1975

A Search for a Computer Graphics System for the Small-scale Computer User

Larry D. Holley
University of Central Florida

A SEARCH FOR A
COMPUTER GRAPHICS SYSTEM
FOR THE
SMALL-SCALE COMPUTER USER

BY

LARRY D. HOLLEY
B.S.E.E., University of Florida, 1969
M.B.A., Florida State University, 1973

RESEARCH REPORT

Submitted in partial fulfillment of the requirements
for the degree of
Master of Science in Engineering
in the
Graduate Studies Program
of
The College of Engineering
Florida Technological University

Orlando, Florida
1975

Abstract

A SEARCH FOR A
COMPUTER GRAPHICS SYSTEM
FOR THE
SMALL-SCALE COMPUTER USER

BY
LARRY D. HOLLEY

This paper presents a review of several computer graphics systems. The need for a universal graphics language is discussed and some typical languages are investigated. Several types of display terminals have been researched and their advantages and disadvantages have been listed.

A vector plotting routine has been programmed on a microprocessor system as a part of the research. A description of the vector plotting system is included in this paper. The system could be a "modular unit" for general purpose use with any alphanumeric raster-scan display terminal and a simple graphics language.

TABLE OF CONTENTS

INTRODUCTION

Since the mid-1960's there has been a growing demand for low cost computer graphics for the small scale computer user. This demand has been made apparent in the written works of van Dam and Michener (1). There has also been much research into the use of time-shared computers to support graphic displays. To add to the conflict of selecting large or small computers, there has been a problem with software or computer language. Should there be a universal graphics language? Should each graphics system use its own special language or should a higher order language be developed; or does it already exist? Finally, what type of display terminal is best suited for computer graphics?

This paper will briefly review the history of computer graphics and describe several earlier systems. The need for a Universal Graphics Language is discussed and several professional opinions are stated on how close we are to having a universal language. Several graphic languages and systems such as LOGO, GOLD, and the Terminal Control System will be described. There will also be a discussion on different types of display terminals and

their advantages and disadvantages.

To further the research of computer graphics, this writer has programmed a point and line drawing routine on the Intellec 8 microprocessor. This paper offers a description of this system and how it can be a "plug-in" or "modular unit" for general purpose use with alphanumeric raster-scan display terminals and any simple graphics language.

CHAPTER I

COMPUTER GRAPHICS

The sophisticated computer has evolved rapidly during the last 25 years replacing many mechanical and analog devices such as calculators and controllers. Computer graphics has recently come to the forefront due to high speed circuitry and modern CRT's. Edwin L. Jacks describes the way Research Laboratories (2) of General Motors Corporation started working in the computer graphics field. In the late 1950's, the Research Laboratories addressed the question, "Could computer techniques really improve the design process?" To answer this question, a study was started on the potential role of computers in the graphical phases of design. Prototype hardware and software components were developed to investigate the problems of image processing. A 740 cathode-ray tube recorder attached to the 704 computer was already being used to plot results of engineering computations. It satisfied the requirement for graphical output. The associated 780 display unit provided a graphical on-line display which, along with a simple switchboard, became an elementary man-machine console. A program-controlled film

scanner was devised using the 740 recorder; a photocell detector was substituted for the film magazine, and its output signal was connected to a computer sense switch. With this breadboard setup, lines on film could be digitized under program control. Programs were written for graphic input and output and for manipulation of images in three dimensions. These early software and hardware components were integrated into an operating system that demonstrated the feasibility for using the computer as an aid in the graphic design process.

On the basis of this early feasibility demonstration, the decision was made to establish a more comprehensive laboratory for graphical man-machine communication experiments. The facilities were to permit the computational power of a large-scale digital computer to be brought to bear on the problems of graphical design in a manner which fully recognized the importance of the man in design. This project on Design Augmented by Computers became known as DAC-I.

The initial goal of the DAC-I Project was the development of a combination of computer hardware and software which (a) would permit "conversational" man-machine graphical communication and (b) would provide a maximum programming flexibility and ease of use for experimentation. This goal was achieved in 1963.

CHAPTER II

GRAPHIC LANGUAGES

Techniques

Techniques for computer manipulation of graphical information have grown rapidly in the past few years and have led to the development of what is called "graphic languages" (3). One of today's most unanswered questions is what type of computer language is best suited for graphic displays. This chapter discusses the merits of higher level languages versus specialized assemble languages. A typical high-level language is described as an example and the chapter is concluded with a debate over the issue of a universal language.

High-Level Versus Assembly Language

The opinion as to whether assembly language or compiler-level language should be used has changed over the brief life-span of the computer business. Originally one would have worked exclusively in assembly language or machine language. The compilers that existed were very inefficient and very slow, and they produced very slow running codes. However, according to Abrams and Stein (4), compiler design has improved and today a good efficient

compiler will produce machine language codes almost as good as that which could be constructed by the most experienced programmer. If this claim is true, there appears to be no reason for ever programming in assembly language. Writing in a higher-level language involves much less work than does writing in assembly language.

Part of the decision depends on the exact kind of operation you are attempting to program the computer to perform. If it is the type of operation that the higher-level language was designed to facilitate, then it certainly should be done in the higher-level language. Languages like FORTRAN exist to make algebraic scintific operations easy. A language like COBOL is designed for business applications, and special-purpose languages exist for many applications. There is currently no widely accepted higher-level language that can be used for making sketches and graphs on CRT's. Efforts have been made to use various specialized systems with their own dedicated language but not enough time has elapsed to make an objective evaluation of the success of such efforts.

A General Purpose Language for Graphical Applications

High-level graphical languages are needed to simplify the programming of graphical facilities, according to Robin Williams (5) of the New York University. Programmers like to deal with input/output from a logical point of view

with no regard to equipment. Williams presented a paper at the 1972 IFIP Working Conference on Graphic Languages describing a high level extensible language that allows the user to define new data types for simplifying subsequent programming. Using his language, objects can be created and edited by working interactively. Motions and other functions, such as interpolation, that apply to an object, can be described in the written language or interactively as well. Thus, the user can create his own suitable environment and depending upon the nature of the problem, he can work with a combination of both methods. Such flexibility at a high level should simplify the use of graphical facilities. Applications of Williams' language are listed in the next chapter along with his language.

To summarize this chapter, and to emphasize the dilemma of graphic languages, the following section was extracted from the IFIP Working Conference on Graphic Languages at Vancouver, Canada, May 22-26, 1972.

Universal Graphic Language

During a panel discussion entitled "Are We Anywhere Near a Universal Graphic Language", A. van Dam (6) of Brown University answered "No". He remarked that everyone has totally different ideas about what a graphic

language is. Van Dam has written:

> We talk about, for example, line drawing interactive graphics, and we find that even that is highly specialized: people who work with storage tubes, versus people who work with refresh displays, those who have intensity modulation versus those who don't. Or those who work with refresh displays, but those which operate on the video technique rather than with a conventional line drawing technique . . . . All very different disciplines, each with their own jargon, their own set of literature, and their own savants. Now you understand why I say "no". It's going to be impossible to . . . . agree even on the difinition of terms, let alone on what would make for a useful, constructive language.

Van Dam closed by emphasizing that language is very much in the eye and mind of the beholder and that the panel should not worry about the way things look. Whether a set of subroutines is called a language or not, whether we want to extend an existing language, or whether we wish to define a new special purpose language, is not important. He emphasized the importance of the functions desired in a language or system.

