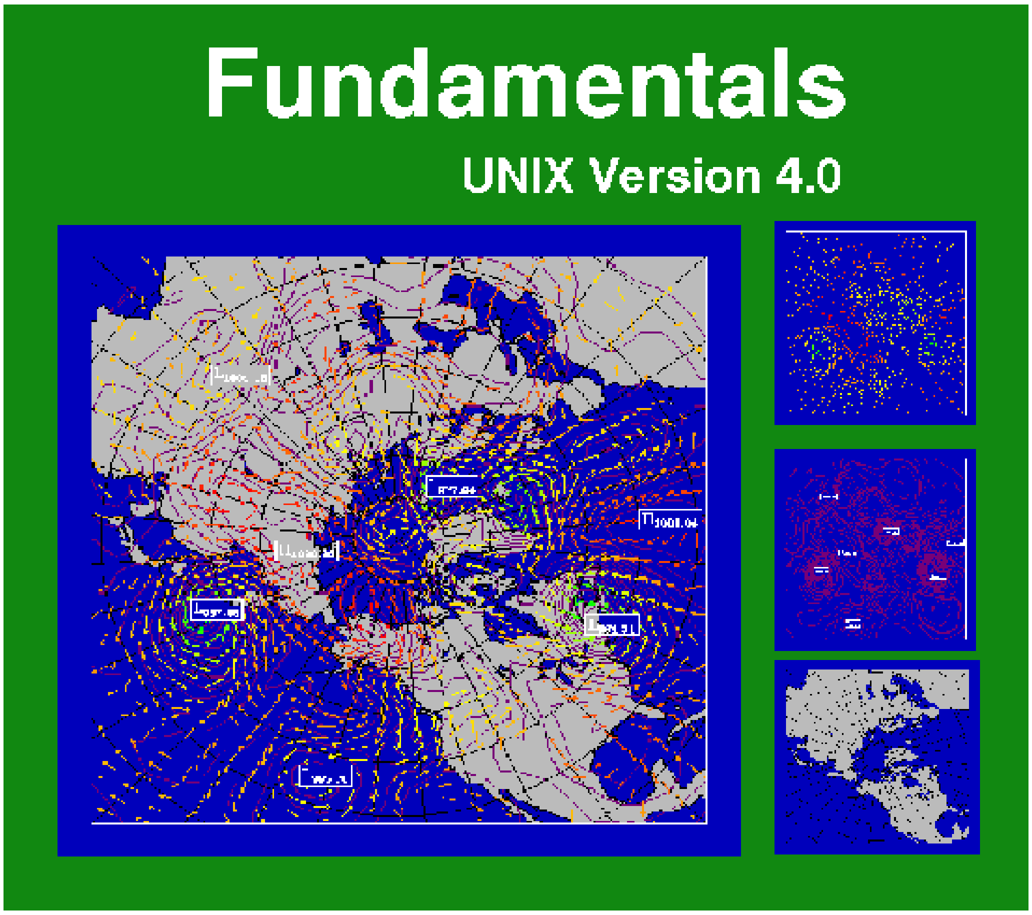
NCAR Graphics Fundamentals

Unix Version 4.0.1



Welcome to the NCAR Graphics Fundamentals documentation.

NCAR Graphics

Fundamentals

UNIX Version 4.0

Document Version 1.0

July 1995

Acknowledgments

Authors: Adrianne Middleton-Link, Dave Kennison, Fred Clare, Bob Lackman, Tim Scheitlin, John Clyne, Jeff Kuehn, Pete Morreale, Dave Brown

Technical writer/editors: Juli Rew, Nancy Dawson, Brian Bevirt

Production: Christine Guzy, Valeri Hart, Jacque Marshall, Janie Young

Special thanks to the following technical reviewers: Dave Kennison, Tim Scheitlin, Tom Parker, Ken Hansen, Pete Morreale, Jeff Kuehn, Bob Lackman, Fred Clare, John Clyne, and Mary Haley.

Thanks also to Jeff Kuehn and Sally Haerer for helping to organize this project and Brian Bevirt for his explanation of the design principles used in the tutorial modules of this manual (Chapter 5).

About the cover

Four NCAR Graphics utilities cooperate to produce the cover graphic using data originating from the U.S. National Meteorological Center. Ezmap creates the polar stereographic map projection and overlying map grid. Conpack draws the contour lines representing surface pressure at mean sea level, and Vectors renders the ground-level wind velocity vectors. Surface pressure also determines the color of the vectors, with red representing the highs, yellow the middle values, and green the lows. The program uses the Areas utility in two ways: first to allow Ezmap to solid fill the oceans and land masses, and then to create an area map that permits the map grid, the contour lines, and the vectors to be masked around the contour label boxes.

The data represent a 24-hour forecast extracted from the National Meteorological Center's Global Product Set for March 13, 1993, at 1200 hours. The National Weather Service's Numerical Product Service broadcasts the data over satellite feed; Unidata's Local Data Manager receives the transmissions and converts the raw data to netdcf format. A prototype version of NCAR Graphics Command Language was used to extract one level and timestep from the netcdf files into a single data file readable by the NCAR Graphics program.

The plot shows most of the Northern Hemisphere during one forecast timestep. The extreme low pressure at the right hand edge of the picture (962.74 hectopascals) represents the center of the severe snow storm that pummelled the East Coast of North America in March 1993. It is interesting to note the other storm of similar shape and intensity, occurring simultaneously at about the same latitude off the eastern coast of Asia. With NCAR Graphics installed on a UNIX system, you can compile and run the code that produced this plot by executing the command:

ncargex fcover

For ordering information, please contact NCAR Graphics ordering information at (303) 497-1201 or ncarginf@ncar.ucar.edu or:

University Corporation for Atmospheric Research
SCD/NCAR Graphics Orders
P.O. Box 3000
Boulder, CO 80307-3000 (USA)

Copyright

Trademarks

Published by: National Center for Atmospheric Research, Scientific Computing Division, P.O. Box 3000, Boulder, CO 80307-3000. The National Center for Atmospheric Research is operated by the University Corporation for Atmospheric Research and is sponsored by the National Science Foundation. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Conventions used in this document

UNIX

- Command names, options, filenames, pathnames, directories, and other items that must be typed as shown are printed in bold letters. Examples:
  man ls
  For more information about the **ls** command, see its **man** page.
- Variables where you provide the substitution are shown in bold italic letters. Examples:
  mv ***filename1 filename2***
  At the prompt, type your ***login_id***.
- The \ (backslash) followed by pressing RETURN when you are entering a command allows you to continue the command on the next line. In UNIX, a \ escapes the next character.

Fortran

- Fortran programs, shell scripts, and screen displays are shown in Courier ("typewriter") font.

```
CALL FLMHDR (35)
```

General

- You must press RETURN after each command, but the RETURN is not shown in the document, except when required for clarity.
- Keystrokes that must be pressed simultaneously are linked by a hyphen, with the modifier key indicated by capital letters before the hyphen. Example:
    To finish the file, enter CONTROLd at the beginning of a new line.

Contents

Preface

Welcome to NCAR Graphics

NCAR Graphics has been distributed to about 1,500 sites around the world. At its heart lie over two dozen high-level Fortran-callable utilities for contouring, mapping, drawing field flows, drawing surfaces, drawing histograms, drawing X/Y plots, labeling, and more. The package includes an ANSI/ISO standard version of GKS, written in FORTRAN 77. Currently two graphics output options are included: a device-independent private encoding of the ANSI/ISO Computer Graphics Metafile (CGM) standard and X11 commands. Recent improvements in NCAR Graphics include more color options and a raster viewing library that allows you to translate between various CGM and raster formats.

History of graphics at NCAR

Since NCAR's inception as a national research center in the early 1960s, graphics has played an important role in supercomputing for the atmospheric sciences. Early on, the Computing Facility of NCAR, now the Scientific Computing Division (SCD), generated a low-level collection of routines to draw lines, curves, and dashed-line patterns called the NCAR System Plot Package. This package was later replaced with some higher level routines now known as SPPS and a GKS package.

By the 1970s, NCAR scientists and SCD staff had developed several higher level graphics utilities, which were integrated into a graphics package that could be distributed throughout the scientific community.

In the early 1980s, graphics and programming language standards had gained wide acceptance, so NCAR Graphics utilities were converted to FORTRAN 77, the metafile was based on a private encoding of the CGM standard, and utilities were moved to depend on the GKS standard.

Since that time, the functionality of NCAR Graphics has been growing: it now contains new utilities, new viewing capabilities, and a variety of translators and filters for the numerous standard file formats. In the next few years, we plan to install an interactive package over the existing FORTRAN 77 interface to allow scientists to modify graphics interactively and study their data as the data are generated.

How to use this documentation set

To provide improved information for users, NCAR Graphics documentation has changed immensely since Version 3.1. This section lists document types, describes what to expect in each type of document, specifies titles of documents in each category, and tells how you can obtain copies of each document.

NCAR Graphics Fundamentals

This user guide, *NCAR Graphics Fundamentals, Unix Version 4.0*, is designed to get you up and running as fast as possible. It contains information about creating and viewing graphics. It also provides short tutorials on how to use the basic functionality of each NCAR Graphics utility. This document is designed to be used with the *NCAR Graphics Contouring and Mapping Tutorial*.

NCAR Graphics Fundamentals, Unix Version 4.0 is available as PostScript file. To get the PostScript file, please contact your NCAR Graphics site representative or installer.

NCAR Graphics Contouring and Mapping Tutorial

The *NCAR Graphics Contouring and Mapping Tutorial* is a step-by-step guide to the important functionality of the contouring and geographic mapping utilities. This guide also covers Areas, an area-processing utility that allows you to fill regions, draw masked lines, and perform other useful functions. The *NCAR Graphics Contouring and Mapping Tutorial* requires basic Fortran program-writing skills and knowledge of how to generate and view graphics. Information about generating and viewing graphics appears in this guide, *NCAR Graphics Fundamentals, Unix Version 4.0.*

Programmer documents and other reference documents

Programmer documents are the software developers' reference notes; they are provided for users who want to explore the information that developers use to maintain a utility's source code. They are organized by utility or function and assume that readers have a strong background in NCAR Graphics programming.

Online man pages

The **man** pages have been completely redesigned and rewritten. There is now a short (one to six screens) **man** page for each utility in NCAR Graphics, each utility's set of parameters (if applicable), each user entry point, and each user-modifiable internal routine. These **man** pages are the most current documentation available for NCAR Graphics because they will be updated with each software release. You can access a **man** page at any time by executing:

man ***routine***

where ***routine*** is the name, in lowercase letters, of the routine (or utility, or a utility's set of parameters) for which you need information. If you have used NCAR Graphics in the past, you need to note the new spelling of some utility names. To avoid confusion between the utility and the driver routine of the same name, we have changed the spelling of the utility name where both of these were formerly identical.

How to get consulting help

If you have questions about using NCAR Graphics, contact your NCAR Graphics site representative. If you do not know who your site representative is and cannot find out from other users or the package installer at your site, you can call the NCAR Graphics Information Line at (303) 497-1201 to obtain the name of your representative. In its sole discretion, NCAR provides limited consulting support on usage and installation on an "as available" basis only. (Please refer to your Software License Agreement for a complete definition of "support.")

Chapter 1: Overview of functionality

NCAR Graphics Version 3.2 can be divided into two parts. The first part will be called "utilities" or "low level utilities" in this document. These utilities are written in Fortran and stored as object or archive libraries, and can be accessed from a C or Fortran program that you write. The second part of NCAR Graphics 3.2 is a set of commands or tools that allow you to compile your programs, run examples, view, print, and edit NCAR Graphics Computer Graphics Metafiles (NCGM) files, and more.

With Version 3.2 there are now user entry points for C as well as Fortran to facilitate new code development by C programmers. We have also added a new display option so that you can run your program, and have graphics displayed directly to as many as 15 different X Windows, or a NCGM file.

Below is a guide to the current functionality of both the NCAR Graphics utilities and the UNIX command tools.

Low level utilities

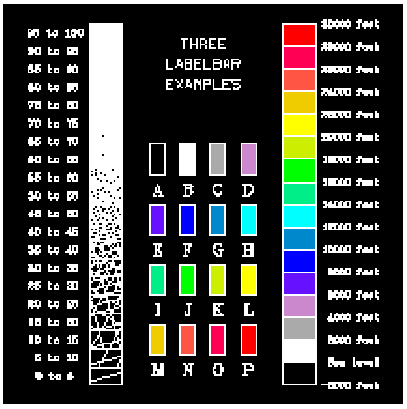
GKS

All NCAR Graphics utilities use the Graphical Kernel System (GKS) to produce output. NCAR Graphics comes with its own Level 0A implementation of GKS with some extensions to the 2B level. NCAR Graphics can run on other implementations of GKS.

Text and label bars

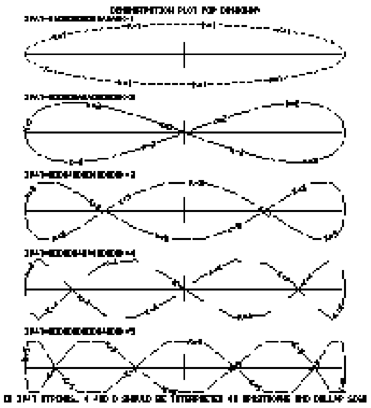
The Plotchar utility draws characters and symbols using 20 different fonts. Characters can be stroked, filled, or drawn using the printer default, giving you options on speed of printing, and very high quality fonts. Plotchar also offers Greek characters and a wide variety of mathematical symbols. With Version 3.2 we now have a utility to draw meteorological symbols and a variety of filled fonts.

The Labelbar utility helps you create a labeled, filled, rectangular bar to serve as a key for filled plots.



Curves

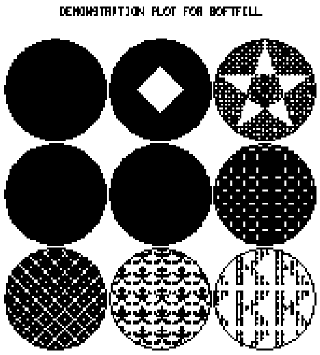
NCAR Graphics offers numerous routines for drawing lines and curves. The Dashline utility allows you a nearly infinite variety of dash and character patterns to draw lines, as well as a variety of smoothing options.



Areas

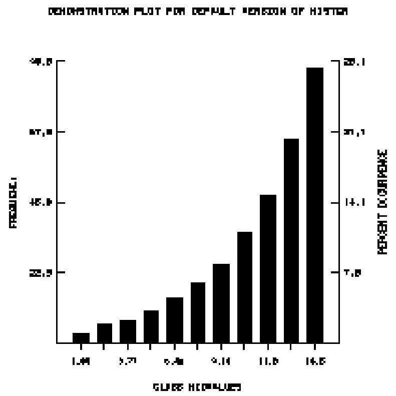
Used in conjunction with line drawing, mapping or contouring utilities, the Areas utility allows you to retrieve (or recover) outline and other information about each region. Also, Areas allows you to overlay two or more groups or sets of lines, and to recover the regions defined by the union of the groups. It can also tell you where each region is located relative to each group of lines. These features allow you to mask contour lines or latitude/longitude lines over the ocean and color-fill or pattern-fill regions, among other things. Areas is covered in detail in the *NCAR Graphics Contouring and Mapping Tutorial.*

NCAR Graphics also offers numerous options for filling areas. The Softfill utility can be used to generate nearly any kind of fill pattern or color, or GKS can be used directly to do solid color fill.



Histograms

The Histogram utility is used to generate histograms for single data arrays or for comparing the distributions of two data arrays.



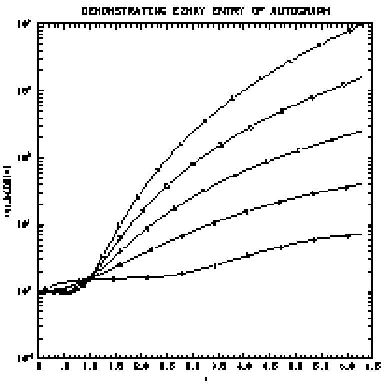
Color conversion

A color space, or color model, is a three-dimensional coordinate system where coordinates in that system represent colors. Colconv allows you to convert between several of the more common color models, namely RGB (Red, Green, Blue) used by most workstations, the YIQ model used by broadcast TV, the HLS (Hue, Lightness, Saturation), and the HSV (Hue, Saturation, Value) models.

X/Yplots

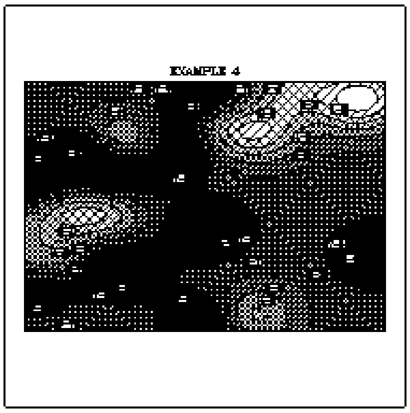
The Autograph utility helps you draw X/Y graphs of one or more curves with labeled backgrounds. Autograph provides facilities for differentiating curves, doing log scaling, and other useful options.

The Gridall utility provides nine options for generating axes, perimeters, and grids in both linear and logarithmic spaces.



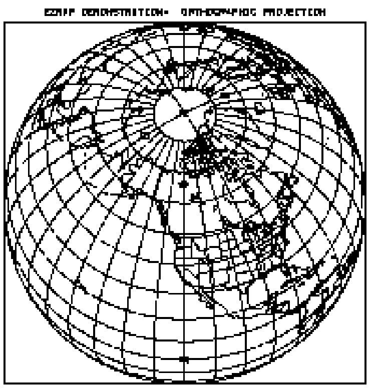
Contours

Conpack is a color contouring utility. Conpack can be used to generate plots of almost any rectangular array of data, and allows an amazing degree of control over how the contours are presented. Bivar, a bivariate interpolation package, has been added to the NCAR Graphics libraries to help users contour irregularly spaced data. In Version 3.2, we have added a cell array option to Conpack to help you draw color or shaded plots of densely packed data. This option can produce very detailed plots in much less time than other Conpack routines. Conpack and Bivar are covered in the *NCAR Graphics Contouring and Mapping Tutorial.*



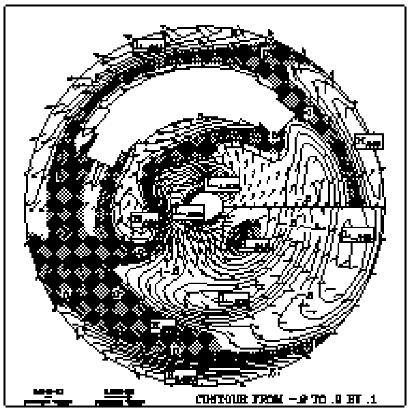
Maps

Ezmap offers ten different projections of the globe, the ability to precisely choose which portion of the globe that you would like to plot, and a selection of political and geographic outlines. In Version 3.2, inverse transformations are available to help you go back from the plotted map to your data values. Ezmap is covered in the *NCAR Graphics Contouring and Mapping Tutorial.*



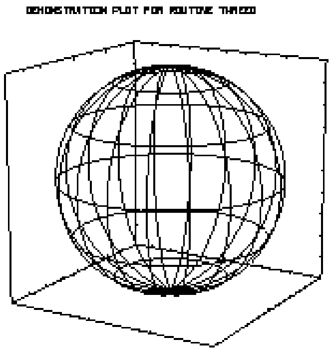
Field flows

The Vectors (previously called Velvct) and Streamlines (previously called Strmln) utilities offer two options for drawing field flows. Both utilities have been updated to allow easy mapping onto globes or other transformations, and masking over regions of your choice. The Vectors utility has been updated to color vectors according to an independent field, such as temperature.



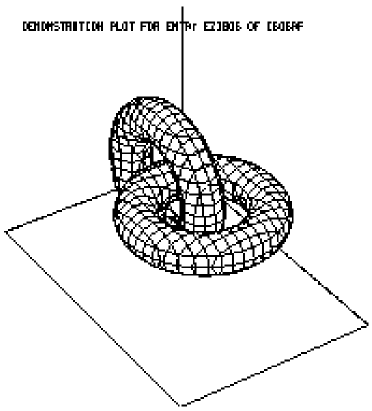
3D

The utility for drawing X/Y/Z plots is Threed. It offers options for drawing text, points, curves, and fences to nearly any perspective.



Wire frame surfaces

The Isosurface utility draws a wire frame approximation of an isosurface from a three- dimensional array. The Surface utility draws wire frame perspectives of functions of two variables.



Movies

Gflash routines store and insert picture segments that appear repeatedly in a movie, greatly reducing the amount of CPU time it takes to generate movies. The Scrolled_title utility provides options for titling movies.

UNIX command tools

NCAR Graphics offers numerous UNIX command tools to make life easier.

Run-time commands. Run-time commands, such as **ncargf77** and **ncargcc**, help compile, link, and load your program in a single step, eliminating needless typing. The **ncargex** command copies the example of your choice into your directory and executes it so that you can study and modify the code to do what you want. This command can greatly reduce development time of new codes, as well as NCAR Graphics spin-up time. See the chapter, "Five quick steps for creating and viewing your graphic."

NCGM viewing and editing. The NCGM viewing and editing tools include command-line tools for text- based terminals and Graphical User Interface (GUI) tools for the X Window System. Both viewing tools allow you to view any frame in your NCGM file in any order. The GUI interface allows you to view your graphics sequentially, animate a NCGM, and zoom in on portions of the picture for a closer look. Editing tools allow you to overlay images, merge and split NCGM files, and delete or add plots from an NCGM.

Raster viewing and editing. With Version 3.2, we have added several new utilities that allow you to view, concatenate, and resample raster images in different formats. See the chapter, "Viewing and editing your CGMs and raster images."

Graphical format conversions. We have also introduced a variety of filters that will help you view NCAR Graphics NCGM files with other graphics systems, and vice versa.

Chapter 2: Five quick steps for creating and viewing your plot

Overview

This chapter briefly discusses the necessary steps for creating and viewing a plot. For detailed information on step 2, please study the "Writing a graphics program" chapter. For more information on step 5, please study the "Viewing and editing your CGMs and raster images" chapter.

The steps

There are five steps required to create and view a plot:

1. Ensure that the environment variable NCARG_ROOT is set.
2. Write a graphics program.
3. Compile and load the program.
4. Execute the program.
5. View your plot.

Step 1: Ensure that the environment variable NCARG_ROOT is set

Before you can do anything, you must make sure that you have set either NCARG_ROOT or the three environment variables NCARG_BIN, NCARG_LIB, and NCARG_INCLUDE, depending on how NCAR Graphics was installed on your system. In some cases, the installer may have set either NCARG_ROOT or NCARG_BIN, NCARG_LIB, and NCARG_INCLUDE globally so that you don't have to worry about it. To figure out if any of these variables have been set, execute:

```
env
```

A list of all the environment variables that have been set will be shown, along with their values.

If you don't have either NCARG_ROOT, or NCARG_BIN, NCARG_LIB, and NCARG_INCLUDE set, then please contact the person who installed NCAR Graphics at your site for information on how to set them.

Step 2: Write a graphics program

A quick way to write a graphics program is to choose an example that produces output similar to what you need for visualizing your data, run **ncargex,** and modify the resulting source code to do what you want.

"Appendix D: A pictorial guide to NCAR Graphics examples" provides grayscale prints of the example plots available via **ncargex**. Choose one that does something like what you want to do, then type:

ncargex ***example***

where ***example*** is the name under the plot of your choice.

The **ncargex** command puts several files in your current working directory. The file ***example*.f** is the main program that you will want to modify, and the file ***example*.ncgm** is the NCAR Graphics Computer Graphics Metafile (NCGM) that is created when the example is compiled and executed. Other files that are created when you run **ncargex** include an executable file named ***example***, and occasionally, a file full of support routines for the main program.

Before you modify an example program, please study the "Writing a graphics program" chapter of this guide so that you understand what the different calls do. Also, when modifying an example program, we recommend that you comment out lines of code and run the example to check the result first. You may then decide to remove some of the code that you commented out: this prevents you from removing something critical, then later finding that you have to locate and replace a deleted line.

Step 3: Compile and load the program

To make it easy to compile and load your program and the NCAR Graphics libraries, NCAR Graphics provides a command to compile and load C programs **(ncargcc**) and a command to compile and load Fortran programs (**ncargf77**).

For a C program, execute:

ncargcc ***example*.c**

Where ***example*.c** is the name of the program you want to compile.

For a Fortran program, execute:

ncargf77 ***example*.f**

Where ***example*.f** is the name of the program you want to compile.

Step 4: Execute the program

Unless you've specified otherwise when compiling and loading your program, you can run your program by executing:

a.out

When you run your program, one of two things will happen depending on the workstation type you have chosen. Either output will be displayed directly on your X Windows workstation, or an NCGM will be created. By default, NCGM files are named **gmeta**.

Step 5: View your plot

NCAR Graphics now offers two output options for your plots. By setting the GKS workstation type, you can make your program output plots directly to as many as 15 different X Windows on your workstation. You can also create a single NCGM file. For details on how to choose your output option, please see the section "What you need to know about GKS workstations." If you choose the X Windows output option, you will need to set the DISPLAY environment variable as shown in examples 1 and 2 below. Then your program will display directly to your workstation, or send an error message to your workstation.

If you choose to create an NCGM file, the rest of this section gives you an overview of your viewing and editing options.

To view an NCGM file, it must be translated into commands that your terminal, workstation, or printer recognizes. The translator that does this is called **ctrans**, and it can be accessed directly or through either a command line interface called **ictrans** or a graphical user interface (GUI) called **idt**. If you are working on a terminal or workstation that is not running X Windows, you should use **ictrans**. If you are working on an X terminal or a workstation running X Windows, you will probably want to use **idt**, although **ictrans** will also work.

Viewing an NCGM on a graphics terminal

The **ictrans** command is an interactive user interface to the **ctrans** translator; **ictrans** is the preferred tool to view an NCGM on a graphics terminal. **ictrans** provides random access to the frames contained in your metafile. You need to specify the graphcap to be used, and there are two ways to do this: you can either set the GRAPHCAP environment variable or specify the graphcap with the **-d** option on the **ictrans** command line. In the following examples, the C shell is used to set environment variables. Please consult the **man** page on the shell that you are using for instructions on how to set environment variables on your system.

If you have run **ncargex** ***example***, which produces an NCGM named ***example*.ncgm**, and you are working on a Tektronix 4107 graphics terminal, you can view ***example*.ncgm** by typing either:

> setenv GRAPHCAP t4107
> ictrans ***example*.ncgm**
> ictrans> **1p**

or:

> ictrans -d t4107 ***example*.ncgm**
> ictrans> **1p**

Typing "**1p**" directs the translator to plot frame 1. When the plotting is complete, a "<READY>" prompt appears; you must then press RETURN in your terminal window before a new "ictrans>" prompt will be displayed. If the example program only produces a single frame, then entering "**2p**" will result in an error message.

Many **ictrans** commands are available for interactive viewing of your metafiles. You can find out about other **ictrans** commands by typing the abbreviation for the help command:

> ictrans> **h**

All ictrans commands can be abbreviated to the shortest unique string, just as the "**h**" above stands for "**help**."

To exit **ictrans**, type:

> ictrans> **q**

More information about **ictrans** appears on the **man** page for **ictrans** and in the "Viewing and editing your NCGMs and raster images."

Viewing an NCGM with the X Window System

The image display tool **idt** is a Graphical User Interface (GUI) to the **ctrans** translator. You need to set the GRAPHCAP and DISPLAY environment variables, then you can view your NCGM. Because X11 is network-based, **idt** can run on one system and produce its graphics locally or on a remote system. The following two examples demonstrate each of these cases.

Example 1. Local application

Create an NCGM named ***example*.ncgm** on your local workstation (called "localstation" here) by executing **ncargex *example*.** To view it on your workstation's display, execute the following three commands:

> localstation> **setenv GRAPHCAP X11**
> localstation> **setenv DISPLAY 0:0**
> localstation> **idt *example*.ncgm**

Example 2. Distributed application

You have an NCGM named ***example*.ncgm** on a computer called "mainframe," and you want to view it on your workstation called localstation. You may need to specify the complete domain name of your workstation on the network *(host.domain)*. Assuming that you are logged on to a window to the remote computer named mainframe, you would execute:

> mainframe> **setenv GRAPHCAP X11**
> mainframe> **setenv DISPLAY localstation.*domain*:0**
> mainframe> **idt *example*.ncgm**

Two **idt** windows will come up. Use the window with the "pushbutton" controls to plot your frames. Complete instructions for using **idt** appear in the chapter "Viewing and editing your CGMs and raster images."

In both of the preceding examples, we have set the environment variables using the C shell. If you are using a different shell, use the **man** page for your shell to learn how to set environment variables. If you frequently use the same type of graphics display, you can set these variables in your **.login**, **.cshrc**, or **.profile** file so that they are set automatically every time you log in. In example 2, the computer named localstation must have been configured to allow X11 requests from the computer named mainframe. See the **man** page on **xhost**, or consult your system administrator if necessary.

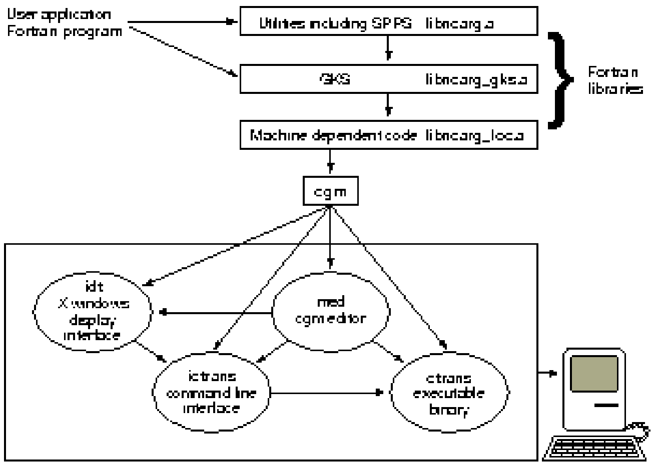
For further details about viewing and editing images, see the chapter "Viewing and editing your CGMs and raster images."

Chapter 3: Writing a graphics program

Before we can talk about the structure of a graphics program, we need to discuss the structure of NCAR Graphics. Once you understand the structure of NCAR Graphics, we will discuss writing a program in

Fortran. Finally we'll show you how to apply those concepts to a C program.

The structure of NCAR Graphics



The NCAR Graphics utilities are built in layers, the lowest of which is GKS. GKS is a specification of how to implement basic functions for computer graphics programming, such as points, lines, text, fill areas, and portable raster images. The GKS standard has been adopted by both ANSI and ISO, and can be implemented at one of 12 levels, depending on graphical input and output capabilities. NCAR has implemented the 0A level, and a few extensions from the 2B level. The GKS standard also specifies the Fortran and C interfaces for its graphics functionality. Because GKS is a standard, and all NCAR Graphics utilities rely on GKS to do the plotting, you can run NCAR Graphics with GKS packages from other vendors, or with the GKS package included in NCAR Graphics.

GKS is not invisible in NCAR Graphics. Things like setting up color tables and doing solid color fill are best done using GKS, so you will call GKS routines directly.

The next layer of NCAR Graphics is the SPPS utility. This utility was designed to do some of the things that GKS does with fewer calls. Other utilities, such as Conpack, Autograph, and Ezmap, frequently rely on you to make the right GKS and SPPS calls to work correctly. In Version 3.2, these other utilities are the highest layer of NCAR Graphics, and will do the most work with the fewest number of calls. In general, knowing which utility a particular routine belongs to is only necessary when you need to look up its documentation in one of the manuals. We've tried to alleviate this difficulty by listing routine names in the index.

Structure of a Fortran program

Every NCAR Graphics program has generally the same structure. Before you can do anything else, you must open GKS, and when you are finished plotting, you must close GKS. Once GKS is open, you, or the utility you are using, need to define a transformation between the user coordinate system (the coordinate system of your data), and the Normalized Device Coordinate (NDC) system. Then it is time to call the various routines needed to get your plot. When you are finished drawing a plot, you must close that plot or frame. Finally, you need to close GKS. In summary, your program contains:

Calls for opening GKS and activating workstations
GKS settings
Normalization transformations
Graphics code
Calls for ending the plot or frame
Calls for deactivating workstations and closing GKS

Example graphics program

In the following example, we discuss a way to do each of these steps, and point you to the chapter of this guide where you can find more detailed information.

```
1       PARAMETER (M=21,N=25)
2       REAL U(M,N), V(M,N), WRK(2*M*N)
3       INTEGER IDM
   C Generate some data.
4       CALL MKDAT(U,V,M,N)
   C Open GKS, open NCGM type workstation, activate workstation.
5       CALL GOPKS (6,IDUM)
6       CALL GOPWK (1, 2, 1)
7       CALL GACWK (1)
   C Select normalization transformation 0 (user coordinates are the
   C same as NDC coordinates), so that title is drawn at top of plot.
8       CALL GSELNT (0)
   C Call PLCHLQ to write the plot title.
9       CALL PLCHLQ (.5,.9765,'Example Streamlines Plot',16.,1 0.,0.)
   C Define normalization transformation 1, and set up linear scaling.
10      CALL SET(0.1, 0.9, 0.1, 0.9,1.0, 21., 1.0, 25.,1)
   C Tell Streamlines that SET has been called, and
   C set spacing of stream lines.
11      CALL STSETI('SET---Set Call Flag', 0)
12      CALL STSETR('SSP---Stream Spacing', 0.015)
   C Initialize Streamlines, and draw stream lines
13      CALL STINIT(U,M,V,M,IDM,IDM,M,N,WRK,2*M*N)
14      CALL STREAM(U,V,IDM,IDM,IDM,WRK)
   C Close Frame
15      CALL FRAME
   C Deactivate workstation, close workstation, and close GKS.
16      CALL GDAWK (1)
17      CALL GCLWK (1)
18      CALL GCLKS
```

Opening GKS and activating workstations

There are three calls necessary to open GKS and activate a workstation: GOPKS to open GKS; GOPWK to open a GKS workstation, which for the NCAR Graphics GKS must be either an X window or an NCGM file; and GACWK to activate the workstation.

```
CALL GOPKS (IER,ISZ)
CALL GOPWK (ID,IC,IT)
CALL GACWK (ID)
```

IER is the logical unit to receive GKS error messages, ISZ is a dummy variable, ID is a GKS workstation identifier, IC is a logical unit to receive the NCGM output, and IT is the workstation type. As shown in lines 5, 6, and 7 of the example code, we recommend that you set IER=6, ID=1, and IC=2 for NCGM output. IT must equal 1 to produce NCGM output. This does assume that you aren't using unit 2 elsewhere in your program for some other purpose.

```
   C Open GKS, open NCGM type workstation, activate workstation.
5       CALL GOPKS (6,IDUM)
6       CALL GOPWK (1, 2, 1)
7       CALL GACWK (1)
```

For complete discussions on opening and closing GKS, and on workstation types, please see Chapter 6 "What you need to know about GKS workstations."

GKS settings

GKS has numerous options that you can set using various GKS routines. Some of the most common options that you may need to set are values for a color table, line width and color, and the fill style for areas that you wnat to fill. The routines that do this are GSCR, GSLWSC, GSPLCI, and GSFAIS, respectively. Although we don't set any GKS options in this example, it is best to set up a color table right after activating your workstation. In other words, you would put your calls to GSCR right after line 7 in the example. Please refer to Chapter 7 "Color tables and color mapping systems" for more information on setting up color tables, Chapter 10 "Drawing lines and curves" for more information on setting line color and width options, and Chapter 13 "Area filling with GKS and Softfill" for more information on filling areas.

Normalization transformations (SET)

It is necessary to set up a transformation between the coordinate system of your data and the Normalized Device Coordinate (NDC) system before attempting to plot your data. The Normalized Device Coordinate system defines the lower left corner of your device to be (0.0, 0.0) and the upper right corner of the device to be (1.0, 1.0). All plots will be drawn in the largest square that can be defined on your device, unless you specify something smaller.

GKS also defines a coordinate system for your data called World Coordinates. This is a Cartesian coordinate system. Because many of our users require non-Cartesian options such as log scaling and mirror imaging, NCAR Graphics uses "user coordinates." These coordinates are the coordinates of your data, and can be ordered from largest value to smallest, smallest to largest, or put on a logarithmic scale. Since GKS does not offer a transformation between NDC coordinates and user coordinates, we recommend that you use the NCAR Graphics call SET to define the transformation.

SET has nine arguments. The first four are the left, right, bottom, and top of the viewport, or your graphics output device. Since these coordinates are in NDCs, they must be between 0.0 and 1.0. Also, the value for the right of the viewport must be larger than the value for the left of the viewport, and the value for the top of the viewport must be larger than the value for the bottom of the viewport. The next four arguments are the left, right, bottom, and top coordinates of your data. If you were plotting something like a pressure function on the Y axis, then you would simply set your largest value at the bottom, and your lowest value at the top. The final argument determines whether any log scaling will be done, as follows:

Value	Result
1	X Linear, Y Linear
2	X Linear, Y Logarithmic
3	X Logarithmic, Y Linear
4	X Logarithmic, Y Logarithmic

```
   C Select normalization transformation 0 (user coordinates are the
   C same as NDC coordinates), so that title is drawn at top of plot.
8       CALL GSELNT (0)
   C Call PLCHLQ to write the plot title.
9       CALL PLCHLQ (.5,.9765,’Example Streamlines Plot’,16.,1 0.,0.)
   C Define normalization transformation 1, and set up linear scaling.
10      CALL SET(0.1, 0.9, 0.1, 0.9,1.0, 21., 1.0, 25.,1)
```

In line 8, we call GSELNT with a value of zero. This is a shortcut for calling SET as follows:

```
      CALL SET (0., 1., 0., 1., 0., 1., 0., 1., 1)
```

This sets the world coordinate system and the NDC system to be the same, and sets the viewport to be the largest square that your graphics output device can handle. It also sets log/linear scaling to be linear in both directions.

In this case, we use GSELNT so that we can draw our title, with line 9, in a known location on the plot, rather than figuring out where the title should go relative to the coordinates of our data.

In line 10, we draw our plot on a slightly smaller square (0.1, 0.9, 0.1, 0.9) than is allowed by GKS. This leaves us room in the viewport to put a title at the top of the plot. Our data have values in the X direction from 1.0 to 21.0, and values in the Y direction from 1.0 to 25.0, as we have defined in the second set of four arguments. Also, this is a linear coordinate system in both directions, so we have set the log scaling value to 1.

More complete discussions of coordinate systems, normalization transformations, and SET appear in Chapter 8 "Coordinate systems in NCAR Graphics," and in Appendix A "The Use of X/Y Coordinates in NCAR Graphics."

Graphics code

At this point, you need to decide what utility or utilities you plan to use to plot your data. Because all NCAR Graphics utilities will draw over what has already been drawn, it is usually necessary to draw and fill in background areas first and to draw text, points, and lines last.

Carefully check the utilities you plan to use. Many of them, especially the easy or do-it-all-in-one-call routines call SET and FRAME for you. If you are overlaying more than one utility, you need to make sure that the SET routine is called only by the first utility. Also, if you want to set parameters special to any utilities, it is generally best to do that before calling the routine that initializes, or initializes and draws for that utility.

```
   C Tell Streamlines that SET has been called, and
   C set spacing of stream lines.
11       CALL STSETI('SET---Set Call Flag', 0)
12       CALL STSETR('SSP---Stream Spacing', 0.015)
   C Initialize Streamlines, and draw stream lines
13       CALL STINIT(U,M,V,M,IDM,IDM,M,N,WRK,2*M*N)
14       CALL STREAM(U,V,IDM,IDM,IDM,WRK)
```

The example code illustrates many of these points. Line 11 tells Streamlines that SET has been called previously, and line 12 sets a parameter that controls the minimum distance between stream lines. We set the Streamlines parameters before initializing Streamlines in line 13. Line 14 draws the stream lines. If we wanted to draw stream lines over a color-filled contour plot, it would be best to call STREAM after the last call to draw the contours.

An easy way to figure out which routines you need to plot your data is to look through Appendix D "Pictorial index to **ncargex**" and find a plot that looks like what you want to do. Then, in the directory where you want to work, execute:

> ncargex ***filename***

where ***filename*** is the name printed under the plot of your choice in Appendix D. **ncargex** will put several files in your current working directory. The file ***filename*.f** will contain the graphics program to draw that plot. To help you make the modifications you need to plot your data, see the tutorial chapters of this guide, the *NCAR Graphics Contouring and Mapping Tutorial*, and the **man** pages.

Ending the plot or frame

There are two ways to end a plot or a frame so that you can either draw another plot in the same program, or so that you can quit. The easiest way is to call the SPPS routine FRAME. As mentioned above, some of the do-it-all-in-one-call routines will call FRAME for you. We called FRAME in line 15 of the example. Depending on your workstation type, FRAME will do different things. Please see Chapter 6 "What you need to know about GKS workstations" for more information on ending a frame.

```
   C Close Frame
15       CALL FRAME
```

Deactivating workstations and closing GKS

Just as it took three routines to open GKS and activate a workstation, it takes three to deactivate a workstation and close GKS. GDAWK deactivates the workstation, GCLWK closes the workstation, and GCLKS closes GKS. The argument that GDAWK and GCLWK require is the workstation identifier, ID, above. It is important for you to make sure that you activate and deactivate the same workstation if you are only working with one workstation, and that you deactivate all workstations before closing GKS. Since we open workstation 1 in line 6, and activate it in line 7, we must deactivate workstation 1 in line 16, and close it in line 17.

```
   C Deactivate workstation, close workstation, and close GKS.
16       CALL GDAWK (1)
17       CALL GCLWK (1)
18       CALL GCLKS
```

For a complete discussion on opening and closing GKS, as well as workstation types, please see Chapter 6 "What you need to know about GKS workstations."

Writing a C program

Writing an NCAR Graphics program in C requires the same elements as writing an NCAR Graphics program in Fortran. Please study the first part of this chapter, "Structure of a Fortran program" for complete details about how to organize an NCAR Graphics program.

This section describes how to use the NCAR Graphics C-binding. The NCAR Graphics GKS C-binding adheres to the ISO/IEC standard, which specifies that the names for the C-bindings are more descriptive than the Fortran names. Hence, the C-binding for setting the color-fill index is **gset_fill_colr_ind**, and the Fortran binding, which can only be 6 characters long, is GSFACI. To get the name and usage of the C-binding, please see the **man** page for the corresponding Fortran routine or the **ncarg_gks_cbind man** page for a complete list of the supported GKS C-bindings. In an effort to make it easy to find the C-bindings for NCAR Graphics routines, all C-bindings for NCAR Graphics routines are of the form **c_*rname*** where *rname* is the Fortran name of the routine in lowercase.

The argument list of each GKS C-binding closely corresponds to the argument list of the Fortran routine, but in many cases, the arguments are represented in the form of a C structure. These structures are defined in the include file **<ncarg/gks.h>**. The argument list of each NCAR Graphics C-binding also closely corresponds to the argument list of the Fortran routine, except when arguments include character strings or multidimensional arrays.

The NCAR Graphics C-bindings are intended to be ANSI C compliant. To get the correct function prototypes you can include **<ncarg/ncargC.h>** and **<ncarg/gks.h>** in your program.

To compile your NCAR Graphics C program with the GKS C-bindings, use the application **ncargcc**. **ncargcc** will take care of loading in the necessary C/Fortran interface libraries as well as the NCAR Graphics C and Fortran libraries.

An example of a C program that calls some of the NCAR Graphics GKS C-bindings has been provided. To copy this C program into your directory, and then compile, link, and run it, type:

```
ncargex c_gtxpac
```

For more information on how to use the C-bindings for the NCAR Graphics utilities, please see **ncarg_cbind**(3NCARG) or the **man** page for any of the NCAR Graphics routines. If you intend to use the GKS C-bindings heavily, you may want to get a copy of the American National Standard's *Computer Graphics---Graphical Kernel System (GKS)---Functional Description.*

Chapter 4: Viewing and editing your CGMs and

raster images

Introduction

The primary token of exchange for graphical imagery in NCAR Graphics is the NCGM, which can be produced directly from NCAR Graphics plotting utilities. This chapter describes a suite of applications that may be used to display and manipulate NCGMs. Many of the applications described in this chapter may also ingest valid ANSI-standard Computer Graphics Metafiles (CGMs), provided they are transformed into NCGM format using the tools discussed later in this chapter. Finally, a limited set of tools for working with raster imagery is discussed at the end of the chapter.

The primary components covered in this chapter are:

- filters to convert CGM files to and from NCAR's privately encoded format
- **ctrans**---a translator for sequential frame display to your terminal or workstation window and for output to many hardcopy devices
- **ictrans**--- a text-based, interactive version of ctrans that may be used from any graphics terminal and the X11 windowing system
- **idt**--- a graphical user interface (GUI) to **ctrans**
- **med**--- a text-based, frame-level interactive NCGM editor
- **rasview**--- a raster file previewer for the X11 window system

```
--------------------------------------------------------------------------
 GUIs are supported by and require the X11 Window System. Text-based
 interfaces are typically equivalent in function to GUIs but only require a
 standard UNIX environment.
--------------------------------------------------------------------------
```

This chapter assumes that you have a working knowledge of the X11 windowing system environment for discussions involving GUIs. For X11 references, see *X Window System User's Guide*, by O'Reilly and Associates, Inc., or contact your system administrator.

Environment

The examples in this chapter assume that you are using **csh**, the C shell. If you are using **ksh**, the Korn shell, or **sh**, the Bourne shell, you may want to get help from your system administrator. If you are not sure what shell you are using, check with your system administrator.

This section discusses the environment variables that must be set. Many of the environment variables are invariant from one session to the next, so you may want to place them in your **.login** or **.cshrc** file. Your site administrator may have already preconfigured your environment for you. To determine which variables are already set, type:

    printenv

Search path

You must make sure that the NCAR Graphics executables and the current working directory, ".", are on your search path. You must also make sure that the appropriate NCAR Graphics environment variables are set. Please see the first step in the "Five quick steps for creating and viewing your graphic" chapter for information on setting critical NCAR Graphics environment variables.

Metafile translators

When using NCAR Graphics metafile translators you must either set the GRAPHCAP environment variable to reflect the graphics device you are using, or provide the device name as a command line argument. To determine which devices are supported, you can type:

```
gcaps
```

The general form for setting your graphcap is:

setenv GRAPHCAP ***device_name***

As an example, if you are running under X11 you would use:

```
setenv GRAPHCAP X11
```

You may want to set this variable in your **.login** or **.cshrc** file so that it will always be available. The *device_name* may also be specified on the command line for the translators, thus overriding the environment variable.

X11 users

If you are an X user, you will also need to set the DISPLAY environment variable. For example, if both your X server and your translator will run on the same local machine then type:

```
setenv DISPLAY:0
```

or possibly:

```
setenv DISPLAY unix:0
```

Check with your system administrator if you are not sure.

