

DISC FILE COPY

(2)

OCT 1988

TRAC - F - TM - 1288

ACN 16306

TRADOC ANALYSIS COMMAND GRAPHICS CAP
USERS MANUAL
(REVISION OF MACRO - FLIK)

AD-A199 465



DTIC
ELECTE
OCT 11 1988
S D
ca

Fort Leavenworth

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

US ARMY

TRADOC ANALYSIS COMMAND - FORT LEAVENWORTH
(TRAC - FLVN)

OPERATIONS DIRECTORATE
FORT LEAVENWORTH, KANSAS 66027

88 10 7 08 0

Technical Memorandum TRAC-F-TM-1288
Oct 1968

TRADOC Analysis Command-Fort Leavenworth (TRAC-FLVN)
Operations Directorate, Technology Applications Branch
Fort Leavenworth, Kansas 66027-5200

TRADOC ANALYSIS COMMAND GRAPHICS CAP

USER'S MANUAL

(REVISION OF MACRO-FLIK)

by

R.H. "Pete" Kaeding

ACN 16306

The views, opinions, and/or findings contained in this report are not to be construed as an official Department of the Army or TRAC-FLVN position, policy, or decision unless so designated by authorized documents issued and approved by the Department of the Army.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
1a. REPORT SECURITY CLASSIFICATION			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for Public Release Distribution Unlimited		
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) TRAC-F-TM-1288			5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION TRAC-FLVN		6b. OFFICE SYMBOL (If applicable) ATRC-FOC	7a. NAME OF MONITORING ORGANIZATION		
6c. ADDRESS (City, State, and ZIP Code) Commander TRAC-FLVN (ATTN: ATRC-FOC) Ft Leavenworth, KS 66027-5200			7b. ADDRESS (City, State, and ZIP Code)		
8a. NAME OF FUNDING / SPONSORING ORGANIZATION		8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER		
8c. ADDRESS (City, State, and ZIP Code)			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.
					WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) TRADOC Analysis Command Graphics Cap Users Manual					
12. PERSONAL AUTHOR(S) R.H. "Pete" Kaeding					
13a. TYPE OF REPORT Final Report		13b. TIME COVERED FROM 5-88 TO 10-88	14. DATE OF REPORT (Year, Month, Day) 1988 October		15. PAGE COUNT 38
16. SUPPLEMENTARY NOTATION					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP			
19. ABSTRACT (Continue on reverse if necessary and identify by block number) This document is a revision of Technical Memorandum TRAC-F-TM-1473, Macro-Ft Leavenworth Improved Kellner Graphics Interface Package (Macro-FLIK) User's Manual. This revision was necessitated by a number of upgrades to the Macro-FLIK software package. Graph Cap (revised Macro-FLIK) consists of a few higher level. FORTRAN subroutines intended to provide the VAX software developer, who has virtually no graphics background, an easy way to incorporate Ramtek graphics into application programs. Graphics effects such as: conventional, pull-down and/or pop-up menuing, point data display in a variety of forms, data manipulation, a variety of shape displays, and picture preservation and manipulation are easily achieved by the programmer calling the appropriate Graph Cap routine. This revision includes a "pocket" reference guide for incorporating graphics into applications programs.					
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION		
22a. NAME OF RESPONSIBLE INDIVIDUAL			22b. TELEPHONE (Include Area Code)		22c. OFFICE SYMBOL

ACKNOWLEDGEMENTS

I would like to express my appreciation to Mr Timothy Bailey, Chf, Computer Systems Division and Mr Ronald Magee, Dir, Operations Directorate for their guidance and evaluation of the Graph Cap package. Several of the original concepts and later enhancements were their recommendations. Also I want to express my gratitude to the Technology Applications Branch Team: Mr Timothy Daniels and Spc Mike Chenault, for their contributions of Ramtek graphics expertise and software debugging.



Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Section
A-1	

Table of Contents

	<u>Page</u>
TITLE PAGE.....	i
DD FORM 1473, Report Documentation Page.....	iii
ACKNOWLEDGEMENTS.....	iv
TABLE OF CONTENTS.....	v
ABSTRACT.....	vii
 MAIN REPORT	
Background.....	1
Preparation.....	2
Initialization.....	2
Graphics input.....	6
Point data display.....	9
Status and picture preservation/manipulation.....	15
Pull-down and pop-up menuing.....	20
Shape manipulation.....	23
General routines.....	24
Summary.....	26
APPENDIX A. REFERENCE GUIDE.....	A-1
 DISTRIBUTION LIST	

FIGURES

<u>Number</u>	<u>Page</u>
1 Text size and direction samples.....	25

ABSTRACT

This document is a revision of Technical Memorandum TRAC-F-TM-1473, Macro-Ft Leavenworth Improved Kellner Graphics Interface Package (Macro-FLIK) User's Manual. This revision was necessitated by a number of upgrades to the Macro-FLIK software package. Graph Cap (revised Macro-FLIK) consists of a few higher level, FORTRAN subroutines intended to provide the VAX software developer, who has virtually no graphics background, an easy way to incorporate Ramtek graphics into application programs. Graphics effects such as: conventional, pull-down and/or pop-up menuing, point data display in a variety of forms, data manipulation, a variety of shape displays, and picture preservation and manipulation are easily achieved by the programmer calling the appropriate Graph Cap routine. This revision includes a "pocket" reference guide for incorporating graphics into applications programs.

1. Background. The majority of the graphics display hardware used by the TRADOC Analysis Command (TRAC) is manufactured by Ramtek. A very low-level software package is provided by Ramtek to accompany this hardware. This package represents the "nuts and bolts" software and, as such, is not very user friendly. To simplify communications with the hardware, a slightly higher level set of routines, designed more with the user in mind, is preferred. Mr. Al Kellner, TRAC-White Sands Missile Range (TRAC-WSMR) attempted to fill this bill by developing what has come to be known within TRAC as the Kellner Graphics Interface Package (KGIP). This package, currently consisting of approximately 208 routines, is considerably more user friendly, and it provides the graphics programmer with a means of utilizing most of the Ramtek's capabilities. The Technology Applications Branch Graphics Team, Operations Directorate, TRAC-Ft Leavenworth (TAB-GT), confronted with several application software packages using KGIP variants, set out to improve the package in several ways.

a. At least seven variants of the package were known to exist. Our initial objective was to provide a single version which would be compatible with all known application packages and all TRAC hardware configurations. To accomplish this, TAB-GT designed and developed a number of routines to make KGIP "intelligent" enough to dynamically determine the hardware configuration on which it's operating as well as that on which it was intended to operate. A byproduct of the evolution of this new package, Fort Leavenworth Improved KGIP (FLIK), was a much better organized package. This improved organization was primarily due to the consolidation of several families of routines (routines performing very nearly the same function) into what we call "super" routines. Other improvements incorporated into FLIK addressed KGIP deficiencies, omissions, or, in some cases, just represented additional sophistication. The end product, FLIK, provides the graphics programmer with a flexible, powerful, and relatively user friendly package.

b. Unfortunately, "user friendly" is "in the eye of the beholder." The typical applications programmer, faced with deadlines and capability requirements to satisfy, doesn't have the time to devote to graphics self-education. This led TAB-GT to develop a package of higher level graphics routines called Macro-FLIK. The Macro-FLIK software package has since been upgraded and renamed Graph Cap. Graph Cap provides several additional capabilities and a good deal more sophistication. Graph Cap is intended to provide the application software developer with an easy to use, minimal set of routines which will enable him to incorporate VAX/Ramtek graphics into any application package. These macro-routines will perform the necessary FLIK calls to execute the desired graphics effect. The following paragraphs discuss this macro-routines package subdivided by effect.

c. On first glance, these routines may seem simple (that's good) and/or restrictive (that's a function of keeping it simple) but the author is certainly open to, and appreciative of, ideas that might improve the utility of the package.

2. Preparation. To use the Graph Cap package the applications programmer must link his application software appropriately. System logicals (automatically assigned at your login at TRAC-FLVN) OMACRO, OKGL, ORMSTN, and OMARQ, will point your link to the appropriate directories. A sample link follows:

LINK APPLICATION, OMACRO:MACRO/OPT

Where MACRO.OPT is as follows:

OMACRO:A/LIB	(The Graph Cap library)
OKGL:A/LIB	(The FLIK library)
ORMSTN:A/LIB	(The Ramtek station/chassis info lib)
OMARQ:QIO_MARQ/LIB	(The vendor-provided Ramtek Marquis driver software library)

3. Initialization. Before communication can be established with one of our Ramtek color monitors, the user must execute a series of initialization routines. These range from opening and assigning a channel to the Ramtek controller and associating the appropriate monitor and graph tablet with your workstation, to defining and loading colors into the Ramtek video lookup table (VLT) for later reference. Graph Cap simplifies this procedure by requiring the application programmer to call only one routine, one time, with no calling arguments, as follows:

INIT_RAMTEK. Calling format:

CALL INIT_RAMTEK

This routine performs all of the necessary initialization to establish the link between the user's terminal and graphics workstation based on several interactive queries. The queries (in bold type), typical response, and discussion of possible responses are shown below.

