REPORT NO.
UCB/SESM-83/09

STRUCTURAL ENGINEERING AND
STRUCTURAL MECHANICS

GENGRAF: A GENERAL PURPOSE SUBROUTINE PACKAGE FOR COMPUTER GRAPHICS

by

R. GORDON OLIVER

DECEMBER 1983

DEPARTMENT OF CIVIL ENGINEERING
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA

GENGRAF
A GENERAL PURPOSE SUBROUTINE PACKAGE FOR COMPUTER GRAPHICS

R. Gordon Oliver

Department of Civil Engineering.
University of California.
Berkeley,
California 94720.

ABSTRACT

GENGRAF is a general purpose subroutine package for generating graphics on computer controlled displays. The package is device independent in that it can create graphics for any number of different display devices. This report explains the general philosophy underlying the proposed use of the package and also details the necessary calling sequences.

November 28, 1983

Table of Contents

II. ACKNOWLEDGEMENTS.

I would like to thank Professor K. S. Pister for providing the opportunity for this work to be undertaken, and for his help and advice. I would also like to thank Professors R. L. Taylor and E. Polak for their advice and suggestions. In addition many other people conveyed their comments and suggestions with respect to both the package and this report, which have been most helpful, especially Chris Thewalt and Joe Landers. Finally, I wish to thank Ken Keller and Giles Billingsley for the use of their Model Frame Buffer graphics device driver package which GENGRAF uses.

This work was supported by the National Science Foundation under Grant PFR-7908261 with the University of California, Berkeley. Computing facilities were provided in part by Equipment Grant ENG-7810992 from the National Science Foundation.

THE GENGRAF GRAPHICS PACKAGE

1. OVERVIEW

There is a strong case for using a standard graphics subroutine library in order to proceed along the path towards producing software which facilitates fast and logically controlled graphics. Such a package would be useful for a number of reasons :

(1) To have a standard set of graphics routines which any application program may utilize.

(2) To segregate the graphical control section of a program from the other operational sections.

(3) To facilitate the use of programs using graphics on a number of different display devices. That is, to achieve a high degree of *device-independence*, both for existing hardware and future additions.

(4) To have the capability of producing *hardcopy* (printed graphical output) without any additional programming.

One standard graphics system which has been proposed and which has had a number of extensive and successful implementations is that of the Special Interest Group on Graphics of the Association for Computer Machinery (SIGGRAPH-ACM). They have established a Graphic Standards Planning Committee which has produced a number of reports on the proposed form of a standard graphics system, which is known as the CORE system [1].

Therefore it was decided to follow the SIGGRAPH outline proposal in attempting to design a graphics package for use in structural engineering applications. The CORE system defines a number of different operation levels, ranging from basic display with no interaction to a more sophisticated and highly interactive level. The CORE system proposal is very comprehensive and the full implementations that have been reported have involved an investment of much time and many programmers [2]. Thus, the GENGRAF graphics package described here is not intended to be a full implementation of the CORE system, but one which provides the basic capabilities and has sufficient flexibility to be extended in the future, without major revisions.

GENGRAF is a *general purpose graphics utility package*. It consists of approximately one hundred routines, which are written in the C programming language [3] and run under the UNIX operating system. The user callable routines are explained in the user manual in Appendix 1. GENGRAF provides routines for the display of two- and three-dimensional graphical primitives in a coordinate system which is under user or application program control.

2. ORGANIZATION OF A GRAPHICS PACKAGE

A graphics package has different types of functions for dealing with the different aspects of graphics control. In the case of the GENGRAF package these can be summarized as follows:

(1) Starting and Finishing.
These routines are used to initialize the graphics device and the variables associated with the package and to close the device and de-allocate storage associated with it.

(2) Viewports and Windows.
These are terms used to describe areas of the device display and regions of the users coordinate space being viewed. A group of routines exist to allow user control of all their associated parameters.

(3) 3D View Control.
If the user space is three-dimensional (3D) a number of parameters can be varied so as to control the resultant view.

(4) Clipping Control.
Parts of an object which would be displayed outside of a certain area of the display (known as the viewport) can be removed or clipped if desired.

(5) World Transformations.
Objects specified within the users (world) coordinate system can have transformations applied to them.

(6) Controlling Attributes.
The visible attributes such as color and pattern types can be specified.

(7) Displaying Primitives.
A primitive is a graphical entity that can displayed with a single function invocation, such as a line, polygon or a string of text.

(8) Coordinate Conversions.
A number of routines exist to convert coordinates from between the various coordinate systems associated with the graphics package.

(9) Inquiry.
Inquire about certain values held within the graphics package.

(10) General Control.
These are routines that do not fit into any other category.

Having outlined the different aspects of a graphics package each one will now be considered in more depth.

2.1. STARTING AND FINISHING

These are a group of routines concerned with opening and closing operations. The *GENOpen* and *GENClose* routines open and close the GENGRAF package and must be called as the first and last routines, respectively. Also included in this group are the routines to open and close individual viewports, as well as a routine to set a particular viewport as the *current* or *working* viewport. These routines are *GENOpenView, GENCloseView* and *GENSetCurView,* respectively. As previously explained a viewport is a section of the device display area which has a window into a world coordinate system mapped onto it. A viewport can extend over the full area of the display surface or just a fraction of it. At any given time more than one viewport can exist on a device display area. Viewports can also be 'stacked up' on top of each or overlapped as the user requires. In this case the application program would control which viewports display information.

2.2. VIEWPORTS AND WINDOWS

Two of the most essential concepts of this graphics system are the *viewport* and *window*. A *viewport* is an area of the display device surface; that is, a section of the screen of a CRT device or a section of the plotting surface of a pen plotter. A *window* is a region of the space, known as the world space or coordinate system, in which the user is modelling an object or phenomenon for display and possibly other purposes. A reader not familiar with these basic components of a computer graphics package may wish to consult a text on the subject [6]. The graphics package performs the operations necessary to project an image of what appears within the bounds of the world space window onto the corresponding device viewport.

Coordinate Systems

GENGRAF provides the capabilities to model both two-dimensional (2D) and three-dimensional (3D) world coordinate spaces. For the purpose of being able to identify points and regions and to be able to define transformations within the world space, it is necessary to define a coordinate system with which to relate. This also applies to the display surface of the graphics device. A coordinate system has to be defined in order to locate viewports. In the case of the display surface a two-dimensional Cartesian system is most appropriate, since display surfaces usually consist of a rectangular area. In this case the x-axis is taken to be horizontal with values increasing from left to right. The y-axis is taken as being vertical with values increasing from bottom to top. The coordinate values are normalized in both x and y directions with values ranging from 0 to 1. These normalized coordinates are known as the *normalized device coordinates* (NDC).

The world coordinate system is also a Cartesian system of either two or three dimensions. A 2D world coordinate system is similar to that of the display surface and viewports. A 3D world coordinate system is a 'right-handed' system, with the x and y axes lying in the horizontal plane and the z axis perpendicular to this plane, values increasing in the upwards direction. A window into a 2D world system is simply a rectangular area orthogonal to the x and y axes. The window area is mapped onto the viewport area, thus if the viewport and window have different aspect ratios (ratio of height to width) then the scaling will be different in x and y directions. If clipping is in effect, everything within the window will be displayed and anything that falls outside will be hidden from view. If clipping is not in effect and primatives are displayed which are partially or wholly outside of the window are displayed, the result is device dependent and in the worst cases can be expected to be "messy".

