Edinburgh Regional Computing Centre

ERCC Graphics Manual

A description of the graphics facilities provided by ERCC.

2nd Edition
July 1979

Edinburgh Regional Computing Centre

ERCC Graphics Manual

A description of the graphics
facilities provided by ERCC.

CONTENTS

Chapter 5

List of Figures and Tables

Chapter 6

List of Figures

Preface to Second Edition

The first edition of the ERCC Graphics Manual, edited by John Murison, consisted of a contents list for Chapter 5 and Chapter 5 which described the graph plotting facilities available at ERCC. This manual was published in June 1977 and the intention was to later issue further chapters describing other graphical facilities. Since the first edition was published changes have been made to both the graphical facilities and the main computers behind them. This edition consists of a contents list for Chapters 5 and 6, Chapter 5 revised to describe the improved graph plotting facilities and access from the ICL 2900 series computers, Chapter 6 which describes interactive graphical facilities and an index covering both chapters.

The intention is still to issue further chapters in due course.

Neil Hamilton-Smith
May 1979

CHAPTER 5

GRAPH PLOTTING

5.1 INTRODUCTION

This chapter describes the basic graph plotting facilities provided by ERCC and explains how to use them. Programs using these facilities may be written in IMP or Edinburgh FORTRAN, and run under EMAS on the ERCC ICL System 4-75 configuration and on the ICL 2970, or under VME/B on the ICL 2980, or under OS on the NUMAC IBM configuration; or written in ICL FORTRAN for running under VME/B on the ICL 2980, or in IBM FORTRAN for running under OS at NUMAC.

Software

Graph plotting software is available in the form of two completely independent packages of routines. The ERCC Graphpack is recommended; however a Calcomp Basic Graphic Software Simulation Package is also made available, since FORTRAN programs obtained from external sources often use the Calcomp manufacturer's plotting routines. In either case a set of routines is provided. These are called by the user in his own program, and the result is that a file of plotter commands (the 'plotter file') is generated. If the user program has called the routines correctly, the plotter file commands, when obeyed by the appropriate graph plotter, cause the desired output to be drawn.

Notes on the packages

* The ERCC Graphpack conforms to the ERCC specification of basic plotting facilities. This states that the plotter output from each job must be 'independent', i.e. can neither corrupt nor be corrupted by plotter output from any other job; and that each job comprises

    one or more independent plotter files, each headed by a 'window' giving file identification information, and having an enforced total paper advance limitation; each such file contains

        one or more 'viewing windows', which may or may not be independent from one another but each of which is contained within the plotter dimensions, and comprises

            one or more drawings, referenced to one or more Cartesian coordinate systems, each comprising

                the basic drawing components - straight lines, curves, special graphic symbols, axes, text, numbers - referenced to the currently declared Cartesian coordinate system.

  The routines concerned with the first three divisions above are termed 'administrative'. They automatically transform the arbitrary coordinate system or systems to which the basic drawing components are referenced into positions on the plotter paper for each drawing in each window in each file.

  The ERCC Graphpack routines are available for use in programs written in IMP, Edinburgh FORTRAN, ICL FORTRAN or IBM FORTRAN.

* The Calcomp Basic Graphic Software Simulation Package conforms, in general, to the Calcomp manufacturer's specification of basic plotting facilities, as described in the Calcomp booklet 'Programming Calcomp Pen Plotters'. Unfortunately the original Calcomp software assumes no responsibility for differentiating between files, or ensuring that plotter output remains within bounds - all is left to the individual programmer. Hence, the Simulation Package has been upgraded by introducing the concepts of 'viewing windows', 'independence' and 'identification' of files and jobs, and by incorporating checking facilities with appropriate comments on receiving invalid data, thus satisfying the standards of plotter output control inherent in the ERCC graph plotting specification. The effect of this upgrading on existing programs which use the original Calcomp manufacturer's software should be minimal; it is discussed below in the Simulation Package description (Section 5.4).

  Calcomp software is FORTRAN based. The Simulation Package routines, although written in IMP, are therefore callable only from Edinburgh FORTRAN, ICL FORTRAN and IBM FORTRAN subprograms.

All of the graph plotting software is written in IMP, notwithstanding the source language of the calling program. Any complications this may cause are fully discussed in this chapter.

Hardware

The software contains references to six different graph plotters, detailed in Table 1 opposite. Any one of these may be chosen as the output device if it is available. In particular, the ERCC graph plotter is an EMAS output device connected via the Network Control Processor. Plotting to the ERCC plotter from NUMAC is controlled by EMAS acting as a Remote Job Entry terminal in such a way that plotter output appears to HASP (the spooling part of the system at NUMAC) as output to a remote Card Punch; a similar mechanism is used for plotting from the ICL 2980 to the plotter at Buccleuch Place Lane. The consequences of this method of plotting are discussed in 'Job Control Requirements' (Section 5.6).

Use of plotter files

Drawings generated by reference to the graph plotting software need not be plotted, immediately or at all. As an aid to program development and to give flexibility between graphic output devices, the routine TVIEW (described elsewhere in this manual) provides for the viewing of graph plotting software output on a Tektronix Storage Tube Display. Conversely, graph plotter output may be obtained from pictures created on a Tektronix display.

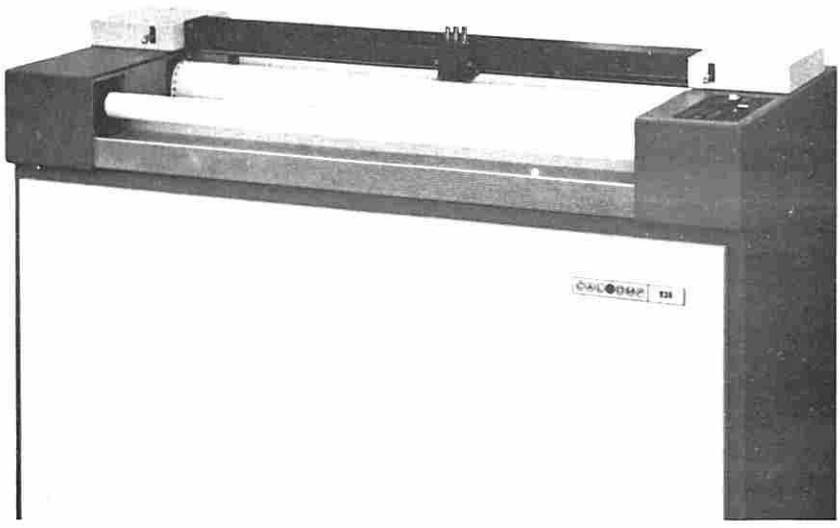
Efficiency

Two rules relating to speed of pen movement and time taken to raise or lower the pen can be deduced from the figures in Table 1; they give a guide to making efficient use of the plotter:

* minimise pen raise/lower commands, since they are relatively expensive

* minimise pen movements with the pen raised, since these movements produce no visible result on the paper

Both packages of routines described in this chapter follow these rules as far as possible, and in particular will resolve successive movements with the pen raised into a single direct movement to the next visible item in a drawing. This introduces the concept of a 'notional' pen position, i.e. where the pen would be if it had followed all the specified movements. The pen's actual position need only coincide with its notional position when the movement requested is with the pen in contact with the paper.



The Calcomp 936 Plotter

5.2 EQUIPMENT

The graph plotting software allows for reference to one of six different types of graph plotter. However they all work on the same principle, only differing in respect of their physical characteristics, and so a single description of the mode of operation suffices. The ERCC graph plotter is a Calcomp 936.

A cylindrical drum revolves on a horizontal axis and carries plotter paper from a feed roll at the rear to a take-up roll in front beneath a fixed carriage located parallel to the drum's axis. The drum revolves both forwards and backwards. The pen cartridge mechanism is mounted on the fixed carriage and traverses the carriage to the right and to the left. Micro-switches at each end of the carriage limit the effective paper width available for plotting. Normally there are no hardware error indications during drum revolution if there is no paper loaded or if the paper runs out. However, the Calcomp 936 at ERCC has had a microswitch fitted to stop the plotter in these circumstances.

By convention, the right-hand end of the carriage is designated the base for plotter measurements and is termed the 'plotter origin'.

X movements are performed by drum revolution

* forwards, advancing the paper (+)
* backwards, retarding the paper (-)

Y movements are performed by pen traverse

* along the carriage to the left (+)
* along the carriage to the right (-)

when the plotter is viewed from the front. Simultaneous drum revolution and pen traverse create diagonal movements in the four 45-degree directions. Each pen movement is made up of a number of small 'increments', each in one of the eight basic directions.

During drum movement and/or carriage movement, the pen mechanism may be raised above the paper, thus simply repositioning the pen, or lowered in contact with the paper, to produce a visible line. The standard pen used is a black biro but blue, green and red biros are also provided. Liquid ink output may also be obtained; at ERCC this must be authorised by Mr M.P. Baillie (031 667-1081 ext. 2902).

Table 1 identifies the different characteristics of plotters which the software recognises.

Characteristics \ Plotter Model	(1) Calcomp 936 ERCC KB	(2) Calcomp 663 ERCC BPL	(3) Calcomp 663 ERCC BPL	(4) Calcomp 564	(5) Calcomp 564	(6) Calcomp 565 SIAE, BUSH
Plotter units	cm	in	in	cm	in	cm
Increment size	.005cm	.0025in	.0025in	.01cm	.005in	.01cm
Speed in inc/sec — pen up	2600	450	450	300	300	300
Speed in inc/sec — pen down	1800	450	450	300	300	300
No. of pen holders	3	1	1	1	1	1
Raise/lower pen time	.1sec	.1sec	.1sec	.1sec	.1sec	.1sec
Max. plotting width	82.5cm	28.5in	10.25in	72.5cm	28.5in	28cm
Length of paper roll	120ft	120ft	120ft	120ft	120ft	120ft
Max. window size	3000cm	1200in	1200in	3000cm	1200in	3000cm
Window separation	5cm	2in	2in	5cm	2in	5cm
Default max. paper usage — students	90cm	36in	36in	90cm	36in	90cm
Default max. paper usage — others	185cm	72in	72in	185cm	72in	185cm

Table 1 Plotter hardware details

Routine Name		Purpose	Page
IMP	FORTRAN		
PLOTTERTYPE	PLTYPE	Sets the plotter code number.	5-5
CHCODE	CHCODE	Sets the plotter program character set type.	5-6
OPENPLOTTER	OPENGR	Initialises the Graphpack prior to plotter file generation.	5-6
PLOTFAULT	IGRERR	Enables most plotting errors to be detected and handled by the user's program.	5-7
GRAPHPAPER	GRPAPR	Sets the graph paper allowance.	5-8
SETPLOT	GRAREA	Specifies a viewing window on the graph paper.	5-9
SCALE	SCALGR	Defines a transformation from the user's coordinate system to the viewing window coordinate system.	5-11
PLOTRECS	IGRREC	Indicates the number of records written to the plotter file to date.	5-12
CLOSEPLOTTER	CLOSGR	Closes the current plotter file.	5-12
FILEGRAPH	FILGR	Saves a single window part-drawing for merging in subsequent graph plots.	5-13
MERGEGRAPH	MERGGR	Merges a part-drawing (saved using FILEGRAPH/FILGR) with the current graph plot.	5-13
CHANGEPEN	CHPNGR	Causes the plotter pen to be changed.	5-14
PLOT	PLOTGR	Draws a straight line between two specified points. The line may be continuous, pecked or invisible.	5-15
AREAFLAG	GRARFL	Enables 'out of area plotting' messages to be suppressed.	5-17
PENPOSITION	PPOSGR	Returns the current pen position coordinates.	5-17
AXIS	AXISGR	Draws a graduated axis from a specified point in a specified direction.	5-17
POINTSYMBOL	PSYMGR	Draws one of fifteen special symbols.	5-19
LINEGRAPH	LINESG	Draws a piecewise straight line through a given array of points, optionally adding a symbol at each.	5-19
CURVE	CURVGR	Draws a piecewise cubic through a given array of points, optionally adding a symbol at each.	5-20
ANNOTATE	ANNOGR	Prior to the output of text, enables its position, size and orientation to be specified.	5-22
PLOTSYMBOL	DRSYMG	Draws a specified character from a standard set.	5-24
PLOTSTRING	DRSTRG	Draws a string of characters.	5-24
PLOTNUMBER	DRNUMG	Outputs a decimal number in a specified format.	5-24

Table 2 ERCC Graphpack Routines

5.3 ERCC GRAPHPACK

Notation

In the presentation of each routine description the IMP externalroutinespec is given first, as this states specifically the type and precision of each parameter. Bracketed with this statement is the IMP routine call. The FORTRAN routine call, for Edinburgh, ICL and IBM FORTRAN, then follows.

The standard FORTRAN variable-naming convention is used throughout, i.e. integer variables begin with one of the letters I, J, K, L, M, N, and all other variables are of type REAL. The one exception is the IMP string type, which does not exist in FORTRAN.

Where underlining is used, as in externalroutinespec above, this is a typographical notation meaning "write the word in capital letters, preceded by a % character and followed by a non-alphabetic character".

Precision

Note that, for consistency with system standards, all real variables passed as parameters to the plotter routines must be in double precision, i.e. longreal in IMP and REAL*8 in FORTRAN. In IMP programs this strictly only applies to those real variables passed by name. In FORTRAN programs, however, all real parameters - including constants - must be explicitly of type REAL*8; e.g. 5.7D0.

Similarly, in IMP programs integer parameters may be passed as byteinteger, shortinteger or integer. In FORTRAN programs all integer parameters must be of type INTEGER*4.

Identification of plotter output

Plotter output is automatically identified with your jobname, the machine on which you are running this job, the date and time of file generation and the file name (if accessible). You are, however, strongly recommended to add your name and delivery point, using a maximum of 30 characters, in order to prevent your output being lost. Simply program the statements

IMP	FORTRAN
OPENPLOTTER(IFILE)	CALL OPENGR(IFILE)
PLOTSTRING('A.N.OTHER, DEPT. ABC')	CALL DRSTRG('A.N.OTHER, DEPT. ABC',20)

i.e. call the appropriate string output routine immediately after opening the plotter. These routines are described below. If no delivery information is available "**** Please set delivery ****" will appear automatically.

5.3.1 ADMINISTRATIVE ROUTINES

******* PLOTTERTYPE / PLTYPE *******

IMP
- externalroutinespec PLOTTERTYPE(integer N)
- PLOTTERTYPE(N)

FORTRAN CALL PLTYPE(N)

This routine may be called at any time to indicate that future calls on graph plotting routines are intended to be with reference to the characteristics of plotter N, where 1<=N<=6 (Table 1). The routine call is ignored if the current plotter type is already N.

The effect of this routine is to close the currently open plotter file, if one is open, and to establish the characteristics of the required plotter in a 'current plotter type' information area. This area is used by the Graphpack and, apart from the effect of this routine, the user need not be aware of its existence.

If this routine call is omitted the characteristics of the Calcomp 936 plotter (which corresponds to N=1) will be assumed by default. If you wish to reference a different plotter you should precede the call on OPENPLOTTER/OPENGR, described below, by a call on this routine.

Of course it is only sensible to send a plotter file to a plotter which has characteristics compatible with those assumed by the Graphpack during the generation of the file, i.e. one which has the same increment size, and dimensions large enough to accommodate the largest viewing window.

Example	IMP	FORTRAN
	PLOTTERTYPE(2)	CALL PLTYPE(2)

would imply that a Calcomp Model 663, with an increment size of .0025 inch and plotting width 28.5 inches, was to be used.

******* CHCODE *******

IMP { externalroutinespec CHCODE(integername N)
    { CHCODE(N)

FORTRAN CALL CHCODE(N)

The graph plotting software needs to know the character code used by your program. This is dependent upon the programming language - IMP and FORTE use ISO, FORTRANG, ICL FORTRAN and IBM FORTRAN use EBCDIC.

The parameter N indicates ISO when even and EBCDIC when odd. By default ISO is assumed, unless IBM FORTRAN is being used when EBCDIC is the default.

The call on this routine should precede calls on any other plotting routine except PLOTTERTYPE/PLTYPE above.

******* OPENPLOTTER / OPENGR *******

IMP { externalroutinespec OPENPLOTTER(integer ICHAN)
    { OPENPLOTTER(ICHAN)

FORTRAN CALL OPENGR(ICHAN)

If you wish to create a plotter file then this routine must be called before any other plotting routine, with the exception of PLOTTERTYPE/PLTYPE, described above.

The effect of this routine call is to close the currently open plotter file, if one is open, and to reset the initialisation parameters for the current plotter type. Consequently, although it is permissible to create several plotter files within a single run of your program, only one of them may be open at a time.

The value of the parameter ICHAN specifies the output channel number which is to be related to this file. The value of ICHAN may be 96, the system-provided plotter file, or may lie in the range 1<=ICHAN<=80; see 'Job Control Requirements', Section 5.6. If the channel is not defined or is defined improperly then the program will terminate with one of the following messages:-

```
***(CHANNEL nn) Call no. mm to 'OPENPLOTTER,
                               OPENGR      :-
***                message                .***
```

where message = No file definition statement
or Plotter file not defined as SQFILE
or Plotter file not F80, i.e. card images

Successive calls on OPENPLOTTER/OPENGR may specify the same ICHAN value but if this points to the same file name or device the software will register plotter fault 12, with the message

```
***(CHANNEL nn) Call no. mm to 'OPENPLOTTER,
                               OPENGR      :-
***File extension not allowed.***
```

and the program will be terminated.

If any plotter file is re-used in the current run of your program then the new drawings will overwrite the previous drawings it contained and they will be lost.

Example	IMP	FORTRAN
	OPENPLOTTER(50)	CALL OPENGR(50)

will initialise a plotter file on channel 50. See 'Job Control Requirements', Section 5.6.

N.B. If the call on OPENPLOTTER/OPENGR is omitted the package is said to be in 'parameter checking' mode. No output file will be created but all routine parameters will be checked for validity and appropriate error messages produced. See PLOTFAULT/IGRERR below.

******* PLOTFAULT / IGRERR *******

externalintegerfnspec PLOTFAULT

IMP I=PLOTFAULT

FORTRAN I=IGRERR(N) where N is a dummy parameter

The graphics software makes various checks on the validity of the values given to parameters passed to the different routines and will, where possible, supply default values when errors occur. In certain circumstances, however, this is impossible and an appropriate message is then sent to a character file with ddname STREAM99. The full list of faults and the routines in which they occur is as follows:

CODE	MESSAGE	ROUTINE
1	Invalid drawing area declaration	SETPLOT/GRAREA
2	Plotter output exceeded	SETPLOT/GRAREA
3	Invalid scaling factor(s)	SCALE/SCALGR
4	Invalid pen up/down code	PLOT/PLOTGR
5	Dash/gap value(s) negative	PLOT/PLOTGR
6	Data and drawing range incompatible	CURVE/CURVGR
7	Independent variable data out of order	CURVE/CURVGR
8	Direction code error	AXIS/AXISGR
9	Illegal use of channel 96	FILEGRAPH/FILGR or MERGEGRAPH/MERGGR
10	Not a valid plotter file	MERGEGRAPH/MERGGR
11	Excessive out-of-area drawing	the current routine call
12	File extension not allowed	OPENPLOTTER/OPENGR
13	Too few data points	CURVE/CURVGR
14	Graduation intervals specified wrongly	AXIS/AXISGR
15	Units not 'INS' or 'CMS' codes	SETPLOT/GRAREA or GRAPHPAPER/GRPAPR
16	No file definition statement	OPENPLOTTER/OPENGR or MERGEGRAPH/MERGGR
17	Plotter file not defined as SQFILE	OPENPLOTTER/OPENGR or MERGEGRAPH/MERGGR
18	Plotter file not F80, i.e. card images	OPENPLOTTER/OPENGR or MERGEGRAPH/MERGGR
19	File is not accessible	MERGEGRAPH/MERGGR

If the initialisation call on OPENPLOTTER/OPENGR has been omitted the graphics software will continue to check the values of other routine parameters and comment on them if appropriate. Normally, however, when a plotter file is being created, the following sequence of events occurs:

* an error flag is set to the code for the error found, and in the case of the routines which may have more than one error the flag values could be

  PLOT/PLOTGR - 4, or 5, or 69 (i.e. 4x16 + 5)

  AXIS/AXISGR - 14, or 8, or 232 (i.e. 14x16 + 8)

  CURVE/CURVGR - 6, or 7, or 13, or 103 (i.e. 6x16 + 7), or 109 (i.e. 6x16 + 13), or 125 (i.e. 7x16 + 13) or 1661 (i.e. 6x16x16 + 7x16 + 13)

* on the occurrence of errors 2, 9, 10, 11, 12, 16, 17, 18 or 19 the program is terminated

An IMP program may then trap any of the non-drastic faults under run-time fault 19 (NUMAC and ICL 4-75) or event 11 (ICL 2970 and 2980), and by calling PLOTFAULT identify the actual error or errors from the function result, which is set to the error flag value. This action also clears the error flag.

Example

```
NUMAC                                   2970
ICL 4-75                                2980

fault 19 -> ERROR                       onevent 11 start
    .                                   -> SW(PLOTFAULT)
    .                                   finish
ERROR: -> SW(PLOTFAULT)
    .
    .
```

Plotter fault recovery in FORTRAN is different. Each Graphpack routine checks the error flag on entry and terminates the program if it has been set (by a previous Graphpack routine). Therefore the recovery operation should, if required, follow every routine which could set the flag. The statement

```
I=IGRERR(N)
```

will identify the error flag value and clear the error; the appropriate recovery sequence can then be followed.

******* GRAPHPAPER / GRPAPR *******

```
IMP    { externalroutinespec GRAPHPAPER(longreal XLNGTH,integer IUNITS)
       { GRAPHPAPER(XLNGTH,IUNITS)

FORTRAN CALL GRPAPR(XLNGTH,IUNITS)
```

File initialisation (via OPENPLOTTER/OPENGR) automatically imposes a paper advance limit for the file thus defined (see Table 1). The limit may be altered by calling GRAPHPAPER/GRPAPR either to reduce it subject to a minimum of 15 inches (38.1 centimetres), or to increase it subject to the length of a roll of plotter paper (see Table 1). If you have a student jobname, however, you will not be allowed to alter the specified limits which are 36 inches (90 centimetres).

Only one alteration of the limit is allowed per file, and then only if the call on GRAPHPAPER/GRPAPR follows the call on OPENPLOTTER/OPENGR and precedes the first call on SETPLOT/GRAREA (see below) in that file.

The new allocation, XLNGTH, is measured in IUNITS, which is specified as follows:

IMP		FORTRAN
0 or M'CMS'	for centimetres	0 or 'CMS'
1 or M'INS'	for inches	1 or 'INS'

XLNGTH comprises (see Figure 1): the file identification window of 2 inches (5.08 cm), 2 inches (5.08 cm) separation, the sum of all paper advances caused by successive viewing window requests and any inter-window separations. See also SETPLOT/GRAREA, described below.

If the IUNITS parameter value is not one of the above options, plotter fault 15 is registered, with the message

```
***(CHANNEL nn) Call no. mm to 'GRAPHPAPER, GRPAPR':-
***Units not 'INS' or 'CMS' codes.***
```

and the program will terminate unless it is in parameter checking mode (see OPENPLOTTER/OPENGR) or you use the fault trapping procedure (see PLOTFAULT/IGRERR).

Example

```
IMP                                     FORTRAN

GRAPHPAPER(200,M'INS')                  CALL GRPAPR(200.D0,'INS')
```

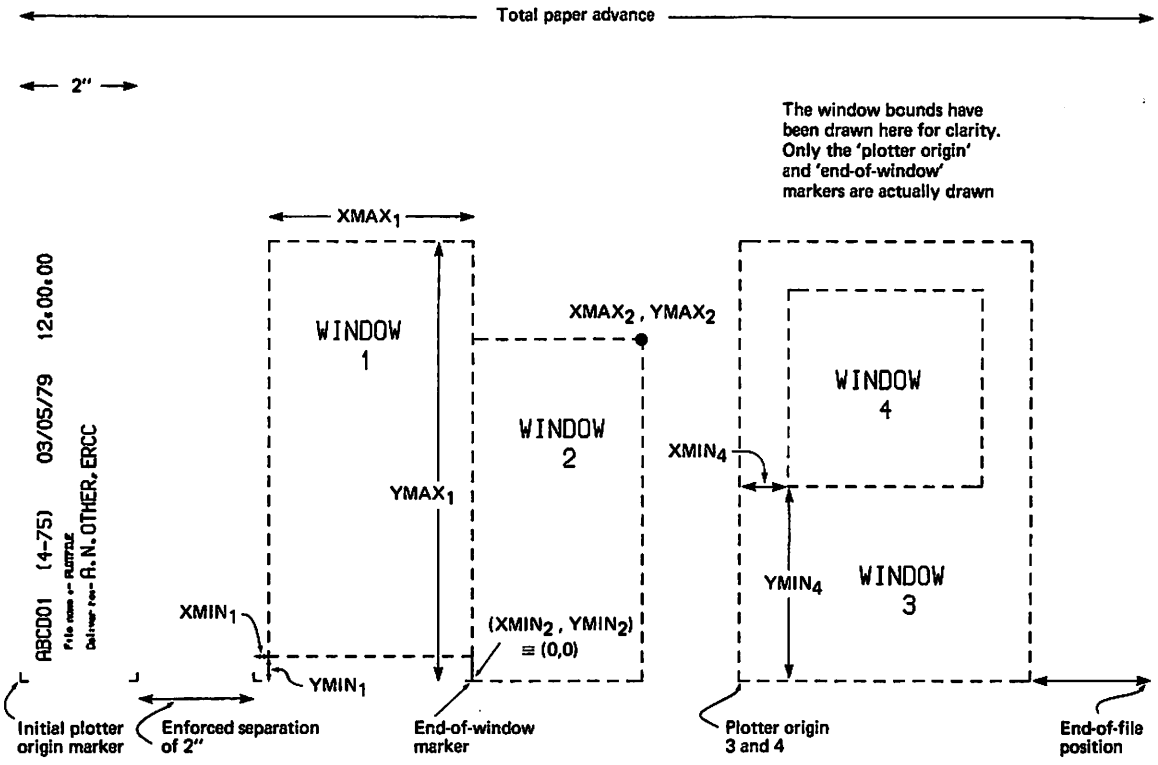


Figure 1 Examples of viewing windows and their relative positions

******* SETPLOT / GRAREA *******

```
IMP      { externalroutinespec SETPLOT(longreal XMIN,YMIN,XMAX,YMAX,integer IUNITS)
         { SETPLOT(XMIN,YMIN,XMAX,YMAX,IUNITS)

FORTRAN  CALL GRAREA(XMIN,YMIN,XMAX,YMAX,IUNITS)
```

After the file initialisation and possibly a resetting of the paper advance limit, this routine defines a new viewing window within which all drawing will be imprisoned until the next window is defined, by another call on the routine. If the window defined is the first in the file it will be preceded by the system-generated file identification window containing your jobname, the machine on which you are running the program, the date and the file run-time. If they are accessible then the file name and your delivery point are also drawn.

Apart from their parameter lists being validated, all references to basic drawing routines will be ignored until this routine has been called to define a window.

There are no restrictions, except that implied by the total paper advance limit, on the number of windows you may define within a particular file. Figure 1 shows a possible series of windows.

The parameter IUNITS determines the position of the 'plotter origin' from which the actual window frame will be referenced, and also specifies the unit of measurement for the window's dimensions. The possible values of IUNITS and their meanings are as follows:

IMP		FORTRAN
0 or M'CMS'	for centimetres	0 or 'CMS'
1 or M'INS'	for inches	1 or 'INS'

causes an 'end-of-window' marker to be drawn at the X-extremity of the previous window, a paper advance of another 2 inches (5.08 cm), and then sets the new plotter origin at this point,

where the plotter origin marker is drawn. This option is enforced for the first window in the file, no matter which option you request.

IMP		FORTRAN
2 or M'CMSA'	for centimetres	2 or 'CMSA'
3 or M'INSA'	for inches	3 or 'INSA'

causes an 'end-of-window' marker to be drawn at the X-extremity of the previous window, and sets this point as the new plotter origin, where the plotter origin marker is drawn - thus abutting the two windows, as shown in Figure 1 (window 2 with window 1).

IMP		FORTRAN
4 or M'CMSV'	for centimetres	4 or 'CMSV'
5 or M'INSV'	for inches	5 or 'INSV'

simply retains the present plotter origin, thus allowing for an inset window (Figure 1, window 4 within window 3), or a separate window further across the pen carriage to utilise the full width of the plotter. No 'end-of-window' marker is drawn.

In each case the new viewing window's unit of measurement, assumed by subsequent routines, is as specified by IUNITS.

If the IUNITS parameter value is not one of the options listed above, plotter fault 15 is registered, with the message

```
***(CHANNEL nn) Call no. mm to 'SETPLOT,
                               GRAREA ':-
***Units not 'INS' or 'CMS' codes.***
```

and the program will terminate, after closing the plotter file, unless it is in parameter checking mode (see OPENPLOTTER/OPENGR) or you use the fault trapping procedure (see PLOTFAULT/IGRERR) .

If IUNITS is correct the new viewing window is defined to have its lower left-hand corner at the point (XMIN,YMIN) and its upper right-hand corner at the point (XMAX,YMAX), both relative to the new plotter origin in the declared unit of measurement; see Figure 1.

The values XMIN, YMIN, XMAX, YMAX are checked to ensure that XMIN<XMAX, YMIN<YMAX, that the window definition lies within the plotter bounds (Table 1) and that it does not attempt to place any part of the window left of the new plotter origin. If any of these conditions is not satisfied, plotter fault 1 is registered, producing the message

```
***(CHANNEL nn) Call no. mm to 'SETPLOT,
                               GRAREA ':-
***Invalid drawing area declaration.***
```

and the program is terminated unless it is in parameter checking mode (see OPENPLOTTER/OPENGR) or you use the fault trapping procedure (see PLOTFAULT/IGRERR).

A check is also kept on the total paper advance in the file. If the paper advance required to clear the new viewing window would exceed the current paper advance limit then the window request is rejected, plotter fault 2 is registered and the program is terminated unless it is in parameter checking mode (see OPENPLOTTER/OPENGR), with the message

```
***(CHANNEL nn) Call no. mm to 'SETPLOT,
                               GRAREA ':-
***Plotter output exceeded.***
```

If no errors are detected the window is defined and the pen is notionally positioned, in raised status, at the point (XMIN,YMIN) - termed the 'window origin' - after having drawn the plotter origin marker.

Example

The successive windows shown in Figure 1 could be defined by a sequence of statements such as the following:

```
IMP                                     FORTRAN

OPENPLOTTER(50)                         CALL OPENGR(50)
PLOTSTRING('name and delivery')         CALL DRSTRG('name and delivery',17)
GRAPHPAPER(200,M'INS')                  CALL GRPAPR(200.D0,'INS')
SETPLOT(2,1,20,30,M'INS')               CALL GRAREA(2.D0,1.D0,20.D0,30.D0,'INS')
SETPLOT(0,0,15,25,M'INSA')              CALL GRAREA(0.D0,0.D0,15.D0,25.D0,'INSA')
SETPLOT(0,0,25,30,M'INS')               CALL GRAREA(0.D0,0.D0,25.D0,30.D0,'INS')
SETPLOT(7,12,18,21,M'INSV')             CALL GRAREA(7.D0,12.D0,18.D0,21.D0,'INSV')
```

N.B. You may find on receiving plotter output that a window is not as wide as you requested, i.e. (XMAX-XMIN). The 'end-of-window' marker will be drawn either

* 2 inches beyond the largest visible X-value reached, or

* at the XMAX position specified (strictly, at the largest XMAX position specified, if two or more windows use the same plotter origin)

relative to the previous plotter origin, whichever produces the shorter paper advance. The total paper advance check, however, will always use the actual XMAX values quoted in the routine calls for its calculations.

