Intergraphic a microprogrammed, graphical-interface computer

Author:
Rose, Gordon A.

Publication Date:
1969

DOI:
https://doi.org/10.26190/unsworks/14141

INTERGRAPHIC - A MICROPROGRAMMED, GRAPHICAL-INTERFACE COMPUTER

GORDON A. ROSE

Being a thesis
submitted as part of the requirements
for the degree of Ph.D.

I certify that the work described in this thesis
has not previously been submitted in whole or in part
towards any university degree.

G.A. Rose
1st November, 1969

ABSTRACT

The thesis describes an economical graphical-communication system which will link a number of graphical terminals to a central processor.

The thesis concentrates on the system's shared graphical-interface computer, Intergraphic, which is complete and operational. Intergraphic will be an intermediary between the terminals and the central processor. Being microprogrammed, the interface is extremely flexible. Also, the interface is fast (3-5 ns logic, 50 ns access-time ROM), and the combination of flexibility and speed has enabled the centralization of vector generation, symbol generation, and many other tasks required by the system. Centralization has led to low per-terminal costs (less than $2000 for component costs only). Incremental plotting rates of 10 MHz have been digitally-generated and applied to an electrostatically-deflected central CRT which displays arbitrary graphics, prior to distribution, from a repertoire of vector types, symbols and circular arcs.

The thesis outlines the conversion of centrally-generated images to television format, the storage of TV images on a multi-track video disc, and the economical distribution of images to TV terminals using commercially-available units.

A survey of some early computer-graphics systems is followed by a discussion of the advances in computer-graphics schemes, particularly those which have reduced per-terminal costs. To justify the design of Intergraphic, the salient features of the scheme are

compared with those of five alternative schemes. The detailed logical description of the interface and the related microcode then follows with emphasis on the integration of vector and symbol generation within the microcode. This integration has been achieved with only minor extensions to an otherwise relatively conventional microprogrammed structure, and has led to high component utilization which is essential for a low-cost system. Moreover, vector and symbol generation has been achieved using the standard digital techniques adopted within the interface. The chosen vector and symbol generation techniques are compared with a number of alternatives and the general question of description and display of arbitrary graphics is discussed.

A simple algorithm is described which enables breakpoints to be inserted in input graphics as they are accepted in real-time from users' light-pens. The problem of providing graphical feedback to users is outlined.

The thesis concludes that low-cost computer-graphics terminals are possible using state-of-the-art devices and that current planning of computer utilities should include extensive computer-graphics networks which link a variety of terminal types via microprogrammed interfaces.

ACKNOWLEDGEMENTS

The author wishes to thank Professor M.W. Allen, Head of the Department of Computation, University of New South Wales, for his constant support throughout the thesis project. Many of the concepts of Intergraphic stemmed from the author's earlier association with the CIRRUS project. During that project, Mr. T. Pearcey introduced the author to microprogramming and read-only stores and Professor Allen introduced the author to computer engineering.

The author wishes to thank C.J. Barter, G.P. Bowen, M. Macaulay, R.B. Stanton and A.A. Thompson for their numerous discussions and helpful criticism on various aspects of Intergraphic. Also, without the laboratory and technical support of Mr. R. Chorley amd Mr. K. Titmuss and the staff of the University's workshop, the construction of Intergraphic would not have been possible. The author wishes to thank Miss P. Rooney and Mrs. J. Greenhill for typing the thesis.

CONTENTS

CHAPTER 2 - LOGICAL STRUCTURE OF THE INTERFACE

CHAPTER 3 - DEFINITION OF THE MICROCODE

APPENDICES

CHAPTER 1

INTRODUCTION

1.1 MOTIVATION

The need for low-cost, general-purpose, graphical terminals for a multi-access computer system motivated the work described in the thesis. By late 1965, when this work began, many applications had shown that graphical terminals were desirable and early multi-access systems had established an operational mode which, potentially, could service hundreds of terminals. Although there had been advances in reducing per terminal costs of computer-graphics systems, an order of magnitude improvement was needed before graphical terminals could be multiplexed in quantity.

1.2 THREE EARLY COMPUTER-GRAPHICS SYSTEMS

By 1964, a number of computer-graphics systems had established that computer-driven cathode ray tube (CRT) displays with input pens and keyboards were extremely useful and versatile terminals. Three notable examples were, Culler and Fried's "On-Line Computing Centre for Scientific Problems" (1, 1963), Sutherland's "Sketchpad" (2, 1963) and the "Design Augmented by Computers (DAC) System" of General Motors Research Laboratories (3, 1964). These systems stemmed from earlier computer-display equipment, particularly equipment in command and control centres of defence systems, e.g. DODDAC (Department of Defense Damage Assessment Centre) (4, 1961). The three systems cited stressed different aspects of graphical-communication and have been frequently referred to in the assessment of later systems.

The user of Culler and Fried's system could define arbitrary

functions by specifying lists of ordinates (typically at 100 abscissae), operate on these functions by keyboard commands and see the results of his commands almost immediately. Also, the user could define new key labels, and build subroutines through a facility which monitored and stored the sequence in which he operated keys. Keyboard overlays conveniently grouped the large number of function keys which were possible. The system coupled the user to the computer very closely so that he could experiment freely with parameters or transformations and so quickly gain insight into his problem.

"Sketchpad" enabled a user to construct and modify arbitrary line drawings with light-pen and function keys. Software interpolated the plotting details for straight-line segments and circular arcs from parameters entered by the user. Sub-pictures could be named, recalled, transformed (scaled, rotated, reflected, etc.) and connected to form further line drawings. Also, by time-sequencing some parameters, drawings could be set in motion, e.g., a four-bar linkage mechanism. Topological relations were coded in a ring structure, so that modifications to one element could be linked to related elements.

The DAC-1 system required precision generation and scanning of arbitrary line-drawings for automobile design. High-precision displays and photo-recording and tracing devices were developed. A "sketchpad"-like system, using an analogue input pen (position-indicating pencil) was used for dynamic communication. The position-indicating pencil simplified pen tracking: it eliminated the time-consuming routines which detected pen movement by probing, at many points, the receptive field of the photo-detector of a conventional light-pen.

1.3 THE PROHIBITIVE COST OF ONE-DISPLAY SYSTEMS

The three systems cited above and other early systems, however, were devised to meet a particular need or to demonstrate the potential

of computer-graphics; system or hardware efficiency was not of first consideration and, typically, single displays were connected directly to a central computer. The cost per display of a dedicated one-display system was virtually the cost of the entire system; this high per-terminal cost severely restricted the installation of graphical terminals.

Fig. 1.1 shows a typical one-display system comprising a central processing unit (CPU); a short-persistence CRT with digital X, Y co-ordinate registers and digital-to-analogue (D/A) converters; a photo-sensor light pen; and a keyboard, possibly with several overlays. The CPU held the application program, display file and routines for vector and symbol generation and pen tracking.

"Sketchpad" was implemented on the TX-2 computer in this general way.

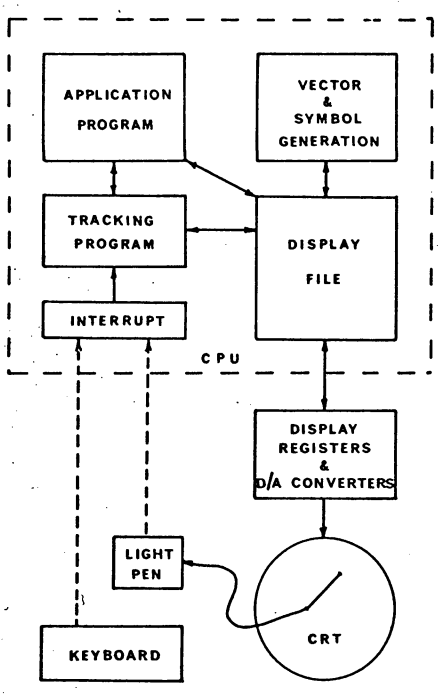


FIG. 1.1. DIRECTLY COUPLED DISPLAY

Most of the system functions were programmed and specialized hardware was minimal; thus, the medium was flexible for experimentation, but imposed restrictions on display complexity for a given flicker rate. Display regeneration alone could almost fully extend the computer so that application programs were limited and time-sharing was impossible. However, the direct connection gave rapid access to the application program. Also, the CRT was closely linked to the CPU so that, within the restrictions of vector and symbol generation, a rapid new image rate or dynamic display was possible.

The principal inefficiencies were:-

1. Regeneration of the display for persistence of vision from the machine core.

2. Software generation of vectors and symbols.

3. Software pen-tracing which required multiple probing of the pen field for each incremental movement of the pen.

Experience with these systems pointed out the many functional requirements of graphical communication, the importance of structuring both application and display programs, and those areas where specialised hardware would be advantageous.

1.4 IMPROVED VECTOR AND SYMBOL GENERATORS

Parallel developments in vector and symbol generation hardware greatly increased plotting rates and so more complex displays could be presented within a persistence of vision period. Also, these devices conserved core storage by eliminating the need for vector and symbol generation programs and tables. The introduction of separate display core-buffers, which held the display file for image regeneration, reduced interrupts on the CPU and freed it for more extensive

application programs or other work. Typically, a display console would comprise a magnetically-deflected large CRT, a symbol generator and a vector generator. The CRT would have deflection resolutions of 1 in 1024, a random position settling time of 30μs, and an incremental plotting rate of 1.5μs per visible point or 300ns per blanked point. Average plotting time for the symbol and vector generator would be 20μs per symbol and 150μs/in respectively[1]. Plotting rates an order faster (2.5μs per random point or symbol; 4μs/in vector rate) were developed using electrostatically-deflected CRT's with vector and symbol generators which were part analogue[2]. These rates allowed even more complex diagrams to be regenerated without apparent flicker.

1.5 TERMINALS FOR MULTI-ACCESS COMPUTING

Multi-access computing as a mode of computer operation was being developed in parallel with the graphical systems and devices above; emphasis was on the operational mode rather than advanced terminal development. By 1964, a multi-access system for about 30 simultaneous users was being realised at Project MAC (5); terminals were electromechanical teletypes. Almost two years later, Corbato and Vyssotsky (6, 1965), in describing the "MULTICS" operating system for project MAC, considered general-purpose graphical terminals highly desirable, but still too costly and too demanding to be multiplexed, particularly in the tens and hundreds implied for multi-access systems. (Costs of single display consoles, either electromagnetically or electrostatically deflected, with vector, symbol and regeneration devices were high - typically in excess of $50,000; in contrast, one would expect per-terminal costs to be less than $5,000 before large numbers of temminals could be multiplexed.)

---

1 DEC Type 338 Display, Digital Equipment Corp., Maynard, Mass., and Elliott Type 4100 Display, Elliott-Automation Computers Ltd., Borehamwood, Herts, England, are representative.

2 CDC Type 250 Display, Control Data Corp., Minneapolis, Minn.

1.6 LOCAL DISPLAY PROCESSORS

There had been, however, a number of advances which reduced the overall cost of computer-graphics systems, mostly by removing some of the trivial,but frequent and therefore time-consuming, tasks from the central processor. Projects MAGIC (7, 1965), GRAPHIC-1 (8, 1965) and portion of an experimental system at the University of California, Berkeley (9, 1965) showed that local processors which controlled symbol and vector generation, regenerated the display, determined pen co-ordinates, etc., greatly reduced demands on the central computer. Supporting hardware in each of these projects, however, was for one display only; thus, although central computer time was saved and the display with its local support could be moved to a remote location, the hardware cost per display was still high. In late 1966, Kennedy (10) described an operating system for a set of three displays. Relatively static displays were refreshed from a drum and, where necessary, highly dynamic displays could be driven directly from the central computer. Again, each display had considerable supporting hardware and there was no significant reduction in per-terminal cost. Fig. 1.2 shows a typical buffered display with local processor. The local buffer might comprise two independent stores (one for the processor, the other for the display logic) or, as in Schooler (11, 1966), might be a single shared store with the display logic having priority during regeneration (cycle-stealing).

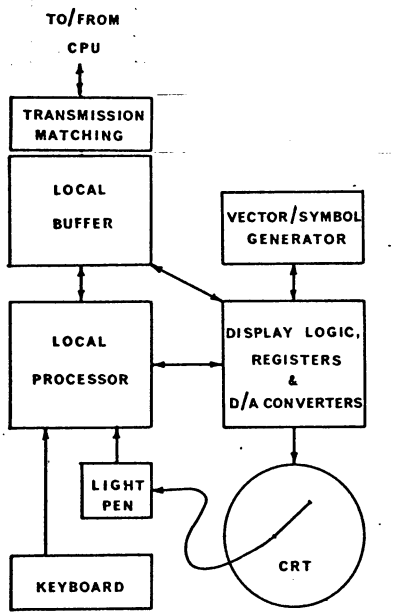


FIG. 1.2. BUFFERED DISPLAY WITH LOCAL PROCESSOR

The possibility of remote installation of these buffered displays introduced two new design parameters (bandwidth and reliability of the CPU-terminal link) and raised the question of which tasks should be done locally. Clearly, transmission characteristics, task division between the central and local processors, and local processing ability are interrelated. Relatively low rate dataphone links enable remote stations to be installed at almost any location, but limit the rate of information exchange and require compact transmission codes. Also, the speed mismatch between the CPU and dataphone line commits the CPU either to handle repetitive interrupts or to provide additional buffering. Detailed tasks which occur frequently in most application programs, e.g., vector plotting, symbol plotting and pen tracking, should be performed locally; unfortunately, many tasks are not so readily classified. Although buffered displays with extensive local ability reduced overall system costs, they were too expensive for the terminals of an economical graphical-communication system. Their main inefficiency was that expensive hardware was not shared and was used merely to regenerate static images during delays from the operator or the CPU.

1.7 GROWTH OF APPLICATIONS

Applications of computer graphics grew rapidly. By 1966, computer-aided design (12) had begun in the automobile (3,13), aerospace (62) and many other industries, e.g., the printing industry (14). Prince (15) has reviewed a number of applications which appeared before or during 1966. These applications, alone, covered a wide field and gave ample motivation to the work described in the thesis. Many more applications have been reported since 1966; examples are, textile pattern input and manipulation for subsequent computer generation of control information for textile machinery (16), generation and display of motion pictures for space research (17), generation of half-tone perspective drawings (18,19), computer aided instruction systems with graphical terminals (20,21), computer generated shading patterns for 3-

dimensional objects (22,23), generation of orthographic views of combinations of plane and quadric surfaces (24), and an interactive graphics system for pattern recognition (25).

Many of these applications demand considerable processor time, and could not be carried out concurrently on the terminals of one system; other applications demand only low information exchange rates. The graphical-communication interface described in the thesis is capable of a high total information exchange rate; within this total capacity, however, there is considerable flexibility, i.e., the interface can service many low information rate terminals, a few high information rate terminals or a mixed set of applications.

1.8 BEGINNING OF THE THESIS STUDY

In 1966, the author discussed (26, reprint enclosed) some inefficiencies in the encoding, displaying and inputting of arbitrary graphics. The paper stressed the need for an interface computer shared between graphical terminals which removed the tasks of detailed generation, simple graphical manipulation and pen tracking from the central computer, and proposed a flexible microprogrammed interface computer, "Intergraphic". There have been changes in design philosophy within Intergraphic since that outline; e.g., the incremental generation of sine and cosine (used for plotting polar vectors) in the "circular mode" described has been replaced by faster, more direct methods and the potentiometer pens for graphical input, proposed from earlier work (27, 1955, reprint enclosed), have been replaced by conventional photo- diode light-pens. However, the main proposal has remained unchanged, viz., a shared, microprogrammed interface with some special micro-orders or modes for central image generation and display servicing.

1.9 SCOPE AND ORGANISATION OF THE CONTENTS OF THE THESIS

The thesis concentrates on the graphical-interface computer which is capable of generating up to 100 independent images per second for distribution to a large number (more than 50) of terminals. Low per-terminal cost will be achieved through video storage and video distribution of images to low-cost television (TV) terminals. The video system is still experimental, and some problems remain in scan-conversion to interlaced TV at the writing speeds and timing intervals proposed.

The video system and the operating system are the responsibilities of co-workers, and therefore a detailed description of these systems is outside of the scope of this thesis. However, it will be necessary to describe these systems generally for background. Also beyond the scope of the thesis are, a comparison of various compound data structures (28 - 38) for applications of computer graphics and specific proposals for structuring graphics within Intergraphic. The interface, however, is flexible and well-suited to experimentation in these areas; there is some space in the microcode for adding special micro-order types and considerable space in the read-only memory which contains microprograms. Generally, sequences coded at microprogram level can be extremely efficient and fast (as illustrated by the micro-programs documented in APPENDIX 3); thus, a microprogrammed interface can be adapted to new tasks through "firmware" modifications with a performance often approaching that of special-purpose hardware.

The remainder of this chapter outlines the Intergraphic system and justifies it by comparison with five alternative schemes. Three of the alternatives have been developed in parallel by other workers; they are, the Advanced Remote Display Station II (ARDS-11) project (39) developed at MIT, the IBM 1500 Instructional Display Sub-System (21,40) and the GLANCE terminal system (31,41) developed at Bell Telephone Laboratories. Chapters 2-7 are organised as follows;

Chapter 2 (LOGICAL STRUCTURE OF THE INTERFACE) describes in detail the three main sections of the interface computer; viz., the general-purpose processing section, the microprogram control section, and the special-purpose display section.

Chapter 3 (DEFINITION OF THE MICROCODE) defines the micro-order types and options within these types, details the mirco-order formats, and illustrates some of the special microcode features such as conditional control, repeated execution, automatic mircoroutine linkage, and vector plotting.

Chapter 4 (VECTOR GENERATION) specifies the basic and special vector types developed within Intergraphic, compares the chosen vector generation algorithm with an alternative algorithm, and reviews alternative techniques (BRM and analogue) for vector and curve generation.

Chapter 5 (SYMBOL GENERATION) describes the symbol generation scheme adopted within Intergraphic and compares the scheme, which interprets basic stroke and increment chains encoded in the read-only memory, with dot matrix and other techniques.

Chapter 6 (GRAPHICAL INPUT) describes the co-ordinate matching method for identifying points or sub-pictures of a display and outlines a breakpoint insertion algorithm for encoding freehand input curves.

Chapter 7 (CONCLUSIONS) summarizes what has been shown by the thesis and the significance of the results. It states the limitations and advantages of the system, and contains recommendations for further work.

The BIBLIOGRAPHY is compiled in order of citation.

The APPENDIX contains details of both the conventional mode of operation of the core store and a mode which enables microcode to be run from core

store for preliminary testing of microprograms; documentation of completed microprograms in detail so that the performance and flexibility of Intergraphic can be assessed accurately; and an illustration of the detailed composition of 88 graphical symbols. Detailed logical and electronic design of gates, the arithmetic units, timing and control circuitry, the read-only memory and the digital-to-analogue converters are not shown because the thesis emphasises the achievement of a high-speed graphics interface by microprogramming assisted by several key micro-orders.

1.10 OUTLINE OF THE INTERGRAPHIC SYSTEM

This section outlines the connection of TV terminals to the interface via a video storage disc and the linkage of the interface to the central processor via a large core store. The operating system is summarized for background, and the salient features of the Intergraphic system are listed.

1.10.1 The Overall System Configuration

Fig. 1.3 shows the proposed system which was first described by the author in December, 1967 (42, reprint enclosed).

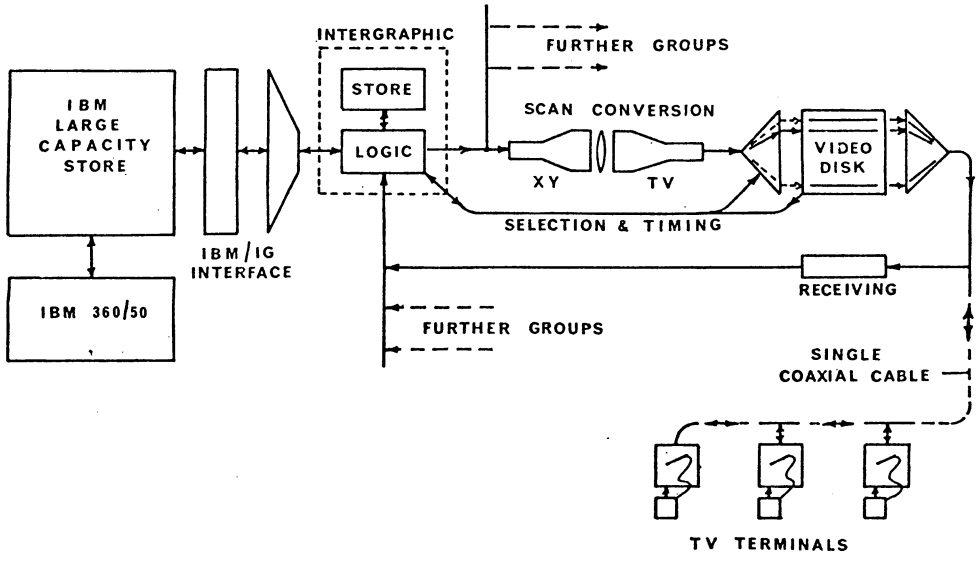


FIG. 1.3. PROPOSED GRAPHICAL-COMMUNICATION SYSTEM

Initially, Intergraphic will link 13 TV terminals to a central computer (IBM 360/50) via a large capacity core store (LCS). Intergraphic will generate all user images once only at high speed on a centrally-located, small, electrostatically-deflected CRT. This CRT, shown as the XY section of the scan converter, will be written in conventional computer-driven display modes, i.e., random point, vector and symbol plotting modes. Each image will be scan converted to a standard TV frame (Australian standard 625 lines, 40 ms) by the TV reading section of the scan-converter (a plumbicon type vidicon) and stored on a single track of a video disc. Each track will refresh a corresponding TV receiver at 25 frames/sec. Thus, regeneration for persistence of vision is regarded as a low-order task and has been placed external even to the interface computer.

Macaulay (43) has described a single coaxial cable circuit for linking 13 TV receivers to their disc tracks. Each receiver is tuned to a standard TV channel frequency. He has also described (op.cit.) simple digital circuits attached to each terminal which determine light-pen "raster co-ordinates" and return these co-ordinates to Intergraphic on the same cable once in every TV frame period. Briefly, a pair of simple counters, running synchronously with the TV raster, determine the horizontal scan line number and position within the line of a "strike" from a simple photo-sensor light pen ("raster-pen") (44). The strike pulse freezes the state of the counters and simultaneously brightens the CRT beam; this spot, a few mm from the pen aperture, provides true raster co-ordinate feedback as it eliminates errors due to delays in the pen photo-sensor response.

The main frame of Intergraphic and the central CRT are operational, and microprograms for plotting lists of Cartesian and polar vectors, circular arcs, symbols and a number of other graphics (detailed in Sect. 4.2.2) have been written. The resolution of the image generated on the central CRT is 1024 x 1024. This resolution is superior to that of the video storage and distribution system; however, the author chose a 1024 x 1024 primary display grid in the hope that

higher frequency video-storage techniques would be developed which would improve the quality of distribution (e.g., to 800 lines). Television displays refreshed at flicker-free rates have inherently fast response times, and the continuously running raster allows simple photo-detector light pens to be used.

The logical structure of Intergraphic is extremely versatile (being sequenced from microprograms held in a read-only memory, ROM) and fast (3-5ns integrated circuits and 100ns ROM cycle time). This enables many functions to be performed within Intergraphic; e.g., the interpretation and plotting of strings of compactly encoded vectors from CPU programs. Several micro-orders have been designed to assist symbol and vector generation; through these orders an incremental plotting rate of 10 MHz has been achieved. Lists of symbols, vectors and other graphics are executed (plotted) as normal machine orders; thus special-purpose vector and symbol generation hardware has been eliminated. A small amount of additional standard digital logic (about 3% of the interface logic) is required to execute the vector and symbol plotting micro-orders. Apart from the ROM and core store, Intergraphic is constructed entirely of integrated-circuit OR-NOR gates, analogue signals first appearing at the outputs of the D/A converters. As Intergraphic has a powerful machine code (also interpreted through fast microcode sequences), it can preprocess blocks of CPU code before plotting; e.g., scale, shift or reflect a graphic.

Although the overall function of Intergraphic is dedicated and therefore specialized, the function is complex and will change with operational experience; hence, the approach has been to specialize a versatile structure by microprograms which can be changed from time to time. Compact image encodings will be buffered in Intergraphic core store at a data rate matching that of the LCS; viz., 1 byte (8 bits)/ μs. For example, a page of text of 1000 symbols or a line drawing comprising 500 short vectors (short Cartesian vectors have X,Y component magnitudes less than 64 increments) will occupy 1000 bytes and require a core-to-core transfer time of 1 ms. The generation time of such dis-

plays will be approximately 5ms; thus, the elapsed time from LCS to generation on the central CRT will be only a fraction of a scan conversion period (40 ms), thereby leaving the interface considerable free time for other functions. Later, further scan converters will be added. It will be possible for Intergraphic to generate 100 new frames per second. However, operational experience with the initial 13 terminals will be necessary to assess the overall system capability before deciding on the final number of terminals.

Highly dynamic displays have a high new image rate; thus, they require direct coupling to Intergraphic rather than to a video track. Such displays will be driven identically to, but in place of, the writing section of a scan converter.

1.10.2 Outline of the Operating System

As previously stated, the operating system is the responsibility of co-workers; this outline is included as background.

User programs will be expressed in PL/1 with embedded graphical orders. PL/1 orders will be executed in the central processor (CP); graphical orders, however, will be interpreted partly by CP subroutines and partly by Intergraphic (IG). For example, "plot rectangle a X b, bottom left corner x_o,y_o" would be expanded into a list header and four vectors by CP subroutines and the list executed (in this case, plotted) by IG.

Communication between the CP and IG will be buffered by two queues in LCS: one, the CP/IG queue, will queue tasks to be executed by IG (EXIG's); the other, the IG/CP queue, will queue data resulting from the execution of EXIG's. The CP/IG queue will be built up by the component of the operating system resident in the central processor (OS/CP) and serviced by the component of the operating system resident in Intergraphic (OS/IG). Examples of EXIG's are "display an

output graphic", "post-process, then display a graphic", and "track an input graphic and encode as a vector string". The IG/CP queue will be built up by OS/IG and serviced by OS/CP; examples of data in this queue are "a string of symbols from a terminal", "a string of vectors encoding an input graphic", and "an item pointed to by the user". Only one EXIG will be resident in the Intergraphic core store at any one time.

In summary, apart from an initial request for service, all terminal activity will be initiated by user programs resident in the central computer; i.e., Intergraphic and the terminals are regarded as a flexible input-output device, the inputs coming from any one user being in modes prescribed by his program.

1.10.3 Salient Features of Intergraphic

Before reviewing and comparing alternative schemes for low-cost graphical terminals, the salient features of Intergraphic will be summarized:-

(1) Independence of Image Generation and Image Distribution

Scan conversion isolates image generation from image storage and distribution. This gives more freedom for developing image generation techniques and more freedom for designing an economical storage and distribution system; thus in Intergraphic, the interface computer generates display points asynchronously in incremental and random point modes, whereas the terminals receive display points synchronously in a regular scanning mode.

(2) Low-Cost Television Storage and Distribution

The advantages of TV storage and distribution are:

(i) Low per-terminal cost (the cost of a TV display refreshed from one track of a video disc is approximately $1000).

(ii) Flicker-free refresh rates and inherently fast response times.

(iii) Frequency multiplexing on a single cable is possible through the standard channel selectors of TV receivers.

(iv) Some noise tolerance, both in the storage and distribution of video signals.

(v) Simple on-off intensity (binary-video) or grey level displays may be selected. (Grey-level storage on a video disc normally uses frequency modulation recording, whereas binary-video can be stored directly and gives higher horizontal resolution.)

(vi) Simple raster-pens, sampled in each TV frame, are possible.

(vii) Several extensive video co-axial cable and twisted pair networks have shown the feasibility of large scale video distribution. Gabriel (45) describes video networks which serve about 7% of all television homes in Great Britain. The networks use multipair or multi-coaxial cables rather than the wide-band, frequency multiplexed cables of community antenna television (CATV) systems.

The disadvantages of TV storage and distribution are:

(i) High data rates are necessary. This follows because TV coding is far less compact than computer display-file coding for most images; e.g., to send 1000 symbols or 500 vectors requires about 10^4 bits of compact computer code,

whereas to send a standard TV image with binary intensity requires about 2×10^5 bits.

(ii) The code length of a TV image is constant, regardless of the complexity of the image.

(iii) Low-cost video storage for high resolution TV is not currently available.

(3) Shared Versatile Interface

The advantages of the central interface computer are:

(i) It is well utilised regardless of the activity patterns at individual terminals (only inexpensive hardware is idle when a particular terminal is inactive).

(ii) Being time-shared, its relative complexity and high performance components have not greatly increased the per-terminal cost.

(iii) It interprets and reassembles (converts) highly compact or complex display codes from the CPU to forms suitable for the simple terminals.

(iv) It removes many tasks specific to graphical communication from the central processor by allowing post-processing of compact central processor codes and pre-processing of graphical inputs.

(v) It generates vectors and symbols in the same way as it executes machine-code; this eliminates special-purpose vector or symbol generators, not only from each terminal but also from the interface.

(vi) It has simplified and unified hardware design as it comprises only OR-NOR gates, a ROM and a core store. Analogue signals first appear at the D/A converters driving the central display(s). This fully digital approach to the generation of graphics is in antithesis to the analogue, or part analogue, generation of specific classes of display curves (cf. Sect. 4.6).

1.11 JUSTIFICATION OF INTERGRAPHIC

The chosen scheme will be justified by showing that it satisfies a number of requirements more economically, or with fewer or less significant shortcomings, than the alternative schemes.

1.11.1 Requirements

An exact and detailed specification of requirements is not intended, but rather, a list of general requirements as a basis for comparing the various schemes.

(1) *Adequate Display Quality;* i.e., adequate resolution, brightness, stability, size, etc. (for definitions, 46,47). A multi-terminal system does not require the precision or stability of a 4096 x 4096 point display as used in the DAC-1 system cited in Sect. 1.2; also, precision terminals are inherently too expensive. For many applications, even 1024 x 1024 point displays are unnecessarily precise and, at the moment, are also expensive for an economical multiterminal system. The author believes that a 512 x 512 point display is the lower limit of resolution for displaying arbitrary graphics in a multi-terminal system, because a lower resolution would prohibit many applications. (For some restricted graphics, a display

of fewer grid points is adequate, provided the points are stable and clearly resolvable, e.g., the 320 x 192 point display of the IBM 1500 System, discussed in Sect. 1.11.2, *Scheme 5*.) Unless the display is used in conjunction with a mechanical overlay (grid, map, etc.), high absolute accuracy is not necessary. A display with noticeable flicker is undesirable - the regeneration or refresh rate of short persistence displays should be at least 20 cps.

(2) *Arbitrary Graphics;* i.e., an ability to display free-form curves and clearly distinguishable symbols from several alphabets.

(3) *Fast Response;* i.e., an ability to display a modified or new image in several seconds. Applications which require a continuous and rapid new frame rate (animated diagrams) demand considerable processing time, and, unless only a few other terminals are active, cannot be accommodated by an economical multiterminal system.

(4) *Graphical and Symbol Input;* i.e., a light-pen or equivalent device for referencing displayed items, moving sub-pictures and inputting arbitrary curves; and a keyboard, or equivalent (e.g., a set of permanent light buttons), for inputting symbols.

(5) *A Flexible and Versatile System;* i.e., a facility to build a repertoire of display oriented instructions at several programming levels which can be efficiently executed (this would enable users to develop new problem-oriented languages and experiment with graphical structures). Also, the system must accommodate an operating system or component of the overall operating system, i.e., it must also be able to execute non-display-oriented functions efficiently

1.11.2 Comparison with Alternative Schemes

Five schemes will be outlined and compared with Intergraphic: three use direct view storage tubes (DVST's), one uses short persistence CRT's regenerated in stroke mode, and the fifth uses modified TV displays, but has no scan converter.

Scheme 1: DVST terminals deflected in parallel from a common pair of analogue deflection buses; terminal selection by DVST unblanking.

If there is no intermediate storage between the interface and the terminals, i.e., if image generation is concurrent with image display, then the image generation rate will be tied to the bandwidth of the deflection system. Thus, the high image generation rates which have been achieved by Intergraphic (10 MHz increments with matching D/A converters) could not be used with the current deflection bandwidth of large DVST's; this would slow the system and, therefore, reduce the number of terminals for a given response time. Also, the transmission of precise, high-frequency analogue signals other than over short distances, is difficult.

Scheme 2: DVST terminals which decode digital image codes (lists of point co-ordinates, increments, etc.); codes may be transmitted on individual circuits or distributed on a common digital bus with terminal selection by coding keys.

A coding compromise exists: compact codes ease transmission specifications, but require elaborate vector and symbol generators at each terminal, whereas expanded codes reduce terminal decoding complexity, but increase the transmission bandwidth for a given image generation time. Reliable code transmission is essential for this scheme. As in *Scheme 1*, the new image generation rate of an interface would be limited by deflection bandwidth (in this case, terminal bandwidth). This mismatch could be absorbed by digital buffers: one buffer per dis-

play (and individual transmission) increases the per-terminal cost significantly, whereas a shared buffer (and common transmission) reduces the new-image rate at a given terminal, i.e., prolongs the average minimum viewing time. A compromise exists whereby terminals are grouped, the terminals within a group sharing a buffer.

An example of *Scheme 2* is the ARDS II terminal (39) which, although originally designed for remote operation over dataphone lines (2000 baud), has been operated at 10,000 baud. The terminal displays arbitrary graphics, but the terminal circuitry is relatively complex. For dataphone connection, the scheme conserves communication bandwidth, but the response time is slow. Fig. 1.4 shows the terminal.

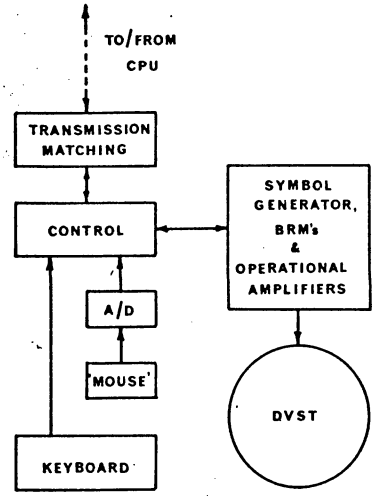


FIG. 1.4. ADVANCED REMOTE DISPLAY STATION II

It comprises a DVST; vector and symbol generators; a keyboard; a "mouse" input device (potentiometer pair); deflection circuitry; and simple control electronics which routes incoming words to the vector or symbol generator, and assembles outgoing words from the keyboard or A/D converters attached to the "mouse". Binary rate multipliers (BRM's) produce vector increments which are added as current pulses into integrating operational amplifiers. Pictorial information, apart from the transient cursor mark which the user positions by moving the "mouse", is accumulated on receipt of digital commands from the central processor. New images require an initial flood erasure. DVST's are less dynamic than regenerated short-persistence CRT's and cannot use conventional light-pen techniques. Several workers (3,48) and the author (27) have developed light-pen techniques for DVST's which are alternatives to the "mouse".

At dataphone rates, even compactly encoded images may need more than five seconds to build up; thus, the choice of codes is of utmost importance. Since the original announcement of the terminal, which described codes for plotting symbols, random points and long vectors, the command form has been changed to mode form, a short-vector code has been included and an incremental mode has been considered[1]. No doubt, alternative combinations of transmission-bandwidth, coding complexity, and response-time will evolve.

Reduction in ARDS-II costs from approximately $10,000 to the projected $3,000 depends upon DVST and integrated array costs. A further per-terminal cost is introduced at the CPU by the mismatch between the data rates of the CPU and dataphone line. Developments in integrated electronics which reduce terminal logic costs, will also reduce the cost of centralized logic. Thus, the relative cost of a system having complex terminals to a system having simpler terminals which share centralized logic, may well remain constant. A disadvant-

---

[1] J.E. Ward, Deputy Director, Electronic Systems Laboratory, MIT --- Private Communication.

age of installing large numbers of terminals having considerable local logic is that transmitted codes and terminal behaviour must be fixed; this would discourage the development of more efficient transmission codes and terminal functions.

Scheme 3: DVST terminals written in TV mode from a central scan converter; terminals time-multiplexed on a single frequency channel and selected by video coding keys, or time-multiplexed within groups each group having a specific channel frequency.

The advantage of this scheme is that video signals need be transmitted once only, thereby eliminating the high information rates necessary for refreshing even static images on short-persistence TV terminals. In a recent review (49, 1968, reprint enclosed), the author advocates TV terminals with local storage (DVST or some other medium) for the terminals of an extensive computer graphics network. This scheme is nearest to Intergraphic, and could become more economical; further advantages are that it eliminates the electro-mechanical video disc, and interlaced TV is not necessary, which simplifies scan conversion.

(Note on DVST costs: When the decision to use a video disc and standard TV terminals was made, the cost of a DVST with power supplies and deflection amplifiers was at least $4000, compared with $1000 for a TV terminal refreshed from a video disc track. Laboratory DVST displays (10cm x 8cm max.) were available and cost approximately $1000, but were considered too small. In late 1967, a 21cm x 16cm DVST display (as used in ARDS II) of good resolution became available (approx. $3000). However, its 20μsec/dot writing time prolongs image generation time (e.g., it would take 0.4 sec to display 1000 symbols, each represented by an average of 20 dots, even ignoring the time needed to position the beam). Thus, buffers would be necessary, at least for every few displays, to maintain an average response time of several seconds. In addition to the cost of this buffering, some further term-

inal circuits would be necessary (decoding and D/A conversion for *Scheme 2*, raster generation for *Scheme 3*). For these reasons, disc refreshed TV displays were chosen in preference to DVST's, but the resolution of 525 or 625 line TV displays is poorer.)

Scheme 4: Short-persistence CRT terminals regenerated in stroke mode from recycled digital codes: the interface generates and routes digital codes once only to core or synchronous (drum, disc or delay line) stores, which are cycled to regenerate displays.

Compact digital codes are preferred for storage, but, as in *Scheme 2*, compact codes increase terminal decoding complexity. A central regeneration store imposes a high total digital transmission rate from store to terminals. Individual terminal regeneration stores, however, reduce the central-to-terminal transmission requirements, but the cost of scattered storage is greater. Core regeneration stores can be filled at a rate matching the interface, but cycled more slowly to match terminal performance. This is not possible with synchronous storage, although several intermediate core buffer areas, written at the interface rates but read more slowly into synchronous regeneration stores once only, could absorb this mismatch.

This scheme is limited by the deflection bandwidth of the large CRT's written in random point or incremental modes (the image must be completed within a persistence of vision period). Also, it demands a reliable digital storage system. An occasional error, or burst of errors, in transmission, however, will produce only one or two faulty regenerations of the display, provided the beam co-ordinates are reset before each regeneration.

An example of *Scheme 4* is the GLANCE terminal system (31,41), developed at Bell Telephone Laboratories. Terminals are 1024 x 1024 displays refreshed at 30 cps from incremental codes stored on a digital disc. The code, expressed in 4-bit groups, specifies one of eight

incremental displacements or controls the intensity level or scaling factor. At a transmission rate of 4×10^6 bits/sec, the display can plot 10^6 points/sec; i.e., a maximum display complexity of about 33,000 points. The cost of storage, as in Intergraphic, is the cost of one track of a disc (in fact, identical discs are used[1]) and cabling is similar; however, the terminal decoding, D/A conversion and deflection circuits are more expensive than TV receiver circuits. A somewhat higher packing density and, therefore, higher bit rate is possible when video signals are stored on the disc because some storage noise can usually be tolerated.The GLANCE and Intergraphic schemes illustrate two uses of high-density disc storage - regenerating an incrementally driven display of about 33,000 points and refreshing a 525 line TV display. For applications which require mostly silhouette type displays, TV coding is preferred, whereas for applications requiring high resolution displays (not limited to 1024 x 1024), incremental coding is essential. For many applications, either scheme is practical.

Scheme 5: TV display terminals refreshed from video disc, but from video patterns computed (assembled) directly in core.

The general concept of this scheme is that a one-to-one image of a display is built in core (i.e., one bit is set in core for each visible display point) in a sequence natural to the description of the display, then the image is read out to the refresh store in a TV scanning sequence. The core is scanned out synchronously with the refresh store, i.e., in a persistence of vision period, and as in Intergraphic, the processor is free for other tasks during this time (assuming the processor has access to other memory). The concept is precise digital scan-conversion, but, if fully developed (i.e., if the entire display area is mapped in core), has two disadvantages:

---

[1] Data Disc Inc., Palo Alto, Calif., U.S.A.

(i) for a 1024 x 1024 display, a memory of some 10^6 bits (possibly organised as 32K of 32 bit words) would be necessary,

(ii) if each display point were set individually, as could be the case for arbitrary curve plotting, a fast memory and addressing scheme would be necessary to generate an image in, say, 30 ms.

An advantage of the scheme is that the expensive storage and logic for the fully developed scheme is shared, and could support say 30 terminals with new images on the average every 2 seconds (assuming equal times for image assembly and image transfer).

A partially developed scheme, however, is well suited for displaying restricted graphics (e.g., symbol arrays in fixed format), because a smaller store is required (e.g., sufficient to hold one line of text only) and groups of display points (e.g., one row of a the dot matrix pattern of a symbol), rather than isolated points, can be assembled in one core access. An example of the partially developed scheme is the IBM 1500 Instructional Display Subsystem (21,40) which has a display format of 40 columns and 16 rows: a character (8 x 12 dot matrix) occupies one column-row intersection, so that the display frame comprises 192 horizontal scan lines each having 320 dot positions. In addition to standard symbols, arbitrary 8 x 12 characters may be defined and placed in the column-row array to constitute a graphic. It is also possible to displace characters vertically by half a row. Although the scheme is not practical for arbitrary graphics (each new graphic would generally require the detailed specification of many new component characters), it is possible to build a restricted class of graphics from a well chosen set of elements. Restricted graphics are adequate for many applications, e.g., annotated block schematics and flow diagrams. Individual CRT displays are specially designed TV receivers with improved positional linearity (the TV raster is non-standard). Each frame of video is buffered on one track of a standard

IBM 2314 disk store which refreshes displays 30 times per second at 2.5×10^6 bits/sec over individual coaxial cable links. For the given flicker rate, the conservative, but accurate, display grid of 61,440 points eases both storage and speed requirements for image assembly.

For arbitrary graphics, conventional stroke mode image generation is inherently simpler and faster than the assembly of a video image in core. Thus, as an ability to display arbitrary graphics was a principal design objective of Intergraphic and the CRT - plumbicon scan converter was inexpensive, the Intergraphic scheme was chosen in preference to *Scheme 5*. (A further advantage of adopting conventional stroke mode generation within Intergraphic was that, as an alternative to driving a set of TV displays, the interface could drive one highly dynamic or highly precise display directly from its X,Y display registers.)

However, the digital precision of scan conversion within *Scheme 5* is attractive, and for a network of 30 terminals, the per-terminal cost of a smaller core store for mapping 512 x 512 point displays would be only about $1000. Further work is recommended in the design of logic for mapping vector points into core (or other digital store).

CHAPTER 2

LOGICAL STRUCTURE OF THE INTERFACE

Tha author uses the term "logical structure" for the functional description of the parts of a machine (i.e., the immediate-access registers, the data paths, the arithmetic-logical unit, the control logic, etc.) and the relationships between these parts. In a microprogrammed machine, the numerous combinations within the logical structure are controlled by microcode. Thus, the logical structure described in this chapter is strongly related to the controlling microcode, the topic of Chapter 3.

The number of registers, the repertoire of basic arithmetic, logical and mapping operations, the extent of additional special logic, the propagation delay of the logic gates, etc., were chosen by trial and error until a set of graphical routines could be executed without numerous register reshuffles, frequent references to core, lengthy microprograms, etc., at speeds corresponding to an average image generation time of 5-10 ms. The logical structure and microcode stemmed from the author's experience as a co-designer of CIRRUS, an earlier microprogrammed computer (50, 1963). There are similarities also with other microprogrammed machines (51, 1964) and proposals (52, 1967), but as far as the author is aware, the "wait", "repeat" and "back" sequencing options, the "microroutine stack", the special micro-order facilities which assist multiplication, division and A/D conversion, and the inclusion of vector and symbol plotting micro-order types within Intergraphic are novel. To a certain extent, the effectiveness of these features, and the structure generally, is shown by the microprograms completed to date, but as in other design procedures, there is no simple measure of merit. Step-by-step justification of the logical structure and microcode is, therefore, not possible, so that much of this chapter and Chapter 3 is purely descriptive.

2.1 DETAILED DESCRIPTION OF GENERAL-PURPOSE PROCESSING SECTION

Fig. 2.1 shows the general-purpose processing section. It comprises,

(1) A Conventional Core-Store[1], cycle time 1.5μs, of 4096, 18-bit words (2 parity bits per word) addressed from register CSA (Core Store Address). Read/write data is held in register CSD (Core Store Data).

(2) A Set of General-Purpose Registers A,B,C,D,E,M,N, each comprising an upper-byte (bits 0-7) and a lower-byte (bits 8-15). Register M is automatically copied from CSD when data is read from core store (read-restore and read-only cycles), and is the source register for writing data into CSD (clear-write and write-only cycles).

(3) A Pair of Register Selection Gates (called the left-operand and right-operand gates), each capable of selecting one of eight, 16-bit registers. The left-operand gate can select a word of 16 zeros (shown "0"), A, B, N, or one of X, Y, S+1, or Q to be defined later. The right-operand gate can select "0", C, D, E, M, or P_i, W, or R to be defined later. The selected 16-bit left-operand is called "a" and the selected 16-bit right-operand, "b".

(4) A Pair of Logical-Negate Selection Circuits which allow $\overline{a}$ to replace a and $\overline{b}$ to replace b. Negation is optional and negation of a is independent of negation of b. Symbol a' denotes a or $\overline{a}$, and b' denotes b or $\overline{b}$.

(5) A Pair of Identical Byte-Length Arithmetic Units (AU's), each capable of addition (+), logical AND ($\wedge$), logical OR (V) and exclusive OR ($\oplus$). The operands for these operations are a'

---

[1] Ampex model RF-1 Ampex Corp., Culver City, Calif., U.S.A.

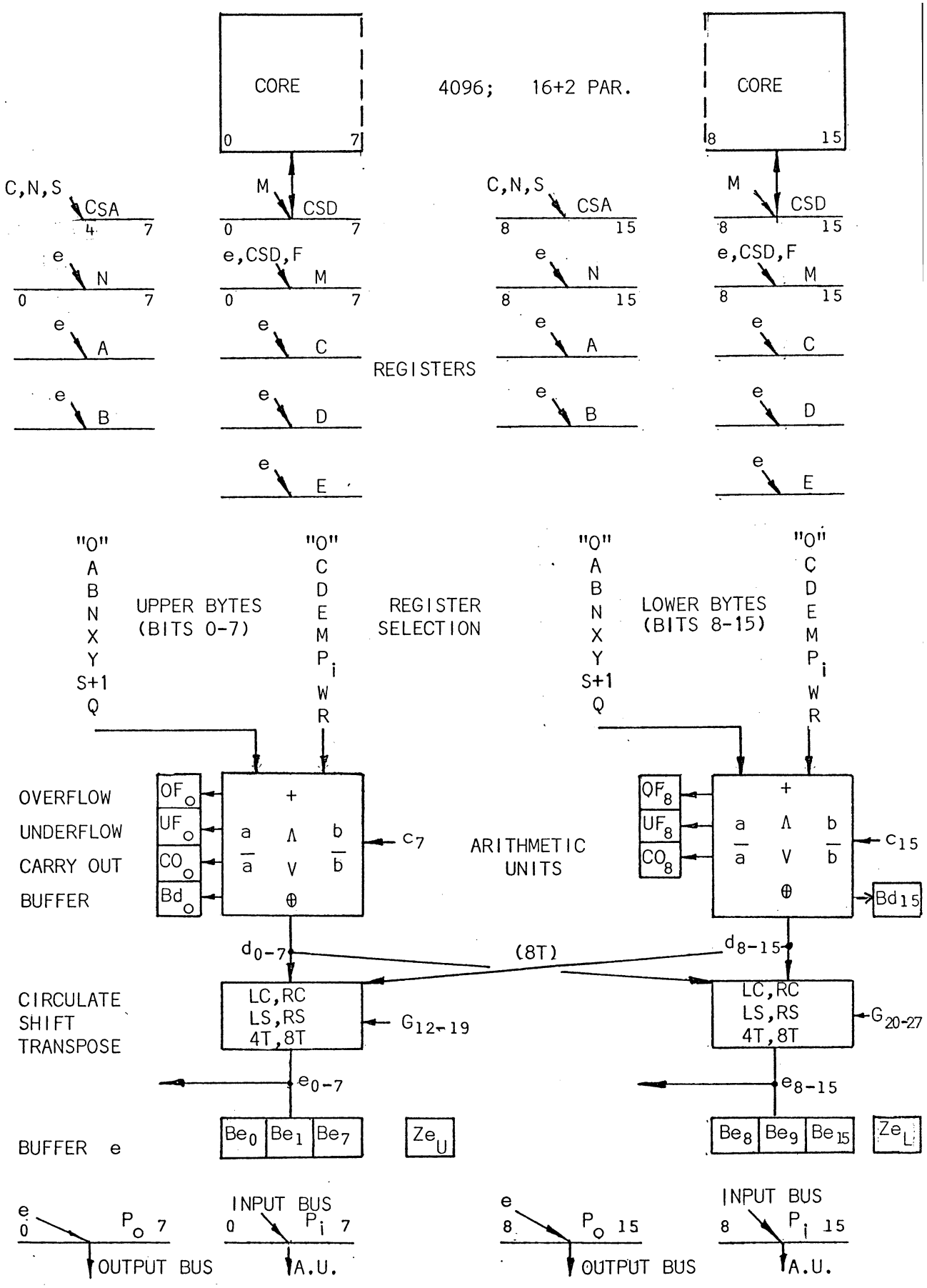


FIG. 2.1. GENERAL-PURPOSE PROCESSING SECTION

and b', and the result of the operation is "d". As the operand gates are each 16-bits wide, the left operands for both the upper and lower AU's must come from the same register; i.e., if $A \vee \overline{M}$ (bits 0-7) is formed in the upper AU, then $A \vee \overline{M}$ (bits 8-15) must be formed in the lower AU.

The two AU's may be operated independently (i.e., carry-out from the lower AU, co_8, does not automatically become the carry into the upper AU, c_7) or concatenated into a 16-bit AU (i.e., $co_8 = c_7$). Overflow and underflow[1] are defined for 2's complement addition as follows,

$$of_o = \overline{a}'_o \, \overline{b}'_o \, c_o$$

$$uf_o = a'_o \, b'_o \, \overline{c}_o \text{ , where,}$$

c_o is the carry into the most significant stage of the adder, and a'_o and b'_o are the sign bits of the operands ("0" = positive). Overflow, of_o, and underflow, uf_o, are buffered in flip flops OF_o and UF_o respectively. Also, the carry-out from the most significant stage, co_o, is buffered in CO_o. Corresponding quantities are defined and buffered for the lower AU, e.g., $of_8 = \overline{a}'_8 \, \overline{b}'_8 \, c_8$ is buffered in OF_8, but these would normally be ignored, although accessible, for 16-bit operation. Result bits d_o and d_{15} are buffered in flip-flops Bd_o and Bd_{15} respectively so that these bits are not lost after a left or right shift respectively (shift specification follows in *(6)*).

Input carry c_{15} can be optionally 0, 1 or CO_o. Option $c_{15} = 1$ allows the negative of one operand to be formed, for, in 2's complement arithmetic, $-A = \overline{A}+1$, the 1 being added

---

1. The thesis uses the term underflow when the result of an operation is more negative than the most negative number which can be accommodated. It is not used in the sense of indicating too small a magnitude in floating point arithmetic.

in the least significant position. (It is noted that it is not possible to form -A -M in the AU.) Option $c_{15} = CO_0$ allows the AU to perform extended length addition. For independent AU operation, c_7 can be optionally 0 or 1, but these options are tied to the options for c_{15}, i.e., $c_7 = c_{15}$ (there is no carry option c = CO for independent AU operation).

(6) Circulate, Shift and Transpose Gates which map d into "e". Seven options are available, one of which is the trivial mapping e ← d. Right shift, RS, preserves sign and left shift, LS, clears the least significant bit. (Only single bit shift or circulate options are available.) Circulate (C) and shift are defined as follows;

For independent AU operation,

RC: $e_{0-15} \leftarrow d_7, d_{0-6}, d_{15}, d_{8-14}$

LC: $e_{0-15} \leftarrow d_{1-7}, d_0, d_{9-15}, d_8$

RS: $e_{0-15} \leftarrow d_0, d_{0-6}, d_8, d_{8-14}$

LS: $e_{0-15} \leftarrow d_{1-7}, 0, d_{9-15}, 0$

(It is noted that the same shift or circulate type applies to both bytes; e.g., RC upper byte, RS lower byte is impossible.)

For 16-bit operation,

RC: $e_{0-15} \leftarrow d_{15}, d_{0-14}$

LC: $e_{0-15} \leftarrow d_{1-15}, d_0$

RS: $e_{0-15} \leftarrow d_0, d_{0-14}$

LS: $e_{0-15} \leftarrow d_{1-15}, 0$

Transpose mappings are the same for independent or concatenated AU operation; they are defined as follows,

4T: $e_{0-15} \leftarrow d_{4-7}, d_{0-3}, d_{12-15}, d_{8-11}$

8T: $e_{0-15} \leftarrow d_{8-15}, d_{0-7}$

Thus, 4T is equivalent to a four position left or right circulation, independent AU operation; 8T simply interchanges the upper and lower bytes of the AU result.

(7) A Machine Distribution Bus which distributes the output from the mapping unit, e. Via this bus, e can be nominated as input to one of 16 destination registers which are, with one exception, the set of operand registers listed in *(3)* above. One of the 16 destination registers is pseudo (denoted NIL) so that, for example, overflow can be detected in buffer OF_o without changing any of the machine registers.

The minimum path delay from source register through the selection, negate, arithmetic and mapping logic is sufficient for a source register to be also nominated as destination, without additional buffering. The maximum delay (i.e., allowing for 16-bit addition) for a complete cycle is approximately 100 ns.

Outputs e_0, e_1, e_7, e_8, e_9 and e_{15} are buffered in flip-flops Be_0, Be_1, etc., at the time the nominated destination register is set (clocked). Also at this time, buffer Ze_U is set to 1 if the upper byte of e is zero, and similarly Ze_L for the lower byte. By nominating destination NIL, these buffers (which are clocked regardless of whether the destination register is NIL or not) allow the sign of a result, or whether a result will be zero or not, etc., to be formed without changing any of the machine registers in the same way as overflow, etc., above. Also, any bit of any source register can be inspected via these buffers; e.g.,

A_o becomes Be_o for the operation,

$A + \text{"0"} \rightarrow NIL,$

D_5 becomes Be_1 for the operation,

$\text{"0"} + D \xrightarrow{4T} NIL.$

(Access to buffers and control are detailed in Chapter 3).

For independent AU operation, there are three options for clocking the destination register: all 16 bits are clocked (i.e., copied from e), the upper byte only is clocked, or the lower byte only is clocked. The double-byte independent AU option is designated U/L. The single-byte options, however, are not nominated directly, but rather by nominating which AU generates the byte of e which is copied: if the upper AU generates the single byte which is copied into the destination register, the option is designated U; and similarly, if the lower AU is the source then the option is designated L. The 8T mapping introduces the need to distinguish between which AU generates the result and which byte of e is copied; for other mappings the upper AU generates the information copied into the upper byte of the destination register, and similarly for the lower byte.

For concatenated AU operation, all 16 bits of the destination register are always clocked; this option is designated UL.

(8) Input and Output Registers P_i and P_o respectively. The connection of peripheral devices is via a peripheral input bus, P_i, and a peripheral output bus, P_o, each of which has 18 bits including bytewise parity. All peripheral transfers will be executed by microcode sequences within Intergraphic, the particular sequence being entered from OS/IG (cf. Section 1.10.2). Status changes in peripheral devices set indicators which, in turn, condition the execution of various jumps distributed within microcode sequences. These traps return con-

trol to OS/IG which identifies the status change in detail and branches control accordingly. Peripheral control of paper-tape equipment, etc., is not central to the thesis and will not be described further.

2.2 DETAILED DESCRIPTION OF THE MICROPROGRAM CONTROL SECTION

Fig. 2.2 shows the microprogram control section. It comprises,

(1) A Read-Only Memory (ROM), access time 50 ns, of 4096, 32-bit words (1 parity bit per word) addressed from bits 4-15 of register S_{0-15}. Register G_{0-31} is the output buffer.
(The ROM contains microprograms, register S is the microprogram instruction counter, and G is the microprogram instruction register.) S may be incremented, decremented, copied from the distribution bus e, or copied from a microroutine stack to be described in *(2)*. Normally, S is incremented and micro-orders are executed in a simple contiguous sequence. The decrement option returns control the preceding micro-order, a mode detailed later (Sect. 3.2, under ABA, BK). Option e → s enables microprogram control to be transferred to a computed address, and popping the microroutine stack into S returns control to a pre-stacked address (detailed in Sect. 3.2, AQS). Register G is partitioned into fields which control the gate and processing options outlined in Sect. 2.1 and other options to be detailed in Chapter 3.