William M. Newman (7) of the University of California, Irvine, took a different approach to the question:

> My answer is: yes, we do have a universal language for interactive graphics and it's called FORTRAN IV. I don't have to explain why that's true, it just happens that time and again you see people proposing graphics languages . . . . all based on FORTRAN IV. I regard it as one of my principal aims to produce something better than FORTRAN IV. This is not very difficult, but having produced it, how do you get the thing accepted?

> People like myself are constantly being accused of avoiding the real world, of spending our time in ivory towers producing one language after another. When you get out and talk to people in the real_world, you find out their reasons for sticking to FORTRAN are not efficiency and things like that; it's their conservatism and lack of courage to make the switch to something better. Even if efficiency is the reason, what does this really mean? It means that FORTRAN runs faster on their machine. The solution to this problem is not to go on using FORTRAN, but to build better machines. This doesn't apply just to graphics, but it happens to be a particular problem in graphics.

Newman concluded by pointing out that there is more to language design than just deciding what functions one would like to have. It is clear that one must try to achieve some degree of simplicity, something rarely done in graphic languages. Almost all graphic languages are needlessly complicated which leads to errors, misunderstanding, misuse, and eventually to discarding the whole system. Newman thinks that simplicity is more important than efficiency in any language.

The panel discussion resulted in the conclusion that a universal language is a forelorn hope, but that a computer graphics is possible.

CHAPTER III

GRAPHIC SYSTEMS

Versatility

The systems and languages in this chapter are described to point out the versatility existing today in the field of Computer Graphics. There are many unique systems with their own languages, each designed to offer a low cost, reliable method for displaying pictures on a CRT screen via computer control.

The IBM 1130/2250

The conversational Graphic Data Processing System that should be discussed first is the IBM 1130/2250 (8,9). This system was one of the first units that offered a systematic approach for computer control of a CRT.

The IBM 2250 display unit is addressed by means of a 1023 X 1023 Cartesian coordinate system whose origin (0,0) is located at the lower left of the tube. Point (1023, 1023) is at the upper right. The distance between any two adjacent horizontal or vertical points is called a "raster unit". When a particular point is addressed, a circular area around that point is illuminated by the

electron beams.

Graphic orders executed by the IBM 2250 cause the electron beam to move from one addressable point to another. The beam can be moved in a "blanked" or OFF fashion, in which no illumination occurs, or in an "unblanked" or ON state, which causes all of the intermediary phosphors to be energized. Movement from one point to another in the ON condition gives the visual appearance of a straight line.

Since the light produced by the energizing of the phosphors fades within a fraction of a second, beam movements must be continually repeated to provide a flicker-free image for human perception. The repetitive redrawing of images on the screen, called "regeneration", can be executed automatically once initiated by the programmer.

Orders for moving the electron beam are fetched from IBM 1130 core storage, where they are stored as 16-bit words. Certain orders occupy two contiguous words. Nongraphic orders used for control and logical purposes, and for maintaining a conversation with the 2250, are also fetched from 1130 memory. The 2250 obtains its orders in sequence from 1130 core. The 2250 CRT has a graphic deflection yoke for creating grosser movements of the beam and a character deflection yoke for generating symbols. Accordingly, the 2250 operates in both Graphic and Character mode.

There have been many improvements to computer graphics since the IBM 1130/2250 system was implemented. One such improvement has been the development of separate data structures that take some of the workload off of the master computer. Robin Williams (5) described a typical simple display structure that could be used with his high-level extensible language.

Display Data Structure

Many practical applications of computer graphics equipment involve large quantities of data, but have very simple graphical structural requirements, according to Williams (5). Examples are the production of engineering drawings for integrated-circuit masks, electronic drawings, logic drawings, point-to-point wiring diagrams, etc. The graphical requirements are usually to store the drawings in terms of points with move and draw commands and to allow editing. If all the points are stored explicitly, a structure as shown in Figure 1 is sufficient. In this case, one display list or table of x, y, z coordinate data is maintained. This table consists of POINT type data and the operation is simply to add points to the table or to delete points from the table.

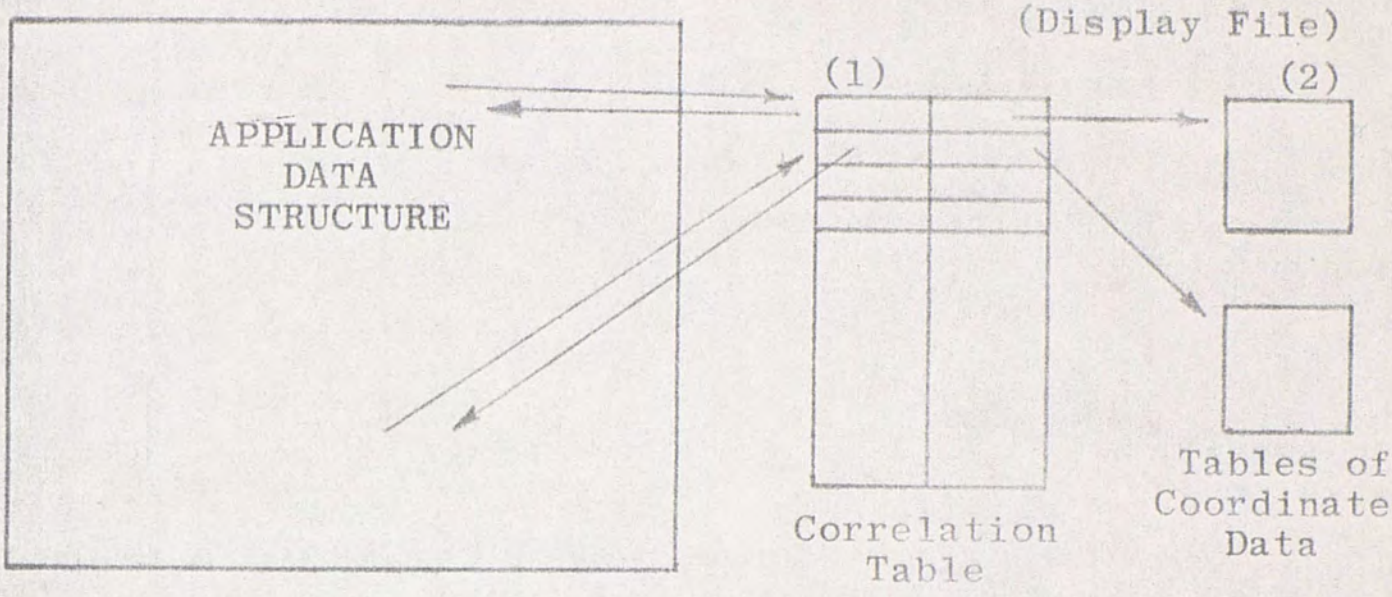


Figure 1. A Simple Display File Structure

An operation that is frequently needed is to make a copy of any section of the table. This operation is required if a part of the line drawing is to be used in another position, possibly with a different scale. Subroutines are called to process the coordinate data and add the new values to the end of the display table. A table for the beginning and the end of these common symbol descriptions would serve to identify them. A correlation table is kept to relate the graphical descriptions to an application data structure. The advantage of this structure is that the points on the screen have exact values in the computer, which makes connectivity, intersection and other calculations relatively straight-forward. The disadvantage is that it requires a lot of memory space or buffer space if the information is to be displayed, since every copy

of the object must be stored explicitly.

Many of the latest graphic display systems use this separate data structure approach.

