AD-A042 997 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 9/2
THE DEVELOPMENT OF A USER ORIENTED INTERFACE FOR A COMPUTER DRI--ETC(U)
JUN 77 C E MCNEIL, D P HOUSTON

UNCLASSIFIED NL

1 OF 3
AD A042997








AD A042997

(2)
P-S1

NAVAL POSTGRADUATE SCHOOL
Monterey, California



THESIS

The Development of a User Oriented Interface for a Computer Driven Graphics Device

by

Charles Estus McNeil
and
Dan Patrick Houston

June 1977

Thesis Advisor: George A. Rahe

Approved for public release; distribution unlimited.

AD NO.
DDC FILE COPY

DDC
RECEIVED
AUG 17 1977
A

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

The Development of a User Oriented Interface for a Computer Driven Graphics Device.

5. TYPE OF REPORT & PERIOD COVERED

Master's Thesis; June 1977

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

Charles Estus McNeil
Dan Patrick Houston

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Naval Postgraduate School
Monterey, California 93940

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS

Naval Postgraduate School
Monterey, California 93940

12. REPORT DATE

June 1977

13. NUMBER OF PAGES

211

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

Naval Postgraduate School
Monterey, California 93940

15. SECURITY CLASS. (of this report)

Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

User-oriented
Graphics Software

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The design, development and implementation of a user-oriented graphics software system is described. The over-all considerations involved in such a design process are discussed, along with the major factors affecting design decisions. The object display system is a RAMTEK GX-100A hosted by a PDP-11/50 computer. Implementation techniques are also presented. A → next page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

251 450

detailed user's manual is appended, and recommendations for further system expansion are offered.

Approved for public release; distribution unlimited

THE DEVELOPMENT OF A USER ORIENTED INTERFACE FOR A COMPUTER DRIVEN GRAPHICS DEVICE

by

Charles Estus McNeil
Major, United States Air Force
B.S. Math, Louisiana Polytechnic Institute, 1962

and

Dan Patrick Houston
Captain, United States Marine Corps
B.S. Mech. Eng., Texas A & M University, 19b6

ACCESSION for
NTIS White Section
DDC Buff Section
UNANNOUNCED
JUSTIFICATION
BY
DISTRIBUTION/AVAILABILITY CODES
Dist. SPECIAL
A 23
O.S.

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the
NAVAL POSTGRADUATE SCHOOL
June 1977

Authors: Charles E McNeil

Dan P. Houston

Approved by: Thesis Advisor

Second Reader

Chairman, Department of Computer Science

Dean of Information and Policy Sciences

ABSTRACT

The design, development and implementation of a user-oriented graphics software system is described. The over-all considerations involved in such a design process are discussed, along with the major factors affecting design decisions. The object display system is a RAMTEK GX-100A hosted by a PDP-11/50 computer. Implementation techniques are also presented. A detailed user's manual is appended, and recommendations for further system expansion are offered.

CONTENTS

I. INTRODUCTION..... 6

II. DESIGN CONSIDERATIONS..... 8

III. LANGUAGE SELECTION..... 14

IV. DESIGN OF SYSTEM FUNCTIONS..... 20

V. DOCUMENTATION..... 28

VI. IMPLEMENTATION AND TESTING..... 31

A. PREFERRED TECHNIQUE..... 31

B. MODIFIED TECHNIQUE..... 32

1. VALIDATION/CONDEMNATION..... 32

2. IMPLEMENTATION..... 34

3. DETAILED TESTING..... 34

4. CONTINUED TESTING..... 35

VII. RECOMMENDATIONS..... 36

VIII. CONCLUSIONS..... 38

IX. APPENDIX A..... 39

X. APPENDIX B.....144

XI. BIBLIOGRAPHY.....209

XII. INITIAL DISTRIBUTION LIST.....211

I. INTRODUCTION

Design techniques for user-oriented graphics software systems are investigated. The principles and techniques gained from this research have been applied in the implementation of such a system for a RAMTEK GX-100A display device.

Primary emphasis is placed on providing a user with the properly constructed software support and accompanying documentation for efficient utilization of the graphics display system. The techniques and decisions involved in performing such a design are discussed in detail.

Additionally, the Vector General Tablet was interfaced with the support system. This interface provided the user with an external interactive device for control of the graphics system. The ability to interact with the system greatly enhanced the desirability of the over-all design.

The basic steps in the application phase were to choose a language, design systems functions, write a detailed user's manual, and implement and test the system. The user's manual (Appendix A) provides even the inexperienced programmer with a tool for rapid access to the system. It is complemented with a tutorial set of examples that more accomplished users might proceed to immediately.

The designed system was implemented in the C programming language and hosted by a PDP-11/50 computer in the Naval Postgraduate School Computer Laboratory.

II. DESIGN CONSIDERATIONS

The following is a quote from Dr. Eric Carlson of the IBM Research Division [14]:

> "In the past, progress in terminal hardware technology and graphics software techniques has not been accompanied by similar progress in understanding the requirements of the user and applications of computer graphics".

The design effort, therefore, in computer graphics must be applied to provide clarity in man/machine communication.

The evolutionary process of a user-oriented system must begin with a firm definition of the user. In this context a user is defined to be some person or group of persons who desire to utilize some graphics system. He, for the most part, is a non-computer scientist who simply wishes to be assisted by the computer in the solution of some problem. If he must expend a great deal of effort to learn how to use the system before any benefits can be accrued, then the system appears not to be a tool but just another hurdle.

Any computer system has the ultimate purpose of solving a problem or set of problems. When a user elects to utilize the power of a computer system to solve a problem, then unknowingly he or she has added the requirement that the prob-

lem be stated in two very distinct ways. The problem must on one hand take on a structure that is amenable to the computer, and on the other hand be stated clearly and concisely to man. As the user's statement of the problem is decomposed to a machine-oriented statement of the same problem, the interface between the user and the system is critical. The ideal interface provides a conservational communication or "symbiosis" between man and machine.

The user must be provided as many aids as physically possible to create an atmosphere that is conducive to creative thought. If he expends the majority of his effort attempting to communicate, he is not likely to be very creative. Thus the man-machine communication must be enhanced in every conceivable manner.

The design effort must, necessarily, begin with considerations of available equipment. If the equipment is already in position, as it was in this endeavor, the designer should strive to interface appropriate interactive tools for a pleasant communication scheme. These devices, though quite varied, "can be treated as a physical realization of one of four virtual devices : the pick, the button, the locator, and the valuator" [12]. The four virtual devices are defined below:

A. A pick is used to designate certain objects. The user will often need to point at objects, just as a person may point his finger to denote a particular place or

thing.

B. A button is for programmatic control -- such as selecting a particular action from several possible actions. (commonly a function keyboard)

C. A locator allows the user to locate specific points within his defined image space. (commonly a tablet, joystick, or mouse)

D. A valuator provides the capability to determine a particular value from some defined real space.

Thus in designing a user-oriented system, certain hardware selections/interface decisions must be performed prior to embarking on the software support design.

The design of a user-oriented graphics software system must provide the full capability of the graphics device, satisfy the user, and not overload the host computer. To achieve these goals, a top down approach seems most appropriate while maintaining a constant focus on user requirements. To realize a successful design of a graphics software system the designer must choose a suitable implementation language, provide adequate software support functions, develop detailed documentation, and finally implement the design and perform detailed testing of the package.

The authorities on computer graphics today see two feasible choices for graphic language selection. A graphics language may be developed for the specific device or an existing high-level language, supplemented by a library of

subroutines, may be employed.

The designer who elects to provide a special purpose , device dependent language is normally directing his attentions to a subset of the entire user population. Such a language will often employ peculiar constructs while striving for increased effiency. In previous studies it has been found that the design and implementation of a special purpose graphics language often provides more of a learning experience to the designer than a useful tool to the user. In addition, the development of such a language invariably requires a great deal of expertise in language structures and often snowballs into a costly and time consuming endeavor.

The most common approach to utilizing graphics equipment is to build around an existing language and generate CALLS to a set of subroutines resident in some library. The host language must provide the necessary facilities , support the appropriate data structures, and possess an efficient subroutine call capability. If such a language exists in the design environment, strong consideration should be given to employing it as the base language for the graphics system.

Having considered all options and selected a base language upon which to build the graphics system, the focal point of the design process must be shifted to the logical formulation of the graphical primitives required by the user

population. William M. Newman and Robert F. Sproull [10] profess that "the only graphical primitives that the programmer needs are functions to define points, lines and display text strings". These basic primitives certainly provide the most fundamental set of tools, but by offering selected options to the user a less hostile and more flexible system may evolve.

The foundation, having been envisioned and logically designed, should, when plausible, be expanded to provide the user with a powerful set of high-level tools. The direction of this expansion is necessarily precipitated by the purpose for the graphics system. This process may provide a very precise set of functions in the design of special purpose systems, or a number of task oriented functions which emphasize overall system generality. Each new function spawned by this expansion must provide the user with a new and powerful, yet simple, tool.

Documentation must parallel the entire design process. A vitally important part of this documentation effort is the creation of a detailed user's manual. This user's manual should be virtually complete prior to the implementation and testing phase. The designers have thus stated how the ultimate system is envisioned.

Having reached this plateau, the implementation and testing phase begins; but the documentation effort must continue in parallel. In the end, the complete set of

documentation efforts must include a detailed user's manual, documented source code, synopses of functions, and a trace of the entire process.

The implementation and testing phase begins with the creation of the actual software to perform the proposed actions. As this software comes into being, detailed testing must be performed to ensure that it functions as envisioned.

In summary, the designers must follow some pre-established plan for the development of a graphics software system. In the absence of such a plan, it is almost inevitable that the resulting system will fall short of its expectations.

III. LANGUAGE SELECTION

As previously stated there are two plausible choices in selecting the language to support a user-oriented graphics system. This choice is driven by the ultimate purpose of the system, time, and money. The system designers may develop a new language tailored exclusively for the device, or utilize an existing high level language augmented by a library of functions for control of the device. Regardless of the course of action the selection should "provide a language which is natural, and which does not add to the boredom, panic, frustration, and confusion of the user"[12].

The environment for the design effort associated with this work was for the most part academic and required great flexibility and generality for the widely varied user population. The users of the projected system would come from all backgrounds and with an infinite set of requirements for the system. Due to this need for extreme flexibility a search began to find a suitable high level language to meet the requirements.

The possibility of constructing a unique language to support the system was ruled out almost immediately due to the time constraint of the effort. Lack of time was certainly a valid reason in itself, but other factors also

deemed this course of action unfeasible. The goal was flexibility, and almost always a new language becomes directed to a subset of the user population. The developed system would require on station software maintenance and would receive little or no attention if it existed as a new and foreign language.

Having made the basic choice to utilize an existing high level language the search began for an acceptable one. The Naval Postgraduate School Computer Laboratory hosts the equipment and as such its capabilities and restrictions were considered in detail. There were several languages commonly in use within the confines of the laboratory (C, Fortran, Basic).The most widely used of all the languages was C [3] and as such research was initiated to ascertain the feasibility of this choice for the proposed system.

The C programming language was well documented and allowed the user to write clear and concise programs. It had nice control capabilities making it a very readable language, and it allowed the user to write code that is very compact but still clear. The language encouraged a modular design and provided all of the necessary constructs to implement a viable graphics system. The fact that the language was commonly used by personnel involved in research within the confines of the laboratory was another plus for selecting it for this application. The feeling was that the C programming language would support a completely flexible

and general purpose graphics system; thus the decision to implement the system with it as the base programming language was made.

A software support package existed for the RAMTEK [18] at the onset of this research effort, but its configuration was unacceptable based on the research to this point. From previous studies it was learned that a high level language supported by a library of functions was the most common approach to utilizing a graphics device. The existing software support package did not resemble this basic design philosophy, as it existed as two very large blocks of object code. Regardless of the size or needs of the user program these two blocks of code were merely appended to the compiled program resulting in a very large execution module. This implementation technique was found to be completely unsatisfactory from a design standpoint.

Having decided to use the C programming language and realizing that the existing software support's organization was not in compliance with the prescribed goals, the next step was to develop the structure and organization of a set of library support functions to provide for specific support of the device. A detailed analysis of the existing system was initiated. The system was found to provide the basic capabilities and modular design that was felt would be required. The decision was made to base the new support system on the C programming language plus a library of rudimen-

tary functions secured from the previous software support system.

Prior to embarking on the task of building such a library, the UNIX operating system constraints were investigated to reveal the proper library structure. This environmentally imposed structure was a major concern as the development process began. The reason that no previous attempts had been made to provide the envisioned structure was that it was to be quite tedious and possibly require a great deal of redesign. In the UNIX environment the library is searched sequentially from start to finish to resolve any undefined references from the compilation of a user program.

This linear search procedure employed by the host system dictates that the library have a very precise structure if it is to be efficient. As the library search is performed all references by internal functions must be forward to circumvent the necessity of multiple searches of the same library. Failure to organize the library around this very real constraint could result in very inefficient utilization. To organize the rudimentary library in such a way required indepth study of the existing software.

The existing software support package was studied in detail to determine, without exception, the precise functioning of each module. It was paramount to explicitly learn the path of function calls from each of the existing procedural functions. This information was found to be best

depicted in tabular format.

A table of cross-references for each function was constructed. This table ultimately displayed the calling sequence for each function and what functions called it. This organizational step provided the insight required to begin to organize the library. Had the original design employed a library in the language decision process vice two large "programs" this step could have been eliminated from this design. The fact that the new user-oriented design will be in a library will facilitate more efficient software maintenance and significantly reduce the size of the user's execution module.

The existing package was written in the C programming language and, as such, fitted the proposed language implementation technique quite well. The structure having been exposed, the task was to build a fundamental support library from previously developed software. The functions were further researched to seek out which variables were used by each function. This process was most involved, but necessary, as the environment of UNIx allowed for only singular initialization of each variable.

An organizational chart was developed to ascertain where and when initialization of each variable was to take place. Once the variables had been laid out in a logical initialization sequence, the actual building process could be embarked upon. The conceptual design could be seen as a

feasible task, but an explicit layout of the proposed library had to be made and manually verified. The structure, to be efficient, had to possess no backward references and provide for singular initialization of all variables.

The base library was designed and implemented. After detailed testing it became apparent that the new concept performed better than the old even though there were no physical improvements to the software functions themselves. The library provided much smaller execution modules, as only the user-requested functions were appended, not the entire package as was previously the case. This newly structured library also possessed the benefit of ease of maintenance and expansion. This represented a major step in the overall goal of the project.

The first step in the design of a user-oriented graphics system had been accomplished. An existing language supported by a library of subroutines had been chosen for implementation of the system. Had this step been performed in the original investigation a more viable and maintainable system would have existed from which to depart. It was felt that this structure provided maximum flexibility, efficiency and created a firm base from which to proceed toward a user-oriented system.

IV. DESIGN OF SYSTEM FUNCTIONS

The trend in the development of graphics software systems is toward high-level software support packages that provide simple, powerful controls over the capabilities of the system, yet shelter the programmer from the low-level features of the hardware. The design of the functions that make up these software packages plays a vital part in determining the success or failure of the system as a whole.

In the environment discussed here, the basic machine-level interface (the "driver") between the host computer and the graphics display device had been installed. In addition, a number of software routines had been written for the system. There was no organized library of system functions. Instead, all of the written routines had been conglomerated into two files, including a very large sample program. The requirement to bring both of these large files into memory for every program's execution was considered unacceptable by established design standards.

It was decided at this point that the most realistic approach would be to modularize the existing functions, organize them into a properly formatted library structure, and thoroughly analyze this package for content and functional capability. The design and development of this system

library was discussed in "LANGUAGE SELECTION".

The ensuing analysis of existing routines revealed many shortcomings that required either modifications to the existing code or a complete redesign of that particular function. In addition, the software package as a whole did not offer the user the full capabilities of the system as envisioned by the designers. As well as not providing the full capability of the device, the system was severely lacking in interactive tools. The system did not provide sufficient control and communication facilities for the user. Numerous additional functions were designed, created, and implemented for this purpose.

This phase of the design process was not a purely creative exercise. It was a case of the designers overlaying the existing software with a "template" or "mold" of the system they envisioned; then modifying, restructuring, and expanding the existing package to fit this mold. The system functions that evolved from this "modified" design process are presented in detail in the User's Manual (Appendix A).

The inclusion of an increased interactive capability involved the logical design and creation of avenues to facilitate a more symbiotic man-machine communication system. The only external device available to the user was the keyboard. A basic graphics system can survive with only this capability, but should not be restricted to this device when the possibility for expansion exists. A concerted effort

then began to provide the user with a flexible communication medium. This medium was the Vector General Data Tablet.

In developing this interactive tool to its fullest, the intent was to give this one physical device the capability to appear as four virtual devices (pick, button, locator, valuator). The pick and locator functions could be satisfied via a tablet driven cursor. This capability became one of the goals of the software design. The tablet could also fulfill the valuator function provided it could take on variable dimensions, and give the user the ability to select values from within this space. Finally, the tablet could be broken into user-defined switches that could respond as buttons or function switches for programmatic control.

By logically envisioning the tablet as discussed above, it provided a nice complement to the RAMTEK keyboard. The user, now having two interactive devices at his disposal, would enjoy a more flexible communication interface with the machine.

A cross-sectional view of the software support package would reveal a structure of "tiered complexity". At the base of this structure would lie the basic primitives for a graphics system's operation. Some of these routines are only one step removed from machine language instructions. These primitives form the foundation for higher level function development, but must also be made accessible to the more advanced users' programs. The next level in the structure

would be comprised of more definitive graphics applications functions. Most users could accomplish all their desired work at this software level, but it would likely require numerous and repetitive calls to these functions. The high-level functions that form the crest of this structure are directly supported by the first two levels. These are the functions that offer the user more powerful, problem-oriented options.

This aspect of three levels of graduated potential was applied to each of the functional areas or control modes of the RAMTEK during this design/evaluation phase. This was also the approach used to ensure that the full range of color manipulation capabilities were available to the user. Thus as each basic hardware capability was investigated, the associated software to exploit that capability was logically designed and conceptually aligned with this structure. If the existing software did not conform to this view or did not function to the envisioned specifications for that particular machine capability, it was modified and/or expanded to do so.

Most of the modifications that were required were to functions of the first and second levels. For instance, the procedure for selecting a particular color entry (from the current color table) was poorly implemented. This function had to be completely redesigned to achieve the proper results. Likewise, the routine that existed for the purpose

of establishing a starting point for generating any display had not taken into account certain control mode/control flag combinations that require more than one machine-level instruction to initialize the proper registers. Minor corrections or improvements were made to numerous other functions in the first two strata of the design structure, primarily in the areas of mapping virtual screen addresses to real screen locations, and performing error checks.

The majority of new functions that were designed and created during this process were high-level functions aimed at improving over-all system performance. During the evaluation of the color display techniques, it was recognized that a user could spend a great deal of time in generating the exact colors he desired for a particular display; yet he had no way of saving these color combinations for subsequent use, since the system color tables are automatically initialized upon each new access to the device. This forced the user to regenerate his colors at each session on the equipment -- a time consuming task. This capability of preserving user-defined colors between intermittent sessions was viewed as an absolute necessity. Thus two new functions were developed : one to copy the status of current color tables into a file in the user's directory, and one to subsequently restore these colors back into the system tables after initialization has taken place.

Another capability that was found lacking, and was therefore generated during this phase, was the ability to simultaneously display multiple alphanumeric character strings, such as blocks of text. This function relieves the user of the distracting task of making repeated calls to the mid-level single string display function.

By application area, the largest number of newly-introduced facilities were in the realm of providing an interactive capability between the RAMTEK display system and a Vector General Tablet. Approximately fifteen functions were designed in the afore-mentioned tiered structure to offer the user a wide variety of interactive techniques. The tablet may thus be used for manipulating or generating color tables; directing the movement of a software-generated cursor on the display screen; selecting program execution options from a menu of logical tablet switches; or, with proper application of lower-level functions, "sketching" a display onto the screen.

The modified design approach that was taken, in conjunction with the tiered-structure view of the software, proved highly efficient by preventing the loss or waste of any previously developed functions, while readily identifying any lack of functional capability in a particular area.

The envisioned system and the newly introduced capabilities, in order to provide a truly user-oriented environment, required a continuous reflection on the user and his

psychological behavior. The user's basic psychological requirements are time dependent [17]. The user expects certain responses in a specific time frame or he becomes frustrated and displeased with the system. Each specific function was evaluated for possible unfavorable psychological responses.

The psychological make-up of the user was of prime importance in the actual design of the interactive color manipulation routine tabcolor(). This function allows the user to interactively modify and/or create system color tables. The expectations and possible frustrations of the user were researched in detail in this regard.

Thus the user was recognized to have certain expectations [17]; and a concerted effort was made to ensure that the system could adequately meet these expectations. The competing properties of efficiency and simplicity remained fundamental considerations throughout the developmental process.

When possible, the system functions were designed to employ "interaction by anticipation"[15]. This approach decreased the required number of diagnostic messages and should result in a more satisfied user.

Satisfactory response times were also considered in the over-all system design. It was found that users could become frustrated and annoyed by the improper timing of responses.

For example, when a user performs incorrectly, the system should not respond immediately, but delay from two to four seconds. This delay allows for psychological closure on the user's part, after which time an error indication is more acceptable.

The logical design of system functions thus involved validation of existing software, design of new software, and constant reflection on the user's psychological needs.

V. DOCUMENTATION

The documentation effort in this thesis work was a continuous process that paralleled all other phases of the project. The lack of sufficient prior documentation was considered as one of the motivating factors behind this work. This deficit was certainly one of the major hindrances in performing the object system design. Therefore, the formulation of proper documentation for this software system, in particular a well-conceived user's manual, was of utmost importance to the designers.