(2) A Microroutine Stack (Q, Q', Q'') pushed and popped from Q. The stack may be pushed from distribution bus e or from S+1, the incremented value of S; it is popped either if Q is nominated as an operand register or on returning from micro-

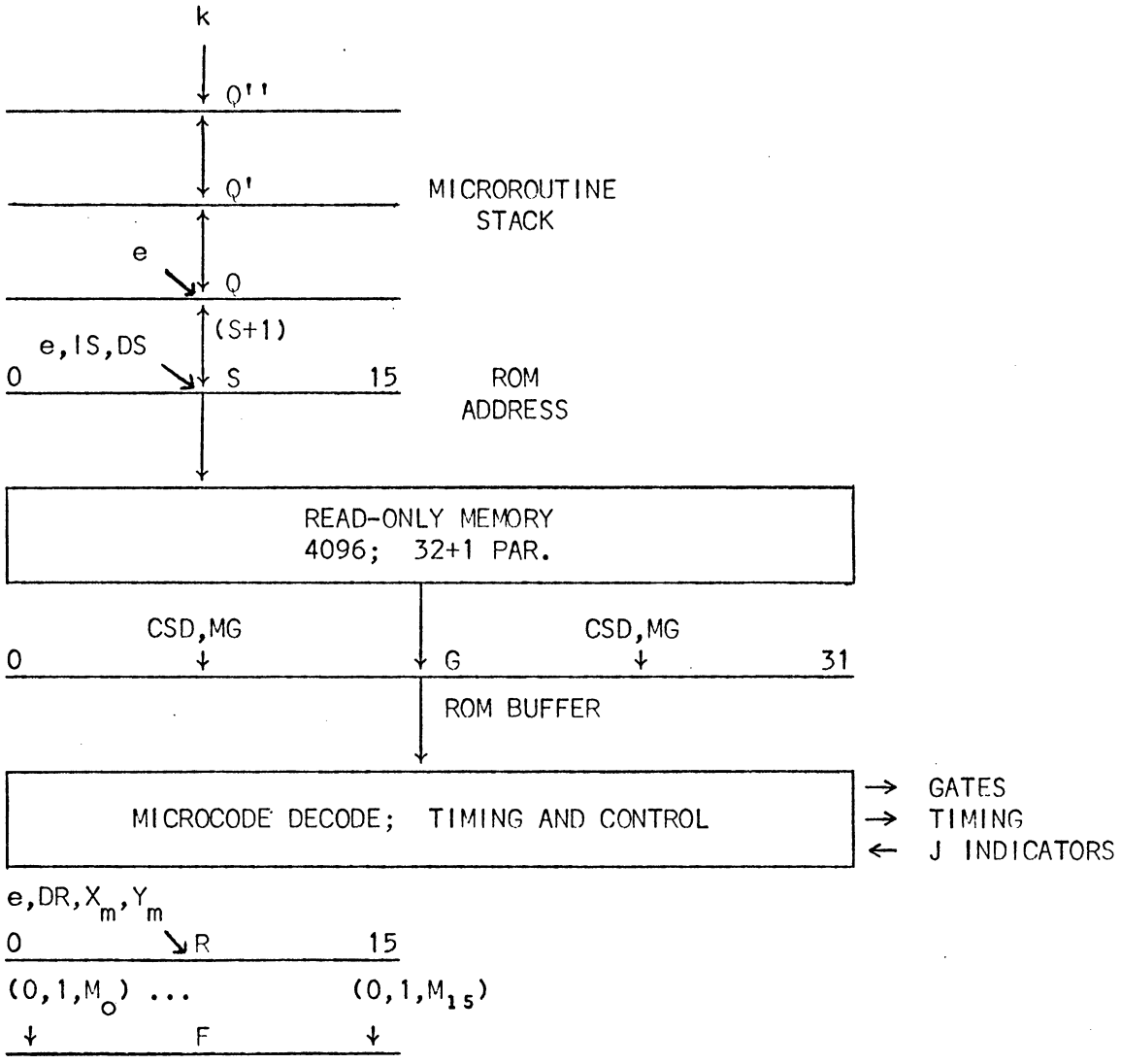


FIG. 2.2. MICROPROGRAM CONTROL SECTION

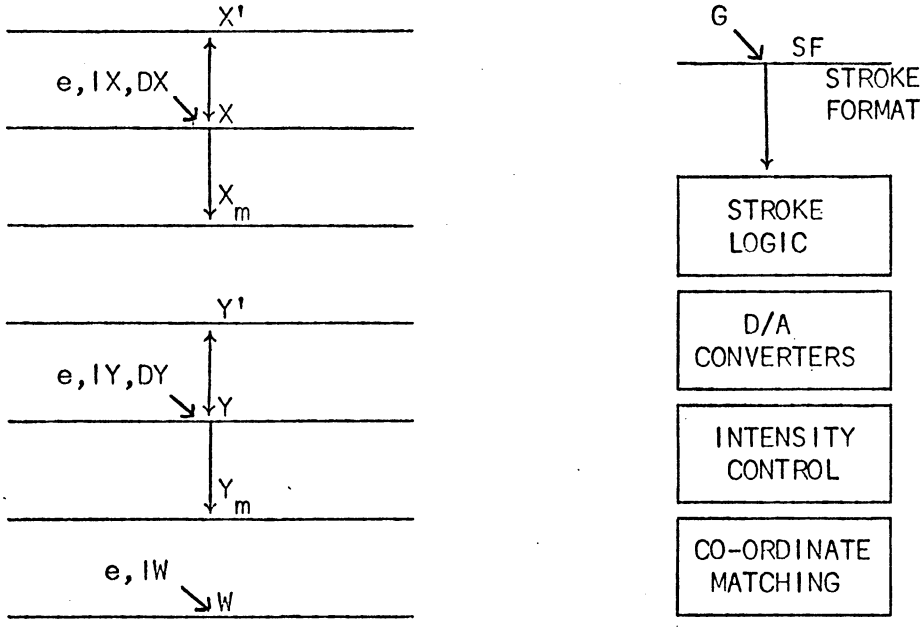


FIG. 2.3. SPECIAL-PURPOSE SECTION

routines which use the microroutine stack facility.

(3) *A Counter* R_{0-15} which can be used as a general purpose register (in the same way as register D for example), as a decrement counter, or as an intermediate buffer which allows registers X_m, Y_m of the display section (Sect. 2.3) to be accessed. Logic also detects when R is zero.

(4) *A Bitwise Programmable Register* F. The 16 flip-flops which constitute F may be individually cleared, set, or copied from corresponding bits of M. Any number of F bits can be nominated (selectively masked) for clearing, setting or copying for M, but these three operations cannot be mixed throughout the F word. F is neither a source nor destination register of the processing section; rather, it communicates via M. F provides a set of programmable indicators, some special-purpose, some general-purpose, for switching mircoprogram control.

2.3 DETAILED DESCRIPTION OF THE SPECIAL-PURPOSE SECTION

Fig. 2.3 shows the special-purpose display section. It comprises,

(1) *The Display Co-ordinate Counters,* X,Y. Both X and Y are source and destination registers of the processing section and are 16 bits long; thus, they can be accessed and jam-set as general-purpose registers. Also, X and Y are incremented/decremented while the interface is operating in vector and symbol plotting modes. The least significant 10 bits of X and Y (bits 6-15), drive the D/A converters which position the beam of the central CRT; bits 0-5 allow a higher resolution CRT to replace the central CRT for special applications,

and allow the "display" to extend beyond the display window.

(2) *The Registers,* X',Y', for buffering X,Y. This register pair is jam-set from X,Y whenever the display co-ordinates need to be temporarily recorded; later the recorded co-ordinates are copied back into X,Y. (Copying the contents of X',Y' into X,Y does not change the contents of X',Y'. One use of X',Y' is to hold an origin to which the display is reset after each vector of a list of vectors is plotted; this results in a set of radial vectors ("spokes") from the origin X',Y', instead of an end-to-end vector plot.)

(3) *The Coordinate-Matching Registers* X_m,Y_m which are preset to a particular co-ordinate pair and subsequently compared with the current co-ordinate pair X,Y. When the corresponding bits of X_m,Y_m and X,Y match, co-ordinate matching logic sets an F indicator. Two indicators are provided for matching: one for an exact match over all 16 bits; the other for a match extending to all but the least significant three bits. (Chapter 6 details the use of co-ordinate matching for light-pen referencing.)

(4) *A Pair of 10-bit D/A Converters.* A settling time of 20 ns has been achieved using non-saturating circuits.

(5) *An Arc Length Counter,* W, which, in small polar vector mode (Sect. 4.2.1), integrates increments of arc length and thus holds distance from the start of an arbitrary curve. An approximation to arc length is recorded in Cartesian and large polar vector modes. A nominated bit of W, or the logical "OR" of several bits, controls the unblanking of the display beam: this gives broken display lines of various on-off patterns. (Sect. 4.3 details this "Line-Form", LF, control.)

(6) *Stroke Generation Logic.* Strokes for symbol plotting, coded four to a ROM word are buffered in stroke-format register SF from G and interpreted by stroke logic. Each stroke in SF produces a pair of X,Y increment/decrement streams and corresponding unblanking pulses. Chapter 5 details the coding and interpretation of symbol strokes.

CHAPTER 3

DEFINITION OF THE MICROCODE

This chapter defines the Intergraphic microcode, i.e., the set of micro-orders which controls the logical structure detailed in Chapter 2. The word-length chosen for the code was 32 bits: this was sufficient to accommodate a range of arithmetic, immediate-constant, transfer, indicator-setting and special graphical micro-orders, as set out below, and also, 32-bits was a multiple of the core-store word-length (16 bits), so that micro-orders, stored one to a pair of core words, could be executed from core store for preliminary testing, before being wired into the ROM. The mode is believed to be novel, and is particularly useful for debugging new microprograms; it is described in APPENDIX 2.

3.1 OUTLINE OF THE MICROCODE

Table 3.1 sets out the microcode formats. Within any column of options shown in the table, the codes corresponding to mnemonics are ascending binary integers; e.g., for field G_{0-3},

0000	codes	ABA
0001	"	ADR
0010	"	ASQ
:		:
1111	"	SFS

Table entries shown "@" are available for future definition; entries shown "#" are forbidden.

There are 16 micro-order types, designated by field G_{0-3}, which are arranged in five main groups: A-orders, D-orders, the TRF

REGISTER G BIT LOCATIONS

0 1 2 3	4 5	6 7 8 9 10 11	12 13	14 15 16	17	18 19 20	21 22 23	24	25 26 27	28 29 30 31
ABA	L			"0"		(OPRN.)				(DEST.)
ADR	U					+,0	"0"		d	NIL
ASQ	U/L			A		+,1	C		LC	A
AQS	UL									B
	RR			B		+,CO	D		RC	N
ACC	CW									
ACN	RO		FW	N		@	E		LS	P_o
	WO		WT	X	a	Λ	M	b	RS	C
	APV									D
ASO	APP		RP	Y	$\overline{a}$	V	P_1	$\overline{b}$	4T	
	@									E
	@		BK	S+1		⊕	W		8T	X
	AMY			Q		@	R		#	Y
AS1	ADV	J								S
	AAD			↓			↓		↓	
	AXS			a			b		e	Q
DMC										M
@	L		MICROPROGRAM CONSTANT							@
DSQ										W
DQS	U		12			19 20			27	R
TRF	#		X ↓ X'	Y ↓ Y'	X' ↓ X	Y' ↓ Y	X ↓ X_m	Y ↓ Y_m	X_m ↓ R	Y_m ↓ R	F ↓ M							0000
	UL																	
FBC	CF			M			P	V										
	SF			M			C	M		V								
	#						D											
	MF																	
IFS SFS	v	r	p	q	ℓ	as 4-10			as 4-10							as 4-10		

TABLE 3.1. INTERGRAPHIC MICROCODE

order, the FBC order and FS-orders. A brief outline follows; the orders (excepting FS-orders, cf. Sect. 5.2) are defined in detail immediately after this outline.

A-orders (Arithmetic) control the general-purpose processing section, i.e., control the register selection gates, logical negation, the AU mode, the mapping gates, and the destination clocks. Concurrently with this basic source-to-destination control, A-orders can also decrement R, push and pop the microroutine stack, initiate a core-store cycle, and establish some special conditions which facilitate vector plotting, point plotting, multiplication, division, analogue-to-digital conversion, and extended length operation.

D-orders (Distribute) bypass the AU's and mapping gates; they simply distribute an immediate-constant nominated in field G_{12-27} (microprogram constant) to a destination register. Concurrently with distribution, D-orders can also push and pop the microroutine stack.

The TRF order (Transfer) controls register-to-register transfers which are not possible via the general-purpose processor.

The FBC order (F register, Bitwise Control) controls the clearing, setting, and copying from M of individual F bits.

FS-orders (Formats) contain four identical fields which encode the detailed incremental or stroke composition of symbols.

3.2 DEFINITION OF A-ORDERS

Eight of the 16 micro-order types are A-orders. The first, designated ABA, is the basic arithmetic micro-order; the remaining 7 augment or modify ABA. The detailed codes following should be read in conjunction with Chapter 2.

ABA (Arithmetic, Basic), $G_{0-3} = 0$.

Field $G_{4,5}$ specifies whether the AU's are operated independently (L, U, or U/L) or concatenated to 16-bits (UL), and, if independently, whether the result received by the destination is actually generated in the lower AU only (option L), the upper AU only (option U) or both AU's (option U/L).

Field G_{6-11}, designated J, nominates one of 64 binary indicators which conditionally control the execution of the current micro-order. Examples of J indicators are OF_0, Be_7 and $\overline{Ze}_U$, designated JOFU, JBe7 and JNZeU respectively. If the nominated J indicator has the binary value "1", the micro-order is executed, otherwise the order is inhibited. J_0 ($G_{6-11} = 0$), instead of being a machine buffer or the output from a logical circuit, is the constant "1"; thus, J_0 specifies unconditional execution. TABLE A3.2 lists the J indicators. (In the following description, J=1 means that the value of the nominated J indicator is currently 1; i.e., J=1 does not represent the indicator J_1.)

Field $G_{12,13}$ specifies the options, FW (Forward), WT (Wait), RP (Repeat) and BK (Back) which are defined in conjunction with the value of J, as follows,

FW: Except for the case of a successful computed jump order (i.e., A-order, J=1, destination register = S), the next mirco-order is read from ROM address S+1: this is the simplest instruction sequence (J controls whether the order is executed or not; the next order is always read from S+1, regardless of J).

WT: If J=1, the order is executed immediately and, except for a computed jump, the next order is read from S+1 (identically to FW). If J=0, the order is indefinitely delayed until the nominated indicator becomes 1. This option enables the machine timing to be tied to external

events. For example JCRA (J Core-Read-Available), which is "0" at the start of every core store cycle and becomes "1" when read-data is available, is nominated to delay a micro-order which requires core data as an operand until the data is available. Unless there is a machine fault, the nominated J of a WT order always becomes 1 and the conditions stated under J=1 then apply.

RP: This option allows a micro-order to be repeated until the nominated J indicator becomes 0. When 0, or if initially 0, the order is not executed and control advances to S+1. (There is no possibility of a computed jump, because a repeated jump is meaningless.) To illustrate the RP option, consider the J indicator JNZR (J Non-Zero R) conditioning the simple arithmetic order which decrements R,

ABA, UL, JNZR, RP, R-1 → R.

Then, if R=8 initially, the order will be repeated 8 times, thereby clearing R. On the 9th attempt, however, JNZR = "0", the order is not executed and control advances to S+1. (A more meaningful example of RP follows under A-order ADR later.)

BK: If J=1, the order is executed and the next order is read from S-1; otherwise, the order is not executed and the next order is read from S+1. (As for RP, BK would be meaningless if used in conjunction with a destination of S, i.e., a computed jump.) AN FW-BK order pair enables two micro-orders to be alternated repeatedly until the J indicator clears (both orders would nominate the same J indicator). This alternation effectively extends RP to a pair of micro-orders; it saves a third test and branch order. For example, an FW-BK pair is used in the multiplication of two 16-bit numbers where two passes of the AU are necessary to

accumulate each partial product and shift the double-length accumulator which finally holds the product.

Field G_{14-16} nominates the left-operand source for the AU's, i.e., which register becomes "a". Before any micro-order held in G is interpreted, the sequence register S is automatically advanced to S+1 in anticipation of a simple advance of control; thus, if the S register is selected to become the left-operand (G_{14-16} = 1,1,0), the value selected is S+1.

Bit G_{17} controls the negation of "a": if $G_{17} = 1$, $\overline{a}$ replaces a.

Field G_{18-20} nominates whether the AU's perform addition or a bitwise logical operation: variants of addition are input carry = 0, 1 or CO denoted by "+,0", "+,1" and "+,CO" respectively; logical operations are AND, OR and exclusive OR denoted by "$\wedge$", "$\vee$" and "$\oplus$".

Field G_{21-23} nominates the right-operand source for the AU's, i.e., "b", and G_{24} controls the negation of "b" (if $G_{24} = 1$, $\overline{b}$ replaces b).

Field G_{25-27} nominates the mapping type, i.e., the transformation of d into e. The entry coded by G_{25-27} = 0,0,0, shown "d", indicates e = d. Shift, circulate and transpose are defined in Sect. 2.1 *(6)*.

Field G_{28-31} codes the destination register. (Byte clocking within the destination register is specified by $G_{4,5}$.)

ADR (Arithmetic, Decrement R), $G_{0-3} = 1$.

This micro-order augments ABA by decrementing R each time the order is executed. Register R cannot be nominated as a source or destination within an ADR order; this avoids the need to define, and

control precisely, when R is decremented within the cycle. R is decremented sufficiently late in the cycle so that JNZR (cf. ABA, RP) is derived from the pre-decremented value of R. The combination ADR, JNZR, RP with R initially set to n, is a convenient way of executing an arithmetic order n times; without the ADR, J and RP facilities each iteration would take 3 micro-orders (the computation step, a decrement order and a test jump). As examples, the compact count facility is used in multiple shifting, byte-length multiplication and vector plotting. In each of the last two examples, the single micro-orders which are iterated are extensions of ADR. They are AMY and APV which are defined later in this section.

ASQ (Arithmetic, S to Q), $G_{0-3} = 2$.

This order augments ABA by also pushing S+1 into the micro-routine stack if the order is executed. The S+1 → Q transfer is early in the cycle; thus, S refers to the address of the current order and not a computed address which would be formed late in the cycle if S were nominated as destination. A single ASQ order, then, with S nominated as destination, plants the return address, S+1, in the microroutine stack and transfers control to a computed address. Q cannot be nominated as a destination or source register within an ASQ order; this avoids the possibility of either double-pushing or pushing-and-popping in one cycle, which would require a complex sequence definition and precise timing.

AQS (Arithmetic, Q to S), $G_{0-3} = 3$.

This order is designed to return control automatically to the calling microprogram at the end of a microroutine, provided the terminating order of the microroutine is arithmetic. AQS augments ABA by popping Q into S if the order is executed. As in ASQ, Q cannot be nominated as a source or destination register. It is meaningless to nominate S as destination register. As many microroutines are short

and do terminate in an arithmetic operation, AQS has contributed to microcode efficiency.

ACC (Arithmetic, Core addressed from C), $G_{0-3} = 4$.

ACC initiates a core-store cycle after the arithmetic operation has been completed. The order addresses the core-store from register C by copying C into CSA, the internal address register of the core-store. The contents of C are not transferred until the destination register of the arithmetic order has taken its new value; thus, the order, ACC, C+1 → C, addresses the core from the incremented value of C. This is useful for fetching instructions or data from contiguous addresses as only one ACC execution per access is necessary. Field G_{4-5}, instead of specifying L,U, etc., nominates the type of memory cycle. Options are, read-restore (RR), clear-write (CW), read-only (RO), and write-only (WO). ACC implies UL operation. APPENDIX 1 defines the memory cycles, the three J indicators which refer to memory cycle events, and some operational constraints.

ACN (Arithmetic, Core addressed from N), $G_{0-3} = 5$.

This order is identical to ACC except that the core is addressed from register N instead of C.

ASO (Arithmetic, Special Group 0), $G_{0-3} = 6$.

This "order" is a group of four orders; only two of the orders within the group, viz., APV and APP, have been defined to date, the remaining two are spare.

APV (Arithmetic, Plot Vector) has been designed for plotting vectors. Each cycle of the AU in an "APV, RP" mode, usually

generates an X increment/decrement or a Y increment/decrement or both, and, as the time for one pass of the AU is approximately 100ns, vector increments are produced at a rate of about 10MHz.

The APV order implies U/L operation, and augments ABA, U/L by;

(i) incrementing X if the upper AU overflows,

(ii) decrementing X if the upper AU underflows,

(iii) incrementing Y if the lower AU overflows,

(iv) decrementing Y if the lower AU underflows,

(v) decrementing R on every execution,

(vi) applying an unblanking pulse (80ns duration) to the central CRT 40ns after X or Y has been incremented or decremented; this plots a point after the D/A converters have settled (20ns) and the CRT beam has moved incrementally (20ns),

(vii) preserving the sign of an addition on overflow or underflow; i.e., d_0 (or d_8) is forced to be "0" if two positive operands overflow the upper (or lower) AU, and d_0 (or d_8) is forced to be "1" if two negative operands underflow the upper (or lower) AU.

The APV order, defined above, performs the iteration within all types of vector plotting. The justification of APV, and a detailed description of its application to vector plotting, follows in Sect. 4.3.

APP (Arithmetic, Plot Point) augments ABA by applying an unblanking pulse of 80ns duration to the central CRT, 40ns

after the destination register has assumed its new value. The 40ns delay has been designed for APV which only increments X or Y. For random point plotting, however, a 200ns settling time is necessary; thus, if X or Y is set by an order in preparation for plotting a point, then the unblanking APP order must be separated from this order by a third order, (this ensures a delay of about 240ns). MI PLT SML CTN PNT D (p A3.33) terminates with the following 3 orders;

ABA	Y + D → Y	SET Y REGISTER
ABA	NIL	NO OPERATION, DELAY ONLY
APP	Q → S	POP STACK (RETURN) & UNBLANK CRT.

AS1 (Arithmetic, Special Group 1) $G_{0-3} = 7$.

This "order" is a group of four orders, AMY, ADV, AAD and AXS which are defined as follows:

AMY (Arithmetic, Multiply) is a variant of ADR which performs the iteration within byte multiplication. (MI BYT MPY BU*EL→E, p A3.49 , illustrates the use of AMY in detail.) AMY implies UL operation, and as well as decrementing R each cycle to control the number of iterations, the order AMY;

(i) conditionally inhibits the "a" operand depending on Be_{15}; i.e., the "a" operand becomes,

$$Be_{15}.a$$

(In the byte-multiply algorithm, Be_{15} contains a copy of the current multiplier bit and "a" nominates the multiplicand; thus AMY accumulates a partial product which is either the multiplicand or "0". The accumulation of partial products is progressively right-shifted to form the product),

(ii) preserves the true sign of a sum, regardless of overflow or underflow, on right-shift. (Normally, if two positive operands overflow the AU, then d_o becomes "1" and this would be extended to both e_o and e_1 on right-shift. AMY, however, forces e_o to be "0" on right-shift for overflow, and forces e_o to be "1" on right-shift for underflow. This modification effectively extends the length of the accumulator by one bit as required by the multiplication algorithm.)

The inclusion of AMY has simplified both byte multiplication and word multiplication (MI WRD MPY B*E→ED, p A3.50) and has resulted in rapid execution; e.g., MI BYT MPY comprises only 9 micro-orders and multiplies two signed bytes to form a signed double-byte product in 1.4μs.

ADV (Arithmetic, Divide) assists non-restoring division by conditionally forming the negative of the nominated "b" operand depending on Be_o. ADV implies UL operation. If $Be_o = 0$, then the nominated fields "+,0" and "b" are replaced by "+,1" and "$\overline{b}$", whereas if $Be_o = 1$, the nominated fields remain unchanged. The non-restoring division algorithm is not included in the thesis.

AAD (Arithmetic, Analogue-to-Digital) has been designed to assist A/D conversion; it is similar to ADV, except that the output of a digital-differential-comparator, DDC, replaces Be_o. The comparator compares the output of the D/A converter driven from the X register with an unknown external voltage: if the unknown exceeds the D/A converter output then DDC = 1, otherwise DDC = 0. The AAD order adds/subtracts progressively-halved increments to/from register X, the number of iterations depending on the precision of the converter and comparator. The algorithm executes two orders (an FW-BK pair) per iteration (i.e., 200ns/bit) and requires a comparator with

a response time not exceeding 100ns.

AXS (Arithmetic, Extended Shift) enables shift and circulate operations to be extended to double-word, or longer, register sets; it implies UL operation.

For right-shift, instead of copying d_0 into e_0 (i.e., copy sign), AXS copies Bd_{15} into e_0. Bd_{15} buffers the least significant bit of the previous UL operation which would normally be lost on right-shift as it would not be mapped into any e bit. Extended right shift sequences begin by applying an "ABA, UL, RS" order to the most significant word of the register set (this copies the sign bit in the usual way), then an "AXS, RS" order shifts the second word component (this ensures that the least significant bit of the first word becomes the most significant bit of the second word), and so on until the last word or byte of the register set is reached.

For left-shift, AXS copies Bd_0 into e_{15}. A left-shift sequence begins at the least significant word component with an "ABA, UL, LS" order, then "AXS, LS" orders follow.

Similar procedures follow for circulate, except that an initial NIL order is required to load Bd_{15} or Bd_0, and all subsequent orders are AXS orders.

3.3 DEFINITION OF THE D-ORDERS

The code provides for four D-orders: three have been defined; the remaining order is spare.

DMC (Distribute Microprogram Constant), $G_{0-3} = 8$.

This order enables a constant, held in field G_{12-27} of DMC, to be immediately loaded into a destination word or byte. There are three options specified by G_{4-5};

(i) option L: the lower byte, G_{20-27}, is transferred to the lower byte of the nominated destination,

(ii) option U: the upper byte G_{12-19}, is transferred to the upper byte of the destination,

(iii) option UL: the whole word G_{12-27} is transferred to the destination word.

The destination and J fields are as defined under ABA. DMC implies FW operation; hence the next order is read from S+1 regardless of J, except for a successful jump order (nominated J = "1", destination = S). DMC with destination S, then, is useful for conditional (or unconditional, if J_0 is nominated) transfer of control to a known address. For other destinations, DMC conditionally clears registers, loads known mask patterns, etc., then increments control to S+1.

DSQ (Distribute Microprogram Constant, S to Q), $G_{0-3} = 10$.

This order augments DMC in the same way as ASQ augments ABA; i.e., a single DSQ order, with destination S, if executed, plants the return address, S+1, in the microroutine stack and transfers control to the address coded in G_{12-27}. This use of DSQ is frequent and has improved microcode efficiency. The J facility conditions the insertion of the microroutine beginning at the address coded in G_{12-27}.

DQS (Distribute Microprogram Constant, Q to S), $G_{0-3} = 11$.

This order is similar to AQS, i.e., it is designed to return control automatically to the calling microprogram at the end of a microroutine, but, in this case, the terminating order of the microroutine must be a distribute order. To avoid both pushing and popping the microroutine stack in the one DQS order, Q cannot be nominated as destination.

The DQS order is particularly efficient for writing tables of constants in the ROM. For example, the sine table (TABLE SIN (DL) → D, p A3.35) comprises 129 values of Sin θ corresponding to the argument range,

$$\theta = 0 \ (\pi/256) \ \pi/2 \ \text{radians},$$

i.e., the first quadrant (including π/2), in steps of π/256 radians. Each table entry is a single DQS order which, when accessed, loads the appropriate Sin θ value into register D and transfers control to the address stored in Q. Without the QS facility, each table entry would require two DMC micro-orders having destinations D and S respectively.

3.4 DEFINITION OF TRF (TRANSFER), $G_{0-3} = 12$.

This order enables a number of register-to-register (or byte-to-byte) transfers to be executed concurrently.

Field G_{12-27} comprises 16 individual bits, each corresponding to a transfer: if a "1" is coded, the appropriate transfer is executed, otherwise it is not; e.g., G_{14} = "1", transfers X' to X. To avoid timing hazards, a register cannot be the destination of one transfer and the source of another transfer in the same TRF order (e.g., G_{12} and G_{14} cannot both be coded "1"). Fields $G_{4,5}$ and J are as defined under D-orders, and the field G_{28-31} must contain

zeros. Concurrent transfers are of the same word type (L, U or UL), and J conditions all transfers. The TRF order enables transfers to be executed which are not possible via the "e" distribution bus. Also, concurrent transfers are convenient for transferring co-ordinate pairs X,Y → X',Y , etc. The vector-plotting option which resets the display to the local origin X',Y' after each vector is plotted, is achieved by an X',Y' → X,Y transfer, the nominated J being a copy of the end-to-end/reset bit (cf. Sect. 4.2.1, *(2)*).

3.5 DEFINITION OF FBC (F REGISTER BITWISE CONTROL), $G_{0-3} = 13$.

This order, like TRF, partitions G_{12-27} into 16 individual bits. The command option coded in G_{4-5} viz., "clear" (CF), "set" (SF) or "copy from M" (MF), applies to each F bit for which the corresponding bit within G_{12-27} is coded "1". (Within any one FBC order, it is impossible to mix CF, SF and MF commands.) F bits 0-7 are reserved for special functions, whereas bits 8-15 are general-purpose programmable indicators. An example of a special-function indicator is F_5 (controlled via G_{17}), designated visible mode (VM), which is a master unblanking control: if F_5 = "0", the CRT beam remains blanked; if "1", other logic can unblank the beam. All F bits are also J indicators, and F can be copied into M by a TRF order, as it is for example in storing the state of the machine.

3.6 FS-ORDERS

There are two FS-orders: Increment Formats (IFS) and Stroke Formats (SFS). Each order comprises the order-type field, G_{0-3}, and four identical fields or formats, G_{4-10}, G_{11-17}, G_{18-24} and G_{25-31}. Sect. 5.2 details the interpretation of these formats, and the application of IFS and SFS to symbol plotting.

CHAPTER 4

VECTOR GENERATION

4.1 THE NEED FOR EFFICIENT VECTOR CODES AND GENERATORS

Vectors or straight-line segments are used so frequently in the construction of arbitrary graphics that efficient vector coding and generation schemes are essential: efficient coding to conserve bandwidth and storage; efficient generation to achieve high plotting rates with a minimum of hardware. Because graphics usually comprise large numbers of vectors drawn end-to-end, precision interpretation of codes is essential, i.e., generation of the exact number of horizontal and vertical increments of a vector. Where errors can accumulate, e.g., in plotting strings of polar vectors which are first converted to Cartesian form, a set of check points (i.e., points at which the display is reset to precise absolute locations) may be necessary. Silhouettes and shaded areas are also included in arbitrary graphics, and these usually comprise large numbers of vectors which cover an area either by a series of parallel scan lines (e.g., the type font synthesis application reported in (14)) or by a series of contours which are derived from the boundary shape.

Although many applications need only horizontal and vertical lines, and display files can be compressed if this is a constraint, arbitrary graphics require vectors of any orientation. Also, vector codes and generators must accommodate a wide range of vector magnitudes. A format designed for long vectors is inefficient in space if used to code a piecewise linear approximation of an intricate curve, and conversely, it is inefficient to build long vectors from individually coded increments or very short vectors. Therefore, several vector formats are necessary and, for a 16-bit core store word, one would expect a long vector format of two words (one containing the horizontal component, the other the vertical component), a short vector format

of one word (each half-word containing a component) and, possibly, an incremental vector format of four increments to a word.

Vectors are most frequently drawn end-to-end; therefore, a block mode in which vectors are coded in variable-length blocks, each block labelled with a header word, is more efficient in space than individual vector commands, each of which must carry a label as well as the pair of components. In block mode, block length may be specified within the header, or alternatively, each vector may carry a link or continue bit. An individual blanking bit for each vector is normally included in vector codes so that isolated groups of visible vectors can be joined by a blanked segment. Short vector, and particularly incremental, codes are often grouped for coding efficiency, the vectors within a group sharing a common link bit and a common blanking bit.

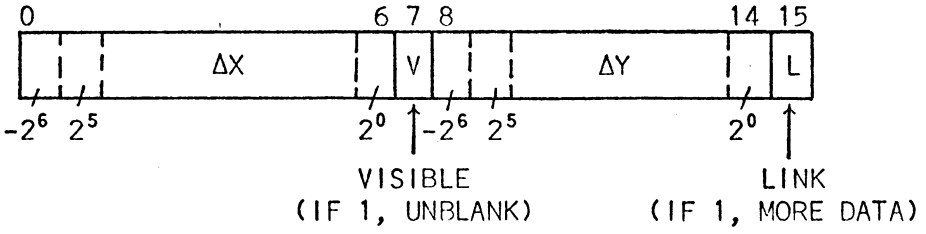
Following, is a description of the range of vector types and corresponding vector formats used in Intergraphic, details of the chosen vector generation algorithm, and a comparison between the chosen and an alternative algorithm. The section concludes with a discussion of two alternative vector generation techniques, viz., binary rate multiplier (BRM) techniques and analogue generation techniques, including some comments on curve generators.

4.2 VECTOR TYPES AND FORMATS USED IN INTERGRAPHIC

The interface can interpret and plot four basic types and six special vector types. The basic vectors are small Cartesian, small polar, large Cartesian and large polar. The special vectors have been included to facilitate the coding and plotting of frequently occurring graphics, e.g., block diagrams, regular grids, circular arcs and diagrams comprising only horizontal and vertical lines.

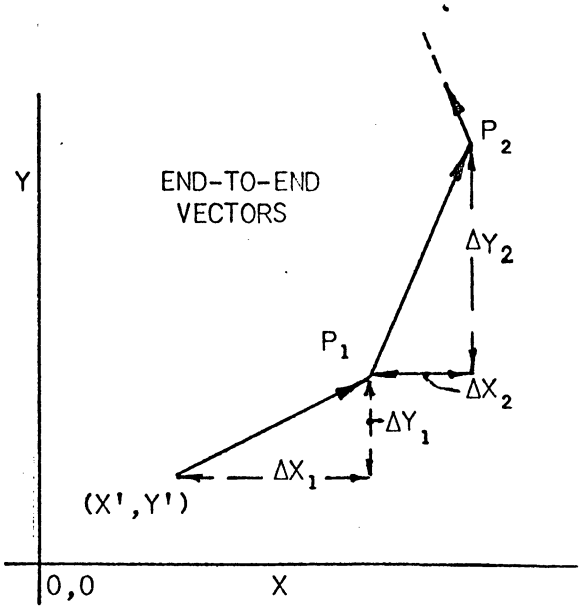
4.2.1 Basic Vector Types

1. *Small Cartesian vectors* are coded one to a 16-bit word, and each vector has an unblanking bit and a link bit. Format 4.1 shows the core-store representation of a small Cartesian vector and states the number representation and range of the components ΔX, ΔY. Figs. 4.1, a and b show a list of small Cartesian vectors plotted end-to-end and reset to X', Y', respectively. Small Cartesian vectors are coded



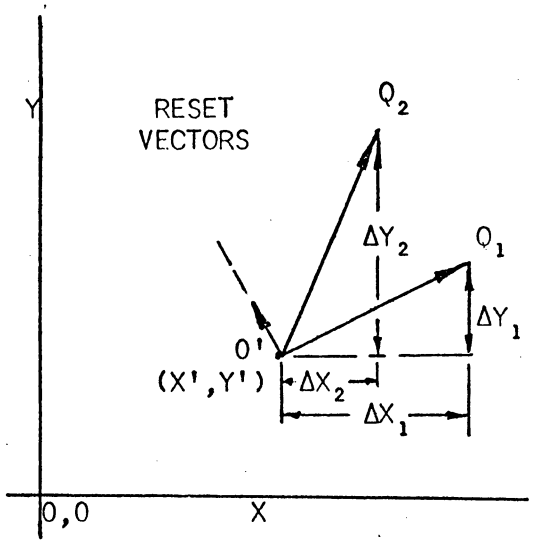
ΔX, ΔY: 7-BIT 2's COMPLEMENT INTEGERS,
RANGE, -64 (1) 63 DISPLAY INCREMENTS.

FORMAT 4.1 CORE REPRESENTATION OF A SMALL CARTESIAN VECTOR.



GRAFN 0; SML CTN VTRS (ETE).

(a)



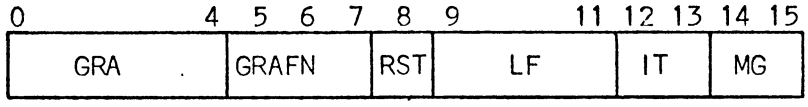
GRAFN 0; SML CTN VTRS (RST).

(b)

FIG. 4.1. SMALL CARTESIAN VECTORS:
(a) End-to-End, and (b) Reset to X', Y'.

in blocks of words under a header word. The header specifies that small Cartesian vectors follow, and also, whether end-to-end or reset plotting is required, the line-form (i.e., continuous or broken lines of various on-off patterns, cf. Sect. 4.3), the intensity level (inapplicable for binary-video storage) and the magnification (X1,X2, X4 or X8).

Graphical functions within Intergraphic are grouped under two machine-code functions, GRAPHICS-A and GRAPHICS-B, designated GRA and GRB respectively. GRA and GRB are further subdivided: GRA has eight sub-functions, designated GRAFN's, and GRB has sixteen sub-functions designated GRBFN's. The four basic vector-types are interpreted by GRAFN's and the six special vector types by GRBFN's. Format 4.2 shows the general GRAPHICS-A header: it precedes a block of small Cartesian vectors if the 3-bit GRAFN field has the value 0. Micro-routine GRA SML CTN VTRS (p A3.7), called by GRAFN = 0, plots a list of small Cartesian vectors.



GRA; MACHINE-CODE FN GRAPHICS-A.

GRAFN; SPECIFIES 1 OF 8 FUNCTIONS WITHIN GRAPHICS-A

RST; IF 1, AFTER PLOTTING EACH VECTOR OR POINT, RESET X,Y TO THE INITIAL VALUES X',Y', ELSE, ADD COMPONENTS END-TO-END (ETE).

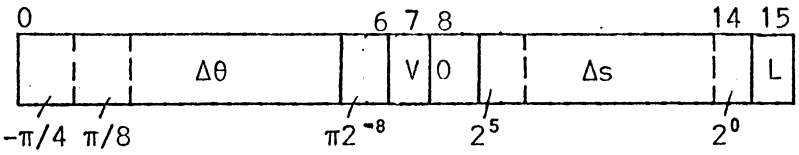
LF; LINE-FORM (APPLICABLE TO VECTORS, FIG. 4.9).

IT; INTENSITY FIELD, 4 LEVELS OF INTENSITY, ALL OF WHICH ARE VISIBLE.

MG; MAGNIFICATION X1, X2, X4, X8 (APPLICABLE TO SMALL CARTESIAN VECTOR/POINT COMPONENTS ONLY.)

FORMAT 4.2. GRAPHICS-A HEADER

2. *Small polar vectors* are also coded one to a 16-bit word, and each vector has an unblanking (or visible) bit V and a link bit L. Format 4.3 shows the core-store representation of small polar vectors and states the representation and range of the components Δθ, Δs. Small polar vectors are also coded in blocks preceded by a GRA header, but the GRAFN field corresponding to small polar vectors has the value 2. Figs. 4.2, a and b show a list of small polar vectors plotted end-to-end and reset, respectively.



Δθ; 7-BIT 2's COMPLEMENT ANGLE (FIG. 4.2)

RANGE; - π/4 (π/256) (π/4 - π/256) RADIANS.

Δs; 7-BIT POSITIVE INTEGER, BIT 8 = 0 (FIG. 4.2)

RANGE; 0 (1) 63 DISPLAY INCREMENTS.

FORMAT 4.3. CORE REPRESENTATION OF THE INCREMENTAL ANGLE AND LENGTH OF A SMALL POLAR VECTOR

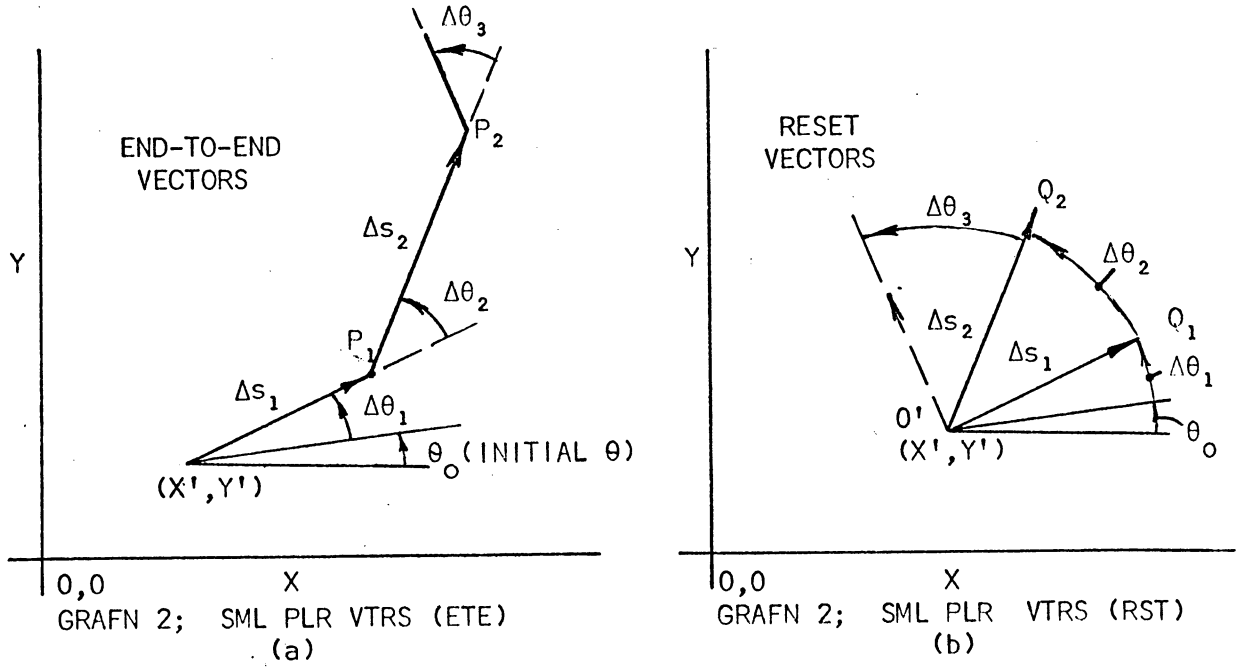


FIG. 4.2. SMALL POLAR VECTORS;
(a) End-To-End, and (b) Reset to X',Y'.

The incremental angular specification $\Delta\theta$ (i.e., the angular difference from the preceding vector, or for the first vector the angular difference from θ_o the initial value of θ) has three advantages over absolute angular specification: it allows graphics to be rotated by modifying θ_o only, it allows circular arcs to be compactly encoded by repeating a $\Delta\theta$, Δs vector in end-to-end mode, and it allows greater angular resolution to be encoded in a given field length because a $\Delta\theta$ specification need not cover 2π radians.

The range of Δs is identical to the range of positive ΔX or ΔY, and the range of $\Delta\theta$ has been selected as

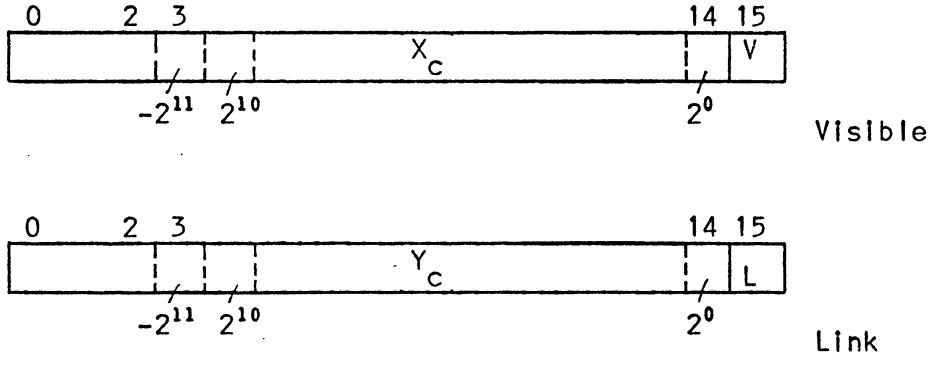
$$-\frac{\pi}{4} \leqslant \Delta\theta < \frac{\pi}{4} \text{ radians,}$$

which, encoded in 7 bits, gives an angular resolution of $\pi/256$ radians. Two reasons for this selection are that the tangential displacement of a polar vector of maximum length, $\Delta s = 64$, rotated by $\pi/256$ is approximately one display increment, and that one revolution comprises an integral number (512) of least significant $\Delta\theta$ increments.

Line-form and intensity are as described under small Cartesian vectors. Magnification, however, is not extended to small polar vector length Δs, because errors in the least significant bits of the sine and cosine routines, which are called before polar vector plotting, would also be magnified. GRA SML PLR VTRS (p A3.10) plots a list of small polar vectors.

3. *Large Cartesian vectors* are coded in two 16-bit words, the first containing the horizontal component X_c and the unblanking (or visible) bit V, the second containing the vertical component Y_c and the link bit, L. Format 4.4 shows the core-store representation and states the number representation and range of the components X_c, Y_c. Although the X_c and Y_c fields could each accommodate 15 bits, the fields have been restricted to 12 bits to be compatible with polar-to-Cartesian conversion (cf. CQ LRG PLR → CTN E,A, p A3.45) and the

special large-vector formats to be defined in this Sect. 4.2.2. The 12-bit fields can accommodate a 4096 x 4096 display, or alternatively, allow temporary display overflow for displays of lower resolution.



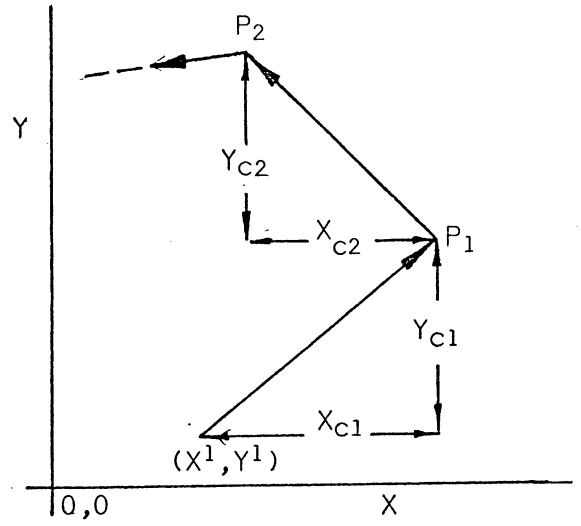
X_c, Y_c ; 12-BIT 2'S COMPLEMENT INTEGERS

RANGE ; - 2048(1)2047 DISPLAY INCREMENTS

BITS 0,1,2 = BIT 3 (I.E., FIELDS 0-2 EXTEND SIGN)

FORMAT 4.4. CORE REPRESENTATION OF THE COMPONENTS OF A LARGE CARTESIAN VECTOR

Fig. 4.3 shows a list of large Cartesian vectors plotted end-to-end; the reset option is also available, but is not shown.

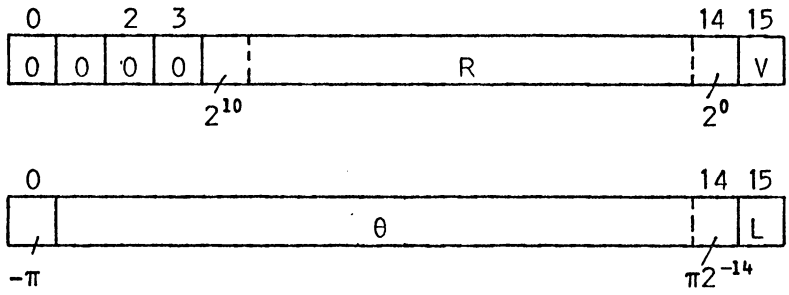


GRAFN 4; LRG CTN VTRS (ETE)

FIG. 4.3. LARGE CARTESIAN VECTORS, END-TO-END

Large Cartesian vectors are coded in blocks under a GRS header with a GRAFN field value of 4. Line-form and intensity apply, but magnificiation is not extended to large vectors. GRA LRG CTN VTRS (p A3.12) plots a list of large Cartesian vectors.

4. *Large polar vectors* are also coded in two core-store words. Format 4.5 shows the R,θ core format and ranges, and Fig. 4.4 shows a list of large polar vectors plotted end-to-end; the reset option is also available. Large polar vectors are distinguished by GRAFN = 6, and line-form and intensity options are applicable.



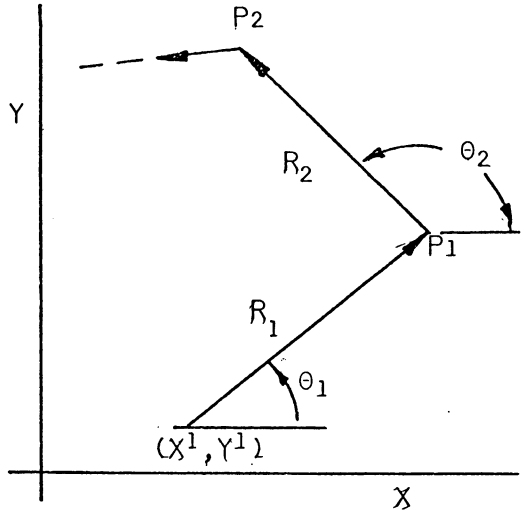
R; 12 BIT POSITIVE INTEGER

RANGE; 0(1)2047 DISPLAY INCREMENTS

θ; 15 BIT 2's COMPLEMENT ANGLE

RANGE; $-\pi(\pi 2^{-14})\ (\pi-\pi 2^{-14})$ RADIANS

FORMAT 4.5 CORE REPRESENTATION OF THE LENGTH AND ANGLE OF A LARGE POLAR VECTOR

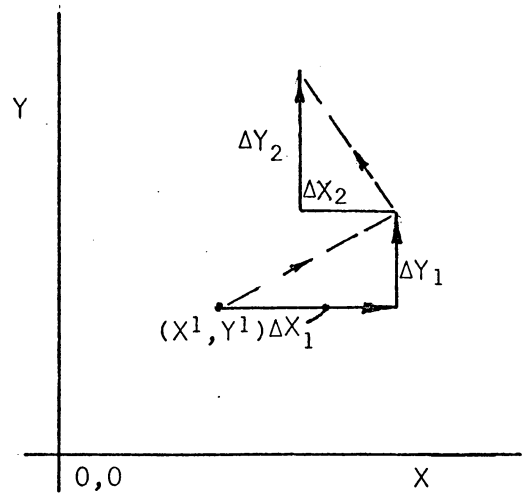


GRAFN 6; LRG PLR VTRS (ETE)

FIG. 4.4. LARGE POLAR VECTORS, END-TO-END

4.2.2 Special Vector Types

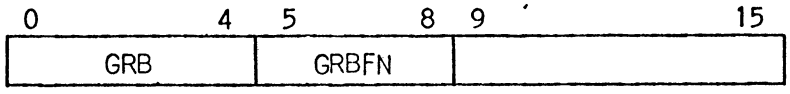
1. *Small "X then Y" vectors* have the same format as small Cartesian vectors (Format 4.1.) but, instead of being the hypotenuse of the right-angled triangle with sides ΔX, ΔY, this "vector" is the sequence ΔX, 0 then 0,ΔY, the terminus being identical to that of the conventional vector. Fig. 4.5 shows two small "X then Y" vectors. There is no reset option.



GRBFN 0; SML XTHENY VTRS

FIG. 4.5. SMALL "X THEN Y" VECTORS

Line-form, intensity and magnification options are as defined under small Cartesian vectors. Small "X then Y" vectors are coded in a block which is headed by a GRB with GRBFN = 0. Format 4.6 shows the general GRB header.



GRB; MACHINE-CODE FN GRAPHICS-B,

GRBFN; SPECIFIES 1 OF 16 FUNCTIONS WITHIN

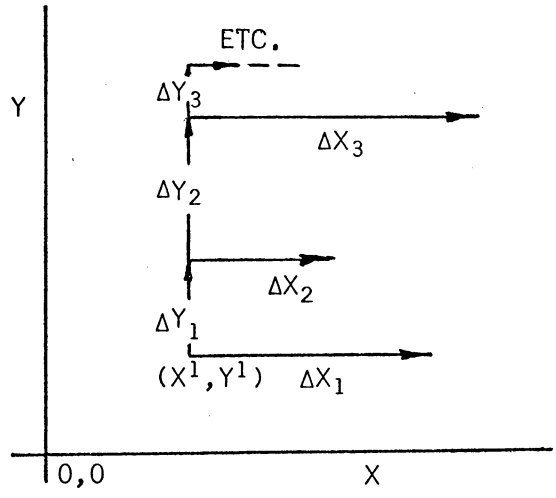
FIELD 9-15; FOR GRBFN's 0-7,AS GRAPHICS-A, OTHERWISE FIELD 9-15 IS GRBFN DEPENDENT.

FORMAT 4.6 . GRAPHICS-B HEADER

Small "X then Y" vectors, appropriately magnified, enable block diagrams to be coded efficiently (blanked "X then Y" vectors allow individual blocks of a diagram to be separated). Stepped waveforms are also efficiently encoded by small "X then Y" vectors. The vectors are plotted by GRB SML XTHENY VTRS (p A3.16).

There is no facility for plotting large "X then Y" vectors, partly because most block diagrams do not require block dimensions to be specified to more than seven significant bits (i.e., magnified small vectors are adequate), but mainly because a pair of large "X or Y" vectors, to be defined shortly, can achieve the same result.

2. *Small "X-reset, increment Y" vectors* are shown in Fig. 4.6. Each "vector" comprises a horizontal vector $\Delta X, 0$ which is reset to X' and a vertical displacement ΔY. The vertical displacement is blanked.



GRBFN 1; SML X RST VTRS IY

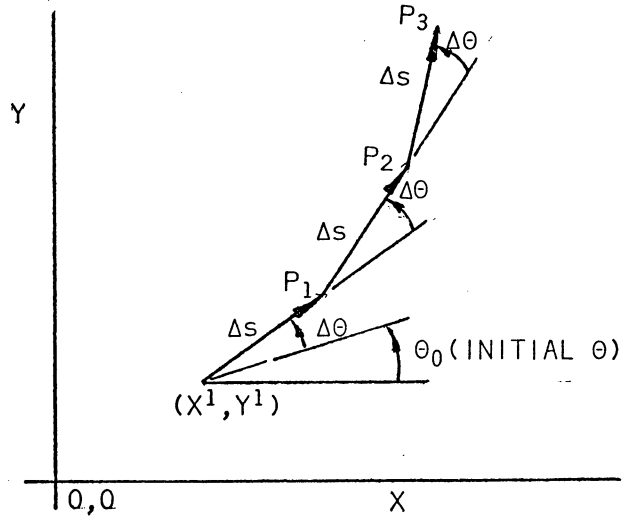
FIG. 4.6. SMALL "X-RESET, INCREMENT Y" VECTORS

Format 4.1 codes the $\Delta X, \Delta Y$ components and V,L, and "vectors" are contained in blocks under a GRB header with GRBFN = 1. Line-form, intensity and magnification options are as defined under small Cartesian vectors. This plotting mode is useful for plotting the horizontal lines of tables: the lines, of arbitrary length and separation, may be aligned on the left, as shown, or on the right ($\Delta X < 0$). GRB SML XRST IY VTRS (p A3.17) plots a list of "vectors" in this mode.

The large version is not provided: as for small "X then Y" vectors, it is less useful and can be composed of two large "X or Y" vectors which also require two core-store words.

3. *Small "Y-reset, increment X" vectors*, interpreted by GRB SML YRST IX VTRS (p A3.18), are the orthogonal versions of small "X-reset, increment Y" vectors. The GRBFN code is 2.

4. *Small repeated polar vectors* are illustrated, in part, by Fig. 4.7 which shows a single "small repeated polar vector". The $\Delta\theta, \Delta s$ elements are drawn end-to-end and form part of a regular polygon or, if $\Delta\theta$ and Δs are sufficiently small, approximate a circular arc. Angle θ, initially θ_o, is incremented by $\Delta\theta$ to form the orientation of the first vector, the line segment Δs is then drawn, and so on.



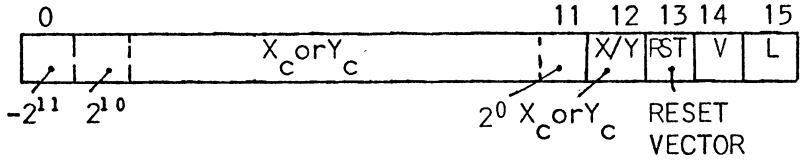
GRBFN 3; SML RPT PLR VTRS (only one list element shown)

FIG. 4.7. A SMALL REPEATED POLAR VECTOR

Two core words are used to code a small repeated polar vector: the first is identical to the small polar vector format (Format 4.3); the second is a positive integer which specifies the number of times the $\Delta\theta$, Δs element is plotted. A list of such word-pairs, then, is a *list* of part polygons or circular arcs. GRB SML RPT PLR VTRS (p A3.19) plots a list of independent, small repeated polar vectors. The list is headed by a GRB header with GRBFN = 3.

This plotting mode, for small $\Delta\theta$, Δs, plots a set of linked circular arcs of different curvatures and total swept angles: it is useful for plotting arbitrary continuous curves which, through the parameter θ_o, may have any orientation. Line-form and intensity apply to small repeated polar vectors, but, as for small polar vectors, magnification is inapplicable. Also, there is no reset option.

5. *Large "X or Y" vectors*, each coded in a single 16-bit word, are restricted to be either horizontal or vertical. Format 4.7 shows the core store representation; it comprises, the 12-bit field "X_c or Y_c" (which specifies either the vector X_c, 0 or the vector 0, Y_c) and the four independent bits (12-15) which specify, horizontal or vertical, end-to-end or reset, blanked or visible, and end or linked, respectively.

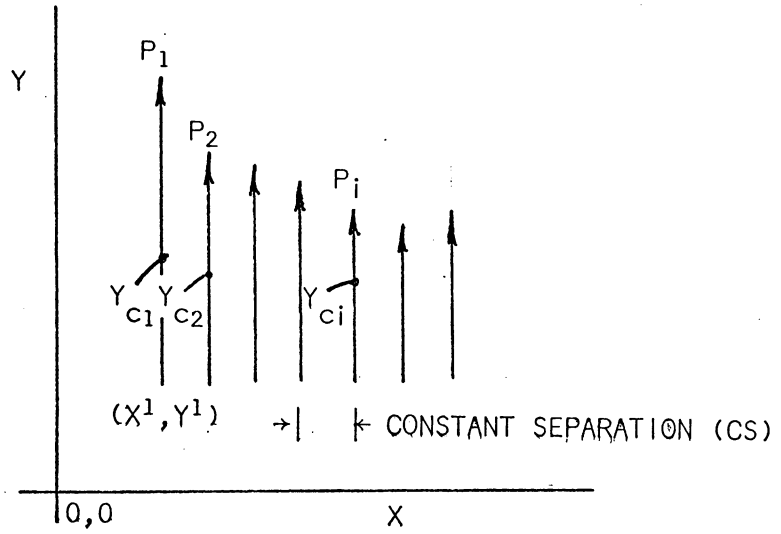


X_c or Y_c; 12-BIT 2's COMPLEMENT INTEGER,
RANGE; -2048(1)2047 DISPLAY INCREMENTS,
BIT 12 = 0 FOR X_c, 1 FOR Y_c; BIT 13 = 1 FOR RESET.

FORMAT 4.7. CORE REPRESENTATION OF LARGE "X OR Y" VECTORS.

Large "X or Y" vectors constitute many graphics and, for a diagram consisting entirely of horizontal and vertical lines, only one half of the storage which would be required for conventional large Cartesian vectors is necessary. The reset option is specified individually for each vector, not for the entire list as in the GRAFN vectors, i.e., the current X,Y is temporarily stored in X',Y' before each vector is plotted and, if reset is nominated, X,Y is restored from X',Y' before the next vector is interpreted. Large "X or Y" vectors are coded under a GRB header with GRBFN = 4. GRB LRG X/Y VTRS (p A3.20) plots a list of these vectors. Line-form and intensity are applicable, but, as for other large vectors, magnification does not apply.

6. *Large "X or Y, constant-separation" vectors* are coded in the large "X or Y" vector format (Format 4.7), but, in addition to plotting the X or Y vector specified, the CRT beam is displaced by a constant amount (the "constant separation", CS) at right angles to the vector (viz., if an X vector is specified, then CS refers to a Y displacement; the displacement has the sign and magnitude of CS, i.e., it is independent of the sign of X_c). Fig. 4.8 shows a list of large Y vectors (reset) which are separated by a constant X displacement of CS. Although each vector is independently labelled as horizontal or vertical and reset or not, these options would normally be constant throughout the list.