The GOLD System

The Graphical On-Line Design (GOLD) System developed at RCA Laboratories(10) is an example of the separate data structure display file approach. The system operates in a stand alone manner on a small computer, yet maintains a large ring type data structure partitioned for secondary storage. In the GOLD system, all editing functions (move, copy, delete, etc.) are performed on the data structure. The resultant changes to the viewed picture are generated by a picture compiler and placed in the display file to appropriately change the viewed picture. The GOLD system utilizes a small computer, a disc storage unit, and a re-fresh display device. The system is being used in the design of circuit artwork at RCA Laboratories. Attempts have been made in the GOLD system to create a flexibility ring type data structure that can efficiently store data and from which data can be rapidly accessed. The power of the picture compiler lies in its ability to operate on a very large ring type data structure, totally in a small computer environment, to produce pictures rapidly such that the interaction speed is high.

Graphical LOGO System

William M. Newman has developed a system that offers many advantages over time-shared multi-access units like the IBM 1130/2250 (11, 12).

The system was designed around the LOGO language. LOGO is essentially a string processing language, contrasting sharply with arithmetic processing languages generally used in graphics systems. LOGO combines the simplicity needed by the beginner with powerful structures needed by the serious programmer.

The use of display procedures in this system required making only three types of additions to the language. These are the additions:

1. Primitive graphical functions for specifying lines, points, etc.

2. Operators for specifying transformations such as translation and scaling.

3. Display procedure calls.

The graphical LOGO system was implemented on an Interdata Model 4 microprogrammed computer, equipped with a writeable control store. This machine is connected by a high-speed parallel interface to an IMLAC PDS-1 display computer. The system organization is shown in Figure 2.

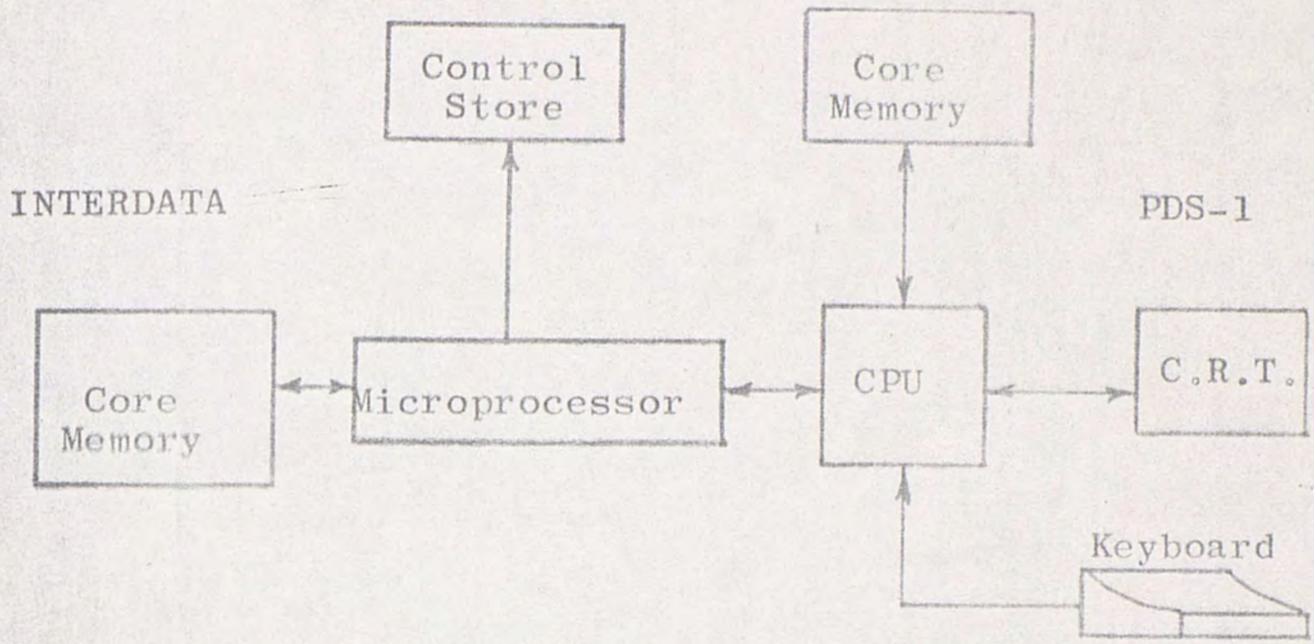


Figure 2. Block Diagram of LOGO System

Microcoding has enabled large sections of the LOGO interpreter to be accommodated in high-speed control memory. This effectively transforms the Interdata computer into a special-purpose LOGO machine. Because of the restricted size of the control store, some parts of the interpreter, including the editor, were executed from core memory.

The technique described here can be applied successfully using any one of a number of languages, according to Newman (11, 12). An algebraic language, provided with more powerful data structure facilities than the simple strings of LOGO, could be very valuable in scientific work.

The advantage of LOGO lies in its extreme simplicity and its suitability for use by non-scientific and novice users.

In a continuing effort to find a computer graphics system suitable for everyone, the next section gives a

a preview of Tektronix' answer to the dilemma.

Terminal Control System

John A. Mead(13) of Tektronix, Inc. agrees that one of the major difficulties in the development of Computer Graphics has been the lack of standard basic graphic software. As a result, there has been a tendency to redevelop the basic software for each installation and, in some cases, for each application. In the past, this software has often been oriented towards one system, applicable to only one type of terminal, and frequently had peculiar features facilitating a particular application and precluding all others. The software thus developed is often too complex for the occasional user to use conveniently and frequently too inflexible for the needs of the sophisticated programmer. As a result, graphic application software using such a base tends to have limited use and life.

To meet the needs of the different users and the multiplicity of the systems, Tektronix (13) developed the Terminal Control System. The Terminal Control System is a comprehensive set of functionally modular subroutines which allows essentially terminal-independent programming. The user needs only to select the proper modules at load time. The design is basically system and computer independent and allows the experienced programmer to work at the basic terminal level and also provides the facilities for the occasional user to operate easily at the conceptual

level. Properly written programs using the Terminal Control System should function with little or no modification on different terminals when loaded with the modules supporting the device.

The limitation of this system is the influence of Tektronix tube terminals on the systems' design. The unit has not been programmed for non-storage terminals.

CHAPTER IV

CRT TERMINALS

Characteristics

CRT Terminals perform two basic functions: they allow you to place data into a computer, and they display output data from a computer. This chapter will review the characterisitics of the display function of terminals. And since this paper is concerned with computer graphics, this chapter is focused on the picture or line drawing capabilities of CRT terminals.

CRT Hardware

Manufacturers sell two basic types of CRTs for alphanumeric display: raster scan and storage tube. The most common CRT type uses the raster scan principle; the same principle used for television sets. One company, Ann Arbor Terminal, even uses a standard TV monitor and a character generator to produce characters on a screen. This arrangement requires some special additional hardware to keep the information displayed on the screen. Character readability is provided by refreshing the same information on the screen 60 times per second, based on 60Hz line

frequency. On some CRTs, an available option allows refreshing certain parts of the information to provide a blinking effect. These facts have been obtained from the January 1975 edition of Digital Design Handbook (14).

Refreshing requires storing information that keeps circulating through the character generator section in order to rewrite the characters in the same locations on the screen. Different manufacturers store the information in different ways. Some of the most popular storage elements are delay lines, and semiconductor and core memories. Since one of the noticeable effects of refresh is image flicker, if you expect an operator to sit and use a CRT for several hours, then you should choose a CRT with the least flicker and with a pleasing contrast.