An attempt was made, throughout the design/evaluation process, to record every finding, every result, every achievement, no matter how insignificant it might have seemed at the time. These notes, though seemingly unorganized, served as valuable reflections during the implementation and testing phase, as well as when this writing began.

It was the ultimate goal of this thesis to develop a maximally user-oriented graphics software system accompanied by an equally user-oriented user's manual. Without the latter, the former would surely go unrecognized and unused. With this in mind, a great deal of thought and preparation were devoted to developing such a manual.

The resulting manual (Appendix A) is organized along the following lines. First there is an introduction to the basic system environment and to the manual itself. Next is a guided tour of the steps required to gain access to this environment. This is followed by a tutorial review of the basic programming knowledge a user should possess to approach this system with some degree of confidence. The next chapter actually begins to reveal some of the features of the graphics display system, and exposes the basic programmatic control functions the user will need. The remaining chapters (except the last) are each devoted to one particular functional capability or control mode of this system. This will, hopefully, provide a more systematic revelation of the over-all capabilities and features of the display system.

As each of these areas is explored, the associated system functions are presented in a top-down fashion. After an introduction to the basic application area, the high-level functions are introduced first, followed by progressively lower level functions. This approach was taken in an attempt to alleviate the difficulties of the novice user in gaining familiarity with the system.

One chapter of the manual is devoted to each of the primary interactive devices : the RAMTEK keyboard and the Vector General Tablet. The final chapter presents routines which are not an integral part of any particular functional

area, and had thus not been introduced earlier, but which might prove useful in some applications.

The individual system functions, in the UNIX format [6], are included as Appendix B for further clarification and/or review by the user. In addition, a copy of the source code for each of these functions (with explanatory comments) is available in the Naval Postgraduate School Laboratory.

VI. IMPLEMENTATION AND TESTING

A. PREFERRED TECHNIQUE

Ideally the implementation phase of a computer system design should only account for 25-30% of the total effort. The functional design is complete and a detailed users manual exists for the proposed system. Actual implementation should then consist of coding, debugging, and localized testing.

The coding effort should offer little challenge, as each function has been laid out in detail. It should be merely a translation of ideas into a suitable medium for the computer. The debugging of each function affords more of a challenge, as without fail logic errors will exist in the basic design.

As the debugging process begins the documentation must be kept current or in the end the documents will not accurately depict the system as it actually performs. If documentation and reality ever begin to diverge then an untenable situation is likely to be created, in the form of an unmaintainable system.

As functions are brought on-line, local testing will be conducted -- local testing being those tests performed by

the person responsible for a particular functions implementation. The problem with this local testing is that often the tester is too close to the problem to provide a good test. After the local testing is complete, the detailed testing of the function should be performed.

This detailed testing should be performed by some person or preferably some group of persons who have no stake in the success or failure of the system. Ideally this testing will be done by a team which would spend as much time and effort developing the test plan as went into the design. All too often this last microscopic testing is omitted, with the end result being a system full of implicit assumptions and surprises for the unsuspecting user.

B. MODIFIED TECHNIQUE

The preferred technique could not be applied to this effort and was thus modified to meet certain restrictions and needs. Hopefully the end result was the same -- leaving a viable and truly user-oriented system. The reasons for the modifications were that a test team for the thesis was unrealistic, and that an existing system afforded a point of departure in search of the user-oriented system.

1. VALIDATION/CONDEMNATION

Due to the fact that a software support package was already in existence for the RAMTEK this modification was

injected. It amounted to having three phases being performed simultaneously. The three phases were FUNCTION DESIGN, DOCUMENTATION, and TESTING.

While operating in parallel, as proposed functional requirements were enumerated, the existing software and its associated documentation were searched to determine if such a function already existed. If the existing system possessed such a function then the validation/condemnation process began.

This process was fundamental, as it either validated the performance of the function or it caused it to be condemned. The process, though simple conceptually, was one of the most difficult and time consuming stages of the systems evolution. If the function performed exactly "as advertised", then the action was to merely document and continue. When the function did not act as expected, then the often lengthy and involved search began for the sometimes illusive logic error.

The need for this unrewarding experience was most certainly created by inadequate testing of the previous system. This inadequate testing had also caused the "old" system to fall prey to idleness, as users do not relish the thought of using a system full of frustration and surprise.

2. IMPLEMENTATION

The fundamental support functions of the predecessor to this user-oriented system had been installed and tested prior to the start of the implementation phase of the modified approach. The volume of change was small when viewed against the entire system, but essential in a truly user-oriented system.

The actual coding effort of the selected additions to the now dependable base system only accounted for approximately 10% of the total effort. This phase was actually conducted exactly as discussed in the PREFERRED TECHNIQUE section. Once the coding, debugging, and localized testing was complete, then detailed system testing began.

3. DETAILED TESTING

The detailed testing of the overall system was conducted in compliance with the preferred technique. The use of an external testing source was not feasible, thus these suggestions were bent slightly to accomodate the circumstances of the design.

The dual thesis effort afforded a viable test via oscillation of duties between designers. The person responsible for the actual implementation of a function was not responsible for it's testing. Thus the designer's counterpart became the devil's advocate during the formal test phase.

In addition to the formal testing, user testing was not only encouraged but sought. The system, when possible, was made available to users -- requesting that they convey their likes and dislikes, successes and failures. The system, having been designed for the user, was then evaluated very critically from user comments. For the most part response was favorable. When unfavorable responses occurred they were studied, and every attempt was made to restructure those aspects that were found to be distasteful.

4. CONTINUED TESTING

In order for a system to really claim to be user-oriented, it must be placed in a continual test posture. This continual testing is informal but allows for response to user criticism. The user is afforded the facility for voicing his problems with the system. These user assessed problems must be evaluated and acted upon. The action need not be drastic, ill-advised system modification, but an honest look at the problem. If corrective actions are in order, then the documentation and software must be modified in unison.

The RAMTEK GX-100A graphics support system is currently installed and undergoing the continued testing phase. There are no known problems, but the facility for voicing such problems exists in the form of a trouble report attached to the device.

VII. RECOMMENDATIONS

The system, as configured, should now offer a productive developmental environment. This newly created environment will, hopefully, encourage fresh innovative research in new applications of color graphics. Two such areas surface almost immediately as likely candidates for meaningful research. These two areas are:

COMMAND, CONTROL, AND COMMUNICATION; and

SIGNAL PROCESSING.

The command, control, and communication applications are bounded by imagination alone, as this field is still evolving. The field of computer graphics will most surely be one of the front-runners in the implementation of such systems. The color capabilities of the RAMTEK should provide a powerful tool in this regard.

Limited signal processing applications have already been performed on the device. The new system should encourage more use of color in this area. The color display will most certainly convey more useful information.

In addition to the new applications, it would be beneficial if the saving of complete screen images were possible. The memory space required for this application would require serious consideration but could be arranged to

permit efficient utilization. The memory readback interface was not installed on this equipment, but with its installation and minor software efforts it would be possible to memorize screen images. This memorization process could eliminate the need for repetitive processing while providing a powerful tool for the user.

The support library exists as a separate entity and is intefaced via the C programming language. Possible research on the capability to access this library via other languages would be a worthy experience.

The ever continuing maintenance of this and all of the systems is of the utmost importance. Similar efforts as conducted on the RAMTEK could be undertaken for other grapics devices in the laboratory.

VIII. CONCLUSIONS

An existing software support package was disassembled, then restructured, tested, and expanded into a usable color graphics display system. The goal, as a user-oriented system, was achieved; but the design approach and the solutions to problems encountered offered the real lessons in this learning process. These were the most important results to the designers.

In redesigning a software system based on a prior design, it was possible to have 20/20 hindsight into the failures of the previous attempt. The most profound judgement derived is that detailed planning must precede implementation efforts. The parent system that spawned this user-oriented child was not planned in detail nor well documented. Had the steps utilized in developing this new design been employed in the previous effort, there would probably have been no need for major revision and exhaustive testing.

The system has been installed and tested. Adequate documentation has been provided to assist users in the efficient use of the device.

APPENDIX A

USERS MANUAL RAMTEX GX-100A

This manual is designed to provide the users of the RAMTEK graphics display device a convienient and simple introduction to the device. The manual allows for as detailed an investigation into the system as the user desires.

CONTENTS

I. INTRODUCTION..... 43

II. OPERATING INSTRUCTIONS..... 46

III. BASIC PROGRAMMING REQUIREMENTS..... 51

IV. PROGRAMMATIC CONTROL OF THE RAMTEK..... 58

A. Opening the Device..... 58

B. Virtual Screen Dimensions..... 59

C. Virtual Screen Addressing..... 60

D. Control Modes and Flags..... 62

V. VECTOR GENERAL DATA TABLET INTERFACE..... 67

VI. COLOR DISPLAYS..... 72

A. BASIC USER GUIDELINES..... 73

1. PROGRAM 1..... 75

2. PROGRAM 2..... 76

3. PROGRAM 3..... 78

B. ADVANCED COLOR METHODS..... 79

1. MODIFYING COLOR TABLES..... 79

a. DATA TABLET METHODS..... 79

(1) PROGRAM 4..... 80

b. KEYBOARD METHODS..... 81

(1) PROGRAM 5..... 81

c. MANUAL COLOR LOADING..... 82

(1) PROGRAM 6..... 83

2. SAVING COLOR TABLES..... 85

VII. ALPHANUMERIC MODE..... 87

A. INTRODUCTION..... 87

B. BASIC USER GUIDELINES..... 88

1. PROGRAM 7..... 91

C. ADVANCED METHODS..... 92

1. PROGRAM 8..... 96

2. PROGRAM 9..... 97

VIII. TRANSVERSE DATA MODE..... 99

A. BASIC USER GUIDELINES.....100

1. SPECIAL SYMBOLS.....100

a. PROGRAM 10.....103

2. IRREGULAR VERTICAL IMAGES.....104

a. PROGRAM 11 105

b. PROGRAM 12.....106

B. ADVANCED METHODS.....107

IX. RASTER DATA MODE.....109

A. PROGRAM 13.....110

X. COMPLEX DATA MODE.....112

A. PROGRAM 14.....114

XI. GRAPHIC VECTOR MODE.....116

A. INTRODUCTION.....116

B. BASIC USER GUIDELINES.....116

C. ADVANCED METHODS.....118

1. PROGRAM 15.....119

XII. GRAPHIC PLOT MODE.....122

A. INTRODUCTION.....122

B. BASIC USER GUIDELINES.....123

1. PROGRAM 16.....124

C. ADVANCED METHODS.....125

XIII. GRAPHIC CARTESIAN MODE.....126

A. INTRODUCTION.....126

B. BASIC USER GUIDELINES.....126

C. ADVANCED METHODS.....127

1. PROGRAM 17.....128

XIV. GRAPHIC ELEMENT MODE.....130

A. INTRODUCTION.....130

B. BASIC USER GUIDELINES.....130

1. PROGRAM 18.....132

C. ADVANCED METHODS.....133

XV. RAMTEK INTERACTIVE KEYBOARD.....134

A. PROGRAM 19 135

B. PROGRAM 20.....136

XVI. SPECIAL APPLICATIONS ROUTINES.....140

I. INTRODUCTION

The RAMTEK GX-100A graphics display system utilizes a raster scan technique, with the display image data being extracted from an internal refresh memory. The RAMTEK system in the Naval Postgraduate School Computer Laboratory is hosted by a PDP-11/50 computer and is accessed through this computer, and its UNIX operating system, from one of the terminals in the Lab.

It is not the intent of this manual to provide a detailed analysis of the hardware or internal functions of the RAMTEK device, but rather to expose the user to the software support package that has been developed for it. This software package consists primarily of a system library of subroutines written in the 'C' programming language. Included in this library are a number of routines that enable the user to interface the RAMTEK with the Vector General Data Tablet in the Lab.

A review of the Table of Contents would reveal the basic organization of this manual to be as described here. Following this Introduction is a chapter devoted to familiarizing the user with the environment in which this graphics system exists. The next chapter presents the fundamental programming knowledge required of a prospective user.

Following this, the basic control functions of this system are described. Next the user is introduced to the data tablet interfacing routines, followed by color manipulation techniques. After this, there are eight chapters that each describe one of the control modes of the GX-100A. Then comes a chapter detailing the use of the interactive keyboard. The final chapter discusses the remainder of the software support package. Where appropriate, the separate chapters are divided into three sections : Introduction, Basic User Guidelines, and Advanced Methods.

The manual will serve as a tutorial on the subroutines available to the user for generating displays on the RAMTEK terminal. There are numerous sample programs included to allow the user to become familiar with operating the system and to exhibit certain features that are available in the RAMTEK software support package. These sample programs, along with several larger demonstration programs, have been compiled into a special directory in the host computer file system (see Operating Instructions). The user is encouraged to execute each of these samples as he progresses through this manual.

For the user that requires a more detailed knowledge of the internal functioning of the device, the RAMTEK GX-100A Programming Manual [1] is available in the Lab.

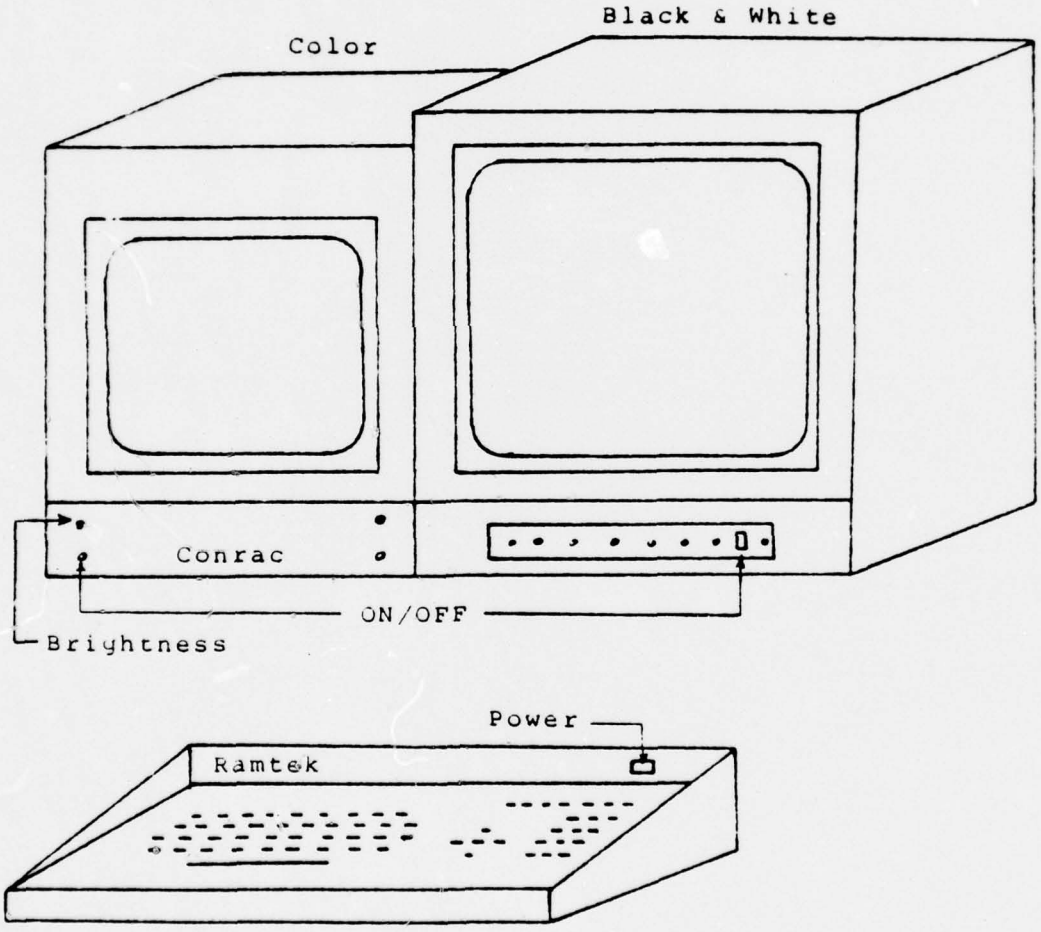


FIGURE 1

II. OPERATING INSTRUCTIONS

The RAMTEK display device is accessed via the "A" side PDP-11/50 in the Computer Lab by using one of the terminals connected to this computer. If the user is not familiar with the operation of these terminals, he should consult the Terminal User's Manual [2] available in the Lab.

In order to utilize the RAMTEK, the device must be "powered-on" and supported by the PDP-11/50 computer. By following the 12 steps listed below, the user will ensure himself of a successful session with the equipment.

1. Ensure the "A" side PDP-11 computer is in operation. (If in doubt, check with Lab staff.)
2. Check the green power supply light/switch (marked "RAMTEK POWER") at the lower rear of the host PDP-11. If this switch is NOT illuminated, contact one of the Lab personnel for assistance. Indiscriminate use of this switch will cause the host system to "crash".
3. Apply power to the RAMTEK keyboard and monitor :
   a. the green "POWER" switch on the upper right portion of the keyboard should be illuminated; if not, depress the switch.
   b. select the "ON" position of the selector knob at the lower left corner of the color display monitor. (See Fig. 1)

c. (optional) select the "ON" position of the power switch on the lower front panel of the black & white monitor (See Fig. 1)

d. (optional) select the "ON" position of the power switch on the Vector General Data Tablet.

4. Choose one of the "A" side terminals and turn it on.

At this point, the user is ready to log into the UNIX operating system and proceed with the session. The user that intends to execute one or more of the sample programs from this manual or view one of the RAMTEK demonstration programs should continue with step 5. For the user that has been established in the system with his own identification and directory, the most common activity is program development and testing. This user is assumed to be familiar with the operating environment and is referred to Chapter IV, Programmatic Control of the RAMTEK. The user with only moderate experience in the UNIX environment may wish to scan Chapter III, Basic Programming Requirements, before proceeding to Chapter IV. These users may return to step 10 below for termination procedures.

5. At the selected terminal, the completion of the following sequence will lead the user to the system directory that contains the sample and demonstration programs (note : lower-case type implies a user-typed command, upper-case type denotes a system reply, "%" is the UNIX prompt signal, and "c/r" means depress the RETURN key) :

LOGIN: cs2000 c/r

PASSWORD: student c/r

% cd ramtek c/r

6. To determine what source programs are available for inspection, the user should type :

ls *.c c/r

This will result in a listing of the names of all files in the cs2000/ramtek directory that are of type ".c". This list of file names will appear on the terminal screen. The files that are named p1.c through p20.c are sample programs that are described later in this manual. The other files listed are of no concern to the user at this time.

7. To inspect one these files, the user should type :

list -c filename.c c/r

c/r

where "filename" denotes the name of the particular chosen file. The second (and subsequent) "carriage return" will cause the next "page" of source code to be printed on the terminal screen. For example, to inspect the source code of the first sample program, the user would type :

list -c p1.c c/r

8. To execute one of the sample programs and view the resulting display on the RAMTEK, the user should type:

filename.x c/r

where "filename" is the name of the selected sample

program. After depressing the RETURN key, the user should turn his attention to the display screen.

9. In addition to the sample programs mentioned above, there are also four demonstration programs in the cs2000/ramtek directory. A detailed explanation of these demo programs can be found in the "User Entry Manual" [2]. To view the first of these demonstration programs, the user should type :

ram1 c/r

Subsequent demo programs are called ram2, ram3, and ram4. These demo programs offer the user the opportunity to interact with the RAMTEK keyboard and the data tablet.

After completing the desired work for one session, the user should proceed to step 10 for termination procedures.

10. For the user who is currently editing a file (ie, creating or modifying a file in his own or a class directory), care must be taken to ensure the latest additions/corrections are incorporated into his program. This is accomplished by typing at the terminal :

w filename c/r

Then typing :

q c/r

will get the user out of the "edit" mode. (see Chap. III)

11. To terminate the session on the terminal, type :

quit c/r

12. To "power down" the RAMTEK, the following steps should be accomplished :

a. Turn off the green power switch on the RAMTEK keyboard by depressing the switch.

b. Select the "OFF" position of the selector knob at the lower left corner of the color display monitor.

c. If the black & white monitor is on, select the "OFF" position of its power switch on the lower front panel.

d. If the Vector General Tablet was being used, select the "OFF" position of its power switch.

III. BASIC PROGRAMMING REQUIREMENTS

In order to utilize the RAMTEK display device the user must have access to the computer laboratory and a login name in the UNIX time sharing system. The fact that the user is established in the UNIX operating system allocates the space required for program development . There are two separate PDP- 11/50 computers in the laboratory and they are referred to as the "A" and "B" sides. The "A" side hosts the graphics display devices of the laboratory; the "B" side is used for general program development. In order for a user to effectively utilize the system he should have access to both sides . This allows the user to create and debug programs on the "B" side, then execute the programs on the "A" side.

The C programming language [3] was used to develop the software support for the RAMTEK and, as such, the user will be required to program in C in order to interface with this software. The C language is similar to FORTRAN but provides a great deal more flexibility. A C program consists of data declarations and one or more functions. This language has five fundamental data types:

int (integer)

char (byte)

float (single precision floating point)

double (double precision floating point)

register (same as int except in a register)

Declarations may be made internally or externally, with respect to a function, indicating either local or global scope respectively. Global scope implies that the variable is known to all functions that follow; local scope implies that only the function that contains the declaration has knowledge of that variable.

A function consists of a set of statements enclosed in braces ({}). Every C program must contain a function called "main" since execution of a C program begins with the first statement of "main". All functions must be explicitly declared (including "main") by a statement of the form : function-name(optional arguments). The optional arguments are the way that data are communicated between functions. The arguments must be enclosed in parenthesis and, even if no arguments exist, the parenthesis must always be present. The statements within a function are always terminated with a semi-colon and, as previously stated, the body of the function is enclosed in braces. "Main" will normally invoke other functions to perform certain tasks; this is accomplished by stating the function name followed by the argument list and terminated by a semi-colon.