KRMINIT: Enter Ramtek logical (like RMA1:) RMA0:

The user must enter the four-character logical associated with the particular work station (there is a label on each workstation at TRAC-FLVN Central Computer Facility (CCF) and Wargame Computer Facility (WCF)).

Draw graphics to Ramtek? [Y/N] [Y] <return> or Y

Save graphics to meta-file? [Y/N] [N] <return> or N

The Vector-in-Commander (VIC) wargame simulation utilizes FLIK for its graphics displays. The above two prompts are necessary for the several modes of execution of that application. Appropriate responses are Yes or No. Typically an application programmer using Graph Cap to incorporate graphics into his own application will respond to these prompts by accepting the default (simply pressing the return key).

Erase screen? [Y/N] [Y] <return> or Y

The hardware dip switch settings on all Ramteks at TRAC-FLVN force software resets to clear all monitors. In this way a display can be left on a monitor for later editing and/or preservation. Appropriate responses are Yes or No. Again, the user would typically accept the default response by pressing the return key.

RELOAD STATUS FROM A PREVIOUS RUN? [Y/N] [N] <return> or N

Appropriate responses are Yes or No. This option allows the user to reload the status of curve displays for review/modification. Typically the user's response will be No. For more information, see paragraph 6a.

COLOR FILE: NEW [N], OLD [O], DEFAULT [D]? N

The first time this software is executed, the user will likely select responses New or Default (by pressing the return key). Typically thereafter, the Old option would be selected to use a previously created and saved color file. The prompts which follow are a result of having selected the New (N) option. Had the user selected Default, the next prompt to appear would be "SAVE THE COLOR FILE JUST CREATED?" (discussed below). Had the user selected Old, he would be prompted to enter the filename of that old color file, followed by the prompt "DO YOU WANT TO SEE YOUR COLOR SCHEME?" (discussed below).

Before discussing the remaining prompts and responses, user's should have at least a cursory understanding of color file production. The Graph Cap color scheme generation software is more sophisticated than it needs to be. For the typical application, the user should simply specify one buffer and build an ample number (current software restriction is 32) of colors by responding to the prompts that follow. However, the capability exists for the educated user to devise a sophisticated overlay scheme. That procedure is described in more detail in the following paragraphs.

Current TRAC Ramtek hardware is configured with anywhere between 8 and 16 usable refresh memory planes per station which will allow addressing a maximum of 256 (2^8) to 4,096 (2^{12}) colors loaded into the VLT. By cleverly loading the VLT and sacrificing

colors, an overlaying effect can be achieved. This clever loading (the details of which are beyond the scope of this manual) is accomplished for the user by a FLIK routine which requires that the user define the color scheme in terms of overlay buffers and colors within each buffer. A buffer consists of a number of refresh memory planes where each plane provides an additional power of two-color capacity. This means that a buffer consisting of three planes would allow the user to load eight colors (2×3). Of course, the advantage to such a buffering/overlay scheme is that graphics drawn in any buffer can be "hidden from view" by drawings in a higher number buffer, but become visible again once the higher buffer drawings are erased.

Currently, the Graph Cap software restricts the user to eight buffers using as many planes as are available at his current work station (but, as mentioned previously, limited to a maximum of five planes per any one buffer). Of course, a single buffer would provide no overlays, simply the generation of 32 colors (five planes or 2×5 colors). The responses made to the sample prompts below would create an overlay scheme consisting of 3 overlay buffers of 32, 2, and 4 colors (5, 1, and 2 planes) respectively.

If the user plans to incorporate pull-down/pop-up menus (see paragraph 7), the software will generate a temporary top level "artificial" buffer of 2 planes (if available at the current station; otherwise, the user is warned that graphics drawn in his top buffer may be "overdrawn" if he uses pull-down/pop-up menus). It is in this temporary buffer that these special menus are drawn, thus allowing them to overlay the user's graphics work without (except in the case just mentioned) ill effect. The pull-down/pop-up grid, text, and highlight colors respectively will be duplicates of the first three colors (excluding the "clear" color number 1) in the user-generated top buffer (if it contains three colors).

For example, if the user, in responding to the sample prompts below, selected colors red, green, and blue as the three colors in buffer number 3 (his top buffer), and then chose to use pull-down menus in his software, the menus would appear with green text on a red background (grid) and upon selection would be highlighted in blue (not a pretty sight). If the user's current work station has more than the eight planes used in this color file definition available, a temporary buffer 4 will be generated with those three colors. If, however, this station has only eight planes available, then the pull-downs/pop-ups will be drawn in buffer number 3, effectively erasing the user's graphics drawn in that buffer. Although the software doesn't prohibit this type of potential damage, it does warn the user at initialization.

The following prompts actually appear on the Ramtek monitor, and

the user's responses are entered via the graph tablet and puck/pen. For this reason, I've chosen to show the response(s) in < >.

SELECT NUMBER OF OVERLAY BUFFERS <3>

Hardware limit (for some of TRAC's hardware) is 8. Entering 1 is acceptable, but implies no "overlying."

SELECT # OF PLANES IN OVERLAY BUFFER #1 <5>

This prompt will appear for each buffer. Current software limit is 32 colors (five planes as in this example) per buffer. As discussed above, the user is limited to the number of planes available at his current station and a total of eight buffers.

SELECT # OF PLANES IN OVERLAY BUFFER #2 <1>

SELECT # OF PLANES IN OVERLAY BUFFER #3 <2>

Note the total number of planes for all buffers is at most (in this case, exactly) eight.

OVERLAY BUFFER CONFIGURATION SELECT COLORS USING THE TABLET

An empty matrix representing the user-specified color scheme is displayed at this time, along with a color pallet of available colors from which the user may select. An "X" appears in each matrix color box and is replaced by the color selected from the pallet as the user progresses through his color scheme definition.

DO YOU WISH TO MAKE ANY CHANGES? <Y>

Appropriate "selections" are Yes or No. If the user selects No, the next prompt to appear will be the "SAVE THE COLOR FILE..." prompt.

SELECT COLOR TO CHANGE

Using the graph tablet and puck, the user selects the box in his (now filled) color matrix which contains the color to be changed.

SELECT NEW COLOR FROM PALLET

The user selects the replacement color from the color pallet. These two prompts are repeated until the "key when finished" box is selected at which time prompts return to the CRT.

SAVE THE COLOR FILE JUST CREATED? Y

ENTER COLOR FILE NAME (NO EXTENSION) <filename>

START OVER? N

Appropriate responses are Yes or No. If the user selects Yes, this option allows him to build another color file by clearing his color matrix and recycling starting with the "SELECT NUMBER OF OVERLAY BUFFERS" prompt.

DO YOU WANT TO SEE YOUR COLOR SCHEME? [Y/N] [N] <return> or N

Appropriate responses are Yes or No. If the user selects Yes, his color scheme is listed to the CRT and the following prompt appears.

IS THIS COLOR SCHEME WHAT YOU WANT? Y

Appropriate responses are Yes or No. If the user selects No, he will return to the "BUILD A NEW..." prompt discussed above.

4. Graphics input. Input data displayed in menus on color monitors and selected via the workstation graphics tablet by light pen/puck was the first application addressed. Since nearly all application software requires user interface, this seemed a high payoff undertaking. The assumption made here is that graphically inputting data is better than keyboard entry because of its aesthetic appeal, user friendliness, and/or simplicity. An application programmer who accepts this assumption would then prefer to incorporate graphics input in his application package if it was not too burdensome to effect and was satisfactorily responsive. This three-routine conventional menu package attempts to give the user that capability. A more sophisticated menuing package, featuring pull-down and pop-up menus, is discussed in paragraph 7.

a. DRAW_MENU Calling format:

CALL DRAW_MENU (LOC_MENU, ERASE, MEN_BUFF, GRID_COLOR,
 TEXT_COLOR, HILITE_COLOR, NUM_BOXES, TEXT)

To keep it simple, this software offers the user four possible locations for menu display. He may have a menu displayed in each location simultaneously or choose to display menus only one at a time. The first calling argument, LOC_MENU, is a numeric code 1 through 4 identifying the desired location of the menu in a clockwise manner with 1 being top of screen (1=top, 2=right, etc.). The next argument is a logical variable indicating whether the this menu is to be automatically erased once a selection is made. If ERASE = .TRUE., the menu will be erased once a selection is made. The next four arguments are

interconnected. MEN_BUFF indicates the graphics overlay buffer (user defined by his responses to routine INIT_RAMTEK described above and the next three arguments (integers) indicate the corresponding colors in which the menu will display. NUM_BOXES notifies the software of how many entries the menu is to contain, and TEXT is a character array containing the actual text to appear in the menu. (User note: the menu size is dynamically determined so lengthy text should be reserved for the side menus). This and other calls to routines in this menu package may be clearer by looking at the sample application program shown in paragraph 4d.