Since raster scan CRT's require constant refresh, choose them for graphics generation, because a graphic image that needs to be changed displays that change in the next refresh cycle from modified inputs. In contrast, the storage tube CRT, which offers most of the external appearances and characteristics of the raster scan CRT, does not require refreshing. Instead, the storage tube's screen is coated with a phosphor that retains the image until the entire screen is cleared. This construction requires much more data transmission to the CRT for each new image display. Image changes require retransmitting the data for the whole image; raster scan tubes require retransmitting only the

changed bits. Since a storage tube does not flicker, its use tires the operator less. A storage tube CRT can also use dedicated storage to make changes appear only as you transmit the changed data from the CPU to the CRT controller. Then the CRT controller can update its image, clear the screen and display the new image.

Direct View Storage Tubes

(DVST's) exhibit slower writing rates than conventionally refreshed or TV tubes, and graphic elements which are less bright (and of poorer contrast with respect to the glowing background) but flicker-free; an unbeatable information density results, allowing, for example, 4000 tiny but still legible flicker-free characters. Based on research by van Dam and Michener (1) , we discover that it is entirely reasonable to time-share driving electronics between two DVST's, allowing split screen techniques (say, one for static overview and one for dynamic local editing) at an incremental cost of under $3,000. The same self-storing characteristic makes selective erasure impossible, and that, coupled with long writing times, in turn makes these tubes unsuitable for dynamic changes. For changes to low density pictures, however, rewriting may be accomplished in several seconds.

Computer Controlled Terminals

One of the least popular techniques for controlling

the CRT is to use a dedicated, small, general-purpose computer. Although the expense limits the use of this arrangement, the expense is often warranted for graphics, because the computer can perform some of the minor calculations and programming, thus relieving the host processor of a considerable part of the bookkeeping. In particular, placing a memory or memory transfer between the host CPU and the CRT processor allows loading the data into the CRT at core speeds and processing by the dedicated processor. Also, in a system consisting of several terminals tied to one host processor, the CRT processor can reduce the host processor's workload, allowing it to handle other activities. If the general purpose processor requires considerable software to implement the system, this need adds to the hardware cost. However, a software-implemented system does allow easy modifying or tailoring to meet specific requirements.

Based on these recommendations by the editors of Digital Design Handbook (14)and the other requests for a separate display data structure requiring a dedicated processor, the investigation in the next chapter was done to further the search for computer graphics for the small scale computer user.

CHAPTER V

MICROPROCESSOR LINE DRAWING FUNCTION

Intellec 8

This chapter concerns a study of a program written on the Intellec 8 for receiving data, reformatting and storing into RAM, and outputting to a refresh memory, raster scan terminal, or large computer data file. For the purpose of this research, the window size for making a picture is limited to 64 x 64 rasters.

The Intellec 8 is a low cost computer system, designed to simplify the development of microcomputer systems which employ Intel 8008 CPU chip processors.

The Intellec 8 uses the 8008-1 as its central processing unit. The 8008-1 has a basic cycle time of 12.5 microseconds. The system contains a control console and provides 16K, 8-bit words of read-write programmable memory(15). The software used for programming the Intellec 8 is MCS-8 assembly language (16).

The input/output device used for this demonstration was a standard teletype terminal. Note that the data had to be converted to ASCII for this purpose.

Input Data Format

The Intellec 8 was programmed to accept specially formatted input data representing vectors. The restrictions are that each vector must be described by four coordinates; 2 octal characters per coordinate. And the coordinates must be in the order: X (initial), Y (initial), X (final), Y (final). Each vector can have a slope of positive or negative and can be described from left to right or right to left.

The only control commands needed are "P" for print data file, and "C" for clear data file.

Data Format and Storage

The unique feature of this program is the way in which data is stored in memory. A block of 512 memory locations are reserved as the data file for the (64 x 64) or 4096 raster points. Each bit in memory represents one raster point. If the memory bit is a "logic one" then the point on the CRT is to be intensified; if the bit is a "logic zero" the point will be blank. These memory locations are arranged so that they can be shifted out serially in the proper order to be stored in the buffer register of a raster scan type CRT. Figure 3 shows the memory locations in the positions they will be displayed on a screen. ie; memory location (100) represents 8 raster points at the lower left portion of the viewing window. Further, the 8 bits of each location are raster

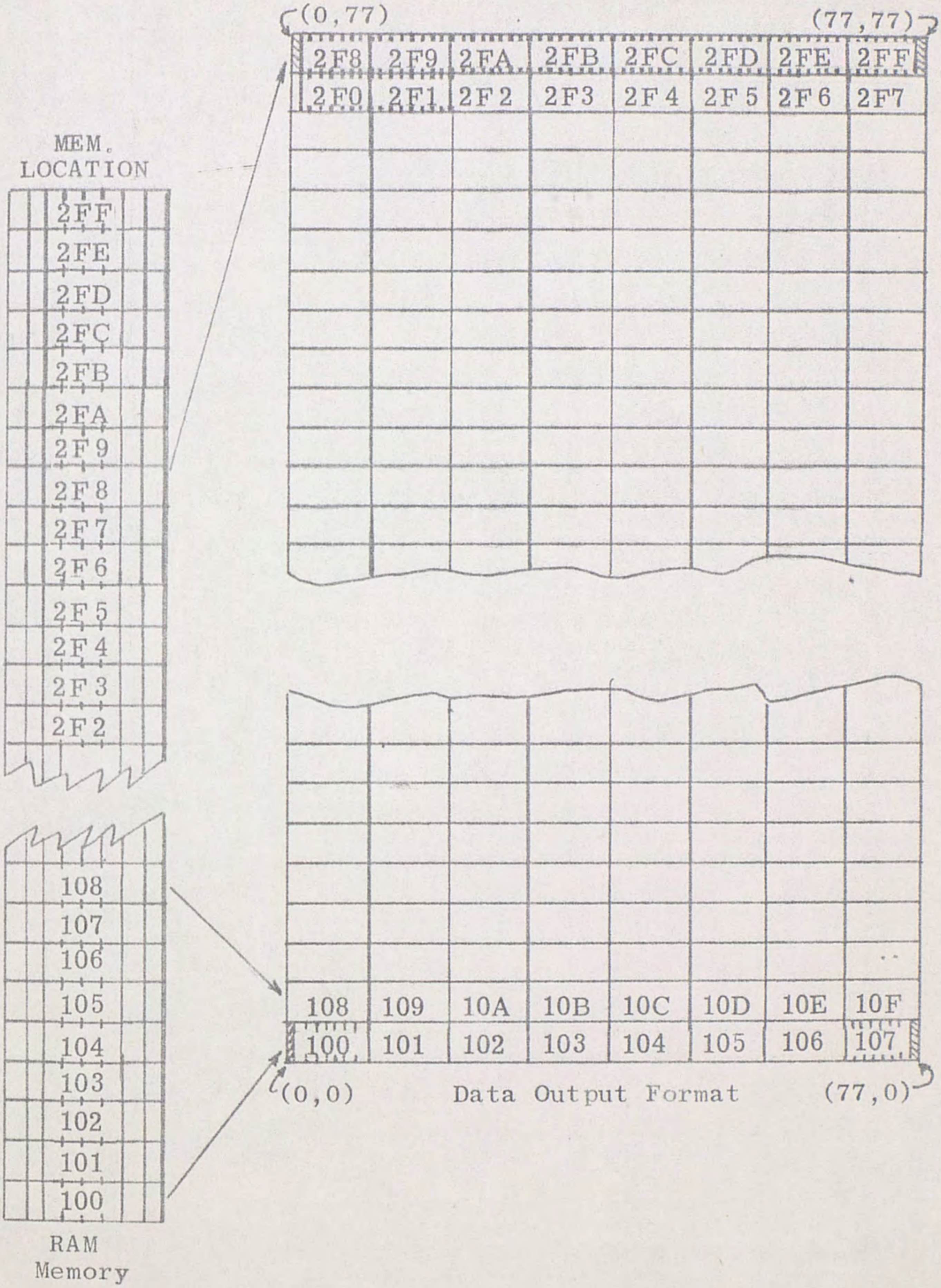


Figure 3. Relationship of Memory to Output

points in the "x" or horizontal plane. Therefore, only 8 locations in memory are required to create one 64 point line across the screen.