If you are running remotely on a machine named pebbles and your local workstation where your X11 server is running is named bambam, then you need to set your DISPLAY environment variable on pebbles with:

```
setenv DISPLAY bambam:0
```

and on bambam execute:

```
xhost + pebbles
```

Translator fonts

Finally, you may optionally want to specify a font other than the default to be used for subsequent metafile translation of text. The FONTCAP environment variable may be set:

setenv FONTCAP ***fontcap_name***

where *fontcap_name* is the desired fontcap. For a list of valid fontcaps, execute the **fcaps** command:

```
fcaps
```

For example:

```
setenv FONTCAP font10
```

specifies that the cursive font, font10, be used.

Getting started --- Translating metafiles with ctrans

This section provides several simple examples that demonstrate the basic steps for using the metafile (NCGM) translator **ctrans.** We assume that your NCARG_ROOT and possibly DISPLAY environment variables have been set correctly as discussed above. Before you start you will need an NCGM. You can create an NCGM by executing:

```
ncargex bnchmk -clean
```

which will leave the file **bnchmk.ncgm** in your current working directory.

Viewing your NCGM

Below are the steps for sequentially displaying the contents of an NCGM on three different devices. Use the command(s) appropriate for your workstation. From a Televideo vt100 graphics terminal:

```
setenv GRAPHCAP vt100
ctrans bnchmk.ncgm
```

or:

```
ctrans -d vt100 bnchmk.ncgm
```

On a workstation running the X Window System:

```
setenv GRAPHCAP X11
ctrans bnchmk.ncgm
```

or:

ctrans -d X11 bnchmk.ncgm

In all three cases, the first of six frames should appear on your screen. You can sequentially advance through the NCGM by pressing the RETURN key after each frame is plotted. In a window-based environment, such as X11, make sure that the mouse sprite is in the graphics window created by **ctrans**. Also, in a window-based environment, clicking the left mouse button has the same effect as pressing RETURN. You can terminate processing at any time by sending **ctrans** an interrupt signal (typing CONTROL-c on most systems).

Printing your NCGM

For most of the output devices supported by **ctrans**, the metafile translation process results in a stream of device-specific commands being written to the standard output. This command stream may be captured for subsequent processing. For example, the command:

ctrans -d ps.mono bnchmk.ncgm > bnchmk.ps

translates the entire **bnchmk.ncgm** metafile into PostScript and writes it to the file **bnchmk.ps**. When the standard output of **ctrans** is redirected to a file (or a pipe) in this manner, it is not necessary to press the RETURN key between frames. **ctrans** will process all the frames in the file and then exit without waiting for any user interaction. The file **bnchmk.ps** may now be sent to a PostScript printer for printing, possibly with the command:

lpr bnchmk.ps

```
------------------------------------------------------------------------
 The lpr command used above is simply an example. Printing commands
 vary widely from site to site. Your system administrator should be able
 to provide you with the printing command appropriate for your site.
------------------------------------------------------------------------
```

ctrans can also generate output suitable for printing on HP LaserJet printers that do not support PostScript:

ctrans -d hppcl bnchmk.ncgm > bnchmk.hppcl

For a complete list of supported output devices, execute the **gcaps** command.

Examining the contents of your NCGM

Sometimes it is desirable for the purposes of debugging to see what CGM commands are actually contained in your metafile. The **ctrans** clear-text driver may be used to convert your binary-encoded NCGM into a human-readable ASCII representation. For example:

ctrans -d CTXT bnchmk.ncgm | more

converts **bnchmk.ncgm** into clear text and sends the output to the UNIX pager, **more.**

The command:

ctrans -d CTXT bnchmk.ncgm | grep -c POLYLINE

counts the number of CGM polyline elements that appear in the file **bnchmk.ncgm.**

```
------------------------------------------------------------------------
 The ANSI CGM standard defines three encoding schemes for CGMs:
 binary, character, and clear text. The clear text produced by the
 ctrans -d CTXT option does not conform to the ANSI definition for clear
 text encoding and is not acceptable as input to other CGM clear-text
 interpreters. It is merely intended as a debugging aid for the user.
------------------------------------------------------------------------
```

Controlling NCGM translation

ctrans offers a suite of command line options for controlling the translation process performed on a metafile which may ultimately determine the appearance of your plot. For a complete description of these options, see the **ctrans man** page.

Interactive metafile translation

Interactive ctrans (ictrans)

ictrans, a text-based interactive version of the metafile translator **ctrans**, will read commands from any terminal. With **ictrans**, you are no longer limited to sequential metafile access. Interactive translation may be desirable if your metafile contains lots of metafile frames or if you simply want to take advantage of some of the functionality in **ictrans** that is not available in **ctrans**. You invoke **ictrans** by typing:

ictrans ***metafile_name***

where *metafile_name* is the path to a valid NCGM. As with **ctrans**, you need to specify the output device you want to translate to either on the command line as in:

ictrans -d X11 bnchmk.ncgm

or with the GRAPHCAP variable:

setenv GRAPHCAP X11
ictrans bnchmk.ncgm

Upon invocation, **ictrans** will respond with an account of the number of frames contained in the metafile followed by the ready prompt ictrans>. Be patient; many of the functions in **ictrans** require information embedded in the metafile. The time required to extract this information depends on the size of the NCGM.

The command for viewing a metafile frame is **plot** (you can type **p** for short). Typing **plot** at the ictrans> prompt will cause the first frame in the metafile to be displayed. **ictrans** is case sensitive; typing **PLOT** will not work. After the frame has been plotted, you can press RETURN to signal **ictrans** to re-enter command mode (you should see the ictrans> prompt). Entering a second **plot** command will cause the next frame, if any, to be displayed:

ictrans> **p**

Note to X users: When **ictrans** is displaying to an X window, you may find it convenient to operate in movie mode. When **ictrans** is in movie mode, a RETURN is not necessary to put **ictrans** into command mode after plotting a frame from your metafile. Enter movie mode by typing:

ictrans> **movie 0**

You can abbreviate the command **plot** by typing **p**; all **ictrans** commands may be abbreviated up to the point that they are unique. For a complete list of **ictrans** commands type:

ictrans> **help**

or just:

ictrans> **h**

Random access

So far, we have only shown sequential metafile access. **ictrans** allows for random frame access as well. To view the fifth frame in a file, type:

ictrans> **5p**

Relative addressing is also possible. To view the last frame in a file type:

ictrans> **$p**

Here, the **$** is an alias for the last frame in the metafile.

For convenience, several frames may be listed on a single command line. To view the first through third, the fifth, and the last frame, type:

ictrans> **1,3 5 $p**

After processing each frame, **ictrans** waits for you to press RETURN before proceeding with the next frame. For a complete description of **ictrans** frame-addressing syntax, see the **man** page for **ictrans**. You can read the **man** page for **ictrans** by typing:

ictrans> **! man ictrans**

(Typing **!** returns you to the UNIX shell for one command.)

Saving a metafile

ictrans allows you to save the metafile describing a frame, or set of frames, to a file. If this file already exists, the saved frames are appended at the end. If the file does not exist, it will be created for you. To save the current frame to a file named **save.ncgm**, type:

ictrans> **save save.ncgm**

The current frame is the last frame plotted. (If no frames have been plotted the current frame is the first frame in the metafile.)

If you type:

ictrans> **s save.ncgm**

ictrans will tell you that the command is ambiguous because the **s** command could stand for either **save** or **spooler**. You can, however, abbreviate **save** with **sa**. The **save** command uses identical syntax to the plot command for addressing frames (as do all **ictrans** commands). To save the third through last frame to the file **foo.ncgm,** type:

ictrans> **3,$s foo.ncgm**

The newly created file, **foo.ncgm**, may now be translated just like any other NCGM. If you type:

ictrans> **file foo.ncgm**

foo.ncgm will be read into the translator for processing. The **file** command is used to change the current file being processed.

Printing metafiles

If the necessary hardware and software exist at your site, **ictrans** may have been configured to translate metafile frames and send them to a printer. To determine whether **ictrans** has been configured for printing to a spooled device, type:

ictrans> **spooler**

If **ictrans** responds with anything other than "" (null), then there exists at least one device to which **ictrans** can send translated code. The current printing device name is given by the **spooler** command when **spooler** is executed without any arguments as above. (The **ictrans** command **alias**, without any arguments, lists all available printing devices, if any.) For example, if **spooler** responds with a device named "postscript," then to send the first three frames of your file to this device after translation, type:

ictrans> **1,3Print**

Don't forget to capitalize the **P** in **Print**. The **P** is capitalized so **Print** can be abbreviated and not conflict with **p** for **plot.**

For more information about configuring **ictrans** for printing metafiles, see the **man** pages for

ictrans(1NCARG) and **ncarv_spool**(5NCARG).

Zooming in on your plot

The ictrans **zoom** command allows you to zoom or pan any area of your plot. For example, the following command will result in the bottom-left quarter of the current frame being drawn so that it fills the entire graphics window:

> ictrans> **zoom .0 .0 .5 .5**

The two pairs of numbers, (.0, .0) and (.5, .5), specify the lower left and upper right coordinates, respectively, of the frame to be viewed. The new coordinate specification remains in place until changed with a subsequent **zoom** command. To restore normal viewing type:

> ictrans> **zoom .0 .0 1. 1.**

Batch processing

It is sometimes desirable to access **ictrans** functionality in a noninteractive, or batch, environment. For example, you may want be able to send selected metafile frames to a printer from inside a shell script. You might be able to use **ctrans** to accomplish this task, but **ctrans** provides no flexible mechanism for choosing which frames get sent; **ctrans** will send the entire file. As we have seen, **ictrans** provides a rich addressing model for selecting pieces of a metafile for processing. To facilitate the use of **ictrans** in a batch mode, **ictrans** accepts all of its commands on the UNIX command line by preceding them with a **-e** flag.

For example:

> **ictrans -e 'spooler postscript' -e '$ Print' bnchmk.ncgm**

would translate the last frame (the $ frame) in the metafile, presumably into PostScript, and send it to the spooled device named "postscript". Don't forget to enclose all **-e** options in single quotes or the shell will mistakenly interpret special characters, like $, for you.

Interactive display tool --- idt

Yet another interactive metafile translator is **idt**. What makes **idt** different from **ictrans** is that **idt** has a point-and-click or graphical user interface (GUI). It may only be used on a workstation (or X terminal) supporting the X11 windowing system.

This section provides examples of using **idt**'s more common features. For a complete description of **idt** functionality, see the **idt man** page.

Once again we assume that you have set your NCARG_ROOT and DISPLAY environment variables correctly as discussed in the "Environment" section. Furthermore, **idt** requires a device specification of X11. One way to accomplish this is by setting your GRAPHCAP environment variable to X11:

setenv GRAPHCAP X11

A note on entering text: In many of the examples that follow, you will be asked to enter text into a "text widget" that appears on your screen as part of the application. To enter text into a text widget, you must move the mouse sprite so that it is completely inside the surrounding text widget box. Depending on your system's configuration, the sprite should change appearances to confirm its position within the text widget. The Athena Text widget provides many keyboard editing commands. These commands allow you to edit the buffer. The default key bindings are patterned after those in the emacs editor. For example, typing CONTROL-k will delete from the cursor to the end of the line, and DELETE key will delete the previous character. If you are familiar with the emacs editor, you should have no trouble entering text into NCAR Graphics GUIs. If you are not familiar with emacs, you may have to experiment a little to get the hang of it.

Translating metafiles (idt)

You are now ready to examine a metafile on your X11 workstation or X11 terminal. See the preceding section "Getting started---Translating metafiles with **ctrans**" for instructions on creating a metafile. The following examples assume you have already created a metafile and its name is **bnchmk.ncgm.** We also assume that you have not modified any of the X11 resources that describe the appearance of **idt.**

Invoke **idt** by typing:

idt bnchmk.ncgm

Two windows should appear on your screen. The position and dimensions of these windows may be adjusted with the mouse after it appears, or it may be pre-set through your X resource file. The details for manipulating window geometry after the window appears depend on the window manager you are running. See your system administrator, or consult the document "The X Window System at NCAR," which offers some guidelines for customizing your X11 environment.

The idt control panel

The control panel (Figure 1) provides a simple mechanism for managing displays (Figure 2). A display represents a metafile, a drawing canvas, and a set of functions for determining how the metafile is plotted on the drawing canvas. The control panel simply allows you to create and destroy displays. Before you can "create" a display, you need to "select" a metafile. A metafile can be selected either by using the <select file> button on the control panel or by placing its name on the command line. Invoking **idt** with a metafile name as a command line argument not only "selects" the metafile but also "creates" the display.

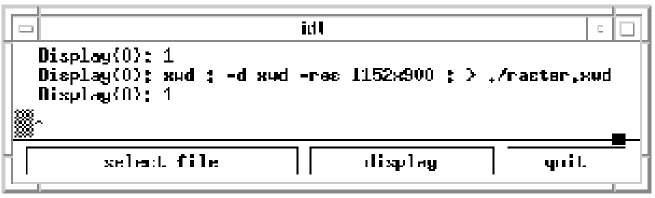


Figure 1.

The display panel

On most systems, the name of the metafile appears in the title bar at the top of the display. The first row in the display contains a scroll bar for randomly selecting a frame to be translated. The label bar to the right displays the number of the frame to which you have scrolled. The display is intended to resemble a video tape recorder. Thus, the second row contains <playback>, <jogback>, <stop>, <jog>, and <play> buttons, respectively. The third and fourth rows contain more complex display commands. See Figure 2.

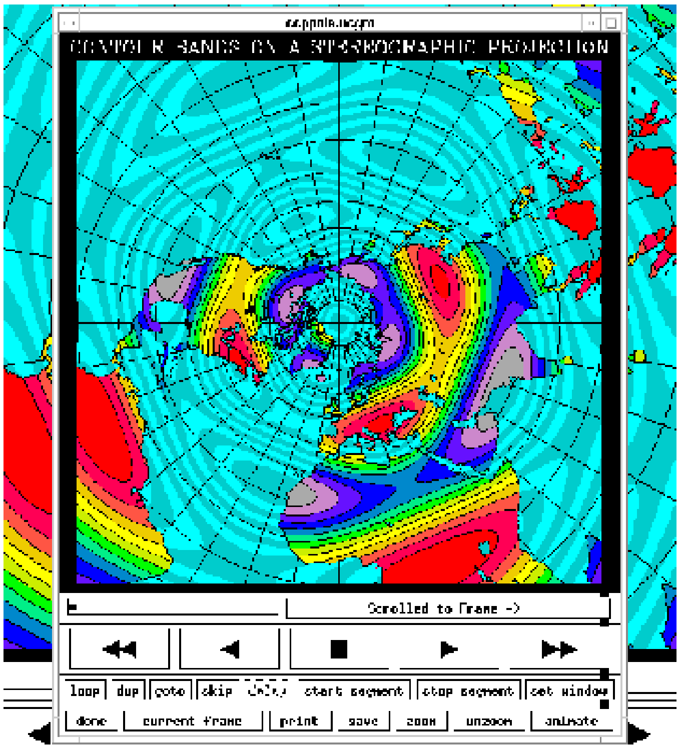


Figure 2.

If you select the <jog> button (the fourth button from the left on the second row), the first frame in the file should be displayed. Selecting this button a second time will cause the second frame to be displayed, and so forth. You can go backward through the file one frame at a time by selecting the <jogback> button, the second button from the left on the second row. The buttons on either end of the second row are for continuous playing in either the reverse or forward directions, respectively. The <stop> button in the middle halts either <play> or <playback>.

There are two ways to randomly access frames in your metafile: the first is by using the scrollbar in the upper left corner of the display panel. The second is by using the <goto> button in the third row. When you select <goto>, a small popup dialog box appears and prompts you for the number of the frame to display. If you enter 2 and then select <ok> from the <goto> popup, the second frame in the metafile will be displayed.

idt segments <start segment> <stop segment>

It is sometimes convenient, particularly with large NCGMs, to work with only a subset of the frames contained in the file. The <start segment> and <stop segment> buttons allow you to define such a subset, called a segment. Initially, the defined segment is the entire file. The <start segment> and <stop segment> buttons allow you to redefine the starting and ending frame boundaries, respectively, of the current segment. Subsequent **idt** commands will treat the metafile as if it only contained the frames defined by the current segment. For example, if you select the <play> button after setting <stop segment> to 4, playing will halt after the fourth frame in the file is displayed.

Continuous playing <loop>

The <loop> button toggles the continuous play mode of **idt** on and off for subsequent "plays" When **idt** is in continuous play mode, the currently defined segment is treated as if it were circular. There is no longer a beginning and ending frame in the segment. Selecting <play> (or <playback>) will cause **idt** to display all the frames in the current segment, and then return to the first (or last) frame and start over. Playing will not stop until aborted by selecting the <stop> button.

Printing frames <print>

Your site may have facilities for printing translated metafiles. **idt** may provide an interface to these devices, depending on how the application has been configured. Selecting the <print> button will present you with a menu of the known printing devices for which **idt** has been configured. If you select <print> and a menu appears, you may select a printer by dragging the mouse sprite down to the desired menu item and releasing the mouse button. Selecting a printer sends the current frame to that device. New printers may be added dynamically to **idt**. Consult the **man** page on **ncarv_spool**(l) for details.

Saving metafiles <save>

The <save> button allows you to write the metafile describing the current frame to a file. This file can translated at a later time. When you select <save>, a dialog popup appears asking you for the name of the file to save to. Enter the name of the file and select <ok>. If the file already exists and it is a valid NCAR CGM, the current frame will be appended to this file. If the file exists and it is not an NCAR CGM, it will be overwritten. If the file does not exist, it will be created.

Enlarging a plot <zoom>

The <zoom> button is used for zooming and panning an area displayed in the drawing canvas. For example, suppose you want the upper left corner of the currently displayed plot to be blown up to fill the entire drawing canvas. Select <zoom> from the display panel. The mouse sprite should change its shape, signaling to you that you are in zoom mode. Move the sprite to the center of the drawing canvas. Click and hold any of the mouse buttons. Move the sprite to the upper left corner of the viewing window and release the mouse button. The rectangular area described by the sprite motions will be redrawn to fill the entire viewing window. To unzoom your plot, select the <unzoom> button from the fourth row of the display panel.

Desktop animation <animate>

One of **idt**'s most useful features is its ability to perform animation on the desktop. Before we discuss how to use this powerful tool, it may help to briefly explain the hows and whys of desktop animation.

You may have already noticed when experimenting with the <play> and <playback> buttons that no matter how fast your X server is, the ensuing sequence of images displayed to your screen does not make a very acceptable animation: To your eye it appears as though images flicker and fail to proceed along in a smooth, animated motion. This is due to an anomaly known as "temporal aliasing." Temporal aliasing results partially from the fact that whatever is drawn on the drawing canvas (lines, filled areas, etc.) is immediately visible. Another factor is the speed at which objects are drawn. As the complexity of an image increases, so does the drawing time.

One way to help alleviate temporal aliasing is to postpone the display of an image until it is completely drawn. This is how the animate function works in **idt**; images (or frames) are predrawn into an area of memory managed by the X server. When it is time to display each frame in an animation sequence, the entire frame can be posted to the drawing canvas instantly. The rate at which successive images can be posted in this manner is not affected by the complexity of the image, but rather on its spatial resolution (width and height in pixels). If the images are small enough in terms spatial resolution, the X server will be able to display them rapidly and at a constant rate to produce a smooth, flicker-free animation.

There is one drawback to this approach, however: the amount of RAM (random access memory) required to store each predrawn image. RAM is a limited resource on any workstation or X terminal. Be aware that it is the X server memory that is of concern here. X servers run on the workstation that is physically attached to your graphics display.

On most systems, the amount of memory required by an animation sequence can be computed by multiplying the number of pixels in the drawing canvas by the number of frames in the sequence. For example, if your drawing canvas is 512 by 512 pixels and you have 10 frames, then the total memory required is about 512 * 512 * 10 = 2,621,440 bytes. The amount of memory that is actually available to your X server will vary depending on how much physical memory is installed on your machine and what other processes are currently running. So how do you determine how "big" an animation you can run? Experimentation is probably the best method. Start with something small, just a few frames, then move up. If the animation gets too big, there will be a noticeable degradation in performance.

One final word of caution: UNIX processes typically do not free up dynamically allocated memory for other processes to use until they exit. Using **idt** to animate imagery may cause your X server to grow substantially. On some systems, this is a trivial issue because each console login session starts a new X server. On other systems, it may be necessary to restart your server manually if it grows too big. Check with your system administrator. Your system administrator should also be able to tell you how to monitor the size of your X server.

Below are the steps, and some guidelines, for successfully animating a sequence of metafile imagery.

The first step is to determine how many and which frames you want to animate from your file. As discussed, it’s likely that it will not be possible to animate your entire file at once. Fortunately, **idt** provides a convenient mechanism for selecting only a portion of your image file: segments (see the section on **idt** segments). When you enter animate mode, only the currently defined segment is preprocessed and loaded into server memory. By default, the current segment is the entire metafile, so you may need to define a new segment. Defining a new segment is a simple process and was discussed earlier in the "**idt** segments" section. We’ll assume we are still working with the **bnchmk.ncgm** file, which only contains six frames. For this example, we will animate the entire file. You may want to check the value of the <start segment> and <stop segment> to verify that the current segment is indeed the entire file.

The next step is to select the <animate> button. **idt** will begin processing the frames in the segment and loading them into server memory. For our example, this should only take a few seconds. For a really large segment, this process might take several minutes (**Note**: the total preprocessing time also depends on the size of the drawing canvas.). The text window in the control panel will keep you informed of **idt**'s progress.

When the preprocessing is complete, the text window in the control panel will display the message "Done loading *x* images," where *x* is the number of frames in your segment. Now you can start your animation. The animation is controlled the same way that the display of normal metafile imagery is controlled. Selecting the <play> button will "play" your animation in the forward direction. The <playback> button will cause your animation to play backward. If <loop> is turned on, the animation will loop repeatedly until the <stop> button is selected.

One animation feature that is only available in animation mode is the <delay> button. If your animation is moving too fast, you can slow it down by setting a delay time, in 1/100's of a second. By default, the delay time is 0.

To terminate your animation and go back to normal metafile processing, select the <animate> button a second time.

Terminating a plotting session <done>

When you are done viewing a particular metafile, you can terminate the display session by selecting the <done> button. Terminating a display session does not terminate the entire application. To exit **idt**, you may select <quit> from the main control panel at any time.

The idt file browser

Once **idt** is up and running, you may want to take advantage of **idt**'s file browser for selecting metafiles for translation. **idt** has a built-in file finder that you can access by selecting the <select file> button from the control panel. Selecting this button at any time causes a popup menu labeled "file finder" to appear. The names of the files in your current working directory should be listed in the text box in the middle of this file selector. If you created the **bnchmk.ncgm** metafile earlier, you should see it listed somewhere in this box. You may need to use the scrollbar to bring it into view. When you have found it, select **bnchmk.ncgm** with the mouse. The string **bnchmk.ncgm** should appear in the box labeled "selection," indicating that **bnchmk.ncgm** is the current selection.

The uppermost box of the popup file selector contains a text widget with an asterisk (*) displayed. The asterisk is the metacharacter used in both the Bourne shell and the C shell for matching zero or more characters in a filename. Hence all the files in the current directory are displayed in the middle box. You can change the mask for the file finder by simply editing the file finder text widget and selecting <finder>. For example, if you enter:

    ***.ncgm**

and select <finder> or press RETURN, all of the files in the current directory with the ".ncgm" name extension will be displayed. Or, if you enter:

/u1/joe/*

all of the files in the user directory **/u1/joe** will be displayed.

When the desired filename is displayed in the selection box, you may confirm your choice by selecting <ok>. You have now informed **idt** of the name of the next metafile to process. **idt** will pop up a new display panel with which to plot the new metafile.

Metafile editing

The med program

The **med** program is a frame-level metafile editor similar in style and syntax to **ed,** the UNIX line editor. **med** allows you to interactively modify metafiles at the frame level. It is important to understand that the smallest atomic object that **med** can work with is a frame. **med** does not allow you to change the appearance of individual frames, except when you composite frames on top of each other.

All of **med**'s editing functions are performed in a temporary buffer. This greatly improves the speed with which **med** can perform its actions. Changes made in the buffer have no effect on the original file until the buffer is written to disk with the **write** command. If you terminate a **med** session without saving the buffer, all changes will be lost.

Under normal invocation, **med** will enter command mode and display the prompt med>. The prompt is a signal that **med** is ready to accept commands from the user. The basic format of a **med** command is:

[***address***] **command**

where ***address*** is an optional field listing a range of frames on which to perform the named command. Frames are numbered starting with 1. A comma-separated pair of numbers specifies an inclusive range of frames on which to operate. If no address is given, the current frame is assumed as default. As with **ictrans, med** commands may be abbreviated up to the point that they are unique. A complete description of addressing syntax is given in the **man** page for **med**.

Assuming you have an NCAR CGM named **bnchmk.ncgm**, invoke **med** with the command:

med bnchmk.ncgm

med will respond with the number of frames in the file. The current frame is set to the last frame in the file. (Note that this is different from **ictrans,** which uses the first frame in the file as the default current frame). If **bnchmk.ncgm** was created with the **ncargex** utility described earlier, **med** should report the existence of six frames.

Listing buffer contents (print)

To list the contents of the buffer at any time, use the **print** command. The **print** command will report a frame's offset, that is, its relative position in the file, the number of 1440-byte records contained in the frame, the offset of the first record of the frame in the file, and an optional description of the frame if it exists. To list the entire contents of the buffer containing the **bnchmk.ncgm** metafile type:

med> **1,6print**

A listing for each of the six frames will appear.

Writing frames to a file (write)

To write the first three frames in **bnchmk.ncgm** to a file called **test.ncgm**, type:

med> **1,3write test.ncgm**

The first three frames in **bnchmk.ncgm** will be written to **test.ncgm**. If the file already exists, it is overwritten.

Appending frames to a file (append)

To append a frame to an existing file, use the **append** command. If you use the **write** command as in the previous example, the file will be overwritten and will only contain the sixth **bnchmk.ncgm** frame. To append the sixth frame of **bnchmk.ncgm** to the end of the file **test.ncgm**, type:

med> **6 append test.ncgm**

The file **test.ncgm** now contains the first through third and the sixth frame from the file **bnchmk.ncgm**.

Reading a file into the buffer (read)

To read the contents of another NCGM into the current buffer without destroying the buffer contents, use the **read** command:

med> **read test.ncgm**

This will read all of the frames contained in the file **test.ncgm** and append them after the current frame. **med** will report the number of frames read successfully. A destination other than the current frame offset may be specified as the target for the incoming frames. The command:

med> **1 read test.ncgm**

will append the contents of the file **test.ncgm** after the first frame in the buffer. The first frame in **test.ncgm** will become the second frame in the buffer.

Splitting a large metafile (split)

Suppose you have a single NCGM that contains a large number of frames. Since large NCGMs can be quite unwieldy, it is often more convenient to have several smaller files than one huge one. The split command can be used to divide a single file into an ordered sequence of smaller ones. For example, suppose you were editing a large metafile with 1,000 frames in it. You could use the command:

med> **1,$ split 10**

to create a sequence of files named **med001** through **med010**, each containing one hundred frames. **med001** would contain the first hundred frames from the original file, **med002** would contain the second hundred frames, and so forth.

The number of split files created does not have to be an integral divisor of the total number of frames. **med** will do its best to keep the same number of frames in each split file.

Copying frames (copy)

To duplicate a set of frames and place them somewhere else in the metafile, use the **copy** command. The general format of the **copy** command is:

[***source_frames***] **copy** [***destination***]

where ***source_frames*** is a list of frames to copy and ***destination*** is the destination frame of the copy. The frames copied are appended after destination. For example, to make a copy of the first two frames and place them at the end of the file, use:

med> **1,2 copy $**

In this example, the $ is an alias for the last frame in a file. This is helpful if you do not know how many frames are contained in the NCGM.

Deleting frames (delete)

To remove some frames from the buffer, use the **delete** command. To delete the last two frames in the buffer:

med> **$-1,$ delete**

The $-1 denotes the offset of the last frame minus 1, that is, the second from last frame. The second $ denotes the last frame, so $-1,$ denotes the last two frames in the buffer.

Moving frames (move)

The **move** command can be used to change the relative position of frames in a metafile. Like the **copy** command, the general format of the **move** command is:

[***source_frames***] **move** [***destination***]

To move the first frame to the end of the file, making the second frame the first frame and the original first frame the last frame, type:

med> **1 move $**

Compositing frames (merge)

The **med** utility can do a limited amount of compositing with the **merge** command. **merge** will append the drawing instructions of one frame to the end of another frame. When plotted, the resulting image appears as if one frame were composited on top of another. The command:

med> **1,2 merge**

appends the drawing instructions contained in frame 2 to frame 1. This changes frame 1, and frame 2 is unaltered.

```
---------------------------------------------------------------------
 This command may produce unexpected results depending on how
 color and GKS point, line, and text attributes are specified in the
 metafile.

---------------------------------------------------------------------
```

Editing a new file (edit)

The **edit** command causes the entire contents of the buffer to be erased and replaced with a new metafile. The command:

med> **edit test.ncgm**

will read the file **test.ncgm** into the buffer.

The **edit** command has a safeguard to prevent you from accidentally destroying changes made to a buffer that have not been saved. If unsaved changes exist, the **edit** command will fail and print a warning message. To prevent this, save the file before executing **edit:**

med> **w**

or append a **!** to the **edit** command. This tells **edit** to disregard any unsaved changes:

med> **edit ! test.ncgm**

Obtaining help (help)

The **help** command, without any arguments, provides a list of all available commands and a brief synopsis of what each command does. It will also give you the appropriate syntax for a particular command. For example, if you type:

med> **help delete**

med will describe the syntax for the **delete c**ommand.

Batch processing

Like **ictrans, med** offers a batch or command line processing capability that is sometimes preferable to the interactive approach.

Like **ictrans, med** performs commands from the command line using the **-e** option.

For example, when issued from the command line, the command:

med -e '1,$ split 3 bnch' bnchmk.ncgm

would split the file **bnchmk.ncgm** into three smaller files: **bnch001.ncgm**, **bnch002.ncgm**, and **bnch003.ncgm**.

The command:

med -e 'r bnch001.ncgm' -e 'r bnch002.ncgm' -e 'r bnch003.ncgm' -e 'w all.ncgm'

would concatenate the **bnch*** files into a single metafile, **all.ncgm**. The **all.ncgm** file should be identical to the original **bnchmk.ncgm**.

Raster imagery

Although the NCGM is the predominant image format in NCAR Graphics, many other visualization and imaging applications work in the realm of raster imagery.

```
--------------------------------------------------------------------------
 Raster imagery is a set of picture elements (pixels) arrayed on a grid;
 these are sometimes called pixmaps. Each pixel in the image contains
 color information specifying the appearance for that particular pixel.
 Raster images have an inherent spatial resolution determined by the
 number of pixels in the grid. By contrast, page-description imagery,
 such as NCGM, CGM, or PostScript, contains a set of instructions for
 drawing a graphic. There is no inherent spatial resolution for page-
 description formats.
--------------------------------------------------------------------------
```

Raster file formats

Raster imagery can be stored in many different formats, and each format may have provisions for several different image encoding schemes. NCAR Graphics attempts to support only the handful of formats most commonly used in scientific visualization. In general, before any of the raster applications in NCAR Graphics can read or write a raster file, the application needs to know what format the file is stored in. By default, this information is determined by looking at the file's name extension. For example, a file named **foo.sun** would be expected to contain a Sun-formatted raster file. A complete list of supported raster file formats and encodings, along with the filename extensions that distinguish them, appears in the **ras_formats man** page.

Because raster utilities often need to read or write raster data from the standard input or output, in which case filenames are not visible to the application, all NCAR Graphics raster commands accept a command line option for specifying file formats that overrides any filename extension. The **-ifmt** option specifies the format of a file to be read. The **-ofmt** option specifies the format of a file to be written.

Raster utilities

NCAR Graphics includes a small collection of utilities for working with raster imagery, and these can help to bridge the gap between NCAR Graphics and other application software. The list below provides a brief summary of what is available:

rasview displays raster images on your workstation

rascat converts raster formats, concatenates files, resizes imagery, and inserts new color palettes

rassplit splits a multiframe raster file into single-frame raster files

rasgetpal extracts the color palette from a raster file

rasls lists raster files

Viewing raster files (rasview)

You may use the **rasview** utility to preview raster files on your X workstation or X terminal. **rasview** is analogous to the metafile previewer, **ctrans**, described in the "Viewing your NCGM" section above. Assuming you have a Sun-formatted raster file named **foo.sun** that you want to preview, you would invoke **rasview** with the command:

```
rasview foo.sun
```

or:

```
rasview -ifmt sun < foo.sun
```

Make sure you place the mouse sprite inside the **rasview** window after you invoke it. **rasview** will not show an image's true colors unless you do so. If you do not have a raster file available, you may create one by converting (rasterizing) an NCGM with the **ctrans** utility. For example, the command:

```
ctrans -d sun bnchmk.ncgm > foo.sun
```

would convert the NCGM **bnchmk.ncgm** into a Sun-formatted raster file named **foo.sun**, at the default resolution of 512x512.

If the raster file contains multiple images, clicking the left mouse button in the **rasview** window will advance to each successive image. In the event that you would just like the images to be displayed to your window as quickly as possible, you can specify the **-movie** option. The following command invokes **rasview** so that it displays all of the frames in the Hierarchical Data Format (HDF) file

temp.hdf to your window without waiting for a mouse-click between frames:

rasview -movie temp.hdf

The **rasview** utility will also allow you to specify a different color table from the one supplied with the image. This is only true for "indexed-color" imagery and does not work for "true-color" or "direct-color" imagery.

Assuming you have an HDF raster file with indexed-color imagery in it, and you've created a new color table using the National Center for Supercomputing Applications' (NCSA) XImage, you can tell **rasview** to use the new color palette:

rasview -pal new.pal temp.hdf

Color palettes are discussed in detail in the "Color palettes" section.

Raster file format conversion, concatenation, and resizing (rascat)

Sometimes it is necessary to convert from one image format to another. Perhaps you have a multisequence NCAR Raster Interchange Format (NRIF) file that you would like to process with the XImage utility from NCSA. XImage only accepts HDF raster files. If your NRIF file is named **bar.nrif** you could convert it to HDF using the **rascat** command:

rascat bar.nrif -ofmt hdf > bar.hdf

or:

rascat bar.nrif -output bar.hdf

rascat will also allow you to concatenate several raster files of different formats into a single raster file. The raster files must all be the same size and encoding type (indexed, for example). The following command demonstrates the concatenation of an HDF file, a Sun file, and an X Window Dump (XWD) file to a Sun file:

rascat title.hdf temp.sun credit.xwd -output all.sun

The **rascat** utility also provides resolution conversion. Let's assume you have a Sun raster file named **video.sun** that has a resolution of 640x480 pixels, and that you would like to shrink it to 240x180. (Note that resizing can "warp" an image; these resolutions were carefully selected to maintain the same image aspect ratio.) Type:

rascat -res 240x180 video.sun -output small-video.sun

If you have indexed-color imagery, rascat will also allow you to insert a new color palette during conversion. (For information on color palette formats, see the "Color palettes" section.) Building on the previous example, we'll have **rascat** both scale the image down and insert a new color palette with the output image:

rascat -res 240x180 -pal new.txt -output small-video.sun

Listing raster files (rasls)

The **rasls** command produces a listing of raster files along with information that is unique to raster files, such as resolution, encoding schemes, and number of frames in the file.

The lines below are the output of **rasls** when it was run in a directory that contained several different types of raster files.

rasls *

```
Sun       RAS_DIRECT 2816X2176       above.256.sun
Sun       RAS_INDEXED 215X215       all.sun
X Window Dump       RAS_INDEXED 512X512       coex03.xwd
Abekas RGB       RAS_DIRECT 720X486       new.a60
NRIF       RAS_INDEXED 640X480       new.nrif
AVS       RAS_DIRECT 440X440       sample.avs
SGI       RAS_DIRECT 646X485       earthstars.rgb
```

Color palettes (rasgetpal)

Indexed-color imagery is stored in a raster file along with a color palette, which tells what color each pixel is. True-color, or red-green-blue (RGB) imagery, on the other hand, does not have a color palette. The remaining discussion on color palettes therefore applies only to indexed-color imagery.

Color palettes can be stored in a file in two formats, binary and textual. Palettes stored in binary format must have a file extension of **.pal**, and palettes stored in textual format must have a file extension of **.txt**. Detailed information on both of these formats is provided later in this section.

NCAR Graphics utilities **rasview** and **rascat** will both accept an option that specifies an alternate color palette. The **rasgetpal** utility lets you extract a color palette from an existing raster file.

The example below demonstrates the extraction of a textual color palette from the raster file **temp.hdf**:

rasgetpal temp.hdf temp.txt

Now that you have a textual copy of the color palette in **temp.txt,** you could change it with a standard text editor. You can now use **temp.hdf**'s color palette, **temp.txt,** to view **pres.hdf**:

rasview -pal temp.txt pres.hdf

Binary color palette format: The binary format is compatible with NCSA HDF and must have the ".pal" file extension. The palette has a red, a green, and a blue table, each with 256 single unsigned byte entries. A 0 represents no intensity for a given color, and a 255 full intensity. In the file, the palette is stored as:

R0 R1 ... R255 G0 G1 ... G255 B0 B1 ... B255 for a total of 768 bytes.

Textual color palette format

The textual format can be created by a program or manually with a text editor. It provides for 256 RGB entries, but it also allows for a sparse color table. In the latter case, missing values are calculated from those provided using linear interpolation. If unspecified, the 0 entry defaults to black and the 255 entry to white. Each color palette entry has the following format:

> ***index_number red green blue***

where *index_number*, *red*, *green*, and *blue* are all integers in the range 0 to 255.

Examples of textual color palettes

Setting entries:

Set color index 0 to black: 0 0 0 0

Set color index 255 to white: 255 255 255 255

Grayscale ramp:

Let’s suppose you have a file named **my.txt** and it has just two lines in it: the black entry and the white entry shown above. This will produce a grayscale color palette, where the table starts at black and ramps up to white; since only entries 0 and 255 are specified, the remaining indices are interpolated.

0 0 0 0

255 255 255 255

(**Note**: An empty palette file (such as **empty.txt**) is legal and, since the 0 index defaults to black and the 255 index to white, produces a grayscale ramp just as in the preceding example.)

Temperature color scale:

Let’s suppose you want entry 0 to be black, and the remaining indices to start at 1 with a fully saturated blue, then shift linearly to 255 with a fully saturated red. Only three entries are required in your file, **temp.txt**.

0 0 0 0

1 0 0 255

255 255 0 0

Splitting multiframe raster files into single-frame raster files (rassplit)

Sometimes it is necessary to extract single frames from a multiframe raster file. NCAR Graphics provides a utility called **rassplit** that provides this function. Let's assume you have a Sun raster file called **temp.sun** and it contains several frames. You can split it into individual frames with the command:

rassplit temp.sun

This command will produce **temp.0001.sun, temp.0002.sun**, and so on for all the frames originally in **temp.sun.**

Now, to produce files from frames 5 and 7 of **temp.sun**:

rassplit -f 5 7 temp.sun

And, building on the previous example, to produce files from frames 5 and 7 but store the result as an HDF file:

rassplit -f 5 7 -ofmt hdf temp.sun

Interoperability with other tools

NCAR Graphics provides utilities to facilitate interoperability with other software. In essence, these tools may provide a bridge between NCAR Graphics and some of the more ubiquitous software packages that may be available at your site. This section identifies these tools and takes a brief look at how they are commonly used.

Non-NCAR Computer Graphics Metafiles

CGMs that were not created by NCAR Graphics are not suitable for processing by NCAR Graphics applications. NCGMs contain information not present in vanilla CGMs; this information helps to speed up the parsing of the file. This parsing information must be added to "vanilla" CGMs before they can be processed by NCAR Graphics applications. The command:

cgm2ncgm < vanilla.cgm > ncar.ncgm

will read in a vanilla CGM named **vanilla.cgm** and produce an NCAR CGM named **ncar.ncgm.**

Similarly, NCGMs cannot be parsed by non-NCAR Graphics applications. To convert the NCGM **ncar.ncgm** to the vanilla CGM **vanilla.cgm,** use:

ncgm2cgm < ncar.ncgm > vanilla.cgm

```
----------------------------------------------------------------------
 A converted CGM may not produce entirely correct results when
 translated. NCAR Graphics translators operate on a subset of elements
 of the CGM standard (as do many CGM translators) and thus may
 encounter valid CGM elements that they do not know how to translate.
 Similarly, NCGMs converted to vanilla CGMs are not guaranteed to be
```

```
 correctly translated or even parsed by third-party translators.
----------------------------------------------------------------------
```

Rasterizing NCGMs

Many commercial and public-domain scientific visualization packages accept raster imagery as an input source. The primary image format used by NCAR Graphics is the NCGM. However, you may use **ctrans** to convert your NCGM into a number of different raster image formats. The general way to make such a conversion is:

ctrans -d ***forma*****t -res** ***resolution metafile > raster_file***

For example, the command:

ctrans -d hdf -res 512x512 myfile.ncgm > myfile.hdf

converts the NCGM **myfile.ncgm** into an HDF raster file, **myfile.hdf**, with a 512x512 pixel resolution. HDF files are accepted by the NCSA application XImage.

The command:

ctrans -d avs -res 200x200 myfile.ncgm > myfile.avs

converts **myfile.ncgm** into a 200x200-pixel AVS file. The AVS raster file format is suitable for ingestion by the Application Visualization System. Finally, an SGI raster file may be created with a command like:

ctrans -d sgi -res 200x200 myfile.ncgm > myfile.sgi

SGI raster files are understood by Silicon Graphics Inc.'s Explorer.

Chapter 5: Introducing the tutorial chapters

Purpose of the tutorial chapters

The following chapters are brief tutorials to help you get started using all of the major features of NCAR Graphics. Since we cannot provide examples and detailed documentation on how to use all of the functionality in NCAR Graphics, we present only the most important features of each of the utilities. For this reason, please study this document to learn how each utility operates, then use the **man** pages for more complete information about additional functionality.

NCAR Graphics software is most commonly used for drawing contours over geographic maps, then filling the resulting regions. The NCAR Graphics utilities most critical to these operations, Areas, Ezmap, and Conpack, require more documentation than the other utilities presented in this guide. We provide a separate document, *NCAR Graphics Contouring and Mapping Tutorial*, that contains much more detail and many more examples than we can offer here. This separate document is available in hardcopy only; ordering instructions appear at the end of the preface of this guide.

The following sections of this guide each discuss a major graphics function like "Drawing lines and curves" and "Drawing text and symbols." Each section contains modules that demonstrate how to perform a specific task in NCAR Graphics like "How to set a dash pattern." The modules are organized as shown on the next two pages: most are laid out as two-page spreads so that all relevant information is available to you at a glance.

Generally, the first modules in a section explain basic functionality, then later modules build increasingly complex examples.

Reference guides throughout

Each section starts with a series of quick reference tables that show all user entry points and parameters for the utility or function being described. More detailed reference materials are included as appendixes.

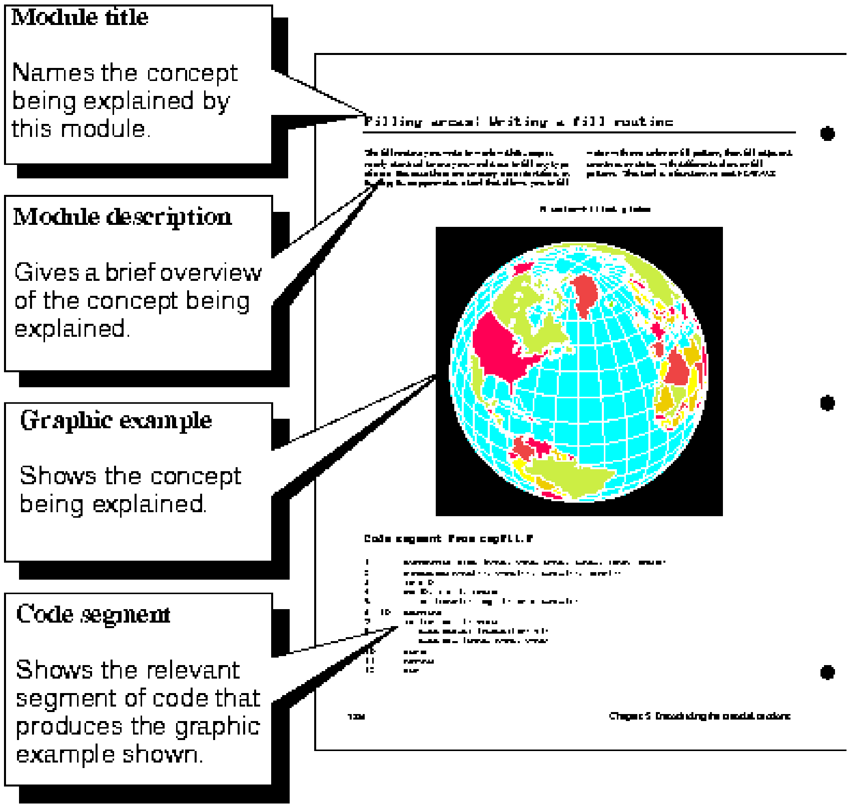
Instructions for performing tasks

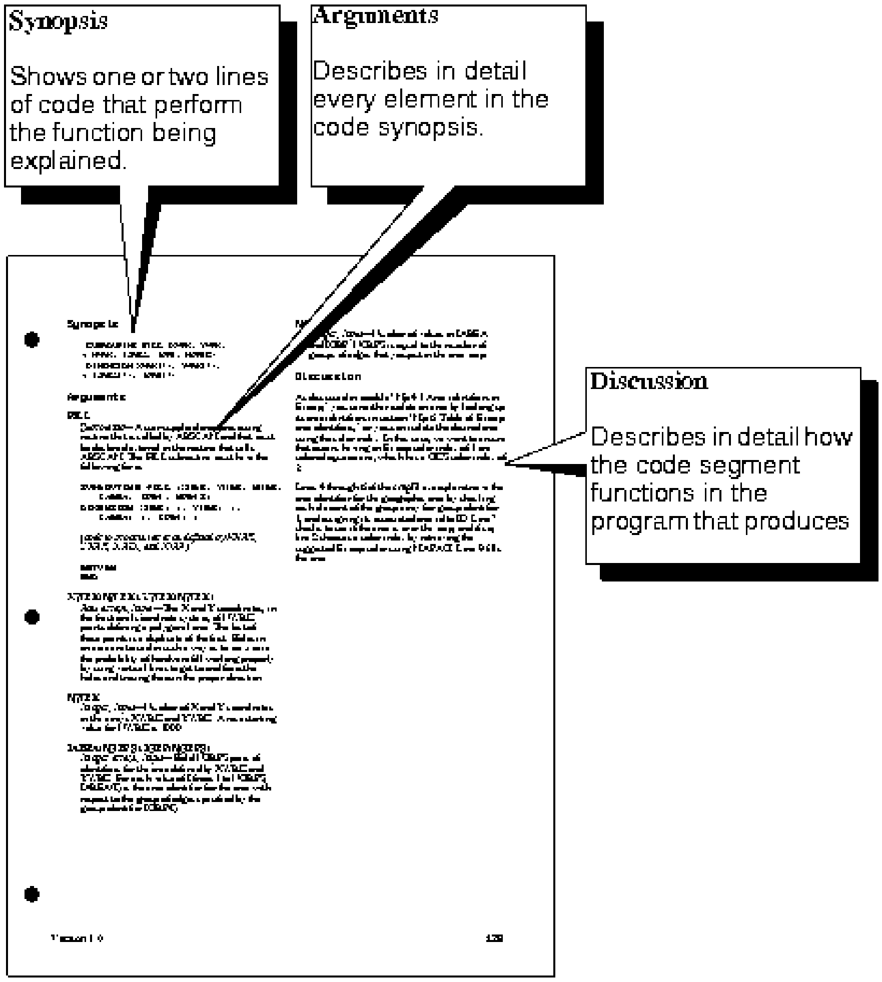
Each module that describes a specific task is structured in a consistent way. The following describes the guidelines used in creating each part of a task-oriented module. This will be useful if you have a question about the way information is presented in any of these modules.