b. MONI_MENU Calling format:

CALL MONI_MENU (MEN1, MEN2, MEN3, MEN4, MEN_OUT, MEN_BOX)

This routine is called to monitor (i.e., await graph tablet selection from) any combination of the menus previously displayed via DRAW_MENU. The user simply identifies which menu(s) is/are to be monitored by the first four arguments (1 indicating that the menu in that location is to be monitored and 0 if not). The software will then wait for the user to respond via the graph tablet (i.e., make a selection using the graphics pen/puck). The arguments MEN_OUT, and MEN_BOX return selection information. MEN_OUT is the code number representing the location of the menu from which the selection was made and MEN_BOX the corresponding box number within that menu.

c. ERAS_MENU Calling format:

CALL ERAS_MENU (MENU_NUMBER)

If the user has menus to be displayed from which multiple selections will be made, he will not want the menu to erase after each selection. By setting the erase "flag" accordingly when the menu is drawn this will not occur. However, a means of eventually erasing this menu may still be desirable. ERAS_MENU allows the user to erase any or all menu(s) by passing the location code number. If the user calls the routine with MENU_NUMBER = 0, menus in all locations will be erased. If called with MENU_NUMBER = -1, the main pull-down menu is erased (see paragraph 7).

d. Menuing application example. Remember the first argument in the DRAW_MENU call represents the menu location, and only coincidentally the menu number.

PROGRAM APPLICATION

```
C *****
C * THIS PROGRAM IS DESIGNED MERELY TO ILLUSTRATE SIMPLE
C * MENU DISPLAYS AND SUBSEQUENT MONITORING. THE PROGRAM
C * DRAWS MENU # 1 AT THE TOP OF THE SCREEN AND, BASED ON
```

```

C      * THE USER'S SELECTION FROM THAT MENU DRAWS, MENU 2 (AT
C      * THE RIGHT) OR MENU 3 (AT BOTTOM) FOR SUBSEQUENT SELEC-
C      * TION. ONE ADDITIONAL MENU, MENU 4 (AT LEFT), IS GENER-
C      * ATED AFTER SELECTION FROM MENU 2.

      CHARACTER*20  TXT1(3), TXT2(8), TXT3(4), TXT4(20)
      LOGICAL ERAS

C      ***** DEFINE 4 MENUS *****
C      *MENU 1
      DATA TXT1 / 'SELECT SYSTEM TYPE', 'PROCESS DATA', 'END
      THE PROGRAM' /

C      *MENU 2
      TXT2(1) = 'FIXED WING AIRCRAFT'      ! NOTE LONGER TEXT
      TXT2(2) = 'ROTARY WING AIRCRAFT'     ! WORKS BEST IN SIDE
      TXT2(3) = 'SP ARTILLERY'             ! MENUS
      TXT2(4) = 'TOWED ARTILLERY'
      TXT2(5) = 'TANKS'
      TXT2(6) = 'ARMORED PERS CARRIERS'    ! NOTE 21ST CHAR TRUN.
      TXT2(7) = 'LOGISTICS'
      TXT2(8) = 'TRUCKS'

C      *MENU 3
      TXT3(1) = 'COMPUTE MEAN'
      TXT3(2) = 'COMPUTE STAND DEV'
      TXT3(3) = 'COMPUTE RANGE'
      TXT3(4) = 'COMPUTE VARIANCE'

C      *MENU 4
      DO I = 1,20
         WRITE (TXT4(I), '(I2)') I          ! LOAD #'S THEMSELVES
      ENDDO

C      ***** MENU DEFINITIONS COMPLETE *****

C      *INITIALIZE RAMTEK (OPEN CHANNEL TO APPROPRIATE STATION,
C      *ESTABLISH COLOR SCHEME TO BE EMPLOYED BY OTHER CALLS)
      CALL INIT_RAMTEK                      ! ONE TIME CALL

C      *DRAW TOP MENU
C      * CALLING ARGUMENTS (INPUT) ARE AS FOLLOWS
      LOC = 1                             ! LOCATES THIS MENU AT TOP
      ERAS = .FALSE.                      ! MENU WILL NOT ERASE AFTER SELECTION
C      *THE COLORS REPRESENTED BY THE FOLLOWING ARGUMENTS ARE
C      *DEPENDENT UPON THE USER'S RESPONSES TO INIT_RAMTEK PROMPTS
      MENBUF = 1                          ! OVERLAY BUFFER
      MGRID = 2                           ! MENU GRID COLOR (USER DEFINED BUFFER 1,
                                         ! COLOR 2
      MTXT = 3                            ! MENU TEXT COLOR
      MHLIT = 4                           ! COLOR IN WHICH MENU SELECTION IS
                                         ! HILITED

```

```

        MBOX = 3                ! NUMBER OF ENTRIES IN THIS MENU
        CALL DRAW_MENU (LOC, ERAS, MENBUF, MGRID, MTXT, MHLIT,
*                               MBOX, TXT1)
100  CONTINUE

C      *MONITOR ONLY THE TOP MENU (AS INDICATED BY THE 1 IN ONLY
C      *THE FIRST OF THE FOUR AVAILABLE MENU LOCATIONS SLOTS)
        CALL MONI_MENU (1, 0, 0, 0, MENOUT, MENBOX)

        IF (MENBOX .EQ. 1) THEN          ! "SELECT SYSTEM TYPE"
            CALL DRAW_MENU (2, .TRUE., 2, 5, 4, 3, 8, TXT2)
        ELSEIF (MENBOX .EQ. 2) THEN       ! "PROCESS DATA"
            CALL DRAW_MENU (3, .TRUE., 1, 7, 5, 4, 3, TXT3)
        ELSE                               ! "END PROGRAM"
            GO TO 9999
        ENDIF

C      *MONITOR BOTH OF THE DISPLAYED MENUS
        CALL MONI_MENU (0, 1, 1, 0, MENOUT, MENBOX)

        IF (MENOUT .EQ. 2) THEN           ! SELECTION FROM RIGHT MENU
            PRINT*, 'HOW MANY OF THIS SYSTEM TO PROCESS?'
            CALL DRAW_MENU (4, .TRUE., 1, 7, 3, 4, 20, TXT4)
            CALL MONI_MENU (0, 0, 0, 1, MENOUT, MENBOX)
            PRINT*, 'PROCESSING ', MENBOX, ' SYSTEMS'
        ELSE                               ! SELECTION FROM BOTTOM MENU
            PRINT*, 'COMPUTING STATS'
        ENDIF

        GO TO 100

9999  CONTINUE

        CALL ERAS_MENU (1)                ! NOTE CALLING WITH 0 ERASES ALL
                                           ! MENUS STILL DISPLAYED
        STOP 'END OF APPLICATION'
        END

```

5. Point data display. Another graphics application that is considered to be relatively high payoff is the ability to graphically display and link point data. This capability has application both to output and input data manipulation. Giving the user dynamic, graphically accomplished, curve-modification capability provides a flexible, user friendly means of developing data files. Allowing the user to display data graphically (especially with the ability to produce hard copies) is useful in analyzing the data and providing presentation assistance. To further enhance the latter capability, curve "fills" and "accumulations" are available allowing the user to generate impressive cumulative distribution type displays. Bar graph data representation is also available if that better fits the user's needs. An application example is provided in paragraph 5g.

a. `LOAD_CURV` Calling format:

`CALL LOAD_CURV (CURV_BUF, CURV_CLR, NUM_PTS, X, Y)`

This routine is called to load a curve for later display. Currently, software limits the user to 20 curves of 30 or less points each. The user controls the overlay buffer, `CURV_BUF` (an integer, 1 if buffering is not being used) and color, `CURV_CLR` (integer), of the curve as determined by his responses to the `INIT_RAMTEK` prompts. WARNING: If the user plans to display filled curves, it is best to be sure all are loaded in the same buffer. Otherwise the results may not be as expected, due to the Ramtek fill technique. `NUM_PTS` passes the number of points on the curve being loaded (which in turn dimensions the `X` and `Y` arrays to follow). The arrays `X` and `Y` define the `X` and `Y`-coordinates respectively of the data points in the order they are to be plotted. (NOTE: Graph Cap software requires that these points be ordered in ascending order of `X`.)

b. `DRAW_CURV` Calling format:

`CALL DRAW_CURV (AXES_BUF, AXES_CLR, FILL, ACCUM)`

This routine will display all curves previously loaded via `LOAD_CURV` and not subsequently erased from memory via `ERAS_CURV`. The axes are dynamically determined with the user having the option to identify each axis' increment if he chooses. The user controls the buffer and color of the axes by the first two integer arguments in this call (for best results, it is recommended that the axes be drawn in a buffer number higher than that in which the curves are to be drawn). The argument `FILL` is a logical which, when `.TRUE.`, will appropriately color fill "below" each curve after plotting it. Otherwise only the curves themselves are plotted. The argument `ACCUM` will produce a cumulative distribution type of display by summing the `Y`-coordinates at each increment point. If `FILL` is `.FALSE.`, the display will plot all curves and the "cumulative" curve unless the data is "nice" (all `X` coordinates the same). In that case the first curve will be drawn and each subsequent curve will reflect the sum of its `Y`-coordinates with those of the previous curves. If `FILL` is `.TRUE.` the user will get an error message unless, as before, the data is "nice," in which case he will get the same set of cumulative curves mentioned above with appropriate color fills.