GRBFN 6; GRB LRG X/Y CS VTRS

FIG. 4.8. LARGE Y, CONSTANT-SEPARATION VECTORS (RESET OPTION)

Applications include bar-graphs and horizontal or vertical shading lines. For shading, two lists, e.g., one having positive Y_c values, the other having negative Y_c values, are required to shade some shapes (e.g., a circle); more lists are required for more complex shapes. Complex shapes, however, are often more conveniently shaded or silhouetted by plotting a series of contours derived from the boundary shape, e.g., a set of concentric circles.

The constant separation, CS, is specified by a 2's complement integer word which is inserted between the GRB header (GRBFN = 6) and the first vector word of the list. GRB LRG X/Y CS VTRS (p A3.22) plots a list of large X or Y vectors at the constant separation requested. Line-form and intensity are applicable, but magnification is not.

4.3 THE CHOSEN VECTOR GENERATION ALGORITHM

This section is descriptive only. The chosen scheme is justified in the next two sections by comparison with an alternative algorithm and with two alternative techniques.

The simplest case of vector plotting, viz., a Cartesian vector with components which are positive byte integers, will be described. The following should be read in conjunction with the definition of APV (Sect. 3.2, *ASO*) and the microroutine MI PLT SML CTN VTR D (p A3.29).

Consider the repeated micro-order,

APV, JNZR, RP, B + D → B,

and assume that B is initially cleared, D_U (upper byte of D) contains the X component of a vector, and D_L contains the Y component. Further, assume that both vector components are positive integers, and that, initially, R contains 128. Since APV decrements R (as ADR), and JNZR, RP is nominated, the arithmetic process is performed 128 times. During these 128 cycles, overflows of $_0$ and of $_8$ will be generated at

rates proportional to D_U and D_L respectively, i.e., X will be incremented at a rate proportional to the X component, and Y will be incremented at a rate proportional to the Y component. Moreover, during these cycles there will be exactly D_U overflows, i.e., X increments, from the upper AU and exactly D_L overflows, i.e., Y increments, from the lower AU. This is derived as follows:

In APV mode with positive D_U, the sign bit B_0 is forced to remain 0 (cf. definition of APV, Sect. 3.2 *APV*, vii); i.e., the expression, $of_0 = \overline{a}'_0\ \overline{b}'_0\ c_0$ (cf. Sect. 2.1, *5*), reduces to, $of_0 = c_0$. But, for APV, if $of_0 = 1$, X is incremented; thus c_0, the carry into the most significant stage of the upper AU, increments X (rather than change B_0). In APV mode, then, the X register extends B_{1-7} to form a single, long accumulator, X_{0-15}, B_{1-7}. The result of 128 repeated additions of D_U into X_{0-15}, B_{1-7} is identical to left-shifting D_U seven places and adding, i.e., is identical to adding D_U, the horizontal vector component, to X. Similarly, Y extends B_{9-15}, and after 128 cycles, Y is incremented by D_L. Thus, the mode produces the horizontal and vertical vector increments uniformly, in the correct ratio, and the pair X,Y is incremented precisely by the vector components. Moreover, after the 128 cycles, B returns to its initial value (in this example, zero).

Negative vector components are also accepted by the same repeated APV order, provided the initial values in the accumulators B_U, B_L are given the same signs as the corresponding components D_U, D_L. For example, the algorithm MI PLT SML CTN VTR D sets an initial value of all 1's into B_U if D_U is negative, and similarly for the lower AU (the description accompanying MI PLT SML CTN VTR D, p A3.29, explains the need for negative initial values).

In the plotting mode above, vector components must each be within the range of a 2's complement byte, i.e.,

$$-128 \leqslant \text{X or Y component} < 127 \text{ display increments.}$$

(For a 1024 x 1024 display, the extremes of this vector range are approximately 13% of the display square side.) Larger vectors are divided into

segments by microroutine MI PLT LRG CTN VTR (p A3.31), each segment being within the range of MI PLT SML CTN VTR D.

Polar vectors are also plotted by microcode: large polar vectors are first converted to Cartesian and then plotted by MI PLT LRG CTN VTR; small polar vectors use MI PLT SML CTN VTR D, where D_U = Cos θ, D_L = sin θ, but the number of cycles (i.e., the initial value of R) is Δs, the length of the vector, not 128.

Vector plotting is improved by prenormalizing vector components and proportionally adjusting the number of cycles (e.g., doubling the vector and halving the number of cycles); this gives a more nearly constant incremental plotting rate and, therefore, a plotting time almost in proportion to vector length. MI PLT SML CTN VTR D begins with a normalize sequence. The routine also accepts a magnification factor of X2, X4 or X8 by increasing the number of cycles proportionally to 256, 512 or 1024, respectively; this method of magnifying vectors preserves the fine structure of vectors (i.e., vector-increments remain single display-increments), but the end-points of magnified vectors coincide with a correspondingly magnified display grid.

A vector is normally plotted as a continuous line of uniform brightness. As an alternative, however, a line-form (LF) option has been provided which plots vectors as broken lines with various on-off intensity patterns along the length of the vector. The LF field of a GRA or GRB header (bits 9-11; Format 4.2, p 4.4 and Format 4.6, p 4.10) specifies the broken-line patterns according to the logical sum-of-products expression

$$\text{Unblank} = \overline{W}_{10}\, LF_0 + \overline{W}_{11}\, LF_1 + \overline{W}_{12}\, LF_2 + \overline{LF}_0\, \overline{LF}_1\, \overline{LF}_2$$

where W_{10-12} are three bits of W_L, a byte-length counter which integrates distance along a vector string, and LF_{0-2} are the bits of field LF. If one display increment is 0.01 in., then the weight of W_{15} is 0.01 in., i.e., the weights of W_{10}, W_{11} and W_{12} are 0.32, 0.16

and 0.08 in. respectively. As examples, LF code 000 specifies an unbroken line; LF code 001 results in the expression

$$\text{Unblank} = \overline{W}_{12},$$

i.e., a broken line of alternate visible and blanked sections each 0.08 in. (Fig. 4.9); and LF code 101 specifies the periodic centre-line pattern - long visible 0.40 in., gap 0.08 in., short visible 0.08 in., and gap 0.08 in.

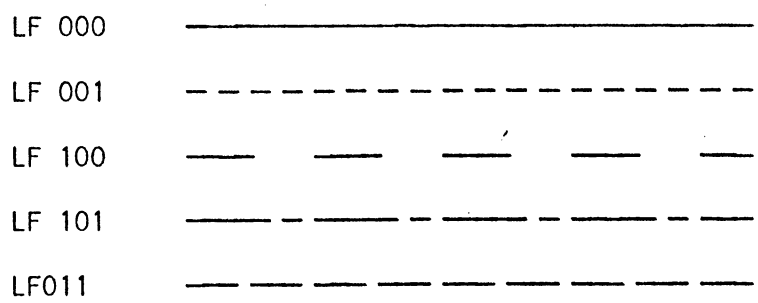


FIG. 4.9. VARIOUS LINE-FORM PATTERNS

In small polar vector mode, distance along the string is recorded simply by incrementing W_L once in each APV cycle (as the average increment/decrement rates of X and Y are Cos θ and Sin θ display increments per cycle respectively, the average beam velocity along the vector is precisely one display increment per cycle). In other vector plotting modes, the beam velocity is not constant and, as an approximation, W is incremented whenever X or Y is incremented/decremented, and doubly incremented when both X and Y are incremented/decremented simultaneously. This increments W at the correct rate for horizontal or vertical vectors, but increments W at $\sqrt{2}$ times the correct rate for vectors of unit slope. Thus, the scale of an on-off pattern varies with slope, but the pattern type is invariant.

4.4 AN ALTERNATIVE VECTOR GENERATION ALGORITHM

Let x,y be a vector and, further, suppose x and y are positive integers and x > y. If IX denotes a horizontal display increment, IY a vertical display increment, and IXY a combined IX,IY increment, then the display sequence for vector x,y would be,

either, blocks of IXY elements separated by single IX elements, the length of the blocks being "uniform" (meaning that the block lengths are not necessarily constant, but, except for the first and last blocks, are within one element in length), e.g., an acceptable sequence for the vector 13,8 is,

IXY, IXY, IX, IXY, IXY, IX, IXY, IX, IXY, IXY, IX, IXY, IX,

or, "uniform" blocks of IX elements separated by single IXY elements, e.g., an acceptable sequence for the vector 15,4 is,

IXY, IX, IX, IXY, IX, IX, IX, IXY, IX, IX, IX, IXY, IX, IX, IX.

These strings can be generated by the following algorithm,,

(i) each step produces a horizontal increment, i.e., produces either an IX element or an IXY element,

(ii) there are exactly x steps, counted, for example, by decrementing at each step a register which initially contains x, until it is cleared,

(iii) in each step, y is subtracted from an accumulator (initially cleared) and, if the result is negative, a Y increment is produced (which, in combination with the X increment of (i), produces an IXY element) and the accumulator is restored to a positive value by the addition of x. This is a balancing process which attempts to null the accumulator; it inserts Y-increments in the stream at a "uniform" rate, but maintains a ratio of Y to X increments which approaches y/x.

The sample strings above were produced by this algorithm. After x steps, the accumulator is diminished by x subtractions of y, but, to maintain the balance, the accumulator must also be augmented by y additions of x; i.e., y vertical increments are produced, and the accumulator is cleared in the terminating step.

Some variations in the actual insertion points of the fixed number (y) of vertical increments within the stream are possible by choosing different initial values for the nulling accumulator, but these variations in fine structure are normally of no consequence in display applications.

The algorithm can be extended to accept negative vector components and the reverse inequality, $|y| > |x|$. For components which are within the range of 2's complement byte integers, the algorithm requires four byte-registers: two to hold x,y, the third for the counter and the fourth for the accumulator. This is one byte-register less than the chosen algorithm (which uses B_U, B_L, D_U, D_L, R_L). Also, only one 8-bit AU is necessary, not two. As in the chosen scheme, some hardware modifications would be necessary to automatically route increments/decrements to the X,Y registers (the paths would be conditioned by the signs and relative magnitudes of x and y). However, the process cannot be reduced to the repetition of a single micro-order, as the chosen algorithm is, unless the operand subtracted from the accumulator is conditional upon the sign of the accumulator, i.e., if the accumulator is positive, y is subtracted (assuming $x > 0$, $y > 0$, $x > y$), whereas if the accumulator is negative, $y-x$ is subtracted (i.e., the accumulator is incremented by $x-y$). Conditional subtraction requires another byte-register to hold $y-x$.

Magnification by 2, 4 or 8 is simply achieved by increasing the initial value in the counter from x to $2x$, $4x$ or $8x$, respectively. A disadvantage of the algorithm is that small polar vectors $\Delta s \cos \theta$, $\Delta s \sin \theta$ cannot be plotted without first forming $\Delta s \cos \theta$ to control the number of steps. (This premultiplication is not necessary for the chosen algorithm - the number of cycles simply becomes Δs, not 128.)

Although the alternative algorithm is practicable, the need for initially testing for relative magnitude, the need for conditional subtraction, and the need for premultiplication when plotting small polar vectors led to the adoption of the chosen scheme.

The development and comparison of algorithms, especially when logical design modifications are involved, is difficult, and the above comparison does not prove that the chosen scheme is the best solution attainable. However, the chosen scheme does show that efficient vector plotting is possible within a conventional microprogrammed structure provided certain logical modifications are introduced. Microroutine MI PLT SML CTN VTR D, which normalizes and plots a vector, has only 11 micro-orders.

4.5 BINARY RATE MULTIPLIER TECHNIQUES FOR VECTOR GENERATION

Fig. 4.10 shows a binary rate multiplier (BRM). It produces an output pulse train which has an average pulse repetition frequency proportional to the contents of a multiplier register, M.

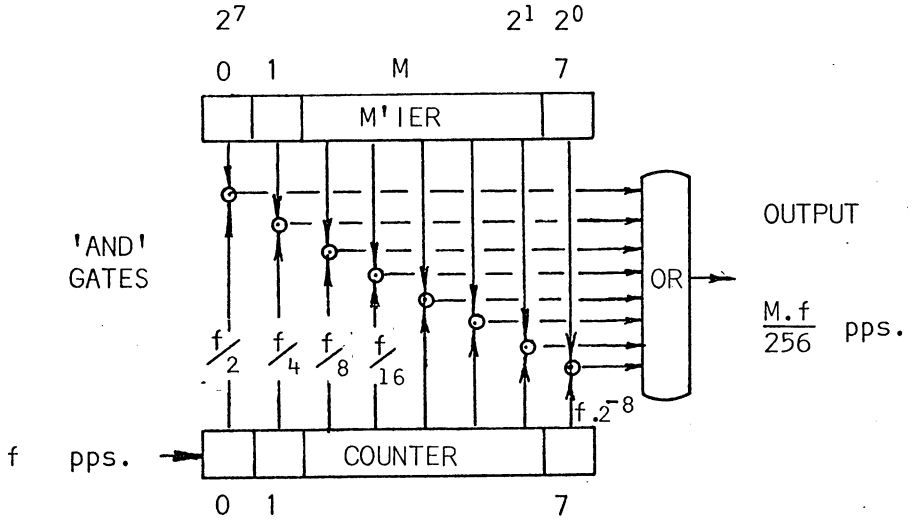


FIG. 4.10. A BINARY RATE MULTIPLIER

The counter is a conventional binary counter, incremented at an input pulse rate of f pulses/sec. (pps). Apart from the internal organization of the counter, each stage of the counter generates an output pulse whenever it undergoes a $0 \rightarrow 1$ transition. These outputs will have rates of

$$f/2,\ f/4 \ldots f/256 \text{ pps},$$

and moreover, the pulses from any two stages never coincide, provided the counter propagation time does not approach $1/f$ sec. (Multiple coincidence of $1 \rightarrow 0$ transitions is frequent and $1 \rightarrow 0$ transitions usually accompany each $0 \rightarrow 1$ transition, but $0 \rightarrow 1$ transitions are always singular.)

A set of "AND" gates selects those pulse trains for which the corresponding multiplier bit is 1, and all selected pulse trains are collected by the "OR" gate (i.e., if multiplier bit $M_o=1$, then the $f/2$ pps train is gated into the "OR" gate and contributes to the output). Thus, since no two pulses coincide, the rate of the output pulse train will be

$$M_0 f/2 + M_1 f/4 + \quad + M_7 f/256 \text{ pps}$$

i.e.
$$(2^7 M_0 + 2^6 M_1 + \quad + M_7) f/256 \text{ pps}$$

i.e.
$$Mf/256$$

where M is the positive integer multiplier. Although the pulses in any one contributing stream are uniformly spaced, the superposition of several streams will not produce a precisely uniform pulse rate and, therefore, $Mf/256$ pps must be interpreted as an average pulse rate.

A vector generator can be produced from two multiplier registers (containing x,y), two sets of "AND" gates, two "OR" gates and one counter which drives both sets of "AND" gates. If 256 pulses are applied to the counter, there will be exactly x pulses collected by one

"OR" gate and exactly y pulses collected by the other. These trains increment the X,Y display registers and so plot the vector x,y. (Although the output trains are not precisely uniform, the fine structure of the trains is not noticeable in vector plotting.)

The BRM uses pulse-rate oriented techniques which are related to DDA (digital differential analyser) techniques. The BRM pair above produces the precise number of display increments, it can accommodate binary magnification by applying 512, 1024,... pulses, and it can also plot small polar vectors from cos θ, sin θ provided Δs, not 128, input pulses are applied.

Moreover, by substituting counters for the x,y registers and cross-coupling the "OR" gate outputs to increment these counters such that,

$$dx = -ky, \qquad dy = kx \qquad (k = \text{constant}),$$

the BRM pair could update sin θ, cos θ(held in the multiplier counters) from increments in θ (expressed as input pulses). Further application of a BRM pair are circle plotting (the streams in the cross-coupled sin θ, cos θ connection above also increment the X,Y display registers) and the generation of right parabolas, hyperbolas and exponentials. An array of BRM's has also been used for co-ordinate transformations (53). Sutherland (54, 1963) proposed DDA techniques for displaying the general conic and estimated a point plotting rate of 1 MHz using 20 MHz serial logic.

The vector generation scheme adopted in Intergraphic resembles the BRM techniques described above (overflow/underflow rates are proportional to the contents of registers), in fact, the scheme stemmed from DDA techniques. In the author's original proposal (26) the upper and lower AU's were designated the "cosine AU" and "sine AU" respectively and the cross-coupled mode, designated the "circular mode", was proposed for the generation of incremental polar vector components. As the design progressed, however, several disadvantages of incremental techniques

became apparent;

(i) "registers" must be able to be incremented/decremented, and for rapid plotting rates, counting logic with short propagation times is necessary (i.e., simple transition generated carry propagation is inadequate);

(ii) to plot parts of circles which have radii of curvature several times larger than the display size requires registers of 12 bits or more;

(iii) for the cross-coupled connection above, the plotting time for a circle is constant regardless of curvature, so that additional rate multipliers or alternative increment insertion points must be provided to maintain more nearly constant plotting rates. (Ideally, the increment rates should match the bandwidth of the display medium, i.e., should be constant.);

(iv) mode changing requires various pulse paths to be switched, e.g., for simple vector plotting, the cross-coupled pulses must be inhibited;

(v) although a wide variety of curves can be generated (especially if four rate generators are used), considerable computer time and software complexity may be needed in the selection of curve types and parameters to approximate segments which synthesize a curve within the DDA's repertoire. Moreover, formats must be designed to transmit the parameters from the CPU, and interface-logic or programs must be provided to, interpret these formats, set initial values, and control the number of cycles.

Thus, a DDA oriented interface using bitwise parallel logic (in contrast to bitwise serial logic) in order to achieve a nominal plotting rate of 10 MHz, would be compatible in size and complexity to the general-purpose processing section of Intergraphic. But the DDA struct-

ure is specialized and therefore not suited to the variety of supervisory and conventional digital computing tasks which must also be performed within a graphical interface. Indeed, a DDA structure may even need to be modified to ensure efficient symbol plotting. Possibly, a hybrid structure with some DDA logic and some conventional digital logic would be the most suitable form for a graphical interface, but the hybrid structure would necessarily involve a wider range of engineering techniques. The microprogrammed interface chosen can generate vectors, strings of circular arcs and high quality symbols at 10 MHz increment rates using compact microcodes, and being a general-purpose computer, is also well suited to the more conventional supervisory and control tasks.

4.6 ANALOGUE GENERATION TECHNIQUES

Johnson (55, 1965) has described an analogue generator for the high speed display of rotated cubics and conics. The generator used conventional analogue-computer concepts, but, by using high-current field-effect transistors for path switching and operational amplifiers with 100 MHz gain-bandwidth products, it achieved a plotting rate equivalent to an incremental rate of 1 MHz. One of the problems introduced when linking curve segments produced by different hardware configurations is that the potential across the capacitors of integrators, and therefore their stored energy, must be changed rapidly.

Roberts (56, 1966) and Blatt (57, 1967) have developed analogue display techniques for the general conic in terms of basic parabolic equations. Wideband multiplying decoders (D/A converters with variable reference voltages) were used. Fourteen decoders were required, three of which were conventional D/A converters and two had digital registers which varied with time, but the remaining nine had only analogue varying inputs during the generation of any one curve segment. Curve plotting rates were also equivalent to an incremental rate of 1 MHz. The scheme can be extended to general 2-D cubic curves (18 multiplying decoders) and to 3-D cubics rotatable in 3-D (34 decoders).

An advantage of this scheme is that it contains no energy storage elements (neglecting stray elements) and the constant configuration eliminated the need for path switching; i.e., it is potentially a very high speed generation device.

Dertouzos and Graham (58, 1966) have described a third analogue technique which segments an arbitrary curve by a set of breakpoints (command points) and approximates the given curve with selected trajectories which link adjacent breakpoints. Trajectories are determined by the relative step responses of a pair of networks which are fed with the D/A converted versions of the breakpoint co-ordinates (e.g., matched networks give a piecewise linear interpolation). The network responses are controlled from segment to segment by switching physical parameters. Intensity control is necessary to compensate for variable plotting velocity. The authors considered both polynomial and simpler exponential trajectories, but stressed that while the more complex realisation yielded a larger class of curve segments, it had two disadvantages; the computing time necessary to select an optimum set of segments to match a given curve was far greater and the implementation was much more complicated and expensive. The technique does allow compact coding (24 bits per segment for the exponential network adopted) and the co-ordinates of each breakpoint are maintained digitally.

The diversity of approaches and techniques referred to above indicate that the field of analogue curve generation within computer graphics is still exploratory. No doubt, for certain applications special hardware for spatial rotations of 3-D objects is justified. However, since there will always be curve types which are outside of the family of curves provided, synthesis techniques will always be necessary. It could be possible that an algorithm specialized for piecewise linear and circular segmentation is simpler and faster than an algorithm for segmentation into conics or cubics. This is similar to the compromise between network complexity and parameter determination stated by Dertouzos and Graham. More experience is required in this area and, as in other comparisons, the outcome will clearly depend upon the range of applications. The approach with Intergraphic has been to limit the

variety of directly interpretable graphics to straight lines and circular arcs, to generate these very rapidly and precisely within the standard digital framework, and to rely on CPU software to partition arbitrary curves (including parabolas, hyperbolas and exponentials) into piecewise linear or circular segments. (The breakpoint insertion algorithm for input graphics (Sect.6.2.2)can be used to form a piecewise linear approximation to a curve which is generated in the CPU by incremental techniques.) This approach has eliminated the need for incorporating BRM's, multiplying decoders, signal switching circuits, analogue networks or operational amplifiers. The plotting rates (100μ sec. for full display deflection in incremental mode) are sufficient to plot 100 new frames/sec. Moreover, using the same codes and logical structure, speeds an order of magnitude greater (100 MHz increment rates) are feasible. This is so because subnanosecond gates are available, the ROM and the D/A converters are nonsaturating techniques which can be extended to high frequencies, and very high frequency CRT electrostatic deflection structures already exist.

CHAPTER 5

SYMBOL GENERATION

5.1 GENERAL CONSIDERATIONS

Loewe, Sisson and Horowitz (59, 1961) and Groll (60, 1964) have reviewed a variety of techniques for the high-speed generation of alphanumeric symbols. They describe three main classes: CRT beam shaping tubes (e.g., the Charactron), raster scanned memories which store symbol dot patterns, and analogue symbol-stroke waveform generators. Also, within these classes, they outline a number of variants.

To eliminate the need for a separate symbol generator, some workers have displayed symbols as strings of vectors using an existing vector generator. The vector codes may be stored in core or read-only memory. However, codes designed primarily for vector plotting are often inefficient for coding intricate symbol details; e.g., long-vector codes accommodate vector components much larger than necessary. Storage can be reduced if symbols are built of standard vectors selected from a small, well-chosen set, because a code which nominates one of a small number of vectors is much shorter than a code which specifies arbitrary vector components. The complexity and cost of the intermediate translation of standard-vector selection codes to vector components for a vector generator, however, must be assessed.

The following considerations led to the symbol generation scheme chosen for Intergraphic.

1. If possible, it is preferable to use the main-frame logic and memory techniques of a machine for symbol generation rather than develop special techniques. Thus, digital techniques were preferred to analogue, and the use of the existing core or read-only memories was preferred to the addition of a separate memory which held only symbol details.

2. It is preferable to build symbols of linked strokes rather than to define symbols within a fixed symbol-matrix, because symbols composed of strokes can extend arbitrarily. (It is possible to assemble two or more "special-symbol" matrices to synthesize one symbol, e.g., an integral sign, but the assembly procedures are not simple and many "special-symbol" matrices may be necessary.)

3. Standard symbol-strokes reduce storage requirements, but must be carefully chosen so that symbols are clearly distinguishable and can be built from a small set of standard strokes.

4. Standard strokes need not be restricted to straight line segments: standard curves can also be encoded compactly, and can be approximated digitally.

5. The storage of symbol-stroke details must be efficient; a comparison with some existing dot matrix and incremental code storage requirements should be favourable.

6. A large number of symbols should be available, and it should be easy to define new symbols.

7. Symbol plotting should fit nicely into the existing control logic of the machine, i.e., special buffering, interface logic and timing should be minimised or, preferably, avoided.

8. Symbol plotting rates should be limited by the bandwidth of the display deflection system, not by the symbol generation hardware, because symbol plotting occurs frequently within a graphical interface.

These points have been satisfied by the chosen scheme which forms symbols from standard straight-lines, quarter circles and quarter ellipses.

5.2 SYMBOL GENERATION WITHIN INTERGRAPHIC

The standard strokes above are encoded in the microprogram read-only memory of Intergraphic, four strokes to a word: these words are distinguished from other micro-orders by the order-type field, G_{0-3}, and are interpreted by a small amount of symbol generation digital logic. There are two such micro-order types, SFS (stroke formats) and IFS (increment formats), the final two orders shown in Table 3.1, p 3.2.

Micro-Order SFS encodes the standard symbol-strokes shown in Fig. 5.1. The four curved strokes, (a) - (d), are quarter circles and ellipses approximated by elemental X,Y increment chains and the four straight strokes, shown at (e), are approximated by the four elemental chains P0 - P3. The display is unblanked for about 80ns after each elemental step, producing a series of overlapping visible dots as illustrated in (d). Although the origin of a stroke is not unblanked, that point is usually the end-point of another stroke so that continuity of dots is maintained. Variants of the strokes illustrated in Fig. 5.1 are also available in the 2nd, 3rd and 4th quadrants. Fig. 5.2 shows all quadrant variants of the small quarter-circle P0.

Detailed formats of the four identical stroke fields within an SFS micro-order are shown in Table 3.1. If v = "1" the stroke is visible, otherwise it is blanked; if r = "1" the stroke is curved, i.e., a quarter circle or ellipse, otherwise it is straight. Field p comprises two bits p_0,p_1 which code P0 - P3 (p_0 p_1 = 00, 01, 10, 11 respectively) and field q comprises two bits q_0, q_1 which code the four quadrants, Q1 - Q4 (q_0 q_1 = 00, 10, 11, 01 respectively, i.e., q_0 is the sign bit of δX, q_1 is the sign bit of δY). The seventh bit, "ℓ", is a link bit: if ℓ = "1" a further stroke follows, otherwise the stroke is the final stroke of the symbol.

Micro-Order IFS encodes smaller standard strokes. The strokes are shown in Fig. 5.3, and are half and quarter-length versions of the elemental chains which constitute the straight strokes of SFS:

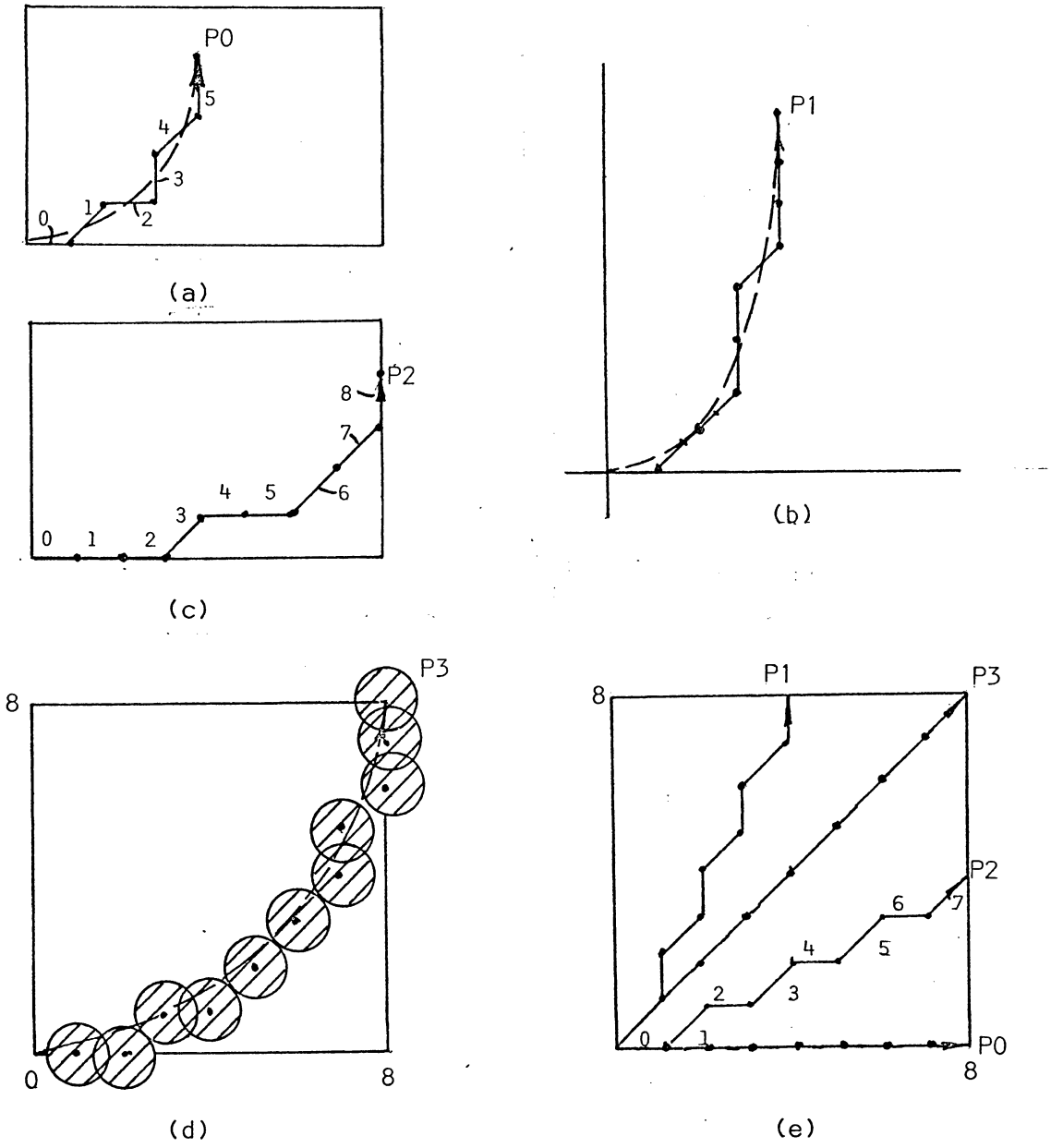


FIG. 5.1. SFS STROKES (FIRST QUADRANT)

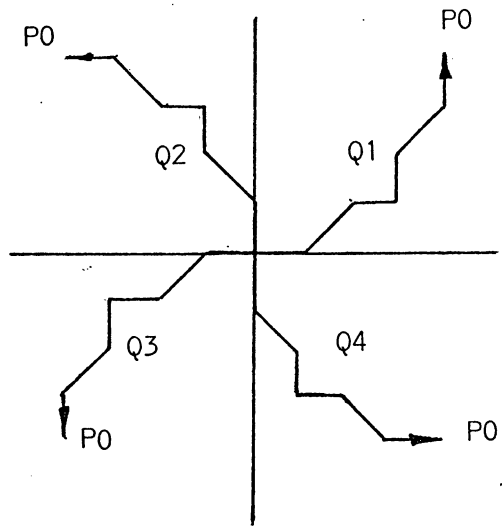


FIG. 5.2. THE FOUR QUADRANT VARIANTS OF SFS, R, P0

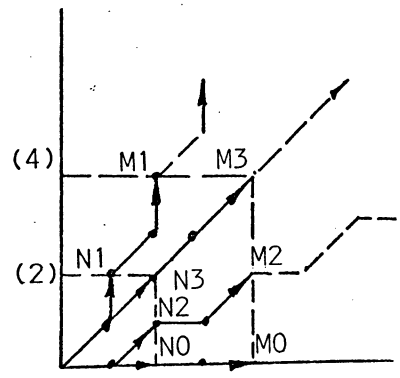


FIG. 5.3. IFS STROKES (FIRST QUADRANT)

points M0 - M3 terminate the half-length versions, and N0 - N3 terminate the quarter-length versions. Detailed coding within IFS is similar to that within SFS, except for bit r. If r = 1, the p field specifies one of M0 - M3, otherwise p specifies one of N0 - N3. As for SFS, all four quadrant versions are available.

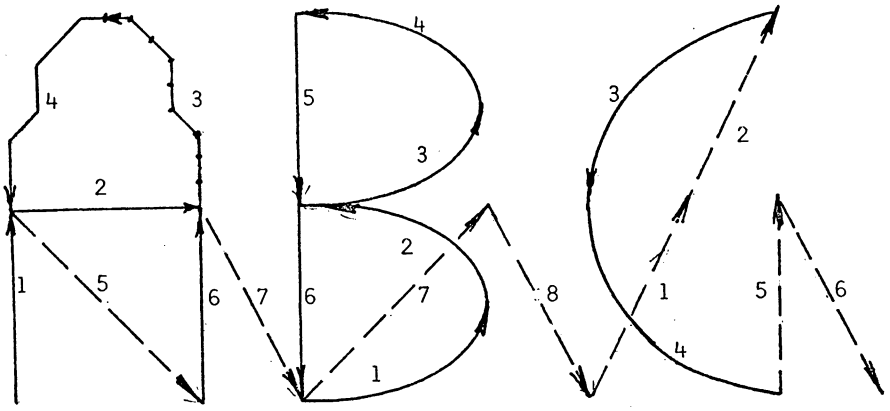
A symbol is usually plotted by executing a short sequence of SFS or IFS micro-orders. SFS and IFS orders can be arbitrarily mixed and, if necessary, other micro-orders can be included in a symbol sequence (e.g., an APV order and some associated orders could plot a long vector as part of an extensive "symbol"). Thus, the scheme integrates symbol generation with other microprogram functions within Intergraphic.

Fig. 5.4 shows a selection of upper and lower-case letters and numbers composed of SFS micro-orders only. The detailed coding for the letter "A" is shown symbolically using the following conventions;

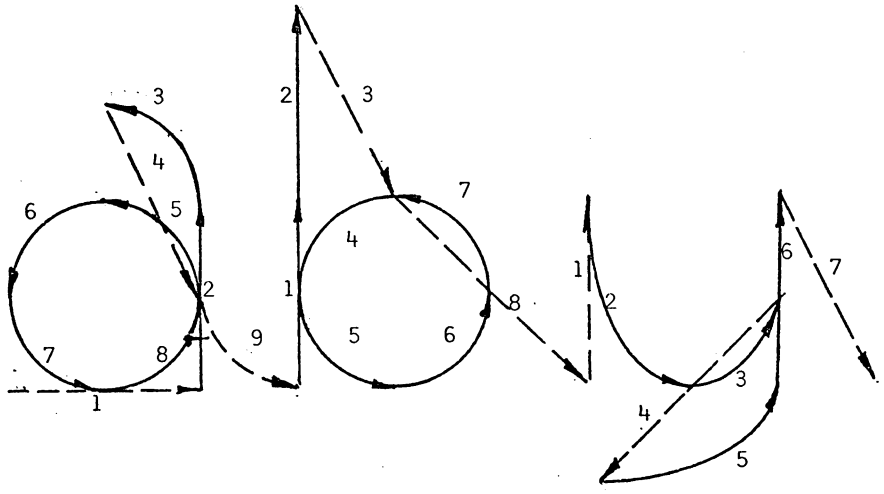
V = visible, S = straight, L = linked
B = blanked, R = rounded, E = end.

The mnemonics P0 - P3 and Q1 - Q4, are as detailed above. Only two SFS orders are necessary to code the 7 strokes of "A": the final stroke (7) leaves the CRT beam at the origin of the next symbol, and the stroke interpretation logic then returns control to the main microprogram by popping the microroutine stack into the ROM address register, S. Some points are plotted twice (e.g., the end of strokes 1 and 4 of "A"); this can often be avoided by recoding the stroke sequence (e.g., for the double-point above, stroke 1 could be blanked, and stroke 5 replaced by a new unblanked downstroke, which replaces the old unblanked 1, followed by a blanked horizontal stroke), but the author believes double-plotting of a few points is not objectionable. For binary-video storage, double plotting cannot be detected.

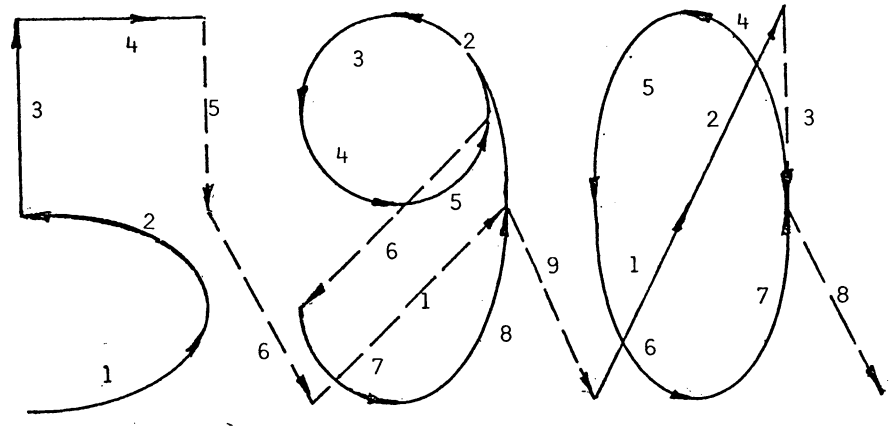
Fig. 5.5 shows the detailed coding of two complex symbols, @ and % using mixed IFS and SFS sequences. The symbolic coding for



"A": SFS VSP0Q2L, VSP0Q1L, VRP2Q2L, VRP1Q3L
SFS BSP3Q4L, VSP0Q2L, BSP2Q4E.



"a"; 3/SFS "b"; 2/SFS "y"; 2/SFS



"5"; 2/SFS "9"; 3/SFS "0"; 2/SFS

FIG. 5.4. SOME SYMBOLS COMPOSED OF SFS STROKES

IFS is identical to SFS, except that "2 or 4" replaces the "S or R" symbol for the r bit:

2 = two-increment strokes (points N, Fig. 5.3)
4 = four-increment strokes (points M, Fig. 5.3)

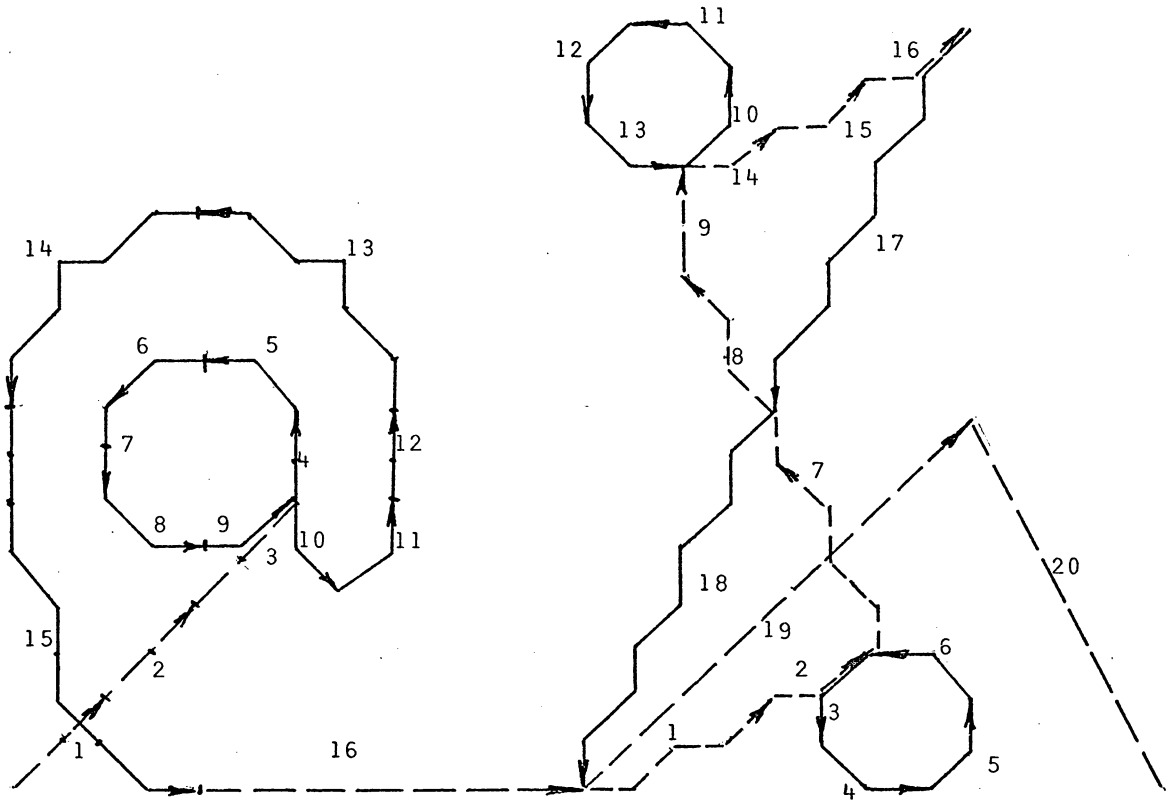
SFS or IFS orders which do not terminate a symbol, must nominate a stroke in each of their four stroke-fields; thus, in some places, a pair of two-increment strokes has been used instead of single, four-increment stroke (e.g., strokes 1 and 2 of @ are equivalent to the four-increment stroke, B, 4, P3, Q1, L.

Symbol-plotting microcodes are accessed via a table of starting addresses (SYMBOL TABLE, p A3.54) which is entered from the GRAPHICS-B order "GRB SYMBOL PAIRS" (p A3.23). The EBCDIC graphic symbol-code has been chosen to be compatible with the central processor (IBM 360/50); symbols are nominated by an 8-bit byte; e.g., "A" = X 'C1' (hexadecimal), 193 (decimal) or B '10100001' (binary). An advantage of a byte code is that 256 "symbols" are possible, so that there is ample space for introducing new symbols, which may be display symbols or control symbols. In either case, the symbols indirectly address microcode execution sequences via the SYMBOL TABLE above; thus the mode of interpretation of all "symbols" is identical - this is an advantage of symbol plotting by microcode.

Double and quadruple-sized symbols are also available: the stroke generation logic plots each elemental increment twice, or four times, respectively, unblanking the display beam each time. The time between successive points plotted is constant (approximately 80ns, i.e., a 12.5 MHz increment rate) regardless of symbol size, so that the larger symbols take proportionally longer to plot.

5.3 COMPARISON WITH DOT MATRIX AND INCREMENTAL CODES

With few exceptions, alphanumeric symbols require only two micro-orders each, i.e., 64 bits including SFS and IFS fields;




```
"@":  IFS  B2 P3 Q1 L,  B2 P3 Q1 L,  B2 P3 Q1 L,  V2 P0 Q2 L
      IFS  V2 P1 Q2 L,  V2 P2 Q3 L,  V2 P0 Q4 L,  V2 P1 Q4 L
      IFS  V2 P2 Q1 L,  V2 P2 Q4 L,  V2 P1 Q1 L,  V2 P0 Q2 L
      SFS  VR P0 Q2 L,  VR P0 Q3 L,  VR P2 Q4 L,  BS P0 Q1 E.

"%":  IFS  B4 P2 Q1 L,  B2 P2 Q1 L,  V2 P1 Q3 L,  V2 P1 Q4 L
      IFS  V2 P1 Q1 L,  V2 P1 Q2 L,  B4 P2 Q2 L,  B4 P2 Q2 L
      IFS  B2 P0 Q2 L,  V2 P1 Q1 L,  V2 P1 Q2 L,  V2 P1 Q3 L
      IFS  V2 P2 Q1 L,  B2 P2 Q1 L,  B2 P2 Q1 L,  B2 P2 Q1 L
      SFS  VS P1 Q3 L,  VS P1 Q3 L,  BS P3 Q1 L,  BS P2 Q4 E.
```

FIG. 5.5. DETAILED COMPOSITION OF @ AND % USING IFS AND SFS MICROCODE

the chosen scheme is more than twice as efficient in storage as a dot matrix encoding for the chosen symbol resolution because the equivalent dot matrix for a 16 x 8 increment grid is 17 x 9 dots; i.e., 153 bits.

Freeman (61, 1961) and McDonald, Ninke and Weller (41, 1967, GLANCE terminal) describe incremental codes which encode one of eight possible increments (right, right and up, up, etc.) using three bits. A fourth bit is included with the GLANCE code which expands the code to sixteen options and thereby allows intensity level and scaling control: scaling control allows rapid beam movement while the beam is blanked. For the symbol set adopted within Intergraphic, an average of approximately five visible strokes per symbol is used; this corresponds to an average of approximately 40 visible points per symbol which would require 120 bits for a 3-bit incremental code (assuming continuity). Thus, ignoring blanked beam movement and, for incrementally specified symbols, the additional unblanking and control bits of the incremental codes cited, the complete SFS and IFS codes are approximately twice as compact. The numbers of bits coding @ and % are 128 and 160 respectively, compared with 153 for equivalent matrix coding, or 150 and 162, respectively, for a simple 3-bit increment code. Thus, even for these intricate symbols, the chosen coding scheme is efficient.

Increment codes, however, are readily converted to beam movements, so that symbol-detail storage requirements cannot be considered in isolation: code interpretation logic must also be examined. In Intergraphic, separate digital logic converts to standard stroke codes to elemental increment strings. A simple four-stage binary counter provides a base of four binary variables to step through the cells of a pair of 4x4 Karnaugh maps for each standard stroke (first quadrant versions only): one map encodes whether the step requires an X increment or not, the other correspondingly for Y. Each step produces either an X or Y increment or both, and the end of the stroke is detected by neither an X nor a Y increment. The maps reduce considerably and have common terms; quadrant versions are derived from the first quadrant strings by transposing X and Y and changing increment signs where necessary. Simple 1 of 4 selection gates select the appropriate strings depending on

P0-P3. A pair of binary rate multipliers was also considered for circular-arc stroke generation, but the non-uniform pulse separation characteristic of BRMs, as outlined in Section. 4.5, was particularly noticeable for circular-arc strokes containing only a small number (typically 8) of increments.

CHAPTER 6

GRAPHICAL INPUT

This chapter discusses the operations of "pointing" and "tracking". "Pointing" is the nomination of a displayed item using a light-pen and associated circuitry and software; "tracking" is the following of light-pen movement, either to input a freehand curve or to move a displayed item along the pen's path.

The nomination, or identification, operation is relatively simple when a photo-sensor light-pen is pointed to a core-regenerated display. As the beam passes under the pen tip, a pulse is generated which interrupts the normal progression through the display-file cycle. If the display is structured as a simple tree (i.e., if displayed entities are identified by an ordered set of pointers or names which specify the nodes at each level of the tree), then the state of the nodal descriptors at the time of interrupt identifies the referenced item. Economical displays, however, are not continuously regenerated from core, so that an alternative scheme is necessary. The approach in Intergraphic has been to determine light-pen co-ordinates, store them in a pair of registers X_m, Y_m, and then begin and proceed with a second pass through the display file until an interruptible point is reached which has been alerted by the proximity of X,Y to X_m, Y_m. Thus, the light-pen interrupt has been replaced by coordinate-matching logic, and, at most, one pass through the display file is needed for each entity referenced. (The display is blanked during a coordinate-matching pass, and the scan-converter does not need to be free before the cycle is commenced.)

The pen-following operation is also based on pen-coordinates which are polled by Intergraphic core once in every TV frame time (40ms).

(The TV raster provides the exploratory scan for the pen [43] and the "raster-coordinates" so determined are readily converted to X,Y co-ordinates.) Pen-following is based on detecting frame-to-frame differences in pen co-ordinates, and encoding the string of significant differences which constitute the path. The TV raster scan has eliminated the need for local plotting patterns (tracking-crosses) which probe the receptive field of a pen to detect the relative position of the pen and pattern. An algorithm is described later (Sect. 6.2) which compactly encodes a freehand input curve into a piecewise-linear string.

6.1 THE "POINTING" OPERATION WITHIN INTERGRAPHIC

This section describes the matching logic, the indicators, and the distribution of interruptible points relevant to identification by pointing. Within these facilities which are at microcode level, tree and other data structures [63] can be alerted as if by conventional light-pen interrupts.

Figure 2.3, p 2.9 shows the coordinate-matching registers X_m, Y_m which are standard word-length registers. They can be copied from X,Y in one TRF micro-order and are accessed, if necessary, via the R register. Indicator F_2 (bit 2 of the bitwise programmable F register; Sect. 2.2(4) and Sect. 3.5) is automatically set when the match between X_m, Y_m and X,Y extends from bit position 0 to bit position 12. Mnemonic MC_{12} (matched co-ordinates to bit position 12) refers to F_2. However, F-indicator F_1, mnemonic MM (matching mode), must be in the 1-state and the current display item must be visible (V = 1) before MC_{12} is set by the matching logic. The MM binary, which is set or reset conventionally by an FBC micro-order, prevents a spurious interrupt from occurring when a match occurs by chance. Matching only to bit location 12 allows some tolerance between the stored pen co-ordinates and the precise display co-ordinates. This tolerance is a necessity, because light-pen

resolution is less than display-point resolution and, moreover, there is some drift in the scan converter and video-distribution system. (Either non-uniform separation between TV lines or curvature or TV lines, provided it is introduced after scan conversion, does not introduce light-pen errors because the pen's Y-ordinate is locked to the ordinates of the image. Non-uniform line velocity at a TV terminal, however, does introduce errors in the pen's X-location because abscissae are determined by a local high frequency counter which is only synchronised at the beginning of each TV line.) The selection of matched co-ordinates to bit location 12 is experimental; it can be readily changed to either a more stringent or more relaxed match, because the matching logic is an interative network and the match from location 0 to any bit location is available.

A match can be detected (i.e., a display file is interruptible) immediately after an isolated point, a vector, or a symbol has been plotted. (An isolated point is not a constituent point of a vector or symbol, but a separate entity, e.g., one point within a list of points plotted by GRA SML CTN PNTS; p A3.9) After any one of these entities has been plotted, control is transferred to an interrupt handling microroutine, "MI INTRPT", conditionally on JINT. JINT is the logical union of peripheral interrupts and internal interrupts, MC_{12} being an internal interrupt. For GRAPHICS-A orders (GRA SML CTN VTRS to GRA LRG PLR PNTS), the transfer conditional on JINT is within the short microroutine "MI RST; INTRPT; LOOP" (p A3.48) which is called within every GRA microprogram after each vector or point has been plotted. The microprograms interpreting GRAPHICS-B orders (GRB SML XTHENY VTRS to GRB EXTRACODE) each call MI INTRPT directly if JINT is 1. (The insertion of traps for testing matched co-ordinates at the end of each point, vector or symbol also ensures that peripheral interrupts from any devices connected to Intergraphic are sampled at relatively short time-intervals. The longest period between samples could be approximately 500 μs, corresponding to a vector having a 12-bit component; normally, however, interruptible points are separated by less than 100 μs.)

During the single regeneration cycle which compares X,Y with X_m,Y_m, the display is blanked by initially clearing F_5, a special-purpose F-bit denoted VM (visible mode). After a coordinate-matching interrupt is serviced, MC_{12} is reset with a normal FBC, CF micro-order. Indicator MC_{12} is the only F bit which is automatically set; others can only be set via an FBC, SF order. Thus, the setting of MC_{12} is analogous to the setting of a binary by the "strike" pulse from a conventional light-pen. The matching logic, indicators and interruptible points, merely provide general microcode-level, pointing facilities; they do not restrict the choice of display-file structure.

6.2 THE "TRACKING" OPERATION WITHIN INTERGRAPHIC

6.2.1 Introduction

The object of tracking may be solely to encode a freehand curve, in which case tracking and encoding can be accomplished entirely within the interface; on completion of the curve, the encoding is transmitted to the user's main program held in the CPU (the parent program). Here, only one transmission of data from the interface to the parent program is necessary. A number of transmissions, however, would be needed if the pen's path defines a locus for successive copies of a rigid subpicture (guided translational movement) or a locus for one point of a picture which is progressively reshaped according to relations held in the parent program (e.g., the pen's ordinate defines the deflection of the free-end of a cantilever frame and the display shows, for each significant ordinate value, the corresponding deformation of the frame). Multiple transmission in guided translational movement is necessary, because the user requires to see the subpicture in its various locations against its background; this, in turn, requires multiple updating of the display file which must reside in the CPU because the shared interface has insufficient memory to hold the display file of any one program for an indefinite period of time. In the reshaped

picture example above, multiple transmission is again necessary to input a parameter (end-point deflection) to the parent program. Here, each significant value of the parameter, possibly after extensive calculation, defines a new shape, i.e., requires the display file to be updated which then sets a new display task for the interface.

For a graphical-communication system to be economical, the number of new images displayed and the amount of data transmitted between the interface and the CPU must be reduced whenever practicable.

Various schemes can be used for the transmission of curves to a parent program. Four examples are; a sequence of co-ordinates sampled at regular time intervals during the drawing of the curve; a sequence of co-ordinates such that the differences between successive co-ordinates have a constant magnitude, or nearly so; a sequence of co-ordinates which define the breakpoints of a piecewise-linear approximation to the curve; and compact parametric specifications derived from an extensive analysis of the curve. Constant-time sampling is extremely inefficient when the real-time detail of data is not required, which is assumed, because a stationary point generates a stream of identical co-ordinates. Constant-separation (spatial) sampling is well-suited to one curvature, or small range of curvatures, but higher curvatures are inadequately encoded and lower curvatures are over detailed. Piecewise-linear encodings are more efficient than the previous two, but require algorithms for breakpoint determination. Compact parametric specification of curves is the most economical in transmission, but also the most demanding in processing time. The compromise is similar to that outlined under output-curve specification (cf. Sect. 4.6).

A piecewise-linear scheme has been adopted in Intergraphic. It is necessary to distinguish between a scheme which declares breakpoints after the curve has been completed (a "static" scheme) and a scheme which inserts breakpoints as the curve is being formed (a "dynamic" scheme). A static scheme can encode the curve more compactly, but it requires a relatively complex algorithm and presupposes that the entire curve is stored before analysis. This storage is impractical, because an input-curve can be arbitrarily long, and, even if inputs are restricted

in extent, the total storage which would need to be reserved in a system with many terminals would be excessive. A dynamic scheme has been chosen because it requires only a small amount of storage, a simple algorithm has been devised, and the compression of the encoding which it generates approaches that of the static scheme. (Because the difference in coding efficiencies between the one extreme of constant-time encoding and the other extreme of compact parametric encoding is so great, the author considered that a detailed comparison of the compressions of static and dynamic piecewise-linear schemes was not justified for the prototype system.)

6.2.2 The Dynamic Breakpoint-Insertion Algorithm

This algorithm selects breakpoints from the stream of X,Y co-ordinates which are determined by sampling the input graphic at regular time intervals (one co-ordinate pair per TV frame-time, namely, 40 ms). It is assumed that the drawing rate is sufficiently slow so that the sample-points adequately represent the input. Figure 6.1 illustrates a typical input curve, the stream of sample-points $(s_0, s_1, .. s_i ..)$ and the sparser stream of breakpoints $(b_0, b_1 .. b_i ..)$ selected by the algorithm.

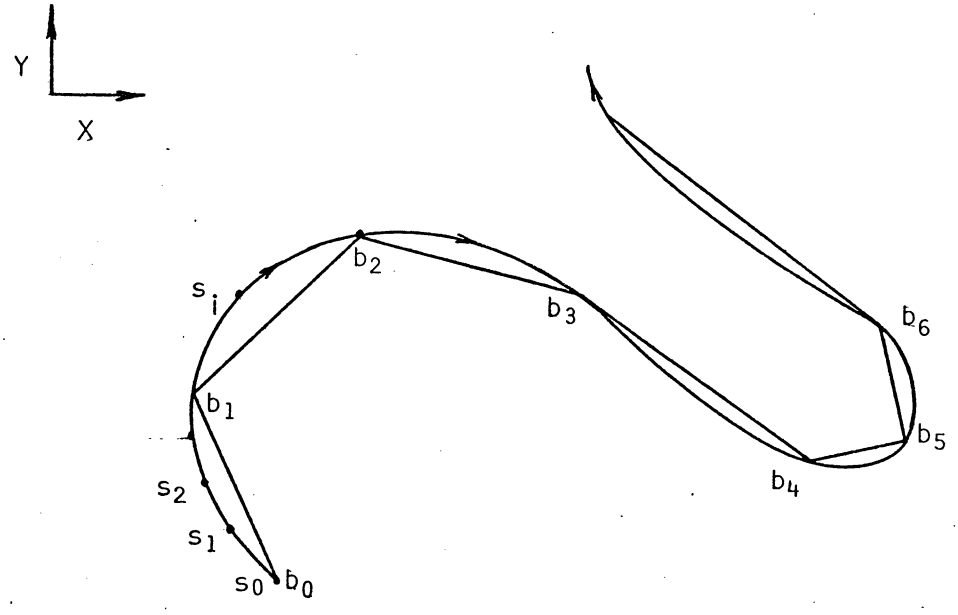


FIG. 6.1. TYPICAL INPUT CURVE, CONSTANT-TIME SAMPLES AND BREAKPOINTS

The algorithm declares the first sample-point s_0 as the initial breakpoint b_0; it then considers s_1, s_2, etc., in sequence, as possible points for b_1. Figure 6.2 illustrates the hypothesis, $b_1 = s_3$.

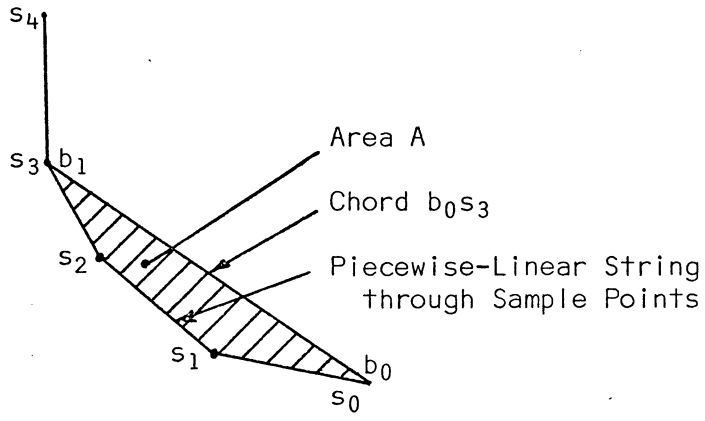


FIG. 6.2. ILLUSTRATION OF THE HYPOTHESIS THAT BREAKPOINT b_1 EQUALS SAMPLE-POINT s_3

Breakpoint b_1 is declared when the mean departure of the chord $b_0 s_i$ from the fine piecewise-linear string $s_0 s_1 .. s_i$ exceeds a preset threshold t. Departure is defined to be measured normal to the chord. Thus, b_1 is declared when,

$$\frac{|A|}{|b_0 s_i|} > t \qquad \text{where,}$$

A is the area between the chord $b_0 s_i$ and the piecewise-linear string (the reckoning of negative area is discussed later), $|A|$ is the magnitude of A, and $|b_0 s_i|$ is the length of the chord $b_0 s_i$.

Breakpoint b_2 is then found in an identical manner, but the origin for this determination is taken at b_1 instead of b_0.

In general, then, b_{k+1} is declared when

$$\frac{|A|}{|b_k s_i|} > t \qquad \text{where,}$$

A is the included area between the chord $b_k s_i$ and the string $s_j s_{j+1} .. s_i$ ($b_k = s_j, i > j$). Figure 6.3 illustrates the calculation of A.