Other resources to use with this manual

Other documentation is also available to help you use NCAR Graphics: the *NCAR Graphics Contouring and Mapping Tutorial*, a full set of **man** pages, and Programmer Documents from the package developers. Examples are also provided to show you NCAR Graphics capabilities and to help you understand and use the code that produces the examples.

Understanding the layout of tutorial modules


Module title
Names the concept being explained by this module.
Module description
Gives a brief overview of the concept being explained.
Graphic example
Shows the concept being explained.
Code segment
Shows the relevant segment of code that produces the graphic example shown.
Filling areas: Writing a fill routine
Code segment from cpfill.f



Details about how tutorial information is presented

Module title and description

The title of the module defines the concept or task that is discussed in it. The text below the title briefly introduces the concept or task.

Graphic example

The graphic illustrates the task discussed in the module. The graphic is produced by the example program named in the code segment that follows.

All color graphics in this guide are printed in gray scale to minimize printing costs. Please run the example, then view the color graphic on your screen for a better understanding of the text.

Code segment

This excerpt from the Fortran code that generates the graphic example shows how the task being explained by the module is accomplished. The numbers at the beginning of each line are for reference in the text only. These numbers have nothing to do with the line numbers in the actual example.

The heading "Code segment from *example*.f" gives you the name of the example file. To obtain files that contain the Fortran code, compile and execute the code, and create the NCGM, type:

ncargex *example*

Do not type the ".f" extension; "*example*.f" denotes the file that contains the Fortran code. All files generated by **ncargex** are placed in your current directory.

All Fortran code is printed in this manual with the Courier (typewriter) font. This monospaced type font preserves the correct relationships of Fortran columns.

Synopsis

This line of code demonstrates the calling sequence of the routines or parameters discussed in the module.

Arguments

This part of the module is a detailed description of every element in the synopsis. This part typically describes each argument to or parameter for the Fortran routine(s) in the synopsis.

Two important pieces of information follow the argument name: its Fortran type and its function in the program. When *parameters* are listed under the Arguments heading, only their Fortran type is given.

Fortran types include INTEGER, REAL, and CHARACTER, plus arrays of these types (the dimension of any array is always indicated). The argument's function in the program may be as *input* (the user supplies a value for the argument), *output* (the routine returns information to the user or the program in the argument), *workspace* (a scratch array for the routines to use in manipulating information, but that the user must declare), or *subroutine* (a special-purpose routine that the user typically creates or modifies to produce a desired result).

Wherever it is appropriate, specific values and ranges of values are explained in text that is indented from the text describing arguments and parameters.

Characters used as variables to represent a number (*n* or *m* for example) are printed in italics.

Discussion

This part of a module provides a line-by-line analysis of how the code segment functions in the program. Discussion may also include other useful information, such as frequently seen error messages or common problems.

All references to example names, filenames, and commands are printed with bold characters.

Chapter 6: What you need to know about GKS workstations

NCAR Graphics is based on GKS, the Graphical Kernel System. GKS is an ANSI standard that standardizes graphics functionality at a fairly low level. GKS is based on the concept of abstract graphical workstations. These workstations provide the logical interface through which an application program controls physical devices.

All GKS workstations are conceptualized as having a single display surface of fixed resolution allowing only rectangular display spaces. These workstations support drawing lines, plotting text, filling polygons, and so on. GKS workstations fall into several types depending on whether they support graphical output or graphical input, or store graphical instructions.

This section presents those parts of GKS that you should know in order to use workstations within the context of NCAR Graphics. This section discusses what types of physical workstations are available in the NCAR GKS package and shows how to associate an abstract GKS workstation with a physical workstation. This section also covers how to direct output to specific workstations and how to store graphics instructions, either in a metafile for permanent storage, or in a segment for temporary storage and subsequent copying to other workstations.

The basic reference for this section is the *User's guide for NCAR GKS-0A Graphics* programmer document; check with your site installer for the online PostScript version. Appendix C of that document also gives several references where complete presentations can be found for the entire GKS standard.

Because of the nature of this section, several of the modules do not contain accompanying graphics. Also, some of the modules have no sample codes since many of the functions are best illustrated in the context of a complete program. To this end, a program illustrating most of the functions covered in this section is presented as the final module.

Table of GKS workstation-related user entry points

This section discusses functions related to manipulating GKS workstations. These functions come from the GKS, Gflash, and SPPS utilities.

GKS

GACWK
Activates a GKS workstation.

GCLKS
Closes the GKS package.

GCLRWK

Clears a GKS workstation.

GCLWK
Closes a GKS workstation.

GDAWK
Deactivates a GKS workstation.

GESC
GKS ESCAPE function used for changing the name of an output metafile.

GOPKS
Opens the GKS package.

GOPWK
Opens a GKS workstation.

GUWK
Updates a GKS workstation

Gflash

GFLAS1
Initiates the creation of a segment for storing graphics instructions.

GFLAS2
Terminates a segment of stored graphics instructions.

GFLAS3
Copies segments of stored graphics instructions to active GKS workstations.

GFLAS4
Processes a previously created segment of stored graphics instructions for use with GFLAS3.

SPPS

FRAME
Updates all open workstations, creates a picture break for NCGM workstations, and pauses in the X11 window of most recent creation if such a window exists.

NGPICT
Combines several workstation control functions.

PLOTIF
Flushes the SPPS polyline buffer.

SFLUSH

Updates all open workstations and flushes all possible buffers.

Opening GKS, opening and activating workstations

This module describes the workstation types that are available to NCAR Graphics and shows how to put them in the proper states to receive graphics instructions.

Before any GKS functions can be called, or before any NCAR Graphics calls can be made, GKS must be opened. Opening GKS initializes all default settings such as character height and marker type. Most GKS functions that are not workstation specific can be called after GKS is open; this includes things like setting line color and thickness, setting fill area interior style and color, setting marker type and size, and so on.

GKS functions that are workstation specific, such as defining color indices, require that the workstation involved be open. In addition, if the GKS output primitives for drawing lines, text, filled areas, markers, and cell arrays are to be sent to a workstation, that workstation must be active.

GOPKS opens the GKS package. GOPWK opens a workstation and is the GKS function that establishes the association between the GKS logical workstation and a physical workstation. GACWK activates a workstation.

Code segment 1 from fgke01.f

```
      CALL GOPKS(6,0)
      CALL GOPWK(1,2,1)
      CALL GOPWK(2,0,8)
      CALL GACWK(1)
      CALL GACWK(2)
```

Synopsis

```
CALL GOPKS (ERRFIL, BUFA)
CALL GOPWK (WKID, CONID, WKTYPE)
CALL GACWK (WKID)
```

Arguments

ERRFIL

Integer, Input ---The Fortran logical unit number to which error messages are to be written. Typically this would be unit 6 (standard out).

BUFA

Integer, Input---The dimension of an internal buffer to be used by GKS. Currently in NCAR GKS, this is ignored and can be set to zero.

WKID

Integer, Input---A number assigned to a workstation as an identifier that can be used in all subsequent calls to GKS functions that require a workstation identifier. In NCAR GKS, WKID can be any non-negative integer.

CONID

Integer, Input---A connection identifier that has different meanings for different workstation types (see the description of WKTYPE and the "Discussion" section below).

WKTYPE

Integer, Input---An identifier specifying the type of output device targeted. Each GKS package has an implementation-dependent set of workstation types that the package supports. There are five legal workstation types in NCAR GKS:

1

NCGM. The output device will be a metafile whose default name is "gmeta". See the module later in this chapter, "Changing an NCGM filename" to change metafile output names. Only one NCGM output file can be open at a time.

3

WISS. This stands for "Workstation Independent Segment Storage" and should be opened before storing graphics in segments. See the module "Using GKS segments, the Gflash utility" later in this section. Only one WISS workstation can be open at a time.

7

Pre-existing X11 window. Directs graphics output to the pre-existing X11 window whose window ID is given as the connection ID.

8

Non-existing X11 window. Opens up a new X11 window.

10

Text dump of graphics output. Writes a human-readable ASCII dump of the output.

Discussion

For workstation types 1 (NCGM) or 3 (WISS), the connection ID is used as a Fortran logical unit number. It is permissible to have workstations of types 1 and 3 open simultaneously, but they must have distinct connection IDs. For workstations of type 7, the connection ID is the X11 window ID for an existing window (an ID as obtained from **xwininfo** for example). For workstations of type 8, an X11 window will be created at open workstation time. The connection ID is irrelevant for workstations of type 8. For workstations of type 10, the ASCII text is written to standard output and the connection ID is irrelevant. There can be a maximum of fifteen simultaneously open workstations of all types.

Line 1 of code segment 1 from **fgke01.f** opens the GKS package and establishes standard output as the error unit. Many GKS calls can be made after GKS has been opened and before any workstation has been opened. For example, you can make calls to set the text alignment, change the character height, and change the marker type. Line 2 opens an NCGM workstation and specifies Fortran logical unit 2 in the connection ID. After the NCGM workstation has been opened, it is legal to make calls to the GKS function GSCR to specify a color table for that workstation. Line 3 opens a new X11 window. After it has been opened, any calls to produce GKS primitives (such as polylines) will not produce any output on workstations 1 or 2, since they are not active. Line 4 activates the NCGM workstation. Calls to produce GKS primitives after this point will be sent to workstation 1, but not to workstation 2, since it is not yet active. Line 5 activates the X11 workstation. All GKS output primitives will now be sent to both workstations.

Deactivating and closing workstations, closing GKS

This module describes how to reverse the operations in the previous module. It describes how to deactivate and close a workstation and how to close GKS.

GDAWK deactivates a workstation, GCLWK closes a workstation, and GCLKS closes GKS.

Code segment 2 from fgke01.f

```
1       CALL GDAWK(1)
2       CALL GDAWK(2)
3       CALL GCLWK(1)
4       CALL GCLWK(2)
5       CALL GCLKS
```

Synopsis

```
CALL GDAWK (WKID)
CALL GCLWK (WKID)
CALL GCLKS
```

Arguments

WKID

Integer, Input---A number identifying the workstation to be deactivated or closed. WKID must be the same number that was used in some previous GOPWK call.

Discussion

Line 1 of the **fgke01.f** code segment calls GDAWK to deactivate the NCGM workstation. After this call, the state of this workstation is exactly as it was immediately after the call to GOPWK that opened it. Unless the NCGM workstation is reactivated, no further GKS primitives will be sent to it. If the NCGM workstation is reactivated before being closed, then GKS output primitives will be once again be sent to it and appended to the file.

Line 2 deactivates the X11 window. The window will stay visible, but subsequent output primitives will not be sent to it unless it is reactivated.

Line 3 closes the NCGM file. The metafile is terminated with the appropriate NCGM elements and no new elements can be appended to it. If another NCGM file were now to be opened with the default name, it would overwrite the previously generated file. If an NCGM were opened with a new name (see "Changing an NCGM filename" later in this section) an entirely new file would be originated and the previous file would remain unchanged.

Line 4 closes the X11 window. The window will be erased from the screen, and that particular window cannot be reopened. Closing workstations removes the workstation identifiers from the set of open workstations and frees the connection identifiers. These numbers can be reused in future calls if desired.

Line 5 closes GKS. This call ensures that the graphics package is properly terminated. No graphics calls (with the exception of GOPKS) are legal while GKS is closed.

Making sure things are current

There are several buffers in the output viewing pipeline of NCAR Graphics. There is the SPPS polyline buffer; there is an output buffer in the metafile generation software; and there are system level I/O buffers. If any of these buffers is only partially filled, it may be that an output workstation does not reflect all of the requests that have been made. There are several functions that can be called to insure that workstations reflect all requests that have been made. This module discuss three such functions. Most of the functionality discussed here is subsumed under the functions discussed in the next module, but in certain circumstances that will be pointed out, you may want to call these lower-level functions directly. GUWK updates a workstation. PLOTIF draws lines and flushes the SPPS polyline buffer. SFLUSH flushes the SPPS polyline buffer and updates all workstations.

Synopsis

```
CALL GUWK (WKID, REGFL)
CALL PLOTIF (X, Y, IPEN)
CALL SFLUSH
```

Arguments

WKID
: *Integer, Input*---A number identifying the workstation to be updated. WKID must be the same as that used in some previous call to GOPWK.

REGFL
: *Integer, Input*---A flag to specify if the current picture should be regenerated. The possible values are "0" for postpone and "1" for perform. For the workstation types supported in NCAR GKS, this flag should always be set to "0".

X,Y
: *Real, Input*---A coordinate pair in normalized device coordinates. For the purposes of this discussion (using PLOTIF to flush the SPPS pen-move buffer) these values are not relevant and can be set to "0.".

IPEN
: *Integer, Input*---A parameter that, for the purposes of this discussion (using PLOTIF to flush the SPPS pen-move buffer), should be set to "2".

Discussion

The function GUWK (update workstation) is a GKS entry point. You would call it if you were interested in writing an application that depended exclusively on GKS. For a metafile, GUWK forces a write of the most current requests and flushes the system I/O buffers. For an X11 workstation, GUWK forces the current picture to reflect all requests that have been made. GUWK does not flush the SPPS polyline buffer.

PLOTIF is an SPPS entry point and is used primarily for drawing lines, but it has the additional use of flushing the SPPS polyline buffer if it is called with the arguments "0., 0., 2". (See the **man** page for PLOTIF for complete details on that function). The polyline buffer in SPPS is designed to reduce metafile sizes by buffering pen-moves in order to call the GKS polyline function with as many points as possible. Without the buffering, GPL would need to be called with only two points in all cases.

The SPSS polyline buffer can produce some unwanted side effects. For example, if you change the line thickness before flushing the SPPS polyline buffer, then the line thickness request can actually be issued before the polyline call that is being buffered, so that the setting of line thickness can appear to have a retroactive effect. Whenever line attributes are being changed, you should flush the SPPS polyline buffer first. Calling PLOTIF is the most direct way of doing this, although other functions will flush this buffer as well, such as SFLUSH, discussed below, and the functions discussed in the next module. If you are using NCAR Graphics calls and want to update a workstation, it is advisable to flush the SPPS polyline buffer first. Thus, calling PLOTIF to flush the SPPS polyline buffer followed by calling GUWK is a logical sequence of calls.

SFLUSH is an SPPS entry. SFLUSH flushes the SPPS polyline buffer and then calls GUWK on all open workstations. The intent of SFLUSH is to flush all possible buffers relating to all open workstations.

There are certain circumstances where it would be more efficient to call PLOTIF rather than SFLUSH. For example, if you were creating dozens of objects with line draws and changing line attributes between the creation of each object, then it would be most efficient to call PLOTIF to flush the SPPS polyline buffer after drawing each object, rather than calling SFLUSH, since SFLUSH would add the overhead of updating all open workstations.

If you don’t want to worry about the subtleties discussed in this module, but you want to ensure that your workstations are current, call SFLUSH and go on your merry way.

Controlling pictures

This module discusses various ways of updating and terminating pictures. For metafiles, this module shows how to insert an "END PICTURE" element into the metafile; for X11 windows, it shows how to pause in a window, how to clear the window, and how to issue a prompt waiting for keyboard or mouse input. GCLRWK clears a workstation.

Synopsis

```
CALL GCLRWK (WKID, COFL)
CALL FRAME
CALL NGPICT (WKID, ACTION)
```

Arguments

WKID

Integer, Input---A number specifying a GKS logical workstation. This number must be the same as that used in some previous GOPWK call.

COFL

Integer, Input---Flag for clearing the display surface. COFL can have two possible values:

0
: Checks to see if the display surface is empty. If it is, nothing is done; if not, the display surface is cleared.

1
: Issues a clear display surface command whether the surface is empty or not.

ACTION
Integer, Input---Specifies an action to be taken on WKID. Legal values are:

0
: Executes an update workstation on WKID (see GUWK in the previous module).

1
: Executes an update workstation on WKID followed by a clear workstation.

2
: Executes an update workstation on WKID followed by a pause waiting for any keyboard or mouse click. The picture is not cleared after the click has been issued.

3
: Same as 2 above, except the picture is cleared after the click has been issued.

4
: Same as 3 above, except a "<READY>" prompt is issued in the lower left corner of the window, after the update of the workstation, to indicate a mouse click is being awaited.

Discussion

GCLRWK is a GKS routine. For workstations of type 1 (NCGM), GCLRWK is interpreted as a picture termination in NCGM generation. If COFL = 0 and no output primitives have been written in the current picture, then a call to GCLRWK is a "do nothing"; otherwise, it generates an END PICTURE element (as well as other picture initializing elements). If COFL = 1, then a call to GCLRWK generates an END PICTURE. When **ctrans** encounters an END PICTURE in NCGM interpretation, it pauses waiting for a mouse click.

For workstations of types 7 or 8 (see the module "Opening GKS, opening and activating workstations" earlier in this section), clear workstation erases all primitives that appear on the screen. For all other workstation types, clear workstation updates the workstation. Clear workstation does not flush the SPPS polyline buffer.

If an NCGM workstation is active, a call to FRAME causes an END PICTURE CGM element to be generated. If there are active X windows, FRAME updates all workstations and pauses in the window of most recent creation. After a mouse click or a key click in the window on pause, *all* workstations are cleared and execution continues. FRAME flushes the SPPS polyline buffer.

FRAME is most applicable to an environment with a single active workstation. When multiple workstations are used, the more versatile NGPICT may be more useful.

In a call to NGPICT, if WKID designates a metafile, then a clear workstation is done. This inserts an END PICTURE into the metafile. The result produced in this case would be the same as that resulting from a call to FRAME. The only valid values for ACTION in a call to NGPICT that apply to metafile workstations are 0 and 1.

NGPICT is designed to be used when more precise control over individual workstations is desired than that offered by FRAME. This might be desirable, for example, when an application is controlling several simultaneously active workstations. NGPICT flushes the SPPS polyline buffer. To use NGPICT open GKS workstations with GOPWK rather than with OPNGKS.

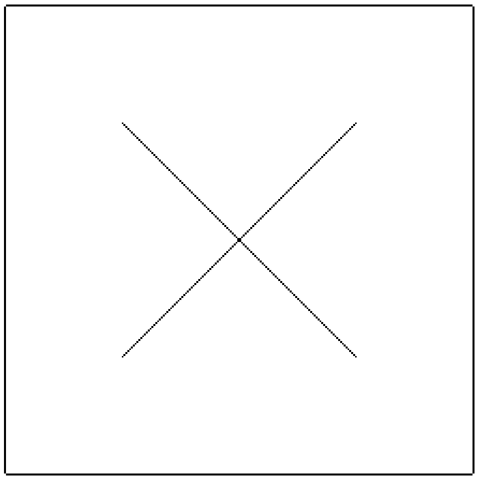
For examples of usage of the functions described in this module, see the final code example at the end of this section.

Using GKS segments, the Gflash utility

This module describes how to store graphics instructions in independent buffers for later plotting. The buffers may be plotted repeatedly in as many pictures as desired, thus saving the time required to regenerate the instructions. This is accomplished by using the functions in the Gflash utility. Gflash offers a limited picture segmentation capability. A full segmentation capability is a part of all GKS packages at level 2A (or higher). NCAR GKS is at level 0A.

If you are using a level 2A GKS package, then you may want to use the segmentation capabilities in that package directly instead of using Gflash. GFLAS1 starts the storage of plotting instructions into a GKS segment; GFLAS2 stops the storage of plotting instructions. GFLAS3 inserts saved plotting instructions into the output stream. GFLAS4 associates plotting instructions stored on a disk file with an identifier that can be used later with GFLAS3.

Gflash usage



Code segment from fgke02.f

```
      CALL GOPKS (6,IDUM)
      CALL GOPWK (1, 2, 1)
      CALL GACWK (1)
      CALL GOPWK(9,1,3)
```

```
      CALL GFLAS1(1)
      CALL GPL(2,RLIN1X,RLIN1Y)
      CALL GFLAS2
      CALL GFLAS1(2)
      CALL GPL(2,RLIN2X,RLIN2Y)
      CALL GFLAS2
      CALL GPL(5,PERIMX,PERIMY)
      CALL GFLAS3(1)
      CALL GFLAS3(2)
      CALL FRAME
      CALL GCLWK(9)
      CALL GDAWK (1)
      CALL GCLWK (1)
      CALL GCLKS
```

Synopsis

```
CALL GFLAS1 (IB)
CALL GFLAS2
CALL GFLAS3 (IB)
CALL GFLAS4 (IB, 'buf_name')
```

Arguments

IB

Integer, Input---A number between 0 and 99 (inclusive) used as an identifier for the file of stored instructions. GFLAS1 automatically assigns the name GNFB*xx* to the file that will receive subsequent plotting instructions. These files may be reused in subsequent job steps by using GFLAS4.

buf_name

Character, Input---A filename of a buffer previously created using a GFLAS1/GFLAS2 sequence.

Discussion

A call to GFLAS1 initiates storage of subsequent plotting instructions in a disk file. A call to GFLAS2 terminates storage of plotting instructions to the disk file originated by GFLAS1. All plotting instructions that occur between a GFLAS1 call and a GFLAS2 call are captured in the buffer identified by the argument to GFLAS1. Between the calls to GFLAS1 and GFLAS2, all output to any other workstation is suspended.

After graphics instructions have been captured in a file with a GFLAS1/GFLAS2 sequence, a call to GFLAS3 will plot the set of saved instructions on all active workstations. If you do not want the instructions of a particular flash buffer plotted to a particular workstation, you will need to deactivate that workstation before calling GFLAS3.

GFLAS4 is used to associate a file of stored instructions (created with a previous GFLAS1/GFLAS2 sequence) with an identifier for subsequent use with GFLAS3. The purpose of GFLAS4 is to allow instructions stored in a file in one job to be used in another job.

NCAR GKS stores the flash buffers on a special GKS workstation called WISS (Workstation Independent Segment Storage). You don't need to understand what WISS is, but the consequence is that

WISS needs to be opened before accessing entries in the Gflash package and closed after Gflash usage. WISS is opened with a call to GOPWK, just like any other GKS workstation. Its workstation type is 3 and the connection identifier is a Fortran logical unit number used for the disk file writes.

In the code segment from **fgke02.f**, lines 1 through 3 open GKS and open and activate an NCGM workstation. Line 4 opens WISS. Line 5 initiates the storage of subsequent instructions in a disk file with the name GNFB01; the local identifier of that file is "1". Line 6 draws a line with positive slope in buffer 1 and line 7 terminates the storage of instructions in buffer 1. Output to active workstations is suspended during the time a GFLAS1/GFLAS2 sequence is under way so that the graphics instructions go to the buffer but not to the active workstations. This represents a difference between the way GKS segments behave and the way that Gflash buffers are created.

Lines 8 through 10 result in the storage of a line of negative slope in buffer 2. Line 11 draws a perimeter around the current picture. This call is outside of any flash buffer and will be sent to the currently active NCGM workstation. Line 12 plots the instructions stored in buffer 1 on all active workstations (only one in this case). Line 13 plots the instructions stored in buffer 2 on all active workstations. Line 14 terminates the picture in the metafile. Line 15 closes the WISS workstation and lines 16 through 18 deactivate and close the NCGM workstation and close GKS.

Color table settings are not stored in Gflash buffers. The color table in effect at the time of a GFLAS3 call will be inherited by the primitives in the buffer being inserted by GFLAS3.

There can be at most 100 segments created in a single job and the legal segment names run from 0 to 99 inclusive. It is not legal to call FRAME, or to clear a workstation, while a segment is open.

GFLAS4 is available only if you are using NCAR GKS. It uses a locally-defined GKS ESCAPE function that is unique to NCAR GKS.

Changing an NCGM filename

This section shows two ways to change the name of NCGM output files from the default value of "gmeta". The first way involves setting a UNIX environment variable that allows you to change the filename for the entire job. The second way is via a call to the GKS ESCAPE function GESC. This way allows for dynamically changing the NCGM name from within your Fortran code, a method that can be used to create several different metafiles from within the same job.

Code segment from fgke03.f

```
      CHARACTER*80 FNAME,CDUM
      CALL GOPKS (6,IDUM)
      FNAME = 'META01'
      CALL GESC(-1391,1,FNAME,1,1,CDUM)
      CALL GOPWK (1, 2, 1)
      CALL GACWK (1)
      CALL GPM(1,.5,.5)
      CALL FRAME
      CALL GDAWK (1)
      CALL GCLWK (1)
      FNAME = 'META02'
      CALL GESC(-1391,1,FNAME,1,1,CDUM)
```

```
13      CALL GOPWK (1, 2, 1)
14      CALL GACWK (1)
15      CALL GPM(1,.5,.75)
16      CALL FRAME
17      CALL GDAWK (1)
18      CALL GCLWK (1)
19      CALL GCLKS
```

Synopsis

```
CALL GESC (FCTID, LIDR, IDR, MXODR, LODR, ODR)
```

Arguments

FCTID

Integer, Input---A function identifier specifying the requested activity. For changing the metafile name, this is "-1391".

LIDR

Integer, Input---Dimension of the IDR data record array.

IDR

*Character*80 array, Input*---An input data record. For calls to change a metafile name, IDR should be dimensioned as "1" and should contain the desired metafile name left justified with blank fill.

MXODR

Integer, Input---Maximum length of the ODR output data record array.

LODR

Integer, Input---Dimension of the ODR output data record array.

ODR

*Character*80 array, Input*---An output data record. For changing a metafile name, nothing is returned in ODR, so set MXODR and LODR to "1" and define ODR to be a CHARACTER*80 array dimensioned as "1".

Discussion

First, consider the example code in **fgke03.f** that shows how to create two metafiles with different names in the same job. After this, we will look at changing the NCGM name using a UNIX environment variable.

Line 1 of the **fgke03.f** code segment establishes two CHARACTER*80 variables used as the input and output data records for the GESC call. Line2 opens GKS and line 3 establishes the name for the first metafile. The specification of the metafile name must occur after GKS has been opened, but always before the NCGM workstation is opened. Line 4 informs NCAR GKS of the metafile name setting. Lines 5 and 6 open and activate an NCGM workstation. Line 7 plots a default marker in the center of the plot and line 8 terminates the picture. Lines 9 and 10 deactivate and close the NCGM file. This output

NCGM has name "META01" and cannot now be changed. Lines 11 through 18 create a second metafile with name "META02". Line 19 closes GKS.

You can have only one NCGM workstation open at a time in NCAR Graphics, so it is not possible to simultaneously create two metafiles. Also, this call to GESC works only with NCAR GKS; the GESC function identifier is unique to NCAR Graphics and will not have a meaning in another GKS package.

If you do not need to create more than one NCGM file in a single job, then specifying the NCGM output file name via the UNIX environment variable NCARG_GKS_OUTPUT is easier than making the GESC call. Just set the NCARG_GKS_OUTPUT environment variable to the name you desire. This method also has the advantage of allowing you to specify different names without having to make any code changes. Also, there are a couple of special settings for NCARG_GKS_OUTPUT that may be of use to you. One is the setting of "**|ctrans**". To make this setting in the C-shell, issue the command:

setenv NCARG_GKS_OUTPUT "|ctrans"

This has the effect of piping the output CGM to the CGM interpreter **ctrans** without creating the metafile in one job step and then invoking **ctrans** in an another job step.

Another special setting of NCARG_GKS_OUTPUT is "**|**". This pipes the metafile output to a default CGM interpreter whose name is specified in the code **wks.h.sed** that is contained in the GKS code that is distributed with the NCAR Graphics package. In the default case, "**|**" is the same as "**|ctrans**".

When using either of these special settings for NCARG_GKS_OUTPUT, you will need to make sure that your graphcap setting for **ctrans** is what you want. This can be done by setting the UNIX environment variable GRAPHCAP appropriately.

Any setting for NCARG_GKS_OUTPUT takes precedence over any settings of metafile output names via the GESC calls.

Putting it all together

This module illustrates many of the concepts in this section with an example code. The example manipulates several workstations in the course of the execution and shows how you must be careful to consider the state of each workstation independently.

Code for fgke04.f

```
1          PROGRAM TEST
2       C
3       C Illustrate workstation control functions by intermixing plotting to
4       C a metafile with plotting to X window workstations.
5       C
6       C
7       C Open GKS
8       C
9          CALL GOPKS(6,0)
10      C
11      C Open a CGM workstation and define color index 6 as red on that
12      C workstation.
```

```
C
   CALL GOPWK(1,2,1)
   CALL GSCR(1,6,1.,0.,0.)
C
C Set character height and set the text alignment to (center, half).
C
   CALL GSCHH(.04)
   CALL GSTXAL(2,3)
C
C Activate the CGM workstation.
C
   CALL GACWK(1)
C
C Create picture 1 in the metafile and call FRAME to terminate
C the first picture.
C
   CALL GSTXCI(6)
   CALL GTX(.5,.5,'Picture 1')
   CALL FRAME
C
C Put another picture in the metafile.
C
   CALL GTX(.5,.5,'Picture 2')
   CALL FRAME
C
C Open and activate an X11 workstation and define color index 6
C to be cyan on that workstation (this workstation will be referred
C to as "The first window".
C
   CALL GOPWK(5,0,8)
   CALL GACWK(5)
   CALL GSCR(5,6,0.,1.,1.)
C
C Create picture 3. This will be plotted to the CGM workstation as
C well as to the X11 window since they are both active.
C
   CALL GTX(.5,.5,'Picture 3')
C
C Terminate the metafile picture.
C
   CALL NGPICT(1,1)
C
C Pause in the X window with a "<READY>" prompt and wait for a
C mouse click. The window will be cleared after the mouse click.
C
   CALL NGPICT(5,4)
C
C Open and activate another X workstation (to be referred to as "The
C second window") and define color index 6 to be green on that
C workstation.
C
   CALL GOPWK(7,0,8)
   CALL GACWK(7)
   CALL GSCR(7,6,0.,1.,0.)
C
C Plot picture 4. This will be sent to the CGM workstation and the
C two X window workstations since they are all active.
C
   CALL GTX(.5,.5,'Picture 4')
C
C Terminate picture 4 in the metafile.
```

```
C
      CALL NGPICT(1,1)
C
C Make the second window current.
C
      CALL NGPICT(7,0)
C
C Pause in the first window with a "<READY>" prompt and wait for a
C mouse click. The window will be cleared after the click.
C
      CALL NGPICT(5,4)
C
C Clear the second window.
C
      CALL NGPICT(7,1)
C
C Deactivate the metafile (but do not close it) and draw picture 5.
C This will go to the two active X11 workstations, but not the CGM.
C
      CALL GDAWK(1)
      CALL GTX(.5,.5,'Picture 5')
C
C Make the second window current.
C
      CALL NGPICT(7,0)
C
C Pause in the first window waiting or a mouse click.
C
      CALL NGPICT(5,4)
C
C Re-activate the metafile and deactivate the second window.
C
      CALL GACWK(1)
      CALL GDAWK(7)
C
C Plot picture 6. This will go to the first window and to the
C metafile.
C
      CALL GTX(.5,.5,'Picture 6')
C
C Terminate the picture in the CGM.
C
      CALL NGPICT(1,1)
C
C Pause in the first window waiting for a mouse click.
C
      CALL NGPICT(5,4)

C Reactivate the second window and clear it.
C
      CALL GACWK(7)
      CALL NGPICT(7,1)
C
C Put out picture 7. This will go to the all active workstations.
C
      CALL GTX(.5,.5,'Picture 7')
C
C Terminate the picture in the CGM.
C
      CALL NGPICT(1,1)
```

```
134     C
135     C Make the first window current.
136     C
137        CALL NGPICT(5,0)
138     C
139     C Pause in the second window waiting for a mouse click.
140     C
141        CALL NGPICT(7,4)
142     C
143     C Deactivate and close the first window; deactivate and close the CGM.
144     C
145        CALL GDAWK(5)
146        CALL GDAWK(1)
147        CALL GCLWK(5)
148        CALL GCLWK(1)
149     C
150     C Put out picture 8. This will go to the second window, the only
151     C active workstation.
152     C
153        CALL GTX(.5,.5,'Picture 8')
154     C
155     C Pause in the second window with a "<READY>" prompt.
156     C
157        CALL NGPICT(7,4)
158     C
159     C Deactivate and close the second window.
160     C
161        CALL GDAWK(7)
162        CALL GCLWK(7)
163     C
164     C Close GKS.
165     C
166        CALL GCLKS
167     C
168        STOP
169        END
```

Discussion

Line 9 of the **fgke04.f** code segment opens GKS. Standard output is specified for the error file.

Line 14 opens an NCGM workstation. The NCGM workstation is given the identifier "1" and it uses "2" as the connection identifier.

Line 15 associates the color red with color index 6 on the NCGM workstation; this association has no bearing on how color indices are defined on other workstations. Note that the NCGM workstation does not have to be active to define color values for it.

Lines 19 and 20 establish some text attributes for drawing subsequent text strings. Notice that no workstations need to be active to make these calls; they will remain in effect for all subsequent calls, since they are never changed.

Line 29 sets color index 6 as the text color. Color index 6 may mean different things for different workstations.

Line 30 draws the text string 'Picture 1'. This will be sent to the only active workstation, which is the

NCGM workstation.

Line 31 calls FRAME to terminate the first picture in the metafile. This call to FRAME is the equivalent of making the two calls GUWK(1,0) and GCLRWK(1,1).

Lines 35 and 36 plot another picture to the NCGM file.

Line 42 opens an X11 workstation. After this call has been executed, a window should appear on your X window workstation. You will need to position this window where you want it and then click on it.

Line 43 activates the new window. This window will now receive all GKS output primitives until it is deactivated.

Line 44 associates color index 6 with the color cyan on the X11 window. Notice that this setting for color index 6 is different from the setting for the same index on the NCGM workstation.

Line 49 plots another text string. This will be sent to the two active workstations. The string will be red in the NCGM and will be cyan in the X11 window.

Line 53 terminates picture 3 in the NCGM. NGPICT will be used to control the actions on each individual workstation from now on. If we were to have called FRAME here, FRAME would first update all open workstations (this means that the NCGM and the first window would be updated) and then FRAME would pause in the window of most recent creation waiting for a mouse click. This means that FRAME would be waiting in the X11 window for mouse input. However, after the call to NGPICT in line 53, the NCGM is updated and cleared, but the X11 window is not necessarily even current.

The NCPICT call in line 58 makes the X11 window current and pauses in that window waiting for a mouse click. A "<READY>" prompt appears in the lower left part of the screen, indicating that the window is waiting for a click. If we were to have called FRAME, this prompt would not appear, although we would still be waiting for a mouse click in that window. After the mouse input is supplied, the window will be cleared, since we have called NGPICT with a second argument of "4".

Lines 64 and 65 of the **fgke04.f** code segment open and activate a second X11 window. There are now three simultaneously active workstations, the NCGM workstation and the two X11 windows. The second X11 window has identifier number 7.

Line 67 associates the color green with color index 6 on the second X11 window. This association applies only to this workstation and is different from the setting of this index for the other two currently active workstations.

Line 71 plots another text string. This will be sent to all three currently active workstations.

Line 75 terminates the picture in the NCGM. This call updates the workstation and clears it. (Remember that a clear workstation on a NCGM terminates the current picture.)

Line 79 brings the second window up to date.

Line 84 makes the first window current and pauses in the window with a "<READY>" prompt, indicating input is being awaited. It was important to bring the second window up to date before executing the pause in the first window, since otherwise that window might not reflect all graphics requests made to it. In this case, it is possible that the text string ’Picture 4’ would not be in the second window unless it had been updated. If we had called FRAME in line 79, all workstations would have been updated and we would be paused in the second window (the window of most recent creation) instead of being paused in the first window as we are. Also, there would be no prompt in the window indicating it is waiting for input. It is for these reasons that using NGPICT can be helpful, although using FRAME can certainly make coding easier if you can keep in mind exactly what it is doing.

Line 88 clears the second window. Without this, the next output primitives would overwrite what is currently there.

Line 93 deactivates the NCGM workstation. This means that output primitives will not be sent to it until it is reactivated. At this point, we have two active workstations, the two X windows.

Line 94 plots another text string. This will appear in the two active X windows.

Line 98 brings the second window up to date and line 102 brings the first window up to date and causes a pause with a prompt.

Line 106 reactivates the NCGM workstation. Line 107 deactivates the second X window. There are now two active workstations: the first X window and the NCGM. Deactivating the second X window means that it will not receive output primitives until reactivated. The window still stays on the screen in the state it was in at the time it was deactivated.

Line 112 plots another text string. It is sent to the two active workstations, but not to the inactive first window, which continues to display what was on it.

Line 116 terminates the picture in the NCGM and line 120 causes a pause in the currently active X window (the first window).

Line 124 of the **fgke04.f** code segment reactivates the second X11 workstation; line 125 clears it. All three workstations are now active again.

Line 129 plots another text string. This will be sent to all workstations.

Line 133 terminates the picture in the NCGM.

Line 137 brings the first window up to date.

Line 141 causes a pause in the second window with a "<READY>" prompt, indicating that window is waiting for input. Notice that we have now switched the window wanting input. Previously, we were using the first window for pausing. Now it is necessary to click in the second window to continue execution and only the second window will be automatically cleared after the pause.

Lines 145 through 148 deactivate and close the NCGM workstation and the first X window. The closing of the NCGM makes that file complete. It cannot be changed after the close. The close of the first X window causes it to disappear from the screen. No further commands can be issued to that window.

Line 153 plots one final text string. It will be sent to the only currently active window.

Line 157 causes a pause in the currently active window with a "<READY>" prompt.

Lines 161 and 162 deactivate and close the final workstation. It disappears from the screen.

Line 166 closes GKS.

After the execution of this job, the NCGM will contain six pictures, the ones numbered 1, 2, 3, 4, 6, and 7. All pictures in the CGM will be red.

During the execution of this job the first X11 window will have plotted five pictures, the ones numbered 3, 4, 5, 6, and 7. All pictures in the first X11 window will be cyan.

During the execution of this job, the second X11 window will have plotted four pictures, the ones numbered 4, 5, 7, and 8. All pictures in the second X11 window will be green.

Chapter 7: Color tables and color mapping systems

A color table is a table that associates color values, such as cyan or yellow, with nonnegative integer indices. This chapter describes how to define color tables in NCAR Graphics and how the associated color indices are used to assign colors to graphics primitives like polylines, polymarkers, text, and filled areas.

In this chapter you will also see how to convert from one "color space" to another.

A color space, or color model, is a three-dimensional coordinate system where coordinates in that system represent colors. There are many popular color spaces; some of them are more suitable for hardware and others are more suitable for human conceptualizing. Two of the more common color models used by hardware are the RGB (red, green, blue) model used by most workstations and the YIQ model used by broadcast TV. Two of the more popular models for human conceptualizing are the HLS (hue, lightness, saturation) and HSV (hue, saturation, value) models.

This chapter describes utilities for converting between the RGB, HLS, HSV, and YIQ models. For a complete description of these models consult the book *Computer Graphics* by James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes, (Addison-Wesley, 1990).

To provide sample graphics plots for the functions in this chapter would require the use of color; for this reason such plots have not been provided.

This chapter uses the notation [x,y), where x and y are real numbers, to mean the set of real numbers between x and y, including x, but not including y.

Table of color table and color mapping user entry points

This chapter lists entry points for setting up color tables and for converting between several systems for representing color values. The table also lists entry points for assigning colors to the GKS primitives of polyline, polymarker, text and filled area.

Color tables

GSCR
: Sets color representation. Used to set up color tables.

Assigning colors to GKS primitives

GSPLCI
: Sets polyline color index. Specifies a color index to be used for GKS polylines.

GSPMCI
: Sets polymarker color index. Specifies a color index to be used for GKS polymarkers.

GSTXCI
: Sets text color index. Specifies a color index to be used for GKS text.

GSFACI
: Sets fill area color index. Specifies a color index to be used for GKS filled areas.

Converting between color systems

HLSRGB
: Converts from the hue, lightness, saturation color model to the red, green, blue color model.

RGBHLS
: Converts from the red, green, blue color model to the hue, lightness, saturation model.

HSVRGB
: Converts from the hue, saturation, value color model to the red, green, blue model.

RGBHSV
: Converts from the red, green, blue color model to the hue, saturation, value model.

YIQRGB
: Converts from the YIQ color model to the red, green, blue model.

RGBYIQ
: Converts from the red, green, blue color model to the YIQ model.

Color tables

Color in NCAR Graphics uses an indexing scheme. A table is set up to associate actual color values with indices, and the indices are used to tell NCAR Graphics what color value to use. The GKS function "Set Color Representation," GSCR, is used to set up the color tables. There are various ways to specify color information in specific situations, but all ways ultimately will use color indices as set up in a color table. In some situations, a given function will inherit its color from the underlying GKS specifications. For example, when drawing lines with LINE, the lines will be drawn using the current GKS line color. This module shows how to specify colors for the basic GKS primitives of polylines (GSPLCI), filled areas (GSFACI), text (GSTXCI), and polymarkers (GSPMCI). For specifying colors in other situations, consult the documentation on color in the appropriate chapter.

Code segment from fcce01.f

```
      CALL GSCR(IWK, 0, 0.0, 0.0, 0.0)
      CALL GSCR(IWK, 1, 1.0, 1.0, 1.0)
      CALL GSCR(IWK, 2, 1.0, 0.0, 0.0)
      CALL GSCR(IWK, 3, 0.0, 1.0, 0.0)
      CALL GSCR(IWK, 4, 1.0, 1.0, 0.0)
      CALL GSCR(IWK, 5, 0.0, 1.0, 1.0)
      CALL GSPLCI(3)
      CALL GPL(5,X1,Y)
      CALL GSMKSC(4.)
      CALL GPM(1,.5,.25)
      CALL GSTXCI(4)
      CALL GTX(0.5,0.5,'Text')
      CALL GSFACI(5)
      CALL GSFAIS(1)
      CALL GFA(5,X2,Y)
      CALL GSPMCI(2)
      CALL GPM(1,.5,.75)
```

Synopsis

```
CALL GSCR (WKID, CI, CR, CG, CB)
CALL GSPLCI (CI)
CALL GSTXCI (CI)
CALL GSFACI (CI)
CALL GSPMCI (CI)
```

Arguments

WKID

Integer, Input ---A number assigned to a workstation as an identifier that is to be used in subsequent calls to GKS functions that require a workstation identifier. In NCAR GKS, WKID can be any nonnegative integer.

CI

Integer, Input ---A color index. CI can be any nonnegative integer and can be used in any subsequent calls that require a color index.

CR

Real, Input ---A number between 0. and 1. (inclusive) that gives an intensity for the red component of the color value that is to be associated with the color index CI.

CG

Real, Input ---A number between 0. and 1. (inclusive) that gives an intensity for the green component of the color value that is to be associated with the color index CI.

CB

Real, Input ---A number between 0. and 1. (inclusive) that gives an intensity for the blue component of the color value that is to be associated with the color index CI.

H

Real, Input for HLSRGB; Output for RGBHLS---A value for hue in the HLS color space. H is in the range [0.,360.).

L

Real, Input for HLSRGB; Output for RGBHLS---A value for lightness in the HLS color space. L is in the range [0.,100.].

S

Real, Input for HLSRGB; Output for RGBHLS---A value for saturation in the HLS color space. S is in the range [0.,100.].

R

Real, Input for RGBHLS; Output for HLSRGB---A value for red in the RGB color space. R is in the range [0.,1.].

G

Real, Input for RGBHLS; Output for HLSRGB---A value for green in the RGB color space. G is in the range [0.,1.].

B

Real, Input for RGBHLS; Output for HLSRGB---A value for blue in the RGB color space. B is in the range [0.,1.].

H

Real, Input for HSVRGB; Output for RGBHSV---A value for hue in the HSV color space. H is in the range [0.,360.).

S

Real, Input for HSVRGB; Output for RGBHSV---A value for lightness in the HSV color space. S is in the range [0.,1.].

V

Real, Input for HSVRGB; Output for RGBHSV---A value for the value component in the HSV color space. V is in the range [0.,1.].

R

Real, Input for RGBHSV; Output for HSVRGB ---A value for red in the RGB color space. R is in the range [0.,1.].

G

Real, Input for RGBHSV; Output for HSVRGB ---A value for green in the RGB color space. G is in the range [0.,1.].

B

Real, Input for RGBHSV; Output for HSVRGB ---A value for blue in the RGB color space. B is in the range [0.,1.].

Y

Real, Input for YIQRGB; Output for RGBYIQ---A value for the Y component of the YIQ color space. Y is in the range [0.,1.].

I

Real, Input for YIQRGB; Output for RGBYIQ---A value for the I component of the YIQ color space. I is in the range [-.6,.6].

Q

Real, Input for YIQRGB; Output for RGBYIQ---A value for the Q component of the YIQ color space. Q is in the range [-.52,.52].

R

Real, Input for RGBYIQ; Output for YIQRGB ---A value for red in the RGB color space. R is in the range [0.,1.].

G

Real, Input for RGBYIQ; Output for YIQRGB ---A value for green in the RGB color space. G is in the range [0.,1.].

B

Real, Input for RGBYIQ; Output for YIQRGB ---A value for blue in the RGB color space. B is in the range [0.,1.].

Discussion

Lines 1 through 3 of code segment 3 from **fcce02.f** assign values to the Y, I, and Q components of the YIQ color space that correspond to red=0., green=1., and blue=0. in the RGB space. The result of the call in line 4 is that R will be returned as 0., G as 1., and B as 0.

Lines 5 through 7 of code segment 3 from **fcce02.f** assign values to red, green, and blue in the RGB space that correspond to values of Y=1., I=0., and Q=0. in the YIQ space. The result of the call in line 8 is that Y will be returned as 1., RI will be returned as 0., and Q will be returned as 0. Y is the grayscale component of a color TV signal that is shown on black and white TVs.