With this very cursory look at the C programming language the user should be able to understand the example programs of this manual and write simple C programs. The user is encouraged to read the C tutorial [4] and the C

reference manual [3] for a more detailed description of the language.

It is assumed here that the user has some knowledge of the UNIX operating system, at least to the level of having read the "UNIX for Beginners" tutorial[5]. Thus he will understand that files may be created by invoking the UNIX TEXT EDITOR, giving the command to append information to the file, inserting the information, terminating the append mode, and writing the file. The following short exercise should refresh the users memory.

```
LOGIN: "user login name"   c/r
PASSWORD: "user password"   c/r
ed sample.c  c/r      (invokes text editor)
a   c/r                (enter append mode)
main()   c/r
  {int a,b,total;   c/r
   a = 1; b = 2;    c/r
   total = a + b;   c/r
   printf("sum is %d0,total);   c/r
  }   c/r
.   c/r
w   c/r                (write file)
q   c/r                (quit edit mode)
```

This example creates a file (C program) called sample.c that, when compiled and executed, sums two integer values a and b into total and then prints the result.

The user should recall that in order to compile a C program he must invoke the C compiler by using the "cc" command. To compile the above example the user would type :

cc sample.c c/r

The result of a C compile is an executable file named a.out. This file is created if one does not already exist or replaces the old a.out file if one existed previously. In order to save an old a.out file the user may rename it by utilizing the UNIX function "mv". For the user to save the a.out file created by compiling sample.c he could type : mv a.out sample c/r . This changes the name of a.out to sample.

Having been refresher on the very basics of the C programming language, UNIX, and compilation procedures, the user is ready to begin writing, compiling, and executing programs specifically for the RAMTEK graphics display device. The user-designed programs for the RAMTEK will be dependent upon the RAMTEK software support library. This library contains a multitude of user oriented routines that allow him to perform certain tasks on the device. These routines are actually C functions and in order to invoke one the user must only mention it's name. Since this support routine is not within the user defined C program, it is undefined unless a link is provided to the support library. This link is provided via a shell command called "ramtek". Thus in order to compile a program that references functions contained in the library, the user types :

ramtek filename.c c/r

This will issue a "cc" command with the appropriate parameters to link to the software support library and resolve the previously undefined function calls. The following example typifies this procedure:

```
LOGIN: "login name"   c/r
PASSWORD: "user password"   c/r
ed CT5.c   c/r
a   c/r
main()   c/r
   {ramtek(); erase();   c/r
    diso(7);   c/r
   }   c/r
.   c/r
w   c/r
q   c/r
ramtek CT5.c  c/r
```

The result of the above example will be the creation of a file CT5.c (the source code) and an a.out file (the executable file).

As previously stated the user, for efficiency reasons, should create and compile his programs on the "B" side of the laboratory facilities. If the user has followed this guideline he now has an executable program that will display some colors and text on the RAMTEK screen (the user should not be concerned with how this is done, but merely with the procedure), but the executable program is on the "B" side and the "A" side actually hosts the graphics display dev-

ices. The user must then transport his executable program from the "B" side to the "A" side.

The transportation of information from the "B" to the "A" side may be accomplished by using the system functions "put" and "get". In order for the user to transfer the a.out file created from the compilation of CTS.c he must:

(1)be logged in on both sides simultaneously

(2)from the "B" side type

put a.out c/r

(the system will respond DONE when the action is complete)

(3)from the "A" side type

get a.out c/r

(the system will respond with DONE when finished)

Now the a.out file created earlier resides on both sides of the laboratory's PDP 11/50 computers. The user is also logged in on both sides and should logout of the "B" side in consideration of other users. The user may now execute his program and see the RAMTEK response. Prior to execution the user should ensure that the GX-100A is powered-on according to the first three steps in the chapter titled OPERATING INSTRUCTIONS. For the user to now execute the program he must type : a.out c/r . The RAMTEK should respond with a display of assorted colors and alphanumeric data on the screen and the terminal should respond with the % prompt.

The user may also use IMP() and EXP() (see UNIX programmers manual [6]) for the transportation of information. These routines provide two way communication but have certain restrictions that make them a second choice.

The user, having followed the CT5.c example to completion, now has two copies of the same file. One file is on the "A" side and the other file is on the "B" side. This duplication of code is a waste of space and as such the user should remove one of the copies. This removal can be accomplished by using the system routine "rm".(type rm a.out c/r)

The user is now prepared to move into the specifics of the RAMTEK support package. For more detailed information on C, UNIX, and the use of system terminals, refer to the appropriate document in the laboratory.

IV. PROGRAMMATIC CONTROL OF THE RAMTEK

A. Opening the Device

There is one library subroutine that MUST be called in every program that applies to the RAMTEK, and it must also be the first such subroutine called. This routine, appropriately named "ramtek", opens the device and establishes user access to it. This will also clear the screen (and the refresh memory) of any prior display. The call may appear as follows :

```
ramtek();
```

As mentioned here, this must be the first RAMTEK subroutine call in the user's program. This routine will return a value of negative one (-1) if unable to open the device, otherwise it returns a zero value. The return of a negative number to indicate an error condition is common practice in this system and its operating environment. The user should consult Appendix B to find the proper diagnostic information for each routine in the software support package.

B. Virtual Screen Dimensions

In order to develop functional programs for application to the RAMTEK, the user must be aware of the concept of the "virtual" screen, which means the user's view of the screen will differ from the real hardware configuration. The display screen may be thought of as being made up of 240 horizontal lines, each consisting of 640 elements. (Each line/element combination is called a "pixel".) In reality, this device has 512 raster lines. Due to certain interfacing restrictions it has been modified to allow only 480 of these lines to be visible. The other 32 lines exist off the bottom of the screen and are of no concern to the user except when the "scroll" routine is employed. (see Special Applications Routines", Chap XVI). All other display routines in the support package map the user's addresses onto the 480 visible lines. Since this GX-100A accesses these visible lines in pairs rather than individually, the result is 240 addressable lines. Thus the initial view of the screen could be that of a 640 by 240 grid with the coordinates (0,0) in the upper left corner and the coordinates (640,240) in the lower right corner. This is NOT the view that is used when operating through the RAMTEK support package.

The software support routines have been designed to transform the screen into the more conventional "Cartesian" grid pattern that has its minimum values in the lower left corner and its maximum values in the upper right corner.

Thus the user views a virtual screen as described by an ordinary Cartesian grid system.

It should be noted here that the initializing "ramtek" routine dimensions the virtual screen to 100 X 100, with the lower left corner representing the point (0.0,0.0) and the upper right corner (100.0,100.0).

By utilizing the routine called "screen" the user may dimension the virtual display screen to any desired size, as long as the coordinates conform to the conventional pattern. The call is of the form :

```
screen(xmin,ymin,xmax,ymax);
```

where the parameters are real numbers that represent the minimum and maximum ranges for the conventional Cartesian coordinates. After dimensioning the screen to the desired size, the user may rely upon the software support package to convert any set of coordinates that are valid for that grid system into the proper real screen location. (see "conve" and "convl" in Chapter XVI, Special Applications Routines)

C. Virtual Screen Addressing

The user has now defined the address space for the virtual screen, and may choose the point from which the display should begin. This involves the location of the Current Operating Position (COP). The COP is initially located at the lower left corner of the screen. This is the case after every call to the "screen" routine. A new location for the

COP may be established by using the "strtxy" routine and passing as parameters the virtual x,y coordinates of the desired starting point. This call should be of the form :

```
strtxy(x,y);
```

where x and y are real numbers. The COP is modified by most of the actual display routines, as explained in subsequent chapters of this manual. Therefore the user must confirm its location prior to attempting to display another image.

There are three modes of addressing the RAMTEK screen : absolute, relative, and indexed.

1. Absolute addressing is the normal or default mode. The starting point (COP) for a display is selected by use of the "strtxy" routine. The ending COP depends on the current control mode (Chap VII - XIV), and the type of display that is generated.
2. Relative addressing is only applicable to the graphics control modes. Its only use is through the routine called "pointr". The real numbers passed as parameters in "pointr" are summed with the x and y values of the last COP to derive the next COP. This allows the user to generate such displays as a sequence of vectors drawn relative to each other, without computing the actual screen coordinates of all the end points. As an example, if the COP was located at (10.0,10.0) and the user issued the command

   ```
   pointr(5.0,-5.0);
   ```

   the new COP would be located at (15.0,5.0). If the

user was operating in the Graphic Vector mode, the above call would have drawn a vector from virtual screen point (10.0,10.0) to point (15.0,5.0).

3. Indexed addressing is turned on and off by use of the "index" routine. A call of the form

```
index(i,x,y);
```

where i has the integer value 1, would initiate indexed addressing. All subsequent instructions would be interpreted with respect to the x and y parameter values in that call. This address mapping is done by summing any subsequent screen address with these values to determine a new COP. For example, if the following subroutine calls were issued

```
index(1,2.0,2.0);
strtxy(3.0,3.0);
```

the COP would be located at the virtual screen coordinates (5.0,5.0). This addressing scheme can be employed in any of the control modes. Indexed addressing is terminated by issuing another index(i,x,y) call, with the parameter i having a value of zero. In this case, the x and y parameters are ignored.

D. Control Modes and Flags

Programmatic control of the GX-100A display device involves selecting one of eight control modes. Each of these modes is affected by certain control flags. A separate chapter in this manual is devoted to each of the control

modes. A description of the five control flags is presented here.

IX - Indexed Addressing

The effect of indexed addressing is explained above in Virtual Screen Addressing, along with a discussion of its controlling routine called "index".

BK - Reverse Background

When set, this causes a "reverse polarity" in the color selection process. It effectively reverses or inverts the current color table. The areas of the screen that were being shown in color entry 0 (normal background color) will be reverted to color entry 15, and vice-versa. (see Chap VI, Color Displays) This flag is turned on and off by the "bkrnd" routine. The flag would be turned ON if the parameter b was given the value 1 in the following call :

bkrnd(b);

If b was given a zero value the flag would be turned OFF.

AW - Additive Write

When set, the generated image is combined with the image already being projected at a particular location. This may be used to combine characters into special characters, create overlay type displays, or display some character close to another image without destroying a portion of that image with the seven by twelve pixel matrix used in display-

ing alphanumeric characters. (see Chap. VII, Alphanumeric Mode) The control of this flag is accomplished by the "writon" routine. As before, if the single parameter is a 1 the flag is turned on; if it is 0 the flag is turned off. The call should appear as follows :

writon(w);

where w represents either 0 or 1. (See Appendix B for error diagnostics)

DW - Double Width

When set, the image being displayed is doubled in width from its normal size; ie, where one element was being "painted" two consecutive elements on that line are displayed in the selected color. The desired character size (normal or double width) may be selected by using the "size" procedure as follows :

size(s);

where s is either 1 or 2, representing normal or double wide respectively. (See Appendix B for error diagnostics)

FP - Fixed Point

When set, this causes the (next) established COP to become the common starting point for all subsequent end points issued in graphic vector and plot generation. (see the Chapters on graphics modes) The fixed point flag is turned on and off by the routine named "fixpt". This single-parameter function accepts either a zero (to turn off the

flag) or a one (to turn on the flag). The call would be of the form :

```
fixot(f);
```

where f is either 0 or 1. (See Appendix B for error diagnostics)

The association of these flags with the different modes is shown below in a listing that indicates which flags are effective in each mode. Note the number assigned to each mode, as these numbers are used in implementing these modes of operation on the RAMTEK.

Mode		Control Flags				
No.	Name	Ix	Bk	AW	DW	FP
0	Alphanumeric	YES	YES	YES	YES	NO
1	Transverse Data	YES	YES	YES	YES	NO
2	Raster Data	YES	YES	YES	YES	NO
3	Complex Data	YES	YES	NO	YES	NO
4	Graphic Vector	YES	YES	NO	NO	YES
5	Graphic Plot	YES	YES	NO	NO	YES
6	Graphic Cartesian	YES	YES	NO	NO	NO
7	Graphic Element	YES	YES	NO	NO	NO

The control modes are selected by use of the "setmode" routine. This routine requires two parameters, the first being the number of the desired mode as indicated above. The second parameter must be either 0 or 1. If zero is passed all control flags are turned off. If one is passed the control flags are left in the state they were in prior to this call. The call is of the form :

```
setmode(a,b);
```

where a is an integer value 0 - 7 and b is either 0 or 1. (See Appendix B for error diagnostics)

V. VECTOR GENERAL DATA TABLET INTERFACE

The Vector General data tablet is an electronic device which allows the user to sample the coordinates of its tablet-like surface (approx 17" by 17"). The tablet is located in the Laboratory and is hosted by the "A" side PDP-11/50 computer.

This tablet is interfaced to the RAMTEK software support routines to provide the user with a valuable interactive tool. To effectively utilize the data tablet, the user must become familiar with it's operation. The following steps will facilitate the familiarization process :

(1) Locate the data tablet (for assistance see the staff).

(2) Note the ON-OFF switch located on the box connected to the tablet surface. To use the tablet the switch must be ON; when not in use the switch should be OFF.

(3) Locate the stylus (a pen-like device connected to the controller by a wire). On the stylus there is a plunger tip that may be depressed. The depression of this tip is significant in certain applications.

(4) Also notice, on the tablet edge, that there are two lights. One light indicates that the

stylus is near the tablet surface and the other indicates that the plunger is depressed.

(5) Turn on the tablet and witness the functions of the stylus and the reactions of the lights.

The data tablet interface allows the user to interactively control the execution sequence of his program. To effectively utilize this tool, the user should be aware of what the software support routines, designed for the tablet, can do for him. There are seven software support routines of interest to the user. An interactive tutorial program is provided for the user's convenience. This program exercises each of the software support routines to give the user a feel for what they can do. To use the tutorial the user types(at one of the "A" side terminals) :

```
LOGIN: cs2000 c/r
PASSWORD: student c/r
cd ramtek c/r
vgtab.x c/r
```

This sequence of steps is for the user who is not already logged in to UNIX. If already logged in, the user need only change to the "ramtek" directory, then execute the tutorial program. This would be accomplished by typing :

```
cd /usr/cs2000/ramtek  c/r
vgtab.x  c/r
```

In either case the result will be that the tutorial program will be activated. It exercises each of the func-

tional routines in the support package. Users are encouraged to "walk through" this session prior to attempting to engage the separate tablet routines. At the completion of the session the UNIX prompt (%) will appear on the terminal.

For reference purposes and for the user who only needs to be refreshed on the functions of the tablet support routines a brief description of each is provided. The following is a synopsis of the routines included in the package :

(1)tabinit() : Opens the tablet and sets the maximum number of switches to 25.

(2)tabdim(txmax,txmin,tymax,tymin) : Allows the user to define his own view of the tablet by stating the maximum and minimum values of his coordinate system.

(3)getxy(n) : Returns the x and y co-ordinates of a point on the tablet, as defined by the location of the stylus. The argument "n" allows the user to select whether or not the tip need be depressed (n=1 pen must be depressed, n=0 pen not depressed).

(4)sixpak() : Defines and allows selection from a standard set of six logical switches on the tablet.(see template 1 in Lab)

(5)buildsw(sw,xhi,xlo,yhi,ylo) : Allows the user to build his own logical switch at a location defined by the users input and assigns the users

switch number (sw) to it.

(6)getsw() : Returns the number of the valid logical switch in which the stylus has been decressed or a -1 if invalid.

(7)tabsw(ns) : Allows the user to define logical switches via the tablet itself by picking the low and high points of each switch with the stylus. The number of switches desired is indicated by the parameter "ns".

For more specific information on these support routines see Appendix B.

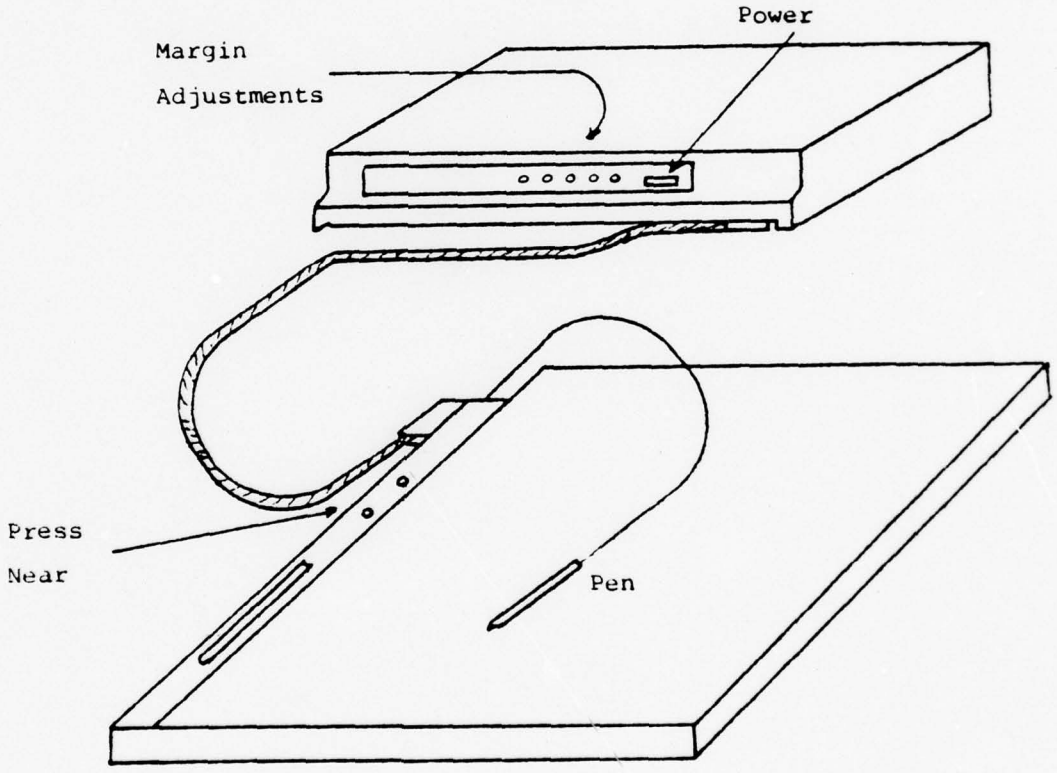


FIGURE 2

VI. COLOR DISPLAYS

The RAMTEK hardware implements color by combining red, blue and green in varying intensities to produce the desired color. The varying intensitities are stored in the video look-up table (VLT) for access by the hardware. The GX-100A can generate 16 levels of intensity for each of the three basic colors, which results in 4096 possible different combinations. The possibility of 4096 different colors does not mean that the user has all 4096 available at once! The RAMTEK software allows for 18 color tables of 16 color entries each. This means that the user has access to 288 colors. The GX-100A can access only one table, with 16 distinct colors, at any given time.

For clarity the hardware video look-up table will be referred to as the color look-up table. The hardware-accessible color look-up table can store sixteen (16) separate colors. This constraint is one that the user should be aware of, but not concerned with, as the software support routines allow the user to overcome this restriction quite easily, through manipulation of the tables.

The software that provides the color support for the RAMTEK allows the user to :

Utilize predefined colors.

Define his own colors.

Select color sets.

Select specific colors within a set.

Interactively modify colors.

Save color tables.

Restore color tables.

Text and/or graphics may be displayed in any defined color. The software support allows the user to define virtually any color that he desires.

A. BASIC USER GUIDELINES

The RAMTEK software preloads a set of color tables for the user. These tables provide for the color needs of most beginning to intermediate users.

The RAMTEK support software provides the user with a set of tools that allow him to select the color to be used for display. The user must cause the appropriate color look-up table to be initialized and then he may select from colors within it. The initialization of a particular color table is accomplished by invoking a support function "colort(n)". When "colort(n)" is called the following actions take place:

1. Color table n is loaded.
2. All display images on the screen are refreshed with the new color set.
3. The background color of the screen is set to entry zero of the new table.

Once the user has loaded a particular color table, he has sixteen colors available for display. To then select a color from the table to be used for display, the system support function "color(n)" is utilized. All images displayed subsequently will be displayed in color "n" of the current color table.

Of the eighteen tables established by the software, nine are filled with predefined colors and nine are blank. Four of these tables are protected from user modification (0-3). These protected tables may be utilized but not modified. The nine predefined tables provide a sufficiently large color selection for most users; however, there are a total of eighteen (18) color tables should the user require more flexibility. The nine blank tables must be defined or filled by the user if he desires to use them (see the explanation in Advanced Color Methods). The colors present in the nine predefined color tables are listed below:

COLOR TABLE	COLOR
0	shades of grey
1	shades of blue
2	shades of green
3	shades of red
4	mixed
5	mixed
6	mixed
7	mixed
8	shades of yellow

The RAMTEK user software allows the viewing of any color table in the system. For the user to look at a particular color table he need only issue the appropriate system call. Two different types of display are available, an expanded and a brief display. The brief display may be selected by the system function "dspyctbl(n)"; the expanded by "disp(n)".

The brief display causes color table "n" to be displayed as a column of boxes in the individual colors of color table "n". The expanded display gives the user not only a column of boxes, but lines and alphanumerics in each color, as well as the octal code that achieves that color combination.

The best way for the user to become familiar with the concepts and support functions introduced is by using them. Three executable programs have been provided for the users convenience.