The window into a 3D world system is not so straightforward, since a transformation from three-dimensional space to two-dimensional space is required. The best way to conceive the region of the three-dimensional world system that will be displayed on the viewport is to imagine an observer sitting in a room, looking out through a window in one of the walls onto a three-dimensional world. If a Cartesian coordinate system was established with a reference origin, then the observer would be located at a three-dimensional point (consider this point to be your eye). All other points in the world would also have unique coordinate triplets. If the projection of everything the observer could see through the window was to be traced onto the window itself, then this window would be like the display surface we attempt to simulate when displaying a 3D world. Everything that is displayed falls within a hypothetical, semi-infinite pyramid which has its apex at the eye of the observer. The four edges of this pyramid extend from the eye of the observer through each of the corners of the rectangular window area. Thus, if an object lies outside of this pyramid then it will not be visible. If objects lie across the boundaries of this pyramid and if clipping is in effect, then only the parts within the pyramid will be displayed.

2.3. 3D VIEW CONTROL

The control over the area of the 3D world viewed through this window is maintained by a number of parameters. One of the most basic is the size of the window. This could be specified so as to produce a distorted view, if the window aspect ratio is different from that of the viewport. However, this situation should be avoided. The actual location of the observer (or more precisely the eye of the observer) has to be specified by a Cartesian triple. This is referred to as the *view reference point*. Also the direction of the view has to given. This is specified in terms of a three-dimensional vector which gives the direction of the view relative to the view reference point. This vector is called the *view direction vector.* One can also imagine the observer being able to twist his or her head around an axis projecting in the viewing direction. This would effect the direction that they perceived to be *up.* Thus the *view up angle* can also be specified by the user; this angle is measured relative to the direction of maximum increasing z values, assuming the x and y axes form a horizontal plane.

Finally, one other parameter can affect the projection of the world system onto the hypothetical window, this is the distance of the window (or projection plane) from the observer. This distance effects the amount of perspective that is perceived, if a perspective projection has been selected. This will not be the case if an oblique or isometric projection has been specified. The perspective projection can, if used appropriately give a more realistic image of the three-dimensional world on a two-dimensional surface than can the other projections. Therefore, four parameter sets control the view of a 3D system. Some graphics systems define viewing parameters such as the angle of the *cone of vision* to control a view. However, this is simply the angle of the hypothetical viewing pyramid and so can be controlled by the size of the window on the projection plane and the distance of this plane from the view reference point.

These routines allow the user to control viewport and window parameters. The extent of the viewport is specified in normalized device coordinates with *GENSetViewport* routine. The dimensionality of the world system is set with *GENSetDimension,* this being either 2D or 3D. The window size is set by the *GENSetWindow* routine. In the case of a three-dimensional world coordinate system, other routines which effect the view are *GENSetViewPoint* (set the view reference point), *GENSetViewDir* (set the view direction vector), *GENSetViewUp* (set the view up angle), and *GENSetViewDist* (set the distance to the projection or window plane).

2.4. CONTROL OF CLIPPING

There are two types of clipping that can be invoked within this package. One is *window clipping,* where all objects are checked to see if they fall either totally or partially outside of the window region. This can be done for a 2D or 3D world system. If clipping is in effect, an object which has parts lying outside of the window will be clipped to the window so that those parts are not displayed. However, if the user is certain that everything will fall inside the window, then window clipping could be turned off. This is less costly in terms of computer processing time. Window clipping is turned on or off with the *GENSetWindClip* routine.

In the case of a 3D world system, it is also possible to invoke *depth clipping.* In this case two planes are defined which are perpendicular to the viewing direction. The plane nearest the view reference point is known as the *front clipping plane* and the one furthest away as the *back clipping plane.* Clipping can be turned on with respect to either or both of these planes so as to produce the image of a semi-infinite world or a slice from the world system. The positions of the clipping planes is set with the *GENSetViewDepth* routine, and clipping with respect to these planes is turned on or off with the *GENSetFrontClip* and *GENSetBackClip* routines.

2.5. WORLD TRANSFORMATIONS

There are two different types of transformations identified within the GENGRAF package. These are :

(i) world or model transformations

(ii) viewing transformations.

A world or model transformation is one which maps a point in the world system to another point in the world system. The different world transformation types are scalings, translations and rotations. A viewing transformation is one which maps a point or region of the world system onto a point or region in a coordinate system which has its origin at the eye of the observer. From this coordinate system the mapping onto the viewport and thus display surface is calculated. For a 2D world system the eye coordinate system is not such a relevant model. In this case the viewing transformation includes scalings and translations to map the window onto the viewport. For a 3D world system the viewing transformations will include scalings, translations, rotations and some kind of projective transformation, such as a perspective transformation.

The user can control the world system transformation directly by calling routines with the appropriate parameters, however the viewing transformations are handled by the package. The user specifies the viewing parameters as explained above.

The world transformation routines define the values of a set of transformation parameters that will be applied to all geometric primitives defined in the world coordinate system. Therefore, all primitives displayed undergo the current transformation in the world coordinate system before being subjected to the current viewing transformation, to map the resultant world coordinates to the viewport and finally to the device coordinates.

World transformations can be defined to act on 'top of each other'; for example, a translation could be specified, followed by a rotation, followed by a scaling. The resultant transformation would be the combination of all of these individual transformations. Therefore the order in which they are specified is significant. Different transformations can result from combining the same component transformations in different ordered sequences. The internal mechanism used for combining the transformations is a *stack*. Each new transformation specified is combined with all the previous transformations and the new resultant transformation is placed at the top of the stack. This provides the capability to *undo* all of the individual transformations, in reverse order, to give any of the previous intermediary transformations. Also the world transformation can be reset to the identity transformation by emptying the stack altogether.

The principles of the world transformation system were explained in the previous section. The actual routines consist of scaling, translation and rotation calls for both two and three dimensions, as well as a routine to undo the last transformation specified and a routine to reset the resultant transformation to the identity transformation (ie. to the identity transformation).

There are also two other routines which can be used to start and end a temporary series of transformations, which will override any transformations previously in effect and then reinstate them. These are the *GENStartTempTrans* and *GENEndTempTrans* routines.

2.6. DISPLAY ATTRIBUTES

These routines control the visible attributes of the primitives displayed. Color is one of the most obvious attributes. GENGRAF maintains two color values for each viewport, one is the *current color* and the other the *background color*. The current color is in effect the foreground color, in which all primitives, both geometric and text, are displayed. The background color is used for erasures whether they are of the whole viewport or of previously displayed primitives. Routines exist to define a color and to both define and set the current and background colors.

The other attributes which can be controlled are linestyle for any line segments displayed, and fill pattern, that defines the pattern used to display solid fill areas. Also included in this section are routines to control which memory planes are being written to and read from, if the device has multiple memory planes. For a more detailed description of this topic see Appendix 1.

2.7. GRAPHICAL PRIMITIVES

Any graphical item which can be generated by the display device in a single action is known as a primitive. GENGRAF defines two classes of these, *geometric primitives* and *text primitives*. Geometric primitives include line segments and polygons. Line segments may be defined by using a *moveto* instruction, followed by a *drawto* instruction. However, only a single primitive is defined by the two instructions. There are three kinds of moves and draws which are available within the package, each relating to a different coordinate system. The *GENMove* and *GENLine* routines relate to the device coordinate system. The *GENMove2* and *GENLine2* routines relate to a 2D world system, and the *GENMove3* and *GENLine3* routines relate to a 3D world system.

To draw multiple line sequences the routines *GENPolyLine, GENPolyLine2* and *GENPolyLine3* are provided for the device, 2D world and 3D world coordinate systems respectively.