Once the input coordinates are received, the microprocessor determines the length, direction, and slope of the vector. Using very simple MCS-8 assembly language, (see Appendix for program listing), the Intellec 8 sets up a loop and stores a "one" bit in the appropriate memory locations to form a vector. Many vectors can be stored into memory by simply inputting their coordinates.

Whenever an ASCII "P" character is received, the microprocessor will send out all 4096 raster points, serially. When an ASCII "C" character is received, the memory file is cleared.

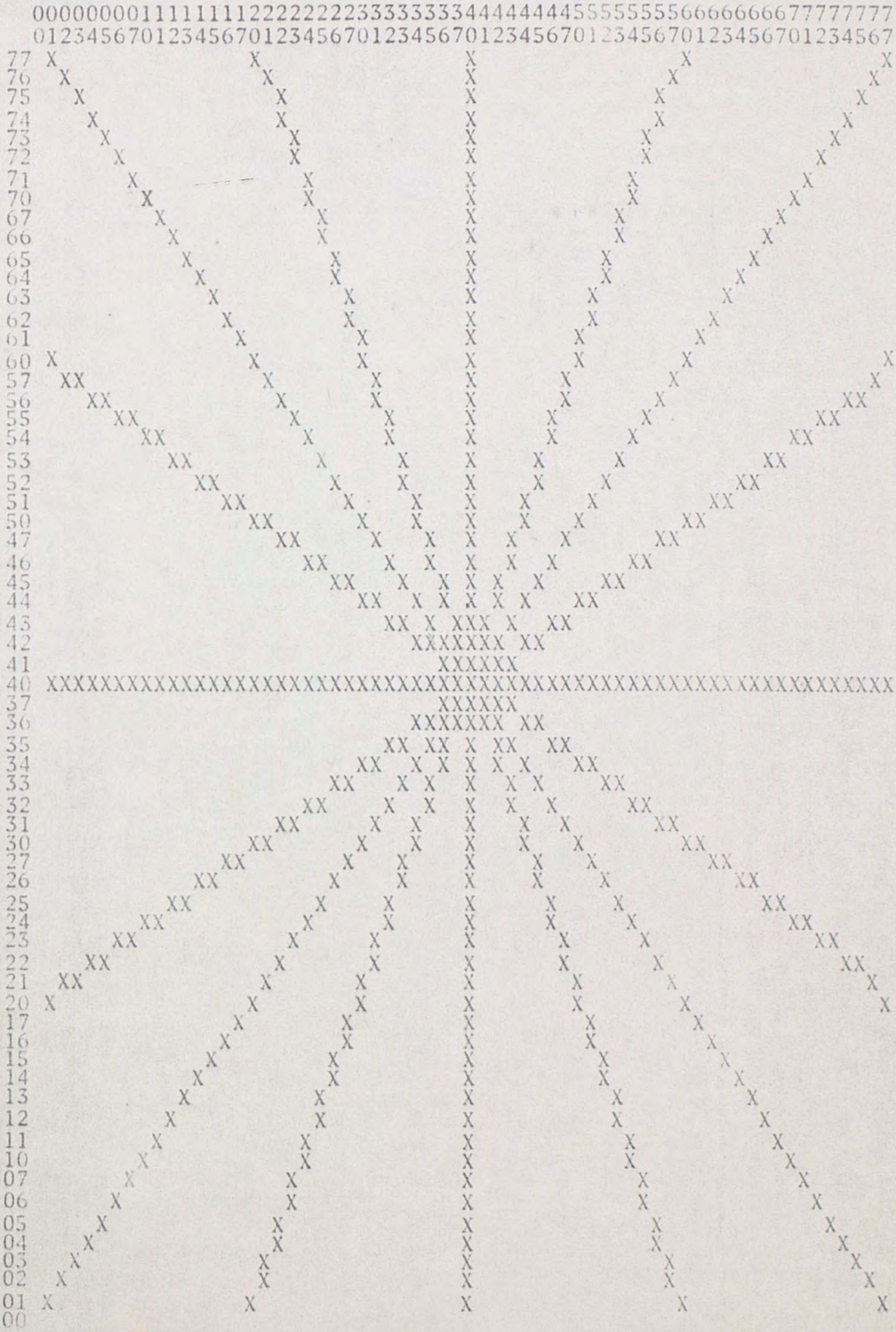
Output Characteristics

For demonstration purposes, a standard teletype terminal was used for inputting and printing data. A special output routine (see Appendix 1 for program listing) was written to type the horizontal (x) and vertical (y) coordinates in octal on the teletype and to represent the raster points with an "X". The following data was typed on the teletype console to be used as an example:

X(i)	Y(i)	X(f)	Y(f)
01	77	77	01
20	77	60	01
40	77	40	01
60	77	20	01

X(i)	Y(i)	X(f)	Y(f)
77	77	01	01
77	60	01	20
77	40	01	40
77	20	01	60

When "P" was typed on the console, the group of vectors were displayed on the teletype terminal. See Figure 4 for a printout of the above vectors.



Firgure 4. Example of Vectors Printed on Teletype

SUMMARY

This report has attempted to show that low cost computer graphics for the small scale computer user or occasional user is possible. A work station with random positioning and vector capabilities as well as character writing speeds many times in excess of a teletype is well within the state-of-the-art. A few years ago time-sharing systems appeared to be the most promising basis for low-cost graphics systems. In the past year or two, however, it has become clear that small, single-user machines offer a very attractive alternative to time-sharing. There are a number of reasons for this. It is much easier to guarantee a rapid response from a local, single-user computer than from a time-shared machine that may be heavily loaded and that is often remote from the terminal. A single-user graphics system involves far less supporting software than a time-shared one. Most persuasive of all, Abrams and Stein (4) agree that a single-user system is often cheaper to use than a terminal to an equivalent time-shared system; this is the result of the recent precipitous drop in hardware costs, a drop that has not been accompanied by any corresponding drop in the cost of

developing large time-shared operating systems.

One of the additional advantages of the single-user graphics system is that it can be tailored to a single, specified language or application, thus avoiding the extra work and operating overhead involved in developing a general-purpose, multi-language system.

APPENDIX

PROGRAM LISTING

The following sequence gives a brief verbal description of the basic steps involved in the Vector Program:

<u>START</u>

1. Read in one 6 bit octal character.

2. Check for "C" (ASCII).

3. If "C", jump to CLEAR routine and set all data bits to zeros, then return to START.

4. Check for "P" (ASCII).

5. If "P", jump to PRINT routine, format data and output; then return to START.

6. Assume character is X(initial), then read next three 6 bit octal characters: Y(initial), X(final), Y(final).

7. Compare X's and Y's.

8. If Vector is single point, form address and store single "1" bit in appropriate mem. location then return to START.

9. If Vector is horizontal or vertical line, determine the distance between (X(initial), Y(initial) and (X(final, Y(final)), increment X (Horz) or Y (Vert) this

distance, store the proper single "1" bits in the appropriate mem. locations, then return to START.

