AD-A050 241 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 9/2
IMPLEMENTATION OF A USER INTERFACE WITH THE VECTOR GENERAL GRAP--ETC(U)
DEC 77 D C ENDICOTT, N P MARTINO

UNCLASSIFIED NL

1 OF 3
AD A050241

AD A050241

AD No.
DDC FILE COPY

NAVAL POSTGRADUATE SCHOOL
Monterey, California

DDC
RECEIVED
FEB 22 1978
D

THESIS

Implementation of a
User Interface with the
Vector General Graphics Display System

by

David C. Endicott
and
Nathan P. Martino

December 1977

Thesis Advisor: G. A. Rahe

Approved for public release; distribution unlimited.

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)
(6) Implementation of a User Interface with the Vector General Graphics Display System.

5. TYPE OF REPORT & PERIOD COVERED
(9) Master's Thesis; December 1977

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
(10) David C. Endicott
Nathan P. Martino

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
(11) Naval Postgraduate School
Monterey, California 93940

12. REPORT DATE
Dec[illegible] 1977

13. NUMBER OF PAGES
199 (12) 199p.

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)

interactive graphics, real-time processes, graphics, segmentation, multiported memory, UNIX, shared core, programming language

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

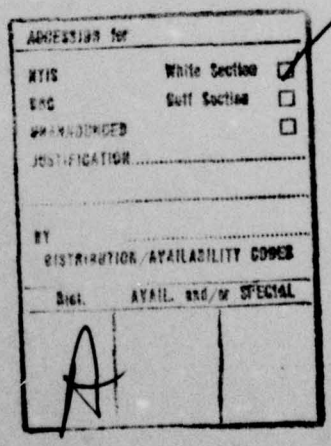
The development and implementation of refresh graphics software for the Vector General Display System on a PDP-11 minicomputer is discussed. Modification and expansion of existing software routines, along with additional software design and development is presented. As an addition to the existing C programming language, a version of BASIC was implemented as a second

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EDITION OF 1 NOV 65 IS OBSOLETE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

251450 01

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

language capable of utilizing the Vector General software. The unsuccessful implementation of a version of FORTRAN with the display system software and the problems involved are discussed. Conclusions and recommendations are offered, and a detailed user's manual is appended.



Approved for public release; distribution unlimited.

Implementation of a User Interface with the Vector General Graphics Display System

by

David C. Endicott
Lieutenant, United States Navy
B.S., United States Naval Academy, 1972

and

Nathan P. Martino
Lieutenant, United States Navy
A.B., University of California, Berkeley, 1971

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
DECEMBER, 1977

DDC
RECEIVED
FEB 22 1978
D

Authors: David C. Endicott

Nathan P. Martino

Approved by: Gus A. Rahe
Thesis Advisor

Second Reader

Chairman, Department of Computer Science

Dean of Information and Policy Sciences

ACCESSION for
NTIS White Section X
DDC Buff Section
UNANNOUNCED
JUSTIFICATION
BY
DISTRIBUTION/AVAILABILITY CODES
Dist. AVAIL. and/or SPECIAL
A

ABSTRACT

The development and implementation of refresh graphics software for the Vector General Display System on a PDP-11 minicomputer is discussed. Modification and expansion of existing software routines, along with additional software design and development is presented. As an addition to the existing C programming language, a version of BASIC was implemented as a second language capable of utilizing the Vector General software. The unsuccessful implementation of a version of FORTRAN with the display system software and the problems involved are discussed. Conclusions and recommendations are offered, and a detailed user's manual is appended.

TABLE OF CONTENTS

I. INTRODUCTION

The Vector General Graphics Display Unit in conjunction with the PDP-11/50 minicomputer provides the Naval Postgraduate School Computer Laboratory with a sophisticated three-dimensional graphics display system. Hardware facilities provide for three-dimensional rotation, translation, and scaling of graphics data. Depth-cueing is accomplished by varying object intensity. A circle/arc generator, in addition to a vector and character generator, further enhances the system by allowing the user to draw arcs and circles with specification of only two or three points. Additionally, external devices in the form of an alphanumeric keyboard, lighted function switches, control dials, trackball, joystick, and lightpen provide interactive control of the display presentation.

Integral to the operation of the Vector General is the display controller. The display controller is a processor which receives data from a display list, decodes the data into graphic commands, and activates the appropriate generator to output the desired information. The construction and maintenance of the display list is accomplished through the use of the Vector General Graphics Library. The library functions as the interface between the user and display controller. Simple primitives are provided allowing the user to build the display list. Additionally,

the software allows the display controller to communicate to the user information obtained from external devices which may be used to programatically control the display presentation.

Previous thesis work by Thorpe [11] [12], Visco [15], and Stankowski [10] provided fundamental graphics software and should be consulted for original design considerations. The objective of work described in this thesis was to review, modify, and extend previous work to provide more useable graphics software for the Vector General Display System. To this extent certain goals were set. All previous work was to be reviewed, revised and updated. A significant portion of time was to be spent in testing and debugging previously untested routines. The design and implementation of software to support the trackball, joystick, and lightpen facilities was to be undertaken. Extension of existing routines to simplify their application had to be accomplished. Expansion of application languages which support vector graphics to include BASIC and FORTRAN was seen as a most significant goal of the research. The culmination of the study was seen to be the compilation, testing, and evaluation of all existing and newly generated software, and the incorporation of all relevant information into one user-oriented reference manual.

II. BACKGROUND

As most of the work discussed in this thesis concerns modifications and extensions to previous thesis work, a description of prior work is in order. Thorpe and Raetz [11] [12] designed the original data structures for constructing and maintaining the display list. The display list was constructed by a series of subroutine calls which produced the correct sequence of graphic commands resulting in a picture generation.

The display was constructed on three hierarchical levels: picture, object, and element. An element is the smallest picture segment that can be referenced. It consists of a sequence of move, line, arc, circle, and character commands. An element must be associated with an object before it can be displayed and an object may comprise a maximum of 10 elements. A picture refers to all the data displayed on the screen and may consist of up to 10 objects.

Visco [15] was concerned with the design of a multi-processor system to support real-time interactive graphics. Due to the high priority of direct memory access (DMA) requests necessary to support refresh graphics, the multi-processor configuration was envisioned as a solution to the DMA bus tie-up problem. Previous work done by Thorpe and Raetz was modified by Visco to conform to the envisioned

system.

Stankowski [10] expanded upon previous work by allowing the user to construct elements and objects with alphanumeric names replacing the previous numbering convention. The object "TREE" could now contain elements "branch1" through "branch10". Additionally, Stankowski designed a number of routines for the user to more easily construct a display list.

O'Dell [5] accomplished the implementation of a segmented-process memory management UNIX operating system that had been previously designed by Emery [3]. Additionally, design and implementation was done in a dual-ported memory environment where designated portions of core were sharable regions. O'Dell's work helped lay the groundwork for a more generalized system in which additional languages could support graphics programs.

Previous paragraphs have presented work completed prior to the commencement of research reported in this thesis. The adequacy of the interface between the user and Vector General was still in doubt. Software for the trackball, joystick, and lightpen facilities was missing and routines for other external devices were not readily useable. Although routines were present to construct and maintain the display list, the majority of them had not been tested. Compatability with O'Dell's SSUNIX and expansion of available languages beyond the C language had yet to be accomplished. With this in mind, a description of the modifications and enhancements to the system is presented in

the following sections.

III. LANGUAGE EXPANSION

A. GENERAL

In reviewing the Vector General Display System and the Naval Postgraduate School Graphics Laboratory, it was noted that most of the software was written in C programming language. This circumstance demanded that any further language introduction would require compatability with C and the UNIX PDP-11 operating system. Two languages became the focus of attention due mainly to their wide use and apparent ability to call C subroutines. Versions of BASIC and FORTRAN were chosen as the languages to attempt implementation with the Vector General software. As work progressed on these languages, strengths and weaknesses were noted for both that could effect further development and use throughout the Graphics Laboratory. The following two sections will explore each language, their origins and difficulties in development, and implementation.

B. BASIC

1. Background

In previous work, Robertson [9] implemented an Extended BASIC compiler. BASIC (Beginner's All-purpose Symbolic Instruction Code) was originally developed at Dartmouth College and has undergone many changes to suit user needs.

The extensions implemented by Robertson include such items as string manipulation, sequential access to external files, multi-dimensional arrays, logical operators for numeric and string quantities, and the ability to call C subroutines.

The Extended BASIC compiler, hereafter referred to as simply BASIC, is not a purely interpretive compiler. It generates an intermediate code (PDP-11 assembly language) from the BASIC source program. BASIC object modules and C subroutine object modules are able to be linked together. One final link and load operation provides the ultimate interface between the two different language modules.

The first step in understanding BASIC was to follow the operation of the shell command which controls the compilation, assembly, and linking. This shell command, LBAX, is a user-accessible executable command on the Naval Postgraduate School's UNIX PDP-11 file system. The command goes through three phases of operation, all of which depend on the option codes included (switches set) when the command is issued. The first phase is the compile phase. This phase compiles the BASIC program source code and emits a PDP-11 assembly language file as the output. The second phase causes the assembly of this assembly language file using the UNIX assembler command "as". This assembly emits an "a.out" file for use in a final linking and loading phase. The final phase uses the UNIX system loader command, "ld", to link and load all necessary object modules. The LBAX shell command automatically loads the respective BASIC and C libraries to reconcile all undefined references. The

output of this final link and load phase is an executable "a.out" file. Within this final stage of the LBAX command, an option is invoked to regulate the UNIX loader. This "-X" switch option causes all C internally generated labels to be disregarded while retaining only those symbols local to the routines themselves. The same switch is also used in the loading phase of the C compile shell command "cc".

Provisions in the compilation phase of a BASIC program ensure the emission of the proper assembly language code that allows interaction between the BASIC program and the C subroutines. Prior to the actual call to a subroutine, the "extern" statement causes the declaration of C subroutine and the designation of the type and number of parameters to be passed. The call to the subroutine is done by the "call" statement. The name of the subroutine and the actual parameters passed are included within this statement. Provisions were also made to allow the user to assign return values from the subroutines to predefined variables.

Initial investigation into Robertson's BASIC compiler proved that the compiler has the capability to allow BASIC programs to call C language subroutines. Robertson's Extended Basic Users Manual should be consulted for further details concerning usage.

2. Testing and Implementation

Robertson indicated that Extended BASIC could interact with subsystems presently used at the Naval Postgraduate School Graphics Laboratory. Implementation of this language

with the Vector General software was never accomplished by Robertson, but the basic feasibility of implementation did exist. The first tests attempted in this development were conducted to determine the correctness of calling C subroutines from the BASIC program. Problems arose in the passing of floating point parameters. In order for such parameters to be passed and aligned correctly within core for the C subroutines, the values expected by the C subroutines had to be declared as double precision variables within the calling program. C language conventions expect both floating point and double precision variables to be stored in 64 bit fields. By declaring the floating point parameters as double precision within the "extern" statement of BASIC, the problem was overcome. Upon the completion of successful parameter passing between BASIC and C subroutines, continued testing was undertaken to use BASIC with the Vector General Library in drawing pictures on the Vector General display.

Since development of the Extended Basic compiler and its accompanying software, the Graphics Library was physically relocated within the UNIX file system. Many of the special device libraries were also renamed. The LBAX shell command had to be updated to reflect these changes. An updated copy of the source and executable versions of the special device libraries were placed on the system for general use.

In preparing a BASIC program for execution envolving the Vector General display, differences were noted in the format of BASIC and C programs. The C program contained, as its

first program statement, an "include" statement. Thorpe [11] and Stankowski [10] required this statement to load the initilized display list in the proper sequence. The position of this statement in the program ensured that upon compilation, the display list would be located at the beginning of the process's data segment in core space. The start of the display list could be located for refreshing by knowing the position of the process's data segment.

The compilation technique mentioned in the preceding paragraph is an important design item in Visco's Vector General device drivers [15]. When designing the device drivers, Visco had to ensure that the start of the display list could be easily located. He also wanted to ensure that with a sharable core region, the only segment of the process located within this region was the data segment. By compiling a C Vector General program with a "-i" switch set, the file generated for execution is designated a 411 filetype. This executable file is distinguished in that the text and data segments reside in separate address space [8]. Upon execution, the start of the data space is aligned at the beginning of a 32 K word boundary in core. The refresh cycle is accomplished by repeated execution of the display list which begins at this 32 K word boundary.

Two major problems had to be overcome in order to use the Vector General display with a BASIC program that called C subroutines. Unlike C, BASIC made no provisions for an "include" statement. BASIC would have to cause the loading of the initialized display list by some other means. A

concatenation of an initilization list could not be appended to the beginning of the BASIC program because of the failure of the language to recognize certain key characters within the list. Such characters as the "←" or underscore character, often used within C variables, are unrecognizable and return error conditions. A decision was made to call a dummy C subroutine which would cause the loading of the initialized display list in the proper sequence. Such a change demanded a modification of the technique by which the Vector General device driver locates the display list data. Visco's temporary drivers had to be modified to find and communicate the offset of the display list from the beginning of the data segment.

The second major problem that was encountered in development of BASIC was the inability of BASIC programs with C subroutine calls to be designated as 411 filetypes upon compilation. The standard file emitted after a BASIC compile was a 407 filetype, text and data being mixed within the file. No provisions were made in the LBAX shell command for inclusion of the "-i" switch to split instruction and data space (I & D space). When the loading phase of the shell command was changed to include this option, execution of the final executable object module proved unsuccessful. As no provisions were made in the compiler to cause data and text instructions to be placed on separate stacks, the passing of various parameters to C subroutines was unable to be accomplished correctly. It was realized that changing the BASIC compiler to emit the proper intermediate assembly

language code was beyond the scope of the original intent of this research.

A different approach was taken in an attempt to overcome the problem. This approach involved implementing a newly constructed UNIX operating system which used a segmented-process memory manager in conjunction with provisions for shared core regions. The shared core regions were recently installed in the hardware. O'Dell [5] implemented such a system called SUNIX for the Naval Postgraduate School Computer Laboratory. One version, SSUNIX, was implemented for the PDP-11/50 processor for use with the graphics display equipment. Although not operational, this hybrid UNIX operating system was planned to be implemented in the computer laboratory. In discussions with O'Dell, SSUNIX was described as a solution to BASIC's problem of intermixing text and data instructions. SSUNIX could distinguish and properly execute either the 411 or 407 filetype processes within the shared memory region. The generalization of the Vector General device drivers and the proper interface between the drivers and the SSUNIX system would be the final items to reconcile in completing the implementation.

Through much testing and debugging, BASIC was implemented as a second high-level programming language for use with the Vector General Graphics Display System. Portions of SSUNIX had to be modified to effect this goal. All documentation for changes to SSUNIX and the Vector General device drivers are discussed in following sections. The use of BASIC as a Vector General programming language is

presented in Appendix A of the Vector General User's Reference Manual.

C. FORTRAN IV-PLUS

1. Background

Much of the work in attempting the implementation of F4P (FORTRAN IV-Plus) with the Vector General software was similar to work accomplished in the BASIC implementation. F4P did require several different techniques for development. More development work is needed before F4P can become completely integrated with the Vector General software.

F4P was developed for the Digital Equipment Corporation [2] by the Commercial Union Leasing Corporation (CULC) to be implemented under the UNIX operating system. The Naval Postgraduate School received executable versions of the compiler but had no access to the source code for it or associated software. This associated software included files for converting regular object modules into F4P object modules, files for placing F4P object modules into library format, and files for linking F4P object modules together into an executable object module. Source code and documentation were not available during the development time frame; this lack of documentation and source for F4P and associated routines made it very difficult to completely develop F4P's use of the Vector General software.

The task of implementing F4P to call C subroutines

proved to be quite different than that of the BASIC and C integration. Such an implementation will require more research and testing before it is fully accomplished. Two important items of imformation were known about F4P and its associated software. F4P uses "DEC standard" object modules; these are quite different than the C object modules. Also, prior to use, C object modules would have to be converted to F4P readable object modules.

Two software items were available which were advertised to allow F4P programs to call C programming language subroutines. The "conv" command could be used to convert a C object module into an F4P readable object module. With all current Vector General software written in C, separate libraries made up of the converted C object modules would have to be constructed. The "callc" routine was the second item designed to complete the interface between the F4P program and the C subroutine. This routine stacks up parameters for C subroutines, executes the external C subroutine, and returns to the calling F4P program.

Differences that did not exist in the BASIC implementation were encountered in the calling of C subroutines by F4P programs. All C subroutines had to be separately compiled by the C compiler into object modules prior to conversion. The F4P "linker" prohibits concatenated subroutines from being compiled together and properly linked with the calling F4P object modules. Also, as previously mentioned, the C subroutine object modules had to be converted to F4P readable object modules by the "conv"

command.

The software that would allow the communication between F4P and C subroutines is the "callc" routine. The source for this routine is an M11 assembly language routine which had to be assembled by the Macro assembler. This Macro assembler is one of the executable files which was included in the associated software supporting the F4P compiler. The resulting "callc.obj" and a seldom used "callcv.obj" modules provide the user with the facility to place the parameters to be passed to the C subroutine in a stack and to jump to the C subroutine which is called. The completion of the subroutine allows control to return to the F4P calling program. The "callc.obj" module would have to be included in the link phase of all the modules in order to obtain an executable F4P file. An important feature of the "callc" routine is that it works with pointers to the parameters rather than the parameters themselves. This C parameter representation necessitated changing the C subroutines to expect pointers to the parameters instead of the values of the parameters.

In dealing with the requirement for separate compilation of subroutines and the call by name parameter passing, a separate Vector General library was needed to accomplish the task. The "libr" command would have to be utilized to place the modules in a single library similar to the archive command "ar" within the C language. Each Vector General Library subroutine had to be updated to reflect the passing of L-values and then recompiled. After conversion of the

object modules into F4P readable object modules, the modules would have to be placed in a new F4P Vector General Library. The following section will discuss the testing and attempted implementation of F4P as a third language capable of using the Vector General software.

2. Testing and Implementation

After general tests were conducted and proved the ability of F4P to correctly call C subroutines, testing was conducted on a simple F4P program that would use the newly constructed Vector General Library of F4P object modules. Two non-fatal errors continued to arise in the testing runs. The two C library subroutines, "ecvt" and "fcvt", were described as multiply defined in the error listing. Upon review of the source code listings for the two items, it became evident that four different assembly language subroutines were involved: "←ecvt", "←fcvt", "ecvt", "fcvt". F4P does not recognize the "←" or underscore character, but unlike BASIC, it disregards the symbol entirely. Through continued testing, the non-distinction between the two similar pairs of subroutine names was found to be no detriment to program loading and execution. Correction of this problem would require rewriting the C subroutine calls.

A second error of some significance was found while testing proceeded. According to C programming language documentation [7], C allows a user to assign a floating point value to an integer variable. Such an assignment performs truncation towards zero on the floating point value

and assigns the whole number value to the integer variable. An inconsistency was found in the methods that C and F4P use in the truncation of floating point numbers.

The converting command "conv" was found to change this basic coding sequence to a sequence which placed the truncated floating point value into a long integer with a length of 32 bits. Such a change would, in effect, cause only the upper 16 bits of information to be used. To remedy the situation, without being able to physically update the source code of the convert routine, all instances of such an operation would have to be located and corrected within the C subroutine. Each instance was changed to reflect that the integer variable was a temporary long integer variable. This was immediately followed by the long integer variable being assigned to a 16 bit integer. The change was proven effective and was instituted in the F4P Vector General Library for the "vgabscale", "vgrelscale", and "vgpscal" routines.

After continued testing, one final, fatal problem was discovered with the interaction between the F4P and the Vector General software. When Vector General subroutines are converted into F4P readable object modules, certain switches are set by default. These switches govern the overlay use in the "linker" phase and the global recognition of different segments which are changed to "psects". It becomes possible for more than one copy of global variables to exist. In checking the display list located in core, two separate display lists were found to exist, each containing

information not held by the other. Information necessary for refreshing the display list was inaccessable to the device drivers. In short, one complete updated display list did not exist.

The inability to implement F4P as the third language capable of utilizing the Vector General Graphics Display System was not the result of any one item. The lack of source code for the software, poor documentation on F4P accompanying software, and inherent differences in the structure of C and F4P were all contributing factors. The implementation of F4P in the Vector General environment should be possible by obtaining proper documentation. The final hurdle to overcome would be the problem of multiple copies of the display list. Recommendations for future work in this area will be made in the final chapter of this thesis.

IV. GRAPHICS SOFTWARE

Modifications and enhancements to graphics software fall within three categories: external device interfacing, display list supporting software, and general system improvements.

A. EXTERNAL DEVICE INTERFACING

The external devices may be used to programatically control the display presentation. Alphanumeric, analog, logical, positioning, and lightpen data may be returned to the user's program through the graphics routines.