1. PROGRAM 1

This program is designed to demonstrate the brief color table display, loading of a color table, selecting a particular color entry from a color table, and the result of a particular color selection. The program will first display color table seven (7) in the brief format, then delay for six (6) seconds and display a box in color four of color table seven. The user may type the program in himself or use the cs2000 file where it exists in both the source

form and an executable form.

a. To use the cs2000 file type :

```
LOGIN: cs2000 c/r
PASSWORD: student c/r
cd ramtek c/r
```

b.To list the program:

```
list -c p1.c c/r
```

c.To execute the program:

```
p1.x c/r
```

a. For the user who desires to input, compile, and execute the program himself a listing of the source code is provided.

```
main()
   {ramtek(); //initialize RAMTEK
   erase(); //erase the screen
   dspyctbl(7); //display color table 7
   colort(7); //select color table 7
   color(4); //select entry 4
   sleep(6); //delay 6 seconds
   block(50.0,50.0,75.0,25.0); //draw block
   }
```

The routine "block" will be explained in a later chapter; it should not concern the user at this stage.

2. PROGRAM 2

This example is designed to introduce the user to the expanded format for displaying a color table and the ef-

fect on the screen image of loading a new color table. Particularly the example displays color table six (6) in the expanded format, delays six (6) seconds and then loads color table five (5). The user should note that when the new color table is loaded it is effectively displayed, but the text still indicates that it is color table six (6). This highlights the fact that all of the information displayed changes color but the information itself remains unchanged.

a. To use cs2000:

LOGIN: cs2000 c/r

PASSWORD: student c/r

cd ramtek c/r

b.To list the program:

list -c p2.c

c.To execute the program:

p2.x c/r

d. To input the program the following listing is provided :

```
main()
   {
   ramtek(); //initialize
   erase(); //erase screen
   color(6); //load color table
   disp(6); //display color table 6
   sleep(6); //delay for 6 seconds
   color(7); //load new color table
   }
```

3. PROGRAM 3

This example is designed to emphasize the effect of the display color selection. Color table seven(7) will be loaded then a box displayed in color 3 of that table. A short delay will occur, then another box will be displayed in color four (4) of that table. The dspyctbl option is used to allow the user to verify the results.

a.To use cs2000:

LOGIN: cs2000 c/r

PASSWORD: student c/r

cd ramtek c/r

b.To list the program:

list -c p3.c c/r

c.To execute the program:

p3.x c/r

d. The program listing follows:

```
main()
   {ramtek();  //initialize ramtek
   colort(7); //select color table 7
   dspyctbl(7); //display color table 7
   color(3); //select color 3
   /*draw box in color 3 */
   block(50.0,75.0,95.0,5.0);
   sleep(6); //delay 6 sec
   color(4); //select color 4
   block(65.0,60.0,80.0,20.0); //draw box color 4
   }
```

The user should now be capable of writing programs that utilize basic color manipulation techniques. There are more complex and sophisticated options that are covered in the next section.

B. ADVANCED COLOR METHODS

The basics for color manipulation require little effort on the part of the user. Since the more advanced user may need to solve a more complex problem, the RAMTEK software has been designed to assist the user and simplify these sometimes complicated tasks.

The higher level user-oriented support routines are primarily to allow the user to modify existing color tables, build new color tables, save a group of color tables, and restore to an active status old color tables.

1. MODIFYING COLOR TABLES

The modification of the 18 color tables may be accomplished by utilizing the data tablet or the RAMTEK keyboard. Each of these techniques will be discussed so that the user may intelligently select the appropriate one.

a. DATA TABLET METHODS

The system software that supports data tablet color manipulation is compact and fast; this will be an advantage to the user who requires on-line color table manipu-

lation . In order to execute this support package the user issues a call of the form : tabcolor();. The system will then prompt for the required inputs to alter and/or build color tables.

The user-oriented routine "tabcolor()" offers one of two options : building an entire table or changing an existing table. The tablet is viewed in this application as per template 2 in the Lab . Once in execution the program is tutorial in nature and will prompt the user for required inputs. A sample session is provided to familiarize the user with color manipulation via the data tablet.

(1) PROGRAM 4

This exercise is designed to familiarize the user with the techniques used by the data tablet color manipulation function.

(1)To use the cs2000 file type :

LOGIN: CS2000 c/r

PASSWORD: student

cd ramtek c/r

(2)To list the source code:

list -c p4.c c/r

(3)To execute the program:

p4.x c/r

(4)For the user who wishes to input the source code himself a listing is provided. This program initializes the RAMTEK, erases the screen and then calls into

action the data tablet color manipulation package .

```
main()
    {ramtek(); //initialize
    tabinit(); //initialize tablet
    erase(); //erase screen
    tabcolor(); //call V.G. tablet routine
    erase();
    }
```

b. KEYBOARD METHODS

The purpose of this interactive program is to allow the user to see and modify the color look-up tables while seated at the RAMTEK keyboard . This extensive software package is very large and should not be used indiscriminately. It is highly recommended that it be used only to build a set of color tables, then save these tables to be loaded into the users production program at a later time (see saving color tables).

For a complete explanation of this option see the description of inter() in appendix B. If the user is interested in utilizing this option he should execute the following program to get an appreciation for it's capabilities.

(1) PROGRAM 5

This example program initializes the RAMTEK and calls the interactive color manipulation routine.

(a) To use the cs2000 file type :

```
LOGIN: cs2000 c/r
PASSWORD: student c/r
cd ramtek c/r
```

(b)To list the program:

```
list -c p5.c c/r
```

(c)To execute the program:

```
p5.x c/r
```

(d) Program listing:

```
main()
   {ramtek(); //initialize
   erase(); //erase the screen
   inter(); //call color manip. routine
   }
```

The utilization of inter() vice tabcolor() will result in the user program being 6948 bytes larger. The size of programs being a consideration in software design implies that inter() should be used selectively. There are specific applications that require it's use and it will perform well in these applications.

c. MANUAL COLOR LOADING

The system provides for the user the capability to programmatically load a particular color table as an array of entries. This process is more cumbersome and less flexible than the other two color manipulation techniques. The programmatic loading of a color table by the user forces

him to preselect colors and manually provide the mechanism for loading , but in small uncomplicated applications affords the user a feasible solution to his immediate problem.

The programmatic loading of a color table is a two step process. The first step in the process is to construct an array in his program that defines the selected color intensities. This definition of colors is accomplished by first declaring an array of 16 elements, since this is the number of elements in a color table . The array must then be given the appropriate color intensities by a system software routine call to triple() (see Appendix B, triple(), for specifics). The triple routine returns a coded value that represents the appropriate color mix. Once the array has the appropriate coded value it is then programmatically loaded into the master array of color tables by calling the system routine "clrtbl(t,a)" (see clrtbl, Appendix B), where "t" is the number of the color table to be replaced and "a" is the name of the color table holding the coded values.

This procedure may seem complicated and is probably better illustrated by an example. The user is encouraged to execute the example program to experience the process.

(1) PROGRAM 6

This program takes a certain predefined set of color mixes and builds color table 9. Once the color

table has been built, it is then displayed for the user to view. In addition to displaying the color table, a set of intersecting lines are drawn that intersect at virtual screen co-ordinates (5.0,5.0). The lines are drawn in color 4 of color table 9, the one just built.

(a) To use the cs2000 file type:

LOGIN: cs2000 c/r

PASSWORD: student c/r

cd ramtek c/r

(b)To list the program:

list -c p6.c c/r

(c)To execute the program:

p6.x c/r

(d) Program listing:

```
int a[16]; //declare array
main()
   {ramtek();  // initialize ramtek
   /* set color intensities into array */
   a[0]=triple(15,15,15); //white
   a[1]=triple(15,0,0);
   a[2]=triple(0,15,0);
   a[3]=triple(0,0,15);
   a[4]=triple(3,14,2);
   a[5]=triple(8,15,0);
   a[6]=triple(8,0,0);
   a[7]=triple(0,8,0);
   a[8]=triple(0,0,8);
   a[9]=triple(8,8,0);
   a[10]=triple(0,8,8);
   a[11]=triple(8,0,8);
   a[12]=triple(10,10,10);
   a[13]=triple(4,9,1);
   a[14]=triple(5,0,5);
   a[15]=triple(9,1,15);
   clrtbl(9,a); //load master array
   dspyctbl(9); //display color table 9
   color(9); color(4); //select color
   screen(0.0,0.0,10.0,10.0); //dimension screen
   axis(5.0,5.0); //draw axis
   }
```

The user, having now executed the suggested sample programs, should have a good appreciation for how to create and manipulate color tables. If more in-depth information is required, the user is encouraged to research References 1 and 3 for the specific hardware and support design.

2. SAVING COLOR TABLES

As mentioned earlier, the system color tables are initialized each time a user begins a session. These colors may be altered by the methods described above. The ability to save the current state of all color tables for use at a later time not only enhances the users capabilities but is an absolute must for production programs that require certain color sets. The user will certainly appreciate the problem if he considers the case where he has defined a large number of specific color sets for distinct applications using "tabcolor()" and/or "inter()", and has no way to save them. Without the color saving option, in order for the user to continue to use a specific set of defined colors in repeated runs of the same program, he must reconstruct them each time.

In utilizing the save color option the user must be aware of two routines:

```
savclr( );
resclr( );
```

The routine "savclr()" saves the entire color array in a

file called "savcol". The routine "resclr()" restores the file "savcol" into the active master color array so that the user may reinstate a specific set of color tables.

VII. ALPHANUMERIC MODE

A. INTRODUCTION

This mode is used for displaying alphanumeric (ASCII) characters on the RAMTEK screen. The basis of this mode is the transformation of the individual ASCII characters into a dot matrix by means of a character font. The font defines this matrix to be five screen elements by seven screen lines in dimension. This character matrix is actually displayed within a seven element by twelve line matrix, to provide separation of characters. This is illustrated in Appendix B of the RAMTEK GX-100 Programming Manual [1].

The ASCII characters are displayed starting at the Current Operating Position (COP) from left to right on the same line. If more than 91 characters are written on one line, they will "wrap around" and over write the beginning characters on that same line. (If the optional "double-width" size characters are used, the maximum number per line is 45.)

The applicable dots (pixels) in the character matrix (as defined by the character font) will appear in the current selected color, while the remainder of the matrix will be "filled in" with the current background color.

B. BASIC USER GUIDELINES

As mentioned earlier in this manual, the software support package for the RAMTEK is contained within the UNIX file system, in the form of a library of subroutines written in the 'C' programming language. These routines are accessed (via any of the terminals on the "A" system) by compiling a user-created 'C' program containing references to functions that reside in that library. The special shell command for compiling a program that calls routines from the RAMTEK library is called "ramtek", and needs only be passed the name of the file to be compiled.

Such a command would be of the form :

```
ramtek  filename.c
```

(note that the file must be of type ".c")

The first graphics routine that must be called from within the user's program is, in every case, the one that is also named "ramtek". (This should not be confused with the shell command above) This routine opens the device, erases the screen, and establishes user access to it. The call should appear as follows :

```
ramtek( );
```

To place the RAMTEK device in the Alphanumeric Mode, the user may utilize the "setmode" routine, with a call of the form :

```
setmode(0,0);
```

(Note : the Alphanumeric mode has been defined to be mode

"0"; the second parameter of "0" turns off any previously-selected control flags; a "1" in this position would preserve these flags.)

Although some display routines will automatically set the proper mode, it is good practice to include this "set-mode" call in user programs.

The user is now ready to select the area on the screen in which he desires alphanumeric data to be displayed. A quick review of the initializing "ramtek( )" routine would reveal that the virtual screen has been dimensioned to 100 X 100, and the Current Operating Position (COP) is located in the lower left corner (coordinates 0,0). If these dimensions are not favorable, the user may select his own with a call to the "screen" routine of the form :

```
screen(xmin,ymin,xmax,ymax);
```

where the parameters are real numbers.

In any case, before a call to one of the display routines, the user should confirm the location of the COP (the point at which the display will begin). The COP may be re-established by a call to

```
strtxy(x,y);
```

where x and y are real numbers for the desired virtual screen coordinates.

For the simple application of displaying a single character on the screen, the procedure called "lttr(ch,size)" is

used. The call is of the form :

```
lttr('x',size);
```

where the character desired to be output is enclosed in single quotes or its ASCII code is given without quotes. The size is passed as either 1 (normal size) or 2 (double width). All control bits or flags remain as previously set. The character is displayed at the existing CUP. No line feed occurs, but the x-value is incremented by 7 real screen elements as a result of the output. (The CUP moves to the right 7 pixels.)

The most straight-forward approach to displaying a set of ASCII characters, such as a block of text, is with a call to the routine "textc", which requires 5 parameters, in order, as follows :

1. The address of an array of pointers to the text; The pointer array should have been declared as pointers to an array of variable-length character strings. The actual array of character strings should be initialized following the array declaration by enclosing each string in double quotes and separating them by commas. The last element should be a zero, without quotes. (see example below)
2. The number of the desired color table (an integer value : 4 - 17)
3. The number of the desired entry from that color table (integer value : 1 - 15)

4. An integer value (1 or 2) to denote desired character size. If this parameter has value 1 the characters are displayed in normal size; if it has value 2, the characters are displayed in double width size.

5. An integer value (0 or 1) to disable/enable the additive write capability. If this parameter has value 1, the corresponding text will be written "on top of" (ie, in addition to -- not in replacement of) the existing screen image. If the value is 0, the desired text will replace the contents of the refresh memory that exists at the specified locations.

1. PROGRAM 7

The execution of this program will display the declared block of text in double-width character size, starting at the virtual screen coordinates (25.0,75.0). The characters will be written in the color specified by color entry 5 of color table 7, and will replace any data previously written there.

This program would normally be compiled using the 'C' compiler in conjunction with the system library of RAMTEK subroutines, via the following command : "ramtek filename.c", where filename is the name of the user file containing the program. However, this example is available in executable form in the "cs2000 - ramtek" directory on the "A" system. (see Chapter II, Operating Instructions)

a. To use the cs2000 directory :

LOGIN: cs2000 c/r

PASSWORD: student c/r

cd ramtek c/r

b. To list the program :

list -c p7.c c/r

c. To execute the program :

p7.x c/r

d. The listing follows :

```
char *txt1[ ]        /* declare and initialize the  */
  {"What you see ",   /* pointer array and text   */
   "    is what you get ",
   "         when you call ....",
   "                          ",
   "                  TEXTO   ",
   0
  };
int ctab1  7;              /* color table 7 */
int col1  5;               /* color entry 5 */
int sz1  2;                /* double width characters */
int wo1  0;                /* no write-over */
main( )
  {ramtek( );                /* open the device  */
   strtxy(25.0,75.0);        /* set starting point */
   texto(txt1,ctab1,col1,sz1,wo1);
  }
```

C. ADVANCED METHODS

After a review of "texto" in Appendix B, the user should realize that its advantages lie in simplicity and generality. In order to perform more complicated alphanumeric display tasks, the user may wish to proceed to a more detailed level of operations.

Further analysis of "texto" would reveal that it actually makes a series of calls to lower-level routines. These

routines may, of course, be employed individually.

As before, the first step in displaying characters is to insure that the system is in the Alphanumeric mode. This is accomplished by calling the routine

```
setmode(a,b);
```

passing a zero value to parameter a. Parameter b must receive either a 0 to turn off all control flags, or a 1 to leave control flags as presently set.

Again, if the user wishes to change the dimensions of the virtual screen, he may call the procedure

```
screen(xmin,ymin,xmax,ymax);
```

and pass the minimum and maximum values of the desired coordinate system. This procedure leaves the COP at the lower left corner of the screen.

In order to select a desired working color table, the user may call

```
colort(t);
```

passing the number of the table to the parameter t. Then the appropriate display color entry is selected by a call to

```
color(e);
```

where e is the entry number from the selected table.

To establish a new COP for a particular display, the procedure

```
strtxy(x,y);
```

is utilized with the starting X and Y coordinates (as shown

before) being passed as real parameters.

The size of the characters displayed may be ordered by a call to

```
size(s);
```

with s being either 1 or 2. If s is passed a value of 1, the characters will be displayed in normal size; if a value of 2 is passed, they will be displayed in double width size.

The selection (or deselection) of the 'additive write' feature (ie, the intention to 'add to' rather than replace the present screen image) is relayed to the system by a call to

```
writon(w);
```

with w receiving a 1 if write-over is desired or a 0 if not.

The actual next-level display routine that is called upon by "texto" is "strout(s)", where s is a pointer to a single string of less than 100 characters. For example, if the user had the following declaration in his program :

```
char string[ ] {"This is a string."};
```

then a call of the form

```
strout(string);
```

would display that sentence on the screen. It should be noted here that after a call to "strout", an automatic line feed/carriage return is performed, leaving the COP on the next available line and at the same element position as the previous COP.

To carry this discussion one step lower in the structure, we point out that "strout" actually counts the number of bytes in the character string and passes this number, along with the address of the string, to the routine "data(name,length)". This routine is also available to the user if he is certain of the number of bytes in his character string and does not attempt to pass a string that is too long. As an example, with the same "char string" declaration as above, the user could write

```
data(string,17);
```

to accomplish the same task as "strout(string);". This routine also leaves the COP on the next line and same element as the previous COP.

After some error checking and setting control bits, "data" goes one step lower by making a system call to the "dump" routine, which transmits the raw data (according to the byte count) to the RAMTEK instruction buffer for interpretation and display.

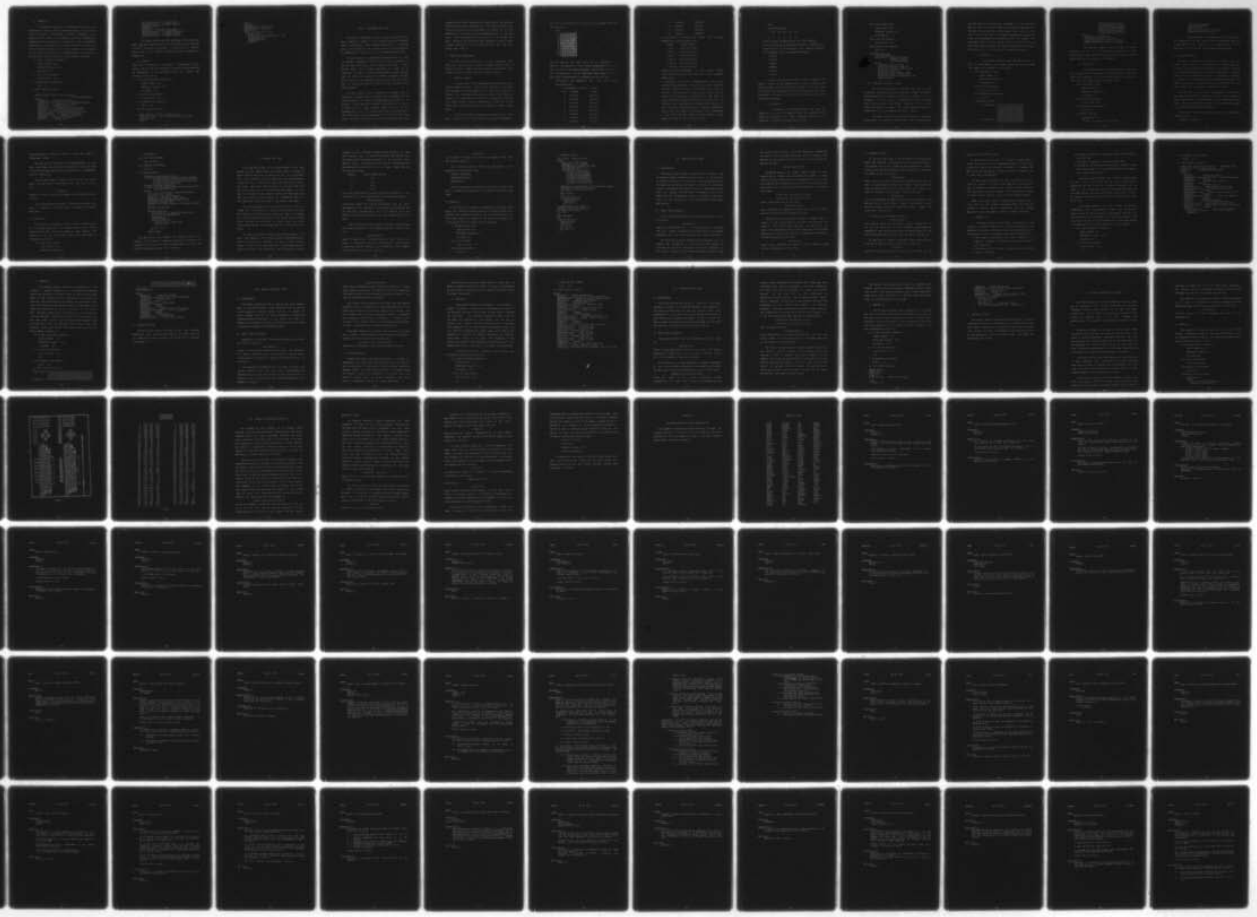
By similar inspection, we find that "size" actually utilizes a lower-level routine called "dblwid" to set the appropriate control bits and "dump" this data to the RAMTEK buffer.

The "dump" routine exists at about the lowest level of the user interface. To directly utilize this routine, the user is advised to gain a greater familiarity with the "RAMTEK GX-100 Programming Manual" [1].