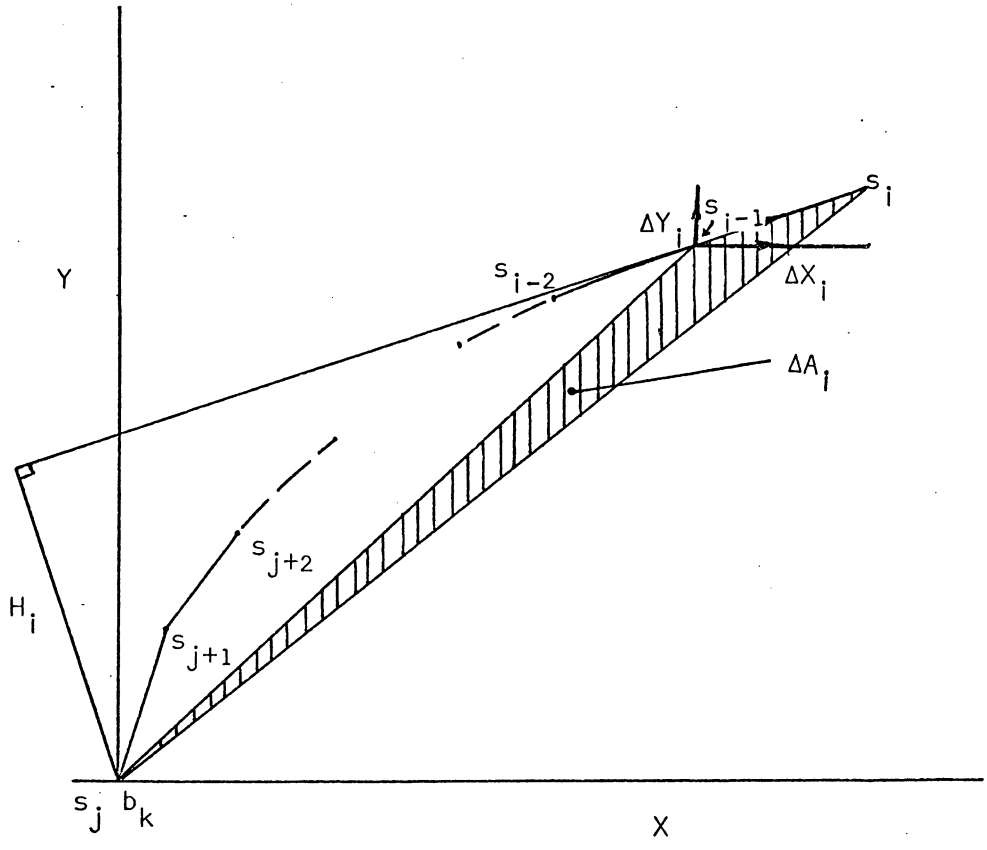


FIG. 6.3. THE CALCULATION OF INCLUDED AREA A

The shaded area labelled ΔA_i is the increase in A from the hypothesis $b_{k+1} = s_{i-1}$ to the hypothesis $b_{k+1} = s_i$.

$$\text{Incremental Area } \Delta A_i = \tfrac{1}{2} \, |s_{i-1}s_i| . H_i$$

where $|s_{i-1}s_i|$ is the base of the triangle $b_k s_{i-1} s_i$, and H_i is its altitude measured perpendicular to the base.

But $|s_{i-1}s_i| . H_i$ is the moment of the vector $s_{i-1}s_i$ about the previous breakpoint b_k. Therefore, $\Delta A_i = \tfrac{1}{2}$ moment of $s_{i-1}s_i$ about b_k.

However, the moment of $s_{i-1}s_i$ about b_k equals the moment of the components of $s_{i-1}s_i$ $(\Delta X_i, \Delta Y_i)$ about b_k, provided the intersection of these components lies on the line through s_{i-1} and s_i. Considering the components intersecting at s_{i-1}, as shown in figure 6.3,

$$\Delta A_i = \tfrac{1}{2}(Y_{i-1}\Delta X_i - X_{i-1}\Delta Y_i)$$

where X_{i-1}, Y_{i-1} are the co-ordinates of s_{i-1} with respect to b_k.

Alternatively, the components ΔX_i, ΔY_i may be displaced to intersect at s_i without affecting their moment about b_k, i.e., alternatively,

$$\Delta A_i = \tfrac{1}{2}(Y_i\Delta X_i - X_i\Delta Y_i)$$

Thus, area A may be calculated by accumulating a pair of simple products for each step along the sequence of sample points. The operands $\Delta X_i, \Delta Y_i$ are fed directly to the algorithm if either is non-zero, and operands X_i, Y_i are readily found by accumulation. It is noted that this determination of A is precise even for coarse sampling. The expression for ΔA_i can clearly be negative, as can the accumulation A.

Chord length $|b_k s_i|$ may be calculated precisely from $\sqrt{X_i^2 + Y_i^2}$, but the estimate

$$\tfrac{1}{2}(|X_i| + |Y_i|)$$

is adequate for breakpoint insertion, and saves considerable computing time. Thus, breakpoint b_{k+1} is declared when

$$\frac{|\Sigma(Y_i \Delta X_i - X_i \Delta Y_i)|}{|X_i| + |Y_i|} > t$$

and, on declaration of b_{k+1}, accumulators X_i, Y_i and A are cleared and the process begins again and proceeds until b_{k+2} is selected.

This technique for breakpoint insertion requires very little computing and storage and, at the speeds possible with efficient microprograms within Intergraphic, the algorithm can issue breakpoints in step with the receipt of sample points. Minor variations on the above process are possible, e.g., declaring s_{i-1} as the breakpoint instead of s_i. The algorithm outlined above can be extended to overcome the following objection. Suppose a straight line were drawn and retraced backwards. A breakpoint would not be inserted at the point of reversal by the algorithm because the included area A would remain zero. Monitoring changes of sign of ΔX and ΔY can detect this situation.

It is also possible to refine the algorithm by defining breakpoints which are not sample-points; e.g., when the basic algorithm above declares a breakpoint, the mean departure of the chord from the sampled graphic is known and the co-ordinates of the chord are known; therefore, it is feasible to displace the breakpoint normal to the chord to reduce the mean deviation. For curves of approximately constant curvature, the breakpoints can be placed outside of the curve such that the mean departure is zero, i.e., the maximum magnitude of the

departure could be approximately halved. However, inflexed graphics introduce difficulties; the author considers that this refinement is possibly not justified in the present application, but recommends further analysis. The author also recommends that the coding efficiencies of several dynamic and static schemes be defined and compared.

An advantage of a dynamic scheme is that the interface can display to the person drawing the curve the piecewise-linear approximation, breakpoint-by-breakpoint, as declared by the algorithm. It is impossible to display every sampled point (40 ms intervals) via the scan-converter to provide feedback from the interface (the TV terminal itself can provide local feedback every frame-time, but this does not verify the samples received by the interface). Moreover, by definition, it is impossible to feedback to the user, breakpoint-by-breakpoint, the piecewise-linear approximation derived by a static scheme.

In the event of a number of users simultaneously inputting curves, the dynamic algorithm updates the various accumulators for each input every 40 ms, checks for breakpoints and adds new breakpoints, when declared, to the breakpoint lists corresponding to each input. If the demand on the interface is such that the vector strings corresponding to these breakpoint lists cannot be displayed "in-step" with the determination of new breakpoints, then some fedback versions will be delayed by then updated by several vectors at a time. The user requiring "in-step" visual feedback will therefore need to reduce his input rate. Provided the individual drawing rate is such that 40 ms sampling adequately defines the input curve, then, even though the fedback version may be delayed by excessive demand on the interface, there is no loss in precision in defining breakpoints and eventually the interface's breakpoint versions of all input curves will be displayed. The proposed scheme, therefore, does not degenerate in precision with increased demand, but causes users to reduce their input rate.

CHAPTER 7

CONCLUSIONS

The work described in the thesis shows that economical graphical communication is possible using state-of-the-art devices. Low cost is achieved by centralizing commonly-used graphical functions at an interface computer and using a low-cost television storage and transmission system for maintaining and distributing images. Scan conversion at the interface allows the functions of image generation and image distribution to be independent of one another; this allows each to be developed more economically than it could be within a system in which distribution considerations constrain generation techniques and vice versa. The interface generates display points synchronously in conventional computer-driven display modes (incremental and random point modes) whereas the terminals receive display points synchronously in a regular scanning mode.

The centralized system chosen is compared with systems which require local display processors, and the particular scheme which was chosen for image generation, storage and distribution is compared with five alternative schemes. Provided video-frequency connection is feasible (very extensive video networks link 7% of all TV homes in the United Kingdom at low per unit cost), the chosen scheme is currently the most economical.

The thesis shows, through the detailed logical structure of the interface and microprograms, that a microprogrammed interface is well-suited to the numerous tasks imposed by graphical communication and a large number of terminals. The microprograms written to date for a variety of tasks, illustrate that the interface's repertoire can be

readily extended by short microprograms (typically 10 micro-orders long). This flexibility, inherent in interpretation by microprogram, is also available to the more conventional tasks within the interface, e.g., supervisory tasks. As multiple-terminal, graphical-communication is still in the exploratory stage, the flexibility of the interface is particularly useful for experiments in graphical languages, operating systems (or more precisely, those components of proposed operating systems which would reside in the interface), post-processing CPU data before display, pre-processing input data before transmission to the CPU, etc. The commonly used functions of vector and symbol generation have been decomposed into short microprograms, the execution of which, in turn, has been assisted by special micro-orders.

Vector generation is integrated within the logical design of the interface in a similar manner to commonly-occurring computational procedures such as multiplication. The thesis details the vector generation mode and compares the chosen method with special vector generation devices; it shows that the general-purpose microprogrammed interface with the small logical extensions for vector generation is competitive, and requires no new circuit devices. Moreover, through microprogramming, the vector repertoire can be readily expanded. To date, the repertoire comprises four basic vector types and six special vector types. The basic vectors are Cartesian or polar in large or small format; special vectors facilitate the coding of block diagrams, regular grids, circular arcs and diagrams consisting of only horizontal and vertical lines.

The thesis discusses the more general question of digital generation of graphics versus analogue and hybrid generation. It defends the approach which,

(i) limits the variety of directly-interpretable graphics to straight lines and circular arcs,

(ii) generates these vector and circular elements very rapidly and precisely within the standard digital framework of the interface,

(iii) relies on CPU software to partition arbitrary curves into piecewise-linear or circular segments.

However, from the diversity of approaches which have been proposed, it is apparent that techniques for curve generation within computer-graphics is still exploratory, and further work is necessary.

Symbol generation has also been integrated within the logical structure of the interface. Symbols are synthesized from standard straight lines, standard quarter circles and standard quarter ellipses or, for intricate sections, from short elements of either 2 or 4 increments in length. Compositions are encoded in the interface's ROM, and a small amount of stroke-detail logic assists in the interpretation of standard strokes. The chosen scheme has shown that it is possible to use standard main-frame logic techniques to generate symbols and that a separate symbol-detail store is not necessary. A comparison with existing dot-matrix and incremental coding schemes shows that, even for intricate symbols, the chosen scheme is very efficient in storage. An advantage of the linked-stroke scheme is that symbols can be of arbitrary extent. Moreover, by integrating stroke generation within the interface, symbols can also contain non-standard strokes which are interpreted by the vector generation logic rather than the stroke generation logic. Provision is made for 256 "symbols", some of which are control symbols; the plotting rate is limited by the deflection bandwidth, not by the stroke generation logic.

The thesis also shows that arbitrary graphics can be accepted as inputs from many users simultaneously (Sect. 6.2.2). A simple algorithm is outlined which encodes an input as a piecewise-linear string having a prescribed maximum mean departure from the graphic. The advantage of the particular dynamic algorithm described is that it declares breakpoints in step with the input and this enables the computer's interpretation (coding) of the graphic to be fedback to the user as he draws. Displaying the piecewise-linear version is practical because there are normally far fewer breakpoints than sample-points

(25 samples per second). Further work is recommended in the refinement and comparison of dynamic encoding schemes. It would appear that the encoding derived from an efficient piecewise-linear scheme could become the input to higher-level algorithms which determine curvature and partition a graphic into elements as a basis for recognition.

At the current interface speeds (3-5 ns logic, 50 ns access-time ROM, and 10 MHz incremental plotting rates for vectors and symbols), images comprising some 500 short vectors or 1000 symbols can be transferred from the CPU to the interface and plotted in less than 10 ms, i.e., in less than a quarter of a TV frame period. The current limitations of the system are the response of the reading section of the scan converter and the limited resolution of the TV storage and distribution system. Further work is recommended on the development of single-beam, electrostatically deflected scan-converter tubes of improved resolution (single-beam to eliminate alignment problems between the reading and writing sections of a dual-tube or dual-ended system; electrostatic deflection to achieve high-frequency response; and higher resolution to match a 1024x1024 computer-driven display rather than a TV monitor of approximately 500 lines). Improved video storage is also required to match a higher quality (say 800-1000 line) TV system. Because a scan converter would be shared by say 32 terminals (a fully developed system of some 100 terminals would need a set of scan converters), the converter can be relatively expensive ($10,000). Alternatively, as outlined in Sect. 1.11.2 *Scheme 5*, studies in core-store (or equivalent) scan-conversion appear to be well worthwhile, because they result in digital conversion.

Local image-retention displays (e.g., DVST's) greatly reduce the total data transmission rate in television distribution systems with predominantly static images, and, as this type of operation could possibly dominate low-cost multi-terminal systems for some time, further developments are justified. Local storage displays also eliminate the need for a multi-track video disc which, being electro-mechanical, is potentially limited in high frequency performance.

The cost of the Intergraphic system developed to 32 terminals is expected to be less than $2,000 per terminal. This cost includes the cost of some 2000 integrated circuits; a 4096 word, 18-bit core store; a 4096 word, 33-bit ROM; a scan converter; a video disc; and the TV receivers and light-pens. The cost does not include development costs for either hardware or software, overheads, etc., but it does indicate that the per-terminal cost of graphical systems will become compatible with that of current electro-mechanical teletype terminals. The advantages of speed, quiet operation and an ability to transmit arbitrary graphics are apparent.

An order of magnitude increase in the speed of the interface is feasible using sub-nanosecond logic and improved output amplifiers for the ROM (the word-drive circuits and basic ferrite coupling elements are non-saturating and are capable of a frequency-response matching a 5 ns access-time ROM). The D/A converters, currently 25 ns settling time, have not limited the performance of the interface to date; they also use non-saturating techniques which are capable of extension to match a 100 MHz incremental system. At these rates, an interface having the same logical structure as Intergraphic could produce some 1000 new images per second, which would enable dynamic displays to be distributed to say 30 terminals, or static displays to more than 1000 terminals. An image distribution system of this performance would impose a heavy load on the CPU(s), but inclusion of sub-nanosecond logic in these main processing units is equally feasible.

Some applications require precision graphics, but although such graphics could be interpreted by the interface described in the thesis, it is unlikely that low-cost distribution schemes will have sufficient resolution for these applications. Rather, precision terminals would be regarded as special cases. Also, highly dynamic displays are so demanding, particularly if good resolution is also required, that a special interface is justified, at least until 100 MHz increment rates are achieved.

For terminals which are located such that only dataphone links are practical, the new image rate is limited by transmission bandwidth; thus, compact codes are necessary and relatively complex decoding circuits are required at each terminal.

A large, comprehensive graphical-communication system, then, will need a variety of terminal types having different resolutions, transmission bandwidths and operating modes. It is expected that many of the terminals in such a system would be TV terminals capable of displaying arbitrary graphics every few seconds. The author has recently proposed a possible configuration for an extensive, yet economical, computer-graphics network [49], and attributes low costs to;

(i) Borrowing from non-computer fields (radar, television, etc.) techniques such as scan conversion, television image transmission, frequency multiplexing and video signal storage;

(ii) Classifying applications into broad classes according to terminal performance requirements, and providing matching terminal types (high-precision, high-resolution; standard television; and dataphone connected DVST's);

(iii) Improving systems organization, e.g., introducing shared graphical-interface computers which allow more efficient task distribution, reduce central processor interrupts and interpret more compact data codes;

(iv) Improving logical design, e.g., plotting vectors and symbols by high-speed microcode, which eliminates the need for special vector and symbol generation hardware;

(v) Replacing core-store image regeneration by video-disc image refreshing or storage-tube image retention.

The significance of economical graphical communication is that current planning of computer utilities should include extensive computer-graphics networks. Such networks will greatly increase the availability of computer-graphics to professional workers and students in many fields, e.g., in education at all levels and in machine-aided design. The thesis has not debated whether computer graphics has sufficient application to justify its widespread availability - the usefulness of computer-graphics has been clearly established by other workers - the thesis has concentrated on the extension of graphical-communication to many users at low-cost.

BIBLIOGRAPHY

1. G.J. CULLER and B.D. FRIED, "An on-line computing centre for scientific problems", TRW Computer Division, Canonga Park, Calif., Rept. M19-3U3, June, 1963.

2. I.E. SUTHERLAND, "SKETCHPAD, a man-machine graphical communication system", 1963 Spring Joint Computer Conf., AFIPS Proc., Vol. 23, pp. 329-346.

3. B. HARGREAVES et al., "Image processing hardware for a man-machine graphical communication system", 1964 Fall Joint Computer Conf., AFIPS Proc., Vol. 26, Pt. 1, pp. 363-386.

4. W.F. BAUER and W.L. FRANK, "DODDAC - An integrated system for data processing, interrogation, and display", 1961 Eastern Joint Computer Conf., Proc., pp. 17-29.

5. R.M. FANO, "The MAC system: the computer utility approach", IEEE Spectrum, Vol. 2, pp. 56-74, January, 1965.

6. F.J. CORBATO and V.A. VYSSOTSKY. "Introduction and overview of the Multics system", 1965 Fall Joint Computer Conf., AFIPS Proc., Vol. 27, Pt. 1, pp. 185-196.

7. D.E. RIPPY et al., "MAGIC, a machine for automatic graphics input to a computer", ibid., pp. 819-830.

8. W.H. NINKE, "Graphic I - a remote graphical display console system", ibid., pp. 839-846.

9. R.W. LICHTENBERGER and M.W. PIRTLE, "A facility for experimentation in man-machine interaction, ibid., pp. 589-598.

10. J.R. KENNEDY, "A system for time-sharing graphic consoles", 1966 Fall Joint Computer Conf., AFIPS Proc. Vol. 29, pp. 211-222.

11. N.A. BALL, H.Q. FOSTER, W.H. LONG, I.E. SUTHERLAND and R.L. WIGINGTON, "A shared memory computer display system", IEEE Trans. Electronic Computers, Vol. EC-15, pp. 750-755, October, 1966.

12. S.A. COONS, "An outline of the requirements for a computer-aided design system", 1963 Spring Joint Computer Conf., AFIPS Proc., Vol. 23, pp. 299-304.

13. E.L. JACKS, "A laboratory for the study of graphical man-machine communication", 1964 Fall Joint Computer Conf., AFIPS Proc., Vol. 26, pp. 343-350.

14. M.V. MATHEWS and J.E. MILLER, "Computer editing, typesetting and image generation", 1965 Fall Joint Computer Conf., AFIPS Proc., Vol. 27, Pt. 1, pp. 389-398.

15. M.D. PRINCE, "Man-computer graphics for computer-aided design", Proc. IEEE, Vol. 54, No. 12, pp. 1698-1708, December, 1966.

16. J.R. LOURIE and J.J. LORENZO, "Textile graphics applied to textile printing", 1967 Fall Joint Computer Conf., AFIPS Proc., Vol. 31, pp. 33-40.

17. G.A. CHAPMAN and J.J. QUANN, "VISTA - Computed motion pictures for space research", ibid., pp. 59-63.

18. C. WYLIE, G. ROMNEY, D. EVANS and A. ERDAHL, "Half-tone perspective drawings by computer", ibid., pp. 49-58.

19. G.W. ROMNEY, G.S. WATKINS and D.C. EVANS, "Real-time display of computer-generated half-tone perspective pictures", 1968 Edinburgh IFIP Conf. Proc., Hardware 2 Section.

20. R.H. TERLET, "The CRT display subsystem of the IBM 1500 instructional system", 1967 Fall Joint Computer Conf. AFIPS Proc., Vol. 31, pp. 169-176.

21. E. HERBERT, "Report on technology for education", International Science and Technology, Vol. 68, pp. 28-49, August, 1967.

22. A. APPEL, "Some techniques for shading machine renderings of solids", 1968 Spring Joint Computer Conf. AFIPS Proc., Vol. 32, pp. 37-45.

23. A. APPEL, "On calculating the illusion of reality", 1968 Edinburgh IFIP Conf. Proc., Hardware 2 Section.

24. R.A. WEISS, "BE VISION, a package of IBM 7090 Fortran programs to draw orthographic views of combinations of plane and quadric surfaces", J. Assoc. Computing Machines, Vol. 13, No. 2, pp. 194-204, April, 1966.

25. D.J. HALL, G.H. BALL, D.E. WOLF and J.W. EUSEBIO, "Promenade - an interactive graphics pattern-recognition system", 1968 Edinburgh IFIP Conf. Proc., Hardware 2 Section.

26. G.A. ROSE, "Economical, graphical-communication techniques for multiple console operation", Third Australian Computer Conf. Proc., pp. 399-402, May, 1966.

27. G.A. ROSE, "Light-pen facilities for direct view storage tubes", IEEE Trans. Electronic Computers, Vol. EC-14, No. 4, pp. 637-639, August, 1965.

28. J.C. GRAY, "Compound data structures for computer-aided design: A survey", Assoc. for Computing Machinery, Proc. 22nd National Conf., 1967, pp. 355-365.

29. D.T. ROSS, "The automated engineering design (AED) approach to generalized computer-aided design", ibid., pp. 367-385.

30. A. VAN DAM and D. EVANS, "A compact data structure for storing, retrieving and manipulating line drawings", 1967 Spring Joint Computer Conf. AFIPS Proc., Vol. 30, pp. 601-610.

31. C. CHRISTENSEN and E.N. PINSON, "Multi-function graphics for a large computer system", 1967 Fall Joint Computer Conf. AFIPS Proc., Vol. 31, pp. 697-711.

32. J.D. JOYCE and M.J. CIANCIOLO, "Reactive displays: improving man-machine graphical communication", ibid., pp. 713-721.

33. R.A. MORRISON, "Graphic language translation with a language independent processor", ibid., pp. 723-729.

34. W.M. NEWMAN, "A system for interactive graphical programming", 1968 Spring Joint Computer Conf. AFIPS Proc., Vol. 32, pp. 47-54.

35. W.M. NEWMAN, "Definition languages for use with the reaction handler", Computer Technology Group Report 67/9, Imperial College, London, October, 1967.

36. W.M. NEWMAN, "The ASP-7 ring structure processor", Computer Technology Group Report, 67/8, Imperial College, London, October, 1967.

37. K. LOCK, "Structuring programs for multi-program, time-sharing, on-line applications", 1965 Fall Joint Computer Conf. AFIPS Proc., Vol. 27, Pt. 1, pp. 457-472.

38. S. BOWMAN and R.A. LICKHALTER, "Graphical data management in a time-shared environment", 1968 Spring Joint Computer Conf AFIPS Proc., Vol. 32, pp. 353-361.

39. R.H. STOTZ and T.B. CHEEK, "A low-cost graphic display for a computer time-sharing console", Society for Information Display, 8th National Symposium, pp. 91-97, May, 1967.

40. R.A. AZIZ, "An instructional display terminal", ibid., pp. 83-90, May, 1967.

41. H.S. McDONALD, W.H. NINKE and D.R. WELLER, "A direct-view CRT terminal for remote computing", Digest of Technical Papers, 1967 International Solid-State Circuits Conf., Vol. 10, pp. 68-69.

42. G.A. ROSE, "Intergraphic - a microprogrammed graphical-interface computer", IEEE Trans., Electronic Computers, Vol. EC-16, No. 6, pp. 773-784, December, 1967.

43. M. MACAULAY, "Low cost terminals using television techniques", IREE (Australia), Vol. 29, No. 9, pp. 307-312, September, 1968.

44. S.B. GRAY, "A computer time-shared display", Society for Information Display, January/February, 1966, pp. 50-51.

45. R.P. GABRIEL, "Wired broadcasting in Great Britain", IEEE Spectrum, pp. 97-105, April, 1967.

46. L.C. HOBBS, "Display applications and technology", December, 1966, pp. 1870-1884.

47. C. MACHOVER, "Graphic CRT terminals - characteristics of commercially available equipment", 1967 Fall Joint Computer Conf. AFIPS, pp. 149-159.

48. M.B. CLOWES and J.R. PARKS, "Improved two-dimensional potentiometer and arrangements for use therewith", British Patent Spec. 982,008, Published February, 1965.

49. G.A. ROSE, "Computer graphics communication systems", 1968 Edinburgh IFIP Conf. Proc., Invited Papers Section.

50. M.W. ALLEN, T. PEARCEY, J.P. PENNY, G.A. ROSE and J.G. SANDERSON, "CIRRUS, an economical multiprogram computer with microprogram control", IEEE Trans. Electronic Computers, Vol. EC-12, pp. 663-671, December, 1963.

51. P. FAGG et al., "IBM System/360 engineering", 1964 Fall Joint Computer Conf., AFIPS Proc., Vol. 26, Pt. 1, pp. 205-231.

52. M.J. FLYNN and M.D. MacLAREN, "Microprogramming revisited", ACM, 22nd National Conf., Proc., pp. 457-464, 1967.

53. R. STOTZ, "Man-machine console facilities for computer-aided design", 1963 Spring Joint Computer Conf., AFIPS Proc., Vol. 23, pp. 323-328.

54. I.E. SUTHERLAND, "Sketchpad, a man-machine graphical communication system", Lincoln Laboratory, Tech. Rept. 296, Appendix E, 1963. (This report contains additional material to (2)).

55. T.E. JOHNSON, "Analog generator for real-time display of curves", M.I.T. Lincoln Laboratory, Lexington, Mass., Tech. Rept. 398, July, 1965.

56. L.G. ROBERTS, "Conic display generator using multiplying digital-analogue converters", Lincoln Laboratory Rept. DS2978, March, 1966.

57. H. BLATT, "Conic display generator using multiplying digital-analog decoders", 1967 Fall Joint Computer Conf. Proc., AFIPS Proc., Vol. 31, pp. 177-184.

58. M.L. DERTOUZOS and H.L. GRAHAM, "A parametric graphical display technique for on-line use", 1966 Fall Joint Computer Conf., AFIPS Proc., Vol. 29, pp. 201-209.

59. R. LOEWE, R.L. SISSON and P. HOROWITZ, "Computer generated displays", Proc. IRE, Vol. 49, pp. 185-195, January, 1961.

60. H. GROLL, "A comparison of various displays of alpha-numerical symbols on cathode-ray tubes", Nachr.-Techn. Z-CJ, No. 3, pp. 133-143, 1964.

61. H. FREEMAN, "On the encoding of arbitrary geometric configurations", IEEE Trans. on Electronic Computers, Vol. EC-10, pp. 260-268, June, 1961.

62. S.H. CHASEN, "The introduction of man-computer graphics into the aerospace industry", 1965 Fall Joint Computer Conf. AFIPS Proc., Vol. 27, pp. 883-891.

63. C.I. JOHNSON, "Principles of interactive systems", IBM Systems Journal, Vol. 7, Nos. 3 and 4, pp. 147-173, 1968.

APPENDIX A1

CORE STORE - CONVENTIONAL MODES

This appendix describes the conventional read and/or write mode control of the core store. APPENDIX A2 describes the core-store mode which simulates the ROM.

Section 3.2 introduced two micro-orders, ACC and ACN, which control the conventional operation of the core store. Both orders are ABA, UL orders, but, after the arithmetic operation is completed, a core cycle is initiated. The four options, introduced on p 3.8 are shown in Fig. A1.1, p. A1.2. Options RR and CW are full cycles which require 1.5μs, whereas RO and WO are part cycles and require only 1.0μs. Option RR automatically restores an accessed word, but RO leaves the location cleared. Option WO requires that the store location be clear initially; thus, the longer CW cycle is necessary if the pre-write contents are unknown, or known to be non-zero.

Three J-indicators have been provided to mark the completion of various core-cycle events: JCNB (J-Core-Not-Busy) becomes 1 at the completion of any core cycle; JCRA (J-Core-Read-Available) becomes 1 in RR and RO cycles when the core data has been read to CSD, transferred to register M and successfully checked for parity; and JCWA (J-Core-Write-Accepted) becomes 1 in CW and WO cycles when the data to be read into store has been transferred from M to CSD. All ACC and ACN micro-orders should await the availability of core-store, i.e., should nominate JCNB, WT. Clearly, write-data in M should not be overwritten by microprogram until JCWA becomes 1, and micro-orders awaiting read-data in M must nominate JCRA, WT. The core store address register, CSA, is automatically copied from C or N during the first 200 ns of any core store cycle, so that the nominated address source (C or N) must not be changed by microprogram during this interval.

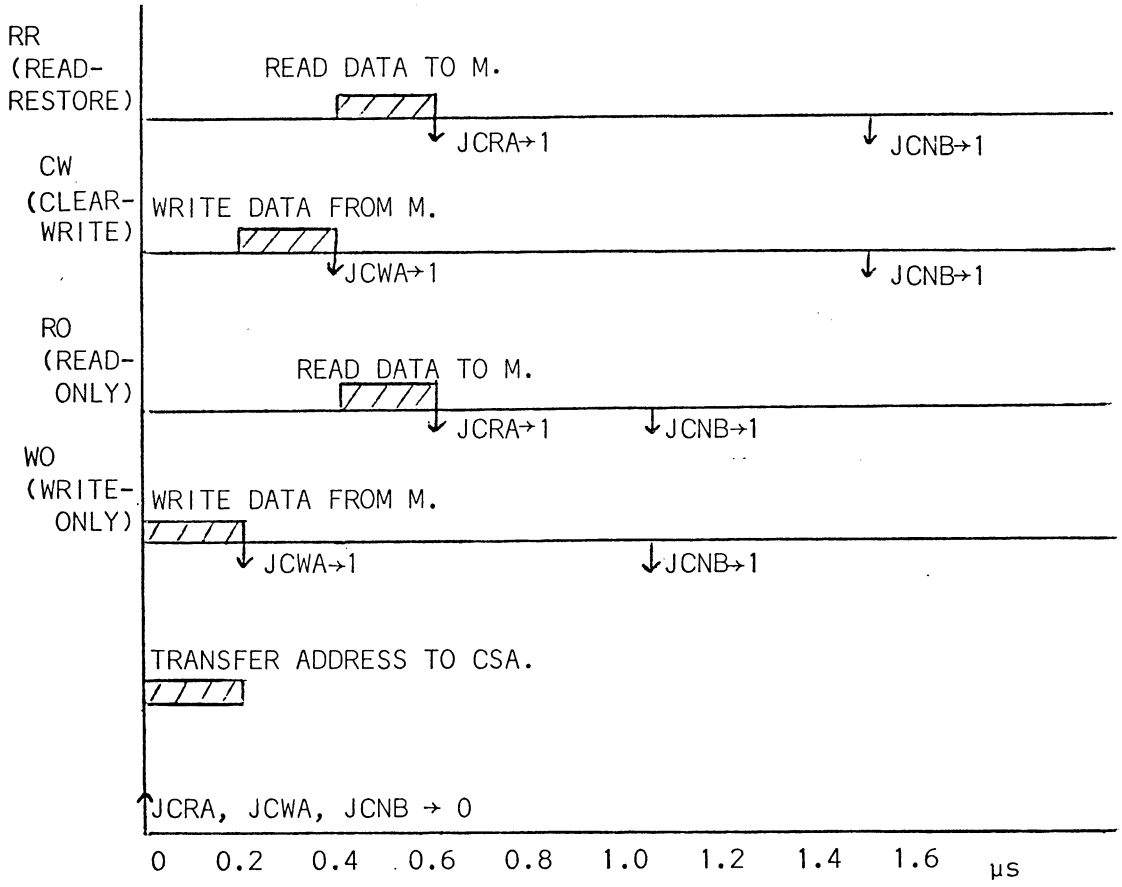


FIG. A1.1. CONVENTIONAL CORE-STORE TIMING

For normal operating speeds, there is time for write-data to be set into M between an ACC or ACN (denoted AC) micro-order which initiates a CW cycle and the beginning of the automatic transfer of M into CSD; also, in read cycles, there is time to access the pre-read contents of M before it is overwritten automatically from CSD. However, single-shot or slow-run execution of micro-orders is desirable and has been provided in Intergraphic. These slowed operational modes prohibit the late setting or accessing of M described above, because the core cycles would be completed before the execution of any orders following the initiating AC orders. In summary, M must be neither accessed nor set between an AC order and an order conditioned either by JCWA or JCRA.

Provided the constraints on M and N above are satisfied, micro-orders can overlap the running of a core-store cycle; in fact, overlap is necessary to exploit the high ratio of core-store to ROM cycle times. The core-store indicators JCWA and JCRA are cleared at the beginning of the next core cycle, not at the end of the current cycle: the slowed operational modes have forced this timing, because there could be an indefinite delay after a core cycle is completed before either of these indicators is inspected.

APPENDIX A2

CORE STORE SIMULATION OF ROM

It is desirable to be able to execute microcode from core store for the following two reasons,

(i) new microcode can be checked and modified from core before it is wired (semi-permanently) into ROM;

(ii) microcode can be mixed with machine-code (M/C) orders: this generalizes the M/C repertoire beyond orders which have existing interpretation sequences wired into ROM.

FIGURES 2.1 and 2.2 show the additional paths which allow core simulation of ROM. Register CSA can be copied from S, and read-data in CSD can be copied either to G_{0-15} or to G_{16-31}. The distinction between a normal access to ROM from S and a simulated access via two core-store cycles is marked by the value of S_0, the most significant digit of S: if $S_0 = 0$, the ROM is accessed (denoted a "firm" cycle), whereas if $S_0 = 1$, the pair of core words is accessed (denoted a "soft" cycle). During a "firm" cycle, S_{4-15} addresses the ROM, and the 32-bit micro-order is read directly into G_{0-31}. (Bits S_{1-3} allow for expansion of ROM.) During a "soft" cycle, S_{5-15} is transferred to CSA_{4-14}, the first core cycle ($CSA_{15} = 0$) transfers the upper half of the micro-order to G_{0-15}, and the second core cycle ($CSA_{15} = 1$) transfers the lower half of the micro-order to G_{16-31}.

Soft-mode core-cycles are borrowed. That is, the paths from S to CSA and CSD to G are normally inhibited, but, if $S_0 = 1$, then the pulse which would drive the ROM is diverted; it sets a binary, denoted "soft" (not the binary S_0), which connects S to CSA and CSD to G (the distinction between G_{0-15} and G_{16-31} being made by another binary, denoted "first"). After the second core-word is read into G_{16-31}, "soft" is cleared (S_0 is unchanged) and the C or N to CSA and CSD to M paths are restored in readiness for the normal execution of the micro-order in G (which happened to be read from core). Bit S_0 is controlled by microcode via e or Q which accommodate

bit location 0. Thus, "soft" operation follows automatically for S addresses $\geq 2^{15}$; bits S_{5-15} then refer to the 2K even addresses of core 0,2,..4K-2. As both SQ and QS transfers include S_0, "soft" microprograms can call, via the microroutine stack, microroutines which are either "firm" or "soft". Provided the constraints which were introduced to allow slowed operation (core simulation of ROM is also slowed operation) are observed (APPENDIX A1), then microcode debugged in core can be transplanted directly into ROM. Any "soft" microroutine should also be wired in; this avoids having to change calling addresses wired in ROM, at a later date.

Microcode may be mixed with M/C instructions as follows. A M/C function, FMIC, heads the inserted microcode. FMIC is interpreted in the standard way by a short microprogram accessed via the M/C table; it causes the transfer,

$$2^{15} + RS(C) + 1 \rightarrow S$$

where RS(C) is the instruction address (held in register C) right shifted. Term 2^{15} sets S_0 to 1 and establishes the "soft" mode, and, because S_{5-15} is connected to CSA_{4-14} in "soft" mode (a "wired" left-shift), the transfer simply begins executing microcode in core at the first even address following FMIC. During the "soft" mode execution of microcode, the instruction address register, C, is not advanced as it would be for normal M/C execution; thus, the final micro-order to be executed from core must advance C to the final core address of the microcode (the Get Next Instruction, GNI, sequence always increments the instruction address register before reading the next M/C instruction) and then return control to GNI via a QS transfer. This is achieved by,

$$\text{AQS} \quad S+1 \quad +,0 \quad \overline{0} \quad \overset{LC}{\rightarrow} \quad C$$

(The pre-incremental value, S+1, is cancelled by $\overline{0}$; the left shift component of LC aligns S to the core-store address which must be advanced

at twice the rate of S, and the 1 value which is circulated to the least significant position from S_0 by LC, increments C to the final core address of the microcode.) The implementation of core store simulation of ROM would be simpler if the core store and ROM word-lengths were identical, say 32 bits (excluding parity). However, the difficulties introduced by unequal word-lengths, are invisible to the user.

APPENDIX A3

DETAILED DOCUMENTATION OF MICROPROGRAMS

This appendix documents 47 microprograms, index p A3.2, which are referred to in the text or by other microprograms. Page A3.3 lists the J-INDICATORS which condition micro-orders.

Comments to the microcode are mostly line-by-line with the microcode, but in some cases comments refer to a group of micro-orders, in which case the group is defined by a vertical bar. Labels local to the microprogram have the prefix LL.

The documentation includes the list of microroutines called by each microprogram, the total set of registers needed to execute each microprogram, and the nesting depth. In some cases, the physical requirements of a call was not known at the time of writing - these are marked with an *.

Subscripts are not depressed, i.e., F_8 is shown as F8; and F8,15 represents F_8,F_{15}, etc.

The microprograms which interpret the conventional machine code instructions, i.e., machine codes other than GRA and GRB, are not shown.

TABLE A3.1. INDEX TO MICROCODE ROUTINES AND TABLES

G_{6-11}	MNEMONIC	VALUE
0	JO	'1'
1	JOFU	OF_0
2	JUFU	UF_0
3	JCOU	CO_0
4	JBdo	Bd_0
5	JBeo	Be_0
6	JBe1	Be_1
7	JBe7	Be_7
8	JNZeU	$\overline{Ze}_U$
9	JOFL	OF_8
10	JUFL	UF_8
11	JCOL	CO_8
12	JBd15	Bd_{15}
13	JBe8	Be_8
14	JBe9	Be_9
15	JBe15	Be_{15}

G_{6-11}	MNEMONIC	VALUE
16	JNZeL	$\overline{Ze}_L$
17	JNZe	$\overline{Ze}$
18	JNZR	$\overline{Z}_R$
19	JNNMU	p A3.29
20	JNNML	"
21	JZe	Ze
22	JMSW	Manual Switch
23		
24		
25		
26		
27		
28		
29		
30		
31		

G_{6-11}	MNEMONIC	VALUE
32	JF0	F_0
33	JF1	F_1
34	JF2	.
35	JF3	.
36	JF4	.
37	JF5	.
38	JF6	.
39	JF7	.
40	JF8	.
41	JF9	.
42	JF10	.
43	JF11	.
44	JF12	.
45	JF13	
46	JF14	.
47	JF15	F_{15}

G_{6-11}	MNEMONIC	VALUE
48	JCU0	Control-Unit 0
49	JCU1	" 1
50	JCU2	" 2
51	JCU3	" .
52	JCU4	" .
53	JCU5	" .
54	JCU6	" .
55	JCU7	" 7
56	JINT	Any Interrupt
57	JPIP	Peripheral Input Parity Error
58	JNPSD	Not Peripheral Status/Data
59		
60		
61	JCNB	Core Not Busy
62	JCRA	Core Read Available
63	JCWA	Core Write Accepted

NOTE: INDICATORS 48-63 ARE BUFFERED TO AVOID THE POSSIBILITY OF A CHANGING J AT THE TIME OF INTERROGATION. INDICATORS 0-47 ARE STATIC AT INTERROGATION TIME.

TABLE A3.2. THE 64 J-INDICATORS

DESCRIPTION OF MACHINE CODE FUNCTION TABLE

This table is entered from the GNI (Get Next Instruction) sequence by a DSQ order. It transfers control to the beginning of a specific machine code function routine (M/C 0 to M/C 31) specified by the FN field (bits 0 - 4 of the instruction). M/C routines terminate with Q → S transfers which return control to the GNI sequence.

A preliminary sequence extracts the FN field from the machine code instruction in M (or the first word of the instruction for multiple-word instructions) and aligns it to bits 11-15 of register D. The M/C function number in D is then added to S+1; this advances S to the appropriate DMC order which transfers control to the specific M/C routine.

INITIAL VALUES

M: THE MACHINE CODE INSTRUCTION WORD.

Q: RETURN ADDRESS TO THE GNI ROUTINE.

FINAL VALUES

M: AS INITIAL.

Q: AS INITIAL.

CALLS - M/C's 0-31.

SELF NEEDS D, M, R.

NO. OF MICRO-ORDERS 37.

MICROCODE

											COMMENTS
1	TABLE M/C FN:	ABA					M	(8T)	D		PUT FN FIELD TO D8-12
2		DMC(U)	0					→	D		CLEAR DU
3		DMC	3					→	R		PUT 3→R FOR SHIFT COUNT
4		ADR	JNZR (RP)				D	(RS)	D		RIGHT SHIFT D 3 PLACES TO ALIGN M/C FN NUMBER TO D11-15 FOR ADDITION TO S
5		ABA			S+1	+0	D	→	S		ADVANCE D+1 ORDERS
6		DMC		M/C 0				→	S		IF M/C = 0 (i.e., IF CONTENTS D = 0), GO TO M/C 0
7		DMC		M/C 1				→	S	1	1 . M/C 1
.		.		.				.	.	.	. .
.		.		.				.	.	.	. .
37		DMC		31				→	S	31	31 GO TO M/C 31.

DESCRIPTION OF MACHINE CODE FUNCTION "GRAPHICS-A"

GRAPHICS-A (GRA) is the first of two machine code functions designed specifically for graphics. GRA is divided into 8 sub-functions (GRAFN's) introduced on p 4.4. A GRA instruction (header) occupies one word, FORMAT 4.2, p 4.4; the relevant data words immediately follow the header.

This routine puts RST, LF, IT, & MG (bits 8-15 of the GRA instruction) from ML to WU, copies the current display register contents X,Y to buffers X',Y', puts the GRAFN field to D13-15 in order to enter the table proper which then transfers control to the beginning of the specific GRAFN encoded. M/C GRA is entered via a DMC order in TABLE M/C FN and M/C GRA then transfers control, also via a DMC order, to the specific GRA routine; return to GNI is the responsibility of the terminating order of each GRA routine.

INITIAL VALUES

M: THE MACHINE CODE INSTRUCTION WORD.

Q: RETURN ADDRESS TO GNI ROUTINE.

FINAL VALUES

M: AS INITIAL

Q: AS INITIAL

WU: RST, LF, IT, MG.

X',Y': X,Y.

CALLS - GRA's 0-7 (SML CTN VTRS -- LRG PLR PNTS).

SELF NEEDS B, D, M, W.

NO. OF MICRO-ORDERS 13.

MICROCODE

1	M/C GRA:	ABA(L)				M	(8T)	W	COPY RST, LF, IT AND MG FIELDS INTO WU.
2		TRF	X→X',Y→Y'						COPY CURRENT X,Y VALUES INTO X',Y' (REQUIRED FOR RST OPTION WITHIN GRAFN's)
3		DMC	X'0700'				→	B	PUT GRAFN SELECTION FIELD IN B.
4		ABA		B	Λ	M	(8T)	D	SELECT GRAFN AND PLACE IN D13-15.
5		ABA		S+1	+0	D	→	S	ADVANCE D+1 ORDERS.
6	(GRA TABLE BEGINS)	DMC	GRA SML CTN VTRS				→	S	IF GRAFN = 0 (i.e., IF CONTENTS D = 0), GO TO GRA SML CTN VTRS
7		DMC	GRA SML CTN PNTS				→	S	1 " 1 . . .
.		.	. . . .				.	.	. . . .
.		.	. . . .				.	.	. . . .
13		DMC	GRA LRG CTN PNTS				→	S	7 7 GO TO GRA LRG PLR PNTS.

DESCRIPTION OF MACHINE CODE FUNCTION "GRAPHICS-B"

GRAPHICS-B (GRB), the second of two machine code functions designed for graphics, is divided into 16 sub-functions (GRBFN's).

As GRA, the GRB instruction (header) occupies one word and the relevant data immediately follows. FORMAT 4.6, p 4.10 applies to GRB.

This routine puts bits 9-15 of the GRB instruction word from M9-15 to W1-7, copies the current X,Y values into X',Y', puts the GRBFN field to D12-15 in order to enter the table proper which then transfers control to the specific GRBFN encoded.

M/C GRB is entered via a DMC order in TABLE M/C FN and M/C GRB then transfers control, also via a DMC order, to the specific GRB routine; return to GNI is the responsibility of the terminating order of each GRB routine.

INITIAL VALUES

M: THE MACHINE CODE INSTRUCTION WORD.
Q: RETURN ADDRESS TO GNI ROUTINE.

FINAL VALUES

M: AS INITIAL
Q: AS INITIAL.
W: BITS 9-15 OF GRB INSTRUCTION.
X',Y': X,Y.

CALLS - GRB's 0-15 (SML XTHENY VTRS TO EXTRACODE).

SELF NEEDS B, D, M, W.

NO. OF MICRO-ORDERS 22.

MICROCODE

1	M/C GRB:	ABA(L)				M	(8T)	W	COPY BITS 9-15 OF GRB INSTRUCTION TO W1-7 (W0 NOT REFERRED TO BY GRBFN's)
2		TRF	X→X',Y→Y'						COPY CURRENT X,Y VALUES INTO X',Y' (REQUIRED FOR SOME GRB RST OPTIONS).
3		DMC	X'0780'				→	B	PUT GRBFN SELECTION FIELD IN B.
4		ABA		B	Λ	M	(LS)	D	SELECT GRBFN FIELD AND ALIGN TO D4-7.
5		ABA				D	(8T)	D	PUT GRBFN IN D12-15.
6		ABA		S+1	+0	D	→		ADVANCE D+1 ORDERS.
7	(GRB TABLE BEGINS)	DMC	GRB SML XTHENY VTRS				→	S	IF GRBFN = 0 (i.e., CONTENTS OF D = 0), GO TO GRB SML XTHENY VTRS.
8		DMC	GRB SML XRST IY VTRS				→	S	" 1 1 .
.		.	.					.	. . .
.		.	.					.	. . .
22		DMC	GRB EXTRACODE				→	S	15 15 GO TO GRB EXTRACODE.

DESCRIPTION OF GRAPHICS-A FUNCTION 'SMALL CARTESIAN VECTORS' (GRAFN 0)

This routine plots a list of small Cartesian vectors coded in FORMAT 4.1, p 4.3. The vector list immediately follows a GRA HEADER (FORMAT 4.2, p 4.4). The upper byte of the header identifies the particular routine (i.e., in this case, GRAFN field contains 0); the lower byte specifies the reset option, line form, intensity level and magnification exponent.

The magnification exponent MG extends the otherwise restricted scope of the small format. At maximum magnification (X8), the range of horizontal or vertical components is

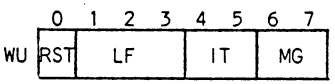
-512 (8) 504 display increments,

which gives ample resolution and scope for many applications. (If 1 display increment = 0.01 in, plotting at X8 is similar to drawing lines terminating at the grid intersection points of graph paper having about 12 divisions/in. Note that the fine texture of vectors is independent of MG.)

Preloading Q to LL LOOP conditionally on L and the use of MI RST; INTRPT;LOOP are detailed under MI RST;INTRPT;LOOP.

INITIAL VALUES

C: IA, THE INSTRUCTION ADDRESS (ADDRESS OF GRA HEADER)
(IA+1) IN CORE: FIRST VECTOR
(IA+2) " " : SECOND VECTOR, ETC.
Q: RETURN ADDRESS TO GNI ROUTINE
WU: BITS 8-15 OF GRAFN HEADER



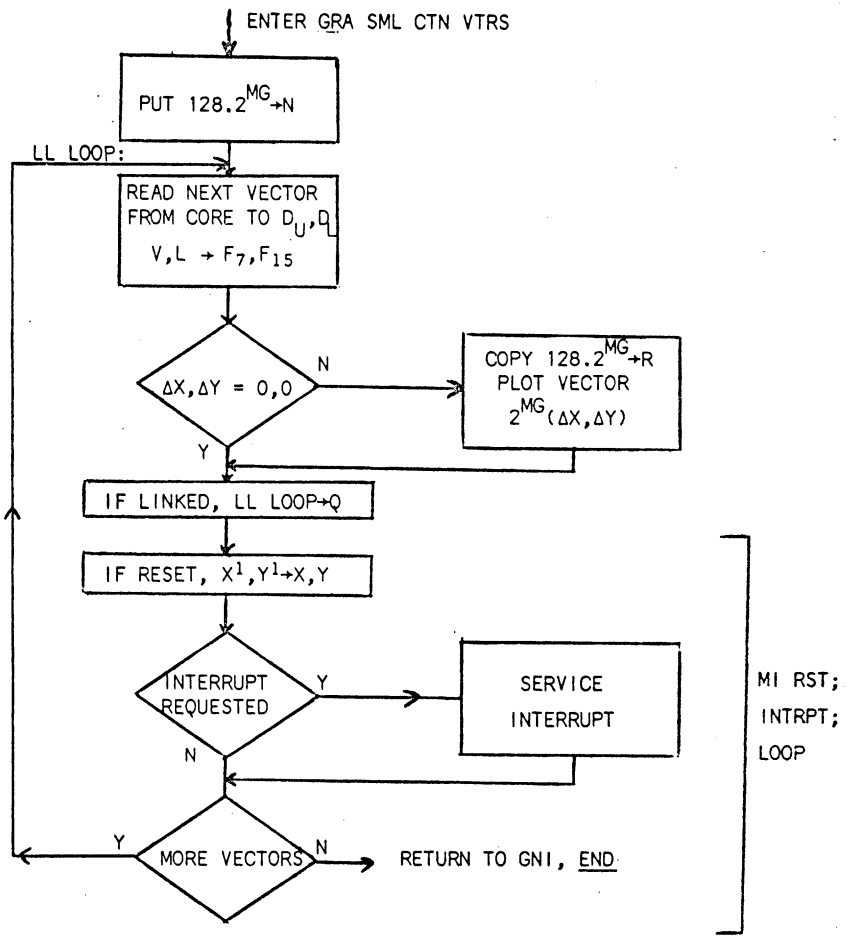
X^1,Y^1: DISPLAY CO-ORDINATES PRECEDING GRA MACHINE ORDER

FINAL VALUES

C: ADDRESS OF FINAL VECTOR IN LIST
(IA+1), ETC.: RESTORED
X,Y: IF RESET, INITIAL VALUES, ELSE INITIAL + $\Sigma\Delta X,\Sigma\Delta Y$
X^1,Y^1: AS INITIAL

NEEDS

CALLS	NEEDS	DEPTH
MI 128*2**MG→N	N,R,WU	NIL
MI READ SML CRD→D	C,D,F7,15,M	NIL
MI PLT SML CTN VTR D	B,D,R	NIL
MI RST;INTRPT;LOOP	Wo,*	2Q
ADDITIONAL FOR SELF	Q	1Q
TOTAL NEEDS & DEPTH	B,C,D,F7,15,M,N,Q,R,W,*	3Q



MICROCODE

1	GRA SML CTN VTRS:	DSQ		MI 128*2**MG->N	→	S
2	LL LOOP:	DSQ		MI READ SML CRD->D	→	S
3		ABA	JNZe	N	→	R
4		DSQ	JNZe	MI PLT SML CTN VTR D	→	S
5		DMC	JF15	LL LOOP	→	Q
6		DMC		MI RST;INTRPT;LOOP	→	S

COMMENTS

PUTS 128.2^{MG} IN N FOR CYCLE COUNT OF MI PLT SML CTN VTR D. (HELD FOR ENTIRE LIST)

INCREMENTS IA IN C, READS VTR CO-ORDINATES TO DU,DL,

RECORDS V,L (NOTE Ze = 1 IF $\Delta X,\Delta Y = 0,0$)

IF $\Delta X,\Delta Y \neq 0,0$ THEN → PUT 128.2^{MG}→R

ELSE OMIT 3,4 → PLOT MAGNIFIED VECTOR $2^{MG}(\Delta X,\Delta Y)$

IF LINKED, PRELOAD Q TO LL LOOP FOR MI RST;INTRPT;LOOP

RESET DISPLAY IF REQUESTED; SERVICE INTERRUPT; RETURN CONTROL TO LL LOOP OR RETURN TO GNI & END

DESCRIPTION OF GRAPHICS-A FUNCTION 'SMALL CARTESIAN POINTS' (GRAFN 1)

Plots a list of points coded in small Cartesian format (FORMAT 4.1, p 4.3 immediately following a GRA HEADER (FORMAT 4.2, p 4.41. The lower byte specifies RST, IT, & MG as detailed under FORMAT 4.2, but LF does not apply.

MI PLT SML CTN PNT D is called even if ΔX,ΔY = 0,0 (i.e., if ΔX,ΔY = 0,0, the point at the last X,Y position is unblanked; this is not so for GRA SML CTN VTRS - a zero vector is bypassed and the display remains blanked).

Preloading Q to LL LOOP conditionally on L and the use of MI RST;INTRPT;LOOP are detailed under MI RST;INTRPT;LOOP.

INITIAL & FINAL VALUES

AS GRAFN SML CTN VTRS

MICROCODE

1	GRAFN SML CTN PNTS:	DMC		X'0300'	→	N
2		ABA		N Λ W (8T)		N
3	LL LOOP:	DSQ		MI READ SML CRD->D	→	S
4		ABA		N	→	R
5		DSQ		MI PLT SML CTN PNT D	→	S
6		DMC	JF15	LL LOOP	→	Q
7		DMC		MI RST;INTRPT;LOOP	→	S

NEEDS

CALLS	NEEDS	DEPTH
MI READ SML CRD→D	C,D,F7,15,M	NIL
MI PLT SML CTN PNT D	D,E,R	NIL
MI RST;INTRPT;LOOP	Wo *	2Q
ADDITIONAL FOR SELF	N,Q,W	1Q
TOTAL NEEDS & DEPTH	C,D,E,F7,15,M,N,Q,R,W *	3Q

NO. OF MICRO-ORDERS 7

COMMENTS

PUT MG SELECTION FIELD IN N (MG IN W6,7)
PUT MG IN N (HELD FOR ENTIRE LIST OF POINTS)
INCREMENTS IA IN C, READS POINT CO-ORDINATES TO DU,DL, RECORDS V,L.
PUT MG EXPONENT IN R FOR MI PLT SML CTN PNT D
PLOTS POINT AT 2^{MG} (ΔX,ΔY) RELATIVE TO CURRENT X,Y
IF LINKED, PRELOAD Q TO LL LOOP FOR MI RPT;INTRPT;LOOP
RESET DISPLAY IF REQUESTED; SERVICE INTERRUPT; RETURN CONTROL TO LL LOOP OR RETURN TO GNI & END

DESCRIPTION OF GRAPHICS-A FUNCTION 'SMALL POLAR VECTORS' (GRAFN 2)

Plots a list of small polar vectors (FORMAT 4.3, p 4.5) immediately following a GRA HEADER (FORMAT 4.2, p 4.4).

The polar co-ordinates are read from core, $\Delta\theta$ updates θ in A and COS,SIN ($\theta + \Delta\theta$) are evaluated and placed in DU,DL (LSB's = 2^{-6}). Distance Δs is then loaded into R (MG is not extended to polar co-ordinate, because errors in COS θ, SIN θ would be magnified) to count cycles within MI PLT SML CTN VTR D, and, if $\Delta s \neq 0$, the equivalent Cartesian vector Δs COS θ, Δs SIN θ is plotted.

Preloading Q to the beginning of the vector plotting loop (in this case GRA SML PLR VTRS itself) conditionally on L and the use of MI RST;INTRPT;LOOP are detailed under MI RST;INTRPT;LOOP.

INITIAL & FINAL VALUES

AS GRA SML CTN VTRS (MG = XI)

MICROCODE

1	GRA SML PLR VTRS:	DSQ		MI READ SML PLR->E,N	→	S
2		DSQ		MI COS,SIN(A+E)->D	→	S
3		ABA		N	(LS)	R
4		DSQ	JNZe	MI PLT SML CTN VTR D	→	S
5		DMC	JF15	GRA SML PLR VTRS	→	Q
6		DMC		MI RST;INTRPT;LOOP	→	S

NEEDS

CALLS	NEEDS	DEPTH
MI READ SML PLR→E,N	C,D,E,F7,15,M,N	1Q
MI COS,SIN(A+E)→D	A,B,D,E,F8	2Q
MI PLT SML CTN VTR D	B,D,R	NIL
MI RST;INTRPT;LOOP	Wo,*	2Q
ADDITIONAL FOR SELF	Q,WU	1Q
TOTAL NEEDS & DEPTH	A,B,C,D,E,F7,8,15,M,N,Q,R,W*	3Q

NO. OF MICRO-ORDERS 6

COMMENTS

INCREMENTS IA IN C, READS $\Delta\theta$ TO E & ALIGNS TO θ IN A, Δs TO N, RECORDS V,L

UPDATES θ IN A, PUTS COS θ → DU, SIN θ → DL (2^6COS, 2^6SIN AS INTEGERS)

PUT $2\Delta s$ IN R FOR MI PLT SML CTN VTR D

IF $\Delta s \neq 0$, PLOT VECTOR $2\Delta s.2^{-7}.2^6$(COS,SIN), i.e., (Δs COS θ, Δs SIN θ)

IF LINKED, PRELOAD Q TO BEGINNING OF THIS GRAFN

RESET IF REQUESTED; SERVICE INTERRUPT; RETURN CONTROL TO ORDER 1 OR RETURN TO GNI & END

DESCRIPTION OF GRAPHICS-A FUNCTION 'SMALL POLAR POINTS' (GRAFN 3)

Plots a list of points coded in small polar format (FORMAT 4.3, p 4.5) immediately following a GRA HEADER (FORMAT 4.2, p 4.4).

The polar co-ordinates are read from core; Δθ updates θ in A; COS θ, SIN θ are formed in DU,DL and then overwritten by ΔsCOS θ,ΔsSIN θ; and the point is plotted (MG = X1) unconditionally on Δs = 0.

Preloading Q and the use of MI RST;INTRPT;LOOP are detailed under MI RST;INTRPT;LOOP.