The solid geometries available at present are the *GENBox* and *GENBox2* routines for displaying rectangles in the device and 2D world coordinate systems, respectively. These routines draw orthogonal, solid filled boxes, if the device is capable of displaying solid fill primitives. The *GENPolygon* and *GENPolygon2* routines display arbitrary shaped polygons, these are filled in if the graphics device has such a capability.

Other geometrical primitives are the *GENArc* and GENArc2 routines for displaying arcs comprising of a series of line segments.

Text primitives are character strings, the basic primitive being the individual character. The *GENText* and *GENText2* routines provide the capability to display text strings, located with respect to the device and world 2D coordinate systems, respectively. There is also a routine for question and answer purposes, this is the *GENQuestReply* routine.

2.8. COORDINATE CONVERSIONS

GENGRAF provides routines to convert coordinates from one system to another, under the currently defined transformations. *GENVtos2* will convert a world 2D coordinate pair to the corresponding device coordinates. *GENVtos3* will do the same for a 3D world triplet. *GENStov2* will convert a device coordinate pair to the corresponding 2D world coordinate point.

Four other routines are also provided, these are in pairs relating to either the 2D or 3D world systems. The *GENWtransform* routines subject either a 2D or 3D point to the appropriate world transformation. The *GENWinvtransform* routines subject either a 2D or 3D point to the inverse of the appropriate world transformation.

2.9. INQUIRY

A set of inquiry routines are provided so that the programmer can interrogate certain display values. These may be contained in the device description data base (of the MFB package) or maintained in the GENGRAF package data area. These routines can be useful when writing programs for use on a number of different devices which have different display characteristics. The program is able to inquire about certain values and make decisions, based on what it learns.

2.10. GENERAL CONTROL

These general control routines could also be termed the miscellaneous routines, since they are routines that do not fit into the other categories. The *GENEraseAll* and *GENEraseView* routines erase the whole display area and individual viewports, respectively. *GENFrame* and *GENNoFrame* are used to invoke and remove a frame around the currently enabled viewport. The *GENPoint* routine is used to invoke the device's pointing mechanism, if it has one. The coordinates of any point selected by the pointing device are returned along with the identity of any button or key pressed to indicate this selection. This routine provides the basis for interactive control of programs, as it can be used for menu selection and identifying other primitives of interest.

3. THE DEVICE DRIVER INTERFACE

The GENGRAF package is primarily for use with frame buffer (raster) devices, therefore the low level routines used to actually control input and output, from and to the device tend to reflect this. GENGRAF makes calls to these low level routines via a *virtual graphics interface.* The most general form of this interface is provided by use of the MFB package of routines.

3.1. THE MFB SYSTEM

The Model Frame Buffer (MFB) system [4] provides the GENGRAF package with a virtual graphics interface for frame buffer (raster) devices. MFB performs the terminal dependent task of encoding and decoding graphics code, thereby allowing the user to write graphics programs to run on almost any graphics device. The MFB package could control vector devices, however, it is more specifically directed towards frame buffer machines. The package displays primitives defined in the device coordinate system, with control over their visible attributes. The package also contains routines for general device control.

The MFB package is written in the C language and was originally intended to run under the UNIX operating system, however, an attempt to run it under the VMS operating system has been made. The package depends on the operating system to control the environment in which it has to run. An example of such control would be the instigation of a *raw* mode of operation, where each character typed in at the terminal is interpreted immediately by the host computer, rather than after a carriage return has been sent. Also, the operating system suspends any interrupts from external processes while in the graphics mode of operation. This is important, as otherwise such interrupts could be wrongly interpreted as graphics instructions. The MFB package has the capability to invoke both of these modes via the UNIX operating system.

Device Description Data Base

MFB uses a device description data base [5]. This contains descriptions of each individual device's capabilities and control syntax. Thus, the GENGRAF system calls the MFB package to initialize the routines for control of a certain type of device. If a new display device is acquired, it can be run with the GENGRAF routines when the appropriate device description is added to the data base. This means no new routines have to be written to cope with a particular display device, unless an operation which is not contained within the MFB package is required. In such a situation it would be necessary to expand both the MFB and GENGRAF subroutine libraries.

4. USING THE GENGRAF LIBRARY

4.1. Writing Programs Using GENGRAF

Writing a program that uses the GENGRAF routines is essentially no different from writing any other program. The GENGRAF routines are called to perform the graphics operations the rest of the program being unaffected. The trivial C example program below shows the typical sequence of calls :

```
main()
/* GENGRAF C test program */
{
    float xcen, ycen, size;

    GENOpen("18", "");
    GENSetWindow(0.0, 100.0, 0.0, 100.0);
    xcen = 50.0;
    ycen = 50.0;
    for (size = 10.0; size <= 90.0; size += 5.0)
    {
        drawbox(xcen, ycen, size);
    }
    sleep(5);
    GENClose();
}

drawbox(xc, yc, size)
float xc, yc, size;
/* draw a box of side length size, centered at xc,yc */
{
    size = size / 2.0;
    GENMove2(xc+size, yc+size);
    GENLine2(xc-size, yc+size);
    GENLine2(xc-size, yc-size);
    GENLine2(xc+size, yc-size);
    GENLine2(xc+size, yc+size);
}
```

GENOpen is the first routine called as this initializes the graphics routine package and the graphics device. The default viewport is used in this example and the window mapped on to it is set next. After that a number of GENMove2 and GENLine2 calls are made which result in a series of concentric rectangles being drawn. Finally the program finishes by calling the GENClose routine to reset the graphics device to its original state.

The equivalent program written in Fortran and calling the GENGRAF Fortran library is shown below :

```
      program test
c     GENGRAF Fortran test program
      real xcen, ycen, size
      call gopen("18", "")
      call gsetwd(0.0, 100.0, 0.0, 100.0)
      xcen = 50.0
      ycen = 50.0
      do 10 i = 2,18
           size = float(i) * 5.0
           call drwbox(xcen, ycen, size)
 10   continue
      call sleep(5)
      call gclose
      stop
      end

      subroutine drwbox(xc, yc, size)
c     draw a box of side length size, centered at xc,yc
      real xc, yc, size
      size = size / 2.0
      call gmove2(xc+size, yc+size)
      call gline2(xc-size, yc+size)
      call gline2(xc-size, yc-size)
      call gline2(xc+size, yc-size)
      call gline2(xc+size, yc+size)
      return
      end
```

4.2. Compiling Programs Using GENGRAF

In order to use the GENGRAF routines the programmer must call them from his/her program code (as shown in the above examples) and then link the GENGRAF library in with their own compiled code to produce the fully executable program file. An example for a 'C' program might be as follows :

```
cc test.c ~rgoliver/lib/gengraf-c.a -lm -o test
```

where test.c is the file containing the C program code which uses graphics. The file *~rgoliver/lib/gengraf-c.a* is the name (including part of a UNIX pathname) of the file containing the GENGRAF 'C' library. The "-lm" tells the loader to load the standard math library and "-o test" to put the executable code in a file called test.

In the case of a Fortran program the compilation command might be as follows :

```
f77 test.f ~rgoliver/lib/gengraf-f.a -o test
```

where test.f is the file containing the Fortran program code which uses graphics. The file *~rgoliver/lib/gengraf-f.a* is the name (including part of a UNIX pathname) of the file containing the GENGRAF Fortran library. The "-o test" tells the loader to put the executable code in a file called test.