EXAMPLE: The user loads (via `LOAD_CURV`) 2 curves defined as follows:

CURVE 1 - (0,0), (10,40), (30,30), (60,80)
CURVE 2 - (0,10), (10,20), (30,50), (60,100)

First of all, the axes will be dynamically computed with origin

(in this case) at 0,0 and X-axis of length 60 (X range) and Y-axis of length 100 (Y range). Increments will default to 6 units on the X-axis (10 equal increments) and 10 on the Y-axis, or the user may specify increments of his choice. Next, the points will be plotted and connected forming the two curves defined (in the colors previously assigned via LOAD_CURV). What occurs next depends on the value of the final two calling arguments as follows:

CASE 1: FILL = .FALSE. ACCUM = .FALSE.
Nothing else to do. What you see is what you get.

CASE 2: FILL = .TRUE. ACCUM = .FALSE.
Curve 2 would display first (has the highest Y) and immediately fill below, followed by curve 1 with corresponding fill.

CASE 3A: FILL = .FALSE. ACCUM = .TRUE. "NICE DATA"
Note: NICE DATA means the X-coordinates of all curves are the same as in our example 0, 10, 30, 60.
All curves will "erase" (since new axes must be computed) and then redraw on the new axes. The first curve will be drawn followed by a second which is, in fact, the accumulation of the first and the second.

CASE 3B: FILL = .FALSE. ACCUM = .TRUE. "NOT NICE DATA"
As in case 3A, all curves will erase. This time they will reappear one at a time and one additional curve (which represents the accumulation of all the others) will appear in the "axes color and buffer."

CASE 4: FILL = .TRUE ACCUM = .TRUE.
In the case where the data is "nice," this display will be the same as case 3B except the appropriate color fills will indicate the portion of the accumulation represented by each curve. If the data is "not nice," the software will issue a warning that a filled cumulative display is impossible and would be meaningless and subsequently produce a nonfilled (a la 3B) display.

c. MONI_CURV Calling format:

CALL MONI_CURV (NUMCURV)

MONI_CURV is a multipurpose routine. On one hand, the user may call this routine to dynamically modify (and subsequently save) the point data, or he may simply wish to attain statistical information regarding the curve(s) being monitored.

By passing NUMCURV as 0 (typically the easiest), all currently displayed curves can be monitored; otherwise, only the curve specified by the calling argument value will be monitored. Note that the numbering sequence is the user's responsibility with the

curves numbered sequentially as they are "loaded" and packed (see paragraph 5d) as they are "erased" from memory.

If the user is monitoring all curves, he is prompted to graphically select one for more detailed inspection. The curve selected (either by the action just described or by specific reference as the calling argument) will then highlight its current data points (nodes). The routine will next display a menu at the top of the Ramtek monitor querying the user for the type of monitoring of interest. The user may choose one of the following:

(1) Move a node. This option allows the user to relocate a data point which, in turn, modifies the point data in memory for that curve.

(2) Delete a node. This option allows the user to erase a data point from the screen, reconnect the preceding and succeeding points, and subsequently erase that data point in memory for that curve.

(3) Add a node. This option allows the user to expand the curve by adding a new data point to the end of the curve (far right). A word of caution here, if the user wishes to expand the curve which extends farthest to the right of all curves displayed, it will be necessary to "create a dummy curve" since the axes are dynamically determined based on the minimums and maximums of the curve(s) to be displayed.

(4) Insert a node. This option allows the user to place a new data point on the curve between two existing data points, with the corresponding effect on the curve in memory.

(5) Stats. This option allows the user to obtain statistical information regarding the selected curve. Information such as range, mean, and standard deviation in both the X and Y directions is available. Also available is point selection information, where the user can identify any point on the curve (or off) by simply locating the light pen/puck and depressing it. The information displayed identifies the exact location selected as well as the point on the curve (perpendicularly) nearest that selected.

(6) Keyboard. This option toggles the expected input format between graph tablet and keyboard. In this way, the user can specify precisely defined nodes to be added, inserted or moved via the keyboard. Upon reselection, input toggles back to tablet.

d. ERAS_CURV Calling format:

CALL ERAS_CURV (NUMBER_CURV, MEMORY)

This routine allows the user to erase a specified curve or all currently displayed curves (NUMBER_CURV = 0) from the screen and optionally from memory. If the user erases the curve(s) from memory (MEMORY = 1), the remaining curves will be packed (i.e., the third curve originally loaded would move to second if either of the first two were erased). WARNING: Remember, it's the user's responsibility to handle the curve ordering. To erase curves from the screen only the user passes MEMORY = 0. WARNING: In the event of a "filled" display, ERAS_CURV will erase only the curve itself not the filled "region".

e. DRAW_BAR Calling format:

```
CALL DRAW_BAR ( AXES_BUF, AXES_CLR, ACCUM )
```

This routine allows another form of point data display. All "curve" data loaded will be displayed as vertical bars, centered about the appropriate X-coordinate. The width of the bars is a function of the user specified X-axis increment. This type of display is best suited for "nice data" (see paragraph 5b). The final argument, ACCUM is a logical which if .TRUE. will result in vertically stacked bars each representing the contribution of a different "curve".

f. DRAW_PIE Calling format:

```
CALL DRAW_PIE ( BUFFER, NUM_VAL, COL_ARRAY, VAL_ARRAY, SIZE, INT)
```

This routine does not operate on curve data previously loaded (as did DRAW_CURV and DRAW_BAR), but does represent another way to depict data graphically. A "pie" representing the sum of all passed in values (real array VAL_ARRAY) is drawn with each wedge illustrating the fractional part of that sum that its corresponding value represents. For example, if VAL_ARRAY contained values 5, 3 and 2, the pie would have wedges consisting of 50%, 30% and 20% of the pie's area respectively depicted. The color of each wedge is determined by the BUFFER and single dimensioned array of integer color values (COL_ARRAY) within that buffer. The size of the pie is determined in two ways: If the user chooses to interactively "size" the pie, he may do so by passing INT as .TRUE., otherwise the size is determined by the integer argument SIZE. If SIZE = 1, the pie is screen centered and occupies approximately one fourth of a low resolution monitor screen. If SIZE = 2, the pie is again screen centered but this time occupies the entire screen. Obviously INT = .TRUE. overrides the SIZE argument allowing the user to interactively locate and "size" an expandable circle.

g. Data point display/monitor example.

PROGRAM APPLICATION2

```

C *****
C * THIS ROUTINE ILLUSTRATES DEFINING, LOADING, DISPLAYING,
C * AND MONITORING CURVES. FIRST THE TANK, LAW, AND HELO
C * CURVES ARE LOADED, DISPLAYED, AND MONITORED. NEXT, THE
C * LAW CURVE IS ERASED, AND THE OTHERS ARE REDRAWN WITH
C * THE FILL OPTION ACTIVATED (AND PAUSES FOR REVIEW).
C * NEXT, THE REMAINING CURVES ARE ERASED FROM THE SCREEN,
C * A NEW CURVE (ARTY) IS LOADED AND ALL ARE REDRAWN WITH
C * BOTH FILL AND ACCUMULATE OPTIONS SELECTED (PAUSE).
C * FINALLY, THE TANK AND HELO CURVES ARE ELIMINATED AND
C * THE ARTY CURVE IS DRAWN ALONE FOR MONITORING.

      REAL HOUR(7),TANK(7), LAW(7), HELO(7), ARTY(7)
      LOGICAL FILL, ACCU

C *DATA MAY BE ROUTINE GENERATED OF COURSE

C *DEFINE "Y-COORDINATES" OF ALL DATA POINTS
      DATA TANK /3.1, 2.6, 8.0, 5.5, 3.1, 1.3, .4/
      DATA LAW /1.4, 1.6, 4.2, 2.2, .8, .2, 0. /
      DATA HELO /0. , 2.1, 9.8, 6.7, 4.0, 0. , .1/
      DATA ARTY /6.2, 8.9,16.2, 10.1,6.4, 3.5, 1.1/

C *DEFINE "X-COORDINATES" OF ALL DATA POINTS
      DATA HOUR / 1., 2., 3., 4., 5., 6., 7. /

      CALL INIT_RAMTEK

C *THE COLORS REPRESENTED BY THE FOLLOWING ARGUMENTS ARE
C *DEPENDENT UPON THE USER'S RESPONSES TO INIT_RAMTEK
      NBUF = 1 ! OVERLAY BUFFER IN WHICH CURVE WILL DRAW
      NCLR = 5 ! 5TH COLOR IN BUFFER 1, USED FOR TANK CUR
      NPTS = 7 ! NUMBER OF DATA POINTS IN THIS CURVE

      CALL LOAD_CURV (NBUF, NCLR, NPTS, HOUR, TANK) !TANK

      CALL LOAD_CURV (NBUF, 3, NPTS, HOUR, LAW) !LAW

      CALL LOAD_CURV (NBUF, 4, NPTS, HOUR, HELO) !HELO

C *DRAW THE 3 CURVES JUST LOADED WITH NO FILL NOR ACCUM.
      NBUF = 1 ! OVERLAY BUFFER IN WHICH AXES WILL DRAW
      NCLR = 6 ! 6TH COLOR IN BUFFER 1, USED FOR AXES
      FILL = .FALSE. ! DO NOT FILL UNDER THE CURVES
      ACCU = .FALSE. ! DO NOT PRODUCE A CUMULATIVE CURVE

      CALL DRAW_CURV (NBUF, NCLR, FILL, ACCU)

C *MONITOR ALL THREE CURVES FOR ADDITIONAL INFORMATION OR
C *MODIFICATION AND SAVE TO FILE IF DESIRED (SAVE_STAT)
      CALL MONI_CURV (0)