******* SCALE / SCALGR *******

```
IMP     { externalroutinespec SCALE(longreal XORIGN,YORIGN,XSCALE,YSCALE,THETA)
        {
        { SCALE(XORIGN,YORIGN,XSCALE,YSCALE,THETA)

FORTRAN  CALL SCALGR(XORIGN,YORIGN,XSCALE,YSCALE,THETA)
```

Having given a successful window definition, you must now provide transformation parameters between some arbitrary Cartesian coordinate system which your program references and the current window, such that the drawing you wish to produce is automatically mapped into the window's bounds. This routine enables you to specify the transformation; it does not produce any pen movement.

Your coordinate system origin transforms into the position (XORIGN,YORIGN), which is measured relative to the current 'window origin', i.e. the point (XMIN,YMIN) specified via the latest call of SETPLOT/GRAREA (see above). The unit of measurement of XORIGN and YORIGN is that specified in the same call of SETPLOT/GRAREA. Whether your system origin actually lies within the window or not is immaterial.

Your system is orientated such that the directions of its X and Y axes lie at an angle THETA (degrees) counter-clockwise to the window X and Y directions.

The scaling factors used to map your X and Y values onto the window's units of measurement are XSCALE and YSCALE, defined as the number of window units representing respectively 1 X- and 1 Y-unit in your coordinate system. Thus if the unit of measurement specified when the window was defined was centimetres, a setting of XSCALE to 10.0 would mean that 1 X-unit in your coordinate system would be represented by 10 centimetres on the graph paper.

Plotter fault 3 is registered, producing the message

```
***(CHANNEL nn) Call no. mm to 'SCALE
                               SCALGR':-

***Invalid scaling factor(s).***
```

unless XSCALE>0 and YSCALE>0, and the program is terminated unless it is in parameter checking mode (see OPENPLOTTER/OPENGR) or you use the fault trapping procedure (see PLOTFAULT/IGRERR).

You may redefine your coordinate system any number of times within a single window.

It should not be re-opened to add more drawing within the same run of the program since this will destroy its previous contents. If further drawing is required you must define and open another file; see OPENPLOTTER/OPENGR above.

If your program terminates for some reason before this routine has been called for the current file, then the 'end-of-file' marker, and possibly a small part of the drawing performed immediately prior to the failure, will be missing.

******* FILEGRAPH / FILGR *******

IMP { externalroutinespec FILEGRAPH
    { FILEGRAPH

FORTRAN CALL FILGR

For some applications there may be a constant component in a complex drawing or drawings, e.g. a coastline map or a grid system. The repeated generation of these constant features in each job requiring them would be extremely wasteful of computer time. There is clearly a need for a facility which allows retention of a file containing a part-drawing such that it may subsequently be recovered and incorporated into another drawing at any time during the generation of a file. This routine provides the facility.

The part-drawing should be created in a plotter file containing a single viewing window whose definition, in terms of size and position relative to its plotter origin, will be compatible with any future window onto which it may be mapped. Calling this routine in preference to CLOSEPLOTTER/CLOSGR above causes a command to be added to the file to position the pen back at this plotter origin, i.e. a null total displacement, followed by closure of the file and the message

(CHANNEL nn) End of plotter file after mm records.

See also 'Job Control Requirements', Section 5.6, for administrative considerations.

If your program fails before this routine has been called then the drawing will be incomplete and you must re-create the file from the beginning.

As before, further reference within the same run of the program to the file in which this drawing was produced, for the purpose of adding more drawing features, will simply overwrite its previous contents.

You are not allowed to use channel 96 for saving part-drawings. If you do the file will not be closed properly, your drawing will be incomplete, and the program will terminate - unless it is in parameter checking mode (see OPENPLOTTER/OPENGR) - with the message

***(CHANNEL nn) Call no. mm to 'FILEGRAPH, FILGR':-

Illegal use of channel 96.

No limit is placed on the number of part-drawings you may save during a single run of your program, but each one must be in a separate file.

******* MERGEGRAPH / MERGGR *******

IMP { externalroutinespec MERGEGRAPH(integer INCHAN)
    { MERGEGRAPH(INCHAN)

FORTRAN CALL MERGGR(INCHAN)

This routine provides for recovery of part-drawings retained by use of the routine FILEGRAPH/FILGR above. See 'Job Control Requirements', Section 5.6, for further administrative considerations. More than one merger is allowed in a single program run.

The plotter file associated with channel number INCHAN is assumed to contain a single viewing window which can be sensibly superimposed upon the current window; i.e. the two windows should be compatible in their positions relative to the current plotter origin and in their dimensions, since no check is made on these factors. Moreover, this routine does not record

any visible X values for SETPLOT/GRAREA to use in its calculation for the end-of-window position - you must issue a visible move in the current window in order to register the merging window width.

Merging of the part-drawing begins after an initial pen repositioning at the current plotter origin, and concludes by notionally repositioning the pen at the coordinate position in force when the merger request was encountered.

If your program fails during merging then the drawing will be in an indeterminate state and will require complete re-drawing.

You are not allowed to use channel 96 for designating a previously created file to be merged: plotter fault 9 would be registered, with the message

```
***(CHANNEL nn) Call no. mm to 'MERGEGRAPH  ':-
                               MERGGR
***Illegal use of channel 96.***
```

If the channel you specify has not been defined or has been defined improperly then one of the following messages will appear:-

```
***(CHANNEL nn) Call no. mm to 'MERGEGRAPH  ':-
                               MERGGR
***              message          .***
```

where message = No file definition statement
or Plotter file not defined as SQFILE
or Plotter file not F80, i.e. card images
or File is not accessible

A check is also made to ensure that the file associated with channel number INCHAN really is a plotter file. If not then plotter fault 10 is registered, with the message

```
***(CHANNEL nn) Call no. mm to 'MERGEGRAPH  ':-
                               MERGGR
***Not a valid plotter file.***
```

The program will terminate on detection of any of these errors unless it is in parameter checking mode (see OPENPLOTTER/OPENGR).

Example

```
IMP                                  FORTRAN

MERGEGRAPH(51)                       CALL MERGGR(51)
```

******* CHANGEPEN / CHPNGR *******

```
IMP     { externalroutinespec CHANGEPEN(integer IPEN)
        { CHANGEPEN(IPEN)

FORTRAN CALL CHPNGR(IPEN)
```

At the start of each plotter file a black biro is automatically selected as the standard pen. You may request a different pen at any time after the first viewing window has been successfully defined by a call on SETPLOT/GRAREA. However, the operation can be costly in terms of pen movement. When a pen change is requested and the pen required is not already mounted - you are using a single-pen plotter say - the current pen is automatically repositioned at the plotter origin related to the current window, so that the registration of the new pen can be checked after it is selected. Following the pen change the pen is notionally returned to the position in force when the pen change request was encountered.

If you have a student jobname any change pen requests will be ignored.

The value of IPEN specifies the required pen as follows:

1	black biro (initially and by default)	
2	blue biro	
3	green biro	
4	red biro	
5	black liquid ink, .1mm diameter pen	
6	black liquid ink, .2mm diameter pen	At ERCC
7	black liquid ink, .3mm diameter pen	use of these pens must
8	black liquid ink, .4mm diameter pen	first be authorised by
9	black liquid ink, .5mm diameter pen	Mr M.P. Baillie
10	black liquid ink, .6mm diameter pen	(031 667-1081 ext. 2902)
11	black liquid ink, .8mm diameter pen	

Any other value of IPEN will cause the black biro to be selected by default. If the pen requested is already selected then this routine call has no effect.

Since the Calcomp Model 936 has a three-pen turret (see Table 1) you can, when using the 936 (plotter type 1), eliminate the time taken to change a pen cartridge physically if you use only pen codes 1, 2, 4; i.e. black, blue and red biro respectively.

Example	IMP	FORTRAN
	CHANGEPEN(3)	CALL CHPNGR(3)

will select the green biro for drawing, until further notice.

It is good practice to draw all you require with one pen colour in a given window, before changing to another colour. This is especially the case with single pen plotters because operator intervention is needed to change the pen cartridge.

At present no limit is placed on the number of pen changes during a single run of your program. This may be reviewed if the good practice described above is not adhered to.

5.3.2 BASIC DRAWING COMPONENT ROUTINES

The following eleven routines constitute the basic drawing components from which you may build complete drawings. All of these routines specify positional parameters in terms of the x- and y-units of your program's co-ordinate system, the parameters already specified in the calls to SETPLOT/GRAREA and SCALE/SCALGR defining how this system is to be automatically mapped onto the current drawing window.

******* PLOT / PLOTGR *******

IMP
```
externalroutinespec PLOT(integer IPEN,longreal TOX,TOY,DASH,GAP)
PLOT(IPEN,TOX,TOY,DASH,GAP)
```

FORTRAN CALL PLOTGR(IPEN,TOX,TOY,DASH,GAP)

This is the basic drawing routine, and is called at some point by all of the other basic component routines. It moves the pen in a straight line from its current position to the point (TOX,TOY) in your coordinate system. The move is subject to a check that it does not contravene the current viewing window bounds.

If IPEN=1 the straight line is invisible and represents a command simply to reposition the pen. If IPEN=2 the line is visible. In both these cases the DASH and GAP parameter values are not used but they must both be non-negative.

If IPEN=0 a dashed line is drawn to (TOX,TOY), with each dash DASH X units long and each gap GAP X units long. The line always begins and ends with a dash.

Any IPEN value other than 0, 1 or 2 will register plotter fault 4, with the message

```
***(CHANNEL nn) Call no. mm to 'PLOT
                               PLOTGR':-
***Invalid pen up/down code.***
```

A negative assignment to either DASH or GAP will register plotter fault 5, with the message

```
***(CHANNEL nn) Call no. mm to 'PLOT'
                              'PLOTGR':-
***Dash/gap value(s) negative.***
```

In either of these cases no pen movement occurs, and the program will terminate unless it is in parameter checking mode (see OPENPLOTTER/OPENGR) or the fault has been trapped (see PLOTFAULT/IGRERR).

If the point (TOX,TOY) transforms to a point outside the viewing window the move will be curtailed at the window boundary. The first time this occurs in a plotter file, and every 50th time thereafter, a warning message is sent to STREAM99:

```
***(CHANNEL nn) Call no. mm to 'name' is drawing out of area.***
```

where 'name' is the name of the routine which caused the boundary contravention. After 20 such messages from one plotter file the program is terminated, with the further message

```
***(CHANNEL nn) Call no. mm to 'name':-
***Excessive out-of-area drawing.***
```

Until this happens the software is continuously checking and 'scissoring' where necessary to ensure that only that part of the drawing which maps onto the viewing window will actually be drawn. See also AREAFLAG/GRARFL below.

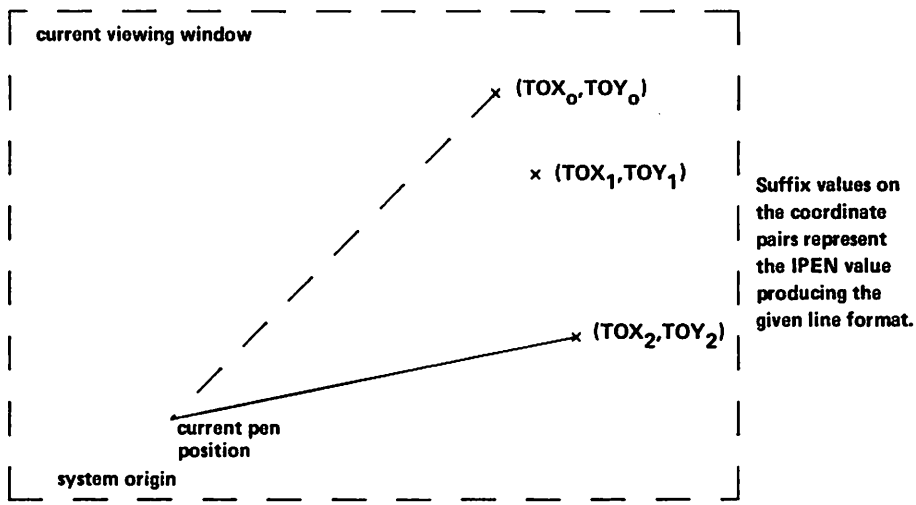


Figure 3 Types of line drawn by PLOT/PLOTGR

Example

The three different types of line demonstrated in Figure 3 could be drawn by the following statements:

IMP

```
PLOT(0,TOX,TOY,DASH,GAP)
PLOT(1,TOX,TOY,0,0)
PLOT(2,TOX,TOY,0,0)
```

FORTRAN

```
CALL PLOTGR(0,TOX,TOY,DASH,GAP)
CALL PLOTGR(1,TOX,TOY,0.D0,0.D0)
CALL PLOTGR(2,TOX,TOY,0.D0,0.D0)
```

******* AREAFLAG / GRARFL *******

```
IMP     { externalroutinespec AREAFLAG(string (3) S)
        { AREAFLAG(S)

FORTRAN  CALL GRARFL(IFLAG)
```

This routine has been provided for those people who wish to use the viewing window as a 'scissoring' tool, and who do not care therefore about plotting out of the window bounds.

The effect of the routine is to switch on or off, according to the parameter value, the monitoring of out-of-area plotting. By default monitoring is switched on at the start of each plotter file.

Example

```
IMP                                  FORTRAN

AREAFLAG('ON')                       CALL GRARFL('ON')
AREAFLAG('OFF')                      CALL GRARFL('OFF')
```

******* PENPOSITION / PPOSGR *******

```
IMP     { externalroutinespec PENPOSITION(longrealname X,Y)
        { PENPOSITION(X,Y)

FORTRAN  CALL PPOSGR(X,Y)
```

If you have forgotten exactly where the pen has been moved to then this routine will inform you. The present pen position, which will be on the viewing window boundary if the last move was curtailed, is returned via the parameters X and Y, in your current coordinate system X- and Y-units.

******* AXIS / AXISGR *******

```
IMP     { externalroutinespec AXIS(longreal X,Y,integer IDIRN,longreal TICINT, c
        {                                       integer INTNO)
        { AXIS(X,Y,IDIRN,TICINT,INTNO)

FORTRAN  CALL AXISGR(X,Y,IDIRN,TICINT,INTNO)
```

Axes are a common feature of graphical output. This routine draws a graduated line parallel to one of your coordinate system's axes. Only that portion of the line which lies within the current viewing window will be visible.

The start point of the graduated line is the point (X,Y) in your coordinate system and the pen is initially positioned there by the routine. The line direction is given by the parameter IDIRN, which may take the following values:

IMP		FORTRAN
1 or M'+X' or 'X'	for the positive X-direction	1 or '+X' or 'X'
2 or M'+Y' or 'Y'	for the positive Y-direction	2 or '+Y' or 'Y'
3 or M'-X'	for the negative X-direction	3 or '-X'
4 or M'-Y'	for the negative Y-direction	4 or '-Y'

INTNO tick marks are drawn along the line at TICINT intervals in the units of the specified direction (X or Y), the total line length drawn being INTNO*TICINT units. The tick marks are perpendicular to the line and, unalterably, of length 0.05 inch (0.125 centimetre) on each side of it.

Subject to non-contravention of the viewing window boundaries the pen is finally placed at the far end of the line from (X,Y), in contact with the paper.

Unless TICINT>0 and INTNO>0, plotter fault 14 will be registered, with the message

```
***(CHANNEL nn) Call no. mm to 'AXIS  ':-
                              AXISGR
***Graduation intervals specified wrongly.***
```

If the value of IDIRN is not one of the options given above, plotter fault 8 is registered, with the message

```
***(CHANNEL nn) Call no. mm to 'AXIS  ':-
                              AXISGR
***Direction code error.***
```

If either condition is violated the program will terminate, unless it is in parameter checking mode (see OPENPLOTTER/OPENGR) or the fault has been trapped (see PLOTFAULT/IGRERR).

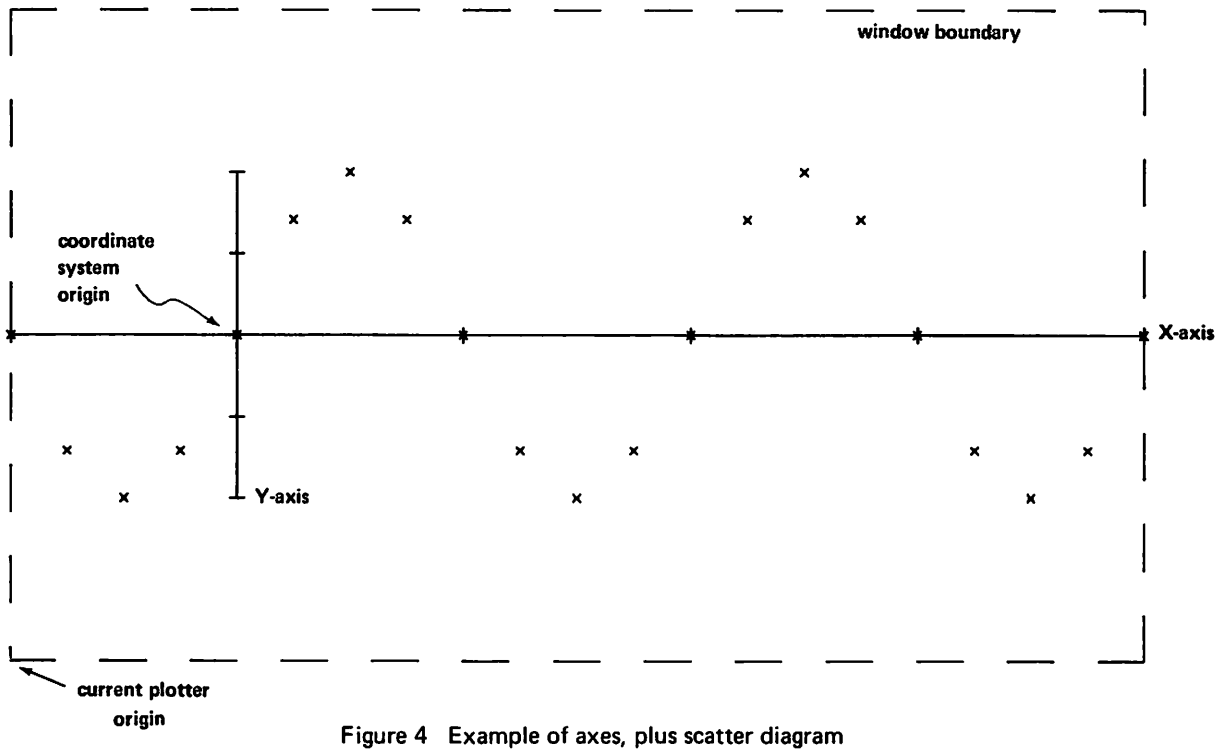


Figure 4 Example of axes, plus scatter diagram

Example

The axes shown in Figure 4 could be drawn in the viewing window, at the positions shown, by the following sequence of statements:

IMP	FORTRAN
SETPLOT(0,0,70,40,M'CMS')	CALL GRAREA(0.D0,0.D0,70.D0,40.D0,'CMS')
SCALE(14,15,14,10,0)	CALL SCALGR(14.D0,15.D0,14.D0,10.D0,0.D0)
AXIS(-1,0,'X',1,5)	CALL AXISGR(-1.D0,0.D0,'X',1.D0,5)
AXIS(0,1,M'-Y',.5,4)	CALL AXISGR(0.D0,1.D0,'-Y',.5D0,4)

The pen will finally be at the point (0,-1) in your coordinate system.

******* POINTSYMBOL / PSYMGR *******

```
IMP      { externalroutinespec POINTSYMBOL(integer ICODE,longreal SIZE)
         {
         { POINTSYMBOL(ICODE,SIZE)

FORTRAN  CALL PSYMGR(ICODE,SIZE)
```

One of 15 special symbols may be added to your drawing at any time. The symbol is drawn using a basic 5-point square grid whose centre coincides with the present pen position. Only that portion of the symbol which lies within the current viewing window will be visible.

The value of ICODE specifies the symbol:

ICODE	SYMBOL	ICODE	SYMBOL
1	×	8	▸
2	+	9	◂
3	□	10	■
4	◇	11	·
5	○	12	-
6	△	13	ı
7	▽	14	/
		15	\

Symbol 5 will be drawn unless 1≤ICODE≤15.

With the exception of symbol 11, which is simply a very small dot, the symbol size, i.e. its width and height, may be varied by the parameter SIZE. Its value is declared in your coordinate system X-units. A default size equivalent to 0.04 inch or 0.1 cm is invoked if your SIZE specification represents less than 4 incremental movements for the current plotter type (see Table 1).

Example

To extend the AXIS/AXISGR example by adding a scatter diagram of special symbols to the axes drawn in Figure 4, each symbol being 4 millimetres wide, the following statements could be added:

```
IMP                                  FORTRAN

SIZE=.4/14                           SIZE=.4D0/14.D0
cycle I=1,1,21                       DO 1 I=1,21
PX=.25*(I-5)                         PX=.25D0*(I-5)
PY=SIN(PI*PX)                        PY=DSIN(3.14159265*PX)
PLOT(1,PX,PY,0,0)                    CALL PLOTGR(1,PX,PY,0.D0,0.D0)
POINTSYMBOL(1,SIZE)                  CALL PSYMGR(1,SIZE)
X(I)=PX                              X(I)=PX
Y(I)=PY                              Y(I)=PY
repeat                             1 CONTINUE
```

******* LINEGRAPH / LINESG *******

```
IMP      { externalroutinespec LINEGRAPH(longrealarrayname X,Y,integer M,N,c
         {                     longreal DASH,GAP,integer ICODE,longreal SIZE)
         {
         { LINEGRAPH(X,Y,M,N,DASH,GAP,ICODE,SIZE)

FORTRAN  CALL LINESG(X,Y,M,N,DASH,GAP,ICODE,SIZE)
```

This routine is provided so that you may join a series of points using a particular format of connecting line, and optionally add a special symbol at each point. The routine combines the actions of PLOT/PLOTGR and POINTSYMBOL/PSYMGR.

The series of points to be used is contained in the arrays X and Y, with point 1 in (Xm,Ym) and the end point in (Xn,Yn). m may be either less than or greater than n, whichever you wish. The direction of movement of the pen from one point to the next is unrestricted.

The pen is repositioned initially at the point (Xm,Ym), and then the remaining |m-n| points are joined in order, using straight line connections, in a format determined by the values of DASH and GAP as follows:

* DASH=0 causes moves with the pen raised (this usually means you wish only to draw a scatter diagram of the data points; see the example below, and compare with the POINTSYMBOL/PSYMGR example)

* DASH>0 and GAP=0 draws solid line connections

* DASH>0 and GAP>0 draws dashed line connections using DASH X-units as the dash length and GAP X-units as the gap length between dashes

Unless ICODE=0 a special symbol is added at each data point as it is passed. ICODE and SIZE have the same meanings as in the routine POINTSYMBOL/PSYMGR.

Only that portion of the drawing lying within the viewing window boundaries will be visible. Subject to this constraint the pen is left at the point (Xn,Yn) on exit from the routine. The pen status on exit is dependent upon the line format chosen and whether symbols were drawn.

Example

The scatter diagram of points in Figure 4 could equally well have been produced by the following single statement, assuming that the X and Y arrays had been evaluated previously.

IMP	FORTRAN
LINEGRAPH(X,Y,1,21,0,0,1,SIZE)	CALL LINESG(X,Y,1,21,0.D0,0.D0,1,SIZE)

******* CURVE / CURVGR ******

```
IMP      { externalroutinespec CURVE(longrealarrayname X,Y,integer M,N,c
                 longreal XON,XOFF,DX,DY,DASH,GAP,integer ICODE,longreal SIZE)

         { CURVE(X,Y,M,N,XON,XOFF,DX,DY,DASH,GAP,ICODE,SIZE)

FORTRAN  CALL CURVGR(X,Y,M,N,XON,XOFF,DX,DY,DASH,GAP,ICODE,SIZE)
```

This routine is similar to LINEGRAPH/LINESG except that a smooth curve is drawn through the series of data points. The method used is to take 4 points at a time, evaluate a third degree polynomial through these points, draw along this curve between points 2 and 3, then drop point 1, introduce the next data point and use this new set of 4 points to extend the curve to the new point 3, etc.

The series of data points to be used is in the arrays X and Y, with point 1 in (Xm,Ym) and the end point in (Xn,Yn). m may be either less than or greater than n, but as the curve is a single-valued function no two X-values can be the same. In fact two stringent conditions must be satisfied:

* there must be at least 4 data points, i.e. $|m-n| \geq 3$

* $Xm < Xm \pm 1 < Xm \pm 2 < \ldots\ldots\ldots\ldots < Xn$

Otherwise plotter faults 13 or 7, respectively, will be registered with the appropriate message; either

```
***(CHANNEL nn) Call no. mm to 'CURVE
                               CURVGR':-
***Too few data points.***
```

or

```
***(CHANNEL nn) Call no. mm to 'CURVE
                               CURVGR':-
***Independent variable data out of order.***
```

Although all of the data points are used in the curve-fitting calculations, you may wish to limit the X-range within which the fitted curve is actually visible. XON declares the X-value at the start of this range and XOFF the end X-value. Obviously the drawing range and the data points must be compatible; i.e. at least some of the data points must lie within the range XON

to XOFF. If any one of the three conditions

* XON < XOFF
* X_m < XOFF
* X_n > XON

is untrue then plotter fault 6 is registered, with the message

***(CHANNEL nn) Call no. mm to 'CURVE / CURVGR':-

Data and drawing range incompatible.

If any of these three plotter faults arises the curve will not be drawn, and the program will terminate unless it is in parameter checking mode (see OPENPLOTTER/OPENGR) or you use the fault trapping procedure (see PLOTFAULT/IGRERR).

If all of the above conditions are satisfied the pen is initially repositioned at the point (Xm,Ym) and the curve is then evaluated step by step and drawn within the stated X-range. The smoothness of the curve is governed by the values of DX and DY. DX X-units is the standard evaluation interval used in stepping the fitted curve, unless this produces a Y-deflection greater than DY Y-units, when Y-intervals of DY are used until the current DX X-interval is complete.

Unless ICODE=0 a special symbol will be drawn at every data point as it is passed (not only those data points lying within the visible curve's X-range). The values of ICODE and SIZE have exactly the same meaning as in POINTSYMBOL/PSYMGR.

The curve format is governed by the values of DASH and GAP:

* DASH=0 causes moves with the pen raised, i.e. an invisible curve (!)
* DASH>0 and GAP=0 draws a solid curved line
* DASH>0 and GAP>0 draws a dashed curve using DASH X-units as the dash length and GAP X-units as the gap between dashes

Again only that portion of the curve and symbols which lies within the viewing window boundaries will be visible. Subject to this constraint the pen is left at the point (Xn,Yn) on exit from the routine. The pen status on exit is dependent on the curve format chosen and whether symbols were drawn.

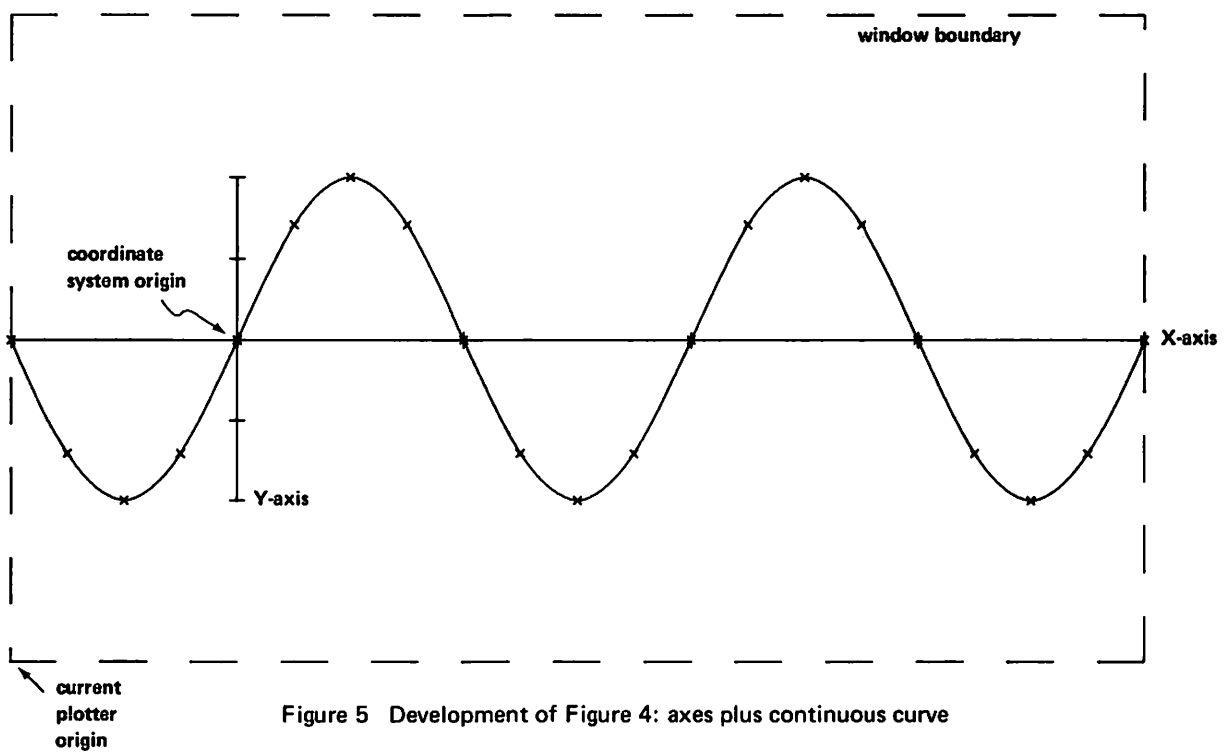


Figure 5 Development of Figure 4: axes plus continuous curve

Example

The curve in Figure 5 has been drawn through the data points used previously to draw the scatter diagram in Figure 4. The necessary call on CURVE/CURVGR is

IMP

```
CURVE(X,Y,1,21,-1,4,.01,.01,1,0,0,0)
```

FORTRAN

```
CALL CURVGR(X,Y,1,21,-1.D0,4.D0,.01D0,
                              .01D0,1.D0,0.D0,0,0.D0)
```

******* ANNOTATE / ANNOGR *******

IMP

```
externalroutinespec ANNOTATE(longreal X,Y,SIZE,THETA)
ANNOTATE(X,Y,SIZE,THETA)
```

FORTRAN

```
CALL ANNOGR(X,Y,SIZE,THETA)
```

Text, numbers, titles, annotation of axes, etc. are all frequent requirements in graphical output. An extensive set of characters is provided for this purpose: see 'Graph Plotting Symbol Sets', Tables 4 and 5.

ANNOTATE/ANNOGR allows you to define the size, orientation and 'start of line' position for strings of characters. The following three routines, viz. PLOTSYMBOL/DRSYMG, PLOTSTRING/DRSTRG, PLOTNUMBER/DRNUMG are provided in order to specify the character strings required to be drawn using the current set of characteristics. These characteristics may be altered any number of times without restriction.

Basically each character, with the exception of the control symbols described later, is contained within a 7x12 point rectangular grid - see Figure 6. The 'base point' of a character is the point (1,3) on this grid and most characters, as shown in Figure 6, lie within the horizontal grid points 1-5 and the vertical grid points 3-10, although lower case characters with tails may descend to the vertical grid point 0 and 'overline' is drawn along the line at vertical grid point 11. Successive characters constituting a line of text have their 7x12 point grids abutted as shown.