AD-A042 997 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 9/2
THE DEVELOPMENT OF A USER ORIENTED INTERFACE FOR A COMPUTER DRI--ETC(U)
JUN 77 C E MCNEIL, D P HOUSTON

UNCLASSIFIED NL

2 OF 3
AD A042997



1. PROGRAM 8

This sample program will demonstrate the use of the alphanumeric display routines mentioned above. It is available for viewing in the "cs2000 - ramtek" directory. Its execution will result in the declared string of alphanumeric characters being displayed, in tandem, at three successive locations on the screen, and in three different colors. Note that the second display call in each instance is not preceded by a call to "strtxy". This demonstrates the location of the COP after calling either of these display routines.

a. To use the cs2000 directory :

LOGIN: cs2000 c/r

PASSWORD: student c/r

cd ramtek c/r

b. To list the program :

list -c p8.c c/r

c. To execute the program :

p8.x c/r

d. The listing follows :

```
char string[] {"This is a string."};
                                    //string declaration
main()
  {ramtek();     // open the device
   screen(0.0,0.0,10.0,10.0);  //dimension screen
   colort(7);    // select color table 7
   color(1);     // select color 1
   setmode(0,0);  // select alphanumeric mode
   size(2);      // select double-width size
   strtxy(0.5,9.0);  //set starting point (COP)
   strout(string);   // display the string
   data(string,17);  // display same string
   sleep(2);     // pause 2 seconds
   color(4);     // change color
```

```
        strtxy(3.0,6.0);  // change COP
        strout(string);  // repeat display
        data(string,17);  //  "      "
        sleep(2);
        color(7);  // change color again
        strtxy(5.5,3.0);  // change COP again
        data(string,17);  // repeat display
        strout(string);  // repeat display
      }
```

This same display could be achieved by the following more concise program, better utilizing some of the features of the 'C' programming language. It also exists in executable form in the "cs2000 - ramtek" directory discussed in Chapter II.

2. PROGRAM 9

Note the initialization of variables in declaration statements, and the use of assignment statements and expressions as parameters in the procedure calls for "color" and "strtxy". Once again :

a. To use the cs2000 directory :

    LOGIN: cs2000 c/r

    PASSWORD: student c/r

    cd ramtek c/r

b. To list the program :

    list -c p9.c c/r

c. To execute the program :

    p9.x c/r

d. The listing follows :

```
char string[] {"This is a string."};
float stx -2.0;   // initialize starting X value
float sty 9.0;   //     "          "     Y   "
main()
```

```
{int i;
 ramtek();
 screen(0.0,0.0,10.0,10.0);
 setmode(0,0);  size(2);
 colort(7);
 for(i = 0;i <= 6;i = i+3)
   {color(i+1);
    strtxy((stx = stx+2.5),(sty - i));
    strout(string);
    data(string,17);
    sleep(2);
   }
}
```

VIII. TRANSVERSE DATA MODE

This mode is designed to allow the user to define and draw irregular images in the vertical plane (8 elements wide) of the RAMTEK. These figures should be in a single color, as the facility for changing colors while processing an image does not exist while in the transverse data mode.

The user must gain a superficial understanding of how the RAMTEK processes his data in this mode to be able to use the mode effectively. The bit pattern of a set of user defined data words (bytes) describes the image to be displayed. The user will define data words or bytes; regardless of which he chooses, the RAMTEK interprets the data byte by byte (a byte consists of eight bits). The bit pattern of each byte turns ON or OFF individual pixels on the RAMTEK screen.

An octal number in the C programming language [3] is denoted by a leading zero. The octal form of numbers will be used exclusively in preparing transverse data, since it is simpler. If the user was to define the data word "x" as "x = 015022 (x is of type integer) then it's bit by bit representation for the system would be: 0001101000010010. The RAMTEK would further decompose this word into two data bytes (00011010 and 00010010). The data bytes are then pro-

cessed by the RAMTEK sequentially, beginning at the user defined starting point (see "strtxy"). The eight pixels to the right of this point are interpreted as either ON or OFF depending on the value of it's corresponding bit in the data byte. A one indicates that the pixel is to be turned ON; zero indicates the OFF state. Each consecutive byte is then written immediately below the last one until all are processed (see "data").

A. BASIC USER GUIDELINES

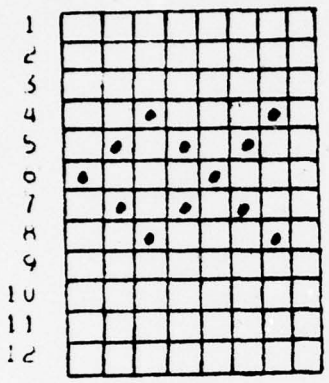
Two very likely applications for the transverse data mode are definition and display of special symbols and irregular vertical images. Software support routines are provided to allow the user to perform these actions.

1. SPECIAL SYMBOLS

A common use of the transverse data mode would be to define special symbols that are not provided in the standard character set. For example, the user may desire to display the lowercase Greek letter alpha. By defining this letter in a linear array of length six (words), and calling the system support function "symbol(sy)", where "sy" is the array name, the user may display this special symbol quite easily.

To define a special character the user should first draw a matrix (8 by 12) and develop a graphical representa-

tion of that character. For example, the lowercase Greek alpha would be:



The next step for the user would be to transform this graphic representation into a data representation that can be utilized by the support package. Transforming the graphical representation of the lowercase Greek alpha into a viable data representation is accomplished as follows:

a. Transform the graphical view into the binary representation.

(- represents a blank ; * a dot)

BYTE	GRAPHIC	BINARY
1	--------	00000000
2	--------	00000000
3	--------	00000000
4	--*----*-	00100010
5	-*-*-*--	01010100
6	*---*---	10001000
7	-*-*-*--	01010100
8	--*----*-	00100010

9	--------	00000000
10	--------	00000000
11	--------	00000000
12	--------	00000000

b. Marry pairs of bytes into words. This is done sequentially from the beginning.

BYTES	BINARY NUMBER
1 and 2	0000000000000000
3 and 4	0000000000100010
5 and 6	0101010010001000
7 and 8	0101010000100010
9 and 10	0000000000000000
11 and 12	0000000000000000

These six words, represented in their binary number form, provide the most basic data form of the lowercase Greek alpha.

c. Transform the binary representations into octal numbers for processing. This binary to octal transformation is done from right to left. Each binary word is divided into 5 groups of three binary digits, with one binary digit left over. The position of the ones in these groupings represent powers of the number two. The leftmost digit represents two squared when a one is in this location. The center digit , when one, represents two to the first power. The rightmost digit represents a one (two to the zero power). Sum the numbers in each group of three to yield one octal digit.

Ex:

0101010000100010

0 101 010 000 100 010

0 5 2 0 4 2

Thus the octal representation would be: 0052042.

a. An octal word, as previously stated, is denoted by a leading zero. The following six octal numbers represent the lowercase Greek alpha described above:

0000000

0000042

0052210

0052042

0000000

0000000

Now that the user has an octal number to define the special symbol, he may generate that symbol by calling the system support routine "symbol(sy)", where "sy" is the name of the user defined array containing the octal definition of the symbol. The user should verify that the octal code listed above is correct.

a. PROGRAM 10

This program is designed to show the user the mechanics of building a special symbol and displaying that symbol on the RAMTEK. The Greek lowercase alpha will be displayed in standard size and double width.

(1)To use cs2000 type:

LOGIN: cs2000 c/r

PASSWORD: student c/r

cd ramtek c/r

(2)To list the program:

list -c p10.c c/r

(3)To execute the program:

p10.x c/r

(4)Program Listing:

```
/*DECLARE ARRAY*/
int alpha[6] {0000000,0000042,
              0052210,0052042,
              0000000,0000000};
main()
   {ramtek(); //initialize
   strtxy(50.0,50.0); //set COP
   colort(7); //select color table
   color(2); //color 2
   symbol(alpha); //output alpha
   dblwid(1); //set double width
   strtxy(50.0,30.0); //establish COP
   symbol(alpha);
   }
```

2. IRREGULAR VERTICAL IMAGES

It is quite plausible to consider the use of the transverse data mode to define irregular images. The system software supports the design and display of columns of information via the function "tdata(tptr,by,ex,ly)", where "tptr" is the name of the array (pointer), "by" is the number of bytes to be output, "ex" is the desired starting x value, and "ly" is the desired starting y value.

The user might desire to develop a set of oversized parentheses for a special application. This is accomplished

by first defining the image as discussed in the previous section, then via "rdata" creating the display. Suppose that the user has defined his virtual screen to be 100 by 100, with (0,0) in the lower left corner. Perhaps he has use for a set of parentheses that encompass 20% of the screen or, more specifically, 48 real screen lines (.2 * 240 = 48). He must then define 48 bytes per image to accomplish this task. The definition of the data is as discussed previously.

a. PROGRAM 11

This program shows the user the definition of a set of parentheses, as discussed above, and how to create the images on the screen of the RAMTEK.

(1)To use cs2000 type:

LOGIN: cs2000 c/r

PASSWORD: student c/r

cd ramtek c/r

(2)To list the program:

list -c p11.c c/r

(3)To execute the program:

p11.x c/r

(4)Program listing:

```
int lparen[24] {0000401,0000403,0001403,
                0003006,0003014,0006014,
                0014030,0014060,0030060,
                0060140,0060300,0140300,
                0140300,0140140,0060140,
                0030060,0030030,0014030,
                0006014,0006006,0003006,
                0001403,0001401,0000401};
int rparen[24] {0100200,0100300,0140300,
                0060140,0060060,0030060,
```

```
                        0014030,0014014,0006014,
                        0003006,0003003,0001403,
                        0001403,0001406,0003006,
                        0006014,0006030,0014030,
                        0030060,0030140,0060140,
                        0140300,0140200,0100200};
main()
   {ramtek(); //initialize
   color(7); color(2); //colors
   tdata(lparen,48,40.0,60.0); //output
   tdata(rparen,48,60.0,60.0); //output
   }
```

The function "tdata" allows the user to paint portions (or all) of the screen by adjusting to the next column when screen overflow occurs. The user need not worry about going off the bottom of the screen.

b. PROGRAM 12

This program shows the reset action of the function "tdata". Data is passed to the routine that is much too long to be displayed in a single column, thus it paints to the right. The image drawn is a dotted line.

(1)To use cs2000 type:

LOGIN: cs2000 c/r

PASSWORD: student c/r

cd ramtek c/r

(2)To list the program:

list -c p12.c c/r

(3)To execute the program:

p12.x c/r

(4)Program listing:

```
int test[300] ;
main()
   {int i; ramtek(); //initialize
```

```
/*initialize array*/
for(i=0;i<300;i++)
test[i] = 01;
color(7); color(2); //colors
tdata(test,600,10.0,40.0);
}
```

The user should now possess the skills required for processing the vast majority of his transverse data applications. The next section will discuss some of the specifics of the mode for more advanced applications.

8. ADVANCED METHODS

The user who desires to execute some process that requires more flexibility than provided in the previous section will have to drop down one level in the support package to the more fundamental functions for processing transverse data. The most fundamental function in the support package, in relation to transverse data, is "data(name,bytes)". The "data" routine simply sends raw data to the TEKTRONIX from the array "name" for the specified number of "bytes".

The user must, at this level, be responsible for setting the mode (see "setmode"), establishing the current operating point (see"strtxy"), and providing his own programmatic control. The data definition process is the same as discussed in the BASIC USER GUIDELINES section.

The control flags that are applicable in the transverse data mode are:

INDEXED ADDRESSING

REVERSE BACKGROUND

ADDITIVE WRITE

DOUBLE WIDTH

The user must select all of his flags and provide the mechanism for their implementation. Complicated transverse data applications require much attention to detail for success. For more specifics the user is encouraged to peruse the RAMTEK Programming Manual [1], in the Lab.

IX. RASTER DATA MODE

The Raster data mode, except for the direction of the writing, is like the Transverse data mode. The Raster data mode writes across the screen vice down the screen. The utility of this mode is that it allows the user to efficiently draw irregular images of various shapes and sizes.

The very nature of user applications for this type of processing dictates that it be utilized at a lower level than previous data modes. The user may define and draw any image that he can conceive. The user must, however, provide all programmatic control himself, as nothing is assumed.

The applicable control flags in this mode are:

INDEXED ADDRESSING (see"index")

REVERSE BACKGROUND (see"bkgrnd")

ADDITIVE WRITE (see"writon")

DOUBLEWIDTH (see"size")

The user should recall from previous chapters that to place the RAMTEK in a particular mode of operation the "setmode" software support function is utilized. To place the RAMTEK in the Raster data mode a call of the form

```
setmode(2,y);
```

would be issued. The argument "y" determines if previous flags are to be saved. If "y" is a one (1), then all prior

flags are saved; if zero (0), then all flags are reset to the default values.

The user who is unfamiliar with the generation of the octal data codes that define an image should review the section on Transverse data. The octal data form is recommended for it's simplicity.

The software support function "data" is used to transport the raw data to the RAMTEK for display. A call of the form

data(x,y);

causes "y" bytes of data to be passed to the RAMTEK from array "x".

The following sample program is provided to assist the user in developing his expertise in the use of the Raster data mode:

A. PROGRAM 13

This program is designed to show the user a specific use of the Raster data mode. The program draws a set of intersecting lines much like a grid system or data record. The versatility of the Raster data mode is only restricted by the users imagination.

1. To use cs2000 type:

LOGIN: cs2000 c/r

PASSWORD: student c/r

```
cd ramtek c/r
```

2.To list the program:

```
list -c p13.c
```

3.To execute the program:

```
p13.x c/r
```

4. Program Listing :

```
/* Define a horizontal line */
ln[18] {0177777,0177777,0177777,0177777,0177777,
        0177777,0177777,0177777,0177777,0177777,
        0177777,0177777,0177777,0177777,0177777,
        0177777,0177777,0177777};
/*Define vertical segments*/
div[18] {0140003,03,03,03,03,03,03,03,03,03,03,03,
         03,03,03,03,03,03};
main()
   {int i,j; float sx,sy; //declare variables
   sx = 160.0; sy = 180.0;
   ramtek(); //initialize system
   colort(7); //establish color table
   setmode(2,0); //establish raster mode
   color(2); //select color
   screen(0.0,0.0,640.0,240.0); //dimension screen
   /*
   OUTSIDE LOOP CONTROL
   */
   for(j=0;j<12;j++)
      {strtxy(sx,sy); //establish start point
      sy = sy - 1; //decrement y
      data(ln,36);
      /*INSIDE LOOP */
      for(i=0;i<18;i++)
         {data(div,36) //send data
         sy = sy - 1;
         }
      sy = sy + 1;
      }
   data(ln,36);
   }
```

The user should now possess the required skills to develop his own special images as required. For further information on this mode the user is encouraged to read the RAMTEK Programmers Manual [1] in the Lab.

X. COMPLEX DATA MODE

This mode writes data to the RAMTEK screen in the same manner as the Raster data mode, except that it allows the user to define the color of each pixel as it is painted. This third dimension of color does have some overhead associated with it, in that now the user must define the color. The color definition requires four (4) bits per pixel vice one (1) bit per pixel in the other two data modes. The impact on the user is that where one data word describes sixteen (16) pixels in either Raster or Transverse data, it only describes four (4) pixels in the Complex data mode.

The obvious benefit of this mode is that the color of images may be modified on a pixel by pixel basis. The four bits required to define a color allow the user to address any one of fifteen (15) colors in the current color look-up table. The user must construct his data words now using binary coded decimal as an added step prior to converting to octal.

The color definition of each pixel must be done by the user, then passed to the software support programmatically. This color definition requires the use of 4-bit binary groups (called binary coded decimal). These groups are combined into 16-bit binary patterns, then converted (by 3-bit

groups) to octal. Suppose the user wishes to define a word that selects four (4) pixels and colors them according to the colors one,four,seven and eleven of the current color look-up table. To accomplish this task he must convert the desired color entries (1,4,7,11) to binary coded decimal form as shown below:

DECIMAL	BINARY CODED DECIMAL
1	0001
4	0100
7	0111
11	1011

The binary coded decimal numbers are then combined into one binary word. For the above example this number would be:

0001010001111011

This binary number must then be transformed into an octal representation as discussed in the chapter on Transverse data. The octal representation for this example would be 0012173. The user must perform this data transformation himself.

Once the data words are properly defined, the user need only place the RAMTEK in the Complex mode with a call of the form

setmode(3,f);

where "f" determines if previous control flags are to be saved or destroyed (f=1 save ; f=0 destroy). Having established the mode of operation the information is now passed to the RAMTEK with a call of the form

data(x,n);

which causes "n" bytes of raw data to be passed to the RAM-TEK from the array "x".

The following control flags are applicable while operating in the Complex mode:

INDEXED ADDRESSING

REVERSE BACKGROUND

DOUBLEWIDTH

The following sample program is provided to assist the user in developing expertise in the use of the Complex data mode:

A. PROGRAM 14

This program is designed to demonstrate the Complex data mode. Its execution causes the entire display screen to be painted with alternating bands of the first seven colors in color table seven. The screen is then partially erased and a triangle with similar properties is displayed.

1. To use cs2000 type:

   LOGIN: cs2000 c/r

   PASSWORD: student c/r

   cd ramtek c/r

2. To list program:

   list -c o14.c c/r

3. To execute program:

   o14.x c/r

4. Program listing

```
int l[161]; //declare array
main()
    {int i,j,k; float sx,sy;
     ramtek(); //init system
     setmode(3,0); //establisn mode
     /*INITIALIZE ARRAY 1 */
     for(i=0;i<161;i = i + 14)
        {l[i]=l[i+1]=0073567;
         l[i+2]=l[i+3]+0063146;
         l[i+4]=l[i+5]=0052525;
         l[i+6]=l[i+7]=0042104;
         l[i+8]=l[i+9]=00314o3;
         l[i+10]=l[i+11]=0021042;
         l[i+12]=l[i+13]=0010421;
         }
    screen(0.0,0.0,640.0,240.0); //dimension screen
    colort(7); //select color table
    k = 320; sx = 0.0; sy = 0.0;
    /*
    PAINT SCREEN
    */
    for(i=0;i<240;i++)
      {strtxy(sx,sy);
       data(l,k);
       sy = sy + 1;
      }
/* ERASE PORTION OF SCREEN */
terse(40.0,10.0,600.0,230.0);
sx = 60.0; sy = 20.0;
k = 100;
/*
DRAW TRIANGLE
/*
for(i=0;i<100;i++)
  {strtxy(sx,sy);
   data(l,k);
   k = k -1;
   sx = sx + 1;
   sy = sy + 1;
   }
}
```

XI. GRAPHIC VECTOR MODE

A. INTRODUCTION

The Graphic Vector Mode is used primarily to draw lines or vectors between user-defined end points. It is relatively simple to use and, by its nature, fits many applications. The user must be aware that except in vertical or horizontal lines, a noticeable quantization error or "stair-casing" effect will appear. This condition is caused by the low line resolution (240 addressable) versus high element resolution (640) of the system, and by the very nature of the raster scan device.

B. BASIC USER GUIDELINES

To put the RAMTEK device into this mode requires a call of the form

```
setmode(4,0);
```

where 4 is the defined number for this mode and 0 indicates that all control flags are to be turned OFF. If the second parameter was 1, all flags would be left as set previously.

The first two routines presented below actually set the proper mode themselves (by calling "setmode" internally); although, as previously mentioned, the use of "setmode" in the user's program is considered standard practice. These

are "vector" and "plotln". The first accepts as parameters two sets of virtual screen coordinate pairs and draws a line between the two points that they define. It's call would appear as follows :

```
vector(x1,y1,x2,y2);
```

The second accepts two equal length arrays of real numbers and an integer value to denote that length. It then plots the points described by the corresponding x,y values and connects each successive point with a line from the previous one. Thus if two arrays were declared and initialized as follows :

```
float px[n] {x1,x2,x3,....,xn};
float py[n] {y1,y2,y3,....,yn};
```

then a subroutine call of the form

```
plotln(px,py,n);
```

would correlate the respective values from the two arrays and plot the points with connecting lines.

Another routine which actually sets the proper mode by itself is one which might be utilized in conjunction with "plotln". This is the "axis" routine. As implied by its name, this subroutine draws a coordinate axis on the screen according to the two real numbers it accepts as parameters. It would be called as follows :

```
axis(x,y);
```

where x and y represent the desired virtual screen values for the intersection of the axes.

C. ADVANCED METHODS

At the next level down in the structure of the software, there are several routines which are highly functional in this system. The first of these is the procedure for establishing a new Current Operating Position (COP). To reiterate its usage, a call of the form

```
strtxy(x,y);
```

where x and y are real numbers, would establish the COP at that virtual screen location. From there, a vector could be drawn to another point by utilizing either the "point" or "pointr" routines in this mode. A call of the form

```
point(x,y);
```

with the parameters representing the x and y values for the desired point, would draw a vector from the COP to that point and establish the COP at the new point. (unless the fixed point flag was ON)

In similar circumstances, a call to

```
pointr(dx,dy);
```

would draw a vector from the COP to a point "relative" to the COP as determined by the "change" values passed as parameters. Here again the COP would be established at the newly calculated point unless the fixed point flag was ON.

As mentioned in "Control Modes and Flags" (Chap IV), the fixed point flag is turned ON and OFF by a call to

```
fixpt(f);
```

where f is given the value 1 to turn ON the flag or the

value 0 to turn OFF the flag.

If this flag is ON, a call to "strtxy" establishes a fixed point from which all subsequent vectors will be drawn by calls to "point" or "pointr". In effect, it freezes the COP until the occurrence of another "strtxy" command or until the flag is turned OFF.

The user should be aware that each time the fixed point flag is turned ON, after having been OFF, a new "strtxy" call is REQUIRED to establish the desired fixed position. The flag cannot be simply toggled ON and OFF amidst a series of "point" and/or "pointr" calls, without erratic results.

Other flags which affect the operation of subroutines in this mode are the Indexed Addressing flag, which was explained in "Virtual Screen Addressing", and the Reverse Background flag, discussed in "Control Modes and Flags".

1. PROGRAM 15

The sample program listed below is available in the "cs2000 - ramtek" directory to illustrate a variety of the described routines. Selective delays have been employed in the program to allow the viewer to notice the following :

1. the change of background color when "colort(7)" is executed (because the "ramtek" routine initializes to color table 0);
2. the efficiency of "plotln" in drawing the first three sides of the star;

3. the use of "pointr" to draw the fourth side relative to the third;

4. the use of "vector" to draw the final side;

5. the use of the fixed point method for drawing lines from the center to the extreme points of the star, with "point" and "pointr".