```

```

C      *ERASE THE LAW CURVE, REDRAW THE OTHERS WITH FILL
      CALL ERAS_CURV (2, 1)      !NOTE CURVE ERASED FROM
                                   !BOTH SCREEN & MEMORY
      CALL DRAW_CURV (1, 6, .TRUE., .FALSE.)

      PRINT*, 'HIT ANY KEY TO CONTINUE'      !PAUSE TO EXAMINE
      READ(5,G) JUNK

      CALL ERAS_SCRN (0,0)      !ERASE SCREEN, ALL BUFFERS

      CALL LOAD_CURV (1, 5, 7, HOUR, ARTY)      !LOAD ARTY CURVE

      CALL DRAW_CURV (1, 6, .TRUE., .TRUE.)

      PRINT*, 'HIT ANY KEY TO CONTINUE'      !PAUSE TO EXAMINE
      READ(5,G) JUNK

      CALL ERAS_SCRN (0,0)      !ERASE SCREEN, ALL BUFFERS
      CALL ERAS_CURV (1, 1)      !ERASE TANK CURVE FROM MEMORY
      CALL ERAS_CURV (2, 1)      !ERASE HELO CURVE (NOTE DATA
                                   !WAS PACKED)

      CALL MONI_CURV (3)      !MONITOR ARTY CURVE

      STOP
      END

```

6. Status and picture preservation/manipulation. The capabilities described in the previous section allow for dynamic display and modification of curve data. Dynamic display has obvious applicability for analysis, especially when the stats option is selected while monitoring the curve(s) of interest. Dynamic modification, on the other hand, is of little use unless the "picture" and/or overall "system status" can be preserved for later use. Three of the Graph Cap callable routines discussed in this section provide the application programmer with this capability, another provides a means for clearing the monitor screen completely, and two others provide picture manipulation capability. Several sample application programs (paragraphs 6g,h,i) are provided for further assistance.

a. SAVE_STAT Calling format:

```
CALL SAVE_STAT
```

This routine can be called directly by the user, as in section 6g, but more likely will be called by the software from routine MONI_CURV at the user's request. In other words, when the user has completed monitoring displayed curves, the option to preserve the current status (which includes color scheme, screen data, and all curve data) is proposed by the software itself. Status files are transportable between low and high resolution systems.

b. LOAD_STAT Calling format

CALL LOAD_STAT (VALID)

This routine is use to reload the status (color scheme, screen data, and all curve data) saved during a previous run via SAVE_STAT. The returned logical calling argument, VALID, simply notifies the user that a valid status file (.STAT) was found.

c. SAVE_PIC Calling format:

CALL SAVE_PIC

This routine saves only the picture currently displayed on the Ramtek monitor. (There is no preservation of curve data or color scheme as with SAVE_STAT). Currently picture files are one-way-transportable between low and high resolution monitors. This means that pictures generated and saved on a low resolution system can be displayed (via LOAD_PIC) on a high resolution system with no ill effects. However, only the lower left quarter of pictures generated and saved on a high resolution system will display on low resolution screen. Upon execution this routine prompts the user as follows:

DO YOU WISH TO SAVE THE ENTIRE SCREEN? [Y/N] [Y] N

If the user selected the default Yes in the above example, the folowing two prompts would not appear.

SELECT ONE CORNER OF RECTANGLE

STRETCH RECTANGLE TO OPPOSITE CORNER & PUSH BUTTON

These prompts direct the user to position an expandable rectangle around the portion of the picture to be saved.

ENTER NAME OF PICTURE FILE TO BE SAVED (NO EXT) <filename>

The user may specify any DEC legitimate file name; however, the extension .PIC will be automatically appended to the file name.

d. LOAD_PIC Calling format:

CALL LOAD_PIC

This routine loads a picture previously preserved with SAVE_PIC. There is no preservation of color when saving a picture, so it's the user's responsibility to assure that an appropriate color scheme is loaded prior to loading the picture. Upon execution this routine prompts the user as follows:

ENTER NAME OF PICTURE FILE TO BE LOADED (NO EXT) <filename>

The user enters the filename (without the .PIC extension) which was saved previously and which he now wants displayed. If the software determines that the picture to be loaded would not completely fill the screen at this user's station, the following prompt will appear.

REPOSTION THE IMAGE? [Y/N] [N] Y

If the user selected the default No in the above example, or, if the software determined the picture being loaded would occupy the full screen, the following prompt would not appear.

LOCATE BOX WHERE PICTURE IS TO APPEAR

A rectangle representing the actual size of the incoming picture will appear on the graphics monitor and should be positioned by the user via the graph tablet puck/pen. Once the user is satisfied with the location, he depresses the puck button to finalize the load.

e. MOVE_PIC Calling format:

CALL MOVE_PIC (MOVE_OR_DUPLICATE)

This routine allows the user to relocate a portion of the picture currently displayed. The single calling argument determines whether the portion selected is actually duplicated (MOVE_OR_DUPLICATE = 1), or actually moved (i.e., the old portion erased) to a new location (MOVE_OR_DUPLICATE = 0). Upon execution the user is prompted as follows:

SELECT ONE CORNER OF RECTANGLE

STRETCH RECTANGLE TO OPPOSITE CORNER & PUSH BUTTON

These prompts direct the user to position an expandable rectangle around the portion of the picture to be moved.

USE CURSOR TO DEFINE LL CORNER OF NEW LOCATION

The user is asked to select a point with the graph tablet puck/pen which will translate to the lower left corner the picture's new location.

f. MERGE_PIC Calling format:

CALL MERGE_PIC

This routine is similar to LOAD_PIC with one major difference.

When a picture file is loaded via LOAD_PIC it completely overdraws the portion of the screen to which it is directed. When a picture is merged via MERGE_PIC it "overlays" the incoming picture atop the current display allowing the latter to "show through" in areas where no data exists in the new picture. Upon execution of this routine the user is prompted as follows:

ENTER PICTURE FILE TO MERGE (NO EXT) <filename>

The user enters the filename (without the .PIC extension) which was saved previously and which he now wants displayed. If the software determines that the picture to be loaded would not completely fill the screen at this user's station, the following prompt will appear.

REPOSTION THE IMAGE? [Y/N] [N] Y

If the user selected the default No in the above example, or, if the software determined the picture being loaded would occupy the full screen, the following prompt would not appear.

LOCATE BOX WHERE PICTURE IS TO APPEAR

A rectangle representing the actual size of the incoming picture will appear on the graphics monitor and should be positioned by the user via the graph tablet puck/pen. Once the user is satisfied with the location, he depresses the puck button to finalize the merged location.

Level-by-level merge? [Y/N/?] [N] <return> or No

Typically the user will not want a level-by-level merge and consequently should accept the default response. Without going into a great deal of detail, a level-by-level merge attempts to merge the pictures, buffer by buffer, to create an intertwined overlapping effect.

g. General screen and status application, example 1.

