. If the gridsize is chosen small enough, this can approach the 0.001 inch limitation of the digitizer. For example, if the gridsize were 0.05 inches, then the LGS region for ring 1 would be 0.00625 inches, only 6.25 times greater than the digitizer resolution. This results in significant uncertainty in whether the ring actually passes the LGS test or not. In an effort to avoid this problem, all codes used in this thesis will result in a minimum LGS region of at least 10 times the digitizer resolution. This is assumed to be large enough to avoid significant error.

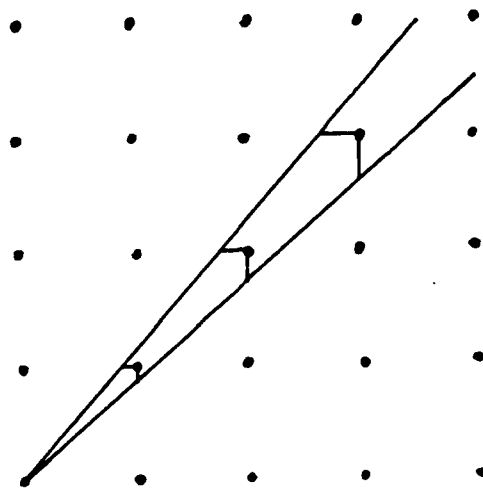


Figure IV-2. Link Gate Sets for a (1,4) Code

Summary

In this chapter, the design and implementation of the multi-ring chain coding programs called CHNCODE and PLOTCODE were discussed. Each of the major subroutines in the two programs were discussed and their purpose and operation explained. The chapter ended with a discussion of a limitation imposed on the gridsize chosen for the encoding process by the resolution of the digitizer.

V. Analysis of Various Line Drawings

This chapter contains an analysis of the performance of the the chain coding system for several different types of line drawings. These drawings include simple geometric figures such as circles and squares, as well as a sine wave and written text. The drawings were all traced by hand using either the cursor or stylus of the digitizer; therefore, they are not geometrically perfect. However, this analysis should yield some information on how well the chain codes perform in a realistic implementation.

The performance of the chain codes is analyzed using the area error per unit length (hereafter referred to as aepl), the number of bits per unit length required for storage (hereafter referred to as bpl), and the smoothness of the chain coded versions of the drawings. The aepl and bpl are analyzed and related to the smoothness of the coded versions of the drawings. The smoothness analysis is a subjective evaluation based on a visual inspection of the coded versions of the drawings as drawn on the plotter. Each drawing is analyzed at various gridsizes and for several different chain codes. The intent of the analysis is to discover how well the chain coding algorithm performs in a realistic chain coding environment, and to determine if there is any correlation between the aepl and bpl metrics and the visual appearance of the drawings. Appendix B contains all the tables and

figures referenced in this chapter pertaining to the performance of the codes.

Circle

First, a circle with a diameter of approximately 4.5 inches was digitized and the performance of the chain codes was examined. The circle, hereafter referred to as CIRCLE, was encoded using several different gridsizes and chain codes. The circle is shown in Figure B-1. The chain codes used for analysis are the (1), (2), (3), (4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4), (1,2,3), (1,2,4), (1,3,4), (2,3,4), and (1,2,3,4). Each of these codes were used at gridsizes of 0.25, 0.2, 0.15, and 0.1 inches. All codes except the (1,3), (1,4), (1,2,3), (1,2,4), (1,3,4), and (1,2,3,4) were also coded at a gridsize of 0.05 inches. These codes were not coded at 0.05 inches because of the uncertainty introduced by the digitizer resolution (see Chapter IV).

Trends in aepl and bpl Performance. In this section, the aepl and bpl metrics were examined from the standpoint of looking for trends in chain code performance. This analysis does not consider the smoothness of the drawing, which is considered in the next section. Tables B-1 through B-5 show the performance of each of the codes for each gridsize.

The aepl increased with gridsize for most of the codes. This relationship was almost linear; however, it

no longer holds for several of the multi-ring codes when the gridsize reaches 0.2 inches. The bpl was inversely proportional to the gridsize for all of the codes, and this relationship held up to the largest gridsize evaluated, 0.25 inches.

Analyzing the performance of each of the chain codes at the various gridsizes and looking for trends, the single ring (2), (3), and (4) codes consistently provided the worst aepl performance and the best bpl performance. The (2,3), (2,4) (3,4), and (2,3,4) codes were sometimes grouped with these single ring codes; the poor aepl performance of these codes demonstrates the significant contribution the first ring level makes in aepl accuracy. The (1) code aepl performance was poor; however, generally not as bad as the other single ring codes. Its bpl performance was also consistently among the poorest. It was even worse than the multi-ring codes which require more bits per point, thus demonstrating the utility of the outer rings in reducing the number of points required to encode the circle. The (1,2) code's aepl performance was mediocre, usually falling in the middle, and its bpl was consistently one of the worst. Generally, the (1,3), (1,2,3), (1,2,4), (1,3,4), and (1,2,3,4) codes had the best performance for the aepl and were usually clustered around the middle for the bpl metric. The (1,2,3,4) code showed the best aepl performance only for the 0.25 inch gridsize and was usually in fourth or fifth place for the

other gridsizes. Also, it showed consistently poor bpl performance; therefore, it seems that a 4-level code does not yield any improvement over the 3-level codes. Apparently, the best combination of aepl and bpl performance came from the 3-level codes (1,2,3), (1,2,4), and (1,3,4). The 2-level code (1,3) was also a strong performer in both areas. None of these codes were superior to the others for all gridsizes; therefore, it is difficult to say which is the best performing code based strictly on an examination of the aepl and bpl metrics.

Smoothness vs aepl and bpl. This section consists of subjective comparisons of the smoothness of the different drawings. Also, an attempt is made to try to determine if there is a relationship between aepl and smoothness or between bpl and smoothness.

Gridsize of 0.25 Inches. None of the drawings coded with a gridsize of 0.25 inches were of good quality. The smoothest drawings appeared to be the (4), (3,4), (2,3,4) and (1,2,3,4) codes. The (1,2,3,4) code is shown in Figure B-2. The rest of the drawings appeared either somewhat deformed (did not appear round) or suffered from abrupt changes in slope like those shown in the (1,3) and (1,4) codes in Figure B-3. Easily the worst drawing was for the (1) code (see Figure B-4). This was obviously due to the limited angular resolution of the (1) code. A problem that occurs for many of the codes and at all the gridsizes is typified by the (2,4) code shown in Figure B-

5. Note that both upper quadrants and the lower left quadrant contain straight line segments which detract from the curvature expected in a circle.

Comparing the smoothness evaluation with the aepl of the codes yields some interesting results. One of the smoothest codes, the (4) code, possessed the highest aepl. The other codes with the highest aepl's were the (2), (3), and (2,4) codes. These codes appeared deformed and asymmetric. The (1,3,4) and (1,2,3,4) codes had two of the lowest aepl ratings; however, both contained abrupt changes of slope in the lower left quadrant which detract from the appearance of the drawing. In general, while a low aepl at this gridsize can indicate a smooth drawing, smoothness is more a function of continuity and symmetry than of aepl. These two attributes are more pleasing to the eye and more resemble a circle than the drawings that possess asymmetries or abrupt changes in slope.

The bpl seemed to have very little to do with smoothness. The drawing with the highest bpl (therefore, the one that might be supposed to contain the most information) was the (1) code, which was the worst performing code with respect to smoothness. It seems that the number of bits used to encode the nodes or even the number of nodes is not as important as the placement and symmetry of the nodes, at least for drawing a circle.

Gridsize of 0.2 Inches. The drawings coded at 0.2 inches were also of poor quality. Several of the

drawings appeared about equal in smoothness; these were the (4), (2,4), and (3,4) codes. Others that were almost as smooth are the (3), (2,3) (1,2,3), (1,2,4), (2,3,4), and (1,2,3,4) codes. Most of the other codes suffered significantly from the abrupt change in slope problem (see the (1), (1,2), (1,3), and (1,4) codes shown in Figure B-6). Again, the worst drawing was the (1) code. Overall, the drawings did not appear as smooth as the drawings done at the 0.25 inch gridsize. None of them seemed symmetrical and the slope changes were generally more severe. Even one of the smoothest drawings, the (4) code, possessed a long straight line segment on the right side, detracting from the expected curvature.

Again, one of the smoothest codes, the (4) code possessed the highest aepl. Two other relatively smooth drawings, the (2,4) and (3,4) codes, also had a high aepl. The drawings with the lowest aepl, the (1,2,3), (1,2,4), (1,3,4), and (2,3,4) were quite smooth; however, each had one or more continuity changes that detracted from their overall appearance. In general, the multi-ring codes possessing the (1) ring contain less aepl; however, they are afflicted with the abrupt changes in slope. The codes with the higher rings such as the (2,4) and (3,4) are smoother because there are no abrupt changes in direction. As in the 0.25 inch gridsize drawings, the value of the bpl seemed to have less effect on smoothness than the placement of the nodes.

Gridsize of 0.15 Inches. The drawings coded with a gridsize of 0.15 inches are of much better quality than those coded at the higher gridsizes. The (3) code, shown in Figure B-7, resulted in the smoothest drawing. Others that are quite good are the (4), (3,4), and (1,2,4) codes. The worst code was again the (1) code, with many abrupt changes in direction detracting from a smooth curvature. Most of the other codes also suffered somewhat from changes in direction. In general, however, the drawings for this gridsize did not possess the wide range of smoothness variations that characterized the drawings at the larger gridsizes of 0.25 and 0.2 inches. Except for the (1) code, the drawings at this gridsize did not differ from each other significantly.

As in the larger gridsizes, some of the smoothest codes possessed the highest aepl ratings. These codes, the (3) and (4) codes, are smooth because of their high angular resolution and because the length of the line segments forming the circle are constant. Again, this was more important than the aepl accuracy. Also, the codes that did perform well in aepl were fairly smooth, just as they were at the higher gridsizes. However, they did possess abrupt changes in slope that detract from a circular appearance. Again, the placement of the nodes seems more related to smoothness than the bpl metric.

Gridsize of 0.1 Inches. The quality of the drawings coded at 0.1 inches was approximately equal to

the (1) ring, and these drawings still have the highest bpl. However, the aepl did not vary widely at this gridsize; there is only 0.009 square inches difference between the maximum and minimum ratings. The smoothest drawing, because it has the least zigzagging, is the (4) code shown in Figure B-21; it also possesses one of the highest aepl ratings and the lowest bpl rating.

The drawings for SQUARE-30 are following the pattern set at the larger gridsizes. The zigzagging problem remains; however, the corners are not chopped off as badly as they have been. The (2,4), and (3,4) codes shown in Figure B-22 produced the smoothest drawings while possessing mediocre aepl ratings. Even at this small gridsize, the drawings are marginal. Again, the drawings with the lowest aepl ratings, the multi-ring codes using the (1) ring, were not the smoothest because of the zigzagging. The (1,3) code shown in Figure B-22 is typical of these codes. The bpl ratings followed the usual pattern of lowest for the single ring high order codes and highest for the multi-ring codes using the (1) ring.

In smoothness, the drawings for SQUARE-45 look much the same as they did for the drawings coded using a gridsize of 0.15 inches. They do not suffer significantly from the zigzagging effect and the chopped off corners are not as bad. The lower order codes using the (1) ring were still smoother than the higher order codes because the

ring. The worst looking drawing was again for the (1) code shown in Figure B-20. None of the drawings at this gridsize were of acceptable quality. In general, the best aepl ratings were for the multi-ring codes which used the (1) ring while they also possessed the highest bpl. The codes with the lowest bpl were the high order single ring codes. Also, the smoothest code, the (3,4) code, possessed a relatively low rating.

At this gridsize, the smoothest drawings were again for SQUARE-45. Although these drawings did not appear as perfect as they did at the 0.2 inch gridsize, they suffered less deformities than did the other squares. None of the drawings were so deformed as to be unrecognizable at this gridsize, but the drawings for SQUARE-30 and SQUARE-60 are still plagued significantly by the zigzagging problem.

Gridsize of 0.1 Inches. Interestingly, SQUARE-0, which had not been plagued by the zigzagging phenomenon at the larger gridsizes, now suffers from the zigzagging problem for every code using a gridsize of 0.1 inches. The zigzagging always occurs near the upper right corner of the drawing. Evidently, the original drawing is distorted enough to cause the chain coding algorithm to select a node one gridsize to the left or right of the present node (recall that the sides of the square are not exactly two inches in length). Generally, the drawings with the lowest aepl are still the multi-ring codes using

comes from the codes using the outer rings, such as the (2,4) and (3,4) codes shown in Figure B-19. This is true despite the fact that these codes possess higher aepl than the codes using the inner ring. At this gridsize, the lowest aepl ratings again come from the multi-ring codes using the (1) ring and the best bpl ratings come from the single ring outer level codes.

The drawings for SQUARE-45 were slightly more deformed at this gridsize than they were for the 0.2 inch gridsize. There was very little zigzagging but almost every code resulted in a small part of some of the corners chopped off. Again, however, the smoothest codes were the multi-ring codes using the (1) ring. Just like the drawings done with the 0.2 inch gridsize, all the codes using the (1) ring possessed the lowest aepl, and they were all equal. The most deformed drawings were the higher order codes, and they possessed the highest aepl. The bpl followed the normal pattern: highest for the multi-ring codes using the (1) ring and lowest for the high level single ring codes.

The smoothness for the SQUARE-60 drawings was again similar to the smoothness for the SQUARE-30 drawings. The high order (3,4) code shown in Figure B-20 was probably the smoothest drawing, although it was deformed by a missing corner and slight zigzagging. Again, this code was smoother than codes with a lower aepl because the zigzagging was more prevalent in the codes using the (1)

ratings followed the usual pattern, lowest for the outer level single ring codes and highest for the multi-ring codes containing the (1) ring.

At this gridsize, the smoothest drawings were for the SQUARE-45 codes. Evidently, the lengths of the sides of the square coincided with the nodes of the grid used by the chain coding algorithm well enough that no corners were lost, and the drawings appear almost perfect. The drawings for SQUARE-0 still suffer from chopped off corners; otherwise, they are very smooth. For the SQUARE-30 and SQUARE-60 drawings, the zigzagging caused by the lack of a node at 30 or 60 degrees is still the major problem. The codes with the higher rings tend to smooth this out to some extent, but they are more prone to chopped off corners.

Gridsize of 0.15 Inches. For SQUARE-0, the drawings coded with a gridsize of 0.15 inches were very smooth except for chopped off corners. The smoothest codes were the ones using the (1) ring. These codes also possessed the lowest aepl, since they have the least amount of corner chopped off. Just as for the drawings with gridsizes of 0.25 and 0.2 inches, all the codes using the (1) ring have essentially equal aepl ratings. Also, these codes have the highest bpl ratings; the lowest bpl again comes from the single ring outer level codes.

The SQUARE-30 drawings still suffer from chopped off corners and zigzagging. The least amount of zigzagging

resolution. The best aepl ratings again came from the multi-ring codes containing the (1) ring and the best bpl ratings came from the outer level single ring codes followed by the outer level multi-ring codes.

The drawings for SQUARE-45 at this gridsize are very smooth. All the codes containing the (1) ring produced drawings with no zigzagging and no chopped off corners. Also, all of these codes possess identical aepl ratings; therefore, except for the nodes used, the drawings are identical (see the (1) code shown in Figure B-17). All of the codes without the (1) ring possessed chopped off corners and some zigzagging. One, the (4) code, was so distorted as to be unrecognizable as a square. Smoothness and aepl correlated well for this drawing, with the smoothest drawings also possessing the least aepl. The highest bpl also belonged to the smoothest drawings.

As in the case of SQUARE-30, none of the codes for SQUARE-60 produced a drawing that was clearly superior to the others, nor were any of the drawings of acceptable quality. All of the drawings were distorted with chopped off corners and zigzagging. Again it appears that the outer ring codes are slightly smoother even though they possess higher aepl ratings. Their longer line segments between nodes do not suffer as much from the zigzagging as the codes containing the (1) ring. The (1) code shown in Figure B-18, with its poor angular resolution, was again the worst code with respect to smoothness. The bpl

Gridsize of 0.2 Inches. For SQUARE-0, the drawings were very smooth, with the only detraction being chopped off corners. All codes produced a drawing with the lower left corner chopped off and the codes without the (1) ring lost other corners as well. The (3), (4), and (3,4) codes were the ones with the most deformities; these three codes also had the worst aepl rating. The best aepl again came from the codes which use the (1) ring. Like the drawings at the 0.25 inch gridsize, the aepl ratings for the codes using the (1) ring were essentially equal. The bpl also followed the same pattern as the drawings with the 0.25 inch gridsize; the lowest bpl was for the codes with the outer rings and the highest bpl came from the multi-ring codes.

Again, the major detraction for the SQUARE-30 drawings was the zigzagging effect caused by the lack of a next node at 30 or 60 degrees. Losing corners was also a problem for these drawings at this gridsize. None of the drawings at this gridsize were of acceptable quality. There were no codes that produced drawings clearly superior to the others; however, the (2,3), (2,4), and (2,3,4) codes shown in Figure B-16 were slightly smoother than the rest. The (2,4) and (2,3,4) codes had mediocre aepl ratings, but the use of the outer rings resulted in less zigzagging for these drawings. The worst code was the (1) code (also shown in Figure B-16); it was again a victim of excessive zigzagging due to its limited angular

was not a very good drawing. Both sides were slightly crushed toward the middle, but it contained the fewest deformities of all the drawings. Most of the rest of the codes produced drawings containing chopped off corners and zigzagging. The worst codes were the (1), (1,3), and (1,4) codes shown in Figure B-15. These codes contained quite a bit more zigzagging than the others which detracted from their smoothness. Their aepl was mediocre and their bpl was high. The (3) and (4) codes were deformed but still recognizable as squares. Their aepl ratings reflected the deformation, as they were the highest of all the codes.

By far the smoothest drawings were for SQUARE-0. They contained no zigzagging and suffered no worse from chopped off corners than the other squares. The next smoothest were the SQUARE-45 drawings. This could be expected because there is always a node 45 degrees from the present node, while this is not true for 30 or 60 degrees. Since none of the rings produce a node that is 30 or 60 degrees from the present node, SQUARE-30 and SQUARE-60 suffered more from zigzagging lines as the coding algorithm tried to follow the sides of the squares and was never able to find a node that was on the line. This effect was less noticeable on the outer ring codes simply because they consist of longer line segments; therefore, they have fewer corrections to make.

zigzagging as the chain coding algorithm was required to compensate for its lack of angular resolution. Both the aepl and bpl ratings for this code were mediocre. The high level single ring codes, the (3) and (4) codes, were very distorted. The corners cut off by the coding algorithm have caused them to lose their identity as rotated squares. This is reflected by their aepl ratings, which are the two highest. The bpl ratings were highest for the multi-ring codes containing the (1) ring and lowest for the outer ring codes, just the opposite of the aepl ratings.

For the SQUARE-45 drawing, no particular drawing stands out as the smoothest; none of the drawings are of acceptable quality. All of the drawings possess some zigzagging or missing corners that detract from the expected shape. The worst codes with respect to the zigzagging are the (1), (1,3), (1,4), and (1,3,4) codes. Again, the high level (4) code and the (3,4) code are so distorted as to be unrecognizable as squares. Both of these codes possess very high aepl ratings, with the (4) code's aepl almost twice as high as the next highest aepl. The (3) code produced a relatively smooth drawing and possessed the lowest aepl rating, as well as a low bpl rating. In general, the codes using the (1) ring had the lowest aepl and the highest bpl.

The smoothest drawing for SQUARE-60 was produced by the (3,4) code shown in Figure B-14. Even this, though,

lost other corners as well. In this case, since the smoothest drawings are the ones with the least amount of the square chopped off, they also are the drawings with the lowest aepl. The bpl was exactly the opposite; the more corners that were missing, the fewer the number of nodes (and the corresponding number of bits) required to encode the square. Another interesting observation is that all the codes containing the (1) ring possess the same aepl rating. This means that although each code produced different nodes because of the different ring levels, they all produced the same drawing when the chain coded versions were plotted.

For SQUARE-30, the smoothness of many of the drawings was reduced because the chain coding algorithm produced a zigzagging effect. For example, the (1) code shown in Figure B-12 contains an excessive amount of zigzagging. Therefore, the smoothest drawings were the ones without any corners chopped off and those containing the least zigzagging. The best combination of these two attributes was the (3,4) code shown in Figure B-13. This code also possessed a low aepl and close to the best bpl rating. It was the only drawing with acceptable quality at this gridsize. Codes that were almost as smooth as the (3,4) code were the (1,2,3) and (1,2,3,4) codes. These codes possessed low aepl ratings but had high bpl ratings. By far the worst looking drawing was produced by the (1) code shown in Figure B-12. This drawing contained a lot of

there is error for each drawing and for every gridsize.

In general, the best aepl and the worst bpl performance came from the multi-ring codes that contained the (1) ring; however, none of these codes were consistently better than the others. Also, the high level, (3) and (4) ring codes provided both the worst aepl and best bpl performance.

No pattern could be discerned when comparing the aepl for the different squares; no square consistently had more or less aepl than the others. For the bpl, however, SQUARE-0 usually had the highest bpl and SQUARE-45 usually had the lowest. Evidently, the codes for SQUARE-45 were able to utilize the outer rings more effectively. Likewise, the codes for SQUARE-0 were not as effective in this respect as the codes for the other squares.

Smoothness vs aepl and bpl. In this section, the smoothness of the chain coded versions of the figures is evaluated examined for a correlation between the smoothness and the aepl and bpl metrics. For each gridsize, each square is examined and then the squares are compared against each other to determine the effect of the rotation.

Gridsize of 0.25 Inches. For the SQUARE-0 drawings coded at 0.25 inches, the only detractions from smoothness are the chopped off corners of the square. Each code chopped off the lower left corner of the square and the higher order codes, the (3), (4), and (3,4) codes,

squares are shown in Figure B-10.

In general, the relationship between aepl and gridsize was not as linear as it was for the circle. This suggests that another factor besides gridsize contributes to the aepl. For a non-rotated square, or any figure consisting solely of straight lines running parallel to the coding grid, it seems obvious that the aepl should be more a function of the ratio of the length of the sides of the figure to the product of the ring level and the gridsize. Also, for a rotated square or similar figure, the aepl should be a function of how close the lines lie to the nodes of the coding grid. For example, a perfect square with sides that are an exact multiple of the gridsize encoded with a (1) code should have an aepl of zero. If the perfect square were not an exact multiple of the gridsize, the aepl would consist of the sum of the areas of small rectangles and triangles like those shown in Figure B-11.

This fact is demonstrated by the aepl data for SQUARE-0 at the gridsize of 0.15 inches. This is the only gridsize evaluated that is not a sub-multiple of the two inch length of the sides of the square. The data shows that the aepl of almost every code is higher for the 0.15 inch gridsize than for the other gridsizes.

Of course, none of the squares used here are perfect squares, as they were hand drawn. The sides of these squares are not exactly two inches in length; therefore,

gridsizes. This suggests that a radius to gridsize ratio of approximately 15 to 1 will result in an adequate representation of a circle. These results are similar to those obtained by Thompson in his study of circular waves (6:V-4).

Squares

The next figures examined were squares with sides approximately two inches in length (recall these figures were hand drawn and are not geometrically perfect). Four different squares were digitized and coded, each with a different angle of rotation relative to a Cartesian coordinate system. The angles of rotation examined were 0, 30, 45, and 60 degrees. The different angles of rotation were used to determine the effect of rotation on the aepl and bpl metrics and the smoothness. The same codes and gridsizes used for the CIRCLE drawing were used for each of the four squares. The performance of the chain codes for each square are examined relative to the results for the other squares to determine the effect of the rotation.

Trends in aepl and bpl Performance. The square at an angle of 0 degrees is referred to as SQUARE-0, the square rotated 30 degrees is known as SQUARE-30, the square rotated 45 degrees is SQUARE-45, and the one rotated 60 degrees is referred to as SQUARE-60. The performance results of the chain codes by gridsize are shown in Tables B-6 through B-25 and the digitized versions of each of the

with the small variation in smoothness between drawings. Again, the placement of the nodes affects the smoothness more than a high or low bpl metric.

Summary of Data for CIRCLE. The best aepl performance generally came from the (1,3), (1,2,3), (1,2,4), and the (1,3,4) codes, while the worst performance was for the (2,3), (2,4), and (3,4) codes. The best performing codes for the bpl metric were naturally the outer ring, single level codes. For the circle, it was noted at all gridsizes that the outer ring codes were smoother than the codes including the inner rings, even though the aepl was generally less for the inner ring codes. This is true because an abrupt change in slope usually results when the coding algorithm, trying to follow the curvature of the circle, switches from an outer ring to an inner one. The conclusion is that aepl is not a good indicator of smoothness for the circle. Also, it was clear that a high bpl is not related to the smoothness. In fact, many times the code with the lowest bpl, the (4) code, was one of the smoothest drawings. In all cases, the placement of the nodes, which determines how symmetrical the circle appears, had more effect on the smoothness than the bpl did.

The smoothness of the drawings was not good at gridsizes higher than 0.15 inches. Also, once this gridsize was reached, there was not as much variation in smoothness between codes as there was at the larger

those coded at 0.15 inches. The (3) and (4) codes were again the smoothest and the (1) code was the roughest. The rest of the drawings were essentially equal; all containing two or three of the abrupt changes in direction that detract from the circular appearance. Codes (1,4) and (3,4) shown in Figure B-8 are typical.

As with all previous gridsizes, the (3) and (4) codes are the smoothest drawings and possess high aepl ratings. The roughest code, the (1) code, also had a high aepl. The drawings at this gridsize did not show much variation in aepl, with the difference between the highest and the lowest only approximately 0.008 inches. This fact correlates well with the low variation in smoothness. Except for the smoothest code, the (3) code, and the worst, the (1) code, these drawings were very close to each other in smoothness. The bpl metric again showed no relevance to smoothness, other than its relationship to the placement of the nodes.

Gridsize of 0.05 Inches. All of the drawings coded with a gridsize of 0.05 inches were of good quality. The (1) and (2) codes suffered somewhat from abrupt changes in slope; however, the gridsize is small enough that it is not as significant a problem as at the larger gridsizes. The (4) code shown, in Figure B-9, is typical of the drawings at this gridsize.

The aepl for these drawings varied less than 0.003 inches between maximum and minimum. This correlates well

corners were not chopped off as badly. The bpl ratings followed the usual pattern.

The characteristics of the SQUARE-60 drawings are again very similar to those of the SQUARE-30 drawings. Although the corners are not chopped off as badly at this gridsize, the zigzagging is still a significant problem. The drawings least effected by the zigzagging are still the ones using the higher order codes, while the lowest aepl ratings are produced by the multi-ring codes using the (1) ring. The bpl ratings followed the same pattern as they have for the larger gridsizes.

The smoothest drawings were again for SQUARE-45. The SQUARE-0 drawings had deteriorated because of the introduction of zigzagging, which had been absent for the larger gridsizes. SQUARE-30 and SQUARE-60 were about equal in smoothness; both still suffering significant deformation because of zigzagging. The corner chopping problem has been vastly reduced for all the squares at this gridsize.

Gridsize of 0.05 Inches. The drawings for SQUARE-0 using a gridsize of 0.05 inches are very smooth. The drawings using the (1) ring are the smoothest, with only a small part of one of the corners missing. The drawings using the outer rings are missing a small amount more of the corners, but these drawings are still very good. The smoothness of all of these drawings is reflected in the aepl ratings. The (1) and (1,2) codes

have the lowest aepl while the (3) code has the highest; however, less than 0.002 inches separates the maximum aepl from the minimum (these drawings are shown in Figure B-23). The relative bpl performance of the codes is unchanged from the higher gridsizes.

The drawings for SQUARE-30 are much improved over the larger gridsizes. In general, the zigzagging problem, though still noticeable, does not detract from the appearance of the square as much as at the larger gridsizes. Also, the smoothness of the codes does not vary as widely; there is no class of codes that is clearly superior to the others. However, the (1) code shown in Figure B-24 is again clearly the worst of the drawings. Its limited angular resolution resulted in significant zigzagging, even though it had a low aepl rating. The bpl again was lowest for the outer ring codes and highest for the inner ring codes.