Chapter 8: Coordinate systems in NCAR Graphics

NCAR Graphics depends on three different coordinate systems and a transformation between them to plot your data to the screen. Before any plotting can be done, this transformation must be defined, so it is a critical part of your graphics program. Each of the coordinate systems and the transformation are defined below.

Illustration of a normalization transformation

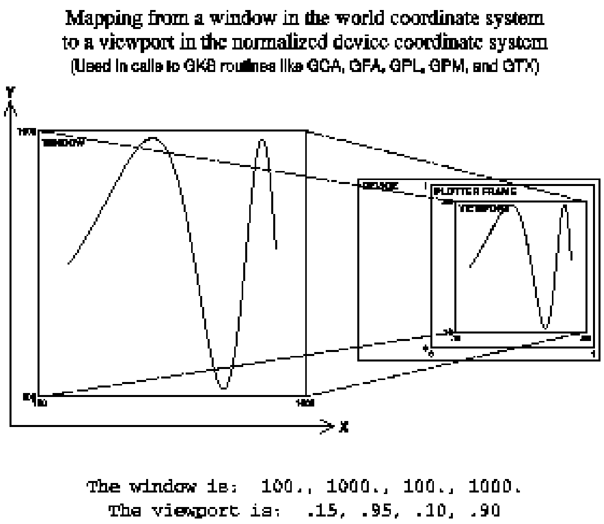


Table of coordinate system user entry points

This chapter discusses GKS and SPPS routines for setting up coordinate systems and normalization transformations.

GKS

GSVP
: Set viewport coordinates.

GSWN
: Set window coordinates.

GSELNT
: Select normalization transformation.

SPPS

SET

Set viewport and user coordinates and logarithmic scaling.

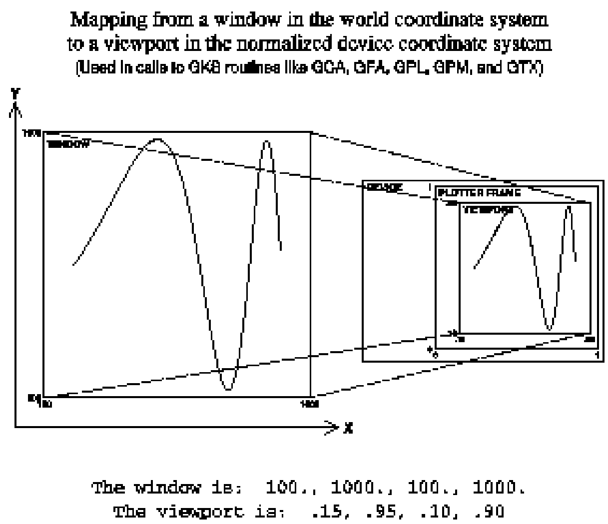
GETSET

Retrieve viewport user coordinate and logarithmic scaling values.

Definitions

This module introduces several important definitions which are used heavily in this guide.

Illustration of a normalization transformation



Discussion

NCAR Graphics uses two GKS coordinate systems: World Coordinates (WCs) and Normalized Device Coordinates (NDCs), as well as its own coordinates, user coordinates. All are expressed in real numbers.

The **World Coordinate System** (WCS) is the system in which you perform your calculations. It can have units of feet, meters, millibars, or whatever engineering units apply to the calculations. World coordinates are Cartesian coordinates expressed as real numbers. You can select a rectangular subspace of world coordinates for output; this space is called a "window." In world coordinates, both axes must monotonically increase starting with a data origin in the lower left coordinate of the frame.

Normalized Device Coordinates (NDCs) is a device-independent coordinate system used by GKS as an addressing system for a virtual (any) plotting device. In GKS, you must perform a transformation from WCs to NDCs. Whereas WCs have variable ranges, NDCs have a constrained range limited to (0.,0.) in the lower left corner and (1.,1.) in the upper right corner. A rectangular subspace of NDCs can be selected for output; this space is called a "viewport."

The **user coordinate system** in NCAR Graphics adds two important extensions to GKS world coordinates: "mirror-imaging" of an axis, and "nonlinear scaling" of an axis. Thus, in user coordinates, windows can be defined in which the horizontal coordinate of the window might be meters varying from 1000. at the left to 100. at the right (mirror-imaging), and the vertical coordinate might vary from 10*2* at the bottom to 10*3* at the top in logarithmic scaling.

If you are working in Cartesian space with linear scaling and standard minimum to maximum axis scaling, then world coordinates and user coordinates are equivalent.

The process of transforming data from WCs into NDCs for output on the plotting device is called a **normalization transformation**. The NCAR Graphics GKS-0A package currently supports two normalization transformation types. Type 0 defines an identity transformation in which the WCs, like NDCs, have horizontal and vertical axes ranging from 0. to 1. This is the transformation used to facilitate putting an object such as a title at a specific location on the frame. A type 1 normalization transformation allows the WCs to have whatever range of values is desired.

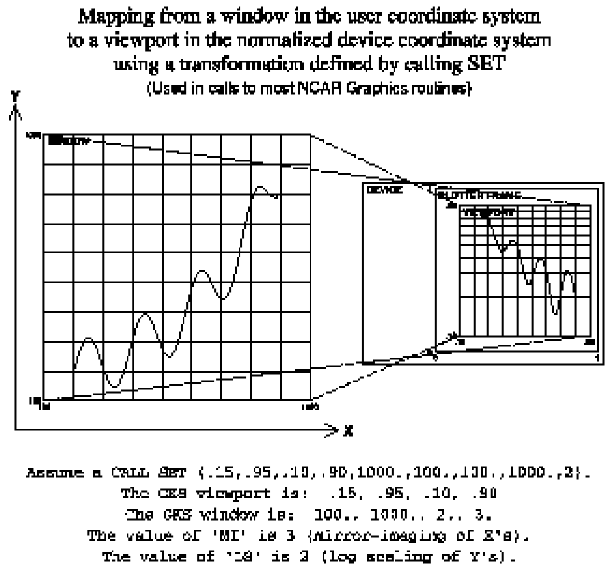
See "Appendix A: The use of X/Y coordinates in NCAR Graphics" for a complete description of NCAR Graphics coordinate systems.

Normalization transformations

To define a normalization transformation within the GKS system requires two calls, one to the GKS routine GSVP to define its "viewport" and a second call to the GKS routine GSWN to define its "window." Using the SPPS routine SET, you can set the window, the viewport, mirror-imaging of axes, and nonlinear scaling of axes all in a single call. Since this routine is more flexible than the two GKS routines, we recommend using SET.

The graphic shows a normalization transformation from world coordinates to NDCs, in which there is mirror-imaging and nonlinear scaling of axes.

Normalization transformation showing mirror-imaging and nonlinear scaling of axes



Code segment from fcoord2.f

```
CALL SET(VPL,VPR,VPB,VPT,1000.,100.,100.,1000.,2)
```

Synopsis

```
      CALL SET (VPL, VPR, VPB, VPT, WDL, WDR, WDB, WDT, LS)
```

Arguments

VPL, VPR, VPB, and VPT

Real, Input---Normalized Device Coordinates (NDCs) that define a viewport. The letters "L," "R," "B," and "T" refer to the left, right, bottom, and top edges of the viewport, respectively. VPL must be less than VPR, and VPB must be less than VPT. All coordinates have values in the range of 0. to 1.

WDL, WDR, WDB, and WDT

Real, Input---User coordinates that define a window. The letters "L," "R," "B," and "T" refer to the left, right, bottom, and top edges of the window, respectively. Axis reversal (also called mirror-imaging of an axis) is controlled by the order of these values. If WDL is greater than WDR, the X coordinates are mirror-imaged. Similarly, if WDB is greater than WDT, the Y coordinates are mirror-imaged.

LS

Integer, Input---Specifies the desired combination of linear or logarithmic axis scaling. Possibilities are:

1 Linear X, Linear Y
2 Linear X, Log Y
3 Log X, Linear Y
4 Log X, Log Y

Note that the window defining coordinates are always input to SET as they would be before any logarithmic transformation.

Discussion

A call to the SPPS routine SET provides a way to completely define the mapping from window user coordinates to viewport Normalized Device Coordinates (NDCs), including the specification of axis scaling reversal and whether axes are to have linear or logarithmic scaling.

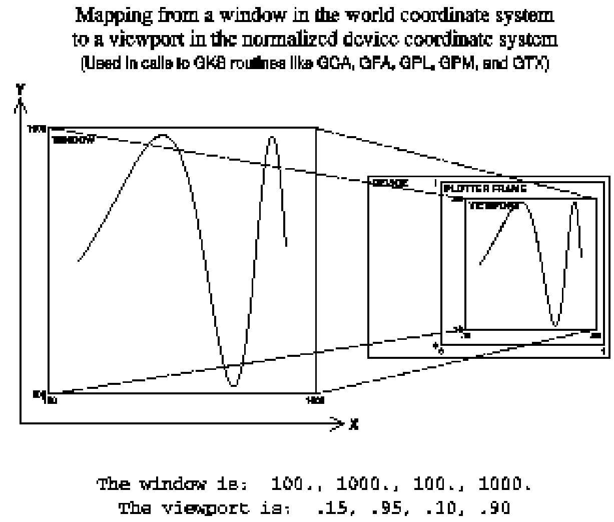
Every call to SET redefines GKS normalization transformation 1 with the new viewport and window coordinates, axis scaling, and axis mirror-imaging. This normalization transformation 1 will be in effect until it is changed by either another SET call or by a call to the GKS routine GSELNT.

Caution: If you use GSELNT to define a new normalization transformation, you will still have the mirror-imaging and linear/log scaling established by the last SET call. If the last SET call used nonlinear axis scaling or mirror-imaging of an axis, you may need to do another SET call to alter these settings.

Restoring the normalization transformation

To save the state of a normalization transformation, set and use a new transformation, and then restore the state of the first transformation, use the GETSET and SET routines. Use GETSET to access the normalization transformation parameters and save them by using unique argument names. You can then call SET with the new transformation and do what is needed. Then, when you are ready to restore the original normalization transformation, call SET again, using the saved parameters.

Restoring a normalization transformation



Code segment from fcoord1.f

```
      CALL GETSET(VPLS,VPRS,VPBS,VPTS,WDLS,WDRS,WDBS,WDTS,LSS)
      CALL SET(VPL,VPR,VPB,VPT,1000.,100.,100.,1000.,2)
      CALL SET(VPLS,VPRS,VPBS,VPTS,WDLS,WDRS,WDBS,WDTS,LSS)
```

Synopsis

```
      CALL GETSET (VPL, VPR, VPB, VPT, WDL, WDR, WDB, WDT, LS)
```

Arguments

VPL, VPR, VPB, and VPT

Real, Output---Normalized device coordinates of the existing viewport. The letters "L," "R," "B," and "T" refer to the left, right, bottom, and top edges of the viewport, respectively. VPL must be less than VPR, and VPB must be less than VPT.

WDL, WDR, WDB, and WDT

Real, Output---User coordinates of the existing window. The letters "L," "R," "B," and "T" refer to the left, right, bottom, and top edges of the window, respectively.

LS

Integer, Output---Current value of the log-scaling parameter.

Discussion

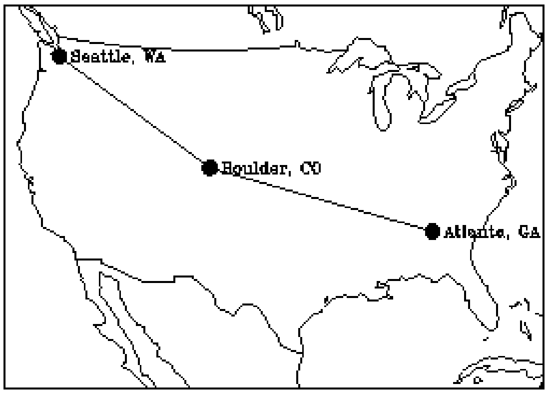
Line 1 of the **fcoord1.f** code segment shows a call to GETSET to save all of the existing transformation parameters. Line 2 defines a new normalization transformation 1 including mirror-imaging on the X axis and logarithmic scaling on the Y axis. Line 3 resets the normalization transformation, which will return mirror-imaging and axis scaling to their original values.

See "Appendix A: The use of X/Ycoordinates in NCAR Graphics" for a complete description of NCAR Graphics coordinate systems.

Chapter 9: Drawing points, markers, and filled dots

NCAR Graphics offers several methods for drawing points and markers on a plot. You can create a single marker or a series of markers at any location and draw lines between them. You can also create filled circular dots with any size and color. The following modules show you how to draw a variety of points and markers, connect them with lines, and set color options using GKS, SPPS, and NCAR Graphics utilities.

Polymarkers and filled dots



User entry points for drawing markers

There are three sets of routines for drawing markers. This table lists the routines in each set: The GKS routines, the SPPS routines, and the NCAR Graphics routines.

GKS routines

GPM
Draws markers at specified locations.

GSMK
Sets a polymarker type (dot, plus, asterisk, circle or cross).

GSMKSC
Controls the size of the marker. Note, however, that the dot polymarker cannot be resized.

GSLWSC
Controls the line width of the segments connecting the markers.

GSCR
Creates a color table.

GSPMCI
Sets the index into the color table that is used to color polymarkers (dots, plus signs, asterisks, circles, and crosses).

GSPLCI
Sets the index into the color table that is used to color polylines (lines and curves).

GQCR
Gets the red-green-blue values when given a color table index.

GQPMCI
Gets the polymarker color index.

GQPLCI
Gets the polyline color index.

GQMK
Gets the polymarker type.

GQMKSC
Gets the polymarker size scale factor.

SPPS routines

POINT
Draws a point at a specified location.

POINTS
Draws a series of markers at specified locations. The markers can optionally be connected by lines.

NCAR Graphics utility routines

NGDOTS

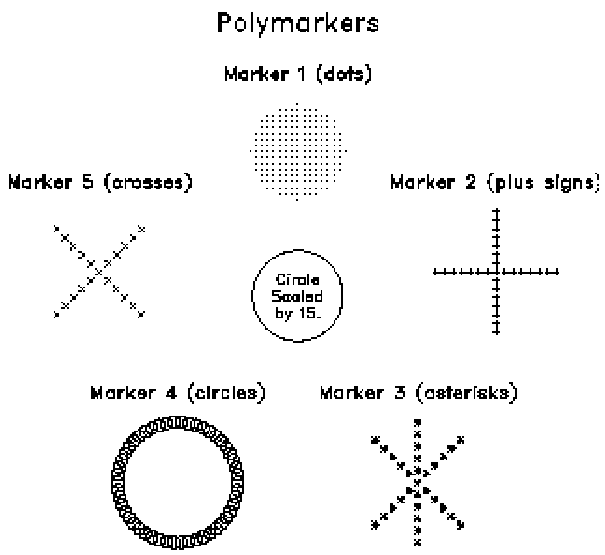
Draws filled circular dots at specified locations. The size and color of the dots are programmable.

Creating markers with GKS routines

The GPM routine draws GKS polymarkers at the locations specified by input coordinates. A polymarker can be either a dot, a plus sign, an asterisk (the default), a circle, or a cross.

This module discusses how to draw, scale, and color a polymarker using GKS routines.

GKS polymarkers



Code segment from fgkgpm.f

```
CALL GSMK(4)
CALL GSPMCI(1)
CALL GSMKSC(2.)
CALL GPM(JL,XM1,YM1)
CALL PCSETI('CD',1)
CALL GSPLCI(1)
CALL PLCHHQ(X0,Y0+R+.05,'Marker 4 (circles)',.018,0.,0.)
```

Synopsis

```
CALL GPM (JL, XM1, YM1)
```

Arguments

JL

Integer, Input---The number of markers to be drawn. (JL>0)

XM1, YM1

Real Array, Input---The X and Y world coordinates of the JL markers.

Discussion

By default, the polymarker type is an asterisk and the scaling factor is 1., so, if you want to change them, GPM should be called after these characteristics are set.

The **fgkgpm.f** code segment is responsible for drawing and labeling the circle of linked circles (Marker 4) in the lower left corner of the output plot.

Line 1 sets the polymarker type to a circle. GSMK takes an integer argument, which specifies the marker type. Valid values are 1 (dot), 2 (plus), 3 (asterisk), 4 (circle), and 5 (cross).

Line 2 specifies the color index in the color table that is used to color the polymarkers. The color table was set up earlier in the code using the GSCR routine.

Line 3 calls the GSMKSC procedure that scales the marker by a factor of two. The argument to GSMKSC must be a real value greater than or equal to 0.

Line 4 calls GPM and draws a series of circles. JL is the number of circles drawn and the XM1 and YM1 arrays specify the X and Y world coordinates of each marker. These coordinates were set prior to entering this code segment.

Line 5 selects the duplex character set.

Line 6 sets the polyline color index that is used for coloring text.

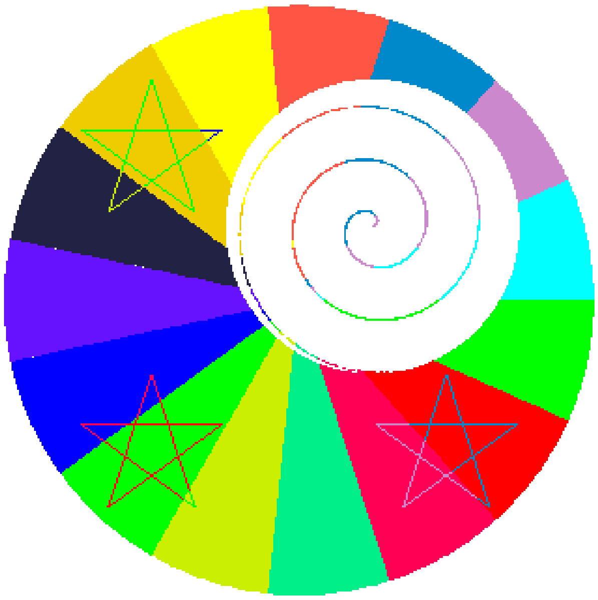
Line 7 draws the text above the set of circle polymarkers.

Creating dots with SPPS routines

The SPPS routine POINT draws a point at a location determined by an X and Y user coordinate. The SPPS POINT routine differs from the GKS GPM routine in that it allows for mirror imaging of axes and non-linear axes through the use of the SET call, which GKS does not allow. Also, GPM uses world coordinates and POINT uses user coordinates.

This module discusses how to create and color points using the SPPS routine POINT.

SPPS POINT



Code segment from fsppoint.f

```
1        CALL SET (.01,.99,.01,.99,-1.,1.,-1.,1.,1)
2        DO 108 ING=1,1500
3               RAD=REAL(ING)/1000.
4               ANG=DTR*REAL(ING-1)
5               XCD=.25+.5*RAD*COS(ANG)
6               YCD=.25+.5*RAD*SIN(ANG)
7               CALL SFLUSH
8               CALL GSPLCI(IT1)
9               CALL POINT (XCD,YCD)
10 108   CONTINUE
11       CALL FRAME
```

Synopsis

```
CALL POINT (PX, PY)
```

Arguments

PX, PY

Real, Input---The X and Y user coordinates defining the location where the dot is to be drawn. The GKS routine GPM can also be used to draw a dot in world coordinates; however, NCAR Graphics user coordinates allow for mirror imaging of axes, which GKS does not. See "Appendix A: The use of X/Y coordinates in NCAR Graphics" for a description of these coordinate systems.

Discussion

The **fsppoint.f** code segment draws the spiral line in the example image using the POINT routine.

Line 1 sets the mapping between the fractional coordinate system and the user coordinate system. The coordinates used by the POINT routine are user coordinates.

Line 2 enters a loop that calculates the coordinates for the points on the spiral and calls POINT for each new coordinate calculated.

Lines 3 through 6 calculate and save the coordinates for the spiral function in the XCD and YCD variables.

Line 7 flushes the output buffer. Because of buffering, objects created by a mixture of calls to SPPS routines and GKS routines may be drawn in the wrong order. Flushing the buffer avoids this problem.

Line 8 sets the color of the dot. Notice that to set point color using the POINT routines, you use the polyline color index and not the polymarker color index. This is because of the way points are drawn by POINT. In this example segment, the code for determining the value of IT1 (and therefore the color) was left out because of space considerations. For a complete example, see the source file, **fsppoint.f**.

Line 9 draws a single dot at the XCD, YCD coordinates.

Line 10 ends the loop.

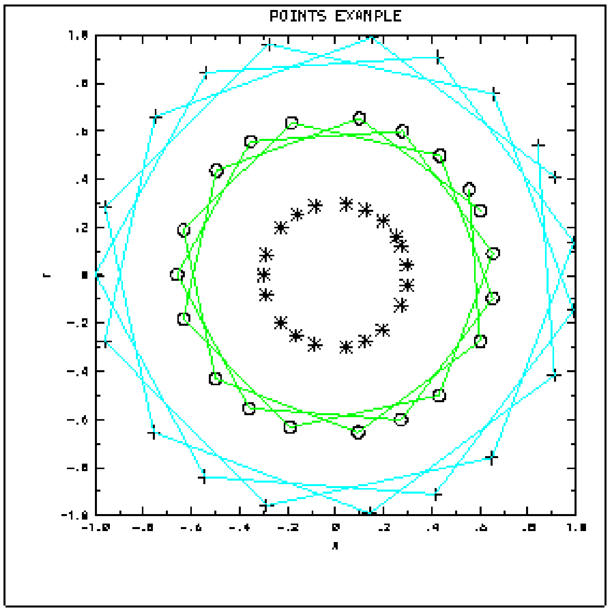
Line 11 closes the picture.

Creating markers connected with lines in SPPS

POINTS is an SPPS routine that draws a series of markers that are optionally connected with line segments. The marker can be one of the following types: dot, asterisk, cross, plus, circle, or the FORTRAN77 character CHAR(*value*).

This module demonstrates how to create a series of markers, connect them with line segments, and select color options.

Connecting markers with the SPPS POINTS routine



Code segment from fsppoints.f

```
CALL AGSETI ('SET.',-1)
CALL EZXY (XDRA,YDRA,NPTS,'POINTS EXAMPLE$')
CALL GSMKSC(2.0)
CALL GSPLCI(7)
CALL GSLWSC(3.0)
CALL POINTS (XDRA,YDRA,NPTS,-2,1)
CALL GSPLCI(9)
CALL POINTS (X2DRA,Y2DRA,NPTS,-4,1)
CALL POINTS (X3DRA,Y3DRA,NPTS,-3,0)
```

Synopsis

```
CALL POINTS (PX, PY, NP, IC, IL)
```

Arguments

PX, PY

Real Array, Input---The X and Y user coordinates where a series of NP markers are to be drawn.

NP

Integer, Input---The number of markers to be drawn.

IC

Integer, Input---Describes the type of marker to be drawn based upon the following values of IC:

0

Draws a dot (.).

-1

Draws a dot (.).

-2
: Draws a plus (+).

-3
: Draws an asterisk (*).

-4
: Draws a circle (o).

-5
: Draws a cross (x).

If IC is 0 or less than 0, then IC specifies the GKS polymarker's type. The polymarker size and color can also be set by calling the appropriate GKS routines, GSMKSC (scaling the polymarker), GSCR (creating a color table), and GSPMCI (setting the polymarker color index), before calling POINTS. Note, however, that the dot polymarker cannot be resized.

>0
: Draws the FORTRAN77 character, CHAR(IC). In these cases, the GKS routine GTX is used to draw the specified character CHAR(IC) at each of the NP points. The text attributes such as orientation, height, and color can be set by calling the GKS text attribute setting routines before the call to the POINTS routine.

IL
: *Integer, Input*---This parameter determines if the markers are to be connected with line segments. If IL equals 0 then the markers are left unconnected. Otherwise the markers are connected with line segments.

Discussion

The polymarker attributes that can be set are discussed above. The polyline attributes that apply to the segments connecting the polymarkers are line type (solid, dashed, dotted, dashed dotted), line width, and line color. The type is set with the GKS routine GSLN. The line width is set with the GKS routine GSLWSC, and the color is set with the GKS routines GSCR and GSPLCI.

Line 1 of the **fsppoints.f** code segment suppresses the drawing of curves by the EZXY routine.

Line 2 draws the background using EZXY.

Line 3 scales the size of the polymarkers up by a factor of 2.0.

Line 4 sets the color of the polyline to color table index 7. The color table was set up prior to entering the code with the GSCR routine.

Line 5 sets the width of the polyline to 3.

Line 6 draws the outer group of plus signs and connects them with line segments.

Line 7 sets the color of the polyline to color table index 9.

Line 8 draws the middle group of circles and connects them with line segments.

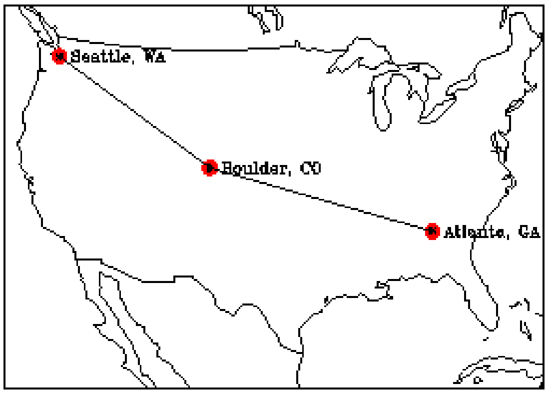
Line 9 draws the inner group of asterisks. These markers are not connected by line segments because the last argument in the POINTS call is 0.

Creating filled circular dots

NGDOTS is an NCAR Graphics routine that draws filled circular dots at specified coordinate locations. The routine also lets you control the size and color of the dots.

This module demonstrates how to draw filled circles with NGDOTS and adjust their size and color.

Drawing filled circles with NGDOTS



Code segment from fngngdts.f

```
      CALL MAPSTC ('OU - OUTLINE DATASET SELECTOR',OUTLN)
      CALL MAPROJ (PROJ,PLAT,PLON,ROTA)
      CALL MAPSET (JLIM,PLIM1,PLIM2,PLIM3,PLIM4)
      CALL MAPSTR ('GR',0.)
      CALL MAPDRW
      DO 100 I=1,NUMSTS
             CALL MAPTRA(LAT(I),LON(I),X,Y)
             XCOOR(I) = X
             YCOOR(I) = Y
             CALL PLCHHQ (X+.015,Y,CITY(I),.015, 0., -1.)
             INDICE = I
 100  CONTINUE
      IF (XCOOR(INDICE).NE.1.E12) THEN
              CALL NGDOTS (XCOOR,YCOOR,NUMSTS,.02,15)
      ENDIF
      CALL POINTS (XCOOR, YCOOR, NUMSTS, -3, 1)
```

Synopsis

```
CALL NGDOTS (X, Y, NUM, SIZE, ICOLOR)
```

Arguments

X, Y

Real Array, Input---The X and Y world coordinates where a series of circular filled dots are to be drawn. The coordinates must be world coordinates and not user coordinates. NGDOTS does not respect the log scaling or axis reversal options of the SET call and will report a warning if these are not set to their default values.

NUM

Integer, Input---The number of dots to be drawn.

SIZE

Real, Input---The dot diameter in world coordinate Y-axis units.

COLOR

Integer, Input---The value of the GKS color index specifying what color the dots will be.

Discussion

The dots created by NGDOTS are scaled so that they will be circular when the normalization transformation does not preserve the aspect ratio.

Line 1 of the **fngngdts.f** code segment sets the type of map outlines.

Line 2 sets the map projection.

Line 3 sets the map limits.

Line 4 turns off the grid lines in the map.

Line 5 draws the map.

Lines 6 through 12 define a loop that transforms the map coordinates to world coordinates for each of the three cities and plots a label (the city name) at each position.

Line 13 tests if the transformation resulted in valid coordinates.

Line 14 draws NUMSTS dots with a size of .02 and a color mapped to the color table entry 15.

Line 16 draws asterisks on top of the filled dots and connects them with line segments.

Chapter 10: Drawing lines and curves

There are several methods for drawing lines and curves (polylines) on a plot and controlling line characteristics, like the dash pattern, color, width, and smoothness. The following modules demonstrate how to draw curves and set their characteristics using GKS, SPPS, and higher level routines.

NCAR Graphics lines and curves

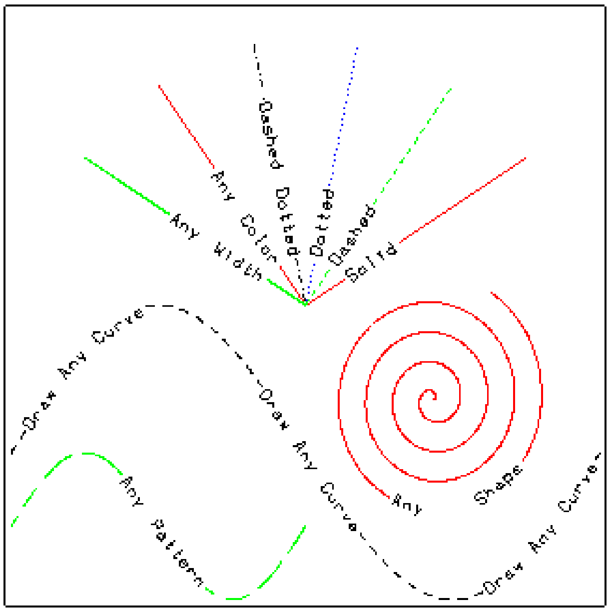


Table of line and curve-drawing user entry points

There are three levels of user entry points for drawing lines and curves. This module organizes the routines according to their level (GKS, SPPS, and higher level NCAR Graphics routines).

GKS routines

GPL
: Draws a connected set of line segments specified by X and Y coordinate arrays.

GSLN
: Sets the line type to either a solid, dashed, dotted, or dashed-dotted line.

GSLWSC
: Sets the line width scale factor (thickness) of a polyline.

GSCR
: Sets up a color table and associates a red-green-blue color value with an index.

GSPLCI
: Sets the polyline color index.

GQLN
: Retrieves the current setting for the line type.

GQLWSC
: Retrieves the current setting for the line width scale factor (thickness).

GQPLCI
: Retrieves the current polyline color index setting.

SPPS routines

LINE
: Draws a line between two points when given two user endpoint coordinates.

CURVE
: Draws a curve defined by X and Y user coordinate arrays.

PLOTIF
: Moves the plotter pen to a fractional coordinate position, or flushes the SPPS polyline buffer.

VECTOR
: Draws a line segment from the current plotter pen position to the given X and Y user coordinates. This works in conjunction with the FRSTPT routine.

FRSTPT
: Sets the current plotter pen position when given user coordinates and is used in conjunction with the VECTOR routine.

Higher level routines

LINED
: Draws a single line between two points in world coordinates. Crowded-line removal is possible with this routine.

CURVED
: Draws a curve defined by X and Y world coordinate arrays. Curve smoothing and crowded-line removal is possible with this routine.

VECTD
: Draws a line from the current plotter pen coordinates to the given world coordinates. Curve smoothing and crowded-line removal is possible with this routine. VECTD works in conjunction with FRSTD and LASTD.

FRSTD
: Sets the current plotter pen position when given world coordinates. FRSTD works in conjunction with VECTD and LASTD.

LASTD
: Flushes buffers and finishes drawing a line after calling FRSTD, VECTD, or LINED.

DASHDC
Chooses a dash pattern with labels.

DASHDB
Chooses a dash pattern without labels.

Table of line and curve-drawing parameters

For a complete description of SPPS parameters, see the next module and the **spps_params man** page.

SPPS parameters

Parameter	Brief description	Fortran type	Routines affected
PB	Polyline Buffer size	Integer (2-50)	PLOTIF

Internal parameters*

Parameter	Brief description	Fortran type	Routines affected
IPAU	Line and gap size	Integer	DASHDC, DASHDB
FPART	First solid segment factor	Real	DASHDC
IGP	Character GaP flag	Integer	DASHDC
ICLOSE	Pen-up parameter	Integer	DASHDC, DASHDB
IOFFS	Smoothing flag	Integer	DASHDC
TENSN	TENSioN factor	Real	DASHDC
NP	Point-interpolation parameter	Integer	DASHDC
SMALL	Saved points min. distance	Real	DASHDC
L1	Max. number of points saved	Integer	DASHDC
ADDLR	Number of free space units	Real	DASHDC
ADDTB	Number of free space units	Real	DASHDC
MLLINE	Max. crowded-LINE length	Integer	DASHDC

* These parameters are not accessible via parameter retrieving and setting routines at this time. They are available in common blocks. Please see the **man** pages for the routines affected.

Line and curve-drawing parameters: What they do and how to use them

The line and curve drawing routines have a single parameter for controlling line characteristics. SPPS parameters are controlled using the SETUSV routine. GKS polyline attributes will affect curves drawn using any NCAR Graphics routine. Please see the **spps_params man** page for more information on setting SPPS parameters.

Synopsis

```
CALL SETUSV (PNAM, IVAL)
```

Routines

SETUSV

Sets the value of one of the SPPS parameters.

SETUSV is a general purpose routine for setting several SPPS parameters; however, this module will only focus on setting the parameter that affects the curve and line drawing routines. For a complete description of all the SETUSV supported parameters, see the online **man** page documentation.

Arguments

PNAM

Character, Input---Consists of a two-letter name contained within single quotation marks (').

PB Sets the polyline buffer size used in the PLOTIF routine. The buffer is an efficiency mechanism for storing several polyline segments and then executing them together. The default size is 50. Setting the size to 2 effectively turns buffering off. Valid values range from 2 to 50, inclusive.

IVAL

Integer, Input---An integer value.

Discussion

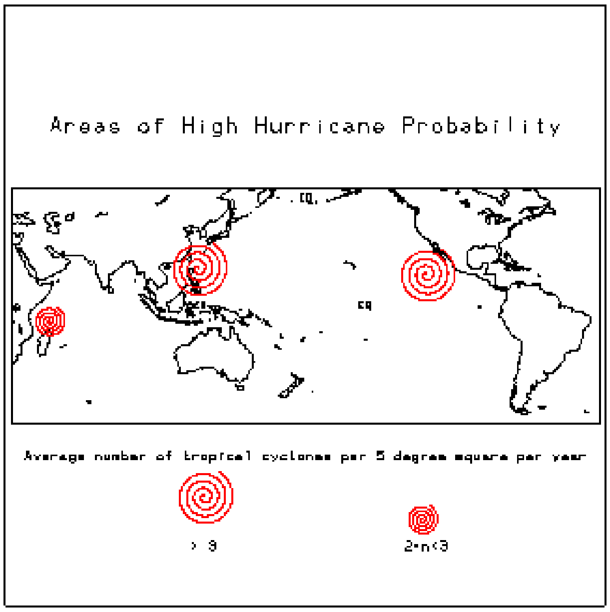
Currently, the internal parameters listed on the previous page for controlling the line characteristics can only be accessed through common blocks. There are no access routines for these parameters at the time of this printing. For more information on accessing these parameters, see the **dashline_params man** page.

Drawing lines and curves with GKS routines

GPL is used to draw a series of connected line segments when given X and Y input arrays of world coordinates.

This module demonstrates how to set the line type and how to draw lines using GKS routines.

Drawing lines and curves with GKS routines



Code segment from fgkgpl.f

```
      REAL    XCD(1500), YCD(1500)
      CALL GSPLCI (2)
      DO 40 ING=1,1500
         RAD=REAL(ING)/1000.
         ANG=DTR*REAL(ING-1)
         XCD(ING)=.25+.5*RAD*COS(ANG)
         YCD(ING)=.25+.5*RAD*SIN(ANG)
 40   CONTINUE
      CALL SET (.24,.37,.48,.61,-1.,1.,-1.,1.,1)
      CALL GPL(1500, XCD, YCD)
      CALL SET (.03,.10,.43,.50,-1.,1.,-1.,1.,1)
      CALL GPL(1500, XCD, YCD)
```

Synopsis

```
CALL GPL (NPTS, XCD, YCD)
```

Arguments

NPTS

Integer, Input---The number of points defining the line to be drawn. NPTS>1.

XCD(NPTS), YCD(NPTS)

Real arrays, Input---The X and Y world coordinates of the points that define the curve to be drawn.

Discussion

The **fgkgpl.f** code segment demonstrates how spiral lines are drawn off the east coast of Africa and Southeast Asia using the GKS GPL routine.

Line 1 defines the X and Y input coordinate arrays.

Line 2 sets the current polyline color index to 2. The color table and associated indices were set up prior to entering this code segment.

Lines 3 through 8 compute the X and Y input coordinates for a spiral line.

Line 9 calls the SET routine to define the mapping from the fractional coordinate system of the plotter to the user coordinate system. This effectively positions the spiral on the frame off the coast of Southeast Asia.

Line 10 calls GPL to draw the spiral. The first argument specifies that there are 1,500 coordinates in the line. The second and third arguments are the arrays of coordinates defining the spiral.

Line 11 defines a new mapping and effectively positions the second spiral off the east coast of Africa.

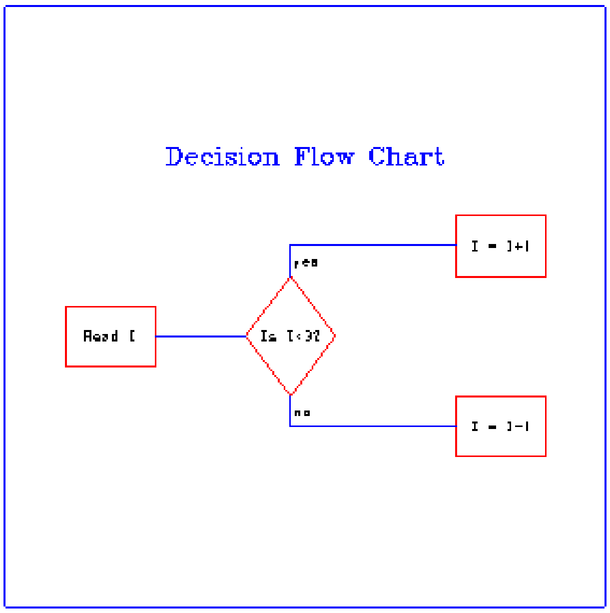
Line 12 draws the second spiral.

Drawing lines and curves with SPPS routines

The SPPS routine LINE can be used to draw a single line on a plot when given the line's endpoint coordinates.

This module demonstrates how to draw lines using the LINE routine.

Drawing a flow chart with the LINE routine



Code segment from fspline.f

```
1       CALL GSPLCI(2)
2       CALL SET (0.,1.,0.,1.,0.,20., 0.,20.,1)
3       CALL GSLWSC(4.)
4       CALL LINE(2.,8.,2.,10.)
5       CALL LINE(2.,10.,5.,10.)
6       CALL LINE(5.,10.,5.,8.)
7       CALL LINE(5.,8.,2.,8.)
8       CALL PLCHLQ(3.5,9.,'Read I',15.,0.,0.)
```

Synopsis

```
CALL LINE (X1, Y1, X2, Y2)
```

Arguments

X1, Y1

Real, Input---The X and Y user coordinates of the starting point of the line segment.

X2, Y2

Real, Input---The X and Y user coordinates of the endpoint of the line segment.

Discussion

The SPPS routine LINE is used to draw a line between two points. Line characteristics like polyline type, line width, and color can be set by calling the GKS routines GSLN (set line type), GSLWSC (set line width), GSCR (create color table), and GSPLCI (set color index) before calling LINE.

The **fspline.f** code segment draws a single box and places the text "Read I" in its center.

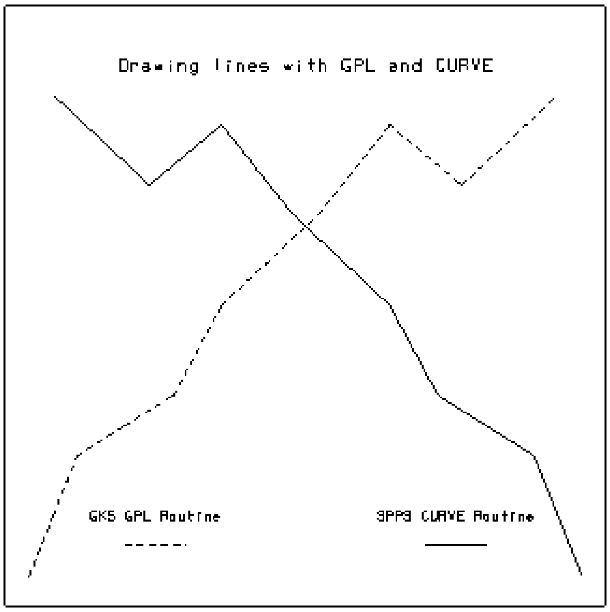
Line 1 sets the current polyline color index. A color table was set up previously in the code using the GKS GSCR routine. Line 2 sets up a mapping between the plotter coordinates and the user coordinate system. Line 3 sets the line width to 4. Lines 4 through 7 make four calls to the LINE routine to draw the four sides of a box. Line 8 draws the text, "Read I", in the center of the box that was just drawn.

Drawing lines and curves with SPPS routines

The SPPS routine CURVE draws a line through a set of X and Y user coordinates.

This module demonstrates how to draw curves using CURVE and compares it with the GKS routine GPL.

Drawing curves with the CURVE routine



Code segment from fspcurve.f

```
DATA XCOORD/1.,2.,3.,5.,7.,9.,13.,16.,19.,23./
DATA YCOORD/1.,3.,5.,6.,7.,10.,13.,16.,14.,17./

CALL SET (0.,1., 0., 1., 25., 0., 0., 20., 1)
CALL GSLN(2)
CALL GPL(10,XCOORD,YCOORD)
CALL GSLN(1)
CALL CURVE(XCOORD,YCOORD,10)
CALL FRAME
```

Synopsis

```
      CALL CURVE (XCD, YCD, NPTS)
```

Arguments

XCD, YCD

Real arrays, Input---The X and Y user coordinates of the points that define the curve to be drawn.

NPTS

Integer, Input---The number of points defining the curve.

Discussion

The SPPS routine CURVE draws a curve defined by the input X and Y user coordinates. The plotter pen is left at the location of the last point in the curve.

The code segment and graphic demonstrate how to use the CURVE routine and how it differs from the GKS curve-drawing routine, GPL.

If the routine SET is called to reverse the axes or use logarithmic mappings, then you should use the CURVE routine instead of GPL, since GPL does not recognize axis reversal or logarithmic mappings.

In the **fspcurve** example, CURVE and GPL plot the same data points. However, since the SET call reversed the X axis, GPL does not draw the curve correctly.

Lines 1 and 2 of the **fspcurve.f** code segment initialize the X and Y coordinates of the line to be drawn.

Line 4 sets up the mapping from plotter space to user coordinates. Notice that we have reversed the X axis to map from 25 to 0.

Line 5 sets the line type to "dashed" using the GKS GSLN routine. GSLN takes a single integer argument, which specifies the type of line to be drawn. The argument may be one of the following:

> 1 Solid line (default)
> 2 Dashed line
> 3 Dotted line
> 4 Dashed-dotted line

Until the line type is reset, all line-drawing routines will draw lines with the type specified by GSLN.

Line 6 calls the GKS routine GPL to draw a dashed line. GPL will ignore the axis reversal specified with the SET call.

Line 7 sets the line type to "solid."

Line 8 calls the SPPS routine CURVE to draw a solid line. CURVE recognizes the axis reversal and therefore mirrors the curve drawn by the GPL routine.

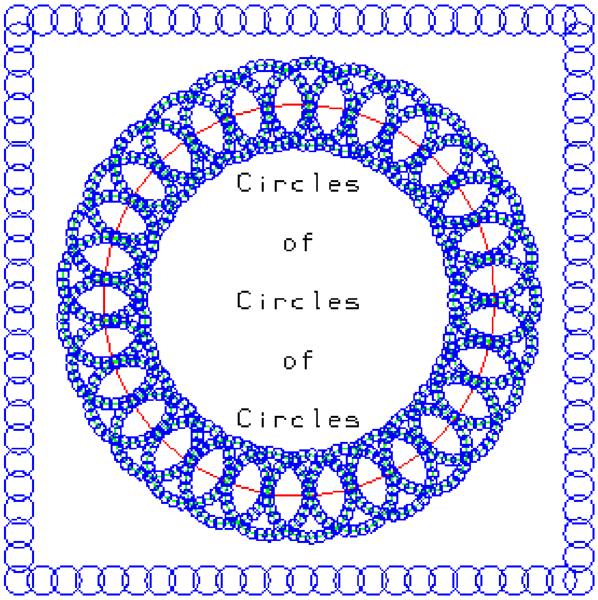
Line 9 advances the frame.

Drawing lines and curves with high level routines

The high level routine CURVED draws curves in the same manner as the SPPS routine, CURVE. CURVED, however, has more functionality, which is discussed in later modules.

This module demonstrates how to draw curves using CURVED and set line characteristics such as color and width.

Drawing circles with the CURVED routine



Code segment from fdlcurvd.f

```
      REAL XCOOR2(120), YCOOR2(120)

      CALL SET(.01,.99,0.01,.99,0.,1.,0.,1.,1)
      CALL GNCRCL(.5,.5,.333,30,XCOOR2, YCOOR2)
      CALL GSLWSC(5.)
      CALL GSPLCI(2)
      CALL CURVED(XCOOR2,YCOOR2,30)
      CALL PLCHLQ(.5,.7,'Circles',.03,0.,0.)
      CALL PLCHLQ(.5,.6,'of',.03,0.,0.)
      CALL PLCHLQ(.5,.5,'Circles',.03,0.,0.)
      CALL PLCHLQ(.5,.4,'of',.03,0.,0.)
      CALL PLCHLQ(.5,.3,'Circles',.03,0.,0.)
```

Synopsis

```
      CALL CURVED (XCD, YCD, NPTS)
```

Arguments

XCD, YCD

Real arrays, Input---The X and Y user coordinates of the points that define the curve to be drawn.

NPTS

Integer, Input---The number of points defining the curve.

Discussion

The high level CURVED routine draws a curve defined by the input X and Y user coordinates.

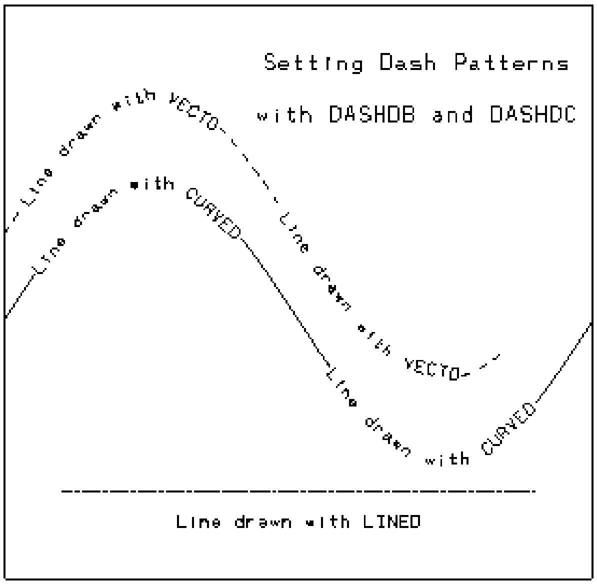
The **fdlcurvd.f** code segment and graphic demonstrate how to use the CURVED routine and set line characteristics like color and line width. Every circle in the graphic was drawn with a call to CURVED. The code segment draws the large circle in which the text "Circles of Circles of Circles" is centered. Although CURVED was used only to draw circles in this example, it can be used to draw curves of any shape.