1. Keyboard

Alphanumeric data entry is accomplished through the Vector General keyboard. Depression of any key enters an 8 bit ASCII character code into the keyboard register and sets the priority interrupt request indicating a character has been entered. Upon executing the interrupt handler, the character code is placed in a keyboard buffer to await a call from the user's program. The "cget" routine was designed to return the data entered from the keyboard. The "cget" routine buffers the data until a carriage return is received, whereupon the entire alphanumeric string is made available to the user's program. The buffering of input data was designed to allow erasing of erroneous data. As no

data is available to the user's program prior to a carriage return, incorrectly entered data may be removed by repeated control-a keystrokes.

The "cget" routine solved the requirement for returning alphanumeric data strings to the user's program. Determination of integer and floating point numbers, however, presented an additional problem. Recognition of integer or floating point numbers from a character string requires the conversion of ASCII data to numerical data. The system-resident C library contained two routines, "geti" and "getf", which accomplish the desired ASCII to integer or float conversions. Use of the system routines was judged more desirable than inventing new software.

Two new routines, "fget" and "iget", were written which set condition flags and pass control to the system "getf" and "geti" routines to accomplish the conversions. A new "getchar" routine was written to replace the system-resident "getchar" routine. Upon executing either the "getf" or "geti" routine, a call is made to the new "getchar" routine to obtain a character. Normally, the default input device would be interrogated for the desired character. However, due to the condition flag set in the previous execution of the "fget" or "iget" routines, the "getchar" routine requests the character from the Vector General keyboard. Repeated calls to "getchar" are made by the system "geti" and "getf" routines until a carriage return is received. The routine determines the integer or floating point number it has within its ASCII string and returns that number to

the calling routine. The result is that the user is able to obtain integer or floating point numbers from either the console or Vector General keyboard without an excessive amount of additional software.

2. Function Switches

The 32 lighted function switches provide for logical true/false information to be available to the user's program. Three routines were added to enhance the use of the function switches. The "fsman" routine was designed to return the status of the desired function switch. If physically depressed, a logical value of true or one (1) is returned to the user. If not depressed, a logical value of false or zero (0) is returned.

The "lamp" routine was designed to light or extinguish the lamp of the desired function switch. No dependence on the status of the function switch is inherent in the operation of the "lamp" routine. The routine can be used to light a function switch to indicate that a certain condition has been attained.

The "fstog" routine was designed to further extend the ease in utilizing the function switches. The "fstog" routine creates the illusion the function switches are toggles which flip-flop their logical value as well as their lamp status with each depression of the switch.

3. Control Dials

Analog information is made available to the user through the use of the 10 control dials. Integer values returned from the control dials vary from -2048 to 2047, the maximum obtainable from the 12 bit system register. Floating point values returned to the user through the dial routine, however, vary from -1.0 to 1.0. This range can be easily extended within the user's program. Analog values obtained through the dial routine are useful for adjusting other program variables such as rotation rates and translation speeds.

4. Trackball

Previously mentioned devices have no connection with what the user is primarily interested in, the display screen. The trackball is an external device designed to provide visual positioning feedback to the user of data on the display screen. This visual feedback is obtained by the appearance of a small, blinking cursor on the display screen. Movement of the user's hand across the round, plastic trackball causes the software-generated cursor to move in an appropriate manner.

The display of the trackball cursor is accomplished in a manner similar to other elements. A sequence of commands is executed by the display controller which draws a small cross at the origin of the X-Y axis. Each refresh cycle the trackball's X and Y coordinate registers are updated and used to modify the coordinate axis of the cursor. The

cursor is moved reflecting any user movement of the trackball. To insure cursor display, all cursor generating code is inserted in the display list prior to the code for any other element.

Two routines were designed to be used with the trackball facilities. The "cursor" routine was designed to activate or deactivate the display of the trackball cursor. Activation is accomplished by ensuring that the display controller executes the cursor drawing code. Deactivation is accomplished by the display controller skipping the cursor generating code within the display list. The "posit" routine was designed to return to the user's program the position of the cursor on the display screen. The "posit" routine returns a floating point value of the cursor's position within the defined coordinate system and is not dependent on the active display of the trackball cursor.

5. Joystick

The joystick is an external device similar to the trackball in that it provides visual positioning feedback to the user. Unlike the trackball, the joystick appears to move three-dimensionally. Movement along the Z axis is accomplished by variance in the intensity level of the blinking joystick cursor. As the user twists the joystick knob to move the cursor further back into the display screen, the intensity level of the cursor display is reduced giving the appearance that it is fading into the distance.

Joystick cursor generating commands are inserted and

updated in the display list just like the trackball cursor commands. Both the "cursor" and "posit" routines work equally well for the joystick with the addition that a Z dimension is available for the joystick "posit" routine. Both the trackball and joystick cursors are useful in providing the user with visual feedback to interactively position display elements.

6. Lightpen

While the trackball, joystick, and associated cursors provide positioning information, the lightpen is unique in its ability for pointing or identifying a portion of the display. The lightpen does not write on the screen, but senses light from the display on its photocell. Not all elements drawn on the display screen cause lightpen interrupts. An element's lightpen enable bit must be set with the "lightpen" routine or no lightpen interrupt will occur. Once the enable bit is set and the operator is pointing the lightpen at a portion of the element, the display controller causes the lightpen interrupt to occur.

The objective of the lightpen interrupt handler is to determine what portion of the display caused the interrupt. After appropriate action is taken, the processor is allowed to continue what is was executing prior to the interrupt. To determine what element caused the interrupt each element is given an element number. This element number is loaded in a register just prior to the actual drawing of the element. Upon encountering a lightpen interrupt, the

interrupt handler interrogates the element number register to determine the element causing the interrupt. Each element has a lightpen flag associated with it within the data structures. Upon determining the element which caused the interrupt the interrupt handler sets the appropriate flag and releases the processor to continue what it was doing prior to the interrupt. The interrupt handler was kept as simple as possible to minimize the amount of time the processor must spend on interrupts.

Two new routines were designed to help support the lightpen facilities. The "penhit" routine was designed to return to the user a logical true/false reply. This reply was in answer to whether a particular element or object had generated a lightpen hit. The routine determines which element or object is in question, checks the appropriate element lightpen flag, and returns the results to the user. The routine was made general enough that it could check for a lightpen hit on an individual element, an object, or the entire picture.

The "clrhit" routine was designed to clear individual, groups, or all elements of previous lightpen strikes. The routine resets lightpen flags to the clear position in preparation for identifying new lightpen strikes.

Although the capabilities of the lightpen make it an ideal device for pointing or identifying portions of the display, the inclusion of additional software made the lightpen an equally good positioning device. By including additional draw commands in the display list in the same

area as the commands for the trackball and joystick cursors, a lightpen cursor was made available to the user. The lightpen cursor is an octogon-shaped object which may be moved with the help of the lightpen. The lightpen cursor was constructed with eight elements comprising the eight sides of the cursor. Upon approaching the lightpen cursor with the lightpen, the cursor appears to latch on and follow the lightpen about the display screen.

The tracking of the lightpen cursor is accomplished by determining which side of the lightpen cursor first initiates a lightpen hit. Moving of the coordinate axis of the cursor is then accomplished, minimizing the distance between the center of the cursor and the lightpen. The determination of the side initiating the lightpen strike is accomplished within the interrupt handler. Distinction between the sides of the lightpen cursor and other lightpen hookable elements is accomplished by providing the lightpen cursor with 8 unique element numbers.

As with the trackball and joystick, two routines provide lightpen cursor positioning information to the user. The "cursor" routine is essential in activating and deactivating the display of the lightpen cursor. Unlike the trackball and joystick cursors, the lightpen cursor must be displayed to be updated. The "posit" routine returns the floating point position of the lightpen cursor within the defined coordinate system.

B. DISPLAY LIST SUPPORTING SOFTWARE

In reviewing, revising, and extending existing software, many tests and changes were required. The "arc" and "circle" routines were rewritten to provide for correct manipulation of associated center and endpoint arguments. The "setvector" and "endele" routines were revised to ensure proper termination of display list element blocks and to correct deficiencies in the generation of auto-increment arcs and circles. The "intscale" routine was revised and extended to allow the user to specify whether intensity cutoff plane provisions were to be in effect. Changes to other routines included additional software to inhibit error conditions, and correlating routines to provide optimal use of available display list space. Three routines received particular attention and require a fuller explanation.

1. Remove Routine

As originally designed the graphic system data structures provide for a display list of a fixed length in size. Should all the display list be utilized, no additional data would be displayed. For example, should the first element of the first object the user builds use up the entire display list, no additional elements could be displayed even though only one element of one object had been used. Normally such a situation would not occur; however, should numerous large elements be constructed this same problem, lack of available display list space, could arise. The "remove" routine was designed to help solve the

problem. The user specifies those elements no longer essential to the display which can be eliminated, thereby creating available space within the display list for additional elements.

A somewhat analogous situation occurs within the memory management techniques of most operating systems. Various methods have been devised to recover available memory segments and reduce fragmentation. The importance of recovering unused or fragmented memory space in memory management techniques cannot be overemphasized. The efficiency of the entire system may depend on the judicious use of available memory. As the importance of recovering available display list space for a process already running in memory is somewhat lower than for memory management techniques, a less sophisticated method was designed for the "remove" routine.

Upon receiving a list of nonessential elements from the user, the "remove" routine sequentially processes the removal of each block of element commands. The freed block is burped or bubbled through the display list by moving succeeding element blocks up within the display list. The movement of element blocks within the display list requires the adjustment of address pointers to these element blocks to ensure proper operation of the display. As the location of active element blocks are moved to free unused memory locations, display list refreshing is inhibited during this operation. This procedure, though costly in execution time for a series of elements, was deemed more suitable than a

more sophisticated method which would have cost the user additional memory space. Additionally, this simple technique was preferred because the "remove" routine was not expected to be used so frequently within a program that a momentary delay in execution would be noticed.

2. Printv Routine

As originally implemented, the graphics software provided for only one type of element containing alphanumeric data. The "charele" routine was used to specify the contents of the character string which could contain any of the characters available in the Vector General character set plus a few format instruction codes. Provisions for changing the contents of an element's character string were nonexistent.

A search was conducted to determine the most feasible way of providing for an element which could handle changing character strings and also provide expanded formatting capabilities. The success of the "fget" and "iget" routines in using system routines focussed attention on the system "printf" routine. The "printf" routine converts, formats, and prints its parameters after the first argument, under control of the first argument. The first argument, the format argument, consists of characters and conversion specifications. It was felt that a combination of the "charele" and "printf" routines could expand the use of character strings used within the Vector General Display

System.

In accomplishing the merger of the "printf" and "charele" routines two new routines were written. The "printv" routine contains arguments from both the "charele" and "printf" routines. Positioning and character string initialization arguments are similar to those in "charele" while format and conversion arguments are similar to the "printf" routine. The "printv" routine passes its format and conversion arguments to the system "printf" routine to accomplish the actual converting and formatting of the ASCII character string. As the "printf" routine must call the system "putchar" routine to output characters, a newly designed "putchar" routine was substituted for the system-resident routine to output characters to both the console and Véctor General display screen.

The "printv" routine gives the user a greatly expanded capability in handling character strings. Although the "printv" element requires additional display list space over that of a "charele" element, the expanded formatting and conversion capabilities of the "printv" element, in addition to its ability to be used repeatedly for different character strings, makes it a highly useful enhancement to the display list supporting software.

3. Display Routine

As mentioned previously, an element-object link must be established to display elements on the screen. While both the "remove" and "erase" routines caused the disappearance

of an element from the display screen, no routine was available to activate or deactivate the display of an element without affecting the element-object association. The "display" routine was designed to accomplish such a purpose. By replacing certain graphic commands within the display list with subroutine return commands, the display controller may be made to skip execution of an element's draw commands. This effectively deactivates the display of the element without affecting the element-object associations. Activation of the element is then accomplished by reinserting the original commands in the display list such that the display controller executes the element's draw commands on the next refresh cycle. The "display" routine allows the user to turn an element's display on or off without affecting the necessary element-object associations. Thus the number of routines required to activate or deactivate an element's display is less than if either the "remove" or "erase" routine had been called.

C. GENERAL SYSTEM IMPROVEMENTS

Many of the revisions and extensions to system software previously mentioned could equally qualify as general system improvements as their presence ensures a smoother interface between the user and the display controller. Of particular concern in this category are the areas of system initialization and error diagnostics.

1. System Initialization

The "sysinit" routine is the first graphics routine called within a user's program. The routine was designed to establish links from the user's process to the Vector General display unit so that refresh data could be sent to the display unit and external device information could be returned to the user. The presence of two Vector General display units required differentiation between units. The user specifies the desired display unit with the argument passed to the "sysinit" routine. To insure maximum opportunity for execution of a user's program, software was added that attempts to link a user with the secondary display unit should the system be unable to link to the user's primary choice. The user's process is terminated only when the system has failed in attempts to link the user with both display units.

2. Error Diagnostics

Recognizing that users will make errors, the design and implementation of error detection and recovery facilities was identified as a necessary expense. The price paid for error detection and diagnostics depends upon the sophistication desired. The PL/C compiler is an example of extremely forgiving and diagnostically helpful software. Recognizing that such a system is not cheap in terms of required supporting software, the requirements for a reasonable system of error diagnostics were reviewed.

It was felt that all detected errors should be

identified to the user. Presentation of error diagnostics, however, is as important as the actual error detection. Cryptic error messages with numbers which must be looked up in manuals, do little to help the user diagnose problems. Concise error messages which identify the error, the routine where it occurred, plus give reference to additional information were deemed more suitable and could be obtained with minimal overhead.

The error diagnostic routine was rewritten to provide the user with four items upon error detection. An error type is specified so the user may realize what has caused the error. Additionally, the argument in error and the routine in which it occurred are specified to ensure that the user pinpoints the problem. Should the preceding information be insufficient, the last item provided is an error number with which the user may obtain a detailed explanation of the error from the Vector General User's Reference Manual.

Graceful error recovery could be considered an art. Termination of a user's program upon initial error detection, while easily accomplishable, would tend to not only deflate egos but also discourage potential users. All software routines were designed and implemented to detect, identify, and "gracefully" recover from user errors. Although the detection and identification of errors is fairly standard throughout the routines, the recovery methods vary from routine to routine. Within some routines, default parameters are assumed upon error detection to

ensure continuation of the routine. Other instances may occur, however, where no default parameter could be assumed. In such instances the only form of recovery is to exit the routine and return control to the user's program. With the exception of the "sysinit" routine, no routine will cause termination of the user's program. This was done to ensure that some part of the user's display is presented, providing the user some amount of self-confidence and a starting point for using the error diagnostics to pinpoint remaining errors in the program.

V. MODIFICATIONS TO SSUNIX SYSTEM SOFTWARE

A. GENERAL

Investigation was begun into the development of SSUNIX to further generalize the use of other languages with the Vector General Graphics Display System. As a result of the Extended BASIC compiler's inability to separate the instruction and data segments of an executable file, SSUNIX was implemented on the PDP-11/50 A processor within the Computer Laboratory. After studying the UNIX and SSUNIX systems and reviewing O'Dell's recommendations for final integration with the Vector General software, modifications were made to complete the implementation. Further modifications were made when problems arose in the SSUNIX system of core allocation. The modifications made to SSUNIX software were in two general areas of the system software. Modifications were made to the SSUNIX source code and to the Vector General device driver code.

O'Dell [5] implemented a segmented-process memory management version of the UNIX operating system which was previously developed by Emery [3]. O'Dell also implemented an additional version, SSUNIX, which was designed for a shared memory environment. The UNIX operating system [6], from which the above modified systems were constructed, is a time-sharing system developed at Bell Laboratories. The

UNIX system was currently in operation at the Naval Postgraduate School Computer Laboratory on the PDP-11/50 processors. The notable accomplishment of the SSUNIX operating system is that it aligns a process's data segment or combined data and test segments into a region of shared, dual-ported memory. The process image is then locked in this memory region for execution.

Prior to O'Dell's accomplishments, Visco [15] had suggested that Emery's original design was a promising technique for allocating a Vector General process's data segment into a shared memory region. This dual-ported memory region could subsequently be accessable by a slave processor which would continuously access the display list data by DMA (Direct Memory Access). Visco's slave processor design would alleviate the PDP-11/50 bus tie-up problem which was caused by the high interruot precedence of a refresh cycle. The accessability of the shared memory region by the main processor would provide the real-time requirements for multiprogramming in the refresh graphics environment. Visco believed that by ensuring only data segments would be located in the 32 K word shared core region there would be greater efficiency and less likelihood of having programs too large for execution.

A key factor for the implementation of SSUNIX was its propensity to distinguish between processes where data and text were separated segments in core (411 filetypes) and files where data and text were mixed (407 filetypes). SSUNIX causes both filetypes to be loaded into the shared

memory region and executed alike. SSUNIX proved to be the solution to the problem of executing the BASIC executable object module where test and data reside together. Looking beyond BASIC's problems, SSUNIX is of such a general nature that other user languages can be more easily implemented with the Vector General software. The future integration of F4P with the SSUNIX system should require no further modfications to SSUNIX.

B. MODIFICATIONS TO SSUNIX SYSTEM CODE

Modifications to SSUNIX were made in two general areas: the interface code between the user and the system software, and the correction of the process for allocating core prior to locking the process in the shared core region. These changes were tested and instituted in the SSUNIX system. The nature and explanations for the changes are discussed in the following paragraphs in this section.

The first modification to SSUNIX was the updating of the shared memory region parameters located within the "main" subroutine of the system. SSUNIX is alerted by "getshr" that a shared memory region of core is requested for execution of a process. The first parameter passed to the system is the region of shared core that is requested. These regions of shared core are assigned the identifying numbers 0 (blocks 2000 - 4000) and 1 (blocks 5000 - 6000). Each block of core represents 64 bytes of core space. Region 0 is 32 K words in size while region 1 is 16 K words

in size. Region 0 was originally declared as 1000 blocks in length beginning at block 3000. The size of this first region had to be corrected in SSUNIX's "main" routine. The true size of the region is 2000 blocks in length beginning at block 2000.

The second modification to the assembly language code of "getshr" was the addition of a parameter to be passed to the system. The parameter passed is the address of the first word of the display list. The address represents the offset value of the first word of the display list from the beginning of the data segment. If the text and data segments are not split, the offset is from the beginning of the mixed text and data. This method identifies the first word of the display list to the device driver whether or not text and data were split. If I & D space were split and "vgdata" was included at the beginning of the C program, the offset would be zero indicating that the display list starts at the beginning of the process's data space. If, as in a BASIC process, I & D space were not split the offset would be added with the offset of the data segment from the beginning of the shared memory region.

Prior to the inclusion of the offset address in the "getshr" routine, both "getshr" and "freeshr" had to be introduced into the Vector General Library. The previous calls to "rtime" and "nonrtime" under the UNIX system were changed to "getshr" and "freeshr" calls. The difference was that the shared memory configuration of core required the Vector General processes to request this shared core region

for loading and execution.

In order to use the offset value passed to the system by the "getshr" call, a small routine called "vgdsoff" was added to SSUNIX. When "getshr" is called by the Vector General user code, a new variable "firstwdaddr" is assigned the value of the the offset of the first word of the display list. The "vgdsoff" routine returns the value of "firstwdaddr" to a calling routine in the device driver. The only routine that would call for this value would be "vgdev50.c" which is the PDP-11/50 device driver for the Vector General Display Processor. This offset value passed to the driver code is a key factor in allowing various user languages to use the Vector General software. Without this offset, the ability to generalize the refresh method for different languages would be difficult.

The final modification to the SSUNIX system was the result of a condition that had previously gone unnoticed. With the help of O'Dell, the source of an error in core allocation by SSUNIX was traced to a routine called "malloc". This routine allocates space in core to the process being loaded by first checking the core "free map". The "free map", which is a mapping of free core areas, keeps track of the areas of core which are free to be used for storage of a process. SSUNIX first swaps a process out of core onto disk. The process is then copied back into core in an area not within the shared memory region so it is out of its own way. "Malloc" failed to take into account that a large free core area could exist from just below the upper

address of the shared memory region to the top of available core area. If the area beneath the shared core region was not large enough to store the process, the next free area to be checked would be this large area. This area was unavailable for storage as the set flag "sharflg" indicated the region was in the shared memory region. As the beginning of the area was located in the shared core region, the entire area was marked unavailable for process swapping.

A routine called "splitmap" was designed and implemented to remedy the problem by updating the core "free map" to reflect that the region above the upper boundary of the shared core was available for storage. "Splitmap" is called in the "shalloc" routine just prior to a call to "ceswap". "Ceswap" originally called the "malloc" routine to assign the core area for temporary storage. "Splitmap" changes the representation of the free core region to two free core regions which would be separated by the upper boundary of the shared core region. The area within the shared core region would not be used due to the set "sharflg", while the area above the boundary would be properly allocated.

Other core allocation conditions were reviewed to determine if additional problems could arise. No other problems were noted, and the updated version of SSUNIX was found to function properly.