INITIAL & FINAL VALUES

AS GRAFN SML CTN VTRS

NEEDS

CALLS	NEEDS	DEPTH
MI READ SML PLR→E,N	C,D,E,F7,15,M,N	1Q
MI COS,SIN(A+E)→D	A,B,D,E,F8	2Q
MI N*COS,N*SIN→D	B,D,E,N,R	1Q
MI PLT SML CTN PNT D	D,E,R	NIL
MI RST;INTRPT;LOOP	Wo *	2Q
ADDITIONAL FOR SELF	Q,WU	1Q
TOTAL NEEDS & DEPTH	A,B,C,D,E,F7,8,15,M,N,Q,R,W*	3Q

NO OF MICRO-ORDERS 7

MICROCODE

1	GRA SML PLR PNTS:	DSQ		MI READ SML PLR->E,N	→	S
2		DSQ		MI COS,SIN(A+E)->D	→	S
3		DSQ		MI N*COS,N*SIN->D	→	S
4		DMC		0	→	R
5		DSQ		MI PLT SML CTN PNT D	→	S
6		DMC	JF15	GRA SML PLR PNTS	→	Q
7		DMC		MI RST;INTRPT;LOOP	→	S

COMMENTS

INCREMENTS IA IN C, READS Δθ TO E & ALIGNS TO θ IN A, Δs TO N, RECORDS V,L

UPDATES θ IN A, PUTS COS θ → DU, SIN θ → DL (2^6COS,2^6SIN AS INTEGERS)

MULTIPLIES Δs BY COS,SIN, i.e., ΔX → DU, ΔY → DL

CLEAR R (MG = X1 FOR POLAR CO-ORDINATES)

PLOTS POINT AT ΔsCOS θ, ΔsSIN θ RELATIVE TO CURRENT X,Y

IF LINKED, PRELOAD Q TO BEGINNING OF THIS GRAFN

RESET IF REQUESTED; SERVICE INTERRUPT; RETURN CONTROL TO ORDER 1 OR RETURN TO GNI & END

DESCRIPTION OF GRAPHICS-A FUNCTION 'LARGE CARTESIAN VECTORS' (GRAFN 4)

Plots a list of large Cartesian vectors (FORMAT 4.4, p 4.7) immediately following a GRA HEADER (FORMAT 4.2, p 4.4).

The vector components Xc,Yc are read from core and, if Xc,Yc ≠ 0,0, plotted (MI PLT LRG CTN VTR begins with a zero-vector test).

Preloading Q and the use of MI RST;INTRPT;LOOP are detailed under MI RST;INTRPT;LOOP.

INITIAL VALUES

C: IA THE INSTRUCTION ADDRESS (ADDRESS OF GRA HEADER)
(IA+1) IN CORE: FIRST HORIZ. COMPONENT Xc & V
(IA+2) " " : " VERT. " Yc & L
ETC. IN PAIRS
Q: RETURN ADDRESS TO GNI ROUTINE
WU: BITS 8-15 OF GRA HEADER
X^1,Y^1: DISPLAY CO-ORDINATES PRECEDING GRA MACHINE ORDER

MICROCODE

1	GRA LRG CTN VTRS:	DSQ		MI READ LRG CRD->E,A	→	S
2		DSQ		MI PLT CTN VTR	→	S
3		DMC	JF15	GRA LRG CTN VTRS	→	Q
4		DMC		MI RST;INTRPT;LOOP	→	S

FINAL VALUES

C: ADDRESS OF LAST Yc,L WORD
(IA+1), ETC: RESTORED
X,Y: IF RESET, INITIAL VALUES, ELSE INITIAL + ΣXc,ΣYc.
X^1,Y^1: AS INITIAL

NEEDS

CALLS	NEEDS	DEPTH
MI READ LRG CRD→E,A	A,C,E,F7,15,M	NIL
MI PLT LRG CTN VTR	A,B,D,E,R	1Q
MI RST;INTRPT;LOOP	Wo *	2Q
ADDITIONAL FOR SELF	Q,W	1Q
TOTAL NEEDS & DEPTH	A,B,C,D,E,F7,15,M,Q,R,W *	3Q

NO. OF MICRO-ORDERS 4

COMMENTS

READS LARGE VECTOR Xc,Yc → E,A & V,L → F7,F15
PLOTS VECTOR Xc,Yc IF NON-ZERO
IF LINKED, PRELOAD Q TO BEGINNING OF THIS GRAFN
RESET IF REQUESTED; SERVICE INTERRUPT; RETURN CONTROL TO ORDER 1 OR RETURN TO GNI & END

DESCRIPTION OF GRAPHICS-A FUNCTION 'LARGE CARTESIAN POINTS' (GRAFN 5)

Plots a list of points in large Cartesian format (FORMAT 4.4, p 4.7) immediately following a GRA HEADER (FORMAT 4.2, p 4.4).

The point co-ordinates (relative to the current display co-ordinates X,Y) are read from core, added to the current X,Y position and plotted (unconditionally on Xc,Yc = 0,0).

Preloading Q and the use of MI RST;INTRPT;LOOP are detailed under MI RST;INTRPT;LOOP.

INITIAL & FINAL VALUES

AS GRA LRG CTN VTRS

MICROCODE

1	GRA LRG CTN PNTS:	DSQ		MI READ LRG CRD->E,A	→	S
2		DSQ		MI PLT LRG CTN PNT	→	S
3		DMC	JF15	GRA LRG CTN PNTS	→	Q
4		DMC		MI RST;INTRPT;LOOP	→	S

NEEDS

CALLS	NEEDS	DEPTH
MI READ LRG CRD→E,A	A,C,E,F7,15,M	NIL
MI PLT LRG CTN PNT	A,D,E	NIL
MI RST;INTRPT;LOOP	Wo *	2Q
ADDITIONAL FOR SELF	Q,W	1Q
TOTAL NEEDS & DEPTH	A,C,D,E,F7,15,M,Q,W *	3Q

NO. OF MICRO-ORDERS 4

COMMENTS

READS Xc,Yc → E,A & V,L → F7,F15
PLOTS POINT Xc,Yc RELATIVE TO CURRENT X,Y
IF LINKED, PRELOADS Q TO BEGINNING OF THIS GRAFN
RESET IF REQUESTED; SERVICE INTERRUPT; RETURN CONTROL TO ORDER 1 OR RETURN TO GNI & END

DESCRIPTION OF GRAPHICS-A FUNCTION 'LARGE POLAR VECTORS' (GRAFN 6)

Plots a list of large polar vectors (FORMAT 4.5, p 4.9) immediately following a GRA HEADER (FORMAT 4.2, p 4.4).

The vector components R,θ are read from core, converted to Cartesian co-ordinates and, if R COS θ, R SIN θ ≠ 0,0, plotted.

Preloading Q and the use of MI RST;INTRPT;LOOP are detailed under MI RST;INTRPT;LOOP.

INITIAL & FINAL VALUES

AS GRA LRG CTN VTRS

NEEDS

CALLS	NEEDS	DEPTH
MI READ LRG CRD→E,A	A,C,E,F7,15,M	NIL
CQ LRG PLR→CTN E,A	A,B,D,E,F8,9,M,N,R	3Q
MI PLT LRG CTN VTR	A,B,D,E,R	1Q
MI RST;INTRPT;LOOP	Wo *	2Q
ADDITIONAL FOR SELF	Q,W	1Q, CQ
TOTAL NEEDS & DEPTH	A,B,C,D,E,F7,8,9,15,M,N,Q,R,W,*3Q, 1CQ	

NO. OF MICRO-ORDERS 7

MICROCODE

1	GRA LRG PLR VTRS:	DSQ		MI READ LRG CRD->E,A	→	S
2		ABA		A	CLS1	A
3		DSQ		MI PUSH CQ	→	S
4		DMC		CQ LRG PLR->CTN E,A	→	S
5		DSQ		MI PLT LRG CTN VTR	→	S
6		DMC	JF15	GRA LRG PLR VTRS	→	Q
7		DMC		MI RST;INTRPT;LOOP	→	S

COMMENTS

READS R,θ TO E,A (MSB θ = -2π) & V,L TO F7,F15

ALIGN θ IN A TO MSB = -π

PUSH CQ FOR CQ ROUTINE FOLLOWING

CONVERTS R,θ TO Xc,Yc IN E,A

PLOTS VECTOR R COS θ, R SIN θ IF NON-ZERO

IF LINKED, PRELOAD Q TO BEGINNING OF THIS GRAFN

RESET IF REQUESTED; SERVICE INTERRUPT; RETURN CONTROL TO ORDER 1 OR RETURN TO GNI & END

DESCRIPTION OF GRAPHICS-A FUNCTION 'LARGE POLAR POINTS' (GRAFN 7)

Plots a list of points in large polar format (FORMAT 4.5, p 4.9) immediately following a GRA HEADER (FORMAT 4.2, p 4.4).

The point co-ordinates are read from core, added to the current X,Y position and plotted (unconditionally on R COS θ, R SIN θ = 0,0).

Preloading Q and the use of MI RST;INTRPT;LOOP are detailed under MI RST;INTRPT;LOOP.

INITIAL & FINAL VALUES

AS GRA LRG CTN VTRS

NEEDS

CALLS	NEEDS	DEPTH
MI READ LRG CRD→E,A	A,C,E,F7,15,M	NIL
CQ LRG PLR→CTN E,A	A,B,D,E,F8,9,M,N,R	3Q
MI PLT LRG CTN PNT	A,D,E	NIL
MI RST;INTRPT;LOOP	Wo *	2Q
ADDITIONAL FOR SELF	Q,W	1Q, CQ
TOTAL NEEDS & DEPTH	A,B,C,D,E,F7,8,9,15,M,N,Q,R,W*	3Q,1CQ

NO. OF MICRO-ORDERS 7

MICROCODE

1	GRA LRG PLR PNTS:	DSQ		MI READ LRG CRD->E,A	→	S
2		ABA		A	(LS)	A
3		DSQ		MI PUSH CQ	→	S
4		DMC		CQ LRG PLR->CTN E,A	→	S
5		DSQ		MI PLT LRG CTN PNT	→	S
6		DMC	JF15	GRA LRG PLR PNTS	→	Q
7		DMC		MI RST;INTRPT;LOOP	→	S

COMMENTS

READS R,θ TO E,A (MSB θ = -2π1 & V,L TO F7,F15

ALIGN θ IN A TO MSB = -π

PUSH CQ FOR CQ ROUTINE FOLLOWING

CONVERTS R,θ TO Xc,Yc IN E,A

PLOTS POINT R COS θ, R SIN θ RELATIVE TO CURRENT X,Y

IF LINKED, PRELOAD Q TO BEGINNING OF THIS GRAFN

RESET IF REQUESTED; SERVICE INTERRUPT; RETURN CONTROM TO ORDER 1 OR RETURN TO GNI & END

DESCRIPTION OF GRAPHICS-B FUNCTION 'SMALL X THEN Y VECTORS' (GRBFN 01)

Small X then Y vectors are shown in FIGURE 4.5, p 4.10. Q
The two sides ΔX & ΔY of a right-angled triangle are plotted instead of the hypotenuse, the order being ΔX then ΔY. Sequences of alternate horizontal and vertical lines are particularly useful when plotting block diagrams. GRB HEADER fields LF, IT and MG are applicable, and the comments on MG under GRA SML CTN VTRS are relevant.

The vectors, coded in FORMAT 4.1, p 4.3 immediately follow a GRB HEADER shown in FORMAT 4.6, p 4.10.

INITIAL & FINAL VALUES

AS GRA SML CTN VTRS, EXCEPT X THEN Y VTRS ARE NOT RESET

NEEDS

CALLS	NEEDS	DEPTH
MI 128*2**MG→N	N,R,W,	NIL
MI READ SML CRD→D	C,D,F7,15,M	NIL
MI PLT SML CTN VTR D	B,D,R	NIL
MI INTRPT	*	1Q
ADDITIONAL FOR SELF	E	1Q
TOTAL NEEDS & DEPTH	B,C,D,E,F7,15,M,N,R,W*	2Q

NO. OF MICRO-ORDERS 14

MICROCODE

								COMMENTS	
1	GRB SML XTHENY VTRS:	DSQ		MI 128*2**MG->N	→	S		PUTS 128.2^{MG}→N FOR CYCLE COUNT OF MI PLT SML CTN VTR D	CONSTANTS FOR ENTIRE LIST
2		DMC(U)		0	→	E		CLEAR EU	
3	LL LOOP	DSQ		MI READ SML CRD->D	→	S		INCREMENTS IA IN C, READS ΔX,ΔY → DU,DL, RECORDS V,L	
4		ABA(L)		D	→	E		PUTS ΔY IN EL (E NOW HOLDS 0,ΔY)	
5		DMC(L)		0	→	D		CLEAR DL (D NOW HOLDS ΔX,0)	
6		ABA		N	→	R		PUT 128.2^{MG} IN R	
7		ABA		D	→	NIL		PREPARE Ze FOR VECTOR ΔX,0 ZERO TEST	
8		DSQ	JNZe	MI PLT SML CTN VTR D	→	S		IF ΔX ≠ 0, PLOT ΔX VECTOR	
9		ABA		N	→	R		PUT 128.2^{MG} IN R	
10		ABA		E	→	D		PUT 0,ΔY → DU,DL & PREPARE Ze	
11		DSQ	JNZe	MI PLT SML CTN VTR D	→	S		IF ΔY ≠ 0, PLOT ΔY VECTOR	
12		DSQ	JINT	MI INTRPT	→	S		IF INTERRUPT REQUEST, INSERT MI INTRPT	
13		DMC	JF15	LL LOOP	→	S		IF LINKED, RETURN CONTROL TO LL LOOP	
14		DQS				NIL		ELSE RETURN TO GNI, END	

DESCRIPTION OF GRAPHICS-B FUNCTION 'SMALL X VECTORS RESET THEN INCREMENT Y' (GRBFN 1)

FIGURE 4.6, p 4.11 shows this mode of plotting. The mode is useful for plotting the horizontal lines of tables: the lines may be aligned on the left (ΔX>0) as shown in FIGURE 4.6 or on the right (ΔX<0), each line having a specified length and separation to the next line. The comments on MG under GRA SML CTN VTRS are applicable.

INITIAL & FINAL VALUES

AS GRA SML CTN VTRS WITH X RESET, Y NOT RESET

NEEDS

AS GRB SML XTHENY VTRS

NO. OF MICRO-ORDERS 16

MICROCODE

											COMMENTS
1	GRB SML XRST IY VTRS:	DSQ		MI 128*2**MG->N				→	S		PUTS 128.2^{MG}→N FOR CYCLE COUNT OF MI PLT SML CTN VTR D | CONSTANT FOR
2	LL LOOP:	DSQ		MI READ SML CRD->D				→	S		INCREMENTS IA IN C, READS ΔX,ΔY → DU,DL, RECORDS V,L | ENTIRE LIST
3		ABA					D	→	E		PUT ΔY IN EL
4		DMC(U)		0				→	E		CLEAR EU, E NOW HOLDS 0,ΔY
5		DMC(L)		0				→	D		CLEAR DL, D NOW HOLDS ΔX,0
6		BBA			N			→	R		PUT 128.2^{MG} IN R
7		ABA					D	→	NIL		PREPARE Ze FOR VECTOR ΔX,0 ZERO TEST
8		DSQ	JNZe	MI PLT SML CTN VTR			D	→	S		IF ΔX ≠ 0, PLOT ΔX VECTOR
9		TRF		X^1→X							RESET X TO X^1
10		DMC		X'0300'				→	B		PUT MG SELECTION FIELD IN B (MG IN W6,7)
11		ABA			B	Λ	W	(8T)	R		PUT MG IN R TO MAGNIFY ΔY
12		ADR	JNZR (RP)				E	(LS)	E		FORM 2^{MG}ΔY IN E
13		ABA			Y	+0	E	→	Y		INCREMENT Y BY 2^{MG}ΔY (POINT NOT PLOTTED)
14		DSQ	JINT	MI INTRPT				→	S		IF INTERRUPT REQUEST, INSERT MI INTRPT
15		DMC	JF15	LL LOOP				→	S		IF LINKED, RETURN CONTROL TO LL LOOP
16		DQS							NIL		ELSE RETURN TO GNI, END

DESCRIPTION OF GRAPHICS-B FUNCTION 'SMALL Y VECTORS RESET THEN INCREMENT X' (GRBFN 2)

This routine is similar to GRB SML XRST IY VTRS with X and Y interghanged. There are minor differences in the microcode; the code is set out in full.

INITIAL & FINAL VALUES

AS GRA SML CTN VTRS WITH Y RESET, X NOT RESET

MICROCODE

1	GRB SML YRST IX VTRS:	DSQ		MI 128*2**MG->N			→	S	
2	LL LOOP	DSQ		MI READ SML CRD->D			→	S	
3		ABA				D	(8T)	E	PUT ΔX IN EL
4		DMC(U)		0			→	E	E NOW HOLDS 0,ΔX
5		DMC(U)		0			→	D	CLEAR DU, D NOW HOLDS 0,ΔY
6		ABA			N		→	R	
7		ABA				D	→	NIL	PREPARE FOR 0,ΔY ZERO VECTOR REST
8		DSQ	JNZe	MI PLT SML CTN VTR		D	→	S	IF ΔY ≠ 0, PLOT ΔY VECTOR
9		TRF		$Y^1 \rightarrow Y$					RESET Y TO Y^1
11		DMC		X'0300'			→	B	
11		ABA			B Λ	W	(8T)	R	
12		ADR	JNZR (RP)			E	(LS)	E	FORM $2^{MG}\Delta X$ IN E
13		ABA			X +0 ·	E	→	X	INCREMENT X BY $2^{MG}\Delta X$
14		DSQ	JINT	MI INTRPT			→	S	
15		DMC	JF15	LL LOOP			→	S	
16		DQS						NIL	

NEEDS

AS GRB SML XTHENY VTRS

NO. OF MICRO-ORDERS 16

COMMENTS (DIFFERENCES FROM GRB SML XRST VTRS IY ONLY)

DESCRIPTION OF GRAPHICS-B FUNCTION 'SMALL REPEATED POLAR VECTORS' (GRBFN 3)

FIGURE 4.7, p 4.12 shows a repeated $\Delta\theta,\Delta s$ vector. This GRBFN plots a list of repeated vectors; i.e., for small $\Delta\theta$ and Δs, plots a set of linked circular arcs of different curvatures and total swept angles.

Following each small polar vector word (FORMAT 4.3, p 4.5) is a repeat insert - a positive integer word which specifies the total number of occurrences of the vector. These word pairs immediately follow a GRB HEADER (FORMAT 4.6, p 4.10).

This description should be read in conjunction with GRA SML PLR VTRS.

INITIAL & FINAL VALUES

AS GRA SML CTN VTRS REPEAT INSERT, VECTORS NOT RESET & MG = X1 ONLY

NEEDS

CALLS	NEEDS	DEPTH
MI READ SML PLR→E,N	C,D,E,F7,15,M,N	1Q
MI COS,SIN(A+E)→D	A,B,D,E,F8	2Q
MI PLT SML CTN VTR D	B,D,R	NIL
MI INTRPT	*	1Q
ADDITIONAL FOR SELF	W	1Q
TOTAL NEEDS & DEPTH	A,B,C,D,E,F7,15,M,N,R,W,*	3Q

NO. OF MICRO-ORDERS 10

MICROCODE

1	GRB SML RPT PLR VTRS:	DSQ		MI READ SML PLR->E,N	→	S
2		ACC(RR)	JCNB (WT)	+1 C	→	C
3	LL RPT LOOP:	DSQ		MI COS,SIN(A+E)->D	→	S
4		ABA		N	(LS)	R
5		DSQ	JNZe	MI PLT SML CTN VTR D	→	S
6		DSQ	JINT	MI INTRPT	→	S
7		ABA	JCRA	$\overline{0}$ +0 M	→	M
8		DMC	JNZe	LL RPT LOOP	→	S
9		DMC	JF15	GRB SML RPT PLR VTRS	→	S
10		DQS				NIL

COMMENTS

INCREMENTS IA IN C, READS $\Delta\theta$ TO E & ALIGNS TO θ IN A, Δs TO N, RECORDS V,L
" " , READS REPEAT INSERT TO M
UPDATES θ IN A, PUTS COS θ → DU, SIN θ → DL
PUT $2\Delta s$ IN R FOR MI PLT SML CTN VTR
IF $\Delta s \neq 0$, PLOT VECTOR
IF INTERRUPT REQUEST, INSERT MI INTRPT
DECREMENT REPEAT BY 1 PREPARE Ze FOR ZERO TEST
IF MORE COPIES, LOOP & PLOT AGAIN
IF LINKED, RETURN TO BEGINNING OF THIS GRBFN
ELSE RETURN TO GNI, END

SIGNIFICANCE OF BIT WEIGHTS AS GRAFN SML PLR VTRS

DESCRIPTION OF GRAPHICS-B FUNCTION 'LARGE X OR Y VECTORS' (GRBFN 41

By restricting a vector to be either horizontal or vertical, vector coding can be compressed 2:1. Many plotting applications use mostly horizontal or vertical lines. The data words for this routine (FORMAT 4.7, p 4.13) are individually tagged as horizontal or vertical and reset or not; they also contain V and L bits. The vectors immediately follow a GRB HEADER.

INITIAL VALUES

C: IA THE INSTRUCTION ADDRESS (ADDRESS OF GRB HEADER)
(IA+1) ETC. IN CORE: X OR Y VECTOR WORDS
Q: RETURN ADDRESS IN GNI ROUTINE

FINAL VALUES

C: ADDRESS OF LAST VECTOR IN LIST
(IA+1) ETC.: RESTORED

NEEDS

CALLS	NEEDS	DEPTH
MI READ X/Y CRD→E,A	A,C,E,F7,J2,J3,J5,M,R	NIL
MI PLT LRG CTN VTR	A,B,D,E,R	1Q
MI INTRPT	*	1Q
ADDITIONAL FOR SELF	NIL	1Q
TOTAL NEEDS & DEPTH	A,B,C,D,E,F7,J2,J3,J5,M,R*	2Q

NO. OF MICRO-ORDERS 6

MICROCODE

1	GRB LRG X/Y VTRS:	DSQ		MI READ X/Y CRD->E,A	→	S
2		DSQ		MI PLT LRG CTN VTR	→	S
3		TRF	JF13	$X^1 \rightarrow X, Y^1 \rightarrow Y$		
4		DSQ	JINT	MI INTRPT	→	S
5		DMC	JF15	GRB LRG X/Y VTRS	→	S
6		DQS				NIL

COMMENTS

PUTS X,Y → X^1Y^1; Xc,Q OR Q,Yc TO E,A: Y,X/Y, RST,L → F7,J2,J3,J5
PLOTS XC OR YC VECTOR IF NON-ZERO
IF RESET, RESET DISPLAY TO X,Y VALUE IMMEDIATELY PRECEDING THIS VECTOR
IF INTERRUPT REQUESTED, INSERT MI INTRPT
IF LINKED, RETURN TO BEGINNING OF THIS GRBFN
ELSE RETURN TO GNI, END

DESCRIPTION OF GRAPHICS-B FUNCTION 'LARGE X OR Y POINTS' (GRBFN 5)

This routine is almost identical to GRB LRG X/Y VTRS, the only difference being a call to MI PLT LRG CTN PNT rather than to MI PLT LRG CTN VTR.

FORMATS, INITIAL & FINAL VALUES

AS GRB LRG X/Y VTRS

NEEDS

CALLS	NEEDS	DEPTH
MI READ X/Y CRD→E,A	A,C,E,F7,12,13,15,M,R	NIL
MI PLT LRG CTN PNT	A,D,E	NIL
MI INTRPT	*	1Q
ADDITIONAL FOR SELF	NIL	1Q
TOTAL NEEDS & DEPTH	A,C,D,E,F7,12,13,15,M,R	*2Q

NO. OF MICRO-ORDERS 6

MICROCODE

					COMMENTS
READ POINT FOR VECTOR & REPLACE ORDER 2 OF GRB LRG X/Y VTRS BY,					
2		DSQ	MI PLT LRG CTN PNT	→ S	PLOTS POINT AT XC OR YC RELATIVE TO CURRENT X,Y

GRB LRG X/Y PNTS

DESCRIPTION OF GRAPHICS-B FUNCTION 'LARGE X OR Y CONSTANT SEPARATION POINTS' (GRBFN 7)

This routine is almost identical to GRB LRG X/Y CS VTRS. The reset YC option enables a conventional graph, i.e., a set of ordinates at regularly spaced abscissae, to be plotted efficiently (points P1,P2 .. of FIGURE 4.8, p 4.14).

FORMATS, INITIAL & FINAL VALUES

AS GRB LRG X/Y CS VTRS

NEEDS

AS GRB LRG X/Y PNTS, EXCEPT ADDITIONAL NEEDS FOR SELF BECOME N, NOT NIL

TOTAL NEEDS & DEPTH	A,C,D,E,F7,12,13,15,M,N,R	2Q

NO. OF MICRO-ORDERS 13

MICROCODE

					COMMENTS
READ POINT FOR VECTOR, RENAME LL Y VTR INC X TO LL Y PNT INC X, AND REPLACE ORDER 4 OF GRB LRG X/Y CS VTRS BY,					
4		DSQ	MI PLT LRG CTN PNT	→ S	PLOTS POINT AT XC OR YC RELATIVE TO CURRENT X,Y

GRB LRG X/Y CS PNTS

DESCRIPTION OF GRAPHICS-B FUNCTION 'LARGE X OR Y CONSTANT SEPARATION VECTORS' (GRBFN 6)

This routine extends GRB LRG X/Y VTRS by displacing the display beam by a constant amount (constant separation, CS) at right angles to the direction of the plotted vector. FIGURE 4.8, p 4.14 shows a list of Y vectors which are separated by a constant X displacement of CS. The sign of the displacement is specified by CS; it is independent of the sign of X_c or Y_c.

INITIAL & FINAL VALUES

AS GRB LRG X/Y VTRS, EXCEPT FOR THE INSERTION OF THE 2's COMPLEMENT 'CONSTANT SEPARATION' WORD, CS, IMMEDIATELY FOLLOWING THE GRB HEADER.

NEEDS

AS GRB LRG X/Y VTRS, EXCEPT ADDITIONAL NEEDS FOR SELF BECOME N, NOT NIL

TOTAL NEEDS & DEPTH A,B,C,D,E,F7,12,13,15,M,N,R 2Q

NO. OF MICRO-ORDERS 13

MICROCODE

										COMMENTS
1	GRB LRG X/Y CS VTRS:	ACC(RR)	JCNB (WT)			+1	C	→	C	INCREMENT IA IN C READ 'CONSTANT SEPARATION'→M
2		ABA	JCRA (WT)				M	→	N	STORE CS IN N FOR ENTIRE LIST
3	LL LOOP:	DSQ		MI READ X/Y CRD->E,A				→	S	PUTS X,Y→X Y ; Xc,0 OR 0,Yc IN E,A; V,X/Y,RST,L→F7,12,13,15
4		DSQ		MI PLT LRG CTN VTR				→	S	PLOTS X OR Y VECTOR IF NON-ZERO
5		TRF	JF13	$X^1 \rightarrow X, Y^1 \rightarrow Y$						IF RESET, RESET DISPLAY TO X,Y VALUE IMMEDIATELY BEFORE THIS VECTOR
6		ABA			N			→	E	PUT CONSTANT SEPARATION → E
7		DMC	JF12	LL Y VTR INC X				→	S	IF F12, I.E. IF Y VECTOR, GO TO LL Y VTR INC X
8		ABA			Y	+0	E	→	Y	X VECTOR ∴ INCREMENT Y BY 'CONSTANT SEPARATION' NOW IN E
9	LL TAIL:	DSQ	JINT	MI INTRPT				→	S	IF INTERRUPT REQUESTED, INSERT MI INTRPT
10		DMC	JF15	LL LOOP				→	S	IF LINKED, RETURN TO LL LOOP
11		DQS							NIL	ELSE RETURN TO GNI, END
12	LL Y VTR INC X:	ABA			X	+0	E	→	X	Y VECTOR ∴ INCREMENT X BY 'CONSTANT SEPARATION' NOW IN E
13		DMC		LL TAIL				→	S	RETURN TO COMMON TAIL OF ROUTINE

DESCRIPTION OF GRAPHICS-B FUNCTIONS 'SYMBOL PAIRS' (GRBFN 10)

This routine reads symbol-pair words from core and transfers control, through SYMBOL TABLE, to the specific symbol execution routines (SYM 0 to SYM 255). In each symbol-pair word, the symbol in the upper byte is executed before the symbol in the lower byte. After each symbol execution, control is returned to this routine (via the terminating QS transfer of SYM) and the next symbol in the list then drives SYMBOL TABLE, etc.

The symbol list is terminated by the END-OF-LIST symbol which returns control to the GNI sequence by popping the Q stack twice: first to NIL (this destroys the return address to GRB SYMBOL PAIRS) and then to S (Q holds the GNI return address on entry to the symbol execution routines). The interrupt test is applied after each symbol is executed.

INITIAL VALUES

C: IA, THE INSTRUCTION ADDRESS (AT GRB HEADER ADDRESS)
(IA+1), ETC. IN CORE: SYMBOL-PAIR WORDS
W1-7: BITS 9-15 OF GRB HEADER
Q: RETURN ADDRESS IN GNI ROUTINE

MICROCODE

1	GRB SYMBOL PAIRS:	DMC		0			→	D
2	LL READ SYM PAIR->M:	ACC(RR)	JCNB (WT)		+1	C	→	C
3		ABA(U)	JCRA (WT)			M	(8T)	D
4		DSQ		SYMBOL TABLE			→	S
5		DSQ	JINT	MI INTRPT			→	S
6		ABA(L)				M	→	D
7		DSQ		SYMBOL TABLE			→	S
8		DSQ	JINT	MI INTRPT			→	S
9		DMC		LL READ SYM PAIR→M			→	S

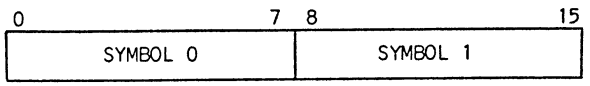
FINAL VALUES

C: THE ADDRESS OF THE FINAL SYMBOL-PAIR WORD
(IA+1), ETC. IN CORE: AS INITIAL

NEEDS

CALLS	NEEDS	DEPTH
SYMBOL TABLE	D, SYM 0 - SYM 255	
MI INTRPT	*	1Q
SELF NEEDS & DEPTH	C,D,M,	1Q
TOTAL NEEDS & DEPTH	C,D,M,SYM 0-SYM 255,*	2Q

NO. OF MICRO-ORDERS 9



COMMENTS

CLEAR D
INCREMENT IA IN C & READ SYMBOL PAIR TO M
PUT UPPER SYMBOL TO DL
GO TO SYMBOL TABLE
ON RETURN FROM SYMBOL EXECUTION (EXCEPT END-OF-LIST SYMBOL), TEST FOR INTRPT
PUT LOWER SYMBOL TO DL
GO TO SYMBOL TABLE
ON RETURN FROM SYMBOL EXECUTION (EXCEPT END-OF-LIST SYMBOL), TEST FOR INTRPT
READ NEXT SYMBOL PAIR

DESCRIPTION OF GRAPHICS-B FUNCTION 'MODIFY X,Y,θ' (GRBFN 12)

This function is specified by a GRB HEADER and up to three 2's-complement data words which immediately follow it. FORMAT A3.1, this page, details the header. The data words are interpreted in sequence by scanning the header field from bits 9 to 14. The four possible outcomes for X (bits 9 and 10) are,

0,0	X unchanged
0,1	X incremented by following data word
1,0	X cleared
1,1	X set to first data word,

and similarly for Y and θ. Angle θ is held in register A.

INITIAL VALUES

C: IA, THE INSTRUCTION ADDRESS, GRB HEADER ADDRESS

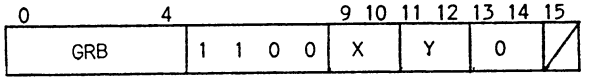
(IA+1), ETC. IN CORE: A LIST OF UP TO THREE 2's COMPLEMENT WORDS TO BE ADDED TO X,Y, OR θ INTERPRETED IN CONJUNCTION WITH HEADER

Q: RETURN ADDRESS TO GNI ROUTINE

WU: BITS 8-15 OF THE GRB HEADER

FINAL VALUES

X: AS INITIAL, CLEARED,
Y: INCREMENTED BY, OR SET TO
A: 2's COMPLEMENT DATA WORDS



FORMAT 3.1

IF BIT 9 = 1 CLEAR X
" " 10 = 1 ADD NEXT WORD TO X (16-BITS, 2's COMP.)
" " 11 = 1 CLEAR Y
" " 12 = 1 ADD NEXT WORD TO Y (16-BITS, 2's COMP.)
" " 13 = 1 CLEAR θ
" " 14 = 1 ADD NEXT WORD TO θ (BITS 0-14)

CALLS NIL

NEEDS A,C,F9-14,M DEPTH 1Q

NO. OF MICRO-ORDERS 14

MICROCODE

1	GRB MODIFY X,Y,THETA:	ABA					W	(8T)	M
2		FBC(MF)		M9-14→F9-14					
3		DMC	JF9	0				→	X
4		DMC	JF11	0				→	Y
5		DMC	JF13	0				→	A
6		DSQ	JF10	LL READ NEXT WORD				→	S
7		ABA	JF10		X	+0	M	→	X
8		DSQ	JF12	LL READ NEXT WORD				→	S
9		ABA	JF12		Y	+0	M	→	Y
10		DSQ	JF14	LL READ NEXT WORD				→	S
11		ABA	JF14		A	+0	M	→	A
12		DQS							NIL
13	LL READ NEXT WORD:	ACC(RR)	JCNB (WT)			+1	C	→	C
14		AQS	JCRA (WT)						NIL

COMMENTS

PUTS HEADER BITS 9-14 IN M9-14
COPY HEADER BITS 9-14 TO F9-14 (NOW AVAILABLE AS J INDICATORS)
IF HEADER BIT 9 = 1, CLEAR X
" " " 11 = 1, " Y
" " " 13 = 1, " A
IF HEADER BIT 10 = 1, GO TO LL READ NEXT WORD & ADD WORD TO X
IF " 12 = 1, " Y
IF " 14 = 1, " A
RETURN TO GNI, END
INCREMENT IA IN C & READ NEXT DATA WORD → M
WAIT UNTIL DATA AVAILABLE IN M THEN RETURN VIA Q

DESCRIPTION OF MICRO-ROUTINE 'READ SMALL CO-ORDINATES TO D'

This routine reads and restores a small co-ordinate pair (one 16-bit word, FORMAT 4.1, p 4.3 or FORMAT 4.3, p 4.5 from core store location IA + 1, where IA is the initial value of the instruction address held in register C. The core word is right shifted into D to remove the unblanking (V) and link (L) bits from the Cartesian or polar co-ordinates. Bits V & L are recorded in F7 and F15 respectively.

INITIAL VALUES

C: IA, THE INSTRUCTION ADDRESS
(IA+1) IN CORE: ΔX,V; ΔY,L OR $\Delta\theta$,V; Δs,L

MICROCODE

1	MI READ SML CRD->D:	ACC(RR)	JCNB (WT)		+1	C	→	C
2		ABA(U/L)	JCRA (WT)			M	(RS)	D
3		FBC(MF)		M7→F7,M15→F15				
4		DQS						NIL

FINAL VALUES

C: IA+1
(IA+1) IN CORE: RESTORED.
DU: ΔX OR $\Delta\theta$ LSB = 1 DISPLAY INCREMENT OR $\pi.2^{-8}$ RADIANS.
DL: ΔY OR Δs LSB = 1 " "
F7,F15: V,L.

CALLS NIL.

NEEDS C,D,F7,15,M.

NO. OF MICRO-ORDERS 4

COMMENTS

INCREMENT IA IN C & START RR CORE CYCLE
PUT ΔX,ΔY OR $\Delta\theta$,Δs TO D
RECORD V,L → F7,F15
RETURN, END

DESCRIPTION OF MICRO-ROUTINE 'READ SMALL POLAR CO-ORDINATES TO E,N'

This routine begins with MI READ SML CRD→D for polar co-ordinates, then extends it by,

1. expanding Δθ to 16 bits and aligning it ot the θ format described in MI SIN(D TRC 9)→D.
2. expanding Δs to 16-bit positive integer format in N.

INITIAL VALUES

C: IA, THE INSTRUCTION ADDRESS

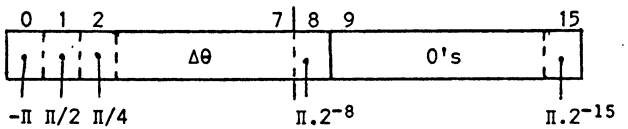
(IA+1) IN CORE: Δθ,V; Δs,L AS FORMAT 4.3, p 4.5.

FINAL VALUES

C: IA+1

(IA+1) IN CORE: RESTORED

E: Δθ, ALIGNED TO θ FORMAT



BITS 0,1 = BIT 2 BITS 9-15 = 0's

FINAL VALUES (CONT)

N: Δs POSITIVE INTEGER WORD LSB = 1 DISPLAY INCREMENT

BITS 0-9 = 0's

F7,F15: V,L

NEEDS

CALLS	NEEDS	DEPTH
MI READ SML CRD D	C,D,F7-15,M	NIL
ADDITIONAL FOR SELF	E,N	1Q
TOTAL NEEDS & DEPTH	C,D,E,F7-15,M,N	1Q

NO. OF MICRO-ORDERS 4

MICROCODE

1	MI READ SML PLR->E,N:	DSQ	MI READ SML CRD→D				→	S
2		DMC	X'F,F,0,0' .				→	N
3		ABA		N	Λ	D	(RS)	E
4		AQS		$\overline{N}$	Λ	D	→	N

COMMENTS

PUTS Δθ,Δs TO DU,DL & V,L TO F7,F15

1's TO NU, 0's TO NL TO SELECT Δθ

Δθ EXTENDED TO 16 BITS & ALIGNED TO θ (MSB = -Π)

1's OF $\overline{NL}$ SELECT Δs RETURN, END (IF Δs = 0, Ze = 1 ON EXIT)

DESCRIPTION OF MICRO-ROUTINE 'READ LARGE CO-ORDINATES TO E,A'

Reads a large-format Cartesian or polar co-ordinate pair from core store locations IA + 1 and IA + 2 (IA is the instruction address, held in register C), and places Xc (or R) in register E and Yc (or θ) in register A. FORMAT 4.4, p 4.7, shows the core representation of large Cartesian co-ordinates and FORMAT 4.5, p 4.9, shows the core representation of large polar co-ordinates.

All co-ordinate words from core are right shifted; thus, Xc,Yc and R are aligned in E or A to LSB = 1 display increment. Angle θ in A, however, after the right shift, has MSB = -2π; hence, A must be left shifted by the calling sequence to align θ to the θ format of MI SIN(A)→D, etc.

INITIAL VALUES

C: IA, THE INSTRUCTION ADDRESS
(IA + 1) IN CORE: Xc,V or R,V
(IA + 2) IN CORE: Yc,L or θ,L

MICROCODE

1	MI READ LRG CRD->E,A;	ACC(RR)	JCNB (WT)		+1	C	→	C
2		ABA	JCRA (WT)			M	(RS)	E
3		FBC(CF)		0→F7				
4		FBC(SF)	JBd15	1→F7				
5		ACC(RR)	JCNB (WT)		+1	C	→	C
6		ABA	JCRA (WT)			M	(RS)	A
7		FBC(MF)		M15→F15				
8		DQS						NIL

FINAL VALUES

C: IA + 2
(IA + 1),(IA + 2) IN CORE: RESTORED
E: Xc or R, LSB = 1 DISPLAY INCREMENT
A: Yc (LSB = 1 DISPLAY INCREMENT) or θ (MSB = -2π)
(*NOTE: θ IN REGISTER A MUST BE LEFT SHIFTED TO ALIGN WITH θ FORMAT OF MI SIN(A)→D, ETC.)
F7,F15: V,L THE UNBLANKING AND LINK BITS

CALLS NIL.

NEEDS A,C,E,F7,15,M.

NO. OF MICRO-ORDERS 8.

COMMENTS

INCREMENT IA, READ Xc OR R & V TO M
PUT Xc OR R TO E, V → Bd15
CLEAR F7
IF V = 1 set F7 | V → F7
INCREMENT IA, READ Yc OR θ & L TO M
PUT Yc OR θ TO A
L → F15
RETURN, END

DESCRIPTION OF MICRO-ROUTINE 'READ LARGE X OR Y CO-ORDINATE TO E,A'

This routine reads a large X or Y co-ordinate word (FORMAT 4.7, p 4.13) from core location IA + 1. If an X word is coded, i.e., if bit 12 = 0, XC is placed in E AND A is cleared; whereas if bit 12 = 1, YC is placed in A and E is cleared. A record of bit 12 is held in F12.

RESET (bit 13) is recorded in F13, and the V & L bits are recorded in F7 & F15 respectively. The routine begins by storing X,Y in X^1,Y^1: this enables the reset option to be extended to each item plotted. The procedure assumes that XC is intended and puts the co-ordinate into E and clears A; if the assumption is incorrect (F12 = 1) orders 10 & 11 transpose E & A.

INITIAL VALUES

C: IA, THE INSTRUCTION ADDRESS
(IA+1) IN CORE: XC OR YC, X/Y, RST, V & L WORD

FINAL VALUES

C: IA+1
(IA+1) IN CORE: RESTORED
E: IF X/Y BIT = '0', THEN XC ELSE ZERO
A: " , " ZERO " YC
F7,12,13,15: V, X/Y, RST, L

CALLS NIL

NEEDS A,C,E,F7,12,13,15,M,R

NO. OF MICRO-ORDERS 12

MICROCODE

1	MI READ X/Y CRD->E,A:	TRF		$X \rightarrow X^1, Y \rightarrow Y^1$				
2		ACC(RR)	JCNB (WT)		+1	C	→	C
3		FBC(CF)		0→F7				
4		ABA	JCRA (WT)			M	(RS)	E
5		FBC(SF)	JBe15	1→F7				
6		FBC(MF)		M12,13,15→F12,13,15				
7		DMC		3			→	R
8		ADR	JNZR (RP)			E	(RS)	E
9		DMC		0			→	A
10		ABA	JF12			E	→	A
11		DMC	JF12	0			→	E
12		DQS						NIL

COMMENTS

COPY X,Y TO X^1,Y^1 SO THAT DISPLAY CAN BE RESET AFTER EACH ITEM
INCREMENT IA IN C, READ XC OR YC; X/Y, RST, V & L TO M
CLEAR F7 | V TO Be15 | IF Be15, SET F7 — V→F7
X/Y IN F12, RST IN F13, L IN F15
PUT 3 IN R TO COUNT SHIFTS
ASSUMES XC, E NOW HOLDS XC ALIGNED LSB = 1 DISPLAY INCREMENT
" " , A CLEARED
IF YC, OVERWRITE ASSUMED ZERO IN A WITH YC
" " , " " XC IN E WITH ZERO
RETURN, END

DESCRIPTION OF MICRO-ROUTINE 'PLOT SMALL CARTESIAN VECTOR DU,DL'

Plots the Cartesian vector with components DU,DL each of which are 2's complement byte integers.

Plotting requires 128 cycles of the arithmetic unit in APV (U/L) mode, with overflows and underflows automatically incrementing and decrementing the display registers as described in SECTION 3.2, p 3.8. Magnification is achieved by proportionally increasing the number of cycles. In general, the vector plotted is,

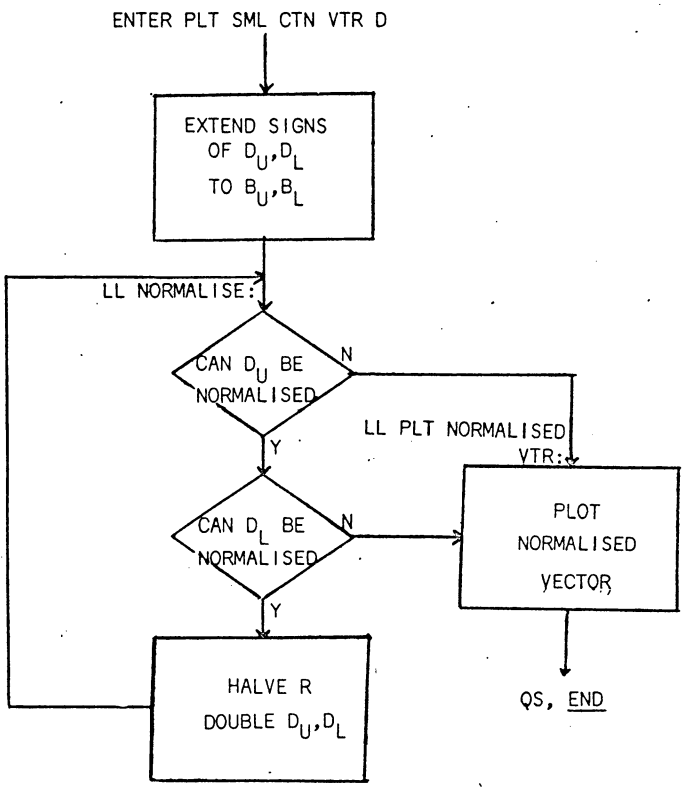
$n2^{-7}$ (DU,DL) where n is the initial value in counter R. The vector increments are added to the initial display co-ordinates in the X,Y registers.

Approximately constant plotting rates for vectors are achieved by normalising vectors before plotting. A vector can be normalised if both components can be normalised, i.e., a normalise step (double DU, double DL and halve the count in R) is executed if the most significant two bits of DU are identical and if the most significant two bits of DL are identical. Indicators JNNMU and JNNML assist normalisation,

JNNMU = Be0 ⊕ Be1 , JNNML = Be8 ⊕ Be9

(J indicator NNMU (NOT NORMALISE UPPER) is the EXCLUSIVE-OR of Be0 and Be1, which are temporary copies of DU bits 0 and 1. Similarly for JNNML.)

The sign bits of DU and DL are extended to the accumulators BU and BL respectively. Thus, if DU is negative, the initial value of BU is set to -1; this ensures exactly |DU| underflows throughout 128 accumulation cycles. As an example, consider DU = -1; after 128 cycles this accumulates to -128 which, without an initial value of -1 in BU, would not produce the single underflow required as a 2's complement byte accommodates -128. A DU value of +1 however, accumulates to +128 which, with an initially cleared BU, does produce the required single overflow.



INITIAL VALUES

DU: HORIZONTAL COMPONENT, 2's COMPLEMENT INTEGER BYTE
DL: VERTICAL " " " " "
R: n, NUMBER OF CYCLES, POSITIVE INTEGER WORD

CONDITION

DU,DL ≠ 0,0

FINAL VALUES

$X: X_o + n2^{-7}DU$

$Y: Y_o + n2^{-7}DL$

CALLS NIL

NEEDS B,D,R

NO. OF MICRO-ORDERS 11

MICROCODE

```
1   MI PLY SML CTN VTR D: ABA                                    D  →  NIL
2                         DMC          0                            →  B
3                         DMC(U)  JBe0 X'FF'                        →  B
4                         DMC(L)  JBe8 X'FF'                        →  B
5   LL NORMALISE:         DMC     JNNMU LL PLT NRMLSD VTR           →  S
6                         DMC     JNNML LL PLT NRMLSD VTR           →  S
7                         ABA                                    R (RS) R
8                         ABA(U/L)                               D (LS) D
9                         DMC           LL NORMALISE                →  S
10  LL PLT NRMLSD VTR:    APV     JNZR (RP)        B    +0       D  →  B
11                        DQS                                          NIL
```

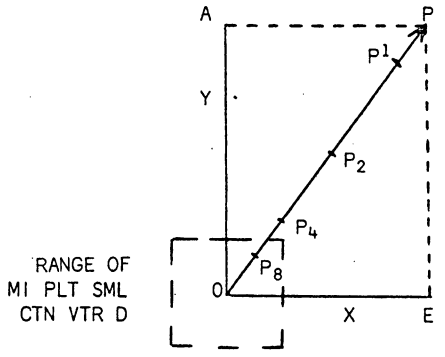
COMMENTS

```
DU→Be0,1;DL→Be8,9 FOR SIGN & NORMALISE TESTS
CLEAR B
IF DU < 0  PROPAGATE 1's THROUGH BU  |  EXTEND SIGNS OF
IF DL < 0       "       "     "  BL  |  DU,DL TO BU,BL
IF DU CANNOT BE NORMALISED GO TO LL PLT NRMLSD VTR
ELSE IF DL CANNOT BE NORMALISED GO TO LL PLT NRMLSD VTR
     ELSE HALVE CYCLES IN R
          DOUBLE VECTOR DU,DL
          GO TO LL NORMALISE
PLOT NORMALISED VECTOR
RETURN, END
```

DESCRIPTION OF MICRO-ROUTINE 'PLOT LARGE CARTESIAN VECTOR E,A'

Plots the Cartesian vector with components E,A each of which are 2's complement integers (with the restriction that within each component word, bits 0 & 1 are identical).

The large vector is plotted as a series of scaled sections, each of which is within the range of MI PLT SML CTN VTR D.

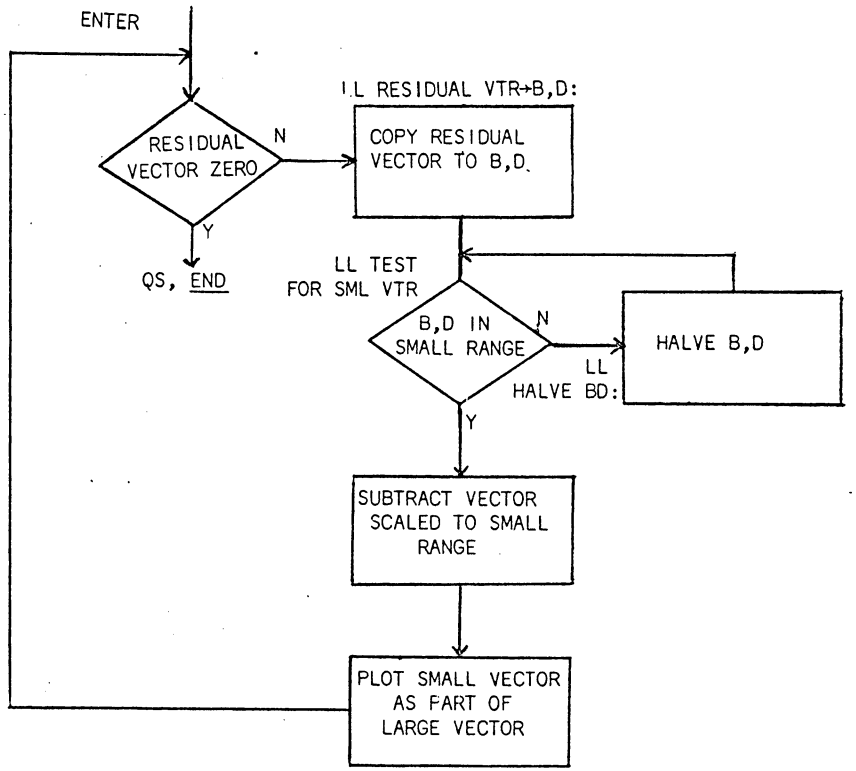


In the illustration, vector OP is halved three times (OP_2,OP_4,OP_8) before it falls within the range of MI PLT SML CTN VTR D.

Vector OP_8 is plotted using MI PLT SML CTN VTR D and subtracted from OP leaving the residual vector OP' which then replaces OP in the above argument.

(The small range vector is subtracted before entry to MI PLT SML CTN VTR D because it could be normalised before plotting; e.g., if E,A initially falls within the small range.) If initially E,A = 0,0, no points are plotted, i.e., the procedure is bypassed.

The vector B,D is within the range of MI PLT SML CTN VTR D if the sign bits of B & D each propagate down to bit position 9.



INITIAL VALUES

E: HORIZONTAL COMPONENT
A: VERTICAL COMPONENT

2's COMPLEMENT INTEGERS
RESTRICTION: $E_0 = E_1$, $A_0 = A_1$

FINAL VALUES

X: $X_0 + E$
Y: $Y_0 + A$

NEEDS

CALLS	NEEDS	DEPTH
MI PLT SML CTN VTR D	B,D,R	NIL
ADDITIONAL FOR SELF	A,E	1Q
TOTAL NEEDS & DEPTH	A,B,D,E,R	1Q

NO. OF MICRO-ORDERS 22

	MICROCODE							COMMENTS
1	MI PLT LRG CTN VTR:	ABA			E	→	NIL	PUT E TO Ze FOR ZERO TEST | TEST
2		DMC	JNZe	LL RESIDUAL VTR->B,D		→	S	IF E ≠ 0 THEN GO TO LL RESIDUAL VTR→B,D | E,A = 0,0
3		ABA		A		→	NIL	ELSE PUT A TO Ze FOR ZERO TEST
4		DMC	JNZe	LL RESIDUAL VTR->B,D		→	S	IF A ≠ 0 THEN GO TO LL RESIDUAL VTR→B,D |
5		DQS					NIL	ELSE RETURN, END
6	LL RESIDUAL VTR->B,D:	ABA			E	→	B	COPY RESIDUAL E TO B | COPY RESIDUAL E,A TO B,D FOR
7		ABA		A		→	D	COPY RESIDUAL A TO D | REDUCTION TO SMALL VECTOR
8	LL TEST FOR SML VTR:	ABA		B		(LS)	NIL	PUT B_{1-8} → ZeU TO TEST IF 0's PROPAGATE DOWN TO B_9 (B_0=B_1)
9		ABA	JNZeU	$\overline{B}$		(LS)	NIL	IF B_{1-8} ≠ 0's THEN → PUT $\overline{B}_{1-8}$ TO ZeU TO TEST PROPAGATION OF 1's
10		DMC	JNZeU	LL HALVE B,D		→	S	IF B_{1-8} ≠ 1's THEN GO TO LL HALVE B,D
11		ABA			D	(LS)	NIL	ELSE PUT D_{1-8} TO ZeU | ELSE PUT D_{1-8} TO ZeU
12		ABA	JNZeU		$\overline{D}$	(LS)	NIL	IF D_{1-8} ≠ 0's THEN → PUT $\overline{D}_{1-8}$ TO ZeU
13		ABA	JNZeU	LL HALVE B,D		→	S	IF D_{1-8} = 1's THEN GO TO LL HALVE B,D
14		ABA		$\overline{B}$ +1	E	→	E	ELSE → SUBTRACT SCALED - VECTOR | ELSE → SUBTRACT SCALED - VECTOR
15		ABA		A +1	$\overline{D}$	→	A	
16		ABA(L)		B		(8T)	D	PUT HORIZONTAL COMPONENT (NOW REDUCED TO SMALL RANGE) TO DU
17		DMC		128		→	R	LOAD 128 → R FOR MI PLT SML CTN VTR D (NOTE: VERTICAL COMPONENT ALREADY IN DL)
18		DSQ		MI PLT SML CTN VTR D		→	S	PLOT SMALL VECTOR AS PART OF LARGE VECTOR
19		DMC		MI PLT LRG CTN VTR		→	S	RETURN TO BEGINNING
20	LL HALVE B,D:	ABA		B		(RS)	B	HALVE B
21		ABA			D	(RS)	D	HALVE D
22		DMC		LL TEST FOR SML VTR		→	S	GO TO TEST FOR SMALL VECTOR

DESCRIPTION OF MICRO-ROUTINES 'PLOT SMALL CARTESIAN POINT DU,DL' AND 'PLOT LARGE CARTESIAN POINT E,A'

The first routine expands the small Cartesian components (initially held in DU,DL) to full word lengths in registers E,D; magnifies them by left-shifting registers E,D MG times; and adds the magnified components to the display registers X,Y. The beam is then unblanked by an APP (arithmetic & plot point) order which also returns control to the calling sequence. MG must be held in R before entering the routine.

The second routine simply adds Xc (held in E) and Yc (held in A) to X and Y respectively, D being used as an intermediate register for addition of Yc. MG does not apply to large co-ordinates.

INITIAL VALUES

MI BLT SML CTN PNT D

DU: ΔX 2's COMPLEMENT BYTE
DL: ΔY " " "
R: MG, MAGNIFICATION EXPONENT

MI PLT LRG CTN PNT

E: Xc 2's COMPLEMENT WORD
A: Yc " " "

MICROCODE

1	MI PLT SML CTN PNT D:	ABA					D	(8T)	E
2		DMC(U)		0				→	E
3		DMC(U)		0				→	D
4		DMC(U)	JBe8	X'FF'				→	E
5		DMC(U)	JBe0	X'FF'				→	D
6		ABA	JNZR (FW)				E	(LS)	E
7		ADR	JNZR (BK)				D	(LS)	D
8		ABA			S+1	+1		→	S
9	MI PLT LRG CTN PNT:	ABA			A			→	D
10		ABA			X	+0	E	→	X

FINAL VALUES

X: Xo + $2^{MG}\Delta X$ OR Xo + Xc
Y: Yo + $2^{MG}\Delta Y$ OR Yo + Yc
Xo,Yo = INITIAL X,Y
E,A: FOR MI PLT LRG CTN PNT, AS INITIAL

CALLS

NIL

NEEDS (MI PLT SML CTN PNT D)

D,E,R.

NEEDS (MI PLT LRG CTN PNT)

A,D,E.

NO. OF MICRO-ORDERS

13

COMMENTS

PUT ΔX FROM DU TO EL (SIGN ΔX→Be8, SIGN ΔY→Be0)
CLEAR EU
CLEAR DU
EXTEND SIGN BIT OF ΔX, E NOW HOLDS WORD ΔX
" " " " ΔY, D " " " ΔY
DOUBLE ΔX,ΔY MG TIMES
(INITIAL R = MG)
SKIP NEXT ORDER
PUT Yc TO D
ADD 2^{MG} ΔX (OR Xc) TO X

MICROCODE (CONT)								COMMENTS (CONT)
11	ABA	Y	+0	D	→	Y		ADD 2^{MG} ΔY (OR Yc) TO Y
12	ABA					NIL		DELAY FOR BEAM SETTING
13	APP	Q			→	S		UNBLANK POINT, RETURN, END

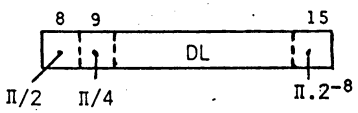
DESCRIPTION OF TABLE SIN(DL)→D

A direct tabulation of the sine function for 129 equally spaced values of the argument (held in DL) in the first quadrant. A MSB of -2 is chosen for SIN(DL) to accommodate the range -1 to +1 inclusive which is needed by related microroutines.

Byte DL is extended to full word length in D and added to the base address of the table (defining 'base' as the ROM address of the table entry corresponding to argument = 0). The value of SIN(DL) overwrites the argument in register D and control is returned via Q automatically as each table entry is a DQS order.

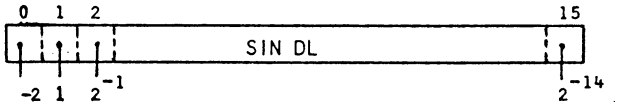
INITIAL VALUES

DL: ARGUMENT, A POSITIVE 8-BIT ANGLE
RANGE 0($\pi.2^{-8}$) $\pi/2$ RADIANS



FINAL VALUES

D: SIN(DL), 2's COMPLEMENT WORD, MSB = -2, ERROR $\pm 2^{-15}$



CALLS - NIL

NEEDS D

NO. OF MICRO-ORDERS 131.