Line 1 declares the array of user coordinates. Line 3 sets up the mapping between the plotter and user coordinates. In this example, the user coordinates range from 0. to 1. in both the X and Y directions. Line 4 calls a routine that initializes the XCOOR2 and YCOOR2 arrays with the coordinates of a circle centered at (.5,.5), a radius of .333, and 30 points, which define it. Line 5 sets the line width to 5 (1 is the default). Line 6 sets the color of the line. The color table was set up prior to entering this code segment with the GKS GSCR routine. The argument (2) passed to the GSPLCI routine represents an index in the color table. Line 7 draws the large circle in the center of the graphic. Lines 8 through 12 draw text in the center of the large circle.

Setting the dash pattern of lines and curves

When using the higher level routines like CURVED, LINED, and VECTD, you can represent lines with any dash pattern. Whereas the GKS GSLN utility only lets you set four line types (solid, dashed, dotted, and dotted-dashed), the DASHDC and DASHDB routines let you set any line type, including lines with labels. This module demonstrates how to set any dash pattern for your lines.

Setting line dash patterns



Code segment from fdldashd.f

```
        STR = '$''$''Line''drawn''with''VECTD$''$''$''$'''
        CALL DASHDC(STR,15,20)
        CALL FRSTD(0.,.60)
        DO 22 I=1,360
               Y = (SIN(REAL(I)*(TWOPI/360.)) * .25) + .60
               CALL VECTD(REAL(I)/360.,Y)
```

```
7   22  CONTINUE
8       STR = '$$$$$$Line''drawn''with''CURVED$$$$$$$$$$$$'
9       CALL DASHDC(STR,15,20)
10      DO 23 I=1,360
11              XCOORD(I) = REAL(I)/360.
12              YCOORD(I) = (SIN(REAL(I)*(TWOPI/360.)) * .25) + .45
13  23  CONTINUE
14      CALL CURVED(XCOORD, YCOORD, 360)
15      CALL DASHDB(63903)
16      CALL LINED(0.1,.15, .9,.15)
17      CALL PLCHLQ (0.5,0.10,'Line drawn with LINED' ,20.,0.,0.)
```

Synopsis

```
CALL DASHDC (CPAT, JCRT, JSIZE)
CALL DASHDB (IPAT)
```

Arguments

CPAT

Character string, Input---CPAT is a string of arbitrary length. However, 60 characters is a practical limit. The string specifies the dash pattern for lines drawn with CURVED, VECTD, and LINED. A dollar sign ($) indicates a solid, a single quotation mark (') indicates a gap, and blanks are ignored. Any other character is considered to be part of the line label. For example, if CPAT were set to $$$'A'Label'$$$, lines would be drawn similar to this: --- A Label ------ A Label ---. The pattern is repeated for the entire length of the line.

JCRT

Integer, Input---Specifies the length, in plotter address units, of the line or space represented by the dollar sign ($) or single quotation mark (') in the CPAT argument.

JSIZE

Integer, Input---Specifies the size of the plotted characters. If JSIZE is between 0 and 3, then the size represents 1., 1.5, 2., and 3. times an 8-plotter address unit width. Otherwise, it represents the character width in plotter address units.

IPAT

Integer, Input---IPAT is a 16-bit dash pattern (1=solid, 0=blank). For example, the binary number 1111000011110000, when converted to base 10, will give dashes of medium length.

Discussion

Either the DASHDB or the DASHDC routines can be used to set the dash pattern of a line. However, the DASHDC routine provides more flexibility by allowing the user to specify a label and character sizes along with the dash pattern.

The **fdldashd.f** code segment and graphic demonstrate how to set the dash pattern and line label of a line using DASHDC and DASHDB.

Line 1 initializes a string that specifies a dash pattern. Dollar signs ($) represent solid portions and single quotation marks (') represent spaces. Note that since FORTRAN77 string constants are enclosed in single quotation marks ('), a single quotation mark (') is represented by two adjacent single quotation marks, (''). Line 2 sets the dash pattern and specifies that the length of a line or space represented by the dollar sign ($) or single quotation mark (') is 15 plotter address units. The last argument specifies the character width in the line label to be 20 plotter address units.

Line 3 calls the FRSTD routine to put the plotter pen down at a point specified in world coordinates. FRSTD must be called prior to calling VECTD to position the plotter pen. This represents the starting point of a line segment drawn by the VECTD routine.

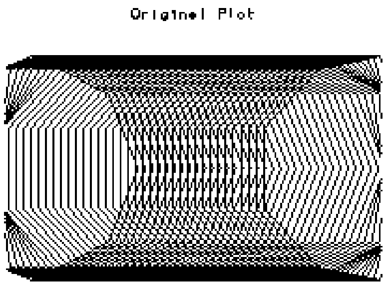
Lines 4 through 7 draw a dashed sine curve using the VECTD routine. VECTD takes two arguments, the X and Y world coordinates of the new plotter pen position. Line 8 initializes a string with a new dash pattern. Line 9 sets the dash pattern. Lines 10 through 13 specify the coordinates for the second sine curve. Line 14 draws the second curve using CURVED.

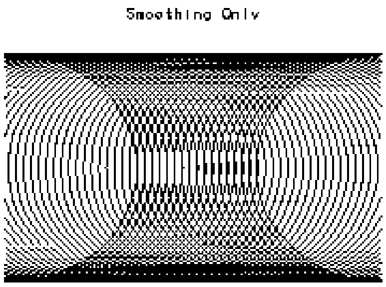
Line 15 sets a new dash pattern using DASHDB. 63903 (decimal) = 1111100110011111 (binary); therefore, line 15 specifies a dashed line with unequal lengths of solid portions and gaps. Line 16 draws a straight line with the LINED routine. LINED takes four arguments. The first two are the X and Y world coordinates of the starting point of the line, and the second two are the X and Y world coordinates of the ending point of the line. Line 17 draws a label under the last line.

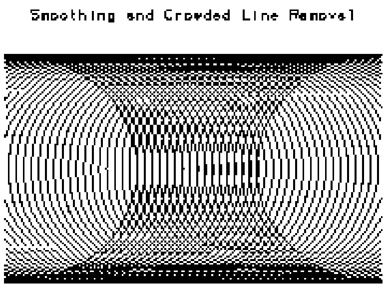
Smoothing curves and removing crowded lines

You can smooth curves and remove crowded lines by loading special libraries when you link your NCAR Graphics program. This module demonstrates the effect of smoothing and crowded-line removal and how to load the libraries that control this.

The effect of curve smoothing and crowded line removal







Code segment from fdlsmth.f

```
      REAL XCOORD(10), YCOORD(10)

      CALL RESET
      DO 10 I=1,50
             CALL GNCRCL(REAL(I)/50.,.5,.333,10,XCOORD, YCOORD)
             CALL CURVED(XCOORD,YCOORD,10)
  10  CONTINUE
      CALL PLCHLQ(.5,.95,'Smoothing and Crowded Line Removal',.02,0.,0.)
      CALL FRAME
```

Synopsis

```
CALL RESET
```

Arguments

None.

Discussion

The RESET routine only needs to be called when you want to remove crowded lines; it is not needed for line smoothing. The RESET routine initializes a special buffer that is used in crowded-line removal. It should be called every time you begin to draw into a new frame.

The three graphics on the opposite page demonstrate the effect of smoothing and crowded-line removal. The effects of curve smoothing are obvious in this example. However, the effects of crowded-line removal are harder to detect. The lines are most crowded along the top and bottom of the plot. If you look closely at the third image, you will notice some of the crowded lines that are present in the two previous images have been removed, especially near the top and bottom of the plot.

Smoothing is turned on by loading the libdashsmth.o library when you link your program. This is done by adding the **-smooth** option to your **ncargf77** command line:

ncargf77 -smooth file.f

or by explicitly loading libdashsmth.o when you compile:

f77 file.f $NCARG_ROOT/lib/ncarg/robj/libdashsmth.o ...

Crowded-line removal and smoothing are turned on by loading the libdashsupr.o library when you link your program. This is done by adding the **-super** option to your **ncargf77** command line:

ncargf77 -super file.f

or by explicitly loading libdashsupr.o when you compile your program:

f77 file.f $NCARG_ROOT/lib/ncarg/robj/libdashsupr.o ...

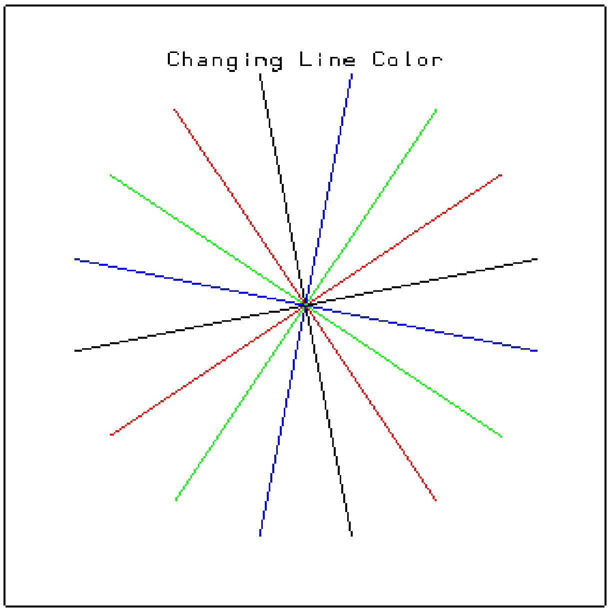
Each of the three example graphics was generated from the same source code, but the last two graphics were compiled with the **-smooth** and **-super** options, respectively. The first graphic was created without using any options.

Line 1 of the **fdlsmth.f** code segment declares the X and Y coordinate arrays that will contain the user coordinates for drawing the curves. Line 3 calls RESET to initialize the plot buffer. This call was only needed for the last graphic, which used crowded-line removal. If we only wanted to do curve smoothing, the RESET routine could have been omitted. Lines 4 through 7 draw 50 circles across the frame. In this example, we call CURVED. However, line smoothing and crowded-line removal work with the VECTD routine also. Smoothing does not effect the output of the LINED routine, but crowded-line removal does. Line 8 labels the plot. Line 9 advances the frame.

Changing line color

The color of curves and lines are changed by calling the GKS routine GSPLCI. This module demonstrates how to change the color of lines.

Changing line color



Code segment from fgklnclr.f

```
      CALL GSCR (1,0,1.,1.,1.)
      CALL GSCR (1,1,0.,0.,0.)
      CALL GSCR (1,2,1.,0.,0.)
      CALL GSCR (1,3,0.,1.,0.)
      CALL GSCR (1,4,0.,0.,1.)
      DTHETA = TWOPI/NPTP
      DO 10 I=1,16
         ANG = DTHETA*REAL(I) + .19625
         XC = RP*COS(ANG)
```

```
10         YC = RP*SIN(ANG)
11         CALL GSPLCI (MOD(I,4)+1)
12         CALL GSLWSC(6.)
13         CALL LINED(X0P,Y0P,X0P+XC,Y0P+YC)
14  10  CONTINUE
```

Synopsis

```
      CALL GSCR (WKID, CI, CR, CG, CB)
      CALL GSPLCI (INDEX)
```

Arguments

WKID
: *Integer, Input*---The workstation identifier.

CI
: *Integer, Input*---A color index to be assigned a color of CR, CG, CB (red, green, blue). Color index 0 defines the background color.

CR
: *Real, Input*---An intensity of red between a minimum of 0. and a maximum of 1.

CG
: *Real, Input*---A green intensity between 0. and 1.

CB
: *Real, Input*---A blue intensity between 0. and 1.

INDEX
: *Integer, Input*---An index into the color table, which was created with previous GSCR calls.

Discussion

The GSCR and GSPLCI routines are used in combination to first create a color table and then pick a line color.

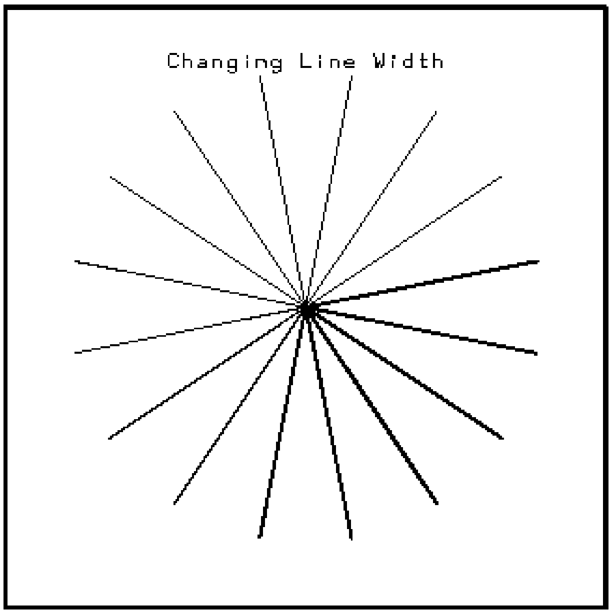
Lines 1 through 5 in the **fgklnclr.f** code segment call the GSCR routine to create a color table with five entries (indices 0 through 4). Line 1 defines the background color white (CR=1., CG=1., CB=1.) Lines 2, 3, 4, and 5 define the colors black, red, green, and blue, respectively. Line 6 initializes a constant. Line 7 enters a loop that draws 16 lines. Line 8 computes the line angle. Lines 9 and 10 compute the endpoint of a line. Line 11 sets the line color by specifying an index into the color table created by GSCR. Line 12 sets the line width. Line 13 draws the line. Line 14 ends the loop.

More information on color tables appears in Chapter 7 "Color tables and color mapping systems."

Changing line width

The line width of polylines is set using the GKS routine GSLWSC. This module demonstrates how to set the line width of curves and lines.

Setting line widths



Code segment from fgklnwth.f

```
      DO 10 I=1,16
         ANG = DTHETA*REAL(I) + .19625
         XC = RP*COS(ANG)
         YC = RP*SIN(ANG)
         CALL GSPLCI (1)
         CALL GSLWSC(REAL(I))
         CALL LINED(X0P,Y0P,X0P+XC,Y0P+YC)
  10  CONTINUE
```

Synopsis

```
CALL GSLWSC (WIDTH)
```

Arguments

WIDTH

Real, Input ---The scale factor to control the line width of the polyline to be drawn. It must be greater than or equal to 0. WIDTH is applied to the nominal line width on a given device, and the line widths are mapped to the nearest available line width representable on that device.

Discussion

The GSLWSC routine is used to set the width of polylines drawn on a plot. The lines in the example graphic were each drawn after setting the line width to a new value with the GSLWSC routine.

The **fgklnwth.f** code segment shows how the lines were drawn by increasing the line width on every iteration of the loop.

Line 1 begins the loop. Line 2 computes the line angle. DTHETA is a constant used to space the lines equally around a circle, and it is defined before entering the loop. Lines 3 and 4 compute the endpoint of a line. Line 5 sets the line color. Line 6 sets the line width. Line 7 draws a single line, and line 8 completes the loop.

Chapter 11: Drawing text and symbols

There are several methods for drawing text and symbols on a plot and controlling their characteristics, such as their color, angle, font, and position. The following modules demonstrate how to draw text and symbols and set their characteristics using GKS and higher level routines.

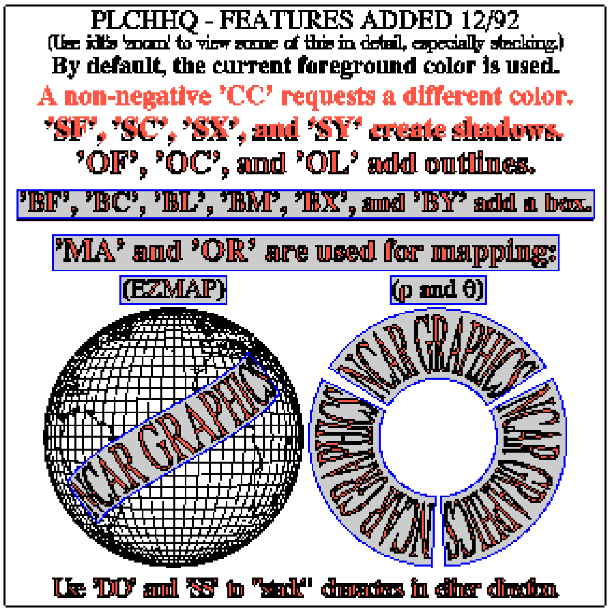


Table of text-drawing user entry points

There are two levels of user entry points for drawing text on a plot. This module lists the routines according to their level (either GKS or higher level NCAR Graphics routines).

GKS routines

GTX
: Draws a text string at a specified coordinate.

GSTXP
Sets the direction in which a string is drawn.

GSTXAL
Sets the horizontal and vertical alignment of text.

GSTXFP
Sets the GKS text font and precision.

GSCHH
Sets the character height.

GSCHSP
Sets the spacing between characters.

GSCHUP
Sets the angle at which a string is drawn.

GSCHXP
Sets the character aspect ratio.

GSCR
Creates a color table by associating a red-green-blue color value with an index.

GSTXCI
Sets the GKS text color.

GSPLCI
Sets the GKS polyline color.

GSLN
Sets the GKS polyline type.

GSLWSC
Sets the GKS line width.

GSELNT
Selects a predefined transformation from world coordinates to normalized device coordinates.

GQTXP
Gets the string direction as set by GSTXP.

GQTXAL
Gets the text alignment values.

GQTXFP
Gets the text font and precision.
Gets the character height.

GQCHSP
Gets the spacing between characters.

GQCHUP
Gets the string angle.

GQCHXP
Gets the character aspect ratio.

GQTXCI
Gets the GKS text color.

GQCNTN
Gets the current transformation value.

Higher level routines

PLCHHQ or PCHIQU
Plots characters of high quality from the fontcap databases and from two 564-character subsets of the PWRITX databases.

PLCHMQ or PCMEQU
Plots characters of medium quality from a Roman font with special characters.

PLCHLQ or PCLOQU
Plots characters of low quality by calling GKS character-plotting routines.

PCHIQU
Alternative name for PLCHHQ.

PCMEQU
Alternative name for PLCHMQ.

PCLOQU
Alternative name for PLCHLQ.

NGWSYM
Draws symbols from the standard WMO/NOAA meteorological fonts.

PCDLSC
Defines the default list of "special colors" used by PLCHHQ in drawing certain characters from the filled fonts. At the time of this printing, however, there are no characters for which the special colors are used. In the future there will be, and the routine will be of more use.

PCMPXY
Routine called by PLCHHQ and PLCHMQ to request mapping of characters from one X/Y coordinate system to another when the mapping flag MA is nonzero. Users can supply their own

version of this routine to create their own mappings.

PCPNWI
A character function that returns the names of parameter arrays.

PWRZI
Plots characters in 3-space when using the Isosurface utility.

PWRZS
Plots characters in 3-space when using the Surface utility.

PWRZT
Plots characters in 3-space when using the Threed utility.

PCSETI
Sets integer values for a specified Plotchar parameter.

PCSETR
Sets real values for a specified Plotchar parameter.

PCSETC
Sets character values for a specified Plotchar parameter.

PCGETI
Gets integer values for a specified Plotchar parameter.

PCGETR
Gets real values for a specified Plotchar parameter.

PCGETC
Gets character values for a specified Plotchar parameter.

Table of text-drawing parameters

For a complete description of the character-plotting parameters, see the **plotchar_params man** page or the Plotchar programmer document.

PLCHHQ or PCHIQU parameters

```
----------------------------------------------------------------------
Parameter  Brief Description                       Fortran type
----------------------------------------------------------------------
AS         Sets chAracter Spacing                  Real
BC         Box Color                               Integer array
BF         Box outline, Fill, and shadow           Integer
BL         Box Line width                          Real
BM         Box Margin width                        Real
BX         Box shadow X offset                     Real
BY         Box shadow Y offset                     Real
```

CC	SpeCial Color indices	Integer array
CD	Complex or Duplex dataset	Integer
CE	String CEntering	Integer
CH	Digitized Cartographic Height	Real
CL	PrinCipal Line width	Real
CS	Constant-Spacing flag	Real
CV	Cartographic Vertical spacing	Real
CW	Digitized Cartographic Width	Real
DB	Bottom text extent box Distance	Real
DL	Left text extent box Distance	Real
DO	Character Drawing Order	Integer
DR	Right text extent box Distance	Real
DT	Top text extent box Distance	Real
FB	Bezier curve Fidelity parameter	Real
FC	Function-Code character	Character
FN	Font Number or name	Character or Integer
IH	Digitized Indexical Height	Real
IS	Indexical size Sub- or superscripting offset	Real
IV	Indexical Vertical spacing	Real
IW	Digitized Indexical Width	Real
MA	MApping flag	Integer
OC	Outline Color specifier	Integer
OF	Outline Flag	Integer
OL	Outline Line width	Real
OR	Out-of-Range flag	Integer
PH	Digitized Principal Height	Real
PS	Principal size Sub- or superscripting offset	Real
PV	Principal Vertical spacing	Real
PW	Digitized Principal Width	Real
QU	QUality flag	Integer
SA	Character size multiplier	Real
SC	Shadow Color	Integer
SF	Shadow Flag	Integer
SL	Shadow Line width	Real
SS	Subtract Space flag	Real
SX	Shadow X offset	Real
SY	Shadow Y offset	Real
TE	Text-Extent computation flag	Integer
XB	Last string X Beginning -- output	Real
XC	Last string X Center -- output	Real
XE	Last string X End -- output	Real
YB	Last string Y Beginning -- output	Real
YC	Last string Y Center -- output	Real
YE	Last string Y End -- output	Real

PLCHMQ or PCMEQU parameters

Parameter	Brief description	Fortran type
HW	Character aspect ratio	Real
MA	MApping flag	Integer
OR	Out-of-Range flag	Real

PLCHLQ or PCLOQU parameters

Parameter	Brief Description	Fortran type
MA	MApping flag	Integer
OR	Out-of-Range flag	Real

Text-drawing parameters: What they do and how to use them

Plotchar parameters control many text characteristics such as shadow offset, character spacing, outline width, font choice, etc.

This module explains how to set and retrieve these parameters.

Synopsis

```
CALL PCGETC (PNAM, CVAL)
CALL PCGETI (PNAM, IVAL)
CALL PCGETR (PNAM, RVAL)
CALL PCSETC (PNAM, CVAL)
CALL PCSETI (PNAM, IVAL)
CALL PCSETR (PNAM, RVAL)
```

Routines

PCGETC, PCGETI, PCGETR

The "get" routines are used to retrieve the current value of a parameter PNAM. The value is returned in the second argument.
PCGETC is used to retrieve character parameters, PCGETI is used to retrieve integer parameters, and PCGETR is used to retrieve real parameters.

PCSETC, PCSETI, PCSETR

The "set" routines are used to assign a value (the second argument) to a parameter PNAM.
PCSETC is used to set character parameters, PCSETI is used to set integer parameters, and PCSETR is used to set real parameters.

Arguments

PNAM

Character expression, Input---A string that specifies the name of the parameter. This string begins with either one of the two-character mnemonics listed in the preceding parameter table or a two-character mnemonic followed by an index enclosed in parentheses when referencing a parameter with an array type. Only these leading characters are examined by Plotchar and must all be either uppercase or lowercase.

CVAL

Character, Input or Output---A character value. The characters must all be uppercase or lowercase when used as input.

IVAL

Integer, Input or Output---An integer value.

RVAL

Real, Input or Output---A real value.

Discussion

Most of the parameters are singly dimensioned, but a couple of them are arrays. The name of a particular element of a parameter array is formed by appending the index of the element, enclosed in parentheses, to the two-character name of the parameter array. If the index is a constant, this is easily done, however, if the index is a variable, forming the name is rather clumsy. The function PCPNWI is provided to make things a little easier for the user.

The following function call could be used to retrieve the name of the element of a parameter array:

```
PCPNWI (PNAM, IPAI)
```

PNAM is the parameter name and IPAI is an integer variable containing the desired index value. Note that any routine that uses PCPNWI must contain the following declaration:

```
CHARACTER * 16 PCPNWI
```

So, for example if IPAI equals 3, then the value of:

```
PCPNWI ('CC', IPAI)
```

is the character string 'CC(3)' (padded with blanks to the right), and the call:

```
PCSETI (PCPNWI ('CC', IPAI), 62)
```

sets element 3 of the CC array to 62. In this case, it would be easier to write:

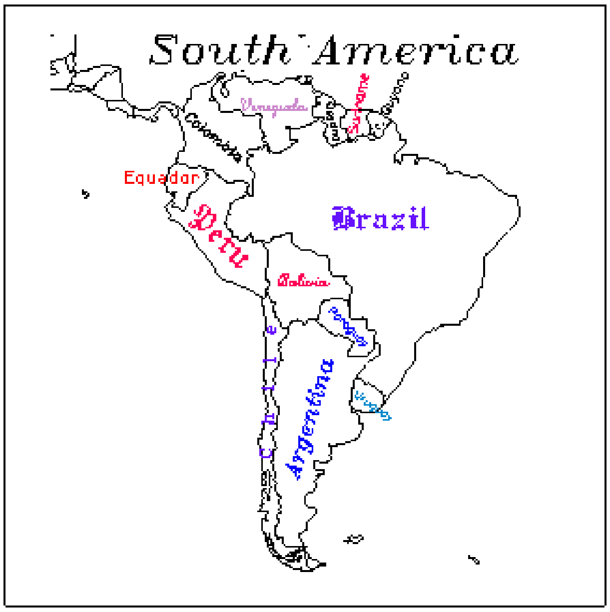
```
PCSETI (CC(3), 62)
```

but that misses the point that the variable IPAI will not always be known.

Drawing text with GKS routines

GTX is the GKS routine for drawing text. Text attributes like color, spacing, font, and so forth can be set by using GKS attribute-setting routines and then calling GTX. This module demonstrates how to set attributes and draw text with GKS routines.

Drawing text with GTX



Code segment from fgkgtx.f

```
      CALL DFCLRS
      CALL GSTXCI (4)
      CALL GSCHH (3.)
      CALL GSTXFP (-16, 2)
      CALL MAPTRA (-10., -60., X, Y)
      CALL GTX (X, Y, 'Brazil')

      CALL GSTXCI (4)
      CALL GSCHH (1.5)
      CALL GSTXFP (-7, 2)
      CALL MAPTRA (-40., -72., X, Y)
      CALL GSCHUP (-1., 0.03)
      CALL GSCHSP (2.)
      CALL GTX (X, Y, 'Chile')
```

Synopsis

```
CALL GTX (XCOORD, YCOORD, STRING)
```

Arguments

XCOORD, YCOORD

Real, Input---The X and Y user coordinates of the first character in STRING.

STRING

Character, Input---The string of characters to be drawn.

Discussion

GTX is used to draw GKS text-output primitives at the coordinates specified by XCOORD and

YCOORD. There are several attribute-setting routines for selecting the size, font, precision, and orientation of text. For detailed information about using these routines, see the *User's Guide for NCAR GKS-0A Graphics*.

The **fgkgtx.f** code segment plots the strings "Brazil" and "Chile" in the graphic. This code segment demonstrates how to set the character color, spacing, height, orientation, and font. These attributes must be set prior to making the GTX call that draws the text.

Line 1 calls a routine that creates a color table for specifying the character color.

Line 2 sets the character color. The argument passed to the GSTXCI routine is an index in the color table created in line 1. All text drawn with GTX will have this color until GSTXCI is called again.

Line 3 sets the character height. The argument to GSCHH is a real value representing the character height in user coordinates. This value is any real number greater than 0.

Line 4 sets the font. The first argument to GSTXFP is the integer value that determines the font. The default is 1, which specifies the ASCII font. This example sets the font to Hershey Gothic English. The second argument sets the type of precision used in subsequent calls to the GTX output primitive. The value 2 (the default) specifies the best possible stroke precision. Valid font numbers are 1 and any integer in the range -2 to -20. The absolute value of these font numbers corresponds to the fontcap databases 1 through 20 listed in a later module.

Line 5 calls the Ezmap routine MAPTRA; this converts the input latitude and longitude values into coordinates that can be used in the GTX routine. The MAPTRA call should be made after the Ezmap routine MAPINT or MAPDRW is called.

Line 6 draws the string "Brazil" using the attributes set above.

Line 8 sets the text color. In this segment the color is the same as the one used for the "Brazil" string, so this call is redundant.

Line 9 sets the character height.

Line 10 selects the Hershey complex Roman font. Please see the module "Fontcap databases 1-20" for a list of all valid GTX fonts.

Line 11 converts the input coordinates from latitude/longitude values to coordinates that can be used by GTX.

Line 12 establishes a vector direction parallel to a vertical line through a character's center. This sets the angle at which text will be drawn. The first argument is the X coordinate of a vector tip, and the second is the Y coordinate. The vector tail is at (0, 0). Thus the arguments 0.0, 1.0 would imply normal character positioning. This example, however, sets the vector at almost a right angle to this, and, therefore, any strings will be drawn in a direction that runs from the bottom to the top of the plot.

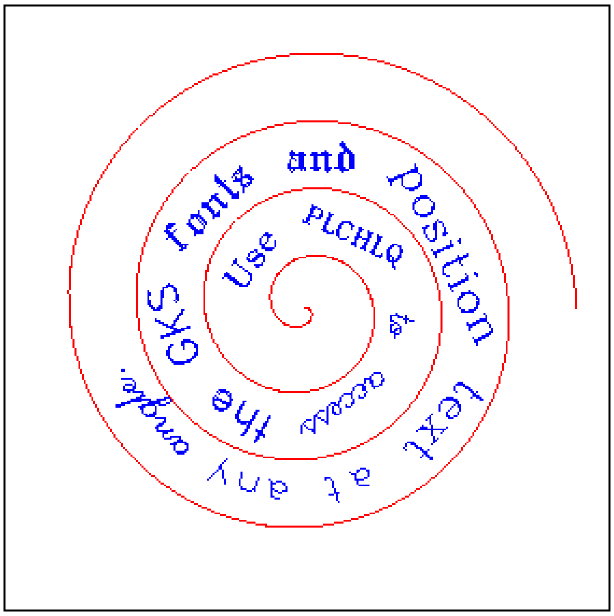
Line 13 sets the amount of white space between characters. The argument is a fraction of the character height, so in this example there are two white spaces between every character in the string "Chile."

Line 14 draws the string "Chile" with the attributes set above.

Drawing text with the PLCHLQ routine

The Plotchar routine PLCHLQ (or PCLOQU) is a higher level NCAR Graphics routine that draws characters of "low quality" by calling the GKS character-drawing routines. This module demonstrates how to draw text with PLCHLQ and how to select fonts, character size, and text color.

PLCHLQ text



Code segment from fpcloqu.f

```
      CALL GSTXCI (4)
      CALL GSTXFP (-13, 2)
      CALL PLCHLQ (.41, .58, 'Use', .04, 55., 0.)
      CALL PLCHLQ (.58, .62, 'PLCHLQ', .03, -25., 0.)
      CALL GSTXFP (-6, 2)
      CALL PLCHLQ (.66, .47, 'to', .04, -90., 0.)
      CALL PLCHLQ (.57, .33, 'access', .05, -145., 0.)
```

Synopsis

```
CALL PLCHLQ (XPOS, YPOS, CHRS, SIZE, ANGD, CNTR)
```

Arguments

XPOS, YPOS

Real, Input---The X and Y user coordinates of the characters to be plotted.

CHRS

Character, Input---A string consisting of the characters to be plotted. The string may include any of the following 95 characters: A-Z a-z 0-9 ! " # $ % & ’ ( ) * + - / \\ : ; < = > ? @ ‘ [ ] ^ _ { | } ~ , . blank
The function codes, like those described later in the "Drawing text with the PLCHHQ routine" module, may not be used.
The number of characters in CHRS is taken to be LEN(CHRS). Use the FORTRAN 77 substring notation CHRS(*m*:*n*) to draw only characters *m* through *n* (where *m* and *n* are integer expressions) from the CHRS character string.

SIZE

Real, Input---Defines a multiplier for character size based upon the following values of SIZE:

<=0.
The absolute value specifies the character size as a multiple of a default digitized size on a 1024x1024 grid on which blanks are 16 units wide.

0.<SIZE<1.
Specifies the desired width of a blank as a fraction of the distance across the plotter frame.

>=1.
Specifies the desired width of a blank in plotter coordinates, as defined by default or by a call to the SPPS routine SETI.

ANGD

Real, Input---The angle, in degrees counterclockwise from the positive X axis, at which the character string is plotted.

CNTR

Real, Input---Specifies the centering option based on the following values of CNTR:

-1
The point (XPOS, YPOS) is the center of the left edge of the first character.

1.
The point (XPOS, YPOS) is the center of the right edge of the last character.

0.
The point (XPOS, YPOS) is the midpoint of the line joining the center of the left edge of the first character to the center of the right edge of the last character.

Discussion

Drawing text with PLCHLQ (or PCLOQU, which has the same arguments) results in a smaller metafile than if PLCHMQ or PLCHHQ is used.

An idiosyncrasy of using PLCHLQ, PLCHMQ, and PLCHHQ is that PLCHLQ uses calls to the GKS routine GTX to draw text, whereas PLCHMQ and PLCHHQ call the GKS routines GPL and GFA. The implication is that changing text characteristics such as its color requires you to use GSTXCI for PLCHLQ and GSPLCI, and GSFACI for PLCHMQ and PLCHHQ. A better way to change character color when using PLCHHQ is to use the parameters provided for color-setting.

Line 1 of the **fpcloqu.f** code segment sets the color of text by specifying an index (4) in the color table that was set up previously with the GKS routine GSCR.

Line 2 chooses a font (Hershey triplex Roman) and sets the stroke precision to "best."

Line 3 plots the character string "Use" at coordinates (.41, .58) with characters that correspond to a blank size that is 4 percent of the width of the plotter frame. The string is drawn at a 55-degree angle, and (.41, .58) is the center of the string.

Line 4 plots the character string "PLCHLQ". The arguments specify a new location, size, and angle.

Line 5 changes the font to Hershey simplex script.

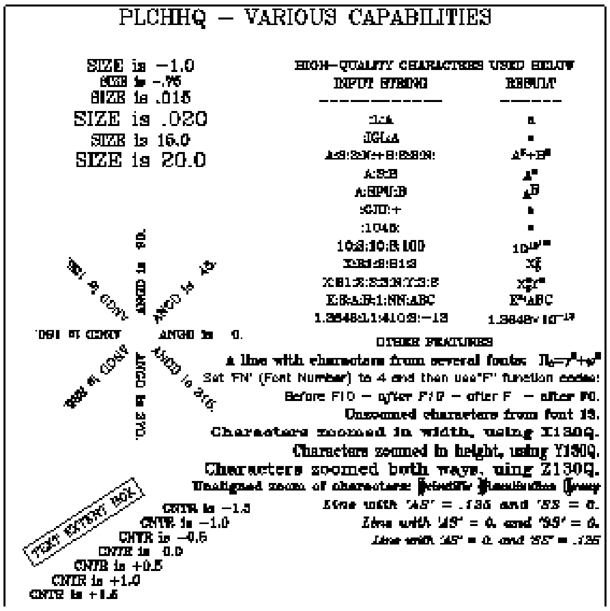
Lines 6 and 7 plot the strings "to" and "access".

Drawing text with the PLCHHQ routine

The Plotchar routine PLCHHQ (or PCHIQU) plots high-quality characters from font databases that contain Roman, Greek, Cyrillic, and Gothic characters; math, music, and weather symbols; and some filled fonts, to name a few. PLCHHQ also supports subscripting, superscripting, shadow, and outlined characters.

This module demonstrates how to draw text with PLCHHQ and set text characteristics such as font type, size, superscripting, subscripting, and angle.

PLCHHQ features



Code segment from fpchiqu.f

```
1   CALL PLCHHQ (.225, .453, '   ANGD is  45.', .012, 45., -1.)
```

```
  CALL PLCHHQ (.875, .750, 'A:S:2:N:+B:S:2:N:', .012, 0., 0.)
  CALL PCSETC ('FC - FUNCTION CODE CHARACTER', '/')
  CALL PLCHHQ (.990, .410, '/F13/A line with characters from several f
 +onts:  /F8/P/BF13/0/N/=/F5/g/SF13/2/N/+/F5/j/SF13/2/N/',
 +.012, 0., 1.)
```

Synopsis

```
      CALL PLCHHQ (XPOS, YPOS, CHRS, SIZE, ANGD, CNTR)
```

Arguments

XPOS, YPOS

Real, Input---The X and Y world coordinates of the characters to be plotted.

CHRS

Character, Input---A string consisting of the characters to be plotted. CHRS may contain embedded function codes that are used to reset parameters affecting the selection of characters from the database and the way in which those characters are plotted.
The characters in CHRS are scanned from left to right and plotted. However, if a function-code signal character (a colon by default; it is defined by the parameter FC) is encountered, the following characters are interpreted as function codes until the next function-code signal character appears in CHRS. Thus if CHRS is the string 'ABC:L:DEF', then "A", "B", and "C" are treated as characters to be plotted, "L" is treated as a function code, and "D", "E", and "F" are treated as characters to be plotted. In this example, the function code "L" requests lowercase. Therefore, the string "ABCdef" would be drawn.
The function codes can take on the following values:

F, F0, F*n*

Where *n* is a font number, selects the character database. By default, the database used is the one specified by the parameter FN. The default for FN is 0; this selects the PWRITX database. F0 returns to the PWRITX database, and F returns to the database specified by the text parameter FN.

R

Selects the Roman font when using the PWRITX database.

G

Selects the Greek font when using the PWRITX database.

P

Selects characters of principal size when using the PWRITX database.

I

Selects characters of indexical size when using the PWRITX database.

K

Selects characters of cartographic size when using the PWRITX database.

U

Selects uppercase characters when using the PWRITX database.

L

Selects lowercase characters. This affects characters from both the PWRITX and fontcap databases.

A

Draws characters across the frame.

D
Draws characters down the frame.

B
Selects a subscript level.

S
Selects a superscript level.

E
Ends subscripting or superscripting.

N
Causes a return to normal level.

H*n* or H*n*Q
Where *n* is an integer, moves horizontally *n* digitized units or *n* blank widths if using H*n*Q.

V*n* or V*n*Q
Where *n* is an integer, causes a vertical move in the same manner as H*n* and H*n*Q.

C
Causes a return to the initial position offset by one blank height. For example, the string 'ABC:C:DEF' causes the characters "DEF" to be written directly below the characters "ABC".

X*n*
Where *n* is an integer, causes character width to zoom to *n* percent of normal.

Y*n*
Where *n* is an integer, causes characters to be made *n*/100 times as high.

Z*n*
Where *n* is an integer, causes characters to be made *n*/100 times as wide and high.

CHRS
A complete description of the function codes, their meanings, and their options appears in the NCAR Graphics Programmer Document *Plotchar, a Character-Plotting Package.*

SIZE
Real, Input---Defines a multiplier for character size based on the following values of SIZE:

<=0.
The absolute value specifies the size as a multiple of a default digitized size on a 1024x1024 grid; blanks on this grid are 16 units wide.

0.<SIZE<1.
Specifies the desired width of a blank as a fraction of the distance across the plotter frame.

>=1
Specifies the desired width of a blank in plotter coordinates, as defined by the default or by a call to the SPPS routine SETI.

ANGD
Real, Input---The angle, in degrees counterclockwise from the positive X axis, at which the character string is plotted. The text extent or distance (in the fractional system) from the point (XPOS, YPOS) to the left, right, top, and bottom edge of a box enclosing the string can be

computed by setting ANGD to 360. and the parameter TE to a nonzero value. In this case, PLCHHQ does not draw any characters; instead it stores the text extent information in the parameters DL, DR, DT, and DB. This provides a way to access this information without drawing the string.

CNTR

Real, Input---Specifies the centering option based on the following values of CNTR:

-1
: The point (XPOS, YPOS) is the center of the left edge of the first character.

1.
: The point (XPOS, YPOS) is the center of the right edge of the last character.

0.
: The point (XPOS, YPOS) is the midpoint of the line joining the center of the left edge of the first character to the center of the right edge of the last character.

s
: Where *s* is any real number, the point (XPOS, YPOS) is obtained by linear interpolation along the line joining the points associated with CNTR values of -1 and +1.

Discussion

PLCHHQ (or PCHIQU, which has the same arguments) accesses a wider variety of fonts and controls a larger range of text characteristics than PLCHLQ or PLCHMQ. A disadvantage of using PLCHHQ, however, is that the resulting metafile tends to be larger.

Line 1 of the **fpchiqu.f** code segment uses the default font (principal Roman) and character database (PWRITX) to draw the text "ANGD is 45.". The text is drawn at a 45-degree angle because of the ANGD argument.

Line 2 uses function codes to draw the equation "A2+B2". The "A" is written with the default font. The ":S:" turns on one level of superscripting. The "2" is written as a superscript. ":N:" returns to the normal level. "+ B" is drawn normally. ":S:" turns on superscripting again. "2" is written as a superscript. Finally, ":N:" returns to the normal level.

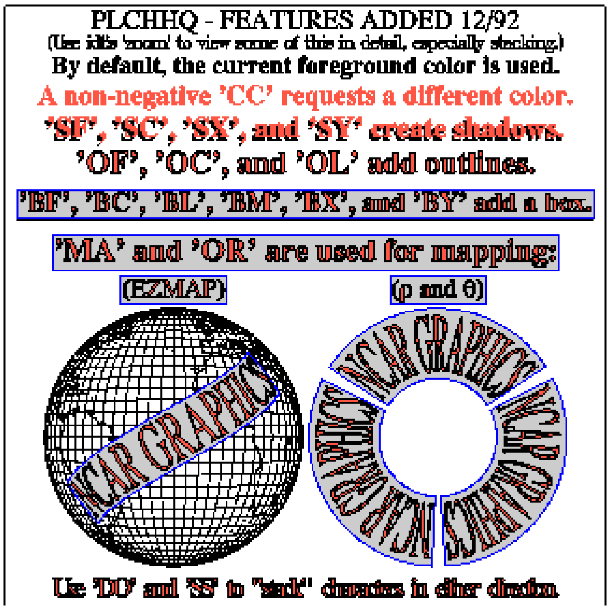
Line 3 switches the function code character from a colon to a slash. Line 4 draws a text string that includes several fonts, subscripting, and superscripting.

The "/F13/" in the CHRS argument selects font 13 from the fontcap database. The characters "A line with ... fonts" is drawn in font 13. The "/F8/" selects font 8. The "P" draws the P from font 8, which is the pi symbol. "/BF13/0" turns on subscripting using font 13 and draws the character zero. The "/N/=" returns drawing to the normal level and draws an equal sign. "/F5/g" selects font 5 and draws the gamma character. The "/SF13/2" turns on superscripting, selects font 13 and then draws "2" as a superscript. The "/N/+" returns drawing to a normal level and draws the plus sign. "/F5/j" selects font 5 and draws a "j." The "/SF13/2" turns on superscripting, selects font 13 and draws a superscripted 2. Finally, the "/N/" returns drawing to the normal level.

Setting color and other PLCHHQ text characteristics

Many text characteristics are adjustable when using the PLCHHQ routine. This module demonstrates how to set characteristics like color, shadows, and character outlines.

Setting PLCHHQ text characteristics



Code segment from fpchiqu.f

```
 CALL PCSETI ('CC - CHARACTER COLOR', 4)
 CALL PLCHHQ (.5, .850, ':F26:A non-negative ''CC'' requests a differ
+ent color.', .026, 0., 0.)
 CALL PCSETI ('SF - SHADOW FLAG', 1)
 CALL PCSETR ('SX - SHADOW OFFSET IN X', -.15)
 CALL PCSETR ('SY - SHADOW OFFSET IN Y', -.15)
 CALL PCSETI ('SC - SHADOW COLOR', 1)
 CALL PLCHHQ (.5, .796, ':F26:''SF'', ''SC'', ''SX'', and ''SY'' crea
+te shadows.', .028, 0., 0.)
 CALL PCSETI ('SF - SHADOW FLAG', 0)
 CALL PCSETI ('OF - OUTLINE FLAG', 1)
 CALL PCSETI ('OC - OUTLINE COLOR', 1)
 CALL PCSETI ('OL - OUTLINE LINE WIDTH', 1)
 CALL PLCHHQ (.5, .738, ':F26:''OF'', ''OC'', and ''OL'' add outlines
+.', .030, 0., 0.)
```

Synopsis

```
CALL PCSETI ('CC', icol)
```

Arguments

CC

Integer, Array---An array of color indices. Element 0 specifies the principal character color, and elements 1 through 16 specify special colors to be used for certain characters. The value -1 means "unspecified"; a value greater than or equal to 0 is a particular color index to be used. The routine PCDLSC may be called to define a standard default set of colors for elements 1 through 16. To specify element 0, use the parameter name CC(0); to access element 1, use CC(1); etc. The name CC by itself refers to element 0.

Discussion

When setting the character color, use CC(0), which is the name of the first element of the parameter for character color. CC is an array of 17 elements (0-16). The first element (CC(0)) specifies the color of the principal character. Elements 1 through 16 specify special colors to be used for certain characters. Specifying CC is equivalent to specifying CC(0).

Most text characteristics are controlled using the parameter access routines discussed at the beginning of this section. In particular, text color is controlled by using PCSETI along with the parameter name CC and a color table index that specifies a red-green-blue color value.

Line 1 of the **fpchiqu.f** code segment sets the character color by specifying the color index 4 in the color table created previously with the GKS routine GSCR.

Lines 2 and 3 draw text using font 26.

Line 4 turns the shadow flag on. A nonzero value means that characters will be drawn with shadows.

Lines 5 and 6 set the X and Y shadow offsets, respectively. The offset is stated as a fraction of the principal character height in the direction of ANGD for X and ANGD+90 for Y.

Line 7 sets the shadow color by specifying the color table index 1.

Lines 8 and 9 draw text using font 26 and shadows.

Line 10 turns off shadows.

Line 11 turns on character outlines.

Line 12 sets the outline color by specifying the color table index 1.

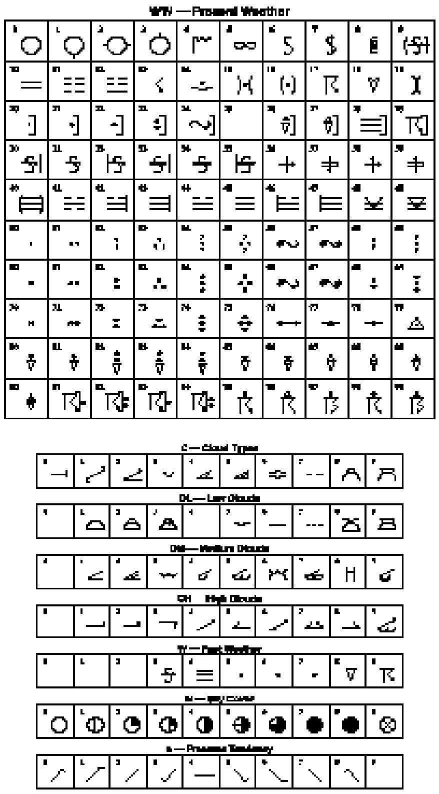
Line 13 sets the outline width. The value is specified as a fraction of "normal."