10. Vector is sloped line, divide the greater of X(final) - X(initial) or Y(final) - Y(initial) by the lesser of the two. The coordinate X or Y (with the greater difference) will be incremented Q times more than the other coordinate; Q = quotient of X(final) - X(initial) and Y(final) - Y(initial). Store the proper single "1" bits into the appropriate mem. locations, then return to START.

```
0000                      ORG       0A00H
3C57                      TTOUT     EQU     3C57H
3C9B                      TTIN      EQU     3C9BH
3C4B                      BLANK     EQU     3C4BH
3CC7                      CRLF      EQU     3CC7H
0010                      X1        EQU     10H
0011                      Y1        EQU     11H
0012                      X2        EQU     12H
0013                      Y2        EQU     13H
0014                      CTR1      EQU     14H
0015                      DXY       EQU     15H
0016                      QUOTI     EQU     16H
0017                      REMAN     EQU     17H
0018                      CONST     EQU     18H
0019                      XSWAP     EQU     19H
001A                      YSWAP     EQU     1AH
0C00                      DELIM     EQU     0C00H
0A00 C0          BEGIN:   NOP
0A01 469B3C               CALL    TTIN
0A04 46000C               CALL    DELIM
0A07 2E003618             LXI     H,CONST
0A0B 3E01                 MVI     M,1
0A0D 2E003619             LXI     H,XSWAP
0A11 3E00                 MVI     M,0
0A13 30                   INR      L
0A14 3E00                 MVI     M,0
0A16 1604                 MVI     C,4
0A18 2E003610             LXI     H,X1
0A1C 44230A               JMP     READ
0A1F 469B3C      TTYIN:   CALL    TTIN
0A22 30                   INR     L
0A23 12          READ:    RAL
0A24 12                   RAL
0A25 12                   RAL
0A26 2438                 ANI     070Q
0A28 E0                   MOV     E,A
0A29 469B3C               CALL    TTIN
0A2C 2407                 ANI     7
0A2E B4                   ORA     E
0A2F F8                   MOV     M,A      ;STORE COORDINATES
0A30 11                   DCR     C
0A31 481F0A               JNZ     TTYIN
0A34 CF                   MOV     B,M      ;LOAD YF IN REG B
0A35 31                   DCR     L
0A36 D7                   MOV     C,M      ;LOAD XF IN REG C
0A37 31                   DCR     L
0A38 DF                   MOV     D,M      ;LOAD YI IN REG D
0A39 31                   DCR     L
0A3A E7                   MOV     E,M      ;LOAD XI IN REG E
0A3B C2                   MOV     A,C      ;LOAD XF IN A
0A3C BC                   CMP     E        ;COMPARE XI WITH XF
```

```
0A3D 40520A           JNC   DX       ;XI LESS THAN OR EQUAL
0A40 D4               MOV   C,E      ;XI GREATER THAN XF
0A41 E0               MOV   E,A
0A42 2E003619         LXI   H,XSWAP
0A46 3E01             MVI   M,1
0A48 2E003610         LXI   H,X1
0A4C FC               MOV   M,E      ;STORE SMALL X AT X1
0A4D 2E003612         LXI   H,X2
0A51 FA               MOV   M,C      ;STORE LARGE X AT X2
0A52 C2       DX:     MOV   A,C
0A53 94               SUB   E        ;DX = X2 - X1
0A54 D0               MOV   C,A      ;LOAD DX IN REG C
0A55 C1               MOV   A,B
0A56 BB               CMP   D        ;COMPARE YI WITH YF
0A57 406C0A           JNC   DY       ;YI LESS THAN OR EQUAL
0A5A CB               MOV   B,D      ;YI GREATER THAN YF
0A5B D8               MOV   D,A
0A5C 2E00361A         LXI   H,YSWAP
0A60 3E01             MVI   M,1
0A62 2E003611         LXI   H,Y1     ;STORE SMALL Y AT Y1
0A66 FB               MOV   M,D
0A67 2E003613         LXI   H,Y2     ;STORE LARGE Y AT Y2
0A6B F9               MOV   M,B
0A6C C1       DY:     MOV   A,B
0A6D 93               SUB   D        ;DY = Y2 - Y1
0A6E BA               CMP   C        ;COMPARE DY WITH DX
0A6F 40780A           JNC   STOD     ;DY GREATER THAN OR EQU
0A72 DA               MOV   D,C      ;DY LESS THAN DX, LOAD
0A73 D0               MOV   C,A
0A74 A8               XRA   A        ;SET A = 0
0A75 447B0A           JMP   SMLDY
0A78 D8       STOD:   MOV   D,A      ;LOAD DY IN D
0A79 0601             MVI   A,1H     ;SET A = 1
0A7B 2E003614 SMLDY:  LXI   H,CTR1   ;STORE LARGER OF DX OR
0A7F FB               MOV   M,D
0A80 2E003615         LXI   H,DXY    ;SET DXY=0 IF DX>DY
0A84 F8               MOV   M,A      ;SET DXY=1 IF DY=>DX
0A85 A8               XRA   A
0A86 C8               MOV   B,A      ;SET REGISTER B=0
0A87 C3               MOV   A,D
0A88 3C00             CPI   0H       ;COMPARE LARGE DIFF.
0A8A 68200B           JZ    FORM     ;(DX=DY=0),SINGLE POI
0A8D C2               MOV   A,C
0A8E 3C00             CPI   0H       ;COMPARE SMALL DIFF.
0A90 48A00A           JNZ   SLOPE    ;(DX NOR DY=0),SLOPED
0A93 2E003616         LXI   H,QUOTI;(DX OR DY=0),HORZ.
0A97 F8               MOV   M,A      ;SET QUOTI=0
0A98 2E003617         LXI   H,REMAN
0A9C F8               MOV   M,A      ;SET REMAN=0
0A9D 441B0B           JMP   SETC
```

```
ØAAØ C3        SLOPE:  MOV   A,D
ØAA1 Ø4Ø1              ADI   1
ØAA3 D8                MOV   D,A    ;ADD 1 TO TO D (LARGE DIF
ØAA4 C2                MOV   A,C
ØAA5 Ø4Ø1              ADI   1      ;ADD 1 TO C (SMALL DIFF.)
ØAA7 D3                MOV   C,D    ; INTER CHANGE REGISTER'S
ØAA8 D8                MOV   D,A
ØAA9 2EØ2              MVI   H,2    ;SET DIV. CTR TO 2
ØAAB 26Ø9      DIV:    MVI   E,9
ØAAD C1                MOV   A,B
ØAAE C8        DIVD:   MOV   B,A
ØAAF C2                MOV   A,C
ØABØ 12                RAL
ØAB1 DØ                MOV   C,A
ØAB2 21                DCR   E
ØAB3 68CØØA            JZ    DIV1
ØAB6 C1                MOV   A,B
ØAB7 12                RAL
ØAB8 93                SUB   D
ØAB9 4ØAEØA            JNC   DIVD
ØABC 83                ADD   D
ØABD 44AEØA            JMP   DIVD
ØACØ Ø6FF      DIV1:   MVI   A,ØFFH ;COMPLEMENT QUOTIENT
ØAC2 AA                XRA   C
ØAC3 29                DCR   H      ;DECR. DIV.CTR H
ØAC4 68D3ØA            JZ    DIV2   ;JUMP TO DIV2 IF FINISH
ØAC7 2EØØ3616          LXI   H,QUOTI
ØACB F8                MOV   M,A    ;STORE INTEGER QUOTIENT
ØACC 2EØ1              MVI   H,1
ØACE 16ØØ              MVI   C,ØH
ØADØ 44ABØA            JMP   DIV    ;INTEGER QUOT COMPLETE
ØAD3 2EØØ3617  DIV2:   LXI   H,REMAN
ØAD7 F8                MOV   M,A    ;STORE DECIMAL IN LOC
ØAD8 A8                XRA   A
ØAD9 C8                MOV   B,A    ;SET B=Ø
ØADA 2EØØ3619          LXI   H,XSWAP
ØADE E7                MOV   E,M
ØADF 2EØØ361A          LXI   H,YSWAP
ØAE3 C7                MOV   A,M
ØAE4 BC                CMP   E
ØAE5 681BØB            JZ    SETC
ØAE8 2EØØ3618          LXI   H,CONST
ØAEC 3EFF              MVI   M,-1
ØAEE 2EØØ3615          LXI   H,DXY
ØAF2 C7                MOV   A,M
ØAF3 3CØØ              CPI   ØH
ØAF5 68ØBØB            JZ    YSWP
ØAF8 2EØØ361Ø          LXI   H,X1
ØAFC C7                MOV   A,M
ØAFD 2EØØ3612          LXI   H,X2
```

```
ØBØ1 E7                  MOV    E,M
ØBØ2 F8                  MOV    M,A
ØBØ3 2EØØ361Ø            LXI    H,X1
ØBØ7 FC                  MOV    M,E
ØBØ8 441BØB              JMP    SETC
ØBØB 2EØØ3611  YSWP:     LXI    H,Y1
ØBØF C7                  MOV    A,M
ØB1Ø 2EØØ3613            LXI    H,Y2
ØB14 E7                  MOV    E,M
ØB15 F8                  MOV    M,A
ØB16 2EØØ3611            LXI    H,Y1
ØB1A FC                  MOV    M,E
ØB1B 2EØØ3616  SETC:     LXI    H,QUOTI
ØB1F D7                  MOV    C,M      ;SET REG.C TO VALUE