C. MODIFICATIONS TO DEVICE DRIVER CODE

As previously mentioned, the device driver code for the Vector General was modified to allow the execution of processes with or without split text and data segments. Visco designed the original device drivers, "nvgdev34.c" and "nvgdev50.c", to support the Vector General in a multi-processor mode of operation. The two processors were to interact in refreshing and updating the Vector General display list. Inherent in the design was the assumption that all processes would be of the 411 filetype, where text and data segments are separated. The data segment would be located alone in the sharable core region. In order to discuss the modifications to the drivers, a brief description of each driver is presented.

The "nvgdev34.c" driver is the proposed PDP-11/34 resident device driver for the Vector General. It communicates to the PDP-11/50 by messages that are passed through a hardware interface device called the DR-11-K. The driver is passed several items of important information by the message process: the offset of D space within the shared memory; the use of Vector General numbers 0 or 1; and, the time to begin refreshing the display list. This driver is also responsible for handling the interrupts generated by the Vector General display controller.

The "nvgdev50.c" driver is the PDP-11/50 processor resident device driver that supports the Vector General in the two processor mode. This driver formats and sends

messages to the PDP-11/34 processor through the DR-11-K device. The messages sent by this driver designate the offset of the data space from the beginning of the 32 K word shared memory region to determine where the display list begins in core.

The temporary device drivers, "tvgdev34.c" and "tvgdev50.c", were designed to support the Vector General in a single processor mode. The two drivers were originally written by Visco to simulate and test the operation of the two processor design until the PDP-11/34 processor could be installed. Both tempory device drivers will continue to drive the Vector General until the slave processor is integrated into the system.

The drivers were designed with two goals in mind. The first goal has been discussed before. The PDP-11/34 driver, as the refresh processor, would refresh the display list. The second design goal was that the PDP-11/50 driver had to inform the slave processor of the location of the display list for refreshing purposes. In generalizing the driver code these two goals remained preeminent. The following paragraphs document the exact modifications made to the device drivers currently used under the SSUNIX operating system.

1. Modifications to TVGDEV50.C

Only one addition was made to this driver. Within the "vgopen" routine, a check was made to determine if I & D space was split. If it was not split, indicating a 407

process, a message was sent to the 34 device driver which passed the offset of the first word of the display list from the beginning of the process. The message was distinguished in the 34 driver from other messages by virtue of its sign bit being set. The message was constructed by setting the sign bit for the value returned from the previously developed "vgdsoff" routine within SSUNIX. A return to the 50 driver was made after the message was processed by the 34 driver. The use of the messages in the 34 driver will be discussed in the following section.

2. Modifications to TVGDEV34.C

The functioning of the 34 device driver depends on the information passed to it by the 50 device driver. Several modifications had to be made to the driver to generalize the code for additional language implementation. The first modification was the processing of the offset message passed by the 50 driver in cases where I & D space was not split. This message is received by the "vg34com" routine and is converted to a positive value. A new variable, "sharmemoff", is set to the passed value. The value of "sharmemoff" remains zero if a 411 process is executed because the variable had previously been initialized as zero and no additional message would have been passed to the driver. If the variable had a positive value, indicating a 407 process, the item would be used in the "vgopen" routine located within the driver code. Three separate variables depend on this variable for determining values that the

refresh cycle and the PDP-11/34 memory management operation are based on.

The PDP-11/34 is unable to address items in a user's display list by simply referencing symbol names. It is able to address only the 32 K words of the dual-ported shared memory. By using the memory management hardware and the message information passed by the 50 driver, the correct physical addresses of the display list items can be located in core. The page address registers, when properly set, allow this mapping or conversion of virtual to physical addresses in core [1].

Three variables called "vgpar", "vgreal", and "pp" all involve the use of memory management. The "pp" variable is a virtual address pointer that can be used in addressing any item in the display list which resides in the shared memory region of core. This variable can be used to address up to 4 K words of core which is more than adequate for addressing the several hundred words in the display list that might be addressed.

The result of the modifications made to this device driver was the correct refreshing of both 407 and 411 filetype processes. All other added items within this driver reflect additional interrupt handling capabilities for the external devices of the system.

VI. CONCLUSIONS AND RECOMMENDATIONS

The primary objective of work described in this thesis was the implementation of enhanced user software with the Vector General Display System. This objective was attacked on two fronts. The first approach was to test, correct, and expand the software previously available on the system, which would improve user interaction. The second approach was to implement other user languages, specifically versions of BASIC and FORTRAN, with the Vector General software. Together, the two approaches provided the successful implementation of a package which better supports user interaction with the Vector General Display System. This chapter provides further recommendations to improve the system in the areas of the display list design, language development, and system software implementation.

A. DISPLAY LIST DESIGN

In the present design of the Vector General software, a structure within the display list is utilized to store user commands to be executed on the Vector General display. This structure "vgdlbuf" is of a fixed length, 4 K words of the shared memory. An improvement in this technique of having a fixed length display list could be investigated. One possibility would be the implementation of software to

increase the size of the structure which would utilize the unused portion of the sharable memory region. This software implementation would be useful when no other process needs the shared memory region and the parent process requires additional core space.

B. LANGUAGE DEVELOPMENT

F4P (FORTRAN IV-Plus) was not completely implemented with the Vector General software due to problems encountered with the interaction between the F4P and C languages. It is recommended that investigation continue in reconciling the "overlay" problem caused by C converted subroutines having more than one copy of globally declared variables. Prerequisite to understanding this problem is the acquisition of source code and documentation for the F4P compiler, the "conv" subroutine, and the F4P "linker". Completion of the the F4P and C interface would further expand the utilization of the Vector General Graphics System to users with knowledge of other languages.

The inability of Robertson's BASIC compiler to split I & D space is viewed as a less important problem. SSUNIX and driver design modifications were implemented to overcome execution problems of the BASIC Vector General programs. The expense of BASIC's inability to split text and data segments required the allotment of more shared memory in loading the combined text and data segment. By re-working the compiler to allow the stacking of data and text in

separate data and text segments, more complex BASIC programs could be executed in the shared core region allotted for execution. Additional testing of large BASIC programs could help determine the limitations of loading both text and data segments into the shared memory region.

C. SYSTEM SOFTWARE IMPLEMENTATION

An increasing item of importance to the software developed in this thesis and previous software work is the continuing support of the SSUNIX operating system in the Naval Postgraduate School Graphics Laboratory. It is recommended that increasing emphasis and testing be placed on SSUNIX while running in a day-to-day operational environment to ensure its reliability.

The implementation of the PDP-11/34 processor in a dual-processor mode with the PDP-11/50 processor should be of primary importance for the Vector General Display System to alleviate the current bus tie-up problem. An investigation into the communication requirements between the slave and the master processors should be conducted. The hardware interfaces for the 32K word shared core region and the Vector General display device should also be implemented.

APPENDIX A

USER'S REFERENCE MANUAL

The purpose of this document is to provide the user with the information necessary to execute programs employing the Vector General graphics display. This manual was originally prepared by B. J. Stankowski in June 1976 and revised by N. P. Martino and D. C. Endicott in December 1977.

Section I provides a brief introduction to the Vector General display system, the software support library, and the prerequisites for their effective utilization. Section II describes the Vector General system in more detail, expanding upon the hardware features, system interface, and external device capabilities. Section III contains information on the initialization of the Vector General display system. Section IV describes the display composition, element construction, and picture, object, and element relationships. Section V provides a brief description of the user support routines available in the support library. Finally, Section VI provides information necessary for compiling and executing user programs employing the Vector General graphics display unit.

TABLE OF CONTENTS

I. INTRODUCTION

The Vector General is an interactive graphics display system which has been interfaced with the PDP-11/50 computer. The display interacts with a PDP-11 user by displaying pictorial information on the surface of a cathode ray tube and by accepting information from its external control devices. The external devices consist of an alphanumeric keyboard, 32 lighted function switches, 10 control dials, a trackball, a joystick, and a lightpen.

An interactive graphics software program library has been constructed to interface the user and the Vector General display controller. The software package provides the user with sufficient routines to easily construct and manipulate graphic displays. Although the routines were written in C, the user has the choice of two high-level languages with which to program. BASIC programs may now also interface with the Vector General through the use of the routines in the software support library.

This manual does not attempt to discuss in detail the electronic functions of the Vector General, or its interface with the PDP-11. The purpose of this manual is to instruct a user in the creation and manipulation of pictorial data on the Vector General display. A knowledge of C or Basic is assumed.

II. THE VECTOR GENERAL DISPLAY SYSTEM

A more detailed discussion of the Vector General can be found in the Users Manual for the Vector General Display Unit [12] and the Design Manual for the Vector General Display Unit [11].

A. THE DISPLAY

The Vector General is a cathode ray tube (CRT) display on which a visible pattern can be created by the movement of an electron beam. The electron beam causes a florescent spot to appear on the face of the display tube. The movement of the beam is controlled by a method called random scan, which in effect steers the spot in a straight line between two points on the display screen. The resulting line or vector, combined with others, creates a picture or pattern on the display screen.

To maintain a clear picture on the display screen requires that the pattern be redrawn on the tube repeatedly at approximately thirty to forty times a second. Each repetition is called a frame and the frequency with which it is redrawn is called the refresh rate. If the pattern is not repeated often enough a degradation of the picture will occur on the display screen. This degradation is called flicker.

B. HARDWARE FEATURES

The system has several hardware features, in addition to a vector generator, which greatly extend its capabilites. These include the ability to produce three-dimensional figures, an ASCII character set, and the hardware generation of arcs and circles. Other features provide the hardware mechanisms for the rotation and translation of user specified picture segments. These hardware features are controlled and coordinated by the display controller. The controller is responsible for handling the communications between the user, the external control devices and the display hardware.

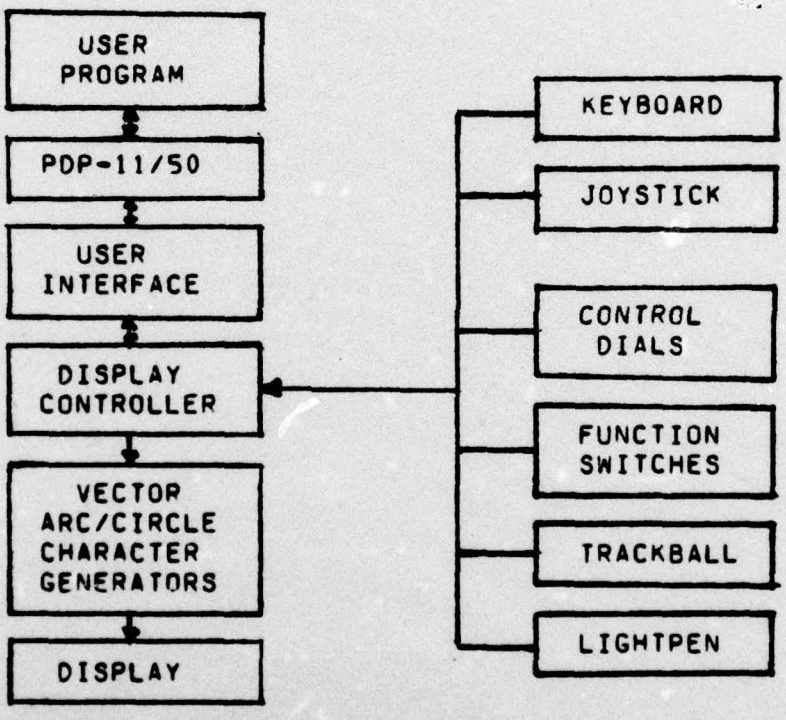
The main purpose of the external control devices is to facilitate user interaction with the display. These devices include an alphanumeric keyboard, 32 lighted function switches, 10 control dials, a trackball, a joystick, and a lightpen.

Additional information on the Vector General hardware can be found in the Graphics Display Reference Manual [13] and the Graphics Display System Technical Manual [14].

C. THE SYSTEM INTERFACE

A Vector General user defines pictorial data and its manipulation within a Basic or C program. The execution of the program causes the Vector General software interface to be activated. It is this software which communicates user requests and receives information from the Vector General

controller. The software passes user requests, in the proper form, to the display controller. The controller will activate the proper hardware generator (ie. character, vector, arc/circle) which will output the desired information on the display screen. The controller will also pass information from the external control devices back to the user via the interface software. This relationship is illustrated in figure A-1.



Interface Relationship with the Vector General
FIGURE A-1

D. INTERFACE WITH EXTERNAL DEVICES

Each of the the external devices communicate directly to the Vector General controller. Information from these devices is returned to the user via the controller and the user interface. The user program may utilize the information

returned from these devices to control program flow. This allows a user to interactively control and manipulate the pictorial information at display time. Specific user routines which activate these external devices and provide a communications channel with the user are discussed in detail in later sections of this manual.

1. Alphanumeric Keyboard

The alphanumeric keyboard allows the user to input information in the form of ASCII character codes. Through the user interface the user can display the information on the Vector General display screen. The data entered from the keyboard can also be returned to the user program for processing.

2. Function Switches

The 32 lighted function switches provide the user with information which can be used to interactively manipulate pictorial data at display time. Each function switch can be assigned specific meaning by the user program. The user interface returns to the user information on which function switches have been pressed. A user program could use this information to selectively rotate, translate or perhaps scale particular picture segments.

3. Control Dials

The 10 control dials provide numeric information to the display controller, specifying the degree to which each dial has been turned. This information, through the

software, can be provided to the user. A user program may utilize the values of the variable control dials in determining the distance or rate at which a portion of the picture may be moved or rotated.

4. Trackball

The trackball is an external device which uses a software-generated blinking cursor to provide visual positioning feedback to the user. The rolling action of the user's hand across the round trackball steers the cursor about the display screen. A routine is available to return the trackball cursor position within the defined coordinate system so the user may programatically use this information. Additionally, a routine is available to activate or deactivate the display of the software-generated cursor. Positioning information may be returned to the user whether or not the cursor is displayed.

5. Joystick

The joystick is an external device, which like the trackball, uses a software-generated blinking cursor to provide positioning feedback to the user. Unlike the trackball which moves about only the X-Y plane, the joystick cursor appears to move in three dimensions. Movement about the X-Y plane is accomplished by pushing the joystick fore-aft, and side-to-side. Movement along the Z axis (perpendicular to the display screen) is accomplished manually by twisting the knob on top of the joystick and visually on the screen by using the depth-cueing facilities

and varying the cursor intensity. Like the trackball, routines are available to return to the user the joystick cursor position and activate or deactivate the display of the cursor.

6. Lightpen

The lightpen, a wand containing a photo cell, can be used to selectively point to different picture segments on the display screen. The interface provides a user program with information on which picture segment was pointed to by the lightpen. A user program can turn the lightpen selectability of specific picture segments on or off. For example, a user might select sections of a picture for erasure by pointing to them with the lightpen.

The software additionally provides for generation of a lightpen cursor. The lightpen cursor is a small non-blinking octagon which may be moved about the display screen with the lightpen. Two speeds of lightpen movement are available. Routines for returning the lightpen cursor position and activation/deactivation of the cursor display are available within the software package.

III. INITIALIZATION

A. INTERFACE INITIALIZATION

The Vector General display system and the graphics software with the PDP-11 must be initialized before any data can be displayed. The initialization routine sysinit must be called before any other routines are utilized. This routine sets all the system default parameters, such as the screen coordinate system.

If for some reason the initialization cannot be completed the user program will be terminated. This error usually occurs because another user is accessing the Vector General.

B. DISPLAY INITIALIZATION

1. Coordinate System

The user can specify a two or three-dimensional cartesian coordinate system. All display coordinate values referenced by the user will be interpreted according to this coordinate system definition. A user may redefine the coordinate scale at any time in a program. The user will define the coordinate system in a call to the routine coordsys.

```
coordsys(dim,minx,maxx,miny,maxy [,minz,maxz]);
```

The routine requires the user to specify if the coordinate system is to be two or three-dimensional and the range of each coordinate. If the parameter 'dim' is two, indicating a two-dimensional coordinate system is desired, the range of the z coordinate can be omitted, and will be ignored if it should be included.

If this routine is not called by the user the default coordinate system will be used. This default system is defined as three-dimensional with the x, y, z coordinates ranging from -100.0 to 100.0. All coordinate values will be interpreted by this default system when coordsys is not called by the user.

As will be discussed later in this manual, both absolute and relative vectors may be utilized in constructing display elements. Should an absolute coordinate be specified which falls outside the defined coordinate system, a value of the nearest coordinate within the defined system will be assumed. Should a relative increment be specified which exceeds one-half the coordinate range (ie: 100 for default coordinate system) a value of zero will be assumed.

2. Picture Scale

The rectangular, 13 by 14 inch, portion of the display screen that can be viewed by the user is called the visible space. The maximum picture space is larger than the visible space, covering an area of 30 by 30 inches. This extra area allows a user to rotate or move part of the picture to the extreme boundaries of the visible space

without any distortion. It also permits limited zooming.

The pictorial data being displayed can be adjusted in size, or scaled, by two different controls. One, the gain control dials on the Vector General display unit allow the user to manually manipulate the picture scale. The second provides scale control within the user's program by calling the routine pscale. This routine is discussed in detail in Section V of this manual.

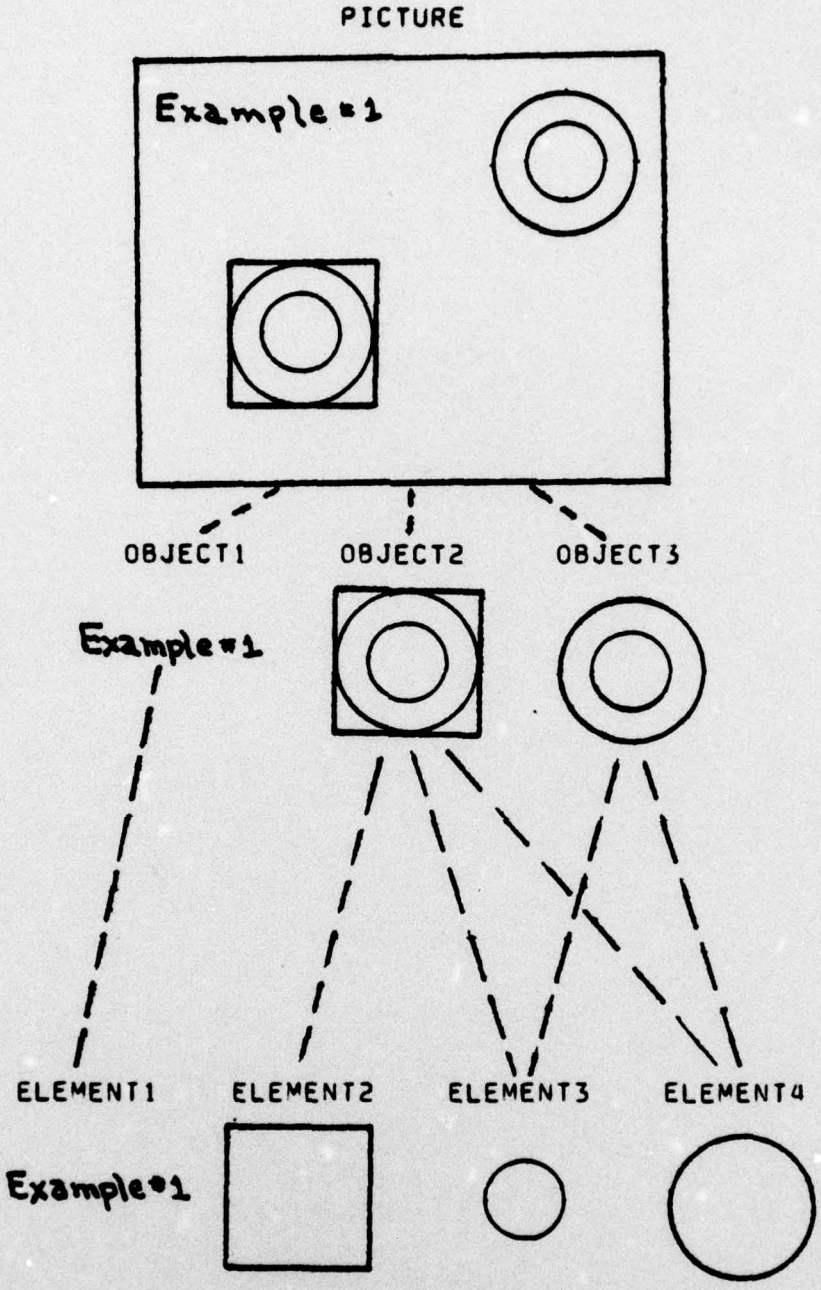
IV. CREATING A PICTURE

A. PICTURE STRUCTURE

All of the information that a user desires to display on the Vector General must be incorporated into the hierarchical picture structure defined by the interface software. The three hierarchical levels are defined as: picture, object, element. These levels specify the underlying structure of the graphical display and determine the operations a user can perform on information associated with each level.

The term picture refers to all of the data that is to be displayed on the Vector General display space. The term element refers to the smallest picture segment which can be independently referenced and changed without affecting the remainder of the picture. Each element, or independent picture segment, has a unique name associated with it. A collection, or meaningful grouping, of elements is called an object. Each object is also labeled by a unique name so it can be easily referenced by a user. Figure A-2 illustrates the relationship of the three levels in the actual structure of a picture.