Lines 14 and 15 draw text with font 26 and outlines.

Drawing weather symbols with the NGWSYM routine

The Ngmisc routine NGWSYM draws a symbol from the standard WMO/NOAA meteorological fonts by reference to the font name and symbol number within that font. This module demonstrates how to draw weather symbols using the NGWSYM routine.

NGWSYM symbols



Code segment from fngwsym.f

```
      DO 30, J=1, NR
         DO 40, I=1, NC
            NUM = (NR-J) * NC + I - 1
            CALL NGWSYM (FONT, NUM, XC(I), YC(J), .43 * YINC, 1, 0)
 40      CONTINUE
 30   CONTINUE
```

Synopsis

```
      CALL NGWSYM (FTYPE, NUM, X, Y, SIZE, ICOLOR, IALT)
```

Arguments

FTYPE

Character, Input---Specifies the desired weather font. Possible values are:

WW
: Present weather.

C
: Cloud types.

CL
: Low clouds.

CM

Medium clouds.

CH

High clouds.

W

Past weather.

N

Sky cover.

a

Pressure tendency.

NUM

Integer, Input---Specifies the number of the desired symbol within the FTYPE font.

X, Y

Real, Input---The X and Y world coordinates of the symbol position.

SIZE

Real, Input---Specifies the height of the symbol in world coordinates.

ICOLOR

Integer, Input---Specifies a GKS color index for the symbol color.

IALT

Integer, Input---Indicates whether an alternate representation for the symbol is to be used. If IALT is 1, then an alternate symbol is used. This only applies to a few symbols (such as 7, 93, 94, 95, 97 in the WW font and 3 in the W font).

Discussion

The NGWSYM routine is an interface to the Plotchar utility for plotting weather symbols from fonts 36 and 37. NGWSYM looks up the appropriate symbol in either font 36 or 37 of Plotchar and uses Plotchar to draw the symbol.

The **fngwsym** example demonstrates how to use the NGWSYM routine to draw a weather symbol chart.

Lines 1 and 2 of the **fngwsym.f** code segment enter two nested loops for drawing a table of symbols. NR specifies the number of rows, and NC specifies the number of columns in the chart.

Line 3 assigns a symbol number to the NUM variable based on the current indices in the nested loops.

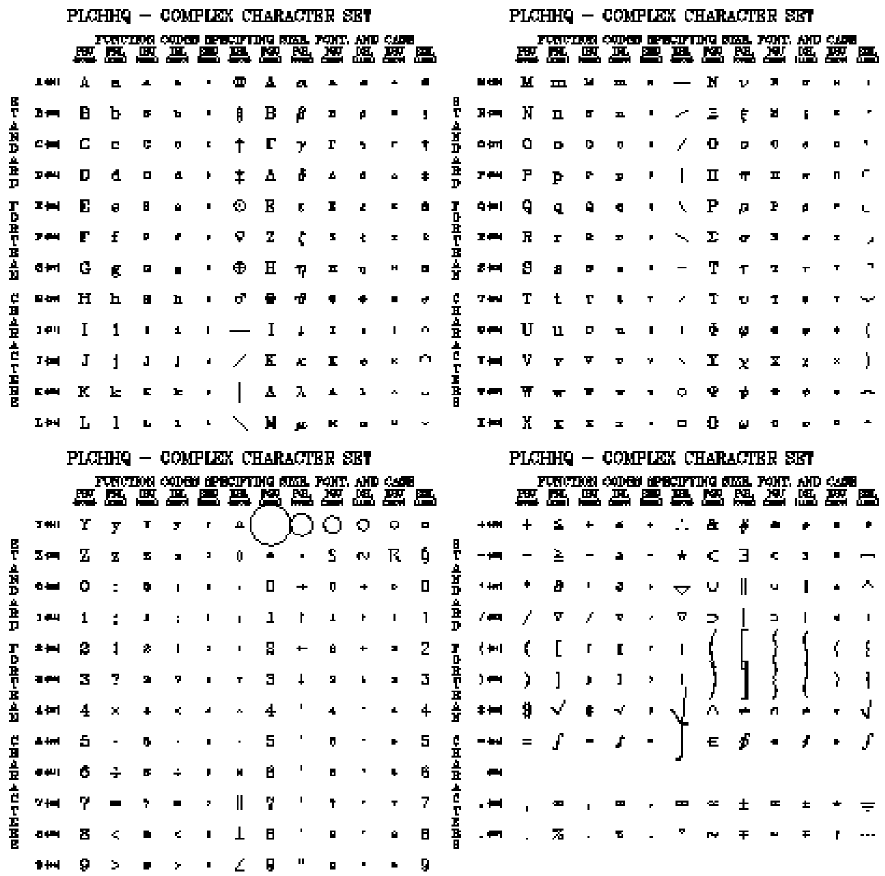
Line 4 draws symbol number NUM from the font specified by FONT. FONT is a character variable that contains one of the weather font character codes described previously under "FTYPE" in the argument section. The XC and YC arrays specify the position of the symbol. The fifth argument specifies a symbol size. The sixth argument, "1", specifies the GKS color index to use to color the symbol. The color table was set up prior to entering this code segment using the GKS routine GSCR. The final argument, "0", specifies not to use the alternate symbol representation.

Lines 5 and 6 terminate the nested loops.

Complex character set

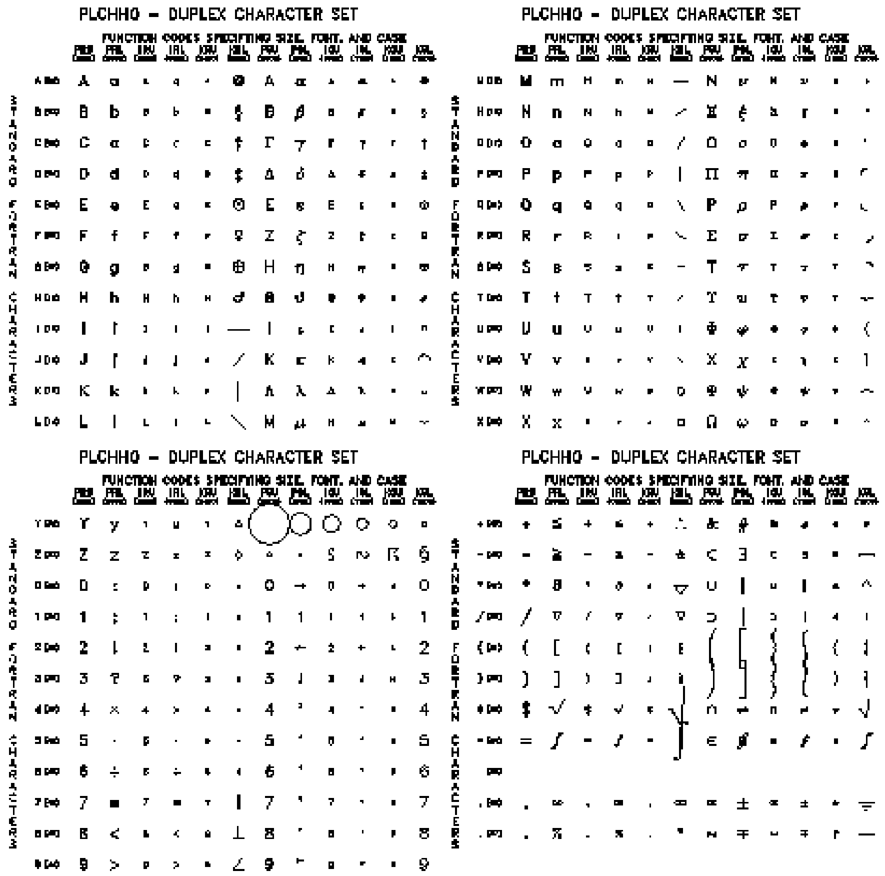
By default, PLCHHQ draws characters from a subset of the PWRITX database called the "complex" character set; it contains 564 characters. These characters are divided into 12 groups of 47 characters specified by font (Roman or Greek), size (principal, indexical, or cartographic), and case (upper or lower).

This character set is selected by first setting the parameter CD to 0 (the default) and then selecting the PWRITX database by either using the "F" function codes as described in the PLCHHQ module or setting the parameter FN to 0 (the default).



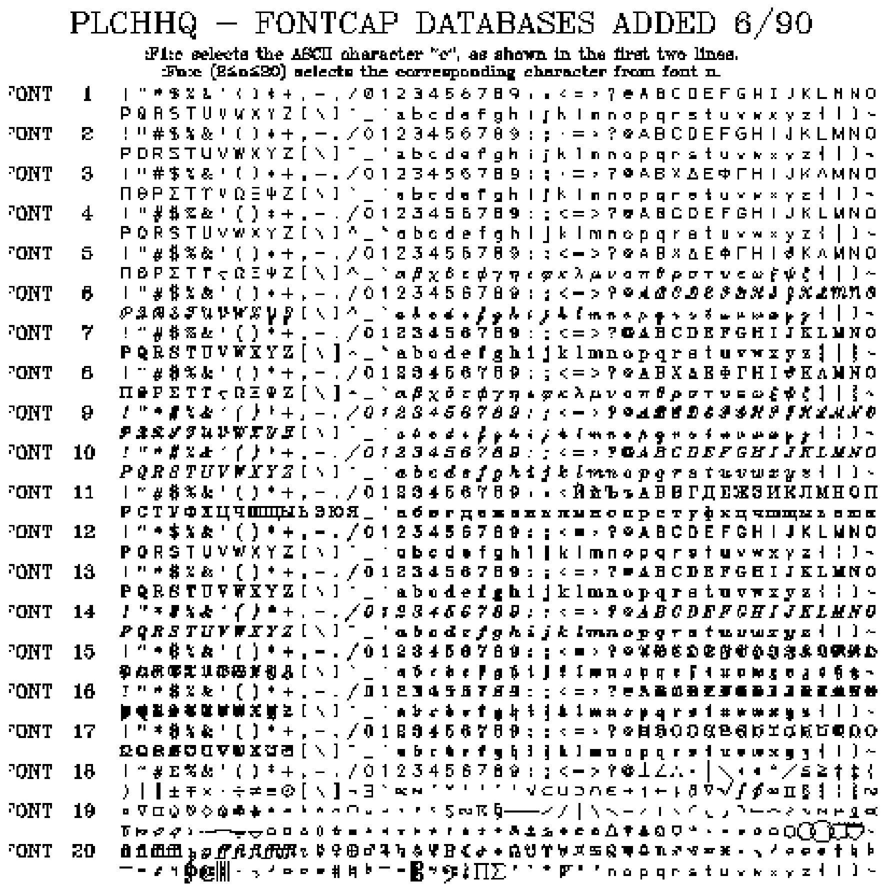
Duplex character set

When using the Plotchar routine PLCHHQ (or PCHIQU), the duplex character set is an alternative to using the complex character set. The duplex set has characters that are simpler than the complex set (no serifs, fewer curves, and so on.), but are still high quality. Only the alphanumeric characters differ; all other characters are the same. To access the duplex characters, reset the parameter CD to 1 and then select the PWRITX database by either using the "F" function codes as described in the PLCHHQ module or setting the parameter FN to 0 (the default).



Fontcap databases 1-20

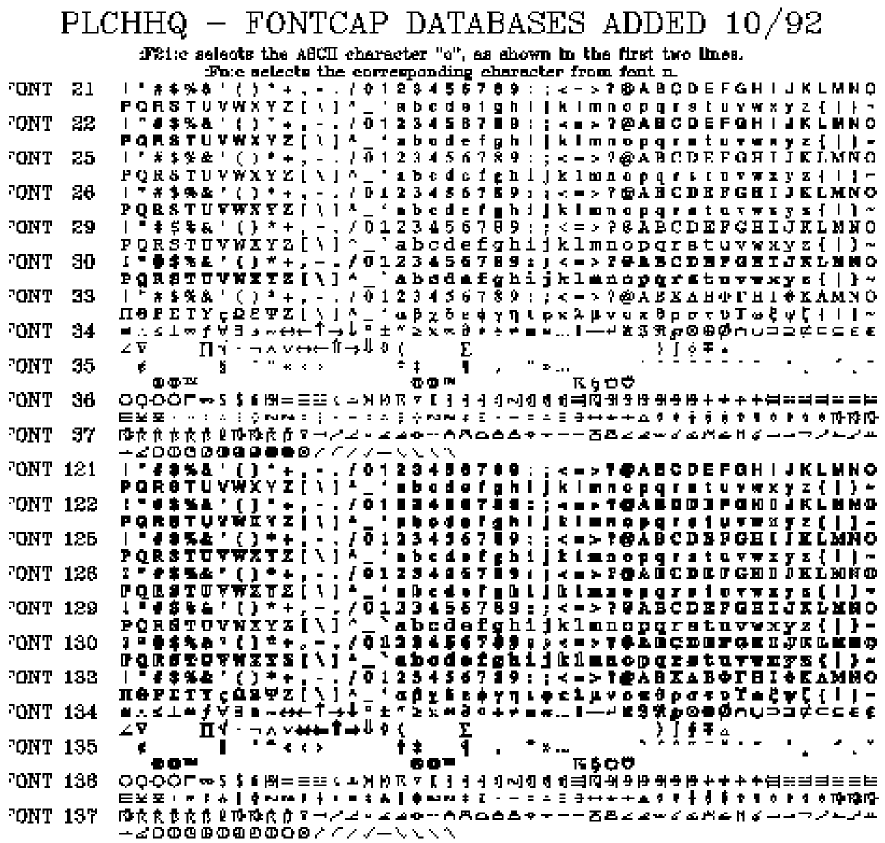
PLCHHQ can also draw characters from databases other than the PWRITX database. The "other" databases are defined by "fontcaps." You select the fontcap databases by setting the parameter FN to *n*, which corresponds to fontcap *n*. Fontcap databases can also be selected by using the "F" function code described in the PLCHHQ module. Listed below are fontcap databases 1 through 20. These are the same fonts used with the GKS routine GTX.



Fontcap databases 21-137

Fontcap databases 21 through 137 appear below (although there are gaps in the font numbers, the databases that appear here and in the preceding modules comprise the entire set of available fonts). You select these databases in the same manner that you select fontcap databases 1 through 20. Database selection is described in the preceding module.

These fontcaps can only accessed by the Plotchar routine PLCHHQ (or PCHIQU).



Chapter 12: Portable raster images (cell arrays)

Most modern graphics display devices use raster image technology. This means that the physical display surface is made up of a two-dimensional array of pixels, each of which can receive a color value. A displayed picture is nothing more than the aggregate setting and display of all physical pixel color values.

The GKS graphics standard (see the *User's Guide for NCAR GKS-0A Graphics* for details on GKS) abstracts the concept of a raster image in its cell array function. This function can be invoked, alongside any other NCAR Graphics function, to draw a two-dimensional array of colored rectangles. This array may be of arbitrary size.

Cell array

Representation of the Mandelbrot set where the colors indicate rate of convergence

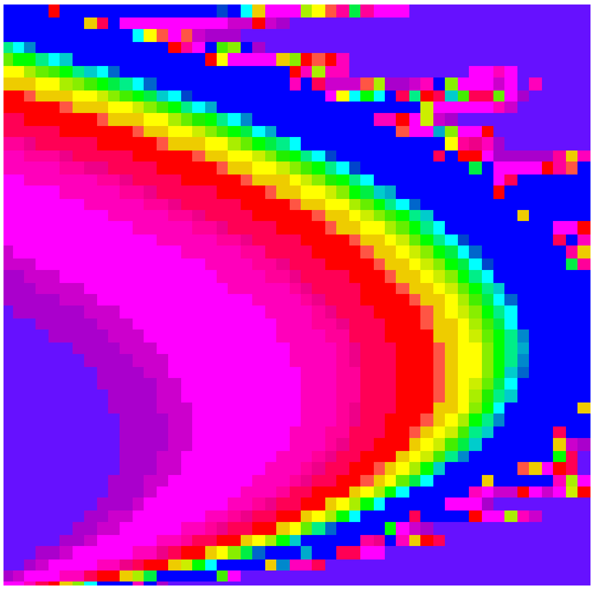


Table of cell array user entry points

There is only a single entry point discussed in this chapter.

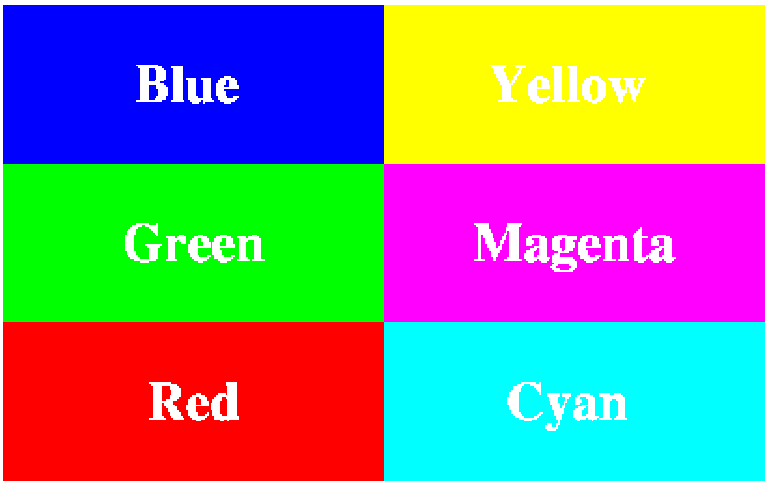
GCA

Takes two corner points defining a rectangle and a number of divisions in the X and Y directions that divide the rectangle into subrectangles, or cells. A color index is specified for each cell, and the resultant cells are drawn in the colors specified.

Creating cell arrays

This module describes how to make calls to the GKS cell array function. This function can be called alongside any other NCAR Graphics function. We will present an example that makes a simple cell array call and uses Plotchar to label the cells.

Simple cell array



Code segment from fcell0.f

```
        CALL GSCR(1,0,1.,1.,1.)
        CALL GSCR(1,1,1.,1.,1.)
        CALL GSCR(1,2,1.,0.,0.)
        CALL GSCR(1,3,0.,1.,0.)
        CALL GSCR(1,4,0.,0.,1.)
        CALL GSCR(1,5,0.,1.,1.)
        CALL GSCR(1,6,1.,0.,1.)
        CALL GSCR(1,7,1.,1.,0.)
        DO 10 I=1,IDX
             DO 20 J=1,IDY
                COLIA(I,J) = IDY*(I-1)+J+1
    20       CONTINUE
    10  CONTINUE
        CALL GCA(0.10, 0.25, 0.90, 0.75, IDX, IDY, 1, 1, IDX, IDY, COLIA)
```

Synopsis

```
CALL GCA (PX, PY, QX, QY, DIMX, DIMY, ISC, ISR, DX, DY, COLIA)
```

Arguments

PX

Real, Input---A world coordinate specifying an X value for a first corner point of a rectangle.

PY

Real, Input---A world coordinate specifying a Y value for a first corner point of a rectangle.

QX

Real, Input---A world coordinate specifying an X value for a second corner point of a rectangle.

QY

Real, Input---A world coordinate specifying a Y value for a second corner point of a rectangle.

DIMX

Integer, Input---The first dimension of the color index array COLIA (described below).

DIMY
Integer, Input---The second dimension of the color index array COLIA.

ISC
Integer, Input---The index of the starting column to be used from COLIA.

ISR
Integer, Input---The index of the starting row to be used from COLIA.

DX
Integer, Input---The number of columns to use from COLIA (this will equal the number of divisions along the X axis).

DY
Integer, Input---The number of rows to use from COLIA (this will equal the number of divisions along the Y axis).

COLIA
Integer array, Input---An array of color indices dimensioned DIMX by DIMY that is used to assign colors to the individual cells.

Discussion

The points (PX, PY) and (QX, QY) define diagonally opposite corner points of a rectangle. This rectangle is divided into DX by DY cells. The two-dimensional array COLIA specifies a color index for each cell. These color indices are used to color the individual cells.

Lines 1 through 8 of the **fcell0.f** code segment set up a color table that associates color values with color indices. See the "Color tables and color mapping systems" chapter in this document for details on setting up a color table. An unusual feature of this color table is that the foreground color (color index 1) and the background color (color index 0) are both defined to be white. This is because a white background was desired and the foreground color is used to write text on top of colors that are different from the background color.

Lines 9 through 13 define the color index array where IDX=2 and IDY=3. The six color indices from 2 through 7 are used. These indices are associated with the primary and secondary colors in the color table.

Line 14 calls GCA to produce the cell array. The black-and-white reproduction is less than optimal, so we have used Plotchar to label the cells to indicate what colors you would see if you plotted this picture on a color device.

Cell arrays provide for a way of displaying raster image data along with text, lines, markers, and filled areas. Some data are more suitable for display as a raster image. For example, you may have data from scientific equipment that is in the form of an array of color intensities. Or you may have a screen dump from a graphics output device that is in the form of a raster image. These arrays would have to be converted to a form suitable for using GCA, but once that is done you can integrate such data with the rest of NCAR graphics and display them on multiple output devices.

One disadvantage in using GCA is the case where the resolution of the cell array is not well matched with the resolution of the graphics output device. In this case you may see some "jaggies" or "staircasing" that are consequences of the algorithm that converts the cell array to the raster image on the screen. You could make the size of the cell array match the resolution of a desired output device, but that defeats the intended portability of GCA.

Chapter 13: Area filling with GKS and Softfill

NCAR Graphics provides two methods for filling areas. The simplest is to use GKS calls directly to fill an area. However, the Softfill utility gives you much more flexibility and many more fill options than GKS.

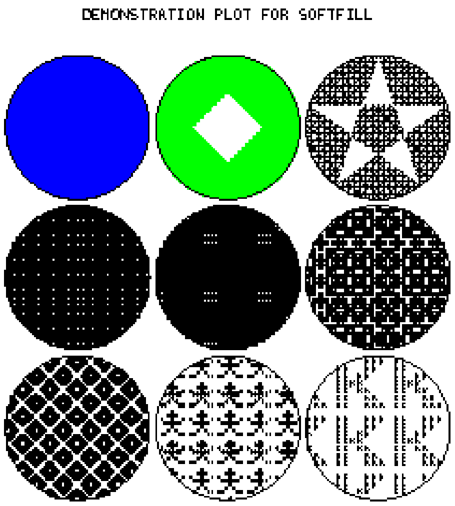


Table of user entry points for area filling

This module organizes the GKS area-filling routines and all Softfill routines according to their functions.

GKS fill routines

GSFAIS
Establishes the interior fill style to be used in subsequent calls to GFA.

GFA
Fills the specified polygon with the fill style set by the last call to GSFAIS.

Softfill fill routines

SFNORM
Fills a polygon defined in NDCs with evenly spaced parallel lines or with rectangular patterns of dots or characters.

SFWRLD
Fills a polygon defined in world coordinates with evenly spaced parallel lines or with rectangular patterns of dots or characters.

SFSGFA
Fills a polygon defined in world coordinates with solid color fill (if available on your graphics device); otherwise it uses a suitable pattern fill.

Softfill parameter routines

SFSETC
Sets character parameters.

SFSETI
Sets integer parameters.

SFSETR
Sets real parameters.

SFSETP
Sets the dot pattern parameter.

SFGETC
Gets character parameters.

SFGETI
Gets integer parameters.

SFGETR
Gets real parameters.

SFGETP
Retrieves the dot pattern parameter.

Table of Softfill parameters

For a more complete description of Softfill parameters, see the **softfill_params man** page or the Softfill programmer document.

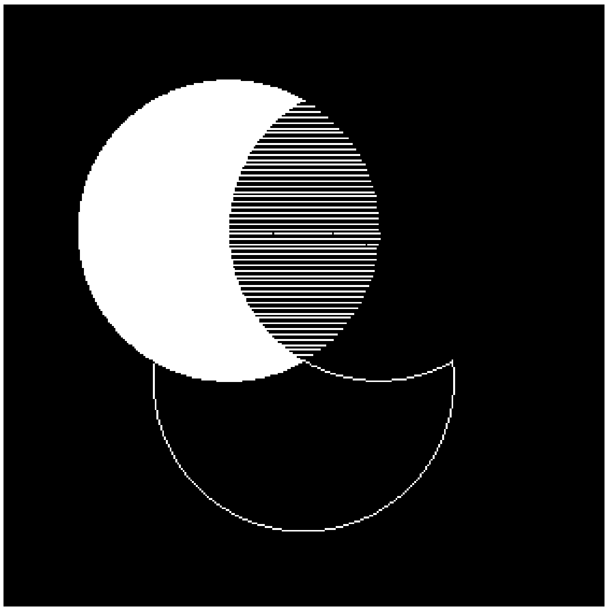
```
-----------------------------------------------------------
Parameter  Brief description          Fortran type
```

```
------------------------------------------------------------
AN        ANgle of fill lines          Real
CH        CHaracter selector           Integer or Character
DO        DOt fill selector            Integer
SP        SPacing between fill lines   Real
TY        TYpe of fill selector        Integer
LDP       Dot pattern                  8 x 8 Integer array
------------------------------------------------------------
```

Filling areas

This module describes the two calls you must make to fill an area using GKS calls. GSFAIS sets the style or type of fill to be used, and GFA fills an area defined in world coordinates.

Areas filled using GKS



Code segment from fcirc.f

```
1        CALL GSFAIS (MOD (ICOLOR, 4))
2        CALL GFA (NC-1, XC, YC)
```

Synopsis

```
      CALL GSFAIS (ISTYLE)
      CALL GFA (N, X, Y)
```

Arguments

ISTYLE

Integer, *Input*---The style of fill to be used in subsequent calls to GFA.

0
: Hollow fill. This is the default.

1
: Solid fill.

2
: Pattern fill: not implemented in NCAR Graphics GKS.

3
: Hatch fill.

N
: *Integer*, *Input*---The number of points in the polygon to be filled. N must be greater than 2.

X(N), Y(N)
: *Real arrays*, *Input*---The X and Y world coordinates of the polygon to be filled.

Discussion

Notice that "hollow fill" is the default GKS fill style. If you don't call GSFAIS before calling GFA, as shown in the **fcirc** example, your polygons or areas will not be filled---in other words, they will be filled with nothing. This is the most common mistake that users make when trying to fill areas using GKS.

Line 1 of the fcirc.f code segment sets the fill style to be one of the four styles, and in line 2 we fill the areas. Hollow fill is used for the bottom portion of the lower circle, solid fill is used for the left portion of the left circle, pattern fill is used as the right portion of the right circle, and is empty because NCAR Graphics does not support this option, and hatch fill is used in the intersection of the two upper circles.

Softfill parameters: What they do and how to use them

Softfill relies on parameters to control such things as fill patterns, distance between fill lines, and angles of crosshatching. There are eight routines to help you retrieve and set parameter values.

Synopsis

```
CALL SFSETC (PNAM, STRING)
CALL SFSETI (PNAM, IVAL)
CALL SFSETR (PNAM, RVAL)
CALL SFSETP (IARRAY)
CALL SFGETC (PNAM, STRING)
CALL SFGETI (PNAM, IVAL)
CALL SFGETR (PNAM, RVAL)
CALL SFGETP (IARRAY)
```

Routines

SFSETC
: Gives a character value to a parameter.

SFSETI
: Gives an integer value to a parameter.

SFSETR
Gives a real value to a parameter.

SFSETP
Sets the dot pattern array.

SFGETC
Retrieves character parameter information.

SFGETI
Retrieves integer parameter information.

SFGETR
Retrieves real parameter information.

SFGETP
Retrieves the dot pattern array.

Arguments

PNAM
Character, Input---A character expression that represents a parameter name. Softfill uses only the first two characters in the string. Many users include additional characters that describe the parameter to make their code easier to maintain.

IVAL
Integer, Input or Output---An integer value or integer variable.

RVAL
Real, Input or Output---A real value or real variable.

STRING
Character expression, Input or Output---A character string or a character variable.

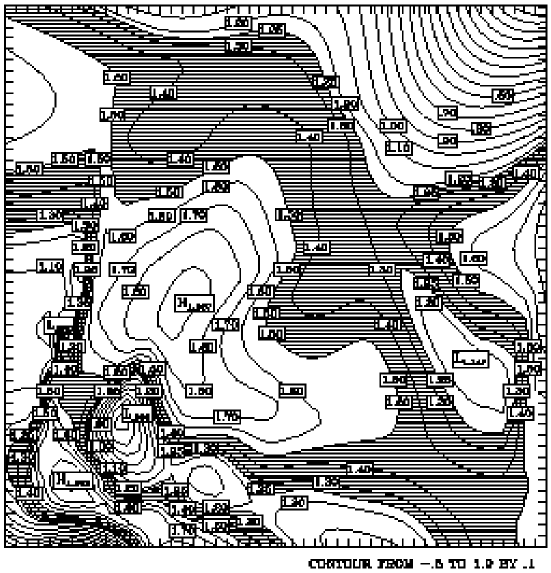
IARRAY
Integer array, Input or Output---An 8 x 8 integer array.

Note: To avoid a common user problem, be sure to double-check your "get" and "set" calls to ensure that you use the correct routine with the correct variable type. For example, do **not** use SFSETI to set a real parameter value.

Pattern-filling areas in NDCs

SFNORM is used to fill polygons or areas with evenly spaced parallel lines, rectangular patterns of dots, or with rectangular patterns of single characters. SFNORM should be used when the points defining the areas to be filled are in NDCs.

Shading a few contour levels



Code segment from ccpscam.f

```
      DO 10, I=1, NGRPS
         IF (IGRP(I) .EQ. 3) IAREA3 = IAREA(I)
  10  CONTINUE
      IF (IAREA3 .EQ. 1) THEN
         CALL SFSETR ('SP - SPACING BETWEEN FILL LINES', .006)
         CALL SFNORM (XWRK, YWRK, NWRK, RSCR, 5000, ISCR, 5000)
      ENDIF
```

Synopsis

```
      CALL SFNORM (XWRK, YWRK, NWRK, DST, NST, IND, NND)
```

Arguments

XWRK(NWRK), YWRK(NWRK)

Real arrays, Input---The X and Y coordinates of the points that define the boundary of the polygon to be filled. Values in these arrays must be in NDCs. You do not need to repeat point 1 at the end of the boundary to close the polygon.

NWRK

Integer, Input---The number of points defining the area to be filled. NWRK must have a value greater than 2.

DST(NST)

Real array, Workspace.

NST

Integer, Input---The length of the real workspace array. Make NST>=NWRK+NIM, where NWRK is the number of points defining the area to be filled, and NIM is the largest number of intersection points of any fill line with the boundary lines. To be sure DST is dimensioned large enough, you can use NIM=NWRK; in practice, NIM rarely needs to be that large.

IND(NND)

Integer array, Workspace.

NND

Integer, Input---The length of the integer workspace array. Make NND>=NWRK+2*NIM, where NWRK is the number of points defining the area to be filled, and NIM is the number of intersection points of any fill line with the boundary lines. To be sure IND is dimensioned large enough, you can use NIM=NWRK; in practice, NIM rarely needs to be that large.

Discussion

In practice, SFNORM is most frequently called by a user-supplied subroutine called by the Areas routine ARSCAM. Because ARSCAM passes the fill routine polygon coordinates in NDCs, SFNORM should be used to do pattern fill. The Areas and Conpack chapters of the *NCAR Graphics Contouring and Mapping Tutorial* provide several examples of how to write this fill routine; see module "Ar 3.5 Writing an area-fill routine" for one example.

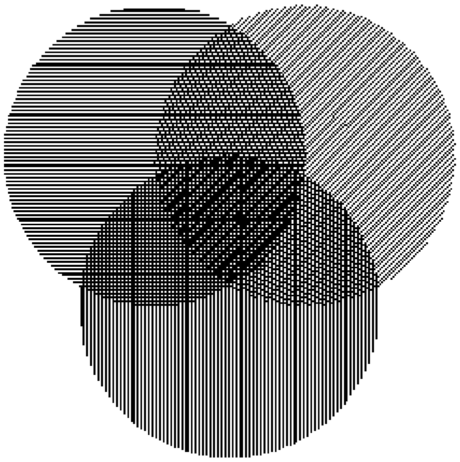
The code segment for the **ccpscam** example is from the routine SFILL, which is a user-supplied subroutine called by the Areas routine ARSCAM to fill certain contour levels. Lines 1 through 3 of the **ccpscam.f** code segment retrieve the area identifier for the contour level. Line 4 checks to see if the area identifier is 1, as we set the area identifier for the shaded area to be. If it is, then line 5 sets the Softfill spacing option, and line 6 calls SFNORM to fill the contour level.

Note: SFNORM does not draw the boundaries around an area. To draw the boundaries, use CURVED or the appropriate line-drawing routine for the utility. In the **ccpscam** example, CPCLDR is used to draw contour lines.

Pattern-filling areas in world coordinates

SFWRLD is used to fill polygons or areas with evenly spaced parallel lines, rectangular patterns of dots, or rectangular patterns of single characters. SFWRLD should be used when the points defining the areas to be filled are in world coordinates.

Filled areas in world coordinates



Code segment from fsfwrld.f

```
      DO 100, II=1, NPTS
         ANG = D2R * 3.6 * FLOAT (II-1)
         X = COS (ANG)
         Y = SIN (ANG)
         X1(II) = X - .5
         Y1(II) = Y + .5
 100  CONTINUE
      CALL SFSETR ('SP - SPACING BETWEEN FILL LINES', 0.006)
      CALL SFSETR ('AN - ANGLE OF FILL LINES', 0.)
      CALL SFWRLD (X1, Y1, NPTS, RWRK, LRWK, IWRK, LIWK)
      CALL SFSETR ('AN - ANGLE OF FILL LINES', 45.)
      CALL SFWRLD (X2, Y2, NPTS, RWRK, LRWK, IWRK, LIWK)
      CALL SFSETR ('AN - ANGLE OF FILL LINES', 90.)
      CALL SFWRLD (X3, Y3, NPTS, RWRK, LRWK, IWRK, LIWK)
```

Synopsis

```
      CALL SFWRLD (XWRK, YWRK, NWRK, DST, NST, IND, NND)
```

Arguments

XWRK(NWRK), YWRK(NWRK)

Real arrays, Input---The X and Y world coordinates of the points that define the boundary of the polygon to be filled. Values in these arrays must be in world coordinates. You do not need to repeat point 1 at the end of the boundary to close the polygon.

NWRK

Integer, Input---The number of points defining the area to be filled. NWRK must have a value greater than 2.

DST(NST)

Real array, Workspace.

NST

Integer, Input---The length of the real workspace array. Make NST>=NWRK+NIM, where NWRK is the number of points defining the area to be filled, and NIM is the largest number of intersection points of any fill line with the boundary lines. To be sure DST is dimensioned large enough, you can use NIM=NWRK; in practice, NIM rarely needs to be that large.

IND(NND)

Integer array, Workspace.

NND

Integer, Input---The length of the integer workspace array. Make NND>=NWRK+2*NIM, where NWRK is the number of points defining the area to be filled, and NIM is the number of intersection points of any fill line with the boundary lines. To be sure IND is dimensioned large enough, you can use NIM=NWRK; in practice NIM rarely needs to be that large.

Discussion

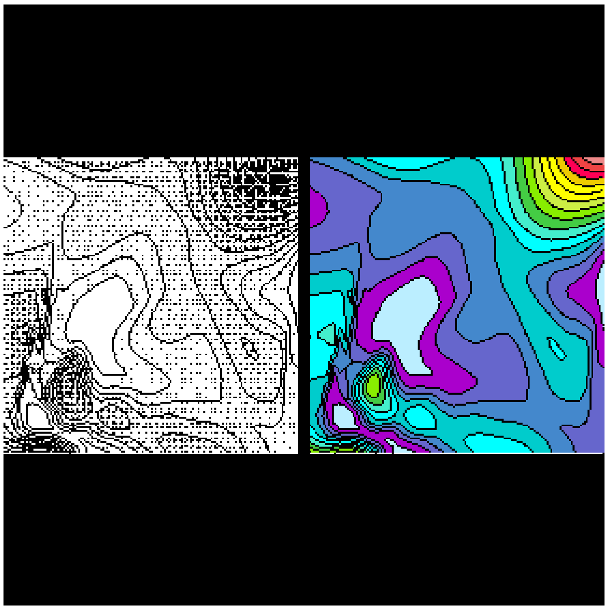
In practice, SFWRLD is most frequently called directly to shade a region for which the user already knows the boundary points. Lines 1 through 7 of the **fsfwrld.f** code segment establish points that define the circle containing horizontal fill lines. Line 10 fills this circle with horizontal lines. Lines 8 and 9 set parameters to control fill line spacing and the angle from horizontal used for drawing the fill lines. Lines 11 through 14 fill two other circles with lines at different angles.

Notice that the boundaries around the circles are not drawn. Neither SFWRLD nor SFNORM will draw the boundary lines around an area. If you need boundary lines, draw them using CURVED, or with an appropriate line-drawing routine for the specific utility you are currently using (such as CPCLDR to draw contour lines in Conpack).

Solid fills and pattern-fill substitutes

SFSGFA provides a way to use hardware-controlled solid color fill (if that capability is available on your graphics device); otherwise it uses a suitable software-controlled pattern fill. Using SFSGFA gives you the advantage of being able to change how areas are filled by modifying only a single variable.

Two fill options with SFSGFA



Code segment from fsfsgfa.f

```
      SUBROUTINE SFILL (XWRK, YWRK, NWRK, IAREA, IGRP, NGRPS)

      REAL XWRK(*), YWRK(*), ISCR(5000)
      INTEGER IAREA(*), IGRP(*), RSCR(5000)

      DO 10, I=1, NGRPS
         IF (IGRP(I) .EQ. 3) IAREA3 = IAREA(I)
 10   CONTINUE
      IF (IAREA3 .GE. 1) THEN
         CALL SFSGFA (XWRK, YWRK, NWRK, RSCR, 5000, ISCR, 5000, IAREA3 + 1)
      ENDIF
```

Synopsis

```
CALL SFSGFA (XRA, YRA, NRA, DST, NST, IND, NND, ICI)
```

Arguments

XRA(NRA), YRA(NRA)

Real arrays, Input and Output---Contain the X and Y coordinates of the points that define the boundary of the polygon to be filled. Use the world coordinate system for input values. After a call to SFSGFA, this array contains the X and Y coordinates transformed to NDCs. You do not need to repeat point 1 at the end of the boundary to close the polygon.

NRA

Integer, Input---The number of points defining the area to be filled. NRA must have a value greater than 2.

DST(NST)

Real array, Workspace---A real scratch array, dimensioned NST. This array is used only when

SFSGFA makes calls to SFWRLD and/or SFNORM to generate fill lines.

NST

Integer, Input---The dimension of DST. Make NST>=NRA+NIM, where NRA is the number of points defining the area to be filled, and NIM is the largest number of intersection points of any fill line with the boundary lines. To be sure DST is dimensioned large enough, you can use NIM=NRA; in practice, NIM rarely needs to be that large. For example, NIM=2 is sufficient for an octagon.

IND(NND)

Integer array, Workspace---This integer scratch array is used only when SFSGFA makes calls to SFWRLD and/or SFNORM to generate fill lines.

NND

Integer, Input---The dimension of IND. Make NND>=NRA+2*NIM, where NRA is the number of points defining the area to be filled, and NIM is the number of intersection points of any fill line with the boundary lines. To be sure IND is dimensioned large enough, you can use NIM=NRA; in practice, NIM rarely needs to be that large.

ICI

Integer, Input---ICI is a fill area index used to determine how an area is filled. The value of the internal parameter TY directly affects the function of ICI. See the next module for more information about TY.

TY=0

SFSGFA does color fill by calling GKS routine GFA. This is the default value.

ICI>=0 specifies the color index of the fill area.

ICI<0 specifies that the fill area color index is not to be set before calling GFA.

TY=1

SFSGFA fills the area with parallel lines spaced SP units apart, and at the angle AN, by calling SFWRLD.

ICI>=0 specifies the polyline color index.

ICI<0 specifies that the polyline color index is not to be set before calling SFWRLD.

Note: If CH and DO are set to select dot fill or character fill, the values of ICI will not affect the color of the dots or characters.

TY=2

SFSGFA calls SFWRLD to fill the area with parallel lines at the angle AN, and calls SFNORM to fill the area again with parallel lines at the angle AN+90; all lines are spaced SP units apart.

ICI>=0 specifies the polyline color index.

ICI<0 specifies that the polyline color index is not to be set before calling SFWRLD.

Note: If CH and DO are set to select dot fill or character fill, the values of ICI will not affect the color of the dots or characters.

TY = -4, -3, -2, -1
SFSGFA fills the area with line patterns by calling SFWRLD and/or SFNORM. The ABS(TY) determines the maximum number of fill-line angles used in a pattern. ICI determines the density of the lines drawn at each angle.

Discussion

In most cases, it is relatively easy to use SFSGFA. Since the Areas routine ARSCAM sets world and NDC coordinates to be the same before calling the processing or fill routine, SFSGFA can be used in place of SFNORM for most Areas applications. The argument list for SFSGFA is identical to those of SFNORM and SFWRLD, except that it has an extra argument, ICI. It is easiest to think of ICI as a fill index, which does the appropriate thing if the parameter TY is set.

In the **fsfsgfa** example, SFILL is called from the driver program by Areas routine ARSCAM. Line 10 of the **fsfsgfa.f** code segment calls SFGSFA to fill each area. The argument ICI in SFGSFA is set equal to the area identifier in SFILL, so that the fill option is related to the area being filled. In the next module, we examine the driver program to see how to get both black and white cross-hatching and color fill with the same call to SFGSFA.

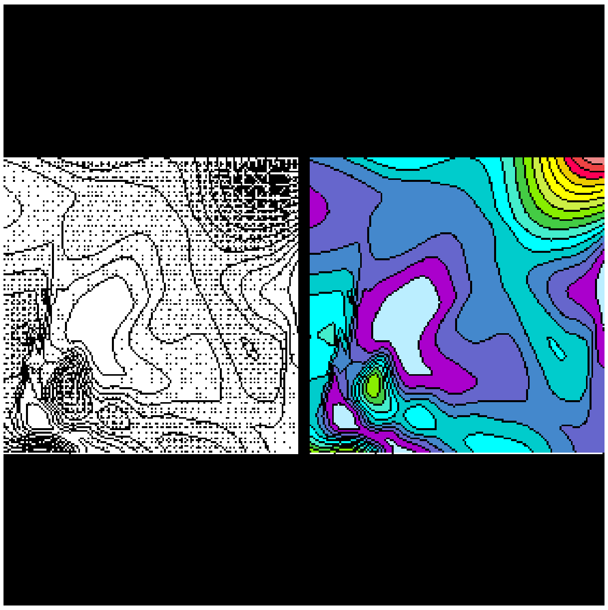
When TY<>0, ICI is used to select the density of lines in each direction. A zero or negative value of ICI selects a blank pattern. Positive values of ICI select patterns that increase in density as ICI increases from 0 to 15. The largest usable value of ICI is approximately 5*ABS(TY); beyond that, the pattern becomes essentially solid. For example, if TY=-4, then 20 is about the largest value of ICI that you can use and still see a pattern.

For each increase in ICI, fill lines are added at one of the usable angles. The first time lines are added at a given angle, they are added between existing lines so that the distance between lines at that angle is halved. An ICI value that is evenly divisible by the absolute value of TY yields a pattern that is evenly dense at all angles. For example, if TY=-2, the patterns associated with the first three values of ICI are formed as follows: ICI=1 uses lines at the angle AN, spaced 32*SP units apart; ICI=2 uses lines at the angles AN and AN+90, both spaced 32*SP units apart; ICI=3 uses lines at AN, spaced 32*SP units apart, and lines at AN+90, spaced 16*SP units apart.

Choosing how fill is done

The parameter TY controls exactly what type of fill you get when you call a Softfill routine.

Two fill options with SFSGFA



Code segment from fsfsgfa.f

```
      CALL CPRECT (ZREG, MREG, MREG, NREG, RWRK, LRWK, IWRK, LIWK)
      CALL SFSETI ('TY - TYPE OF FILL', -4)
      CALL CPBACK (ZREG, RWRK, IWRK)
      CALL ARINAM (MAP, LMAP)
      CALL CPCLAM (ZREG, RWRK, IWRK, MAP)
      CALL ARSCAM (MAP, XWRK, YWRK, NWRK, IAREA, IGRP, NOGRPS, SFILL)
      CALL CPCLDR (ZREG, RWRK, IWRK)
      CALL CPRECT (ZREG, MREG, MREG, NREG, RWRK, LRWK, IWRK, LIWK)
      CALL GSFAIS (1)
      CALL SFSETI ('TY - TYPE OF FILL', 0)
      CALL CPBACK (ZREG, RWRK, IWRK)
      CALL ARINAM (MAP, LMAP)
      CALL CPCLAM (ZREG, RWRK, IWRK, MAP)
      CALL ARSCAM (MAP, XWRK, YWRK, NWRK, IAREA, IGRP, NOGRPS, SFILL)
      CALL CPCLDR (ZREG, RWRK, IWRK)
```

Synopsis

```
CALL SFSETI ('TY', ity)
```

Arguments

TY

Integer---The type of fill to be done by the routine SFSGFA.

0

SFSGFA does color fill by calling GKS routine GFA. This is the default value.

ICI>=0 specifies the color index of the fill area.

ICI<0 specifies that the fill area color index is not to be set before calling GFA.

1

SFSGFA fills the area with parallel lines spaced SP units apart, and at the angle AN, by calling SFWRLD.

ICI>=0 specifies the polyline color index.

ICI<0 specifies that the polyline color index is not to be set before calling SFWRLD.

Note: If CH and DO are set to select dot fill or character fill, the values of ICI will not affect the color of the dots or characters.

2

SFSGFA calls SFWRLD to fill the area with parallel lines and calls SFNORM to fill the area again with parallel lines perpendicular to the first set.

ICI>=0 specifies the polyline color index.

ICI<0 specifies that the polyline color index is not to be set before calling SFWRLD.

Parameters AN, CH, DO, and SP can be used to further affect the nature of the fill. If CH and DO are set to select dot fill or character fill, the values of ICI will not affect the color of the dots or characters.

-4, -3, -2, -1

SFSGFA fills the area with line patterns by calling SFWRLD and/or SFNORM. The ABS(TY) determines the maximum number of fill-line angles used in a pattern. ICI determines the density of the lines drawn at each angle.