4.3. Running Programs Using GENGRAF

Once an executable program has been compiled and linked it is ready to run. This should be straight forward unless the program utilizes some non-standard version of GENGRAF. The standard version uses the MFB package to provide the low level device driver. It is possible however, to link GENGRAF to some other device driver package to drive a single device type in order to reduce memory requirements. If the MFB version is being used then the program must be able to read the graphics data base description file (known as the mfbcap file). This can be declared by use of the MFBCAP environmental variable which gives a string containing a pathname to the data base file. If no MFBCAP variable exists in the current environment then MFB searches in a directory called /cad/lib for a file called mfbcap.

GENGRAF sets up some non-standard line modes when operating some graphics devices. Therefore, if the program should crash while executing a graphics program, the terminal line mode will have to be reset manually. This can usually be achieved by typing the following commands

```
tset [control-j]
stty -litout [control-j]
```

If the routines do not behave as expected one should consult ones local "graphics guru".

REFERENCES

[1] Graphics Standards Planning Committee, Special Interest Group on Graphics, Association for Computer Machinary (SIGGRAPH-ACM), *Computer Graphics, Vol. 13, No. 3, August 1979.*

[2] J.D. Foley, P.A. Wenner, Department of E.E.C.S., George Washington University, Washington D.C. 20052. *The George Washington University Core System Implementation,* Computer Graphics, July 1981.

[3] B.W. Kernighan, D.M.Ritchie, Bell Laboratories, Murray Hill, N.J. *The C Programmmers Manual.* Prentice-Hall, Inc.,Englewood Cliffs, N.J. 07632, 1978.

[4] G. Billingsley, K. Keller, University of California, Berkeley, CAD Group, Department E.E.C.S. *Model Frame Buffer (MFB) manual,* UNIX system document, 1982.

[5] G. Billingsley, K. Keller, University of California, Berkeley, CAD Group, Department E.E.C.S. *Graphics Terminal Capability Data Base (MFBCAP) manual,* UNIX system document, 1982.

[6] W.M. Newman, R.F. Sproul, *Principles of Interactive Computer Graphics,* 2'nd Edition, Mcgraw-Hill, N.Y., 1979.

APPENDIX 1

GENGRAF SUBROUTINE USERS MANUAL

This appendix describes the GENGRAF routines and the calling sequences necessary to invoke them. The package is written in the C language and so the names and necessary parameters for C program calls are given. The routines are also callable in Fortran, as an intermediary set of C routines has been provided to make Fortran calls compatible with the corresponding C routines. This connecting set of routines is necessary in order to conform to Fortran's requirement of a maximum of six characters for variable and subroutine names and also to make procedure call formats compatible. The Fortran subroutine names and passed parameter types are given below those for C.

The routines have been segregated into groups relating to the different aspects of operation and control of the graphical display. If default values are set up at initialization time without a program having to make explicit calls to set them up then this is indicated below the subroutine description.

1. Starting and Finishing

C **GENOpen(devicename,devicefile)**
*char *devicename, *devicefile;*
FORTRAN **gopen(devnam,devfil)**
*character*30 devnam,devfil*

Initialize the GENGRAF package for use on the specified device. The name of the graphics device (devicename) must be given. This device name should be one of those in the Model Frame Buffer terminal CAPabilities (MFBCAP) file. In addition a UNIX device file descriptor (devicefile) may be specified if the program is to be run from a device other than the graphics display. This routine must be called before any other GENGRAF routine. A viewport is established and set to be subsequently enabled for the convenience of those applications requiring only a single viewport. This default viewport is named "single".

Defaults : There is no default device name; this must be specified. The default device file is the device from which the program is initiated. Therefore, the null string as the device file descriptor will specify the graphics to appear on the device from which the program is invoked.

C **GENClose()**
FORTRAN **gclose**

Close the view package and free all memory allocated for its use. Called when all viewing operations have been completed. This routine must be called so as to leave the device in the same state in effect before the package was initialized.

C **GENOpenView(viewname)**
*char *viewname;*
FORTRAN **gopvp(vname)**
*character*10 vname*

Create a viewport by specifying the alpha-numeric name of the viewport. The name, which must be no more than ten characters long, is checked to see that it has not already been used and if not, all the necessary variable allocation for the viewport is performed. If it is required to direct graphics to the viewport then it must be established as the *currently enabled* or *working viewport* with a call to GEN-SetCurView. In this case the viewport is referenced with the same name as used when it is originally defined.

C **GENCloseView(viewname)**
*char *viewname;*
FORTRAN **gclvp(vname)**
*character*10 vname*

Close the *viewname* viewport This will free all storage allocated for the viewport when it was opened. Another viewport of the same name could subsequently be defined, but none of the original viewport information will be retained.

2. Viewports and Windows

C **GENSetCurView(viewname)**
*char *viewname;*
FORTRAN **gscvp(vpname)**
*character*10 vpname*

Set the *viewname* viewport as the *currently enabled* or *working* viewport. The viewport must have already been opened with a call to GENOpenView. All subsequent graphical output will be directed to this viewport, until another call to GENSetCurView is made, specifying another viewport.

C **GENSetViewport(xmin,xmax,ymin,ymax)**
float xmin,xmax,ymin,ymax;
FORTRAN **gsvp(xmin,xmax,ymin,tmax)**
real xmin,xmax,ymin,ymax

Define the limits of the currently enabled viewport on the output device display area. This is specified in terms of a normalized device coordinate system (NDC). On initialization the normalized coordinates of the device are set to range from 0 to 1.0 in the x display direction, increasing from left to right. The normalized coordinate range in the y direction is also defined to range from 0 to 1.0 from bottom to top.

Default : The viewport is set to fill the full device area when it is initialized with a call to GENOpen-View. That is,

xmin = 0
xmax = 1.0
ymin = 0
ymax = 1.0

C **GENSetDimension(dim)**
int dim;
FORTRAN **gsdim(dim)**
int dim

Set the type of coordinate system to be used in the currently enabled viewport. The *dim* parameter is set to either 2 for a two-dimensional world system (2D) or 3 for a three-dimensional world system (3D). In the case of a 2D system the only other routine called is the window routine, which defines the area of the world coordinate system which is mapped onto the viewport. For a three-dimensional world coordinate system, other routines besides the windowing routine need to be called in order to establish the viewing transformation to display the world system window within the viewport. In the case of a 3D system the view into the world coordinate system comprises a hypothetical 'pyramid of vision' which has as its apex the eye of the observer.

Default : A two dimensional coordinate system is established when the viewport is opened.

C **GENSetWindow(xmin,xmax,ymin,ymax)**
float xmin,xmax,ymin,ymax;
FORTRAN **gswd(xmin,xmax,ymin,ymax)**
real xmin,xmax,ymin,ymax

Define the window into the world coordinate system which will be mapped onto the current viewport. Appropriate translations and scaling are performed automatically so as to ensure the window area maps onto the viewport exactly. To ensure a uniform transformation between the window and the viewport (that is to prevent any relative distortion between any two perpendicular directions) the window and viewport aspect ratios must be equal. In the case of a 2D world system, the window specifies a rectangle in the world coordinate plane. In the case of a 3D world system, the window specifies a rectangle on the projection plane to be mapped onto the viewport. The projection plane is a plane which lies perpendicularly to the viewing direction, and which is centered about the intersection of the viewing direction vector. All the points,lines and planes within the world system are projected onto this plane which is the projected onto the appropriate viewport. The values specified are the minimum and maximum coordinates in the x and y directions, respectively. In each case the minimum values must be less than the the maximum values.