The SQUARE-45 drawings for this gridsize are very smooth; however, some zigzagging is now noticeable that had been missing from the drawings coded at larger gridsizes. The zigzagging is very minor and only occurs in one or two places on each drawing. It seems that the gridsize is now small enough to cause the drawing to be affected by the inaccuracies in the slope of the lines of the original hand drawn square. In other words, where the slope of the original drawing was not exactly 45 degrees, the nodes of the chain code grid are now close enough to

each other to result in the selection of a node not 45 degrees from the present node. The smoothest drawings, the (1,2), (2,3), and (2,3,4) drawings shown in Figure B-25, also possessed the lowest aepl ratings. The drawings with the highest aepl were the outer ring codes, which were also the most distorted. The bpl ratings followed the usual pattern of lowest for the single ring high level codes and highest for the multi-ring codes.

Like the drawings for SQUARE-30, the drawings for SQUARE-60 are much improved at this gridsize. The zigzagging is much less significant and most of the drawings are very smooth. The (1,2) code produced the smoothest drawing; however, it is not significantly better than the (2,4) and (3,4) drawings (these drawings are shown in Figure B-26). The aepl for the (1,2) code was the lowest of all the codes, while the aepl for the (2,4) and (3,4) codes was neither among the highest or lowest when compared to the rest of the codes. The worst drawings were the (1) and (2,3,4) codes shown in Figure B-27). These codes contained more zigzagging than the other drawings, which detracted from their smoothness. The aepl and bpl ratings followed the familiar pattern set at the larger gridsizes of low aepl and high bpl for the multi-ring codes and high aepl and low bpl for the single ring outer level codes.

Because of the zigzagging in the drawings for SQUARE-45, the smoothest drawings at this gridsize were for the

SQUARE-0 drawings. The drawings for SQUARE-30 and SQUARE-60 were very much improved with the zigzagging much less noticeable than it had been at larger gridsizes. Also, this gridsize is small enough that the problem of chopped off corners is not as significant, though it is still noticeable in drawings that do not use the (1) ring.

Summary of Data for Squares. Generally, the drawings for SQUARE-0 and SQUARE-45 were much smoother than the drawings for SQUARE-30 and SQUARE-60. This is probably attributable to the fact that a node can always be found 0 or 45 degrees from the present node for any chain code, while no node is exactly 30 or 60 degrees from a present node. This makes the zigzagging problem more likely to occur for lines at an angle of 30 or 60 degrees. The other major problem noted for all the squares was loss of some of the corners. All of the squares suffered from this problem to some extent; however, it was most noticeable on SQUARE-0, since it was generally the only deformity suffered by that square. Usually, a low aepl for SQUARE-0 and SQUARE-45 indicated that the drawing would be smooth, while this was not true for SQUARE-30 and SQUARE-60. Many times for SQUARE-30 and SQUARE-60, the smoothest drawings were the ones using the outer rings, such as the (2,4) and (3,4) codes, the codes with mediocre or high aepl ratings. The zigzagging was less apparent for these codes because of the greater distance between nodes. It appears that the (1) ring is necessary for the

corners, but detracts from overall smoothness. So, similar to the circle, the conclusion is that low aepl does not necessarily indicate smoothness; smoothness is more a function of the placement of the nodes.

Sine Wave

The next drawing digitized and encoded was two cycles of a sine wave. The period of the sine wave is approximately 1.5 inches and it has a peak of approximately 1.625 inches. This drawing is shown in Figure B-28. The same codes and gridsizes used for the previous drawings were used for the sine wave.

Trends in aepl and bpl Performance. This section analyzes the aepl and bpl data for the sine wave to see if any trends in performance patterns for these metrics develop. The performance data for each of the codes is shown in Tables B-26 through B-30.

Generally, an increase in gridsize leads to an increase in the aepl rating; however, the relationship was not as close to being linear as it was for the circle. The bpl relationship to 1/gridsize is as highly linear as it was for the circle and the square.

The outer ring codes, the (3), (4), and (3,4) codes, consistently provided the worst aepl performance and the best bpl performance. The best aepl performance was for the multi-ring codes containing the inner, (1) ring, while these codes also provided the worst bpl performance. The (1,2,3,4) code's aepl rating was lower than the other

codes for all gridsizes except 0.1 inches, where it was the next to the lowest. The performance of the other codes containing the (1) ring varied over the gridsizes enough that no conclusion could be drawn as to which one was the best. The best bpl performance came from the codes using the outer rings, the (2), (3), (4), and (3,4) codes. This is the same pattern noted for the other drawings.

Smoothness vs aepl and bpl. In this section, the smoothness of the chain coded versions of the sine wave is evaluated and an attempt is made to correlate the smoothness with the aepl and bpl metrics. The data is examined for each of the five gridsizes in the following paragraphs.

Gridsize of 0.25 Inches. None of the drawings coded with a gridsize of 0.25 inches were of good quality. The major problems are chopping off at the peaks of the sine wave and zigzagging along the straight portions. Also, the (3) and (4) codes were extremely distorted because they could not adequately follow the sine wave due to the relatively large gridsize. The smoothest drawings were for the (1,2,3,4), (1,2,4), and (1,3,4) multi-ring codes. The smoothest drawing, the (1,2,3,4) code, shown in Figure B-29, possessed the lowest aepl rating while the worst two drawings, the (3) and (3,4) codes, possessed the highest aepl. The (1) code suffered the most from the zigzagging, just as it did for the SQUARE-30 and SQUARE-60

drawings; it also had a mediocre aepl rating.

Gridsize of 0.2 Inches. The quality of the drawings coded with a gridsize of 0.2 inches was also very poor. The (1,2,3,4) code again produced the smoothest drawing. Other relatively smooth drawings were the (1,2), (1,2,3), and (1,2,4) codes. The aepl ratings of these codes were generally low. The (3) and (4) codes were still very distorted at this gridsize. The (1) code shown in Figure B-30, with a mediocre aepl ratings, is one of the worst drawings because of excessive zigzagging. The multi-ring codes seem effective in smoothing out a large part of the zigzagging at this gridsize.

Gridsize of 0.15 Inches. The quality of the drawings coded with a gridsize of 0.15 inches was still poor for most codes. Chopped off peaks and zigzagging are still significant problems. Again, the (1) code, with its limited angular resolution, suffered greatly from zigzagging. The smoothest codes were the (1,2,3,4), (1,3), and (1,4) codes (the (1,4) code is shown in Figure B-31). These codes possessed relatively low aepl ratings, except for the (1,2,3) code, which was in the middle of the grouping. The (4) code was still very distorted at this gridsize; this was reflected in its high aepl rating.

Gridsize of 0.1 Inches. Except for the (1) code, shown in Figure B-32, the zigzagging for the codes at a gridsize of 0.1 inches is not as severe as it was at the larger gridsizes. Also, the chopping off of the peaks

is only noticeable on the outer ring codes, such as the (3,4) code shown in Figure B-33. Overall, the quality of these drawings was an improvement over the drawings at the larger grid sizes. In general, the smoothest codes were those using some combination of the (1) ring and the outer, (3) or (4), rings. The worst codes were those with the peaks chopped off, the (3), (4), and (3,4), and the (1) code, which still is plagued by zigzagging. The aepl was generally lower for the smooth codes (those containing the (1) and an outer ring) than for the rough ones. These codes still possessed the highest aepl ratings at this grid size.

Gridsize of 0.05 Inches. With a grid size of 0.05 inches, most of the codes produced extremely smooth representations of the sine wave. The zigzagging problem still exists for the (1) code, see Figure B-34; however, it is much less objectionable at this small grid size. The smoothness of the other drawings does not vary significantly; the (2,3,4) code shown in Figure B-35 is typical. The aepl ratings were lowest for the codes using the (2) ring (recall that the digitizer resolution limits the codes that can be used at this grid size).

Summary of Data for the Sine Wave. In general, the best aepl performance came from the codes which use a combination of the (1) ring and the outer, (3) and (4), rings. Although the aepl increased with grid size for every code, the relationship was not linear. On the other

hand, the relationship between bpl and 1/gridsizes seems to be very linear, just as it was for the circle and the squares.

The smoothest codes for every gridsizes were the ones using the (1) ring in combination with one of the outer rings, such as the (1,3,4) and the (1,2,3,4) codes. These drawings also had the lowest aepl ratings. At the higher gridsizes, the major problems were zigzagging and chopping off the peaks of the sine wave; these problems were not nearly as noticeable at the lower (0.1 and 0.05 inch) gridsizes. The (1) ring suffered the most from the zigzagging problem and consistently had a relatively high aepl rating.

The bpl metric for the sine wave followed the pattern established by the circle and the squares. That is that the bpl is lowest for the codes using the outer rings, such as the (3), (4), and (3,4) codes, and highest for the multi-ring codes using the (1) ring.

Written Text

The last drawing to be digitized and coded was a sample of written text. The word "hello" was chosen because it possesses a nice blend of long sloping lines and curves with both a large and small radius of curvature. The digitized drawing, hereafter referred to as TEXT, is shown in Figure B-36. The letter "h" in "hello" is approximately one inch high and the letters "e"

and "o" are approximately 1/2 inch high. Each of the codes and grid sizes used for the other drawings were used for TEXT.

For all of the coded versions of TEXT, the coded drawing consists of a small number of line segments which cross each other one or more times before intersecting with the digitized drawing. This creates multiple closed loops between intersections of the digitized and coded drawings which the area algorithm in the ERROR program cannot handle (see page IV-13 in reference 2). For this reason, the error results are not tabulated. Therefore, the performance metrics discussed in the remaining paragraphs of this section of the chapter are smoothness and readability of the text.

Grid sizes of 0.25 and 0.2 Inches. The drawings coded with a grid size of 0.25 inches were unrecognizable. The (1,3) code shown in Figure B-37 is an example of one of the better codes, and it is unreadable. The drawing becomes recognizable as a written word with a grid size of 0.2 inches, but only for codes that use the (1) ring. All of the drawings at this grid size are extremely jagged and angular; there is no smoothness evident. Codes (1,3) and (1,2,3,4), shown in Figure B-38, are representative examples.

Grid size of 0.15 Inches. With a grid size of 0.15 inches, a noticeable difference in the quality of the codes emerges. However, the codes that do not use the (1)

ring are still unrecognizable. All of the drawings are still jagged and angular, but the utility of the multi-ring codes in smoothing the drawing can easily be seen by comparing the (1) and the (1,2,3,4) codes in Figure B-39. It seems that the best drawings are the ones that use any combination of the (1) and (2) rings. The curves in the letters are too small to take advantage of the outer rings, the (3) and (4) rings; therefore, the (1,3) and (1,4) codes look very similar to the (1) code shown in Figure B-39.

Gridsize of 0.1 Inches. At a gridsize of 0.1 inches, the drawings with the (2) ring as the lowest ring are now recognizable; however, the drawings using the higher rings are still unreadable. The (1), (1,3), and (1,4) codes still produced drawings at this gridsize that are jagged and angular, while the (1,2), (1,2,3), (1,2,4), and (1,2,3,4) codes are beginning to appear smooth. The (1,2,4) code shown in Figure B-40 is representative of these drawings.

Gridsize of 0.05 Inches. With a gridsize of 0.05 inches, all of the codes produced recognizable drawings (recall that several of the codes cannot be used because of the digitizer resolution). The (1) and (4) codes shown in Figure B-41 are the worst codes, each for a different reason. The (1) code is jagged and angular because of its limited angular resolution while the (4) code just cannot adequately follow the original digitized

drawing. The (2,4) and (2,3,4) codes shown in Figure B-41 are the smoothest drawings. These codes utilize the inner ring and the outer ring to provide a good combination for encoding the various radii of curvature encountered in the drawing.

Summary of Data for TEXT. The TEXT drawing needed a small gridsize to be readable. With a gridsize of 0.25 inches, the drawing was unrecognizable, and only the codes using the (1) ring produced recognizable drawings at gridsizes of 0.2 and 0.15 inches. At the gridsizes above 0.1 inches, all the drawings were very jagged and angular and of unacceptable quality. With a gridsize of 0.1 inches, the drawings began to become smoother, with the codes using the (1) and (2) rings performing the best. However, the drawings at this gridsize were still judged to be of unacceptable quality. All of the drawings coded with a gridsize of 0.05 inches were recognizable as written words; however, the quality of the (1) and (4) codes was unacceptable, and only marginal for the other codes. These results indicate that the minimum acceptable letter height to gridsize ratio is approximately 20 to 1 for marginal quality.

For all gridsizes except 0.05 inches, the codes that use a combination of the (1) and an outer ring, the (3) or (4) rings, did not perform well. Apparently, the twists and curves in the text did not allow the outer rings to be used, and the (1) ring's limited angular resolution

produced very jagged letters. Therefore, the best performing codes were those that used the (1) and (2) rings with some added combination of the outer rings, such as the (1,2,3) code.

Summary

In this chapter, the results of an analysis of several line drawings have been discussed. This analysis was based on the performance of several chain codes at different gridsizes using the precision, compactness, and smoothness performance criteria. The next chapter will discuss the conclusions that have been reached as a result of this analysis and propose recommendations for future study.

VI. Conclusions and Recommendations

This chapter summarizes the conclusions reached as a result of the analysis of the performance of the chain codes. Also, recommendations for future study are made.

Conclusions

Generally, the aepl metric was lowest for the multi-ring codes containing the (1) ring. These codes usually possessed aepl ratings that were very close to each other; however, none of these codes were consistently better than the others. The aepl was consistently highest for the outer level (3) and (4) codes. The multi-ring codes that do not use the (1) ring, such as the the (2,3) and (3,4) codes, generally had a high or mediocre aepl rating. The single ring codes, including the (1) code, consistently had a high aepl rating compared to the multi-ring codes. Another general observation is that the aepl usually increases with gridsize for every code; however, this did not occur in every case.

No definite conclusions can be drawn from these drawings as to which code is the best to use to obtain a low aepl rating. Though no particular code had the lowest aepl rating at every gridsize for any of the drawings, a general conclusion, however, can be drawn. As mentioned earlier, the multi-ring codes that use the (1) ring

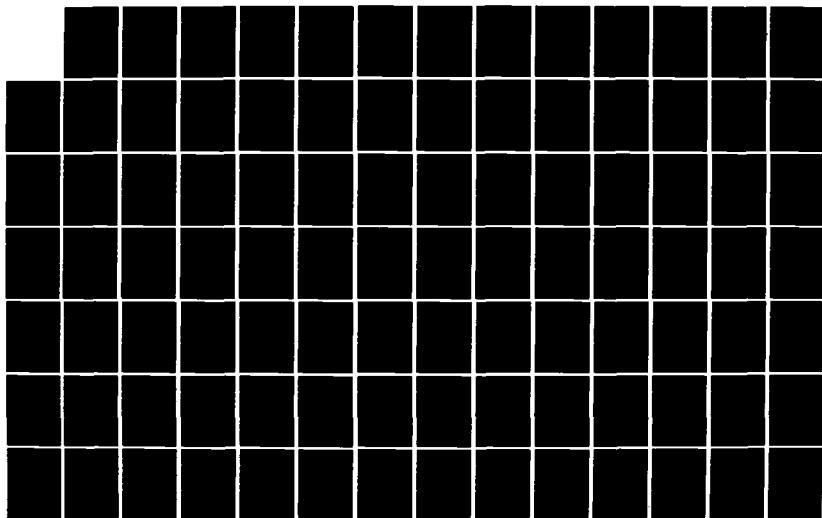
AD-A152 008

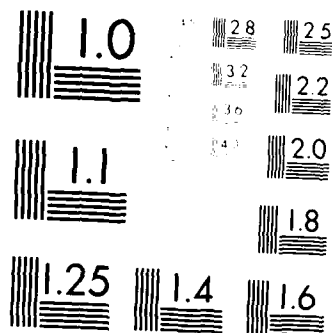
AN EXTENSION OF A MICROCOMPUTER BASED SYSTEM FOR
ANALYSIS OF LINE DRAWING. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI. T A MORRIS
DEC 84 AFIT/GE/ENG/84D-48 F/G 9/2

2/3

UNCLASSIFIED

NL





MICROCOPY RESOLUTION TEST CHART
 NATIONAL BUREAU OF STANDARDS-1963-A

generally had aepl ratings that were very close. Although none of these codes were consistently better than the others, the use of any of these codes will almost certainly result in a relatively low aepl rating.

The (2), (3), (4), and (3,4) codes consistently had the lowest bpl ratings for all drawings, while the codes using the (1) ring usually had the highest bpl ratings. The multi-ring codes using the (2), (3), and (4) rings had mediocre bpl ratings. The bpl ratings were inversely proportional to the gridsize for all codes. In fact, this relationship was very close to linear for the gridsizes and codes analyzed in this thesis. Therefore, if the bpl for any code is known for one gridsize, it can be predicted for the other gridsizes with reasonable accuracy.

It seems that the codes with the best combination of aepl and bpl performance are the multi-ring codes that use the (2), (3), and (4) rings, such as the (2,3,4) code. The performance of these codes was generally mediocre for both metrics; however, since the strong performing codes for both metrics are mutually exclusive (the best codes for aepl are the worst ones for bpl and vice versa), these codes are the best compromise.

The results of the smoothness evaluation were largely dependent upon the drawing being analyzed. A general observation for all drawings is that a low aepl rating is not a guarantee of a smooth drawing. The codes using the

high order rings, which generally possessed the highest aepl ratings, sometimes resulted in smoother drawings because there was less zigzagging caused by constant corrections as the coding algorithm tried to follow the original drawing. Also, it was evident that the placement of the nodes was more important than a low aepl for providing a smooth drawing. In other words, if the nodes of the coded drawing were symmetrically spaced and followed the shape of the drawing well, this was more visually pleasing than a drawing that contained abrupt changes in slope or direction, but possessed a low aepl rating.

Although the results of the smoothness evaluations were drawing dependent, general conclusions can be drawn on which class of codes should be used to provide the smoothest drawings. If the drawing to be coded consists mostly of curves with a fairly constant radius of curvature, such as the CIRCLE drawing used in this analysis, then the high order codes are generally smoother, even though they have the highest aepl ratings. Also, these codes worked best for straight lines at an angle relative to the coding grid that does not pass through any nodes, such as the SQUARE-30 and SQUARE-60 drawings used in this analysis. The higher order codes produce less zigzagging for this type of drawing. For random drawings or those with a wide range of radii of curvature, such as the sine wave in this analysis, the

smoothest codes were those using the inner, (1), ring in combination with one or more outer rings.

Another aspect of coding a drawing is the selection of the gridsize used for the encoding process. This analysis also provides some insight into this area. The coded versions of the circle used in this analysis were of unacceptable quality at gridsizes above 0.15 inches. This indicates that for drawings with a constant radius of curvature, that a radius to gridsize ratio of approximately 15 to 1 will result in acceptable drawings if the proper codes are used. For the drawings containing lines of arbitrary slope, such as the rotated squares, no general conclusions could be reached identifying a line length to gridsize ratio that would provide adequate performance in every situation. The coded versions of the written text used in this analysis were unacceptable at gridsizes above 0.05 inches. Therefore, a letter height to gridsize ratio of approximately 20 to 1 is indicated.

Of course, the selection of the code and gridsize used are extremely application dependent, and, as expected, there are trade-offs between aepl, bpl, and smoothness. The application dictates which parameter is the most important; however, these results can provide guidelines to aid in the selection of the code and gridsize that provide the best compromise for a particular type of drawing.

Recommendations for Future Study

The following recommendations are made for future study with this system:

1. Implement the parallel quantizing scheme. This would be a relatively simple change to the COMPCODE procedure of the CHNCODE program.
2. Implement a procedure in the ERROR program to determine the statistics of the usage of the different ring levels.
3. Using the statistics gained from recommendation 2, implement a node encoding procedure that minimizes the number of bits required to encode the node based on the probability of that node being used.
4. Implement a procedure in the CHNCODE program that simulates the effect of an additive white Gaussian noise communications channel at various bit error rates and study the effect on the drawings.

APPENDIX A

This appendix contains the Pascal source code for the CHNCODE and the PLOTCODE multi-ring chain coding programs.

This is the source listing for the multi-ring chain coding program CHNCODE.

PROGRAM CHNCODE;

```
(*
(* WRITTEN BY: THOMAS A. MORRIS
(* DATE: 15 AUG 1984
(* THE PURPOSE OF THIS PROGRAM IS TO CONVERT A DRAWING
(* REPRESENTED BY A SET OF X-Y COORDINATES INTO A
(* MULTI-RING CHAIN CODED VERSION OF THE DRAWING.
(* THE OUTPUT FILE WILL CONTAIN THE STARTING X-Y
(* COORDINATE, GRID SIZE, AND THE RING LEVELS USED FOR
(* THE CODE. THE OUTPUT FILE FORMAT IS: PEN UP/DOWN,
(* X COORDINATE, Y COORDINATE, CHAIN CODE. A CHAIN
(* CODE OF -1 INDICATES THAT THE POINT IS THE FIRST
(* OR LAST IN A LINE SEGMENT. THE INPUT FILE NEED
(* ONLY BE A LIST OF X-Y COORDINATES WITH EACH POINT
(* SEPARATED BY CARRIAGE RETURN/LINEFEED.
```

```
TYPE POINTER = ^RINGPNTS;
   RINGPNTS = RECORD
       LOCATION : ARRAY [1..2] OF INTEGER;
       NEXT : POINTER
   END; (* RECORD *)
```

```
(* THIS POINTER AND RECORD ARE USED TO FORM A LINKED
(* LIST TO STORE THE POINTS AS THEY ARE READ IN FROM
(* THE DIGITIZED FILE. LOCATION IS THE X-Y
(* COORDINATE OF THE POINT AND NEXT IS A POINTER
(* VARIABLE WHICH POINTS TO THE NEXT SEQUENTIAL
(* POINT IN THE LIST
```

VAR

```
F, G: TEXT;
A, RESULT, J, NUMFILES, OUTRING: INTEGER;
CODE: EXTERNAL INTEGER;
FILENAME: STRING;
NODE: EXTERNAL ARRAY [1..2] OF INTEGER;
GRIDSIZE, NUMRINGS: ARRAY [1..25] OF INTEGER;
LEVEL: ARRAY [1..25,1..5] OF INTEGER;
CODEFILE: ARRAY [1..25] OF STRING;
DELTA: ARRAY [1..5] OF INTEGER;
FIRST, FINISH: BOOLEAN;
PEN: CHAR;
POINT: ARRAY [1..3,1..2] OF INTEGER;
INTSCT: ARRAY [1..5] OF BOOLEAN;
DELTAXY, INTPT: ARRAY [1..5,1..2] OF REAL;
LASTPNT: ARRAY [1..5,1..4] OF INTEGER;
FRONT, BACK, PTR: POINTER;
```

```

(*) THESE VARIABLES ARE DEFINED AS FOLLOWS: (*)
(*)
(*) F,G:  THESE ARE THE NAMES USED FOR THE TEXT FILES (*)
(*) REPRESENTING THE INPUT (DIGITIZED) FILE AND (*)
(*) THE OUTPUT (CODED) FILE (*)
(*)
(*) A,J:  INTEGERS USED AS LOOP COUNTING VARIABLES (*)
(*)
(*) RESULT:  USED AS A STORAGE LOCATION FOR THE TEXT (*)
(*) FILE CLOSE PROCEDURE RETURN VALUE (*)
(*) (INTERNAL TO COMPILER) (*)
(*)
(*) NUMFILES:  THE NUMBER OF FILES TO BE CREATED BY (*)
(*) CODING THE DIGITIZED FILE (*)
(*)
(*) OUTRING:  OUTERMOST RING THAT HAS NOT FAILED THE (*)
(*) LGS TEST (*)
(*)
(*) FILENAME:  TEMPORARY VARIABLE TO HOLD NAMES OF THE (*)
(*) TEXT FILES FOR THE INTERNAL ASSIGN (*)
(*) PROCEDURE (*)
(*)
(*) GRIDSIZE:  ARRAY TO HOLD THE GRIDSIZE USED FOR (*)
(*) EACH FILE TO BE CODED (*)
(*)
(*) NUMRINGS:  ARRAY HOLDS THE NUMBER OF RING LEVELS (*)
(*) USED FOR EACH FILE TO BE CODED (*)
(*)
(*) LEVEL:  ARRAY HOLDS THE VALUES OF THE RING LEVELS (*)
(*) USED FOR EACH FILE TO BE CODED (*)
(*)
(*) CODEFILE:  ARRAY HOLDS THE NAMES OF EACH FILE TO (*)
(*) BE CODED (*)
(*)
(*) DELTA:  ARRAY HOLDS THE DISTANCE FROM THE CURRENT (*)
(*) NODE TO EACH RING OUT TO RING 5 (*)
(*)
(*) FIRST:  TRUE IF THE NEXT POINT READ FROM THE (*)
(*) DIGITIZED FILE IS THE FIRST POINT IN A (*)
(*) LINE SEGMENT (*)
(*)
(*) FINISH:  TRUE IF THE LAST POINT IN THE DIGITIZED (*)
(*) FILE HAS BEEN READ (*)
(*)
(*) PEN:  HOLDS A VALUE OF 'D' IF THE PEN IS DOWN AND (*)
(*) A VALUE OF 'U' IF THE PEN IS UP (*)
(*)
(*) POINT:  [1] IS THE X-Y COORDINATES OF THE CURRENT (*)
(*) NODE, [2] AND [3] ARE THE COORDINATES OF (*)
(*) THE LAST TWO POINTS READ IN FROM THE (*)
(*) DIGITIZED FILE (*)

```

```

(*)  INTSCT:  HOLDS A VALUE OF TRUE FOR EVERY RING      *)
(*)           INTERSECTED                               *)
(*)                                                    *)
(*)  DELTAXY:  HOLDS THE DIFFERENCE BETWEEN THE GRID    *)
(*)             CENTER AND THE INTERSECTION POINT FOR   *)
(*)             EACH RING                               *)
(*)                                                    *)
(*)  INTPNT:  HOLDS THE INTERSECTION POINT COORDINATES  *)
(*)             FOR EACH RING THAT IS INTERSECTED      *)
(*)                                                    *)
(*)  LASTPNT:  HOLDS THE LAST TWO POINTS READ AFTER A  *)
(*)             RING IS INTERSECTED.                   *)
(*)                                                    *)
(*)  FRONT:  POINTER TO THE FIRST POINT IN THE LINKED  *)
(*)             LIST                                   *)
(*)                                                    *)
(*)  BACK:  POINTER TO THE LAST POINT IN THE LINKED    *)
(*)             LIST                                   *)
(*)                                                    *)
(*)  PTR:  POINTER TO THE CURRENT POINT IN THE LINKED  *)
(*)             LIST                                   *)
(*)                                                    *)

```

```

EXTERNAL PROCEDURE COMPCODE; (* THIS PROCEDURE PERFORMS *)
  (* THE LGS TEST AND CALCULATES THE VALUE OF THE      *)
  (* CHAIN CODE                                         *)

```

```

PROCEDURE INSERT; (* INSERTS POINTS INTO THE REAR OF  *)
  (* THE QUEUE                                         *)

```

```

VAR P: POINTER; (* P IS USED AS A TEMPORARY STORAGE  *)
  (* POINTER TO MANIPULATE PTR                       *)

```

```

BEGIN (* INSERT *)
  NEW(P);
  P^.LOCATION[1] := POINT[3,1];
  P^.LOCATION[2] := POINT[3,2];
  P^.NEXT := NIL;
  IF BACK = NIL
  THEN
    BEGIN
      BACK := P;
      FRONT := P
    END
  ELSE
    BEGIN
      BACK^.NEXT := P;
      BACK := P
    END;
  PTR := NIL
END; (* INSERT *)

```

```

PROCEDURE READLIST; (* THIS PROCEDURE READS POINTS IN *)
                    (* FROM THE LINKED LIST AND *)
                    (* UPDATES THE PTR POINTER *)

```

```

BEGIN (* READLIST *)
  POINT[3,1] := PTR^.LOCATION[1];
  POINT[3,2] := PTR^.LOCATION[2];
  PTR := PTR^.NEXT
END; (* READLIST *)

```

```

PROCEDURE ALIGN; (* ALIGNS LIST OF POINTS UP TO THE *)
                 (* CODED NODE ALSO UPDATES THE FRONT *)
                 (* PCINTER *)

```

```

VAR P: POINTER; (* P IS USED AS A TEMPORARY STORAGE *)
                (* POINTER TO MANIPULATE PTR *)