Discussion

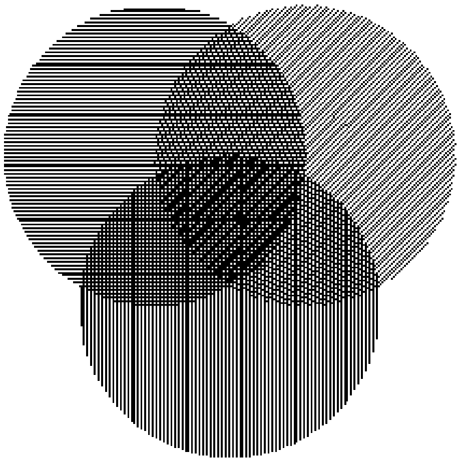
Lines 1 through 7 in the **fsfsgfa.f** code segment set up the area map and call Conpack to draw the contours. Line 2 chooses cross-hatching at four different angles as the type of fill for each contour level by setting TY=4. The lines are then drawn by SFGSFA in subroutine SFILL by the call to ARSCAM in line 6. The call to SFGSFA and the subroutine SFILL are discussed in the preceding module.

Lines 10 through 17 do exactly the same thing, however, this time, line 11 sets the GKS solid fill option, and line 12 sets solid color as the type of fill with TY=0. Line 16 fills the contour by calling ARSCAM, which in turn calls SFILL to do the filling. Notice that to change the type of fill, it is only necessary to change the value of the parameter TY.

Controlling fill lines

There are two parameters to control fill lines: AN controls the angles at which lines are drawn, and SP controls the spacing between lines.

Changing fill line angle and spacing



Code segment from fsfwrld.f

```
1        DO 100, II = 1, NPTS
2        ANG = D2R * 3.6 * FLOAT (II - 1)
3           X = COS (ANG)
4           Y = SIN (ANG)
5           X1(II) = X - .5
6           Y1(II) = Y + .5
7   100  CONTINUE
8        CALL SFSETR ('SP - SPACING BETWEEN FILL LINES', 0.006)
9        CALL SFSETR ('AN - ANGLE OF FILL LINES', 0.)
10       CALL SFWRLD (X1, Y1, NPTS, RWRK, LRWK, IWRK, LIWK)
11       CALL SFSETR ('AN - ANGLE OF FILL LINES', 45.)
12       CALL SFWRLD (X2, Y2, NPTS, RWRK, LRWK, IWRK, LIWK)
13       CALL SFSETR ('AN - ANGLE OF FILL LINES', 90.)
14       CALL SFWRLD (X3, Y3, NPTS, RWRK, LRWK, IWRK, LIWK)
```

Synopsis

```
      CALL SFSETR ('AN', angle)
      CALL SFSETR ('SP', space)
```

Arguments

AN

Real---The angle, in degrees, at which fill lines are to be drawn. By default, AN=0.

SP

Real---The spacing between each fill line, or if dot fill is selected, the distance between any pair of adjacent dots along a fill line. The value is given in NDCs. For this reason, 0.0<=SP<=1.0. By default, SP=0.00125. This produces nearly solid fill on many devices.

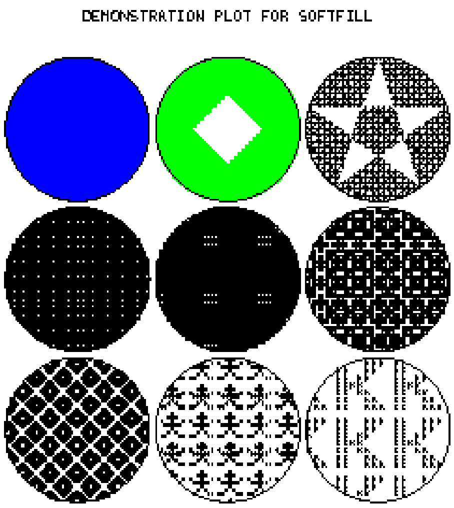
Discussion

Lines 1 through 7 of the **fsfwrld.f** code segment define the perimeters of the circles. Line 8 sets the spacing between fill lines to a reasonably large value, then line 9 specifies that the lines be drawn horizontally. Line 10 fills the first circle. Then using the same spacing, lines 11 and 12 fill the second circle with lines drawn at a 45 degree angle. Lines 13 and 14 fill the third circle with vertical fill lines.

Filling with dots or characters

Softfill uses two parameters and a pattern array to control two things: how areas are filled (dots, characters, or lines) and the pattern of the fill.

Sample dot, character, and line



Code segment from tsoftf.f

```
1        DATA ID4 / 0, 0, 0, 0, 0, 0, 0, 0,
2       +           0, 1, 1, 0, 0, 1, 1, 1,
3       +           0, 1, 1, 0, 0, 1, 1, 0,
4       +           0, 1, 1, 0, 1, 1, 0, 0,
5       +           0, 1, 1, 1, 1, 0, 0, 0,
6       +           0, 1, 1, 0, 1, 1, 0, 0,
7       +           0, 1, 1, 0, 0, 1, 1, 0,
8       +           0, 1, 1, 0, 0, 1, 1, 1/
9              CALL SFSETI ('DO - DOT-FILL FLAG', 1)
10 109         CALL GSCHH (.008)
11             CALL SFSETR ('SP - SPACING OF FILL LINES', .012)
12             CALL SFSETC ('CH - CHARACTER SPECIFIER', 'K')
13             CALL SFSETP (ID4)
14             CALL SFWRLD (XRA, YRA, 100, DST, 102, IND, 104)
```

Synopsis

```
      CALL SFSETI ('DO', ido)
      CALL SFSETC ('CH', char)
      CALL SFSETP (IARRAY)
```

Arguments

DO

Integer---The dot fill parameter determines whether dots, characters or lines are drawn.

0
: Solid lines are drawn. This is the default.

1
: Dots or characters are drawn.

CH

Integer or Character---The character parameter determines what is drawn at each dot.

0
: The fill character is a dot. This is the default.

***n*>0**
: The fill character is specified by CHAR(*n*), where *n* is the integer you specify.

***n*<0**
: The fill character is a GKS polymarker of type ABS(*n*), where *n* is the integer you specify:

-1 (dot)
-2 + (plus)
-3 * (asterisk)
-4 o (circle)
-5 x (cross)

[***a_character***]
: The fill character is the ASCII character that you specify.

IARRAY

Integer array---An 8x8 array in which the dot pattern is to be set or retrieved. Each dot is associated with a particular element of IARRAY. If that element is a 1, then the dot is drawn; if it is a 0, then the dot is not drawn. The association is done so that the pattern is replicated across the entire filled area. By default, IARRAY consists of all 1s.

Discussion

When dot fill is selected, each fill line is drawn using dots; if CH is nonzero, then each fill line is drawn using the specified character or symbol. Dots along a fill line are the same distance apart as the lines are from each other; these dots appear at the intersection points of a square grid.

If AN=0, then incrementing the first subscript of IARRAY corresponds to horizontal motion across the plot from left to right, and incrementing the second subscript of IARRAY corresponds to a vertical motion across the plot from top to bottom. This allows the contents of IARRAY to be declared in a DATA statement that creates a picture of the pattern, as shown in lines 1 through 8 of the **tsoftf.f** code segment.

By changing AN, it is possible to rotate the fill pattern. This dot pattern is used until another call to

SFSETP changes the contents of the array. By setting CH, you can change the elements used to define the dot pattern.

Line 10 sets the character size, and line 12 sets the character to "K," so that small printed Ks define a K pattern in the lower right circle in the plot. Line 13 sets the K pattern with a call to SFSETP, and line 14 draws the pattern.

Chapter 14: Drawing label bars

The Labelbar utility creates a specialized label consisting of a bar of filled areas that are labeled to correspond with areas filled by another utility. Label bars can be filled with black-and-white pattern fills, with color, or with both. Label bars are commonly used with contour plots and with other types of plots where area pattern fills or color are used to differentiate values in the plot.

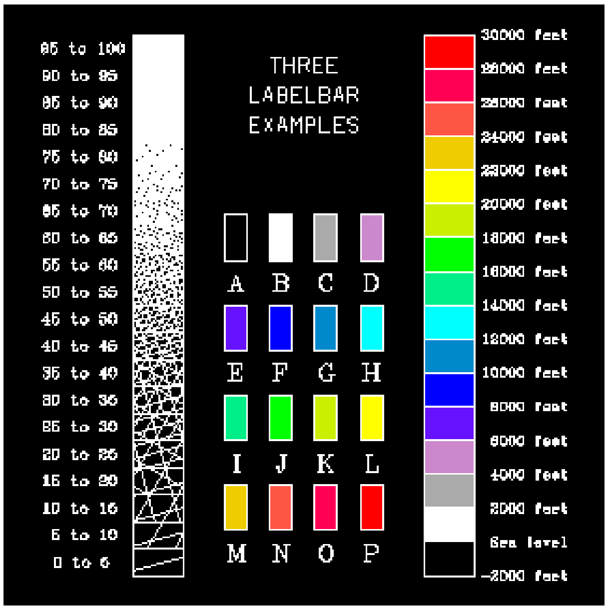


Table of Labelbar user entry points

This module organizes the Labelbar routines according to their functions.

Drawing routine

LBLBAR
    Draws a complete label bar.

Filling routine

LBFILL

Fills portions of the label bar. LBFILL is a user-modifiable routine.

Parameter setting and retrieving routines

LBGETI

Retrieves a Labelbar integer parameter value.

LBSETI

Sets a Labelbar integer parameter value.

LBGETR

Retrieves a Labelbar real parameter value.

LBSETR

Sets a Labelbar real parameter value.

Table of Labelbar parameters

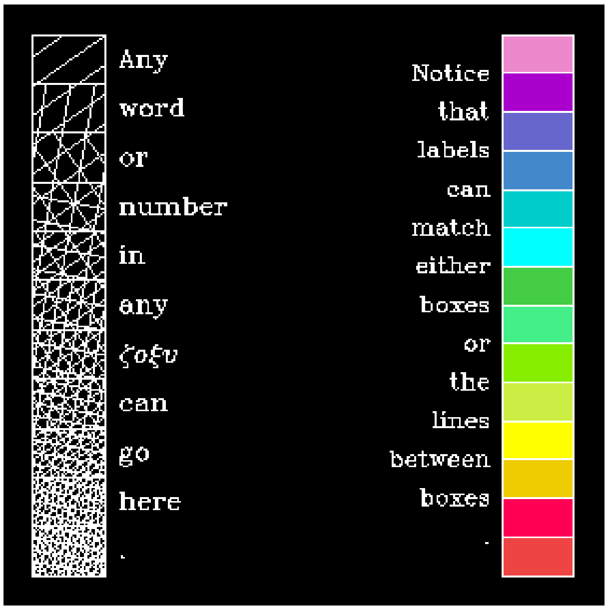
Currently, Labelbar supports six parameters. Three can be retrieved by the LBGETI routine or set with the LBSETI routine, and three can be retrieved by the LBGETR routine or set with the LBSETR routine. For a more complete description of Labelbar parameters, see the **labelbar_params man** page or the Labelbar programmer document.

```
--------------------------------------------
Parameter  Brief description     Fortran type
--------------------------------------------
CBL        Color of Box Lines    Integer
CFL        Color of Fill Lines   Integer
CLB        Color of LaBels       Integer
WBL        Width of Box Lines    Real
WFL        Width of Fill Lines   Real
WLB        Width of LaBel lines  Real
--------------------------------------------
```

Drawing the label bar

A single call to LBLBAR is all that is necessary to draw either horizontal or vertical label bars.

Simple label bars



Code segment from clblbr.f

```
      DATA LAB1 /'.','here','go','can',':G:font','any','in',
     + 'number','or','word','Any'/
      DATA LAB2 /'.','boxes','between','lines','the','or','boxes',
     + 'either','match','can','labels','that','Notice'/
      DATA IFILL1 /11,10,9,8,7,6,5,4,3,2,1/
      DATA IFILL2 /3,4,5,6,7,8,9,10,2,11,12,13,14,15/
      CALL GSFAIS(1)
      CALL SFSETR('AN',35.)
      CALL SFSETI('TY',-4)
      CALL LBLBAR(1,.05,.45,.05,.95,11,.3,1.,IFILL1,0,LAB1,11,1)
      CALL SFSETI('TY',0)
      CALL LBLBAR(1,.55,.95,.05,.95,14,.3,1.,IFILL2,1,LAB2,13,2)
```

Synopsis

```
      CALL LBLBAR (IHOV, XLEB, XREB, YBEB, YTEB, NBOX, WSFB, HSFB, LFIN,
     +             IFTP, LLBS, NLBS, LBAB)
```

Arguments

IHOV

Integer, Input---An integer specifying whether the bar should be oriented horizontally or vertically.

0

Draw the bar horizontally.

1

Draw the bar vertically.

XLEB, XREB, YBEB, YTEB

Real, Input---Four values in the range 0.0 to 1.0 that specify the left, right, bottom, and top coordinates, respectively, in NDCs in which the entire bar, including labels, is to fit.

NBOX

Integer, Input---A value specifying the number of boxes into which the bar should be divided. All boxes are equal in size. A horizontal bar is divided into boxes by vertical lines, and a vertical bar is divided into boxes by horizontal lines. The boxes in a horizontal bar are ordered from left to right, and the boxes in a vertical bar are ordered from bottom to top. This ordering determines the sequence in which the boxes are filled and labeled. Box outlines can be omitted by using a negative value of NBOX.

WSFB, HSFB

Real, Input---0.0<=WSFB, HSFB<=1.0. WSFB specifies a fraction of the width of the box to be filled, and HSFB specifies a fraction of the height of the box to be filled. For example, if you use WSFB=0.333 and HSFB=1.0, the width of the filled portion of each box will be one-third of the box's width, and fill will extend all the way from the bottom of the box to its top. The exact position of each filled portion within its box depends on how you choose to label the bar (see the description of the argument LBAB below); positioning is automatically handled for you by LBLBAR.

LFIN

Integer array, Input---The list of fill indices, one for each box in the bar.

IFTP=0

The elements of LFIN will be passed to the Softfill subroutine SFSGFA as values for the argument ICI.

IFTP<>0

The elements of LFIN will be passed to the subroutine LBFILL as values for the argument INDX.

By default, LFIN is used as an argument of the GKS subroutine GSFACI to specify a color-fill index. It is affected by how the IFTP parameter is set.

IFTP

Integer, Input---Specifies how the filled portion of the boxes should be filled:

0

LBLBAR calls the Softfill routine SFSGFA to fill the box. The value of the Softfill internal parameter TY determines whether filling is done by calls to the GKS routine GFA, by drawing closely spaced colored lines to simulate solid fill, or by drawing lines at varying angles and spacings to form patterns.

<>0

The Labelbar routine LBFILL is called to fill the boxes. By default, LBFILL calls GKS routine GFA to color-fill the box.

LLBS

Character array, Input---A list of labels for the bar. If the number of labels is equal to the number of boxes, LBLBAR associates the labels with the boxes themselves. If the number of labels is one less than the number of boxes, LBLBAR associates the labels with the divisions between boxes. If the number of labels is one more than the number of boxes, LBLBAR associates the first label with the beginning of the bar, the last label with the end of the bar, and the labels in between with the divisions between the boxes.

NLBS

Integer, Input---The number of labels in the array LLBS. NLBS must be such thatNBOX-1<=NLBS<=NBOX+1.

LBAB

Integer, Input---Specifies on which side or sides of the bar the labels are to be drawn.

0
The bar is unlabeled.

1
The labels are to be below a horizontal bar or to the right of a vertical bar.

2
The labels are to be above a horizontal bar or to the left of a vertical bar.

3
The labels are to be repeated on both sides of the bar.

Discussion

LBLBAR calls the Plotchar routine PLCHHQ to draw the labels and automatically scales labels so they do not overlap one another or run outside the area for the label bar. Labels at the ends of the bar may run outside this area, however. Any font can be used; this is demonstrated by using the PLCHHQ code to set Greek characters for the word "font" in line 1 of the **clblbar.f** code segment.

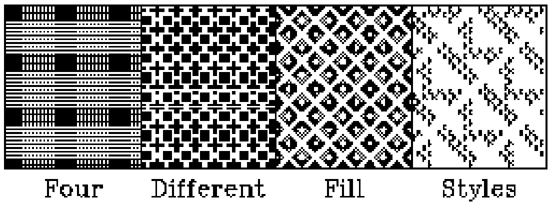
When IFTP is zero, as shown in line 10, LBLBAR does fill by calling the Softfill routine SFSGFA. The argument array LFIN, called IFILL1 in the **clblbar** example, contains the values that are passed to SFSGFA as values of its argument ICI. In line 5, values of IFILL1 go from highest to lowest, so that the most densely filled boxes will be at the bottom of the label bar. Before calling LBLBAR, line 8 sets the Softfill parameter to draw lines at an initial angle of 35 degrees, and line 9 sets the Softfill parameter to draw lots of crisscrossing lines. For a detailed explanation of the use of TY, see the module titled "Choosing how fill is done" in Chapter 13 "Area filling with GKS and Softfill."

By default, TY=0; this forces SFSGFA to do fill by calling the GKS subroutine GFA. GFA does either hollow, solid, or hatch fill. Hollow fill (only boundaries are drawn) is the GKS default, but you can change the type of fill used by calling the GKS subroutine GSFAIS. Notice that line 7 uses GSFAIS to do solid fill. Line 12 draws the label bar on the right side of the plot; this call to LBLBAR uses the GKS color-fill option by setting IFTP to 1.

Advanced filling of label bars

This module covers material that most readers will not need. When the LBLBAR argument IFTP<>0, LBFILL is called to fill the label bar. The default version of LBFILL does color fill by calling GSFACI with color indices set in LBLBAR argument LFIN, and GKS routine GFA.

Special fill example



Code segment from clbfil.f

```
      SUBROUTINE LBFILL (IFTP,XCRA,YCRA,NCRA,INDX)
      IF (INDX.EQ.1) THEN
         CALL SFSETR ('SP - SPACING OF FILL LINES',.012)
         CALL SFSETI ('DO - DOT-FILL FLAG',0)
         CALL SFWRLD (XRA,YRA,NRA,DST,500,IND,500)
         CALL SFSETR ('SP - SPACING OF FILL LINES',.006)
         CALL SFSETI ('DO - DOT-FILL FLAG',1)
         CALL SFNORM (XRA,YRA,NRA,DST,500,IND,500)
      ENDIF
```

Synopsis

```
CALL LBFILL (IFTP, XCRA, YCRA, NCRA, INDX)
```

Arguments

IFTP

Integer, Input---A fill-type selector. When LBFILL is called by LBLBAR, this will be the argument IFTP from your call to LBLBAR, and it will always be nonzero.

XCRA(NCRA), YCRA(NCRA)

Real arrays, Input---The X and Y values, in NDCs, that define an area to be filled. The last point is a duplicate of the first.

NCRA

Integer, Input---The number of points defining the area to be filled. When LBFILL is called by LBLBAR, its value will always be 5.

INDX

Integer, Input---A color index or pattern selector. When LBFILL is called by LBLBAR, this will always be an element from the array LFIN in your call to LBLBAR. Its value may be used as a color index or as a pattern selector.

Discussion

LBFILL is called when the LBLBAR argument IFTP is nonzero. It is called once for each filled portion of the label bar. It is expected to fill that portion of the box defined by the argument arrays XCRA and YCRA. Here is the default version of the routine:

```
      SUBROUTINE LBFILL (IFTP, XCRA, YCRA, NCRA, INDX)
      DIMENSION XCRA(*), YCRA(*)
      CALL GSFACI (INDX)
      CALL GFA (NCRA-1, XCRA, YCRA)
      RETURN
      END
```

The default version fills an area by calling GFA. The argument INDX is used as a color index. In the **clbfil.f** code segment, however, INDX determines which special pattern fill is to be used. Lines 3 through 8 use several Softfill routines to draw alternating lines and dots.

Setting and retrieving Labelbar parameters

Labelbar has four parameter setting and retrieving routines. Labelbar parameters control things like line width and color, as well as text width and color. All Labelbar parameters are described in the next module, in the **labelbar_params man** page, and in the Labelbar programmer document.

Synopsis

```
      CALL LBGETI (PNAM, IVAL)
      CALL LBSETI (PNAM, IVAL)
      CALL LBGETR (PNAM, RVAL)
      CALL LBSETR (PNAM, RVAL)
```

Routines

LBGETI
: Retrieves an integer parameter value.

LBSETI
: Sets an integer parameter value.

LBGETR
: Retrieves a real parameter value.

LBSETR
: Sets a real parameter value.

Arguments

PNAM
: *Character String, Input*---A character string denoting the parameter name. The parameter name is enclosed by single quotation marks in Fortran calls to ensure that it is treated as a string rather than a real or integer variable.

RVAL
: *Real, Input or Output*---A real value used for setting or retrieving a parameter value.

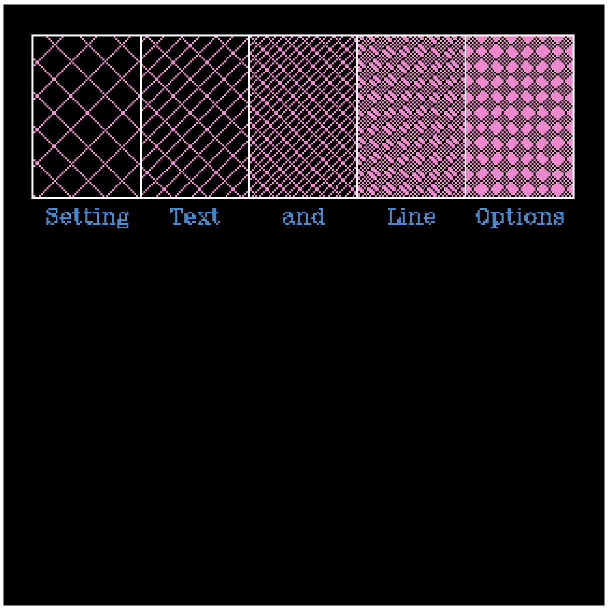
IVAL

Integer, Input or Output---An integer value used for setting or retrieving a parameter value.

Setting Labelbar parameters

Labelbar currently has six parameters, three for controlling line or character width, and three for controlling line or character color.

Modified label bar



Code segment from clblpr.f

```
      CALL COLOR
      CALL LBSETI('CBL',1)
      CALL LBSETI('CFL',15)
      CALL LBSETI('CLB',12)
      CALL LBSETR('WBL',1.5)
      CALL LBSETR('WFL',2.0)
      CALL LBSETR('WLB',5.0)
      CALL SFSETI('TY',-2)
      CALL SFSETR('AN',45.)
      CALL LBLBAR(0,.05,.95,.05,.95,5,1.,.3,IFILL1,0,LAB1,5,1)
```

Synopsis

```
CALL LBSETI('CBL',icbl)
CALL LBSETI('CFL',icfl)
CALL LBSETI('CLB',iclb)
CALL LBSETR('WBL',wbl)
CALL LBSETR('WFL',wfl)
CALL LBSETR('WLB',wlb)
```

Arguments

CBL

Integer---The color of box lines flag sets the color index of the lines defining the boundaries of the boxes in a label bar. By default CBL=-1, and the current polyline color index will be used to draw the lines.

CFL

Integer---The color of fill lines flag sets the color index for Softfill lines drawn by Labelbar. This option is only used if IFTP=0, and -4<=TY<=-1. By default, CFL=-1; the current polyline color index will be used to draw the lines.

CLB

Integer---The color of labels flag sets the color index for text drawn by Labelbar. By default CLB=-1, and the current text color index will be used to draw the labels.

WBL

Real---The width of box lines parameter sets the line width scale factor for box lines.

WFL

Real---The width of fill lines parameter sets the line width scale factor for fill lines drawn by Softfill. This parameter is only used if IFTP=0, and -4<=TY<=-1.

WLB

Real---The width of label parameter sets the line width scale factor for drawing text.

Discussion

The line width scale factor mentioned in WBL, WFL, and WLB sets the line width so that it is the scale factor times the "normal" line width. For example, if WLB is set to 3.0, then the box line width will be three times as thick as the normal line width on most output devices.

Lines 5, 6, and 7 of the **clblpr.f** code segment set the box lines to be 1.5 times the normal thickness, cross-hatch lines to be twice the normal thickness, and text lines to be five times the normal thickness. Line 1 sets up the color table. Then lines 2 through 4 use it to set box line, cross-hatch line, and text colors, respectively. Lines 8 and 9 set Softfill options to do cross-hatching at a 45-degree angle. Line 10 draws the label bar.

Chapter 15: Drawing axes, perimeters, and grids

The Gridall utility allows you to draw backgrounds for X/Y plots. It contains routines for drawing grids, perimeters, and pairs of axes. It also provides mechanisms for labeling your grids, perimeters, and axes, and for setting colors.

The following modules demonstrate how to use the GRIDAL routine to draw your choice of background, how to label your plot, and how to set color options.

Grid and axes options

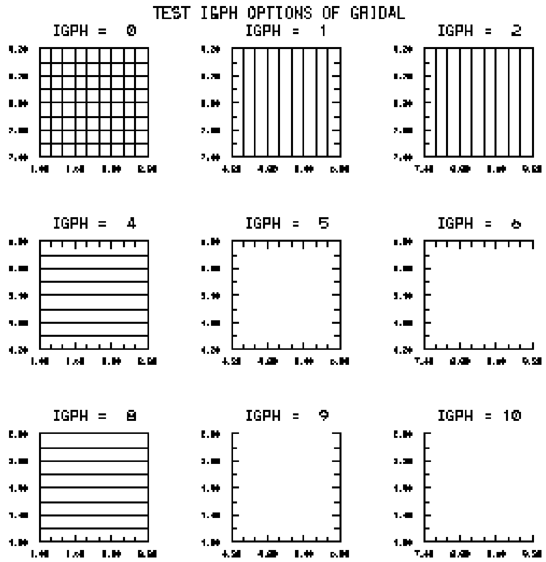


Table of Gridall user entry points

This module organizes Gridall routines by their function. Please note that the GRIDAL entry point allows you to draw any of the options with a single routine call.

Drawing routines

GRID
: Draws an unlabeled grid.

GRIDL
: Draws a labeled grid.

HALFAX
: Draws a pair of intersecting axes.

PERIM
: Draws an unlabeled perimeter.

PERIML
: Draws a labeled perimeter.

GRIDAL
: Draws any of the above.

Control routines

GACOLR
Sets the color of grids, perimeters, axes, and labels.

LABMOD
Controls the format and direction of labels.

TICK4
Changes the length and direction of tick marks.

Parameter access routines

GAGETI
Retrieves the value of an integer parameter.

GASETI
Sets the value of an integer parameter

GAGETR
Retrieves the value of a real parameter.

GASETR
Sets the value of a real parameter.

GAGETC
Retrieves the value of a character parameter.

GASETC
Sets the value of a character parameter.

Table of Gridall parameters

For a complete description of the Gridall parameters, see the **gridall_params man** page or the Gridall programmer document.

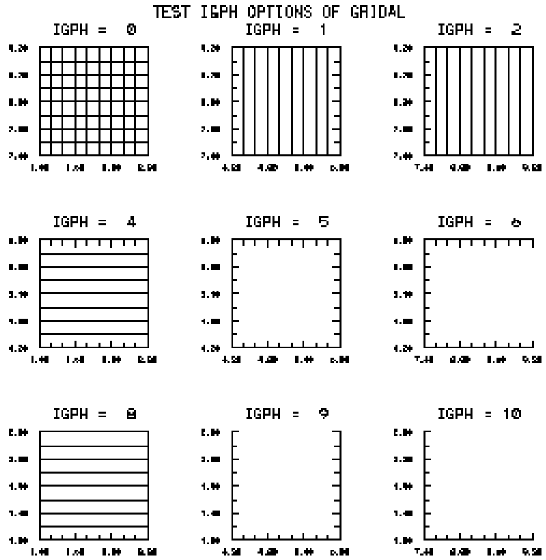
```
------------------------------------------------------------
Parameter  Brief description             Fortran type
------------------------------------------------------------
CAX        Color index for AXes          Integer
CLB        Color index for LaBels        Integer
CMJ        Color index for MaJor ticks   Integer
CMN        Color index for MiNor ticks   Integer
LTY        Labeling TYpe                 Integer
WAX        Line Width of AXes            Real
WLB        Line Width of LaBels          Real
WMJ        Line Width of MaJor ticks     Real
WMN        Line Width of MiNor ticks     Real
XLF        X-axis Label Format           Character
XLL        X-axis Label Length           Integer
```

XLO	X-axis Label Offset	Real
XLS	X-axis Label Size	Real
XMJ	X-axis MaJor tick length	Real
XMN	X-axis MiNor tick length	Real
XOR	X-axis label ORientation	Integer
YLF	Y-axis Label Format	Character
YLL	Y-axis Label Length	Integer
YLO	Y-axis Label Offset	Real
YLS	Y-axis Label Size	Real
YMJ	Y-axis MaJor tick length	Real
YMN	Y-axis MiNor tick length	Real

Drawing grids, perimeters, and axes with GRIDAL

The GRIDAL routine is the Gridall utility's "do it all" routine. By changing a single value in the GRIDAL argument list, you can draw axes, perimeters, grid lines, or some combination of the three.

Grid and axes options



Code segment from gridal.f

```
      CALL PLCHLQ(.5,.9,'DEMONSTRATION PLOT FOR GRIDAL',16.,0.,0.)
      CALL GRIDAL(5,2,6,3,1,1,IGPH,.3,.13)
      CALL FRAME
```

Synopsis

```
CALL GRIDAL (MJRX, MNRX, MJRY, MNRY, IXLAB, IYLAB, IGPH, X, Y)
```

Arguments

MJRX, MJRY

Integer, Input---The number of major divisions in the X and Y axes, respectively.

MNRX, MNRY

Integer, Input---The number of minor divisions in the X and Y axes, respectively.

The meanings of these arguments depend on the current setting of the linear/log flag in SPPS. This flag is set by the final argument in the SET call.

If the axis is linear, MJRX specifies the number of major divisions of the X/Y axis, and MNRX specifies the number of minor divisions within each major division. These values specify the number of *spaces* between GRIDAL lines or ticks, not the number of lines or ticks. Counting those at the ends, there is always one *more* major tick than the number of major divisions specified by MJRX. There is always one *less* minor tick than the number of minor divisions specified by MNRX.

If the axis is logarithmic, major division points occur at a value 10*MJRX* times the previous point. So if the minimum and maximum X-axis values are 3. and 3000., and MJRX is 1, then the major division points are 3., 30., 300., and 3000. If MNRX<=10, nine minor divisions between major divisions. For example, between 3. and 30., there would be minor division points at 6., 9., 12., 15., 18., 21., 24., and 27. If MNRX>10., minor divisions are omitted.

IXLB, IYLB

Integer, Input---IXLB and IYLB determine whether or not axes and axis labels are drawn. IXLB affects the X axis and labels, while IYLB affects the Y axis and labels.

-1

No axis or axis labels are drawn.

0

Axis drawn without axis labels.

1

Axis and axis labels are drawn.

IGPH

Integer, Input---Specifies the background type as indicated by one of the following integers:

IGPH	X axis	Y axis
0	grid	grid
1	grid	perimeter
2	grid	axis
4	perimeter	grid
5	perimeter	perimeter
6	perimeter	axis
8	axis	grid
9	axis	perimeter
10	axis	axis

XINT, YINT

Real, Input---The user "world" coordinates of the point of intersection of the two axes when IGPH=10. For other values of IGPH for which one of the axes is the axis type, XINT and/or YINT specifies the position of that axis.

Discussion

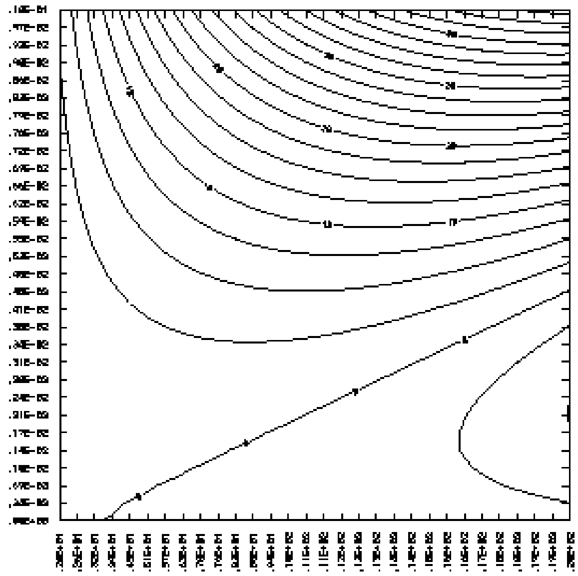
Line 1 of the **gridal.f** code segment draws a title for the plot using Plotchar routine PLCHLQ. Line 2 uses a call to GRIDAL to draw the grid lines in the graphics and to draw the grid labels. Line 3 calls FRAME to end the picture. This is necessary since none of the Gridall routines automatically call FRAME.

Labeling grids, perimeters, and axes

Sometimes you may need to change the way that labels are formatted or change their orientation on the plot. In both cases, the Gridall routine LABMOD gives you the flexibility that you need.

LABMOD does not draw the labels; it must be called before the Gridall drawing and labeling routines, such as GRIDAL, GRIDL, PERIML, or HALFAX.

Setting perimeter labels



Code segment from ccpga.f

```
      CALL LABMOD('(E7.2)','(E7.2)',0,0,10,10,0,0,1)
      CALL GRIDAL(K-1,0,N-1,0,1,1,5,0.,0.)
      CALL CPCLDR(Z,RWRK,IWRK)
      CALL FRAME
```

Synopsis

```
CALL LABMOD (FMTX, FMTY, NUMX, NUMY, ISZX, ISZY, IXDC, IYDC, IXOR)
```

Arguments

FMTX, FMTY

Character, Input---Contain format specifications for the X and Y axis numerical labels produced by GRIDAL, GRIDL, HALFAX, or PERIML. The specification must begin with a left parenthesis and end with a right parenthesis, and must not be more than ten characters long. Conversions of Fortran Format types E, F, G, and I are allowed. The default format is '(E10.3)'.

NUMX, NUMY

Integer, Input---If nonzero, NUMX is the number of characters in each X-axis numeric label. Labels with length less than NUMX will be padded at the end with blanks, allowing you to change the centering of labels. NUMY behaves in the same fashion.

ISZX, ISZY

Integer, Input---The character sizes for the labels. SIZE=ISZX/PAU, where SIZE is the size in NDCs, and PAU=1024 unless you've made a call to SETI.

IXDC

Integer, Input---The distance, in 1024ths of a screen width, from the left edge of the current viewport to the label specified by FMTY, NUMY, and ISZY. There are two special values of IXDC:

0

The Y-axis labels end 20/1024ths of a screen width (0.02 NDCs) to the left of the viewport. This is equivalent to setting IXDC=20 and is the default.

1

The Y-axis labels begin 20/1024ths of a screen width to the right of the viewport. This is equivalent to setting IXDC=20*w*, where *w*/1024 is the width of the viewport in NDCs.

Negative values of IXDC specify the distance from the right edge of the viewport.

When GRIDAL is called with IGPH=2, 6, or 10, IXDC is the distance from the Y axis, rather than from the minimum viewport coordinate, and special values 0 and 1 are equivalent to 20 and 20.

IYDC

Integer, Input---The distance, in 1024ths of a screen width, from the bottom edge of the current viewport to the label specified by FMTX, NUMX, and ISZX. There are two special values of IYDC:

0

The X-axis labels end 20/1024ths of a screen width (0.02 NDCs) below the viewport. This is equivalent to setting IYDC=20.

1

The X-axis labels begin 0.02 NDCs above the viewport. This is equivalent to setting IYDC=20*h*, where *h*/1024 is the height of the viewport in NDCs.

The default value is 20.

Negative values of IYDC specify the distance from the top edge of the viewport.

When GRIDAL is called with IGPH=8, 9, or 10, IYDC is the distance from the X axis, rather than from the minimum viewport coordinate, and special values 0 and 1 are equivalent to 20 and 20.

IXOR

Integer, Input---Specifies the orientation of the X-axis labels.

0

Horizontal.

1

Vertical.

The default orientation is horizontal.

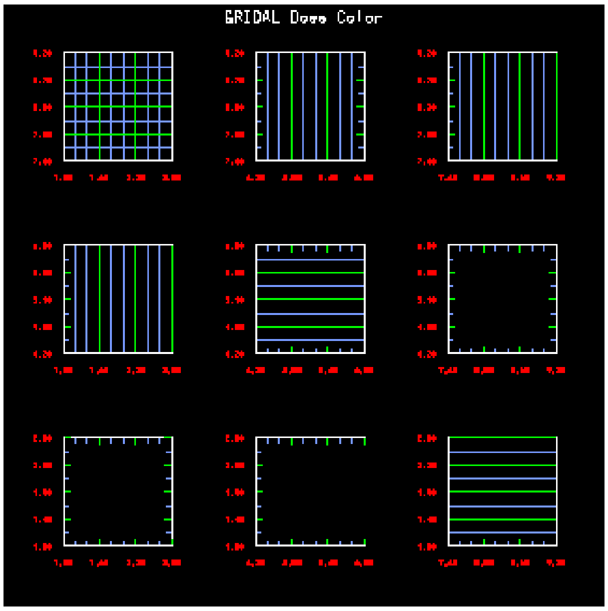
Discussion

Line 1 of the **ccpga.f** code segment chooses our own label format and uses the default label size. Because the tick marks are so close together, it is necessary to draw the X axis labels vertically, which we do by setting IXOR to 1. Line 3 draws contour lines after the perimeter and labels are drawn.

Setting grid, perimeter, and axis color

GACOLR helps you set colors in your plot. GACOLR sets color indices for axes, labels, and major and minor grid lines or tick marks.

Gridall does color



Code segment from gaclr.f

```
1        CALL COLOR
2        CALL GACOLR(1,2,3,4)
3        DO 100 IGPH=0,10
4           CALL SET(X1,X2,Y1,Y2,H1,H2,V1,V2,1)
5           CALL GRIDAL(3,3,4,2,1,1,IGPH,H1,V1)
6   100  CONTINUE
7        CALL FRAME
```

Synopsis

```
      CALL GACOLR (KAXIS, LABEL, MJTICK, MNTICK)
```

Arguments

KAXIS
Integer, Input---The color index of the desired color for the axes. KAXIS=0 by default.

LABEL
Integer, Input---The color index of the desired color for the axis labels. LABEL=0 by default.

MJTICK
Integer, Input---The color index of the desired color for the major tick marks or grid lines. MJTICK=0 by default.

MNTICK
Integer, Input---The color index of the desired color for the minor tick marks or grid lines. MNTICK=0 by default.

A value of less than 1 for any of the preceding arguments implies that no call is to be done to set the color before drawing the associated item.

Discussion

GACOLR, like LABMOD, must be called before the plot is drawn by any of the Gridall drawing routines.

Line 1 of the **gaclr.f** code segment sets up a color table by calling a user-defined routine COLOR. Line 2 sets color choices for the grids, axes, and labels, using a call to GACOLR. Lines 3 and 6 set up a DO loop to draw each of the different kinds of plot that GRIDAL is capable of. Line 4 uses a SET call to determine where on the plot the grid, perimeter, or axes will be drawn. Line 5 calls GRIDAL to draw the grid, perimeter, or axes. Line 7 calls FRAME to close the picture.

Note: The **gaclr** example produces a color plot that does not reproduce well in black and white. Please view it on your screen.

Chapter 16: Drawing graphs with Autograph

The Autograph utility allows you to draw X/Y plots of one or more curves on a labeled background. The appearance of an Autograph plot is controlled by a series of parameters that control characteristics such as line color and axis labels. These parameters are set and retrieved through user-callable entry points in Autograph, to which you supply the name of the parameter and the value to be assigned to it. For plots containing multiple curves, you may draw curves with user-selectable dash patterns and (or) colors.

The following modules show you how to use the Autograph utilities to draw and label your plots. Additionally, examples show how to use dash patterns and colors.

X/Y plot capabilities

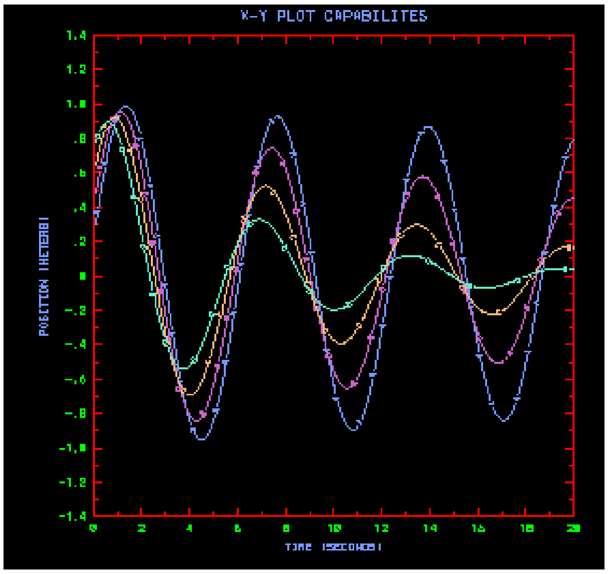


Table of Autograph user entry points

This module organizes Autograph routines by their functions.

Simple plotting routines

EZY
: Plots a complete graph of a single curve through the points (i,y(i)).

EZXY
: Plots a complete graph of a single curve through the points (x(i),y(i)).

EZMY
: Plots a complete graph of M curves through the points (i,y(i,1)), (i,y(i,2)), ... ,(i,y(i,M)). Additionally, EZMY can accept transposed y arrays if the appropriate parameters are set.

EZMXY

Plots a complete graph of M curves through the points (x(i),y(i,1)), (x(i),y(i,2)), ... , (x(i),y(i,M)). EZMXY is also capable of plotting (x(i,1),y(i,1)), ... , (x(i,M),y(i,M)), and EZMXY can accept transposed x and y arrays if the appropriate parameters are set.

Internal plotting routines

AGSTUP

Initializes Autograph.

AGBACK

Draws the background (axes, labels, etc.) for an Autograph plot.

AGCURV

Draws the curves (one per call) for an Autograph plot.

Parameter control routines

AGSETP

Sets an array of parameters to Autograph.

AGSETR

Sets a single real parameter to Autograph.

AGSETI

Sets a single integer parameter to Autograph.

AGSETC

Sets a single character parameter to Autograph.

AGGETP

Retrieves an array of parameters to Autograph.

AGGETR

Retrieves a single real parameter to Autograph.

AGGETI

Retrieves a single integer parameter to Autograph.

AGGETC

Retrieves a single character parameter to Autograph.

AGSAVE

Saves Autograph's current state to a file.

AGRSTR

Restores Autograph's state from a file.

Color/intensity control routines

AGCHAX
Called by Autograph before and after drawing axes, tick marks, and axis numbering, AGCHAX can be replaced by a user-written routine to change intensities and colors for the next object to be drawn.

AGCHCU
Called by Autograph before and after drawing curves, AGCHCU can be replaced by a user-written routine to change intensities and colors for the next object to be drawn.

AGCHIL
Called by Autograph before and after drawing information labels, AGCHIL can be replaced by a user-written routine to change intensities and colors for the next object to be drawn.

AGCHNL
Called by Autograph before drawing each numeric label for the axes, AGCHNL can be replaced by a user-written routine to change the format or appearance of a numeric label.

Dash pattern control routines

AGBNCH
Converts binary dash patterns to character dash patterns.

AGDSHN
Generates the parameter names for the individual members of the dash pattern group to be passed to an AGSETC call for setting the dash patterns.

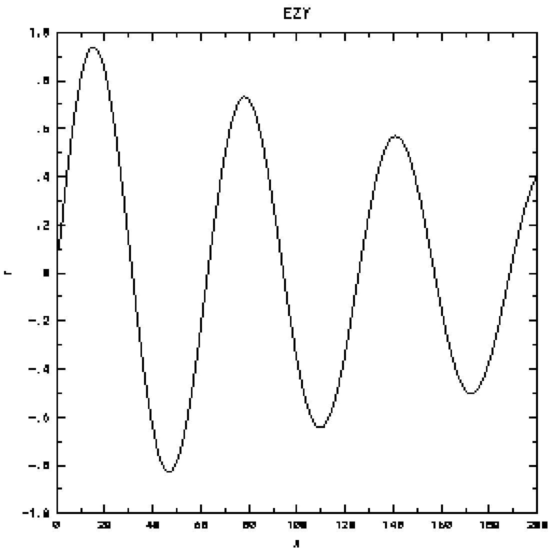
Miscellaneous routines

AGUTOL
Called by Autograph, AGUTOL controls the user-coordinate system to label-coordinate system mapping. AGUTOL can be replaced by a user-written routine to alter this mapping for any of the four (left, right, top, bottom) axes.

AGPWRT
Draws the numeric, axis, and informational labels. AGPWRT can be replaced by a user-written routine to alter the appearance of text strings on the plot.

Drawing one curve per plot with EZY

EZY allows you to draw an X/Y plot of a single, one-dimensional array. EZY draws the background axes and plots a curve through the ordered pairs (i,y(i)). The plots drawn by EZY can be embellished by any of the techniques discussed later in this chapter.



Code segment from fagezy.f

```
1        CALL OPNGKS
2        CALL EZY (YDRA,NPTS,'EZY$')
3        CALL CLSGKS
```

Synopsis

```
CALL EZY (YDRA,NPTS,TITLE)
```

Arguments

YDRA

Real, Input---Array at least NPTS in length containing the Y-coordinate values to be plotted.

NPTS

Integer, Input---Contains the number of points to be plotted.

TITLE

Character, Input---Contains the title for the plot.

Discussion

Line 1 of the **fagezy.f** code segment opens and initializes the GKS package. Line 2 passes the real array YDRA(NPTS) to EZY for plotting. The plot title will be 'EZY$'. Note the dollar sign ($) at the end of the title string, which behaves as a string terminator. A common mistake is to drop the dollar sign from the end of the title string. Line 3 closes GKS.

EZY is intended to be a simple interface to the Autograph package. It actually handles most of the details of producing a plot for you. EZY calls the following routines in this order:

AGSTUP

Autograph set up, which initializes the Autograph package and calls SET if necessary.

AGBACK

Autograph background, which draws the background for the plot, including axes and all labels.

AGCURV

Autograph curve, which draws the actual curve on the background.

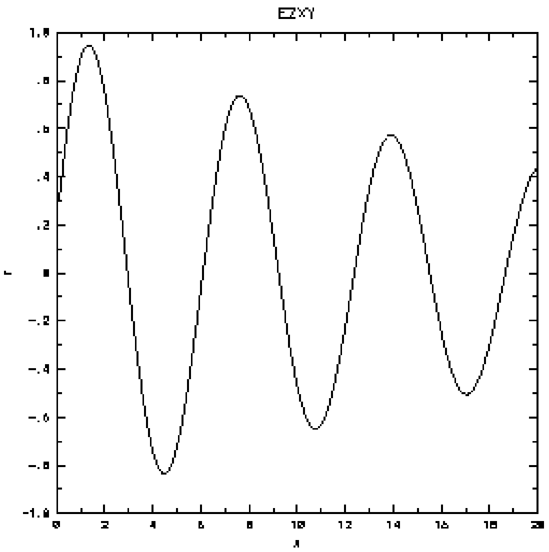
FRAME

which terminates the current metacode plot frame and advances to the next plot frame.