PROGRAM APPL_LOAD

C *THIS PROGRAM SIMPLY LOADS SOME CURVES & SAVES THEM
REAL HOUR(7),TANK(7), LAW(7), HELO(7), ARTY(7)

C *DATA MAY BE ROUTINE GENERATED OF COURSE

DATA TANK /3.1, 2.6, 8.0, 5.5, 3.1, 1.3, .4/
DATA LAW /1.4, 1.6, 4.2, 2.2, .8, .2, 0. /
DATA HELO /0. , 2.1, 9.8, 6.7, 4.0, 0. , .1/
DATA ARTY /6.2, 8.9,16.2, 10.1,6.4, 3.5, 1.1/

DATA HOUR / 1., 2., 3., 4., 5., 6., 7. /

```

CALL INIT_RAMTEK

CALL LOAD_CURV (1, 5, 7, HOUR, TANK) !LOAD TANK CURVE

CALL LOAD_CURV (1, 3, 7, HOUR, LAW) !LOAD LAW CURVE

CALL LOAD_CURV (1, 4, 7, HOUR, HELO) !LOAD HELO CURVE

CALL SAVE_STAT      ! SAVE STATUS FOR LATER UPDATE

STOP
END

```

h. General screen and status application, example 2.

```

PROGRAM APPL_MODIFY
C *THIS ROUTINE ALLOWS THE USER TO DISPLAY, ANALYZE,
C *MODIFY, AND SUBSEQUENTLY SAVE CURVES LOADED BY APPL_LOAD
C *AND THEN CONTINUE MODIFYING THOSE CURVES IF DESIRED

CHARACTER RESP*1 /'Y'/

CALL INIT_RAMTEK      !SAY YES TO RELOAD PROMPT
                      ! AND ENTER NAME OF STATUS FILE
                      ! SAVED IN APPL_LOAD

DO WHILE (RESP .EQ. 'Y')
  CALL DRAW_CURV (1, 8, .FALSE., .FALSE.)
  CALL MONI_CURV (0)  !OPTIONALLY SAVE STATUS
  CALL SAVE_PIC      !OPTIONALLY SAVE PICTURE
  CALL ERAS_SCRN (0,0) !ERASE ALL BUFFERS, BUT NOT
                      !CURVE DATA

  PRINT*, 'CONTINUE?'
  READ(5,10) RESP
10  FORMAT (A1)
ENDDO

STOP
END

```

i. General screen and status application, example 3.

```

PROGRAM APPL_RELOAD
C *THIS ROUTINE ALLOWS A USER TO LOAD PREVIOUSLY PRESERVED
C *STATUS OR PICTURE FILES.

C *WARNING: REMEMBER PICTURE FILES ARE NOT TRANSPORTABLE

CHARACTER RESP*1

```

```

CALL INIT_RAMTEK

10 PRINT*, 'LOAD STATUS (S), OR PICTURE (P)?'
   READ(5,20) RESP
20 FORMAT (A1)

   IF (RESP .EQ. 'S') THEN
      CALL LOAD_STAT (VALID)
      CALL DRAW_CURV (1, 8, .FALSE., .FALSE.)
      CALL MONI_CURV (0)
      CALL SAVE_PIC
   ELSEIF (RESP .EQ. 'P') THEN
      CALL LOAD_PIC
   ELSE
      GO TO 9999
   ENDIF

PRINT*, 'ERASE SCREEN?'
READ(5,20) RESP
IF (RESP .EQ. 'Y') CALL ERAS_SCRN (0,0)

GO TO 10
9999 STOP
END

```

7. Pull-down and pop-up menuing. Paragraph 4 discusses Graph Cap's initial, generic, and simplistic approach to menuing. Pull-down and pop-up menus provide significantly more aesthetic appeal, but they aren't an appropriate formats for every menuing application. A pull-down menu consists of a main menu, which appears at the top of the monitor, and a set (currently software restricted to 8 or less) of submenus, which visually "pull down" from the main menu as its entries are "touched" via the graph tablet pen/puck. A pop-up menu is one which appears in the center of the graphics monitor for a single selection and automatically disappears after that selection is made. Again, an application program to illustrate pull-down and pop-up implementation is provided in paragraph 7d. The nature of these menus (they overlay previous displays) mandates that they be drawn in the highest overlay buffer defined by the user in his responses to routine INIT_RAMTEK (paragraph 3a), or preverably in a temporary, artificial, software generated higher buffer so as not to disturb the underlying graphics. The first three (excluding "clear" color 1) colors in that buffer are reserved to define the menu grid, text, and hilight colors.

a. LOAD_PULL Calling format:

```

CALL LOAD_PULL (NUM_BOX_MAIN, TXT_MAIN, MAX_SUB, NUM_BOX_SUB,
                TXT_SUB)

```

This routine loads the user-defined pull-down menu text into Graph Cap commons for later display and monitoring. Typically, the user will define the menu text and call LOAD_PULL from one routine and then display and monitor the menu as needed. The first two calling arguments refer to the main menu portion of the pull-down with NUM_BOX_MAIN being the number of entries (boxes) in the main menu (note the above restriction of 8) and TXT_MAIN is the character array containing the text corresponding to each of these entries. The last three arguments describe the submenu portion. NUM_BOX_SUB is an array indicating the number of entries in each submenu. For example: if there are three main menu entries, there would most likely be three corresponding submenus. The submenus can each have a different number of entries, say 3, 5, and 7. In this case, the array NUM_BOX_SUB would be dimensioned 3 and could be defined as follows: DATA NUM_BOX_SUB / 3,5,7/. MAX_SUB represents the largest of all the NUM_BOX_SUB values (7 in this example), and TXT_SUB is a two-dimensional character array containing the submenu text where the second dimension represents the submenu, and the first the box in the submenu.

b. MONI_PULL Calling format:

CALL MONI_PULL (WHICH_SUB, WHICH_BOX_IN_SUB)

This routine displays the pull-down menu previously loaded into common via LOAD_PULL and monitors it in the following fashion. As the user slides the pen/puck (no button depression required) into a main menu entry box, the appropriate submenu will appear below that main menu entry (the submenu text length is not restricted by that of the main menu entry). As the user slides the pen/puck through the submenu, each entry will backlight to clearly identify which would be selected should the user depress the puck button (#1 on a four-button puck). Once the user does depress the puck button, his selection will highlight and the submenu subsequently disappear, however, the main menu remains displayed to allow for successive monitoring until the user erases it via ERAS_SCRN (paragraph 9b). The selection information is returned to the calling routine through the calling arguments as follows: the number of the displayed submenu (left to right) from which the selection was made is returned in WHICH_SUB, while the number (top to bottom) of the selected entry box in that submenu is returned in WHICH_BOX_IN_SUB.

c. MONI_POP Calling format:

CALL MONI_POP (NUMBOXES, TEXT, WHICH_BOX_IN_POP)

Since a pop-up menu appears only long enough for a single selection, this routine both displays and monitors the user-defined pop-up menu. The user provides the number of entries

(boxes) for the pop-up in argument NUMBOXES and a text array containing the pop-up text in TEXT. WHICH_BOX_IN_POP is returned to the user as the number (top to bottom) of the selected entry.

d. Pull-down/pop-up menuing example program.

```

PROGRAM APPL_POP
C *****
C * THIS ROUTINE DISPLAYS AND ALLOWS CYCLICAL SELECTION
C * FROM A PULL-DOWN MENU UNTIL THE END PROGRAM SUBMENU
C * ENTRY IS SELECTED. ADDITIONALLY, IF THE USER SELECTS
C * THE EDIT SUBMENU AND SUBSEQUENTLY THE DELETE OPTION,
C * A POP-UP MENU REQUESTING CONFIRMATION WILL APPEAR.

CHARACTER MTXT(4)*7, STXT(6,4)*30, PTXT(3)*15
INTEGER NSUB(4)

C ***** DEFINE THE PULL-DOWN MENU *****
C * MAIN MENU PORTION
  MTXT(1) = 'SELECT'
  MTXT(2) = 'PROCESS'
  MTXT(3) = 'NUMBER'
  MTXT(4) = 'EDIT'
  NMAIN = 4

C * SUBMENU PORTION
  STXT(1,1) = 'FIXED-WING AIRCRAFT'
  STXT(2,1) = 'ROTARY-WING AIRCRAFT'
  STXT(3,1) = 'SP ARTILLERY'
  STXT(4,1) = 'TOWED ARTILLERY'
  STXT(5,1) = 'TANKS'
  STXT(6,1) = 'ARMORED PERSONNEL CARRIER'
  NSUB(1) = 6

  STXT(1,2) = 'COMPUTE MEAN'
  STXT(2,2) = 'COMPUTE STANDARD DEVIATION'
  STXT(3,2) = 'COMPUTE RANGE'
  STXT(4,2) = 'COMPUTE VARIANCE'
  NSUB(2) = 4

  DO I = 1, 20
    WRITE ( STXT(I,3), '(I2)' ) I
  ENDDO
  NSUB(3) = 20

  STXT(1,4) = 'ADD'
  STXT(2,4) = 'CHANGE'
  STXT(3,4) = 'DELETE'
  STXT(4,4) = 'DUPLICATE'
  STXT(5,4) = 'END PROGRAM'
  NSUB(4) = 5