An element is completely described and named by the user. It can describe either graphical or ASCII character data. An element can be independently added or erased from



Hierarchical Picture Structure
FIGURE A-2

the display screen. It can be caused to blink or be specified as being lightpen selectable.

Each element must be associated with at least one object before it can be displayed on the Vector General. An object may consist of from one to ten elements. A user is responsible for establishing the desired object-element association, and for specifying a unique name for each object. Each user defined object can be independently rotated, or translated to any section of the 30 by 30 inch display space. An object can be added or erased from a picture, scaled, and specified to blink or to be lightpen selectable. Each object's intensity can be varied in order to give three-dimensional objects their depth queuing. These actions, when applied to a specific object, affect every element that has been associated with it by the user.

A picture may contain from one to ten objects. The coordinate scale and picture scale defined by the user affects the entire picture. A picture's coordinate scale can be varied but this action will affect every object defined as part of the picture. An entire picture can also be erased, specified to blink or be lightpen selectable.

A summary of the operations for each level of the hierarchy is outlined in Figure A-3.

PICTURE:

Define picture coordinate system
Picture scale
Picture offset
Erase
Display
Remove
Refresh
Blink
Lightpen selectable

OBJECT:

Translate
Rotate
Scale
Intensity scale
Intensity offset
Erase
Display
Remove
Blink
Lightpen selectable

ELEMENT:

Erase
Display
Remove
Blink
Lightpen selectable

Summary of Operations Associated with Each Hierarchial Level
FIGURE A-3

B. CONSTRUCTING AN ELEMENT

Every element is completely described by the user within an element block. There are two types of element blocks. A draw element block describes graphical information. A character element describes ASCII characters that are to be displayed. Each element is uniquely named and this name

will be used to reference this particular structure.

1. Draw Element Block

A draw element block represents a group of draw instructions that describe a specific structure, or picture segment. These draw instructions include setvector, move, line, arc and circle. A draw element block begins with a call to the routine drawele and is terminated by a call to the routine endele. The draw instructions that are executed between drawele and endele describe the actual picture segment.

The only parameter required by drawele is a quoted character string, or pointer to a character string, specifying the name the user wants to associate with this element. This name will be used throughout the program to reference this element block.

The basic draw element block, and the related draw instructions are represented in the following format:

```
-- drawele("element-name");
|       setvector(vtype,vmode,[inc],[scale]);
|       .
|       .
|       .
|   ---setvector(vtype,vmode,[inc],[scale]);
|   |   move(x,[y],[z]);
|   |   line(x,[y],[z]);
|   |   circle(dir,centx,[centy],[centz]);
|   --- arc(dir,centx,[centy],[centz],endx,[endy],[endz]);
|       .
|       .
|       .
--- endele();
```

The user can select one or more of twelve vector types

in constructing an element. These vector types describe how the coordinate data will be interpreted in drawing a vector on the display screen. The choice of a vector effects the parameters that will be passed in each of the move, line, circle or arc instructions. The user specifies a vector selection by calling the routine setvector.

a. Setvector

This draw instruction must be called immediately after drawele, and may be called any number of times within the element block. Each setvector, and the draw instructions that follow it, comprise a subgroup. The setvector instruction determines the manner in which the line, move, arc and circle instructions in the subgroup will be interpreted, as well as their visual appearance on the display screen. The routine is called with the following parameters:

```
setvector(vtype,vmode [,inc] [,scale]);
```

The parameter vtype specifies which one of the twelve vector types the user wants the following group of line, move, arc and circle instructions to utilize. The parameter vmode indicates the vector mode, or appearance of the vectors to be drawn (ie, solid line, dotted line, etc.). Certain vector types require additional information; this information is specified by the parameters inc and scale.

A brief summary of the twelve vector types is listed in Table A-I.

TABLE A-I

SUMARY OF VECTOR TYPES

NAME	DESCRIPTION
VA	- vector absolute; each coordinate is specified with respect to the origin. Each point(x,y,z) references a unique point on the display screen.
VAX	- vector absolute auto-increment x; every draw instruction causes the y and z absolute values to be updated while x is stepped by a constant value.
VAY	- vector absolute auto-increment y; every draw instruction causes the x and z absolute values to be updated while y is stepped by a constant value.
VAZ	- vector absolute auto-increment z; every draw instruction causes the x and y absolute values to be updated while z is stepped by a constant value.
VR	- vector relative; each x, y, z coordinate value indicates the amount that is to be added or subtracted from the previous absolute coordinate point.
VRX	- vector relative auto-increment x; each draw instruction causes the y and z coordinate values to be incremented by the specified value while x is stepped by a constant value.
VRY	- vector relative auto-increment y; each draw instruction causes the x and z coordinate values to be incremented by the specified value while y is stepped by a constant value.
VRZ	- vector relative auto-increment z; each draw instruction causes the x and y coordinate values to be incremented by the specified value while z is stepped by a constant value.

TABLE A-I
(continued)

INC2- two-dimensional incremental vector; a relative vector that optimizes storage requirements. The coordinate increment values are limited to values approximately 3% of the user's coordinate range.

INCX- two-dimensional incremental auto-increment x; is a relative vector that optimizes storage requirements. The y coordinate value is incremented by a small value while x is stepped by a constant value.

INCY- two-dimensional incremental auto-increment y; is a relative vector that optimizes storage requirements. The x coordinate value is incremented by a small value while y is stepped by a constant value.

INC3- three-dimensional incremental vector; a relative vector that optimizes storage requirements. The x, y, z coordinate increment values are limited to values that are approximately 3% of the user's coordinate range.

A more detailed account of each vector type and the parameters required by setvector are listed in Appendix B. Figure A-4 illustrates the five different vector modes that are available.

VECTOR MODE	PARAMETER VALUE	VISUAL APPEARANCE
line	LN / 00	———————
dashed	DSH / 020	— — — — —
dotted line	DOT / 040	- - - - - - -
end point	PNT / 060	•
dash-dot-dash	DD / 0120	— - — - —
dash-dot-dash	DDD / 0140	— — — - — -

Vector Modes
FIGURE A-4

b. Move

The draw instruction move is used to reposition the beam on the display screen. It will produce no visible line or pattern. The format of the instruction is:

```
move(x [,y] [,z]);
```

The coordinate values x, y, z will be either absolute or relative values. The vector type selected in the preceeding setvector instruction will determine how the value of these parameters will be interpreted.

A move instruction under an absolute vector type will reposition the beam at the absolute coordinate given in the move instruction parameter list. Absolute parameter values which exceed the limits of the defined coordinate system will be assigned a value of the nearest coordinate within the defined system.

A move instruction under a relative vector type will reposition the beam at a location determined by adding the parameter values to the present beam location. Relative parameter values which exceed one-half the coordinate range will be set to zero.

The bracketed values indicate parameters that may be optional. If the user coordinate system is not three-dimensional, the z parameter can be omitted, and will be ignored if it should be included.

c. Line

The line instruction draws a visible line or vector on the display screen. The line is drawn from the present beam locaton to the specified end point. The format of the instruction is:

```
line(x [,y] [,z]);
```

The coordinate values x, y, z will be either absolute or relative values. The vector type selected in the previous setvector determines how the value of these parameters will be interpreted.

A line instruction under an absolute vector type will

draw a line from the present beam position to the absolute coordinate specified in the line instruction parameter list. Absolute parameter values which exceed the limits of the defined coordinate system will be assigned a value of the nearest coordinate within the defined system.

A line instruction under a relative vector type will draw a line from the present beam position to a point determined by adding the parameter values to the present beam location. Relative parameter values which exceed one-half the coordinate range will be set to zero.

If the user has defined a two-dimensional coordinate system, the z parameter can be omitted, and will be ignored if it should be included.

d. Circle

The circle instruction will draw a circle beginning at the present beam location about the center point specified by the user. The difference between the present beam location and the center point determines the radius of the circle. The instruction is used by the following format:

```
circle(dir,centx [,centy] [,centz]);
```

The parameter dir indicates in which direction the circle is to be drawn, clockwise or counterclockwise. The number of parameters required and their values are determined by the vector type selected in the previous setvector instruction.

Circle commands issued under absolute vector types will cause the circle to be drawn about a center point as specified in the parameter list. Absolute parameters which exceed the limits of the defined coordinate system will assume a value of the nearest coordinate within the defined system.

Circle commands issued under relative vector types will cause the circle to be drawn about a center point determined by adding the parameter values to the present beam location. Relative parameter values which exceed one-half the coordinate range will be set to zero.

If the user coordinate system is two-dimensional, the z parameter can be omitted, and will be ignored if included. A circle cannot be drawn by any of the four incremental vectors.

e. Arc

The arc instruction will draw an arc from the present beam location, about the specified center point, to the desired end point. The distance between the starting point and the center point determines the radius of the arc being drawn. The instruction format is :

arc(dir,centx [,centy] [,centz],endx [,endy] [,endz]);

The parameter dir, gives the direction the arc is to be drawn, clockwise or counterclockwise. The arc's actual center and end points are determined by the vector type selected in the previous call to setvector, and by the

supplied parameter values.

Arc commands issued under absolute vector types will result in the center and end points being absolute coordinates as given in the command parameter list. Absolute parameters which exceed the limits of the defined coordinate system will assume a value of the nearest coordinate within the defined system.

Arc commands issued under relative vector types will result in the end point of the arc being determined by adding the endx, endy, and endz values to the present beam location. The center point is then obtained by adding the centx, centy, and centz values to the end point. Relative parameters which exceed one-half the coordinate range will be set to zero.

If the user coordinate system has been defined as two-dimensional, the z parameters can be omitted, and will be ignored if included. An arc cannot be drawn by any of the four incremental vectors.

If the distance from the starting point of the arc to the center point, and the distance from the end point to the center point are not equal, the resulting arc will contain a straight line. The straight line results from the arc generator trying to compensate for the two different distances to the center point. One distance will be used to determine the radius of the arc; the arc will then be drawn using this radius. The arc will stop at the systems new defined end point and a straight line will be drawn to the end point that had been specified by the user. The resulting

arc appears in the following form:

2. Character Element

A character element represents ASCII character data that is to be incorporated into the picture structure. A user can specify a character element containing ASCII, special Vector General characters and formatting symbols to be displayed on the Vector General display screen. A user can select from four character sizes and has the option of selecting a slanted character set. The text can be displayed horizontally or vertically on the screen. The vertical position causes the characters to appear as if they were on a page that had been rotated ninety degrees counter clockwise. The user can select the position on the screen where the string is to begin, or can output it relative to the present beam position.

Each character element is given a unique name by the user. This name will allow the user to easily reference each character element. There are two routines which can produce character element strings.

a. Charele

The charele instruction produces a character element string of a given length. The charele character element generally requires less display list space than the

printv character element and is designed for elements which do not change. The character element is represented by the following format:

```
charele("element-name",string,size,wdir,slant,x,y);
```

The parameter string can be either a quoted string within the parameter list or a pointer to a character string or array. The character string will begin at the point(x,y) or can be output relative to the present beam location by replacing the x and y parameter with the constant VGREL. For example, the following character element, when linked to an object, would be output relative to the present beam position:

```
charele("element-name","Now is the time",SZ4,HOR,SLNT,VGREL);
```

A summary of the character element parameters is presented in Table A-II. The character set available on the Vector General is illustrated in Figure A-5. All of the Vector General characters can be represented within a character string. The special formatting symbols and special Vector General characters are listed in Appendix B.

TABLE A-II

SUMMARY OF CHARACTER ELEMENT PARAMETERS

size: specify character size

SZ / 00 - use previously defined character size

SZ1 / 0100 - set size to 100 columns by 60 lines

SZ2 / 0120 - set size to 81 columns by 41 lines

SZ3 / 0140 - set size to 60 columns by 30 lines

SZ4 / 0160 - set size to 32 columns by 16 lines

wdir: write direction

HOR / 00 - write characters horizontally

VER / 0200 - write characters vertically

slant: specifies regular or slanted characters

SLNT / 00 - slanted characters

NSLNT/ 01 - regular characters

Character Set
FIGURE A-5

b. Printv

The printv instruction provides for a formatted output character element string of variable length. Maximum character length for a printv character element string is 100 characters. The printv character element is designed for output character strings which change frequently. The printv character element is represented by the following format:

```
printv("elename",size,wdir,slant,xpos,ypos,"format"
                    [ ,arg1,arg2,.....,arg10 ] );
```

The first six parameters are identical in use to the charele format in initializing and positioning the character string. The printv character string cannot, however, be positioned relatively but must be supplied with both X and Y absolute coordinates. The format parameter and ten optional arg parameters are used as in a standard printf statement in formatting output.

An example of the use of the printv character element would be as follows:

```
printv("RESULT",SZ4,HOR,SLNT,5.0,0.0,"result = %d",
                                        result);
```

and could be changed later in the program to:

```
printv("RESULT",SZ3,HOR,NSLNT,-10.0,15.0,"score = %d",
                                          score);
```

C. LINKING ELEMENTS TO OBJECTS

Draw and character elements represent the smallest picture segment that can be independently referenced by the user. In order to display an element it must be associated with at least one object. An object represents the smallest entity that can be displayed independently on the Vector General screen.

1. Object Routine

Each object is given a unique name so it can be easily referenced by the user. Elements can be linked to an object at one time or by several different calls to the routine called object. An object can consist of one or more elements. A specific element can be linked to several different objects, or may be linked to one object several times. This object-element association or linking is established by the routine object. This routine has the following format:

```
object(num,"object-name","element-name",...,"element-name");
```

The parameter num indicates the number of elements that are to be linked to the named object by this call. The elements are referenced by the names specified by the user in the preceding drawele, charele, and printv routines. Since each object is associated with a unique name, a second call to object, with a duplicated object name, will cause the elements to be added to the object first associated with

that name.

When an element is linked to a specific object several times, the user can no longer reference a specific occurrence of this element within the object, for example:

```
object(3,"Tree","branch","branch","branch");
```

In this case the element branch has been linked to the object tree three times. When displayed, element branch will appear three times. Now, however, the user cannot uniquely reference a specific occurrence of the element branch. If the element called branch was selected to be erased, the first occurrence of branch would be erased from the screen. In many instances it may be desirable to uniquely reference each occurrence of an element within a specific object. This can be accomplished by associating several unique names with an element. The routine copyele provides this capability.

2. Copyele Routine

This routine allows a user to assign several unique names to a specific element structure. In this way, an element structure can be associated with an object several times and each occurrence can be uniquely referenced. The routine is represented by the following format:

```
copyele(num,"element-name","copy1-name","copy2-name",...);
```

The parameter num indicates the number of additional names a user wishes to associate with the named element structure. The "element-name" refers to a previously defined draw element or character element block. Each of the "copy-names" must be unique.

Now reconsider the previous example. If the element branch is associated with two other unique names and these three names are used in the object-element linking, each occurrence of the structure can now be uniquely referenced by the user. The following two statements will accomplish this task.

```
copyele(2,"branch","branch1","branch2");
object(3,"tree","branch","branch1","branch2");
```

V. USER SUPPORT ROUTINES

After the user has incorporated all data that is to be displayed into the desired picture structure, it can now be manipulated and transformed. The following user routines provide the means to manipulate the picture, objects and elements that have been created by the user.

A description of each user routine, calling format and error diagnostics are included in Appendix B.

A. PICTURE REPRESENTATION

1. Blink

The display blink mode can be set for the entire picture, single object or for any number of elements associated with a specific object. Modifying the blink mode of an object affects all the elements associated with that object. The routine blink will turn the blink mode on or off for the specified picture segment.

2. Erase

The entire picture, a single object or any number of elements associated with a specific object can be erased from the display screen. The picture segments that are erased from the screen can be redisplayed by again establishing the desired object-element association. This is accomplished by calling the routine object, as described

earlier. Erasing an object will affect all elements associated with the named object.

3. Remove

The routine remove provides a user with the ability to release the memory locations associated with a specific element structure. The user can remove the picture and release all the memory locations that have been used to describe all the existing elements. Additionally the picture will be erased from the screen. Each element that is removed from memory can no longer be referenced or linked to objects. An individual element can also be removed from memory. This will cause every occurrence of the element to be erased from the screen and the memory locations associated with the element's description will be released. The user can specify an element for removal by either its original name or by any of the copy names associated with it. Remove results in the elimination of all occurrences and all copies of an element from the display screen. An element that has been removed can be redisplayed only by reconstructing the element block and by again establishing the desired object-element association.

4. Display

The display of the entire picture, an object, or individual elements may be activated or deactivated with this routine. Object-element associations are not affected with this routine as they are with erase and remove.

5. Refresh

The default display refresh rate is 40 Hz and may be altered from 15 to 120 Hz with this routine. High refresh rates ensure no flicker but may not allow enough time to execute all desired draw commands before refresh begins again. Low refresh rates allow more draw commands but with an increase in flicker.

6. Offset

Picture coordinate axis positioning is normally centered on the display screen but may be altered with this routine. Altering of the picture coordinate axis will affect all objects in the picture.

7. Pscale

Modification of the picture scale may be accomplished by calling this routine. All objects within the display are affected by a modification with this routine.

8. Intensity Offset

The routine intoffset allows the user to vary the intensity level of a three-dimensional object, or impose a screen cut-off plane for the named object.

9. Intensity Scale

The routine intscale allows a user to vary the intensity of a three-dimensional object. This provides the depth-cueing or shading for a three-dimensional object.

B. PICTURE MANIPULATION

1. Rotate

This routine allows a user to rotate an object about the x, y, and z axis. The rotation of an object affects all elements associated with the named object. Arcs, circles and characters are always drawn in a plane parallel to the screen, and are rotatable in a three-dimensional coordinate system about the z-axis.

2. Scale

The routine scale allows a user to independently scale any object of the picture. All elements associated with the object will be scaled by the specified scale factor.

3. Translate

The routine trans allows the user to move an object anywhere in the display space. The object and all its associated elements can be moved in the x, y, and z planes. Continually translating an object by very small increments will cause it to appear as if it is moving across the display screen.

4. Place

The place routine allows the user to place or position an object anywhere in the display space. Specification of an absolute coordinate value will cause the coordinate axes of the object to be centered on the specified value.

C. EXTERNAL DEVICES

1. Cursor

This routine allows the user the option of activating or deactivating the display of the trackball, joystick, or lightpen cursor.

2. Posit

This routine will return to the user a floating point value of the position of the trackball, joystick, or lightpen cursor. The cursor need not be displayed for this routine to work.

3. Keyboard

a. Cget

The routine cget allows a user to receive and output characters from the Vector General keyboard onto the display screen. The ASCII character data is also placed in a user specified character array for processing by the user's program. Up to 121 characters can be entered. Data entry is terminated by a carriage return. The termination of the data entry also erases the output characters from the display screen.

b. Fget

The fget routine returns to the user a floating point number entered from the Vector General keyboard. This routine uses both the cget and getf routines.

c. Iget

The iget routine returns to the user an integer entered from the Vector General keyboard. This routine uses both the cget and geti routines.

4. Lightpen

a. Lghtpen

The lightpen selectability of the picture, a single object, or any number of elements associated with a specific object can be turned on or off by this routine. This determines what picture segments will be affected by lightpen interactions.

b. Penhit

After an element's lightpen selectability has been activated with the lghtpen routine, determination of whether or not an element has registered a lightpen hit is accomplished with this routine. A one is returned if a lightpen hit has occurred; otherwise a zero is returned.

For an element to register a lightpen hit it is necessary not only for the element's lightpen selectability to have been activated, but also the lightpen must be pointing at the element and the lightpen sense switch must be activated. The lightpen sense switch is activated by bridging the small rubber gap on the forward part of the lightpen.

c. Clrhit

This routine is used to clear the elements of lightpen hits. Element lightpen selectability is unaffected by this routine.

5. Function Switches

a. Fsman

This routine returns to the user an indication of whether or not a specified function switch is depressed. A one is returned to the user if depressed, otherwise a zero if returned.

b. Fstog

This routine makes the function switches appear as toggle switches. Odd (1st,3rd,...) selections of a switch light the lamp and return a value of one. Even (2nd,4th,...) selections turn the lamp off and return a value of zero.

c. Lamp

The lamp routine will light or extinguish the specified function switch lamp.

d. Manint

Each depression of the manual interrupt function switch increments the manual interrupt count by one. The manint routine either returns to the user the present manual interrupt count or sets that count to zero.

AD-A050 241 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 9/2
IMPLEMENTATION OF A USER INTERFACE WITH THE VECTOR GENERAL GRAP--ETC(U)
DEC 77 D C ENDICOTT, N P MARTINO

UNCLASSIFIED NL

2 OF 3
AD A050241

6. Control Dials

The dial routine returns a floating point number from -1.0 to 1.0 to the user to indicate the relative position of the specified control dial.

