To be published in the Proceedings
of the AEC Scientific Computer
Information Exchange Meeting,
New York, N. Y., May 2-3, 1974

RECEIVED
LAWRENCE
RADIATION LABORATORY

MAY 6 1974

LIBRARY AND
DOCUMENTS SECTION

LBL-2676
c.2

A BARELY INTELLIGENT TERMINAL

Harvard H. Holmes

March 1974

Prepared for the U. S. Atomic Energy Commission
under Contract W-7405-ENG-48

TWO-WEEK LOAN COPY

*This is a Library Circulating Copy
which may be borrowed for two weeks.
For a personal retention copy, call
Tech. Info. Division, Ext. 5545*

LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA

LBL-2676
c.2

DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

A BARELY INTELLIGENT TERMINAL

H. H. Holmes
Lawrence Berkeley Laboratory
University of California
Berkeley, California 92720

ABSTRACT

A system for effective use of an intelligent terminal for graphics applications is described. It provides extensions to the basic hardware capabilities such as display subroutines and display scaling, and light pen tracking and inking. It supports a variety of local manipulations on display files which have been supplied by a host.

An interrogation facility allows the host to send a list of questions together with the ranges of acceptible answers to the terminal. Thereafter, a single command will invoke an interrogation of the user. Each answer, in turn, is checked for validity and is transmitted to the host only upon completion of correct input. Menus are used when the user must choose one of several alternatives. Each choice of an item may lead to a subsequent menu. These selections are accumulated and the host is interrupted only when the entire sequence is complete.

The editing operations allow changes to be made in the local display image, with or without sending these changes to the host. These operations are sufficient to allow a complete drawing to be constructed locally, without using the host at all. Our applications include ordinary graphs, symbolic modeling , and a drafting application.

The terminal hardware is a DEC GT40 display: a CPU, 8K of 16 bit memory, a display processor, and a communication line to the host. The local data structures comprise a display list, menus, and directories, supported by a simple brute-force memory allocation scheme.

Our goals for this terminal system are to provide fast response for display manipulations and editing. In addition, we anticipate a substantial reduction in computing load on the host for those applications primarily involving editing of displays. A substantial reduction in communication bandwidth makes remote use feasible, eg, at experimental sites via the ARPA net.

Table of Contents

		Page
I.	Introduction	1
II.	Bandwidth vs. Intelligence	1
III.	Software Support	4
IV.	Hardware	7
V.	Implementation	8
IV.	Conclusion	10

I. INTRODUCTION

One of the problems faced by graphics programmers today is how best to use the intelligent terminals which are being produced in ever increasing numbers. At large installations there is an enormous investment in existing hardware and software. One cannot just convert overnight to one of the new devices, but rather it must be integrated into the existing systems. And yet one cannot afford to overlook the possibilities offered by a new device.

We will describe the general tradeoffs involved in the selection and use of a graphics terminal and then we will describe the particular facilities which we plan to offer with our terminal, followed by a rough sketch of the projected implementation and some applications.

II. BANDWIDTH VERSUS INTELLIGENCE

The primary tradeoffs involved in choosing a graphics terminal can be characterized in terms of bandwidth and intelligence. These are two relatively independent variables which are easily understood. The selection of a bandwidth and intelligence will determine the computing load on the host and the applications served. Referring to Figure 1, we have arbitrarily chosen some benchmark tasks which are appropriate for a graphics terminal. These tasks are to display, edit or view in motion either simple or complex pictures. The response time for each of these tasks is considered to be 30 seconds for a display, 1 second for an edited display and 1/30 second for the display of each frame of a moving

display. A simple picture has 1,000 vectors and a complex picture has 10,000 vectors.

We have selected several milestones in our search for a definition of intelligence. Milestone 0 comprises no intelligence, for example a storage tube display or a television monitor. The cost per terminal is about $5,000. Milestone 1 adds refresh memory, a display processor and a CPU. Current systems, such as the DEC GT40 and the IMLAC PDS-4 are in the range of $15,000. These systems are intelligent terminals, able to alter a picture from coded commands and to search a simple data structure. Milestone 2 adds a disk, more memory and more processor and/or CPU power. These systems cost about $50,000. Depending on the application they may either be very intelligent terminals or a minimal satellite processor. Examples are the DEC GT44 and the IBM 1130 with a disk and 2250 display. These systems support a high level language and have enough power for continuous alteration of displays (animation) and for searching complicated data structures (disk resident). Milestone 3, in the range of $150,000, adds enough computing power to generate all the parameters for simple motion, or put another way, to simulate simple dynamic systems in real time. Large mini and midi computer systems fall into this category.