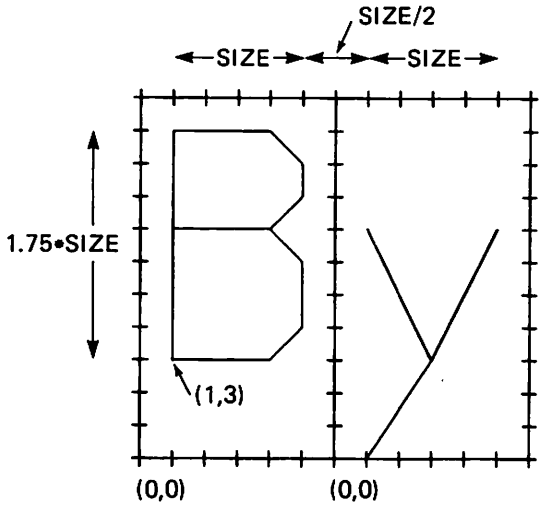
The ANNOTATE/ANNOGR parameters take the following meanings:-

X,Y — is the co-ordinate position of 'start of text line' and (initially) coincides with the 'base point' of the first symbol in a text string. The pen is positioned here by ANNOTATE/ANNOGR. The 'start of text line' position is modified by the 'newline', 'newpage', 'carriage return' symbols described below. Generally, however, the 'base point' of a symbol is the present pen position, i.e. symbols are drawn wherever the pen is at the time.

SIZE — specifies, in your X-units, the width of each character, i.e. the distance between the horizontal points 1-5 of the 7x12 point grid; this automatically ensures a character height = 1.75*SIZE and SIZE/2 spacing between successive characters (the length of a text line is thus 1.5*SIZE*no. of characters).

THETA — specifies the orientation of the line of text in degrees counter-clockwise to your X-axis direction.

N.B. SIZE will be set to the equivalent of 0.04 inch (.1 centimetre) if the character width specified represents fewer than 4 increments for the current plotter type (see Table 1).



The word 'By' with the 7 x 12 point grids used to draw it. The grids of course do not normally appear.
The bottom left-hand corner of the 'B' is at the letter's base point – (1,3) on the grid.

Figure 6 Example of grid used in symbol drawing

The characteristics of the control symbols (see Section 5.5) are:

* 'null' has no effect whatsoever

* 'newline' or 'newpage' causes the 'start of line' position, and hence the next character position, to be moved to the next line, i.e. 3xSIZE X-units below the previous line's start point and in the orientation THETA

* 'carriage return' sets the next character position back to the current 'start of line' position, thus allowing for underlining, overprinting, etc.

* 'space' or 'backspace' set the next character position one symbol position forward or backward respectively

* 'italics on' causes each succeeding character to 'lean' 15 degrees to the right

* 'italics off' resets characters to their normal rectangular shape

* 'subscript mode' can have different effects:

  a) if the previous mode was 'standard' then succeeding characters will be drawn as half-size subscripts
  b) if the previous mode was 'superscript' then succeeding characters return to 'standard' mode and full-size

  Thus two successive 'subscript mode' requests are needed to transfer directly from 'superscript mode' to 'subscript mode'

* 'superscript mode' can have different effects:

  a) if the current mode is 'standard' then succeeding characters will be drawn as half-size superscripts
  b) if the current mode is 'subscript' then succeeding characters return to 'standard' mode and full-size

  Thus two successive 'superscript mode' requests are needed to transfer directly from 'subscript mode' to 'superscript mode'

N.B. A call on ANNOTATE/ANNOGR always returns characters to 'standard' mode and switches to the 'italics off' state.

******* PLOTSYMBOL / DRSYMG *******

```
        ⎧ externalroutinespec PLOTSYMBOL(integer ICODE)
IMP     ⎨
        ⎩ PLOTSYMBOL(ICODE)

FORTRAN  CALL DRSYMG(ICODE)
```

This routine draws a single character at the present pen position, using the current size and orientation characteristics set by ANNOTATE/ANNOGR. Remember to use CHCODE to specify whether characters are ISO or EBCDIC.

The full set of possible characters is described in Tables 4 and 5 - see 'Graph Plotting Symbol Sets' (Section 5.5). However, only those characters which are standard members of either the IMP Extended Symbol Set or are Hollerith symbols, as appropriate, may appear within quotes or in Hollerith format as the value of ICODE. Any other symbol required must be specified by the value of its internal representation. Any undefined symbol value will be replaced by '-'.

******* PLOTSTRING / DRSTRG *******

```
        ⎧ externalroutinespec PLOTSTRING(string (255) ICHARS)
IMP     ⎨
        ⎩ PLOTSTRING(ICHARS)

FORTRAN  CALL DRSTRG(ICHARS,N)
```

This routine draws a succession of characters by repeated application of PLOTSYMBOL/DRSYMG, thus producing a line of text of the current size in the current orientation. Use CHCODE to specify whether characters are ISO or EBCDIC.

In IMP the parameter is a string variable name or a string expression; the string length is implicit.

In FORTRAN the parameter ICHARS is either the name of an A4 format integer or integer array, or a Hollerith string of symbols. The string length N must be explicitly stated.

Normally the string may contain up to 255 characters. However, when this routine is called to specify further job identification information, i.e. between calls on OPENPLOTTER/OPENGR and SETPLOT/GRAREA, the string may only contain up to 30 characters (see 'Identification of plotter output' in para 5.3).

******* PLOTNUMBER / DRNUMG *******

```
        ⎧ externalroutinespec PLOTNUMBER(longreal X, integer M,N)
IMP     ⎨
        ⎩ PLOTNUMBER(X,M,N)

FORTRAN  CALL DRNUMG(X,M,N)
```

This routine converts the floating-point value in X into a written decimal number according to the format specified by the values of the parameters M and N, by repeated application of the routine PLOTSYMBOL/DRSYMG. Use CHCODE to specify whether the characters are ISO or EBCDIC.

There are three format options:

* M>0 and N=0 produces an M-digit integer value without a decimal point
* M>0 and N>0 produces a fixed-point number with M digits preceding the decimal point and N digits following it
* M=0 and N>0 produces a floating-point number with 1 digit preceding the decimal point, N digits following it, and a signed 2-digit exponent occupying 4 character positions

If M has a negative value then M=0 is assumed by default; if N has a negative value then N=7 is assumed by default.

Leading zeros in the integer part of the number appear as spaces, and the sign character precedes the first significant digit (the positive sign character is replaced by a space). If more than M digits are needed to specify the integer part of the number then all the digits are drawn and the number is displaced to the right in consequence.

In IMP the exponent indicator is '@' and in FORTRAN 'E'.

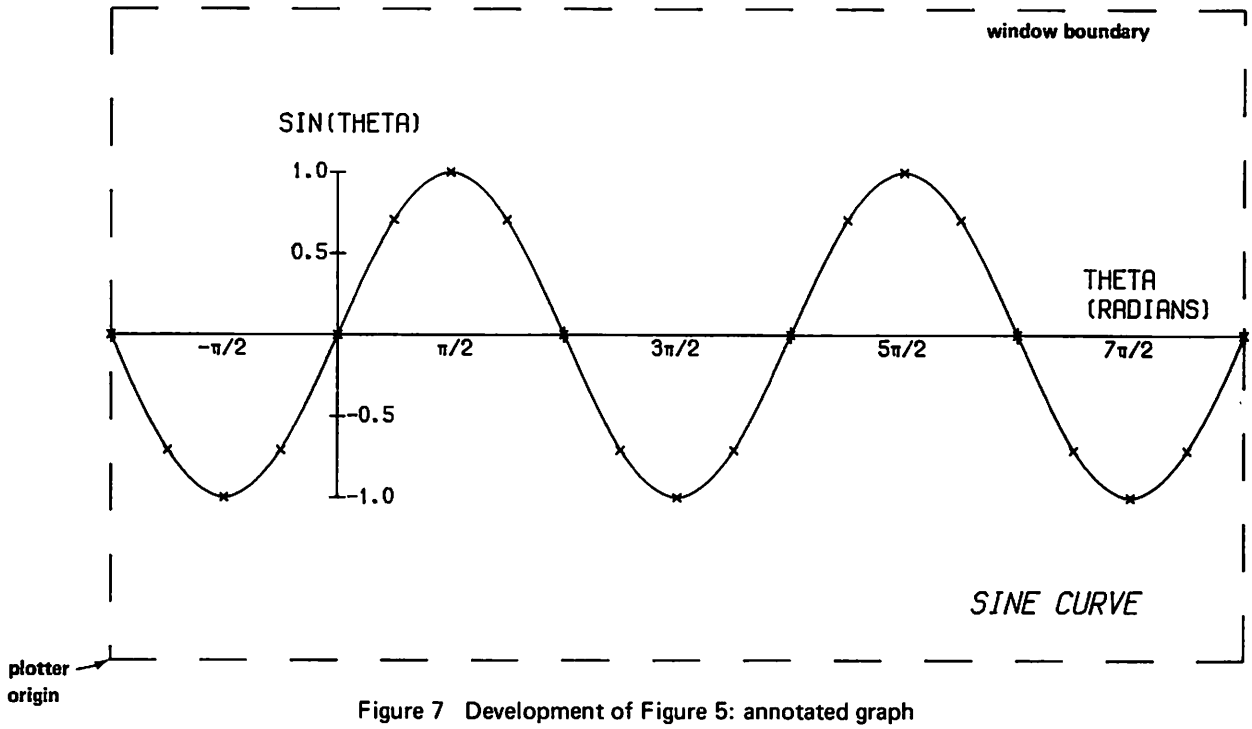


Figure 7 Development of Figure 5: annotated graph

Example

The complete example program being written to produce the drawings in Figures 4 and 5, and now 7, can be rewritten using more efficient pen movements (i.e. minimising movements with the pen raised), as shown in the following IMP and FORTRAN examples.

IMP PROGRAM

```
realslong
begin
externalroutinespec OPENPLOTTER(integer N)
externalroutinespec GRAPHPAPER(real X,integer N)
externalroutinespec PLOTSTRING(string (255)S)
externalroutinespec SETPLOT(real XL,YL,XH,YH,integer N)
externalroutinespec SCALE(real XO,YO,XS,YS,THETA)
externalroutinespec AXIS(real X,Y,integer M,real D,integer N)
externalroutinespec ANNOTATE(real X,Y,SIZE,THETA)
externalroutinespec PLOTNUMBER(real X,integer M,N)
externalroutinespec PLOTSYMBOL(integer N)
externalroutinespec CURVE(realarrayname X,Y,integer M,N,real XN,XF,DX,DY,DASH,GAP,c
                          integer IC,real SIZE)
externalroutinespec CLOSEPLOTTER
integer I; real PX,PY; realarray X,Y(1:21)
ownbyteintegerarray CH(1:5)='-',' ','3','5','7'
OPENPLOTTER(50)                          ;! plotter file on channel 50
GRAPHPAPER(40,M'INS')                    ;! lower setting than the default
PLOTSTRING('ERCC')                       ;! extra job i.d. information
SETPLOT(0,0,70,40,M'CMS')                ;! window size (70-0) cm. by (40-0) cm.
SCALE(14,20,14,10,0)                     ;! 14cm/X-unit, 10cm/Y-unit
AXIS(0,-1,'Y',0.5,4)                     ;! Y-axis drawn upwards
ANNOTATE(-0.25,1.25,0.015,0)             ;! 0.21cm wide characters horizontal
                                          ! (14 x 0.015cm)
PLOTSTRING('SIN(THETA)')                 ;! Y-axis title
PY=1                                     ;! Y-range from 1 to -1
cycle I=1,1,5
if I<=3 then PX=-0.07 else PX=0.01
ANNOTATE(PX,PY-0.012,0.01,0)             ;! 0.14cm wide characters horizontal
PLOTNUMBER(PY,1,1) unless I=3            ;! numbering the Y-axis, except zero
PY=PY-0.5
repeat
PX=-0.5                                  ;! X-range from -PI to +4PI
cycle I=1,1,5
ANNOTATE(PX-3*0.01,-0.05,0.01,0)         ;! 0.14cm wide characters horizontal
PLOTSYMBOL(CH(I))                        ;! numbering the X-axis
PLOTSYMBOL(176)                          ;! at intervals of PI
PLOTSTRING('/2')                         ;! from -PI/2 to +7PI/2
PX=PX+1
repeat
ANNOTATE(3.5,0.2,0.015,0)
PLOTSTRING('THETA
(RADIANS)')                              ;! X-axis title
AXIS(4,0,M'-X',1,5)                      ;! X-axis drawn backwards
cycle I=1,1,21                           ;! 21 data points
X(I)=0.25*(I-5)                          ;! (X,Y) coordinate pairs
Y(I)=SIN(PI*X(I))
repeat
CURVE(X,Y,1,21,-1,4,0.008,0.01,1,0,1,0.01) ;! solid curve plus special symbols with
                                            ! evaluation intervals 0.112cm (X) and
                                            ! 0.1cm (Y) if needed.
ANNOTATE(2.7,-1.4,0.02,0)                ;! drawing title position
PLOTSYMBOL(224)                          ;! set italic mode
PLOTSTRING('SINE CURVE')                 ;! draw title
CLOSEPLOTTER                             ;! file complete
endofprogram
```

```
      INTEGER*4 PI,NLINE,CH(5)
      REAL*8 PX,PY,X(21),Y(21)
      DATA CH/'-',' ','3','5','7'/,NLINE/10/,PI/176/
C  IF NOT FORTE COMMENT OUT THE PREVIOUS LINE AND REPLACE IT BY
C     DATA CH/'-',' ','3','5','7'/,NLINE/21/,PI/35/
C     CALL CHCODE(1)
C  OPEN PLOTTER FILE, SET PAPER LIMIT, AND ADD EXTRA JOB I.D. INFORMATION
      CALL OPENGR(50)
      CALL GRPAPR(40.D0,'INS')
      CALL DRSTRG('ERCC',4)
C  DECLARE REQUIRED WINDOW AND COORDINATE SYSTEM WITHIN IT
      CALL GRAREA(0.D0,0.D0,70.D0,40.D0,'CMS')
      CALL SCALGR(14.D0,20.D0,14.D0,10.D0,0.D0)
C  DRAW Y-AXIS, ADD TITLE, AND NUMBER IT
      CALL AXISGR(0.D0,-1.D0,'Y',0.5D0,4)
      CALL ANNOGR(-0.25D0,1.25D0,0.015D0,0.D0)
      CALL DRSTRG('SIN(THETA)',10)
      PY=1.D0
      DO 1 I=1,5
      PX=-0.07D0
      IF(I .GT. 3) PX=0.01D0
      CALL ANNOGR(PX,PY-0.012D0,0.01D0,0.D0)
      IF(I .NE. 3) CALL DRNUMG(PY,1,1)
    1 PY=PY-0.5D0
C  NUMBER THE X-AXIS, ADD ITS TITLE, DRAW IT BACKWARDS
      PX=-0.5D0
      DO 2 I=1,5
      CALL ANNOGR(PX-3.*0.01D0,-0.05D0,0.01D0,0.D0)
      CALL DRSYMG(CH(I))
      CALL DRSYMG(PI)
      CALL DRSTRG('/2',2)
    2 PX=PX+1.D0
      CALL ANNOGR(3.5D0,0.2D0,0.015D0,0.D0)
      CALL DRSTRG('THETA',5)
      CALL DRSYMG(NLINE)
      CALL DRSTRG('(RADIANS)',9)
      CALL AXISGR(4.D0,0.D0,'-X',1.D0,5)
C  CALCULATE 21 DATA POINT CO-ORDINATE PAIRS AND DRAW SOLID CURVE WITH SYMBOLS
      DO 3 I=1,21
      X(I)=0.25D0*(I-5)
    3 Y(I)=DSIN(3.14159265*X(I))
      CALL CURVGR(X,Y,1,21,-1.D0,4.D0,0.008D0,0.01D0,1.D0,0.D0,1,0.01D0)
C  ADD DRAWING TITLE USING ITALIC CHARACTERS
      CALL ANNOGR(2.7D0,-1.4D0,0.02D0,0.D0)
      CALL DRSYMG(224)
      CALL DRSTRG('SINE CURVE',10)
C  PLOTTER FILE COMPLETE
      CALL CLOSGR
      STOP
      END
```

5.4 CALCOMP BASIC GRAPHIC SOFTWARE SIMULATION PACKAGE

Notation

The description of each routine is headed by the FORTRAN call statement for that routine; this package is not available from IMP programs.

The standard FORTRAN variable naming convention is used throughout, i.e. integer variables begin with one of the letters I, J, K, L, M, N, and all others are real.

Precision

For consistency with the original Calcomp software all variables passed as parameters to these routines are in single precision. Real variables, names or constants, must be REAL*4, and all integer variables must be INTEGER*4.

Routine Name	Purpose	Page
PLTYPE	Sets the plotter code number.	5-29
CHCODE	Sets the plotter program character set type.	5-29
PLOTS	Initialises the Calcomp Simulation Package prior to plotter file generation, and enables delivery information etc. to be specified.	5-30
IGRREC	Indicates how many records have been written to the current plotter file to date.	5-32
NEWPEN	Causes the plotter pen to be changed.	5-32
OFFSET	Enables a secondary Cartesian coordinate system to be specified.	5-33
PLOT	Draws a straight line between two specified points. The line may be visible or invisible. The routine can also be used to reposition the standard or secondary origin, and to close the file.	5-34
FACTOR	Enables a scaling factor to be specified. This factor is applied equally to subsequent X- and Y-coordinate values.	5-36
GRARFL	Enables 'out of area plotting' messages to be suppressed.	5-36
WHERE	Returns the current pen position coordinates.	5-36
SYMBOL	Draws a specified character or characters from a standard set, or from a special set.	5-37
NUMBER	Outputs a decimal number in a specified format.	5-39
SCALE	Enables a given set of data values to be scaled to fit into a specified length when drawn.	5-40
AXIS	Draws a graduated line, numbered at one-inch intervals, from a specified point in a specified direction. A title can also be drawn.	5-41
LINE	Draws a piecewise straight line through a given array of points and adds a special symbol at each point. The line or the symbols can be omitted.	5-42

Table 3 Calcomp Simulation Package Routines

Identification of plotter output

Your plotter output is automatically identified with your jobname, the machine on which you are running the job, the date and time of file generation, and the file name (if accessible). You are, however, strongly recommended to add your name and delivery point, using a maximum of 30 characters, in order to minimise delays in returning your output. Simply program, for example:

```
CALL PLOTS('A.N.OTHER, DEPT. X',18,ICHAN)
```

when initialising your plotter files. PLOTS is decribed below. If there is no delivery information available then

"**** Please set delivery ****"

will be drawn automatically.

Error messages

In certain circumstances the value of a parameter passed to one of the routines may not be intelligible to that routine. When this occurs an appropriate message will be sent to a character file with ddname STREAM99 (see 'Job Control Requirements', Section 5.6), and the program will be terminated unless it is in parameter checking mode (see PLOTS below).

5.4.1 ADMINISTRATIVE AND DRAWING ROUTINES

***** PLTYPE *****

```
CALL PLTYPE(NTYPE)
```

This routine may be called at any time to indicate that future calls on graph plotting routines should be with reference to the characteristics of plotter NTYPE, where 1<=NTYPE<=6 (see Table 1). The routine call is ignored if the current plotter type is already NTYPE.

The effect of this routine is to close the currently open plotter file, if one is open, and to establish the characteristics of the required plotter type in the current plotter type information area.

If this routine call is omitted the characteristics of the Calcomp 936 plotter (NTYPE=1) will be assumed by default. If you wish to reference a different plotter you should precede the call on PLOTS, described below, by a call on this routine.

Of course it is only sensible to send a plotter file to a plotter which has characteristics compatible with that referenced by the file, i.e. the same increment size and dimensions large enough to accommodate the largest viewing window you declare.

Example

```
CALL PLTYPE(2)
```

would cause future reference to a Calcomp Model 663 plotter with an increment size of .0025 inch and maximum plotting width 28.5 inches.

***** CHCODE *****

```
CALL CHCODE(N)
```

The graph plotting software needs to know the character code used by your program. This is dependent upon the programming language - FORTE uses ISO, FORTRANG, ICL FORTRAN and IBM FORTRAN use EBCDIC.

The parameter N indicates ISO when even and EBCDIC when odd. ISO is assumed by default unless IBM FORTRAN is being used when EBCDIC is the default.

The call on this routine should precede calls on any other plotting routine except PLTYPE above.

******* PLOTS *******

```
CALL PLOTS(IDENT,N,ICHAN)
```

If you wish to create a plotter file you must call this routine before any other plotting routine, with the exception of PLTYPE and CHCODE above.

The effect of this routine call is to close the currently open plotter file, if one is open, and to reset the initialisation parameters for the current plotter type (see Table 1). Consequently, although it is permissible to create several plotter files within a single run of your program, only one of them can be open at a time.

The value of the parameter ICHAN specifies the output channel number to be related to this file. The value may be 96, the system plotter file, or may lie in the range 1<=ICHAN<=80. See also 'Job Control Requirements', Section 5.6. If the channel has not been defined or is defined improperly then the program will terminate with one of the following messages:-

```
***(CHANNEL nn) Call no. to 'PLOTS':-
***          message              .***
```

where message = No file definition statement
or Plotter file not defined as SQFILE
or Plotter file not F80, i.e. card images

Successive calls on PLOTS may specify the same value for ICHAN but if this points to the same file name or device the program will terminate with the message

```
***(CHANNEL nn) Call no. mm to 'PLOTS':-
***File extension not allowed.***
```

If any plotter file is re-used in this run of your program then the new drawings will simply overwrite the previous drawings it contained and they will be lost.

On creation of this file a job identification 'window' is declared, 2 inches wide and the full width of the plotter, and your jobname, the machine on which you are running the program, the date, file run-time and file name (if accessible) are drawn across the width of the plotter. Further identification, such as your name and delivery point, may be added by use of the parameters IDENT and N.

IDENT is either a Hollerith string containing N characters, or the name of an INTEGER*4 array containing N characters left-justified in A4 format. N is limited to 30.

Following completion of the job identification window the pen is moved on 2 inches. A large viewing window is then automatically declared within which all of the drawings you create in this file will be imprisoned. The window size is the full width of the plotter (i.e. the Y direction - see Table 1), and either

* 120 inches (300 centimetres) along the paper (the X-direction), or
* 32 inches (80 centimetres) along the paper - X-direction - if you have a student jobname.

The following initial conditions are then established:

* the standard black biro pen cartridge is selected
* a 'logical origin' is declared to be 1 inch horizontally and vertically from the 'plotter origin' and the pen is notionally positioned at this point (see para 5.1 Efficiency for the definition of a 'notional' position)
* the program X and Y directions are declared to be the same as those of the window, and the program units of measurement in both these directions are declared to be inches; this constitutes the 'standard coordinate system' recognised by the software, with the current logical origin as its origin

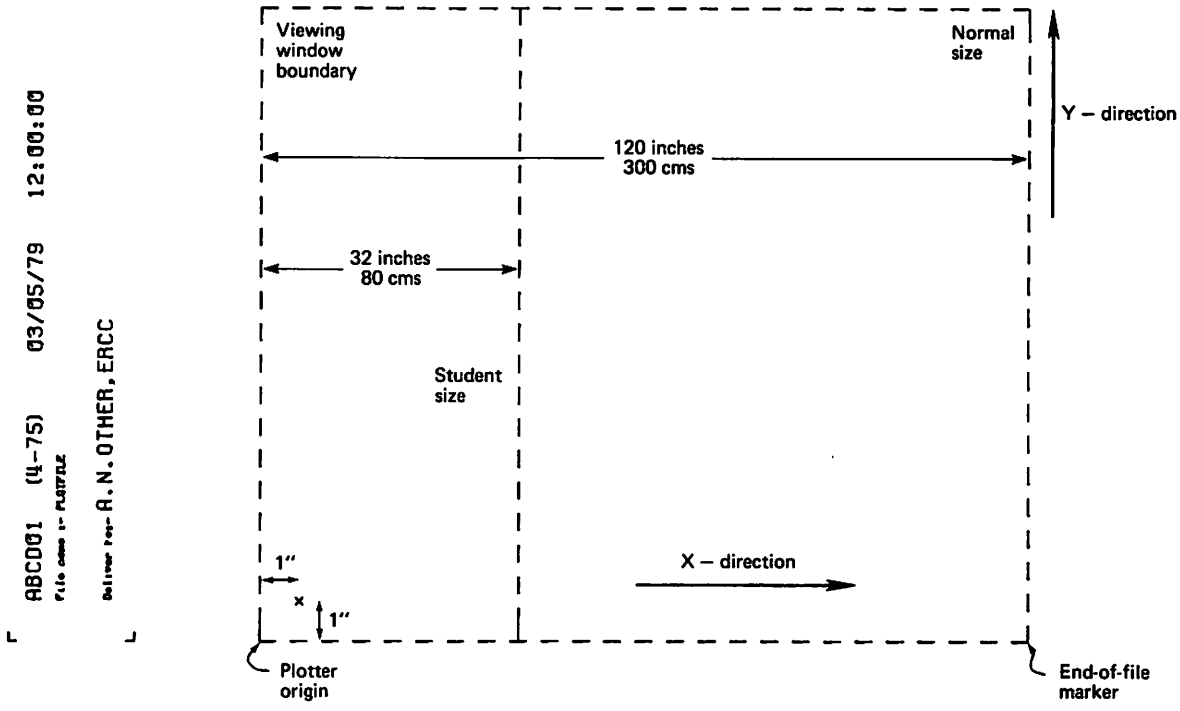


Figure 8 Calcomp package: plotter output layout

N.B. When compared with the original Calcomp software specification, as described in the Calcomp booklet 'Programming Calcomp Pen Plotters', the use of the parameters IDENT and N is non-standard. It is possible in this package, however, since the plotter output buffer which these parameters normally define is totally internal to the Simulation Package software; it is definitely not available to the user program.

The window feature is also non-standard, but provides a fail-safe method for ensuring that pen movements stay within bounds. This therefore precludes use of the plotter micro-switches as a device for pen registration and origin setting as described in the Calcomp booklet. You must keep track of the pen's position yourself; see also the routines PLOT and WHERE, described later.

Example

```
CALL PLOTS('A.N. OTHER, DEPT. X',19,50)
```

will create a plotter file on channel 50, add the 19 characters of delivery information specified to the job identification window, and then initialise the drawing area.

N.B. If the call on PLOTS is omitted the package is said to be in 'parameter checking' mode. No output file is produced but all routine parameters will be checked for validity and appropriate error messages produced.

******* IGRREC *******

```
I = IGRREC(N)          where N is a dummy parameter
```

This function indicates how many records have been written to date to the current plotter file. The function may be called at any time; a zero result is returned if no plotter file is open at the time of the call.

******* NEWPEN *******

```
CALL NEWPEN(IPEN)
```

You may request a different pen from the standard black biro at any time after the call on PLOTS. The operation can be costly in terms of pen movement, however. If this is a single-pen plotter or the requested pen is not already mounted, the pen change is effected at the 'plotter origin' so that the new pen's registration can be checked. (Only if the new pen is immediately available is the pen change done in situ.) After the pen change the pen is notionally re-positioned at its position when the pen change request was encountered.

The value of IPEN determines the pen type required as follows:-

1	black biro (initially and by default)	
2	blue biro	
3	green biro	
4	red biro	
5	black liquid ink, .1mm diameter pen	
6	black liquid ink, .2mm diameter pen	At ERCC
7	black liquid ink, .3mm diameter pen	use of these pens must
8	black liquid ink, .4mm diameter pen	first be authorised by
9	black liquid ink, .5mm diameter pen	Mr M.P. Baillie
10	black liquid ink, .6mm diameter pen	(031 667-1081 ext. 2902)
11	black liquid ink, .8mm diameter pen	

Any other value of IPEN will cause the standard black biro to be selected. If the pen requested is already selected this routine call has no effect.

If you have a student jobname any pen change requests will be ignored.

N.B. Since the Calcomp Model 936 plotter has a three-pen turret (see Table 1), when using that plotter you can eliminate the time taken to change the pen cartridge if you use only pen codes 1, 2 and 4, i.e. black, blue and red biros respectively.

Example

```
CALL NEWPEN(3)
```

will select the green biro for drawing until further notice.

It is good practice to draw all you require with one pen colour before requesting a change of pen because of the time it takes, and the effort involved if the current plotter type has only a single-pen cartridge. At present there is no limit on the number of pen changes during a run of your program, but this may be reviewed if the good practice just described is not adhered to.

***** OFFSET *****

CALL OFFSET(XOFF,XS,YOFF,YS)

This routine seems to be common to some versions of the Calcomp software. It is provided to allow you to refer to a different Cartesian coordinate system from the standard one described in PLOTS.

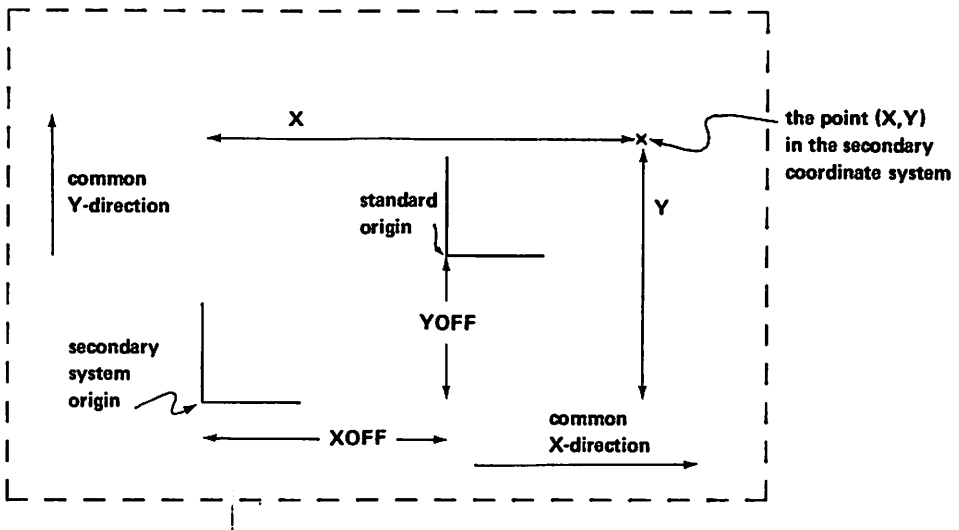


Figure 9 Calcomp secondary coordinate system

Figure 9 shows the meaning attached to XOFF and YOFF. The X- and Y-directions are the same for both coordinate systems. The transformation from the secondary system to the standard one may be simply stated as

$$X_{standard} = (X_{secondary} - XOFF)/XS \quad \text{and} \quad Y_{standard} = (Y_{secondary} - YOFF)/YS$$

where XS is the number of secondary X-units per inch
YS is the number of secondary Y-units per inch

Thus the standard coordinate system's origin is at the point (XOFF,YOFF) in the secondary coordinate system.

This routine simply sets the values of XOFF, XS, YOFF and YS in an information area in the software. You can only refer to the secondary system by using the routine PLOT, described below; none of the other drawing routines can access the offset information.

When a plotter file is initialised the two coordinate systems are made coincident, i.e. XOFF=YOFF=0 and XS=YS=1.

If you attempt to set either XS<0 or YS<0 the program will terminate, unless it is in parameter checking mode (see PLOTS above), with the message

```
***(CHANNEL nn) Call no. mm to 'OFFSET':-
***Negative or zero scaling factor(s).***
```

Example

If your program X-units are seconds, you wish to represent 1 minute by one inch on your drawing, 10 events are to be represented by 1 inch in the Y direction, and the current standard origin is to be coincident with your graph origin, then the statement

```
CALL OFFSET(0.,60.,0.,10.)
```

would automatically cause your program units to be transformed to the standard system units, but only via calls on the routine PLOT.

******* PLOT *******

CALL PLOT(X,Y,IPEN)

This is the basic drawing routine, called at some point by all of the other drawing routines. The pen is moved in a straight line from its current position to the point specified, subject to a check that this move does not contravene the viewing window boundaries.

Normally the position of the point (X,Y) is expressed in the standard coordinate system units, i.e. as inches from the current standard origin position. In this case the possible values of IPEN are:

+2	move to (X,Y) with the pen in contact with the paper, i.e. draw the line
+3	move to (X,Y) with the pen raised, i.e. re-position the pen
999	move the pen clear of all previous drawing and close the file
-2 or -3	as for +2, +3 above except that additionally the point (X,Y) is now the standard origin position from which all future standard coordinate positions will be measured

If you wish to use the secondary coordinate system defined by the parameters passed to the routine OFFSET described above then the point (X,Y) is expressed in terms of this system and the values IPEN may then take are:

12 or 13	as for +2, +3 above
999	as for 999 above
-12 or -13	as for +12, +13 and additionally reset the secondary coordinate system origin at the point (X,Y)

The values +12, +13, -12, -13 are the indication that the secondary system is to be referenced. This reference is available only via the PLOT routine.