```

```

BEGIN (* ALIGN *)
  PTR := FRONT;
  IF PTR <> NIL THEN
    WHILE NOT ((LASTPNT[OUTRING,3] = PTR^.LOCATION[1]) AND
              (LASTPNT[OUTRING,4] = PTR^.LOCATION[2])) DO
      IF PTR <> NIL THEN
        BEGIN
          P := PTR;
          PTR := P^.NEXT;
          DISPOSE(P)
        END;
        FRONT := PTR
      END; (* ALIGN *)

```

```

PROCEDURE CLRLIST; (* CLEARS LIST OF ALL POINTS *)

```

```

VAR P: POINTER; (* P IS USED AS A TEMPORARY STORAGE *)
                (* POINTER TO MANIPULATE PTR *)

```

```

BEGIN (* CLRLIST *)
  PTR := FRONT;
  WHILE PTR <> NIL DO
    BEGIN
      P := PTR;
      PTR := P^.NEXT;
      DISPOSE(P)
    END;
    BACK := NIL
  END; (* CLRLIST *)

```

```

PROCEDURE STARTUP; (* PARAMETER INITIALIZATION *)

```

```

VAR K, L: INTEGER; (* K AND L ARE USED AS LOOP COUNTING *)
                    (* VARIABLES *)

BEGIN (* STARTUP *)
  WRITELN ('THIS PROGRAM COMPUTES MULTI-RING CHAIN CODES');
  WRITELN;
  WRITE('ENTER THE DIGITIZED DATA FILENAME : ');
  READLN(FILENAME);
  ASSIGN(F,FILENAME);
  WRITELN;
  WRITE ('ENTER THE NUMBER OF CODED FILES YOU WISH TO
    CREATE: ');
  READLN (NUMFILES);
  WRITELN;
  FOR K := 1 TO NUMFILES DO
    BEGIN (* FOR STATEMENT *)
      WRITE ('ENTER CODED DATA FILENAME ',K,' : ');
      READLN (FILENAME);
      CODEFILE[K] := FILENAME;
      WRITE ('ENTER GRIDSIZE DESIRED: ');
      READLN (GRIDSIZE[K]);
      WRITE ('ENTER THE NUMBER OF RINGS USED BY THE CODE: ');
      READLN (NUMRINGS[K]);
      WRITELN ('ENTER THE RINGS USED BY THE CODE STARTING
        WITH THE LOWEST');
      WRITE ('(EXAMPLE FOR A (1,3,5) CODE: ENTER 1 3 5): ');
      FOR L := 1 TO (NUMRINGS[K] - 1) DO
        READ (LEVEL[K,L]);
        READLN (LEVEL[K,NUMRINGS[K]]);
        WRITELN
      END (* FOR STATEMENT *)
    END; (* STARTUP *)

PROCEDURE INITIALIZE; (* FILE AND VARIABLE INITIALIZATION *)

VAR M: INTEGER; (* M IS A LOOP COUNTER VARIABLE *)

BEGIN (* INITIALIZE *)
  REPEAT
    WRITELN ('CODING ',CODEFILE[J]);
    FILENAME := CODEFILE[J];
    ASSIGN (G,FILENAME);
    RESET (F);
    REWRITE (G)
  UNTIL IORESULT <> 255;
  FOR M := 1 TO LEVEL[J,NUMRINGS[J]] DO
    DELTA[M] := GRIDSIZE[J] * M;
  FOR M := 1 TO 5 DO
    INTSCT[M] := FALSE;
  FINISH := FALSE;
  BACK := NIL
END; (* INITIALIZE *)

```

```

PROCEDURE FIRSTPNT; (* THIS PROCEDURE FINDS THE FIRST *)
                    (* PEN DOWN IN A DATA FILE AND *)
                    (* KEEPS THE LAST PEN UP AS THE *)
                    (* FIRST POINT IN THE CODED FILE *)

```

```

VAR NUMBER, N: INTEGER;

```

```

(* VARIABLE DEFINITIONS: *)
(* *)
(* NUMBER IS A COUNT OF THE NUMBER OF POINTS READ FROM *)
(* THE FILE TILL A PEN DOWN CONDITION IS READ *)
(* *)
(* N IS A LOOP COUNTING VARIABLE *)

```

```

BEGIN (* FIRSTPNT *)
  FIRST := FALSE;
  NUMBER := 0;
  REPEAT (* FIND THE FIRST PEN DOWN... KEEP LAST PEN UP, *)
    (* POINT[2], AND FIRST PEN DOWN, POINT[3] *)
    POINT[2] := POINT[3];
    READLN (F, PEN, POINT[3,1], POINT[3,2]);
    NUMBER := NUMBER + 1;
    FINISH := EOF(F)
  UNTIL (PEN = 'D') OR FINISH;
  INSERT;
  IF NOT FINISH
  THEN
    BEGIN (* IF STATEMENT *)
      IF NUMBER < 2 (* ? FIRST ELEMENT ALREADY PEN DOWN *)
      THEN
        BEGIN (* THEN CLAUSE *)
          POINT[2] := POINT[3];
          READLN (F, PEN, POINT[3,1], POINT[3,2]);
          INSERT
        END; (* THEN CLAUSE *)
        POINT[1] := POINT[2];
        POINT[3] := POINT[2];
        WRITE (G, 'U', ' ', POINT[1,1], ' ', POINT[1,2], ' -1 ',
          GRIDSIZE[J], ' ');
        FOR N := 1 TO (NUMRINGS[J] - 1) DO
          WRITE (G, LEVEL[J,N], ' ');
        WRITELN (G, LEVEL[J,NUMRINGS[J]])
      END (* IF STATEMENT *)
    END; (* FIRSTPNT *)

```

```

IF POINTS INTERSECT; (* THIS PROCEDURE FINDS THE *)
                    (* INTERSECT POINTS FOR EACH OF *)
                    (* THE RINGS OUT TO THE OUTERMOST *)
                    (* RING *)

```

```

VAR A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z: INTEGER;

```

```

(* VARIABLE DEFINITIONS: *)
(*)
(*) A IS A LOOP COUNTING VARIABLE *)
(*)
(*) DISTX AND DISTY ARE USED TO DETERMINE IF THE LAST *)
(*) POINT IS THE ONE THAT INTERSECTS A RING *)

PROCEDURE FINDINT; (* THIS PROCEDURE FINDS THE X-Y *)
                  (* COORDINATES OF THE ACTUAL *)
                  (* INTERSECTION POINT *)

VAR
  TEMP: REAL;
  SIGNX: INTEGER;

(* VARIABLE DEFINITIONS: *)
(*)
(*) TEMP AND SIGNX ARE USED AS INTERMEDIATE VARIABLES IN *)
(*) THE PROCESS OF FINDING THE INTERSECTION COORDINATES *)

BEGIN (* FINDINT *)
  INTSET[A] := TRUE; (* THE RING IS INTERSECTED BEFORE *)
                    (* EOF OR END OF LINE SEGMENT *)
  IF POINT[3,1] = POINT[2,1]
  THEN
    BEGIN (* THEN CLAUSE *)
      IF POINT[1,2] <= POINT[3,2]
      THEN DELTAXY[A,2] := DELTA[A]
      ELSE DELTAXY[A,2] := -1 * DELTA[A];
      INTENT[A,2] := POINT[1,2] + DELTAXY[A,2]
    END (* THEN CLAUSE *)
  ELSE
    BEGIN (* ELSE CLAUSE *)
      TEMP := (POINT[3,2] - POINT[2,2]) / (POINT[3,1] -
        POINT[2,1]);
      (* SLOPE OF DIGITIZED LINE CROSSING THE GRID *)
      IF POINT[3,1] >= POINT[2,1]
      THEN SIGNX := 1
      ELSE SIGNX := -1;
      TEMP := TEMP * (SIGNX * DELTA[A] - POINT[2,1] +
        POINT[1,1]);
      DELTAXY[A,2] := TEMP + POINT[2,2] - POINT[1,2];
      INTENT[A,2] := POINT[1,2] + DELTAXY[A,2];
      IF ABS (DELTAXY[A,2]) > DELTA[A]
      THEN (* INTERSECTS ON TOP OR BOTTOM OF GRID *)
        BEGIN (* THEN CLAUSE *)
          IF DELTAXY[A,2] > 0
          THEN DELTAXY[A,2] := DELTA[A]
          ELSE DELTAXY[A,2] := -1 * DELTA[A];
          INTENT[A,2] := POINT[1,2] + DELTAXY[A,2]
        END (* THEN CLAUSE *)
      END; (* ELSE CLAUSE *)
    END;
  IF (POINT[3,2] - POINT[2,2]) <= (ABS(DELTAXY[A,2]) *

```

```

    PLOTLINE[15] := ' '
END; (* PLOTIN *)

```

```

PROCEDURE INSERT; (* INSERTS POINTS INTO THE REAR OF      *)
                  (*   THE QUEUE                            *)

```

```

VAR P: POINTER; (* P IS USED AS A TEMPORARY STORAGE      *)
                (*   POINTER TO MANIPULATE PTR          *)

```

```

BEGIN (* INSERT *)
    NEW(P);
    P^.LOCATION[1] := POINT[3,1];
    P^.LOCATION[2] := POINT[3,2];
    P^.NEXT := NIL;
    IF BACK = NIL
    THEN
        BEGIN
            BACK := P;
            FRONT := P
        END
    ELSE
        BEGIN
            BACK^.NEXT := P;
            BACK := P
        END;
    PTR := NIL
END; (* INSERT *)

```

```

PROCEDURE READLIST; (* THIS PROCEDURE READS POINTS IN    *)
                   (*   FROM THE LINKED LIST AND          *)
                   (*   UPDATES THE PTR POINTER           *)

```

```

BEGIN (* READLIST *)
    POINT[3,1] := PTR^.LOCATION[1];
    POINT[3,2] := PTR^.LOCATION[2];
    PTR := PTR^.NEXT
END; (* READLIST *)

```

```

PROCEDURE ALIGN; (* ALIGNS LIST OF POINTS UP TO THE      *)
                 (*   CODED NODE                            *)

```

```

VAR P: POINTER; (* P IS USED AS A TEMPORARY STORAGE      *)
                (*   POINTER TO MANIPULATE PTR          *)

```

```

BEGIN (* ALIGN *)
    PTR := FRONT;
    IF PTR <> NIL THEN
        WHILE NOT (LASTENT[OUTRING,2] = PTR^.LOCATION[1] AND
                   LASTENT[OUTRING,4] = PTR^.LOCATION[2]) DO
            IF PTR <> NIL THEN
                BEGIN

```

```

(* SCALE:  SCALE FACTOR FOR PLOTTER                                *)
(*)                                                (*)
(* XYDATA:  X-Y COORDINATES SENT TO THE PLOTTER                    *)
(*)                                                (*)
(* TRANS:   TRANSLATION FACTOR FOR PLOTTER                          *)
(*)                                                (*)
(* KEYNUMBER:  VALUE REPRESENTS WHICH FUNCTION KEY ON              *)
(*)   THE DIGITIZER HAS BEEN DEPRESSED .                           *)
(*)                                                (*)

EXTERNAL PROCEDURE BUSIN (DEVICE:INTEGER; VAR ERFLAG:CHAR;
  VAR INLINE:LINE);
EXTERNAL PROCEDURE BUSOUT (DEVICE:INTEGER; VAR ERFLAG:CHAR;
  VAR OUTLINE:LINE);
EXTERNAL FUNCTION BUSINT:CHAR;
EXTERNAL PROCEDURE PORTIN;
EXTERNAL PROCEDURE CHAROT (VAR OUTPUT:CHAR; VAR ERFLAG:CHAR);
EXTERNAL PROCEDURE CHARIN (VAR SIGNAL:CHAR; VAR ERFLAG:CHAR);
EXTERNAL PROCEDURE LINOUT (VAR CHARAC:LINE; VAR ERFLAG:CHAR);
EXTERNAL PROCEDURE GET_POINT (VAR DATA_LINE:LINE; VAR ERFLAG:
  CHAR; VAR KEYNUMBER:INTEGER);
EXTERNAL FUNCTION PREPARE:CHAR;
EXTERNAL PROCEDURE WHATNUM (VAR CHARAC:LINE; VAR LENGTH, NUMB:
  INTEGER);
EXTERNAL PROCEDURE WHATCHAR (X:INTEGER; VAR LINEIN:LINE; VAR
  CNTR, LENGTH:INTEGER);
EXTERNAL PROCEDURE SAMPLING (VAR ERFLAG:CHAR);
EXTERNAL PROCEDURE CALCODE;

PROCEDURE PLOTIN; (* THIS PROCEDURE INITIALIZES THE                *)
                  (* PLOTTER                                         *)
                  (*)

BEGIN (* PLOTIN *)
  WRITE('ENTER PLOTTER LINE TYPE (0-8): ');
  READLN(LINE_TYPE);
  PLOTLINE[14] := LINE_TYPE;
  WRITE('ENTER SCALING FACTOR (DIG:PLOT)- 1:');
  READLN(SCALE);
  WRITE('ENTER PLOTTER TRANSLATION (X Y): ');
  READLN(TRANS[1],TRANS[2]);
  WRITELN;
  PLOTLINE[1] := ',';
  PLOTLINE[2] := ':';
  PLOTLINE[3] := 'I';
  PLOTLINE[4] := ' ';
  PLOTLINE[5] := '0';
  PLOTLINE[6] := 'D';
  PLOTLINE[7] := ' ';
  PLOTLINE[8] := '0';
  PLOTLINE[9] := ' ';
  PLOTLINE[10] := 'A';
  PLOTLINE[11] := 'U';
  PLOTLINE[12] := ' ';
  PLOTLINE[13] := 'L';

```

The rest of this appendix contains the source code listing for the PLOTCODE program.

```
(* THIS PROGRAM COMPUTES THE CHAIN CODE OF A DRAWING AS *)
(* IT IS BEING DRAWN ON THE DIGITIZER AND IMMEDIATELY *)
(* SENDS THE DATA POINTS TO THE PLOTTER FOR PLOTTING *)

PROGRAM PLOTCODE;

TYPE POINTER = ^RINGPNTS;
   RINGPNTS = RECORD
       LOCATION : ARRAY [1..2] OF INTEGER;
       NEXT : POINTER
   END; (* RECORD *)
   LINE = ARRAY [1..40] OF CHAR;

VAR
   A, RESULT, OUTRING, PLOT CNT: INTEGER;
   COUNTER, LOOP, CNTNUM, LENGTH, I: INTEGER;
   CODE: EXTERNAL INTEGER;
   NODE: EXTERNAL ARRAY [1..2] OF INTEGER;
   GRIDSIZE, NUMRINGS: INTEGER;
   LEVEL, DELTA: ARRAY [1..5] OF INTEGER;
   FIRST, FINISH: BOOLEAN;
   PEN, ERROR_FLAG, SIGNAL, LINE_TYPE, ERFLAG: CHAR;
   POINT: ARRAY [1..3,1..2] OF INTEGER;
   INTSCT: ARRAY [1..5] OF BOOLEAN;
   DELTAXY, INTPNT: ARRAY [1..5,1..2] OF REAL;
   LASTPNT: ARRAY [1..5,1..4] OF INTEGER;
   FRONT, BACK, PTR: POINTER;
   NUMARRAY, DATA, PLOTLINE: LINE;
   TEMP, SCALE: REAL;
   XYDATA, TRANS: ARRAY [1..2] OF INTEGER;
   KEYNUMBER: EXTERNAL INTEGER;

(* ONLY THE VARIABLES THAT ARE UNIQUE TO PLOTCODE ARE *)
(* DEFINED HERE. THE REMAINDER OF THE VARIABLES ARE *)
(* DEFINED IN THE CHNCODE PROGRAM LISTING *)
(* *)
(* PLOT CNT: COUNTER VARIABLE TO KEEP TRACK OF THE *)
(* VALUES IN THE PLOTLINE ARRAY *)
(* *)
(* CNTNUM: HOLDS THE NUMBER OF DIGITS READ IN FROM THE *)
(* DIGITIZER *)
(* *)
(* LENGTH: USED AS A PARAMETER FOR THE WHATCHAR *)
(* PROCEDURE TO TELL THE PROCEDURE HOW MANY *)
(* CHARACTERS TO CONVERT THE NUMBER TO *)
(* *)
(* PLOTLINE: THIS ARRAY CONTAINS THE INFORMATION SENT *)
(* TO THE PLOTTER FOR PLOTTING *)
(* *)
(* TEMP: USED AS A TEMPORARY STORAGE VARIABLE *)
```

```

        CODE := CODEB
    END (* THEN CLAUSE *)
ELSE CODE := CODEB
END; (* ENCODE *)

BEGIN (* COMPCODE *)
    IF OUTRING = LEVEL[J,1]
    THEN
        BEGIN
            FINDNODE;
            ENCODE
        END
    ELSE
        BEGIN (* ELSE CLAUSE *)
            REPEAT
                FINDNODE;
                PASSLGS := FALSE;
                RINGTEST;
                IF PASSLGS
                THEN ENCODE
                ELSE
                    BEGIN (* ELSE CLAUSE *)
                        VALUE := 1;
                        REPEAT
                            OUTER := LEVEL[J, NUMRINGS[J] - VALUE];
                            VALUE := VALUE + 1
                        UNTIL OUTER < OUTRING;
                        OUTRING := OUTER
                    END (* ELSE CLAUSE *)
                UNTIL (OUTRING = LEVEL[J,1]) OR PASSLGS;
                IF OUTRING = LEVEL[J,1]
                THEN
                    BEGIN
                        FINDNODE;
                        ENCODE
                    END
                END; (* ELSE CLAUSE *)
            POINT[3,1] := LASTPNT[OUTRING,1];
            POINT[3,2] := LASTPNT[OUTRING,2];
            POINT[1,1] := NODE[1];
            POINT[1,2] := NODE[2]
        END; (* COMPCODE *)

MODEND. (* MODULE COMPCODE *)

```

This concludes the program listing for CHNCODE.

```

        IF (NODE[2] - POINT[1,2]) = DELTA[OUTRING]
        THEN U_L_CORNER
        ELSE
            IF (POINT[1,2] - NODE[2]) = DELTA[OUTRING]
            THEN L_L_CORNER
            ELSE
                BEGIN (* ELSE CLAUSE *)
                    RIGHT := FALSE;
                    SIDE
                END (* ELSE CLAUSE *)
            ELSE (* NODE IS ON TOP OR BOTTOM *)
                BEGIN
                    IF (NODE[2] - POINT[1,2]) = DELTA[OUTRING]
                    THEN TOP := TRUE
                    ELSE TOP := FALSE;
                    TOP_OR_BOT;
                END
            END; (* RINGTEST *)

```

```

PROCEDURE ENCODE; (* THIS PROCEDURE CALCULATES THE          *)
                  (* CHAIN CODE                               *)

```

```

VAR RING, CODEB, I: INTEGER;

```

```

(* VARIABLE DEFINITIONS:                                     *)
(*                                                             *)
(* RING AND I ARE LOOP COUNTING VARIABLES                     *)
(*                                                             *)
(* CODEB IS THE VALUE OF THE CHAIN CODE FOR A NODE FOR      *)
(* A SINGLE RING CHAIN CODE                                  *)

```

```

BEGIN (* ENCODE *)
    IF XCODE >= YCODE
    THEN CODEB := XCODE + YCODE
    ELSE CODEB := OUTRING * 8 - XCODE - YCODE;
    IF CODEB < (3 * OUTRING)
    THEN CODEB := CODEB + OUTRING * 5
    ELSE CODEB := CODEB - OUTRING * 3;
    IF NUMRINGS[J] > 1
    THEN
        BEGIN (* THEN CLAUSE *)
            FOR I := 1 TO NUMRINGS[J] DO
                IF OUTRING = LEVEL[J,I] THEN RING := I;
            FOR I := 1 TO (RING - 1) DO
                BEGIN (* FOR STATEMENT *)
                    CASE LEVEL[J,I] OF
                        1 : CODEB := CODEB + 8;
                        2 : CODEB := CODEB + 16;
                        3 : CODEB := CODEB + 24;
                        4 : CODEB := CODEB + 32
                    END (* CASE STATEMENT *)
                END; (* FOR STATEMENT *)
            END
        END
    END

```

```

        ELSE PASSLGS := FALSE;
        COUNT := COUNT + 1
    UNTIL (NOT PASSLGS) OR (COUNT >= OUTRING)
END; (* TOP_OR_BOT *)

```

```

PROCEDURE SIDE; (* THIS PROCEDURE PERFORMS THE LGS TEST *)
    (* WHEN THEN NODE IS ON EITHER THE *)
    (* LEFT OR RIGHT SIDE OF THE RING *)

```

```

BEGIN (* SIDE *)
    MIDPNT[1,1] := NODE[1];
    MIDPNT[1,2] := NODE[2] + GRIDSIZE[J] / 2;
    MIDPNT[2,1] := NODE[1];
    MIDPNT[2,2] := NODE[2] - GRIDSIZE[J] / 2;
    COUNT := 1;
    REPEAT
    (* ESTABLISH UPPER BOUND *)
        M := (MIDPNT[1,2] - POINT[1,2]) / (MIDPNT[1,1] -
            POINT[1,1]);
        IF RIGHT
            THEN X := NODE[1] - GRIDSIZE[J] * COUNT
            ELSE X := NODE[1] + GRIDSIZE[J] * COUNT;
        UPPER := M * (X - MIDPNT[1,1]) + MIDPNT[1,2];
    (* ESTABLISH LOWER BOUND *)
        M := (MIDPNT[2,2] - POINT[1,2]) / (MIDPNT[2,1] -
            POINT[1,1]);
        LOWER := M * (X - MIDPNT[2,1]) + MIDPNT[2,2];
        IF (INTPNT[OUTRING - COUNT, 2] <= UPPER) AND (INTPNT
            [OUTRING - COUNT, 2] >= LOWER) AND (ABS(X - INTPNT
            [OUTRING - COUNT, 1]) < 1)
            THEN PASSLGS := TRUE
            ELSE PASSLGS := FALSE;
        COUNT := COUNT + 1
    UNTIL (NOT PASSLGS) OR (COUNT >= OUTRING)
END; (* SIDE *)

```

```

BEGIN (* RINGTEST *)
    IF (NODE[1] - POINT[1,1]) = DELTA[OUTRING]
        THEN (* NODE IS ON RIGHT SIDE--MAY BE A CORNER *)
            IF (NODE[2] - POINT[1,2]) = DELTA[OUTRING]
                THEN U_R_CORNER
            ELSE
                IF (POINT[1,2] - NODE[2]) = DELTA[OUTRING]
                    THEN L_R_CORNER
                ELSE (* NODE IS ON THE RIGHT SIDE *)
                    BEGIN
                        RIGHT := TRUE;
                        SIDE
                    END
            ELSE
                IF (POINT[1,1] - NODE[1]) = DELTA[OUTRING]
                    THEN (* NODE IS ON LEFT SIDE--MAY BE A CORNER *)

```

```

BEGIN
  MIDPNT[1,1] := NODE[1] - GRIDSIZE[J] / 2;
  MIDPNT[1,2] := NODE[2];
  MIDPNT[2,1] := NODE[1];
  MIDPNT[2,2] := NODE[2] - GRIDSIZE[J] / 2;
  COUNT := 1;
  REPEAT
    TEMPNODE[1] := NODE[1] - GRIDSIZE[J] * COUNT;
    TEMPNODE[2] := NODE[2] - GRIDSIZE[J] * COUNT;
  (* ESTABLISH X BOUNDARY *)
    M := (MIDPNT[1,2] - POINT[1,2]) / (MIDPNT[1,1] -
      POINT[1,1]);
    XBOUND := MIDPNT[1,1] + (TEMPNODE[2] - MIDPNT[1,2]) / M;
  (* ESTABLISH Y BOUNDARY *)
    M := (MIDPNT[2,2] - POINT[1,2]) / (MIDPNT[2,1] - POINT[1,1]);
    YBOUND := M * (TEMPNODE[1] - MIDPNT[2,1]) + MIDPNT[2,2];
    NUMBER1 := INTPTNT((OUTRING - COUNT),1);
    NUMBER2 := INTPTNT((OUTRING - COUNT),2);
    IF (NUMBER1 <= TEMPNODE[1]) AND (NUMBER1 >= XBOUND) AND
      (NUMBER2 <= TEMPNODE[2]) AND (NUMBER2 >= YBOUND)
      THEN PASSLGS := TRUE
      ELSE PASSLGS := FALSE;
    COUNT := COUNT + 1
  UNTIL (NOT PASSLGS) OR (COUNT >= OUTRING)
END; (* U_R_CORNER *)

```

```

PROCEDURE TOP_OR_BOT; (* THIS PROCEDURE PERFORMS THE *)
  (* LGS TEST WHEN THE NODE IS ON *)
  (* THE TOP OR BOTTOM OF THE RING*)

```

```

BEGIN
  MIDPNT[1,1] := NODE[1] - GRIDSIZE[J] / 2;
  MIDPNT[1,2] := NODE[2];
  MIDPNT[2,1] := NODE[1] + GRIDSIZE[J] / 2;
  MIDPNT[2,2] := NODE[2];
  COUNT := 1;
  REPEAT
  (* ESTABLISH LEFT BOUND *)
    M := (MIDPNT[1,2] - POINT[1,2]) / (MIDPNT[1,1] -
      POINT[1,1]);
    IF TOP
      THEN Y := NODE[2] - GRIDSIZE[J] * COUNT
      ELSE Y := NODE[2] + GRIDSIZE[J] * COUNT;
    LFT := MIDPNT[1,1] + (Y - MIDPNT[1,2]) / M;
  (* ESTABLISH RIGHT BOUND *)
    M := (MIDPNT[2,2] - POINT[1,2]) / (MIDPNT[2,1] -
      POINT[1,1]);
    R := MIDPNT[2,1] + (Y - MIDPNT[2,2]) / M;
    IF (INTPTNT[OUTRING - COUNT,1] <= R) AND (INTPTNT[OUTRING -
      COUNT,1] >= LFT) AND (ABS(Y - INTPTNT[OUTRING -
      COUNT,2]) < 1)
      THEN PASSLGS := TRUE

```

```

NUMBER1 := INTPT[(OUTRING - COUNT),1];
NUMBER2 := INTPT[(OUTRING - COUNT),2];
IF (NUMBER1 <= TEMPNODE[1]) AND (NUMBER1 >= XBOUND) AND
   (NUMBER2 <= YBOUND) AND (NUMBER2 >= TEMPNODE[2])
  THEN PASSLGS := TRUE
  ELSE PASSLGS := FALSE;
COUNT := COUNT + 1
UNTIL (NOT PASSLGS) OR (COUNT >= OUTRING)
END; (*L_R_CORNER *)

```

```

PROCEDURE U_L_CORNER; (* THIS PROCEDURE IS USED TO      *)
                      (* THE LGS TEST WHEN THE NODE     *)
                      (* IS ON THE UPPER LEFT CORNER    *)
                      (* OF THE RING                     *)

```

```

VAR NUMBER1, NUMBER2: REAL; (* NUMBER1 AND NUMBER2 ARE *)
                           (* INTERMEDIATE VARIABLES *)

```

```

BEGIN
  MIDPNT[1,1] := NODE[1] + GRIDSIZE[J] / 2;
  MIDPNT[1,2] := NODE[2];
  MIDPNT[2,1] := NODE[1];
  MIDPNT[2,2] := NODE[2] - GRIDSIZE[J] / 2;
  COUNT := 1;
  REPEAT
    TEMPNODE[1] := NODE[1] + GRIDSIZE[J] * COUNT;
    TEMPNODE[2] := NODE[2] - GRIDSIZE[J] * COUNT;
    (* ESTABLISH X BOUNDARY *)
    M := (MIDPNT[1,2] - POINT[1,2]) / (MIDPNT[1,1] -
      POINT[1,1]);
    XBOUND := MIDPNT[1,1] + (TEMPNODE[2] - MIDPNT[1,2]) / M;
    (* ESTABLISH Y BOUNDARY *)
    M := (MIDPNT[2,2] - POINT[1,2]) / (MIDPNT[2,1] -
      POINT[1,1]);
    YBOUND := M * (TEMPNODE[1] - MIDPNT[2,1]) + MIDPNT[2,2];
    NUMBER1 := INTPT[(OUTRING - COUNT),1];
    NUMBER2 := INTPT[(OUTRING - COUNT),2];
    IF (NUMBER1 <= XBOUND) AND (NUMBER1 >= TEMPNODE[1]) AND
       (NUMBER2 <= TEMPNODE[2]) AND (NUMBER2 >= YBOUND)
      THEN PASSLGS := TRUE
      ELSE PASSLGS := FALSE;
    COUNT := COUNT + 1
  UNTIL (NOT PASSLGS) OR (COUNT >= OUTRING)
END; (* U_L_CORNER *)