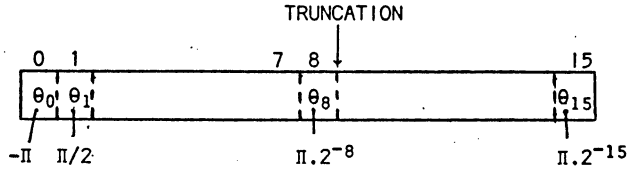
MICROCODE

1	TABLE SIN(DL)→D:	DMC(U)	0			→	D	CLEAR DU
2		ABA		S+1	+,0 D	→	S	ADVANCE BY ARGUMENT +1 TO ENTER TABLE.
3		DQS	X'0000'			→	D	SIN 0 (=0) TO D RETURN, END.
4		DQS	.			→	D	SIN $\pi.2^{-8}$ " " "
.		"	.			"	"	" $\pi.2^{-7}$ " " "
.		.	.			.	.	" ETC . . .
131		DQS	X'4000'			→	D	*SIN π/2 (=X'4000')* *TO D RETURN, END*

129 ENTRIES
0($\pi 2^{-8}$)$\pi/2$

DESCRIPTION OF MICRO-ROUTINE 'SIN(D TRUNCATED TO 9 BITS)→D'

This routine evaluates the sine of an angle θ which is first truncated to 9-bits to match the angular resolution of the first quadrant sine table, 'TABLE SIN(DL)→D'.



FORMAT OF ANGLE θ

QUANDRANT 1 $(\theta_0,\theta_1 = 0,0)$:

The range of θ_{1-8} is $0(\Pi 2^{-8})$ $[\Pi/2-\Pi 2^{-8}]$ which is directly within the range of the table. Thus, a preliminary alignment of θ_{1-8} to DL only is required; a left shift followed by an 8T transpose of register D achieves this. (DU is cleared by TABLE SIN(DL)→D).

QUANDRANT 2 $(\theta_0,\theta_1 = 0,1)$:

$$\theta_{0-8} = \Pi/2 + \theta_{2-8}$$

but $SIN(\Pi/2+x) = SIN(\Pi/2-x)$

i.e. $SIN\ \theta_{0-8} = SIN(\Pi/2-\theta_{2-8})$ for this quadrant

$$\Pi/2-\theta_{2-8} = 0,1,0\text{'s} + 1,1,\overline{\theta}_{2-8} + \Pi 2^{-8}$$
$$= 0,0,\overline{\theta}_{2-8} + \Pi 2^{-8}$$

which has the range

$$\Pi 2^{-8}(\Pi 2^{-8})\Pi/2$$ i.e., is within the range of the table.

but, $0,0,\overline{\theta}_{2-8} + \Pi 2^{-8} = 0,\overline{\theta}_{1-8} + \Pi 2^{-8}$ for $\theta = 1$

Thus, for quadrant 2, register D is left-shifted, DU is negated as an 2's complement byte, and the register bytes are transposed.

QUANDRANTS 3 & 4 $(\theta_0 = 1)$

$$SIN(-\Pi+x) = -SIN\ x$$

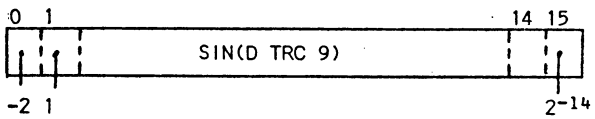
Thus, for quadrants 3 & 4, the procedures for quadrants 1 & 2 respectively are followed, but the results are negated.

INITIAL VALUES

D: ANGLE θ 2's COMPLEMENT WORD (FORMAT, SEE DESCRIPTION)
RANGE $-\Pi\ (\Pi.2^{-15})\ [\Pi-\Pi.2^{-15}]$ RADIANS
BITS 9-15 ARE ARBITRARY

FINAL VALUES

D: SIN(D TRC 9); 2's COMPLEMENT WORD, MSB = -2, ERROR = $\pm 2^{-15}$



NEEDS

CALLS	NEEDS	DEPTH
TABLE SIN(DL) D	D	NIL
ADDITIONAL FOR SELF	F8	1Q
TOTAL NEEDS & DEPTH	D,F8	1Q

NO. OF MICRO-ORDERS 8.

MICROCODE

									COMMENTS
1	MI SIN(D TRC 9)->D:	ABA				D	(LS)	D	θ_0→Bdo, θ_1→Beo
2		FBC(CF)		0→F8					CLEAR F8 | θ_Q→F8 RECORDS θ_Q FOR QUAD 3,4
3		FBC(SF)	JBdo	1→F8					IF θ_Q, SET F8 | CORRECTION
4		ABA(U)	JBeo		+1	$\overline{D}$	→	D	IF θ_1 = 1, REPLACE θ_{1-8} BY $\overline{\theta}_{1-8}$ + $\Pi 2^{-8}$
5		ABA				D	(8T)	D	TRANSPOSE D FOR TABLE SIN(DL)→D
6		DSQ		TABLE SIN(DL)->D			→	S	GO TO TABLE SIN(DL)→D
7		ABA	JF8		+1	$\overline{D}$	→	D	IF QUAD 3,4, REPLACE SIN θ_{1-8} BY -SIN θ_{1-8}
8		DQS						NIL	RETURN, END

DESCRIPTION OF MICRO-ROUTINE 'SIN(A TRUNCATED TO 9 BITS)→D'

This routine differs from MI SIN(D TRC 9)→D in that register A holds angle θ throughout the routine.

The formats of θ and the result are identical to those of MI SIN(D TRC 9)→D.

A DMC, not DSQ, micro-order calls MI SIN(D TRC 9)→D; thus, the final DQS order of MI SIN(D TRC 9)→D returns control to the sequence calling MI SIN(A TRC 9)→D. This saves one nesting level and a terminating AQS order for MI SIN(A TRC 9)→D.

INITIAL VALUES

A: AS D IN MI SIN(D TRC 9)→D

FINAL VALUES

A: AS INITIAL
D: AS MI SIN(D TRC 9)→D

NEEDS

CALLS	NEEDS	DEPTH
MI SIN(D TRC 9)→D	D,F8	1Q
ADDITIONAL FOR SELF	A	NIL
TOTAL NEEDS & DEPTH	A,D,F8	1Q

NO. OF MICRO-ORDERS 2.

MICROCODE

							COMMENTS
1	MI SIN(A TRC 9)->D:	ABA		A	→	D	LOAD D
2		DMC		MI SIN(D TRC 9)->D	→	S	GO TO MI SIN(D TRC 9)→D , RETURN , END

MI SIN(A TRC 9)→D

DESCRIPTION OF MICRO-ROUTINE 'SIN(A ROUNDED TO 9 BITS)→D'

This routine rounds, rather than truncates, angle θ; otherwise the routine is indentical to MI SIN(A TRC 9)→D.

$$\theta \text{ RND } 9 = (\theta + \pi.2^{-9}) \text{ TRC } 9$$

Thus, $\pi.2^{-9}$ is added to register D before the call to MI SIN(D TRC 9)→D.

NO. OF MICRO-ORDERS 3.

MICROCODE

									COMMENTS
1	MI SIN(A RND 9)->D:	DMC		X'0040'			→	D	PUT $\pi.2^{-9}$ IN D
2		ABA		A	+0	D	→	D	PUT $\theta + \pi.2^{-9}$ IN D
3		DMC		MI SIN(D TRC 9)->D			→	S	GO TO MI SIN(D TRC 9)→D , RETURN , END

MI SIN(A RND 9)→D

DESCRIPTION OF MICRO-ROUTINE 'COS(A TRUNCATED TO 9 BITS)→D

NO. OF MICRO-ORDERS 3.

This routine evaluates cosine, but otherwise is identical to MI SIN(A TRC 9)→D.

$$\cos(\theta \text{ TRC } 9) = \sin(\pi/2-(\theta \text{ TRC } 9))$$

but $-(\theta \text{ TRC } 9) = (\overline{\theta} + \pi.2^{-8}) \text{ TRC } 9$

where $\overline{\theta} = \overline{\theta}_0, \overline{\theta}_1, ..$

i.e. the transformation,

$$\underline{\overline{\theta} + \pi/2 + \pi.2^{-8} \rightarrow \theta}$$

precedes entry to 'MI SIN(D TRC 9)→D'.

MICROCODE						COMMENTS
1	MI COS(A TRC 9)->D:	DMC	X'4080'	→	D	PUT $\pi/2 + \pi.2^{-8}$ IN D
2		ABA	$\overline{A}$ +0 D	→	D	PUT $\overline{\theta} + \pi/2 + \pi.2^{-8}$ IN D
3		DMC	MI SIN(D TRC 9)->D	→	S	GO TO MI SIN(D TRC 9)→D , RETURN , END

DESCRIPTION OF MICRO-ROUTINE 'COS(A ROUNDED TO 9 BITS)→D'

NO. OF MICRO-ORDERS 3.

This routine evaluates the cosine of θ rounded to 9 bits. Formats, initial and final values and needs are identical to those of MI SIN(A TRC 9)→D.

$$\cos(\theta \text{ RND } 9) = \sin(\pi/2-(\theta \text{ RND } 9))$$

if $\theta_9 = 0$, $-(\theta \text{ RND } 9) = -(\theta \text{ TRC } 9)$

$= (\overline{\theta} + \pi 2^{-8}) \text{ TRC } 9$

which also $= (\overline{\theta} + \pi 2^{-9}) \text{ TRC } 9$

if $\theta_9 = 1$, $-(\theta \text{ RND } 9) = ((\theta + \pi 2^{-8}) \text{ TRC } 9)$

$= \overline{\theta} \text{ TRC } 9$ because, for 2's complement representation in general, the negative of a number can also be found by subtraction of '1' in the least significant position followed by bitwise complementation. For truncation, subtraction of $\pi 2^{-8}$, not '1' in the L.S.B. position, is necessary.

For $\theta_9 = 1$, $\overline{\theta}$ TRC 9 is also $(\overline{\theta} + \pi 2^{-9})$ TRC 9

Thus, for $\theta_9 = 0$ or 1,

$$-(\theta \text{ RND } 9) = (\overline{\theta} + \pi 2^{-9}) \text{ TRC } 9$$

i.e., the transformation,

$$\underline{\overline{\theta} + \pi/2 + \pi.2^{-9} \rightarrow \theta}$$

precedes entry to 'MI SIN(D TRC 9)→D'.

MICROCODE

							COMMENTS
1	MI COS(A RND 9)->D:	DMC	X'4040'		→	D	PUT $\pi/2 + \pi.2^{-9}$ IN D
2		ABA	$\overline{A}$	+0 D	→	D	PUT $\overline{\theta} + \pi/2 + \pi.2^{-9}$ IN D
3		DMC	MI SIN(D TRC 9)->D		→	S	GO TO MI SIN(D TRC 9)→D , RETURN , END

DESCRIPTION OF MICRO-ROUTINES 'SIN(A)→D' & 'COS(A)→D'

These micro-routines evaluate the sine or cosine of the 16 bit angle θ in register A.

$$\text{COS } A = \text{SIN}(\pi/2-\theta)$$ i.e., the preliminary transformation $\pi/2-\theta \rightarrow \theta$ enables a common procedure to be used for SIN & COS. Entry to MI COS A →D is recorded by setting indicator F9. This allows the original θ to be restored in register A after the common procedure is completed.

SIN θ is evaluated by simple linear interpolation between two values of SIN(θ TRC 9) i.e.,

$$\text{SIN } \theta = \text{SIN}(\theta \text{ TRC } 9) + \Delta.\theta_{9-15}/\pi.2^{-8} \text{ where,}$$

first difference $\Delta = \text{SIN}((\theta+\pi.2^{-8})\text{TRC } 9) - \text{SIN}(\theta \text{ TRC } 9)$

The maximum error due to linear interpolation of SIN θ, within an interval of argument of $\pi.2^{-8}$ radians, occurs near $\theta = \pm\pi/2$, but is smaller in magnitude than

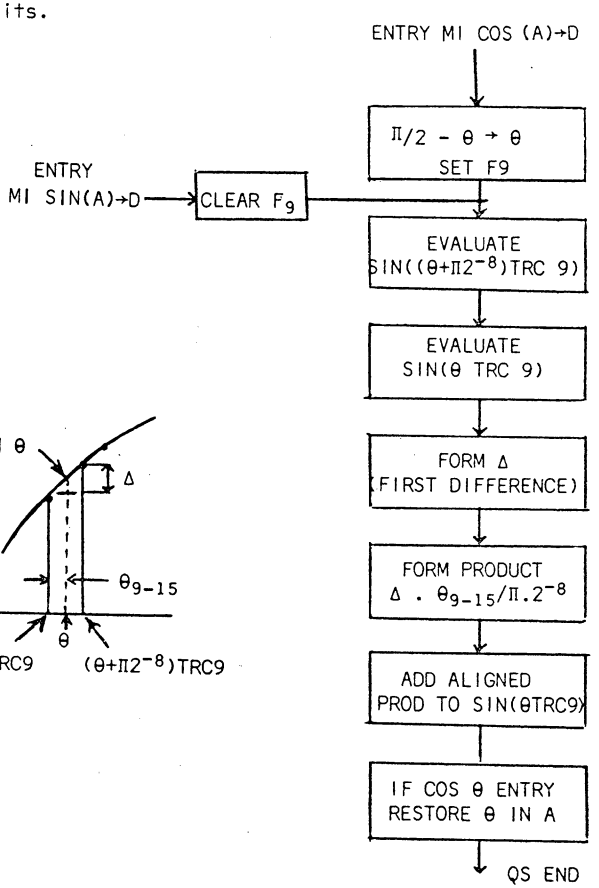
$$(\pi.2^{-9})^2/2 \text{ , i.e., } < \pi.2^{-15}$$

In the vicinity of θ = 0 or π, $|\Delta|$ is almost $\pi.2^{-8}$, i.e., $2^{-6} > |\Delta|_{max} > 2^{-7}$. Thus Δ, coded with LSB = 2^{-14}, may exceed the range of a 2's complement byte by one place, and therefore, to use MI BYT MPY, must first be right shifted one position. Δ/2 then, is chosen as M'IER (LSB of $\Delta/2 = 2^{-14}$).

$\theta_{0-9}/\pi.2^{-8}$ is simply the byte pattern

$0,\theta_9,\theta_{10}..\theta_{15}$, a positive single byte number with LSB = 2^{-7}; it is used directly as M'CAND in MI BYT MPY. The product, $\Delta/2.\theta_{9-15}/\pi 2^{-8}$, has LSB = $2^{-14}.2^{-7} = 2^{-21}$; i.e., the correction $\Delta.\theta_{9-15}/\pi 2^{-8}$ is in register E coded with LSB = 2^{-20}. Therefore, E must be right-shifted 6 places before addition to SIN(θ TRC 9) which has LSB = 2^{-14}.

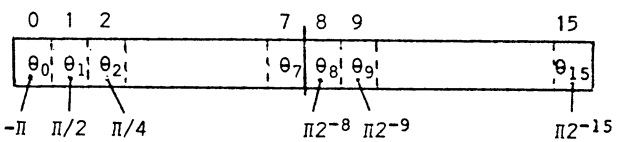
As the error of SIN(θ TRC 9) is $\pm 2^{-15}$, and two calls are made to form Δ, then the error in Δ may be 2^{-14}; also, a further error of 2^{-14} may be introduced by the loss of the LSB in right shifting Δ. This possible error of 2^{-13} multiplied by the maximum value of $\theta_{9-15}/\pi 2^{-8}$, viz 1, gives an error bound of $\pm 2^{-13}$ for SIN or COS θ. This error could be halved by storing the LSB of Δ before shifting, and applying a correction; this has not been done because, for graphical orders, X_c=RSINθ is only required to 12 bits.



INITIAL VALUES

A: ANGLE θ, 2's COMPLEMENT WORD

RANGE $-\pi\ (\pi.2^{-15})\ [\pi-\pi.2^{-15}]$ RADIANS

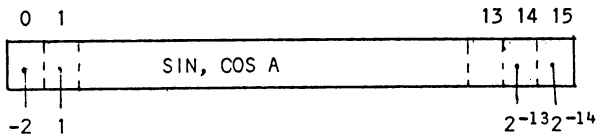


FINAL VALUES

A: AS INITIAL

D: SIN θ or COS θ, 2's COMPLEMENT WORD

MSB = -2, ERROR = $\pm 2^{-13}$



NEEDS

CALLS	NEEDS	DEPTH
SIN(A TRC 9) D	A,D,F8	1Q
SIN(D TRC 9) D	D,F8	1Q
BYT MPY BU*EL→E	B,E,R	NIL
ADDITIONAL FOR SELF	F9	1Q
TOTAL NEEDS & DEPTH	A,B,D,E,F8-9,R	2Q

NO. OF MICRO-ORDERS 21.

MICROCODE

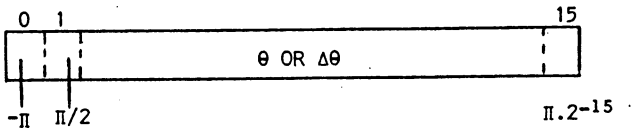
									COMMENTS
1	MI COS(A)->D:	DMC		X'4000'			→	D	π/2 IN D TO FORM π/2-θ
2		ABA			$\overline{A}$	+1	D →	A	π/2-θ REPLACES θ IN A
3		FBC(SF)		1->F9					SET F9 TO RECORD COS(A)→D ENTRY
4		ABA			S+1	+1	→	S	SKIP NEXT ORDER
5	MI SIN(A)->D:	FBC(CF)		0->F9					CLEAR F9 FOR SIN(A)→D ENTRY
6		DMC		X'0080'			→	D	$\pi 2^{-8}$ IN D
7		ABA			A	+0	D →	D	PUT $\theta + \pi 2^{-8}$ IN D
8		DSQ		MI SIN(D TRC 9)->D			→	S	EVALUATE SIN($(\theta + \pi 2^{-8})$ TRC 9)
9		ABA					D →	B	HOLD SIN($(\theta + \pi 2^{-8})$ TRC 9) IN B
10		DSQ		MI SIN(A TRC 9)->D			→	S	EVALUATE SIN(θ TRC 9)
11		ABA			B	+1	$\overline{D}$ (RS)	E	PUT Δ/2 TO EL AS M'TIER FOR MI BYT MPY BU*EL→E LSB = 2^{-14}
12		DMC		X'007F'			→	R	PUT SELECTION FIELD IN R FOR θ_{9-15}
13		ABA			A	Λ	R (8T)	B	θ_{9-15} TO BU AS M'CAND FOR MI BYT MPY BU*EL→E LSB = 2^{-7}
14		DSQ		MI BYT MPY BU*EL->E			→	S	EVALUATE Δ/2 . $\theta_{9-15}/\pi 2^{-8}$, LSB = 2^{-21} i.e., $\Delta.\theta_{9-15}/\pi 2^{-8}$ IN E LSB = 2^{-20}
15		DMC		6			→	R	PUT 6→R | ALIGN CORRECTION
16		ADR	JNZR (RP)				E (RS)	E	RIGHT SHIFT E SIX PLACES | TO LSB = 2^{-14}
17		ABA					E →	B	PUT ALIGNED PROD IN B
18		ABA			B	+0	D →	D	SIN θ TO D
19		DMC	JF9	X'4000'			→	R	IF COSA→D ENTRY, PUT π/2→ R
20		ABA	JF9		$\overline{A}$	+1	R →	A	" RESTORE θ TO ORIGINAL VALUE π/2-(π/2-θ) = θ
21		DQS						NIL	RETURN, END.

DESCRIPTION OF MICRO-ROUTINE 'COS(A+E)→DU, SIN(A+E)→DL'

This routine updates θ in A by Δθ in E (previously aligned to θ by MI READ SML PLR E,N), evaluates COS θ & SIN θ each rounded to byte length, and places them in DU,DL. The routine is a preliminary to plotting small polar vectors, and, since the maximum vector length is 63 display increments, the approximate values COS,SIN(A RND 9), subsequently rounded to one byte, are sufficiently accurate.

INITIAL VALUES

A: ANGLE θ, 2's COMPLEMENT WORD
E: Δθ , " " "



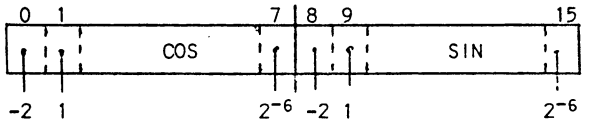
FINAL VALUES

A: AS INITIAL
DU: (COS((θ+Δθ) RND 9)) RND 8 | 2's COMPLEMENT BYTES
DL: (SIN((θ+Δθ) RND 9)) RND 8 | -1 (1/64) 1

MICROCODE

1	MI COS,SIN(A+E)→D:	ABA			A	+0	E	→	A
2		DSQ		MI SIN(A RND 9)->D				→	S
3		ABA					D	→	B
4		ABA(U)	JBe8		B	+1		→	B
5		DSQ		MI COS(A RND 9)->D				→	S
6		ABA					D	→	NIL
7		ABA(U)	JBe8			+1	D	→	D
8		AQS(U)			B			(8T)	D

FINAL VALUES (CONT)



E: AS INITIAL

NEEDS

CALLS	NEEDS	NEST DEPTH
MI SIN(A RND 9)→D	A,D,F8	1Q
MI COS(A RND 9)→D	A,D,F8	1Q
ADDITIONAL FOR SELF	B,E	1Q
TOTAL NEEDS & DEPTH	A,B,D,E,F8	2Q

NO. OF MICRO-ORDERS 8.

COMMENTS

1 UPDATE θ IN A BY Δθ IN E
2 EVALUATES SIN(θ RND 9), RESULT TO D
3 PUT SIN(θ RND 9) TO B & COPY BIT 8 OF SIN TO Be8
4 INC BU TO ROUND SIN TO ONE BYTE
5 EVALUATES COS(θ RND 9), RESULT TO D
6 COPY BIT 8 OF COS TO Be8
7 ROUND COS TO ONE BYTE
8 PUT ROUNDED SIN TO DL, RETURN, END

DESCRIPTION OF MICRO-ROUTINE 'N COS θ→DU, N SIN θ→DL'

This routine comprises two multiplication sequences which form

$$\Delta X = \Delta s \cos \theta$$
$$\Delta Y = \Delta s \sin \theta$$

COS θ, SIN θ, initially in DU,DL are overwritten by the Cartesian components ΔX,ΔY.

INITIAL VALUES

D: COS θ, SIN θ (FINAL VALUES OF MI COS,SIN(A+E)→D)
N: ΔS, POSITIVE INTEGER WORD (BITS 0-9 = 0's)

FINAL VALUES

DU: X = Δs COS θ | 2's COMPLEMENT BYTES, LSB = 1
DL: Y = Δs SIN θ | DISPLAY INCREMENT
N: AS INITIAL

NEEDS

CALLS	NEEDS	DEPTH
MI BYT MPY BU*EL→E	B,E,R	NIL
ADDITIONAL FOR SELF	D,N	1Q
TOTAL NEEDS & DEPTH	B,D,E,N,R	1Q

NO. OF MICRO-ORDERS 10

MICROCODE

								COMMENTS	
1	MI N*COS,N*SIN->D:	ABA(L)	N		(8T)	B		PUT Δs IN BU	COMMON
2		ABA(U)	B		(LS)	B		2Δs IN BU AS M'CAND FOR MI BYT MPY, LSB = 1	M'CAND
3		ABA		D	(8T)	E		PUT COS θ IN EL AS M'IER FOR " " LSB = 2^{-6}	
4		DSQ	MI BYT MPY BU*EL→E		→	S		PUTS Δs COS θ IN E, LSB = 2^{-7}	ΔX→
5		ABA		E	(LS)	E		EU NOW CONTAINS ΔX	DU
6		ABA(U)		E	→	D		PUT ΔX→DU	
7		ABA		D	→	E		PUT SIN θ IN EL AS M'IER FOR MI BYT MPY	
8		DSQ	MI BYT MPY BU*EL→E		→	S		PUTS Δs SIN θ IN E, LSB = 2^{-7}	ΔY→
9		ABA		E	(LS)	E		EU NOW CONTAINS ΔY	DL
10		AQS(U)		E	(8T)	D		PUT ΔY→DL RETURN, END	

DESCRIPTION OF CORE-Q ROUTINE 'LARGE POLAR TO CARTESIAN CONVERSION E,A'

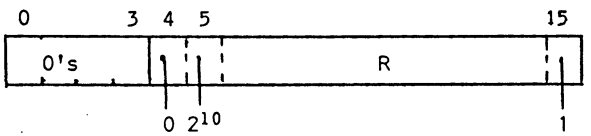
This routine replaces the polar pair R,θ by the Cartesian pair,

$$X_c = R \cos \theta \quad , \quad Y_c = R \sin \theta$$

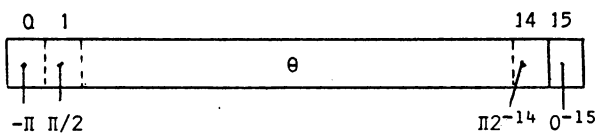
Angle θ is restricted to 15 bits, because the core-store representation of large polar co-ordinates (FORMAT 4.5, p 4.9) requires one bit of the A word to hold L. Also, to code a full 2π radians, the least weight which the MSB of θ can have is π radians. Thus, with these constraints, the finest resolution possible for A is $\pi 2^{-14}$ radians; i.e., approximately 2^{-12} radians. This resolution is adequate, because, when multiplied by $R_{max}(=2^{11})$ it gives a displacement resolution of only one half of a display increment. If θ and R are exact values, the error in X_c or Y_c is controlled by the error in COS θ or SIN θ and has a maximum value of $\pm 2^{11}.2^{-13}$, i.e., $\pm 2^{-2}$ display increments. If θ and R are rounded values, the error in R cos θ is increased by the variability of R and θ. R variations have maximum effect on R cos θ at θ = 0 (max error = $\pm 2^{-1}$), whereas θ variations have maximum effect at θ = π/2, R = 2^{11} (max error due to $\pm\pi.2^{-15}$ in θ, at this R, is $2^{11}.\pi 2^{-15}$, i.e., max error = $\pm 2^{-2}$). The compound error from the three sources is less than ±1 display increment.

INITIAL VALUES

E: RADIAL DISTANCE R, POSITIVE INTEGER WORD
RANGE, 0(1) $[2^{11}-1]$

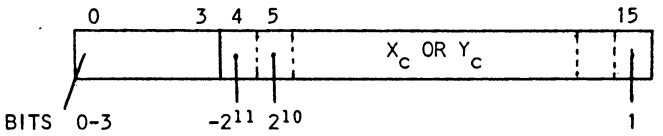


, A: θ, 15-BIT 2's COMPLEMENT WORD



FINAL VALUES

E: $X_c = R \cos \theta$]
A: $Y_c = R \sin \theta$] 2's COMPLEMENT WORDS
$-[2^{11}-1]$ (1) $[2^{11}-1]$; ERROR ±1 MAX.



COPY BIT 4

NEEDS

CALLS	NEEDS	NEST DEPTH
MI SIN(A)→D	A,B,D,E,F8-9,R	2Q
MI COS(A)→D	"	"
MI WRD MPY B*D→ED	B,D,E,R	NIL
MI POP CQ	M,N	NIL
ADDITIONAL FOR SELF	NIL	1Q
TOTAL NEEDS & DEPTH	A,B,D,E,F8-9,M,N,R	3Q ∴ CQ

NO. OF MICRO-ORDERS 10.

MICROCODE

								COMMENTS
1	CQ LRG PLR->CTN E,A:	ABA			E	(LS)	N	TEMP STORE 2R IN N (MI's SIN,COS(A)→D NEED E)
2		DSQ	MI SIN(A)->D			→	S	PUTS SIN θ IN D AS M'IER FOR MI WRD MPY B*D→ED LSB = 2^{-14}
3		ABA		N		(LS)	B	PUT 2^2R TO B AS M'CAND FOR MI WRD MPY (LSB =1)
4		DSQ	MI WRD MPY B*D->ED			→	S	PUTS 2^2R SIN θ (= 2^2Y_c) IN ED (LSB OF D = 2^{-14} ∴ LSB OF E = 2^2 i.e., E CONTAINS Y_c)
5		ABA			E	→	M	TEMP STORE Y_c IN M
6		DSQ	MI COS(A)->D			→	S	PUTS COS θ IN D AS M'IER
7		ABA		N		(LS)	B	PUT 2^2R TO B AS M'CAND
8		DSQ	MI WRD MPY B*D->ED			→	S	PUTS 2^2R COS θ IN ED, i.e., PUTS X_c IN E (ITS FINAL POSITION)
9		ABA			M	→	A	PUT Y_c IN A (ITS FINAL POSITION) CONVERSION COMPLETED
10		DMC	MI POP CQ			→	S	RETURN VIA POP CQ, END

DESCRIPTION OF MICRO-ROUTINE 'PUT $128.2^{MG}\rightarrow N$'

This routine evaluates the number of plotting cycles for MI PLT SML CTN VTR D for magnified small Cartesian vectors. Magnification field MG (= X1, X2, X4 or X8) is held in W6,7 throughout the plotting of a list of vectors.

INITIAL VALUES

W6,7: MG the magnification exponent

FINAL VALUES

N: 128.2^{MG} A POSITIVE INTEGER
W6,7: AS INITIAL

CALLS NIL

NEEDS N,R,W.

NO. OF MICRO-ORDERS 5.

MICROCODE

1	MI 128*2**MG->N:	DMC		X'0300'					→	N	PUT MG SELECTION FIELD IN N (MG IN W6,7)
2		ABA			N	Λ	W	(8T)	R		PUT MG IN R TO COUNT MG LEFT SHIFTS
3		DMC		128					→	N	PUT 128 IN N
4		ADR	JNZR (RP)		N			(LS)	N		128.2^{MG} IN N
5		DQS							NIL		RETURN, END

DESCRIPTION OF MICRO-ROUTINE 'RESET; INTERRUPT; & LOOP'

This routine is the common tail of the eight GRA routines. MI RST;INTRPT;LOOP is entered after each item is plotted: it resets the display co-ordinates to X^1,Y^1 for the reset option; it inserts an interrupt servicing procedure if any peripheral indicator is set, i.e., if JINT = 1; and its final order returns control to the S value held in Q. As the time to plot a list of items under a GRA header is variable and can be long compared to the interrupt servicing interval, then it is necessary to examine requests for service after each item is plotted.

The particular GRA routine which calls MI RST;INTRPT;LOOP does not call via a DSQ order; rather, it preloads Q to a begin-loop address if the current link bit L is 1. Thus, if more display items remain in the list under the particular GRA header, the item plotting part (loop part) of the procedure is repeated: each linked item pushes the Q stack and the loop to the start pops the stack. When L = 0, MI RST;INTRPT;LOOP returns control to an address in the GNI (Get Next Instruction) sequence. (This address is held in Q when GRAFN is first entered.)

INITIAL VALUES

Q: BEGIN-LOOP ADDRESS OF CALLING GRAFN IF L = 1
RETURN ADDRESS IN GNI ROUTINE IF L = 0
Wo: 1 IF RESET OPTION
0 IF END-TO-END OPTION
X^1,Y^1: ORIGINAL X,Y DISPLAY CO-ORDINATES BEFORE GRA HEADER IS INTERPRETED

MICROCODE

1	MI RST;INTRPT;LOOP:	ABA			W	→	NIL
2		TRF	JBe0	$X^1 \rightarrow X, Y^1 \rightarrow Y$			
3		DSQ	JINT	MI INTRPT		→	S
4		DQS					NIL

FINAL VALUES

Q: POPPED
Wo, X^1, Y^1: AS INITIAL
NOTE: MI RST;INTRPT;LOOP IS CALLED BY A DMC ORDER, NOT DSQ

NEEDS

CALLS	NEEDS	DEPTH
MI INTRPT	*	1Q
ADDITIONAL FOR SELF	Wo	1Q
TOTAL NEEDS & DEPTH	Wo, *	2Q

NO. OF MICRO-ORDERS 4

COMMENTS

RESET (Wo) → Be0
IF RESET, PUT X^1,Y^1 → X,Y
IF INTERRUPT, INSERT MI INTRPT
IF LINKED, LOOP TO CALLING GRAFN, ELSE RETURN TO GNI

DESCRIPTION OF MICRO-ROUTINE 'BYTE MULTIPLY BU*EL→E'

Multiplies a 2's complement integer byte multiplicand in BU (M'CAND) by a 2's complement integer byte multiplier in EL (M'IER) to form a 2's complement integer word product in E (PROD).

The algorithm is conventional; i.e., PROD is the sum of partial products, each partial product being either the M'CAND (appropriately shifted for correct significance) or zero depending on the current M'IER bit. Progressive significance of the partial products is achieved by right shifting each accumulation of partial products. The M'IER bits are progressively scanned and destroyed thereby making room for the rightwards expanding product in the accumulator (register E).

Except for the sign bit, the weights of all bits of the M'IER are positive. Thus, the first seven partial products take the sign of the M'CAND. The eighth partial product however, is either zero (positive M'IER) or the negative of the M'CAND (negative M'IER). Micro-order AMY assists the algorithm by,

1. Bringing B conditionally on Be15, a copy of the current multiplier bit in E15.
2. Copying the true sign of the sum of partial products to E0 on right shift, regardless of overflow or underflow.
3. Decrementing R each cycle.

INITIAL VALUES

BU: M'CAND, 2's COMPLEMENT INTEGER BYTE -128 (1) 127
EL: M'IER " "

FINAL VALUES

BU: AS INITIAL
E : PROD, 2's COMPLEMENT INTEGER WORD -2^{15}(1) $2^{15}-1$

CALLS - NIL

NEEDS - B,E,R

NO. OF MICRO-ORDERS 9

MICROCODE

										COMMENTS
1	MI BYT MPY BU*EL->E:	DMC(L)		0				→	B	CLEAR BL
2		DMC(U)		0				→	E	CLEAR EU
3		DMC		7				→	R	LOAD R FOR COUNT OF 7
4		ABA					E	→	NIL	LOAD Be15 WITH LSB OF M'IER
5		AMY	JNZR (RP)		B	+0	E	(RS)	E	FORM & ACCUMULATE FIRST 7 PARTIAL PRODUCTS
6		ABA	JBe15		S+1	+1		→	S	IF SIGN BIT OF M'IER (NOW IN E15 & Be15) = 1 SKIP NEXT MICRO-ORDER
7		AQS					E	(RS)	E	ELSE POSITIVE M'IER ∴ FINAL STEP IS RIGHT SHIFT ONLY, RETURN, END
8		AMY			$\overline{B}$	+0	E	(RS)	E	NEGATIVE M'IER, FINAL STEP IS SUBTRACT M'CAND & RIGHT SHIFT
"										(+0 NOT +1 BECAUSE E15 (NOW '1') IS INCLUDED IN THE SUBTRACTION)
9		DQS							NIL	RETURN, END

DESCRIPTION OF MICRO-ROUTINE 'WORD MULTIPLY B*D→ED'

Multiplies a 2's complement integer word multiplicand in B (M'CAND) by a 2's complement integer word multiplier in D (M'IER) to form a 2's complement double-word product in ED (PROD).

The algorithm is similar to that of MI BYT MPY BU*EL→E except that a 32-bit double register (ED) is required to accumulate partial products. Two orders are therefore required for right shift: the first shift order also accumulates the M'CAND if the current M'IER bit in D15 is 1, the second extends the shift over D (addition is not necessary). This double-word cycle occurs 15 times. The sixteenth partial product is either zero (positive M'IER, sign bit '0' now in D15) or the negative of the M'CAND (negative M'IER, sign bit '1' now in D15) as discussed in MI BYT MPT. Micro-order AMY assists as in MI BYT MPY by, bringing the M'CAND conditionally, maintaining true sign of the accumulated partial products and decrementing R.

INITIAL VALUES

B : M'CAND, 2's COMPLEMENT INTEGER WORD
D : M'IER, "

FINAL VALUES

B : AS INITIAL
ED: PROD, 2's COMPLEMENT INTEGER DOUBLE-WORD

CALLS - NIL

NEEDS - B,D,E,R

NO. OF MICRO-ORDERS 13

MICROCODE

										COMMENTS
1	MI WRD MPY B*D->ED:	DMC		0			→	E		CLEAR E
2		DMC		15			→	R		LOAD R FOR COUNT OF 15
3		ABA				D	→	NIL		LOAD Be15 WITH LSB OF M'IER
4		AMY	JNZR (FW)		B	+0	E	(RS)	E	| FORM & ACCUMULATE FIRST 15 PARTIAL PRODUCTS IN DBL WRD
5		AXS	JNZR (BK)				D	(RS)	D	| ED (ON PASS 15 OF FW-BK PAIR, ORDER 4 IS EXECUTED, R CLEARED &
6		AXS					D	(RS)	D	| 5 SKIPPED. ORDER 6 COMPLETES THE 15TH PASS).
7		DMC	JBe15	LL SUBTRACT M'CAND			→	S		IF M'IER NEGATIVE (D15 & Be15 = 1) GO TO LL SUBTRACT M'CAND
8		ABA				E	(RS)	E		| ELSE M'IER IS POSITIVE & FINAL PARTIAL PRODUCT = 0, .'.
9		AXS				D	(RS)	D		| RIGHT SHIFT DBL WRD ED ONLY
10		DQS						NIL		| RETURN, END
11	LL SUBTRACT M'CAND:	AMY			$\overline{B}$	+1	E	(RS)	E	| SUBTRACT M'CAND (AMY PRESERVES SIGN IF U'FLOW OR O'FLOW),
12		AXS					D	(RS)	D	| & SHIFT DBL WRD ED
13		DQS						NIL		RETURN, END

DESCRIPTION OF MICRO-ROUTINES 'PUSH CORE-Q STACK' & 'POP CORE-Q STACK'

Routines which need to nest micro-routines three deep cannot themselves be called via the hardware stack Q,Q',Q", for, within the deepest microroutine, the Q stack is filled (Q" holds the return address in the outermost calling routine, Q' the return address in the first routine called, and Q the return address in the second routine called). To allow routines to be nested more than three deep, a second stack of return addresses is provided in the core store. This stack "CORE-Q" or "CQ" holds the return addresses from subroutines which either nest MI's three deep or nest other routines which would overflow the hardware stack Q. Such subroutines are called 'CORE-Q ROUTINES' and have the prefix CQ to distinguish them from MI's. Page A3.53 shows the sequence "CALLING SEQUENCE" calling "CQ PLR→CTN E,A" which nests "MI SIN(A)→D", "MI SIN (D TRC 9)→D" and "TABLE SIN(DL)→D". CORE-Q stack is not an automatic extension of Q,Q',Q"; rather, CQ must be pushed by the sequence which calls a CQ routine and popped by the called CQ routine at its termination. MI PUSH CQ is designed to assist the sequence calling a CQ routine: it places S+2 (where S is the ROM address of the DSQ order calling MI PUSH CQ) on top of the CORE-Q stack and increments the CQ POINTER (held in a fixed location 'ADR CQ POINTER' in core) to this top address. Value S+2, not S+1, is placed on top of CQ because S+1 is the return address from MI PUSH CQ (via Q).

At S+1, a DMC order transfers control to the CQ routine being called and, on completion of this CQ routine, CQ is popped and control is returned to S+2. MI POP CQ is designed to assist the called CQ routine pop CQ: it puts S+2 to microprogram sequence register S and decrements the CQ POINTER.

In summary, a calling sequence requires two micro-orders to call a CQ routine: the first, a DSQ, calls MI PUSH CQ; the second, a DMC, calls the CQ routine. No parameters are required, just the entry locations of MI PUSH CQ and the particular CQ or, for symbolic microcode assembly, simply their names. Return of control to S+2 is not the responsibility of the writer of the calling sequence: it is included in all CQ routines, via MI POP CQ. The call to MI POP CQ does not use the hardware Q stack, because return of control after MI POP CQ is to the CALLING SEQUENCE.

MI PUSH CQ

INITIAL VALUES

(ADR CQ POINTER) THE CONTENTS OF CORE LOCATION 'ADR CQ POINTER' = CQ POINTER = i

FINAL VALUES

(ADR CQ POINTER) = i + 1

(CQ POINTER, i.e., i + 1) = S+2 OF THE CALLING SEQUENCE

CALLS - NIL.

NEEDS M,N.

NO. OF MICRO-ORDERS 6.

MICROCODE

										COMMENTS
1	MI PUSH CQ:	DMC		ADR CQ POINTER			→	N		PUT ADDRESS OF CORE-Q POINTER IN N
2		ACN(R0)	JCNB (WT)						NIL	READ CQ POINTER TO M
3		ACN(W0)	JCNB (WT)			+1	M	→	M	INCREMENT POINTER & STORE AT ADR CQ POINTER
4		ABA					M	→	N	PUT ADDRESS OF NEW TOP OF CQ STACK IN N
5		ACN(CW)	JCNB (WT)		Q	+1		→	M	PUT RETURN ADDRESS S+2 (i.e. Q+1) TO TOP OF CQ VIA M
6		DQS							NIL	RETURN, END

MI POP CQ

NOTE:

MI POP IS CALLED BY A DMC MICRO-ORDER, NOT DSQ.

INITIAL VALUES

AS FINAL VALUES OF MI PUSH CQ

FINAL VALUES

AS INITIAL VALUES OF MI PUSH CQ

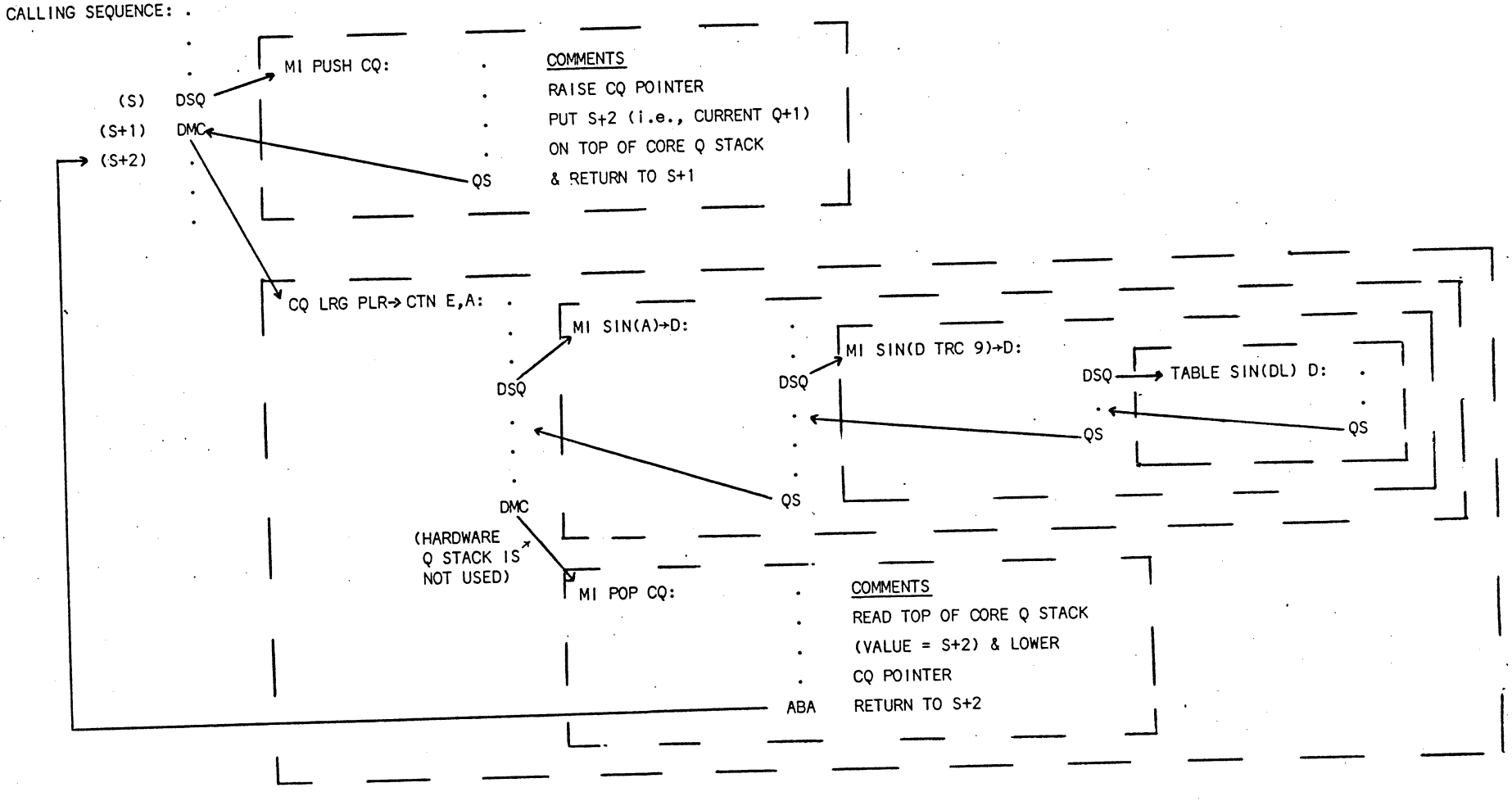
NEEDS

AS MI PUSH CQ

NO. OF MICRO-ORDERS 5.

MICROCODE

										COMMENTS
1	MI POP CQ:	DMC		ADR CQ POINTER			→	N		PUT ADDRESS OF CORE-Q POINTER IN N
2		ACN(RO)	JCNB (WT)					NIL		READ CQ POINTER TO M
3		ACN(WO)	JCNB (WT)		$\overline{0}$	+0	M →	M		DECREMENT POINTER & STORE AT ADR CQ POINTER
4		ACN(RO)	JCNB (WT)			+1	M →	N		RESTORE ADDRESS OF OLD TOP OF STACK IN N & READ RETURN ADDRESS TO M
5		ABA					M →	S		RETURN CONTROL TO S+2, END



CALLING A 'CORE-Q' ROUTINE

DESCRIPTION OF 'SYMBOL TABLE'

This table contains the starting addresses of the microprograms which either plot or execute the command coded by the various "symbols". There are 256 possible symbols. The application of SYMBOL TABLE is described under GRB SYMBOL PAIRS.

INITIAL VALUES

DU: CLEARED

DL: THE SYMBOL CODE

MICROCODE

1	SYMBOL TABLE:	ABA		S+1	+0	D	→	S
2		DMC	SYM 0				→	S
3		"	"				"	"
.		.	.				.	
.		.	.				.	
257		.	255				→	S

FINAL VALUES

D: AS INITIAL

NEEDS

CALLS	NEEDS	DEPTH
SYM 0 - SYM 255		
SELF NEEDS & DEPTH	D	
TOTAL NEEDS & DEPTH	D, SYM 0 - SYM 255	

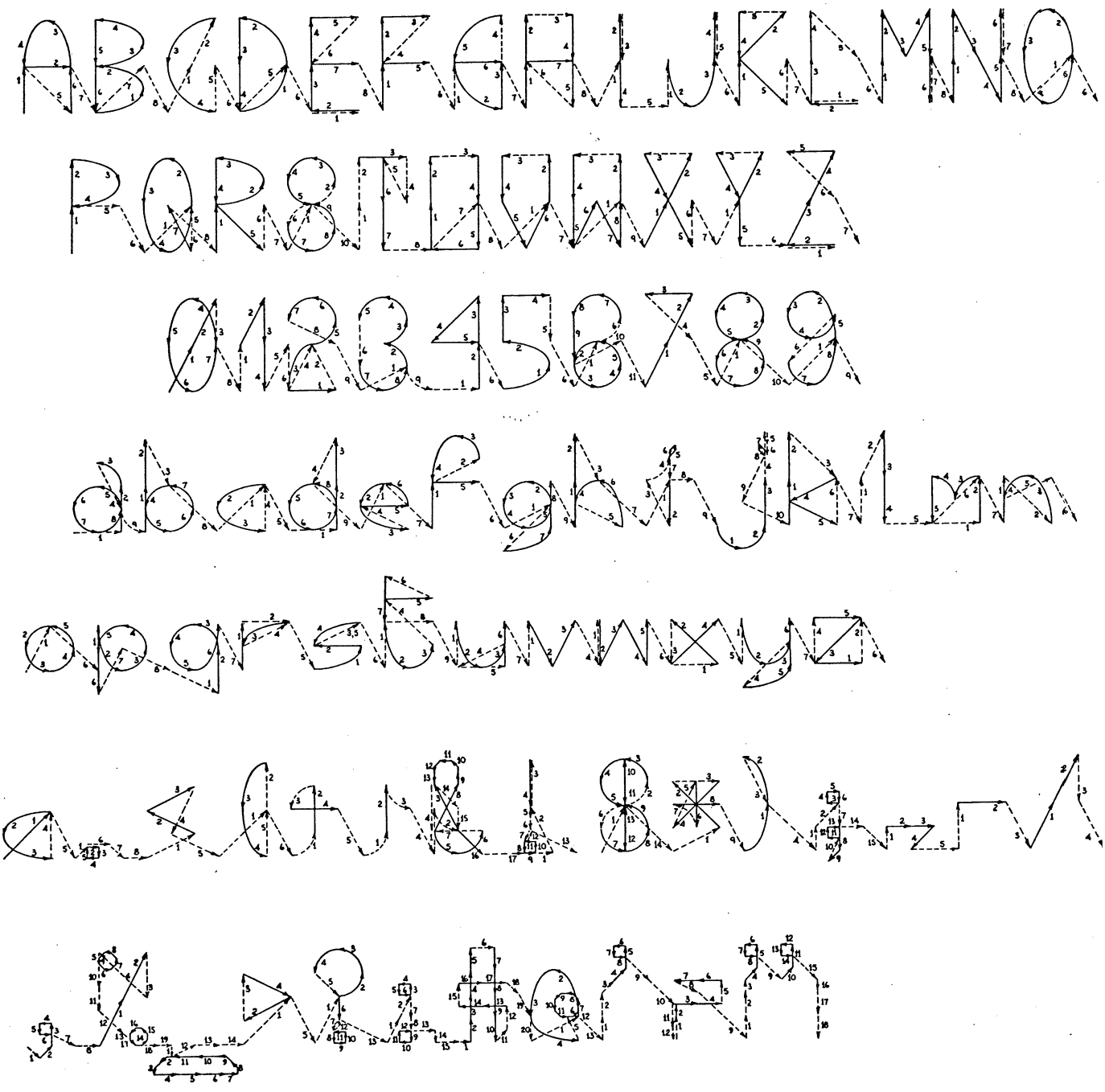
NO. OF MICRO-ORDERS 257

COMMENTS

ADVANCE D + 1 ORDERS			
IF SYMBOL 0, i.e. CONTENTS OF D = 0, GO TO	SYM 0		
" 1 " " = 1, "	1		
. .	.	.	
. .	.	.	
. 255	255	255	

APPENDIX A4

DETAILED COMPOSITION OF 88 GRAPHICAL SYMBOLS



APPENDIX A5

REPRINTS OF AUTHOR'S PAPERS RELEVANT TO THESIS

1. G.A. ROSE, "Economical, graphical-communication techniques for multiple console operation", Third Australian Computer Conf. Proc., pp. 399-402, May, 1966.

2. G.A. ROSE, "Light-pen facilities for direct view storage tubes", IEEE Trans. Electronic Computers, Vol. EC-14, No. 4, pp. 637-639. August, 1965.

3. G.A. ROSE, "Intergraphic - a microprogrammed graphical-interface computer", IEEE Trans., Electronic Computers, Vol. EC-16, No. 6, pp. 773-784, December, 1967.

4. G.A. ROSE, "Computer graphics communication systems", 1968 Edinburgh IFIP Conf. Proc., Invited Papers Section.

Reprinted from: IFIP CONGRESS 68 preprints

North-Holland Publishing Company 1968.

COMPUTER GRAPHICS COMMUNICATION SYSTEMS *

Gordon A. ROSE
Department of Electronic Computation, University of New South Wales, Kensington, N.S.W., Australia

The paper first outlines an early display coupled directly to a computer, and a buffered display with local processor. Three recent schemes which have evolved from these are then reviewed and compared: each aims for low-cost graphical communication within a multi-terminal system; they are the Advanced Remote Display Station II project; the Intergraphic project; and the IBM 1500 Instructional Display System. The paper asserts that the techniques of these recent schemes, supplemented with wired video broadcasting techniques, could be used to link thousands of terminals to a central computer(s) at low cost. A possible configuration is proposed. The significance is that current planning of computer utilities should include extensive computer graphics networks.

1. INTRODUCTION

A number of papers have reviewed the merits and diverse applications of computer graphics [1-3]. Some applications require exacting terminal performance (high resolution, fast response, arbitrary graphics, etc.), while others require only modest performance. Thus, a spectrum of graphical devices linked to either central or peripheral computers has evolved; these installations have demonstrated the versatility of graphical communication. However, despite the desirability of graphical terminals, high per-terminal costs have prevented their widespread use.

This paper reviews three recent economical graphical-communication systems applicable to networks of tens or hundreds of terminals. The systems differ because they place different significance on transmission bandwidth, response time, freedom of graphical form, etc.; collectively, they offer a valuable set of graphical-communication techniques. These advances, as yet mostly in prototype form, show that per-terminal costs can be reduced by at least an order of magnitude (from $50 000 to less than 5 000). The significance is that extensive computer graphics communication systems are now economically feasible.

The first system, the Advanced Remote Display Station II project, is designed for dataphone lines (2000 baud). Its direct view storage tube (DVST) displays eliminate the need for image regeneration, and terminal circuitry expands compact dataphone codes into strings of vector and symbol increments. The approach conserves communication bandwidth and displays arbitrary graphics, but the terminal circuitry is relatively complex and the response time is limited.

The second system, the Intergraphic project, is designed for television distribution. A shared graphical-interface computer generates all terminal images in random point, vector or symbol modes; scan converts them to television format; stores the images on tracks of a video disk recorder; and distributes the images on a single coaxial cable. The approach allows arbitrary graphics to be displayed on simple television terminals, but requires a wide communication bandwidth.

The third system, the IBM 1500 Instructional Display system, is also designed for television distribution. It generates binary-video patterns directly and thereby eliminates the need for scan conversion. Although direct video generation is only practical for restricted graphics, such graphics have wide application.

Before reviewing the three multi-terminal systems, the paper outlines two earlier single-terminal systems. First, the directly-coupled display, because it pioneered later schemes and has set a reference for terminal performance. Second, the buffered display (a single display coupled to a peripheral computer), because it introduced the concept of task division between central and local processors and allowed remote installation; however, one peripheral processor per terminal is expensive.

From the review, the paper outlines a possible configuration for a comprehensive computer graphics communication system.

* Supported by the Australian Research Grants Committee.

2. DIRECTLY-COUPLED DISPLAYS

The earliest displays were directly connected to, and had the total resources of, a large computer. Emphasis was on a working system for graphical communication and not on efficiency or extensive communication networks.

Fig. 1 shows a typical system comprising a central processing unit (CPU); a short-persistence cathode ray tube (CRT) with digital X, Y coordinate registers and digital to analog (D/A) converters; a photo-sensor light pen; and a keyboard, possibly with several overlays. The CPU held the application program, display file and routines for vector and symbol generation and pen tracking. In 1963, Sketchpad [4] was implemented on the TX-2 computer in this general way.

Most of the system functions were programmed and specialized hardware was minimal; thus, the medium was flexible for experimentation, but imposed restrictions on display complexity for a given flicker rate. Display regeneration alone could almost fully extend the computer so that application programs were limited and time-sharing was impossible. However, the direct connection gave rapid access to the application program. Also, the CRT was closely linked to the CPU so that, within the restrictions of vector and symbol generation, a rapid new image rate or dynamic display was possible.

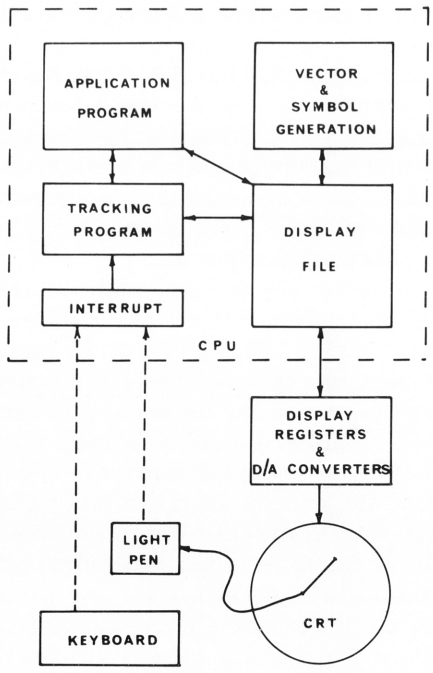


Fig. 1. Directly-coupled display.

The principal inefficiencies were:

1. Regeneration of the display for persistence of vision from the machine core.
2. Software generation of vectors and symbols.
3. Software pen tracking which required multiple probing of the pen field for each incremental movement of the pen.

Experience with these systems pointed out the many functional requirements of graphical communication, the importance of structuring both application and display programs, and those areas where specialized hardware would be advantageous.

3. BUFFERED DISPLAYS WITH LOCAL PROCESSORS: REMOTE INSTALLATION

Vector and symbol generation hardware enabled more complex displays to be presented within a persistence of vision period. Also, the removal of vector and symbol generation programs and tables from the core store conserved core storage. Display core-buffers, local processors and special display logic reduced interrupts on the CPU and freed it for more extensive application programs or other work.

Fig. 2 shows a typical buffered display with local processor. The local buffer might comprise two independent stores (one for the processor, the other for the display logic) or might be a single shared store with the display logic having priority during regeneration (cycle-stealing). GRAPHIC-I [5], SCHOOLER [6] and portion of a project at the University of California, Berkeley [7], are representative. MAGIC [8] uses a cyclic buffer driving simple list processors for local display processing and regeneration.

Remote installation of these buffered displays was possible. This introduced two new design parameters (bandwidth and reliability of the CPU-terminal link) and raised the question of which tasks should be done locally.

Clearly, transmission characteristics, task division between the central and local processors, and local processing ability are inter-related. Relatively low rate dataphone links enable remote stations to be installed at almost any location, but limit the rate of information exchange and require compact transmission codes. Also, the speed mismatch between the CPU and dataphone line commits the CPU either to handle repetitive interrupts or to provide additional buffering. De-

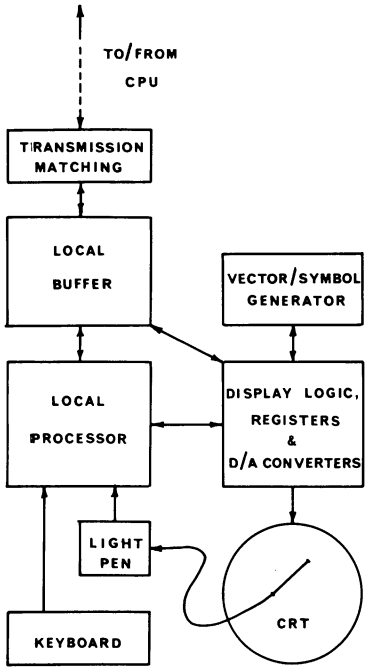


Fig. 2. Buffered display with local processor.

tailed tasks which occur frequently in most application programs, e.g., vector or symbol plotting and pen tracking, should be performed locally; unfortunately, many tasks are not so readily classified.

Although a buffered display with local processor and display logic is less demanding on the CPU than a directly-coupled display and the overall system is more economical, the cost per display is still necessarily high. Therefore, displays with such extensive local ability are not suitable for the terminals of an economical graphical communication system. Their main inefficiency is that considerable hardware is used merely to regenerate static images during delays from the operator or the CPU.

For systems with tens or hundreds of terminals, it would be expected that the total terminal cost would be comparable with the remaining system cost. This has motivated a number of projects aimed at reducing terminal complexity and cost.