N.B. In each of the cases when the origin is reset the 'other' origin is also automatically reset such that the offset between them remains as (XOFF,YOFF).

Any value of IPEN other than those specified will be interpreted as +3.

The software will always use and remember the given values of the PLOT routine parameters when computing the destination coordinates, but a check is continuously made to ensure that only that part of the drawing which lies within the viewing window is actually drawn. Having computed the destination position in terms of the standard coordinate system, a scaling factor (see the routine FACTOR described below) is then applied in order to ascertain the corresponding position on the plotter. If this position lies outside the viewing window the pen will be notionally positioned at this point so far as the user's program is concerned, but the pen movement itself will have been stopped at the window boundary. The first time this occurs, and every 50th time thereafter, the purely informative message

```
***(CHANNEL nn) Call no. mm to 'name' is drawing out of area.***
```

appears, where 'name' is the actual routine which requested this pen movement. After 20 such messages from one plotter file the program is terminated, with the additional message

```
***(CHANNEL nn) Call no. mm to 'name':-
***Excessive out-of-area drawing.***
```

Up to this point the software continuously checks vectors, and 'scissors' them where necessary.

N.B. The Calcomp manufacturer's software specification recommends use of the plotter micro-switches for pen registration, at file initialisation and possibly at other times also, by using the sequence

```
CALL PLOT(0.,-28.5,-3)
CALL PLOT(0.,1.,-3)
```

The first statement would operate the right-hand micro-switch; the second would bring the pen 1 inch away from the plotter edge, this position then being the logical origin for future drawing.

In the ERCC graph plotting environment, with the concept of a 'viewing window', this sequence would have the effect of setting the logical origin immediately after file initialisation at (1-28.5+1), i.e. 26.5 inches off the plotter - and very little of the drawing would then be visible. The ERCC software sets the initial position for you (see the routine PLOTS) but you must keep further origin changes under control.

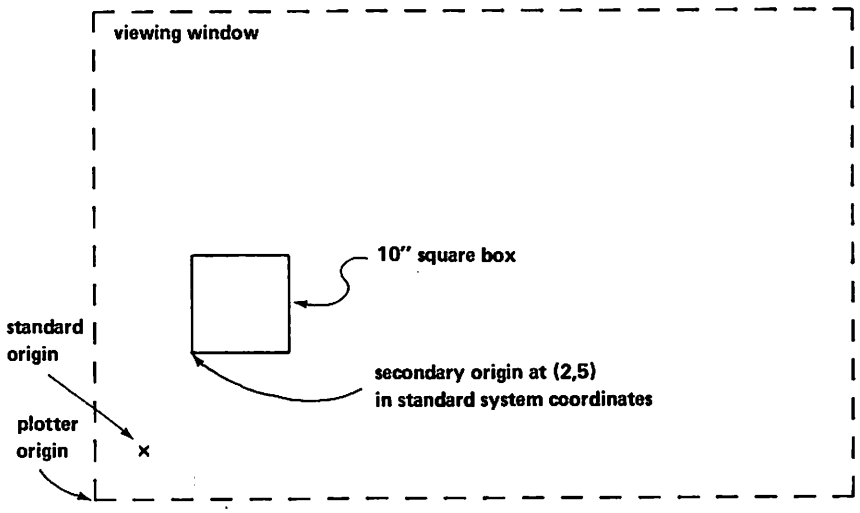


Figure 10 Example of the use of routine PLOT

Example

The box in Figure 10 could be drawn by the statements

```
CALL PLOT(2.,5.,3)
CALL PLOT(12.,5.,2)
CALL PLOT(12.,15.,2)
CALL PLOT(2.,15.,2)
CALL PLOT(2.,5.,2)
```

or, using a secondary coordinate system with the same unit of measurement but with origin at the lower left-hand corner of the box, by the statements

```
CALL OFFSET(-2.,1.,-5.,1.)
CALL PLOT(0.,0.,13)
CALL PLOT(10.,0.,12)
CALL PLOT(10.,10.,12)
CALL PLOT(0.,10.,12)
CALL PLOT(0.,0.,12)
```

******* FACTOR *******

```
CALL FACTOR(F)
```

You may scale your drawing using this routine. The scaling factor F is applied equally to the X- and Y-coordinate values, to produce magnification or diminution of the complete drawing. F may be reset any number of times.

The initial setting is F=1.0. If F is set greater than 1.0 then part of your drawing might disappear outside the viewing window.

F must be greater than 0.0, otherwise the message

```
***(CHANNEL nn) Call no. mm to 'FACTOR':-
***Negative or zero plot size.***
```

will appear, and the program will be terminated unless it is in parameter checking mode (see PLOTS above).

Example

```
CALL FACTOR(2.)
```

will automatically double the size of your drawing from this point on. With reference to Figure 10, the box would then be 20 inches square with its lower left-hand corner at the point (4.,10.) relative to the standard origin.

******* GRARFL *******

```
CALL GRARFL(IFLAG)
```

This routine is not included in the Calcomp manufacturer's software. Its purpose is to accommodate those who wish to ignore any out-of-area pen movement messages, and hence to use the 'viewing window' mechanism as a simple 'scissoring tool'.

IFLAG may take the following values:

* 'ON' meaning 'permit out-of-area messages' - this is the default value at the start of each plotter file

* 'OFF' meaning 'suppress out-of-area messages'

Any other value will be taken to mean 'ON'.

Example

```
CALL GRARFL('OFF')
CALL GRARFL('ON')
```

******* WHERE *******

```
CALL WHERE(X,Y,F)
```

If you wish to determine the notional pen position at any time, i.e. the position where your program believes the pen to be (and this might be outside the viewing window), then this routine will tell you.

After a call of the routine the parameters have the following values:

* X contains the notional X position in inches relative to the current standard origin

* Y contains the notional Y position in inches relative to the current standard origin

* F contains the present value of the scaling factor - see FACTOR above

***** SYMBOL *****

CALL SYMBOL(X,Y,HEIGHT,IBCD,ANGLE,NCHAR)

This routine allows you to add annotation or special graphic symbols to your drawings. The full complement of symbols is detailed under 'Graph Plotting Symbol Sets' (Section 5.5), with the symbol '-' being the default if an invalid code is requested. A call on CHCODE described above should be made in order to specify whether characters will be ISO or EBCDIC.

NCHAR specifies whether a particular symbol is to be used as a special symbol or as a text character:

* NCHAR<0 means that IBCD is an integer whose value specifies a special symbol code

A special symbol is drawn with the 'base point' (X,Y) as centre (in the standard coordinate system), with an overall height and width of HEIGHT inches, and rotated through ANGLE degrees about (X,Y) counter-clockwise to the standard X-direction. The pen is initially moved in a straight line to (X,Y) from its current position. The value of NCHAR affects this move:

* NCHAR<-1 means that the move draws a line

* NCHAR=-1 means that the move is invisible

When the symbol is complete the pen is again at (X,Y), and in contact with the paper.

The symbols usually specified as 'special' (since they are 'centred' symbols) are

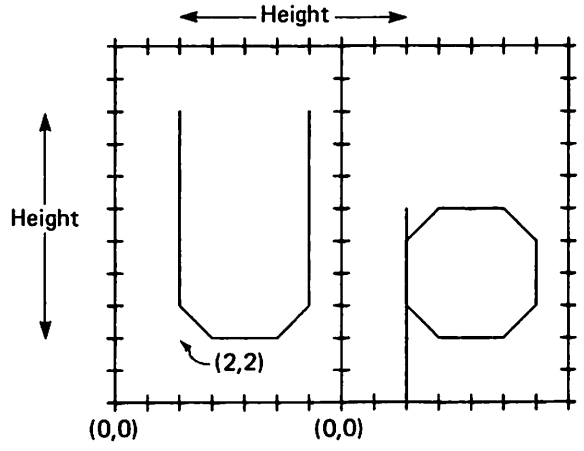
All other symbols are normally used in 'text' mode.

* NCHAR=0 means that IBCD is an integer whose value specifies a text character code

* NCHAR>0 means that IBCD is an integer name, or integer array name, or a Hollerith string, of text characters in A4 format

Text characters are drawn with the 'base point' (X,Y) in the standard coordinate system taken as the lower left-hand corner of the first character in the sequence. The full sequence is drawn along a line at an angle ANGLE degrees counter-clockwise to the standard X-direction.

Each text character, with the exception of the control symbols described below, is contained within an 8x12 point grid, with the base point of the character at the position (2,2) on this grid - see Figure 11. The parameter HEIGHT inches identifies the distance between the vertical positions 2-9, and hence also the full width of the grid. Successive characters have their 8x12 point grids abutted; consequently the line of text appears as it would be written or typed. The length of the line is NCHAR x HEIGHT inches.



The word 'Up' with the 8 x 12 point grids used to draw it.
The grids of course do not normally appear.

Figure 11 Example of grid used in symbol drawing

The characteristics of the control symbols (see Section 5.5) are:

* 'null' has no effect whatsoever

* 'space' and 'backspace' set the next character position HEIGHT inches forward or backward, respectively, along the line of text

* 'newline' sets the next character position HEIGHTx12/7 inches below (X,Y) - allowing for the rotation ANGLE - and re-defines (X,Y) to be this point for future newline reference

* 'italics on' causes each succeeding character to 'lean' 15 degrees to the right

* 'italics off' causes each succeeding character to appear in its normal rectangular shape

* 'subscript mode' may have two different effects:

  a) if the current mode is 'standard' then succeeding characters will be drawn as subscripts, at 0.7 x HEIGHT inches size

  b) if the current mode is 'superscript' then succeeding characters return to 'standard mode' and full size

  Thus two successive 'subscript mode' requests are needed to transfer directly from 'superscript mode' to 'subscript mode'

* 'superscript mode' may have two different effects:

  a) if the current mode is 'standard' then succeeding characters will be drawn as superscripts, at 0.7 x HEIGHT inches size

  b) if the current mode is 'subscript' then succeeding characters return to 'standard mode' and full size

  Thus two successive 'superscript mode' requests are needed to transfer directly from 'subscript mode' to 'superscript mode'

N.B. The SYMBOL routine will always set 'standard mode' and 'italics off' before drawing the symbol sequence; (X,Y) always becomes the new newline reference position.

For both special symbols and text characters the value of HEIGHT will be automatically adjusted to 0.04 inch (0.1 cm) if the HEIGHT value given represents less than 4 increments on the current plotter type (see Table 1) when a special symbol is specified, and less than 7 increments when text characters are specified.

On exit from the SYMBOL routine the pen coordinate position is remembered in order that you may refer to it on the next call of the routine. If you wish to make use of this then the (X,Y) values specified in the next call should be

* X=999.0 if you wish to carry on from the remembered X-position, but select a different Y-position

* Y=999.0 if you wish to carry on from the remembered Y-position, but select a different X-position

* X=999.0 and Y=999.0 if you wish to continue from exactly the same point

This facility can be used when requesting either special symbols or text characters, regardless of which type of symbol was requested in the previous call.

N.B. The Simulation Package SYMBOL routine has been written to be consistent with the original Calcomp specification, which allows all symbols to be used either as special graphics symbols or as text. The consequences of this are that

  * if 'special symbol' mode has just been used, the 'remembered position' is beyond the point (X,Y) on entry to the routine, in the direction ANGLE, by a distance either

    a) HEIGHTx7/4 inches if one of the 'centred' symbols was drawn, or

    b) HEIGHT inches if it was a 'printable' character, or the appropriate adjustment if it was a control character

* if 'special symbol' mode has just used what is normally a text character it will be lower left-hand corner justified, not centre justified, about the point (X,Y)

* if a 'centred' symbol occurs within a 'text' string it will be drawn HEIGHT x 4/7 inches high and centre justified, not lower left-hand corner justified like normal 'text' characters

Example

The drawing of text and connected special symbols in Figure 12 could be produced by the following sequence of statements:

```
      CALL PLOTS('ERCC',4,50)
      CALL SYMBOL(0.,15.,.14,'A POINT PLOT USING CONNECTING LINES',0.,
     X 35)
      DO 1 I=1,10
    1 CALL SYMBOL(X(I),Y(I),.1,IGR,0.,-2)
```

The connecting line from the end of the text string has been drawn because the same SYMBOL call has been used for moving to each of the special symbol drawing positions. IGR will have the value 243 in FORTE and 3 in FORTRANG, IBM FORTRAN and ICL FORTRAN; see Tables 4 and 5 respectively, in 'Graph Plotting Symbol Sets' (Section 5.5).

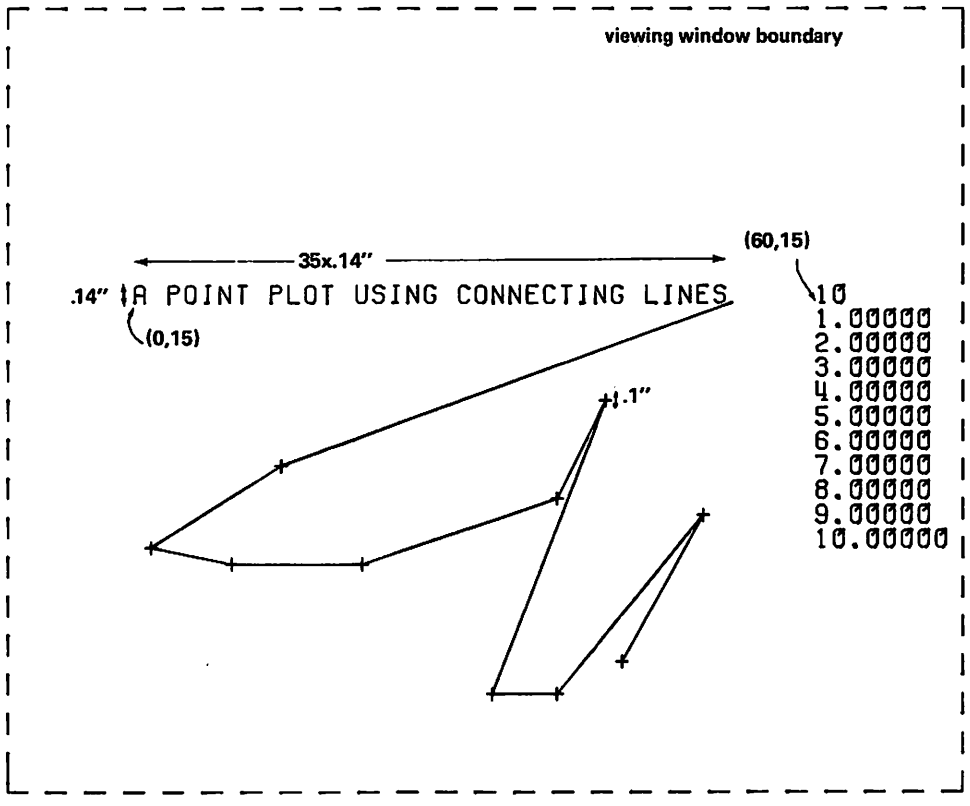


Figure 12 Example of the use of routines PLOTS, SYMBOL and NUMBER

******* NUMBER *******

```
      CALL NUMBER(X,Y,HEIGHT,FPN,ANGLE,NDEC)
```

Drawing numbers is exactly the same as drawing a string of text characters once the required symbols have been derived for the requested number format. The NUMBER routine is simply a pre-processor for the SYMBOL routine described above.

The number drawn is the value assigned to the parameter FPN. The number format is specified by the value assigned to the parameter NDEC as follows:

* NDEC>=0 means the number is to be in F format with NDEC digits following the decimal point

* NDEC=-1 means the number is to be in I format - only the integer part is drawn

* NDEC<-1 means the number is to be in I format, but the integer part will be truncated from the right by |NDEC-1| digits

In each case FPN is rounded properly for the format requested. |NDEC| is limited to a maximum value of 9. If FPN is positive the sign character is omitted.

After initially re-positioning the pen at (X,Y) relative to the standard origin the string of text characters derived for the requested format is drawn using characters HEIGHT inches high, with the line of text at an angle of ANGLE degrees counter-clockwise to the standard X-direction.

As with the SYMBOL routine, the final pen position will be 'remembered' for possible use as the start point of further numbers, text or special graphic symbols. Hence, if in a NUMBER routine call X=999.0 or Y=999.0 or X=Y=999.0 is used, the number will be positioned in the requested position relative to the last position remembered by the SYMBOL routine.

Example

A column of numbers could be drawn by the following statement sequence:

```
      CALL NUMBER(60.,15.,.1,10.,0.,-1)
      DO 1 I=1,10
    1 CALL NUMBER(999.0,15.-I,.08,FLOAT(I),0.,5)
```

as shown in Figure 12.

******* SCALE *******

```
CALL SCALE(ARRAY,AXLEN,NPTS,INC)
```

You may frequently have a set of data values which are required to be drawn to fit some fixed distance on your graph. The SCALE routine takes a set of values and derives suitable start and finish values to fit a specified length of line; it does not produce any pen movement.

The variable ARRAY is either an array name, or the first element of an array, in which the set of data values is stored. The parameter INC specifies the elements within ARRAY which constitute the set of points, i.e.

ARRAY(1), ARRAY(1+|INC|), ARRAY(1+2x|INC|), ... ,ARRAY(1+(NPTS-1)x|INC|)

If INC=0 or NPTS<=0 then the program will be terminated, unless it is in parameter checking mode (see PLOTS), with the message

```
***(CHANNEL nn) Call no. mm to 'SCALE':-
***No data points, or zero increment selection.***
```

The SCALE routine ascertains the minimum and maximum of the values in the set, and hence the units per inch (MAX-MIN)/AXLEN along the line of length AXLEN inches. AXLEN is set to 1.0 if it is given as less than 1.0.

The increment per inch value is often too large for axis annotation purposes, and the increment is therefore adjusted to be of the form

$$n*10^m$$

where n = 1, 2, 4, 5, or 8, whichever fits best. MAX and MIN are also adjusted to be multiples of this number, such that the set of data values lies within the range MIN to MAX.

The starting and finishing points of the line are assigned according to the value of INC, as follows:

* INC>0 means the starting point is the adjusted MIN, the finishing point is the adjusted MAX, and the increment per inch is therefore positive

* INC<0 means the starting point is the adjusted MAX, and the finishing point is the adjusted MIN, and the increment per inch is therefore negative

The calculated starting point and increment per inch values are then stored for future use in the array elements ARRAY(1+NPTSx|INC|) and ARRAY(1+(NPTS+1)x|INC|), respectively, i.e. the 'next' two array positions in the sequence defining the set of points. This means that you must dimension ARRAY large enough to accommodate these elements.

If you have a set of (X,Y) coordinate pairs which you wish to scale to fit fixed-length X- and Y-axes then you must call SCALE twice, once for the X-values and once for the Y-values. The scaling parameters may then be used by the AXIS and LINE routines described later.

Example

The points drawn in Figure 12 could be scaled to fit 10 inch long X- and Y-axes by the following statements:

```
CALL SCALE(X,10.,10,1)
CALL SCALE(Y,10.,10,1)
```

******* AXIS *******

```
CALL AXIS(X,Y,IBCD,NCHAR,AXLEN,ANGLE,START,STEP)
```

This routine will draw a graduated line numbered at one-inch intervals, and add a string of text characters as a title. The numbering and titling information occupy an area approximately 0.5 inch wide alongside the line; this should be allowed for when planning the situation of the line.

(X,Y) defines the starting point of the line in the standard coordinate system; its direction is defined to be at ANGLE degrees counter-clockwise to the standard X-direction. Graduation marks are drawn 0.05 inch in length, perpendicular to the line, but not crossing it.

In general (X,Y) may be any point and ANGLE may take any value but, when this graduated line is to be associated with a line drawing produced by use of the LINE routine (described next), their values are strictly limited to

X=0.0 and ANGLE= 0.0 or ANGLE=180.0 if the line is an X-axis

Y=0.0 and ANGLE=+90.0 or ANGLE=-90.0 if the line is a Y-axis

The length of the line is declared by the value of the parameter AXLEN in inches. If AXLEN<=0 the program will be terminated, unless it is in parameter checking mode - see PLOTS, with the message

```
***(CHANNEL nn) Call no. mm to 'AXIS':-
***Zero or negative length specified.***
```

When the line is numbered the value of START is drawn alongside the tick mark at (X,Y), the value of (START+STEP) one inch further along the line, etc. The values of START and STEP may be those derived previously by calls on the SCALE routine, or specific values you decide to use instead.

The number format used always has two decimal places, and the STEP value may be adjusted locally within the AXIS routine such that $0.01 \leq incr < 100$. If this adjustment is necessary it will be indicated by the routine adding to the axis title

bb*10^n

where b represents a space
n is the adjustment exponent (*10^n is omitted if n=0; n is omitted if n=1)

The axis title is contained in IBCD, as either a Hollerith string or an A4 format integer array, and has |NCHAR| text characters whose type will have been specified by a call on CHCODE. The sign of NCHAR indicates the positioning of the title and line-numbering, and the direction of the tick marks, as follows:

* NCHAR≥0 means the title, numbering and tick marks are all on the counter-clockwise side of the line

* NCHAR<0 means the title, numbering and tick marks are all on the clockwise side of the line

The title, and number adjustment characters if needed, are drawn parallel to the graduated line and centralised along it, using characters 0.14 inches in height, i.e. about 7 characters per inch. Numbering characters are drawn alongside the tick marks using characters 0.105 inches in height, i.e. about 10 per inch with the left-hand edge of the second character in the number immediately above, or below, the tick mark.

The routine exits with the pen back at (X,Y).

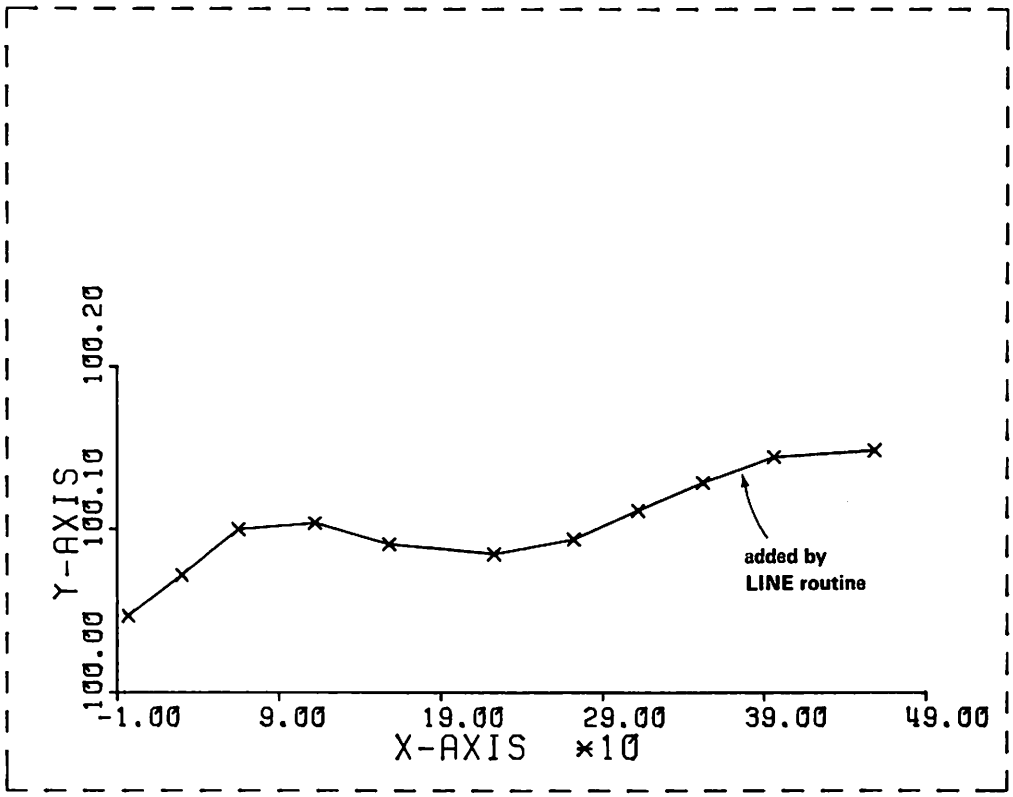


Figure 13 Annotated graph

Example

Figure 13 shows the result of the following two calls on the AXIS routine:

```
CALL AXIS(0.,0.,'X-AXIS',-6,5.,0.,-10.,100)
CALL AXIS(0.,0.,'Y-AXIS',6,2.,90.,YARRAY(23),YARRAY(25))
```

where it is assumed that the array YARRAY has been scaled by the SCALE routine to give an axis start number of 100 and an increment per inch of 0.1 in terms of the program's Y-values, saved in YARRAY(23) and YARRAY(25) respectively. The SCALE call would have been

```
CALL SCALE(YARRAY,2.,11,2)
```

******* LINE *******

```
CALL LINE(X,Y,NPTS,INC,LINTYP,ICODE)
```

You will probably wish at some time during your graph plotting to join a series of data point coordinate pairs using straight line connections, possibly adding a special symbol at some or all of these points. The LINE routine provides this facility, by calling the PLOT and SYMBOL routines, and assumes that the set of points has been processed by the SCALE routine.

The set of points is contained in the two arrays X and Y, with the parameter INC specifying the elements within these arrays which constitute the set; i.e.

(X1,Y1),(X1+|INC|,Y1+|INC|),(X1+2x|INC|,Y1+2x|INC|),.....,(X1+(NPTS-1)x|INC|,Y1+(NPTS-1)x|INC|)

. If INC=0 or NPTS<=0 then the program will be terminated, unless it is in parameter checking mode (see PLOTS), with the message:

```
***(CHANNEL nn) Call no. mm to 'LINE':-
***No data points, or zero increment selection.***
```

The array elements (X1+NPTSx|INC|,Y1+NPTSx|INC|) are assumed to contain the respective X-axis and Y-axis start values.

The array elements (X1+(NPTS+1)x|INC|,Y1+(NPTS+1)x|INC|) are assumed to contain the respective X-axis and Y-axis increments per inch.

These values may be obtained by processing the X and Y arrays in the SCALE routine, or you may supply specific values yourself. Obviously the X and Y arrays must be dimensioned large enough to accommodate these elements.

The LINE routine initially re-positions the pen at the nearer end point of the defined set to the current pen position, then draws connecting lines and/or special symbols through each point of the set in turn. The connecting line format is governed by the value of the parameter LINTYP, as follows:

* LINTYP=0 means only the connecting lines are drawn
* LINTYP>0 means the connecting lines are drawn and the special symbol ICODE is added at every LINTYPth data point in the set
* LINTYP<0 means the connections are invisible and only the special symbol ICODE is drawn at every |LINTYP|th point in the set

The routine exits with the pen at the opposite end of the set of points.

N.B. The LINE routine assumes that the program X- and Y-directions are the same as those of the standard coordinate system, and that the given X-axis and Y-axis starting values coincide with the standard origin.

Example

The line drawing shown in Figure 13 could be drawn by the following statement:

```
CALL LINE(X,Y,11,2,1,ICODE)
```

where the X and Y arrays must have been dimensioned to at least 25 locations, and ICODE has the value 244 in FORTE and 4 in FORTRANG, IBM FORTRAN and ICL FORTRAN; see Section 5.5, Tables 4 and 5 respectively.

The complete example program for creating the drawing shown in Figure 13 is as follows:

```
      DIMENSION X(25),Y(25)
      DATA ICODE/244/
C     IF NOT FORTE COMMENT OUT THE PREVIOUS STATEMENT AND REPLACE IT BY
C       DATA ICODE/4/
C       CALL CHCODE(1)
C     READ DATA VALUES
      READ(5,100) (X(I),Y(I),I=1,21,2)
  100 FORMAT(2F7.2)
C     OPEN PLOTTER FILE ON CHANNEL 50
      CALL PLOTS('ERCC',4,50)
C     SET ORIGIN POSITION
      CALL PLOT(2.,2.,-3)
C     SCALE DATA VALUES TO FIT A 5 INCH X-AXIS AND A 2 INCH Y-AXIS USING
C     EVERY OTHER POINT
      CALL SCALE(X,5.,11,2)
      CALL SCALE(Y,2.,11,2)
C     DRAW X-AXIS AND Y-AXIS
      CALL AXIS(0.,0.,'X-AXIS',-6,5.,0.,X(23),X(25))
      CALL AXIS(0.,0.,'Y-AXIS',-6,2.,90.,Y(23),Y(25))
C     ADD LINE DRAWING
      CALL LINE(X,Y,11,2,1,ICODE)
C     CLOSE PLOTTER FILE
      CALL PLOT(10.,0.,999)
      STOP
      END
```

5.5 GRAPH PLOTTING SYMBOL SETS

IMP and FORTE are both based on the ISO character set. FORTRANG, IBM FORTRAN and ICL FORTRAN on the other hand are based on the EBCDIC character set.

The full complement of graph plotting symbols is available to all types of program, regardless of which graph plotting package is being referenced (ERCC or Calcomp Simulation). However, because of the differences between the two character sets, the internal representation of a graph plotting symbol is dependent upon the language in which the calling program is written.

Table 4 shows the ISO symbol set used in an IMP or FORTE context. Table 5 shows the EBCDIC symbols used in all other contexts. In both cases only those symbols which are accepted members of the standard ISO or EBCDIC character sets may appear in your program within quotes or as a Hollerith string. All others must be referenced by their symbol values.

The style of some characters may differ between the ERCC Graphpack and the Calcomp Simulation Package.

5.5.1 Extended ISO Symbol Set for IMP and FORTE Programs (Table 4)

Notes

* all undefined symbols produce the symbol '-' by default
* the symbols NUL, SP, BS, CR represent 'null', 'space', 'backspace', and 'carriage return without newline', respectively
* the symbols NL and FF are synonymous and represent the 'newline' operation
* the symbol SUB means either 'enter subscript mode from normal mode' or 'return from superscript mode to normal mode'
* the symbol SUP means either 'enter superscript mode from normal mode' or 'return from subscript mode to normal mode'
* the symbol ION means 'switch to italic characters'
* the symbol IOF means 'switch back to standard characters'
* the symbols in column 15 are debarred to ERCC Graphpack routines
* the symbols CR and FF are debarred to Calcomp Simulation Package routines

5.5.2 Extended EBCDIC Symbol Set for FORTRANG, IBM FORTRAN and ICL FORTRAN Programs (Table 5)

Notes

* several symbols are duplicated within this table in order to incorporate standard EBCDIC characters and the original Calcomp defined set, and also to keep the Greek alphabetic characters in a close pattern
* all undefined symbol values will produce the symbol '-' by default
* the symbols NUL, SP, BS, NL represent 'null', 'space', 'backspace', and 'newline' operations respectively
* the symbol SUB means either 'enter subscript mode from normal mode' or 'return from superscript mode to normal mode'
* the symbol SUP means either 'enter superscript mode from normal mode', or 'return from subscript mode to normal mode'
* the symbol ION means 'switch to italic characters'
* the symbol IOF means 'switch back to standard characters'
* the symbols 0 to 13 in column 0 are debarred to ERCC Graphpack routines

TABLE 4.