```

```

PROCEDURE U_R_CORNER; (* THIS PROCEDURE PERFORMS THE  *)
                      (* LGS TEST WHEN THE NODE IS CN *)
                      (* THE UPPER RIGHT CORNER OF     *)
                      (* THE RING                       *)

```

```

VAR NUMBER1, NUMBER2: REAL; (* NUMBER1 AND NUMBER2 ARE *)
                           (* INTERMEDIATE VARIABLES *)

```

```

BEGIN (* L_L_CORNER *)
  MIDPNT[1,1] := NODE[1];
  MIDPNT[1,2] := NODE[2] + GRIDSIZE[J] / 2;
  MIDPNT[2,1] := NODE[1] + GRIDSIZE[J] / 2;
  MIDPNT[2,2] := NODE[2];
  COUNT := 1;
  REPEAT
    TEMPNODE[1] := NODE[1] + GRIDSIZE[J] * COUNT;
    TEMPNODE[2] := NODE[2] + GRIDSIZE[J] * COUNT;
  (* ESTABLISH X BOUNDARY *)
    M := (MIDPNT[2,2] - POINT[1,2]) / (MIDPNT[2,1] -
      POINT[1,1]);
    XBOUND := MIDPNT[2,1] + (TEMPNODE[2] - MIDPNT[2,2]) / M;
  (* ESTABLISH Y BOUNDARY *)
    M := (MIDPNT[1,2] - POINT[1,2]) / (MIDPNT[1,1] -
      POINT[1,1]);
    YBOUND := M * (TEMPNODE[1] - MIDPNT[1,1]) + MIDPNT[1,2];
    NUMBER1 := INTPT[(OUTRING - COUNT),1];
    NUMBER2 := INTPT[(OUTRING - COUNT),2];
    IF (NUMBER1 <= XBOUND) AND (NUMBER1 >= TEMPNODE[1]) AND
      (NUMBER2 <= YBOUND) AND (NUMBER2 >= TEMPNODE[2])
      THEN PASSLGS := TRUE
      ELSE PASSLGS := FALSE;
    COUNT := COUNT + 1
  UNTIL (NOT PASSLGS) OR (COUNT >= OUTRING)
END; (* L_L_CORNER *)

```

```

PROCEDURE L_R_CORNER; (* THIS PROCEDURE PERFORMS THE *)
  (* LGS TEST WHEN THE NODE IS ON *)
  (* THE LOWER RIGHT CORNER OF *)
  (* THE RING *)

```

```

VAR NUMBER1, NUMBER2: REAL; (* NUMBER1 AND NUMBER2 ARE *)
  (* INTERMEDIATE VARIABLES *)

```

```

BEGIN (* L_R_CORNER *)
  MIDPNT[1,1] := NODE[1];
  MIDPNT[1,2] := NODE[2] + GRIDSIZE[J] / 2;
  MIDPNT[2,1] := NODE[1] - GRIDSIZE[J] / 2;
  MIDPNT[2,2] := NODE[2];
  COUNT := 1;
  REPEAT
    TEMPNODE[1] := NODE[1] - GRIDSIZE[J] * COUNT;
    TEMPNODE[2] := NODE[2] + GRIDSIZE[J] * COUNT;
  (* ESTABLISH X BOUNDARY *)
    M := (MIDPNT[2,2] - POINT[1,2]) / (MIDPNT[2,1] -
      POINT[1,1]);
    XBOUND := MIDPNT[2,1] + (TEMPNODE[2] - MIDPNT[2,2]) / M;
  (* ESTABLISH Y BOUNDARY *)
    M := (MIDPNT[1,2] - POINT[1,2]) / (MIDPNT[1,1] -
      POINT[1,1]);
    YBOUND := M * (TEMPNODE[1] - MIDPNT[1,1]) + MIDPNT[1,2];

```

```

(* XCODE AND YCODE ARE INTERMEDIATE VARIABLES USED IN      *)
(* THE PROCESS OF FINDING THE CHAIN CODE VALUE              *)
(*                                                         *)
(* TEMP IS AN INTERMEDIATE VARIABLE USED THROUGHOUT THE    *)
(* PROCEDURE                                                *)

PROCEDURE FINDNODE; (* THIS PROCEDURE FINDS THE NODE      *)
                   (* CLOSEST TO THE RING                *)
                   (* INTERSECTION POINT                  *)

BEGIN (* FINDNODE *)
  TEMP := (DELTA[X][OUTRING,1] + DELTA[OUTRING]) / GRIDSIZE[J];
  XCODE := ROUND(TEMP);
  TEMP := (DELTA[X][OUTRING,2] + DELTA[OUTRING]) / GRIDSIZE[J];
  YCODE := ROUND(TEMP);
  NODE[1] := POINT[1,1] + (XCODE - OUTRING) * GRIDSIZE[J];
  NODE[2] := POINT[1,2] + (YCODE - OUTRING) * GRIDSIZE[J]
END; (* FINDNODE *)

PROCEDURE RINGTEST; (* THIS PROCEDURE DETERMINES WHERE    *)
                   (* THE INTERSECTION POINT IS AT        *)
                   (* UNDER TEST                          *)

VAR
  RIGHT, TOP: BOOLEAN;
  MIDPNT: ARRAY [1..2,1..2] OF REAL;
  TEMPNODE: ARRAY [1..2] OF INTEGER;
  M, XBOUND, YBOUND, LFT, R, UPPER, LOWER: REAL;
  X, Y: INTEGER;

(* VARIABLE DEFINITIONS: *)
(* *)
(* RIGHT IS TRUE IF THE NODE IS ON THE RIGHT SIDE OF      *)
(* THE RING                                                *)
(* *)
(* TOP IS TRUE IF THE NODE IS ON THE TOP OF THE RING     *)
(* *)
(* MIDPNT IS AN ARRAY HOLDING THE X-Y COORDINATES OF     *)
(* THE GRID MIDPOINTS SURROUNDING A NODE--USED FOR      *)
(* LGS TEST                                               *)
(* *)
(* TEMPNODE, M, XBOUND, YBOUND, LFT, R, UPPER, LOWER,    *)
(* X, AND Y ARE INTERMEDIATE VARIABLES IN THE LGS       *)
(* TESTING PROCESS                                        *)

PROCEDURE L_L_CORNER; (* THIS PROCEDURE PERFORMS THE LGS *)
                     (* TEST IF THE NODE IS ON THE LOWER *)
                     (* LEFT CORNER OF THE RING           *)

VAR NUMBER1, NUMBER2: REAL; (* NUMBER1 AND NUMBER2 ARE *)
                           (* INTERMEDIATE VARIABLES *)

```

```

                                CLRLIST
                                END (* PEN OF U *)
                                END (* CASE STATEMENT *)
                                UNTIL FIRST OR FINISH
                                END (* IF NOT FINISH THEN CLAUSE *)
                                END; (* WHILE LOOP *)
                                CLOSE(G,RESULT)
                                END; (* FOR STATEMENT *)
                                WRITELN;
                                WRITELN ('COMPUTATIONS COMPLETE')
                                END. (* MAIN PROGRAM *)

```

```

MODULE COMPCODE;

```

```

VAR

```

```

    DELTAXY, INTPNT: EXTERNAL ARRAY[1..5,1..2] OF REAL;
    DELTA: EXTERNAL ARRAY[1..5] OF INTEGER;
    OUTRING, J: EXTERNAL INTEGER;
    NUMRINGS, GRIDSIZE: EXTERNAL ARRAY[1..25] OF INTEGER;
    NODE: ARRAY[1..2] OF INTEGER;
    POINT: EXTERNAL ARRAY[1..3,1..2] OF INTEGER;
    LEVEL: EXTERNAL ARRAY[1..25,1..5] OF INTEGER;
    CODE: INTEGER;
    LASTPNT: EXTERNAL ARRAY[1..5,1..4] OF INTEGER;

```

```

(* VARIABLE DEFINITIONS: *)
(*)
(* NODE IS AN ARRAY CONTAINING THE X-Y COORDINATES OF *)
(* THE NODE UNDER INVESTIGATION FOR ENCODING *)
(*)
(* CODE IS THE VALUE OF THE CHAIN CODE FOR THE NODE *)
(*)

```

```

PROCEDURE COMPCODE; (* THIS PROCEDURE IS CALLED WHEN *)
                    (* THE RING INTERSECTIONS ARE *)
                    (* FOUND. IT CALCULATES THE *)
                    (* CHAIN CODE FOR EACH LINK OF *)
                    (* THE DRAWING *)
                    (*)

```

```

VAR

```

```

    PASSLGS: BOOLEAN;
    COUNT, VALUE, OUTER, XCODE, YCODE: INTEGER;
    TEMP: REAL;

```

```

(* VARIABLE DEFINITIONS: *)
(*)
(* PASSLGS IS TRUE IF THE NODE PASSES THE LGS TEST *)
(*)
(* COUNT, VALUE, AND OUTER ARE USED AS COUNTING *)
(* VARIABLES TO KEEP TRACK OF WHICH RING IS BEING *)
(* TESTED *)
(*)

```

```

CLRLIST
END; (* DONE *)

```

```

BEGIN (* MAIN PROGRAM *)
  STARTUP;
  FOR J := 1 TO NUMFILES DO
    BEGIN (* FOR STATEMENT *)
      INITIALIZE;
      WHILE NOT FINISH DO
        BEGIN (* WHILE LOOP *)
          FIRSTPNT;
          IF NOT FINISH THEN
            BEGIN (* THEN CLAUSE *)
              PTR := FRONT;
              REPEAT
                INTERSECT;
                CASE PEN OF
                  'E' : BEGIN
                    FOR A := 1 TO NUMRINGS[J] DO
                      IF INTSCT[LEVEL[J,A]] THEN
                        OUTRING := LEVEL[J,A];
                      IF INTSCT[LEVEL[J,1]] THEN
                        BEGIN
                          COMPCODE;
                          ALIGN;
                          WRITELN(G,'D',' ',POINT[1,1],
                                ' ',POINT[1,2],' ',CODE)
                        END;
                      DONE;
                      CLRLIST;
                      FINISH := TRUE
                    END; (* PEN OF E *)
                  'D' : BEGIN
                    OUTRING := LEVEL[J,NUMRINGS[J]];
                    COMPCODE;
                    ALIGN;
                    WRITELN (G,PEN,' ',POINT[1,1],' ',
                          POINT[1,2],' ',CODE);
                  END;
                  'U' : BEGIN
                    FOR A := 1 TO NUMRINGS[J] DO
                      IF INTSCT[LEVEL[J,A]] THEN
                        OUTRING := LEVEL[J,A];
                      IF INTSCT[LEVEL[J,1]] THEN
                        BEGIN
                          COMPCODE;
                          WRITELN(G,'D',' ',POINT[1,1],
                                ' ',POINT[1,2],' ',CODE);
                          ALIGN
                        END;
                      DONE;
                      FIRST := TRUE;

```

```

UNTIL (DISTX >= 0) OR (DISTY >= 0) OR ((PTR = NIL) AND
  FINISH) OR ((PTR = NIL) AND (PEN = 'U'));
IF (DISTX >= 0) OR (DISTY >= 0)
  THEN
    BEGIN
      REPEAT
        FINDINT;
        LAST;
        A := A + 1;
        DISTX := ABS(POINT[3,1] - PCINT[1,1]) - DELTA[A];
        DISTY := ABS(POINT[3,2] - POINT[1,2]) - DELTA[A]
      UNTIL (A >= OUTRING) OR ((DISTX < 0) AND
        (DISTY < 0));
      IF A <= OUTRING
        THEN
          IF (DISTX >= 0) OR (DISTY >= 0)
            THEN
              BEGIN
                FINDINT;
                LAST
              END
            ELSE A := A - 1
          END
        ELSE INTSCT[A] := FALSE;
      IF FINISH THEN PEN := 'E'
      UNTIL A >= OUTRING
    END; (* INTERSECT *)

```

```

PROCEDURE DONE; (* THIS PROCEDURE IS CALLED WHEN *)
(* INTERSECT HAS RETURNED WITH A PEN *)
(* VALUE OF 'E' OR 'U', MEANING THAT *)
(* THE END OF THE DRAWING HAS BEEN *)
(* REACHED BEFORE THE LAST RING WAS *)
(* INTERSECTED *)

```

```

VAR A: INTEGER; (* A IS A LOOP COUNTING VARIABLE *)

```

```

BEGIN (* DONE *)
  REPEAT
    PTR := FRONT;
    INTERSECT;
    FOR A := 1 TO LEVEL[J,NUMRINGS[J]] DO
      IF INTSCT[A] THEN OUTRING := A;
      IF INTSCT[LEVEL[J,1]] THEN
        BEGIN (* IF INTSCT *)
          COMPCODE;
          ALIGN;
          WRITELN(G,'D',' ',POINT[1,1],' ',POINT[1,2],
            ' ',CODE)
        END; (* IF INTSCT *)
      UNTIL PTR = NIL;
      WRITELN(G,'D',' ',POINT[3,1],' ',POINT[3,2],' -1');

```

```

    DELTA[A])
  THEN (* INTERSECTS ON LEFT OR RIGHT SIDE *)
    BEGIN (* THEN CLAUSE *)
      IF POINT[1,1] <= POINT[3,1]
        THEN DELTAXY[A,1] := DELTA[A]
        ELSE DELTAXY[A,1] := -1 * DELTA[A];
      INTPNT[A,1] := POINT[1,1] + DELTAXY[A,1]
    END (* THEN CLAUSE *)
  ELSE
    BEGIN (* ELSE CLAUSE *)
      TEMP := (POINT[3,1] - POINT[2,1]) / (POINT[3,2] -
        POINT[2,2]);
      TEMP := TEMP * (DELTAXY[A,2] - POINT[2,2] +
        POINT[1,2]);
      DELTAXY[A,1] := TEMP + POINT[2,1] - POINT[1,1];
      INTPNT[A,1] := POINT[1,1] + DELTAXY[A,1]
    END (* ELSE CLAUSE *)
  END; (* FINDINT *)

PROCEDURE LAST; (* THIS PROCEDURE STORES THE LAST TWO *)
                (* POINTS READ WHEN A RING IS *)
                (* INTERSECTED *)
                *)

BEGIN (* LAST *)
  LASTPNT[A,1] := POINT[2,1];
  LASTPNT[A,2] := POINT[2,2];
  LASTPNT[A,3] := POINT[3,1];
  LASTPNT[A,4] := POINT[3,2]
END; (* LAST *)

BEGIN (* INTERSECT *)
  OUTRING := LEVEL[J,NUMRINGS[J]];
  PTR := FRONT;
  A := 0;
  REPEAT
    IF NUMRINGS[J] = 1 THEN A := OUTRING ELSE A := A + 1;
  REPEAT
    IF PTR <> NIL
      THEN
        BEGIN (* THEN CLAUSE *)
          POINT[2] := POINT[3];
          READLIST
        END (* THEN CLAUSE *)
      ELSE
        IF NOT FINISH AND (PEN <> 'U') THEN
          BEGIN (* IF NOT FINISH AND PEN <> U *)
            POINT[2] := POINT[3];
            READLN (F, PEN, POINT[3,1], POINT[3,2]);
            FINISH := EOF(F);
            INSERT
          END; (* IF NOT FINISH AND PEN <> U *)
        DISTX := ABS(POINT[3,1]-POINT[1,1])-DELTA[A];
        DISTY := ABS(POINT[3,2]-POINT[1,2])-DELTA[A]

```

```

        P := PTR;
        PTR := P^.NEXT;
        DISPOSE(P)
    END;
    FRONT := PTR
END; (* ALIGN *)

PROCEDURE STARTUP; (* PARAMETER INITIALIZATION *)

VAR L: INTEGER; (* L IS USED AS A LOOP COUNTING VARIABLE *)

BEGIN (* STARTUP *)
    WRITELN ('THIS PROGRAM PLOTS A MULTI-RING CHAIN CODED
        DRAWING AS THE POINTS ARE DIGITIZED');
    WRITELN;
    WRITE('ENTER THE GRIDSIZE DESIRED: ');
    READLN(GRIDSZ);
    WRITE('ENTER THE NUMBER OF RINGS USED BY THE CODE: ');
    READLN(NUMRINGS);
    WRITELN('ENTER THE RINGS USED BY CODE STARTING WITH THE
        LOWEST');
    WRITE('(EXAMPLE FOR A (1,3,5) CODE: ENTER 1 3 5): ');
    FOR L := 1 TO (NUMRINGS - 1) DO
        READ(LEVEL[L]);
    READLN(LEVEL[NUMRINGS]);
    WRITELN;
    FOR L := 1 TO LEVEL[NUMRINGS] DO
        DELTA[L] := GRIDSZ * L;
    FOR L := 1 TO 5 DO
        INTSCT[L] := FALSE;
    FINISH := FALSE;
    BACK := NIL;
    PORTIN;
    PLOTIN;
    ERROR_FLAG := PREPARE;
    SAMPLING(ERROR_FLAG);
    WRITELN('NOW BEGIN TAKING POINTS');
    WRITELN
END; (* STARTUP *)

PROCEDURE READ_DIG; (* THIS PROCEDURE READS THE          *)
                    (* INCOMING DATA FROM THE          *)
                    (* DIGITIZER                        *)

VAR LOOP, COUNTER: INTEGER; (* LOOP AND COUNTER ARE      *)
                            (* COUNTER VARIABLES        *)

BEGIN (* READ_DIG *)
    ERFLAG := '-';
    REPEAT
        KEYNUMBER := 0;
        GET_POINT(DATA, ERROR_FLAG, KEYNUMBER);

```

```

IF KEYNUMBER <> 0 THEN
  BEGIN
    IF KEYNUMBER = 3 THEN PEN := 'D';
    IF KEYNUMBER = 2 THEN FINISH := TRUE;
    IF KEYNUMBER >= 4 THEN PEN := 'U'
  END;
UNTIL (KEYNUMBER = 0) OR (ERROR_FLAG = 'E');
IF ERROR_FLAG <> 'E'
THEN
  BEGIN (* IF ERROR_FLAG STATEMENT *)
    COUNTER := 1;
    FOR LOOP := 1 TO 2 DO
      BEGIN (* FOR STATEMENT *)
        CNTNUM := 1;
        REPEAT
          NUMARRAY[CNTNUM] := DATA[COUNTER];
          CNTNUM := CNTNUM + 1;
          COUNTER := COUNTER + 1;
        UNTIL DATA[COUNTER] = ',';
        COUNTER := COUNTER + 1;
        CNTNUM := CNTNUM - 1;
        WHATNUM(NUMARRAY, CNTNUM, PRINT[3,LOOP]);
      END (* FOR STATEMENT *)
    END (* IF THEN *)
  ELSE (* IF ERROR_FLAG *)
    BEGIN (* IF ELSE *)
      FINISH := TRUE;
      WRITELN('ERROR IN READING FROM DIGITIZER');
      WRITELN('PROGRAM WILL TERMINATE')
    END (* IF ELSE *)
  END; (* READ_DIG *)

PROCEDURE PLOT; (* THIS PROCEDURE IS USED TO SEND THE *)
                (* OUTPUT COORDINATES TO THE PLOTTER *)

VAR LOOP:INTEGER; (* LOOP IS A COUNTER VARIABLE *)

BEGIN (* PLOT *)
  PLOT CNT := 16;
  XYDATA[1] := NODE[1];
  XYDATA[2] := NODE[2];
  FOR LOOP := 1 TO 2 DO
    BEGIN (* FOR STATEMENT *)
      TEMP := XYDATA[LOOP] * SCALE;
      XYDATA[LOOP] := ROUND(TEMP) + TRANS[LOOP];
      LENGTH := 4;
      WHATCHAR(XYDATA[LOOP], PLOTLINE, PLOT CNT, LENGTH);
      PLOT CNT := PLOT CNT + 1;
      PLOTLINE[PLOT CNT] := ',';
      PLOT CNT := PLOT CNT + 1
    END; (* FOR STATEMENT *)
    PLOT CNT := PLOT CNT - 1;
    PLOTLINE[PLOT CNT] := ' ';

```

```

PLOT CNT := PLOT CNT + 1;
PLOT LINE[PLOT CNT] := CHR(13);
PLOT CNT := PLOT CNT + 1;
PLOT LINE[PLOT CNT] := CHR(10);
PLOT CNT := PLOT CNT + 1;
PLOT LINE[PLOT CNT] := '}' ;
LINOUT(PLOT LINE, ERROR_FLAG);
CHARIN(SIGNAL, ERFLAG)
END; (* PLOT *)

```

```

PROCEDURE FIRSTPNT; (* THIS PROCEDURE FINDS THE FIRST *)
                  (* PEN DOWN IN THE DRAWING AND *)
                  (* INITIALIZES POINT *)

```

```

VAR NUMBER, N: INTEGER;

```

```

BEGIN
  FIRST := FALSE;
  FINISH := FALSE;
  NUMBER := 0;
  REPEAT (* FIND THE FIRST PEN DOWN... KEEP LAST PEN UP *)
    (* (POINT[2]) AND FIRST PEN DOWN (POINT[3]) *)
    READ DIG; (* GETS A POINT FROM THE DIGITIZER *)
    NUMBER := NUMBER + 1;
    IF KEYNUMBER = 2 THEN
      BEGIN
        FINISH := TRUE;
        PEN := 'E'
      END;
    IF NOT FINISH THEN
      BEGIN
        NODE[1] := POINT[3,1];
        NODE[2] := POINT[3,2];
        PLOT
      END
    UNTIL (PEN = 'D') OR FINISH;
    POINT[1] := POINT[3];
    POINT[2] := POINT[3];
    PLOT LINE[11] := 'D';
    INSERT
  END; (* FIRSTPNT *)

```

```

PROCEDURE INTERSECT; (* THIS PROCEDURE FINDS THE *)
                  (* INTERSECT POINTS FOR EACH OF *)
                  (* THE RINGS OUT TO THE *)
                  (* OUTERMOST RING *)

```

```

VAR A, DISTX, DISTY: INTEGER;

```

```

PROCEDURE FINDPNT; (* THIS PROCEDURE FINDS THE X-Y *)
                  (* COORDINATES OF THE ACTUAL *)

```

(* INTERSECTION POINT *)

```

VAR
  TEMP: REAL;
  SIGNX: INTEGER;

BEGIN (* FINDPNT *)
  INTSCT[A] := TRUE; (* THE RING IS INTERSECTED BEFORE EOF
    OR END OF LINE SEGMENT *)
  IF POINT[3,1] = POINT[2,1]
  THEN
    BEGIN (* THEN CLAUSE *)
      IF POINT[1,2] <= POINT[3,2]
      THEN DELTAXY[A,2] := DELTA[A]
      ELSE DELTAXY[A,2] := -1 * DELTA[A];
      INTPNT[A,2] := POINT[1,2] + DELTAXY[A,2]
    END (* THEN CLAUSE *)
  ELSE
    BEGIN (* ELSE CLAUSE *)
      TEMP := (POINT[3,2] - POINT[2,2]) / (POINT[3,1] -
        POINT[2,1]);
      (* SLOPE OF DIGITIZED LINE CROSSING THE GRID *)
      IF POINT[3,1] >= POINT[2,1]
      THEN SIGNX := 1
      ELSE SIGNX := -1;
      TEMP := TEMP * (SIGNX * DELTA[A] - POINT[2,1] +
        POINT[1,1]);
      DELTAXY[A,2] := TEMP + POINT[2,2] - POINT[1,2];
      INTPNT[A,2] := POINT[1,2] + DELTAXY[A,2];
      IF ABS (DELTAXY[A,2]) > DELTA[A]
      THEN (* INTERSECTS ON TOP OR BOTTOM OF GRID *)
        BEGIN (* THEN CLAUSE *)
          IF DELTAXY[A,2] > 0
          THEN DELTAXY[A,2] := DELTA[A]
          ELSE DELTAXY[A,2] := -1 * DELTA[A];
          INTPNT[A,2] := POINT[1,2] + DELTAXY[A,2]
        END (* THEN CLAUSE *)
      END; (* ELSE CLAUSE *)
    IF (POINT[3,2] = POINT[2,2]) OR (ABS(DELTAXY[A,2]) <
      DELTA[A])
    THEN (* INTERSECTS ON LEFT OR RIGHT SIDE *)
      BEGIN (* THEN CLAUSE *)
        IF POINT[1,1] <= POINT[3,1]
        THEN DELTAXY[A,1] := DELTA[A]
        ELSE DELTAXY[A,1] := -1 * DELTA[A];
        INTPNT[A,1] := POINT[1,1] + DELTAXY[A,1]
      END (* THEN CLAUSE *)
    ELSE
      BEGIN (* ELSE CLAUSE *)
        TEMP := (POINT[3,1] - POINT[2,1]) / (POINT[3,2] -
          POINT[2,2]);
        TEMP := TEMP * (DELTAXY[A,2] - POINT[2,2] + POINT[1,2]);
        DELTAXY[A,1] := TEMP + POINT[2,1] - POINT[1,1];
        INTPNT[A,1] := POINT[1,1] + DELTAXY[A,1]
      END
    END
  END
END

```

```

        END (* ELSE CLAUSE *)
END; (* FINDPNT *)

```

```

PROCEDURE LAST; (* THIS PROCEDURE STORES THE LAST TWO      *)
                (* POINTS READ WHEN A RING IS                *)
                (* INTERSECTED                                *)

```

```

BEGIN (* LAST *)
    LASTPNT[A,1] := POINT[2,1];
    LASTPNT[A,2] := POINT[2,2];
    LASTPNT[A,3] := POINT[3,1];
    LASTPNT[A,4] := POINT[3,2]
END; (* LAST *)

```

```

BEGIN (* INTERSECT *)
    OUTRING := LEVEL[NUMRINGS];
    PTR := FRONT;
    A := 0;
    REPEAT
        IF NUMRINGS = 1 THEN A := OUTRING ELSE A := A + 1;
        REPEAT
            IF PTR <> NIL
            THEN
                BEGIN (* THEN CLAUSE *)
                    POINT[2] := POINT[3];
                    READLIST
                END (* THEN CLAUSE *)
            ELSE
                IF NOT FINISH AND (PEN <> 'U') THEN
                    BEGIN (* IF NOT FINISH AND PEN <> U *)
                        POINT[2] := POINT[3];
                        READ DIG;
                        IF KEYNUMBER = 2 THEN FINISH := TRUE;
                        INSERT
                    END; (* IF NOT FINISH AND PEN <> U *)
                    DISTX := ABS(POINT[3,1]-POINT[1,1])-DELTA[A];
                    DISTY := ABS(POINT[3,2]-POINT[1,2])-DELTA[A]
                UNTIL (DISTX >= 0) OR (DISTY >= 0) OR ((PTR = NIL)
                    AND FINISH) OR ((PTR = NIL) AND (PEN = 'U'));
                IF (DISTX >= 0) OR (DISTY >= 0)
                THEN
                    BEGIN
                        REPEAT
                            FINDPNT;
                            LAST;
                            A := A + 1;
                            DISTX := ABS(POINT[3,1] - POINT[1,1]) -
                                DELTA[A];
                            DISTY := ABS(POINT[3,2] - POINT[1,2]) -
                                DELTA[A]
                        UNTIL (A >= OUTRING) OR ((DISTX < 0) AND (DISTY < 0));
                        IF A <= OUTRING
                        THEN

```

```

        IF (DISTX >= 0) OR (DISTY >= 0)
        THEN
            BEGIN
                FINDPNT;
                LAST
            END
        ELSE A := A - 1
        END.
        ELSE INTSCT[A] := FALSE;
        IF FINISH THEN PEN := 'E'
        UNTIL A >= OUTRING
    END; (* INTERSECT *)

PROCEDURE DONE; (* THIS PROCEDURE IS CALLED WHEN THE      *)
                (* END OF THE DRAWING HAS BEEN REACHED. *)

VAR A: INTEGER;

BEGIN (* DONE *)
    REPEAT
        PTR := FRONT;
        INTERSECT;
        FOR A := 1 TO LEVEL[NUMRINGS] DO
            IF INTSCT[A] THEN OUTRING := A;
            IF INTSCT[LEVEL[1]] THEN
                BEGIN (* IF INTSCT *)
                    CALCODE;
                    ALIGN;
                    PLOT
                END; (* IF INTSCT *)
            UNTIL PTR = NIL
        END; (* DONE *)

BEGIN (* MAIN PROGRAM *)
    STARTUP;
    WHILE NOT FINISH DO
        BEGIN (* WHILE LOOP *)
            FIRSTPNT;
            IF NOT FINISH THEN
                BEGIN (* IF THEN *)
                    PTR := FRONT;
                    REPEAT
                        INTERSECT;
                        CASE PEN OF
                            'E', 'U' : BEGIN (* CASE PEN OF E OR U *)
                                FOR A := 1 TO NUMRINGS DO
                                    IF INTSCT[LEVEL[A]] THEN
                                        OUTRING := LEVEL[A];
                                    IF INTSCT[LEVEL[1]] THEN
                                        BEGIN (* IF INTSCT *)
                                            CALCODE;
                                            ALIGN;

```

```

        PLOT
        END; (* IF INTSCT *)
    DONE;
    IF PEN = 'E'
        THEN FINISH := TRUE
        ELSE FIRST := TRUE
    END; (* PEN OF E OR U *)
'D' : BEGIN (* CASE PEN OF D *)
    OUTRING := LEVEL[NUMRINGS];
    CALCODE;
    ALIGN;
    PLOT
        END; (* PEN OF D *)
    END (* CASE STATEMENT *)
UNTIL FIRST OR FINISH
END (* IF THEN *)
END; (* WHILE LOOP *)
WRITELN;
WRITELN('ALL DONE')
END. (* MAIN PROGRAM *)

```

The only difference between the COMPCODE procedure for CHNCODE and the COMPCODE procedure for PLOTCODE is the variables NUMRINGS, GRIDSIZE, and LEVEL. For the PLOTCODE program, NUMRINGS and GRIDSIZE are declared as external integers in the VAR section instead of single dimension arrays. Also, LEVEL is now a single dimension array instead of a two dimensional array. Since these are the only differences, the source code for the COMPCODE procedure for PLOTCODE is not listed here.