```

```

MAXSUB = 20

C      ***** BUILD POP UP MENU *****
      PTXT(1) = 'CONFIRM'
      PTXT(2) = 'REJECT'
      PTXT(3) = 'NONE OF THE ABOVE'

C      ***** FINISHED DEFINING MENUS *****

C      *LOAD THE PULL DOWN MENU INTO COMMON
      CALL LOAD_PULL (NMAIN, MTXT, MAXSUB, NSUB, STXT)

C      *DISPLAY AND MONITOR THE PULL DOWN MENU
100    CALL MONI_PULL (NSUB, NBOX)

C      *IF USER SELECTS "END PROGRAM"
      IF (NSUB .EQ. 4 .AND. NBOX .EQ. 5) THEN
        CALL ERAS_SCRN (0,0)
        STOP 'END OF APPL_POP'
      ENDIF

C      *IF USER SELECTS EDIT/DELETE
      IF (NSUB .EQ. 4 .AND. NBOX .EQ. 3) THEN
        CALL MONI_POP (3, PTXT, NPBOX)
C      *AND CONFIRMS THE DELETION
        IF (NPBOX .EQ. 1) THEN
          PRINT*, 'DELETION OPTION SELECTED'
        ENDIF
      ENDIF
      GO TO 100
      STOP
      END

```

8. Shape manipulation. The following routines give the Graph Cap user the capabilities to generate geometric shapes and text and to erase the last version of each shape displayed.

a. DRAW_SHAPE Calling format:

```

CALL DRAW_SHAPE ( BUFFER, COLOR, SHAPE, FILL, TXT_SIZE, TXT_DIR,
                  LEAVE_SHAPE, XLL_CENTX, YLL_CENTY, XUR_RAD, YUR)

```

This routine serves as a "shape driver." The user can generate any of the following shapes simply by passing the appropriate arguments: rectangle, circle, ellipse, polygon, line and text. Each of the first three are expandable (the user expands from the first point selected to the desired size) and can be optionally color filled (as can the polygon). The line and polygon are located using "stretchable" segments (the user stretches from each selected point to the next). The first two arguments define the color of the shape by specifying the BUFFER and COLOR number

within that buffer. The third argument is the shape name itself (e.g. 'rectangle', 'circle', etc.). The fourth argument, FILL, is a logical which when .TRUE. indicates that the shape is to be color filled (ignored when not applicable). The next two arguments pertain only if the shape to be drawn is 'text'. TXT_SIZE is an integer between 1 and 16 inclusive which specifies the size of the drawing text where size 16 is 16 times as wide and high as size 1 text. TXT_DIR is an integer between 1 and 4 inclusive which specifies the direction the text will assume as follows: 1) normal (horizontal), 2) vertical top to bottom with characters facing normally, 3) vertical bottom to top face left (rotated 90 degrees counterclockwise) and, 4) vertical top to bottom face right (rotated 90 degrees clockwise). An example of text size and direction can be seen in Figure 1. The seventh thru eleventh arguments currently apply only to rectangle and circle draws. LEAVE_SHAPE is a logical which if .TRUE. indicates that the shape drawn will remain displayed. At first this may seem like a ridiculous option, but there are times when the applications programmer is interested merely in defining a rectangular or circular area rather than actually drawing a permanent shape. This option allows him to specify that area via an expandable rectangle or circle.

b. DELETE_SHAPE Calling format:

CALL DELETE_SHAPE (SHAPE)

This routine will allow the user to erase the last drawn of the specified shape. Two caveats need emphasis here. This effect is not regressive, i.e. if the user has drawn four rectangles he may erase the fourth by calling DELETE_SHAPE, but once completed a subsequent call to DELETE_SHAPE will not erase the third. Secondly, if the user has color filled the shape to be erased via the FILL argument in the call to DRAW_SHAPE, then the entire shape (fill and all) will be erased. However, if an alternate method of "fill" was used (see FILL_FRMC) the shape will be erased but the "filled region" remains. The one calling argument, SHAPE, represents the literal string 'rectangle', 'circle', etc.

9. General routines. The routines in this section don't fit nicely into any of the previous sections. Their function is more general in nature.

a. FILL_FRMC Calling format:

CALL FILL_FRMC (BUFFER, COLOR, FILL_MODE)

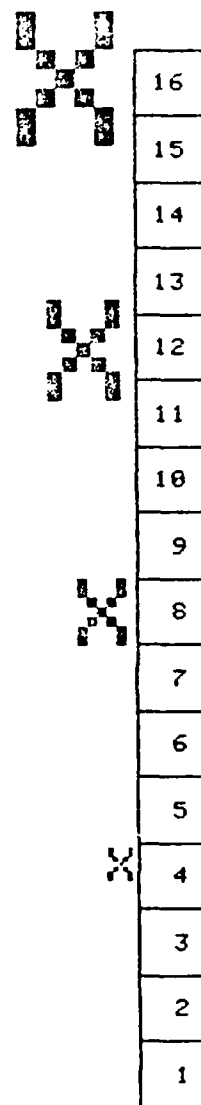
This routine, "fill from cursor", provides the user with a method to "fill" an area, with a specified BUFFER and related COLOR in that buffer. Extreme caution is urged in using this routine since a "fill" can destroy the user's entire display if not

LEFT TO RIGHT

TOP/BOT-FACE RIGHT

BOT/TOP-FACE LEFT

TOP/BOT-FACE RIGHT



16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1

FIGURE 8-1. Text samples

properly "bounded" by a shape drawn in the same buffer and color. If FILL_MODE is 2 (fill will "flow" until it encounters "itself", i.e. the same color in the same buffer). If FILL_MODE is 1, the fill will "flow" until it "hits" any other color in that same buffer which is currently displayed.

b. ERAS_SCRN Calling format:

CALL ERAS_SCRN (MEMORY, BUFFER)

This routine allows the user a means of completely clearing the monitor screen by erasing any or all (BUFFER=0) buffers. The first argument, MEMORY, will typically be 0 (to erase the screen only), but can be set to 1 if the user wishes to clear memory of all previously loaded curve data (to freshly load new curve data).

10. Summary. The Graph Cap (formerly Macro-FLIK) routines described in the previous sections are intended to provide the application software programmer with an easy to use, minimal set of routines which will enable him to incorporate VAX/Ramtek graphics into any application package. Currently, the major graphics functions provided include menuing (graph tablet input), data point display and analysis, data point manipulation and preservation, picture preservation and manipulation and shape manipulation. Other non-application software-specific graphics capabilities could be added to the current package upon request.

a. The Graph Cap software can be found in the disk/directory defined by logical name OMACRO. The source code is separated into two text libraries. Library MACRO.TLB contains that portion of the package designed to be callable by the applications programmer, while library UTLY.TLB contains utility routines called by the routines in the former. The applications programmer calls utility routines at his own risk with no guarantee of the results. The object modules for both text libraries are kept in A.OLB.

b. Most of the sample application programs found in this document (and several others) are available in OMACRO:APPL.TLB for additional assistance.

c. At this writing, several additional enhancements to the Graph Cap package have been suggested and are under consideration. The first of these would involve the incorporation of "alert" menus. Alert menus are similar to pop-up menus in that they appear only for a single selection; but, they differ in that they have space available for user provided text instruction and a number of "buttons" (boxes) for the user to select from. Another enhancement would provide the user with a flexible legend generation capability (especially as it applies to point data

display).

d. Any questions regarding the use of the Graph Cap package, or suggestions as to how to improve its utility, should be directed to Mr. RH "Pete" Kaeding, TRAC-FLVN, AV 552-4261.

APPENDIX A

"POCKET" REFERENCE GUIDE FOR GRAPH CAP TRAC GRAPHICS CAP CALLABLE ROUTINES (23)

INITIALIZATION:

1. INIT_RAMTEK "BUILD/SELECT COLOR SCHEME"

CONVENTIONAL MENUING:

2. DRAW_MENU (LOC, ERAS, NBUF, NGRD, NTXT, NHILT, NBX, TXT)

LOC = 1 - 4 clockwise from top of screen
ERAS = .TRUE. or .FALSE. erase menu after selection?
NBUF = 1 - 8 dependent on color file you intend to use
NGRD = color # to be used for menu grid boxes (dep. on color file)
NTXT = color # to be used for menu text (dep. on color file)
NHILT = color # to be used for highlighting selection ("")
NBX = # of boxes in the menu defined
TXT = single dimensioned character array containing text to appear in each box of the menu

SAMPLE: CHARACTER*20 TXT(3)
DATA TXT /'BOX 1', 'BOX 2', 'BOX 3'/
CALL DRAW_MENU (1, .TRUE., 1, 2, 3, 4, 3, TXT)

3. MONI_MENU (LOC1, LOC2, LOC3, LOC4, MENU_SEL, IBOX_SEL)

LOC1 - LOC4 = 0 or 1, 1 means to monitor a menu in that location
MENU_SEL = <returned> which of the monitored menus the selection came from
IBOX_SEL = <returned> number of the box selected from the above menu

SAMPLE: CALL MONI_MENU (1, 1, 0, 0, MENU_SEL, BOX_SEL)
will return information when a selection is made from the
top or right menus

4. ERAS_MENU (MENU_LOC)

MENU_LOC = 1 to 4 clockwise from top, menu # to be erased.
MENU_LOC = -1 the main pulldown menu is erased.