Default : The window values are set to the corresponding viewport values in the NDC system, which in the case of the default viewport is the full display area. That is,

xmin = 0
xmax = 1.0
ymin = 0
ymax = 1.0

C **GENFrame()**
FORTRAN **gfrm**

Draw a frame around the currently enabled viewport in the current color. If the viewport takes up the whole of the display and there is no room for a frame the result is undefined. Thus it is the users responsibility to ensure that a viewport frame will fit on the display area.

3. 3D View Parameters

C **GENSetViewPoint(xref,yref,zref)**
float xref,yref,zref;
FORTRAN **gsvpt(xref,yref,zref)**
real xref,yref,zref

Specifies the point in a 3D world coordinate system from which the rest of the world system is viewed. Another analogy is to consider the eye of the observer to be at this point.

Default : View point set to the origin. Thus this or the default viewing direction must be changed in order to be able to view any 3-dimension objects.

C **GENSetViewDir(vdx,vdy,vdz)**
float vdx, vdy, vdz;
FORTRAN **gsvdir(vdx,vdy,vdz)**
real vdx, vdy, vdz

Specifies the viewing direction, in a 3D world coordinate system. The parameters given are that of a vector pointing in the viewing direction, relative to the view reference point. The various viewing angles, relative to the orthogonal world coordinate system are calculated using these values. By default no viewing direction is specified, therefore, these values must be initialized to some set of values which specify a non-zero vector, before a view can be implemented.

Default : No direction defined.

C **GENSetViewUp(upangle)**
float upangle;
FORTRAN **gsvup(upang)**
real upang

Specifies the view-up direction in a 3D world coordinate system. This direction is that which appears vertically on the viewport. Alternatively this direction can be thought of as specifying the angle of rotation of the window from its initial position on the projection plane.

Default : For a two-dimensional world the view-up direction is parallel to the y-axis. For a three-dimensional world system the direction of maximum increasing z is defined as the view-up direction.

C **GENSetViewDist(vdist)**
float vdist;
FORTRAN **gsvdst(vdist)**
real vdist

Specifies the distance from the view reference point to the projection plane. This effectively controls the two angles of the 'viewing pyramid' and consequently the scope of the world coordinate system to be viewed. Default : The viewing distance is set to zero; ie. the view reference point lies in the projection plane, which is a paradoxical situation. Thus the viewing distance parameter must be initialized to a positive non-zero value before a view can be implemented.

4. Clipping Control

The following routines deal with clipping, which can be in either of two states *on* or *off*. This is signified by the Boolean variable **onoff** whose value is 1 if the associated clipping is *on* and 0 if the associated clipping is *off*.

C **GENSetViewDepth(minZ,maxZ)**
float minZ,maxZ;
FORTRAN **gsvdth(minz,maxz)**
real minz,maxz

Specifies two planes which are perpendicular to the viewing direction vector and which are at distances minZ and maxZ from the view reference point (observer). Thus, they define a *slice* of the world coordinate system. Clipping of all graphics primitives can be invoked with respect to either or both of these planes. The minZ value must be less than the maxZ value. The default value for both parameters is zero, which if depth clipping were evoked, would produce no visible display. However, depth clipping is turned off by default. Thus, before depth clipping is turned on, valid values for minZ and maxZ must have been set.

Default : No values defined.

C **GENSetWindClip(onoff)**
int onoff;
FORTRAN **gswclp(onoff)**
integer onoff

Turn clipping to the window on or off. Window clipping can be active with either a 2D or 3D world coordinate system. In the case of a 2D system it clips all primitives to the window on the world plane. For the 3D case primitives are clipped to the 'viewing pyramid' which extends from the view reference point. Another way of describing this 3D clipping operation is to imagine that all primatives are projected onto an infinite projection plane. Then 2D clipping is evoked on the projection plane to leave only the part of the display that is mapped onto the viewport. When clipping is turned off, all of the primitives displayed should lie within the specified window. If this is not the case then the effect of the display of primitives lying partially or wholly outside the window is undefined and will be device dependent.

Default : Clipping is turned *on*.

C **GENSetFrontClip(onoff)**
int onoff;
FORTRAN **gsfclp(onoff)**
integer onoff

In the case of a 3D view it is possible to clip primitives to certain depth values as specified by a call to the GENSetViewDepth routine. The GENSetFrontClip routine is a routine which switches clipping to the front of these *depth*planes (the one nearest the view reference point) either *on* or *off*. The values of minZ and maxZ must be set before any depth clipping is turned on.

Default : Clipping to the front depth plane is turned *off*.

C **GENSetBackClip(onoff)**
int onoff;
FORTRAN **gsbclp(onoff)**
integer onoff

Turn clipping to back depth plane on or off. As for the GENSetFrontClip routine except relates to the back depth plane (the one furthest from the view reference point).

Default : Clipping to the back depth plane is turned *off.*

5. World Transformations

C **GENScale2(sx,sy)**
float sx,sy;
FORTRAN **gscal2(sx,sy)**
real sx,sy

Applies scaling in a 2D world system by factors of sx and sy, respectively, in the x and y directions. Scaling is performed with respect to the origin. Thus, if it is desired to scale relative to any other point, then this point must first be translated to the original origin, then the scaling operation performed and finally the scaling point must be translated back to its original position.

Default : Uniform scaling, ie. sx = sy = 1.0

C **GENTranslate2(tx,ty)**
float tx,ty;
FORTRAN **gtran2(tx,ty)**
real tx,ty

Applies a translation in a 2D world system by offset values of tx and ty, respectively, in the x and y directions. The same effect, in terms of what will be displayed in the viewport, could be produced by changing the window values.

Default : No translations in effect, ie. tx = ty = 0.0

C **GENRotate2(angle)**
float angle;
FORTRAN **grot2(angle)**
real angle

Applies a rotation to a 2D world system of *ang* degrees, measured positively in an counter-clockwise direction.

Default : No rotation in effect, ie. angle = 0.0

C **GENScale3(sx,sy,sz)**
float sx,sy,sz;
FORTRAN **gscal3(sx,sy,sz)**
real sx,sy,sz

Applies scaling in a 3D world system by factors of sx,sy and sz, respectively, in the x,y and z directions. See GENScale2 for a description of how to produce scaling with respect to a point other than the origin.

Default : Uniform scaling, ie. sx = sy = sz = 0.0

C **GENTranslate3(tx,ty,tz)**
float tx,ty,tz;
FORTRAN **gtran3(tx,ty,tz)**
real tx,ty,tz

Applies a translation in a 3D world system, by offset values of tx,ty and tz, respectively, in the x,y and z directions.

Default : No translation in effect, ie. tx = ty = tz = 0.0

C **GENRotate3(axis,angle)**
char axis
float angle
FORTRAN **grot3(axis,angle)**
*character*1 axis*
real angle

Applies rotations to a 3D world system. The axis, about which the rotation is to be produced and the angle of rotation to be applied need to be specified. The axis value is a single character and must be either 'x', 'y' or 'z' to produce a rotation. The units of angular measurement are degrees.

Default : No rotations in effect, ie. angle = 0 for all axes.

C **GENUndoTrans()**
FORTRAN **gudtrn**

Undo the last transformation specified. The result is to produce the combined world transformation that was in effect before the most recent transformation was evoked. The routine can be called repeatedly until the there is no world transformation, that is, only the identity transformation which has no effect.
This can also be thought of as *popping* a transformation off the top of a transformation *stack*.

C **GENResetTrans()**
FORTRAN **grttrn**

Reset the world transformation to the identity transformation. This destroys any transformations that might have been in effect.