4. DATAPHONE LINKED DVST DISPLAYS WITH SIMPLIFIED TERMINAL LOGIC: ARDS PROJECT

The Advanced Remote Display Station II (ARDS II) project [10] has shown that low-cost (potentially $3000 - $5000) terminals for arbitrary graphics can be operated over dataphone lines. Fig. 3 outlines the terminal.

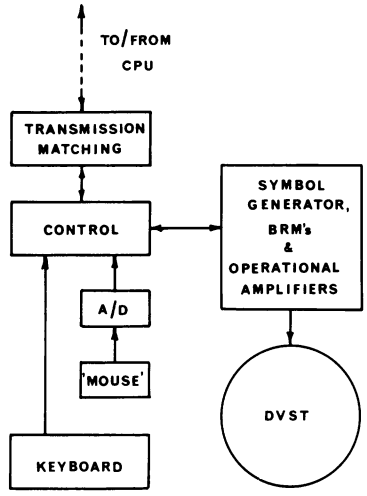


Fig. 3. Advanced remote display station II.

It comprises a DVST; vector and symbol generators; a keyboard; a "mouse" input device (potentiometer pair); deflection circuitry; and simple control electronics which routes incoming words to the vector or symbol generator, and assembles outgoing words from the keyboard or A/D converters attached to the "mouse". Binary rate multipliers (BRM's) produce vector increments which are added as current pulses to integrating operational amplifiers.

Pictorial information, apart from the transient cursor mark which the user positions by moving the "mouse", is accumulated on receipt of digital code from the central processor; thus, reliable digital code transmission is essential. New images require an initial flood erasure. As the dataphone rate is low (up to 2000 baud), even compactly encoded images may need more than five seconds to build up. One side advantage of slow image generation is that the deflection system bandwidth need not be great; however, much wider deflection bandwidths are readily obtained. The DVST and simplified logic combination of ARDS-II is cheaper than the buffered display with local processor typified in section 3, but DVST's are less dynamic than regenerated short-persistence CRT's and cannot use conventional light-pen techniques. Alternatives to the "mouse" input device which are also applicable to DVST's have been reported elsewhere [9].

Simplifying the terminal logic reduces its ability to interpret compact digital codes and so increases the need for information interchange. For example, a string of short vectors, such as a series of segments approximating an arc, is inefficiently encoded in the ARDS-II vector format (five transmitted code words, 50 bits, per vector). More complex logic could accept both short and long vector formats, strings of simple increments and perhaps polar encoded vectors. Alternative coding schemes for digital transmission to remote terminals have been proposed by Bowen [11]. No doubt, further code and functional changes will evolve.

Reduction in ARDS-II costs to the projected $3000 depends upon DVST and integrated array costs (for BRM's, symbol-detail memory, etc.). A further per-terminal cost is introduced at the CPU by the mismatch between the data rates of the CPU and dataphone line as discussed in sect. 3: it is preferable to transmit the display file in bursts at CPU speeds. Developments in integrated electronics which will reduce terminal logic costs will also reduce the cost of centralized logic. Thus, the relative cost of a system having complex terminals to a system having simpler terminals which share centralized logic will probably remain constant. A disadvantage of installing large numbers of terminals having considerable local logic is that transmitted codes and terminal behaviour must be fixed; this would discourage the development of more efficient transmission codes and terminal functions.

5. TELEVISION TECHNIQUES FOR THE STORAGE, DISTRIBUTION AND INPUTTING OF GRAPHICS

Even simpler terminals are possible if picture details are generated centrally and distributed by television: terminal processors and terminal vector and symbol generators are eliminated; standard or monitor quality TV receivers can be used as terminals; and some noise in picture distribution can be tolerated. Simple on-off intensity control (binary video) is adequate for most applications. Video storage disks can store one frame per track and economical multi-track disks are available; grey level storage requires frequency modulation recording, whereas binary video can be stored simply and gives higher horizontal resolution.

The disadvantage of TV distribution is that very high data rates are necessary. This follows because TV coding is far less compact than computer display file coding for most images; e.g., to send 1000 symbols or 500 vectors requires about 10^4 bits of compact computer code, whereas to send a standard TV image with binary intensity requires about 2×10^5 bits. Another disadvantage of TV is that the code length is constant, regardless of the complexity of the image.

However, many terminals can share a common coaxial cable through frequency and/or time multiplexing techniques. Standard TV terminals, each tuned to a different TV channel, can be frequency-multiplexed (in Australia, there are 13 standard channels). Storage TV terminals (DVST or local video storage) can be time-multiplexed on a single frequency channel provided each frame is labelled with a terminal address key [12]. With combined frequency and time multiplexing many TV terminals could share a single coaxial cable; e.g., with 10 channels and a TV frame rate of 25 frames/sec, 500 terminals could receive a new image, on the average, every two seconds. Within the overall system capacity, a few terminals could even display rapid frame sequences.

In many instances, the installation of a single cable, or even a set of cables, is not a major problem, particularly if terminals are grouped as in a teaching laboratory. Already, extensive private communication networks of various bandwidths exist in some organizations [13]. Moreover, extensive cable systems in Great Britain serve about 7% of all television homes [14]. A primary distribution network (video trunk route) of grouped coaxial cables (one for each TV channel), placed in an underground duct, feeds a set of unit areas in a town. Within each unit area a secondary distribution network of multipair cables (a multipair cable contains six wire-pairs for video) serves about 1000 homes. This wired broadcasting system uses multipair or multicoaxial cables rather than wide-band, frequency multiplexed cables as used in community antenna television (CATV) systems. These techniques are not directly applicable to networks of hundreds of independent computer graphic terminals, but they are very relevant. The need for economical local storage of TV images is clearly established.

The user of a TV computer graphics terminal, which has a conventional (non-storage) CRT display, may return information to the computer as follows. A pair of simple counters, running synchronously with the TV raster, can determine the horizontal scan line number and position within the line of a "strike" from a simple photo-sensor light pen ("raster-pen") [12, 15, 16]. The strike pulse freezes the state of the counters; various methods have been used to transmit the count and

reset the counters. Raster coordinates of referenced display items are invariant to terminal display distortion. The raster-pen scheme can be extended to a second smaller "keyboard" CRT which displays standard symbols and a set of function key cells: pen strike coordinates rounded to the grid size correspond to the referenced symbol. Raster-pen counters may be attached to each TV terminal or, alternatively, all terminals may share one pair of centralized counters; these alternatives will be discussed later.

Thus, many economical television techniques already exist for the storage, distribution and inputting of graphics; further techniques, in particular local video storage, need to be developed.

6. ARBITRARY GRAPHICS SCAN CONVERTED TO TELEVISION: INTERGRAPHIC PROJECT

Given the practicality of a large number of simple graphical input/output terminals linked to a centre, it is necessary to generate and maintain the details of all images, and interpret pen positions, at the centre. Much of this work is specialized and should be executed in a dedicated graphical-interface computer closely coupled to the main machine.

This strategy has been adopted in the "Intergraphic" project [17] (fig. 4) * which generates terminal images once only at high speed (10 MHz incremental plotting rates) on a small electrostatically-deflected CRT, scan converts these images, and stores them on specific tracks of a video disk which refresh the terminals. A single coaxial cable distributes images by frequency multiplexing.

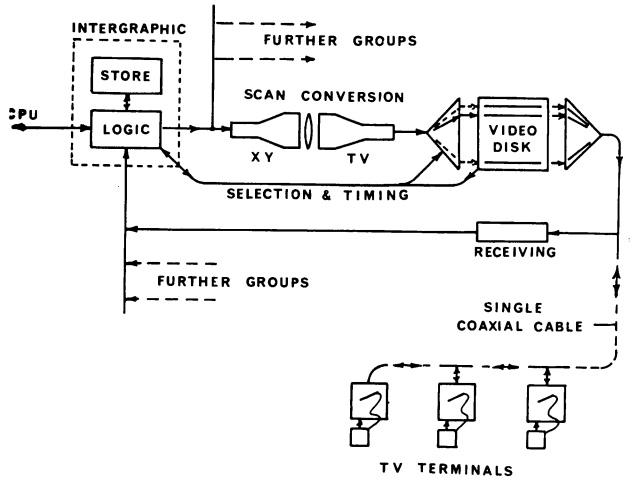


Fig. 4. Intergraphic system.

* Reprinted from [17] by permission of the IEEE.

The graphical-interface computer is extremely versatile (as it is microprogrammed) and fast (3-5 nsec integrated circuits and 100 nsec read-only memory cycle-time), so that complex graphics can be generated well within a TV frame time. Intergraphic has a small conventional core store (8K bytes) and shares the large core store (10^6 bytes) of the main computer; the short data link and the two stores have identical data transfer rates (10^6 bytes/sec).

Vectors and symbols are produced incrementally by short microcode sequences containing several special micro-orders to facilitate plotting. Thus, these functions are executed as normal machine orders and separate vector and symbol generation hardware has been eliminated. As Intergraphic has a powerful machine code (interpreted by 100 nsec microcode sequence steps), blocks of central computer code can be processed before plotting; e.g., scaled, shifted, reflected, etc.

Although the overall function of Intergraphic is dedicated and therefore specialized, the function is complex and will change with operational experience; hence, the approach has been to specialize a versatile structure by microprograms which can be changed from time to time ("firmware" modifications).

Once in every television frame period, each terminal sends its "raster-pen" coordinates and the state of its pen switch to Intergraphic (coordinates are only meaningful when accompanied by a set input switch).

Raster coordinates are determined by a pair of counters attached to each terminal as outlined in sect. 5. Coordinate transmission is distributed over the frame period: X coordinate during one field, Y coordinate during the other. Coordinates are returned on the single coaxial cable by frequency multiplexing; low frequencies are adequate, but redundant coding is necessary because reliability is essential. The terminal pulses which freeze the raster counters (derived from the pen strike pulses) also brighten the CRT beam; this spot, a few mm from the pen aperture, provides true coordinate feedback and eliminates errors due to delays in the pen photosensor response.

In "drawing" mode, pen movement is detected by coordinate differences between successive frame samples; these increments update running parameters of a breakpoint insertion algorithm

held in the interface and shared by all terminals. The breakpoints so determined are sent to the application program - this compression greatly reduces the data flow between the interface computer and the main machine.

In the simplest scan converter mode, an image is written within one frame period and read out to video disk in the following frame period (alternate framing is readily synchronized from the video disk). This gives a distribution rate of $12\frac{1}{2}$ new frames/sec, which, for example, could service 25 terminals with new images on the average every two seconds. By adding further converters, 100 new frames/sec could be distributed before the plotting capability of Intergraphic is exceeded.

Scan conversion is the intermediary between the interface computer and the terminals: the interface computer generates display points asynchronously in incremental and random point modes; the terminals receive display points synchronously in a regular scanning mode.

Direct computer generation of binary video signals for arbitrary graphics is difficult, as even a simple vector becomes a set of discrete points distributed throughout the scanning sequence.

A disadvantage of scan conversion is that to add or delete even a single symbol from a display requires one complete television scan; i.e., an item which is generated in several microseconds commits the converter for at least 40 msec. Thus, it is preferable to add symbols to a display in groups (words), controlled by the space and line return symbols or a special symbol.

Messages of up to one line of text may be displayed by the interface and visually checked before being sent to the application program. This, like breakpoint determination, requires small reserved areas in the interface core for each terminal, but reduces CPU interrupts.

The Intergraphic system is economical because the shared interface computer is well utilized, regardless of the activity patterns at individual terminals; only inexpensive hardware is idle when a particular terminal is inactive.

7. RESTRICTED GRAPHICS BY PROGRAMMED VIDEO: IBM 1500 INSTRUCTIONAL DISPLAY SYSTEM

By restricting the graphical format, binary video signals can be assembled as bit patterns directly by program. This eliminates scan conversion. The IBM 1500 Instructional Display System [16] has a display format of 40 columns and 16 rows: a character (8×12 dot matrix) occupies one column-row intersection, so that the display frame comprises 192 horizontal scan lines each of 320 dot positions. In addition to standard symbols, arbitrary 8×12 characters may be defined and placed in the column-row array to constitute a graphic. It is also possible to displace characters vertically by half a row. Although the scheme is not practical for arbitrary graphics (each new graphic would generally require the detailed specification of many new component characters), it is possible to build a restricted class of graphics from a well chosen set of elements. Restricted graphics are adequate for many applications, e.g., annotated block schematics and flow diagrams.

Individual CRT displays are specially designed TV receivers with improved positional linearity (the TV raster is non-standard). Each frame of video is buffered on one track of a standard IBM 2314 disk store; displays are refreshed 30 times per second at 2.5×10^6 bits/sec over individual coaxial cable links. Further economy is achieved through central determination of light pen raster coordinates in shared hardware. Individual pen input responses are cabled to the centre where they are polled. If the selected terminal has its pen switch depressed, then it has access to the pen locating circuitry throughout the next frame period. The pen strike pulse freezes the raster counters and so records the pen's position. The counters are then reset and run again until frozen in a later frame period by another terminal. The cable delay from a terminal is exactly compensated by sending the TV line synchronizing pulses from the terminal along with the pen strikes: the line pulses start the within-line count at the centre.

In the worst case of a simultaneous request for pen sampling from all 32 terminals, there could be a delay in excess of one second; even this delay is acceptable when pointing to displayed items (the normal use of the pen is this economical system).

In general, shared pen locating hardware cannot follow simultaneous freehand graphical inputs from a number of terminals; rather, each pen needs to be sampled once per frame, which requires a pair of counters per terminal, as discussed in section 6.

For direct video generation by program, a computer must assemble and transfer to video buffer the binary strings which become the TV scan lines. Two schemes are outlined:

1. The computer first forms an image in core of either the whole display or one row of char-

acters. To do this, the dot matrix of a character is read from a table and the matrix rows are distributed individually to appropriate addresses in core. The destination address of the first row of the matrix is determined by the character's location in the display; subsequent rows are placed by address indexing. After the core image is assembled, it is then read and transferred to video buffer in an address sequence corresponding to the TV scan. Thus, this scheme separates image assembly from image transfer.

2. This scheme merges assembly and transfer. The first row of the dot matrix of a character is read from the table and transferred directly to video buffer; then the first row of the next character's matrix is read and transferred, etc. This scheme conserves core storage and is necessarily faster, but it requires a faster memory to synchronize with the video buffer (indexing and indirect addressing within the video time of one dot matrix row).

Clearly, for a given flicker rate, reduced resolution eases both storage and speed requirements. In both of the above schemes the computer is busy for at least the entire transfer phase (i.e., one TV frame period for every new frame generated. The transfer phase is shorter, however, when only one line or character of text is modified.

A scan converter frees the computer from the transfer phase; this is significant when the average time between new frames (i.e., average image generation time plus supervisory overhead) is only a fraction of the TV frame time, as, for example, it is in Intergraphic. The ratio of computer to scan converter costs must also be considered in the justification of scan conversion. Absolute advantages of scan conversion are the acceptance of arbitrary graphics (which allows computer interpolation of compact vector codes, or symbol stroke codes, the latter being more efficient than dot matrix patterns) and the elimination of the need for wordwise synchronization between the computer and the video buffer.

8. PRECISION GRAPHICS

The need for low-cost terminals has introduced displays having less stringent specifications. However, in any comprehensive computer utility, there will always be a need for some displays of the highest quality.

Applications such as template and integrated circuit mask generation or scaled curve following require high resolution, high precision displays. An example is the 4096 × 4096 image processor of the DAC-I system [18]. Such displays require more precise D/A converters, deflection amplifiers and CRT's, but present no inherent programming difficulties. Precision CRT's are mostly small (3-5 " diam) and are not viewed directly; rather, they record onto, or scan through, photographic film. The greater precision usually requires a longer settling time for the CRT beam.

9. A PROPOSED EXTENSIVE GRAPHICAL-COMMUNICATION SYSTEM USING EXISTING TECHNIQUES

The foregoing sections have outlined and compared a variety of existing techniques for linking graphical terminals to a central computer. This section outlines a configuration for a comprehensive graphical communication system based on these existing techniques.

Per-terminal costs of multi-terminal systems have been greatly reduced by one or more of the following:

1. Imposing operational constraints such as restricted graphics, lower resolution or precision, and slower plotting rates or response times.
2. Introducing hardware which is shared by terminals; e.g., graphical-interface computers, centralized raster-pen circuitry and centralized scan converters.
3. Converting images expressed in computer based codes to forms suitable for low-cost distribution; e.g., scan conversion of vector strings to video signals.
4. Storing images at each terminal.

Some applications cannot accept these constraints or devices. Therefore, for overall economy of a comprehensive graphical-communication system, several classes of terminals are necessary.

Class 1 - *Dynamic Precision Terminals*

Dynamic precision displays for animated movie generation would be few in number and each display would be connected directly to a private graphical interface computer. These units would be expensive and shared on an hourly or daily basis. Some laboratory applications require this class of display, e.g., stimulus-response experiments on the visual system which require computer generation of visual stimuli, and analysis of responses from many sensors.

A high precision display for non-repetitive template or mask generation could be located remotely and linked to the interface by a reliable low data rate line which transmits coded display increments.

Class 2 - Television Terminals

Standard television terminals driven from an interface which sends an arbitrary image to each terminal on the average every few seconds would be satisfactory for most applications: the resolution is adequate to detail 1000 symbols; the speed is more than adequate for contemplative reading – the reserve speed allows page scanning, or even faster sequences, at a few terminals. Arbitrary graphics are essential for some applications; also, they can be economically generated and converted to video by an interface which contains a shared scan converter(s).

The following example describes an interface serving 200 standard TV terminals with new images on the average every two seconds. (This is claimed for Intergraphic and an order of magnitude increase is predicted [17]). Five scan converters would be necessary to distribute the 100 new frames/sec. Each converter would transfer 20 new frames/sec to a specific set of 40 tracks on a video buffer refreshing terminals at 25 frames/sec (Australian standard). As soon as a scan converter was written (average writing time 10 msec), the read beam, scanning synchronously with the video buffer, would be enabled and would start transferring the image to the video buffer. A counter would time the transfer to last for exactly one frame period (40 msec), and on completion of the frame transfer, the converter's busy indicator would be cleared. During this transfer, the interface would generate an image on the next converter for a terminal in the next group of 40. This overlapping sequence, regulated by the busy indicators, would continue.

Any set of 40 terminals which is physically clustered would be driven directly from a video buffer located within the cluster - one coaxial cable or twisted wire-pair from each buffer track to its terminal. If the cluster was remote from the interface, one time-multiplexed cable from the converter to the buffer would be sufficient: a track address key would herald each video frame transferred from the converter. Scattered terminals would be frequency multiplexed on a single coaxial cable, rather than radially fed, from the video buffer. Isolated remote terminals have been linked by video wire circuits [14]; however, if this is impractical then a dataphone DVST terminal would be necessary.

Class 3 - Dataphone DVST Terminals

Existing dataphone circuits would dictate the linking of these terminals (ARDS type). A special interface which absorbs the speed mismatch between the central processor and the dataphone lines would be desirable.

Proposed network

A large computer graphics network (fig. 5) would comprise say ten graphical interface computers in two groups of five, each group connected to one of two very large computers. The large computers would be coupled for reliability and load sharing. Within each group of five interfacees, one interface would drive a Class 1 terminal, three interfaces would drive about 600 Class 2 terminals, and the fifth interface would serve 1000 Class 3 terminals with new frames on the average every ten seconds. With faster interfacces and low-cost local storage, it will become practical to time multiplex thousands of simple Classs 2 terminals, directly from banks of scan converrters. With such large systems a careful reassessment of frame formats, scanning times, etc., would be necessary.

10. CONCLUSIONS

This review shows that many economical graphical communication techniques exist, and further developments promise to reduce per-terminal costs to below $1000. Apart from reduced costs of micro-electronics generally, lowcost has been achieved by:

1. Borrowing from non-computer fields (radar, television, etc.) techniques such as scan conversion, television image transmission, frequency multiplexing and video signal storage.
2. Classifying applications into broad classes according to terminal performance requirements, and providing matching terminal typees (high-precision, high-resolution; standard television; and dataphone connected DVST's).
3. Improving systems organization, e.g., introducing shared graphical-interface computers which allow more efficient task distribution, reduce central processor interrupts and interpret more compact data codes.
4. Improving logical design, e.g., plotting vectors and symbols by high-speed microcode, which eliminates the need for special vector and symbol generation hardware.
5. Replacing core-store image regeneration by video-disk image refreshing or storage-tube image retention.

The significance of these advances is that current planning of computer utilities should include

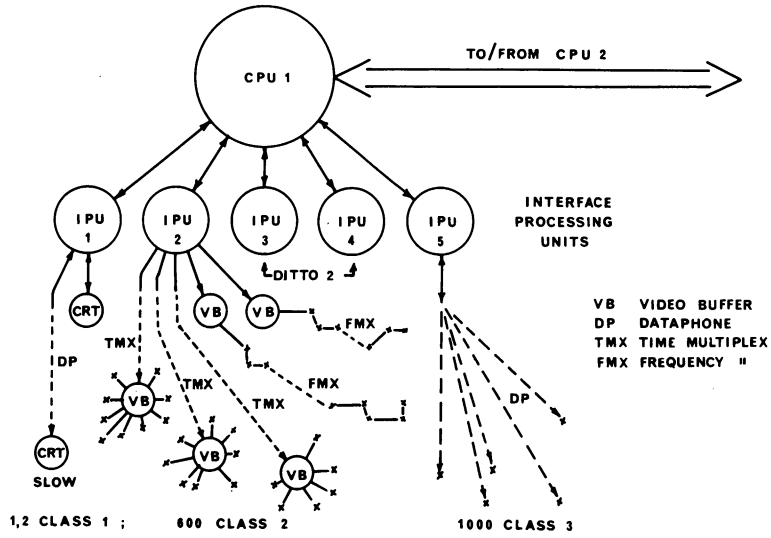


Fig. 5. A large computer-graphics network.

extensive computer graphics networks. Television systems are particularly attractive, and excellent examples of economical video distribution serving thousands of simple terminals exist.

Further development of local image storage is required to exploit time multiplexing of common distribution channels. Fortunately, advances in graphical communication, which ultimately reflect an even greater load on the central processor, have been accompanied by advances in the understanding and processing of complex data structures [19, 20].

ACKNOWLEDGEMENTS

The author wishes to thank Prof. M. W. Allen, Prof. P. D. Jones, T. Pearcey, M. Macaulay, C. J. Barter, G. P. Bowen and R. B. Stanton for many helpful discussions on systems organization, data structures, computer graphics and hardware. The constant support of Prof. M. W. Allen, Head of the Computation Department, is greatly appreciated.

REFERENCES

[1] M. D. Prince, Man-computer graphics for computer-aided design, Proc. IEEE, vol. 54, no. 12, pp. 1698-1708, December 1966.

[2] E. L. Jacks, A laboratory for the study of graphical man-machine communication, 1964 Fall Joint Computer Conf., AFIPS Proc., vol. 26, pp. 343-350.

[3] M. H. Lewin, An introduction to computer graphic terminals, Proc. IEEE, vol. 55, no. 9, pp. 1544-1552, September 1967.

[4] I. E. Sutherland, SKETCHPAD, a man-machine graphical communication system, 1963 Spring Joint Computer Conf., AFIPS Proc., vol. 23, pp. 329-346.

[5] W. H. Ninke, GRAPHIC-I, a remote graphical display console system, 1965 Fall Joint Computer Conf., AFIPS Proc., vol. 27, Pt. I, pp. 839-846.

[6] N. A. Ball, H. Q. Foster, W. H. Long, I. E. Sutherland and R. L. Wigington, A shared memory computer display system, IEEE Trans. Electronic Computers, vol. EC-15, pp. 750-755, October 1966.

[7] R. W. Lichtenberger and M. W. Pirtle, A facility for experimentation in man-machine interaction, 1965 Fall Joint Computer Conf., AFIPS Proc., vol. 27, Pt. I, pp. 589-598.

[8] D. E. Rippy et al., MAGIC, a machine for automatic graphics input to a computer, ibid., pp. 819-830.

[9] G. A. Rose, Light-pen facilities for direct view storage tubes, IEEE Trans. Electronic Computers, vol. EC-14, pp. 637-639, August 1965.

[10] R. H. Stotz and T. B. Cheek, A low-cost graphic display for a computer time-sharing console, Society for Information Display, 8th National Symposium, pp. 91-97, May 1967.

[11] G. P. Bowen, M. E. Thesis, 1968, Contributions to the UNSW graphical data network, School of Electrical Engineering, University of New South Wales, Kensington, N. S. W., Australia.

[12] M. Macaulay, M. E. Thesis, 1968, Input/output terminals for the computing utility, ibid.

[13] J.C.McPherson, Data communication requirements of computer systems, IEEE Spectrum, pp.42-45, December 1967.
[14] R.P.Gabriel, Wired broadcasting in Great Britain, IEEE Spectrum, pp.97-105, April 1967.
[15] S.B.Gray, A computer time-shared display, Information display, January/February, 1966, pp.50-51.
[16] R.A.Aziz, An instructional display terminal, Society for Information Display, 8th National Symposium, pp. 83-90, May 1967.
[17] G.A.Rose, Intergraphic - a microprogrammed graphical-interface computer, IEEE Trans., Electronic Computer, Vol. EC-16, No.6, Dec. 1967.
[18] B.Hargreaves et al., Image processing hardware for a man-machine graphical communication system, 1965 Fall Joint Computer Conf. AFIPS Prooc., vol. 26, pt.I, pp.363-386.
[19] J.C.Gray, Compound data structures for computer-aided design: A survey, Assoc. for Computing Machinery, Proc. 22nd National Conf., 1967, ppp. 355-365.
[20] D.T.Ross, The automated engineering design (AED) approach to generalized computer-aided design, Proc. 22nd National Conf., 1967, pp. 367-385.

Reprinted from IEEE TRANSACTIONS
ON *ELECTRONIC COMPUTERS*
Volume EC-16, Number 6, December, 1967
Pp. 773-784

"Intergraphic," A Microprogrammed Graphical-Interface Computer

GORDON A. ROSE

Abstract—**The paper describes a proposed microprogrammed interface computer, "Intergraphic," which will link many (initially 13; potentially more than 50) general-purpose graphical terminals to a central processor. Intergraphic will generate new images once only, at high speed (10-MHz incremental plotting rates) on one of several small, electrostatically deflected, precision CRT's. The images will be generated on a 1024 by 1024 grid in incremental and random-point display modes by fast microcode sequences which interpret display lists from the central processor. The centrally generated images will be scan converted to standard television video signals and recorded on a multitrack video disk(s), each track refreshing a low-cost standard television terminal. User feedback will be via raster coordinates, determined by a light-pen ("raster-pen") and simple counting circuits at each terminal.**

The paper concentrates on the digital interface structure which is versatile and fast (3–5 ns integrated circuits and 100-ns cycle-time READ-only memory). The description centers on display generation, although the structure is largely general purpose. The proposed operating system is outlined only since it is in an early stage of development. Also, the order-code accompanying data from the central processor is incomplete. However, new orders can be readily interpreted by adding microprograms to the READ-only memory (i.e., "firmware," not hardware extensions); thus, the interface will also be a useful medium for experimentation in graphical structures and communication.

Index Terms—**Compact graphical codes, computer graphics, economical displays, graphical interface, man–machine communication, microprogrammed interface.**

I. INTRODUCTION

BEFORE 1964, a number of projects[1]–[3] had shown that a computer-driven display with an input pen was a useful and versatile terminal. These early systems, however, were devised to meet a particular need or to demonstrate the potential of computer graphics; system or hardware efficiency was not of first consideration and, typically, single displays were connected directly to a central computer.

By 1964, a multiaccess system for about 30 simultaneous users was being realized at Project MAC.[4] Terminals were electromechanical teletypes; two years later Corbató and Vyssotsky[5] considered general-purpose graphical terminals highly desirable, but still too costly and too demanding to be multiplexed in quantity.

At the same time, other projects[6]–[8] showed that local control computers with regeneration stores, which generated and held display details, determined pen coordinates, etc., could reduce demands on the central computer. However, supporting hardware in each of these projects was for one display only; thus, although central computer time was saved and the display with its local support could be moved to remote locations, the hardware cost per display was still high. Another single display project, SCHOOLER,[9] had aims similar to those presented in references [6] and [7], but eliminated the need for a separate regeneration memory by allowing the display unit to borrow cycles from the local control core memory. Also in late 1966, Kennedy[10] described an operating system for a set of three Digigraphic displays. Relatively static displays were refreshed from the drum and, where necessary, highly dynamic displays could be driven directly from the central computer.

A typical specification for the general-purpose displays cited would be a magnetically deflected, large CRT with X, Y deflection resolutions of 1 in 1024, a random position settling time of 30 μs, and an incremental plotting rate of 1.5 μs per visible point or 300 ns per

Manuscript received January 20, 1967; revised June 24, 1967. This work was supported by the Australian Research Grants Committee.

The author is with the Dept. of Electronic Computation, University of New South Wales, Kensington, New South Wales, Australia.

blanked point. A symbol generator (20 μs per symbol) and vector generator (150 μs/in) would usually accompany each display.[1]

Plotting rates an order faster (2.5 μs per random point or symbol; 4 μs/in vector rate) are possible using electrostatically deflected CRT's with vector and symbol generators which are part analog.[2]

Costs of such displays (electromagnetic or electrostatic deflection) with vector, symbol and regeneration devices are high (typically in excess of $50 000); in contrast, per terminal costs not exceeding $2000 would be expected before large numbers (say, more than 50) of graphical terminals are multiplexed.

A previous paper on economical graphical communication[11] discussed a number of inefficiencies in the encoding, displaying, and inputting of arbitrary graphics. It stressed the need for an interface computer, shared between graphical terminals, which removed the tasks of detailed generation, simple graphical manipulation, and pen tracking from the central computer, and proposed a microprogrammed interface computer, "Intergraphic."

This paper details the structure and performance specifications of Intergraphic, planned for operation by the end of 1967. Initially, Intergraphic will link 13 graphical terminals to a central computer (IBM 360/50) via a large capacity core store (LCS). Fig. 1 shows the proposed system.

Intergraphic will generate all user images once only (i.e., regeneration for persistence of vision is regarded as a low-order task which should be external even to the interface computer) at high speed on a single, small, electrostatically deflected CRT in conventional computer-driven display modes.

Each image will be scan converted to a standard TV frame and stored on a video disk (one track refreshing one active terminal). An economical TV distribution system, described by Macaulay,[12] will then continually distribute the images to corresponding standard TV receiver terminals on a single coaxial cable. Macaulay also describes simple digital circuits attached to each terminal which will return pen raster coordinates to Intergraphic on the same cable once in every TV frame period (Australian standard, 40 ms).

The present paper concentrates on the digital interface structure which is extremely versatile (being sequenced from microprograms held in a READ-only memory, ROM) and fast (3–5 ns integrated circuits and 100-ns ROM cycle time).

The paper gives examples of the code which will accompany compactly encoded data between LCS and Intergraphic (Section IV), and the interpretation of this code by microprogram sequences (Section V). The flexibility and speed of Intergraphic will enable many functions to be performed within it; e.g., Cartesian and polar vector strings will be interpreted and plotted by microcode. The paper describes two micro-orders which assist vector and symbol generation; through these orders, the incremental plotting rate of vectors and symbols will be 10 MHz.

Compact image encodings will be buffered in Intergraphic core store at a data rate matching that of LCS; viz., 1 byte (8 bits) per μs. For example, a page of text of 1000 symbols or a line drawing comprising 500 short vectors (short Cartesian vectors have X, Y component magnitudes less than 64 increments) will occupy 1000 bytes and require a core-to-core transfer time of 1 ms. The generation time of such displays will be approximately 5 ms; thus, the elapsed time from LCS to generation on the central CRT will be only a fraction of a scan conversion period (40 ms), thereby leaving the interface considerable free time for other functions. Later, further scan converters will be added. It will be possible for Intergraphic to generate 100 new frames per second. However, operational experience with the initial 13 terminals will be necessary to assess the overall system capability before deciding on the final number of terminals. Highly dynamic displays have a high new image rate; thus, they require direct coupling to Intergraphic rather than to a video track. Such displays will be driven identically to, but in place of, the writing section of a scan converter.

A detailed description of the proposed operating system is not within the scope of this paper. Many operational details will remain experimental; others will follow established multiprogramming procedures. However, to complete an outline of the overall scheme, a brief description follows.

User programs will be expressed in PL/1 with embedded graphical orders. PL/1 orders will be executed in the central processor (CP); graphical orders, however, will be interpreted partly by CP subroutines and partly by Intergraphic (IG). For example, "plot rectangle $a \times b$, bottom left corner x_0, y_0" would be expanded into a list header and four vectors by CP subroutine and the list executed (in this case, plotted) by IG.

Communication between the CP and IG will be buffered by two queues in LCS: one, the CP/IG queue, will queue tasks to be executed by IG (EXIG's); the other, the IG/CP queue, will queue data resulting from the execution of EXIG's. The CP/IG queue will be built up by the component of the operating system resident in the central processor (OS/CP) and serviced by the component of the operating system resident in Intergraphic (OS/IG). Examples of EXIG's are "display an output graphic," "post-process, then display a graphic," and "track an input graphic and encode as a vector string." The IG/CP queue will be built up by OS/IG and serviced by OS/CP; examples of data in this

[1] DEC Type 338 Display, Digital Equipment Corp., Maynard, Mass., and Elliott Type 4100 Display, Elliott-Automation Computers Ltd., Borehamwood, Herts, England, are representative.

[2] CDC Type 250 Display, Control Data Corp., Minneapolis, Minn.

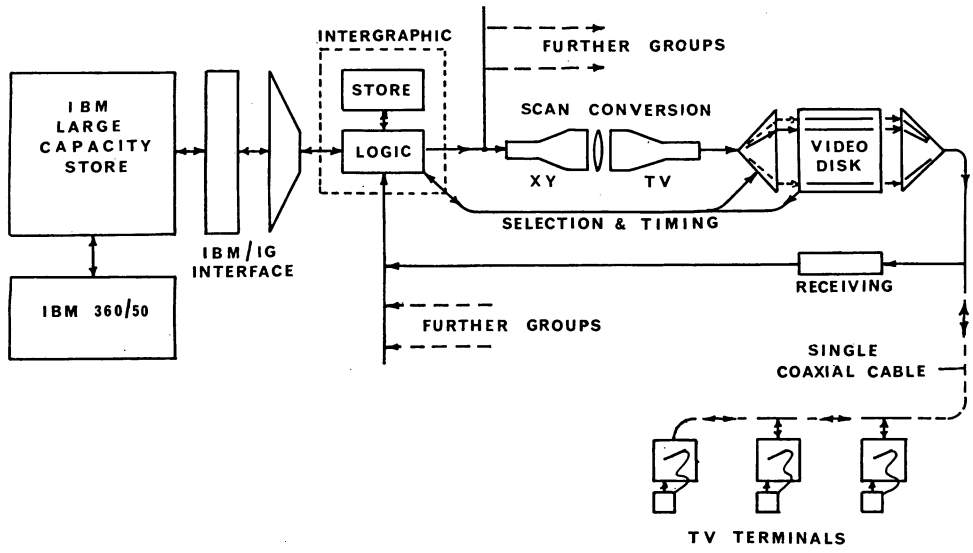


Fig. 1. Proposed graphical-communication system.

queue are "a string of symbols from a terminal," "a string of vectors encoding an input graphic," and "an item pointed to by the user." Only one EXIG will be resident in the Intergraphic core at any one time.

In summary, apart from an initial request for service, all terminal activity will be initiated by user programs resident in the central computer; i.e., Intergraphic and the terminals are regarded as a flexible input-output device, the inputs coming from any one user being in modes prescribed by his program.

II. Justification of System and Interface Structure

An interface which multiplexes graphical terminals must be capable of a variety of processing and supervisory tasks. Some of the tasks planned for Intergraphic have been mentioned in Section I.

Many interface tasks are purely digital. Others, e.g., symbol, vector, or special curve generation, may be digital or analog.[13],[14] The extent of analog techniques is an important engineering decision; other major decision areas are local or central generation devices, storage or regenerated displays, and, if regenerated, digital or video storage.

For a multiple terminal system to be economical, the cost of a terminal must be low. The cost of a standard TV receiver refreshed from a single track on a multitrack video disk is currently lower than that of a direct view storage tube (DVST) display. Moreover, a group of TV receivers can be readily multiplexed on a single coaxial cable.[12]

Although it is possible to compute video patterns directly for some restricted display formats, e.g., typewriter mode, direct video construction of arbitrary graphics is generally difficult. It is simpler to construct video patterns indirectly by scan converting a display previously generated in vector or random point modes. Many terminals can share a single scan converter, provided the average new frame rate per terminal is adequate. A converter frame rate of 12.5 per second, for example, could support 25 terminals with new frames, on the average, every 2 seconds. Video refreshed TV receivers have a constant display frame rate. Thus, flicker is independent of the new frame rate and is negligible.

Alternatives to scan conversion and video regeneration are

1) DVST terminals with no intermediate storage between the interface and terminals.

a) *Scheme 1:* Common analog deflection signals to all terminals with terminal selection by intensity gating—the image generation rates of the interface would be limited by the bandwidth of the deflection system for the large DVST's.

b) *Scheme 2:* Common digital image codes sent to all terminals with terminal selection by coding keys—compact codes would require elaborate vector and symbol generators at each terminal; expanded codes, generated by the interface, could reduce terminal decoding complexity. Reliability of code transmission is important. As in Scheme 1, the image generation rate of the interface would be limited by deflection bandwidth.

2) DVST terminals written in TV mode from a scan converter. The intermediate storage of the scan converter allows the standard TV deflection bandwidth to be used. Intensity signals need be transmitted once only, thereby eliminating the high information rates necessary for refreshing even static images on short persistence displays.

3) Digital regeneration, i.e., the interface generates and routes digital codes once to core or synchronous (drum, disk, or delay line) stores, which are cycled to regenerate displays. Compact digital codes are preferred for storage, but, as in Scheme 2, compact codes increase terminal decoding complexity. A central regen-

eration store causes a high total digital transmission rate from store to terminals which complicates transmission; individual terminal regeneration stores reduce the central to terminal transmission requirements, but the cost of scattered storage is greater. Core regeneration stores can be filled at a rate matching the interface, but cycled more slowly to match terminal performance. This is not possible with synchronous storage, although several intermediate core buffer areas, written at the interface rates but read more slowly into synchronous regeneration stores once only, could absorb this mismatch.

Recent work on meshless DVST's[15] and bistable plasma arrays[16] offers local storage devices at potentially low cost. At the moment, however, the chosen scheme is cheaper than the alternatives listed.

Central generation of images in the chosen scheme eliminates the need for multiple copies of vector and symbol generators. Thus, detailed vector and symbol generation become tasks within the interface. From the variety of tasks expected of the interface, it was decided to control a general-purpose digital structure by microprogramming.[17],[18] This reduces the amount of special-purpose hardware, and allows the system to be readily extended or modified. It has been possible to generate vectors efficiently within the interface entirely by microprogram with the assistance of a special micro-order; the additional logic required to execute this order is trivial. Symbol generation also fits well into the microprogram structure, but a small amount of logic to generate standard strokes has been necessary. Apart from the ROM and core store, Intergraphic is being constructed entirely of integrated-circuit OR–NOR gates, analog signals first appearing at the outputs of the digital-to-analog (D/A) converters.

Microprogramming a structure of OR–NOR elements is a unified approach which has been preferred to the construction of a number of special-purpose devices using a variety of techniques. Moreover, the paper shows that high speeds are possible with the chosen structure and techniques. Using the same codes and logical structure, speeds an order of magnitude greater (100 MHz increment rates) are feasible. This is so because subnanosecond gates are becoming available, the ROM and the D/A converters use nonsaturating techniques which can be extended to high frequencies, and very high frequency CRT electrostatic deflection structures already exist.

III. Interface Logical Structure and Microcode

A. General Description

Figs. 2 through 4 show the interface logical structure, and Table I sets out the controlling microcode. Throughout the description, "register" implies a set of flip-flops.

The general-purpose processing section (Fig. 2) comprises a core store, addressed from N with READ/WRITE register M; a set of general registers (A, B, $\cdots$, E); register selection gates; a pair of eight bit arithmetic units (AU's) with elementary shifting, circulating, and transposing facilities; and an internal distribution bus e.

The control section (Fig. 3) comprises an ROM (holding microprograms), addressed from S with READ register G; a microroutine link register Q; decoding and timing logic; a general-purpose counter R; and a bitwise programmable register F.

The special-purpose section (Fig. 4) comprises X, Y display registers and D/A converters; X', Y' origin registers for resetting X, Y in absolute vector mode (see Section IV); X_m, Y_m matching registers for detecting a particular X,Y pair; line-form (i.e., continuous, dotted, or broken lines) and intensity-control circuits; and symbol-stroke generation logic.

The core store is conventional; 4096, 16-bit words, cycle time 1.5 μs. It buffers compact display codes and instructions to and from the central computer. The ROM, however, is fifteen times faster; during its cycle time (100 ns) one micro-order is executed. This speed allows compact core codes to be interpreted by microcode sequences, often without slowing the core store.

B. Microcodes (Outline)

The microcode will be outlined only here; details are contained in the Appendix.

There are six micro-order types (five are shown in Table I).

1) The *arithmetic/logical* orders (ABA-APV, or briefly, A-orders). These others add, or perform a logical operation on, the contents of two nominated registers and send the result to a third nominated register. Two's complement representation is implied throughout the paper. Most of the machine registers are included, so that many functions can be performed with these orders; two examples are the core address being indexed by the contents of another register, and the ROM address being set to a computed jump. A single bit shift or circulate, or a 4-bit or 8-bit transpose of the result, are optional. Operations may be performed on lower significant bytes of registers only, on upper bytes only, or on both bytes considered either isolated or concatenated. As well as the conventional execute and advance mode, the execution of an A-order may be delayed or repeated until a nominated event occurs, or the next order may be taken from ROM address $S-1$, not $S+1$ (this backstep option allows alternations of two orders until a nominated event occurs). ADR, AMY, $\cdots$, APV are variants of the basic A-order, ABA; they are detailed in the Appendix.

2) The *distribute microprogram constant* orders (DMC, DQS). These orders place 8- or 16-bit constants in a nominated destination register.

3) The *register transfer* order (TRF). This order executes register-to-register transfers which are not possible via the arithmetic unit.

4) The *F register bitwise control* order (FBC). The

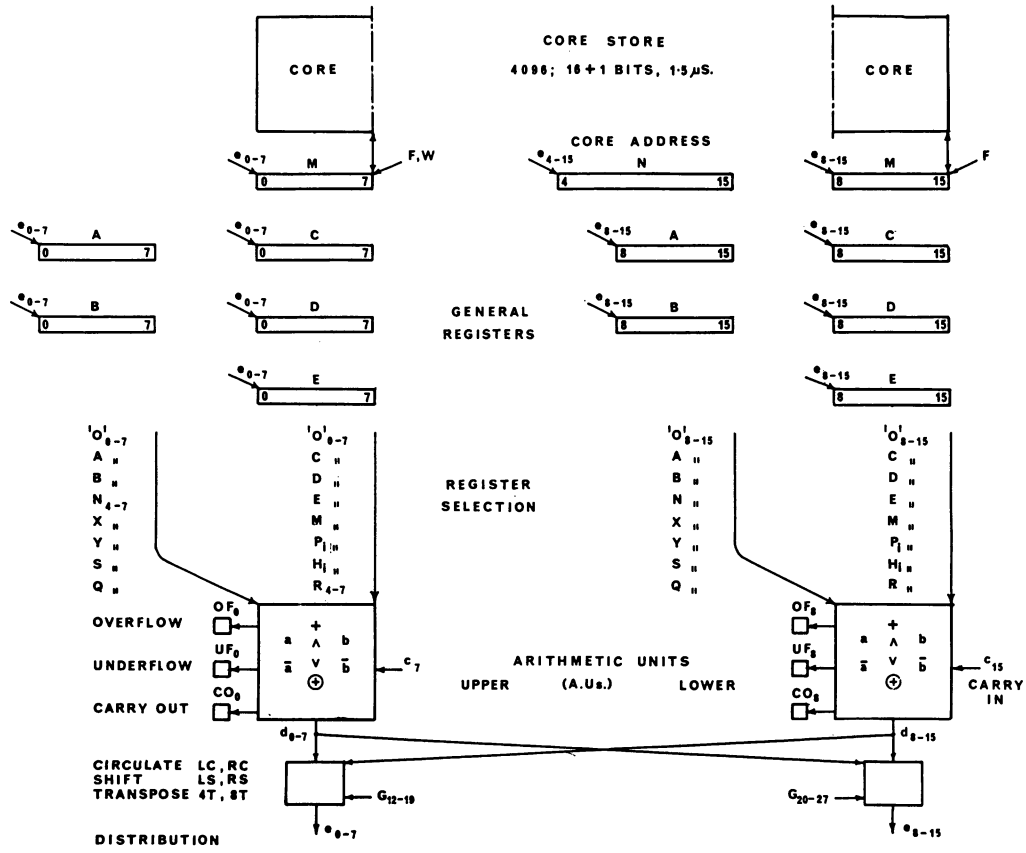


Fig. 2. General-purpose processing section.

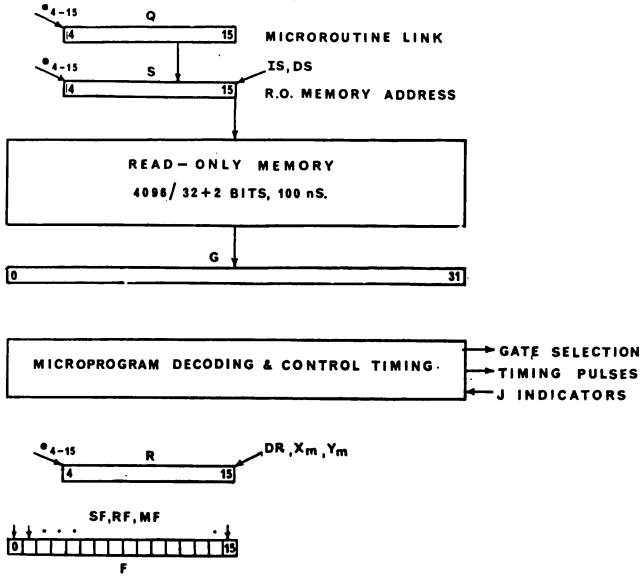


Fig. 3. Intergraphic control section.

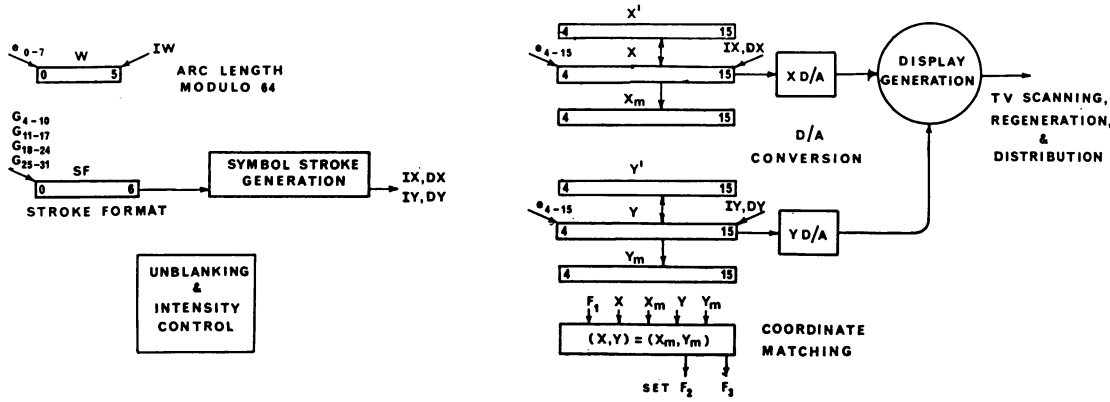
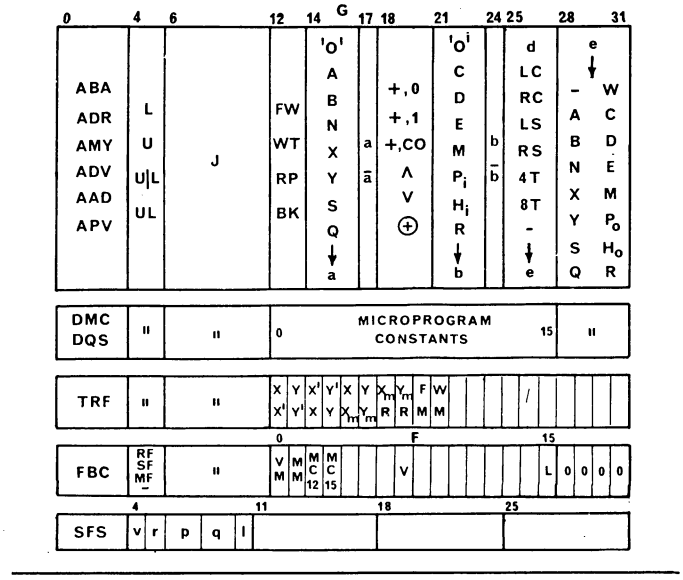


Fig. 4. Special-purpose display section.

TABLE I

MICROPROGRAM CODE



flip-flops of register F may be individually set, reset, or copied from corresponding bits of M. The last option allows nominated code bits from core words to be buffered.

5) The *core store control* order (CSC), not shown in Table I, places the READ and WRITE phases of the core store under microprogram control.

6) The *stroke formats* order (SFS). This order holds four identical code fields, each of which defines an elementary symbol stroke; a string of these strokes constitutes a symbol.

Each order takes 100 ns, except for the wait option of A-orders, and the SFS order which persists until the four strokes are generated. Micro-orders, except SFS, are executed conditionally on the state of a binary indicator nominated from a set of 64 by field J. The indicator state also conditions the sequencing of microprograms (see Appendix).

C. Display Logic

The special-purpose display hardware (Fig. 4) is also controlled by microcode as follows: the display registers X, Y may be nominated operands or destinations. As will be detailed later, X, Y may be incremented/decremented either from an arithmetic overflow/underflow during the plot vector micro-order APV, or from pulse streams generated by the stroke generation logic.

Origin registers X', Y' may be copied from X, Y and later returned to X, Y by TRF micro-orders. Coordinate-matching registers X_m, Y_m may be copied from X, Y and, if necessary, placed in core via R during a program interrupt.

The display registers each contain 12 bits; this capacity allows images to be accumulated beyond the boundaries of a 1024×1024 display, or, alternatively, allows a high resolution (4096×4096) display to be driven.

Coordinate-matching hardware assists display item identification. Pen coordinates are stored in X_m, Y_m and the display reconstructed once until coincidence within a nominated tolerance. For exact coincidence, all X, Y bits must agree with X_m, Y_m before an indicator is set (F_3 is reserved for this purpose). For relaxed coincidence, F_2 is set when the match extends to all but the least significant three bits. Display reconstruction during matching need not be visible. The referenced item may be a point, vector, or set of vectors, depending on the particular micro-order(s) which is match conditioned.

The generation of vector and symbol stroke increments is discussed in Section V.

D. Peripheral Connection and Interrupts

The connection of peripheral devices to Intergraphic will be via input buses P_i and H_i, which may be nominated as operands to the arithmetic unit, and output buses P_0 and H_0 which may be nominated as destinations.

All peripheral transfers will be executed by microcode sequences within Intergraphic, the particular sequence being entered from OS/IG (see Section I). Status changes in peripheral devices set a J indicator(s) which, in turn, conditions the execution of various jumps distributed within microcode sequences. These traps return control to OS/IG which identifies the status change in detail and branches control accordingly.

The flexibility and speed of the microprogram structure will allow complex peripheral controlling tasks to be implemented efficiently.

IV. Examples of Intergraphic Orders and Data Formats

This section outlines some of the orders and data formats executed by Intergraphic. The code is executed from the Intergraphic core store interpretively by microprogram. Orders and data are mostly display file oriented, but not necessarily so; i.e., Intergraphic can be used as a specialized processor for the central computer.

A. Plotting a List of Vectors or Points

Fig. 5 shows the list header order which plots the list of vectors or points following the header, and Fig. 6 shows several vector or point formats.

The header nominates both the data format and the operation to be performed throughout the list. Linking bits in the data (L) define list length; a zero link bit indicates that the next word is a new Intergraphic order which may be an isolated order or another list header.

In the following, we refer to the header of Fig. 5, from left to right:

Line-Form (*LF*, *bits 0–2*): This specifies a periodic on–off, intensity pattern along a vector string according to the logical expression

$$\text{Unblank} = \overline{W}_0LF_0 + \overline{W}_1LF_1 + \overline{W}_2LF_2 + \overline{LF_0}\,\overline{LF_1}\,\overline{LF_2},$$

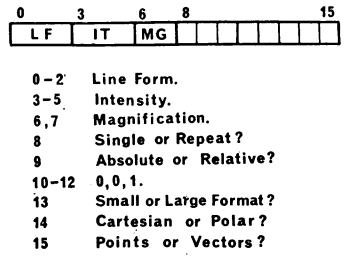


Fig. 5. List header for point or vector strings.

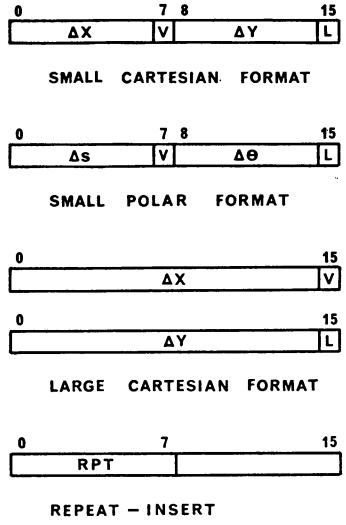


Fig. 6. Data formats for points or vectors.

where W is a 6-bit register (Fig. 4) which accumulates length, modulo 64, along a graphic from increments IW added to the least significant position W_5. Thus, for 0.01 inch display increments, bits W_0, W_1, and W_2 represent 0.32, 0.16, and 0.08 inch, respectively. LF code 001, then, gives

$$\text{Unblank} = \overline{W}_2$$

i.e., a broken line of alternate visible and blanked sections each 0.08 inch.

LF code 101 gives the periodic center-line pattern; long visible 0.40 inch, gap 0.08 inch, short visible 0.08 inch, and gap 0.08 inch.

Intensity (*IT*, *bits 3–5*): This specifies one of eight brightness levels for visible points or vectors (V="1").

Magnification (*MG*, *bits 6, 7*): This specifies that all vector or point coordinates in the list are to be magnified by ×1, ×2, ×4, or ×8. The vector-plotting microprogram maintains a constant incremental plotting rate for a given vector regardless of magnification; magnification extends the total number of increments and, therefore, the plotting time. This technique is preferred to magnification by adding a fixed number of plotting increments to more significant bit positions of the X, Y display registers. The latter technique is faster, but also magnifies the fine step structure of lines and demands a correspondingly greater bandwidth of the display device. The procedure adopted, therefore, preserves line texture within a magnified grid structure.

Single or Repeat (*bit 8*, "*1*" = *repeat*): This specifies whether points or vectors are to be plotted once or a number of times. In repeat mode, each coordinate pair is followed by a repeat-insert word which contains the number of repeats, RPT; the number of executions is RPT+1.

Absolute or Relative (*bit 9*, "*1*" = *relative*): "Plot relative" accumulates data values, i.e., vectors or point coordinates are added end to end. "Plot absolute" draws vectors, or measures point coordinates, from an origin X', Y' (point 0', Fig. 7); thus, after each point or vector is plotted, the X, Y registers are jam-set to X', Y' by the TRF micro-order $X' \rightarrow X$, $Y' \rightarrow Y$.

Small or Large Format (*bit 13*, "*1*" = *large*). The small Cartesian data format encodes a ΔX, ΔY coordinate pair plus an unblanking bit (V) and a linking bit (L) in one 16-bit core store word. For 2's complement representation, therefore, the range of coordinates is

$$-64 \leq \Delta X \quad \text{or} \quad \Delta Y < 64 \text{ increments.}$$

This small format gives compact encoding and is adequate for many line drawings, especially in conjunction with magnification. The large Cartesian format occupies two core words and can readily accommodate the full display range.

Cartesian or Polar (*bit 14*, "*1*" = *polar*): Polar coordinates are specified as a vector length Δs, and an angular difference $\Delta\theta$, from the orientation of the previous vector. This incremental polar code has three advantages: it allows graphics to be rotated by modifying the initial value of θ only, it allows circular arcs to be compactly encoded by repeating a Δs, $\Delta\theta$ pair in relative mode (Fig. 7), and it allows greater angular resolution to be encoded in a given field length because a $\Delta\theta$ specification need not cover 2π radians. The range of Δs is identical to the range of positive ΔX or ΔY, and the range of $\Delta\theta$ (also encoded in 7 bits) has been selected as

$$-\frac{\pi}{4} \leq \Delta\theta < \frac{\pi}{4} \text{ radians;}$$

this gives a resolution of $\pi/256$ radians. Two reasons for this selection are 1) that the tangential displacement of a polar vector of maximum length, $\Delta s = 64$, rotated by $\pi/256$ is approximately one display increment, and 2) that one revolution comprises an integral number (512) of least significant increments. Initially, only small polar coordinates will be implemented.

Points or Vectors (*bit 15*, "*1*" = *vectors*): A point is regarded as the visible terminus of an otherwise blanked vector. In execution, however, a point is plotted by jam-setting the X, Y display registers, whereas a vector is generated by a sequence of increment pulses to X, Y.

B. Plotting a List of Symbols

A list of symbols is preceded by a list header order with an intensity and magnification field. A symbol occupies one 8-bit byte, i.e., two symbols per core store word (Fig. 8). Through byte codes for "begin new line" etc., typewriter formats can be readily defined.

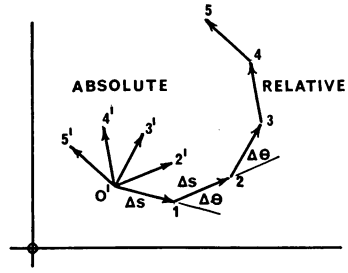


Fig. 7. Repeated polar vectors—absolute and relative plots.

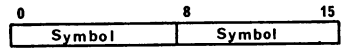


Fig. 8. Symbol format.

C. Isolated Orders

Isolated orders do not announce a following data list; they are used for setting parameter values, nesting display subpictures, etc. Examples are

Set $\theta = n\frac{\pi}{4}$

Increment θ by $n\frac{\pi}{4}$

Clear X, Y

Define Origin $0' \leftarrow X$, Y.

Since orders are interpreted by fast microprogram sequences, there is considerable scope for development of either list-oriented or isolated graphical orders.

V. Microprogram Interpretation of Intergraphic Orders

This section outlines the principal microprogram plotting sequences within Intergraphic; it gives examples of operating times, plotting rates, and microcode lengths.