Appendix B

This appendix contains all the data and figures pertaining to the analysis of the chain codes. The data in the tables is arranged in descending order.

Table B-1

Circle Coded With 0.25 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
4	.0571219	1	11.4869
3	.0460307	1,2	10.636
2	.0437804	1,2,4	10.2106
2,4	.0417099	1,4	10.2106
1	.0399003	1,2,3,4	9.92697
2,3	.0344327	1,2,3	9.78515
1,2,3	.0340151	2,3	9.35971
3,4	.029784	2,4	9.35971
1,2	.0297006	2,3,4	8.93427
1,2,4	.0293341	1,3	8.50883
1,3	.0272705	1,3,4	8.50883
1,4	.0271261	2	7.94157
1,3,4	.0246575	3,4	7.65794
2,3,4	.0235653	3	6.38162
1,2,3,4	.0222014	4	4.96348

Table B-2

Circle Coded With 0.2 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
4	.0433054	1,4	16.5922
3	.042194	1,2	14.1813
3,4	.038012	1	14.0395
2	.0378037	1,2,3,4	12.4087
2,4	.0342058	1,2,4	12.3378
1	.0325768	1,3,4	11.9123
2,3	.0277863	2,4	11.9123
1,4	.0267378	1,2,3	11.9123
1,2	.0259034	2,3,4	11.416
1,2,3,4	.0258092	1,3	11.3451
2,3,4	.0257596	2,3	11.0614
1,3	.025754	2	9.64334
1,3,4	.025387	3,4	8.93427
1,2,4	.0250073	3	8.15429
1,2,3	.0237554	4	6.38162

Table B-3

Circle Coded With 0.15 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
4	.025238	1	18.2939
3	.023449	1,4	17.0176
2	.0218877	1,2	16.6631
3,4	.0216231	2,3	15.3159
1	.0210488	1,2,3	15.3159
1,2	.0206047	1,2,3,4	14.3941
2,3	.0200604	2,3,4	14.3941
2,4	.0195636	1,3	14.1813
1,2,4	.017529	1,2,4	14.103
1,4	.0174938	2,4	14.0395
1,2,3,4	.0169143	1,3,4	13.6141
1,2,3	.0167028	2	12.4796
2,3,4	.0165344	3,4	11.0614
1,3	.0161368	3	10.636
1,3,4	.0151618	4	8.15429

Table B-4

Circle Coded With 0.1 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
4	.0206494	1	27.2282
2,4	.0181527	1,4	27.2282
2	.0178209	1,2	25.881
3,4	.0164992	1,2,3,4	25.3137
3	.0159846	1,2,3	25.101
1	.0159659	1,3,4	24.6756
2,3,4	.0152976	1,3	24.1083
2,3	.0151722	1,2,4	23.3992
1,2,3,4	.0148214	2,3,4	23.3283
1,4	.0139625	2,3	22.9738
1,3,4	.0139252	2,4	21.6975
1,2,4	.0138379	2	18.4358
1,2,3	.0132626	3,4	17.0176
1,2	.0128413	3	15.5995
1,3	.0126179	4	11.6996

Table B-5

Circle Coded With 0.05 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
2	.00899887	1	54.2438
4	.0088706	1,2	53.8892
2,4	.00825768	2,3,4	45.1677
3,4	.0082074	2,3	42.5441
3	.00798065	2,4	42.1187
2,3,4	.00789176	2	36.3043
2,3	.00740921	3,4	32.759
1	.00731703	3	30.4899
1,2	.00686694	4	23.0447

Table B-6

SQUARE-0 Coded With 0.25 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
3	.0865381	1,4	14.593
4	.0782665	2,3,4	13.1337
3,4	.0524139	1,3,4	13.1337
2	.033646	1,3	12.1609
2,3	.03357	1	11.6744
2,4	.0333724	1,2,3,4	11.0664
2,3,4	.0257822	1,2	10.9448
1,4	.023153	1,2,3	10.9448
1,2	.023153	2,3	10.2151
1,3	.023153	1,2,4	10.2151
1,2,3,4	.023153	2,4	8.75584
1,2,3	.023153	2	7.78297
1,2,4	.023153	3,4	7.29654
1,3,4	.023153	3	6.08044
1	.023153	4	4.86436

Table B-20

SQUARE-45 Coded With 0.05 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
4	.0120054	1,2	56.7843
3,4	.0113016	2,3,4	45.3053
3	.0112894	1	43.2293
2,4	.0106233	2,4	40.2985
2	.0106233	2,3	39.5658
1	.0104661	2	28.8195
1,2	.0101697	3,4	28.5753
2,3,4	.00993329	3	24.4233
2,3	.00992719	4	18.3175

Table B-19

SQUARE-45 Coded With 0.1 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
4	.0170141	1,4	32.2388
3	.014881	1,2,3,4	28.209
3,4	.0148688	1,3,4	27.8426
2,3	.0126287	1,2,4	25.6445
2,3,4	.0126165	1,2,3	25.6445
2,4	.0123596	1,2	24.4233
2	.0123596	1,3	23.8127
1,2,4	.0119108	1	21.6146
1,3,4	.0119108	2,3,4	20.5156
1,2	.0119108	2,3	19.0502
1,2,3,4	.0119108	2,4	18.3175
1	.0119108	2	14.654
1,3	.0119108	3,4	13.9213
1,4	.0119108	3	12.8222
1,2,3	.0119108	4	9.76935

Table B-18

SQUARE-45 Coded With 0.15 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
4	.0360339	1,4	19.7829
3,4	.0278093	1	14.2876
3	.027791	1,2	14.0434
2,4	.0195481	1,3,4	13.9213
2	.0195481	1,2,4	13.9213
2,3,4	.0168188	1,2,3,4	13.677
2,3	.0168004	1,3	12.8222
1,2,4	.0140711	2,3,4	12.8222
1,3,4	.0140711	2,4	12.4559
1,2	.0140711	1,2,3	12.4559
1,2,3,4	.0140711	2,3	11.7232
1	.0140711	2	9.76935
1,3	.0140711	3,4	9.52511
1,4	.0140711	3	8.54818
1,2,3	.0140711	4	6.71643

Table B-17

SQUARE-45 Coded With 0.2 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
4	.0787137	1,2,3,4	17.0963
3	.0494789	1,4	16.8521
3,4	.0494545	1,2,4	15.3867
2,3	.0397096	1,3,4	15.3867
2,4	.0396852	1,2,3	15.3867
2	.0396852	1,2	14.0434
2,3,4	.0348249	1,3	12.8222
1,2,4	.0299402	2,3,4	11.1126
1,3,4	.0299402	1	10.6241
1,2	.0299402	2,3	10.2578
1,2,3,4	.0299402	2,4	9.52511
1	.0299402	2	7.32701
1,3	.0299402	3,4	7.32701
1,4	.0299402	3	6.71643
1,2,3	.0299402	4	4.88467

Table B-16

SQUARE-45 Coded With 0.25 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
4	.106124	1,4	14.654
3,4	.0617745	1,3,4	13.1886
1	.0410334	1,2,3,4	12.8222
1,4	.0410334	1,2,4	11.7232
2	.0392058	1,2	10.9905
2,4	.0392058	1,3	10.9905
1,2,4	.0390922	1,2,3	10.9905
1,2	.0390922	1	9.52511
2,3,4	.0388089	2,3,4	9.403
2,3	.0387783	2,4	8.05971
1,3	.0372728	2,3	8.05971
1,3,4	.0372728	3,4	6.59431
1,2,3,4	.0372357	2	6.35007
1,2,3	.0372357	3	5.49526
3	.035274	4	4.27409

Table B-15

SQUARE-30 Coded With 0.05 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
2,3	.0102448	1	51.6472
2	.00979521	1,2	51.1656
4	.00971353	2,3,4	46.35
2,4	.00939931	2,3	44.0626
3	.00915299	2,4	40.4509
2,3,4	.00892546	2	34.1906
3,4	.00798058	3,4	31.0605
1	.00722729	3	28.8935
1,2	.0049792	4	21.6701

Table B-14

SQUARE-30 Coded With 0.1 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
4	.027109	1,4	29.6158
2,3,4	.0214104	1,2	25.8837
2,3	.0202131	1	25.643
3	.0193276	1,2,3,4	25.2818
2	.0182949	1,3	24.6799
1,2	.0170821	1,2,3	24.5595
2,4	.0170821	2,3,4	24.4391
3,4	.0168885	1,3,4	23.1148
1	.0149234	1,2,4	23.1148
1,3	.0149234	2,3	22.3924
1,2,4	.0143164	2,4	20.2254
1,4	.0137629	2	17.3361
1,2,3	.0137092	3,4	16.6137
1,3,4	.0136997	3	14.4467
1,2,3,4	.0130003	4	10.835

Table B-13

SQUARE-30 Coded With 0.15 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
4	.0406664	1,4	23.8371
3	.0355717	1,2,3,4	20.2254
2	.0314643	1,2,4	19.5031
2,4	.0302024	1,2,3	18.0584
2,3	.0296651	1	17.3361
2,3,4	.026365	1,3,4	16.6137
3,4	.0239979	2,3,4	16.0118
1	.0224846	1,3	15.6506
1,2,3	.0224699	2,3	15.1691
1,3	.0218679	2,4	13.7244
1,3,4	.0208045	3,4	11.5574
1,2,3,4	.0205804	2	11.5574
1,2,4	.0186802	3	9.63117
1,4	.0185656	1,2	8.30635
1,2	.0175606	4	7.22338

Table B-12

SQUARE-30 Coded With 0.2 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
3,4	.061025	1,4	17.3361
3	.052909	1,2,3,4	16.8545
2	.0481822	1,3,4	15.8914
4	.0476209	1,2,4	15.1691
2,4	.0456853	1,2	15.0487
1	.0338033	1,2,3	13.7244
1,3	.0330289	1,3	13.2428
1,4	.0325602	1	13.002
1,2,4	.0317454	2,3,4	11.7981
2,3,4	.030705	2,3	10.835
1,3,4	.0302271	2,4	10.1127
1,2,3	.0297057	2	8.66806
2,3	.0282764	3,4	7.94572
1,2,3,4	.026627	3	7.22338
1,2	.0259995	4	6.01948

Table B-11

SQUARE-30 Coded With 0.25 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
4	.0764967	1,4	13.002
3	.0679133	1,2	11.437
2	.0466314	1,2,3,4	10.9554
2,4	.0466163	1	10.835
2,3,4	.0410383	1,3	10.835
2,3	.0369186	1,2,4	10.835
1	.0364514	2,3,4	10.1127
1,3	.0352499	1,3,4	10.1127
1,2,3	.0313723	1,2,3	10.1127
1,2,4	.0287189	2,4	9.3904
1,4	.0284036	2,3	9.3904
1,2	.0272566	2	7.22338
1,3,4	.0258869	3,4	6.50104
3,4	.0249573	3	6.01948
1,2,3,4	.0231457	4	4.81559

Table B-10

SQUARE-0 Coded With 0.05 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
3	.00995035	1,2	61.4125
3,4	.00916263	1	59.1019
2,3	.00874405	2,3,4	51.0757
4	.00861047	2,3	48.1571
2,3,4	.00844003	2,4	46.6978
2,4	.00842635	2	39.4013
2	.00841455	3,4	35.753
1	.00818465	3	32.8344
1,2	.00818465	4	24.9298

Table B-9

SQUARE-0 Coded With 0.1 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
3	.0253283	1,4	37.2123
3,4	.0205395	1,3,4	31.3751
4	.0186636	1,2,3,4	30.6454
2	.0184423	1,2	29.7942
2,4	.0183207	1	29.5509
1	.0178789	1,2,3	28.4565
1,4	.0176783	1,2,4	28.4565
1,2	.0169646	1,3	27.362
1,2,4	.0168308	2,3,4	24.6866
2,3	.0166915	2,3	24.0785
2,3,4	.0166063	2,4	22.6192
1,3	.016587	2	19.9438
1,3,4	.0165201	3,4	17.5117
1,2,3	.0163076	3	15.8091
1,2,3,4	.0162407	4	12.7689

Table B-8

SQUARE-0 Coded With 0.15 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
4	.0480225	1,2,3,4	29.7942
3	.0433962	1,2,3	26.9972
3,4	.0432776	1,4	26.9972
2	.0387158	1,3,4	26.2675
2,3	.0385793	1,2,4	26.2675
2,4	.0385269	1,2	24.9298
2,3,4	.0385269	1,3	23.7137
1	.0315528	1	19.3358
1,3	.0313704	2,3,4	17.8765
1,3,4	.0312792	2,3	16.0523
1,4	.0312792	2,4	15.3227
1,2	.031259	2	12.6473
1,2,3	.0311678	3,4	11.6744
1,2,3,4	.0310766	3	10.9448
1,2,4	.0310766	4	8.51263

Table B-7

SQUARE-0 Coded With 0.2 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
4	.0764335	1,2,3,4	16.174
3,4	.0617601	1,3,4	16.0523
3	.0401376	1,4	16.0523
2,3	.0286988	1,2,3	15.3227
2	.028496	1,2	15.2011
2,3,4	.0283501	1,3	14.593
2,4	.0283014	1	14.593
1	.0217971	1,2,4	13.8634
1,2	.0216755	2,3,4	12.7689
1,2,3	.0215539	2,3	11.6744
1,3	.0215539	2,4	10.9448
1,2,3,4	.0214323	2	9.72872
1,2,4	.0214323	3	8.51263
1,3,4	.0214323	3,4	8.02619
1,4	.0214323	4	6.08044

Table B-21

SQUARE-60 Coded With 0.25 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
3	.0694956	1,4	12.8
4	.066657	1,3	11.8518
2,4	.0501826	1,2,3,4	11.6148
2	.0500789	1,3,4	11.3778
2,3,4	.041145	1,2	11.2593
1	.0367181	1	10.6667
1,4	.0328972	1,2,4	10.6667
1,3	.0321207	1,2,3	10.6667
1,3,4	.0316774	2,3,4	9.95559
1,2,4	.0313565	2,3	8.53336
3,4	.0307504	2,4	8.53336
2,3	.0301129	2	7.11113
1,2,3,4	.0287303	3,4	6.40002
1,2	.0228083	3	5.33335
1,2,3	.0220892	4	4.74075

Table B-22

SQUARE-60 Coded With 0.2 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
3,4	.0519887	1,4	14.9333
4	.0496305	1,2,3	14.2222
2	.0431296	1,2	14.2222
2,4	.0431296	1,2,3,4	14.1037
3	.0415733	1,2,4	13.5111
2,3,4	.0390647	1,3	13.037
2,3	.0357547	1	12.8
1,2,4	.0326856	1,3,4	12.0889
1	.0315811	2,3,4	11.6148
1,2,3,4	.031371	2,4	10.6667
1,3,4	.030425	2,3	10.6667
1,4	.0303095	2	8.53336
1,3	.0282973	3,4	7.82225
1,2,3	.0266533	3	7.70373
1,2	.0251806	4	5.33335

Table B-23

SQUARE-60 Coded With 0.15 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
4	.0448341	1,2	19.5556
3	.036445	1,4	19.2
2	.0289377	1,2,3,4	19.0815
2,4	.0279655	1,2,4	18.4889
2,3,4	.0268246	1	17.0667
1,3	.024342	1,3,4	17.0667
1	.0236559	1,3	16.5926
1,4	.0236198	1,2,3	16.5926
3,4	.0228392	2,3,4	14.9333
1,2,3	.0214367	2,4	14.2222
1,2,4	.0207672	2,3	13.5111
1,3,4	.0202818	2	11.3778
1,2,3,4	.0193619	3,4	10.6667
2,3	.0192783	3	9.48151
1,2	.01703	4	7.11113

Table B-24

SQUARE-60 Coded With 0.1 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
4	.028357	1,4	27.7334
3	.0193231	1,2	26.6667
2	.0189168	1	25.6
2,4	.0186962	1,2,3,4	24.8889
3,4	.0166949	1,3	23.7037
2,3,4	.0156298	1,3,4	23.4667
2,3	.0149809	1,2,4	23.4667
1	.0148037	1,2,3	23.4667
1,2,4	.0141943	2,3,4	21.5704
1,4	.0138881	2,4	20.6223
1,3	.013315	2,3	20.6223
1,2	.0124786	2	17.0667
1,3,4	.0121346	3,4	15.6445
1,2,3,4	.0118581	3	14.2222
1,2,3	.0114837	4	10.6667

Table B-25

SQUARE-60 Coded With 0.05 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
4	.00939215	1,2	50.3705
2	.00916857	1	50.1335
2,3,4	.0090146	2,3,4	46.4594
3	.00898931	2,3	40.5334
2,4	.00836145	2,4	37.689
3,4	.00777994	2	33.6593
1	.00696494	3,4	30.5779
2,3	.00631943	3	27.8519
1,2	.00464985	4	21.3334

Table B-26

Sine Wave Coded With 0.25 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
4	.177014	1,4	12.8262
3,4	.101479	1,2,4	11.585
3	.0674033	1,3,4	11.585
2	.057571	1,2	11.0333
2,3	.0547971	1	10.9644
2,4	.0516681	1,2,3	10.7575
2,3,4	.0500062	1,3	10.6885
1	.0469225	1,2,3,4	10.6196
1,2,3	.0404034	2,3	9.93002
1,2	.0393538	2,3,4	9.65418
1,3	.0371786	2,4	9.51626
1,4	.0333935	2	7.17168
1,2,4	.0326867	3	6.20626
1,3,4	.0312101	3,4	5.37876
1,2,3,4	.0276557	4	3.44792

Table B-27

Sine Wave Coded With 0.2 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
4	.0951982	1,4	16.55
3	.066723	1,2	14.4812
3,4	.0563303	1,2,3	14.4812
2	.0447947	1,2,4	14.4812
2,3	.0387108	1	14.0675
2,4	.0383211	1,3	13.1021
1	.0351407	1,2,3,4	13.0331
1,2	.0319325	1,3,4	12.8262
2,3,4	.0294085	2,3	11.9987
1,4	.026748	2,4	11.585
1,2,3	.0264903	2,3,4	11.1023
1,2,4	.0258608	2	9.37835
1,3,4	.0253898	3,4	7.44751
1,3	.0235785	3	7.24064
1,2,3,4	.022521	4	5.17188

Table B-28

Sine Wave Coded With 0.15 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
4	.059578	1,2,3	19.86
3	.0405581	1,4	19.86
3,4	.0300417	1,2	18.9635
2	.0275403	1	18.8256
2,3	.0272906	1,3,4	18.205
1	.0266828	1,2,3,4	17.8602
2,4	.0242809	1,2,4	17.3775
1,2,3	.0222929	2,4	16.9637
1,2	.0220264	1,3	16.8948
1,3,4	.0190452	2,3	16.1362
2,3,4	.0190356	2,3,4	14.9639
1,3	.0178728	2	12.6883
1,2,4	.0167421	3,4	11.1712
1,4	.0166428	3	10.3437
1,2,3,4	.0152194	4	7.58543

Table B-29

Sine Wave Coded With 0.1 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
4	.0285503	1	28.135
3	.0244672	1,4	27.3075
3,4	.0200061	1,2,3	26.8938
1	.0180841	1,2	26.549
2,3	.0172292	1,3	26.2042
2	.0160311	1,2,4	25.2388
1,2,3	.0142483	1,2,3,4	25.1008
1,3	.0140129	2,3	24.4113
1,2	.0135987	2,4	23.17
2,4	.0128029	2,3,4	22.2046
2,3,4	.0127397	1,3,4	21.9288
1,2,4	.0120049	2	18.7567
1,4	.0115023	3,4	16.55
1,2,3,4	.0111887	3	15.5156
1,3,4	.00914258	4	11.7229

Table B-30

Sine Wave Coded With 0.05 Inch Gridsize

Code	aepl (sq in/in)	Code	bpl (bits/in)
4	.0135882	1	56.4769
3	.0091371	1,2	51.7188
1	.00859574	2,3	47.5813
2	.00834132	2,3,4	46.8228
2,3	.00795684	1,4	44.685
3,4	.00744803		37.7892
1,2	.00743075	1,4	33.9275
2,4	.00631215	3	31.3761
2,3,4	.0058975	4	23.4458

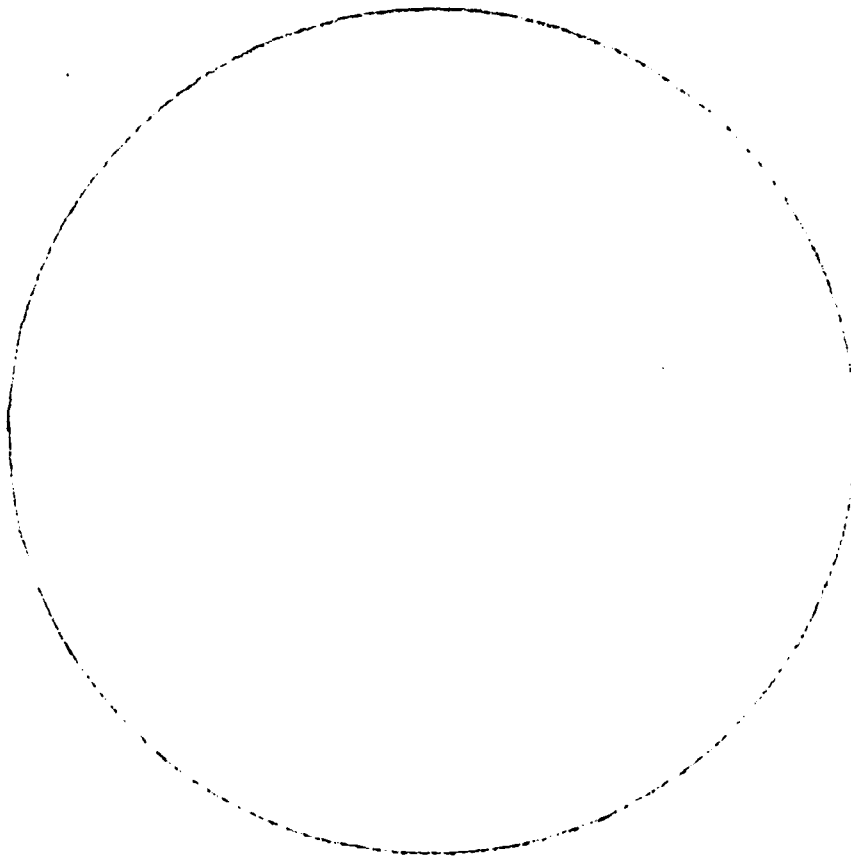


Figure B-1. Digitized CIRCLE Drawing

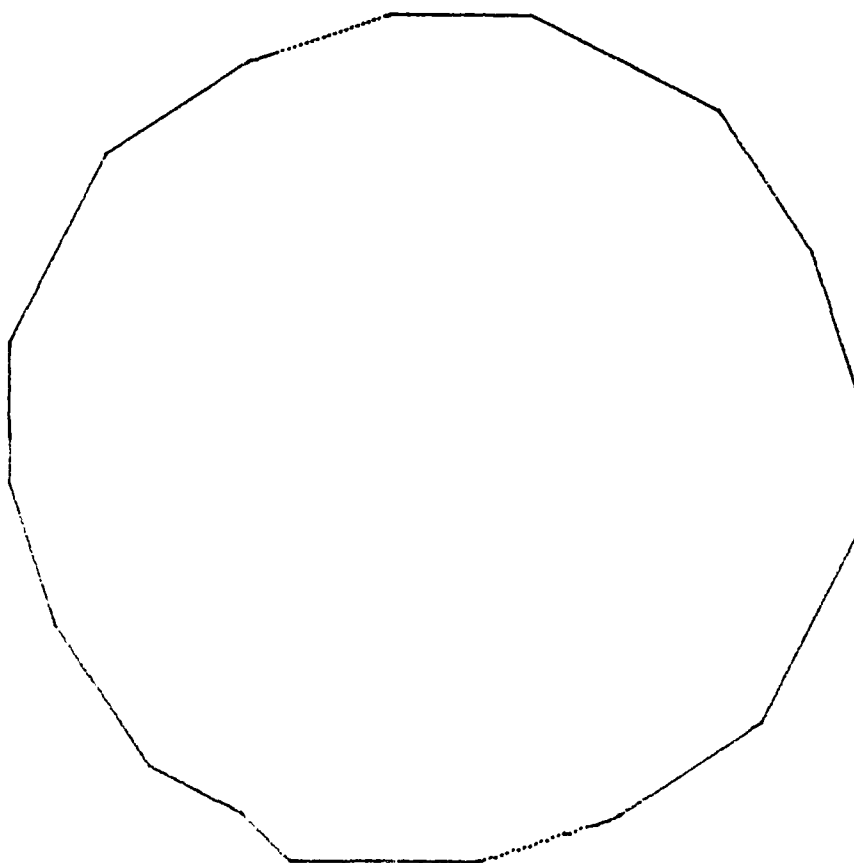
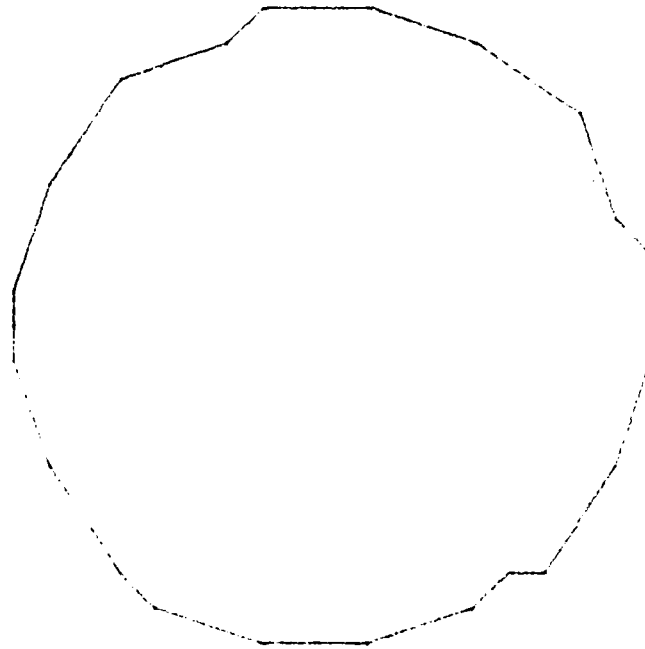


Figure B-2. CIRCLE: (1,2,3,4) Code With 0.25 Inch Gridsize

(1,3)



(1,4)