C **GENStartTempTrans()**
FORTRAN **gstemp**

Start a temporary transformation sequence or stack. This will override any transformations previously in effect until the GENEndTempTrans routine is called to restore the original transformations and end the temporary stack. This routine could be useful for temporarily overriding the current world transformation or to display primatives with no transformation in effect, such as a coordinate triad.

C **GENEndTempTrans()**
FORTRAN **getemp**

End a temporary transformation sequence or stack. This is called to end a temporary stack evoked by a call to the GENStartTempTrans routine.

C **GENDeg(dx,dy)**
float dx,dy;
FORTRAN **gdeg(dx,dy)**
real dx,dy

This function returns an angle in degrees corresponding to offsets in two mutually perpendicular directions, dx and dy.

6. Controlling Attributes

C **GENDefineColor(colorid,R,G,B)**
int colorid, R,G,B;
FORTRAN **gdcol(color,red,green,blue)**
integer color,red,green,blue

Define a color. The color is identified by the code number colorid. This is specified by giving the intensity values of the red, green and blue components of which the color is to be composed. The intensity values for each of the colors can range from 0 to 1000. Thus a color region consisting of a cube with coordinates ranging from zero to 1000 can be envisaged.

C **GENSetColor(colorid)**
int colorid;
FORTRAN **gscol(color)**
integer color

Set color *colorid* as the current color.

C **GENSetCursorColor(color1,color2)**
int color1,color2;
FORTRAN **gscurc(color1,color2)**
. *integer color1,color2*

Set the color of the device cursor, if it has one, to alternate between *color1* and *color2*. The colors alternate only if the device has this capability otherwise only the first color is used.

C **GENSetBlinker(colorid,r,g,b,onoff)**
int colorid,r,g,b,onoff;
FORTRAN **gsblk(color,r,g,b,onoff)**
integer color,r,g,b,onoff

Set color *colorid* to blink to another color specified by the red, green and blue. The frequency with which it blinks is specified by the *onoff* value, which is an integer number of machine refresh cycles. A single refresh cycle is usually of the order of 1/30th to 1/60th of a second long. This feature is only available on devices that support it in their hardware/firmware.

C **GENDefineLineStyle(styleid,bitarray)**
int styleid, bitarray;
FORTRAN **gdlsty(style,bitarr)**
integer style,bitarr

Define a linestyle. The linestyle is referenced by the code number styleid. The actual style of the line is specified by giving the repeatable pattern of the line style in the 'binary' array bitarray. Thus, if bitarray was set to a value of decimal 85 (that is, binary '01010101'), then the line would displayed as dotted, with alterate pixels being lit in the current color.

C **GENSetLineStyle(styleid)**
int styleid;
FORTRAN **gslsty(style)**
integer style

Set the line style for all subsequent line segments. Styleid could be a default device line style or a user defined style.

Default : Solid line.

C **GENDefineFillPattern(styleid,bitarray)**
*int styleid, *bitarray;*
FORTRAN **gdfill(style,bitarr)**
integer style bitarr

Define a fillpattern for boxes and other polygons. The fillpattern is referenced by the code number styleid. The actual pattern is specified by giving the pattern of a small block in the 'binary' array bitarray, which is repeated all over the polygons area. Thus, if bitarray was set to the following decimal values 85, 170, 85, 170, 85, 170, 85, 170, then the two dimensional binary array would be as follows :

```
01010101
10101010
01010101
10101010
01010101
10101010
01010101
10101010
```

Thus the pattern displayed in a polygon would be one of diagonal stripes with the pixels in each row being alterately lit in the current color.

C **GENSetFillPattern(style)**
int style;
FORTRAN **gsfill(style)**
integer style

Set the fillpattern for all subsequent boxes and polygons. Pattern could be a default device pattern or a user defined pattern.

Default : Solid fill.

C **GENSetWriteMask(maskpattern)**
int maskpattern;
FORTRAN **gswmsk(mask)**
integer mask

Set the device write mask to that of maskpattern. This is only appropriate if the device has multiple display planes and they can be selectively written to. This routine and GENSetReadMask could be used to 'toggle' between different displays, which are stored in separate display planes. The maskpattern value is interpreted as a binary array with a each bit corresponding to a display plane. If a bit is set to 1, then the appropriate display plane is written to. If it is set to zero, then the display plane is not written to (ie., it is masked off).

C **GENSetReadMask(maskpattern)**
int maskpattern;
FORTRAN **gsrmsk(mask)**
integer mask

Set the device read mask to that of maskpattern. This is only appropriate if the device has multiple display planes and they can be selectively read from. As in the case of the GENSetWriteMask routine, the maskpattern value is interpreted as a binary array with a each bit corresponding to a display plane.

7. Geometric Primitives

C **GENMove(x,y)**
int x,y;
FORTRAN **gmove(x,y)**
integer x,y

Move the *pen* or *cursor* to the position (x,y) in the actual device coordinate system.

C **GENMove2(x,y)**
float x,y;
FORTRAN **gmove2(x,y)**
real x,y

Move the pen or cursor to the position (x,y) in the world system. If a 3D view is in effect then the z coordinate value defaults to zero.

C **GENMove3(x,y,z)**
float x,y,z;
FORTRAN **gmove3(x,y,z)**
real x,y,z

Move the pen or cursor to the position (x,y,z) in the world system. If a 2D view is in effect then the z coordinate value is ignored.

C **GENLine(x,y)**
int x,y;
FORTRAN **gline(x,y)**
integer x,y

Draw a line from the current cursor position to the point (x,y) in the device coordinate system. No clipping will be performed should any part of the line fall outside of the viewport or display area.

C **GENLine2(x,y)**
float x,y;
FORTRAN **gline2(x,y)**
real x,y

Draw a line from the current world cursor position to the (x,y) world system point specified, leaving the cursor at the point (x,y) whether this is inside the current window or not.

C **GENLine3(x,y,z)**
float x,y,z;
FORTRAN **gline3(x,y,z)**
real x,y,z

Draw a line from the current world cursor position to the (x,y,z) world system point specified, leaving the cursor at the point (x,y,z) whether this is inside the current window or not.

C **GENPolyLine(nvert,xy)**
int nvert,xy;
FORTRAN **gplin(nvert,xy)**
integer nvert,xy[]

Draw a series of connected line segments joining the *nvert* vertices, in the device coordinate system. No clipping will be performed should any part of the line sequence fall outside of the viewport or display area. The x and y coordinates are ordered in consecutive pairs in the linear array *xy*.

C **GENPolyLine2(nvert,xy)**
int nvert;
float xy[];
FORTRAN **gplin2(nvert,xy)**
integer nvert
real xy

Draw a series of connected line segments joining the *nvert* vertices, in the 2D world coordinate system. The x and y coordinates are ordered in consecutive pairs in the linear array *xy*.

C **GENPolyLine3(nvert,xyz)**
int nvert;
float xyz[];
FORTRAN **gplin3(nvert,xyz)**
integer nvert
real xyz

Draw a series of connected line segments joining the *nvert* vertices, in the 3D world coordinate system. The x,y and z coordinates are ordered in consecutive triples in the linear array *xyz*.

C **GENArc(x,y,r,astart,astop,s)**
int x,y,r,astart,astop,s;
FORTRAN **garc(x,y,r,astart,astop,s)**
integer x,y,r,astart,astop,s

Display an arc in the device coordinate system with its origin at the point *x,y* and of radius *r*. The arc starts at an angle *astart* measured from the positive direction of the x axis, and goes through to an angle *astop*. The arc comprises of *s* line segments.