The user should realize that "plotln" establishes its own initial CDP at the first point and leaves the CDP located at the last point that it plots. He should also be aware of the location of the CDP before and after each of the other display routines are called. Notice that "strtxy" is used only once -- immediately after turning on the fixed point flag.

The figure displayed by this program illustrates dramatically the 'stair-casing' effect that occurs when drawing lines at various angles on the screen. The user may also notice that a horizontal line is slightly 'fatter' than a vertical one due to the afore-mentioned difference between line resolution and element resolution in this system.

a. To use the cs2000 directory :

```
LOGIN: cs2000  c/r
PASSWORD: student   c/r
cd ramtek  c/r
```

b. To list the program :

```
list -c p15.c   c/r
c/r
```

c. To execute the program :

```
p15.x   c/r
```

d. The listing follows :

```
float px[4] {-4.0,0.0,4.0,-6.5};     //declare array
float py[4] {-6.0,6.5,-6.0,2.0};     //    "        "
int pn 4;          //declare array length
main()
  {int i;
   ramtek();          //open the device
   screen(-10.0,-10.0,10.0,10.0);  //set screen size
   sleep(2);          //pause
   color(7);          //select color table 7
   sleep(2);     //pause
   color(5);          //select color 5
   setmode(4,0);       //set graphic vector mode
   plotln(px,py,pn);     //plot the array points
   sleep(2);         //pause
   pointr(13.0,0.0);      //draw line relative to COP
   sleep(2);          //pause
   vector(6.5,2.0,-4.0,-6.0);   //draw with "vector"
   sleep(2);         //pause
   color(1);          //select color 1
   fixpt(1);      //turn ON fixed point flag
   strtxy(0.0,0.0);     //set starting point (COP)
   pointr(6.5,2.0);  //draw first line relative to COP
   sleep(2);          //pause
   for(i=0;i<4;i++)          //draw remaining points
     {point(px[i],py[i]);      //  from fixed COP
      sleep(2);
     }
  }
```

XII. GRAPHIC PLOT MODE

A. INTRODUCTION

The primary function of the graphic plot mode is to generate the display of a histogram style plot. In effect, this means to shade or color the area between a selected x axis and a line defined by user-generated points. This shading is actually accomplished by drawing a series of vertical lines adjacent to each other. The software that accomplishes this is built around a RAMTEK hardware feature in this mode which automatically increments the element address by one after each line is drawn. Thus after the starting element has been established, each successive element (proceeding left to right) is individually addressed, until the final point is reached. As each element is addressed, a vertical line is drawn from a pre-selected common line or 'x axis' to the proper line address or y value. The most obvious application is plotting a 'function' and shading the area between that 'function' and a certain x axis.

B. BASIC USER GUIDELINES

To select the graphic plot mode the user should make a subroutine call of the form

```
setmode(5,0);
```

where 5 is the defined number for this mode and 0 indicates that all control flags are to be turned OFF. If a 1 were in this position, all control flags would be left as set previously.

This mode is, in effect, implemented in one routine called "ploth". This routine plots a set of x,y coordinate points, as did "plotln" in the previous chapter, and then shades the area 'under the curve' as described above. As with "plotln", the "ploth" routine requires two equal length arrays of real numbers that describe the 'function' to be plotted, and an integer value that denotes the length of the arrays. It also requires, as a fourth parameter, the real y value (representing the x axis) from which the plot should start. Thus a call to this routine would be as follows :

```
ploth(px,py,n,base);
```

where "px" and"py" are the names of the arrays, "n" is the length of the arrays, and "base" is the desired base of the histogram plot. For proper results, the x values must be arranged in ascending order in the array.

1. PROGRAM 16

This sample program, available for execution in the "cs2000 - ramtek" directory, will illustrate the histogram style plot achieved by "ploth". Prior to plotting the histogram, a set of axes, that will coincide with the center of the plot, are positioned on the screen. The user should note that the histogram overlaps and wipes out these axes in the areas that are common to the two displays. If the "axis" call had been placed after the "ploth" call, this condition would be reversed. The routine "plotln" is also called with the same data to point out the relationship between these two routines. (Note : The "setmode" calls could have been omitted, since both "ploth" and "plotln" set their own required modes internally.)

a. To use the cs2000 directory :

LOGIN: cs2000 c/r

PASSWORD: student c/r

cd ramtek c/r

b. To list the program :

list -c p16.c c/r

c/r

c. To execute the program :

p16.x c/r

d. The listing follows :

```
float px[40] {3.0,3.1,3.2,3.3,3.4,3.5,3.6,3.7,3.8,3.9,
              4.0,4.1,4.2,4.3,4.4,4.5,4.6,4.7,4.8,4.9,
              5.0,5.1,5.2,5.3,5.4,5.5,5.6,5.7,5.8,5.9,
              6.0,6.1,6.2,6.3,6.4,6.5,6.6,6.7,6.8,6.9};
float py[40] {3.0,3.0,3.0,3.1,3.2,3.4,3.6,3.9,4.3,4.6,
```

```
                    5.3,5.6,5.9,6.1,6.3,6.5,6.6,6.7,6.8,6.9,
                    6.9,6.8,6.7,6.6,6.5,6.3,6.1,5.9,5.6,5.3,
                    4.6,4.3,3.9,3.6,3.4,3.2,3.1,3.0,3.0,3.0};
int pn 40;
float pbase 5.0;

main()
   {ramtek();       //open the device
    screen(0.0,0.0,10.0,10.0);  //set dimensions
    colort(7);     //color table 7
    color(1);    //color 1
    axis(4.95,5.0);     //draw axes
    sleep(3);     //pause
    color(7);     //color 7
    setmode(5,0);     //select graphic plot mode
    ploth(px,py,pn,pbase);    //plot histogram
    sleep(3);     //pause
    color(5);     //change color
    setmode(4,0);     //change mode
    plotln(px,py,pn);    //plot the 'function'
   }
```

C. ADVANCED METHODS

The use of the graphic plot mode for any more detailed operations than can be accomplished by the "ploth" routine will require the user to consult the RAMTEK GX-100 Programming Manual [1].

XIII. GRAPHIC CARTESIAN MODE

A. INTRODUCTION

The graphic cartesian mode is used to draw solid rectangles on the RAMTEK screen. These rectangles must be defined by two diagonally opposite corner points, the second of which becomes the new CCP after completion. Since these rectangles may be displayed in any color, the user, by choosing the background color, can effectively use this mode for selective erasure of rectangular areas.

B. BASIC USER GUIDELINES

Selection of the graphic cartesian mode can be achieved by a call of the form

```
setmode(6,f);
```

where 6 is the defined number for this mode. The parameter "f" should have the value 0 to turn OFF all control flags, or the value 1 to leave all flags as they were set prior to this call.

The simplest implementation of this mode is through the "block" routine. The user must first have selected his desired color by using the "color" routine. Then the call to "block", which requires four real number parameters, would appear as follows :

```
block(x1,y1,x2,y2);
```

where the two respective coordinate pairs define two diagonally opposite corners of the desired block to be displayed. The resulting COP will be located at the point (x2,y2), the last to be transmitted to the RAMTEK.

The "block" routine does put the RAMTEK into the graphic cartesian mode by itself and it also does some fairly elaborate error checking. But a quick scan of the source code would reveal that the actual rectangular display is generated by issuing a "strtxy" call followed by a "point" call. The user is, of course, free to utilize these routines also. In fact, in some applications they are essential.

Thus after selecting the proper mode and color, the user could create the same display as the "block" call above by issuing the following two subroutine calls :

```
strtxy(x1,y1); point(x2,y2);
```

(assuming the parameter values were the same as above).

C. ADVANCED METHODS

Although the basic results are the same, a variety of approaches are available. Some users may find the "pointr" routine convenient in applications where relative addressing becomes easier to compute than absolute. Thus after using the "strtxy" routine to establish an initial COP, the "pointr" routine could be called repeatedly (as could "point") to generate a series of linked rectangles.

There also may be certain applications in which the indexed addressing feature is useful. This may be implemented by the "index" routine explained in Chapter IV under "Virtual Screen Addressing".

1. PROGRAM 17

This sample program is available in the "cs2000 - ramtek" directory to graphically demonstrate the effect of the separate routines in this mode. The user should observe the following : (1) the use of "block" to draw a block in the upper left quadrant of the screen; (2) the use of "strtxy" and "point" to draw a similar block in the upper right quadrant; (3) the use of relative addressing when using "pointr" to generate two small boxes in the lower right quadrant; (4) the use of indexed addressing to position three small boxes in the lower left quadrant; (5) the 'selective erasure' effect achieved by selecting color 0 and drawing over part of the block in the upper right quadrant.

To assist the user in accessing this program the following guidelines are provided :

a. To use the cs2000 directory :

LOGIN: cs2000 c/r

PASSWORD: student c/r

cd ramtek c/r

b. To list the program :

list -c p17.c c/r

c/r

c. To execute the program :

```
p17.x   c/r
```

d. The listing follows :

```
main()
  {ramtek();     //open the device
   screen(0.0,0.0,20.0,20.0);  //set dimensions
   setmode(6,0);    //select graphic cartesian mode
   colort(7);     //select color table 7
   color(1);      //select color 1
   block(3.0,13.0,7.0,17.0);  //draw a block
   sleep(2);      //pause
   strtxy(13.0,17.0);    //establish new COP
   point(17.0,13.0);     //draw a block
   sleep(2);      //pause
   color(3);      //select color 3
   strtxy(17.0,7.0);     //establish new COP
   pointr(-2.0,-2.0);    //draw "relative" to COP
   pointr(-2.0,-2.0);    // "       "       "  "
   sleep(2);      //pause
   color(1);      //color 1
   index(1,1.0,1.0);  //"index" from (1.0,1.0)
   strtxy(2.0,2.0);    //set new COP
   point(4.0,4.0);     //draw a block
   color(5);   //select color 5
   strtxy(3.0,3.0);    //set new COP
   point(5.0,5.0);    //draw a block
   color(7);   //select color 7
   strtxy(4.0,4.0);    //set new COP
   point(6.0,6.0);    //draw a block
   sleep(2);      //pause
   index(0,0.0,0.0);  //turn OFF index flag
   color(0);   //select background color
   block(13.0,13.0,15.0,15.0); //"erase" part of a block
   }
```

XIV. GRAPHIC ELEMENT MODE

A. INTRODUCTION

This mode involves activating or 'painting' individual elements (pixels) on the RAMTEK screen. When operating in the graphic element mode the user must compute, either programmatically or by hand, the virtual screen coordinates of each pixel that he desires activated. As such, its use may may not seem as attractive as some other modes, yet for some applications this mode may prove extremely useful.

B. BASIC USER GUIDELINES

The graphic element mode may be selected by a call such as

```
setmode(7,f);
```

where 7 is the designated number of this mode. The second parameter (f) should be a '0' to turn OFF all control flags, or a '1' to leave the flags as they were.

By its nature, this mode demands an increased awareness of the virtual screen dimensions. The screen may be dimensioned to any size the user desires by a call of the form :

```
screen(xmin,ymin,xmax,ymax);
```

which will result in the minimum coordinate values representing the lower left corner of the screen and the

maximum values representing the upper right. There are some instances when viewing the screen in its 'real' dimensions will help. This may be achieved by using the "screen" routine to set the 640 by 240 dimensions by screen(0.0,0.0,640.0,240.0). One of the more useful application routines in this mode is the "plotpt" routine. This procedure, as with "ploth" and "plotln" in previous chapters, requires as parameters two equal length arrays of real numbers and an integer value defining that length. Thus if the following arrays were declared and initialized :

```
float px[n] {x1,x2,x3,....,xn};
float py[n] {y1,y2,y3,....,yn};
```

then the suproutine call

```
plotpt(px,py,n);
```

would correlate the 'n' values from "px" and "py" and activate those 'n' individual pixels on the screen according to the currently selected color.

The "plotpt" routine actually makes a series of calls to the familiar "point" function. In this mode, the "point" routine is used to address individual elements on the screen, not to define the end point of some vector or block drawing procedure as before. Thus the user may utilize "point" to activate arbitrary pixels. This can be helpful when plotting some function whose values are not known before-hand. (see sample program below)

The "pointr" routine may also be used to address some element relative to the previously addressed element (location of COP). The reverse background and indexed addressing flags which were explained in "Control Modes and Flags" (Chap IV) are applicable to this mode.

1. PROGRAM 18

The following program, when executed will compute and plot the points along the circumference of a circle. At the same time, it uses the results of its SIN function computations to plot a portion of a sine curve within the circle. This program is in the "cs2000 - ramtek" directory and can be accessed as follows :

a. To use the cs2000 directory :

LOGIN: cs2000 c/r

PASSWORD: student c/r

cd ramtek c/r

b. To list the program :

list -c p18.c c/r

c/r

c. To execute the program :

p18.x c/r

d. The listing follows :

```
double sin();
double cos();
float px;
float py;
float pr 3.0;     //radius of circle

main()
  {float i,t;
```

```
    ramtek();     //open the device
    screen(-5.0,-5.0,5.0,5.0);  //set dimensions
    color7(7);     //color table 7
    color(5);      //color 5
    setmode(7,0);    //select graphic element mode
    for(i = 0.0;i < 6.28318;i = i + 0.01)
      {px = pr * cos(i);
       t = sin(i);
       py = pr * t;
       point(px,py);     //plot the circle
       if(i <= 6.0) point((i-3),t);  //plot sine
      }
  }
```

C. ADVANCED METHODS

The graphic element mode seems best suited for plotting a series of points that are computed by the application program itself. Depending on the desired display, some projects that require the user to pre-compute his data points might be more adaptable to one of the Data modes. (see Raster and Transverse Data Modes)

XV. RAMTEK INTERACTIVE KEYBOARD

The RAMTEK keyboard is software adapted to be utilized as an interactive tool by the RAMTEK user. This keyboard (located in front of the GX-100A) is ON when the green switch/light (in the upper right corner) is glowing; to activate the keyboard when it is OFF, depress the switch/light. The keyboard is configured as shown in Figure 3.

Referring to Figure 3, the user will note that there are eighty-nine (89) keys. Each key may have from one (1) to three (3) states (uppercase, lowercase, control). The user has available 237 distinct codes that may be transmitted via the keyboard, with the ASCII code being utilized for all keys. The decimal representation of the actual code returned from selecting a particular key is shown in Figure 4.

The system software has been designed to allow the user to read keyboard input quite easily. The three functions that allow the user to interact with the keyboard can provide individual ASCII characters, integer numbers, or real numbers depending on the function utilized.

In addition to reading from the keyboard, the user may echo print back to the RAMTEK screen individual ASCII characters as they are read. The system support routine "out(n)"

provides this capability ("n" is an ASCII coded character). The function "out(n)" automatically repositions the COP one character to the right to prevent over-writing.

The function that returns individual ASCII characters to the user is "retchar()". To read a character from the RAMTEK keyboard the user would issue a call of the form

```
x = retchar();
```

This will result in "x" being assigned the value of the selected key.

A. PROGRAM 19

This program will read, and echo print to the RAMTEK screen, characters selected from the RAMTEK keyboard (NOT ALL SELECTIONS ARE PRINTABLE). The program terminates when a single carriage return is input.

1.To use the cs2000 directory:

```
LOGIN: cs2000 c/r
PASSWORD: student c/r
cd ramtek c/r
```

2.To list the program:

```
list -c p19.c c/r
```

3.To execute the program:

```
p19.x c/r
```

4.Program listing follows:

```
#define CR 13
main()
   {int x; //integer variable
   ramtek(); //initialize
```

```
colort(7); color(2); //select colors
while ((x = retchar()) != CR) //loop
   {strtxy(50.0,50.0); //set positon
   out(x); //echo character
   }
}
```

To read an integer from the RAMTEK, the user depends upon the system function "geti()". To read an integer quantity from the key board the user issues a call of the form

```
y = geti();
```

The function "geti()" recognizes a carriage return as the end of the input value. To input the value one hundred the user would type (at the RAMTEK keyboard) :

```
100   c/r
```

A negative one (-1) is returned if a carriage return is typed as the first character. The only valid inputs are lowercase numbers; all others are ignored.

B. PROGRAM 20

This program allows the user to input integer values from the keyboard until a carriage return is typed as the first character in a new number. The integer value is output to the RAMTEK screen for user verification (see "itoal(x)").

1.To use the cs2000 directory:

```
LOGIN: cs2000 c/r
PASSWORD: student c/r
cd ramtek c/r
```

2.To list the program:

```
list -c p20.c   c/r
```

3.To execute the program:

```
p20.x c/r
```

4.Program listing follows:

```
#define CR 13
main()
   {int k; //integer variable
   ramtek(); //initialize
   color(7); color(2); //select colors
   while ((k = geti()) != CR) //loop
      {strtxy(50.0,50.0); //set COP
      itoal(k); //output value
      }
   }
```

To read a real number from the RAMTEK keyboard the user will call "getf()". The user would issue a call of the form

```
y = getf()
```

to read a real number. The result would be the assignment of a real number to the variable "y". The function "getf()" recognizes a carriage return as the end of the input value. Only lowercase numbers, "e" for exponential, "+" or "-", and "." are valid inputs to the routine; all others are ignored.

The user should now be able to effectively use the keyboard as an interactive tool. For specifics on the functions see Appendix B.

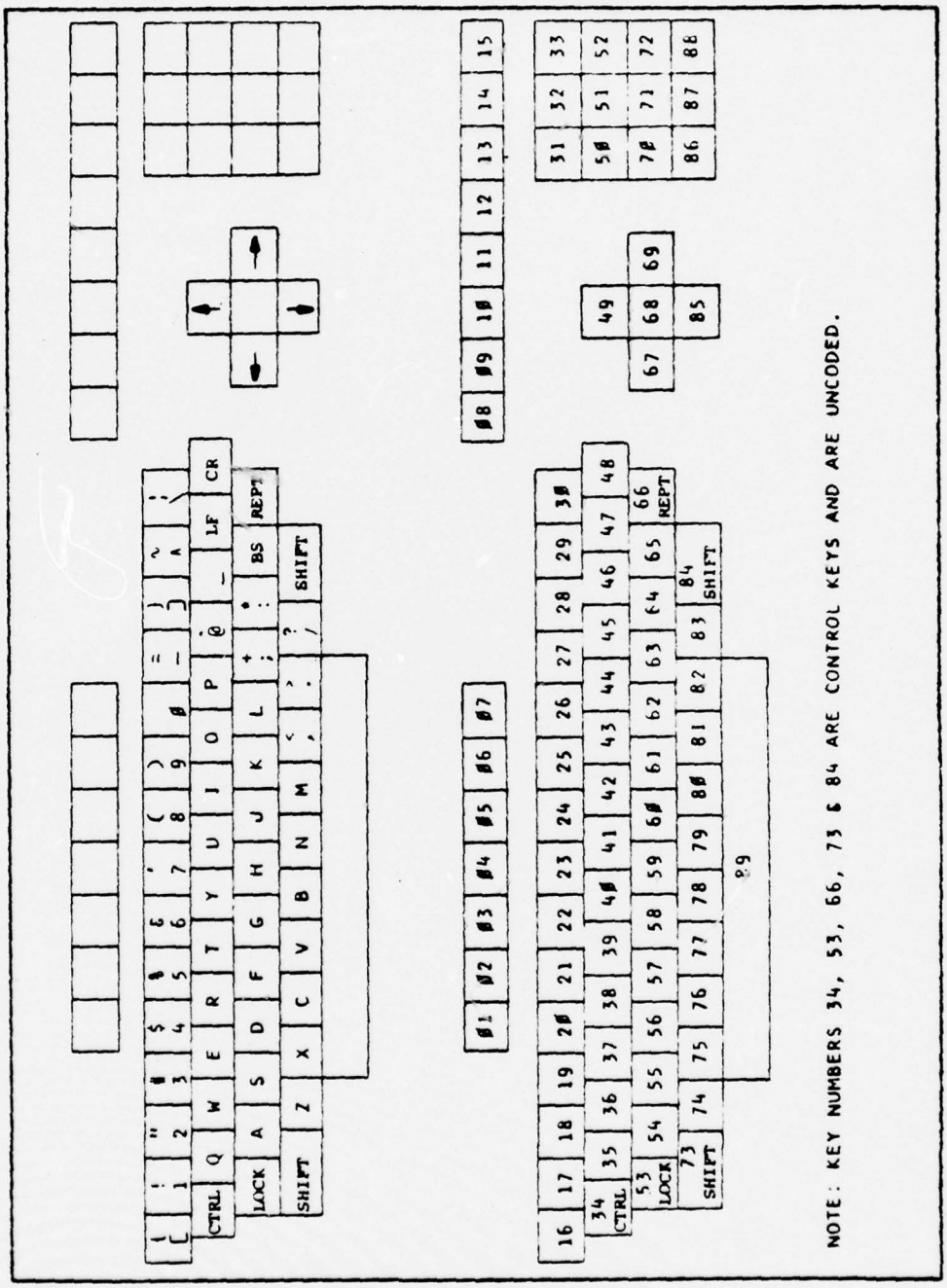

CTRL
LOCK
SHIFT
BS
REPT
LF
CR
NOTE: KEY NUMBERS 34, 53, 66, 73 & 84 ARE CONTROL KEYS AND ARE UNCODED.