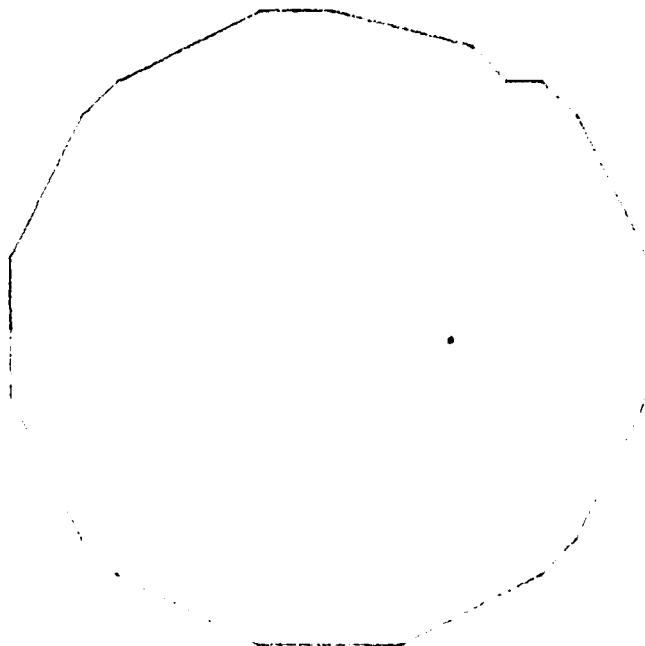


Figure B-3. CIPCLE: (1,3) and (1,4), Codes with 0.25
Inch Gridsize (Reduced to 3/4 original size,

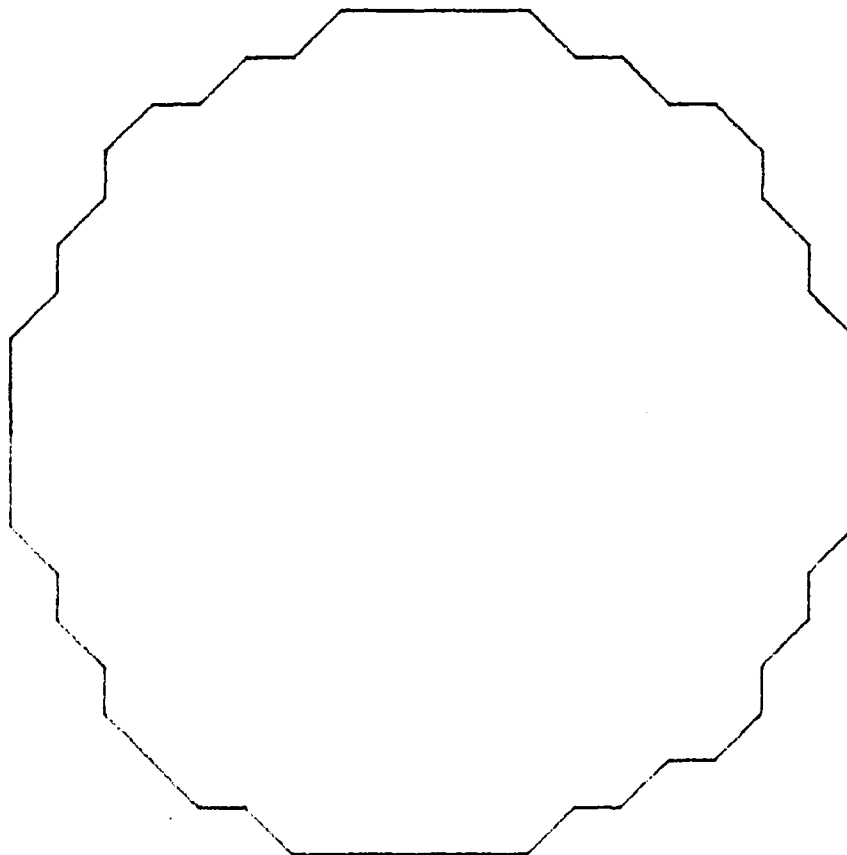
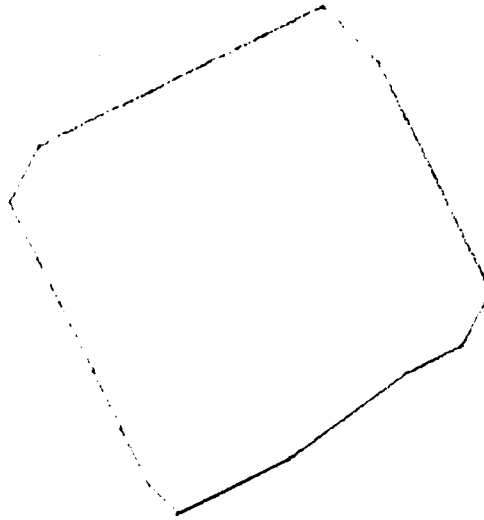


Figure B-4. CIRCLE: (1) Code With 0.25 Inch Gridsize

(2,4)



(3,4)



Figure B-19. SQUARE-30: (2,4) and (3,4) Codes With
0.15 Inch Gridsize

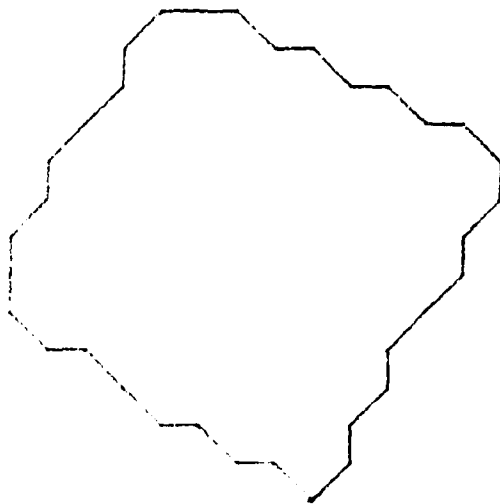


Figure B-18. SQUARE-60: (1) Code With 0.2 Inch Gridsize

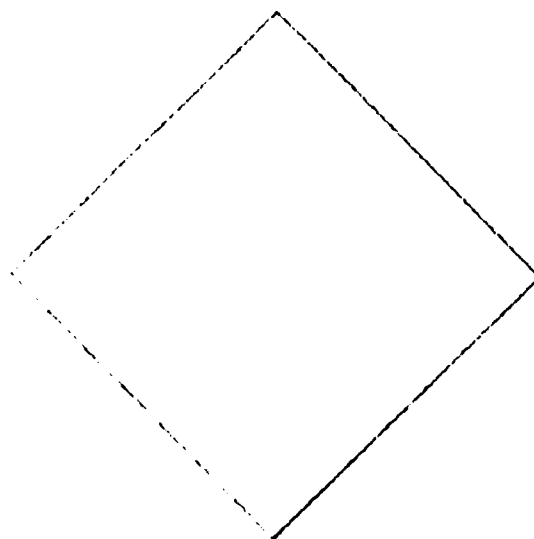


Figure B-17. SQUARE-45: (1) Code With 0.2 Inch Gridsize

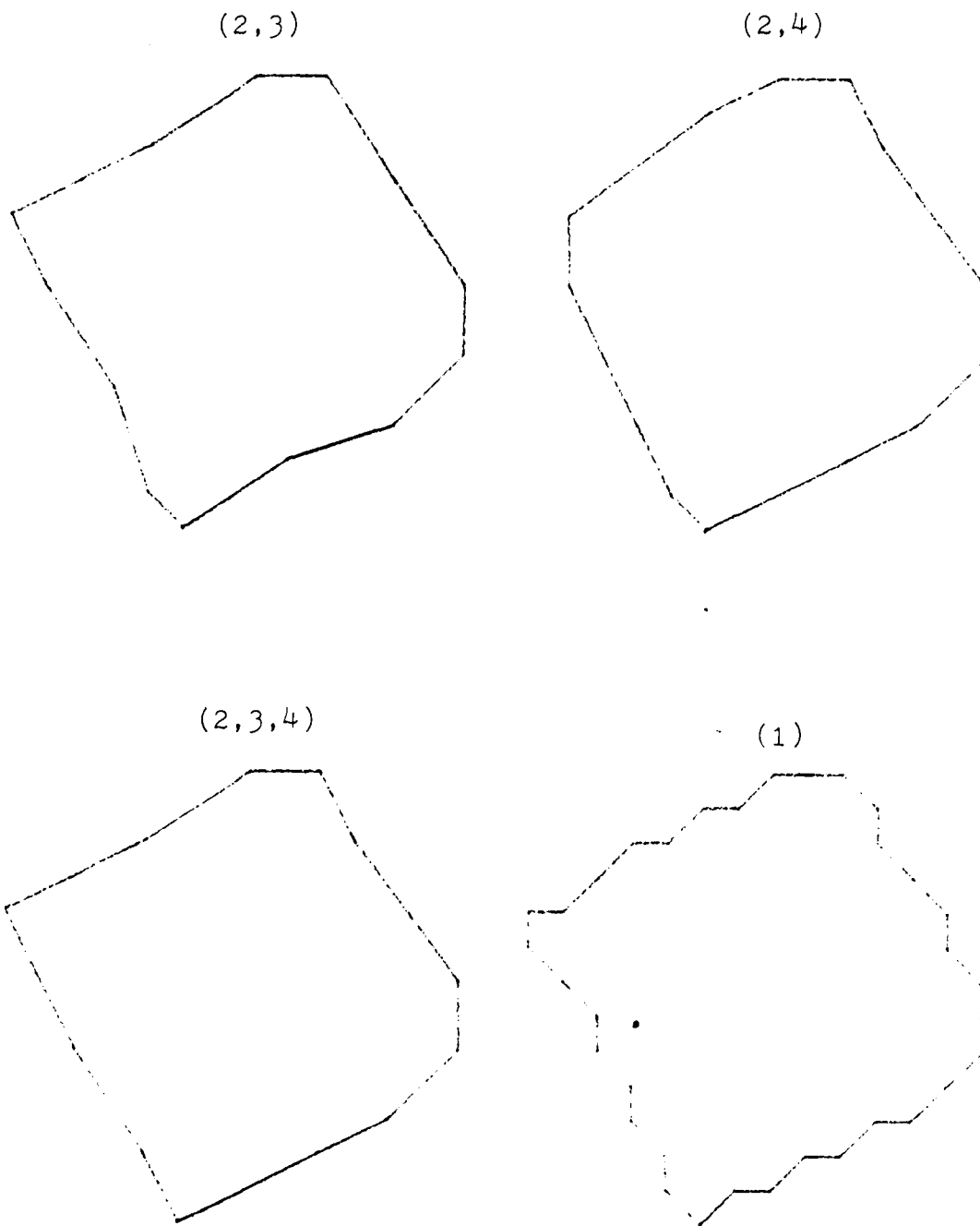


Figure B-16. SQUARE-30: (2,3), (2,4), (2,3,4), and (1) Codes
With 0.2 Inch Gridsize

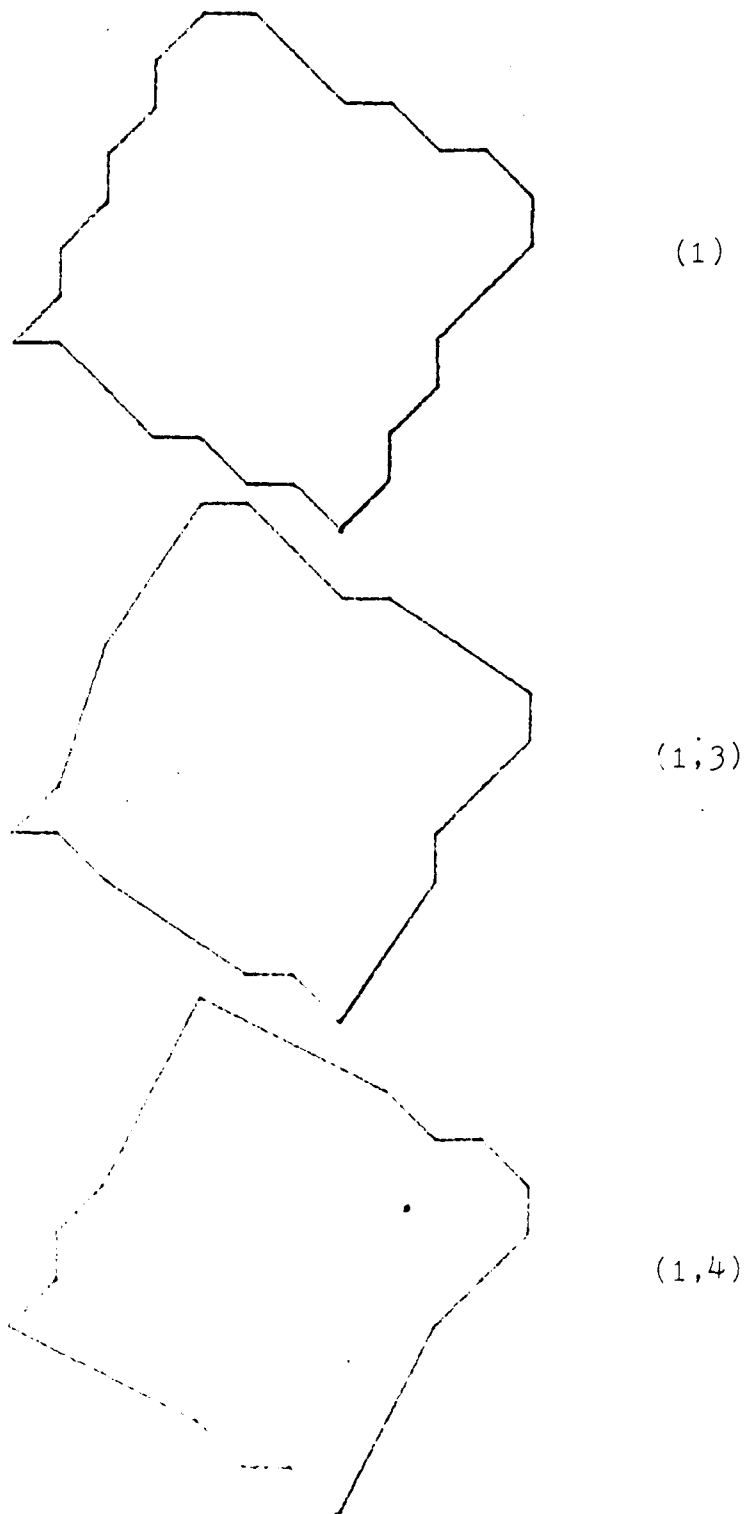


Figure B-15. SQUARE-60: (1), (1,3), and (1,4) Codes With
0.25 Inch Gridsize

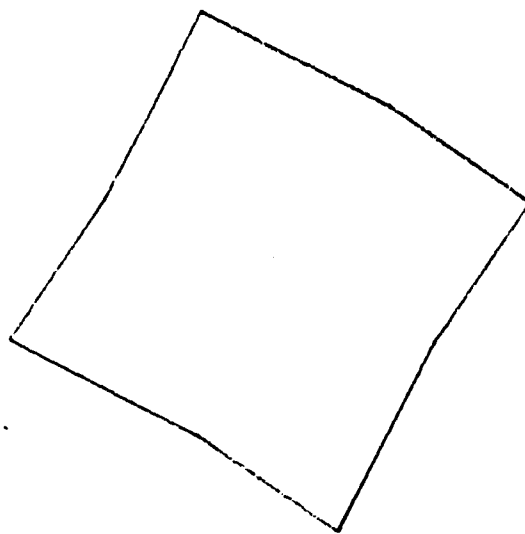


Figure B-14. SQUARE-60: (3,4) Code With 0.25 Inch Gridsize

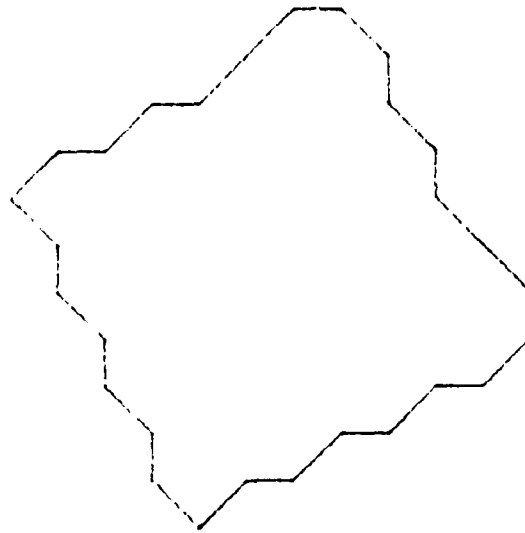


Figure B-12. SQUARE-30: (1) Code With 0.25 Inch Gridsize

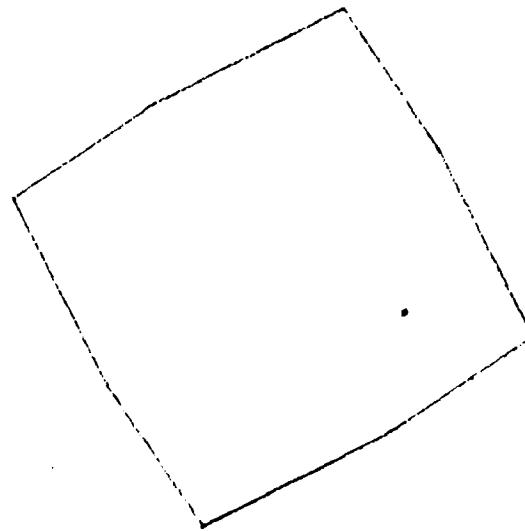


Figure B-13. SQUARE-30: (3,4) Code With 0.25 Inch Gridsize

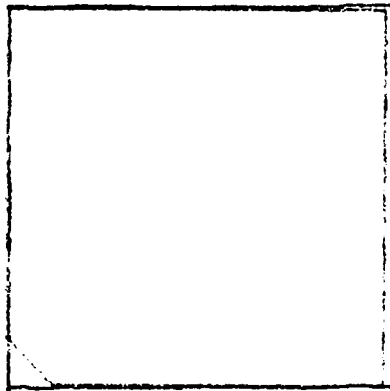
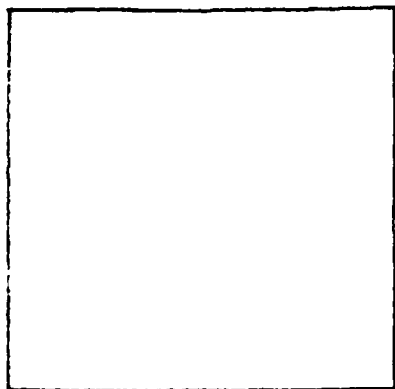
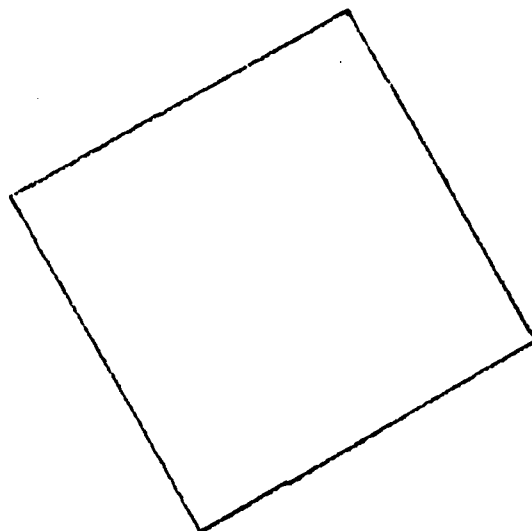


Figure B-11. Example of Error For Non-Rotated Square

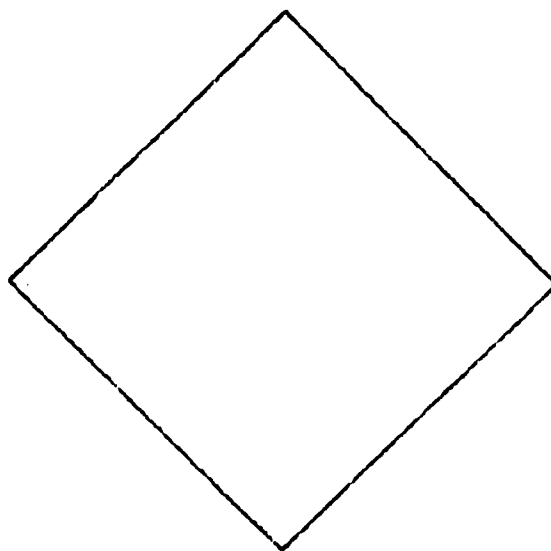
SQUARE-0



SQUARE-30



SQUARE-45



SQUARE-60

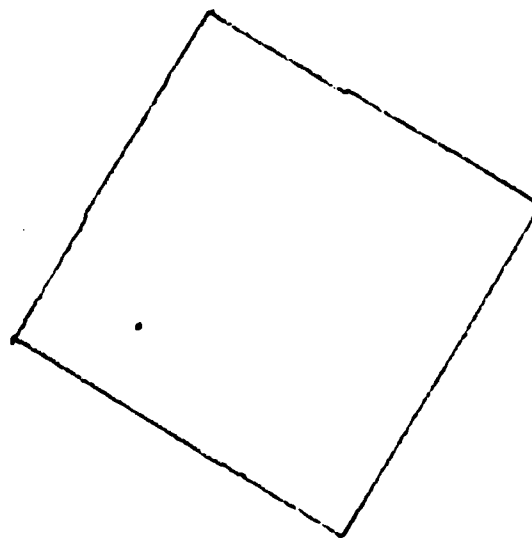


Figure B-10. Digitized Square Drawings

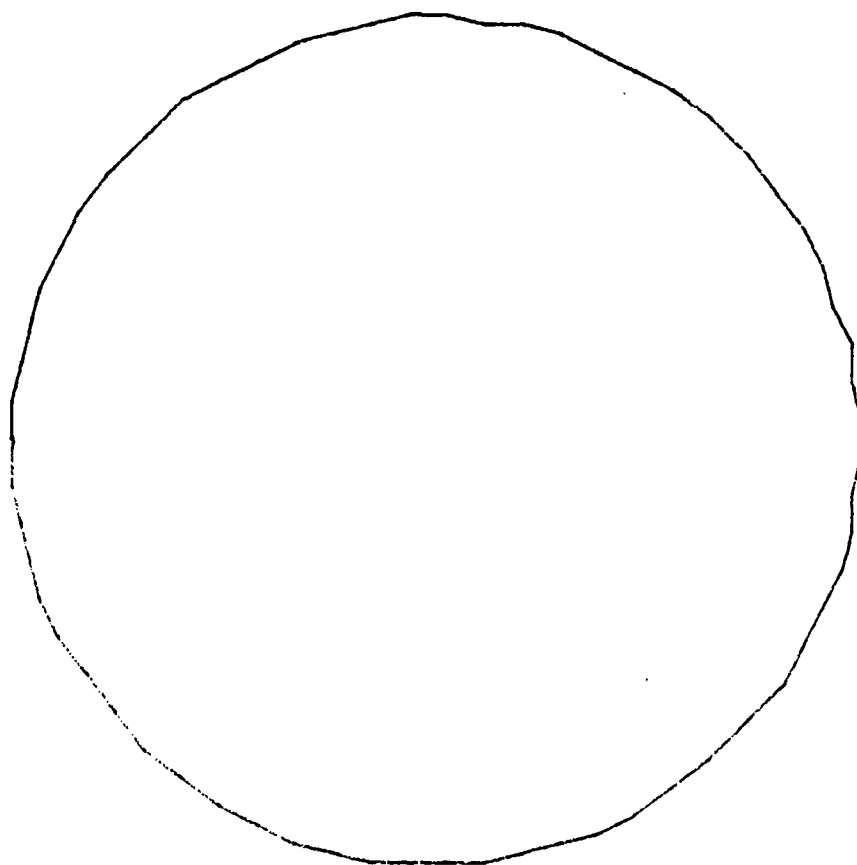
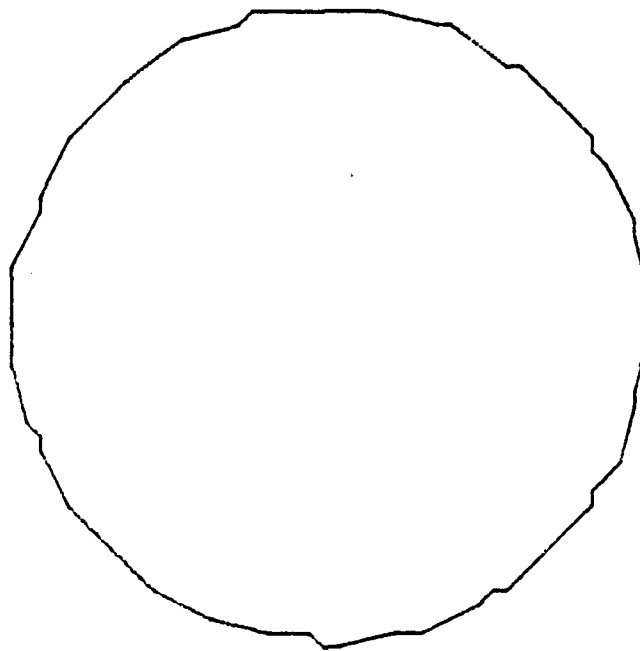


Figure B-9. CIRCLE: (4) Code With 0.05 Inch Gridsize

(1,4)



(3,4)