ISO SYMBOL SET AND SYMBOL VALUES FOR IMP AND EDINBURGH FORTRAN GRAPH PLOTTING PROGRAMS

				b_8	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
				b_7	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
				b_6	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
				b_5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
b_4	b_3	b_2	b_1	COL / ROW	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	NUL	←	SP	0	@	P	`	p		Π		π			ION	▣
0	0	0	1	1	~	∧	!	1	A	Q	a	q	Α	Ρ	α	ρ			IOF	⊙
0	0	1	0	2	▫	∨	"	2	B	R	b	r	Β	Σ	β	σ	ς		↑	Δ
0	0	1	1	3	≠	⊃	#	3	C	S	c	s	Γ	Τ	γ	τ			¬	+
0	1	0	0	4	±	¢	£	4	D	T	d	t	Δ	Υ	δ	υ				×
0	1	0	1	5	÷	×	%	5	E	U	e	u	Ε	Φ	ε	φ				◇
0	1	1	0	6	<	⌟	&	6	F	V	f	v	Ζ	Χ	ζ	χ				♠
0	1	1	1	7	>	ı	'	7	G	W	g	w	Η	Ψ	η	ψ				♯
1	0	0	0	8	BS	_	(	8	H	X	h	x	Θ	Ω	θ	ω				♥
1	0	0	1	9	→	*	)	9	I	Y	i	y	Ι		ι					♀
1	0	1	0	10	NL	‡	*	,	J	Z	j	z	Κ	·	κ					♂
1	0	1	1	11	↓	‡	+	;	K	[	k	{	Λ		λ					♃
1	1	0	0	12	FF	¯	,	<	L	\	l	|	Μ	,	μ					♄
1	1	0	1	13	CR	—	-	=	M	]	m	}	Ν		ν					♮
1	1	1	0	14	SUP	∫	.	>	N	^	n	~	Ξ	.	ξ					
1	1	1	1	15	SUB	∞	/	?	O	_	o	$	Ο	;	ο					

SYMBOL VALUE = 16*COL + ROW

TABLE 5.

EBCDIC SYMBOL SET AND SYMBOL VALUES FOR FORTRANG, ICL & IBM FORTRAN GRAPH PLOTTING PROGRAMS

				b_8	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
				b_7	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
				b_6	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
				b_5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
b_4	b_3	b_2	b_1	COL / ROW	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	□	ı	}	Σ	SP	&	–	0	€	•	S	`			ION	0
0	0	0	1	1	⊙	BS	{	+	A	J	/	1	a	J		^	A	J	IOF	1
0	0	1	0	2	△	∧	μ	<	B	K	S	2	b	k	s		B	K	S	2
0	0	1	1	3	+	=	π	>	C	L	T	3	c	l	t		C	L	T	3
0	1	0	0	4	×	→	φ	Δ	D	M	U	4	d	m	u		D	M	U	4
0	1	0	1	5	◇	NL	θ	[	E	N	V	5	e	n	v		E	N	V	5
0	1	1	0	6	↑	≠	ψ	]	F	O	W	6	f	o	w		F	O	W	6
0	1	1	1	7	⊠	±	χ	\	G	P	X	7	g	p	x		G	P	X	7
1	0	0	0	8	z	_	ω	γ	H	Q	Y	8	h	q	y		H	Q	Y	8
1	0	0	1	9	Y	NUL	λ	⌟	I	R	Z	9	ı	r	z		I	R	Z	9
1	0	1	0	10	⊠	¯	α	‡	¢	!	∞	ı	α	η	v	τ	A	H	N	T
1	0	1	1	11	*	∫	δ	‡	.	$	,	#	β	θ	ξ	υ	B	Θ	Ξ	Υ
1	1	0	0	12	⊠	⊃	ε	←	<	*	%	@	γ	ι	o	φ	Γ	I	O	Φ
1	1	0	1	13	ı	v	η	×	(	)	—	'	δ	κ	π	χ	Δ	K	Π	X
1	1	1	0	14	*	~	SUP	↑	+	;	>	=	ε	λ	ρ	ψ	E	Λ	P	Ψ
1	1	1	1	15	_	~	SUB	↓	ı	¬	?	"	ζ	μ	σ	ω	Z	M	Σ	Ω

SYMBOL VALUE = 16*COL + ROW

5.6 JOB CONTROL REQUIREMENTS

The relevant User's Guides contain introductions to the process of running user programs; the following paragraphs describe the particular requirements for graph plotting jobs. Note that at NUMAC and under VME/B on the ICL 2980 plotter jobs must be run in solo, not batch, mode.

Reference should be made to:

EMAS User's Guide - Chapter 4
EMAS 2900 User's Guide - Chapter 6
NUMAC OS User's Guide - Section E
2980 User's Guide - Section E

There are three aspects to the job control requirements for plotting jobs, no matter which language your program uses or on which computer your program is run. These are:

* an indication of the package your program references

* a definition of the file or files your plotting output is written to

* an indication of the plotter you wish your files to be drawn on

These three items are discussed separately for each of the computer installations on which the packages are available.

5.6.1 EMAS on the ICL 4-75 Twin Configuration

Normally EMAS is accessed in foreground mode via an interactive terminal. Jobs may be detached from foreground mode, to be run later in background mode. In either mode the commands required to perform a particular task are the same.

Package Access

The two graph plotting packages are held in the CONLIB process, in the libraries

CONLIB.GRAPHLIB the ERCC Graphpack
CONLIB.CLCMPLIB the Calcomp Simulation package

which are permitted for everyone to access.

For your program to be able to access a library you must first issue the command

APPENDLIB(libraryname)

and in the case of these two plotting libraries you must be sure that the correct library is searched first, since both packages have routines with the same names - CHCODE, IGRREC, JOBINF, FILEINF, PLOT, AXIS, SCALE, GRARFL, PLTYPE. If you already have CONLIB.CLCMPLIB appended to your library structure, but now wish to access the ERCC Graphpack routines in the library CONLIB.GRAPHLIB, you must issue the commands

REMOVELIB(CONLIB.CLCMPLIB)
APPENDLIB(CONLIB.GRAPHLIB)

Otherwise your program will fail because the wrong package will have been initialised. Similarly if you wish to change from CONLIB.GRAPHLIB to CONLIB.CLCMPLIB.

If in doubt, the command

LIBANAL(userlib)

will indicate the libraries you currently have included in your library structure and in what order. 'userlib' in this context is usually SS#LIB.

Normally the APPENDLIB command need only be given once. It will stay in force from one log-on session to the next, until you alter the library structure.

File Definition

Graph plotter files have two important characteristics:

* they are IMP sequential files (since the plotting software is written in IMP), regardless of the language of the calling program

* their structure is always fixed 80-byte records, i.e. card images; if a file of a different format were sent to a plotter the computer controlling the plotter would reject it

Hence file definition takes the form of the command

```
DEFINE(SQFILEnn,plotfile)
```

where `nn` is the output channel number referred to in the appropriate file initialisation routine OPENPLOTTER, OPENGR, PLOTS

`plotfile` may be the name of a new file, or an existing one if it was originally created as a graph plotter file, or one of the .GP or .SGP type device codes

Normally omission of the fourth parameter to DEFINE will cause the file to have a V1024 format. However, when a graph plotter program initiates a SQFILE the format F80 is automatically enforced. Remember though that the format of a file is governed by the DEFINE statement in force when it was created, and this format cannot be altered other than by first DESTROYing the file.

.GP is the standard plotter device code; .SGP is used to indicate that this file should be drawn using liquid ink, again on the standard plotter. See HELP(REMOTES) for other .GPnn devices.

The default file size of 255K bytes will cope with a file of more than 3000 card images, which is a very large plotter file. At present this is the maximum size of plotter file which EMAS will allow to be transmitted to a plotter.

N.B. If the output channel you specify in the initialisation routine call is the system plotter file on channel 96, the file definition is automatically performed for you by the EMAS Subsystem. DEFINE will fail if you try to define channel 96.

Output to Plotter

Once you have run your program, and therefore created the plotter file, you may

* keep the file for plotting later

* keep the file for merging with another drawing - if you have used the ERCC Graphpack routine FILEGRAPH/FILGR

* direct it to the relevant plotter - as chosen by PLOTTERTYPE/PLTYPE or by default

The standard plotter attached to 4-75 EMAS is the Calcomp 936 (plotter type 1 in Table 1) with the device code .GP or .SGP, as described above. If your DEFINE command references one of these device codes or the plotter file was created on channel 96 then the plotter file will be sent straight to this plotter and you will not have a copy of the file in your file index. Otherwise you may cause the plotter file to be sent to the plotter, following the program run which created it, by specifying

```
        LIST(plotfile,device)       if you wish to retain the file
or      SEND(plotfile,device)
```

Other devices may be available from time to time; see the .GPnn alternatives under HELP(REMOTES) or in the EMAS Information card.

Example

```
APPENDLIB(CONLIB.GRAPHLIB)      if needed
DEFINE(SQ50,plotfile)           SQ is an accepted abbreviation of SQFILE
RUN(program)                    IMP or FORTE program referencing the ERCC Graphpack
LIST(plotfile,.GP)              drawing the plotter file just created and retaining
                                the file
```

N.B. Since the EMAS commands APPENDLIB, DEFINE, RUN, LIST, SEND all take the form

externalroutinespec command(string (63) S)

they may in fact be called from within a routine or program, directly in IMP and indirectly (via the routine EMASFC) in FORTE; see the EMAS User's Guide.

5.6.2 EMAS on the ICL 2970

Normally EMAS is accessed in foreground mode via an interactive terminal. Jobs may be detached from foreground mode, to be run later in background mode. In either mode the commands required to perform a particular task are the same.

Package Access

The two graph plotting packages are held in the CONLIB process, in the directories

CONLIB.GRAPHICS	the ERCC Graphpack
CONLIB.CALCOMPICS	the Calcomp Simulation package

which are permitted for everyone to access.

For your program to be able to access a directory you must first issue the command

OPTION(SEARCHDIR=directory name)

and in the case of these two directories you must ensure that the one you want takes precedence since several entry points are duplicated, viz. CHCODE, IGRREC, JOBINF, FILEINF, PLOT, AXIS, SCALE, GRARFL, PLTYPE. If you already have CONLIB.CALCOMPICS in your directory search list then the command

OPTION(REMOVEDIR=CONLIB.CALCOMPICS,SEARCHDIR=CONLIB.GRAPHICS)

will ensure that the ERCC Graphpack takes precedence, and vice versa. If both are already in the directory search list then

OPTION(SEARCHDIR=directory name)

will promote the package you require to the head of the list. If in doubt

OPTION(?)

will inform you of the current state of the directory search list, among other information.

File Definition

Graph plotter files have two important characteristics:

* they are IMP sequential files since the plotting software is written in IMP - regardless of the language of the driving programs producing the files

* their structure is always fixed 80-byte records, i.e. card images; if a file of a different format were sent to a plotter the computer controlling the plotter would reject it

Hence file definition is performed by the command

DEFINE(nn,plotfile)

where nn is the output channel number specified in the appropriate initialisation routine call - OPENPLOTTER, OPENGR, PLOTS

plotfile may be the name of a new file, or an existing file, or .GP or .SGP

The graph plotting software automatically ensures that the file will have format F80, there is no need to use the fourth parameter of DEFINE for this purpose.

.GP is the standard plotter device code; .SGP requests liquid ink on the standard plotter. If nn=96, the system defined plotter file, you must not DEFINE it - DEFINE will fail.

The default file size of 255K bytes will cope with more than 3000 card images, which is a very large plotter file. This size is the present limit for acceptance at the computer driving the standard plotter.

Output to Plotter

Once you have run your program and created a plotter file you may

* keep the file for plotting later
* keep the file for merging with another drawing - if you have used the ERCC Graphpack routine FILEGRAPH/FILGR
* direct it to the relevant plotter - as chosen by default or by PLOTTERTYPE/PLTYPE

The standard plotter available from EMAS 2900 is the Calcomp 936 (plotter type 1 in Table 1) with the device code .GP or .SGP, as described above. If your DEFINE command specifies one of these device codes, or the plotter file was created on channel 96, then the plotter file will be sent directly to this plotter and you will have no copy of the file in your file index. Otherwise you may direct it to the relevant plotter at your leisure by issuing the command

```
        LIST(plotfile,device)       if you wish to retain the file
or      SEND(plotfile,device)
```

Other devices than .GP and .SGP may become available; see the .GPnn alternatives under HELP(DEVICES) or in the EMAS 2900 Information card.

Example

```
OPTION(SEARCHDIR=CONLIB.GRAPHICS)      if needed
DEFINE(50,plotfile)                    connect plotter file to channel 50
RUN(program)                           program referencing the ERCC Graphpack
LIST(plotfile,.GP)                     draw a copy of the file contents on the
                                       Calcomp 936 at The King's Buildings
```

N.B. Since the commands OPTION, DEFINE, RUN, LIST, SEND all take the form

```
externalroutinespec command(string (255) parameters)
```

they may be called from within a program or routine, directly in IMP and indirectly (via the routine EMASFC) in FORTE - see the EMAS 2900 User's Guide.

5.6.3 VME/B on the ICL 2980

Graph plotting jobs must be run in solo mode; they are not allowed within a Scientific Jobber batch.

Package Access

The two graph plotting packages are held in OMF libraries owned by :SYSTEM. To obtain access call one of the macros

```
EXTEND_LIBRARY_LIST(:SYSTEM.GRAPHLIB)       ERCC Graphpack
EXTEND_LIBRARY_LIST(:SYSTEM.CLCMPLIB)       Calcomp Simulation Package
```

- EXLBL is the accepted short form of EXTEND_LIBRARY_LIST. The macro call should be placed within the SCL block where program loading and execution occurs.

If the plotting software is to be accessed from an ICL FORTRAN program you must also include the SCL statement

```
ICL9CEJINIT
```

before running the program.

File Definition

Graph plotter files have two important characteristics:

* they are IMP sequential files since the plotting software is written in IMP - regardless of the language of the driving program

* their structure is always fixed 80-byte records, i.e. card images; if a file of a different format were sent to a plotter the computer controlling the plotter would reject it

Hence file definition must be performed by the following SCL statements:-

```
NEWFILE(plotfile)
ASSIGNFILE(NAME=plotfile,LNAME=ICL9CEnn,ACCESS=W)
```

or

```
WORKFILE(LNAME=ICL9CEnn,DESC=*STDGP)
```

if the file is to be sent directly to the plotter,

where 'nn' is the channel number specified in the relevant plotter initialisation routine - OPENPLOTTER, OPENGR or PLOTS

Output to Plotter

Once you have produced a plotter file you may

* keep the file for plotting later - call the SAVEFILE(plotfile) macro

* keep the file for merging with another drawing, if you have used the ERCC Graphpack routine FILEGRAPH/FILGR - again call SAVEFILE(plotfile)

* direct it to the relevant plotter as chosen by default or by PLOTTERTYPE/PLTYPE

The standard plotter available from the ICL 2980 is the Calcomp 663 situated in Buccleuch Place Lane - plotter type 2 or 3 in Table 1. You must provide a description of the file to be plotted; this is either

```
DESC = *STDGP               (as in WORKFILE above) for biro output
or   DESC = :STD.STDGP1111   for liquid ink output
```

If WORKFILE is used the plotter file will be placed in a queue for output and routed to Buccleuch Place Lane by the 2980 operators. Otherwise you should call the macro

```
LISTFILE(NAME=plotfile,DESC=description,DEVICE=GP14)
```

Other devices are not yet connected to the 2980 system.

Example

```
.
.
.
EXLBL(:SYSTEM.GRAPHLIB)                  to access the ERCC Graphpack
NEWFILE(PLOTOUT)                         create a new file
ASSIGNFILE(PLOTOUT,ICL9CE50,ACC=W)       assign it to channel 50 for writing to
IMP(INPUT=programfile)                   compile and run program
----
    data for program writing plotter
    output to channel 50
++++
LISTFILE(PLOTOUT,DESC=*STDGP,DEV=GP14)   list file to plotter at Buccleuch Place Lane
.
.
```

5.6.4 OS on the NUMAC IBM 360/370 Configuration

Package Access

The following table indicates the catalogued procedures which may be used when running graph plotter programs, and the relevant libraries containing the plotting package routines.

Catalogued Procedure	Program Language	Library	
		ERCC Graphpack	Calcomp Simulation Package
IMP IMPCLG SIMRUN	IMP	SYS5.SYSLIB	not available
FORTE FORTECLG SIMRUN	FORTE	SYS5.SYSLIB	SYS5.CLCMPLIB
FORTRAN FORTRANG FORTRANH LINKANGO LOADANGO	IBM FORTRAN	SYS5.OSSIMLIB	SYS5.OSSIMLIB

Notes:

* SYS5.SYSLIB is automatically available via the catalogued procedures IMP, IMPCLG, FORTE, FORTECLG and SIMRUN - no reference need be made to it.

* To access SYS5.CLCMPLIB from the procedures FORTE and SIMRUN, the JCL statement required is

```
//G.USERLIB  DD DSN=SYS5.CLCMPLIB,DISP=SHR
```

  and from the procedure FORTECLG

```
//L.USERLIB  DD DSN=SYS5.CLCMPLIB,DISP=SHR
```

* All of the IBM FORTRAN procedures contain a DD statement with the ddname GRAPHLIB; you must provide an INCLUDE card specifying which package your program requires from SYS5.OSSIMLIB.

  For the procedures FORTRANG, FORTRANH and LINKANGO the following cards should be added after the program in your job deck:

```
//L.SYSIN    DD *
   INCLUDE   GRAPHLIB(packname)
```

  and for the procedures FORTRAN and LOADANGO

```
//G.SYSLIN2  DD *
   INCLUDE   GRAPHLIB(packname)
```

  where 'packname' is ERCCPACK for the ERCC Graphpack, and
  CLCMPACK for the Calcomp Simulation Package.

* You may make use of overlay facilities with IBM FORTRAN programs, but the graph plotting package used must be in the root segment.

Error Messages

The graph plotting routines in both the ERCC Graphpack and the Calcomp Simulation Package are written in IMP. Consequently the various error messages, which may appear in the event of invalid parameter values being passed to the routines, are written to a line printer file with ddname STREAM99.

The relevant DD statement is automatically included in the JCL statements of the procedures IMP, IMPCLG, FORTE, FORTECLG and SIMRUN.

For the procedures FORTRAN, FORTRANG, FORTRANH, LINKANGO, and LOADANGO you must provide the JCL statement

```
//G.STREAM99 DD SYSOUT=A,DCB=(RECFM=FA,BLKSIZE=133)
```

File Definition

Two facts should be borne in mind about graph plotter files:

* they are IMP sequential files (since the plotting software is written in IMP), regardless of the language of the calling program
* their structure is always fixed 80-byte records (card images); if a file of a different format were sent to a plotter, the computer controlling the plotter would reject it

Hence a JCL statement defining a plotter file takes the following form:

```
//G.SQFILEnn DD DCB=(RECFM=F|FB,LRECL=80,BLKSIZE=80|3120),'device characteristics'
```

where 'nn' is the output channel number referred to in the appropriate file initialisation routine (OPENPLOTTER, OPENGR or PLOTS)

RECFM and BLKSIZE are given values dependent upon the 'device characteristics'

Notes

* If the plotter file is to be saved on disk, the full DD statement becomes

```
//G.SQFILEnn DD DSN=&&tempname|jobname.filename,DISP=(,PASS|CATLG),UNIT=DISK,
//       SPACE=(TRK,(3,1)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
```

  where the SPACE parameter will allow for a plotter file of approximately 451 records in the primary extent, a larger than average plotter file.

* If the plotter file is to be saved on magnetic tape, the full DD statement becomes

```
//G.SQFILEnn DD DSN=jobname.filename,DISP=(,KEEP),UNIT=TAPE,
//     VOL=SER=serial,DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120),LABEL=(label,,,OUT)
```

  and a /*TAPE card is also required specifying that tape 'serial' is to be written to.

* If the plotter file is to be merged with another new plotter file, the full DD statement describing the file to be merged becomes

```
//G.SQFILEnn DD DSN=&&tempname|jobname.filename,DISP=OLD
```

  for a disk file, and for a magnetic tape file,

```
//G.SQFILEnn DD DSN=jobname.filename,DISP=OLD,UNIT=TAPE,VOL=SER=serial,
//       LABEL=(label,,,IN)
```

  In the latter case a /*TAPE card is also required.

* If the plotter file is to be saved on punched cards, the full DD statement becomes

```
//G.SQFILEnn DD SYSOUT=B,DCB=(RECFM=F,BLKSIZE=80)
```

  and the CARDS (or C) parameter in the JOB card controlparams field should reflect the number of cards you expect the file to occupy: 400 is an average plotter file. If you save more than one file on cards in this way the CARDS parameter should be an estimate of the total number of cards required.

Output to Plotter

The standard graph plotter, the Calcomp 936 (plotter type 1 in Table 1), is connected to EMAS running in the ERCC ICL 4-75 complex. This appears to HASP as a Remote Job Entry terminal. To direct a graph plotter file to this plotter from NUMAC you require a /*ROUTE card, of the form

```
/*ROUTE PUNCH REMOTE34
```

since the plotter is regarded by HASP as a remote card punch. Other remote graph plotters may become available.

Notes

* If the plotter file is to be plotted directly from your program the DD statement becomes

```
//G.SQFILEnn DD SYSOUT=B,DCB=(RECFM=F,BLKSIZE=80)
```

or, when liquid ink plotting is required (i.e. pen code>4),

```
//G.SQFILEnn DD SYSOUT=(K,,1111),DCB=(RECFM=F,BLKSIZE=80)
```

The CARDS parameter in the JOB card controlparams field should reflect the plotter file size in card images.

The /*ROUTE card causes all files designated SYSOUT=B to be sent to the specified remote. If you require to output punched cards from the same program you must save them as card images in a disk or magnetic tape file, for punching later in another job.

* If you have a previously created plotter file (saved on disk or magnetic tape) which you now wish to be plotted, then a typical job to perform this operation would be

```
//jobname    JOB 'R=64K,C=500','delivery information'
/*ROUTE  PUNCH REMOTE34
//      EXEC PGM=IEBGENER
//SYSPRINT   DD SYSOUT=A
//SYSIN      DD DUMMY
//SYSUT1     DD DSN=jobname.filename,DISP=OLD
//SYSUT2     DD SYSOUT=B,DCB=(RECFM=F,BLKSIZE=80)
//
```

assuming that the file was on a disk. If the file were on magnetic tape the SYSUT1 statement would be

```
//SYSUT1     DD DSN=jobname.filename,DISP=OLD,UNIT=TAPE,VOL=SER=serial,
//       LABEL=(label,,,IN)
```

together with a /*TAPE card; or, if the file had been kept on cards, SYSUT1 would be

```
//SYSUT1     DD DATA
  <data cards>
/*
```

Examples

1. IMP program accessing the ERCC Graphpack and outputting to the plotter.

```
//jobname    JOB 'R=200K,C=400','delivery information'
/*ROUTE  PUNCH REMOTE34
//      EXEC IMP,REGION.C=200K,REGION.G=150K,PARM.G=MAP
//C.SYSIN    DD *

   <IMP program writing the plotter file to channel 10>

//G.SQFILE10 DD SYSOUT=B,DCB=(RECFM=F,BLKSIZE=80)
//G.SYSIN    DD *
   <data>
//
```

2. FORTE program accessing the ERCC Graphpack and saving the plotter file on disk.

```
//jobname    JOB 'R=150K,C=400','delivery information'
//      EXEC FORTE,REGION.C=150K,REGION.G=150K,PARM.G=MAP
//C.SYSIN    DD *

   <FORTE program writing the plotter file to channel 20>

//G.SQFILE20 DD DSN=jobname.filename,DISP=(,CATLG),UNIT=DISK,
//    DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120),SPACE=(TRK,(3,1))
//G.SYSIN    DD *
   <data>
//
```

3. Pre-compiled FORTE program accessing the Calcomp Simulation Package and outputting to the plotter.

```
//jobname    JOB 'R=150K,C=400','delivery information'
/*ROUTE  PUNCH REMOTE34
//      EXEC SIMRUN,REGION=150K,PARM=MAP
//LINKIN     DD DSN=jobname.program,DISP=SHR
//USERLIB    DD DSN=SYS5.CLCMPLIB,DISP=SHR
//SQFILE10   DD SYSOUT=B,DCB=(RECFM=F,BLKSIZE=80)
//SYSIN      DD *
   <data>
//
```

4. IBM FORTRAN program accessing the Calcomp Simulation Package and outputting to the plotter.

```
//jobname    JOB 'R=192K,C=400','delivery information'
/*ROUTE  PUNCH REMOTE34
//      EXEC FORTRANG,REGION.G=192K,PARM.L=MAP
//C.SYSIN    DD *

   <IBM FORTRAN program writing the plotter file to channel 10>

//L.SYSIN    DD *
     INCLUDE GRAPHLIB(CLCMPACK)
//G.STREAM99 DD SYSOUT=A,DCB=(RECFM=FA,BLKSIZE=133)
//G.SQFILE10 DD SYSOUT=B,DCB=(RECFM=F,BLKSIZE=80)
//G.SYSIN    DD *
   <data>
//
```

5. IBM FORTRAN program accessing the ERCC Graphpack, creating a new plotter file and merging onto it an old plotter file saved on disk by a program, IMP or FORTRAN, using the routine FILEGRAPH/FILGR. Output is again to the plotter.

```
//jobname    JOB 'R=192K,C=400','delivery information'
/*ROUTE  PUNCH REMOTE34
//      EXEC FORTRAN,REGION.C=150K,REGION.G=192K,PARM.G=MAP
//C.SYSIN    DD *

   <IBM FORTRAN program writing the plotter file to channel 10
    and merging the old plotter file from channel 50>

//G.SYSLIN2  DD *
    INCLUDE  GRAPHLIB(ERCCPACK)
//G.SQFILE50 DD DSN=jobname.mergefile,DISP=SHR
//G.SQFILE10 DD SYSOUT=B,DCB=(RECFM=F,BLKSIZE=80)
//G.STREAM99 DD SYSOUT=A,DCB=(RECFM=FA,BLKSIZE=133)
//G.SYSIN    DD *
   <data>
//
```

CHAPTER 6

THE EDINBURGH TEKTRONIX INTERACTIVE GRAPHICS PACKAGE

Tektronix Terminals

The 4000 Series of terminals manufactured by the Tektronix Company are versatile, dual-purpose devices, capable of transmitting, receiving and displaying data in either alpha-numeric (character) or graphical (drawing) mode, in an interactive fashion. The terminals are of the storage tube type and hence are able to sustain a display indefinitely. This also means that any graphical display cannot be altered without completely re-drawing it and that, consequently, moving pictures are impossible. Figure 1 shows a typical model from the Tektronix series.

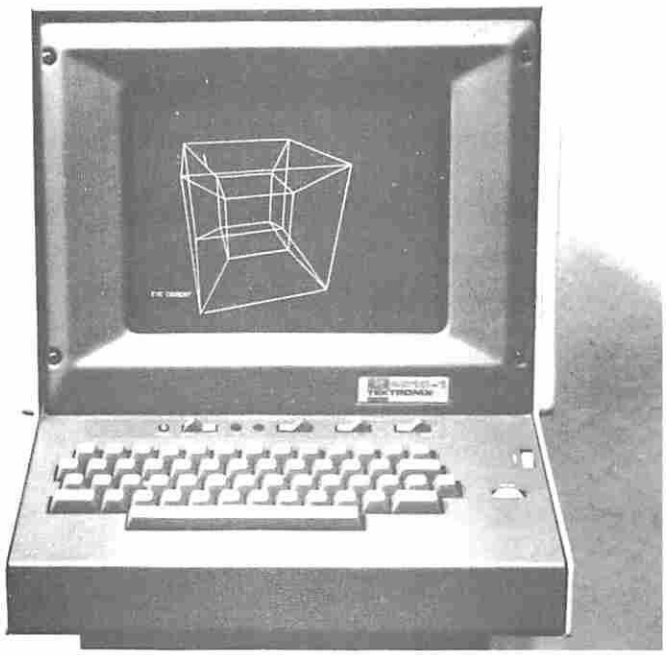


Figure 1. A Tektronix model 4010 terminal.

Within the Edinburgh Regional Computing Centre, a model 4006 terminal is accessible at the Buccleuch Place Lane installation, a 4010 at 59, George Square and a 4014 in the James Clerk Maxwell Building.

Each terminal in the range (models 4002, 4006, 4010, 4012, 4014, etc.) has a rectangular screen; in the case of the 4014, it is some 15 inches wide and 12 inches high, and in the other models, 8.5 inches wide and 6.5 inches high. All models are fitted with a key-board compatible with that of a standard teleprinter. Except for the 4002 and 4006, they also have thumb-wheels or a 'joy-stick' to control a pair of horizontal and vertical cross-hairs (the 'Cursor'), which may be displayed on the screen and moved independently to intersect at any desired point on it.

For graphical purposes, the screen is considered as a rectangular array of points, arranged in 780 rows of 1024 points - that is, nominally 1024 rows of 1024 points, of which only 780 rows are guaranteed to be visible. The points are addressed by X- and Y-co-ordinates, numbered 0 to 1023, horizontally from left to right and 0 to 779, vertically from bottom to top, respectively. The model 4014 terminal has an 'enhanced graphics option' allowing the resolution to be increased four-fold in each direction (i.e. nominally 4096 rows of 4096 points, of which 3120 rows are visible).

Drawing is conducted by moving the electron beam arbitrarily from point to point, tracing its movement, as in a child's 'join-the-dots' game. The beam may also be moved without leaving a trace, where the drawing is disjointed. The result is a 'wire-frame' line drawing, which may be interpreted as being two-dimensional or as a projection of a three-dimensional object. The enhanced graphics option of the model 4014 terminal permits drawing of dashed and dotted lines and display of points with various levels of intensity.

When the terminal is first switched on, the entire screen becomes illuminated as it warms up. The 'PAGE' key must be pressed to clear the screen before the terminal is used.

The terminals have a facility built in to protect the phosphor coating on the screen. After displaying a picture inactively for some seconds, the intensity of the display is automatically reduced. It can be restored by pressing any key (a non-printing key such as 'SHIFT' is sufficient).

The Sigma Graphics Option Controller

A standard alphabetic frame-video terminal, on whose screen symbols appear as patterns of illumated dots in a rectangular matrix (typically of 7 rows by 5 columns), e.g. the ITT and Dacoll terminals commonly used in the ERCC public terminal areas, can be used to produce low-resolution graphics in an interactive fashion, simulating the action of Tektronix terminals, by interfacing them with a Sigma Graphics Option Controller (GOC).

This device has the effect of allowing the video screen to be considered as an array of 256 rows of 256 points, i.e. one quarter of the resolution of the standard Tektronix terminals, but with very similar characteristics otherwise. A key-pad is attached to the device to control a cross-hair cursor. It is fitted with five keys, which have the functions of moving the cursor upward, downward, leftward and rightward and recording a 'hit', i.e. fixing the cursor position. The keys may also be used to reset the device in certain contingencies.

A typical configuration incorporating the GOC is shown in Figure 2.

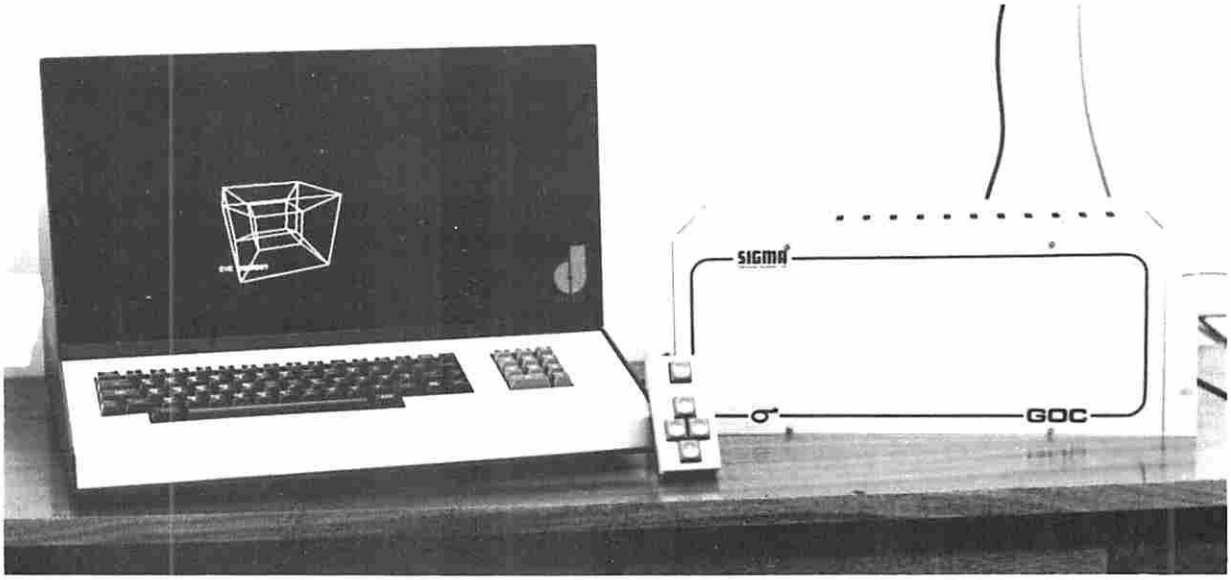


Figure 2. A Sigma GOC with a Dacoll terminal.