VI. RUNNING A VECTOR GENERAL PROGRAM

A. PROGRAM FORMAT AND COMPILATION

In order to properly utilize the Vector General display system, routines to initialize and terminate the system must be called. These routines initialize the Vector General, start the actual visual display and properly terminate the display at the end of a user's program. Each user program must include a call to these three routines:

```
sysinit();
vgpicture();
vgterm();
```

The first routine, sysinit, must be called by the user before any other user interface routine is called. To display the picture that a user has described on the Vector General screen, the routine vgpicture must be called. This routine is called only once, and nothing will appear on the screen until it has been called. Finally, to properly terminate, the system requires a call to the routine vgterm at the end of the user program.

1. C Programs

To properly initialize various data structures in the interface routines and utilize the Vector General program constants used in this manual the first line in the user's program should be:

```
#include "/usr/graph/vgdata"
```

Once a C program has been written it must be compiled with the user interface graphics subroutine library. To accomplish this the following command must be issued:

```
vgc filename.c
```

The result of a successful compilation is an executable a.out file.

2. Basic Programs

The first subroutine call made by the BASIC program should be:

```
call vgdata()
```

This causes the loading of the initialized display list. Care must be made in previously declaring the "vgdata" call in an "extern" statement before this call. This call would be in lieu of the "include" statement used by the C programs.

The program constant names used in the C Vector General programs do not apply in BASIC Vector General programs. Conversion values must be utilized instead of the string constant names when programming in BASIC. The following table lists the constant names and the equivalent values in decimal number representation to be used:

```
                DEFINE VECTOR TYPES
VR   = 4096                 VAY = 4102
VRX  = 4097                 VAZ = 4103
VRY  = 4098                 INC2 = 4104
```

```
VRZ = 4099                    INCX = 4105
VA  = 4100                    INCY = 4106
VAX = 4101                    INC3 = 4107
                DEFINE VECTOR MODES
LN  = 0                       PT  = 48
DSH = 16                      DD  = 80
DOT = 32                      DDD = 96
                DEFINE COORDINATE FIELD VALUES
AIR = 0                       X = 0
XR  = 1                       Y = 1
YR  = 2                       Z = 2
ZR  = 3
                DEFINE OF AND CF FIELDS FOR ARC COMMANDS
C  = 4
CC = 8
                DEFINE FIELDS FOR INCREMENTAL VECTORS
                     INCREMENT SCALE
NMG = 0.0
MG  = 128.0
                DEFINE CONSTANTS FOR CHARACTER GENERATION
SLNT = 0                      VGREL = -.00001
NSLNT = 1
                CHARACTER WRITE DIRECTION
HOR = 0
VER = 128
                CHARACTER SIZE
SZ  = 0                       SZ3 = 96
SZ1 = 64                      SZ4 = 112
SZ2 = 80
                DEFINE ACTION PARAMETER
ON  = 1                       CLEAR = 0
OFF = 0                       COUNT = 1
                DEFINE CASE STATEMENT VALUES FOR OBJ-ELE-PIC
PIC = 0                       ELE = 1
OBJ  = -1
                DEFINE DEVICE SWITCH NUMBERS
LPEN = 0                      JOY = 2
TRK  = 1
```

Once a BASIC program has been written, it must be compiled with the various user subroutine libraries. To accomplish the task, the following system resident command must be issued:

```
vgb filename.b
```

The result of a successful compilation is an executable "a.out" file.

B. SAMPLE PROGRAMS

1. C Programs

The following is an example of a C-language Vector General graphics program. The actual picture produced by this program is illustrated in Figure A-6.

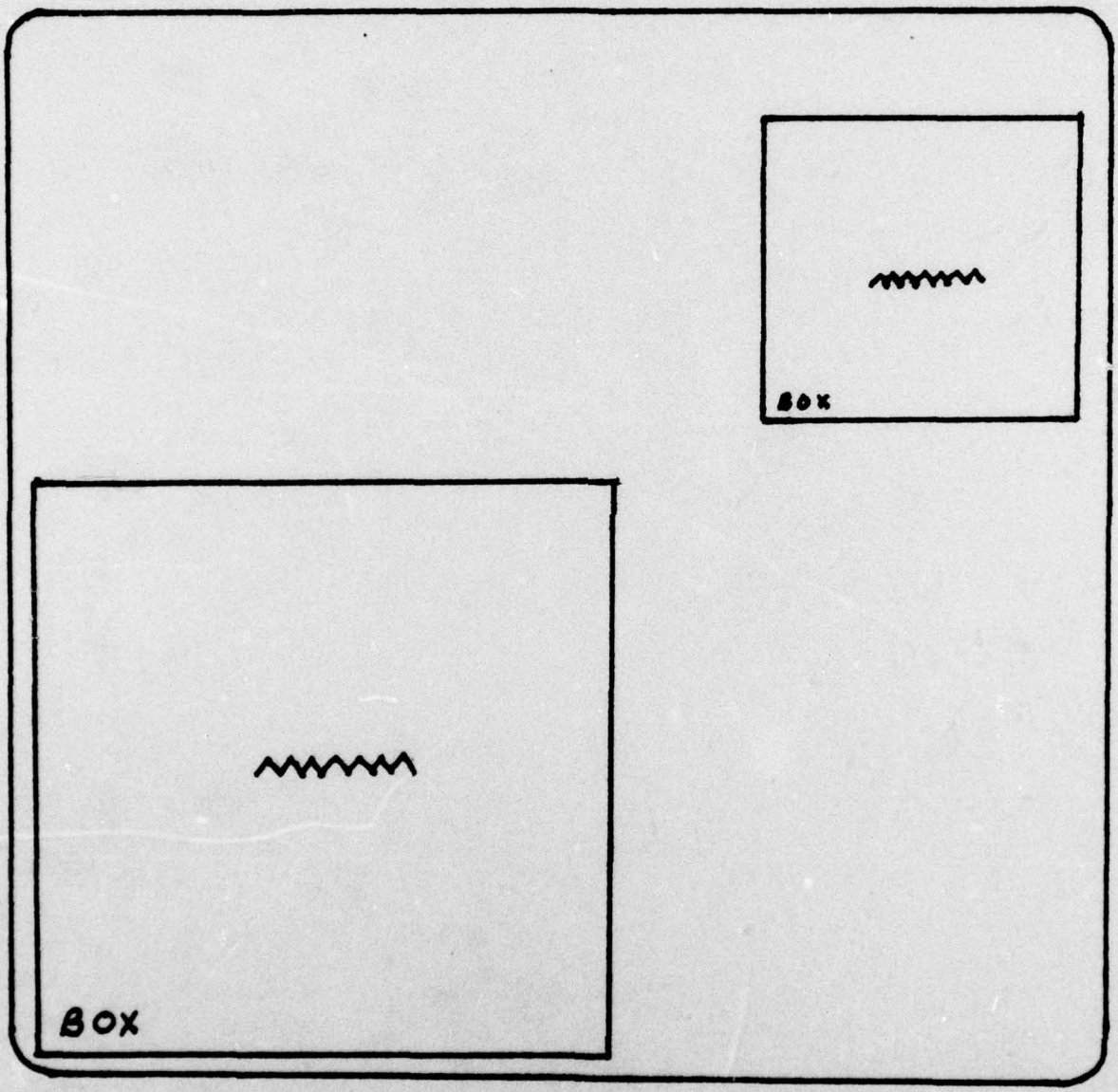
```
#include "/usr/graph/vgdata"

main()
{
 int i;
    sysinit(0); //initialize the Vector General

    //define coordinate system
    coordsys(2,-1.0,1.0,-1.0,1.0);

    drawele("box");
        setvector(VA,LN);
            move(-1.0,-1.0);
            line( 1.0,-1.0);
            line( 1.0, 1.0);
            line(-1.0, 1.0);
            line(-1.0,-1.0);
    endele();

    charele("name","#I BOX",SZ4,HOR,NSLNT,VGREL);

    drawele("zigzag");
        setvector(VA,LN);
            move(-0.5,-0.5);
        setvector(INCX,LN,.03,NMG);
            for(i=0 ; i< 7 ; i++ )
            line(.03,-.03);
    endele();

    //establish object-element relationship
    object(3,"bigbox","box","name","zigzag");
    object(3,"smallbox","box","name","zigzag");

    scale("bigbox",0.5);
    trans("bigbox",-0.5,-0.5);
    scale("smallbox",0.25);
    trans("smallbox",0.5,0.5);

    vgpicture();    //display picture
    sleep(30);  //display picture for 30 sec
    vgterm();   //terminate Vector General process
}
```

2. Basic Program

The following is an example of a BASIC Vector General graphics program. The actual picture produced by this program is illustrated in Figure A-6.

```
     rem    external subroutine declarations
extern vgdata()
extern sysinit(integer)
extern coordsys(integer,double,double,double,double)
extern drawele(&char)
extern setvector(integer,integer,double,double)
extern move(double,double)
extern line(double,double)
extern endele()
extern charele(&char,&char,integer,integer,integer,double)
extern object(integer,&char,&char,&char,&char)
extern scale(&char,double)
extern trans(&char,double,double)
extern vgpicture()
extern sleep(integer)
extern vgterm()

     rem    initialize display list
call vgdata()
     rem    initialize the Vector General
call sysinit(0)
     rem    define coordinate system
call coordsys(2,-1.0,1.0,-1.0,1.0)

call drawele("box")
   call setvector(4100,0,0,0)
   call move(-1.0,-1.0)
   call line(1.0,-1.0)
   call line(1.0,1.0)
   call line(-1.0,1.0)
   call line(-1.0,-1.0)
   call endele()

call charele("name","#I BOX",112,0,1,-.00001)

call drawele("zigzag")
   call setvector(4100,0,0,0)
      call move(-0.2,0.0)
   call setvector(4105,0,.03,0)
      for i = 0 to 6
         call line(.03,-.03)
      next i
   call endele()

     rem    establish object-element relationship
call object(3,"bigbox","box","name","zigzag")
```

```
call object(3,"smallbox","box","name","zigzag")

call scale("bigbox",0.5)
call trans("bigbox",-0.5,-0.5)
call scale("smallbox",0.25)
call trans("smallbox",0.5,0.5)

     rem    display picture
call vgpicture()
     rem    display picture for 30 sec
call sleep(30)
     rem    terminate Vector General process
call vgterm()
end
```



Picture Displayed by Sample Program
FIGURE A-6

C. ERROR DIAGNOSTICS

A user can control the printing of error messages through the use of the errormsg routine. The general format of an error message will be as follows:

E-nn TTTT: "eeee" in routine rrrr()

where:

nn = error number

TTTT = error type

eeee = error

rrrr = routine in error

For example:

1. E-1 BLOCK: in routine line()

   Message indicates a block error in routine line(). Probable cause is a line() command issued outside a drawele() block. The line() command will be disregarded and control returned to the user's program.

2. E-5 NO OBJ: "Tree" in routine erase()

   Message indicates that the object 'Tree' has been used in the erase() routine and no such object currently exists. Routine will be disregarded and control returned to the user's program.

3. E-15 X: "150.000000" in routine printv()

   Message indicates an improper X coordinate value in the printv() routine. The erroneous X coordinate value of 150 will be reduced to a value within the defined coordinate system (ie: 100 for default coordinate system) and the routine will continue execution.

4. E-25 NUM DIAL: "36" in routine dial()

   Message indicates an improper dial number of 36 used in the dial() routine. Routine will return a value of 0.0 to the user.

A brief idea of the cause of the error may be gained

from the error type specification in the error message. Hopefully with a specification of the error, the error type and routine where the error occured, the user will be able to quickly trace and correct the problem. Should a more detailed explanation be desired, a fuller description of the specified error may be obtained with the help of the error number and error diagnostic table, or by reviewing the diagnostic section under the routine in question.

The software support package is designed so that most errors will not terminate a user's program, and in fact most errors will usually result in the termination of the specified routine and return control to the user's program. Every effort will be made to execute the support routines and user's program to completion so that at least some of the user's data will be displayed.

TABLE-III
ERROR DIAGNOSTIC TABLE

E-1 ELEMENT BLOCK ERROR - The specified routine is in error in one of two respects. Either the routine has been called outside a drawele() block when it should be within, or the routine has been called within a drawele() block when it should be outside. The instruction will be ignored and control returned to the user's program. Block errors have a tendency to occur in groups when the programmer is not careful in constructing element blocks.

E-2 SPACE ERROR - Space allocated by the system for the building of elements has been exceeded. See the routine remove in order to free up unnecessary element memory locations. The instruction executing when this error occurs will be ignored and control will be returned to the user's program.

E-3 DIRECTION ERROR - The value of the parameter 'dir'

was not C or CC in the specified routine. A direction of C will be assumed and the routine will continue execution.

E-4 CHARACTER ERROR - A character symbol included in an ASCII character string is undefined. The symbol will be ignored and control will be returned to the user's program.

E-5 OBJECT ERROR - The named object does not exist; it has not been defined in a call to the routine object(). A specification of a non-existent object will terminate the routine and return control to the user's program.

E-6 ELEMENT ERROR - The named element does not exist; it has not been defined in a call to drawele(), charele(), or printv(). For routines that specify a number of element names, this type of error normally results in skipping the specified element and continuing the routine with the next element.

E-7 DUPLICATE ELEMENT ERROR - An element name has been duplicated in a drawele() or charele() call. Each element structure created by drawele() or charele() must have a unique name. This error usually results in either skipping the designated element and continuing the routine, or termination of the routine and return to the user's program.

E-8 ELEMENT NUMBER ERROR -The total number of elements allowed by the system has been exceeded. Maximum number of elements allowed is 70. The user should consider using the remove() routine to delete unnecessary elements so new elements may be introduced.

E-9 OBJECT NUMBER ERROR - The number of objects allowed by the system has been exceeded. Maximum number of objects is 10.

E-10 VECTOR MODE ERROR - Specified vector vmode in setvector() routine is in error. A vector mode of LN will be assumed and the routine will continue execution. See the setvector() summary for allowable vector mode codes.

E-11 DIMENSION ERROR - The coordinate dimension specification is in error. Either the 'dim' parameter in coordsys() is in error, or a vector type of VAZ or VRZ has been specified with a two-dimensional coordinate system. The routine in question will be skipped and control returned to the user's program.

E-12 ADD ELEMENT ERROR -An error has occurred in cget() or object() in attempting to build a new element. Routine terminates and control is returned to the user's program.

E-13 MAKE OBJECT ERROR - An error has occurred in cget() or object() in attempting to build a new object. Routine terminates and control is returned to the user's program.

E-14 VECTOR TYPE ERROR - An improper vector type was specified in routine setvector(). A vector type of VA will be assumed and the routine will continue executing. See the setvector() summary for allowable vector type codes.

E-15 X COORDINATE/INCREMENT ERROR - The specified x parameter exceeds allowable limits. If an absolute x coordinate exceeds the defined coordinate system, the x value will be set to the value of the nearest coordinate within the defined system (xmin or xmax). If the parameter specifies an x increment which exceeds one-half the coordinate range, the parameter is set to zero.

E-16 Y COORDINATE/INCREMENT ERROR - The specified y parameter exceeds allowable limits. If an absolute y coordinate exceeds the defined coordinate system, the y value will be set to the value of the nearest coordinate within the defined system (ymin or ymax). If the parameter specifies a y increment which exceeds one-half the coordinate range, the parameter is set to zero.

E-17 Z COORDINATE/INCREMENT ERROR - The specified z parameter exceeds allowable limits. If an absolute z coordinate exceeds the defined coordinate system, the z value will be set to the value of the nearest coordinate within the defined system (zmin or zmax). If the parameter specifies a z increment which exceeds one-half the coordinate range, the parameter is set to zero.

E-18 ELEMENT NUMBER ERROR - The parameter 'num' specifying the number of elements being passed in this routine is not -1 to 10. A 'num' value of 10 will be assumed and the routine will continue executing.

E-19 CIR/ARC ERROR - A circle or arc cannot be drawn by an incremental vector. The routine is skipped and control returned to the user's program.

E-20 USER BUFFER ERROR - The size of the supplied user buffer has been exceeded. Either enter a carriage

return or erase (cntrl-a) part of the buffer.

E-21 ADD OBJECT ERROR - An error has occurred in cget() or object() in attempting to add an object to the display. Routine terminates and control returned to the user's program.

E-22 OBJECT LINK ERROR - The specified element is not linked to the specified object. Error usually results in skipping the element in question and continuing the routine with the next element.

E-23 FUNCTION SWITCH NUMBER ERROR -An improper function number was specified. The routine terminates and control is returned to the user's program. A value of zero is returned to the user.

E-24 ACTION VALUE ERROR - An improper action value was specified in this routine. Refer to routine summary diagnostics for specific action taken.

E-25 DIAL NUMBER ERROR - An improper dial number was specified in routine dial(). The routine terminates and a value of 0.0 is returned to the user's program.

E-26 ELEMENT ERROR -Element specified in printv routine was specified earlier as other than a printv element. Routine will be skipped and control returned to the user's program.

E-27 DEVICE NUMBER ERROR -An improper device number was used in this routine. For sysinit() a device number of zero will be assumed and the routine will continue. For cursor() or posit() the routine will terminate and control will return to the user's program. The posit() routine will also return a value of 0.0 to the user.

E-28 COORDINATE SPECIFICATION ERROR - The specification of a coordinate in this routine is in error. The routine terminates and a value of 0.0 is returned to the user.

E-29 DIMENSION ERROR - Neither the lightpen nor the trackball have a z coordinate specification. The routine terminates and a value of 0.0 is returned to the user.

E-30 VALUE ERROR - The 'val' parameter in intscale() or intoffset(), or the 'rate' parameter in refresh() is in error. The routine terminates and control returns to the user's program.

E-31 CUT ERROR - The 'cut' parameter in intscale() is

not zero or one. The routine terminates and control returned to the user's program.

E-32 DUPLICATE LINK ERROR - The specified element is already linked to the object. The routine continues execution with the next specified element.

E-33 BUFFER LIMIT ERROR - Either the supplied user buffer has been filled or 121 characters have been read in the cget() routine. The routine will wait for either a carriage return or erasure (cntrl-a) of part of the buffer contents.

E-34 FORMAT ERROR - The desired output string exceeds 100 characters in the printv() routine. The routine will print only the first 100 characters.

E-35 WRITE ERROR - The write direction parameter is in error. A default value of HOR will be assumed and the routine continued.

E-36 SLANT ERROR - The slant parameter is in error. A default value of NSLNT will be assumed and the routine continued.

E-37 SIZE ERROR - The size parameter is in error. A default value of SZ4 will be assumed and the routine continued.

APPENDIX B

USER SUPPORT ROUTINE DESCRIPTIONS

The description of the user interface routines appear in a format similar to the routine descriptions in the UNIX Reference Manual at the Naval Postgraduate School Computer Laboratory.

NAME:

arc - draw an arc

SYNOPSIS:

```
2-dimensional coordinate system
  VR/VA  : arc(dir,centx,centy,endx,endy)
  VRX/VAX: arc(dir,centy,endy)
  VRY/VAY: arc(dir,centx,endx)
  VRZ/VAZ: not allowed

3-dimensional coordinate system
  VR/VA  : arc(dir,centx,centy,centz,endx,endy,endz)
  VRX/VAX: arc(dir,centy,centz,endy,endz)
  VRY/VAY: arc(dir,centx,centz,endx,endz)
  VRZ/VAZ: arc(dir,centx,centy,endx,endy)

int dir: CC / 004 - counter clockwise
         Ç / 010 - clockwise

float centx,centy,centz,endx,endy,endz;
```

DESCRIPTION:

An arc is drawn from the present beam location, about a center point to an end point. The location of the center and end points depend upon the specified vector type and supplied center and end point parameters. The parameters passed for the center and end points of the arc are either relative or absolute values. The number and value of these parameters are determined by the vector type selected in the previous call to set-vector(). If the center point is not an equal distance from the starting point of the arc and the end point, an arc will be drawn using one of the distances as the radius. The result is a straight line drawn from the actual terminating point of the arc to the end point.

If the vector type is an absolute vector, the center and end points will be as given in the parameter list. Auto-increment vector types will cause the center and end points to have the same coordinate value as the starting point for the designated auto-increment coordinate. Under absolute vector types, parameters which fall outside the defined coordinate system will assume a value of the nearest coordinate within the defined system.

If the vector type is a relative vector, the end point of the arc will be obtained by adding the endx, endy, and endz values to the starting point. The center point is then obtained by adding the centx, centy, and centz values to the previously obtained endpoint. As with absolute vectors, auto-increment modes cause the center and end points to have the same coordinate value as the starting point for the designated auto-increment coordinate. Under relative vector types, parameters which exceed one-half the coordinate range will be set to zero.