ØB2Ø 2EØØ3611  FORM:     LXI    H,Y1     ;LOAD Y1 IN REG. E
ØB24 E7                  MOV    E,M
ØB25 2EØØ361Ø            LXI    H,X1
ØB29 DF                  MOV    D,M      ;LOAD X1 IN REG D&A
ØB2A C3        LAST2:    MOV    A,D
ØB2B ØA                  RRC
ØB2C ØA                  RRC
ØB2D ØA                  RRC
ØB2E 24Ø7                ANI    7        ;PLACE BITS 3,4&5 OF X
ØB3Ø FØ                  MOV    L,A      ;Ø,1 AMD 2 OF ADDR REG L
ØB31 C4                  MOV    A,E
ØB32 Ø2                  RLC
ØB33 Ø2                  RLC
ØB34 Ø2                  RLC
ØB35 EØ                  MOV    E,A
ØB36 24F8                ANI    ØF8H     ;OR BITS Ø,1,2,3,&4 OF Y
ØB38 B6                  ORA    L        ;AND 7 OF ADDR PEG L
ØB39 FØ                  MOV    L,A      ;L=LOW ORDER ADDRESS
ØB3A C4                  MOV    A,E
ØB3B 24Ø1                ANI    1
ØB3D Ø4Ø1                ADI  1
ØB3F E8                  MOV    H,A      ;H=HIGH ORDER ADDR  ..
ØB4Ø CØ                  NOP
ØB41 CØ                  NOP
ØB42 C3                  MOV    A,D
ØB43 24Ø7                ANI    7
ØB45 D8                  MOV    D,A      ;LOAD REG A WITH XD
ØB46 Ø6Ø1                MVI    A,1
ØB48 685ØØB    DATA:     JZ     STORE    ;FIND DATA =2**XD
ØB4B 19                  DCR    D
ØB4C Ø2                  RLC
ØB4D 4448ØB              JMP    DATA
ØB5Ø D8        STORE:    MOV    D,A
ØB51 C7                  MOV    A,M
ØB52 B3                  ORA    D        ;OR DATA IN LOC H,L
ØB53 F8                  MOV    M,A
```

```
0B54 2E003614          LXI    H,CTR1
0B58 C7                MOV    A,M
0B59 3C00              CPI    0H
0B5B 68000A            JZ     BEGIN   ;HAVE ALL POINTS
0B5E 1401              SUI    1       ;DECR.CTR1
0B60 F8                MOV    M,A
0B61 68B00B            JZ     LAST1   ;ALL BUT LAST POINT
0B64 2E003615          LXI    H,DXY
0B68 C7                MOV    A,M
0B69 3C00              CPI    0H
0B6B 68750B            JZ     LOADX   ;DXY=0 (DX>DY) INCR.
0B6E 2E003611          LXI    H,Y1    ;DXY=1 (DY=>DX) INCR.
0B72 44790B            JMP    OVERX
0B75 2E003610 LOADX:   LXI    H,X1
0B79 C7       OVERX:   MOV    A,M
0B7A 0401              ADI    1       ; INCRE. LARGER OF X1
0B7C F8                MOV    M,A
0B7D 11                DCR    C
0B7E 48200B            JNZ    FORM    ;DONT INCR SMALLER OF
0B81 2E003617 ADREM:   LXI    H,REMAN
0B85 C7                MOV    A,M
0B86 81                ADD    B
0B87 C8                MOV    B,A     ;ADD DECIMAL QUOT. TO
0B88 60AA0B            JC     RESET   ;DOES B HAVE OVERFLOW?
0B8B 2E003615          LXI    H,DXY
0B8F C7                MOV    A,M
0B90 3C00              CPI    0H
0B92 2E003610          LXI    H,X1
0B96 489B0B            JNZ     OVRY
0B99 3601              MVI     L,01H
0B9B C7       OVRY:    MOV     A,M
0B9C C7                MOV     A,M
0B9D D5                MOV     C,H
0B9E DE                MOV     D,L
0B9F 2E003618          LXI     H,CONST
0BA3 87                ADD     M
0BA4 EA                MOV     H,C
0BA5 F3                MOV     L,D
0BA6 F8                MOV    M,A
0BA7 441B0B            JMP    SETC
0BAA A8       RESET:   XRA    A
0BAB C8                MOV    B,A     ;CLEAR REG. B
0BAC 10                INR    C       ;SET REG C TO +1
0BAD 44200B            JMP    FORM
0BB0 2E003613 LAST1:   LXI    H,Y2    ;NEXT POINT WILL BE
0BB4 E7                MOV    E,M
0BB5 2E003612          LXI    H,X2    ;USE COORDINATE X2,Y2
0BB9 DF                MOV    D,M
0BBA 442A0B            JMP    LAST2
0BBD 00                HLT
0000                   END
```

```
0000            ORG    0C00H
3C57            TTOUT  EQU   3C57H
3C9B            TTIN   EQU   3C9BH
3C4B            BLANK  EQU   3C4BH
3CC7            CRLF   EQU   3CC7H
0A00            BEGIN  EQU   0A00H
0C00 C0         DELIM:  NOP
0C01 3CC3               CPI    0C3H
0C03 68A90C             JZ     CLEAR
0C06 3CD0               CPI    0D0H
0C08 0B                 RNZ
0C09 440C0C             JMP    PRINT
0C0C C0         PRINT:  NOP
0C0D 46C73C             CALL   CRLF
0C10 464B3C             CALL   BLANK   ;PRINT 3 SPACES
0C13 464B3C             CALL   BLANK
0C16 464B3C             CALL   BLANK
0C19 1E08               MVI    D,8
0C1B 162F               MVI    C,2FH
0C1D 10         NEWC:   INR    C
0C1E 2608               MVI    E,8
0C20 CA         SAMEC:  MOV    B,C
0C21 46573C             CALL   TTOUT
0C24 21                 DCR    E
0C25 48200C             JNZ    SAMEC
0C28 19                 DCR    D
0C29 481D0C             JNZ    NEWC
0C2C 46C73C             CALL   CRLF
0C2F 464B3C             CALL   BLANK
0C32 464B3C             CALL   BLANK
0C35 464B3C             CALL   BLANK
0C38 1E08               MVI    D,8     ;SET OCTAL COUNTER
0C3A 1630       CZERO:  MVI    C,30H
0C3C 2608               MVI    E,8     ;SET UNITS COUNTER
0C3E CA         STEPC:  MOV    B,C
0C3F 46573C             CALL   TTOUT   ;PRINT NUMBER
0C42 10                 INR    C       ;INCREMENT NUMBER
0C43 21                 DCR    E
0C44 483E0C             JNZ    STEPC   ;RETURN AND PRINT NEXT
0C47 19                 DCR    D
0C48 483A0C             JNZ    CZERO   ;RETURN AND PRINT ZERO
0C4B 2E02               MVI    H,2
0C4D 36F8               MVI    L,370Q
0C4F 0E0D       NXLIN:  MVI    B,0DH
0C51 46573C             CALL   TTOUT
0C54 0E0A               MVI    B,0AH
0C56 46573C             CALL   TTOUT
0C59 1602               MVI    C,2
0C5B C5                 MOV    A,H
0C5C 1401               SUI    1
0C5E 86                 ADD    L
```

```
0C5F 02          SECCH:  RLC
0C60 02                  RLC
0C61 D8                  MOV    D,A
0C62 2407                ANI    7
0C64 0430                ADI    30H
0C66 C8                  MOV    B,A
0C67 46573C              CALL   TTOUT
0C6A C3                  MOV    A,D
0C6B 02                  RLC
0C6C 11                  DCR    C
0C6D 485F0C              JNZ    SECCH
0C70 0E20                MVI    B,' '
0C72 46573C              CALL   TTOUT   ;PRINT BLANK
0C75 2608                MVI    E,8
0C77 D7          NEWL:   MOV    C,M
0C78 1E08                MVI    D,8
0C7A C2                  MOV    A,C
0C7B 2401        BACK:   ANI    1
0C7D 68850C              JZ     PRBLK
0C80 0E58                MVI    B,'X'
0C82 44870C              JMP    OVER
0C85 0E20        PRBLK:  MVI    B,' '
0C87 46573C      OVER:   CALL   TTOUT
0C8A C2                  MOV    A,C
0C8B 1A                  RAR
0C8C D0                  MOV    C,A
0C8D 19                  DCR    D
0C8E 487B0C              JNZ    BACK    ;MORE BITS IN DATA WORD
0C91 21                  DCR    E
0C92 68990C              JZ     LCPLT   ;MORE WORDS IN LINE
0C95 30                  INR    L
0C96 44770C              JMP    NEWL
0C99 06F1        LCPLT:  MVI    A,-15   ;LINE COMPLETED
0C9B 86                  ADD    L
0C9C F0                  MOV    L,A
0C9D 3CF8                CPI    -8H
0C9F 484F0C              JNZ    NXLIN
0CA2 29                  DCR    H
0CA3 68000A              JZ     BEGIN
0CA6 444F0C              JMP    NXLIN
0CA9 1602        CLEAR:  MVI    C,2
0CAB 2E013600            LXI    H,100H
0CAF 3E00        CLR:    MVI    M,0
0CB1 30                  INR    L
0CB2 48AF0C              JNZ    CLR
0CB5 28                  INR    H
0CB6 11                  DCR    C
0CB7 68000A              JZ     BEGIN
0CBA 44AF0C              JMP    CLR
0000                     END
```