A. Small Cartesian Vectors

The micro-order APV (A-order, plot vector) has been designed for vector plotting. Small format Cartesian vector plotting uses the U/L option of APV. This micro-order augments the basic arithmetic operation ABA by

1) incrementing $X(IX)$ if the upper AU overflows (upper overflow of_0 is the logical function $\bar{a}_0\bar{b}_0c_0$, where a_0, b_0 are the sign digits ("0" = positive) of the operands, and c_0 is the carry into the most significant stage of the adder; OF_0 is the buffered version of_0);

2) decrementing $X(DX)$ if the upper AU underflows (underflow uf_0 is $a_0b_0\bar{c}_0$);

3) incrementing $Y(IY)$ if of_8;
4) decrementing $Y(DY)$ if uf_8; and
5) decrementing R on every execution.

Also, APV preserves the sign of an addition on overflow or underflow, i.e., d_0 remains "0" after two positive numbers overflow the adder—normally, overflow results in a negative representation for the "sum" ($d_0 = 1$).

Thus, the upper and lower AU's add positive operands modulo 128 for micro-order APV, U/L, and the number of whole multiples of 128 are accumulated in the display registers.

In execution, C_{0-6} contains ΔX, and C_{8-14} contains ΔY (bits C_7, C_{15} are cleared after unblanking and link bits (V, L) have been removed and stored in F_7, F_{15}). Let register A be zero initially, and consider the repeated accumulation of C into A

$$\text{APV}, \quad U/L, \quad RP, \quad A + C \rightarrow A.$$

For positive ΔX and ΔY, overflows of_0 and of_8 will be generated at rates proportional to ΔX and ΔY, respectively; i.e., X and Y will be incremented proportionally to ΔX and ΔY. After 64 cycles there would be exactly ΔX overflows from the upper AU and exactly ΔY overflows from the lower AU. Thus, X would be incremented to $X+\Delta X$ and Y to $Y+\Delta Y$, as required. Setting R initially to 64, and nominating J as nonzero R (NZR) automatically controls the number of cycles. The micro-order would not be executed in cycle 65 (NZR = "0"), and the microprogram would advance to the following order. For magnified vectors, R is set initially to 64 multiplied by the magnification.

The plotting time for vectors having ΔX and ΔY less than 32 is reduced by prenormalization. If ΔX and ΔY can be simultaneously normalized, then C is left-shifted and R correspondingly right-shifted. This tends to maintain a constant incremental plotting rate regardless of the size of vectors.

The greater normalized vector component will always generate an increment in the second addition cycle, provided the vector is nonzero. Both components generate increments in the last addition cycle (provided the lesser component is also nonzero). This bias, which aligns the increment streams to coincide on the last cycle, appears as a small differential delay between the two streams. It is eliminated by setting the initial values in the upper and lower halves of register A to 32 increments for positive ΔX, ΔY. (The actual values would be +64, since one display increment aligns with bit positions 6 and 14, not 7 and 15.) A component of one unit will then generate its increment in cycle 32, rather than in the last cycle. After 64 cycles, the values in register A are identical to their initial values. Thus, repeating a vector does not require resetting of A; likewise, magnification and normalization sequences need be executed only once.

Actual Cartesian plotting rates vary between 5×10^4 and 14×10^4 in/s for 0.01 inch X and Y increments. An example of the slowest rate is $(\Delta X, \Delta Y) = (32, 0)$, which generates an X increment every second cycle, i.e., 5 MHz, and an example of the fastest rate is $(\Delta X, \Delta Y) = (63, 63)$, which generates both X and Y increments practically every cycle, i.e., X and Y rates both approach 10 MHz.

The overhead (including magnification and repeat facilities) for plotting a Cartesian vector in small format is approximately 2.0 μs. To this must be added the actual plotting time which is

$$\frac{6.4\,MG}{2^N}\ \mu\text{s}$$

where MG is the magnification, and N the number of shifts during normalization. The overhead for repeating a small Cartesian vector is 0.6 μs. The initial interpretation of the block header is executed within a core cycle time; thus, there is negligible block header overhead to be added to vector plot times, even for blocks containing only a few vectors. The small Cartesian vector microprogram (including absolute/relative option) has a total of 28 micro-orders.

B. Small Polar Vectors

Polar vector plotting requires the preliminary updating of θ by $\Delta\theta$, and the evaluation of cos θ, sin θ. Rapid table lookup for cos θ, sin θ in the ROM has reduced evaluation time to a minimum. A polar vector is plotted as a Cartesian vector with $\Delta X = \cos\theta$, $\Delta Y = \sin\theta$, except that the number of APV cycles is Δs, not 64. Normalization is not required for cos θ, sin θ, and the plotting rate is exactly 10^5 in/s for 0.01-inch increments. Repeated polar vectors, including magnification, have an overhead of 2.6 μs. This is necessarily greater than the Cartesian case because each repeat must update θ. The small polar vector microprogram has a total of 50 micro-orders, and the cos θ, sin θ table contains 65 micro-orders.

As a summary example of small polar vector performance, a circle comprising 16 vectors, $\Delta s = 40$ increments, $\Delta\theta = 32/64$ of $\pi/4$ (approximately 2 inches in diameter for display increments of 0.01 inch) would require three core words—block header, $(\Delta s, \Delta\theta)$ vector, and repeat insert. It would be executed in approximately 100 μs, i.e., an average plotting rate, allowing for all overheads, of 6×10^4 in/s.

C. Symbol Plotting

Symbols, each nominated by a one-byte core code, are plotted in detail by short microprogram sequences, mostly SFS micro-orders. Each SFS order specifies a sequence of four, or fewer, standard strokes. Six standard strokes, on the average, constitute an uppercase symbol. Thus, two SFS orders, i.e., 64 ROM bits including

the SFS labels, are sufficient for most uppercase symbols; completed uppercase symbols appear as elemental vectors linking adjacent nodes of a 17 by 9 rectangular grid. The speed of symbol plotting is limited by the adopted maximum incremental X or Y plotting rates of 10 MHz, not by microprogram rates. Half-scale plotting of uppercase symbols reduces the node matrix to 9×5, and almost halves the average plotting time (3 μs per symbol).

Symbols, however, may be arbitrary and extend over any number of grid increments by linking sufficient strokes. Moreover, symbols need not be composed entirely of standard strokes; micro-orders which plot vectors or points may be mixed with SFS orders.

Each standard stroke field within SFS comprises an unblanking bit v; a bit r which distinguishes between curved and straight stroke elements; a two-bit field p which specifies one of four standard curved strokes or one of four standard straight strokes; a two-bit field q nominating which quadrant version of standard stroke; and a link bit l which, when "0," marks the completion of a symbol.

VI. Conclusions

Detailed planning of Intergraphic has shown that a medium sized, microprogrammed structure which is largely general purpose, but with some special-purpose hardware and microcode, will be able to interface many (more than 50) graphical terminals to a central processor. The proposed structure:

1) eliminates many of the trivial, but frequent, interrupts on a central processor which occur in unbuffered systems;

2) replaces special-purpose vector and symbol generation hardware, often required for each terminal, by fast microprogram sequences which generate vectors and symbols for all terminals;

3) removes many tasks specific to graphical communication from the central processor by allowing postprocessing of compact central processor codes and preprocessing of graphical inputs;

4) provides an extremely flexible medium both for performing the variety of processing and supervisory tasks required and for experimentation through new microcode sequences, not hardware extensions;

5) has simplified and unified hardware design, as it comprises only OR–NOR gates and an ROM; analog signals first appear at the D/A converters driving the central display(s). This fully digital approach to the generation of graphics is in antithesis to the analog, or part analog, generation of specific classes of display curves.

The projected number of terminals is large because the "new-image" generation rate is rapid and will not be dissipated by regenerating short persistence displays merely for persistence of vision. Moreover, the cost per terminal will be low, because the shared scan converter(s) and multitrack video disk(s) allow low cost, standard TV terminals to be used.

A limitation of the system will be an average "new-image" rate per terminal of about one per second; this is believed adequate because many terminals will be static for seconds. Highly dynamic or synchronous displays will be limited to one or two in number; they will be large, electrostatically deflected CRT's, not TV receivers, connected directly to Intergraphic.

It is expected that the system will become central processor limited, not peripheral limited, as graphical terminals greatly increase the access to, and therefore the demand on, the central processor.

The significance of a system supporting many general-purpose, graphical terminals at low cost is that many display-dependent applications, e.g., computer-aided teaching or design, will become more widely used.

Appendix
Microcode Details

Reference is made to Table I.

Order Type

The first field, G_{0-3}, is common to all micro-orders; it nominates the order type.

Upper and Lower Bytes

Field $G_{4,5}$ is common to A-orders, DMC, DQS, and TRF. Option L indicates lower significant bytes (bits 8–15) of registers, i.e., L indicates that only the lower arithmetic unit (LAU) is to be used for A-orders, and that microprogram constants or register-to-register transfers apply to bits 8–15 only. Option U nominates upper significant bytes only (bits 0–7 for 16-bit registers, or bits 4–7 for 12-bit registers). U/L specifies simultaneous execution of the L and U options, i.e., UAU (the upper arithmetic unit) and LAU operate independently without carry, shift, or circulate linkages between bit positions 7 and 8. UL, however, specifies operations on full registers. Options U/L and UL are identical for logical operations, microprogram constants, and register transfers.

J Indicators and Sequencing

Field G_{6-11} (J) applies to all orders except SFS; it nominates one of 64 binary indicators to condition the execution of the micro-order. Two examples of J indicators are A_0 (the sign digit of register A), and NZR (nonzero R). If the nominated J indicator is binary "1", the micro-order is executed; if not, the order is inhibited. J_0 is the constant "1"; thus, J_0 specifies unconditional execution. For the forward (FW, field $G_{12,13}$) option of A-orders and for all other orders, regardless of the state of the J indicator, the next micro-order is read from address $S+1$. However, for A-order options wait

(*WT*), repeat (*RP*), and back (*BK*), sequencing depends on the state of the *J* indicator.

The execution of the wait option is delayed until the nominated indicator becomes "1"; an example being an addition micro-order waiting for the arrival of an operand on completion of the READ phase of the core store (indicator "core-READ complete").

For the repeat option, the order is repeatedly executed until the nominated indicator clears. When cleared, execution is inhibited and the next micro-order is taken from $S+1$. Two examples of repeated arithmetic/logical operations are vector plotting (Section V) and multiplication. For the back sequence option, if the *J* indicator is "1", the order is executed and *S* decremented to $S-1$; otherwise, the order is inhibited and *S* advanced to $S+1$. An *FW–BK* order pair enables two micro-orders to be alternated repeatedly until the *J* indicator clears (both orders would nominate the same *J*). An example of a two-order repeat is vector normalization; the components of a vector are doubled (one order) and the number of plotting cycles halved (the other order) until the vector is normalized.

Sequencing may also be controlled by nominating *S* as the destination for either the result of an A-order (computed jump) or a microprogram constant (absolute jump). These jumps override the normal sequencing.

The DQS order augments DMC (DMC sends a microprogram constant to a register) by transferring *Q* to *S*. This variant is useful for the construction of tables in ROM; e.g., the table of $\cos\theta$,$\sin\theta$, used for polar vector plotting, consists entirely of DQS orders which send $\cos\theta$, $\sin\theta$ to register *C* and return control to address *Q*. Without the *Q* to *S* variant, each table entry would need to be followed by a jump order.

A-Orders

ABA, the basic arithmetic/logical order, combines operands *a* and *b* (nominated by fields G_{14-16} and G_{21-23}) by addition, or by the digitwise logical operations of AND, OR, or EXCLUSIVE OR. Operand complements $\bar{a}$ and $\bar{b}$, may also be used. Addition may include an input carry *c* of 0, 1, or the carry out (*CO*) of the previous operation (shown as "+, 0," "+, 1," and "+, *CO*" in G_{18-20}). Field G_{25-27} allows the AU output *d* to be directly connected, circulated, or shifted one position, or transposed in 4- or 8-bit bytes before being distributed as *e*.

Right shift *RS* preserves sign, and left shift *LS* clears the least significant digit. Circulate and shift are interpreted in conjunction with *U*, *L*, etc. For example,

$$U/L,\ RC\colon e_{0-15} \leftarrow d_7, d_{0-6}, d_{15}, d_{8-14}$$
$$UL,\ RC\colon e_{0-15} \leftarrow d_{15}, d_{0-14}.$$

Transpose mappings are

$$4T\colon e_{0-15} \leftarrow d_{4-7}, d_{0-3}, d_{12-15}, d_{8-11}$$
$$8T\colon e_{0-15} \leftarrow d_{8-15}, d_{0-7}.$$

For the *L*, $8T$ combination, *L* applies to the operands and *d* (bit positions 8–15). After the $8T$ transpose, these relevant bits become e_{0-7}, i.e., the *U* byte of the destination register is clocked for the *L*, $8T$ combination, and vice versa. The destination for *e* is specified by field G_{28-31}.

ADR (A-order, decrement *R*) augments ABA by also decrementing *R* on each execution. Inclusion of "repeat" and "decrement *R*" in one micro-order gives a threefold speed advantage over the 3-loop consisting of an arithmetic order, a decrement counter order, and a test jump.

AMY (A-order, multiply) is a variant of ADR which assists multiplication; the nominated *a* operand or "0" is selected depending upon whether D_{15} is "1" or "0." Consider an 8-bit multiplicand in A_{0-7}, 0's in A_{8-15}, and initially 0's in D_{0-7}, an 8-bit multiplier in D_{8-15}, and 10_8 in *R*. Then the 16-bit product appears in D_{0-15} in 0.8 μs by executing

$$\text{AMY, } UL, \text{ NZR, } RP, A + D \xrightarrow{RS} D \quad (c = 0).$$

ADV (A-order, divide) assists division by selecting the "*b*" operand or its negative according to B_0, the sign digit of *B*, register *B* being used as the nulling register in nonrestoring division.

AAD (A-order, analog to digital) assists analog-to-digital conversion. This variant is similar to ADV, except that the sign of the output of an analog comparator replaces B_0.

APV is detailed in Section V=A.

Distribute Microprogram Constant Orders

These orders use bits G_{4-11} and G_{28-31} identically to A-orders. The microprogram constant field G_{12-27} becomes *e* via a path in the $d \rightarrow e$ logic. The DQS variant has been previously defined under sequencing.

Transfer Register Order

This order executes register-to-register transfers according to 1's in the field G_{12-31}. A number of simultaneous transfers are possible, provided they are independent. Options *L*, *U*, or *UL* (or *U/L*), and conditional indicator *J* apply.

F Register Bitwise Control Order

Sixteen flip-flops may be reset individually (option *RF*, $G_{4,5}=0$, 0) by placing 1's in the *F* field (G_{12-27}). Option *SF* sets nominated flip-flops; option *MF* copies register *M* onto *F* through the mask pattern nominated in G_{12-27}. The state of *F* may be stored in core via an $F \rightarrow M$ transfer order. Some *F* bits have a special purpose; e.g., F_1 indicates matching mode. *F* bits are also available as *J* indicators.

Stroke Formats Order

The formats are covered in Section V-C.

Acknowledgments

The author is indebted to Prof. M. W. Allen, Prof. P. D. Jones, M. Macaulay, G. P. Bowen, and R. B. Stanton for their many discussions, and to R. B. Chorley and K. W. Titmuss for laboratory assistance.

References

[1] I. E. Sutherland, "SKETCHPAD, a man-machine graphical communication system," *1963 Spring Joint Computer Conf., AFIPS Proc.*, vol. 23. Baltimore, Md.: Spartan, 1963, pp. 329–346.

[2] T. Marill *et al.*, "Cyclops I, a second generation recognition scheme," *1963 Fall Joint Computer Conf., AFIPS Proc.*, vol. 24. Baltimore, Md.: Spartan, 1963, pp. 27–33.

[3] G. J. Culler and B. D. Fried, "An on-line computing centre for scientific problems," TRW Computer Division, Canonga Park, Calif., Rept. M19-3U3, June 1963.

[4] R. M. Fano, "The MAC system: the computer utility approach," *IEEE Spectrum*, vol. 2, pp. 56–64, January 1965.

[5] F. J. Corbató and V. A. Vyssotsky, "Introduction and overview of the Multics system," *1965 Fall Joint Computer Conf., AFIPS Proc.*, vol. 27, pt. I. Washington, D.C.: Spartan, 1965, pp. 185–196.

[6] D. E. Rippy *et al.*, "MAGIC, a machine for automatic graphics input to a computer," *ibid.*, pp. 819–830.

[7] W. H. Ninke, "Graphic I—a remote graphical display console system," *ibid.*, pp. 839–846.

[8] R. W. Lichtenberger and M. W. Pirtle, "A facility for experimentation in man-machine interaction," *ibid.*, pp. 589–598.

[9] N. A. Ball, H. Q. Foster, W. H. Long, I. E. Sutherland, and R. L. Wigington, "A shared memory computer display system," *IEEE Trans. Electronic Computers*, vol. EC-15, pp. 750–755, October 1966.

[10] J. R. Kennedy, "A system for time-sharing graphic consoles," *1966 Fall Joint Computer Conf., AFIPS Proc.* vol. 29. Washington, D. C.: Spartan, 1966, pp. 211–222.

[11] G. A. Rose, "Economical, graphical-communication techniques for multiple console operation," *Third Australian Computer Conf. Proc.*, pp. 399–402, May 1966.

[12] M. Macaulay, "Low-cost terminals using television techniques," *Nat'l Radio & Electronics Engrg. Conv. Abstracts*, IREE (Australia), p. 222, May 1967.

[13] T. E. Johnson, "Analog generator for real-time display of curves," M.I.T. Lincoln Laboratory, Lexington, Mass., Tech. Rept. 398, July 1965.

[14] M. L. Dertouzos and H. L. Graham, "A parametric graphical display technique for on-line use," *1966 Fall Joint Computer Conf., AFIPS Proc.*, vol. 29. Washington, D. C.: Spartan, 1966, pp. 201–209.

[15] N. H. Lehrer and R. D. Ketchpel, "Recent progress on a high resolution, meshless, direct-view storage tube," *ibid.*, pp. 531–540.

[16] D. L. Bitzer and H. G. Slottow, "The plasma display panel—a digitally addressable display with inherent memory," *ibid.*, pp. 541–547.

[17] M. W. Allen, T. Pearcey, J. P. Penny, G. A. Rose, and J. G. Sanderson, "CIRRUS, an economical multiprogram computer with microprogram control," *IEEE Trans. Electronic Computers*, vol. EC-12, pp. 663–671, December 1963.

[18] P. Fagg *et al.*, "IBM System/360 engineering," *1964 Fall Joint Computer Conf., AFIPS Proc.*, vol. 26, pt. I, pp. 205–231.

"Light-Pen" Facilities for Direct View Storage Tubes—An Economical Solution for Multiple Man-Machine Communication

GORDON A. ROSE

Abstract

Techniques are presented which enable conventional "light-pen" tracking and pointing functions to be extended to the direct view storage tube (DVST). The schemes eliminate the high data transfer rates or considerable buffer storage required even for static displays when short persistence cathode ray tubes (CRTs) are used.

Two solutions, a "quadruple photo-sensor pen" and the "potentiometer pen" (a transparent resistive sheet and stylus combination which covers the display), are described and compared.

Introduction

A versatile CRT display with "light-pen" tracking and pointing facilities is well recognized as a creative tool for on-line machine-aided design [1], [2], problem solving [3], text and program editing, etc. In many instances, the extension of pen displays to a number of independent stations is desirable (e.g., a teaching machine laboratory).

Continual regeneration of a short persistence CRT for flicker-free viewing demands high data transfer rates, or considerable buffer storage; but in return for this outlay comes ready communication by single photo-sensor pens and a medium for highly dynamic displays. However, the attendant costs are related to regeneration rates established by persistence of vision criteria and are disproportionate to the dynamic content of the majority of displays where static text and diagrams prevail for seconds.

The DVST eliminates the need for continual regeneration and can provide excellent displays. Often, the retention of past events enhances a display by portraying trends or envelopes. Flood erasure followed by display regeneration at infrequent intervals is adequate for most presentations. Developments in short erase times, fast writing speeds and selective erasure techniques have produced an attractive display medium. In a large time-shared installation, a small number of highly dynamic problems could be accommodated with short persistence displays.

Unfortunately, conventional "light-pen" techniques are inapplicable to DVSTs, both in the tracking mode (which senses and follows pen movement by probing the visual field of the pen), and in the pointing mode (which associates the instant of pen response with an item in the display file), for the following reasons:

1) Tracking pen movement by beam perturbations within the pen field would result in a thick stored trail—the complete history of local probing.
2) Any exploratory scan designed to locate the pen and initiate the trail would overwrite the stored image.
3) Pointing to a stored image on the DVST, subsequent to writing, cannot generate any dynamic signals synchronized with the display file.

Solution I. The Quadruple Photo-Sensor Pen

Objection 1) has been overcome by using a set of four photo-sensors arranged in horizontal and vertical pairs which sense the local (X, Y) displacement of the display spot relative to the pen axis (Fig. 1). The pen has been assembled from miniature silicon photo transistor sensors and silicon chip difference amplifiers housed in the pen. The sign of a difference signal will gate either incrementing or decrementing pulses to the corresponding display register. An alternative and elegant solution would be to employ a partitioned set of fiber optics.

Objection 2) is avoided by starting a track initially from an "inkwell" and subsequently from the current beam position, which appears brighter than the stored image.

It is necessary to distinguish between hand drawn track segments to be filed by the computer, and incidental tracking trails linking such segments. The distinction is made by a pressure operated pen tip switch. Wanted track coordinates are filed by interrupting the computer *either* at regular intervals (20 ms), *or*, when the track is started or terminated, or crosses grid lines of preselected spacing.

The need to transport the beam to every track start and the attendant unwanted image trails are disadvantages of the photo-sensor method. The clutter of incidental trails (particularly noticeable when building up a complex freehand sketch of symbols or diagrams involving frequent "pen lifts" and return referencing), may be removed by flood erasure and redisplay; but this can be demanding on computer time if repeated each time a wanted track is commenced, at any of a number of stations.

The quadruple photo-sensor pen loses control if rotated around its longitudinal axis in excess of $\pm 30°$. Shaping the pen grip is sufficient to constrain this rotation.

Objection 3) has been overcome in several ways. A single flood-erase and redisplay cycle initiated each time a feature is pointed to enables the conventional photo-sensor pointing mode to be used. Alternatively, the tracking mode may be used to transport the display spot to a feature (cursor mode), followed by an interrupt. Feature identification by coordinates is discussed under Solution II.

Solution II. The Transparent Two-Dimensional Potentiometer

A. Outline of the System

This technique does not employ photo-sensors, and thereby avoids unwanted transport trail generation on the DVST screen.

The (X, Y) position of a hand-held stylus is detected from uniform potential gradients, switched alternately between the X and Y directions on a transparent conductive sheet placed over the display.

Digital versions of the analog stylus coordinates are generated using the existing digital registers, D/A converters, and deflection amplifiers of the display console in a feedback loop, which equates the deflection signals to the analog stylus coordinates (Fig. 2). A suitable choice of scale factors brings the display spot almost directly beneath the stylus tip. Thus, the "true" position of the stylus (i.e., the position corresponding to the digital coordinates accepted by the computer on interrupt), is seen by the operator and becomes his focal point for tracking. Misalignment of the controlled spot and the stylus does not appear as an error in the system, but rather as an annoyance to the operator. There is some tolerance here, as in conventional light-pen operation, because manual inputs cannot be controlled with the precision implied by the display resolution (typically 10^4 points per inch^2).

"On-off" intensity control of the beam has been readily achieved by detecting stylus contact electrically, and gives the impression of the stylus drawing the track. In the tracking mode the computer is

Manuscript received January 4, 1965; revised April 27, 1965.
The author is with the University of Adelaide, Adelaide, South Australia.

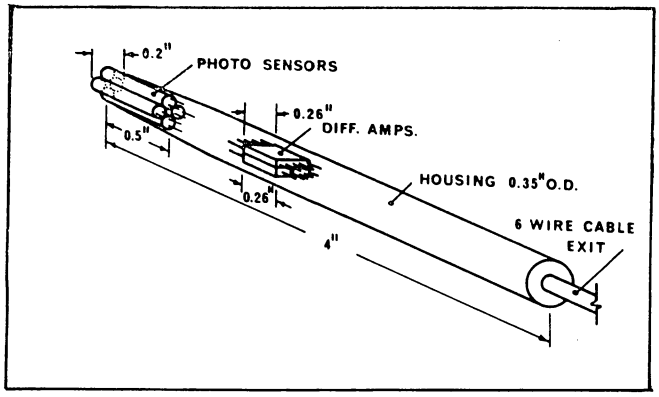


Fig. 1. The quadruple photo-sensor pen.

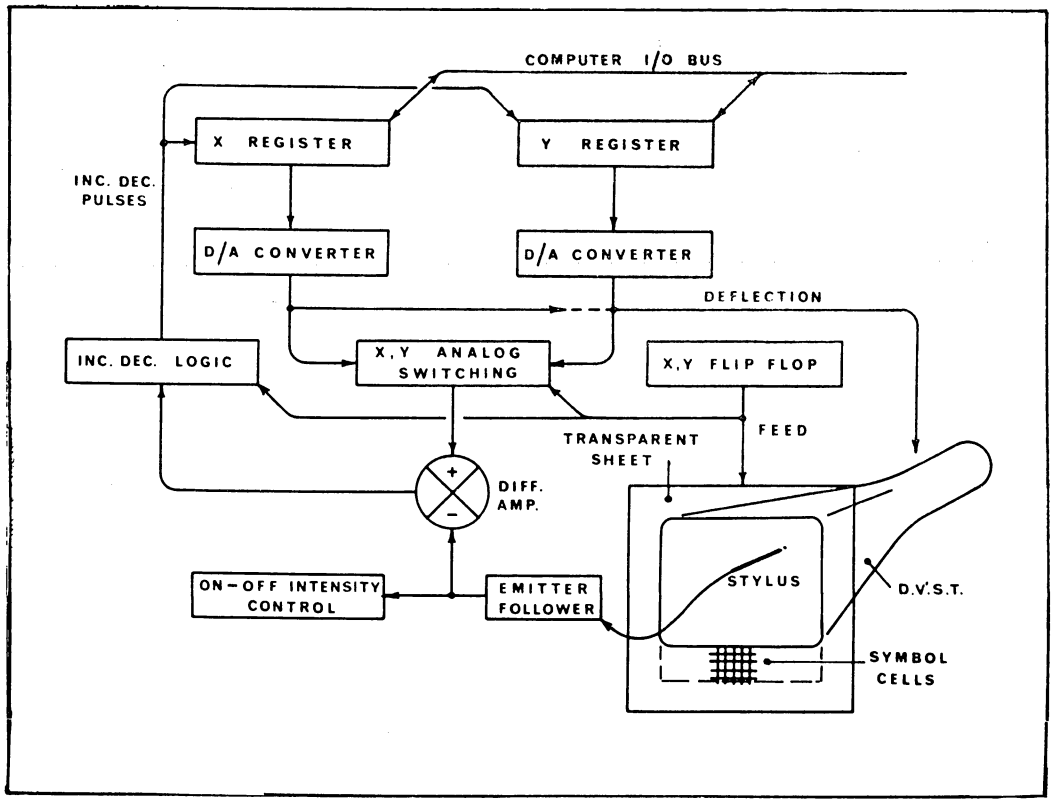


Fig. 2. The "potentiometer-pen" system.

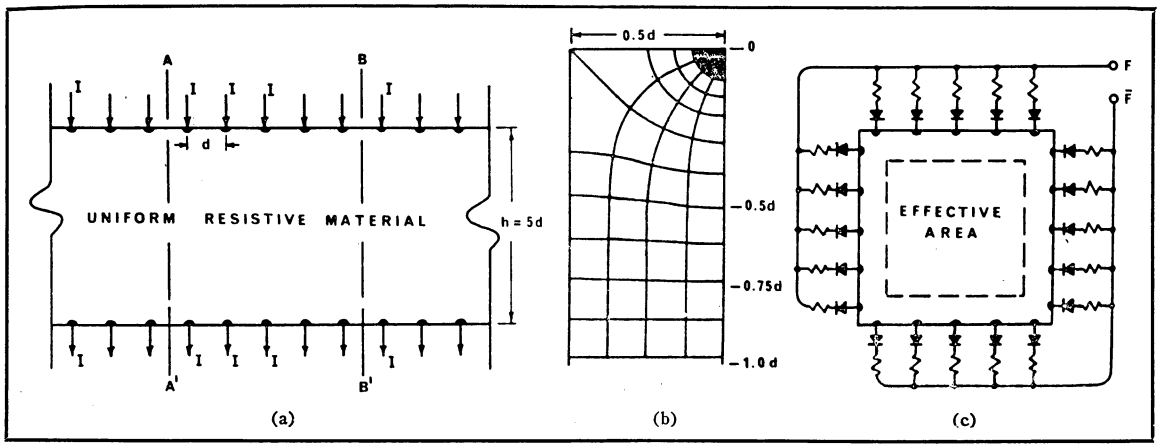


Fig. 3. The transparent two-dimensional potentiometer.

interrupted during stylus contact with the same options as in Solution I.

In the pointing mode, a feature is indexed by pointing directly to it. The corresponding digital coordinates are sent to the computer when the stylus is lifted from the sheet. (Thus, before lifting the stylus, the display spot may be placed precisely on the feature.)

Identification of a feature from its digital coordinates may not be possible in some cases (e.g., if a line generator is used, the intermediate line points will not be directly recognized by the computer). In this case, the feature coordinates are stored locally in the display console and the display is regenerated once (blanked), until coincidence of the display and stored coordinates interrupts the computer.

B. Two-Dimensional Potentiometer Details

Consider an infinite strip of uniform resistive material fed with equal direct current magnitudes [Fig. 3(a)]. The contacts are identically shaped and equally spaced along the top and bottom edges. The top edge feed contacts will be equal in potential (V_T), and the bottom edge contacts will be equal in potential (V_B). For contact separations (d) that are small compared with the strip height (h) there will be a uniform potential gradient between the top and bottom edges, except for local deformations in the vicinity of these edges.

By symmetry, there will be zero transverse component of current across vertical lines midway between the feed points. Lines AA' and BB' are in this category and therefore cuts could be made along AA' and BB' without changing the flow within the finite section between them. Also, from symmetry, the field plot consists of a number of replications and reflections of the section shown [Fig. 3(b)]. A margin of $0.75d$ has been found adequate to avoid potential deformations.

Choosing $h=5d$, a square section has been formed with five feed points along the top and bottom edges. A similar set of contacts are now placed along AA' and BB' [Fig. 3(c)], and provided they are open circuited, will adjust in potential between V_T and V_B and not significantly effect the flow, except in their own vicinity (again a margin of $0.75d$ has been adequate).

Switching the roles of the horizontal and vertical contact sets establishes a uniform horizontal current flow. Thus, alternate analog samples of the X and Y coordinates of a high impedance stylus are generated.

Automatic feed switching has been achieved using diodes and a standard flip-flop. For the "1" state of the flip-flop, ($F=0$V, $\overline{F}=-3.0$ V), only the top and bottom sets of diodes conduct. For the "0" state only for the side diodes conduct. A durable, highly transparent stannic oxide coating (approximately 200 ohms per square), has shown no physical or chemical deterioration after several months of use. The stylus used is simply a good quality hard pencil lead which moves freely over the coating, and feeds a 5-megohm input impedance emitter follower. The slight film deposited by the pencil lead has negligible conductance compared with the oxide coating (an exaggerated film on glass measured many megohms per square), and is readily removed every few days. Contact of the coating by the operator's hand does not produce any visible misalignment of the display spot and stylus. Differences in contact shape and diode drops have been compensated for by setting the feed flip-flop to a particular state, and adjusting resistances in series with the conducting diodes until the near boundary of the effective area of the sheet is an equipotential. This adjustment is carried out at construction time only.

Misalignment errors of one per cent of the sheet side are attributed to nonuniformity of the oxide coating (produced commercially at low cost, by a hot spraying process). Controlled coatings deposited by sputtering would result in very low misalignment.

Although no sheet feed or stylus details are described, an apparently parallel development has been briefly reported by Hargreaves, et al. [4], in conjunction with nonstorage CRTs. The input device forms part of the comprehensive DAC-1 system. Tracking is effected by program.

C. Further Applications of the Potentiometer-Pen

A "keyless" typewriter has been formed as follows. The effective area of the conductive glass is extended beyond the useful area of the DVST and one of a set of replaceable alphabet cards is inserted beneath the conductive glass. Symbols are arranged in a matrix of cells (0.5 cm×0.5 cm cell size) and, when pointed to by the stylus, digital coordinates (most significant bytes) corresponding to the cell, and therefore the symbol, are sent to the computer. The method incorporates the typewriter facility with the display without sacrificing valuable display area.

Handwriting can be separated into X, Y component waveforms for dynamic analysis or transmission using the analog or digital versions of stylus position.

Conclusions

1) Man-machine communications based on short persistence CRT displays and single photo-sensors are unnecessarily demanding on a computer system.
2) Saving in average data rates associated with DVST displays make multiple pen-displays economically feasible.
3) Low-cost "light-pen" facilities for DVST displays have been simply constructed and are extremely effective.
4) The potentiometer-pen method is preferred:
   a) Tracks may be initiated or features indexed at random without storing unwanted transport trails on the DVST screen.
   b) A "keyless" typewriter is readily implemented at virtually no extra cost. The compact symbol set and pencil-like stylus has resulted in higher input symbol rates for nontypists.

Acknowledgment

The work of this paper was motivated from discussions held during the opening phase of M.I.T.'s Project MAC, August 1963. The author is grateful to Prof. R. M. Fano for enabling him to participate.

References

[1] R. Stotz, "Man machine console facilities for computer-aided design," *1963 Am. Fed. of Information Processing Societies Conf, Proc.*, vol. 23, pp. 323–328.
[2] I. E. Sutherland, "Sketchpad, a man-machine graphical communication system," *1963 Am. Fed. of Information Processing Societies Conf. Proc.*, vol. 23, pp. 329–346.
[3] G. J. Culler and B. D. Fried, "An on-line computing centre for scientific problems," Rept. M19-3U3, T.R.W. Computer Division, Thompson Ramo Wooldridge, Inc., Canoga Park, Calif., June 1963.
[4] B. Hargreaves, et al., "Image processing hardware for a man-machine graphical communication system," *1964 Am. Fed. of Information Processing Societies Conf. Proc.*, vol. 26, pt. I, pp. 363–386.

Reprinted from IEEE TRANSACTIONS
ON *ELECTRONIC COMPUTERS*
Volume EC-14, Number 4, August, 1965
Pp. 637-639

Economical, Graphical Communication Techniques For Multiple Console Operation

By GORDON A. ROSE
Department of Electronic Computation
University of New South Wales
P.O. Box 1, Kensington, N.S.W.

SUMMARY

The paper outlines the output and input sequences of an on-line, graphical communication system which links many users to a central processor via an interface computer. Output includes arbitrary graphics and symbols; input includes real-time freehand sketches and symbols, referenced symbol cells and image pointing.

The paper cites a number of inefficiencies in the encoding, plotting and input tracking of graphics. It presents a compact incremental polar encoding which specifies a graphic either by a piecewise linear approximation or by linked sections of constant curvature.

The paper specifies the logical structure of a graphical interface computer, INTERGRAPHIC, which is responsible for detailed image generation, input graphic encoding and vector string manipulation. The interface operates mainly in two high speed incremental modes — rectangular and circular, and is capable of incremental plotting rates of several Mc/s. The symbol generator shares the interface read-only micro-program control store and generates symbols from standard straight lines, quarter circles and quarter ellipses at a point plotting rate of 10 Mc/s.

The paper emphasises the need for simple, unifying concepts for graphical communication, and proposes that the flow of direction and curvature of a graphic with arc length is a useful basis for both recognition and display.

SECTION I. INTRODUCTION

GRAPHICAL COMMUNICATION is, for the purposes of this paper, on-line, man-machine communication through line drawings and symbols. Such communication exploits the user's visual perception and his ability to point, sketch, print and write. It requires the machine display and recognition of arbitrary shapes at rates matched to the user.

A number of notable systems[1 2 3] have proved the utility of graphical communication, and time-sharing can extend it to many users. However, such networks are rare, mainly because many of the techniques require excessive computer time or storage. The work outlined in this paper attempts to eliminate or reduce these inefficiencies.

The paper regards graphical communication as consisting of an output path, from machine to user, and an input path from user to machine. The paths convey graphical forms, or simply graphics.

The output path converts the highest machine language description of a graphic, e.g., its name and parameters, into its visual form. The sequence is:—

(i) Highest machine description, e.g., field plot TM_{01}.
(ii) Expansion of (i) into its elements, e.g., every field line and every boundary. Each element being defined by; a set of incremental relations, or the flow of direction or curvature with arc length, or a set of sample points.
(iii) Efficient encoding from (ii) into forms accepted by display generation hardware, e.g., strings of line segments, expressed in Cartesian or polar form.
(iv) Generation of individual display points by vector and symbol generators.

Graphics may be modified at the highest machine level, e.g., change parameter, or at the display generation level—the interface level, e.g., rotate graphic. For simple graphics, (ii) may be trivial. Commonly used graphics, once encoded at (iii), may be stored in a dictionary under a name in (i).

The input path accepts a freehand graphic and classifies it as an object or operation in the highest machine language. The sequence is:—

1. Detection of the flow of input co-ordinates (tracking).
2. Encoding the flow in the interface language.
3. Determination of the geometrical attributes of the graphic, e.g., its curvature flow, orientation, or total arc length.
4. Classification or recognition of the graphic.

This sequence is bypassed when the user touches a symbol key or points to a displayed item.

The paper centres mainly on the interface problem [items (iii), (iv), (1), (2) and (3)] which requires high-performance hardware of low cost per user for its solution.

SECTION II of the paper presents a number of observations on graphical communication. SECTION III is more specific; it describes the system structure of the graphical interface computer, INTERGRAPHIC.

SECTION II. BACKGROUND OBSERVATIONS

The observations of this section are general; often, comparisons are made between techniques or devices, but the argument is not continued to a statement of preference.

1. Inefficiency of Independent Point Plotting

A typical display system presents an image as a set of intensified points on a 1024 x 1024 square grid. Thus, for a 10" x 10" screen, the grid line separation is approximately 0.010", and good quality line drawings are possible.

To specify a display by its constituent points, i.e., to consider each point independently of the others, requires 20 bits per point. Such encoding is inefficient; it does not exploit the continuity of lines, or the often simple incremental relations which define them.

2. Incremental Generation of Straight Lines and Circles

For a straight line from point A, (x_0, y_0), to point B, $(x_0 + X, y_0 + Y)$, the incremental relations are;

$$\frac{dy}{dx} = \frac{Y}{X},$$

or, in terms of the parameter t,

$$\frac{dx}{dt} = kX, \quad \frac{dy}{dt} = kY \text{ (k constant)}$$

Thus, two streams of unit grid increments, having time rates proportional to X and Y respectively, are required to generate the line. A pair of 10-bit binary counters, set initially to (x_0, y_0), may integrate the increment streams (physically, pulse trains) to form the display co-ordinates (x, y). At most, 40 bits are required to define AB. If lines are plotted end to end, from a specified origin, then the co-ordinates of A exist. If, also, the ranges of X and Y are less than the total display size, then an even shorter code is required, e.g., 12 bits.

For a circle, centre (x_0, y_0), the incremental relations are;

$$\frac{dx}{dt} = -\omega (y-y_0), \quad \frac{dy}{dt} = \omega (x-x_0) \qquad (\omega \text{ constant})$$

Two increment streams are again required; here the stream rates are not constant, because the rate controlling registers containing $x-x_0$ and $y-y_0$ must also be updated by the increment streams.

Two increment streams are also sufficient to generate right parabolas, hyperbolas and exponentials.

3. Increment Rate Generators

Several techniques[4] exist for generating pulse rates proportional to the contents, X, of a register.

Repeated addition of X into an accumulator generates overflows at the required rate—a digital differential analyser (DDA) technique (bit parallel operation implied).

Alternatively, a binary rate multiplier (BRM), which gates and merges binary weighted pulse streams according to X, may be used.

4. Specific versus Universal Curve Generation

Both algorithms and special purpose hardware are common for straight line generation (vector generators). Sutherland[1] has proposed DDA techniques for the general conic. Obviously, a compromise arises between the extent and utilisation of special hardware for the generation of specific curve families.

It is unlikely that there is a universal procedure, or a single set of hardware, for the efficient incremental generation of all curves—rather, the flexibility of the central processor should be available for this purpose. Regardless of the particular generation scheme, it is feasible that there are universal curve encoding schemes which are particularly efficient for storage and interface operations.

5. Efficient Encoding of Arbitrary Curves

Freeman[5,6] has encoded arbitrary continuous curves using strings of three bit groups, each group specifying one of the eight incremental vectors, (1, 0), (1, 1), (0, 1), (—1, 1), (—1, 0), (—1, —1), (0, —1) and (1, —1) in units of one grid division. Similar coding strings have been used to control mechanical plotting devices. Although general, this coding is inefficient for curve storage and unsuitable for curve processing: inefficient, because it encodes unnecessary detail for sections of low curvature; unsuitable for processing, because it cannot be easily changed to accommodate the simple operations of rotation and scaling.

A string of piecewise linear sections (vectors) may represent an arbitrary curve. Curve rotation and scaling are simply performed by applying these operations to the constituent vectors. The method automatically provides greater detail with increasing curvature, and higher directional resolution over the relatively long straight portions of a curve. This adaption is not inherent in the Freeman coding, and the determination of direction requires averaging over a variable number of increments.

In summary, the encoding of graphics must be universal, efficient, and operable; so that arbitrary curves may be compactly encoded and stored in a form suitable for, at least, translation, rotation and scaling.

6. Symbol Generation

Displays of printed text or programs form a large part of graphical communication; thus, comprehensive symbol sets which may be extended are desirable. Small dot matrices, say 7 x 5, although modest in storage requirements, restrict the character set. Large dot matrices rapidly become inefficient in storage, since the method does not exploit the continuity of symbol strokes. Special-purpose symbol generation devices incorporated in each display unit, e.g., beam shaping symbol templates in display CRTs cannot be shared and are therefore generally not economical.

Some displays use the system's vector generator to construct symbols. This technique, although unifying, leads to inefficient coding, because most symbol sections are short relative to the maximum vector length. Also, the scheme is not well matched to the numerous small arcs of symbols.

7. Graphical Input Devices —

The Light Pen, The Potentiometer Pen and the Rand Tablet

The user may input, in real time, freehand sketches, hand printed symbols, and, to a lesser extent, cursive script. He may also point to a displayed item, or press a key.

The "light pen,"[4] a pen tipped with a single photo sensor, has been used extensively for pointing to a displayed item, or, in conjunction with a tracking program, for drawing an input graphic. With multiple stations such tracking programs become demanding. Starting a light pen track ("inking") requires either an exploratory scan, or the return of the pen to an "inkwell" or an existing light spot; the former requires the display of many points, the latter is inconvenient for the user. When the user points to a displayed item, the pen responds at the instant the display spot comes within its field of view. This response immediately interrupts the display program at the particular item. Thus, an interrupt may require one regeneration cycle—this presents no problem when short persistence CRTs are regenerated at 30 c/s.

The "potentiometer pen"[3,7] is a transparent, resistive coating of stannic oxide on glass, placed directly over the display, which is contacted by a pen stylus. It is not an optical device and therefore the preliminary of "inking" is eliminated. Also, mechanical keysets may be eliminated, without sacrificing valuable display area, by defining symbol cells on an area of the coated glass sheet beyond the display. These cells are defined by replaceable alphabet cards placed under the sheet. The user may readily mix graphical strokes and "keyboard" requests using the one input pen.

The Rand Tablet[8], a graphical input surface, encodes the position of a stylus from digital signals on orthogonal sets of printed conductors under a tough mylar surface. This opaque device cannot be used directly over the display; the user references an item by homing a displayed copy of the stylus position onto it. A transparent version of this digital device is feasible and would make sketching easier. Unfortunately, the tablet is much more complex than the light pen or potentiometer pen and therefore more difficult to multiplex.

The Rand Tablet generates the co-ordinates of its stylus directly; the potentiometer pen, indirectly. Thus, item identification by pointing, using either of these techniques, requires a comparison of the stylus co-ordinates with the co-ordinates of every display point. Again, one display cycle may be required, but, unlike the optically dependent light pen, the cycle need only extend to the shared digital display registers. Hence, the dependence of pointing on visual image regeneration is eliminated.

8. Input Graphic Encoding

The techniques discussed in the previous section ultimately determine a sequence of unit x and y grid increments. Section 5 discussed the inefficiency of encoding a graphic at, or near, this elementary level (Freeman coding), and outlined the advantages of piecewise linear encoding for curve storage and processing. The following comments discuss the construction of piecewise linear codes from unit grid input sequences. Identical considerations apply to the encoding of output graphics generated from incremental expressions.

Static and dynamic schemes will be outlined.

A static scheme stores a number of successive input points. It then approximates this section of graphic with a minimum of straight line segments, consistent with some criterion of fit. The scheme requires considerable computation and storage. It is called static because the set of input points are regarded as unordered with respect to time. A problem arises as to the number of points to be stored prior to analysis. Storing an entire graphic, from pen down to pen up, is undesirable; it prevents the real-time detection of specific geometrical features, e.g., cusps or inflexions, which may be required by the central processor to initiate intermediate responses. Storing a variable length of graphic according to local geometry is difficult, because this geometry is itself the object of the analysis.

A dynamic scheme does not store individual points; rather it updates a set of descriptors on receipt of each input increment, and declares breakpoints from these running quantities. After each breakpoint, the descriptors are reset and the procedure repeated. A relatively small amount of computation, mostly simple addition, is required for each input increment. Although dynamic encoding may not produce as good a fit as a static analysis, it does enable the geometrical properties to be readily determined almost in step with the generation of the graphic—this allows a lively interaction between the user and machine.

9. Determination of Geometrical Properties

Piecewise linear encoding generates a string of vectors. In polar form, the magnitude of a vector is an element of arc length, $\triangle s$; thus, total distance along the curve, s, follows by accumulation of $\triangle s$. The direction of a vector is the local curve orientation, θ; thus, the flow of curve direction with arc length, $\theta(s)$, is available. An estimate of the derivative of $\theta(s)$ with respect to s, i.e., curvature, K(s), follows by dividing the differences of successive values of θ, viz., $\triangle\theta$, by the local $\triangle s$. Checks on the continuity of $\theta(s)$ and K(s) must be maintained.

Moments, area, and centroids of various regions bounded by a graphic, and centroids of graphics, considered as lines of uniform density, are readily determined from either the piecewise linear encoding or the elementary input increments. The choice of methods, and the distribution of these tasks between the interface and the central processor are important considerations in the interface order structure.

Recognition or classification schemes require access to the central processor; they are beyond the scope of an interface, and will not be discussed further in this paper.

10. Some Simplifying Concepts

A diversity of approaches to graphical communication have been recorded[1 2 3 9 10]—each approach emphasises a particular aspect, e.g., "Sketchpad" is directed towards machine aided design; another[9] emphasises the recognition of cursive script. The merging of these schemes into a general-purpose graphical communication system raises many problems. However, any simple concepts which have broad application are valuable.

This paper has illustrated that piecewise linear encoding is applicable to both output and input of arbitrary graphics. The central processor encodes an output graphic which it generates incrementally; the interface encodes an input graphic which it receives incrementally—in both cases an identical algorithm can be used.

The determination of the flow of direction and curvature along an input graphic, from its encoding, has been discussed. The reverse process, i.e., the display of a graphic from a specification of $\theta(s)$ or $K(s)$ would further unify input and output. This is facilitiated by the inclusion of an order which is a variant of the plot polar vector order, viz., plot ($\triangle s$, $\triangle\theta$) which specifies the increase in θ from the previous vector; repetition of this order automatically generates a regular polygon approximation to a graphic of constant curvature.

Piecewise linear encoding is applicable to line drawings of any complexity. Each intensified (inked) section of a line drawing is encoded; the line drawing is then considered either as a set of such sections with specified initial points, or as a concatenation of the sections linked with blanked vectors. The linking vectors can be readily included in any particular form of the encoding. A complex drawing can therefore be reduced to a one dimensional expression of the form $\theta(s)$.

A recognition program might well start with a set of graphic sections, each described in terms of its global position, its directional and curvature flow, etc. Furthermore, it is possible to modify, or preprocess, a graphic so described by algorithms, e.g., delete a region of excessive curvature. Such algorithms offer simple solutions to some spatial filtering operations which are useful in recognition.

SECTION III. INTERGRAPHIC— A GRAPHICAL INTERFACE COMPUTER

This section specifies the system structure of a graphical interface computer, INTERGRAPHIC. Many of the decisions have been based on the observations of SECTION II.

Efficient communication at reasonable cost (less than several thousand dollars per user) has influenced many of the decisions. Time-sharing the interface will considerably reduce the cost per user station, but it demands high-speed circuitry for acceptable response times.

INTERGRAPHIC is being constructed of high-speed (nominally, 5 nanosecond) integrated micro-logic. It will link eight consoles to the central processing unit (CPU) via a 4096 word, 16-bit core store. INTERGRAPHIC will be responsible for detailed display generation and distribution, and will service all console requests, including the encoding of freehand graphical inputs.

Interface generated points will be plotted at several megacycles, i.e., a writing rate of $2\text{-}4 \times 10^4$ inches per second, on a display grid of $2^{10} \times 2^{10}$. Arbitrary graphics will be represented by piecewise linear approximations, or, alternatively, by linked sections of constant curvature (specified by compact repeat ($\triangle s$, $\triangle\theta$) orders as discussed in SECTION II, 10). Radii of curvature from approximately 1/100", ($\triangle s$ = I div., $\triangle\theta$ = I radian), to 40", ($\triangle s$ = 64 div., $\triangle\theta$ = 1/64 radian), will be possible. Vectors will be restricted to the range ($\pm$ 64 div., $\pm$ 64 div.) in either Cartesian, (X, Y), or incremental polar ($\triangle s$, $\triangle\theta$) form. The interface will specify $\triangle\theta$ in preference to θ, because it allows greater angular resolution for a given byte length—whereas θ must be encompass 2π radians, $\triangle\theta$ need not; its range will be $\pm$ 1 radian. Values of $\triangle\theta$ in excess of 1 radian will be accomodated by including the order—add a multiple of $\frac{\pi}{2}$ to θ.

It will be possible to set θ directly to any value by including the order—set θ to a multiple of $\frac{\pi}{2}$. Error growth in θ, due to long strings of θ can usually be avoided at the encoding phase; otherwise a string can be controlled by punctuation with set θ orders.

Other interface orders will allow conversion between Cartesian and polar vectors, random point plotting (Cartesian coordinates only), and symbol plotting. Symbols, selected from a set of 128, may be either isolated or linked (typewriter mode).

INTERGRAPHIC will consist of a set of general-purpose registers under high-speed microprogram control in a manner similar to the CIRRUS[11] computer. It will generate increment rates by repeated addition and overflow detection (SECTION II, 3), because the process may share arithmetic hardware required for other operations; special-purpose BRM hardware has no significant speed advantage.

The 14-bit arithmetic unit (AU) will add, subtract, increment, and decrement and will execute the logical operations of AND, OR and EQUIVALENCE.

The AU will also operate as two separate 7-bit units—the cosine arithmetic unit (CAU) and the sine arithmetic unit (SAU). Thus, two independent rate streams will be available simultaneously. SECTION II, 2, has discussed the potential of two rate generators. However, the limited precision of 7-bit (including sign) accumulators would restrict the size of any circles, parabolas or exponentials generated directly in this way. Rather, the 7-bit accumulators will be used to manipulate and display piecewise linear approximations.

There will be two high-speed incremental modes within the interface—rectangular and circular.

The rectangular mode will plot a straight line by continued addition of X in the CAU and Y in the SAU. The additions will be synchronised by common timing pulses and repeated 64 times. There will be X overflows from the CAU and Y from the SAU. These overflows will increment the x and y display registers respectively, in a sequence which generates the line. It will be possible to plot small vectors more rapidly by magnifying their X and Y components by 2^N, and reducing the number of addition cycles to 2^{6-N}.

The circular mode will generate $\cos\theta$ and $\sin\theta$; it will not generate any display points. Initially, $\cos\theta$ will be set to 1 and $\sin\theta$ to 0 corresponding to $\theta = 0$. For each unit of $\triangle\theta$, (1/64 radian), $\cos\theta$ will be added into the CAU and $\sin\theta$ will be added into the SAU. Any overflow from the CAU will increment $\sin\theta$ at its least significant end; any overflow from the SAU will decrement $\cos\theta$ at its least significant end. This process will be similar to the circle generator of SECTION II, 2; except that the x, y display registers will not be incremented. The circular mode will thus update $\cos\theta$ and $\sin\theta$ on receipt of each $\triangle\theta$ value by repeating the cycle $\triangle\theta$ times.

The order, plot ($\triangle s$, $\triangle\theta$), will update $\cos\theta$ and $\sin\theta$ in the circular mode, ($\triangle\theta$ cycles), then display the vector in the rectangular mode with $X = \cos\theta$ and $Y = \sin\theta$ ($\triangle s$ cycles); thereby plotting the vector ($\triangle s \cos\theta$, $\triangle s \sin\theta$). Usually, the direction of a graphic will be a continuous function; thus most $\triangle\theta$ values, and therefore the number of circular mode cycles, will be small. Conversions between Cartesian and polar forms will be executed in the circular mode.

The remainder held in the accumulator of an overflow rate generator must be preserved when hardware is shared, because any spurious additions will cause irregular overflow generation. The interface will provide separate remainder registers for both circular and rectangular modes, so that these modes may be interleaved without reference to core store.

INTERGRAPHIC will form symbols from a set of standard straight lines, quarter circles and quarter ellipses in any of four quadrant variants. Each standard section will require 7 bits for its specification including a blanking and a continue bit. An upper case letter will require an average of 6 sections or 42 bits, and will have a quality equivalent to a 16 x 8 dot matrix. Constituent points will be plotted at 10 Mc/s and the average symbol rate will be 10^5/sec. The microprogram read only store will contain the string codes for each symbol, and very little additional micrologic will be necessary for standard section generation.

There will be no restrictions on the extent or complexity of a symbol—the sections will be joined end to end, blanked where necessary, and terminated by a zero continue bit. Constraining the terminal point to coincide with the starting point will simplify the typewriter mode. A mixed mode option will allow the inclusion of a general vector string for symbols of unusual extent.

INTERGRAPHIC will use analogue potentiometer pens, since they are simple and inexpensive, have adequate precision, do not depend on visual image regeneration and allow off-screen symbol cell indexing. The following binary decision sequence (outlined for x only, in the range $-1 \leqslant x < 1$) will determine the stylus position on initial contact — is the pen to the left or right of $x = 0$? If right then repeat for $x = 0.5$, if left then repeat for $x = -0.5$, etc. After initial contact, the pen position will change only incrementally at rates of less than one grid division per millisecond. Thus, simple counting techniques will be adequate to follow the stylus. The x register will be incremented or decremented according to the sign of the difference between the stylus position voltage and the analogue converted version of x. A common tracking program will service all consoles within a millisecond; hence the interface will update, in real time, the position of any pen in contact from the flow of its x and y grid increments.

Graphic inputs will be encoded as piecewise linear strings. Breakpoints on the graphic will be inserted dynamically by a simple algorithm operating on the flow of pen increments. It will determine the enclosed area between the curve and the chord joining the last breakpoint (or initial point) to the present position, and declare a new breakpoint when the ratio of this area to the chord length exceeds a preset value. Incremental area determination will require simple addition only; chord length determination will require an interface Cartesian to polar conversion; ratio determination will require a short interface sequence equivalent to division. Thus, each tracked point will be rapidly assessed as a possible breakpoint. The preset value will control the fineness of the chordwise approximation and will be a useful parameter for recognition.

Each chord will be available within the interface in Cartesian and polar forms. The compact incremental polar angle $\triangle\theta$, to be used for polar co-ordinate transmission, will be derived by subtracting the accumulation of previous $\triangle\theta$ values from the present θ; this will prevent error growth in the $\triangle\theta$ string.

The flexibility of the INTERGRAPHIC structure under high-speed microprogram control will be adequate for geometric determinations and spatial filtering. However, any procedure requiring many multiplications or divisions, or other extensive computation, will be executed in the CPU.

CONCLUSIONS

A simple low-cost, graphical interface computer can considerably improve the overall efficiency of a graphical communication system by off-loading from the central processor the tasks of detailed image generation, vector string co-ordinate conversion, simple graphic manipulation, input pen tracking and input graphic preprocessing.

The choice of graphic code(s) is of utmost importance; they must be universal, efficient, and operable, so that arbitrary curves may be compactly encoded for transmission and storage in a form suitable for, at least, translation, rotation and scaling. The incremental polar code presented satisfies these requirements. Also, it enables an input graphic to be readily reduced to a one dimensional flow of curve direction or curvature with arc length. Operations on this flow can perform the equivalent of spatial filtering, which is useful for recognition.

Time-sharing an interface demands high performance; this is possible using high-speed micrologic in a structure which is well matched to the graphic code(s). The rectangular and circular modes described have rates which are independent of either core or read only store cycle times, so that megacycle plotting rates are possible.

The potentiometer pen can be easily multiplexed and it unifies tracking, pointing and keying.

REFERENCES

1. I. E. Sutherland, "Sketchpad, A Man-Machine Graphical Communication System", 1963, Am. Fed. of Information Processing Societies Conf. Proc., *23*, 329-346.
2. G. J. Culler and B. D. Fried, "An On-Line Computing Centre for Scientific Problems", June, 1963, Report M19-3U3, T.R.W. Computer Division, Thompson Ramo Wooldridge Inc., Canonga Park, California.
3. B. Hargreaves, et al., "Image Processing Hardware for a Man-Machine Graphical Communication System", 1964, Am. Fed. of Information Processing Societies Conf. Proc., *26*, pt. I, 363-386.
4. R. Stotz, "Man-Machine Console Facilities for Computer-Aided Design", 1963, Am. Fed. of Information Processing Societies Conf. Proc., *23*, 323-328.
5. H. Freeman, "On the Encoding of Arbitrary Geometric Configurations", June, 1961, IEEE Trans. on Electronic Computers, *EC-10*, 260-268.
6. H. Freeman, "Techniques for the Digital Computer Analysis of Chain-Encoded Arbitrary Plane Curves", 1961, National Electronics Conf. Proc., *17*, 421-432.
7. G. A. Rose, "Light-Pen Facilities for Direct View Storage Tubes—An Economical Solution for Multiple Man-Machine Communication", August, 1965, IEEE Trans. on Electronic Computers, *EC-14*, 637-639.
8. M. R. Davis and T. O. Ellis, "The Rand Tablet, A Man-Machine Graphical Communication Device", 1964, Am. Fed. of Information Processing Societies Conf. Proc., *26*, pt. I, 325-331.
9. M. Eden, "Handwriting and Pattern Recognition", Feb., 1962, IEEE Trans. on Information Theory, *IT8*, 160-166.
10. T. Marril et al., "CYCLOPS I A Second Generation Recognition Scheme", 1963, Am. Fed. of Information Processing Societies Conf. Proc., *24*, 27-33.
11. M. W. Allen, T. Pearcey, J. P. Penny, G. A. Rose and J. G. Sanderson, "CIRRUS, An Economical Multiprogram Computer with Microprogram Control", Dec., 1963, IEEE Trans. on Electronic Computers, *EC-12*, No. 6, 663-671.