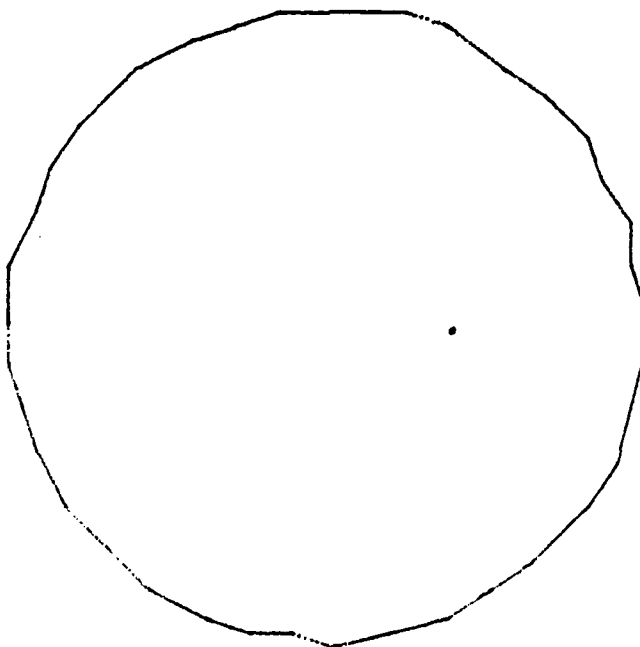


Figure B-8. CIRCLE: (1,4) and (3,4) Codes With
0.1 Inch Gridsize

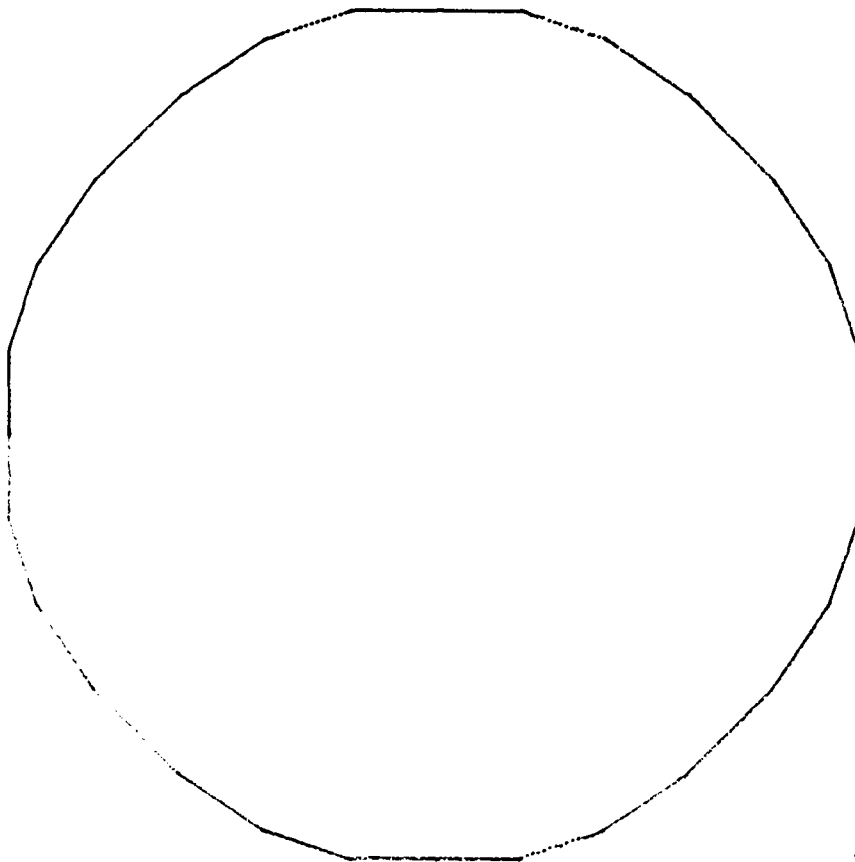


Figure B-7. CIRCLE: (3) Code With 0.15 Inch Gridsize

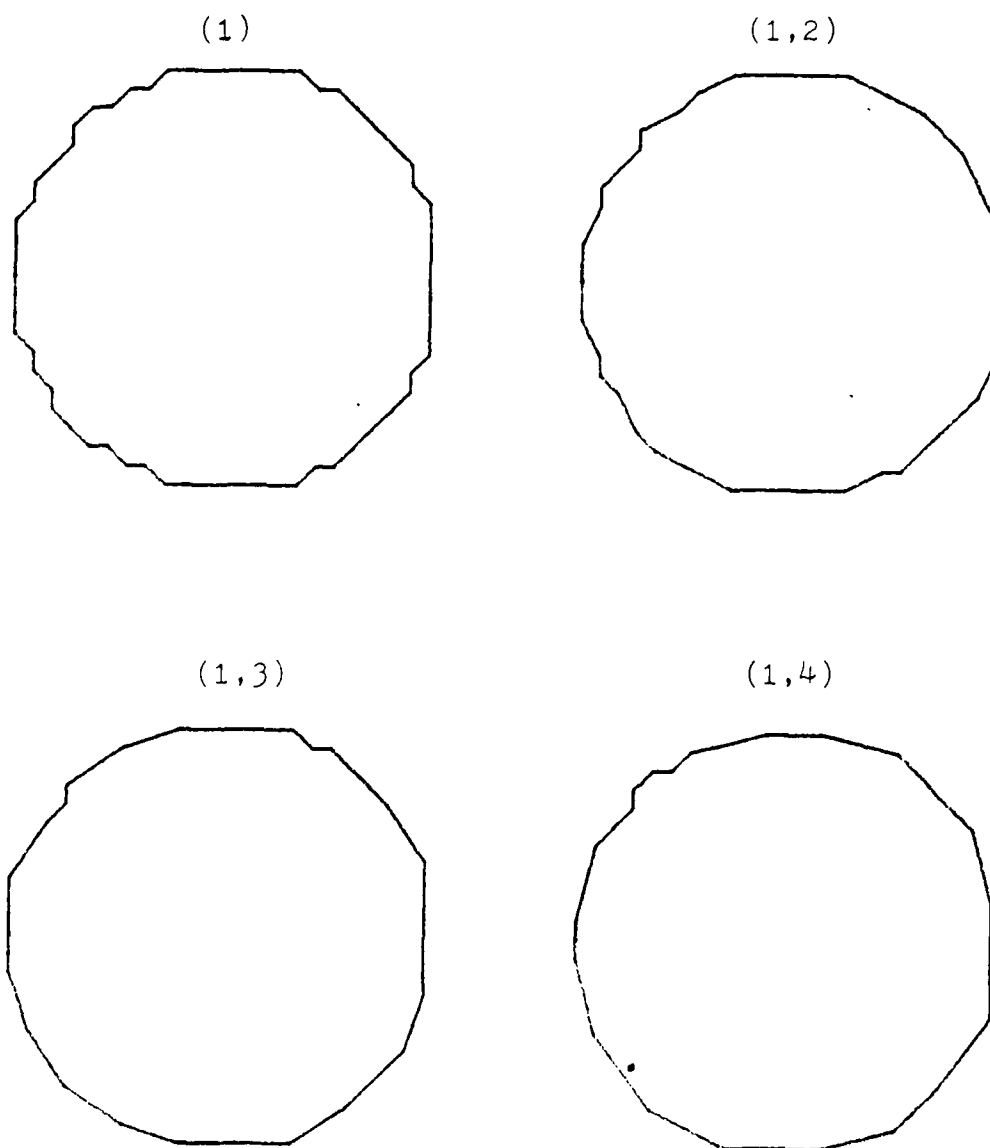


Figure B-6. CIRCLE: (1),(1,2),(1,3), and (1,4) Codes With
0.2 Inch Gridsize (Reduced to $\frac{1}{2}$ original size)

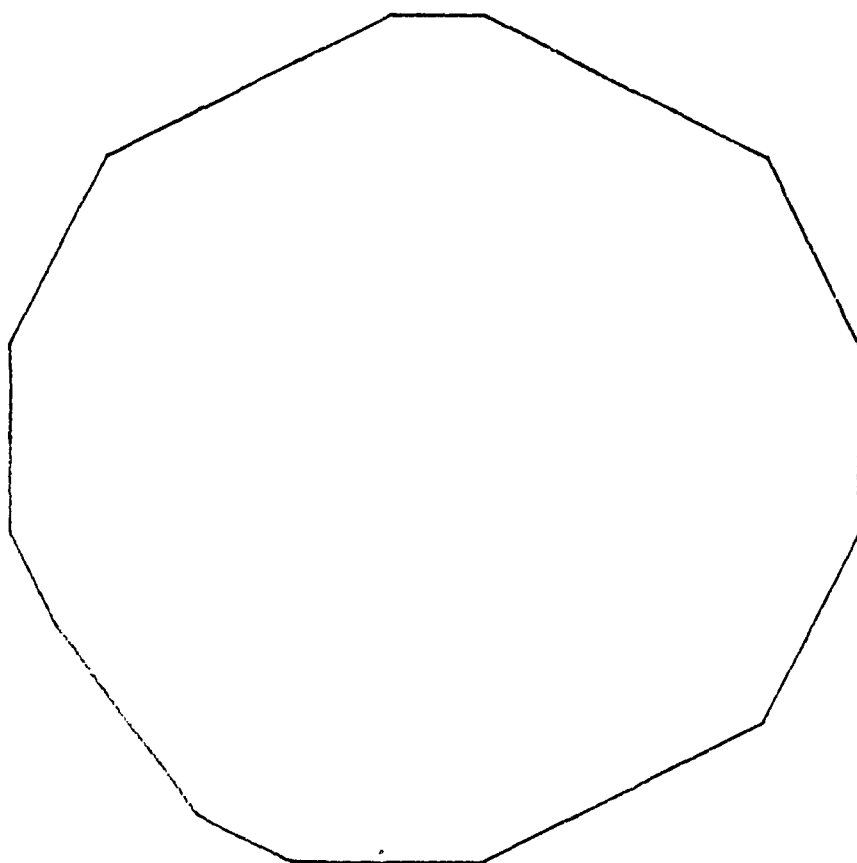


Figure B-5. CIRCLE: (2,4) Code With 0.25 Inch Gridsize

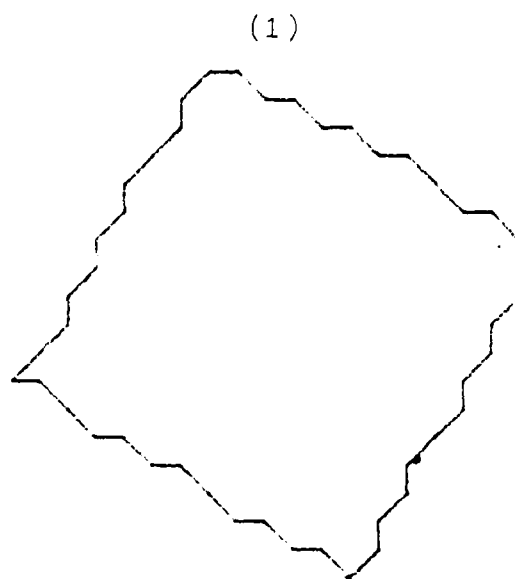
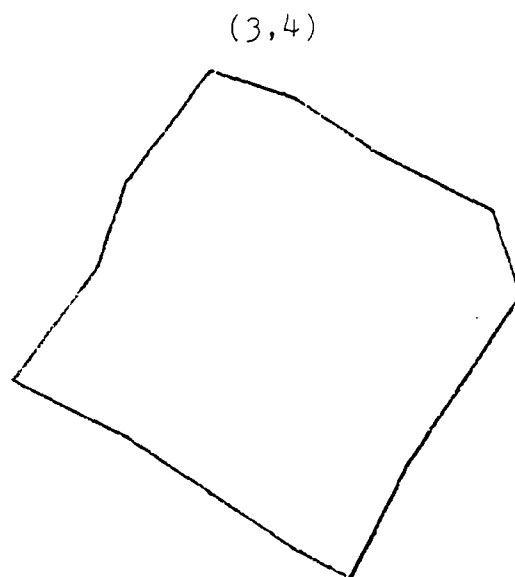


Figure B-20. SQUARE-60: (3,4) and (1) Codes With
0.15 Inch Gridsize

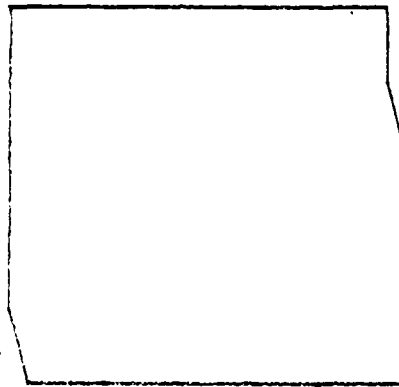


Figure B-21. SQUARE-0: (4) Code With 0.1 Inch Gridsize

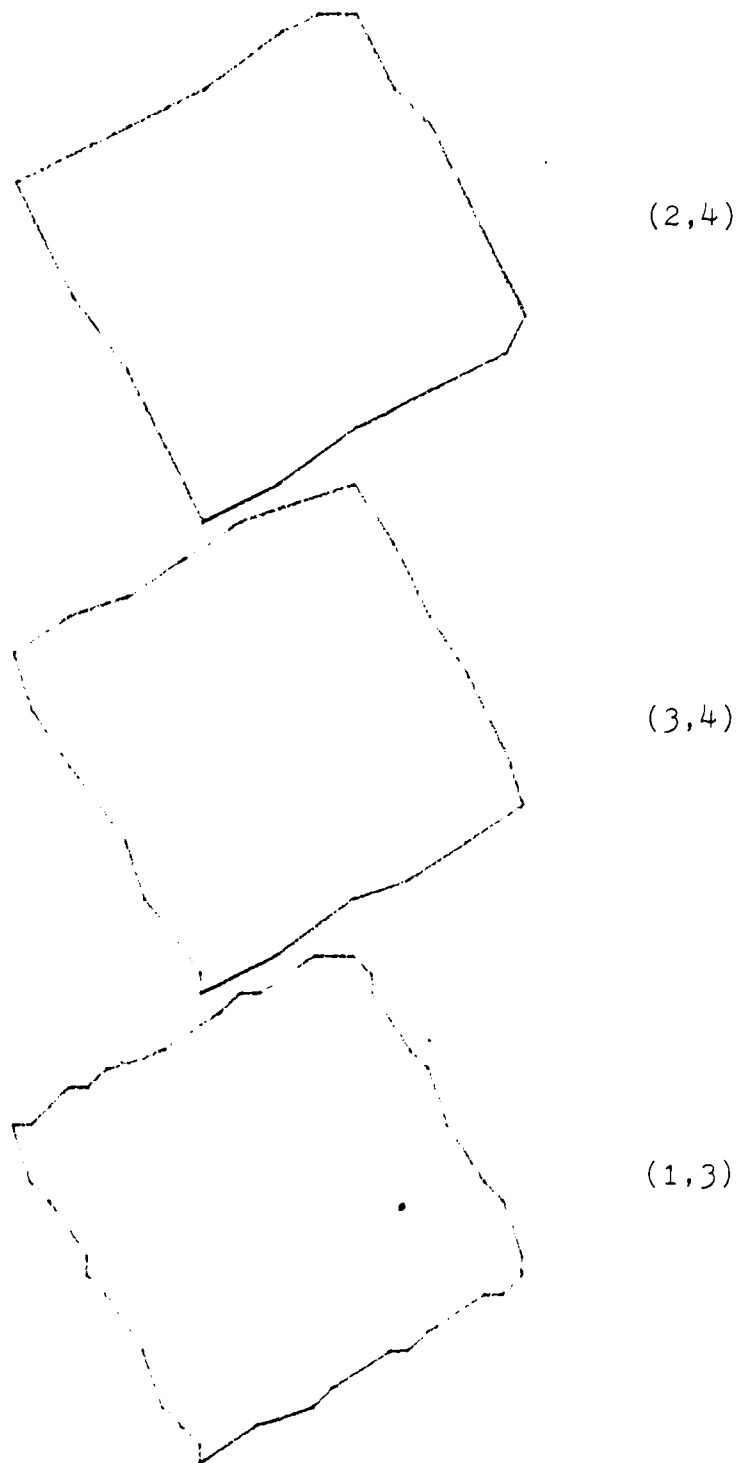
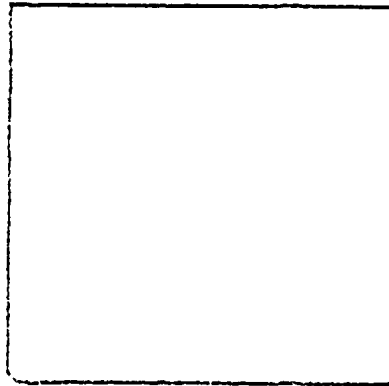
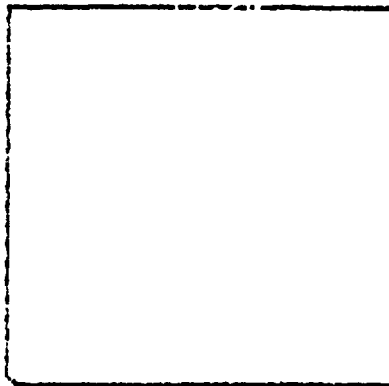


Figure B-22. SQUARE-30: (2,4), (3,4), and (1,3) Codes With
0.1 Inch Gridsize

(1)



(1,2)



(3)

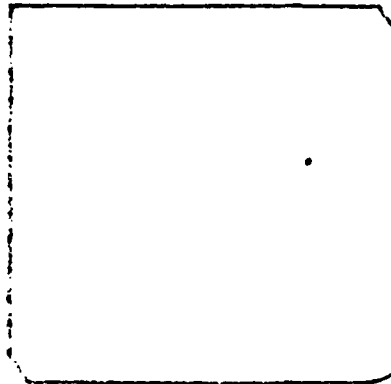


Figure B-23. SQUARE-C: (1), (1,2), and (3) Codes With
0.05 Inch Gridsize

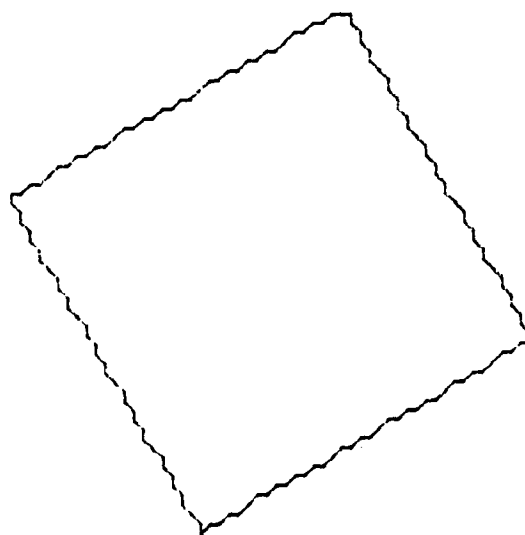


Figure B-24. SQUARE-30: (1) Code With 0.05 Inch Gridsize

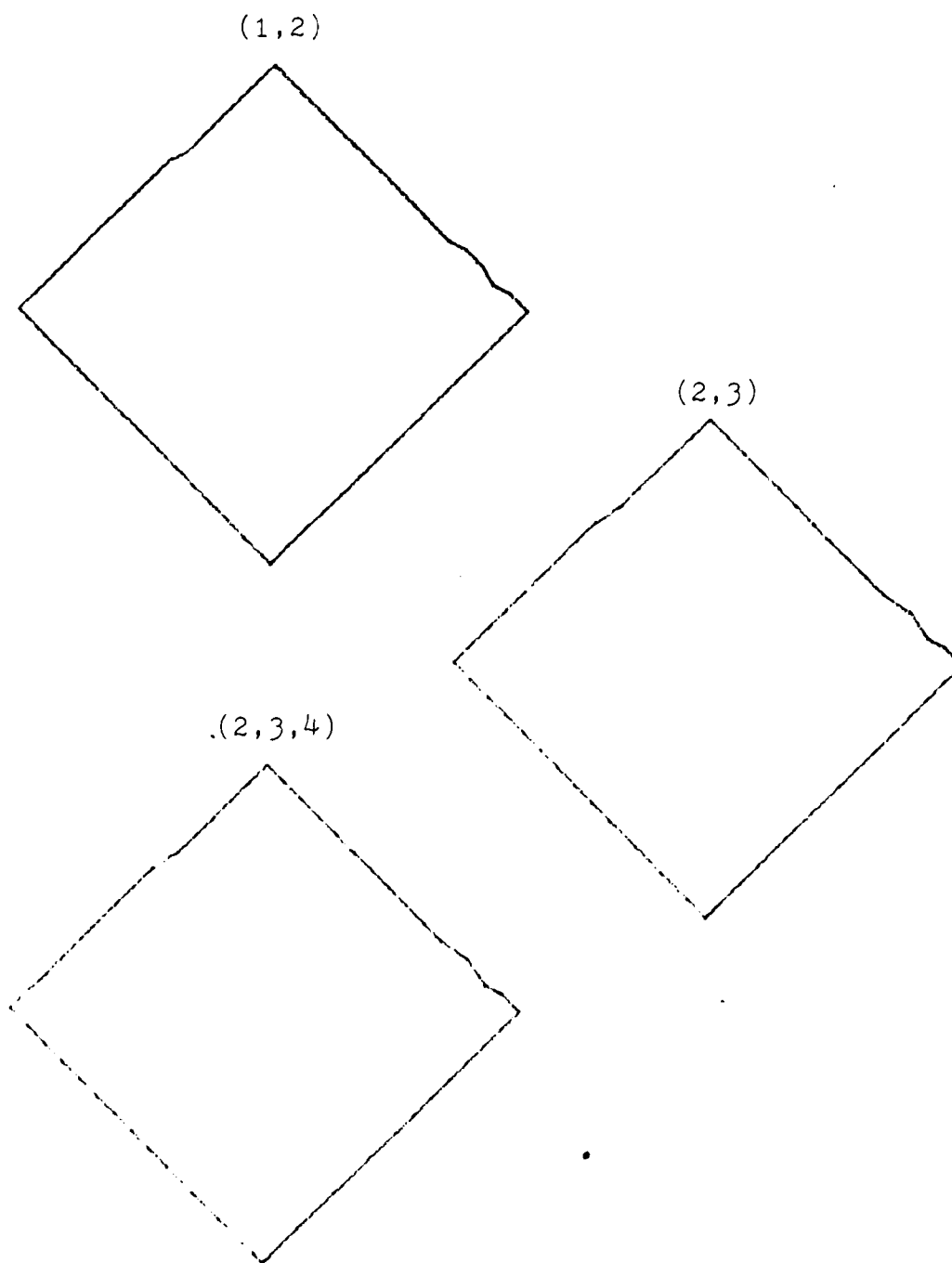


Figure B-25. SQUARE-45: $(1,2)$, $(2,3)$, and $(2,3,4)$ Codes
With 0.05 Inch Gridsize

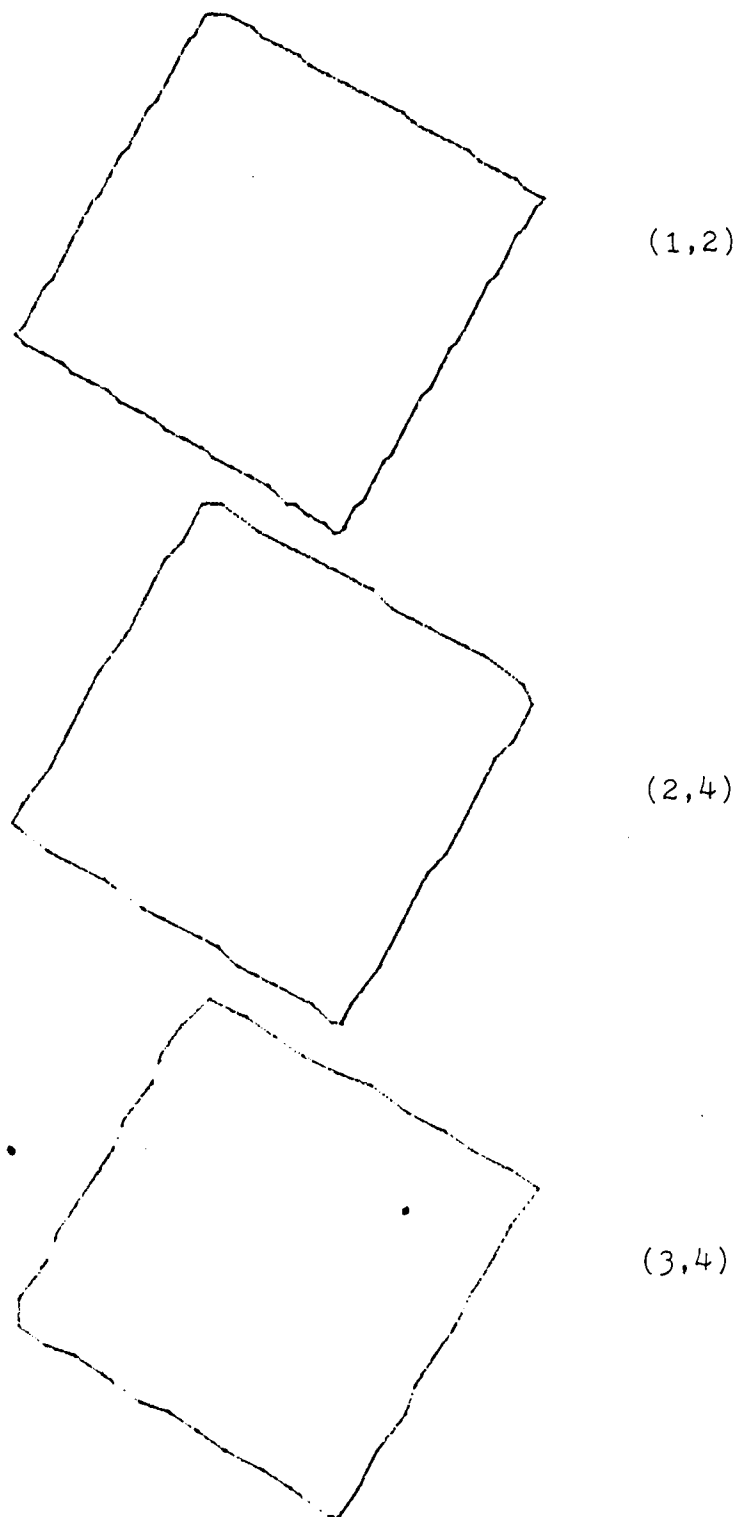


Figure B-26. SQUARE-60: (1,2), (2,4), and (3,4) Codes
With 0.05 Inch Gridsize

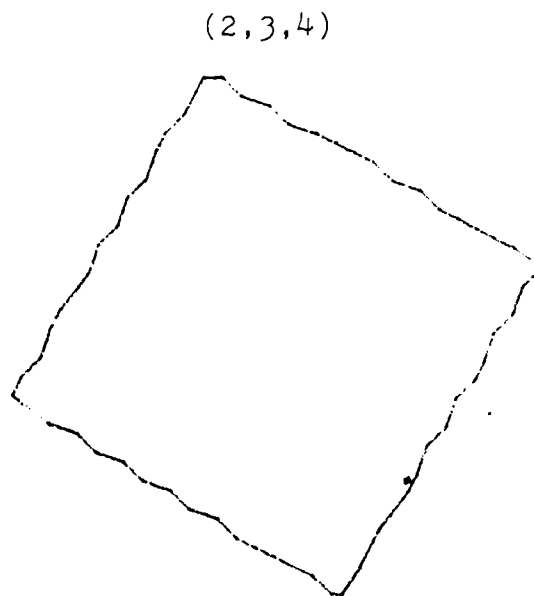
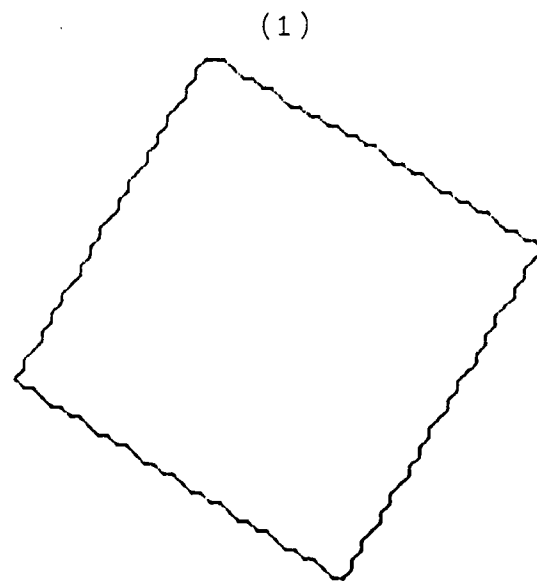