Arcs cannot be drawn by any of the four incremental vectors.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-1	Routine has been called outside a drawele() block	Return
E-2	No more space available in display list	Return
E-3	The draw direction was not defined as C or CC	dir = C
E-11	3D not defined	Return
E-15	The x value was out-of-bounds	x = xmin or xmax or 0.0
E-16	The y value was out-of-bounds	y = ymin or ymax or 0.0
E-17	The z value was out-of-bounds	z = zmin or zmax or 0.0
E-19	Illegal arc instruction from incremental vector	Return

ALSO SEE:

circle, setvector

NAME:

blink - blink the entire picture or the specified object or elements

SYNOPSIS:

```
blink(action,num,"objname","ele1","ele2",...,"ele10");

int action:  ON / 01    blink
            OFF / 00    stop blinking action

int num:    PIC / 00    picture
            OBJ / -1    object
             1 - 10     elements
```

To blink the entire picture:

```
blink(action,PIC);
```

To blink a specific object:

```
blink(action,OBJ,"objname");
```

To blink elements of a specific object:

```
blink(action,num,"objname","ele1","ele2",...);
```

DESCRIPTION:

The blink mode for the entire picture, a single object or any number of elements of a specific object, can be turned on or off.

Modifying the blink mode of an object affects all elements linked to that object.

The parameter 'num' specifies the number of element names being passed in this routine and can vary between 1 and 10.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-5	Specified objname does not exist	Return

E-6	Specified element does not exist	Skip element
E-18	Parameter 'num' not -1 to 10	num = 10
E-22	Element not linked to specified object	Skip element
E-24	Improper action value	action = OFF

NAME:

cget - get ASCII characters from the VG keyboard

SYNOPSIS:

```
cget(buffer,length,xpos,ypos);

char *buffer;

int length;

float xpos,ypos;
```

DESCRIPTION:

This routine allows for buffered input from the VG keyboard to be displayed on the display screen and inserted in a user-supplied buffer. Parameters allow for the positioning of the display buffer on the screen in absolute coordinates. The length of the user-supplied buffer must be specified in the parameter 'length'.

A blinking cursor indicates the beginning of the buffer and the character size. Characters may be entered into the buffer and onto the screen up to a maximum of either 121 characters or the size of the supplied user buffer, whichever is smaller.

Contents may be erased from the screen and the user buffer by repeated cntrl-a keystrokes. A C/R will terminate the input string, delete the display of the buffer contents, and insert a '\0' as the next character in the user buffer to indicate termination of the string.

A listing of the ASCII character code for every keyboard entry is listed on the following page.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-9	Object limit exceeded	Return
E-15	Specified xpos out-of-bounds	xpos = xmin or xmax

E-16	Specified ypos out-of-bounds	ypos = ymin or ymax
E-20	Either 121 characters entered or user buffer limit reached	Wait for CR or CNTRL-A

ALSO SEE:

iget, fget

OCTAL CODE	CHARACTER	GENERATED SYMBOL	VG KEYBOARD SYMBOL
000	NULL	(ignored)	@ ctrl
001	SO	erase last char.	A ctrl
002	STX	(ignored)	B ctrl
003	ETX	(ignored)	C ctrl
004	EOT	(ignored)	D ctrl
005	ENQ	(ignored)	E ctrl
006	ACK	(ignored)	F ctrl
007	BEL	(ignored)	G ctrl
010	BS		BS
011	HT	(LF, cent)	I ctrl
012	LF		LF
013	VT	(top, cent)	K ctrl
014	FF	(top, left)	L ctrl
015	NL	(CR, LF)	CR
016	SE	(ignored)	N ctrl
017	SI	(ignored)	O ctrl
020	DLE	(clear queue)	P ctrl
021	DC1	(-LF)	Q ctrl
022	DC2	(-SZ)	R ctrl
023	DC3	(+SZ)	S ctrl
024	DC4	(terminate)	T ctrl
025	NAK	(ignored)	U ctrl
026	SYN	(ignored)	V ctrl
027	ETB	(ignored)	W ctrl
030	CAN	(ignored)	X ctrl
031	EM	(ignored)	Y ctrl
032	SUB	(ignored)	Z ctrl
033	ESC	(escape)	[ctrl
034	FS	(ignored)	\ ctrl
035	GS	(ignored)	] ctrl
036	RS	(ignored)	^ ctrl
037	US	(ignored)	_ ctrl
040	Space		sp bar
041	!		1 shift
042	"		2 shift

cget

043	#	3 shift
044	$	4 shift
045	%	5 shift
046	&	6 shift
047	'	7 shift
050	(	8 shift
051	)	9 shift
052	*	: shift
053	+	; shift
054	,	,
055	-	-
056	.	.
057	/	/
060	0	0
061	1	1
062	2	2
063	3	3
064	4	4
065	5	5
066	6	6
067	7	7
070	8	8
071	9	9
072	:	:
073	;	;
074	<	, shift
075	=	- shift
076	>	. shift
077	?	/ shift
100	@	@
101	A	A shift
102	B	B shift
103	C	C shift
104	D	D shift
105	E	E shift
106	F	F shift
107	G	G shift
110	H	H shift
111	I	I shift
112	J	J shift
113	K	K shift
114	L	L shift
115	M	M shift
116	N	N shift
117	O	O shift
120	P	P shift
121	Q	Q shift
122	R	R shift
123	S	S shift
124	T	T shift
125	U	U shift
126	V	V shift

127	W		W shift
130	X		X shift
131	Y		Y shift
132	Z		Z shift
133	[		[
134	\		\
135	]		]
136	^	(superscript)	^
137	_	(subscript)	_
140	`		@ shift
141	a		A
142	b		B
143	c		C
144	d		D
145	e		E
146	f		F
147	g		G
150	h		H
151	i		I
152	j		J
153	k		K
154	l		L
155	m		M
156	n		N
157	o		O
160	p		P
161	q		Q
162	r		R
163	s		S
164	t		T
165	u		U
166	v		V
167	w		W
170	x		X
171	y		Y
172	z		Z
173	{		{ shift
174	!		\ shift
175	}		] shift
176	~		^ shift
177	del		DEL
240	■		space spec
241	◆		1 shift spec
242	▒		2 shift spec
243	°		3 shift spec
244	±		4 shift spec
245	v	(centered)	5 shift spec
246	┘		6 shift spec
247	┐		7 shift spec
250	┌		8 shift spec
251	└		9 shift spec
252	┼	(subscript)	: shift spec

253	+		; shift spec
254	◀		, spec
255	■		- spec
256	↑		. spec
257	↓		/ spec
260	●		0 spec
261	↑		1 spec
262	↓		2 spec
263	□		3 spec
264	←		4 spec
265	→	(centered)	5 spec
266	∂		6 spec
267	∆		7 spec
270	∪		8 spec
271	∩		9 spec
272	·	(center dot)	: spec
273	×		; spec
274	←		, shift spec
315	■		M shift spec
316	∧		N shift spec
317	α		O shift spec
320	π		P shift spec
321	∩		Q shift spec
322	↓		R shift spec
323	∞		S shift spec
324	∅		T shift spec
325	-		U shift spec
326	∫		V shift spec
327	∧		W shift spec
330	∫		X shift spec
331	∩		Y shift spec
332	⌐		Z shift spec
333	⌐		[spec
334	→		\ spec
335	⊤		] spec
336	-		^ spec
337	•		_ spec
340	┘	(blinking)	@ shift spec
341	↓		A spec
342	∅		B spec
343	∂		C spec
344	∫		D spec
345	∫		E spec
346	∅		F spec
347	∀		G spec
350	∀		H spec
351	⊤		I spec
352	∩		J spec
353	/		K spec
354	/		L spec
355	∧		M spec
356	∧		N spec

cget

357	ｗ	O spec
360	ｒ	P spec
361	ｂ	Q spec
362	ｅ	R spec
363	ｒ	S spec
364	ｔ	T spec
365	ｕ	U spec
366	ｏ	V spec
367	ｊ	W spec
370	ｋ	X spec
371	ｌ	Y spec
372	ｖ	Z spec
373	ｆ	{ spec shift
374	｜	\ spec shift
375	ｈ	} spec shift
376	～	^ spec shift
377	■	DEL spec

NAME:

charele - displays an ASCII character string
on the Vector General

SYNOPSIS:

```
charele("elename",charptr,size,wdir,slant,xpos,ypos);

int size:  SZ  / 0000    use previous character size
           SZ1 / 0100    size = 100 col x 60 lines
           SZ2 / 0120    size = 81 col x 41 lines
           SZ3 / 0140    sixe = 60 col x 30 lines
           SZ4 / 0160    size = 32 col x 16 lines

int wdir:  HOR / 0000    write horizontally
           VER / 0200    write vertically

int slant:  SLNT  / 00   slant characters
            NSLNT / 01   no slant characters

float xpos, ypos;

      xpos:  VGREL - to output characters relative
                     to the present beam position
                     the parameter xpos should be
                     replaced with VGREL and the
                     parameter ypos omitted.
```

DESCRIPTION:

This routine displays the ASCII character string, specified by charptr, on the vector general screen. The characters can be output by specifying the desired coordinate starting point (xpos,ypos) or by outputting the string relative to the present beam location. The symbols available include formatting symbols and an extended character set. These can be included in any character string by including the proper identifying symbols. The character string formatting symbols are:

KEYBOARD SYMBOL	FORMAT
#@	backspace; moves back one character space
#A	line feed, position center screen
#B	line feed
#C	position at top center of screen
#D	position at top left corner of screen
#E	carriage control, line feed

#F	ignored
#G	ignored
#H	special character
#I	neg. line feed; moves up one line
#J	decreases current character size by one
#K	increases current character size by one

Due to the extended character set available on the vector general, all special character symbols are preceded by a pound sign. In order to have a pound sign appear on the screen, two pound signs (##) must appear in the character string. A list of the special characters available are listed on the next page under Extended Character Set.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-1	Routine has been called within a drawele() block	Return
E-2	No space available in display list	Return
E-4	Incorrect ASCII character symbol found in character string	None
E-7	Duplicate element name	Return
E-8	Maximum number of elements exceeded	Return
E-15	Specified xpos out-of-bounds	xpos = xmin or xmax
E-16	Specified ypos out-of bounds	ypos = ymin or ymax
E-35	Improper 'wdir' parameter	write = HOR
E-36	Improper 'slant' parameter	slant = NSLNT
E-37	Improper 'size' parameter	size = SZ4

Extended Character Set

DATAMEDIA KEYBOARD SYMBOL	CHARACTER
#space	¤
#!	↓
#"	‖
#L	○
#$	£
#H	∨
#&	∫
#'	√
#(	⊂
#)	⊃
#*	∘
#+	÷
#,	≤
#-	≡
#.	≥
#/	¬
#0	°
#1	↑
#2	π
#3	□
#4	←
#5	∧

charele

#6	∂
#7	∠
#8	∪
#9	∩
#:	•
#;	X
#<	←
#M	■
#N	∩
#O	Ω
#P	∓
#Q	∩
#R	ϕ
#S	Σ
#T	θ
#U	ι
#V	∫
#W	∧
#X	↑
#Y	∫
#Z	—
#[	L
#\	→
#]	⊢
#↑	—
#←	•

charele

#'	
#a	
#b	
#c	
#d	
#e	
#f	
#g	
#h	
#i	
#j	
#k	
#l	
#m	
#n	
#o	
#p	
#q	
#r	
#s	
#t	
#u	
#v	
#w	
#x	
#y	

#z	▚
#(	┌
#|	|
#)	┤
#¬	~
#?	■

NAME:

circle - draw a circle

SYNOPSIS:

```
2-dimensional coordinate system
  VR/VA   :  circle(dir,centx,centy);
  VRX/VAX:  circle(dir,centy);
  VRY/VAY:  circle(dir,centx);
  VRZ/VAZ:  not allowed

3-dimensional coordinate system
  VR/VA   :  circle(dir,centx,centy,centz);
  VRX/VAX:  circle(dir,centy,centz);
  VRY/VAY:  circle(dir,centx,centz);
  VRZ/VAZ:  circle(dir,centx,centy);

int dir:  C / 010    clockwise
         CC / 004    counter-clockwise

float centx,centy,centz;
```

DESCRIPTION:

A circle is drawn from the present beam location about a center point. The location of the center point depends upon the specified vector type and supplied center point parameter. The parameters passed for the center point of the circle are either relative or absolute values. The number and value of these parameters are determined by the vector type selected in the previous call to setvector().

If the vector type is an absolute vector then the center point of the circle will be as given in the parameter list. Auto-increment vector types will cause the center point to have the same coordinate value as the starting point for the designated auto-increment coordinate. Under absolute vector types, parameters which fall outside the defined coordinate system will assume a value of the nearest coordinate within the defined system.

If the vector type is a relative vector the center point will be obtained by adding the centx, centy, and centz values to the starting point. As with absolute vectors, auto-increment modes cause the center point to have the same coordinate value as the starting point for the designated auto-increment coordinate.

Under relative vector types, parameters which exceed one-half the coordinate range will be set to zero.

Circles cannot be drawn by any of the four incremental vectors.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-1	Routine has been called outside a drawele() block	Return
E-2	No space available in display list	Return
E-3	Draw direction not defined as either C or CC	dir = C
E-11	3D not defined	Return
E-15	The x value was out-of-bounds	x = xmin or xmax or 0.0
E-16	The y value was out-of-bounds	y = ymin or ymax or 0.0
E-17	The z value was out-of-bounds	z = zmin or zmax or 0.0
E-19	Illegal circle instruction from incremental vector	Return

ALSO SEE:

arc, setvector

NAME:

clrhit - clear picture/object/element lightpen hit

SYNOPSIS:

```
clrhit(num,"objname","elename");

int num:  PIC / 00   clear all element lightpen hits
          OBJ / -1   clear lightpen hits on all
                     elements of specified object
          ELE / +1   clear lightpen hit on specified
                     element of specified object
```

To clear all elements of lightpen hits:

```
clrhit(PIC);
```

To clear all elements of object 'tree':

```
clrhit(OBJ,"tree");
```

To clear element 'branch5' of object 'tree':

```
clrhit(ELE,"tree","branch5");
```

DESCRIPTION:

This routine clears lightpen hits on a single element, group of elements, or all elements. If PIC is specified, all elements are cleared of lightpen hits. If OBJ is specified, all elements attached to the specified object are cleared of lightpen hits. If ELE is specified, only the indicated element of the specified object is cleared of a lightpen hit. Invalid parameters will result in no action being taken.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-5	Indicated objname does not exist	Return
E-6	Indicated elename does not exist	Return
E-22	Element not linked to specified object	Return

NAME:

coordsys - define user coordinate system

SYNOPSIS:

```
coordsys(dim,minx,maxx,miny,maxy [,minz,maxz] );

int dim:  2 or 3

float minx,maxx,miny.maxy,minz,maxz;
```

DESCRIPTION:

Defines a two or three dimensional cartesian coordinate system, of any scale, for the user. The parameter 'dim' specifies the number of dimensions required. If 'dim' is 2 then only the range of x and y need to be specified.

minx must be strictly less than maxx.

miny must be strictly less than maxy.

minz must be strictly less than maxz.

All subsequent user coordinate values are interpreted according to this user defined coordinate system.

If coordsys() is not called by the user the default values will be taken. The default coordinate system is three dimensional with x, y, z ranging from -100.0 to 100.0. All coordinate values received will be interpreted according to these default values unless the coordinate system is redefined by the user.

DIAGNOSTICS:

Any values which fall outside of the defined coordinate system will produce an error message specifying which coordinate value was out-of-bounds. If the value was an absolute coordinate, the nearest coordinate within the defined coordinate system will be assumed. If the value was a relative value that was out-of-bounds, a value of zero will be assumed.

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-11	Dimension not specified as 2D or 3D	Return

NAME:

copyele - specify additional names to be used in referencing a specific element block

SYNOPSIS:

```
copyele(num,"elename","name1","name2",....,"name10");

int num:   1 to 10
```

DESCRIPTION:

This routine allows the user to give several unique names to a specific element block. In this manner, a user can link one element to an object several times and can uniquely reference each occurrence of the element within the object.

The parameter 'num' specifies the number of element names being passed in this routine and can vary between 1 and 10.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-6	Specified elename does not exist	Return
E-7	Duplicate element name	Skip element
E-8	Maximum number of elements exceeded	Return
E-18	Parameter 'num' not 1 to 10	num = 10

ALSO SEE:

drawele, object

NAME:

cursor - activate/deactivate display of specified device cursor

SYNOPSIS:

```
cursor(dev,action);

int dev:  LPEN / 00     lightpen
           TRK / 01     trackball
           JOY / 02     joystick

int action:  OFF / 00      turn cursor display off
              ON / 01      turn cursor display on
```

DESCRIPTION:

This routine will activate/deactivate the display of the specified device cursor depending on the specified 'action' value.

The trackball and joystick cursors are displayed as blinking plus signs. The trackball cursor is two dimensional and may be moved along the X-Y plane. The joystick cursor is drawn two dimensionally but can move in all three dimensions. As the joystick cursor moves along the Z axis its intensity cueing is affected. Both trackball and joystick cursors may be moved without being displayed.

The lightpen cursor is a small non-blinking octogon. It is drawn in two dimensions and can only move along the X-Y plane. The lightpen may only be moved while being displayed. Lightpen cursor movement may be accomplished in one of two speeds, depending upon whether the lightpen sense switch is on.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-24	Improper 'action' value	Return
E-27	Improper 'dev' number	Return

NAME:

dial - return value of specified dial

SYNOPSIS:

```
dial(num);

int num:  1-10

double dial();
```

DESCRIPTION:

This routine returns to the user a double precision value from -1.0 to +1.0 of the specified dial. A zero (0) is returned to the user if an improper dial 'num' is specified.

To obtain the double precision value from this routine the function declaration "double dial();" must be made in the users program.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-25	Improper dial 'num'	Return 0.0

NAME:

display - activate/deactivate display of the entire picture, an object, or an element

SYNOPSIS:

```
display(action,type,"objname","ele1",....,"ele10");

int action:  ON / 01    turn display on
            OFF / 00    turn display off

int type:   PIC / 00    entire picture
            OBJ / -1    specified object
             1 - 10     individual elements
```

To activate display of entire picture:

```
display(ON,PIC);
```

To deactivate display of an object:

```
display(OFF,OBJ,"objname");
```

To deactivate display of 3 elements in an object:

```
display(OFF,3,"objname","ele1","ele2","ele3");
```

DESCRIPTION:

This routine will activate or deactivate the display of the entire picture, an object, or elements within an object depending upon the 'action' parameter. Unlike the erase() and remove() routines, this routine does not affect the element-object relationships, but simply turns the display of its parameters either on or off. The routine should not be confused with the vgpicture() routine which must be called to initiate the display of elements.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-5	Specified objname not found	Return
E-6	Specified element not found	Skip element

E-18	Parameter 'type' not -1 to 10	type = 10
E-22	Element not linked to specified object	Skip element

ALSO SEE:

vgpicture, erase, remove

NAME:

drawele - start a draw element block

SYNOPSIS:

```
drawele("elename");
```

DESCRIPTION:

This routine specifies the beginning of a draw element block. It associates a unique name with the group of draw instructions that fall in between this call and a call to endele(). The resulting picture segment will then be referenced by the name specified as the element name in this routine.

A user may want to repeat a specific element block structure several times within one object. Instead of specifying several element blocks which describe the same structure a user can indicate a group of names which refer to one specific element block. These unique names each referring to the same structure can then be linked to an object, and each can be uniquely referenced. This association of several names with one element can be accomplished by the routine copyele().

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-1	Routine has been called within a drawele() block	Return
E-2	No space available in display list	Return
E-7	Duplicate element name	Return
E-8	Maximum number of elements exceeded	Return

ALSO SEE:

copyele, object

NAME:

endele - end of the current element block

SYNOPSIS:

```
endele();
```

DESCRIPTION:

Specifies the termination of a list of draw instructions describing a specific element. The picture segment described by a group of draw instructions that fall between a drawele() and endele() call will be referenced by the name specified in the drawele() call.

A new element block cannot begin until the previous block has been properly terminated by a call to endele().

DIAGNOSTICS:

This routine must be called to properly end a draw element block. If a block is not properly terminated prior to the beginning of a new element block, all drawele() calls will be ignored and any draw instructions that follow will be associated with the element block that has not been terminated.

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-1	Routine has been called outside a drawele() block	Return

NAME:

erase - erase the specified portions of the picture

SYNOPSIS:

```
erase(num,"objname","ele1","ele2",....,"ele10");

int num:   PIC / 00    all elements
           OBJ / -1    all elements within specified
                       object
            1 - 10     specified elements within
                       specified object
```

To erase the entire picture:

```
erase(PIC);
```

To erase a specific object:

```
erase(OBJ,"objname");
```

To erase elements of a specific object:

```
erase(num,"objname","ele1","ele2",..);
```

DESCRIPTION:

The entire picture, specified object or the listed elements of a specific object, will be erased from the vector general display screen. The elements still exist. To redisplay any portion of the erased picture, the appropriate element-object linking must again be done by the user.