The Package

The Edinburgh Tektronix Package, which is available on EMAS on both the System 4 and 2970 implementations, provides an interactive user interface to the model 4002, 4006, 4010, 4012 and 4014 terminals. In addition, it may be used to produce low-resolution graphical displays on an alphanumeric display video terminal fitted with the Sigma GOC interface. It provides a means of producing a hard copy of the resultant drawing on the ERCC Calcomp model 936 Graph Plotter. The package also allows a drawing recorded in a file in ERCC Graph Plotter File format to be viewed on a display for inspection before committing to hard copy.

On the System 4 machines, the package is accessed by the command:

```
APPENDLIB(CONLIB.TEKLIB)
```

while on the 2970, it is obtained by the command:

```
OPTION(SEARCHDIR=CONLIB.GRAPHICS)
```

The Package consists of a number of routines which provide the building blocks from which graphics-based applications programs may be constructed. They may be called from IMP or FORTRAN user programs.

Graphics Mode

Before attempting to execute any program invoking the Tektronix Package, the user must first ensure that the terminal in use will communicate with the host system in 'Graphics Mode', that is, without translation of lower-case input characters and disabling control and format characters occurring in output. This is achieved by first pressing the 'CTRL' and 'A' keys on

the keybord simultaneously, followed by 'G' and then 'RETURN'. The response will be:

```
        SET [G ]
```
or:
```
        SET [ GRAPH MODE ]
```

depending upon the level of Terminal Control Processor software in use.

Defining Terminal Type

The various terminals in the Tektronix range have rather different characteristics and hence it is necessary to identify the model in use. This is achieved by invoking the routine:

```
        externalroutine TEKTYP(integer M)
```
or:
```
        SUBROUTINE FTKTYP(M)
```

in an IMP or FORTRAN context, respectively. The parameter 'M' is the Tektronix terminal number, e.g.

```
        TEKTYP(4010)
```
or:
```
        CALL FTKTYP(4014)
```

The Sigma GOC interface controlling a standard alphabetic video terminal is identified by the terminal number 999.

Erasing the Screen

Before drawing starts, and periodically thereafter, it is desirable to be able to clear the screen. This can be done either manually, by pressing the 'PAGE' key on the Tektronix terminal keyboard ('CLEAR' or equivalent key on other terminals), or from within the program, by invoking the routine:

```
        externalroutine ERASE
```
or:
```
        SUBROUTINE ERASE
```

The Pseudo-Display File

Whilst drawing is progressing on the screen, the Package maintains a record of the drawing in its working space, termed the 'Pseudo-display File', which subsequently may be copied to a permanent file for transmission to a graph plotter. This data space must normally be initialised before drawing sta.ts, by calling the routine:

```
        externalroutine NEWPIC
```
or:
```
        SUBROUTINE NEWPIC
```

Following completion of a drawing, the Pseudo-display file may be re-initialised in preparation for a further drawing, if so desired, by calling the same routine. This does not cause the screen to be erased, thus several drawings may be superimposed.

It may be desirable to suppress the storage of certain parts of a drawing in the pseudo-display file. This may be done selectively by calling the routine:

```
        externalroutine STOREOFF
```
or:
```
        SUBROUTINE FSTOFF
```

and storage in the pseudo-display file may be re-enabled by calling the routine:

```
        externalroutine STOREON
```
or:
```
        SUBROUTINE FSTON
```

These routines may be called repeatedly following the call of NEWPIC, which sets an initial default situation equivalent to STOREON.

Similarly, it may be required to store the drawing in the pseudo-display file, without displaying it on the screen. This may again be done selectively, using the routine:

```
        externalroutine VIEWOFF
or:
        SUBROUTINE FVWOFF
```

to supress the display and the routine:

```
        externalroutine VIEWON
or:
        SUBROUTINE FVWON
```

to re-enable the display. VIEWON is implied by calling NEWPIC.

Windows

The Package allows drawings to be constructed in a 'Virtual Picture Space', in which X- and Y-co-ordinates (horizontal and vertical respectively) may be defined in the range -32768 to +32767, in unit increments. Within this space, the user may define any rectangular 'Virtual Window', which delimits the extent of the drawing he requires to display on the screen.

The routine NEWPIC automatically sets default limits for the virtual window to be 0 <= X, Y <= 1023. The user is at liberty to redefine these subsequently, by calling the routine:

```
        externalroutine VWINDO(integer VXO, VYO, VXM, VYM)
or:
        SUBROUTINE FVWNDO(VXO, VYO, VXM, VYM)
```

which defines the virtual window as the rectangle whose lower left-hand corner has virtual co-ordinates (VXO,VYO) and whose upper right-hand corner is (VXM,VYM). NEWPIC defines the virtual window by default as:

```
        VWINDO(0,0,1023,1023)
```

On the screen itself, the user may define any rectangular area, known as the 'Screen Window', within which he wishes the drawing in the virtual window to be displayed. The drawing is scaled and 'clipped' automatically to fit the screen window exactly. The screen window may be defined to have X- and Y-co-ordinates limited to any range from 0 to 1023, although points with Y-co-ordinates from 0 to 779 only are visible on the screen. This is achieved by calling the routine:

```
        externalroutine SWINDO(integer SXO, SYO, SXM, SYM)
or:
        SUBROUTINE FSWNDO(SXO, SYO, SXM, SYM)
```

The screen window is the rectangle whose lower left-hand corner is the point (SXO,SYO) and whose upper right-hand corner is the point (SXM,SYM) on the screen. The default definition of the screen window, which is set by NEWPIC, is:

```
        SWINDO(0,0,1023,1023)
```

thus, the default virtual window maps directly on to the whole screen.

This concept of windowing is illustrated by Figure 3.

The virtual window and screen window may be redefined dynamically, to enable the user to see different parts of his drawing at different magnifications and orientations within different areas of the screen, simultaneously, if he so wishes.

A rectangular frame may be drawn around the current screen window by calling the routine:

```
        externalroutine FRAME
or:
        SUBROUTINE FRAME
```

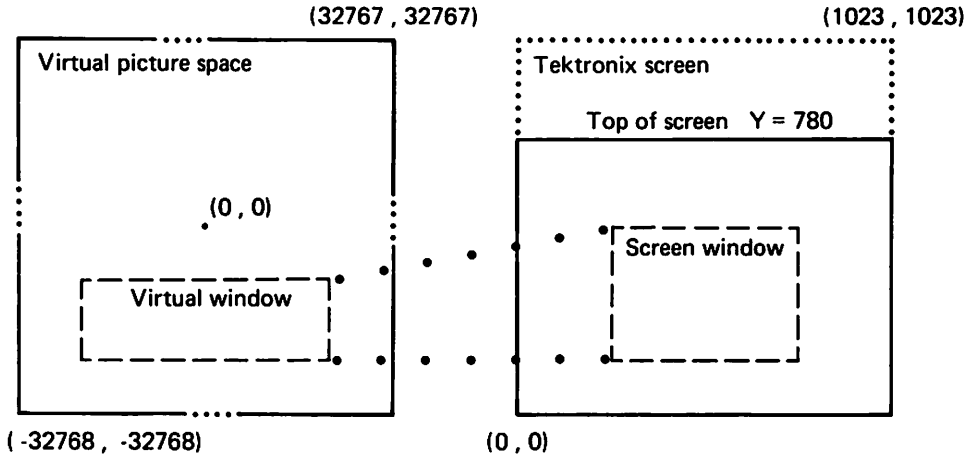


Figure 3. Virtual and screen windows.

Drawing Operations

Drawing may commence immediately following the initialisation of the pseudo-display file and definition of the virtual and screen windows (explicitly or by default). Drawing is conducted in terms of the co-ordinates of the virtual working space, but only that part of the drawing lying within the virtual window will be seen on the screen. Movement around the virtual picture space may be defined either in terms of the absolute co-ordinates of points within it or as vectors relative to the 'current virtual position'. Initially (i.e. immediately after calling NEWPIC), the current position is the virtual origin (the point with virtual co-ordinates (0,0)).

The routines provided for drawing are as follows:

```
        externalroutine DRAWA(integer X, Y)
or:
        SUBROUTINE FDRAWA(X, Y)
```

draws a visible vector from the current position to the point with absolute virtual co-ordinates (X,Y), which becomes the current position.

```
        externalroutine POINTA(integer X, Y)
or:
        SUBROUTINE FPNTA(X, Y)
```

moves invisibly to a new current position at the point with absolute virtual co-ordinates (X,Y) and draws a dot at that point.

```
        externalroutine MOVEA(integer X, Y)
or:
        SUBROUTINE FMOVEA(X, Y)
```

moves to a new current position at the point with absolute virtual co-ordinates (X,Y) without leaving a trace.

```
        externalroutine DRAWR(integer DX, DY)
or:
        SUBROUTINE FDRAWR(DX, DY)
```

draws a vector in the direction specified by the virtual X- and Y-co-ordinate increments DX and DY relative to the current position. A positive value of DX implies movement to the right and a negative value to the left, while a positive value of DY implies upward movement and a negative value downward.

```
        externalroutine POINTR(integer DX, DY)
or:
        SUBROUTINE FPNTR(DX, DY)
```

traverses invisibly the vector specified by the increments DX and DY and draws a dot at the new current position.

```
        externalroutine MOVER(integer DX, DY)
```
or:
```
        SUBROUTINE FMOVER(DX, DY)
```

similarly moves to a new position relative to the current position, but does not leave a trace.

All these routines cause the terminal to operate in 'graphics' mode automatically. Provided that the display is enabled (VIEWON), and that they lie within or traverse the currently defined virtual window, the vectors will appear on the screen and, provided the pseudo-display file is enabled (STOREON), they will be recorded there.

The drawing operation fails, and the user program is terminated summarily, if the end-point of the vector is not inside the Virtual Picture Space (i.e., the co-ordinates of the current position do not lie in the range -32768 <= X, Y <= 32767).

Enhanced Graphics Mode

The model 4014 terminal has a number of sophisticated facilities which are not available on the other terminals in the Tektronix series. These are collectively termed 'Enhanced Graphics Options'. They include the ability to increase the resolution of the screen four-fold, considering it to be nominally as comprising 4096 rows of 4096 points, of which only 3120 rows are guaranteed visible. Other facilities are the ability to draw dashed and dotted lines and to display points at various levels of intensity on the screen.

These facilities are invoked by calling the routine:

```
        externalroutine ADVGPH(integer I)
```
or:
```
        SUBROUTINE FADVG(I)
```

The parameter I is set to 1 to enable the enhanced graphics options and to 0 to disable them. The routine has an effect only when the model 4014 terminal is in use; it is ignored otherwise. After calling:

```
        ADVGPH(1)
```
or:
```
        CALL FADVG(1)
```

the user program may define a screen window with X- and Y-co-ordinates in the range 0 <= X, Y <= 4096 (although Y-co-ordinates greater than 3120 do not define visible points).

The routine:

```
        externalroutine SETLINE(integer I)
```
or:
```
        SUBROUTINE FSTLN(I)
```

may be used to affect the appearance of visible vectors in subsequent drawing operations. If the parameter I is set to 0, normal vectors are to be drawn; a value of 1 defines dotted vectors; 2 implies vectors are composed of dots and dashes; 3 means short dashes and 4 long dashes. This will apply to all vectors until SETLINE or FSTLN is called again.

The routines:

```
        externalroutine SPOINTA(integer X,Y,I)
```
or:
```
        SUBROUTINE FSPNTA(X,Y,I)
```

and:

```
        externalroutine SPOINTR(integer DX,DY,I)
```
or:
```
        SUBROUTINE FSPNTR(DX,DY,I)
```

have identical effects to those of POINTA (FPNTA) and POINTR (FPNTR) respectively, except that the brightness of the dot may be controlled by the parameter I, whose value may vary from 1 (almost invisible) to 62 (full intensity, the default brightness). This enables part of the display to stand out from the rest; it is particularly useful in applications involving the projection of three-dimensional images.

Example 1

The following IMP program shows how a simple drawing of a rectangle with its diagonals (one appearing as a dotted line) could be displayed.

```
externalroutinespec TEKTYP(integer N)
externalroutinespec ERASE
externalroutinespec NEWPIC
externalroutinespec MOVEA(integer X,Y)
externalroutinespec DRAWR(integer DX,DY)
externalroutinespec MOVER(integer DX,DY)
externalroutinespec POINTR(integer DX, DY)
externalroutinespec FRAME
begin
integer I
TEKTYP(4010)                          ; ! define model number
ERASE                                 ; ! clear screen
NEWPIC                                ; ! initialise pseudo-display file
MOVEA(411,339)                        ; ! below and to left of screen centre
DRAWR(200,0)                          ; ! base, horizontal left to right
DRAWR(0,100)                          ; ! right-hand side, upward
DRAWR(-200,0)                         ; ! top, right to left
DRAWR(0,-100)                         ; ! left-hand side, downward
DRAWR(200,100)                        ; ! diagonal, lower left to upper right
MOVER(-200,0)                         ; ! to upper left-hand corner
cycle I=1,1,10; POINTR(20,-10); repeat; ! diagonal, upper left to lower right
MOVER(-200,0)                         ; ! back to lower left corner
FRAME                                 ; ! around whole screen
endofprogram
```

The same effect would be achieved by the following FORTRAN program:

```
      CALL FTKTYP(4010)
      CALL ERASE
      CALL NEWPIC
      CALL FMOVEA(411,339)
      CALL FDRAWR(200,0)
      CALL FDRAWR(0,100)
      CALL FDRAWR(-200,0)
      CALL FDRAWR(0,-100)
      CALL FDRAWR(200,100)
      CALL FMOVER(-200,0)
      DO 1 I=1,10
      CALL FPNTR(20,-10)
    1 CONTINUE
      CALL FMOVER(-200,0)
      CALL FRAME
      STOP
      END
```

The resulting display would be as shown in Figure 4.

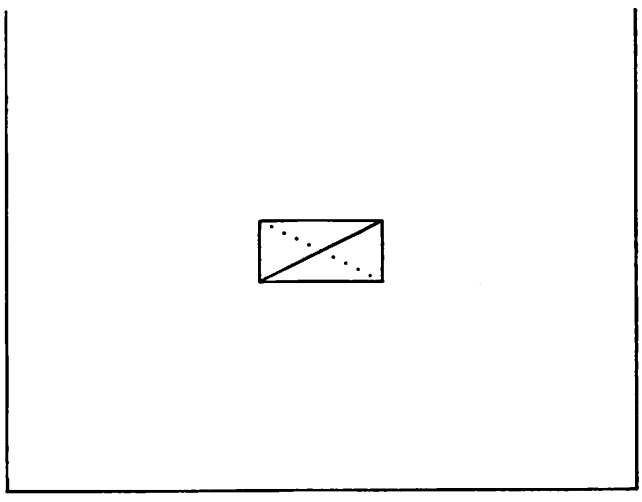


Figure 4. A simple picture.

Example 2

The effect of defining virtual and screen windows is illustrated by the following modified version of the program described in Example 1:

```
externalroutinespec TEKTYP(integer N)
externalroutinespec ERASE
externalroutinespec NEWPIC
externalroutinespec VWINDO(integer VXO,VYO,VXM,VYM)
externalroutinespec SWINDO(integer SXO,SYO,SXM,SYM)
externalroutinespec MOVEA(integer X,Y)
externalroutinespec DRAWR(integer DX,DY)
externalroutinespec MOVER(integer DX,DY)
externalroutinespec POINTR(integer DX,DY)
externalroutinespec FRAME
begin
integer I
TEKTYP(4010)
ERASE
NEWPIC
VWINDO(300,300,600,600)
SWINDO(200,100,800,700)
MOVEA(411,339)
DRAWR(200,0)
DRAWR(0,100)
DRAWR(-200,0)
DRAWR(0,-100)
DRAWR(200,100)
MOVER(-200,0)
cycle I=1,1,10
POINTR(20,-10)
repeat
MOVER(-200,0)
FRAME
endofprogram
```

or, in the FORTRAN version:

```
      CALL FTKTYP(4010)
      CALL ERASE
      CALL NEWPIC
      CALL FVWNDO(300,300,600,600)
      CALL FSWNDO(200,100,800,700)
      CALL FMOVEA(411,339)
      CALL FDRAWR(200,0)
      CALL FDRAWR(0,100)
      CALL FDRAWR(-200,0)
      CALL FDRAWR(0,-100)
      CALL FDRAWR(200,100)
      CALL FMOVER(-200,0)
      DO 1 I=1,10
      CALL FPNTR(20,-10)
    1 CONTINUE
      CALL FMOVER(-200,0)
      CALL FRAME
      STOP
      END
```

In these cases, the display would have the form shown in Figure 5.

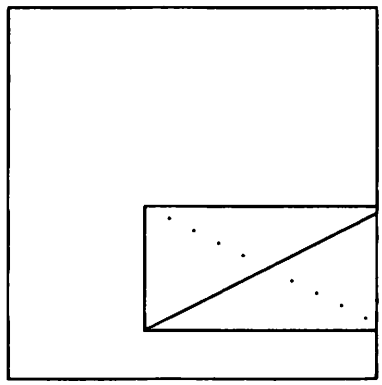


Figure 5. The use of windows.

Dynamic Windowing

Once a drawing has been completed and stored in the pseudo-display file, various views of it may be displayed on the screen by defining different virtual and screen windows and reviewing the pseudo-display file, without actually reconstructing the picture.

The routine:

```
      externalroutine REVU
or:
      SUBROUTINE REVU
```

scans the pseudo-display file, applies to it the current virtual and screen window definitions, and displays the resulting drawing. In combination with calls of VWINDO and SWINDO, it may be called repeatedly to produce a sequence of views of the drawing. If the screen is not erased between calls, these views will be superimposed.

Example 3

This effect is illustrated by another modification to the program described in Examples 1 and 2:

```
externalroutinespec TEKTYP(integer N)
externalroutinespec ERASE
externalroutinespec NEWPIC
externalroutinespec VWINDO(integer VXO,VYO,VXM,VYM)
externalroutinespec SWINDO(integer SXO,SYO,SXM,SYM)
externalroutinespec MOVEA(integer X,Y)
externalroutinespec DRAWR(integer DX,DY)
externalroutinespec MOVER(integer DX,DY)
externalroutinespec POINTR(integer DX,DY)
externalroutinespec FRAME
externalroutinespec REVU
begin
integer I
TEKTYP(4010)
ERASE
NEWPIC
MOVEA(411,339)
DRAWR(200,0)
DRAWR(0,100)
DRAWR(-200,0)
DRAWR(0,-100)
DRAWR(200,100)
MOVER(-200,0)
cycle I=1,1,10
POINTR(20,-10)
repeat
MOVER(-200,0)
FRAME
VWINDO(300,300,600,600)
SWINDO(200,100,800,700)
REVU
endofprogram
```

or, in the FORTRAN version:

```
      CALL FTKTYP(4010)
      CALL ERASE
      CALL NEWPIC
      CALL FMOVEA(411,339)
      CALL FDRAWR(200,0)
      CALL FDRAWR(0,100)
      CALL FDRAWR(-200,0)
      CALL FDRAWR(0,-100)
      CALL FDRAWR(200,100)
      CALL FMOVER(-200,0)
      DO 1 I=1,10
      CALL FPNTR(20,-10)
    1 CONTINUE
      CALL FMOVER(-200,0)
      CALL FRAME
      CALL FVWNDO(300,300,600,600)
      CALL FSWNDO(200,100,800,700)
      CALL REVU
      STOP
      END
```

In each case, the display would at first have the form of the drawing shown in Figure 4, then the drawing in Figure 5 would be superimposed. Had the routine ERASE been called immediately before the call of REVU, the first picture would have disappeared and been replaced by the second.

Sub-pictures

Often, it is required to display several manifestations of the same object within a drawing. This could clearly be achieved by including the drawing instructions within a program subroutine and executing it as often as required. This has two drawbacks. Firstly, the different drawings may require to be of different sizes, at different orientations and at different points on the screen; this would be rather difficult to achieve in general. Secondly, each copy of the drawing would be recorded in full in the pseudo-display file and hence the complexity of the display would be limited.

The concept of 'Sub-pictures' eliminates these difficulties, but allows multiple copies of objects to be displayed. A sub-picture is, in fact rather like a program subroutine, in that it contains a sequence of DRAW and MOVE instructions, which may be invoked repeatedly; but only one copy of it is stored in the pseudo-display file. Each instance of the sub-picture in the display may define scaling and orientation transformations of the vectors defined by these drawing instructions. In order to ensure that each instance may appear at a different place on the screen, the vectors defined in the sub-picture must be relative to the current position in the virtual picture space on entry to the sub-picture; that is, they must be defined by the routines DRAWR and MOVER, not DRAWA or MOVEA. In order to ensure that side effects are not caused by any instance of the sub-picture, it is advisable to return to the original current position at the end of the sub-picture; this is not done automatically by the sub-picture mechanism.

A sub-picture definition is initiated by calling the routine:

```
        externalroutine DEFSUB(integer N)
or:
        SUBROUTINE FDEFSB(N)
```

where N is any integer which uniquely identifies the sub-picture. Up to 128 sub-pictures may be defined.

The drawing instructions required to construct the sub-picture, together with any computations necessary to complete them, should follow immediately after the call of DEFSUB. They will have no effect on the screen, since DEFSUB automatically calls VIEWOFF to disable the display. The drawing, will, however, be stored in the pseudo-display file.

The drawing of the sub-picture should be terminated by calling the routine:

```
        externalroutine ENDSUB
or:
        SUBROUTINE ENDSUB
```

This may be omitted if a further sub-picture definition follows; i.e. DEFSUB calls ENDSUB automatically if the previous sub-picture definition is not terminated explicitly. This means that it is not possible to nest sub-picture definitions. ENDSUB re-enables the display (by calling VIEWON) if it was enabled before entry to the sub-picture definition.

Once it has been defined, a sub-picture may be invoked as often as desired, by calling the routine:

```
        externalroutine INSTAN(integer N, SC, OR, XR, YR)
or:
        SUBROUTINE FINST(N, SC, OR, XR, YR)
```

where N is the indentity number of the sub-picture required. SC and OR specify the scale and orientation which are to be applied to the sub-picture and XR and YR indicate whether reflection in the X- or Y-axis respectively is required.

A value of 128 for SC indicates that no scaling is to be applied, i.e. the scaling factor is unity. A larger value will represent a magnification and a smaller value a reduction in size, in each case by a factor of SC/128, in relation to the nominal size of the drawing. Care must be taken to ensure that the scaled drawing still remains entirely within the Virtual Picture Space (i.e. the co-ordinates of all points lie in the range -32768 <= X, Y <= 32767).

The orientation parameter, OR, may take values of 0, 1, 2 or 3 to indicate the drawing is to be rotated anti-clockwise through 0, 90, 180 or 270 degrees respectively about the point at which the sub-picture starts.

The parameter XR may take values of 0 or 1. The latter value indicates that the drawing is to appear as if reflected in a horizontal line passing through the point at which the sub-picture starts. A value of 0 indicates that no such reflection is to be applied.

The parameter YR similarly indicates whether reflection in a vertical axis passing through the start of the sub-picture is required.

Reflection operations are performed before scaling and orientation transformations are applied.

It is possible to instance one sub-picture within the definition of another. In such a case, the transformations defined are applied cumulatively, those defined for the instanced sub-picture being applied before those of the instancing sub-picture in any manifestation.

Example 4

The following modified version of the program described in the previous examples shows the effects of the scaling, orientation and reflection transformations on a sub-picture defining the rectangle with diagonals.

```
externalroutinespec TEKTYP(integer N)
externalroutinespec ERASE
externalroutinespec NEWPIC
externalroutinespec VWINDO(integer VXO,VYO,VXM,VYM)
externalroutinespec SWINDO(integer SXO,SYO,SXM,SYM)
externalroutinespec MOVEA(integer X,Y)
externalroutinespec DRAWA(integer X,Y)
externalroutinespec MOVER(integer DX,DY)
externalroutinespec POINTR(integer DX,DY)
externalroutinespec DRAWR(integer DX,DY)
externalroutinespec FRAME
externalroutinespec REVU
externalroutinespec DEFSUB(integer N)
externalroutinespec ENDSUB
externalroutinespec INSTAN(integer N,S,O,XR,YR)
begin integer I
TEKTYP(4010)
ERASE
NEWPIC
DEFSUB(1)              ; ! picture of rectangle
DRAWR(200,0)
DRAWR(0,100)
DRAWR(-200,0)
DRAWR(0,-100)
DRAWR(200,100)
MOVER(-200,0)
cycle I=1,1,10
POINTR(20,-10)
repeat
MOVER(-200,0)          ; ! back to start
ENDSUB                 ; ! end of picture
VWINDO(0,0,1023,779)
SWINDO(0,0,1023,779)
MOVEA(511,511)         ; ! centre of screen
INSTAN(1,128,0,0,0)    ; ! normal size, no rotation or reflection
MOVER(0,100)           ; ! up a little
INSTAN(1,64,1,0,0)     ; ! half size, rotated 90 degrees
MOVER(-50,0)           ; ! left a little
INSTAN(1,256,0,1,1)    ; ! twice normal size, reflected down left
MOVER(0,-100)          ; ! down a little
INSTAN(1,192,1,1,1)    ; ! 1.5 times normal size, rotated 90
                       ; ! degrees and reflected in both axes
FRAME
endofprogram
```

The FORTRAN version would be:

```
      CALL FTKTYP(4010)
      CALL ERASE
      CALL NEWPIC
      CALL FDEFSB(1)
      CALL FDRAWR(200,0)
      CALL FDRAWR(0,100)
      CALL FDRAWR(-200,0)
      CALL FDRAWR(0,-100)
      CALL FDRAWR(200,100)
      CALL FMOVER(-200,0)
      DO 1 I=1,10
      CALL FPNTR(20,-10)
    1 CONTINUE
      CALL FMOVER(-200,0)
      CALL ENDSUB
      CALL FVWNDO(0,0,1023,779)
      CALL FSWNDO(0,0,1023,779)
      CALL FMOVEA(511,511)
      CALL FINST(1,128,0,0,0)
      CALL FMOVER(0,100)
      CALL FINST(1,64,1,0,0)
      CALL FMOVER(-50,0)
      CALL FINST(1,256,0,1,1)
      CALL FMOVER(0,-100)
      CALL FINST(1,192,1,1,1)
      CALL FRAME
      STOP
      END
```

The result of running this program would be to be to produce a display like that shown in Figure 6.

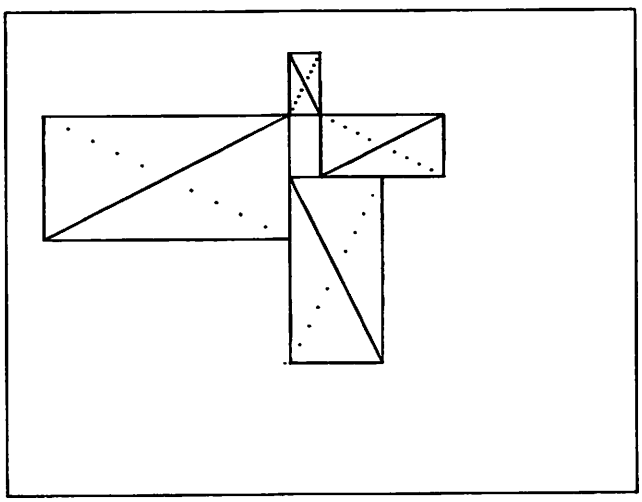


Figure 6. The use of sub-pictures.

The Cross-hair Cursor

A cursor, comprising a pair of horizontal and vertical lines, is provided on the more sophisticated models in the Tektronix terminal range and on other terminals controlled by the Sigma GOC. These cross-hair lines appear on the screen like other vectors, but rather less brightly. They have, however, a special property of mobility; they may moved independently to intersect at any point on the screen by using the thumb-wheels, joy-stick control or, in the case of the Sigma GOC, the cursor key-pad.

The cursor enables the user to identify the co-ordinates (in the virtual picture space) of any point in his drawing. This is useful in two principal situations: firstly, when constructing a drawing, the user may use the cursor to specify or establish the positions of its various components, and secondly, the cursor may be used to define the boundary of the virtual window which the user requires (by identifying the co-ordinates of its lower left-hand and upper right-hand corners).

The sequence of operations involving the cursor is as follows. First, the cursor cross-hairs are displayed (by the program) on the screen, intersecting at an arbitrary point on the screen (depending upon the current status of the thumb-wheels, joy-stick or key-pad). Its appearance is accompanied by an audible signal from the terminal. The user may then use the control provided to move the cross-hairs to any position desired. When it reaches the required position, the user may press any key on the key-board, representing a visible character, to report the cursor's position back to the program. In certain cases, it may be necessary also to press the 'RETURN' key. In the case of a terminal controlled by the Sigma GOC, it is sufficient simply to press the 'HIT' key on the key-pad. The cross-hairs then disappear from the screen.

The cursor is made to appear on the screen, and its eventual position is reported back to the controlling program, by the routine:

```
        externalroutine CURSOR(integername CH, X, Y)
or:
        SUBROUTINE CURSOR(CH, X, Y)
```

CH represents the (ISO) value of the character typed to record the cursor position (in the case of the Sigma GOC, this is always zero); X and Y are the co-ordinates, in virtual space, of the point at which the cursor position was reported. FORTRAN users should note that the value of CH returned is an integer between 0 and 127 (in fact, between 32 and 95), and a transformation is necessary if it is to be compared with character values in normal FORTRAN format.