REFERENCES

(1) Van Dam, Andries, and Michener, J. C. "Storage Tube Graphics-a Comparison of Terminals". Advanced Computer Graphics-Economics Techniques and Applications, pp. 852-57. Edited by R. R. Parslow and R. Elliot Green. New York: Plenum Press, 1971.

(2) Jacks, Edwin L. "A Laboratory for the Study of Graphical Man-Machine Communication". Conversational Computers, p. 144. William D. Orr. New York: John Wiley and Sons, 1968.

(3) O'Callaghan, John F. "Use of a Picture Language to Generate Descriptions of Line Drawings in an Interactive System". Graphic Languages, pp. 125-27. Edited by F. Nake and A. Rosefeld. London: North Holland Publishing Co., 1972.

(4) Abrams, Marshall D., and Stein, Philip G. Computer Hardware and Software-An Interdisciplinary Introduction. Reading, Mass.: Addison-Wesley Publishing Co., 1973.

(5) Williams, Robin A. "General Purpose Graphical Language". Graphic Languages, pp. 334-46. Edited by F. Nake and A. Rosefeld. London: North-Holland Publishing Co., 1972.

(6) Van Dam, Andries. "Are We Near a Universal Graphic Language". Graphic Languages, pp. 354-55. Edited by F. Nake and A. Rosefeld. London: North-Holland Publishing Co., 1972.

(7) Newman, William M. "Are We Near a Universal Graphic Language". Graphic Languages, pp. 355-56. Edited by F. Nake and A. Rosefeld. London: North-Holland Publishing Co., 1972.

(8) Desmonde, William H. A Conversational Graphic Data Processing System: the IBM 1130/2250. Englewood Cliffs, N. Y.: Printice-Hall Inc., 1969.

(9) Louden, Robert K. Programming the IBM 1130 and 1800. Englewood Cliffs, New York: Printice-Hall Inc., 1967.

(10) French, L. J. "Picture Generation for Interactive Circuits". Proceedings of the ACM. IEEE Design Automation Workshop. New York: The Association of Computing Machinery and the IEEE, Inc., 1972.

(11) Newman, William M. "A Low Cost Single-User Graphics System". Graphics Language, pp. 291-97. Edited by F. Nake and A. Rosefeld. London: North-Holland Publishing Co., 1972.

(12) Newman, William M. "Graphics and Education-an Informal Graphics System Based on the LOGO Language". AFIPS Conference Proceedings 42 (April 1973): 651-55.

(13) Meads, Jon A. "A Terminal Control System". Graphic Languages, pp. 271-72. Edited by F. Nake and A. Rosefeld. London: North-Holland Publishing Co., 1972.

(14) "CRT Terminal". Digital Design Handbook (January 1975): 32-35.

(15) Cropper, Leigh C. and Whiting, John. "An Evaluation of the Use of Microprocessors in CRT Display Terminals". IEEE 1974 COMPCON Spring. Los Angeles, Ca.: IEEE Press, 1974.

(16) MCS-8 Assembly Language Programming Manual. Santa Clara, Ca.: Intel Corp., (1973).