FIGURE 5

LC-lowercase
UC-uppercase
CTRL-control

1	145	177	209	2	146	178	210
3	147	179	211	4	148	180	212
5	149	181	213	6	150	182	214
7	151	183	215	8	152	184	216
9	153	185	217	10	154	186	218
11	155	187	219	12	156	188	220
13	157	189	221	14	158	190	222
15	159	191	223	16	91	123	27
17	49	33	17	18	50	34	18
19	51	35	19	20	52	36	20
21	53	37	21	22	54	38	22
23	55	39	23	24	56	40	24
25	57	41	25	26	48	48	---
27	45	61	29	28	93	125	29
29	94	126	30	30	92	124	28
31	133	165	197	32	137	169	201
33	141	173	205	34	---	---	---
35	113	81	17	36	119	87	23
37	101	69	5	38	114	82	18
39	116	84	20	40	121	89	25
41	117	85	21	42	105	73	9
43	111	79	15	44	112	80	---
45	64	96	---	46	95	95	31
47	10	10	10	48	13	13	13
49	---	160	192	50	134	166	198
51	138	170	202	52	142	174	206
53	---	---	---	54	97	65	1
55	115	83	19	56	100	68	4
57	102	70	6	58	103	71	7
59	104	72	8	60	106	74	10
61	107	75	11	62	108	76	12
63	59	43	27	64	58	42	26
65	8	8	8	66	---	---	---
67	---	161	193	68	---	146	196
69	---	162	194	70	135	167	199
71	139	171	203	72	143	175	207
73	---	---	---	74	122	90	26
75	120	88	24	76	99	67	3
77	118	86	22	78	98	66	2
79	110	78	14	80	109	77	13
81	44	60	12	82	46	62	14
83	47	63	15	84	---	---	---
85	---	163	195	86	---	168	200
87	---	172	204	88	---	176	208
89	32	32	32				

FIGURE 4

XVI. SPECIAL APPLICATIONS ROUTINES

The purpose of this chapter is to present those routines from the software support package which are not an integral part of any topic discussed earlier, and which would otherwise be omitted. Some of the routines in the support package are merely subordinate parts of larger routines and serve no stand-alone purpose. These will not be discussed. The routines presented here are those that might be employed by a user's application program.

There is only one routine in the support package that requires knowledge of the fact that this device actually has 512 raster lines -- 480 of which are visible and 32 of which are off the bottom of the screen. This is the "scroll" function, which is used to make a display move up or down by a certain number of real lines. If a portion of the display is 'scrolled' off the top of the screen, it will 'wrap around' and reappear at the bottom (and vice versa). Thus the 32 non-visible lines that exist off the bottom of the screen must be taken into consideration when making use of this feature. The subroutine call would appear as

```
scroll(direction,numlines);
```

where the parameter "direction" must be either a "u" for UP or a "d" for DOWN, and the "numline" parameter is a real number denoting the number of real screen lines the display

should be moved.

In previous chapters, screen addressing has been presented in terms of the virtual screen. There are two routines that map these virtual addresses into real (and visible) screen locations for the actual display generator. These are "conve" and "convl". The "conve" function converts the X coordinate into a real screen element position. The "convl" function converts the Y coordinate into a visible real screen line number. The combination of these two define an element/line intersection or pixel. These functions are used internally by all the software display routines to convert user-defined addresses to real screen locations before passing these to the RAMTEK buffer for display. These two procedures may be employed by the user with individual calls such as

```
conve(x); convl(y);
```

where "x" and "y" are the real numbers of the desired virtual screen address.

In order to display a given integer value (in the range of -32767 to +32767) on the screen, the "itoal" routine may be used. This routine will convert the given integer parameter into its ASCII code and display the desired integer value on the screen at the COP. The call would be

```
itoal(n);
```

where "n" is any valid integer value.

Likewise, the "ftoa" function may be used to display a real number with no more than nine digits to the left of the decimal point and no more than four digits to the right. Thus if "x" were a valid real number, the call

```
ftoa(x);
```

would display the decimal representation of that number (complete with decimal point) on the RAMTEK screen at the location of the COP.

The "out" function takes, as a single parameter, the ASCII code for any valid character and "dumps" it to the RAMTEK to display that character. Since the ASCII code for the character is required, this may be achieved in two ways. First, the character may be enclosed in single quotes within the subroutine call, such as

```
out('x');
```

to display the letter x; or, second, if the user pre-defines a character, such as with

```
#define COMMA 054
```

then the call

```
out(COMMA);
```

would display a comma at the location of the COP. The x value or element position of the COP is incremented by 7 elements to prevent destroying that character with the next to be displayed. (see "Alphanumeric Mode")

As briefly mentioned in the "Alphanumeric Mode", raw data is passed to the RAMTEK by the "dump" routine. This

procedure sends a pre-declared number of bytes of data from the software buffer (called "buff") to the RAMTEK hardware buffer. The number of bytes to be passed is normally kept in "bytecnt". The address of the next available location in "buff" is kept in a pointer called "ptrbuff". Thus, for example, to send the octal value 060017 to the RAMTEK, the following instructions would be used :

```
*ptrbuff = 060017;
ptrbuff++;
bytecnt = bytecnt + 2;
dump(buff,bytecnt);
```

To operate at this level in the RAMTEK environment, the user should familiarize himself with the RAMTEK GX-100 Programming Manual [1] and the RAMTEK software source code available in the Lab.

APPENDIX B

SOFTWARE SUPPORT FUNCTION DESCRIPTIONS

This Appendix contains the descriptions of the user interface routines in the format that is followed in the documentation of the UNIX operating system in the Naval Postgraduate School Computer Laboratory.

RESERVED WORDS

ADOFF	enthold	LL1	setup
adon	epage1	LL2	sixpak
ALPHA	epage2	LTA	size
axis	enthold	LTD	skip
BKON	erase	lttr	SSCALL
BKOFF	ERS	LXD	status
bkrnd	fixpt	maxsw	strout
blank	flip	moreinst	strtxy
BLK	fname	msw	swary
block	fp	n0-n17	symbol
box	FPOFF	octbl	systbl
bracket	FPON	out	tabcolor
buf	ftoa	pause	tabdim
buff	getf	pick	tabinit
buildsw	geti	plotpt	tabsw
butbl	getnum	plotln	tblwho
bytecnt	getsw	ploth	tdata
c	getxy	point	terse
chcol	GRAPHCRT	pointr	texto
chnge	GRAPHELM	proc1	tfpi
clrhold	GRAPHVEC	proc2	TRANSD
clrtbl	head	proc3	triple
code	headptr	proc4	tx
codeit	heat	proc5	txdim
coke	holdx	pst	txmax
colint	holdy	ptrbuff	txmin
color	index	putup	tv
colort	inptrs	qptr	tydim
colortbl	instr	qptr1	tymax
colsw	inst1-inst80	qt	tymin
comb	inter	quest	upcnt
COMMA	int53	q1	vector
COMPD	int60	ramtek	wait
conve	itoa	RASTERD	WDOFF
convl	itoal	redim	WDON
copy	IXOFF	resclr	writon
CR	IXON	retchar	xaxis
cursor	LCM	savclr	xmin
data	lcmhold	savstat	xmax
datap	LER	scissor	xswitch
dblwid	LEx	SCR	yaxis
disp	LE1	screen	yaxisf
dspyctbl	LE2	scroll	ymin
dump	LLR	SDCU	ymax
einst	LLx	setmode	

NAME:

axis - draw coordinate axis

SYNOPSIS:

```
axis(x,y)
float x, y;
```

DESCRIPTION:

Draws a cartesian coordinate x and y axis on the screen with center at user-defined screen coordinate (x,y).

The operation is totally independent of any control mode issued previously.

The COP is left at (max x-value,y).

Normal return is zero.

DIAGNOSTICS:

Returned -1 indicates no portion of the axis lies on the user-defined screen.

NAME:

bkrnd - change reverse background flag

SYNOPSIS:

```
bkrnd(a)
int a;
```

DESCRIPTION:

If a is zero, the reverse background flag in the control mode is set to zero, i.e. turned off.

If a is equal to one, the reverse background flag in the control mode is set to one, i.e. turned on.

Normal return is zero.

DIAGNOSTICS:

Returned -1 indicates the passed integer a is not equal to zero or one.

NAME:

block - draw solid block

SYNOPSIS:

```
block(x1,y1,x2,y2)
float x1,y1,x2,y2;
```

DESCRIPTION:

A solid block is drawn with opposite corners as defined by user-defined screen coordinates (x1,y1) and (x2,y2).

The COP is left at (x2,y2). The operation is independent of any mode issued previously but is sensitive to all flags applicable to Graphic Cartesian mode.

Normal return is zero.

DIAGNOSTICS:

Returned -1 indicates the block can not be drawn on the user-defined screen.

SEE ALSO:

index(), pkrng(), dblwid()

NAME:

buildsw - build logical switches on the tablet

SYNOPSIS:

```
buildsw(sw,xh,xl,yh,yl)
float xh,xl,yh,yl;
int sw;
```

DESCRIPTION:

Allows the user to define individual logical 'switches' on the Vector General Tablet by passing the following parameters :

1-user's logical switch number (integer)
2-high x value (real)
3-low x value (real)
4-high y value (real)
5-low y value (real)

-- thus defining a selection box and assigning it the desired number. Maximum number of switches is 25.

DIAGNOSTICS:

Normal return is the switch number.
Returned -1 indicates attempt to create more than 25 switches.

SEE ALSO:

sixpak(); tabsw();

NAME:

chnge - change color table entry

SYNOPSIS:

```
chnge(numb,entry,cont)
int numb, entry, cont;
```

DESCRIPTION:

Changes the specified entry in the indicated color table to the passed parameter cont. triple() should be used to put cont into the proper form.

numb is the table number in which the entry is to be changed. entry is the entry number to be changed. cont is the integer twelve bit code which is loaded in the specified entry.

Normal return is zero.

DIAGNOSTICS:

All errors are indicated by negative returned values. The error values and their meanings are as follows:

-1 Indicates numb is less than four or greater than seventeen.

-2 Indicates entry is negative or greater than fifteen.

SEE ALSO:

triple()

NAME:

clrtbl - load color table

SYNOPSIS:

```
clrtbl(n,name)
int n, *name;
```

DESCRIPTION:

Loads color table number n with the codes contained in the array pointed to by name;

The array is an integer array containing sixteen integers which represent the octal code of the color table entries between zero and fifteen. The array is assumed to be declared sixteen words in length.

n is an integer value which ranges from four to seventeen.

The color tables from zero to four are system defined tables. They contain the following:

Table 0 - Fifteen shades of grey.
Table 1 - Fifteen shades of blue.
Table 2 - Fifteen shades of green.
Table 3 - Fifteen shades of red.

If the user desires to modify a system table see inter().

Normal return is zero.

DIAGNOSTICS:

Returned -1 indicates the passed integer n is less than four or greater than seventeen.

SEE ALSO:

triple()

NAME:

color - select color

SYNOPSIS:

```
color(s)
int s;
```

DESCRIPTION:

The passed integer s is the number of the desired entry in the current color table. The color located in that entry will be used for all subsequent entities displayed until a different color is issued.

s is between zero and fifteen.

Normal return is zero.

DIAGNOSTICS:

Returned -1 indicates the passed integer s is negative or greater than fifteen.

SEE ALSO:

chnge()

NAME:

colort - select display color table

SYNOPSIS:

```
colort(t)
int t;
```

DESCRIPTION:

The passed parameter t is the number of the color table that is sent to the RAMTEK device for display.

t is between zero and seventeen.

Normal return is zero.

DIAGNOSTICS:

Returned -1 indicates the passed integer t is negative or greater than seventeen.

SEE ALSO:

clrtbl()

NAME:

conve - convert an x value to an element location

SYNOPSIS:

```
conve(x)
float x;
```

DESCRIPTION:

Converts a virtual screen x-dimension value or address into a real screen element position. The floating point parameter must be a valid address component for the current virtual screen space.

DIAGNOSTICS:

Returns the number of the real screen element position.

SEE ALSO:

convl();

NAME:
convl - convert a v value to a real screen line number

SYNOPSIS:
```
convl(y)
float y;
```

DESCRIPTION:
Converts a virtual screen y-dimension value into a real screen line number. The floating point parameter must be a valid address component for the current virtual screen space.

DIAGNOSTICS:
Returns the number of the real screen line.

SEE ALSO:
conve();

NAME:

cursor - software cursor (V.G. tablet driven)

SYNOPSIS:

```
cursor();
extern float tx,ty;
```

DESCRIPTION:

This routine places a cursor on the screen of the RAMTEK in color 15 of the current color table. The cursor is destructive in that it erases everything that it passes over. Control of the cursor is via the VECTOR GENERAL tablet. When the pen is depressed the cursor disappears and exits. The x and y values of the pen/cursor location are saved in tx and ty (must be global to the user program) for the user.

DIAGNOSTICS:

none

SEE ALSO:

terse() , getxy() , tabdim() , tabinit() , screen()

NAME:

data - display raw data

SYNOPSIS:

```
data(name,l)
int *name, l;
```

DESCRIPTION:

The raw data passed in the linear array pointed to by name is displayed on the RAMTEK according to the current control mode.

l is the length of the array in bytes!

Normal return is zero.

DIAGNOSTICS:

Returned -1 indicates the passed integer l is negative or zero.

SEE ALSO:

strout(), lttr()

NAME:

dblwid - change double width flag

SYNOPSIS:

```
dblwid(x)
int x;
```

DESCRIPTION:

If x is equal to zero, the double width flag in the control mode is set to zero, i.e. turned off.

If x is equal to one, the double width flag in the control mode is set to one, i.e. turned on.

Normal return is zero.

DIAGNOSTICS:

Returned -1 indicates the passed integer x is not equal to zero or one.

SEE ALSO:

size()

NAME:

disp - display a selected color table (long form)

SYNOPSIS:

```
disp(n);
int n;
```

DESCRIPTION:

The color table denoted by the integer parameter 'n' is displayed on the screen, along with the octal code required to generate each entry.

SEE ALSO:

dspvctbl()

NAME:
dspyctbl - display a selected color table

SYNOPSIS:
```
dspyctbl(j);
int j;
```

DESCRIPTION:
The color table denoted by the integer parameter 'j' is displayed on the RAMTEK. Background color is entry 0; display color is entry 15.

SEE ALSO:
disp( )

NAME:

dump - send raw data to the RAMTEK

SYNOPSIS:

```
dump(buff,bytes);
char *buff;
int bytes;
```

DESCRIPTION:

Writes raw data to the RAMTEK instruction buffer. This routine is for low level control of the RAMTEK and is very restrictive. The raw data is interpreted according to the mode and flags set within the device.

DIAGNOSTICS:

none

SEE ALSO:

write() , RAMTEK PROGRAMMERS MANUAL

NAME:
erase - erases the screen

SYNOPSIS:
erase()

DESCRIPTION:
The RAMTEK screen is erased to the current background color (ie, the state of the reverse background flag).

NAME:

fixpt - change fixed point flag in control mode

SYNOPSIS:

```
fixpt(x)
int x;
```

DESCRIPTION:

If x is equal to zero, the fixed point flag in the control mode is set to zero, i.e. turned off.

If x is equal to one, the fixed point flag in the control mode is set to one, i.e. turned on.

Meaningful only in Graphic Vector or Graphic Plot mode. In Graphic Vector mode it causes all subsequent vectors to be drawn from a common point as issued by a subsequent (or the last previous) strtxy(). In Graphic Plot mode this flag is used to achieve the histogram style plot of "ploth()".

Normal return is zero.

DIAGNOSTICS:

Returned -1 indicates the passed integer x is not equal to zero or one.

NAME:

ftoa - converts real numbers to ASCII for output

SYNOPSIS:

```
ftoa(in);
float in;
```

DESCRIPTION:

Converts a floating point number to an ASCII code so that it may be output to the RAMTEK. The passed floating point number is first converted to the ASCII representation and then output to the RAMTEK screen at the user defined CDP. Allows for only four digits right of the decimal place and eight digits left of the decimal place.

DIAGNOSTICS:

none

SEE ALSO:

data() , itoal()

NAME:

getf - read floating point number from RAMTEK keyboard

SYNOPSIS:

getf()

DESCRIPTION:

Returns a floating point number from the RAMTEK keyboard. The only numbers recognized are: an optional minus sign followed by a string of digits optionally containing one decimal point, then followed optionally by the letter 'e' followed by a signed integer.

Normal return is the floating point number.

SEE ALSO:

getnum(), retchar()

NAME:
geti - return an integer from the RAMTEK

SYNOPSIS:
geti();

DESCRIPTION:
Returns an integer value from the RAMTEK keyboard. ASCII characters between 0 and 9 , inclusively , are recognized; all others are ignored. Upon typing a carriage return the process begins; hence carriage return denotes end of string.

DIAGNOSTICS:
none

SEE ALSO:
atoi() , retchar()

NAME:

getnum - read number from RAMTEK keyboard

SYNOPSIS:

```
getnum(base)
int base;
```

DESCRIPTION:

Returns a positive or negative integer number from the RAMTEK keyboard. Numerals typed up to a comma or a c/r on the keyboard are considered to be the number. The routine will not return any value until a c/r or a comma is typed, nor will it accept characters other than digits from zero to nine when base is equal to ten, or digits from zero to seven in case of base eight.

base is the base of the integer number returned. It is restricted to eight (octal) or ten (decimal).

Normal return is the integer number.

DIAGNOSTICS:

All errors are indicated by negative returned values. The error values and their meanings are as follows:

-1 Indicates a c/r was struck without any previous entries.

-2 Indicates the passed parameter base was not equal to eight or ten.

SEE ALSO:

retchar(), getf()

NAME:
getsw - get the number of a selected logical switch

SYNOPSIS:
```
getsw();
```

DESCRIPTION:
Determines if the user has selected a valid logical switch on the Vector General tablet. If so, it returns the number of that switch.

DIAGNOSTICS:
Returned -1 indicates invalid selection.

SEE ALSO:
buildsw(); sixpak(); tabsw();

NAME:

getxy - get x,y coordinates of a point on the tablet

SYNOPSIS:

```
getxy(p)
int p;
extern float tx,ty;
```

DESCRIPTION:

Reads (into global variables tx and ty) the x,y coordinate values from the tablet at the point defined by the tip of the stylus. If the parameter 'p' is a 1, the stylus must be depressed on the tablet surface to select a point; if 'p' is a 0, the coordinates nearest the tip of the pen are read. These coordinates are automatically converted to virtual tablet coordinates if the tablet has been redimensioned (by a prior call to tabdim).

NAME:

index - select indexing

SYNOPSIS:

```
index(i,x,y)
float x, y;
int i;
```

DESCRIPTION:

If i equals zero, indexing is deselected and the indexing flag in the control mode is turned off. The index registers are then loaded with zero.

If i equals one, indexing is selected and the indexing flag in the control mode is turned on. The index registers are then loaded with the displacements passed in x and y. x is loaded in the x-index (element) register and y is loaded in the y-index (line) register.

x must be no greater than the user-defined screen width. y must be no greater than the user-defined screen height.

Normal return is zero.

DIAGNOSTICS:

All errors are indicated by negative returned values. The error values and their meanings are as follows:

-1 Indicates the passed integer i is not equal to zero or one.

-2 Indicates one of the passed displacements x and y is greater than the screen width or height.

SEE ALSO:

screen()

NAME:

inter - interactive color table modification routine

SYNOPSIS:

inter()

DESCRIPTION:

Sets up and runs a tutorial program that enables the user to look at, select and modify color tables during program execution. There are two basic modes in which the routine operates. Each of these modes has a series of commands which the user can execute.

1. Paging Mode. Upon execution the paging mode is immediately entered and a set of instructions is displayed. The following commands are then acceptable:

p - Indicates a particular table number is to be displayed. The number must then be entered between zero and seventeen.

n - Increments table number displayed by one.

b - Decrements table number displayed by one.

i - Displays instructions.

q - Quits from the interactive routine.

e - Enters the edit mode.

2. Edit Mode. When the edit mode is entered, a set of instructions is displayed listing the commands that can be issued and the edit method they initiate. The commands are:

t - Enters table assignment method. Permits entries from any table to be copied into the table designated by the user. The table number which is to receive the copied entries must be entered first.

o - Enters octal assignment method. Entries are specified in a designated table by entering the ordered triple that defines the color desired. The table number (4-17) in which the entries are to be defined must be en-

tered first.

c - Enters combining assignment method. This method allows the user to logically OR two separate color lookup table entries into a selected user table entry. Whole tables may also be combined into a selected user table. The user table number (4-17) must be entered first.

f - Enters inverting tables method. This method inverts the user table designated by the user. Entry zero becomes fifteen, entry one becomes fourteen, etc. The act of entering the user table number executes the inversion.

m - Enters copying tables method. After entry of the user table number that is to receive a copied table the number of the table to be copied is entered. The act of entering the second table number causes the copy to take place.

Upon entry, the editing methods display a list of instructions on how the method is to be used and the results obtained. Each of the methods have commands that may be issued. The legal commands for each edit method are summarized below.

- Table Assignment Method.
  - n - Indicates the table number and entry to be copied follows.
  - c/r-Increments through the entries in the receiving user table.
  - p - Displays the color lookup table being modified. To return from the display, type a q.
  - q - Quits back to edit mode instructions.

- Octal Assignment Method.
  - n - Indicates the octal triple to be entered as the entry follows.
  - c/r-Increments through the entries in the receiving user table.
  - p - Displays the color lookup table being modified. To return from the display, type a q.
  - q - Quits back to edit mode instructions.

- Combining Assignment Method.
    - n - Indicates the two ordered pairs designating the table number and entry number to be ORd are to follow.
    - c/r-Increments through the entries in the receiving user table.
    - p - Displays the color lookup table being modified. To return from the display, type a q.
    - w - Indicates two whole tables are to be combined (ORd) and their numbers are to follow.
    - q - Quits back to edit mode instructions.

- Inverting Tables Method.
    - p - Displays the color lookup table that is being or was inverted.
    - q - Quits back to edit mode instructions.

- Copying Tables Method.
    - p - Displays the receiving table.
    - q - Quits back to edit mode instructions.

NAME:

itoal - converts integers to ASCII for output

SYNOPSIS:

```
itoal(in);
int in;
```

DESCRIPTION:

Itoal takes an integer value and converts it to an ASCII coded character string. Having converted the value it is then displayed at the user defined COP.