Example 5

This example shows how a drawing may be built up by use of the cursor to position the components of the drawing and how the resulting display may be examined selectively by defining various virtual windows, again using the cursor. The program starts by displaying the cursor. The user is expected to move it to the desired position and identify, by typing 'R' or 'H', whether he wishes a rectangle or hexagon to appear in that position; if he types any other symbol, except 'Z', a dot will appear instead. This is repeated until the user types 'Z' (for 'zoom') in which case the program frames the drawing and enters a second phase. The user is then expected to use the cursor to specify which part of the drawing he wishes to see magnified, by moving the cursor first to the lower left-hand corner and then the upper right-hand corner of the area he wishes to examine. This is displayed in place of the original picture and the procedure is repeated until the user types 'S' (for 'stop') or defines a window which is inside-out.

```
externalroutinespec TEKTYP(integer N)
externalroutinespec ERASE
externalroutinespec NEWPIC
externalroutinespec VWINDO(integer VXO,VYO,VXM,VYM)
externalroutinespec SWINDO(integer SXO,SYO,SXM,SYM)
externalroutinespec MOVEA(integer X,Y)
externalroutinespec POINTA(integer X,Y)
externalroutinespec DRAWR(integer DX,DY)
externalroutinespec MOVER(integer DX,DY)
externalroutinespec POINTR(integer DX,DY)
externalroutinespec FRAME
externalroutinespec REVU
externalroutinespec DEFSUB(integer N)
externalroutinespec ENDSUB
externalroutinespec INSTAN(integer N,S,O,XR,YR)
externalroutinespec CURSOR(integername CH,X,Y)
begin
integer I, C, X, Y, XX, YY
TEKTYP(4010)
ERASE
NEWPIC
VWINDO(0,0,1023,779)
SWINDO(0,0,1023,779)
DEFSUB(1)       ; ! picture of rectangle
DRAWR(200,0)
DRAWR(0,100)
DRAWR(-200,0)
DRAWR(0,-100)
DRAWR(200,100)
MOVER(-200,0)
cycle I=1,1,10
POINTR(20,-10)
repeat
MOVER(-200,0)   ; ! back to start
ENDSUB          ; ! end of rectangle
DEFSUB(2)       ; ! picture of hexagon
MOVER(-50,-87)
DRAWR(100,0)
DRAWR(50,87)
DRAWR(-50,87)
DRAWR(-100,0)
DRAWR(-50,-87)
DRAWR(50,-87)
MOVER(50,87)    ; ! back to start
ENDSUB          ; ! end of hexagon
cycle
CURSOR(C,X,Y)
exitif C='Z'
if 'R'#C#'H' then POINTA(X,Y) elsestart
MOVEA(X,Y)
if C='R' then INSTAN(1,128,0,0,0)
if C='H' then INSTAN(2,128,0,0,0)
finish
repeat
FRAME
cycle
CURSOR(C,X,Y)
exitif C='S'
CURSOR(C,XX,YY)
exitif XX<=X or YY<=Y
ERASE
VWINDO(X,Y,XX,YY)
REVU
FRAME
repeat
ERASE
endofprogram
```

The FORTRAN version of this program would be:

```
      INTEGER I, C, X, Y, XX, YY, R, H, Z, S
      DATA R, H, Z, S /'R', 'H', 'Z', 'S'/
      R=R/16777216
      H=H/16777216
      Z=Z/16777216
      S=S/16777216
C     CONVERT FORTRAN CHARACTERS TO RANGE 0-127
      CALL FTKTYP(4010)
      CALL ERASE
      CALL NEWPIC
      CALL FDEFSB(1)
      CALL FDRAWR(200,0)
      CALL FDRAWR(0,100)
      CALL FDRAWR(-200,0)
      CALL FDRAWR(0,-100)
      CALL FDRAWR(200,100)
      CALL FMOVER(-200,0)
      DO 1 I=1,10
      CALL FPNTR(20,-10)
    1 CONTINUE
      CALL FMOVER(-200,0)
      CALL ENDSUB
      CALL FDEFSB(2)
      CALL FMOVER(-50,-87)
      CALL FDRAWR(100,0)
      CALL FDRAWR(50,87)
      CALL FDRAWR(-50,87)
      CALL FDRAWR(-100,0)
      CALL FDRAWR(-50,-87)
      CALL FDRAWR(50,-87)
      CALL FMOVER(50,87)
      CALL ENDSUB
      CALL FVWNDO(0,0,1023,779)
      CALL FSWNDO(0,0,1023,779)
    2 CALL CURSOR(C,X,Y)
      IF(C.EQ.Z) GOTO 4
      IF(C.NE.R .AND. C.NE.H) GOTO 3
      CALL FMOVEA(X,Y)
      IF(C.EQ.R) CALL FINST(1,128,0,0,0)
      IF(C.EQ.H) CALL FINST(2,128,0,0,0)
      GOTO 2
    3 CALL FPNTA(X,Y)
      GOTO 2
    4 CALL FRAME
    5 CALL CURSOR(C,X,Y)
      IF(C.EQ.S) GOTO 6
      CALL CURSOR(C,XX,YY)
      IF(XX.LE.X .OR. YY.LE.Y) GOTO 6
      CALL ERASE
      CALL FVWNDO(X,Y,XX,YY)
      CALL REVU
      CALL FRAME
      GOTO 5
    6 CALL ERASE
      STOP
      END
```

The reader may wish to note that the above programs could be modified very simply to enable the user to create 'free-hand' drawings. This is left as an exercise for those who have need of such a facility for non-frivolous applications.

Annotation of Displays

A display may be made more informative by including annotation in it. This might include titles to describe a drawing, labels for the axes of a graph, dimensions of a design or simply instructions to the user who is operating the terminal. The last type of annotation would not normally be required to be recorded in the pseudo-display file if a hard copy of the drawing were subsequently produced.

The Package provides facilities for including text in a display, in the form of either 'Hardware Characters' or 'Software Characters'. The former are the very symbols displayed when the terminal is acting in alphabetic mode, rather than graphics mode. They are of standard size (in the case of the model 4014 terminal, four sizes may be used) and always appear in horizontal lines, reading from left to right. The latter are drawn by the terminal, like other drawing components, and may be scaled and oriented as required by the user. The user may choose to include them in the pseudo-display file or not, in the case of software characters, by using the routines STOREOFF and STOREON and, for hardware characters, by using alternative printing routines. The character set in each case is the same (the Edinburgh implementation of the ISO Code symbols, excluding lower case letters), although the appearance of the characters differs in the display.

Hardware Characters and Alpha-numeric Mode

Hardware characters appear on the screen as patterns of dots in a matrix of 7 rows of 5 dots, each character contained in a rectangular area of the screen, measuring 22 points high and 14 points wide (one sixth of an inch by one tenth). The effect (much magnified) is, roughly, as shown in Figure 7.

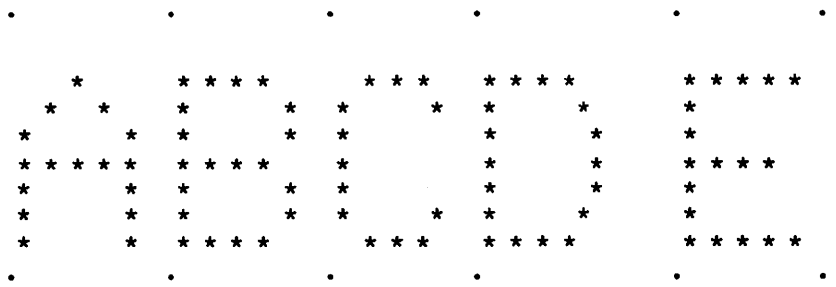


Figure 7. Hardware characters.

Whenever hardware characters are required, the terminal is set in 'alpha-numeric mode', rather than graphical mode. This is normally done automatically by the routines described below. When working in alpha-numeric mode, the terminal can display a maximum of 35 lines, each of up to 72 characters in length, on the entire screen (using 770 rows of 1008 screen points). If further lines are printed, they appear in the right-hand half of the screen. The current printing position is indicated by the 'Alpha-cursor', which appears as a dimly-blinking solid rectangle.

The initial position of the alpha-cursor may be defined by a movement in graphical mode (MOVEA or MOVER), provided this defines an actual screen position (i.e., within the current virtual window); it does not necessarily coincide with a line position normally used in alpha-numeric mode. A position outside the current virtual window will be transformed by the terminal hardware into an arbitrary screen position, so that the characters actually appear on the screen. The resultant position and any subsequent movements of the alpha-cursor, while in alpha-numeric mode, do not affect the 'current virtual position' when the terminal eventually returns to graphical mode.

The current position of the alpha-cursor may be determined by calling the routine:

```
externalroutine ALPHAPOS(integername X,Y)
```

or:

```
SUBROUTINE FALPOS(X,Y)
```

The values of X and Y returned are the virtual co-ordinates of the lower left-hand corner of the cursor. It may be necessary for the user to press the RETURN key on the keyboard to ensure that the required information is returned by the terminal; this is not normally the case.

It may occasionally be necessary to ensure that the terminal is operating in alpha-numeric mode explicitly, before printing or receiving characters. This may be achieved by calling the routine:

```
        externalroutine RTAM
or:
        SUBROUTINE RTAM
```

The return to alpha-numeric mode is indicated by the appearance on the screen of the alpha-numeric cursor, at the current screen position.

A single character may be displayed by calling the routine:

```
        externalroutine PRINTSYMBOL(integer CH)
or:
        SUBROUTINE FPSYM(CH)
```

where CH is the ISO code for the symbol required. In the case of FORTRAN, the standard representation of characters must be transformed into a number between 0 and 127 (or, rather, 32 and 95). IMP programmers should note that this routine is not the routine of the same name supplied implicitly by the IMP Compiler; it must be specified as an external routine in the user program.

A string of characters may similarly be printed on the screen by the routine:

```
        externalroutine PRINTSTRING(string(255) S)
or:
        SUBROUTINE FPSTRG(S,L)
```

where S is the string of characters to be printed and, in the case of the FORTRAN subroutine, L is the length of that string. NEWLINE (i.e. RETURN) characters within the string are ignored, while spaces are included in the display.

A NEWLINE character can be sent to the terminal by calling the routine:

```
        externalroutine NLINE
or:
        SUBROUTINE NLINE
```

The effect is to move the alpha-cursor to the beginning of the next line on the screen. Normally this will be a position at the extreme left of the screen, 22 screen points below the previous line of text. If this would be below the bottom of the screen, the alpha-cursor is moved to the top centre of the screen.

In the above cases, the characters printed are not stored in the pseudo-display file as part of the picture displayed; the text is treated merely as screen annotation.

It may be required to include hardware characters in the display in such a way that they are stored in the pseudo-display file and thus feature in any transcription of that display. The appearance of the individual hardware characters is not affected by any scaling, rotation or reflection of the display, although their position on the screen clearly is (like any other display component).

The following routines allow hardware characters to be included in the pseudo-display file:

The routine:

```
        externalroutine CHAR(integer CH)
or:
        SUBROUTINE FCHAR(CH)
```

has an effect similar to that of PRINTSYMBOL (FPSYM) except that the symbol is stored in the pseudo-display file and the graphics position is moved 12 screen points horizontally to the right after displaying it. This displacement is slightly less than that applied to the alpha-cursor by PRINTSYMBOL, but is subject to scaling, orientation and reflection transformations. The symbol always remains upright and of standard size.

The routine:

```
externalroutine DRAWTEXT(string(255) S)
```

or:

```
SUBROUTINE FDRTXT(S,L)
```

has an effect as defined for PRINTSTRING (FPSTRG) above, except that the characters are displaced 12 screen points from one another and they are stored in the pseudo-display file. The displacement is subject to scaling, orientation and reflection effects.

The routine:

```
externalroutine TWRITE(integer N,M)
```

or:

```
SUBROUTINE FWRITE(N,M)
```

prints the value of N, as a decimal integer of up to M digits, preceded by a space or minus sign. The resultant characters are stored in the pseudo-display file and their relative displacements are subject to scaling, orientation and reflections.

The routine:

```
externalroutine TPRINT(real X, integer M,N)
```

or:

```
SUBROUTINE FPRINT(X,M,N)
```

similarly prints the real number X with up to M digits before and N digits after the decimal point.

Example 6

The following IMP program shows the use of hardware characters and the effect of scaling and rotation transformations on them.

```
externalroutinespec TEKTYP(integer T)
externalroutinespec ERASE
externalroutinespec NEWPIC
externalroutinespec MOVEA(integer X,Y)
externalroutinespec MOVER(integer DX,DY)
externalroutinespec CHAR(integer CH)
externalroutinespec NLINE
externalroutinespec PRINTSTRING(string(255) S)
externalroutinespec TPRINT(real X, integer M,N)
externalroutinespec DEFSUB(integer I)
externalroutinespec ENDSUB
externalroutinespec INSTAN(integer N,S,R,XR,YR)
begin
integer K
TEKTYP(4010)
ERASE
NEWPIC
DEFSUB(1)
cycle K='A',1,'E'
CHAR(K)
repeat
TPRINT(123.456,3,3)
ENDSUB
MOVEA(300,200)
PRINTSTRING("FGHIJ"); NLINE
PRINTSTRING("KLMNO")
MOVEA(512,390)
INSTAN(1,128,0,0,0)
MOVEA(490,400)
INSTAN(1,160,1,0,0)
MOVEA(460,380)
INSTAN(1,200,2,0,0)
MOVEA(512,360)
INSTAN(1,256,3,0,0)
endofprogram
```

The equivalent FORTRAN program would be:

```
      INTEGER K,A(5)
      DATA A/'A','B','C','D','E'/
      CALL FTKTYP(4010)
      CALL ERASE
      CALL NEWPIC
      CALL FDEFSB(1)
      DO 1 J=1,5
      K=A(J)/16777216
      CALL FCHAR(K)
    1 CONTINUE
      CALL FPRINT(123.456,3,3)
      CALL ENDSUB
      CALL FMOVEA(300,200)
      CALL FPSTRG('FGHIJ',5)
      CALL NLINE
      CALL FPSTRG('KLMNO',5)
      CALL FMOVEA(512,390)
      CALL FINST(1,128,0,0,0)
      CALL FMOVEA(490,400)
      CALL FINST(1,160,1,0,0)
      CALL FMOVEA(460,380)
      CALL FINST(1,200,2,0,0)
      CALL FMOVEA(512,360)
      CALL FINST(1,256,3,0,0)
      STOP
      END
```

The resultant display would be as shown in Figure 8.

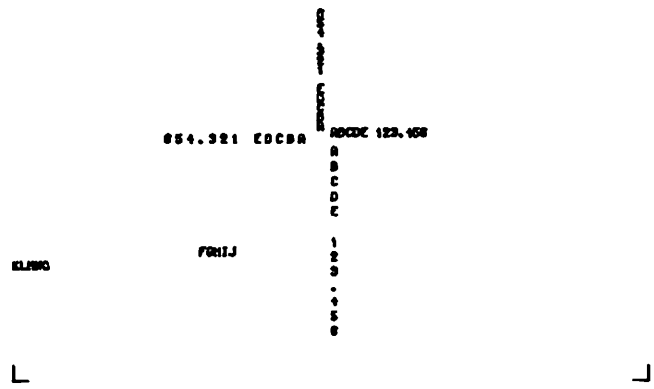


Figure 8. Scaling and rotation of hardware characters.

Software Characters

Software characters are drawn as sequences of visible and invisible vectors, just like other drawing components, while the terminal is in graphical mode. They appear as shown in Figure 9.

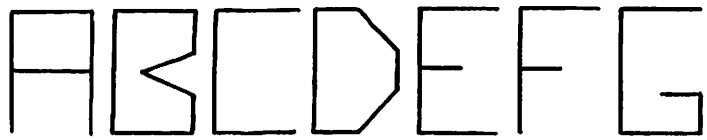


Figure 9. Software characters

The basic size of the characters is approximately one sixteenth of an inch high and one twentieth of an inch wide. This is illegible on most screens; a scaling factor may be applied to increase the size. The characters may also be rotated, individually or as a line of text, so as to suit their use in the display, as labels for axes of a graph, etc. Whatever scaling and orientation is applied to these characters is compounded by any scaling, orientation and reflection transformations defined for any sub-picture in which they appear.

The routines used to display software characters are as follows.

The routine:

```
externalroutine CHARS(integername CH,OR,X,Y,SC)
```

or:

```
SUBROUTINE CHARS(CH,OR,X,Y,SC)
```

prints the symbol whose ISO Code value (betwen 32 and 95) is represented by CH. X and Y represent the virtual co-ordinates of the position at which it is to be drawn; it is necessary to MOVE to this position before drawing the character; X and Y are updated by the displacement upon completion. SC and OR are respectively the scale (between 1 and 128) and orientation (0, 1, 2, 3 for rotations of 0, 90, 180 and 270 degrees anticlockwise, respectively). A scale of 1 is basic size (about a sixteenth by a twentieth of an inch, including displacement), while a scale of 128 fills half the screen with one character (about six inches by four).

The routine:

```
externalroutine BIGSTRING(string(255) S integer OR, SC)
```

or:

```
SUBROUTINE FBGSTR(S,L,OR,SC)
```

draws the characters stored in the string S, in the case of FORTRAN, comprising L characters, as a line of text. Both the line and the individual symbols are oriented as defined by OR, thus: OR = 0 means horizontal, upright, reading from left to right; OR = 1 means vertical, rotated 90 degrees anticlockwise, reading upward; OR = 2 is horizontal, upside-down, reading from right to left and OR = 3 is vertical, rotated 90 degrees clockwise, reading downward.

Example 7

The following example shows the effects of scaling and orientation on software characters.

```
externalroutinespec TEKTYP(integer T)
externalroutinespec ERASE
externalroutinespec NEWPIC
externalroutinespec MOVEA(integer X,Y)
externalroutinespec CHARS(integername CH,OR,X,Y,SC)
externalroutinespec BIGSTRING(string(255) S, integer R,SC)
begin
integer K,X,Y,R,S
TEKTYP(4010)
ERASE
NEWPIC
K='A'
R=0
X=0
Y=0
S=1
MOVEA(X,Y)
CHARS(K,R,X,Y,S)
K='B'
R=1
X=1000
S=2
MOVEA(X,Y)
CHARS(K,R,X,Y,S)
K='C'
R=2
Y=600
S=4
MOVEA(X,Y)
CHARS(K,R,X,Y,S)
K='D'
R=3
X=0
S=8
MOVEA(X,Y)
CHARS(K,R,X,Y,S)
MOVEA(512,390)
BIGSTRING("ABCDE",0,1)
MOVEA(500,410)
BIGSTRING("FGHIJ",1,2)
MOVEA(460,390)
BIGSTRING("KLMNO",2,4)
MOVEA(500,360)
BIGSTRING("PQRST",3,8)
endofprogram
```

The FORTRAN version of this program would be:

```
INTEGER K,X,Y,R,S,A,B,C,D
DATA A,B,C,D/'A','B','C','D'/
CALL FTKTYP(4010)
CALL ERASE
CALL NEWPIC
K=A/16777216
X=0
Y=0
CALL FMOVEA(X,Y)
CALL CHARS(K,0,X,Y,1)
K=B/16777216
X=1000
CALL FMOVEA(X,Y)
CALL CHARS(K,1,X,Y,2)
K=C/16777216
Y=600
CALL FMOVEA(X,Y)
CALL CHARS(K,2,X,Y,4)
K=D/16777216
X=0
CALL FMOVEA(X,Y)
CALL CHARS(K,3,X,Y,8)
CALL FMOVEA(512,390)
CALL FBSTRG('ABCDE',5,0,1)
CALL FMOVEA(500,410)
CALL FBSTRG('FGHIJ',5,1,2)
CALL FMOVEA(460,390)
CALL FBSTRG('KLMNO',5,2,4)
CALL FMOVEA(500,360)
CALL FBSTRG('PQRST',5,3,8)
STOP
END
```

In each case, the display would have the form shown in Figure 10.

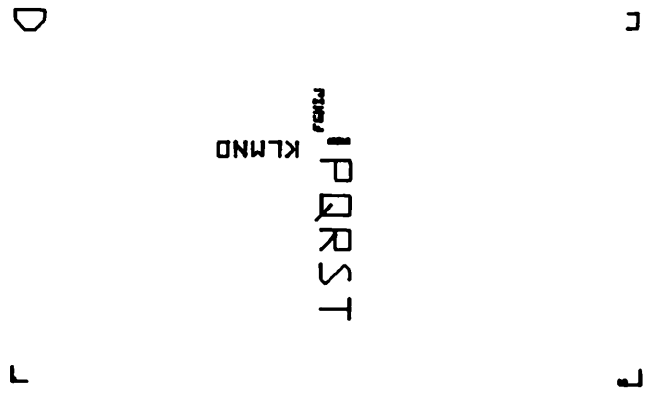


Figure 10. Scaling and rotation of software characters.

The Package provides a facility for displaying and using a 'Menu' - a device commonly used in interactive graphics programs. This consists of a list of options, representing the various activities provided by the graphics program, which is displayed at one side of the screen, while the remainder of the screen is used for drawings. The user controls the program by selecting from the options provided the action which he wishes to invoke. The program reacts accordingly, resulting in an effect upon the display. The user may then select another option, to cause a different effect. Thus the required picture may be built up in a series of operations.

The menu provided by the package is confined to an area at the left-hand side of the screen, measuring about two inches wide and six inches high, bounded by a rectangular frame. When the menu is in use, drawings must be confined to the remaining square area (six inches or so wide) at the right of the screen. (The actual dimensions of the various areas of the screen described here are nominal, for the standard Tektronix screen and they may vary for other terminal models.) The screen co-ordinates of the menu area are (0,0) at the lower left corner to (263,760) at the upper right, while those of the drawing area may perhaps be best considered to be (263,0) at the lower left to (1023,760) at the upper right.

The upper part of the menu area (some two inches square) is termed the 'Header', in which whatever information or instruction is required to enable the user to operate the menu may be printed. No more than 12 lines of up to 19 characters each may be included in this area.

The lower part of the menu area contains the menu itself. This consists of up to eleven lines of text (double spaced), each representing one of the options provided by the program. Each may be of up to 9 characters in length, but should include an identifying character which the user may type to select that option. There is always room at the bottom of the menu area for a prompt (such as 'SELECT'), which the program may generate, and to which the user is expected to respond. This he does by typing the appropriate character(s), representing his choice of option, followed by a RETURN.

Figure 11 shows a typical screen configuration including a menu.

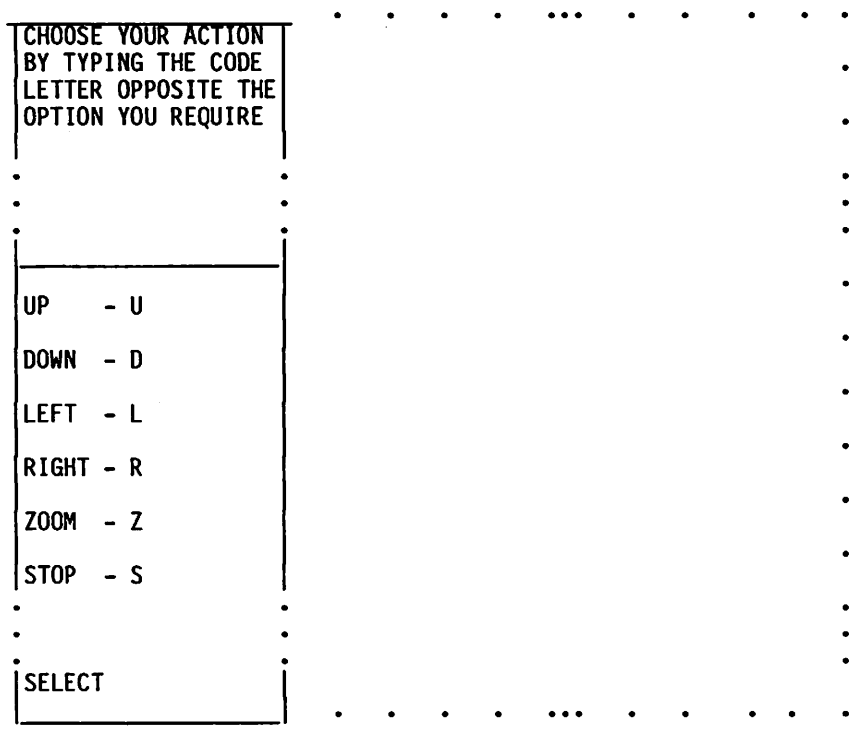


Figure 11. A menu.

The method by which the user is requested to select his option, and the way in which his response is handled, is entirely the responsibility of the graphics programmer. No specific method is defined by the Package; in particular, there is no direct means of establishing which line of the menu has been selected unless it is explicitly numbered by the user as part of the expected response.

The routine:

```
externalroutine DEFHEAD(string(230) H)
```

or:

```
SUBROUTINE FDEFHD(H,L)
```

may be used to define the text which is to be placed in the header area. The string H (of length L characters, in the FORTRAN version) may contain up to 12 lines, each of no more than 19 characters, and each terminated by a NEWLINE character (ISO code 10). The routine does not itself display the header, but merely stores the text for subsequent display when the complete menu is defined.

The routine:

```
externalroutine DEFMENU(string(9) M, integer N)
```

or:

```
SUBROUTINE FDFMNU(M,L,N)
```

defines the characters held in the string M (of length L, in the FORTRAN version) as the text which is to be displayed in the Nth line of the menu. The string may not exceed 9 characters in length. The menu line number (N) may be between 1 and 11, thus a complete menu may be defined by a sequence of up to 11 calls of this routine. Again, this merely stores the texts (representing the various program options) for later display.

The routine:

```
externalroutine DRAWMENU
```

or:

```
SUBROUTINE FDRMNU
```

draws a frame around the menu area and causes the texts defined for the header and menu options to be displayed within it. It is important to note that the routine calls both VWINDO (FVWNDO) and SWINDO (FSWNDO) in order to define the menu area; the virtual and screen windows must, therefore, be redefined by the graphics program subsequently.

It may be desirable to define alternative menus at different stages in the process of generating the required picture. The above routines may be employed repeatedly to replace one menu by another. In this situation, the routine:

```
externalroutine DELLINE(integer N)
```

or:

```
SUBROUTINE FDLLIN(N)
```

may also be used, to delete the Nth line from the menu. If N has a value of 0, the header is deleted. The effect is not evident until the menu is redrawn (by DRAWMENU or FDRMNU).

Example 8

The following example shows how the menu described in Figure 11 may be generated. A mechanism for interpreting the user response is suggested. The program draws a hexagon in the centre of the drawing area and displays the menu. The options permit the user to alter the view of the hexagon by moving the virtual window upward, downward, to the left or right, to magnify the picture by reducing the size of the virtual window or to terminate the program.

```
externalroutinespec TEKTYP(integer T)
externalroutinespec ERASE
externalroutinespec NEWPIC
externalroutinespec VWINDO(integer XO,YO,XM,YM)
externalroutinespec SWINDO(integer XO,YO,XM,YM)
externalroutinespec MOVEA(integer X,Y)
externalroutinespec DRAWR(integer DX,DY)
externalroutinespec MOVER(integer DX,DY)
externalroutinespec FRAME
externalroutinespec REVU
externalroutinespec RTAM
externalroutinespec PRINTSTRING(string(255) S)
externalroutinespec DEFHEAD(string(230) H)
externalroutinespec DEFMENU(string(9) M, integer N)
externalroutinespec DRAWMENU
externalroutinespec PROMPT(string(15) P)
```

```
begin
integer I, S, T, XO, YO, XM, YM
switch ACT(1:6)
constintegerarray CH(1:6)= 'U','D','L','R','Z','S'
TEKTYP(4010)
ERASE
NEWPIC
SWINDO(263,0,1023,760)
MOVEA(511,511)
MOVER(-50,-87)
DRAWR(100,0)
DRAWR(50,87)
DRAWR(-50,87)
DRAWR(-100,0)
DRAWR(-50,-87)
DRAWR(50,-87)
MOVER(50,87)
FRAME
DEFHEAD("CHOOSE YOUR ACTION
BY TYPING THE CODE
LETTER OPPOSITE THE
OPTION YOU REQUIRE
")
DEFMENU("UP    - U",1)
DEFMENU("DOWN  - D",2)
DEFMENU("LEFT  - L",3)
DEFMENU("RIGHT - R",4)
DEFMENU("ZOOM  - Z",5)
DEFMENU("STOP  - S",6)
XO=0; YO=0; XM=1023; YM=1023;      ! Initial virtual window
cycle
DRAWMENU
RTAM;                              ! to avoid corruption of prompt
PROMPT("SELECT")
READSYMBOL(S)
READSYMBOL(T) until T='
'
cycle I=1,1,6
->ACT(I) if S=CH(I)
repeat
PRINTSTRING("INVALID OPTION")
-> AGAIN
ACT(1): YO=YO+100; YM=YM+100;      ! Move virtual window upward
->VW
ACT(2): YO=YO-100; YM=YM-100;      ! Move virtual window downward
->VW
ACT(3): XO=XO-100; XM=XM-100;      ! Move virtual window left
->VW
ACT(4): XO=XO+100; XM=XM+100;      ! Move virtual window right
->VW
ACT(5): XO=XO+100; XM=XM-100;      ! Zoom in to smaller
YO=YO+100; YM=YM-100;              ! virtual window
VW: ERASE
VWINDO(XO,YO,XM,YM)
SWINDO(263,0,1023,760)
REVU
FRAME
AGAIN: repeat
ACT(6): ERASE
endofprogram
```

The FORTRAN version of this program might be written as follows:

```
      INTEGER I, S, T, XO, YO, XM, YM
      INTEGER CH(6) /'U','D','L','R','Z','S'/
      INTEGER HEAD(20) /'CHOO','SE Y','OUR ','ACTI','ON ',
     * 'BY T','YPIN','G TH','E CO','DE ',
     * 'LETT','ER O','PPOS','ITE ','THE ',
     * 'OPTI','ON Y','OU R','EQUI','RE '/
      DO 10 I=5,20,5
   10 HEAD(I)=HEAD(I)-22
C     CONVERT SPACE AT END OF EACH LINE TO NEWLINE
      CALL FTKTYP(4010)
      CALL ERASE
      CALL NEWPIC
      CALL FSWNDO(263,0,1023,760)
      CALL FMOVEA(511,511)
      CALL FMOVER(-50,-87)
      CALL FDRAWR(100,0)
      CALL FDRAWR(50,87)
      CALL FDRAWR(-50,87)
      CALL FDRAWR(-100,0)
      CALL FDRAWR(-50,-87)
      CALL FDRAWR(50,-87)
      CALL FMOVER(50,87)
      CALL FRAME
      CALL FDEFHD(HEAD,80)
      CALL FDFMNU('UP    - U',9,1)
      CALL FDFMNU('DOWN  - D',9,2)
      CALL FDFMNU('LEFT  - L',9,3)
      CALL FDFMNU('RIGHT - R',9,4)
      CALL FDFMNU('ZOOM  - Z',9,5)
      CALL FDFMNU('STOP  - S',9,6)
      XO=0
      YO=0
      XM=1023
      YM=1023
   11 CALL FDRMNU
      CALL RTAM
      CALL FPRMPT('SELECT',6)
      READ(5,501) S
  501 FORMAT(A1)
      DO 12 I=1,6
      IF(S.EQ.CH(I)) GOTO (1,2,3,4,5,6),I
   12 CONTINUE
      CALL FPSTRG('INVALID OPTION',14)
      GOTO 11
    1 YO=YO+100
      YM=YM+100
      GOTO 9
    2 YO=YO-100
      YM=YM-100
      GOTO 9
    3 XO=XO-100
      XM=XM-100
      GOTO 9
    4 XO=XO+100
      XM=XM+100
      GOTO 9
    5 XO=XO+100
      XM=XM-100
      YO=YO+100
      YM=YM-100
    9 CALL ERASE
      CALL FVWNDO(XO,YO,XM,YM)
      CALL FSWNDO(263,0,1023,760)
      CALL REVU
      CALL FRAME
      GOTO 11
    6 CALL ERASE
      STOP
      END
```

Contour Drawing

A common requirement in scientific research is to represent a function or distribution pictorially as an undulating surface. A number of methods are available to make this possible. One is to indicate the various levels or ranges of values of the distribution by contour lines, like an Ordnance Survey contour map. It may be used to represent any function of two independent variables. In the case of a relief map, the altitude at points within the area represented by the map is the function concerned and the Easting and Northing co-ordinates of these points are the independent variables.

The Edinburgh Tektronix Package provides a facility for contour drawing. It is based on the assumption that the value of the function under inspection is known at a number of points which are distributed evenly in a rectangular mesh over a unit square area (or some transformation of such an area). The points are identified by X- and Y-co-ordinates, where 0 <= X, Y <= 1 for the unit square; the user may specify what transformation he wishes applied to these co-ordinates to suit his problem.

Drawing is performed by the routine:

```
externalroutine TCONTOUR(longrealarrayname Z integer M,N c
                         longreal LVL,CHS routine TRANS)
```

or:

```
SUBROUTINE TCONTR(Z,M,N,LVL,CHS,TRANS)
```

The parameter Z represents a two-dimensional array, of M rows and N columns, which contains the values of the function under consideration, computed for N values of X and M values of Y. The rows and columns of the array Z are mapped on to the unit square as shown in Figure 12.