C **GENArc2(x,y,r,astart,astop,s)**
float x,y,r,astart,astop;
int s;
FORTRAN **garc2(x,y,r,astart,astop,s)**
real x,y,r,astart,astop
integer s

Display an arc in the 2D world coordinate system with its origin at the point *x,y* and of radius *r*. The arc starts at an angle *astart* measured from the positive direction of the x axis, and goes through to an angle *astop*. The arc comprises of *s* line segments.

C **GENBox(x1,y1,x2,y2)**
int x1,y1,x2,y2;
FORTRAN **gbox(x1,y1,x2,y2)**
integer x1,y1,x2,y2;

Display a box with diagonal coordinates (x1,y1) and (x2,y2), where the coordinate values are in device units. The box will be displayed in the current fillpattern and color.

C **GENBox2(x1,y1,x2,y2)**
float x1,y1,x2,y2;
FORTRAN **gbox2(x1,y1,x2,y2)**
real x1,y1,x2,y2;

Display a box with diagonal coordinates (x1,y1) and (x2,y2), where the coordinate values are in world units. The box will be displayed in the current fillpattern and color.

C **GENPolygon(nvert,xy)**
int nvert,xy;
FORTRAN **gpgon(nvert,xy)**
integer nvert,xy[]

Display a filled polygon, in the current fill pattern, defined by *nvert* vertices, in the device coordinate system. No clipping will be performed should any part of the polygon fall outside of the viewport or display area. The x and y coordinates are ordered in consecutive pairs in the linear array *xy*.

```
C          GENPolygon2(nvert,xy)
           int nvert;
           float xy[];
FORTRAN    gpgon2(nvert,xy)
           integer nvert
           real xy
```

Display a filled polygon, in the current fill pattern, defined by *nvert* vertices, in the 2D world coordinate system. The x and y coordinates are ordered in consecutive pairs in the linear array *xy*.

8. Text Primitives

```
C          GENText(string,x,y,pos,ang)
           char *string;
           int x,y,pos,ang;
FORTRAN    gtext(string,x,y,pos,ang)
           character*100
           integer x,y,pos,ang
```

Display the text string *string* at the device coordinates (x,y) using the relative position code *pos* at the angle *ang*. The relative position code determines how the string will be justified with respect to the locating coordinates. The position code can be any value from one to nine inclusive. The positions corresponding to each code are shown below :

```
7 - 8 - 9

4 - 5 - 6

1 - 2 - 3
```

```
C          GENText2(string,x,y,pos,ang)
           char *string;
           float x,y;
           int pos,ang;
FORTRAN    gtext2(string,x,y,pos,ang)
           character*100 string
           real x,y
           integer pos,ang
```

Display the text string *string* at the world coordinates x,y using the relative position code *pos* at the angle *ang*. The position codes are the same as for GENText above.

C **GENQuestReply(x,y,promtstr,replystr)**
int x,y;
*char *promtstr, *replystr;*
FORTRAN **gqrep(x,y,pstr,rstr)**
integer x,y
*character*100 pstr,rstr*

Write a prompt string starting at location (x,y) in device coordinates, to be followed by the users typed in response, which is returned in the string pointed to by replystr.

9. Coordinate Conversions

C **GENWtransform2(x,y)**
*float *x, *y;*
FORTRAN **gtfm2(x,y)**
real x,y

Apply the current 2D world transformation to the (x,y) coordinate pair and return their transformed values.

C **GENWtransform3(x,y,z)**
*float *x, *y, *z;*
FORTRAN **gtfm3(x,y,z)**
float x,y,z

Apply the current 3D world transformation to the (x,y,z) coordinate set and return their transformed values.

C **GENWinvtransform2(x,y)**
float x,y;
FORTRAN **gitfm2(x,y)**
real x,y

Apply the inverse of the current 2D world transformation to the (x,y) coordinate pair and return their transformed values.

C **GENWinvtransform3(x,y,z)**
float x,y,z;
FORTRAN **gitfm3(x,y,z)**
real x,y,z

Apply the inverse of the current 3D world transformation to the (x,y,z) coordinate set and return their transformed values.

C **GENWtos2(xv,yv,xs,ys,valid)**
float xv,yv;
*int *xs, *ys, valid;*
FORTRAN **gwts2(xv,yv,xs,ys,valid)**
real xv,yv
integer xs,ys, valid

Transform the 2D world system coordinate pair (xv,yv) to the corresponding device coordinate pair (xs,ys). If the point xs,ys is outside of the current viewport then the valid flag is set to zero, otherwise it is set to 1.

C **GENWtos3(xv,yv,zv,xs,ys,valid)**
float xv,yv,zv;
*int *xs, *ys, valid;*
FORTRAN **gwts3(xv,yv,zv,xs,ys,valid)**
real xv,yv,zv
integer xs,ys, valid

Transform the 3D world system coordinate set (xv,yv,zv) to the corresponding device coordinate pair (xs,ys). If the point (xs,ys) is outside of the current viewport then the valid flag is set to zero, otherwise it is set to 1.

C **GENStow2(xs,ys,xv,yv,valid)**
int xs,ys;
*float *xv, *yv;*
int valid;
FORTRAN **gstw2(xs,ys,xv,yv,valid)**
integer xs,ys
real xv,yv
integer valid

Transform the device coordinate pair (xs,ys) to the corresponding 2D world coordinate system pair (xv,yv). If the point (xs,ys) is outside of the current viewport then the valid flag is set to zero, otherwise it is set to 1.

10. Inquiry

10.1 Device Values

C **GENInqDevRes(xres,yres)**
*int *xres, *yres;*
FORTRAN **gidres(xres,yres)**
integer xres,yres

The device resolution in the x and y directions is returned. The values correspond to the maximum addressable values in each direction. It is assumed that the lowest addressable values are zero in both the x and y directions.

C **GENInqNumColors(maxnum)**
*int *maxnum;*
FORTRAN **gincol(numcol)**
integer numcol

The maximum number of colors a device can simultaneously display is returned. This value must always be greater or equal to two, since even a monochrome device has one foreground and one background color.

C **GENInqNumBitPlanes(maxnum)**
*int *maxnum;*
FORTRAN **ginbit(numcol)**
integer numcol

The number of bit planes the device uses to display images is returned. This value must always be greater or equal to one, since a monochrome device has a single bit plane.

C **GENInqNumBlinkers(maxnum)**
*int *maxnum;*
FORTRAN **ginblk(numblk)**
integer numblk

The number of simultaneously blinking colors the device can support is returned.

C **GENInqNumLines(maxnum)**
*int *maxnum;*
FORTRAN **ginlin(numlin)**
integer numlin

The maximum number of linestyles a device can simultaneously addressed is returned. The device may be capable of having many more types of linestyle displayed, however, they can not be address simultaneously.

C **GENInqNumPatterns(maxnum)**
*int *maxnum;*
FORTRAN **ginfil(numfil)**
integer numfil

The maximum number of fillpatterns a device can simultaneously addressed
is returned.
The device may be capable of having many more types of fillpatterns
displayed, however, they can not be addressed simultaneously.

C **GENInqTextSize(xtext,ytext)**
*int *xtext, *ytext;*
FORTRAN **gitsz(xtext,ytext)**
integer xtext,ytext

The size of the text character block is returned. The width is returned as xtext, and the height as ytext.

C **GENInqNumButtons(num)**
*int *num;*
FORTRAN **ginbut(num)**
integer num

The number of buttons the pointing device has is returned.