Bandwidth is the data transmission speed of the host to terminal connection. The cost of bandwidth is proportional to the rate and the distance involved. Telephone lines cover the range from 100 baud to 10 kilobaud. Acoustic couplers can be used at up to 300 baud, thus giving true portability. The ARPANET, using special lines, achieves 50 kilobaud. Computer I/O channels achieve bandwidths of 10 megabaud or more, but a channel connection usually requires that the equipment be in the same

room.

Our chart is constructed by assigning a bandwidth to each type of graphics activity: motion, editing or display of simple or complex pictures. Starting at the left of the chart we extend each activity to Milestone 1. After some reflection, we estimate the bandwidth reduction made possible by this much intelligence. To edit a simple picture, for example, we can keep the display in the terminal and transmit only change information. We estimate that this allows a bandwidth reduction of 30 to 1. Complex pictures cannot be so well structured in the limited memory of the terminal so we estimate a bandwidth reduction of 10 to 1 for editing complex pictures. The rest of the chart is constructed in a similar fashion. We then review the chart and make revisions as necessary to eliminate inconsistencies or clashes with common sense or experience.

We can now consider the computing costs of using the host. These costs include both the monetary costs and the costs in user frustration caused by poor response time. We have divided the hosts into two kinds: cheap and expensive. If we assume that the cost of the host is directly proportional to the bandwidth required, then we obtain the lines of constant cost as shown in Figure 2. We have also drawn lines of optimum cost effectiveness for the two types of hosts. If these lines are overlaid on Figure 1, they suggest what we all knew: as hosts become more expensive, the optimum tradeoff moves toward the more expensive terminal. Cheap, responsive hosts can make effective use of less expensive terminals. Unfortunately, the addition of several terminals will often cause the host to saturate, changing it from a cheap host to an expensive host. This is

an obvious reflection of the fixed capacity of the host. If a terminal is connected to a host, it would be wise to expect to process the extra workload in the terminal itself unless the host is suitably upgraded.

III. SOFTWARE SUPPORT

The introduction of terminals at our installation is providing an incentive to re-examine the software situation with a view to providing graphics wherever we now provide a teletype. We will not replace all our teletypes, of course, but we will provide enough graphics terminals so that they will be available to whoever needs them. While we have had CRT consoles for as long as we have had teletypes, they have not been as well received. The reason is that while the teletypes have gone out to the users' work area, the CRT consoles have remained isolated both physically and in terms of programming expertese. Now with the opportunity of providing graphics in the users' work area, we must also make the graphics terminal as easy to program as the teletype.

The first step in this direction has been the establishment of a standard device-independent interface at the FORTRAN subroutine level. We are now in the process of writing device drivers for each type of hardware to interface at this level. Each device driver can clear the screen, plot a sequence of lines, or plot a sequence of characters beginning at a specific point. Drivers for interactive terminals can also read user defined coordinates from a cursor or a tracking cross. This interface allows the user to specify his graphics device when the program is loaded. He may also control two or more devices. The second step in developing

graphics is to modify the several high level graphics languages at our installation to conform to this standard interface.

We will also develop some high level routines of our own: a menu routine, a questionaire routine and a modeling-edit routine. We will implement these on our intelligent terminal as part of the capabilities of the terminal itself. We feel that the implementation within the terminal will be easier and will provide much faster response while reducing the computing load on the host. This immediate response will make it much easier for the novice to learn to use this facility, since the results of each action will be instantly apparent. Frustration for the novice and expert alike will be reduced, since they no longer must wait for a response from the host. The menu facility will allow the host to define a tree structure of labeled nodes and send it to the terminal. When the menu is invoked via a short command from the host or internally within the terminal, the top of the tree is displayed as a menu of light pen sensitive items, either text or symbols; when an item is selected the branches below that node are displayed until a terminal item is selected. The entire path through the tree is then communicated to the host. In this way the user can rapidly select the items which lead him through a complicated command structure while the interaction with the host can be reduced to a few I/O requests.

Figure 3 is a sample menu. When the menu is invoked only the first two lines are shown; after choosing "output", the types of output are shown. After the type of output is selected, the user must confirm his selection. Until the selection is confirmed, the user may change his mind either by starting over at the top or by backing up one level at a time.

The questionaire is similar in spirit to the menu. It is a common facility in programs and it is implemented in the terminal to provide fast response and reduce the computational load on the host. The user is directed to a page of questions and as he answers, each answer is checked for validity. Default values may be provided and any value may be modified until confirmation.