PULLDOWN & POPUP MENUING:

5. LOAD_PULL (NBX_MN, TXT_MN, MAX_SUB, NBOX_SUB, TXT_SUB)

NBX_MN = # of boxes (entries) in the main menu
TXT_MN = single dimensioned character array of main menu
text
MAX_SUB = maximum # of entries in any of the sub-menus
NBOX_SUB = single dimensioned array of # of boxes in each
sub-menu
TXT_SUB = two dimensional character array containing text
of all sub-menus where 2nd dimension is the
submenu number and the first the box in that
submenu.

SAMPLE: CHARACTER*20 TXT_SUB(3,2), TXT_MAIN(2)
INTEGER BOX_SUB(2)

```
DATA TXT_MAIN /'MAIN 1', 'MAIN 2'/  
TXT_SUB(1,1) = 'BOX 1 SUB 1'  
TXT_SUB(2,1) = 'BOX 2 SUB 1'  
TXT_SUB(1,2) = 'BOX 1 SUB 2'  
TXT_SUB(2,2) = 'BOX 2 SUB 2'  
TXT_SUB(3,2) = 'BOX 3 SUB 2'
```

```
BOX_SUB(1) = 2  
BOX_SUB(2) = 3  
CALL LOAD_PULL (2, TXT_MAIN, 3, BOX_SUB, TXT_SUB)
```

6. MONI_PULL (NSUB_SEL, NBOX_IN_SUB_SEL)

NSUB_SEL = <returned> # of sub-menu selection was made
from
NBOX_IN = <returned> # of box in that sub-menu

7. MONI_POP (NBOX_POP, TXT_POP, NBOX_IN_POP_SEL)

NBOX_POP = # of boxes (entries) to be in the pop up menu
TXT_POP = single dimensioned character array of pop-up
 menu text
NBOX_IN = <returned> # of box selected from pop-up menu

SAMPLE: CHARACTER*20 TXT_POP(2)
 DATA TXT_POP /'BOX 1' , 'BOX 2'/
 CALL MONI_POP (2, TXT_POP, BOX_IN_POP_SEL)

POINT DATA DISPLAY:

8. LOAD_CURV (NCURV_BUF, NCURV_COL, NUM_PTS, X, Y)

NCURV_BUF = # of buffer curve is to be drawn in (color
 file dependent)
NCURV_COL = # of color in that buffer (color file dep.)
NUM_PTS = # of points on this curve (software limit is
 30)
X = single dimensioned array containing the X-coordinates
 of all points to later be displayed
Y = ditto Y-coordinates

SAMPLE: DATA X / 1, 2, 3, 4, 5/
 DATA Y /100.5, 200, 6.8, 999.9, 20000/
 CALL LOAD_CURV (1, 2, 5, X, Y)

9. DRAW_CURV (NAXES_BUF, NAXES_COL, FILL, ACCUMULATE)

NAXES_BUF = buffer # axes are to be drawn in (color file
 dependent)
NAXES_COL = color # axes are to be drawn in (color file
 dependent)
FILL = logical, .TRUE. means curves will be filled
 below *
ACCUMULATE = logical, .TRUE. means cumulative curve will
 be drawn

* WARNING: Fill is very buffer/color sensitive. If the
user plans to display filled curves, it is best to load
them all in the same buffer to avoid undesirable results.
It is also recommended that the axes be drawn in a buffer
higher than that in which the curves will be drawn.

SAMPLE: CALL DRAW_CURV (2, 2, .TRUE., .FALSE.)

10. DRAW_BAR (NAXES_BUF, NAXES_COL, ACCUMULATE)

All arguments are the same as DRAW_CURV, except there is no fill option (bars are always filled).

11. ERAS_CURV (NUM_CURV, ISCRN_AND_OR_MEMORY)

NUM_CURV = # of the curve to be erased (in the order they were loaded), 0 means all curves

ISCRN_OR_MEM = 0 means just erase from the screen (curve data remains "loaded"), 1 means clear the curve completely (it will no longer display when draw_curv is called)

POINT DATA REVIEW/MODIFICATION:

12. MONI_CURV (NCURV_NUMBER)

NCURV_NUMBER = # of the curve the user will be allowed to "monitor" (change data points or view statistical info), 0 means all curves displayed are to be monitored

POINT DATA PRESERVATION/REDISPLAY:

13. SAVE_STAT "NAME FILE"

14. LOAD_STAT (VALID) "RETURN ARGUMENT INDICATING EXISTENCE"

VALID = logical, .TRUE. means the file specified exists and reads error free

PICTURE PRESERVATION/REDISPLAY:

15. SAVE_PIC "NAME FILE"

16. LOAD_PIC "NAME FILE"

SHAPE MANIPULATION:

17. DRAW_SHAPE (NBUF, NCOL, SHAPE, FILL, NTXT_SIZ, NTXT_DIR,
LEAVE_SHAPE, XLL-CENTX, YLL-CENTY, URX, URY)

NBUF = buffer # in which shape is to be drawn
(colorfile dep)
NCOL = color # in which shape is to be drawn
(colorfile dep)
SHAPE = character string identifying the shape to be
drawn. One of: rectangle, circle, ellipse,
line, polygon, text.
FILL = logical, .TRUE. means shape will be filled
(if applicable)
NTXT_SIZ = 1 - 16 from smallest to largest text
NTXT_DIR = 1 - 4, 1 means normal (horizontal) text,
2-4 are vertical with text facing up, left or
right
LEAVE_SHAPE = logical, .TRUE. means the rectangle (or
circle) drawn (obviously originally applied
only to rectangle) is to remain on the screen
after the draw is completed
XLL, YLL = <returned> the lower left virtual (in users
coordinate system) coordinates of the
rectangle drawn (or the circle's center)
URX, URY = <returned> the upper right virtual
coordinates of the rectangle drawn (URX is
the radius of the circle)

SAMPLE: drawing a rectangle

CALL DRAW_SHAPE (1, 2, 'RECTANGLE', .TRUE., 0, 0, .TRUE.,
XLL, YLL, URX, URY)

drawing text

CALL DRAW_SHAPE (1, 2, 'TEXT', .FALSE., 4, 1,
.FALSE., , , ,)

18. DELETE_SHAPE (SHAPE)

SHAPE = character string of shape to be deleted (only the last of each shape drawn can be deleted)

GENERAL:

19. ERAS_SCRN (ISCRN_OR_MEMORY, NBUFFER)

ISCRN_OR_MEMORY = *redundant*, 0 means just erase screen in buffer indicated, 1 means erase all curve data

NBUFFER = # of buffer in which all graphics is to be erased.

NOTE: if buffer 2 is erased, displays drawn in buffer 1 will "reappear"

20. MOVE_PIC (MOVE_OR_DUPLICATE)

MOVE_OR_DUPLICATE = 0 means move portion of picture selected to another location, 1 means actually duplicate the portion of the picture to another location

21. MERGE_PIC "NAME FILE"

22. DRAW_PIE (NBUF, NVAL, NCOL_ARRAY, VAL_ARRAY, SIZE,
INTERACTIVE)

NBUF = # of buffer in which pie is to be drawn
NVAL = # of values to be passed in next two arrays
NCOL_ARRAY = single dimensioned array of colors to
associate with each piece of the pie
VAL_ARRAY = single dimensioned array of values to
represent pieces of the pie, where the entire
pie represents the sum of those values
SIZE = overridden if INTERACTIVE is .TRUE., 1 means
the pie will be drawn with radius equal to
1/4 of a low res screen, 2 means the pie
will be drawn with radius equal to 1/2 of a
low res screen. In both cases the center of
the pie will be center screen.
INTERACTIVE = logical, .TRUE. means the user will
dynamically determine the pie's center and
radius by using an expandable circle which
appears on the screen. Obviously overrides
SIZE.

23. FILL_FRMC (NBUF, NCOL, MODE)

NBUF = # of buffer in which fill color is located
NCOL = # of color to fill with
MODE = 1 means fill until it encounters any other color (in that
buffer)
2 means fill until it encounters "itself" (the same
color/buffer combination)

*WARNING: Since FILL is so buffer/color sensitive the user should
exercise caution whenever filling.

DISTRIBUTION LIST

	<u>No. Copies</u>
Defense Technical Information Center ATTN: DTIC, TCA Cameron Station Alexandria, VA 22314	2
US Army Library Army Study Documentation and Information Retrieval System (ASDIRS) ANRAL-RS ATTN: ASDIRS Room 1a518, The Pentagon Washington, D.C. 20310	1
US Army TRADOC Analysis Command-WSMR ATTN: ATRC-WSL (Technical Library) White Sands Missile Range, NM 88002-5502	1
US Army TRADOC Analysis Command-FLVN ATTN: ATRC-FOA (Technical Info Center) Fort Leavenworth, KS 66027-5200	1
US Army Combined Arms Research Library (CARL) ATTN: ATCL-SWS-L Fort Leavenworth, KS 66027-5000	1