The parameter 'num' specifies the number of element names being passed in this routine and may vary between 1 and 10.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-5	Specified objname does not exist	Return
E-6	Specified element does not exist	Skip element

E-18	Parameter 'num' not -1 to 10	num = 10
E-22	Element not linked to specified object	Skip element

ALSO SEE:

remove, display, object

NAME:

errormsg - print error messages on PDP-11 terminal screen

SYNOPSIS:

```
errormsg(action);

int action:   ON / 01    print error messages
              OFF / 00   do not print error messages
```

DESCRIPTION:

All error messages will automatically be printed on the PDP-11 terminal screen. The user can control the printing of error messages during any portion of a program by calling this routine.

NAME:

fget - get a floating point number from the VG keyboard

SYNOPSIS:

```
fget([num,xpos,ypos]);

int num:  ABS / 01   absolute coordinate indicator

float xpos,ypos;     (x,y) absolute coordinate
```

To return a floating point number and begin input display buffer at (50.0,75.0) absolute:

```
fget(ABS,50.0,75.0);
```

To return a floating point number and begin input display buffer at (xmin,ymin) absolute:

```
fget();
```

DESCRIPTION:

This routine returns to the user a floating point number from the VG keyboard. The routine is similar to getf() and in fact uses getf() to accomplish its task.

The optional parameters allow the user to position the input display buffer anywhere on the display screen. The 'xpos' and 'ypos' parameters are absolute floating point coordinates. If no parameters are given the input buffer will begin at (xmin,ymin). Coordinate parameters which fall outside the defined coordinate system will assume a value of the nearest coordinate within the defined system.

The input display buffer is drawn with the help of a call to the cget() routine. The input display buffer has a maximum size of 20 characters before an error message is displayed.

To obtain the floating point value from this routine the function declaration "float fget();" must be made in the users program.

DIAGNOSTICS:

All errors will be printed of the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-15	Specified xpos out-of-bounds	xpos = xmin or xmax
E-16	Specified ypos out-of-bounds	ypos = ymin or ymax
E-33	Buffer limit of 20 characters reached	Wait for CR or CNTRL-A

ALSO SEE:

iget, cget

NAME:

fsman - check function switch depressed

SYNOPSIS:

```
fsman(num);

int num:  0 - 31
```

DESCRIPTION:

This routine checks the status of the specified function switch and returns a one (1) if depressed and a zero (0) if not depressed. Unlike fstog(), lamps are not lighted by this routine and a one is returned only so long as the switch is physically depressed.

It is not recommended that both fstog() and fsman() be used in the same program. Due to the operation of the routine fstog(), if both routines are used the lamp status on switches queried by fsman() should be disregarded.

A zero is returned if an improper function switch 'num' is specified.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-23	Improper function switch 'num' specified	Return 0

ALSO SEE:

fstog, lamp

NAME:

fstog - check function switch toggle

SYNOPSIS:

```
fstog(num);

int num:  0 - 31
```

DESCRIPTION:

This routine checks the specified function switch and returns a one (1) and lights the appropriate lamp on the odd (1st,3rd,5th,....) depressions of the function switch. The return value and lamp status remain the same until the next even (2nd,4th,6th,....) depression of the function switch whereupon a zero (0) is returned and the lamp extinguished.

It is not recommended to use both fstog() and fsman() routines in the same program; however, if both are used, disregard the lamp status of function switches queried by fsman().

A zero is returned if an improper function switch 'num' is specified.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-23	Improper function switch 'num' specified	Return 0

ALSO SEE:

fsman, lamp

NAME:

iget - get an integer from the VG keyboard

SYNOPSIS:

```
iget([num,xpos,ypos]);

int num:  ABS / 01   absolute coordinate indicator

float xpos,ypos;   (x,y) absolute coordinate
```

To return an integer and begin input display buffer at (-25.0,17.0) absolute:

```
iget(ABS,-25.0,17.0);
```

To return an integer and begin input display buffer at (xmin,ymin) absolute:

```
iget();
```

DESCRIPTION:

This routine returns to the user an integer from the VG keyboard. The routine is similar to geti() and in fact uses geti() to accomplish its task.

The optional parameters allow the user to position the input display buffer anywhere on the display screen. The 'xpos' and 'ypos' parameters are absolute floating point coordinates. If no parameters are given the input display buffer will begin at (xmin,ymin). Coordinate parameters which fall outside the defined coordinate system will assume a value of the nearest coordinate within the defined system.

The input display buffer is drawn with the help of a call to the cget() routine. The input display buffer has a maximum size of 20 characters before an error message is displayed.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-15	Specified xpos out of bounds	xpos = xmin or xmax

E-16	Specified ypos out of bounds	ypos = ymin or ymax
E-33	Buffer limit of 20 characters reached	Wait for CR or CNTRL-A

ALSO SEE:

fget, cget

NAME:

intoffset - object intensity offset

SYNOPSIS:

```
intoffset("objname",val);

float val:  0.0 to 1.0
```

DESCRIPTION:

The intensity range of the specified object is determined by the parameter 'val'. If 'val' is one, the maximum intensity range is achieved. If the value is zero, the intensity is constant and the image has no depth-cueing.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-5	Specified object does not exist	Return
E-30	Parameter 'val' not 0.0 to 1.0	Return

ALSO SEE:

intscale

NAME:

intscale - modify object intensity scale

SYNOPSIS:

```
intscale("objname",val,cut);

float val:   0.0 to 1.0

int cut:    0 or 1
```

DESCRIPTION:

The intensity range of an object, which gives a three-dimensional object its depth cueing, is determined by the parameter 'val'. The range of 'val' is from zero to one. If 'val' is one the maximum intensity range is obtained; if it is zero the object has no depth cueing.

The parameter 'cut' provides for the use of a screen cut-off plane. If the value of 'cut' is one then the cut-off plane provisions are in effect. If the value is zero there will be no screen cut-off plane.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-5	Specified objname does not exist	Return
E-30	Parameter 'val' not 0.0 to 1.0	Return
E-31	Parameter 'cut' not 0 or 1	Return

ALSO SEE:

intoffset

NAME :

lamp - light/extinguish specified lamp

SYNOPSIS:

```
lamp(num,action);

int num:  0 - 31

int action:  OFF / 00   turn lamp off
              ON / 01   turn lamp on
```

DESCRIPTION:

This routine lights/extinguishes the specified lamp depending on the specified 'action' parameter. An 'action' value of OFF will extinguish the specified lamp, while a value of ON will light the appropriate lamp. An improper lamp 'num' or 'action' will result in no action being taken.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-23	Improper lamp 'num' specified	Return
E-24	Improper 'action' value specified	Return

ALSO SEE:

fstog

NAME:

lghtpen - set lightpen sensitivity of
the picture,object or elements

SYNOPSIS:

```
lghtpen(action,num,"objname","ele1","ele2",....,"ele10");

int action:  ON / 01    set lightpen sensitivity
             OFF / 00   clear lightpen sensitivity

int num:    PIC / 00    picture
            OBJ / -1    object
             1 - 10     elements
```

To set sensitivity of the entire picture:

```
lghtpen(action,PIC);
```

To set sensitivity of an object:

```
lghtpen(action,OBJ,"objname");
```

To set sensitivity of elements of a specific object:

```
lghtpen(action,num,"objname","ele1","ele2",...);
```

DESCRIPTION:

The user can specify which picture segments will be lightpen sensitive. These elements designated as lightpen sensitive will be affected by lightpen interaction with the vector general display screen.

The parameter 'num' specifies the number of element names being passed in this routine and may vary from 1 to 10.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-5	Specified objname does not exist	Return
E-6	Specified element does not exist	Skip element

E-18	Parameter 'num' not -1 to 10	num = 10
E-22	Element not linked to specified object	Skip element

ALSO SEE:

penhit, clrhit

NAME:

line - draw a line

SYNOPSIS:

```
2-dimensional coordinate system
  VR/VA  :  line(parx,pary);
  VRX/VAX:  line(pary);
  VRY/VAY:  line(parx);
  VRZ/VAZ:  not allowed

3-dimensional coordinate system
  VR/VA  :  line(parx,pary,parz);
  VRX/VAX:  line(pary,parz);
  VRY/VAY:  line(parx,parz);
  VRZ/VAZ:  line(parx,pary);

float parx,pary,parz;
```

DESCRIPTION:

A line is drawn from the present beam location to a position (x,y,z). The location of (x,y,z) depends upon the vector type and the specified parameters.

If the vector type is absolute, the location of (x,y,z) will be (parx,pary,parz). If an auto-increment mode is in effect, the designated auto-increment coordinate is left out of the parameter list and is incremented by the designated increment to find the termination point of the line. Under absolute vector types, parameters which fall outside the defined coordinate system will assume a value of the nearest coordinate within the defined system.

If the vector type is relative, the location of (x,y,z) will be the present beam location plus the specified parameter values. As with absolute vectors, auto-increment modes do not specify the designated auto-increment coordinate in the parameter list. Under relative vector types, parameters which exceed one-half the coordinate range will be set to zero.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-1	Routine has been called outside a drawele() block	Return
E-2	No space available in display list	Return
E-15	Parameter parx out-of-bounds	parx = xmin or xmax or 0.0
E-16	Parameter pary out-of-bounds	pary = ymin or ymax or 0.0
E-17	Parameter parz out-of-bounds	parz = zmin or zmax or 0.0

ALSO SEE:

setvector

NAME:

manint - Clear or return manual interrupt count

SYNOPSIS:

```
manint(action);

int action:  CLEAR / 00     set count to zero
             COUNT / 01     return count
```

DESCRIPTION:

This routine either returns the present value of the manual interrupt count or resets the count to zero. Each time the manual interrupt function switch is depressed the manual interrupt count is incremented by one.

An 'action' value of COUNT will return to the user the present manual interrupt count, while a value of CLEAR will reset the count to zero. An improper 'action' value will result in returning a value of zero but not affecting the manual interrupt count.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-24	Improper 'action' value	Return 0

NAME:

move - move to the specified location

SYNOPSIS:

```
2-dimensional coordinate system
  VR/VA   :  move(parx,pary);
  VRX/VAX:   move(pary);
  VRY/VAY:   move(parx);
  VRZ/VAZ:   not allowed

3-dimensional coordinate system
  VR/VA   :  move(parx,pary,parz);
  VRX/VAX:   move(pary,parz);
  VRY/VAY:   move(parx,parz);
  VRZ/VAZ:   move(parx,pary);

float parx,pary,parz;
```

DESCRIPTION:

The beam is moved from the present position to a position (x,y,z). The location of (x,y,z) depends upon the vector type and the specified parameters.

If the vector type is absolute, the location of (x,y,z) will be (parx,pary,parz). If an auto-increment mode is in effect, the designated auto-increment coordinate is left out of the parameter list and is incremented by the designated increment to find the termination point. Under absolute vector types, parameters which fall outside the defined coordinate system will assume a value of the nearest coordinate within the defined system.

If the vector type is relative, the location of (x,y,z) will be the present beam location plus the specified parameter values. As with absolute vectors, auto-increment modes do not specify the designated auto-increment coordinate in the parameter list. Under relative vector types, parameters which exceed one-half the coordinate range will be set to zero.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-1	Routine has been called outside a drawele() block	Return
E-2	No space available in display list	Return
E-15	Parameter parx out-of-bounds	parx = xmin or xmax or 0.0
E-16	Parameter pary out-of-bounds	pary = ymin or ymax or 0.0
E-17	Parameter parz out-of-bounds	parz = zmin or zmax or 0.0

ALSO SEE:

setvector

NAME:

object - link elements to specified object name

SYNOPSIS:

```
object(num,"objname","ele1","ele2",.....,"ele10");

int num:  1 - 10
```

DESCRIPTION:

Associates with the named object each of the listed elements. This picture segment, or element grouping will be referenced by the object name specified in this routine.

Every element must be linked to at least one object in order for it to be displayed on the Vector General screen. A user can link one or more element names to an object. Elements can be linked to an object by one or several calls to this routine. An element can be linked to several different objects, or one element may be linked several times to the same object.

If an object or element has been erased from the display screen, a user can redisplay the desired object or elements by again establishing the desired object-element association.

The parameter 'num' specifies the number of element names being passed in this routine and may vary from 1 to 10.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-6	Specified element does not exist	Skip element
E-8	Maximum number of elements exceeded	Return
E-9	Maximum number of objects exceeded	Return
E-12	Add element error	Return
E-18	Parameter 'num' not 1 to 10	num = 10
E-21	Add object error	Return

object		object
E-32	Element already linked	Skip element

ALSO SEE:

drawele, copyele, erase, remove

NAME:

offset - offset picture coordinate axis

SYNOPSIS:

```
offset(xoff,yoff);

float xoff,yoff:  -1.0 to +1.0
```

DESCRIPTION:

This routine will alter the positioning of the entire picture on the display screen. Normal or default picture positioning is centered on the display screen and corresponds to offset(0.0,0.0). Both the X and Y axes may be altered with this routine. Parameters outside the range -1.0 to +1.0 will result in no altering of the specified coordinate axis.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-15	Specified xoff value out-of-bounds	Skip X axis altering
E-16	Specified yoff value out-of-bounds	Skip Y axis altering

ALSO SEE:

trans

NAME:

penhit - determine if object/element lightpen hit

SYNOPSIS:

```
penhit(num,"objname","elename");

int num:  OBJ / -1      check penhit on object
          ELE / +1      check penhit on element
```

To check for lightpen hit on object 'tree':

```
penhit(OBJ,"tree");
```

To check for lightpen hit on element 'branch5' of object 'tree':

```
penhit(ELE,"tree","branch5");
```

DESCRIPTION:

This routine checks to see if an object or element within an object has had a lightpen hit. The routine returns to the user a one (1) if there has been a lightpen hit on the element or object, otherwise a zero (0) is returned. When testing for a lightpen hit on an object only the first two parameters are necessary, and a one is returned if any element within the object has had a lightpen hit.

The clrhit() routine must be called to clear an object or element of lightpen hits. Invalid parameters will result in a zero being returned to the user.

For an element to register a lightpen hit the element's lightpen sensitivity must be activated with the lghtpen() routine and the lightpen, with the sense switch selected, must be held over the element.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-5	Indicated objname does not exist	Return 0
E-6	Indicated elename does not exist	Return 0

E-22	Element not linked to specified object	Return 0

ALSO SEE:

lghtpen, clrhit

NAME:

place - position object

SYNOPSIS:

```
place("objname",xabs,yabs [,zabs] );

float xabs,yabs,zabs;
```

DESCRIPTION:

The specified object, with all its elements, is positioned at absolute_location (xabs,yabs,zabs).

The z parameter is required only when the coordinate system is defined as three dimensional, and will be ignored if included under a two-dimensional system.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-5	Specified object does not exist	Return
E-15	Specified xabs out-of-bounds	xabs = xmin or xmax
E-16	Specified yabs out-of-bounds	yabs = ymin or ymax
E-17	Specified zabs out-of-bounds	zabs = zmin or zmax

ALSO SEE:

offset, trans

NAME:

posit - return coordinate value of specified device cursor

SYNOPSIS:

```
posit(dev,coord);

int dev:   LPEN / 00     lightpen cursor
           TRK / 01      trackball cursor
           JOY / 02      joystick cursor

int coord:  X / 00       x coordinate
            Y / 01       y coordinate
            Z / 02       z coordinate

float posit();
```

DESCRIPTION:

This routine returns to the user a floating point value of the specified coordinate and device. The value returned is a value within the user specified coordinate system.

This routine will return the present position of the device cursor even though the cursor may not be currently displayed. A coordinate value of zero is returned if an improper parameter is received.

To obtain the floating point value from this routine the function declaration "float posit();" must be made in the users program.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-27	Improper 'dev' number	Return 0.0
E-28	Improper 'coord' number	Return 0.0
E-29	No 3D with LPEN or TRK	Return 0.0

ALSO SEE:

cursor

NAME:

printv - formatted printf routine for VG display

SYNOPSIS:

```
printv("elename",size,write,slant,xpos,ypos,"format"
              [,arg1,arg2,....,arg10] );

int size:  SZ / 00      use previous character size
          SZ1 / 0100    size = 100 col x 60 lines
          SZ2 / 0120    size = 81 col x 41 lines
          SZ3 / 0140    size = 60 col x 30 lines
          SZ4 / 0160    size = 32 col x 16 lines

int write:  HOR / 00    horizontal printing
            VER / 0200  vertical printing

int slant:  SLNT / 00   slanted printing
           NSLNT / 01   no slanted printing

float xpos,ypos;      absolute coordinates
```

A sample printed at (-25.0,-10.0) absolute:

```
    printv("test",SZ4,HOR,SLNT,-25.0,-10.0,
                    "bravo[5] = %d",bravo[5]);
```

DESCRIPTION:

This routine is a formatted print routine similar to the printf() routine except output is to the VG display screen. The routine is a combination of the charele() and printf() routines. The routine uses the first 6 parameters as the charele() routine does in positioning and initializing an element structure. Coordinate parameters which fall outside the defined coordinate system will assume a value of the nearest coordinate within the defined system.

Unlike charele() the routine allows multiple calls upon the same element, each call updating the element's character string. The routine allows for formatted output strings of up to 100 characters before an error message is displayed.

Each new element declared in a printv() routine uses up 57 words of the available display list so care must be taken not to declare too many new elements in this routine. It should be remembered that an element may

be used repeatedly with this routine.

Elements originally declared elsewhere as either vector drawing elements or character drawing elements should not be used in this routine. In other words, element names found in this routine should be unique and should not be found in other drawele() or charele() routines.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-2	No more space available in dislay list	Return
E-8	Maximum number of elements exceeded	Return
E-15	Parameter xpos out-of-bounds	x = xmin or xmax
E-16	Parameter ypos out-of-bounds	y = ymin or ymax
E-34	Formatted output length exceeds 100 characters	Chop to 1st 100 char
E-35	Improper 'write' parameter	write = HOR
E-36	Improper 'slant' parameter	slant = NSLNT
E-37	Improper 'size' parameter	size = SZ4

ALSO SEE:

charele

NAME:

pscale - scale the picture

SYNOPSIS:

```
pscale(val);

float val:  0.0 to 1.0
```

DESCRIPTION:

This routine will scale the entire picture. Normal or default picture scaling corresponds to pscale(1.0). An improper pscale() parameter will result in the picture scale remaining the same as before calling this routine.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-30	Parameter 'val' not 0.0 to 1.0	Return

ALSO SEE:

scale

NAME:

refresh - change refresh rate

SYNOPSIS:

```
refresh(rate);

int rate:  1 to 8
```

DESCRIPTION:

This routine will change the refresh rate of the VG display screen. Normal or default refresh rate is 40 Hz and corresponds to refresh(3). To determine the refresh rate for different values use the following equation:

120 / rate = new refresh rate

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-30	Parameter 'rate' not 1 to 8	Return

NAME:

remove - erase the indicated portion of the picture and remove the associated element block from the display list

SYNOPSIS:

```
remove(num,"ele1","ele2",.....,"ele10");

int num:    PIC / 00     all elements
             1 - 10      specified elements
```

To remove the entire picture:

```
remove(PIC);
```

To remove element structures:

```
remove(num,"ele1","ele2",..);
```

DESCRIPTION:

The entire picture, or every occurrence of the named elements are erased from the vector general display screen. Each named element structure will be removed from the display list. The routine removes all occurances of the specified elements including elements constructed with the copyele() routine.

To redisplay any portion of the removed picture requires that the user rebuild each element and relink it to the appropriate objects.

The parameter 'num' specifies the number of element names being passed in this routine and may vary from 1 to 10.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-6	Specified element does not exist	Skip element
E-18	Parameter 'num' not 1 to 10	num = 10

ALSO SEE:

erase, display

NAME:

rotate - rotate an object

SYNOPSIS:

```
rotate("objname",xradians,yradians [,zradians] );

float xradians,yradians,zradians;
```

DESCRIPTION:

The named object will be rotated about the x, y, z coordinate axis. The parameters, given in radian measure, specify the degree of rotation desired about each axis.

The z parameter is required only when the coordinate system is defined as three dimensional and will be ignored otherwise.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-5	Specified objname does not exist	Return

NAME:

scale - modify object scale

SYNOPSIS:

```
scale("objname", scale);

float scale:  0.0 to 1.0
```

DESCRIPTION:

All elements associated with the named object are scaled by the value scale at display time. The range of scale is from zero to one. Normal or default object scaling corresponds to scale("objname",1.0).

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-5	Specified objname does not exist	Return
E-30	Parameter 'scale' not 0.0 to 1.0	Return

ALSO SEE:

pscale

NAME:

setvector - specify the vector type and vector mode

SYNOPSIS:

```
setvector(vtype,vmode [,inc][,scale] );

int vtype:
        VR    - vector relative
        VRX   - vector relative auto-increment x
        VRY   - vector relative auto-increment y
        VRZ   - vector relative auto-increment z
        VA    - vector absolute
        VAX   - vector absolute auto-increment x
        VAY   - vector absolute auto-increment y
        VAZ   - vector absolute auto-increment z
        INC2  - vector incremental 2-dimensional
        INCX  - vector incremental auto-increment x
        INCY  - vector incremental auto-increment y
        INC3  - vector incremental 3-dimensional

int vmode:
        LN  /  00 - line
        DSH / 020 - dashed line
        DOT / 040 - dotted line
        PNT / 060 - end point
        DD  /0120 - dash-dot-dash line
        DDD /0140 - dash-dot-dash line

float scale:
       MG / 128.0 - add the coordinate increments
                    to the high order bits of the
                    specified register.
      NMG / 0.0   - add the coordinate increments
                    to the low order bits of the
                    specified register.


float inc;
```

DESCRIPTION:

This routine specifies which one of the 12 vector types is to be drawn in the line, move, circle and arc instructions which follow. It also specifies the vector mode (i.e., line, dotted, dashed, etc.), and if required, the increment value for auto-increment vectors and the scale factor for incremental vectors. The coordinate values passed in the following draw

instructions are dependent on the vector type selected in this routine.