The final and most extensive facility provided in the terminal is the modeling-edit facility which allows the host to send a picture to the terminal for modification by the user. Picture elements are grouped into subpictures and each subpicture may be included in any other subpicture. Any subpicture may be edited by the user; he may add or delete lines, alphanumerics and subpictures. Each change in the picture is sent to the host as it is made, so that when computation is desired, there is no time lost in sending the drawing back to the host. A simple application of this system is for layout of text, drawings, and other documentation. The host may supply some paragraphs of text and some drawings. If each item is a subpicture, then the text and drawings may be moved about on the screen until the proper composition is achieved. This especially suitable for preparing charts, tables and graphs for publication. The editing facility can achieve a pleasing layout far more easily and cheaply than multiple batch runs using trial and error.

The greatest gains can be made in the area of modeling. A typical sequence of the terminal would have the host initialize the terminal with symbols for electronics: resistors, transistors, etc. The user would connect these symbols to form a circuit and the circuit would be analyzed by the host. If the terminal had enough memory, the circuit could remain

in the memory ready for immediate editing as soon as the previous results had been viewed. Several response curves could also be accumulated for comparison by overlaying one on another. Programs for such analysis of symbolic drawings are already available, but the difficulties of using current terminals have prevented their widespread use.

IV. HARDWARE

Our terminal is a DEC GT40 having 8k of 16 bit memory, a PDP-11/05 CPU, a display processor, and a 2400 baud communication line to a CDC 6600. The CPU has six general purpose registers and a hardware stack. Interrupts are handled using an interrupt vector for each device and four priority levels are available. Each I/O device is assigned a pseudo-memory address and all of the CPU instructions may be used to manipulate data at the device address. The display processor includes a vector generator, a character generator and logic for direct memory access. A light pen is also part of the display processor. Light pen and other display processor - CPU interactions are handled with interrupts. The hardware does not provide a display subroutine jump so this is simulated as follows: (1) The display processor executes a stop and interrupt instruction followed by an address, (2) the interrupted CPU finds the address by reading the display program counter and (3) the CPU stacks the address and starts the display processor at the new address. If the new address is zero, then step 2 is reversed and the display processor is returned to a prior picture using the address from the stack.

The host is a CDC 6600 with 128k of 60 bit memory and the usual complement of I/O devices. All teletypes and the GT40 as well interface

through a PDP-8 which handles buffering and local control of the teletype lines. The PDP-8 is line oriented and will only pass information on to the host when a complete line has been received. Characters may be automatically converted from ASCII to the 6600 internal character code or they may be passed in image mode allowing every possible 8 bit character to be sent.

V. IMPLEMENTATION

The terminal program is implemented using the display file as the primary data structure. This file is continuously executed by the display processor so that each change in the data structure is immediately reflected on the screen. Memory is allocated in fixed sized blocks (usually 16 or 32 words) using a bit map to find unused blocks. As each block is filled with display code, the bit map is examined to see if the next block is free; if so, then the display code is continued into the next block. If not, the bit map is then searched for any empty block and a display jump is inserted from the old block to the new block. The bit map allows most blocks to be sequentially allocated and this avoids the overhead of a display jump most of the time. With 32 word blocks, the entire bit map for 8k words requires only a 16 word table, so the overhead required to find new blocks is very small.

The display file proper is organized into a master display list, a directory, and all the subpictures. A subpicture is displayed by putting a display subroutine jump for it into the master display list. The directory contains the names and the first addresses of all the subpictures. It also contains a response code which indicates what is to

be done when a light pen hit occurs on this subpicture. This response code identifies each item as a menu, a questionaire, an ordinary subpicture, or an internal element. This mechanism allows menus and questionaires to be stored as ordinary display elements, while providing proper response to these items. This scheme also allows the terminal to utilize menus and questionaires in its internal operation. The edit commands use this menu facility, for example.

Each subpicture is ordinarily closed, that is, it is defined entirely with relative beam positioning and the last operation returns the beam to its original position. This convention allows a subpicture to be modified subsequent to its inclusion in another picture without disturbing the location of items in the picture.

The terminal software has four discernable levels: memory management, display code generation, command interpretor, and interrupt routines. Every action by the user produces an interrupt which is put into a FIFO queue. As time permits, the command interpretor or the display code generation routines remove these actions and process them. The command interpretor uses the current state and a table to decide which of the code generator routines to call. The code generators ultimately call the memory management routines to alter the data structure.