DIAGNOSTICS:

none

SEE ALSO:

ftoa() , data()

NAME:
lttr - display single character

SYNOPSIS:
```
lttr(ch,size)
int ch, size;
```

DESCRIPTION:
Will display the character passed with the size indicated at the current operating point.

After display, the current operating point is on the same line and seven actual screen elements to the right of its last position.

ch contains the ASCII code for the character to be displayed, or the actual character enclosed in single quotes.

If size is equal to one, the character is displayed in standard size.

If size is equal to two, the character is displayed in double width size.

The operation is independent of any mode issued previously, but is sensitive to the flags applicable to Alphanumeric mode.

Normal return is zero.

DIAGNOSTICS:
Returned -1 indicates the passed integer size was not equal to one or two.

SEE ALSO:
strout(), data(), size(), index(), bkrnd(), writon()

NAME:
out - outputs ASCII characters to the RAMTEK

SYNOPSIS:
```
out(ch);
```

DESCRIPTION:
Echos the passed ASCII coded character to the RAMTEK screen. The CCP is then incremented to the right to prevent overwriting.

Typical call:
```
out(48);
```

DIAGNOSTICS:
none

SEE ALSO:
dump() , lttr() , retchar()

NAME:

pick - output an integer between zero and seventeen

SYNOPSIS:

```
pick(i);
int i;
```

DESCRIPTION:

Outputs an integer to the RAMTEK screen (between 0 and 17) depending entirely upon the passed parameter. Inflexible but useful in displays involving color table manipulation.

DIAGNOSTICS:

none

SEE ALSO:

data()

NAME:

ploth - plot data as a histogram

SYNOPSIS:

```
ploth(x,y,n,base)
float *x, *y, base;
int n;
```

DESCRIPTION:

The values y[i] are treated as functions of x[i]. As the function is plotted, the area between the curve and the x-axis, as designated by base, is filled in with the color last selected.

That portion of the histogram that lies on the screen will be plotted.

The operation is totally independent of any control mode issued previously.

x and y are pointers to linear arrays.
n is the number of points to be plotted.

Normal return is zero.

DIAGNOSTICS:

All errors are indicated by negative returned values. The error values and their meanings are as follows:

Returned -1 indicates the passed parameter base is not on the user-defined screen.

SEE ALSO: plotln(), plotpt()

NAME:

plotln - plot data with connected lines

SYNOPSIS:

```
plotln(x,y,n)
float *x, *y;
int n;
```

DESCRIPTION:

The values y[i] are treated as functions of x[i]. As the function is plotted, the successive points are connected by straight lines of the color last selected.

That portion of the plotted curve which lies on the screen will be plotted.

The operation is totally independent of any control mode issued previously.

x and y are pointers to linear arrays.
n is the number of points to be plotted.

SEE ALSO:

ploth(), plotpt()

NAME:
plotpt - plot data with points

SYNOPSIS:
```
plotpt(x,y,n)
float *x, *y;
int n;
```

DESCRIPTION:
The values of y[i] are treated as functions of x[i]. The function is plotted with dots for each (x[i],y[i]) coordinate. The dots are in the color last selected.

If a point does not lie on the user-defined screen, it is not plotted.

The operation is totally independent of any control mode issued previously.

x and y are pointers to linear arrays.
n is the number of points to be plotted.

SEE ALSO:
plotn(), plotln()

NAME:

point - define point

SYNOPSIS:

```
point(x,y)
float x, y;
```

DESCRIPTION:

Defines a point on the virtual screen. The point must lie within the user-defined screen.

If in Graphic Vector mode, (x,y) defines the endpoint of a vector and causes it to be drawn. The current operating point is then (x,y).

If in the Graphic Cartesian mode, (x,y) defines the diagonally opposite corner (from the CUP) of a proposed rectangle. The rectangle thus defined is shaded in the currently selected color. The current operating point is then (x,y).

If in the Graphic Element mode, (x,y) defines a single point on the user-defined screen and causes a dot to be drawn there. The current operating point is then (x,y).

Normal return is zero.

DIAGNOSTICS:

Returned -1 indicates the passed point does not lie on the user-defined screen.

SEE ALSO:

strtxy()

NAME:
pointr - define a point relative

SYNOPSIS:
```
pointr(x,y)
float x, y;
```

DESCRIPTION:
Defines a point on the user-defined screen relative to the last current operating point.

If in Graphic Vector mode a vector from the last current operating point to a point defined by the last current operating point plus the displacements x and y is drawn.

If in the Graphic Cartesian mode, a rectangle is defined by the current operating point and the point computed by adding x and y to the COP. This rectangle is displayed in the current color.

If in Graphic Element mode a dot is drawn at a point defined by the last current operating point plus the displacements x and y.

The COP is left at the calculated location.

SEE ALSO:
strtxy()

NAME:
ramtek - initiates RAMTEK system

SYNOPSIS:
ramtek()

DESCRIPTION:
Initiates the RAMTEK system and sets the default conditions as follows:

1 - Initialize the user-defined screen to 0.0 to 100.0 in x and y.
2 - Loads a shades of grey color table (0) in the RAMTEK.
3 - Selects color fifteen in table zero for display.
4 - Selects Alphanumeric control mode.
5 - Opens the RAMTEK for reading and writing.
6 - Erases the screen.

Normal return is zero.

DIAGNOSTICS:
Returned -1 indicates RAMTEK device could not be opened.

NAME:

redim - redimension the values read from the tablet

SYNOPSIS:

```
redim();
extern float tx,ty;
```

DESCRIPTION:

Redimensions or converts the current x,y coordinates read by "getxy" to virtual tablet coordinates. This is applicable after the routine "tabair" has been used to redimension the virtual tablet to a user-defined size. The converted coordinate values are placed in global variables tx and ty. No parameters are passed either way by this function.

SEE ALSO:

getxy();

NAME:

resclr - restore the user's color tables to the system

SYNOPSIS:

```
resclr( );
char *savcol;
extern colortbl[][];
```

DESCRIPTION:

The user's version of the color tables are brought into the system from a user file called "savcol" which must have been created by a prior call to savclr( ).

Normal return is the number of bytes read from "savcol" into "colortbl[][]" (should be 448).

DIAGNOSTICS:

Returned -1 indicates an unsuccessful attempt to open the file "savcol".
Returned -2 indicates an error in reading from "savcol" to colortbl[][].

SEE ALSO:

savclr( )

NAME:

restat - restore the RAMTEK (and VG Tablet) to previous status

SYNOPSIS:

```
restat( );
```

DESCRIPTION:

The control status that was in effect at the time of the last call to savstat( ) is restored to the RAMTEK. This includes mode, control flags, color table, and the dimensions of the virtual screen and virtual tablet.

SEE ALSO:

savstat( )

NAME:
retchar - read a character from RAMTEK keyboard

SYNOPSIS:
retchar()

DESCRIPTION:
An ASCII code representing the typed character is returned in the lower half of an integer.

SEE ALSO:
getnum(), getf(), geti()

NAME:

savclr - save the current color tables

SYNOPSIS:

```
savclr( );
char *savcol;
extern colortbl[][];
```

DESCRIPTION:

The current user-available color tables (4 - 17) are saved into a newly-created file called "savcol" in the user's directory. This file may subsequently be re-opened and read back into the system by a call to resclr( ). This allows the user to preserve desired color schemes between sessions.

Normal return is the number of bytes read into "savcol" -- (should be 448).

DIAGNOSTICS:

Returned -1 indicates an unsuccessful attempt to create the file "savcol".
Returned -2 indicates an error in writing to "savcol" from colortbl[].

SEE ALSO:

resclr( )

NAME:
savstat - save the status of the RAMTEK display

SYNOPSIS:
savstat( );

DESCRIPTION:
The current control status of the RAMTEK and VECTOR GENERAL Tablet is saved for subsequent use. This data includes the mode, control flags, current color table, and the dimensions of the virtual screen and virtual tablet.

SEE ALSO:
restat( )

NAME:

screen - define user screen

SYNOPSIS:

```
screen(x1,y1,x2,y2)
float x1, y1, x2, y2;
```

DESCRIPTION:

Defines a standard cartesian coordinate system of any scale for the user. The point (x1,y1) becomes the coordinate of the lower left corner of the screen. The point (x2,y2) becomes the coordinate of the upper right corner of the screen.

x1 must be strictly less than x2.

y1 must be strictly less than y2.

All subsequent user coordinates are interpreted according to this user-defined screen.

Normal return is zero.

DIAGNOSTICS:

Returned -1 indicates x1 is not less than x2 or y1 is not less than y2. An error message is also printed on the terminal screen.

NAME:

scroll - scroll screen

SYNOPSIS:

```
scroll(a,cnt)
char a;
float cnt;
```

DESCRIPTION:

The current displayed picture on the screen is scrolled up or down. Information scrolled off the top of the screen will be scrolled in the bottom and vice-versa.

If a is the character 'd', the direction of the scroll will be down.

If a is the character 'u', the direction of the scroll will be up.

cnt is the number of user-defined y-units (lines) that the picture is to be scrolled. cnt can be no larger than the virtual screen height.

Normal return is zero.

DIAGNOSTICS:

All errors are indicated by negative returned values. The error values and their meanings are as follows:

-1 Indicates the passed parameter cnt was less than zero or greater than the screen height.

-2 Indicates the passed parameter a was not a 'd' or 'u'.

AD-A042 997 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 9/2
THE DEVELOPMENT OF A USER ORIENTED INTERFACE FOR A COMPUTER DRI--ETC(U)
JUN 77 C E MCNEIL, D P HOUSTON

UNCLASSIFIED NL

3 OF 3
AD
A042997



END
DATE
FILMED
9-77
DDC

NAME:

setmode - select control mode

SYNOPSIS:

```
setmode(a,b)
int a, b;
```

DESCRIPTION:

Selects the control mode according to the passed parameters a and b. a represents the control mode as follows:

0 - Alphanumeric
1 - Transverse Data
2 - Raster Data
3 - Complex Data
4 - Graphic Vector
5 - Graphic Plot
6 - Graphic Cartesian
7 - Graphic Element

All flags are turned off if b is equal to zero. If b is equal to one, any flags set in the previous mode are also set with the mode selected by a. All entities displayed subsequent to this call are displayed according to the selected mode. Routines that disregard the selected mode are axis(), block(), inter(), lttr(), plotb(), plotln(), plotpt(), texto(), and vector().

Normal return is zero.

DIAGNOSTICS:

All errors are indicated by negative returned values. The error values and their meanings are as follows:

-1 Indicates the passed parameter a was less than zero or greater than seven.

-2 Indicates the passed parameter b was not equal to zero or one.

SEE ALSO:

ramtek(), bkrnd(), dplwid(), fixpt(), index(), writpn()

NAME:
sixpak - select from six standard switches on tablet

SYNOPSIS:
sixpak();

DESCRIPTION:
Establishes a package of six selection boxes or switches on the Vector General tablet (according to Template 1) and returns the number of the switch selected by the user with the stylus.

DIAGNOSTICS:
Normal return is the number of the switch selected (1-6).
Returned -1 indicates invalid selection.

SEE ALSO:
buildsw(); getsw(); tabsw();

NAME:

size - select letter size

SYNOPSIS:

```
size(r)
int r;
```

DESCRIPTION:

If r is equal to one the standard character size is used.

If r is equal to two the double width character size is used.

Sets the double width flag in the control mode. Therefore size takes the same action as dblwid(1).

Normal return is zero.

DIAGNOSTICS:

Returned -1 indicates the passed parameter r is not equal to one or two.

SEE ALSO:

dblwid(), strout(), lttr()

NAME:

strout - output character string

SYNOPSIS:

```
strout(sp)
char *sp;
```

DESCRIPTION:

Outputs a character string no greater than 100 characters long beginning at the current operating point and continuing on the same line. After completion, an automatic line feed occurs which defines a new current operating point on the next line at the same starting point as the previous line.

sp points to the character string to be output.

Normal return is zero.

DIAGNOSTICS:

Returned -1 indicates the string contained more than 100 characters. In this case no characters will be displayed.

SEE ALSO:

data(), size(), lttr(), texto()

NAME:

strtxy - establish current operating point

SYNOPSIS:

```
strtxy(x,y)
float x, y;
```

DESCRIPTION:

Establishes the current operating point on the screen for subsequent instructions. If the current mode is Graphic Vector with the fixed point flag set, it establishes the base from which the vectors are drawn.

x is a user-defined screen location in x.
y is a user-defined screen location in y.

(x,y) must lie within the user-defined screen.

Normal return is zero.

DIAGNOSTICS:

Returned -1 indicates (x,y) does not lie on the user-defined screen.

SEE ALSO:

fixpt(), screen(), point()

NAME:

symbol - display a special symbol

SYNOPSIS:

```
symbol(sy);
int sy[6];
```

DESCRIPTION:

Draws the symbol that has been defined by the user. Thus the user may define special symbols to augment the standard ACSII character set. Requires the pointer sy to the user-defined array. This routine automatically outputs 12 bytes in the Transverse Data mode.

typical data declaration:

```
int alpha[] {0000000,0000042,
             0052210,0052042,
             0000000,0000000};
```

typical call:

```
tdata(alpha,12,50.0,50.0);
```

SEE ALSO:

tdata()

NAME:

tabcolor - tablet color manipulation function

SYNOPSIS:

```
tabcolor();
```

DESCRIPTION:

Allows for the interactive modification of color tables via the VECTOR GENERAL TABLET. The system prompts for required inputs from the tablet. A template for the tablet is required for input. The tablet must be initialized prior to calling tabcolor (see tabinit).

Typical call:

```
tabinit();
tabcolor();
```

15	16	17	18	HALT
10	11	12	13	14
5	6	7	8	9
0	1	2	3	4
ENTER	CHANGE	TABLE	REVISE	

Template Description

DIAGNOSTICS:

interactive to the RAMTEK raw access terminal.

SEE ALSO:

inter() , savcolor() , resclr() , chnge() , color() , colort() , tabinit()

NAME:

tabdim - redimension the virtual tablet

SYNOPSIS:

```
tabdim(xmax,xmin,ymax,ymin)
double xmax,xmin,ymax,ymin;
```

DESCRIPTION:

Accepts as parameters a maximum and minimum value for the x and y dimensions (in that order) and 'sets' the tablet to these dimensions. Allows the user to view the virtual tablet in any size pattern that conforms to the standard Cartesian grid (minimum x,y values represented at the lower left corner; maximum x,y values located at the upper right corner).

NAME:
tabinit - open the tablet for the user

SYNOPSIS:
tabinit();

DESCRIPTION:
Opens the Vector General tablet and establishes user access via file-pointer fpi. Sets the maximum number of logical switches available to 25. Prints an error message if unable to open the device.

NAME:

tabsw - interactively define switches on the tablet

SYNOPSIS:

```
tabsw(sw)
int sw;
```

DESCRIPTION:

Allows the user to define a set of logical switches on the tablet by utilizing the stylus to select the low set of coordinates, then the high set of coordinates (ie, the lower left corner of the desired selection box, then the upper right corner of the box). The number of switches to be defined is indicated by the parameter 'sw'. The switches are numbered sequentially as they are defined.

Prints the x,y values as selected, along with the switch number assigned after each selection process.

SEE ALSO:

buildsw(); getsw(); sixpak();

NAME:

texto - display a block of text

SYNOPSIS:

```
texto(txt,ctab,col,sz,wo);
int ctab,col,sz,wo;
char *txt[ ] { };
```

DESCRIPTION:

The multiple-entry array of variable length character strings addressed by *txt is displayed on the RAMTEK according to the other parameters :

ctab => desired color table number
col => color entry from that table
sz => desired character size (1 or 2)
wo => 1 if additive write is desired, 0 if not

Normal return is zero

DIAGNOSTICS:

Returned -1 indicates one of the character strings is too long (over 100 characters).
NOTE: any strings over 91 characters in length will "wrap around" the screen on the same line.

NAME:

tblwho - request for current color table number

SYNOPSIS:

tblwho()

DESCRIPTION:

Returns an integer value which is the table number of the color table that is currently being used for display.

NAME:

tdata - outputs transverse data for the user

SYNOPSIS:

```
tdata(trd,by,x,y)
int trd [];
int by;
float x,y;
```

DESCRIPTION:

The variable trd is the name of an array containing the transverse data definition for the image. by is the number of bytes to be passed ; x , y are the starting points for the desired display. The transverse data is processed byte by byte with each byte being displayed directly beneath the last one.

typical data declaration:

```
int alpha[] {000000,0000042,
             005220,0052042,
             000000,0000000};
```

typical call:

```
tdata(alpha,12,50.,50.0);
```

The function will not permit moving off of the screen as it bumps to the right one byte and repositions the user selected y value and continues.

DIAGNOSTICS:

none

SEE ALSO:

setmode() , symbol()

NAME:

terse - selectively erases the screen

SYNOPSIS:

```
terse(era,erb,erc,erd);
float era,erb,erc,erd;
```

DESCRIPTION:

Allows the user to selectively erase a portion of the screen by passing the coordinates of two opposite corners of a box. The area within the defined box will be painted to the background color.

DIAGNOSTICS:

none

SEE ALSO:

strtxy() , point() , setmode() , block()

NAME:

triple - code color from three-number triple

SYNOPSIS:

```
triple(b,g,r)
int b,g,r;
```

DESCRIPTION:

The three input parameters representing the intensities of blue(b), green(g) and red(r) are coded into an integer which is suitable for insertion into a color table entry.

b is between 0 and 15 and represents the intensity of blue desired.

g is between 0 and 15 and represents the intensity of green desired.

r is between 0 and 15 and represents the intensity of red desired.

Normal return is an integer representing the code.

DIAGNOSTICS:

Returned -1 indicates input parameters b, g or r are negative or greater than 15.

SEE ALSO:

chnge(), clrtbl()

NAME:

vector - draw single vector

SYNOPSIS:

```
vector(x1,y1,x2,y2)
float x1, y1, x2, y2;
```

DESCRIPTION:

A vector is drawn on the screen from user-defined screen coordinate (x1,y1) to (x2,y2). The current operating point is left at (x2,y2).

The operation is independent of any mode issued previously but is sensitive to the flags applicable to the Graphic Vector control mode. Undesired results may be obtained if the fixpt flag is set previous to calling vector().

Normal return is zero.

DIAGNOSTICS:

Returned -1 indicates no portion of the indicated line lies on the user defined screen.

SEE ALSO:

bkrnd(), dblwid(), fixpt(), index()

NAME:

writon - additive write

SYNOPSIS:

```
writon(w)
int w;
```

.

DESCRIPTION:

If w is a one, the additive write flag in the control mode is turned on causing subsequent entities in Alphanumeric, Raster Data and Transverse Data modes to write on top of previous entities without destroying them.

If w is a zero, the additive write flag in the control mode is turned off.

Normal return is zero.

DIAGNOSTICS:

Returned -1 indicates the passed integer w is negative

or greater than one.

REFERENCES

[1] "Ramtek GX-100A Programming Manual", Ramtek Corp., 1974.

[2] "User Entry Manual", Cloyes and Muller Thesis, Naval Postgraduate School, 1977.

[3] Ritchie, D. M., "C Reference Manual", Bell Telephone Laboratories, Murray Hill, New Jersey 07974, 1975.

[4] "Programming in C - A Tutorial", Bell Laboratories, Murray Hill, New Jersey, 1975.

[5] "UNIx For Beginners", Bell Laboratories, Murray Hill, New Jersey.

[6] "UNIx Programmers Manual", Bell Laboratories, 1973

[7] Newman, W. M. and Sproull, R. F., "Principles of Interactive Computer Graphics", p. 359, McGraw Hill, 1973.

[8] Sutherland, I. E., "SKETCHPAD - A Man-Machine Graphical Communication System", AFIPS Conf. Proc., Vol. 23, p. 329-346, SJCC 1963.

[9] "Ramtek GX-100B Programming Manual", Ramtek Corp., 1975.

[10] Newman, W.M. and Sproull, R.F., "An Approach to Graphics System Design", Proceedings of the IEEE, Vol. 62, No. 4, p. 471, 1974.

[11] Sutherland, I.E., "Three Dimensional Data Input by Tablet", Proceedings of the IEEE, Vol. 62, No. 4, p. 453, 1974.

[12] Foley, J.D. and Wallace, V.L.,"The Art of Natural Graphic Man-Machine Conversation", Proceedings of the IEEE, Vol. 62, No. 4, p. 462, 1974.

[13] Lucido, A.P., "Software Systems for Computer Graphics", Computer, Vol. 9, No. 9, p. 23, SNAP Aug. 1976.

[14] Carlson, E.D., "Graphics Terminal Requirements for the 1970's", Computer, Vol. 9, No. 9, p. 37, SNAP Aug. 1976.

[15] Smith, L.B., "The Use of Interactive Graphics to Solve Numerical Problems", Communications of the ACM, Vol. 13, No. 10, p.625, Oct.1970.

[16] Sutherland, I.E., "Computer Graphics", Datamation 1976.

[17] Miller, R.B., "Response Time in Man-Computer Conversational Transactions", Fall Joint Computer Conference, 1968.

[18] Nesslage, R.L., "The Design of a User Interface for a Color Raster Scan Graphics Device", Masters Thesis, Naval Postgraduate School, 1976.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
4.	Professor George A. Rahe, Code 52Ra Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
5.	LTCOL R.J. Roland, USAF, Code 52R1 Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
6.	Air Force Institute of Technology CID Wright-Patterson AFB, Ohio 45433	1
7.	MAJ Charles E. McNeil, USAF 200 Holly Dr. Natchez, Mississippi 39120	1
8.	CAPT Dan P. Houston, USMC 608 S. 27th Ct. Renton, Washington 98055	1