Figure B-27. SQUARE-60: (1) and (2,3,4) Codes With
0.05 Inch Gridsize

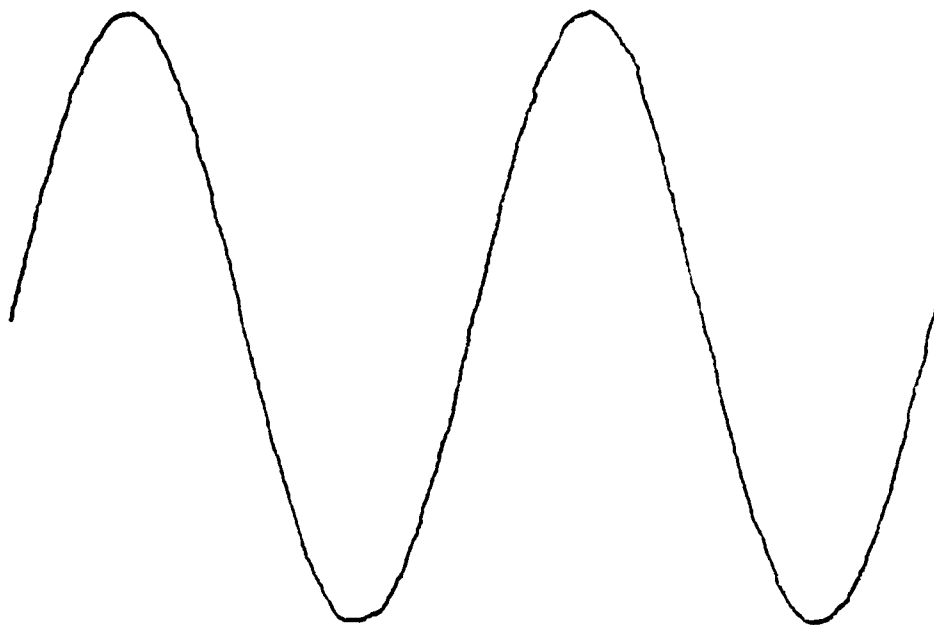


Figure B-28. Digitized Sine Wave Drawing

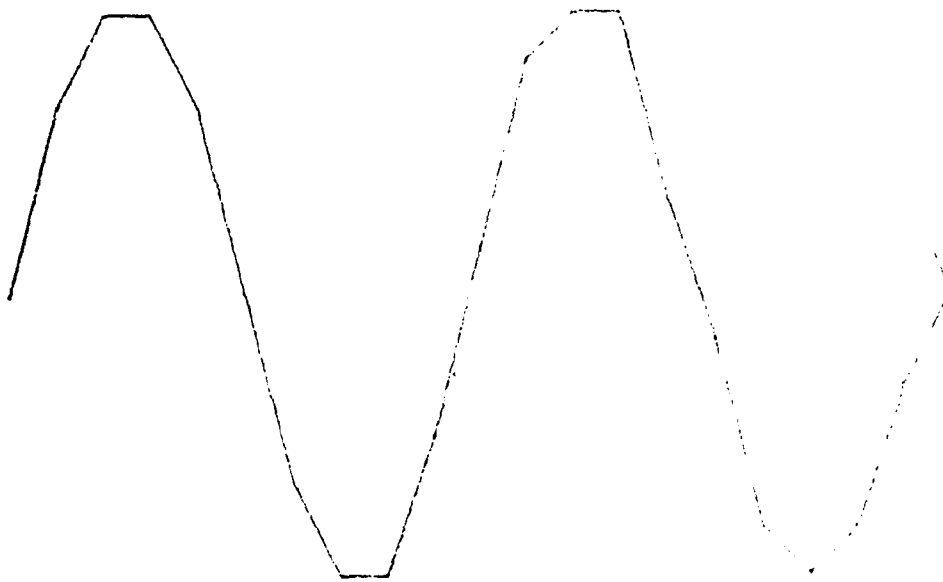


Figure B-29. SINE WAVE: (1,2,3,4) Code With 0.25
Inch Gridsize

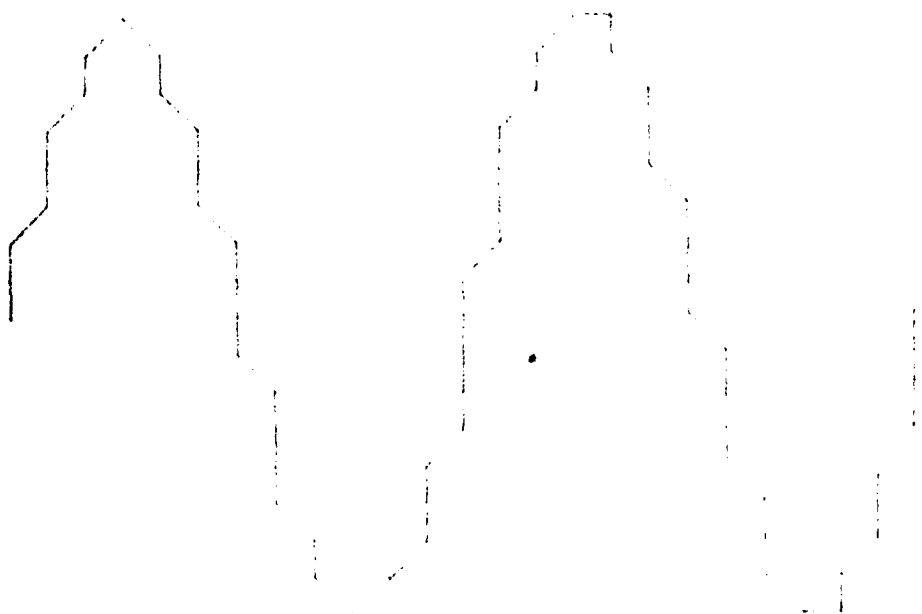


Figure B-30. SINE WAVE: (1) Code With 0.2 Inch Gridsize



Figure B-31. SINE WAVE: (1,4) Code With 0.15 Inch Gridsize

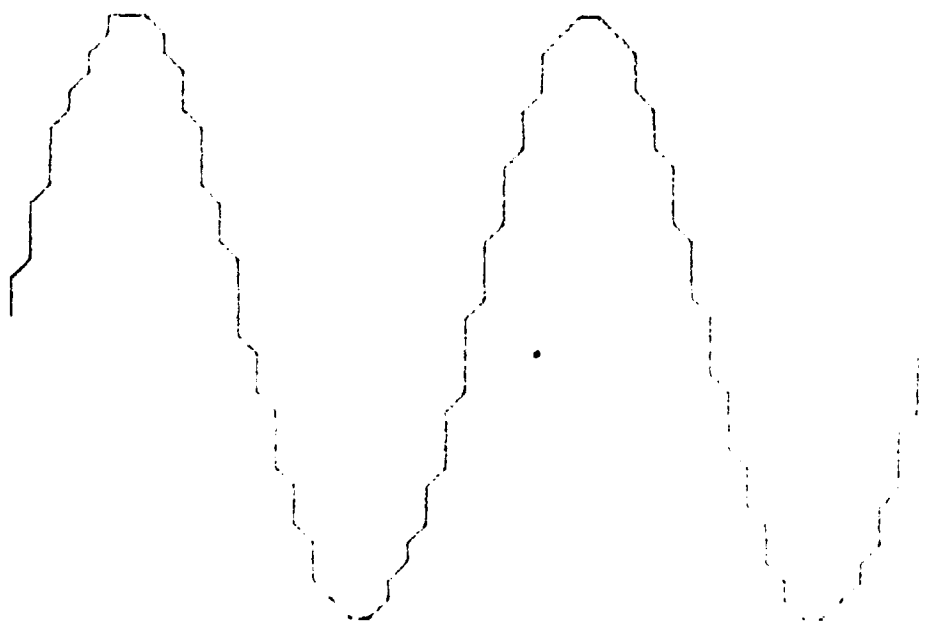


Figure B-32. SINE WAVE: (1) Code With 0.1 Inch Gridsize

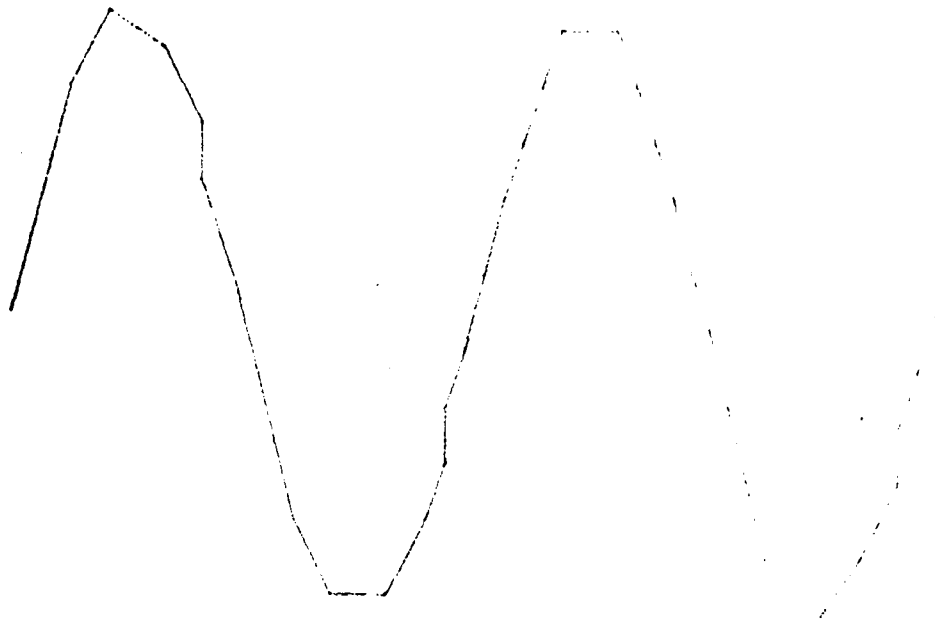


Figure B-33. SINE WAVE: (3,4) Code With 0.1 Inch Gridsize

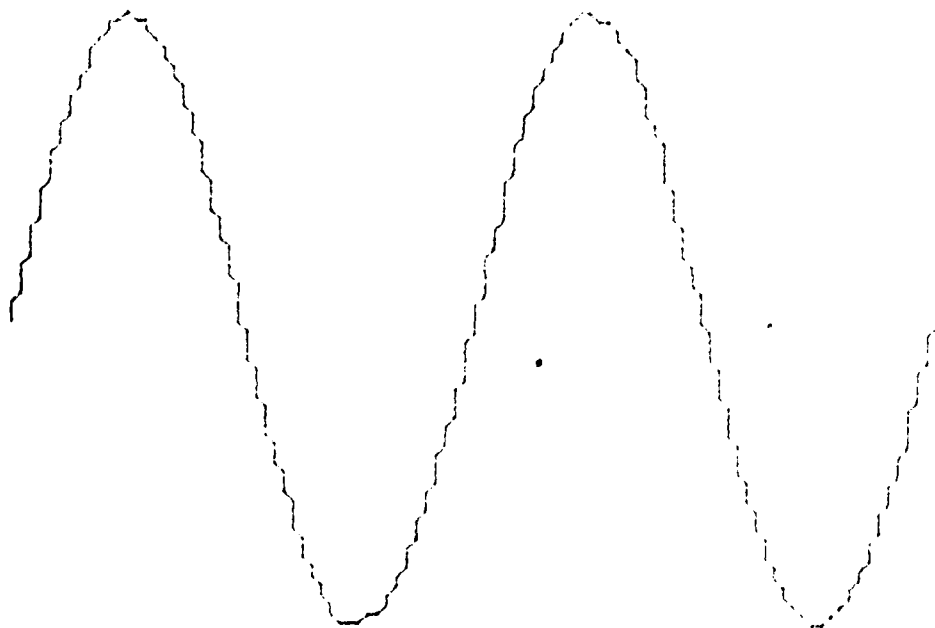


Figure B-34. SINE WAVE: (1) Code With 0.05 Inch Gridsize

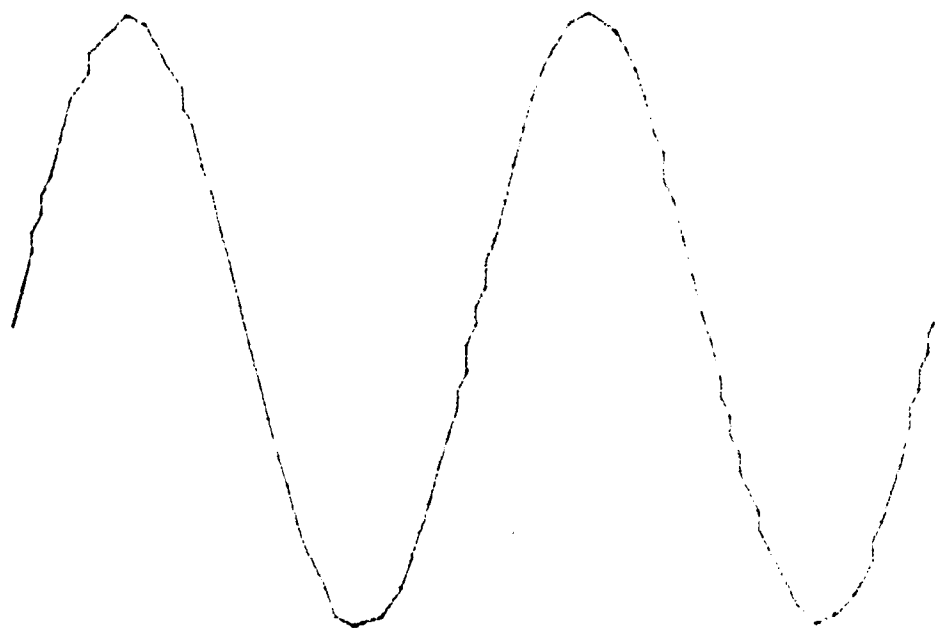


Figure B-35. SINE WAVE: (2,3,4) Code With
0.05 Inch Gridsize

hello

Figure B-36. Digitized Written Text Drawing

AD-A152 088

AN EXTENSION OF A MICROCOMPUTER BASED SYSTEM FOR
ANALYSIS OF LINE DRAWING. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. T A MORRIS
DEC 84 AFIT/GE/ENG/84D-48 F/G 9/2

3/3

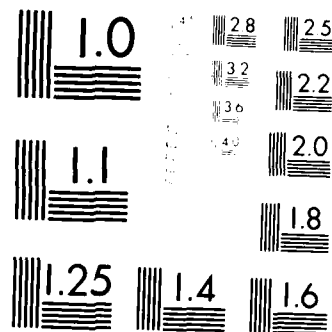
UNCLASSIFIED

NL

END

FILED

ENC



MICROCOPY RESOLUTION TEST CHART
 NATIONAL BUREAU OF STANDARDS-1963-A

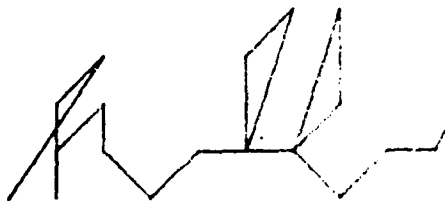


Figure B-37. TEXT: (1,3) Code With 0.25 Inch Gridsize

(1,3)

(1,2,3,4)



Figure B-38. TEXT: (1,3) and (1,2,3,4) Codes With
0.2 Inch Gridsize

(1)

(1,2,3,4)

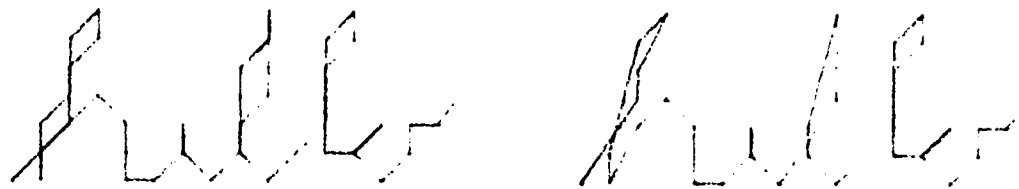
The image shows two handwritten versions of the word "Hello". The first version, labeled (1), is written in a simple, blocky style. The second version, labeled (1,2,3,4), is written in a more cursive, slanted style. Both are drawn on a grid.

Figure B-39. TEXT: (1) and (1,2,3,4) Codes With
0.15 Inch Gridsize

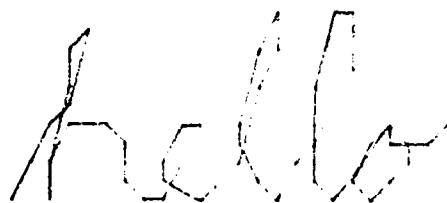
The image shows a single handwritten version of the word "Hello" in a cursive, slanted style, labeled (1,2,4). It is drawn on a grid.

Figure B-40. TEXT: (1,2,4) Code With 0.1 Inch Gridsize

(1)

hello

(4)

hello

(2,4)

hello

(2,3,4)

hello

Figure B-41. TEXT: (1),(4),(2,4), and (2,3,4) Codes
With 0.05 Inch Gridsize

Appendix C

This appendix is a user's manual for the software described in this thesis.

User's Manual

A list of the available programs and a brief statement of their purpose is shown below.

DIGITIZE : digitize a line drawing on the digitizer and store the data in a disk file.

PLOTFILE : plot the data stored in a disk file.

DIGPLOT : echo the points being digitized on the digitizer to the plotter.

LABELS : provide direct control of the plotter.

CHNCODE: compute the multi-ring chain code of a line drawing described by a series of coordinates stored in a disk file and output the coordinates of the nodes and the chain code to a disk file.

ERROR : compute the area error between the line drawings described by the data in two disk files, also computes the length of the line in each disk file and the number of bits needed to store the chain code.

PLOTCODE: plots a multi-ring chain coded version of a drawing as it is drawn on the digitizer.

A more complete guide to using each of these programs follows.

DIGITIZE

An example of program operation is shown below. Program operation will be shown as a sequence of steps in

which the user interactively communicates to the computer all necessary input data.

Step 1: In response to the CP/M prompt, A>, type DIGITIZE.

The computer will respond with:

PROGRAM TO PLOT DIGITIZED POINTS IN A FILE
SINGLE OR CONTINUOUS SAMPLING MODE (SG/CN) :

Step 2: If you want to digitize selected points on a drawing, enter SG, then go to step 5. If you want continuous sampling of the points on the drawing, enter CN and go to step 3.

Step 3: The computer responds with:

CONTINUOUS SAMPLING MODE SELECTED:
DEFAULTS ARE :

DELTA d = 20.25 inches

DELTA t = 20 msec

DELTA t = ? (20 to 32767 msec) (CR for default) :

User responds with the minimum sampling time required for the digitizer.

The computer then prompts for the DELTA d with:

DELTA d = ? (.001 to 20.25 in.. 1 = .001 in) (CR FOR DEFAULT) :

The user then responds with the minimum sampling distance.

Step 4: The computer prompts:

SWITCH NORMAL OR SWITCH FOLLOW MODE (SN/SF):

The user then enters SN for the switch normal mode or SF for the switch follow mode. In the switch normal mode, the DIGITIZE switch on the cursor must be depressed for points to be taken.

In the switch follow mode, the DIGITIZE switch acts as a toggle; if points are not being taken, pressing the switch causes the digitizer to start taking points, pressing it again causes it to stop.

Step 5: The computer then prompts:

THIS PROGRAM WILL WRITE OVER ANY EXISTING FILE
WITH THE SAME NAME
ENTER FILENAME FOR DATA POINTS (EX: A:DATA1.DAT):

The user then enters any CP/M acceptable filename. If there are no problems with the disk, such as the disk being full, the computer will respond with:

FILE OPENED SUCCESSFULLY

CONTROL KEYS ON THE DIGITIZER ARE:

Fa = RESET DIGITIZER MODE

Fb = CLOSE CURRENT FILE AND ASK FOR MORE

Fc = PEN DOWN

ANY OTHER KEY = PEN UP

NOW BEGIN TAKING POINTS

Step 6: The user now begins taking points. If the cursor is being used, then the DIGITIZE switch must be depressed; if the stylus is being used, then the DIGITIZE switch is activated by the pressure of writing.

The digitizer control keys are necessary for proper operation of the program. If you wish to change digitizer modes (single or continuous sampling or the DELTA t or DELTA d) then press Fa. Pressing Fc puts the pen down for the

drawing while any key other than a, b, or c puts the pen up. When the drawing is completed, press Fb to close the file.

An important fact to note is that the ERROR routine needs at least one point at the beginning of the digitized file to have the pen in the up position. Therefore, the user should press the digitize switch and then the Fc key in order to get a few points with the pen up at the beginning of the file.

Step 7: When the drawing has been digitized, the user should press the Fb key on the digitizer to close the file. The computer will respond with:

MORE TO DIGITIZE (Y/N)?

If there is more to digitize, press Y and the program repeats starting with Step 2.

If there is no more to digitize, press N. The computer responds with:

ALL DONE

The computer then returns to CP/M.

PLOTFILE

The program to plot a set of data points is invoked by:

A>PLOTFILE (CR)

The program will prompt the user for the number of files to be plotted. Beginning with the first file, the program

then prompts for the filename and the parameters for the
plotter. The plotter parameters are :

Line Type (integer 0-8) : The available line types are:

L0 _____
L1 _____
L2
L3 - - - - -
L4 - - - - -
L5 _____
L6 - - - - -
L7 - - - - -
L8 - - - - -

Scaling Factor: In order to give the user complete control over the size and location of the plotted drawing, the scaling factor and translation factors were implemented. The x and y coordinates of the input file are multiplied by the scaling factor, added to their respective translation factors, and output to the plotter. The scaling factor is a real number and must be entered as x.y (a value of .2 would be entered as 0.2). The coordinates in the files normally have units of 0.001 in. The plotter uses units of 0.005 inches for all numbers.

Therefore, a 1 in = 1 in scale for a digitized drawing would be:

$$X * (\text{in} / 1000) * (200 / \text{in}) = X * 0.2 \quad (\text{scaling factor} = 0.2)$$

Translation factor: The translation number is in units of 0.005 in (200 = 1 in).

DIGPLOT

The digitizer to plotter program is invoked by

A>DIGPLOT (CR)

The digitizer to plotter program prompts the user for all of the input it requires. The input required has been described in the instructions for DIGITIZE and PLOTFILE (except for the difference in the function of the digitizer keys shown below). The program is exited by entering a CONTROL-C from the keyboard.

The digitizer special function keys are defined as:

Fa : reset digitizer sampling parameters

Fb : reset plotter parameters

Fc : indicate pen down

Fd - prefix Fe : indicate pen up

LABELS

The plotter direct control program is invoked by

A>LABELS (CR)

The program prompts the user for all input. If a mistake is made in entering the command line then the only way to

correct it is to specify a repetition number of zero. The x and y increments are integer numbers with unit of 0.005 in.

A summary of the plotter commands follows:

O - set origin: the current pen location becomes the new origin

D - pen down: puts the pen down at the current location

U - pen up: immediately picks the pen up

H - home: moves the pen to the home location (lower left corner) and defines that location as the new origin

A - absolute plotting mode: all coordinates will be plotted with respect to the currently defined origin

R - relative plotting mode: all coordinates will be plotted with respect to the point plotted immediately prior to the point being currently plotted

Ln - set line type: define line type as n (see line type definition above)

Srnbsss_ - symbol plotting: plot ASCII character string sss with rotation r and height h ('_' indicates end of character string, b is a space) (r is an integer 1 - 4 and the rotation is: rotation = (r-1)*90 degrees, 0 degrees is straight right and the rotation angle is positive clockwise) (h is an integer 1 - 5 which corresponds to heights of: 1 = 0.07", 2 = 0.14", 3 = 0.28", 4 = 0.56", and 5 = 1.12")

T - self-test routine: perform self-test program

x,y - move pen to x,y: move the pen to x,y with respect to the origin or previous point (A vs R) with the pen up or down (U vs D) (x and y are integers sent as ASCII character strings)

CHNCODE

The operation of the multi-ring chain coding program is described by the following sequence of steps:

Step 1: To use the program, enter CHNCODE in response to the CP/M prompt.

Step 2: The program responds with:

THIS PROGRAM COMPUTES MULTI-RING CHAIN CODES

ENTER THE DIGITIZED DATA FILENAME:

At this point, enter the name of the digitized data file. The program then responds with:

ENTER THE NUMBER OF CODED FILES YOU WISH TO CREATE:

No more than 25 different files can be created at one time. Enter now the number of files you wish to create (entry must be an integer).

Step 3: The program now begins interactively obtaining the necessary data for each coded file. For example, to create a (1,3) code with a gridsize of .05 inches and a (2,4) code with a gridsize of .1 inches, the user would have entered 2 for the number of files to be created and the program would respond as follows:

ENTER CODED DATA FILENAME 1: (here the user

enters a filename, say CODE.CD1)
ENTER GRIDSIZE DESIRED: 50 (remember 1 = .001 inches)
ENTER THE NUMBER OF RINGS USED BY THE CODE: 2
ENTER THE RINGS USED BY THE CODE STARTING WITH THE LOWEST
(EXAMPLE FOR A (1,3,5) CODE: ENTER 1 3 5): 1 3

ENTER CODED DATA FILENAME 2: CODE.CD2
ENTER GRIDSIZE DESIRED: 100
ENTER THE NUMBER OF RINGS USED BY THE CODE: 2
ENTER THE RINGS USED BY THE CODE STARTING WITH THE LOWEST
(EXAMPLE FOR A (1,3,5) CODE: ENTER 1 3 5): 2 4

Step 4: The program now begins creating the coded files.
It provides the user with updates of its progress as shown:

CODING CODE.CD1 (program informs user that it is creating the first coded file)

CODING CODE.CD2 (program informs user that it is creating the second coded file)

COMPUTATIONS COMPLETE (program informs user that it is finished)

The program now returns the user to CP/M.

ERROR

The area error computation program is invoked by

A>ERROR (CR)

The program then prompts the user for all the information that it needs. As an example, consider the example used for the CHNCODE program. The sequence of events would be as follows:

Step 1: Call the ERROR program by entering ERROR (CR) in response to the CP/M A> prompt.

Step 2: The program responds with:

ENTER DIGITIZED DATA FILENAME: enter the digitized data filename

ENTER NUMBER OF CODED FILES TO BE ANALYZED: 2
ENTER CODED DATA FILENAME 1: CODE.CD1 (first coded file)
ENTER CODED DATA FILENAME 2: CODE.CD2 (second coded file)

Step 3: The program then starts analyzing CODE.CD1. It keeps the user informed on which file is being analyzed as follows:

ANALYZING CODE.CD1

The program then lists the output when the analysis is completed and proceeds to the next coded file.

A useful way to obtain the results for many files is to have the CRT output also sent to the line printer. This is easily done by invoking the CONTROL-P toggle from CP/M before calling the ERROR program. This prevents the user from having to sit in front of the CRT during program operation to obtain the ERROR results, since these results are also automatically recorded by the line printer.

PLOTCODE

To call the PLOTCODE program, the user simply enters

A>PLOTCODE (CR)

The program then interactively prompts the user for all necessary input data using similar menus to the CHNCODE and DIGPLOT programs. These menus prompt for the chain code ring levels and gridsize, the plotter translation and scale factors, and the digitizer sampling parameters. The

digitizer keys have the same meaning for PLOTCODE that they do for DIGITIZE except Fb does not close a file, it simply allows the user to start a new drawing.

BIBLIOGRAPHY

1. Freeman, Herbert. The Generalized Chain Code for Map Data Encoding and Processing. Technical Report CRL-59. Air Force Office of Scientific Research. June 1978.
2. Rock, Joseph R., Jr. A Microcomputer Based System for Analysis of Line Drawing Quantization Techniques. MS thesis, AFIT/GE/EE/83D-77. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1983.
3. Jones, Keith R. Grid Based Line Drawing Quantization. MS thesis, AFIT/GE/EE/82D-41. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1982.
4. Christensen, Eric R. Grid Based Line Drawing Quantization. MS thesis, AFIT/GE/EE/83D-16. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1983.
5. Saghri, John A. Efficient Encoding of Line Drawing Data with Generalized Chain Codes. Tech Report IPL-TR-79-003, Image Processing Laboratory, Rennsselaer Polytechnic Institute, Troy, New York, August 1979.
6. Thompson, Edward A. Grid Based Line Drawing Quantization. MS thesis, AFIT/GCS/EE/83D-20. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1983.
7. Freeman, H. "Analysis of Line Drawings", in J.C. Simon and A. Rosenfeld, Digital Image Processing and Analysis, Noordhoff, Leyden, 1977, pp 187-199.
8. Freeman, Herbert. Computer Processing of Line Drawings, Tech Report 403-30, Department of Electrical Engineering and Computer Science, New York University, Bronx, New York, May 1973.

VITA

Captain Thomas A. Morris was born on 25 July 1953 in Gilmer, Texas. He graduated from Gilmer High School in 1971 and enlisted in the Air Force in 1973. He attended Texas A&M University under the Airman Education and Commissioning Program and he received the degree of Bachelor of Science in Electrical Engineering. Upon graduation, he attended Officers' Training School and received a commission. He then served as a Communications-Electronics engineer with the 1842 Electronics Engineering Group, Scott AFB, Illinois, until entering the School of Engineering, Air Force Institute of Technology, in June 1983.

Permanent address: Route 6, Box 104

Gilmer, Texas 75644

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified		1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE		4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/ENG/84D-48	
5. MONITORING ORGANIZATION REPORT NUMBER(S)		6a. NAME OF PERFORMING ORGANIZATION School of Engineering	
6b. OFFICE SYMBOL (If applicable) AFIT/ENG		7a. NAME OF MONITORING ORGANIZATION	
6c. ADDRESS (City, State and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433		7b. ADDRESS (City, State and ZIP Code)	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION		8b. OFFICE SYMBOL (If applicable)	
8c. ADDRESS (City, State and ZIP Code)		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
11. TITLE (Include Security Classification) See Box 19		10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. PROJECT NO. TASK NO. WORK UNIT NO.	
12. PERSONAL AUTHOR(S) Thomas A. Morris, B.S., Captain, USAF			
13a. TYPE OF REPORT MS Thesis		13b. TIME COVERED FROM _____ TO _____	
14. DATE OF REPORT (Yr., Mo., Day) 1984 December		15. PAGE COUNT 210	
16. SUPPLEMENTARY NOTATION			
17. COSATI CODES FIELD GROUP SUB. GR 02 02		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Line Drawing Chain Codes Computer Graphics	
19. ABSTRACT (Continue on reverse if necessary and identify by block number) Title: AN EXTENSION OF A MICROCOMPUTER BASED SYSTEM FOR ANALYSIS OF LINE DRAWING QUANTIZATION SCHEMES Thesis Chairman: Kenneth G. Castor, Major, USAF			
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT CLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ OTIC USERS ☐		21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Kenneth G. Castor, Major, USAF		22b. TELEPHONE NUMBER (Include Area Code) 513-255-5533	
		22c. OFFICE SYMBOL AFIT/ENG	

This paper describes the extension of a microcomputer based system to analyze the performance of various grid based line drawing quantization schemes. The system is developed on a Heathkit H-89 microcomputer with a Hewlett-Packard 9874A digitizer and a Hiplot digital plotter. The capabilities of the existing system were expanded to allow real time digitizing/plotting operations and to provide for encoding and analysis using variable ring quantization schemes. Comparisons of specific drawings were then made for various quantization schemes based on a distortion metric (area between original and quantized image), a rate metric (number of bits in the quantization), and a subjective evaluation of the smoothness of the quantized image.

Results indicate that multi-ring codes usually produce less distortion in the quantized image than single ring codes while the single ring codes require less bits for storage. Also, neither metric is a good indicator of the smoothness of the quantized image.

END

FILMED

5-85

DTIC