A typical interaction would begin with the terminal initialized to display a menu. The user points to an item on the menu; after several hits, the interrupt routines put this action on the queue. The command interpretor removes the action, finds the response code and activates the proper routine. This routine may remove the menu and display another menu. The user may now ask to erase a line. The command interpretor sets the proper state and then waits for a light pen hit on a line. When the

hit occurs, the address of the line is passed to the proper code generation routine. This routine will remove the intensity bit from the line. It then must remove the line from the data structure if possible. It will search forward and backward to see if the line is surrounded by invisible lines. If so, these are combined and the deleted words filled with display no-operations. If an entire block of 32 words has been deleted in this manner, the block then must be de-linked and returned to the pool of available memory. The routine then returns to the command interpretor.

This system is implemented in assembly language using a cross assembler running on the host. The host also has available a text editor and a loader which operates over the communications line. Program development proceeds by typing in the new code to be added to the existing program. The code is then assembled and loaded into the terminal over the communication line. This usually takes less than one minute. The code is tested and any revisions can be made and immediately tested again. At the conclusion of the session the new source replaces the old one on the permanent file system. This is a great improvement over most minicomputer facilities; the line printers, magnetic tape drives, and other peripherals of the host are also immediately available.

IV. CONCLUSION

This system will enable an evaluation of the menu, the questionaire and the model editing facilities. If they are satisfactory, then they can be implemented for the PP (peripheral processor) driven CRT consoles

with a minimum of trial and error. This ability to proceed without trial and error is very important since the CDC 6600 operating system is not protected against errors in PP routines.

We feel that this system will go far toward enhancing graphical communication with the user. Our goal is for the user to regard these facilities with the same confidence that he has for the teletype and the text editor.

ACKNOWLEDGEMENT

Worked performed under the auspices of the U.S. Atomic Energy Commission.

REFERENCES

1. Holmes, Harvard., and Austin, Donald M., "Picasso: A General Graphics Modeling Program", ACM SIGPLAN: Symposium on Two-Dimensional Man-Machine Communication, Los Alamos, New Mexico, Vol. 7, No. 10, October, 1972.

2. Newman, William M., and Sproull, Robert F., Principles of Interactive Computer Graphics, McGraw-Hill, 1973.

3. van Dam, Andries, and Stabler, George M., "Intelligent Satellites for Interactive Graphics," NCC, 42, 1973, pp. 229-238.