This must follow a drawele() call and preceed any draw instructions. This routine can be called any number of times within a draw element block.

The bracketed z parameters are required only when the coordinate system has been defined as three dimensional.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-1	Routine has been called outside a drawele() block	Return
E-2	No space available in display list	Return
E-10	Improper vector mode	vmode = LN
E-14	Improper vector type	vtype = VA
E-15	Specified xinc out-of-bounds	xinc = xmin or xmax or 0.0
E-16	Specified yinc out-of-bounds	yinc = ymin or ymax or 0.0
E-17	Specified zinc out-of-bounds	zinc = zmin or zmax or 0.0

NAME:

VA - vector absolute

SYNOPSIS:

```
drawele("elename");
    .
    .
    setvector(VA,vmode);
    move(coordx,coordy [,coordz] );
    line(coordx,coordy [,coordz] );
    circle(dir,centx,centy [,centz] );
    arc(dir,centx,centy [,centz],endx,endy [,endz] );
    .
    .
endele();

float coordx,coordy,coordz;
float centx,centy,centz,endx,endy,endz;
```

DESCRIPTION:

The x,y,z coordinates of absolute vectors are specified with respect to the origin, or zero position of the user defined coordinate system. Each x, y, z coordinate value references a unique point on the display screen. Under this vector type, coordinate parameters for move/line/arc/circle commands which fall outside the defined coordinate system will assume a value of the nearest coordinate within the defined system.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-1	Routine has been called outside a drawele() block	Return
E-2	No space available in display list	Return
E-10	Improper vector mode	vmode = LN
E-14	Improper vector type	vtype = VA

NAME:

VAX - vector absolute auto-increment x

SYNOPSIS:

```
drawele("elename");
    .
    .
    setvector(VAX,vmode,xinc);
    move(coordy [,coordz] );
    line(coordy [,coordz] );
    circle(dir,centy [,centz] );
    arc(dir,centy [,centz],endy [,endz] );
    .
    .
endele();

float xinc,coordy,coordz;
float centy,centz,endy,endz;
```

DESCRIPTION:

The vector absolute auto-increment x causes the initial x coordinate value to be incremented or decremented by the value specified by 'xinc'. With every move/line/arc/circle command the specified y and z coordinate values are updated while the x coordinate value is incremented by the constant value 'xinc'. For example, the point (x,y,z) becomes point (x+xinc, coordy,coordz). Under this vector type, coordinate parameters for move/line/arc/circle commands which fall outside the defined coordinate system will assume a value of the nearest coordinate within the defined system.

A 'xinc' parameter exceeding one-half the coordinate range will be set to zero.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-1	Routine has been called outside a drawele() block	Return
E-2	No space available in disolay list	Return

E-10	Improper vector mode	vmode = LN
E-14	Improper vector type	vtype = VA
E-15	Specified 'xinc' out-of-bounds	xinc = 0.0

NAME:

VAY - vector absolute auto-increment y

SYNOPSIS:

```
drawele("elename");
    .
    .
    setvector(VAY,vmode,yinc);
    move(coordx [,coordz] );
    line(coordx [,coordz] );
    circle(dir,centx [,centz] );
    arc(dir,centx [,centz],endx [,endz] );
    .
    .
endele();

float yinc,coordx,coordz;
float centx,centz,endx,endz;
```

DESCRIPTION:

The vector absolute auto-increment y causes the initial y coordinate value to be incremented or decremented by the value specified by 'yinc'. With every move/line/arc/circle command the specified x and z coordinate values are updated while the y coordinate value is incremented by the constant value 'yinc'. For example, the point (x,y,z) becomes the point (coordx,y+yinc,coordz). Under this vector type, coordinate parameters for move/line/arc/circle commands which fall outside the defined coordinate system will assume a value of the nearest coordinate within the defined system.

A 'yinc' parameter exceeding one-half the coordinate range will be set to zero.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-1	Routine has been called outside a drawele() block	Return
E-2	No space available in display list	Return

E-10	Improper vector mode	vmode = LN
E-14	Improper vector type	vtype = VA
E-16	Specified 'yinc' out-of-bounds	yinc = 0.0

NAME:

VAZ - vector absolute auto-increment z

SYNOPSIS:

```
drawele("elename");
    .
    .
    setvector(VAZ,vmode,zinc);
    move(coordx,coordy);
    line(coordx,coordy);
    circle(dir,centx,centy);
    arc(dir,centx,centy,endx,endy);
    .
    .
endele();

float zinc,coordx,coordy;
float centx,centy,endx,endy;
```

DESCRIPTION:

The vector absolute auto-increment z causes the initial z coordinate value to be incremented or decremented by the value specified by 'zinc'. With every move/line/arc/circle command the specified x and y coordinate values are updated while the z coordinate value is incremented by the constant value 'zinc'. For example, the point (x,y,z) becomes the point (coordx,coory,z+zinc). Under this vector type, coordinate parameters for move/line/arc/circle commands which fall outside the defined coordinate system will assume a value of the nearest coordinate within the defined system.

A 'zinc' parameter exceeding one-half the coordinate range will be set to zero.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-1	Routine has been called outside a drawele() block	Return
E-2	No space available in display list	Return

E-10	Improper vector mode	vmode = LN
E-14	Improper vector type	vtype = VA
E-17	Specified 'zinc' out-of-bounds	zinc = 0.0

NAME: VR - vector relative

SYNOPSIS:

```
drawele("elename");
    .
    .
    setvector(VR,vmode);
    move(deltax,deltay [,deltaz] );
    line(deltax,deltay [,deltaz] );
    circle(dir,centx,centy [,centz] );
    arc(dir,centx,centy [,centz],endx,endy [,endz] );
    .
    .
endele();

float deltax,deltay,deltaz;
float centx,centy,centz,endx,endy,endz;
```

DESCRIPTION:

Relative vectors specify an increment value that is to be added to or subtracted from the initial x,y,z beam position. For example, the point (x,y,z) becomes point (x+deltax,y+deltay,z+deltaz).

Under this vector type, coordinate parameters for move/line/arc/circle commands which exceed one-half the coordinate range will be set to zero.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-1	Routine has been called within a drawele() block	Return
E-2	No space available in display list	Return
E-10	Improper vector mode	vmode = LN
E-14	Improper vector type	vtype = VA

NAME:

VRX - vector relative auto-increment x

SYNOPSIS:

```
drawele("elename");
    .
    .
    setvector(VRX,vmode,xinc)
    move(deltay [,deltaz] );
    line(deltay [,deltaz] );
    circle(dir,centy [,centz] );
    arc(dir,centy [,centz],endy [,endz] );
    .
    .
endele();

float xinc,deltay,deltaz;
float centy,centz,endy,endz;
```

DESCRIPTION:

A vector relative auto-increment x causes the initial x coordinate value to be incremented or decremented by the constant value 'xinc'. With every move/line/arc/circle instruction that follows, the y and z coordinate values will be updated, while x is stepped by the value 'xinc'. For example, the point (x,y,z) becomes the point (x+xinc,y+deltay,z+deltaz).

Under this vector type, coordinate parameters for move/line/arc/circle commands which exceed one-half the coordinate range will be set to zero.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-1	Routine has been called outside a drawele() block	Return
E-2	No space available in display list	Return
E-10	Improper vector mode	vmode = LN
E-14	Improper vector type	vtype = VA
E-15	Specified 'xinc' out-of-bounds	xinc = 0.0

NAME:

VRY - vector relative auto-increment y

SYNOPSIS:

```
drawele("elename");
    .
    .
    setvector(VRY,vmode,yinc)
    move(deltax [,deltaz] );
    line(deltax [,deltaz] );
    circle(dir,centx [,centz] );
    arc(dir,centx [,centz],endx [,endz] );
    .
    .
endele();

float yinc,deltax,deltaz;
float centx,centz,endx,endz;
```

DESCRIPTION:

A vector relative auto-increment y causes the initial y coordinate value to be incremented or decremented by the constant value 'yinc'. With every move/line/arc/circle instruction that follows, the x and z coordinate values will be updated, while y is incremented by the value 'yinc'. For example, the point (x,y,z) becomes the point (x+deltax,y+yinc,z+deltaz).

Under this vector type, coordinate parameters for move/line/arc/circle commands which exceed one-half the coordinate range will be set to zero.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-1	Routine has been called outside a drawele() block	Return
E-2	No space available in display list	Return
E-10	Improper vector mode	vmode = LN
E-14	Improper vector type	vtype = VA
E-16	Specified 'yinc' out-of-bounds	yinc = 0.0

NAME:

VRZ - vector relative auto-increment z

SYNOPSIS:

```
drawele("elename");
    .
    .
    setvector(VRZ,vmode,zinc)
    move(deltax,deltay);
    line(deltax,deltay);
    circle(dir,centx,centy);
    arc(dir,centx,centy,endx,endy);
    .
    .
endele();

float zinc,deltax,deltay;
float centx,centy,endx,endy;
```

DESCRIPTION:

A vector relative auto-increment z causes the initial z coordinate value to be incremented or decremented by the constant value 'zinc'. With every move/line/arc/circle instruction that follows, the x and y coordinate values will be updated, while z is incremented by the value 'zinc'. For example, the point (x,y,z) becomes the point (x+deltax,y+deltay,z+incz).

Under this vector type, coordinate parameters for move/line/arc/circle commands which exceed one-half the coordinate range will be set to zero.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-1	Routine has been called outside a drawele() block	Return
E-2	No space available in display list	Return
E-10	Improper vector mode	vmode = LN
E-14	Improper vector type	vtype = VA
E-17	Specified 'zinc' out-of-bounds	zinc = 0.0

NAME:

INC2 - incremental vector; two-dimensional

SYNOPSIS:

```
drawele("elename");
    .
    .
    setvector(INC2,vmode,scale);
    move(deltax,deltay);
    line(deltax,deltay);
    .
    .
endele();

float deltax,deltay;

float scale:
    MG / 128.0  - add the coordinate increments to
                  the 7 high order bits of the
                  specified register.
   NMG /  0.0   - add the coordinate increments to
                  the 7 low order bits of the
                  specified register.
```

DESCRIPTION:

A two-dimensional relative vector that halves vector storage requirements and doubles the data rate. This vector type should be used when element storage space is critical. This reduced storage requirement results from a limitation on the maximum size of each relative increment. The values passed in move/line instructions are limited to approximately 3% of the user minimum and maximum coordinate ranges. Should coordinate parameters for move/line commands exceed the 3% limit, they will be reduced to satisfy the constraint.

The parameter 'scale' specifies whether this increment value should be added to the 7 high order bits of the specified coordinate or to the 7 low order bits. This scale factor then determines if the increments will be applied over a fine(NMG) or coarse(MG) grid.

Arcs and circles cannot be drawn with this vector type.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-1	Routine has been called outside a drawele() block	Return
E-2	No space available in display list	Return
E-14	Improper vector type	vtype = VA

NAME:

INC3 - incremental vector; three-dimensional

SYNOPSIS:

```
drawele("elename");
    .
    .
    setvector(INCX,vmode,scale);
    move(deltax,deltay,deltaz);
    line(deltax,deltay,deltaz);
    .
    .
endele();

float deltax,deltay,deltaz;

float scale:
    MG / 128.0  - add the coordinate increments to
                  the 7 high order bits of the
                  specified register.
   NMG /  0.0   - add the coordinate increments to
                  the 7 low order bits of the
                  specified register.
```

DESCRIPTION:

A three-dimensional relative vector that halves vector storage requirements and doubles the data rate. This vector type should be used when element storage space is critical. This reduced storage requirement results from a limitation on the maximum size of each relative increment. The values passed in move/line instructions are limited to approximately 3% of the user minimum and maximum coordinate ranges. Should coordinate parameters for move/line commands exceed the 3% limit, they will be reduced to satisfy the constraint.

The paramater 'scale' specifies whether this increment value should be added to the 7 high order bits of the specified coordinate or to the 7 low order bits. This scale factor then determines if the increments will be applied over a fine(NMG) or coarse(MG) grid.

Arcs and circles cannot be drawn with this vector type.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-1	Routine has been called outside a drawele() block	Return
E-2	No space available in display list	Return
E-14	Improper vector type	vtype = VA

NAME:

INCX - incremental vector; two-dimensional
auto-increment x

SYNOPSIS:

```
drawele("elename");
    .
    .
    setvector(INCX,vmode,xinc,scale);
    move(deltay,deltay);
    line(deltay,deltay);
    .
    .
endele();
```

```
float xinc,deltay;

float scale:
    MG / 128.0  - add the coordinate increments to
                  the 7 high order bits of the
                  specified register.
   NMG /  0.0   - add the coordinate increments to
                  the 7 low order bits of the
                  specified register.
```

DESCRIPTION:

A two-dimensional relative vector that halves vector storage requirements and doubles the data rate. This vector type should be used when element storage space is critical. This reduced storage requirement results from a limitation on the maximum size of each relative increment. The values passed in move/line instructions are limited to approximately 3% of the user minimum and maximum coordinate ranges. Should the coordinate parameters for move/line commands exceed the 3% limit, they will be reduced to satisfy the constraint.

The parameter 'scale' specifies whether this increment value should be added to the 7 high order bits of the specified coordinate or to the 7 low order bits. This scale factor then determines if the increments will be applied over a fine(NMG) or coarse(MG) grid.

This incremental, auto-increment vector causes the x coordinate value to be incremented or decremented by the constant value 'xinc', while the y coordinate value is increment by small relative values. The 'xinc' value need not be limited to 3% of the

coordinate range but may not exceed one-half the coordinate range or it will be set to zero.

Arcs and circles cannot be drawn with this vector type.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-1	Routine has been called outside a drawele() block	Return
E-2	No space available in display list	Return
E-14	Improper vector type	vtype = VA
E-15	Specified 'xinc' out-of-bounds	xinc = 0.0

NAME:

INCY - incremental vector; two-dimensional
auto-increment y

SYNOPSIS:

```
drawele("elename");
    .
    .
setvector(INCY,vmode,yinc,scale);
move(deltax,deltax);
line(deltax,deltax);
    .
    .
endele();

float yinc,deltax;

float scale:
    MG / 128.0  - add the coordinate increments to
                  the 7 high order bits of the
                  specified register.
    NMG /  0.0  - add the coordinate increments to
                  the 7 low order bits of the
                  specified register.
```

DESCRIPTION:

A two-dimensional relative vector that halves vector storage requirements and doubles the data rate. This vector type should be used when element storage space is critical. This reduced storage requirement results from a limitation on the maximum size of each relative increment. The values passed in move/line instructions are limited to approximately 3% of the user minimum and maximum coordinate ranges. Should the coordinate parameters of move/line commands exceed the 3% limit, they will be reduced to satisfy the constraint.

The parameter 'scale' specifies whether this increment value should be added to the 7 high order bits of the specified coordinate or to the 7 low order bits. This scale factor then determines if the increments will be applied over a fine(NMG) or coarse(MG) grid.

This incremental, auto-increment vector causes the y coordinate value to be incremented or decremented by the constant value 'yinc', while the x coordinate value is increment by small relative values. The 'yinc' value need not be limited to 3% of the

AD-A050 241 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 9/2
IMPLEMENTATION OF A USER INTERFACE WITH THE VECTOR GENERAL GRAP--ETC(U)
DEC 77 D C ENDICOTT, N P MARTINO

UNCLASSIFIED NL

3 OF 3
AD A050241

END
DATE FILMED
3-78
DDC

coordinate range but may not exceed one-half the coordinate range or it will be set to zero.

Arcs and circles cannot be drawn with this vector type.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-1	Routine has been called outside a drawele() block	Return
E-2	No space available in display list	Return
E-14	Improper vector type	vtype = VA
E-16	Specified 'yinc' out-of-bounds	yinc = 0.0

NAME:

sysinit - vector general display initialization

SYNOPSIS:

```
sysinit(devnum);

int devnum:  0 or 1
```

DESCRIPTION:

This routine establishes a link with the vector general, initializes its display system and sets all the user default parameters. The routine allows for specification of which VG device the user desires to utilize. The VG devices are numbered 0 and 1. Should the system be unable to allocate the specified VG device, allocation of the other VG device will be attempted.

This routine must be called before any other display instructions.

DIAGNOSTICS:

If initialization of the specified VG device cannot be completed, an error message will be printed out and the system will attempt to initialize the other VG device. The system will terminate the process only if both initialization attempts fail.

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-27	Improper VG devnum specified	devnum = 0

NAME:

trans - translate object

SYNOPSIS:

```
trans("objname",deltax,deltay [,deltaz] );

float deltax,deltay,deltaz;
```

DESCRIPTION:

All of the elements associated with the named object are translated from their present location (X,Y,Z) to (X+deltax,Y+deltay,Z+deltaz).

The z parameter is required only when the coordinate system is defined as three dimensional, and will be ignored if included under a two-dimensional system.

DIAGNOSTICS:

All errors will be printed on the PDP-11 terminal screen.

ERROR	CAUSE	ACTION
E-5	Specified object does not exist	Return
E-15	Specified deltax out-of-bounds	deltax = 0.0
E-16	Specified deltay out-of-bounds	deltay = 0.0
E-17	Specified deltaz out-of-bounds	deltaz = 0.0

ALSO SEE:

offset, place

NAME:

vgpicture - start refresh of VG picture display

SYNOPSIS:

```
vgpicture();
```

DESCRIPTION:

This routine must be called to initiate the display of elements on the VG display screen. The routine need be called only once. The display() routine may be called after this routine to activate or deactivate the display of all or parts of the display picture.

DIAGNOSTICS:

none

ALSO SEE:

display, erase, remove

NAME:

vgterm - terminate display and close VG device

SYNOPSIS:

```
vgterm();
```

DESCRIPTION:

This routine should be included at the end of the user's program to properly terminate the display and close the VG device. Should the user's program contain an endless loop, depressing the 'rubout' key will accomplish the same task.

DIAGNOSTICS:

none

LIST OF REFERENCES

1. Digital Equipment Corporation, PDP-11/34 Processor Handbook, 1976.

2. Digital Equipment Corporation, FORTRAN IV-PLUS User's Guide, 1975.

3. Emery, H. W., The Development of a Partitioned Segmented Memory Manager for the UNIX Operating System, M.S. Thesis, Naval Postgraduate School, 1977.

4. Newman, W. M. and Sproull, R. F., Principles of Interactive Computer Graphics, McGraw-Hill, 1973.

5. O'Dell, J. M., The Development of a Segmented Memory Manager for the UNIX Operating System with Applications in a Multiported Memory Environment, M.S. Thesis, Naval Postgraduate School, 1977.

6. Ritchie, D. M. and Thompson, K., "The UNIX Time-sharing System", Communications of the ACM, v. 17, no. 7, p. 365-375, July, 1974.

7. Ritchie, D. M., C Reference Manual, Bell Telephone Laboratories, Murray Hill, New Jersey, 1974.

8. Ritchie, D. M., UNIX Assembler Manual, Bell Telephone Laboratories, Murray Hill, New Jersey, 1974.

9. Robertson, M. D., An Extended Basic Compiler with Graphics Interface for the PDP-11/50 Computer, M.S. Thesis, Naval Postgraduate School, June, 1977.

10. Stankowski, B. J., The Design and Implementation of a General Purpose Interactive Graphics Subroutine Library, M.S. Thesis, Naval Postgraduate School, September 1976.

11. Thorpe L. A. and Raetz G. M., Design Manual for the Vector General Interface Display Unit, Naval Postgraduate School Technical Report NPS72Rr76032.

12. Thorpe L. A. and Raetz G. M., User Manual for the Vector General Display Unit, Naval Postgraduate School Technical Report NPS72Rr76031.

13. Vector General Inc., Graphics Display System Reference Manual, August 1974.

14. Vector General Inc., Graphics Display System Technical Manual, June 1974.

15. Visco, D. W., The Design of an Inter-processor Support System for an Interactive Graphics Package, M.S. 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 G. A. Rahe, Code 52Ra Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
5.	LCDR Stephen T. Holl, USN, Code 52Hl Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
6.	Computer Laboratory, Code 52ec Department of Computer Science Naval Postgraduate School Monterey, California 93940	2
7.	LT David C. Endicott, USN Route 1, Box 461-E Stoneville, North Carolina 27048	1
8.	LT Nathan P. Martino, USN 1894 Bailey Road Concord, California 94521	1