C **GENInqButton(button,num)**
*int button, *num;*
FORTRAN **gibut(button,num)**
integer button,num

The number returned when button number *button* is activated on the pointing device is returned.

C **GENInqMasks(maskbool)**
*int *maskbool;*
FORTRAN **gimask(mbool)**
integer mbool

A boolean value indicating whether the device has read and write masks is returned. If the value is 1 the device has masks and if the value is 0 it does not.

10.2. Viewport Values

C **GENInqCurView(curview)**
*char *curview;*
FORTRAN **gicvp(vpname)**
*character*10 vpname*

The alpha-numeric name of the currently enabled viewport is returned. If a viewport is not enabled a null string is returned.

C **GENInqViewport(vminX,vmaxX,vminY,vmaxY)**
*float *vminX, *vmaxX, *vminY, *vmaxY;*
FORTRAN **givp(vminx,vmaxx,vminy,vmaxy)**
real vminx,vmaxx,vminy,vmaxy

The NDC values of the bounds of the viewport are returned.

C **GENInqVratio(vratio)**
*float *vratio;*
FORTRAN **givrat(vratio)**
real vratio

The aspect ratio of the viewport is returned. This is the ratio of the vertical range of the viewport to its horizontal range.

C **GENInqDimension(currentdim)**
*int *currentdim;*
FORTRAN **gidim(dim)**
integer dim

The dimensionality of the world system associated with the viewport is returned. This value will be either two or three for a 2D or 3D system, respectively.

C **GENInqWorldRes(xres,yres)**
*float *xres, *yres;*
FORTRAN **giwres(xres,yres)**
real xres,yres

The world coordinate distances corresponding to one addressable unit on the display device is returned. In the case of a 2D world system these values is easily definable. In the case of a 3D world system these values are the distances measured on projection plane. The values can be of use when determining at what scale to display certain types of data.

10.3 Viewing Values

C **GENInqWindow(wminX,wmaxX,wminY,wmaxY)**
*float *wminX, *wmaxX, *wminY, *wmaxY;*
FORTRAN **giwd(wminx,wmaxx,wminy,wmaxy)**
wminx, wmaxx, wminy, wmaxy

The values of the window bounding edges are returned.

C **GENInqViewPoint(xref,yref,zref)**
*float *xref, *yref, *zref;*
FORTRAN **givpt(xref,yref,zref)**
real xref,yref,zref

If the world system is a 3D system the view reference point coordinates are returned.

C **GENInqViewDir(vdirdx,vdirdy,vdirdz)**
*float *vdirdx, *vdirdy, *vdirdz;*
FORTRAN **givdir(dx,dy,dz)**
real dx,dy,dz

If the world system is a 3D system the view direction vector values are returned.

C **GENInqViewUp(upangle)**
*float *upangle;*
FORTRAN **givup(upang)**
real upang

If the world system is a 3D system the view up angle value is returned.

C **GENInqViewDist(vdist)**
*float *vdist;*
FORTRAN **givdst(vdist)**
real vdist

If the world system is a 3D system the distance to the projection plane from the view reference point is returned.

C **GENInqCurWpos(curx,cury,curz)**
*float *curx, *cury, *curz;*
FORTRAN **giwpos(curx,cury,curz)**
real curx,cury,curz

The world coordinates of the present cursor position are returned.

C **GENInqCurSpos(screenx,screeny)**
*int *screenx, *screeny;*
FORTRAN **gispos(scx,scy)**
integer scx,scy

The device coordinates of the present cursor position are returned.

10.4 Transformation Values

C **GENInqTrans(trans)**
float trans[4][4];
FORTRAN **gitran(trans)**
real trans
dimension trans(4,4)

The 4 by 4 final transformation matrix is returned. This matrix is the combination of the world transformation matrix and the projection transformation matrix. If the world system is a 2D system then the first, second and fourth columns and rows contain the appropriate 3 by 3 matrix.

C **GENInqWorldTran(worldtran)**
float worldtran[4][4];
FORTRAN **giwtrn(wtran)**
real wtran
dimension wtran(4,4)

The 4 by 4 world transformation matrix is returned. If the world system is a 2D system then the first, second and fourth columns and rows contain the appropriate 3 by 3 matrix.

C **GENInqViewTran(viewtran)**
float viewtran[4][4];
FORTRAN **givtrn(vtran)**
real vtran
dimension vtran(4,4)

The 4 by 4 viewing projection transformation matrix is returned. If the world system is a 2D system, then this matrix will be the identity matrix, since no projection is performed.

10.5 Clipping Values

C **GENInqWindClip(clipbool)**
*int *clipbool;*
FORTRAN **giwclp(onoff)**
integer onoff

A Boolean flag indicating whether or not window clipping is in effect is returned. If the flag is 1 clipping is in effect and if the flag is zero, it is not.

C **GENInqFrontClip(frontbool)**
*int *frontbool;*
FORTRAN **gifclp(onoff)**
integer onoff

A Boolean flag indicating whether or not clipping to the front depth plane is in effect is returned. If the flag is 1 clipping is in effect and if the flag is zero, it is not.

C **GENInqBackClip(backbool)**
*int *backbool;*
FORTRAN **gibclp(onoff)**
integer onoff

A Boolean flag indicating whether or not clipping to the back depth plane is in effect is returned. If the flag is 1 clipping is in effect and if the flag is zero, it is not.

C **GENInqViewDepth(minz,maxz)**
*float *minz, *maxz;*
FORTRAN **givdth(minz,maxz)**
real minz,maxz

The distances to the front and back depth planes are returned.

10.6 Attribute Values

C **GENInqCurColor(currentcolor)**
*int *currentcolor;*
FORTRAN **giccol(color)**
integer color

The value of the identity of the currently defined foreground color is returned.

C **GENInqLineStyle(linestyle)**
*int *linestyle;*
FORTRAN **gilsty(lstyle)**
integer lstyle

The value of the identity of the currently defined linestyle is returned.

C **GENInqFillPattern(fillpattern)**
*int *fillpattern;*
FORTRAN **gifpat(filpat)**
integer filpat

The value of the identity of the currently defined fillpattern is returned.

11. General Control

C **GENEraseAll()**
FORTRAN **geall**

Erase everything displayed on the display surface.

C **GENEraseView()**
FORTRAN **gevp**

Erase everything within the viewport. The viewport is left displaying the background color.

C GENHalt()
FORTRAN ghalt

Suspend graphics mode operation. The line modes that were previously in effect are reinstated. This may be useful when putting a job in background mode.

C GENInit()
FORTRAN ginit

Initialize the device for graphics mode, usually after GENHalt has been called at an earlier time. The appropriate line modes for graphics are set. This may be useful when bringing a job from a background state into foreground operation.

C GENPoint(xpoint,ypoint,key,button,valid)
*int *xpoint, *ypoint;*
*char *key;*
*int *button, *valid;*
FORTRAN gpoint(xpoint,ypoint,key,button,valid)
integer xpoint,ypoint
*character*1 key*
integer button,valid

Enable the device's pointing device to select a point on the display surface. The device coordinates of the point *(xpoint,ypoint)* are taken when the stylus point is depressed (if a stylus is being used) or when a mouse button is depressed (if a mouse is being used) or when a keyboard key is hit (if the keyboard keys are being used). The key number *key* or mouse button number *button* are returned, if appropriate. Also a Boolean flag *valid* is returned and set to 1, if the selected point is within the currently enabled viewport and set to zero, if it is not.