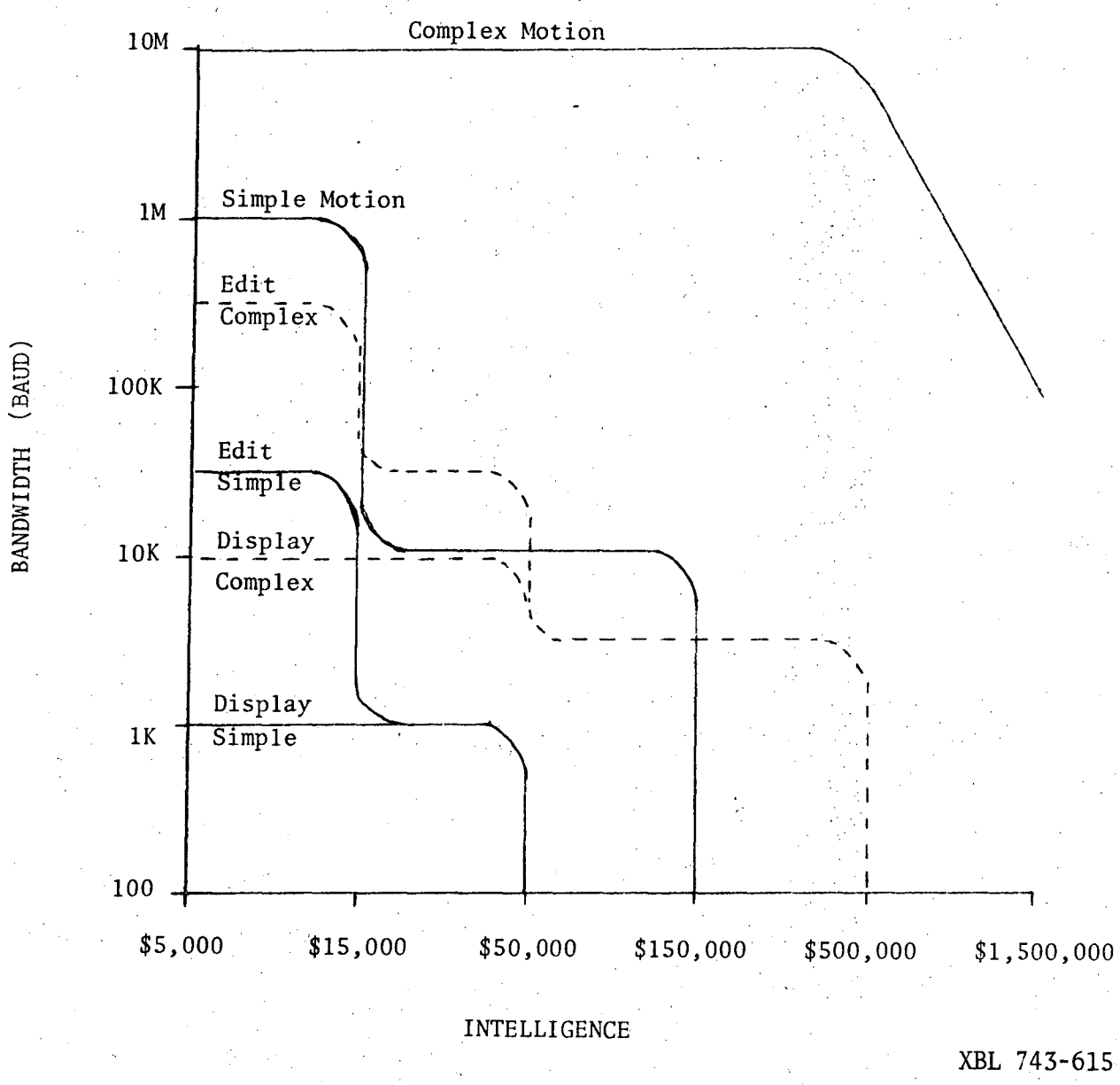