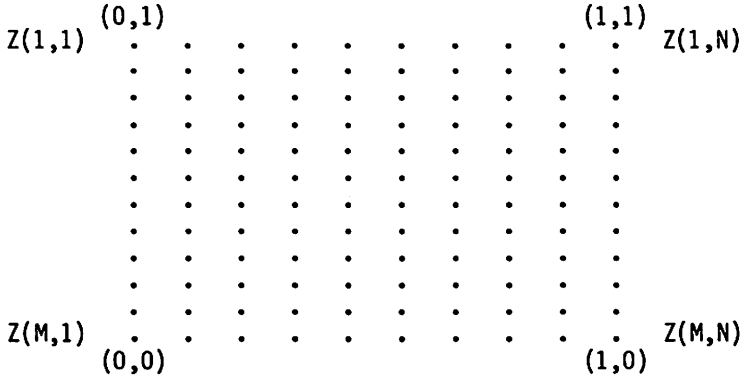


Figure 12. Function values in the unit square.

This implies that the values of X for which the function is known are 0, 1/(N-1), 2/(N-1), ... (N-2)/(N-1) and 1, while those of Y are 0, 1/(M-1), 2/(M-1), ... (M-2)/(M-1) and 1. The value of the function for the arbitrary point (I/(N-1), J/(M-1)) is held in the array element Z(M-J+1,I+1).

The scalar parameter LVL is the level at which the required contour is to be drawn (i.e. the value of the function which it represents). It must be noted that the routine draws only one contour at a time, and must, therefore, be invoked repeatedly (with different values for LVL) to produce a complete contour diagram.

CHS indicates whether numerical annotation of the contour is to be included (CHS > 0) or not (CHS <=0). Annotation is produced by displaying (in hardware characters) the nearest integer to the value of LVL at some point in the contour line.

The parameter routine TRANS must be supplied by the user, with the following specification:

```
routinespec TRANS(longrealname X,Y)
```

or:

```
SUBROUTINE TRANS(X,Y)
```

(where X and Y are DOUBLE PRECISION variables in the FORTRAN version.) Its function is to transform the co-ordinates X and Y from those of the unit square to those of the space defined by the user, returning the transformed values through the same parameters, X and Y. These values will be further transformed automatically by the Package into screen co-ordinates for display. Some transformation is necessary, since the unit square represents a tiny dot on the

screen. At least a scaling operation is required to produce an intelligible drawing. The user writing in FORTRAN should note that his transformation routine must be declared as EXTERNAL in the program invoking TCONTR.

It is important to note that the routine TCONTOUR or TCONTR is, in effect, simply a sequence of DRAW and MOVE operations, interspersed with certain annotation. This means that the normal sequence of calls on the administrative routines of the Package, (TEKTYP, NEWPIC, etc.) must be invoked before drawing may commence. It also implies that the appearance of the contour drawing is subject to transformations due to the virtual and screen windows, scaling and orientation defined at the time of drawing.

Example 9

The following program illustrates the use of the contour drawing facility. It accepts a rectangular array of data, representing values of a function under consideration. This array may be of any size, specified by the preceding data. Contours are drawn at any number of levels, which are defined by the data. The resulting contour drawing appears in an area of 600 screen units square at the lower left-hand corner of the screen.

```
externalroutinespec TEKTYP(integer N)
externalroutinespec ERASE
externalroutinespec NEWPIC
externalroutinespec MOVEA(integer X,Y)
externalroutinespec TCONTOUR(longrealarrayname Z integer M,N c
                        longreal LVL,CHS routine TRANS)
externalroutinespec PROMPT(string(I5) P)
begin
integer NX,NY,NL,I,J
routine SQUARE(longrealname X,Y)
X=600*X; Y=600*Y;    ! transform to fill square 600 units across
end
PROMPT("X POINTS?"); READ(NX)
PROMPT("Y POINTS?"); READ(NY)
PROMPT("LEVELS?"); READ(NL)
begin
longrealarray Z(1:NY,1:NX), HT(1:NL)
PROMPT("HEIGHT?")
cycle NL=1,1,NL
READ(HT(NL))
repeat
PROMPT("FN VALUES?")
cycle I=1,1,NY
cycle J=1,1,NX
READ(Z(I,J))
repeat
repeat
TEKTYP(4010)
ERASE
NEWPIC
MOVEA(0,0)
cycle NL=1,1,NL
TCONTOUR(Z,NY,NX,HT(NL),1,SQUARE)
repeat
end
endofprogram
```

The FORTRAN version of the program might be written as follows:

```
      EXTERNAL SQUARE
      INTEGER NX,NY,NL,I,J,L
      DOUBLE PRECISION Z(10000), HT(32)
      CALL FPRMPT('X PTS,Y PTS?',12)
      READ, NX, NY
      CALL FPRMPT('LEVELS?',7)
      READ, NL
      CALL FPRMPT('HEIGHT?',7)
      READ, (HT(L), L=1,NL)
      CALL FPRMPT('FN VALUES?',10)
      K=1
      L=(NX-1)*NY+1
      DO 1 I=1,NY
      READ, (Z(J), J=K,L,NY)
      K=K+1
    1 L=L+1
      CALL FTKTYP(4010)
      CALL ERASE
      CALL NEWPIC
      CALL FMOVEA(0,0)
      DO 2 L=1,NL
      CALL TCONTR(Z,NY,NX,HT(L),1,SQUARE)
    2 CONTINUE
      STOP
      END
      SUBROUTINE SQUARE(X,Y)
      DOUBLE PRECISION X,Y
      X=600.*X
      Y=600.*Y
      RETURN
      END
```

The following represents a sample set of data, in which the value of the function is known for 10 values of X and 20 values of Y and contours are to be drawn at all integral values of the function between -10 and +10.

```
10  20
21
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10
  0.0  0.7  1.3  1.6  2.1  2.5  3.1  3.6  3.9  4.2
 -0.8  0.2  0.7  1.4  2.2  2.8  3.4  4.0  4.7  5.1
 -1.2 -0.3  0.1  1.3  2.4  3.3  3.7  4.7  5.6  7.3
 -2.1 -1.1 -0.3  0.4  2.1  3.4  4.0  5.3  6.4  9.8
 -4.3 -2.2 -0.8  0.2  1.8  3.4  4.3  6.1  7.8 10.6
 -6.2 -3.1 -1.2 -0.8  1.4  3.4  4.2  5.6  7.4  9.8
 -8.3 -4.7 -2.1 -0.6  0.8  2.7  4.0  4.8  6.3  8.7
-10.1 -5.6 -3.4 -1.2  0.4  2.3  3.7  4.5  6.0  7.8
 -8.8 -4.3 -2.6 -0.8 -0.1  1.7  2.9  3.8  4.9  6.8
 -5.9 -2.7 -1.8 -0.5  0.0  0.9  1.7  3.0  4.2  5.7
 -3.9 -2.3 -1.1 -0.3  0.0  0.6  1.4  2.8  4.0  5.1
 -3.6 -2.0 -0.9 -0.1  0.0  0.4  1.2  2.3  3.6  4.2
 -2.8 -1.5 -0.5  0.0  0.0  0.2  0.7  1.8  2.5  3.7
 -1.9 -0.8 -0.3  0.0  0.0  0.1  0.5  1.1  2.0  2.9
 -1.1 -0.6 -0.2  0.0  0.0  0.0  0.3  0.8  1.4  2.2
 -0.9 -0.6 -0.1  0.0  0.0  0.0  0.1  0.3  0.9  1.8
 -0.4 -0.4 -0.0  0.0  0.0  0.0  0.0  0.1  0.5  0.8
  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.3  0.4
  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.1  0.2
  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

The effect of processing this set of data by the above program would be to display the drawing shown in Figure 13.

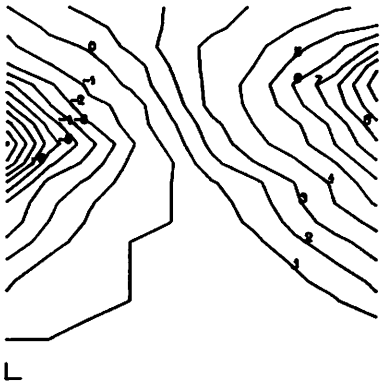


Figure 13. A contour drawing

Three-dimensional Drawing

The full potential of interactive graphics is made evident by the ability to draw a projection of a three-dimensional object and to modify the appearance of the projection, so as to give the impression of viewing the object from different directions, in proper perspective. The most sophisticated graphics systems permit continuous movement from one view to another, giving the impression of the object rotating in space, and are able to identify parts of the object which would normally be hidden from view and to eliminate them from the display; they may also be able to include differential shading of the various surfaces of the object, to lend considerable realism to the picture.

The Edinburgh Tektronix Package provides a simple three-dimensional viewing facility, without such sophistications. The picture appears as a 'wire-frame' drawing, with a perspective effect.

To use this facility, the user is permitted to consider that the object he is drawing exists in a 'real' world, of three dimensions; i.e. each point within it is referenced by X-, Y- and Z-co-ordinates, which may have values within any desired range. The user may then choose any point within the object as a focus for viewing; this point is termed the 'current origin' - it may be displaced from the actual origin of the three-dimensional drawing space, the point with co-ordinates (0,0,0). He may then choose any other point in the drawing space as his 'viewing position'. Having done so, he may use the projection facility to transform the salient points of his object into their equivalent co-ordinates on a plane surface (a hypothetical 'screen'), representing the 'virtual picture space' defined by the Package. He may then require to scale or otherwise transform these into co-ordinates for display on the real terminal screen by means of the windowing technique described above.

The routine:

```
externalroutine AIM(real VX,VY,VZ,CX,CY,CZ integername EF)
```

or:

```
SUBROUTINE FAIM(VX,VY,VZ,CX,CY,CZ,EF)
```

computes the transformation coefficients required to produce a three-dimensional perspective view of an object. The parameters VX, VY and VZ represent the co-ordinates of the 'viewing point' and CX, CY and CZ are those of the 'current origin' - some point within the object being viewed. EF is an error indicator, whose value is zero if the transformation can be computed and non-zero otherwise. This arises when the X- and Z-co-ordinates of the current origin and viewing point coincide. The computed transformation coefficients are not directly accessible to the user; they are stored for subsequent use.

The routine:

```
externalroutine PROJECTION(integer X,Y,Z integername XE,YE,EF)
```

or:

```
SUBROUTINE FPRJCT(X,Y,Z,XE,YE,EF)
```

applies the transformation computed by AIM to a point within the object, whose co-ordinates are (X,Y,Z). It computes the co-ordinates (XE,YE) of the projection of that point on to the plane of the virtual picture space. The parameter EF is an error indicator, whose value is zero if no error arises and non zero if the projection fails. Failure occurs if any point within the object coincides with, or lies behind, the viewing point.

Example 10

The following program illustrates the use of the three-dimensional drawing facility. It accepts a sequence of vector specifications, representing drawing and moving operations in three-dimensional space, relative to the current position in that space, similar to the two-dimensional operations of DRAWR and MOVER provided by the Package. Each vector is defined by four integers. The first is either 1, representing a visible line, or 0, representing an invisible movement. The remaining three values represent the increments in the X-, Y- and Z-co-ordinates defining the vector which is to be traversed, relative to the current position in 'real' space. It is assumed that drawing commences initially at the origin. The sequence of vectors is terminated by the dummy vector:

```
0 0 0 0
```

The relative vectors so defined are first transformed into absolute co-ordinates in the three-dimensional space and are stored. Subsequently, the program requests the co-ordinates of the user's 'viewing position', assuming that the 'current origin' for the view is the actual origin of the real space - the point (0,0,0). The appropriate transformations are applied and a perspective view of the object is displayed. Any sequence of views is permitted, terminated by the specification of a viewing position at the origin (0,0,0).

```
externalroutinespec TEKTYP(integer T)
externalroutinespec ERASE
externalroutinespec NEWPIC
externalroutinespec VWINDO(integer XO,YO,XM,YM)
externalroutinespec SWINDO(integer XO,YO,XM,YM)
externalroutinespec MOVEA(integer X,Y)
externalroutinespec DRAWA(integer X,Y)
externalroutinespec RTAM
externalroutinespec PRINTSTRING(string(255) S)
externalroutinespec TWRITE(integer I,N)
externalroutinespec NLINE
externalroutinespec AIM(real VX,VY,VZ,CX,CY,CZ integername EF)
externalroutinespec PROJECTION(integer X,Y,Z integername XE,YE,EF)
externalroutinespec PROMPT(string(15) P)
begin
integerarray DORM,VX,VY,VZ(1:1000)
real XC,YC,ZC
integer I,NP,D,DX,DY,DZ,X,Y,Z,XS,YS
X=0; Y=0; Z=0                ; !  Start at origin
cycle NP=1,1,1000
PROMPT("VECTOR?"); READ(D); READ(DX); READ(DY); READ(DZ)
exitif D=DX=0 and DY=DZ=0    ; ! dummy vector terminator
DORM(NP)=D                   ; ! store vector type
X=X+DX; VX(NP)=X             ; ! compute absolute position and store
Y=Y+DY; VY(NP)=Y
Z=Z+DZ; VZ(NP)=Z
repeat
NP=NP-1                      ; ! discount dummy vector
TEKTYP(4010)
cycle
RTAM                         ; ! to avoid corruption of prompt
PROMPT("EYE COORDS?"); READ(XC); READ(YC); READ(ZC)
exitif XC=YC=0 and ZC=0      ; ! terminator
AIM(XC,YC,ZC,0,0,0,I)        ; ! view to origin
if I≠0 start
PRINTSTRING("AIM FAILS "); TWRITE(I,2); NLINE; stop
finish
ERASE; NEWPIC
```

```
VWINDO(-256,-256,255,255)
SWINDO(0,0,780,780)
cycle NP=1,1,NP
D=DORM(NP); X=VX(NP); Y=VY(NP); Z=VZ(NP)
PROJECTION(X,Y,Z,XS,YS,I); ! convert to virtual co-ordinates
if I≠0 start
PRINTSTRING("PROJECTION FAILS "); TWRITE(I,2); NLINE; stop
finish
if D=0 then MOVEA(XS,YS) else DRAWA(XS,YS)
repeat
MOVEA(-200,-200);          ! avoid text interfering with drawing
repeat
ERASE
endofprogram
```

The following is an equivalent program in FORTRAN:

```
      INTEGER DORM(1000),VX(1000),VY(1000),VZ(1000)
      REAL XC,YC,ZC
      INTEGER I,N,NP,D,DX,DY,DZ,X,Y,Z,XS,YS
      X=0
      Y=0
      Z=0
      DO 1 NP=1,1000
      CALL FPRMPT('VECTOR?',7)
      READ, D,DX,DY,DZ
      IF(D.EQ.0 .AND. DX.EQ.0 .AND. DY.EQ.0 .AND. DZ.EQ.0) GOTO 2
      DORM(NP)=D
      X=X+DX
      VX(NP)=X
      Y=Y+DY
      VY(NP)=Y
      Z=Z+DZ
    1 VZ(NP)=Z
      NP=1001
    2 NP=NP-1
      CALL FTKTYP(4010)
    3 CALL RTAM
      CALL FPRMPT('EYE COORDS?',11)
      READ, XC,YC,ZC
      IF(XC.EQ.0 .AND. YC.EQ.0 .AND. ZC.EQ.0) GOTO 7
      CALL FAIM(XC,YC,ZC,0,0,0,I)
      IF(I.EQ.0) GOTO 4
      CALL FPSTRG('AIM FAILS ',10)
      CALL FWRITE(I,2)
      CALL NLINE
      STOP
    4 CALL ERASE
      CALL NEWPIC
      CALL FVWNDO(-256,-256,255,255)
      CALL FSWNDO(0,0,780,780)
      DO 6 N=1,NP
      D=DORM(N)
      X=VX(N)
      Y=VY(N)
      Z=VZ(N)
      CALL FPRJCT(X,Y,Z,XS,YS,I)
      IF(I.EQ.0) GOTO 5
      CALL FPSTRG('PROJECTION FAILS ',17)
      CALL FWRITE(I,2)
      CALL NLINE
      STOP
    5 IF(D.EQ.0) CALL FMOVEA(XS,YS)
      IF(D.NE.0) CALL FDRAWA(XS,YS)
    6 CONTINUE
      CALL FMOVEA(-200,-200)
      GOTO 3
    7 CALL ERASE
      STOP
      END
```

The following sample data represents a drawing of a 'tasseract' - a cube contained within another cube, with their corresponding edges joined. The inner cube is of side 100 units and the outer one of 200 units. Both are centred on the origin. A single view is defined, from the point (128,256,512).

```
0  -50   50   50
1  100    0    0
1    0 -100    0
1   50  -50   50
1    0  200    0
1  -50  -50  -50
1    0    0 -100
1 -100    0    0
1  -50   50  -50
1    0    0  200
1  200    0    0
1    0    0 -200
1 -200    0    0
1    0 -200    0
1   50   50   50
1    0  100    0
1    0    0  100
1  -50   50   50
1    0 -200    0
1    0    0 -200
1  200    0    0
1    0  200    0
1  -50  -50   50
1    0 -100    0
1 -100    0    0
1    0    0  100
1  -50  -50   50
1  200    0    0
1    0    0 -200
1  -50   50   50
1    0    0  100
1 -100    0    0
1    0  100    0
0    0    0    0

128  256  512
  0    0    0
```

The resultant display would be as shown in Figure 14.

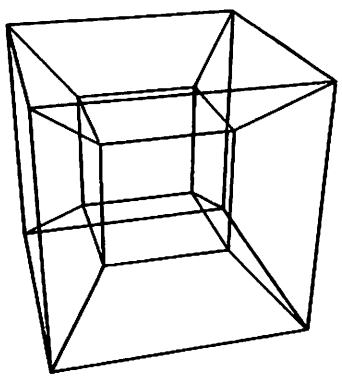


Figure 14. A three-dimensional projection.

Storage and Viewing of Pseudo-display Files

When a drawing has been constructed, using the interactive graphics facilities described in the previous sections, the user may find it desirable to make a permanent record of the resulting display.

The routine:

```
externalroutine SNAPSHOT(string(24) F)
```

or:

```
SUBROUTINE FSNAP(F,L)
```

copies the current content of the pseudo-display file maintained by the Package, to a Store-Map file whose name is F (of L characters in length, in the FORTRAN version). The content of this file will include whatever window transformations apply to the current display. The routine may be invoked repeatedly, thus allowing different views of the picture to be stored.

It must be noted that the form of the file so stored is some what different in the implementation on EMAS 2970 from that of the System 4 version, and that consequently transmission of pseudo-display files between these implementations is not recommended.

For example, the final picture produced by the programs listed in Example 10 could be recorded by including the specification:

```
externalroutinespec SNAPSHOT(string(24) F)
```

at the head of the IMP version and preceding the final call of the routine:

```
ERASE
```

by the call:

```
SNAPSHOT("CUBES")
```

or:

```
CALL FSNAP('CUBES',5)
```

in the IMP and FORTRAN versions respectively. The effect would be to create a store-map file called 'CUBES', containing the drawing.

A permanently stored pseudo-display file may be reviewed by means of the EMAS foreground command:

```
TEKVIEW
```

This command first requests:

```
Tektronix No:
```

to which the user is expected to respond with the model number of the terminal in use (4002, 4006, 4010, 4012 or 4014, or 999 for a terminal controlled by a Sigma GOC). The enhanced graphics option is enabled automatically if the model 4014 terminal is in use.

Then the program prompts:

```
Tekfile Name:
```

to which the user should type the name of the store-map file containing his pseudo-display file copy.

The drawing is displayed, together with a menu including the following options:

```
NEWFILE_F      (to select another pseudo-display file)

WINDOW__W      (to select a window, using the cursor)

RETURN__R      (to review the original picture)

STOP____S      (to terminate the display)
```

The prompt issued is:

```
SELECT
```

On completion, the screen is cleared.

Graph Plotter Transcription

Once a permanent pseudo-display file has been created, the user may wish to transcribe its content into a form suitable for hard-copy drawing on a graph plotter or similar device.

This requirement is catered for in the Package by the EMAS foreground command:

CREATE GPFILE

The program first prompts:

TEKFILENAME?

to which the user responds with the name of his permanent pseudo-display file.

Next, the prompt:

PLOTTER FILE?

is offered. The user should type the name of a sequential file, comprising 80-byte formatted records, into which he wishes the display to be transcribed, in the form of an ERCC standard graph plotter file.

The prompt:

DELIVERY INFO?

requests that the user types his name and address for delivery of the graph plotter output.

The prompt:

SIZE (CM)?

requests the width, between 1 and 82.5 centimetres, of the SETPLOT area within which the drawing is to be plotted.

The program then issues the request:

```
PEN COLOUR....1=BLACK:2=BLUE:3=GREEN:4=RED
1:2:3:4?
```

to which the user should respond with the appropriate colour code.

Then the program issues the prompt:

DASHING(Y OR N)?

to which the user should reply:

Y

if he wishes the drawing to comprise dashed lines, or:

N

if he does not. In the former case, the prompt:

DASH 1:2:3?

requests the user to specify the dashing pattern he requires:

1 - Long dashes and gaps

2 - Medium dashes and short gaps

3 - Short dashes and gaps

The prompt:

```
OVERPLOT(Y,N)?
```

expects the user to reply:

```
Y
```

if he wishes a further pseudo-display file (which he is subsequently required to name) plotted within the same SETPLOT area as the first; this may be useful where one pseudo-display file is a standard set of axes and the other is a graph to be superimposed. The alternative response is:

```
N
```

if no overplotting is required.

The prompt:

```
ANOTHER SETPLT?
```

expects the response:

```
Y
```

if the user wishes to transcribe a further pseudo-display file in a separate SETPLOT area on the graph plotter. The name of that file is requested. The user should respond:

```
N
```

if no further file is to be plotted.

The program continues to request:

```
OVERPLOT(Y,N)?
```

and:

```
ANOTHER SETPLT?
```

until both responses are:

```
N
```

at which point the program terminates. A complete graph plotter file is then ready to despatch to the graph plotter, by means of the command:

```
LIST(filename,.GP)
```

Viewing Graph Plotter Files

Before committing to hard copy a graph plotter file created in the fashion described above, or by conventional use of the Graph Plotter Package (q.v.), the user may wish to check the contents of the plotter file he has created. This is best achieved by invoking the command:

```
TVIEW
```

which may be used to display selectively, on a Tektronix or similar terminal, drawings stored in various SETPLOT segments of such a file.

The program commences by isssuing the prompt:

```
Tektronix type?
```

to which the user should respond by typing the model number (4002, 4006, 4010, 4012, 4014 or 999) of the terminal in use.

The program reminds the user to enter Graph Mode before proceeding. It then issues the prompt:

```
Help Info?
```

The user may respond:

```
Y
```

if he wishes explanatory information to precede each subsequent prompt, or:

```
N
```

if he is more experienced in the use of the program.

The prompt:

```
File name?
```

requests the name of a sequential file of 80-byte formatted records, containing a specification of a drawing in ERCC Graph Plotter File Standard format, i.e. as produced by GRAPHPACK (q.v.) or CREATE GPFILE.

The program then issues the message:

```
Max setplot size in centimetres/inches
```

(where the units specified are appropriate to the file named), followed by the prompts:

```
Length:
```

and:

```
Height:
```

to which the user should reply with the appropriate dimensions of the largest SETPLOT area in the file being viewed.

The prompt:

```
Which setplot?
```

then requests the number (1,2,3, etc.) of the first SETPLOT area which the user wishes to select from the file for display. If that area exists, it is drawn on the screen, together with a menu including the following options:

```
NEWVIEW_A      (return to original display of current setplot)
NEXTSP__N      (display next setplot area in current file)
WINDOW__W      (window on current display, using cursor)
LOOK____L      (review current setplot after windowing)
ANYSP___I      (choose any other setplot area in this file)
RESET___R      (review first setplot area in this file)
NEWFILE_F      (view a different plotter file)
HELP____H      (request help information)
STOP____S      (terminate program)
```

The prompt issued is:

```
SELECT
```

The program continues to respond to menu selections until the option:

```
S
```

is selected.

Error Messages

The following errors may be reported when calling routines in the Package:

FAULT 1 ATTEMPTING TO DRAW OUTSIDE VIRTUAL PICTURE

The co-ordinates of the current virtual position have been defined outside the range -32768 to +32767.

FAULT 2 SCREEN WINDOW CO-ORDINATES OFF SCREEN

The screen window limits have been defined outside the range 0 to 1023, or 0 to 4095 in enhanced graphics mode.

FAULT 3 ATTEMPTING TO CLOSE NON-EXISTANT SUBPIC DEFINITION

The routine ENDSUB has been called without previously invoking DEFSUB.

FAULT 4 SUBPICTURE NOT DEFINED

The routine INSTAN or FINST has attempted to instance a subpicture which has not been defined by DEFSUB or FDEFSB.

FAULT 5 REFLECTION PARAMETERS OUT OF RANGE

Reflection parameters in call of INSTAN or FINST must be 0 or 1 they have been defined outside this range.

FAULT 6 PSEUDO DISPLAY FILE SPACE EXCEEDED

The pseudo display file has a capacity of up to 40000 vectors only; an attempt has been made to exceed this.

FAULT 7 SUB-PICTURE ALREADY DEFINED

Two calls of DEFSUB or FDEFSB have nominated the same sub-picture identifier.

FAULT 8 MULTIPLICATION OVERFLOW

A scaling factor has been defined in INSTAN, FINST, CHARS, BIG STRING or FBGSTR which results in drawing beyond the virtual picture limits.

FAULT 9 SUB-PICTURE NAME TABLE OVERFLOW

An attempt has been made to define more than 128 different subpictures.

All these faults result in the program terminating.

Summary

The following reference list of the contents of the Edinburgh Tektronix Interactive Graphics Package is included for the convenience of more experienced users.

In each case, the name of the IMP version of the routine is given first, with its parameter specification, and the FORTRAN version follows if its name or parameter specification is different. All INTEGER type parameters are of 4-byte length and strings are passed in FORTRAN by giving the name and size of an A4-format INTEGER array or Hollerith string.

Routine	Parameters	Description
TEKTYP FTKTYP	(integer TYPE) (TYPE)	Sets the Tektronix model in use (4002,4006,4010,4012,4014,999).
ERASE		Erases picture from screen.
NEWPIC		Clears and initialises pseudo-display file for new picture. Calls VIEWON and STOREON. Sets default values for windows (0,0,1023,1023). Sets default values for scale, reflections and rotations.
VIEWON FVWON		Turns on incremental viewing.
VIEWOFF FVWOFF		Turns off incremental viewing (vectors are stored in pdf, but not shown on screen).
STOREON FSTON		Enables storage to pdf.
STOREOFF FSTOFF		Disables storage to pdf.
VWINDO FVWINDO	(integer VXO,VYO,VXM,VYM) (VXO,VYO,VXM,VYM)	Sets virtual window with origin (VXO,VYO) and top right-hand corner (VXM,VYM).
SWINDO FSWINDO	(integer SXO,SYO,SXM,SYM) (SXO,SYO,SXM,SYM)	Sets screen window with origin (SXO,SYO) and top right-hand corner (SXM,SYM).
FRAME		Draws a frame for current screen window.
DRAWA FDRAWA	(integer X,Y) (X,Y)	Draws a line from current virtual position to the point (X,Y).
MOVEA FMOVEA	(integer X,Y) (X,Y)	Moves to virtual position (X,Y).
POINTA FPNTA	(integer X,Y) (X,Y)	Moves to virtual position (X,Y) and draws a dot.
DRAWR FDRAWR	(integer DX,DY) (DX,DY)	Draws a line from current virtual position (VX,VY) to point (VX+DX,VY+DY).
MOVER FMOVER	(integer DX,DY) (DX,DY)	Moves to virtual position (VX+DX,VY+DY).
POINTR FPNTR	(integer DX,DY) (DX,DY)	Moves to virtual position (VX+DX,VY+DY) and draws a dot.
DEFSUB FDEFSB	(integer NAME) (NAME)	Starts sub-picture definition. Calls VIEWOFF. Calls ENDSUB if last sub-picture not closed.
ENDSUB		Ends subpicture definition. Calls VIEWON if display was enabled prior to last call of DEFSUB.

Routine	Parameters	Description
INSTAN FINST	(integer NAME,SIZE,ROT, XREF,YREF) (NAME,SIZE,ROT,XREF,YREF)	Instances a sub-picture with scale factor, rotation and reflections specified.
CHARS	(integername CHAR,ROT,CX, CY,SCALE)	Draws a software character at current position with rotation and scale, stores it in the pdf.
BIG STRING FBGSTR	(string(255) S, integer ROT,SIZE) (A,I,ROT,SCALE)	Draws software character string with rotation and scale, storing it in the pdf.
RTAM		Sets terminal into alphanumeric mode.
ALPHAPOS FALPOS	(integername X,Y) (X,Y)	Returns the virtual position of the alphanumeric cursor.
PRINT SYMBOL FPSYM	(integer CHAR) (CHAR)	Sets terminal into alphanumeric mode if necessary and transmits a hardware character. Does not store it in the pdf.
PRINT STRING FPSTRG	(string(255) S) (A,I)	Sets terminal into alphanumeric mode and sends a string of hardware characters. Does not store them in the pdf.
NLINE		Sets terminal into alphanumeric mode if necessary, and transmits a newline.
CHAR FCHAR	(integer CVAL) (CVAL)	Displays a hardware character at current virtual position. Moves 12 units in X-direction.
DRAW TEXT FDRTXT	(string(255) S) (A,I)	Prints a hardware character string and stores it in the pdf.
TWRITE FWRITE	(integer NO, PLACES) (NO,PLACES)	Prints integer as hardware characters and stores it in the pdf.
TPRINT FPRINT	(real NO, integer PLACES, DEC PLACES) (NO,PLACES,DEC PLACES)	Prints number as hardware characters in fixed point form and store it in the pdf.
DEFHEAD FDEFHD	(string(230) HEAD) (A,I)	Defines header text for menu.
DEFMENU FDFMNU	(string(9) T, integer L) (A,I,L)	Defines menu line option text.
DELLINE FDLLIN	(integer L) (LINE)	Deletes option line from menu.
DRAWMENU FDRMNU		Draws previously defined menu.
CURSOR	(integername CVAL,X,Y)	Sets cross-hairs on screen. Returns the cursor virtual position and identifying symbol upon user response.
TCONTOUR FTCNTR	(longrealarrayname Z, integer M,N, longreal LVL,CHS, routine TRANS) (Z,M,N,LVL,CHS,TRANS)	Draws a single contour line, representing a level of a function of two variables.
AIM FAIM	(real VX,VY,VZ,CX,CY,CZ, integername EF) (VX,VY,VZ,CX,CY,CZ,EF)	Computes transformations for a three-dimensional view.
PROJECTION FPRJCT	(integer X,Y,Z, integername XE,YE,EF) (X,Y,Z,XE,YE,EF)	Applies computed transformations to a single point within a three-dimensional object.

Routine	Parameters	Description
REVU		Regenerates current picture, applying current windows.
SNAPSHOT FSNAP	(string(20) FILE) (A,I)	Stores the pdf in a permanent file.

The following four routines apply to the model 4014 terminal only.

ADVGPH FADVG	(integer I) (I)	Sets enhanced graphics mode if I=1, otherwise clears mode.
SETLINE FSTLN	(integer I) (I)	Sets type of following vectors: I = 0 Normal I = 1 Dotted I = 2 Dot-dashed I = 3 Short-dashed I = 4 Long-dashed
SPOINTA FSPNTA	(integer X,Y,I) (X,Y,I)	Moves to virtual position (X,Y) and draws a dot with intensity 1 <= I <= 62.
SPOINTR FSPNTR	(integer DX,DY,I) (DX,DY,I)	Moves to virtual position (VX+DX,VY+DY) and draws a dot with intensity 1 <= I <= 62.

The following foreground commands are available on all terminals.

TEKVIEW	Displays a permanent pdf file created by SNAPSHOT.
CREATE GP FILE	Converts a permanent pdf file created by SNAPSHOT into an ERCC standard graph plotter file.
TVIEW	Displays an ERCC standard graph plotter file.

INDEX