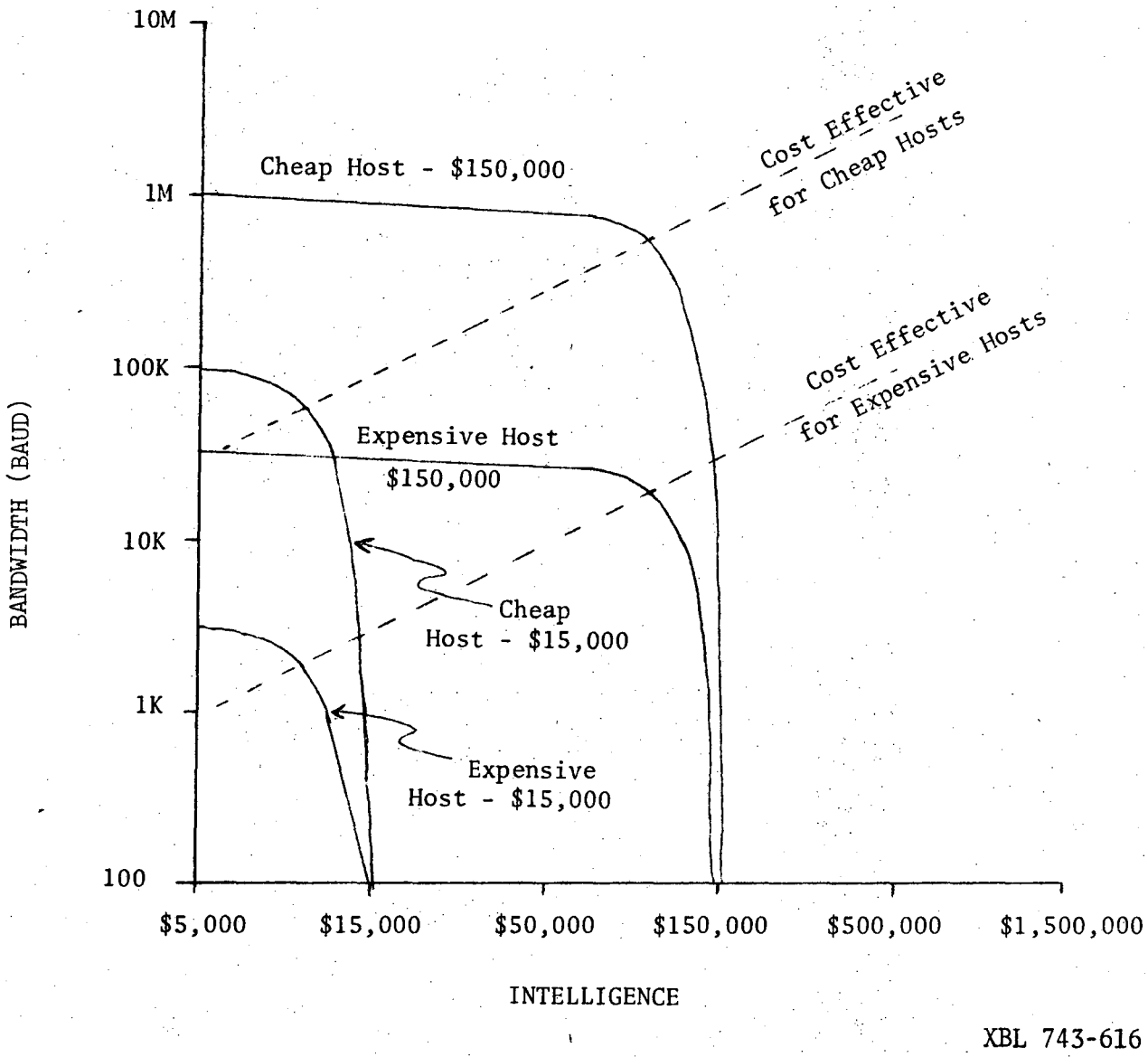


Figure 1. Bandwidth - Intelligence Tradeoffs for Selected Graphics Tasks



XBL 743-616

Figure 2. Lines of Constant Total Cost (Terminal Plus Computing) and Lines of Cost Effectiveness

SELECT NEXT COMMAND

INPUT	OUTPUT	COMPUTATION
	SELECT OUTPUT TYPE	
	HISTOGRAM	
	SCATTER PLOT	
	PERSPECTIVE	
	TIME SLICE	
		XBL 743-617

Figure 3. A Sample Menu

Number of Magnets (0 to 8)? _ _ _ 4

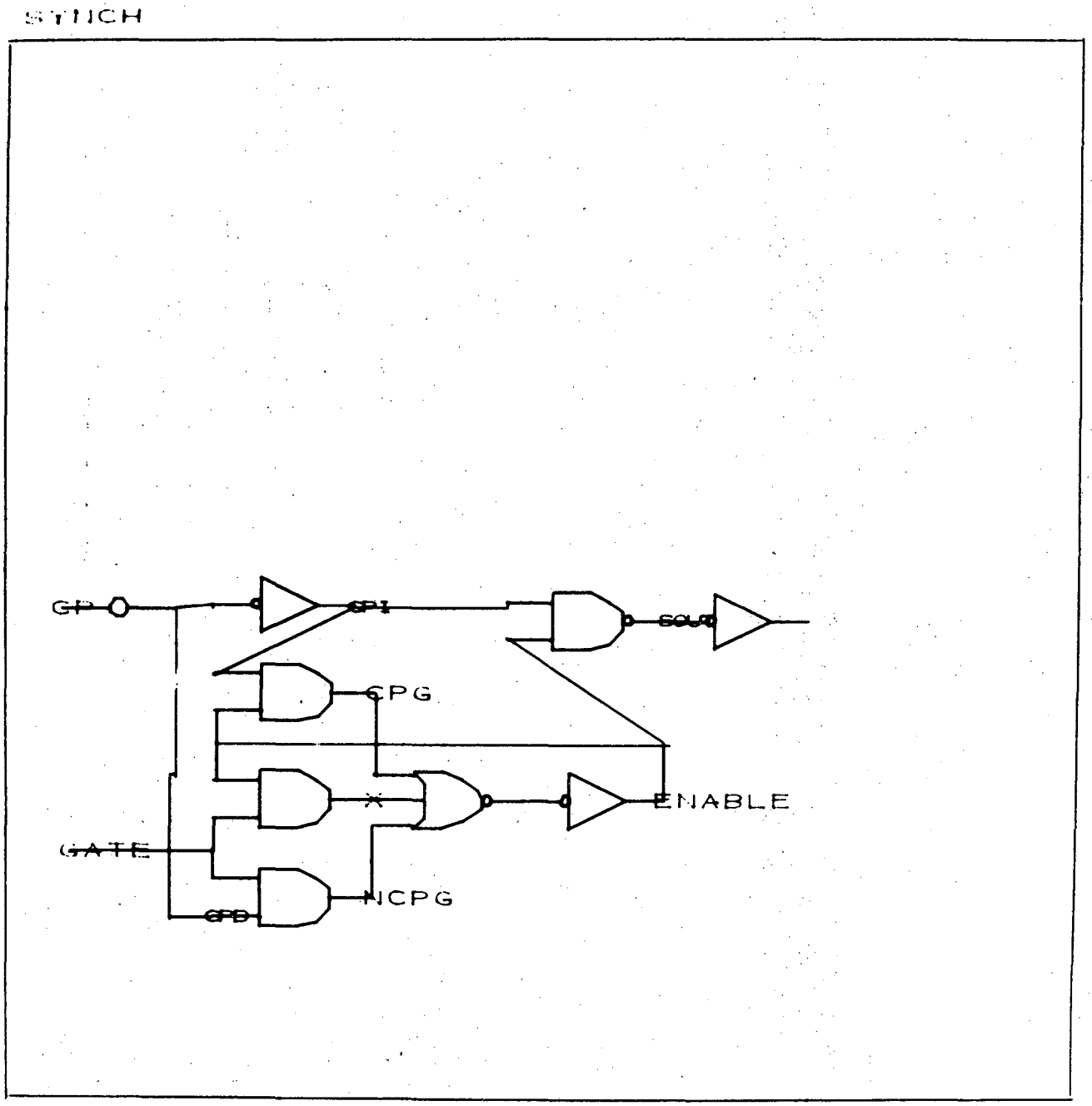
Type of Magnet (Bending or Quadrapole)? _ B _ _ _

Number of iterations (1 to 100)? _ 1 0

CONFIRM (Y or N)

XBL 743-618

Figure 4. Sample Questionaire



XBL 743-619

Figure 5. A Simple Model

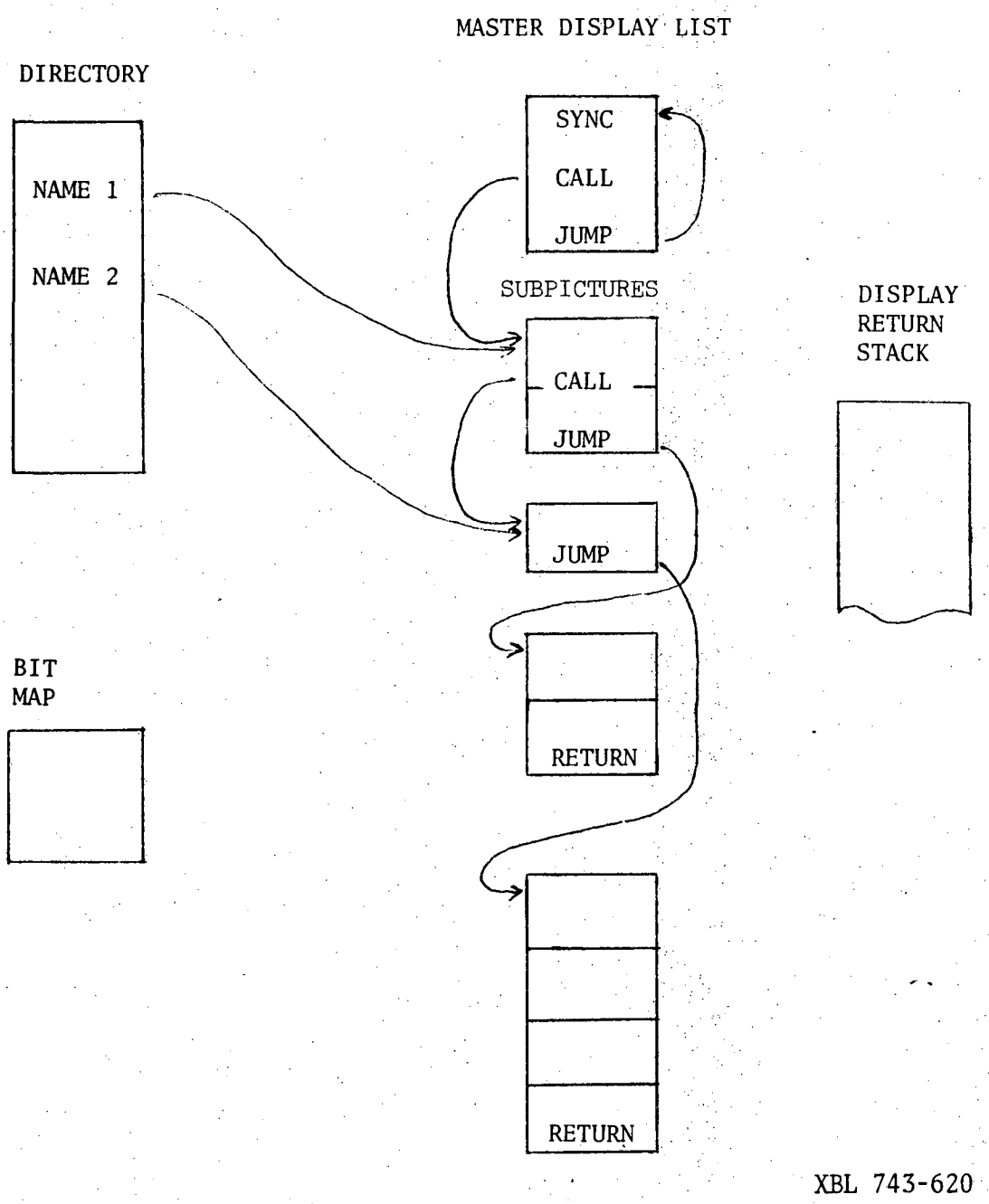
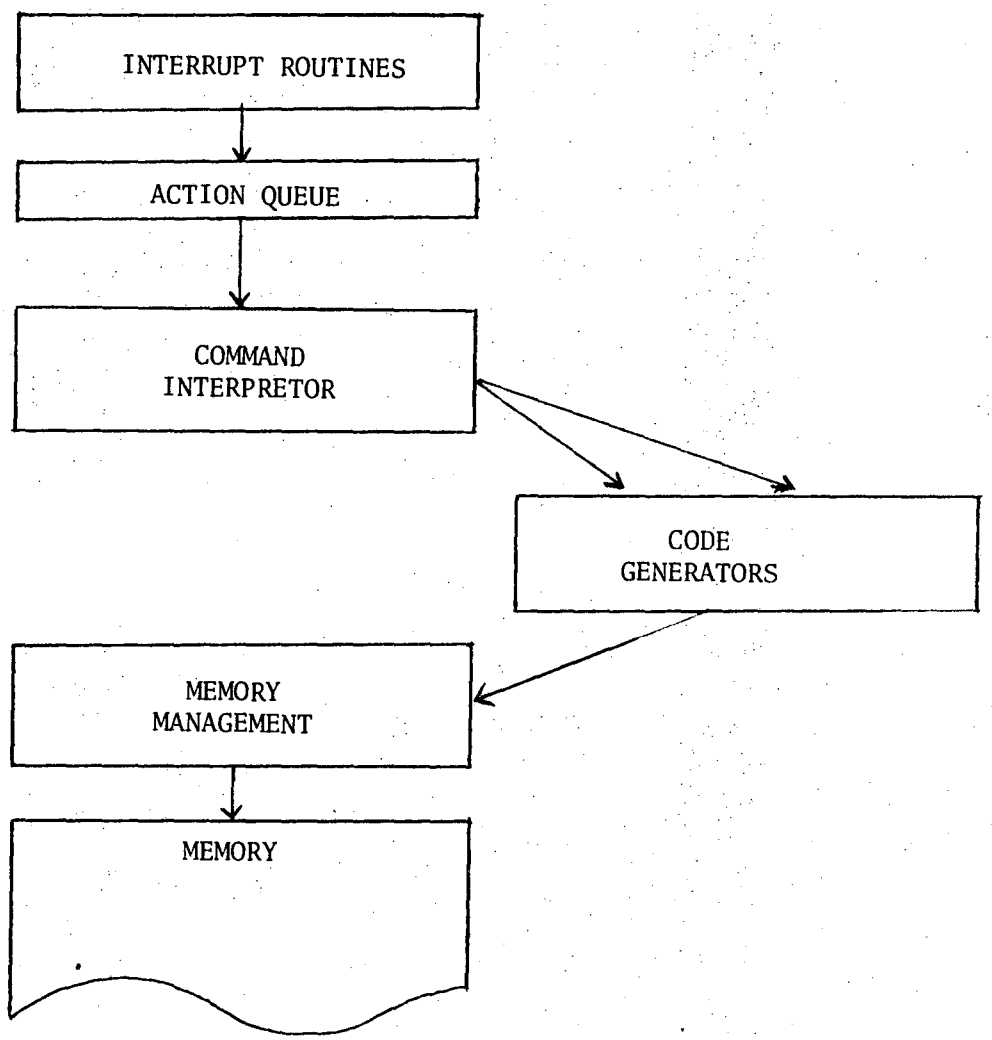


Figure 6. Terminal Data Structures



XBL 743-621

Figure 7. Terminal Program Structure

LEGAL NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Atomic Energy Commission, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

TECHNICAL INFORMATION DIVISION
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720