It is possible to place multiple plots on a single frame. See the manual pages for the subroutine SET and the Autograph parameters SET and FRAME in the **autograph_params man** page.

Drawing one curve per plot with ordered pairs using EZXY

EZXY allows you to draw an X/Y plot of ordered pairs. EZXY draws the background and axes and plots a curve through the ordered pairs (x(i),y(i)). The plots drawn by EZXY can be embellished by any of the techniques discussed later in this chapter.



Code segment from fagezxy.f

```
      CALL OPNGKS
      CALL EZXY (XDRA,YDRA,NPTS,'EZXY$')
      CALL CLSGKS
```

Synopsis

```
      CALL EZXY (XDRA,YDRA,NPTS,TITLE)
```

Arguments

XDRA

Real, Input---A real array at least NPTS in length containing the X-coordinate values to be plotted.

YDRA

Real, Input---A real array at least NPTS in length containing the Y-coordinate values to be plotted.

NPTS

Integer, Input---Contains the number of points to be plotted.

TITLE

Character, Input---Contains the title for the plot.

Discussion

Line 1 of the **fagezy.f** code segment opens and initializes the GKS package. Line 2 passes the real arrays YDRA(NPTS) and XDRA(NPTS) to EZXY for plotting. The plot title will be 'EZXY$'. Note the dollar sign ($) at the end of the title string which behaves as a string terminator. A common mistake is to drop the dollar sign from the end of the title string. Line 3 closes GKS.

EZXY is intended to be a simple interface to the Autograph package. It actually handles most of the details of producing a plot for you. EZXY calls the following routines in this order:

AGSTUP

Autograph set up, which initializes the Autograph package and calls SET if necessary.

AGBACK

Autograph background, which draws the background for the plot, including axes and all labels.

AGCURV

Autograph curve, which draws the actual curve on the background.

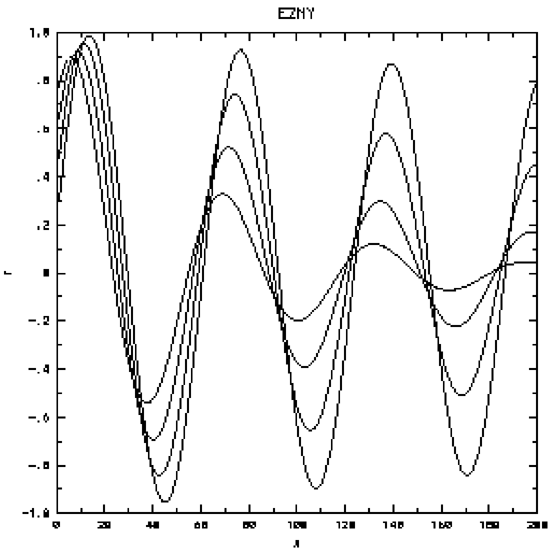
FRAME

which terminates the current metacode plot frame and advances to the next plot frame.

It is possible to place multiple plots on a single frame. See the manual pages for the subroutine SET and the Autograph parameters SET and FRAME in the **autograph_params man** page.

Drawing multiple curves per plot with EZMY

EZMY allows you to draw an X/Y plot of a single, two-dimensional array in which one dimension represents the curve number and the other dimension represents the points along that curve. Thus EZMY draws a background axis and plots a curve through the ordered pairs (i,y(i,1)) for i=1 to NPNTS, another through the points (i,y(i,2) for i=1 to NPNTS, and so on for NCURVE curves. The plots drawn by EZMY can be embellished by any of the techniques discussed later in this chapter.



Code segment from fagezmy.f

```
      CALL OPNGKS
      CALL EZMY (YDRA,NPTS,NCURVE,NPTS,'EZMY$'
      CALL CLSGKS
```

Synopsis

```
CALL EZMY (YDRA, IMAX, NCURVE, NPTS, TITLE)
```

Arguments

YDRA

Real, Input---A real array declared as YDRA(IMAX,NCURVE) which contains the Y-coordinate values to be plotted. The data in the plot are arranged (by default) such that the second index refers to the curve number and the first index refers to a point along a curve.

IMAX

Integer, Input---The maximum value of the first dimension of YDRA.

NCURVE

Integer, Input---The number of curves to be drawn. This number should be less than or equal to the maximum value of the second dimension (by default) of YDRA.

NPTS

Integer, Input---Contains the number of points to be plotted on each curve.

TITLE

Character, Input---Contains the Title for the plot.

Discussion

Line 1 of the **fagezmy.f** code segment opens and initializes the GKS package. Line 2 passes the real array YDRA(IMAX,NCURVE) to EZMY for plotting. The plot title will be 'EZMY$'. Note the dollar sign ($) at the end of the title string, which behaves as a string terminator. A common mistake is to drop the dollar sign from the end of the title string. Line 3 closes GKS.

EZMY is intended to be a simple interface to the Autograph package. It actually handles most of the details of producing a plot for you. EZMY calls the following routines in this order:

AGSTUP
: Autograph set up, which initializes the Autograph package and calls SET if necessary.

AGBACK
: Autograph background, which draws the background for the plot, including axes and all labels.

AGCURV
: Autograph curve, which draws the actual curves on the background. AGCURV is called NCURVE times: once for each curve.

FRAME
: which terminates the current metacode plot frame and advances to the next plot frame.

If the array is transposed from what Autograph expects, you can use the following call to set a parameter to inform Autograph of this.

```
CALL AGSETI('ROW.',-1)
```

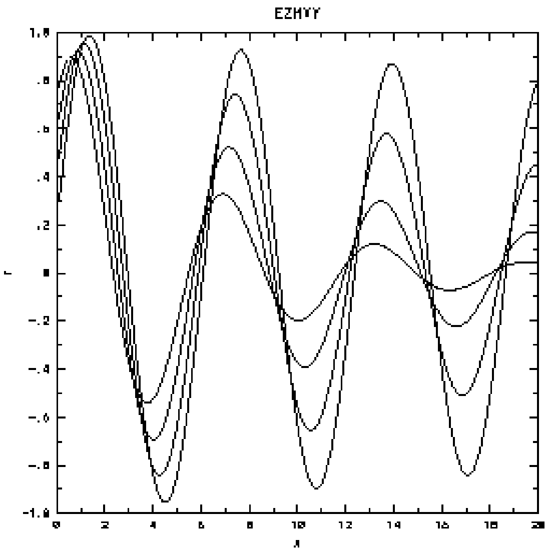
This call would be placed between the calls to OPNGKS and EZMY. After this parameter has been set, Autograph will expect the input arrays to a multicurve plot to look like YDRA(NCURVE,NPTS). To return Autograph to the default behavior (expecting YDRA(NPTS,NCURVE) set ROW back to a positive value.

```
CALL AGSETI('ROW.',1)
```

Common mistakes with the AGSET*x* calls are to forget the period (.) after the parameter name or to forget to enclose the parameter name in quotation marks (').

Drawing multiple curves with ordered pairs using EZMXY

EZMXY allows you to draw an X/Y plot of a two- dimensional array containing the Y-coordinates to be plotted and a one-dimensional array containing the corresponding X-coordinates. For the two-dimensional array, one dimension represents the curve number, and the other dimension represents the points along that curve. (Optionally, the Autograph parameter ROW can be set to allow a two-dimensional X-coordinate array.) Thus EZMXY draws a background axis and plots a curve through the ordered pairs (x(i),y(i,1)) for i=1 to NPNTS, another through the points (x(i),y(i,2) for i=1 to NPNTS, and so on for NCURVE curves. The plots drawn by EZMXY can be embellished by any of the techniques discussed later in this chapter.



Code segment from fagezmxy.f

```
1        CALL OPNGKS
2        CALL EZMXY (XDRA,YDRA,NPTS,NCURVE,NPTS,'EZMXY$')
3        CALL CLSGKS
```

Synopsis

```
CALL EZMXY (XDRA, YDRA, IMAX, NCURVE,NPTS, TITLE)
```

Arguments

XDRA

Real, Input---A real array at least NPTS in length containing the X-coordinate values of the ordered pairs to be plotted.

YDRA

Real, Input---A real array declared as YDRA(IMAX,NCURVE) that contains the Y-coordinate values of the ordered pairs to be plotted. The data in the plot are arranged (by default) such that the second index refers to the curve number and the first index refers to a point along a curve.

IMAX

Integer, Input---The maximum value of the first dimension of YDRA.

NCURVE

Integer, Input---The number of curves to be drawn. This number should be less than or equal to the maximum value of the second dimension (by default) of YDRA.

NPTS

Integer, Input---Contains the number of points to be plotted on each curve.

TITLE

Character, Input---Contains the title for the plot.

Discussion

The first line of the code segment opens and initializes the GKS package. The second line passes the real array YDRA(IMAX,NCURVE) to EZMXY for plotting. The plot title will be 'EZMXY$'. Note the dollar sign ($) at the end of the title string, which behaves as a string terminator. A common mistake is to drop the dollar sign from the end of the title string. The last line of the Fortran segment closes GKS.

EZMXY is intended to be a simple interface to the Autograph package. It actually handles several of the details of producing a plot for you. EZMXY calls the following routines in this order:

AGSTUP

Autograph set up, which initializes the Autograph package and calls SET if necessary.

AGBACK

Autograph background, which draws the background for the plot, including axes and all labels.

AGCURV

Autograph curve, which draws the actual curves on the background. AGCURV is called NCURVE times: once for each curve.

FRAME

which terminates the current metacode plot frame and advances to the next plot frame.

If the YDRA array is transposed from what Autograph expects, you can use the following call to set a parameter to inform Autograph of this.

```
CALL AGSETI('ROW.',-1)
```

This call would be placed between the calls to OPNGKS and EZMXY. After this parameter has been set, Autograph will expect the input arrays to a multicurve plot to look like YDRA(NCURVE,NPTS). To return Autograph to the default behavior (expecting YDRA(NPTS,NCURVE), set ROW back to a positive value.

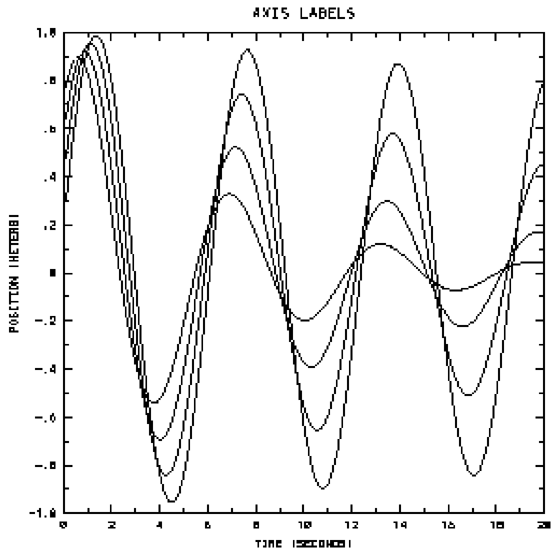
```
CALL AGSETI('ROW.',1)
```

Common mistakes with the AGSET*x* calls are to forget the period (.) after the parameter name or to forget to enclose the parameter name in quotation marks (').

Changing the axis text labels

You may change an axis text label on a plot by setting three parameters (two character parameters and one integer parameter) before an EZ-routine is called (or more specifically, before AGBACK is called). The parameters should be set in order to avoid unexpected results.

Changing the axis text labels



Code segment from fagaxlbl.f

```
C       CHANGE BOTTOM (B) AXIS LABEL
        CALL AGSETC('LABEL/NAME.','B')
        CALL AGSETI('LINE/NUMBER.',-100)
        CALL AGSETC('LINE/TEXT.','TIME (SECONDS)$')
C       CHANGE LEFT (L) AXIS LABEL
        CALL AGSETC('LABEL/NAME.','L')
        CALL AGSETI('LINE/NUMBER.',100)
        CALL AGSETC('LINE/TEXT.','POSITION (METERS)$')
        CALL EZMXY (XDRA,YDRA,NPTS,NCURVE,NPTS,'AXIS TEXT LABELS$')
```

Synopsis

```
CALL AGSETC ('LABEL/NAME.', cval)
CALL AGSETI ('LINE/NUMBER.', ival)
```

Arguments

LABEL/NAME

Character---Works in conjunction with LINE/NUMBER to refer to the label to be changed. After they are set, LINE/TEXT can be set to alter the text of the label. The values of LABEL/NAME and LINE/NUMBER combine as follows to select a specific axis label.

LABEL/NAME	LINE/NUMBER
B---Bottom label	-100
L---Left label	100
R---Right label	-100
T---Top label	100

Discussion

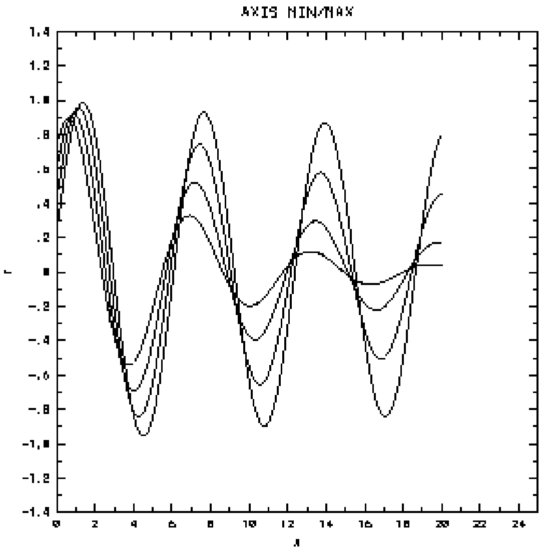
Lines 2 and 3 of the **fagaxlbl.f** code segment specify that we are about to modify information about the text label on the bottom axis. Line 4 changes the contents of that label's text string. Lines 6, 7, and 8 make a similar change to the information buffer for the left axis text label.

The most common mistakes with setting axis labels are to either drop one of the parameter-setting calls or to confuse the order of the calls. This can produce unexpected results. Additionally as has been mentioned before, the parameter strings must be terminated with a period. Finally, the label text strings must be terminated with a dollar sign ($).

Changing the axis minimum and maximum values

The minimum and maximum values on an axis can be changed with a group of three subroutine calls to set three Autograph parameters.

Changing axis minimum and maximum



Code segment from fagaxmax.f

```
      CALL AGSETR('Y/MINIMUM.',-1.4)
      CALL AGSETR('Y/MAXIMUM.', 1.4)
      CALL AGSETR('Y/NICE.',    0.0)

      CALL AGSETR('X/MINIMUM.', 0.0)
      CALL AGSETR('X/MAXIMUM.',25.0)
      CALL AGSETR('X/NICE.',    0.0)

      CALL EZMXY (XDRA,YDRA,NPTS,NCURVE,NPTS,'AXIS MIN/MAX$')
```

Synopsis

```
CALL AGSETR ('X/MINIMUM.', rval)
CALL AGSETR ('X/MAXIMUM.', rval)
CALL AGSETR ('X/NICE.', rval)
```

Arguments

Y/MINIMUM
Real---The minimum value for the plot's Y-axis.

Y/MAXIMUM
Real---The maximum value for the plot's Y-axis.

Y/NICE
Real---Flag to indicate to Autograph that the Y-axis minimum and maximum have already been set by the user and should not be changed. The value of this flag will dictate the appearance of the tick marks on the axis:

-1.0
Adjust to force major tick marks at the endpoints of the left axis. (Default)
0.0
Do not adjust either vertical axis.
+1.0
Adjust to force major tick marks at the endpoints of the right axis.

X/MINIMUM
Real---The minimum value for the plot's X-axis.

X/MAXIMUM
Real---The maximum value for the plot's Y-axis.

X/NICE
Real---Flag to indicate to Autograph that the X-axis minimum and maximum have already been set by the user and should not be changed. The value of this flag will dictate the appearance of the tick marks on the axis:

-1.0
Adjust to force major tick marks at the endpoints of the bottom axis. (Default)
0.0
Do not adjust either horizontal axis.
+1.0
Adjust to force major tick marks at the endpoints of the top axis.

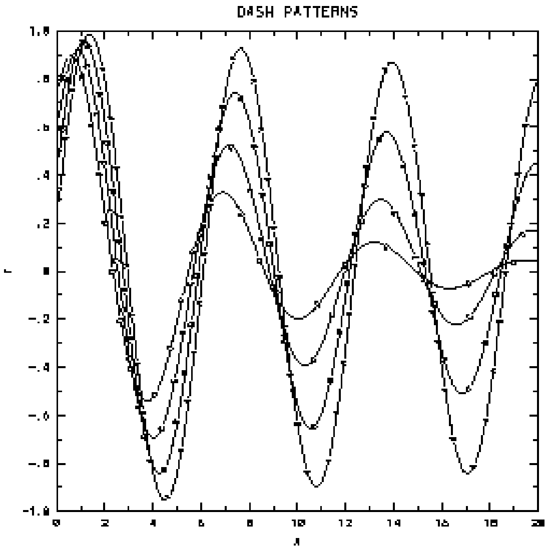
Discussion

In the **fagaxmax.f** code segment, we set the Y-axis minimum in line 1 and set the Y-axis maximum in line 2. In line 3, we set the Y/NICE flag to notify Autograph that we have set the minimum and maximum values for this plot and that Autograph should not attempt to rescale the Y-axis for the plot. Lines 5 through 7 accomplish the same on the X-axis.

Differentiating multiple curves with a dash pattern

Sometimes we would like a simple method for differentiating multiple curves on one plot. The simplest way to do this is to mark each curve with a letter indicating the order in which the curves were drawn. More complex methods of marking the curves and creating your own dash patterns are also available.

Changing the dash pattern



Code segment from fagcudsh.f

```
      CALL OPNGKS
      CALL AGSETR('DASH/SELECTOR.',-1.0)
      CALL EZMXY (XDRA,YDRA,NPTS,NCURVE,NPTS,'DASH PATTERNS$')
      CALL CLSGKS
```

Synopsis

```
      CALL AGSETR (PNAM, rval)
```

Arguments

PNAM

Character, Input---Specifies which parameter is to be changed.

DASH/SELECTOR

Real---Selects DASH/SELECTOR as the parameter to be updated. DASH/SELECTOR controls the dash patterns used for drawing curves. The default is +1.0.

-n

Specifies that Autograph use the 26 built-in "alphabetic" dash patterns. If more than 26 curves are to be drawn, then the dash patterns for the curves are chosen round-robin fashion from this list of 26 dash patterns.

+n

A positive integral real number less than 26 specifies the use of the first *n* user-defined dash patterns. If more than *n* curves are to be drawn, then the dash patterns for the curves are chosen round-robin fashion from this list of *n* dash patterns. The default set of user-defined dash patterns produces all solid lines.

Discussion

Line 2of the **fagcudsh.f** code segment sets the Autograph parameter DASH/SELECTOR to -1.0, telling Autograph to use the alphabetic dash patterns for drawing the curves. This is followed by a call to EZMXY, which draws a set of NCURVE curves (using the alphabetic dash patterns).

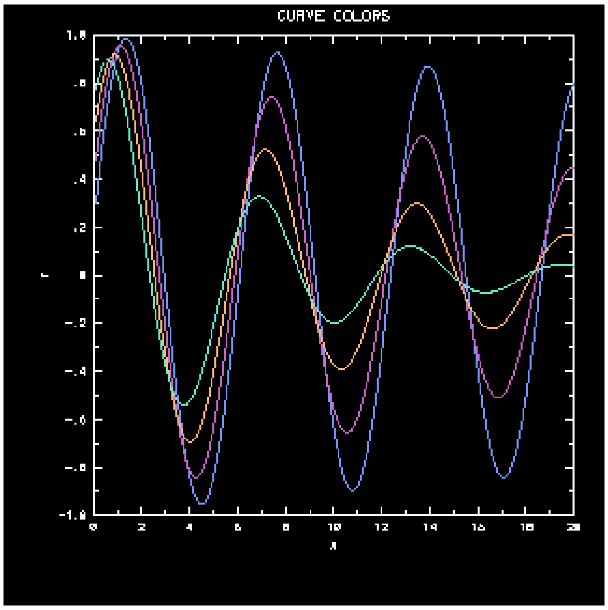
The most common mistake, as has been mentioned before, is to forget to terminate the parameter name string with a period. Also note that the parameter value type must match the AGSET*x* routine being used to set the parameter.

For more information on user-definable dash patterns, see the **autograph_params man** page for the parameters DASH, DASH/SELECTOR, DASH/LENGTH, DASH/CHARACTER, DASH/DOLLAR-QUOTE, DASH/PATTERNS/1 through DASH/PATTERNS/26, and the DASHD, AGDSHN, and AGCURV **man** pages.

Changing the curve colors

Autograph has several routines called before and after drawing various portions of a plot. One such routine is AGCHCU, which is called before and after drawing each curve. The default version of AGCHCU does nothing, but you can write your own version of this routine which changes the GKS polyline color index and the GKS text color index before drawing a line.

Changing the curve colors



Code segment from fagcuclr.f

```
1        SUBROUTINE AGCHCU(IFLG,KDSH)
2        CALL PLOTIF (0.,0.,2)
3        IF (IFLG .EQ. 0) THEN
4           CALL GSPLCI( ABS(KDSH)+3 )
5           CALL GSTXCI( ABS(KDSH)+3 )
6        ELSE
7           CALL GSPLCI(1)
8           CALL GSTXCI(1)
9        ENDIF
10       RETURN
11       END
```

Synopsis

```
      CALL AGCHCU (IFLG, KDSH)
```

Arguments

IFLG

Integer, Input---Indicates whether this call is occurring before or after the object is drawn. If IFLG is 0, Autograph is about to draw a curve. If IFLG is nonzero, Autograph has just finished drawing a curve.

KDSH

Integer, Input---Indicates the curve number/dash pattern being used to draw this curve. The curve number will be the absolute value of KDSH. KDSH will be negative if the alphabetic dash patterns are being used and positive if the default or user-definable dash patterns are being used.

Discussion

Line 2 of the **fagcuclr.f** calls the SPPS routine PLOTIF to move the "pen" to the origin in the "pen-up" position. This has the effect of flushing the SPPS polyline buffer so that any changes we make actually show up in the plot. Lines 3, 6, and 9 contain an IF construct that tests whether AGCHCU is being called before or after a curve has been drawn.

Lines 4 and 5 are executed before a curve is drawn. Line 4 sets the GKS polyline color index according to which curve is being drawn, indicated by KDSH, and line 5 sets the GKS text color index to the same value to handle cases in which the alphabetic dash patterns are used. Lines 7 and 8 are executed after a line has been drawn and serve to reset the GKS polyline and text color indices to the foreground color.

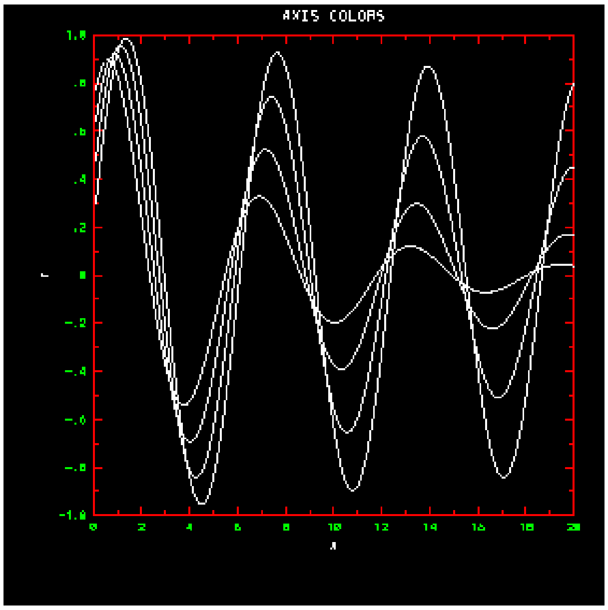
There are three common errors in setting colors. The first is to forget to set both the polyline and text colors back to the foreground color after drawing an object. The second common error results from trying to use GSCR to change the foreground color within AGCHCU. And last, if you forget to call PLOTIF before changing the color indices, GKS calls made while there is data in the SPPS polyline buffer will not have the desired effect.

Changing the axis colors

Autograph has several routines called before and after drawing various portions of a plot. One such routine is AGCHAX, which is called before and after drawing each axis object. The default version of AGCHAX does nothing, but you can write your own version of this routine which changes the GKS polyline color index and the GKS text color index before drawing a line.

Your modified version of AGCHCU must be loaded before the NCAR Graphics libraries are loaded; otherwise, the default AGCHCU will be used.

Changing the axis color



Code segment from fagaxclr.f

```
      SUBROUTINE AGCHAX(IFLG,IAXS,IPRT,VILS)
      CALL PLOTIF (0.,0.,2)
      IF (IFLG .EQ. 0) THEN
         CALL GSPLCI(3)
         CALL GSTXCI(2)
      ELSE
         CALL GSPLCI(1)
         CALL GSTXCI(1)
      ENDIF
      RETURN
      END
```

Synopsis

```
CALL AGCHAX (IFLG, IAXS, IPRT, VILS)
```

Arguments

IFLG

Integer, Input---Indicates whether this call is occurring before or after the object is drawn. If IFLG is 0, Autograph is about to draw a curve. If IFLG is nonzero, Autograph has just finished drawing a curve.

IAXS
: *Integer, Input*---Indicates which axis is being drawn.

1
: Left axis

2
: Right axis

3
: Bottom axis

4
: Top axis

IPRT
: *Integer, Input*---Indicates which part of the axis is being drawn.

1
: The axis line

2
: A major tick mark

3
: A minor tick mark

4
: The mantissa of a numeric label

5
: The exponent of a numeric label

VILS
: *Real, Input*---Indicates the point in the label system at which the object is being drawn.

Discussion

Line 2 of the **fagaxclr.f** code segment calls the SPPS routine PLOTIF to move the "pen" to the origin in the "pen-up" position. This has the effect of flushing the SPPS polyline buffer so that any changes we make actually show up in the plot. Lines 3, 6, and 9 contain an IF construct that tests whether AGCHAX is being called before or after an object has been drawn. Lines 4 and 5 are executed before an object is drawn. Line 4 sets the GKS polyline color index, and line 5 sets the GKS text color index. Lines 7 and 8 are executed after an object has been drawn and serve to reset the GKS polyline and text color indices to the foreground color.

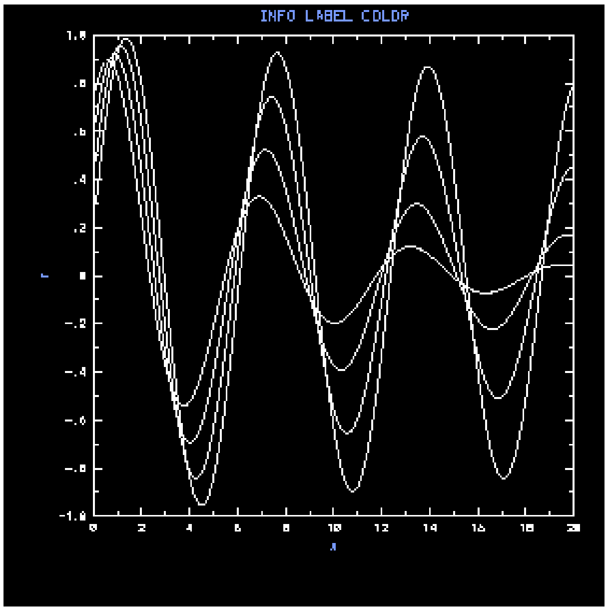
There are three common errors in setting colors. The first is to forget to set both the polyline and text colors back to the foreground color after drawing an object. The second common error results from trying to use GSCR to change the foreground color within AGCHAX. And last, if you forget to call PLOTIF before changing the color indices, GKS calls made while there is data in the SPPS polyline buffer will not have the desired effect.

Your modified version of AGCHAX must be loaded before the NCAR Graphics libraries; otherwise, the default AGCHAX will be used.

Changing the info label colors

Autograph has several routines called before and after drawing various portions of a plot. One such routine is AGCHIL, which is called before and after drawing each informational label. The default version of AGCHIL does nothing, but you can write your own version of this routine that changes the GKS polyline color index and the GKS text color index before drawing a label.

Changing the informational label color



Code segment from fagilclr.f

```
      SUBROUTINE AGCHIL(IFLG,LBNM,LNNO)
      CALL PLOTIF (0.,0.,2)
      IF (IFLG .EQ. 0) THEN
         CALL GSTXCI(4)
      ELSE
         CALL GSTXCI(1)
      ENDIF
      RETURN
      END
```

Synopsis

```
CALL AGCHIL (IFLG, LBNM, LNNO)
```

Arguments

IFLG

Integer, Input---Indicates whether this call is occurring before or after the object is drawn. If IFLG is 0, Autograph is about to draw a curve. If IFLG is nonzero, Autograph has just finished drawing a curve.

LBNM
Character, Input---Indicates the name of the label being drawn.

LNNO
Integer, Input---Indicates the number of the label being drawn.

Discussion

Line 2 of the **fagilclr.f** code segment calls the SPPS routine PLOTIF to move the "pen" to the origin in the "pen-up" position. This has the effect of flushing the SPPS polyline buffer so that any changes we make actually show up in the plot. Lines 3, 5, and 7 contain an IF construct that tests whether AGCHIL is being called before or after a label has been drawn. Line 4 is executed before a label is drawn; it sets the GKS text color index. Line 6 is executed after a label has been drawn and serves to reset the GKS text color index to the foreground color.

There are three common errors in setting colors. The first is to forget to set the text color back to the foreground color after drawing an object. The second common error results from trying to use GSCR to change the foreground color within AGCHIL. And last, if you forget to call PLOTIF before changing the color indices, GKS calls made while there is data in the SPPS polyline buffer will not have the desired effect.

Your modified version of AGCHIL must be loaded before the NCAR Graphics libraries; otherwise, the default AGCHIL will be used.

Chapter 17: Creating bar charts using the Histogram utility

Histograms (bar charts) are used to show the distribution of values in a like-sample dataset. The individual values are sorted by size into a number of value ranges called "class intervals." The number of samples out of the total that fall into a class interval are represented as a histogram bar height. The height of each histogram bar is proportional to the number of samples in that class interval.

Bar chart created using Histogram

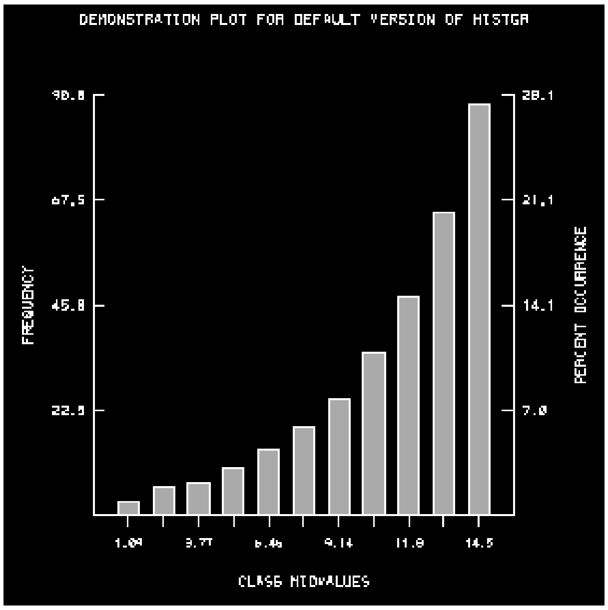


Table of Histogram user entry points

There are five user-callable entries within the Histogram utility. HISTGR is called to generate a histogram. HSTOPC, HSTOPI, HSTOPL, and HSTOPR are called before HISTGR to set parameters (options) that affect the output histogram. All parameters have an original default setting.

Action routine

HISTGR
: Generates histograms of several types. Raw sample data can be sorted into a default range of classes or an assigned range of classes. Class interval data, which was sorted at an earlier step, can be input and displayed. Two histograms can be displayed simultaneously.

Parameter routines

HSTOPC
: Sets parameters of type character, such as titles and labels.

HSTOPI
: Sets parameters of type integer, including the color indices to be assigned to the various components of the graphic and the size and orientation of the class (bar) labels.

HSTOPL
: Sets parameters of type logical, which is basically an on/off switch for these options.

HSTOPR
: Sets parameters of type real, including the location on the plot device where the histogram is to be placed, the spacing of the histogram classes (bars), and if two histograms are present, the overlap

of the two.

Table of Histogram parameters

An arbitrary number of options can be set through successive calls to any of the parameter-setting routines. A call to the HISTGR entry will then generate the desired histogram. The four parameter-setting routines are used to set parameters of type character, integer, logical, and real.

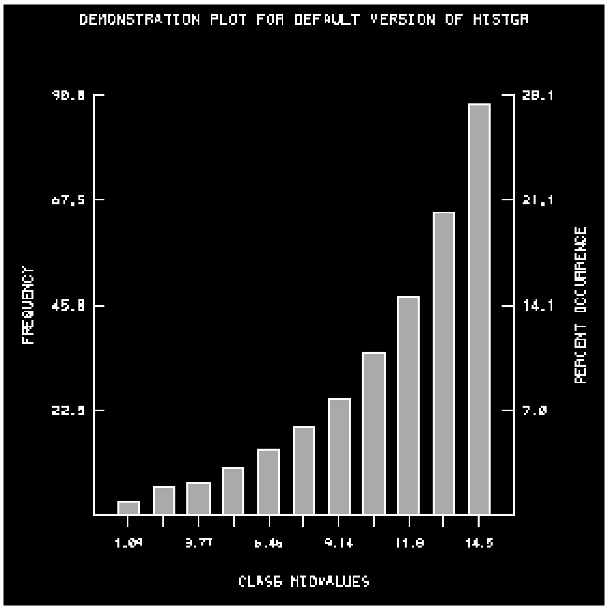
For a complete description of the Histogram parameters, see the **histogram_params man** page.

Parameter	Fortran type	Set by	Brief description	Default value
FOR	Character	HSTOPC	FORmat for class labels	'(G10.3)'
TIT	Character	HSTOPC	A main TITle	No title
LAB	Character	HSTOPC	Class interval axis LABel	'CLASS INTERVALS
				('MID=OFF')
				'CLASS MIDVALUES
				('MID=ON')
FQN	Character	HSTOPC	FreQueNcy axis label	'FREQUENCY'
CHR	Character	HSTOPC	Class interval label string	Internal numeric
COL	Integer	HSTOPI	8 COLorable graphic components	All are white on
CLA	Integer	HSTOPI	Size of CLAss labels	Medium size char
			Orientation of class labels	Horizontal label
HOR	Logical	HSTOPL	Direction of histogram bars	Vertical
PER	Logical	HSTOPL	A PERcentage axis opposite the	The axis is draw
			frequency axis	
MID	Logical	HSTOPL	Location of class interval labels	Placed at interv
SHA	Logical	HSTOPL	SHAding of histogram bars	Bars are shaded
DRL	Logical	HSTOPL	Grid lines through bars	No lines are dra
MED	Logical	HSTOPL	A line drawn at data MEDian	No line is drawn
PRM	Logical	HSTOPL	A PeRiMeter around the histogram	No perimeter is
FRA	Logical	HSTOPL	Advance the FRAme	The frame is adv
LIS	Logical	HSTOPL	LISt parameter values on output	No printed list
DEF	Logical	HSTOPL	Reset all parameters to DEFaults	
WIN	Real	HSTOPR	Region of frame to put histogram	The entire frame
SPA	Real	HSTOPR	SPAcing between histogram bars	Spacing = 2.0
			Dual histogram overlap	Overlap = -1.0

Generating a histogram

The HISTGR routine generates histograms for a single data array or compares the histogram distributions of two data arrays. Data values are partitioned into classes. You can make histogram bars represent either the number of occurrences within each class, or a Y-value associated with that class.

A histogram with default options



Code segment 1 from thstgr.f

```
      CALL HSTOPL('DEF=ON')
      CALL HISTGR(DAT1, NDIM, NPTS, IFLAG, CLASS, NCLASS, WORK, NWRK)
```

Synopsis

```
CALL HISTGR (DAT1, NDIM, NPTS, IFLAG, CLASS, NCLASS, WORK, NWRK)
```

Arguments

DAT1(NDIM,2)

Real, Input---An NDIM by 2 array. See argument IFLAG for a description of input data options.

NDIM

Integer, Input---The first dimension of DAT1.

NPTS

Integer, Input---Number of points to be sorted or number of previously sorted class intervals. The NPTS argument depends on the value of IFLAG:

When IFLAG=0 or 1, NPTS is the number of points in array DAT1 to be sorted into class intervals (bins).

When IFLAG=2 or 3, NPTS is the number of previously sorted class intervals (bins) in array CLASS.

IFLAG

Integer, Input---Used to select one of four histogram options:

0

A single array of length NPTS is loaded into the DAT1 array. HISTGR computes NCLASS equally sized class intervals that vary from the minimum value in DAT1 to the maximum value in steps of (MAX-MIN)/NCLASS.

The DAT1(NPTS) values are sorted into these bins. The result is a histogram with NCLASS bars (bin counts). The bar height can be labeled by number of points in the bin or by the percentage this is of the total number of points.

1

Similar to IFLAG = 0, except the user can specify the class intervals. For example, the following code sorts DAT1 values into five equally sized bins in the value range from 0. to 10.:

```
NCLASS = 5
CLASS(1) = 0.
CLASS(2) = 2.
CLASS(3) = 4.
CLASS(4) = 6.
CLASS(5) = 8.
CLASS(6) = 10.
```

Note: Class intervals need not be equally spaced.

2

To display previously computed histograms, load into the DAT1 array either the number of values in each class interval (bin) or the percentage-of-total value.

NPTS is the number of class intervals (histogram bars). The midpoint value of each class interval is loaded into array CLASS.

For example, the previous example would involve putting a five point histogram in DAT1 and assigning class interval midpoints as follows:

```
NPTS = 5
CLASS(1) = 1.
CLASS(2) = 3.
CLASS(3) = 5.
CLASS(4) = 7.
CLASS(5) = 9.
```

3

Similar to IFLAG=2, except two histograms can be displayed for comparison purposes. The first histogram is loaded into DAT1(NPTS,1). The second histogram is loaded into DAT1(NPTS,2). The first histogram can partially shade or obscure the second histogram by the appropriate selection of the SPAC and OVERLP options.

CLASS

Real, Input---Class interval values, dimensioned (NCLASS+1). The CLASS argument depends on the following IFLAG values:

0
: CLASS is not used.

1
: NCLASS+1 class interval endpoints are loaded into array CLASS in a monotonically increasing order. The intervals need not be of equal width.

2 or 3
: NCLASS midpoint intervals are loaded into array CLASS. They must be in monotonically increasing order, but need not be of equal widths. The histogram bars will, however, be displayed with equal widths.

NCLASS
Integer, Input---Number of class intervals (histogram bars) specified.

WRK
Real, Input---Scratch array dimensioned by NWRK.

NWRK
Integer, Input---The dimension size of array WRK determined from:

NDIM + 3 * (NCLASS + 1)

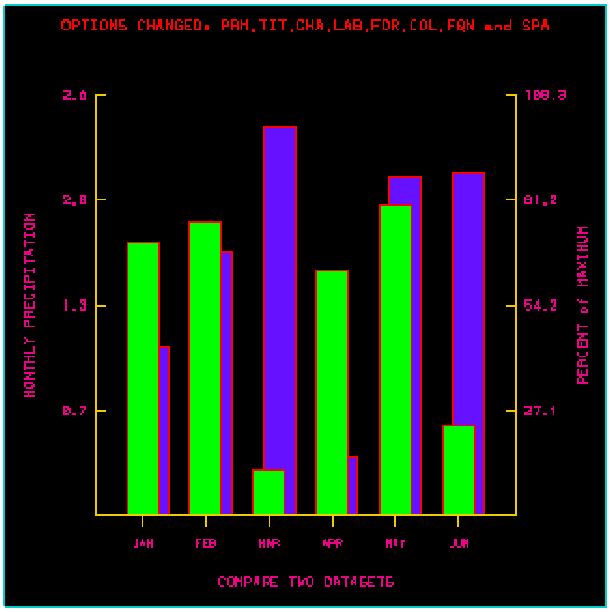
Discussion

Line 1 of code segment 1 from **thstgr.f** shows a quick way to reset all internal parameters to their default values. This can be quite useful when you are generating a sequence of histograms that include many parameter-setting calls. It allows you to return every parameter to its default value in a single call. Line 2 generates the histogram.

Comparing two histograms

The Histogram utility can be used to compare dual histograms. In the following example, two histograms, each with six class intervals, are compared.

Comparison of two histograms using overlapping bars



Code segment 2 from thstgr.f

```
1        IFLAG  = 3
2        NCLASS = 6
3        NPTS2  = 2
4        CALL HSTOPL('DEF=ON')
5        CALL HSTOPI('COL=ON',3,0,COLORS,8)
6        CALL HSTOPC('TIT=ON',
        + 'OPTIONS CHANGED: PRM,TIT,CHA,LAB,FOR,COL,FQN and SPA',7,3)
7        CALL HSTOPL('PRM=ON')
8        CALL HSTOPC('FQN=ON','MONTHLY PRECIPITATION',7,3)
9        CALL HSTOPC('FOR=ON','(F3.0)',9,3)
10       MON='JANFEBMARAPRMAYJUNJULAUGSEPOCTNOVDEC'
11       CALL HSTOPC('CHR=ON',MON,12,3)
12       CALL HSTOPC('LAB=ON','COMPARE TWO DATASETS',7,3)
13       SPAC(1)  = 2.0
14       SPAC(2)  = -1.5
15       CALL HSTOPR('SPA=ON',SPAC,2)
16       CALL HISTGR(DAT1, NDIM, NPTS2, IFLAG, CLASS, NCLASS, WORK, NWRK)
```

Discussion

Lines 1, 2, and 3 of code segment 2 from **thstgr.f** set arguments for the HISTGR call. Line 1 sets IFLAG=3 to tell HISTGR that two histograms will be displayed. Line 2 sets NCLASS=6 to specify that six midpoint intervals will be loaded into array CLASS. Line 3 sets NPTS2=2 to specify that that the second histogram is being input in array DAT1(NDIM,NPTS2).

Line 4 resets all internal parameters to their default values.

Line 5 assigns colors to the eight components of the graphic. (Specific colors are not shown since this document is not produced in color.)

Line 6 creates a main title that shows which parameters were reset for this plot.

Line 7 requests a perimeter (box) be drawn around the histogram.

Line 8 sets the frequency axis title.

Line 9 assigns an F3.0 format to class labels.

Line 10 concatenates a string of twelve 3-character alphanumeric labels representing the months of a year.

Line 11 assigns these labels to the classes (histogram bars).

Line 12 sets a class title.

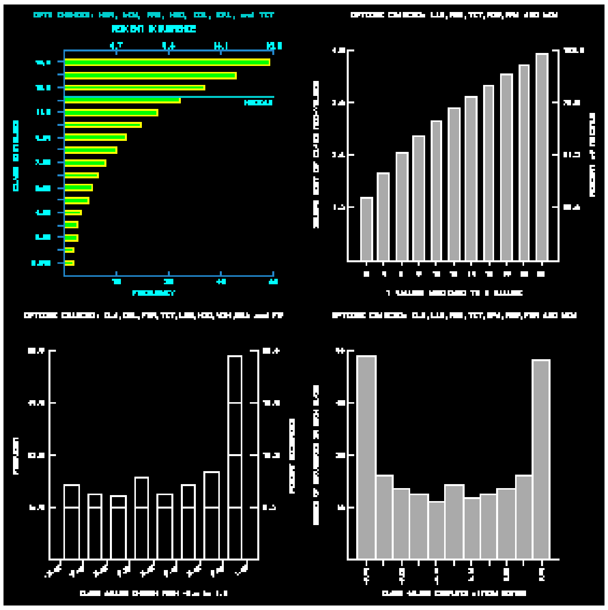
Lines 13 defines the amount of histogram bar spacing, and line 14 defines the amount by which the two histograms are to overlap. Line 15 sets the two spacing parameters.

Line 16 generates the histogram.

Four histograms on a frame

For comparisons or to save space, it is often advantageous to put more than one histogram on an output frame. To produce the following frame, HISTGR is called four times. On the first three calls, the frame is not advanced. After the fourth call, the frame is advanced.

Producing four histograms in one frame



Code segment 3 from thstgr.f

```
1       IFLAG = 0
2       NCLASS = 17
3       NPTS = 320
4       CALL HSTOPL('DEF=ON')
5       CALL HSTOPC('TIT=ON',
       + 'OPTS CHANGED: HOR, WIN, FRA, MED, COL, SPA, and TIT',9,3)
6       CALL HSTOPL('HOR=ON')
7       CALL HSTOPL('MED=ON')
8       ARR7(1) = 0.
9       ARR7(2) = .5
10      ARR7(3) = .5
11      ARR7(4) = 1.
12      CALL HSTOPR('WIN=ON',ARR7,4)
13      CALL HSTOPI('COL=ON',2,0,COLORS,8)
14      SPAC(1) = 3.0
15      SPAC(2) = 0.0
16      CALL HSTOPR('SPA=ON',SPAC,2)
17      CALL HSTOPL('FRA=OFF')
18      CALL HISTGR(DAT1, NDIM, NPTS, IFLAG, CLASS, NCLASS, WORK, NWRK)
```

Discussion

Lines 1 and 2 of code segment 3 from **thstgr.f** determine that the input dataset will be binned into 17 equal intervals.

Line 3 defines the size of the DAT1 array at 320 data values.

Line 4 resets all internal parameters to their default values.

Line 5 assigns a main title that relates which internal parameters are set for this histogram.

Line 6 requests that the histogram bars be drawn horizontally.

Line 7 requests a median line be drawn through the histogram.

Lines 8 through 12 set the location where this histogram will be drawn on a plotter frame that has a range of 0. to 1. in both dimensions. This particular call will place this histogram in the upper left quadrant.

Note: The process for putting multiple histograms on a page was to use the parameter WIN instead of using four calls to the SET routine of SPPS. This is because window and viewport scaling are done within the HISTGR routine, which would override any SET call put in the user code.

Line 13 sets colors for the graphics (not shown in the output here).

Lines 14 through 16 set the spacing between histogram bars.

Line 17 turns off the frame advance so that additional histograms can be drawn on the frame.

Line 18 generates the histogram shown in the upper left quadrant of the graphic.

Caveat: We regret the name choice of the parameter "WIN." In the terminology of GKS, this parameter should be called the viewport, not the window, since the viewport defines the plotting area on the output device.

Chapter 18: Field flows

The field flow utilities, Vectors and Streamlines, allow you to create plots representing two-dimensional field flows in a variety of styles, given arrays containing the components of the field vectors on a uniform grid. Vectors additionally provides the capability of representing a third associated scalar dataset, using a graded-color palette to color the vectors. A variety of options also exist in both utilities that allow you to control coordinate system mapping, overlay masking, and other features.

Uniform field over an azimuthal equidistant projection of the globe plotted using Vectors and Streamlines

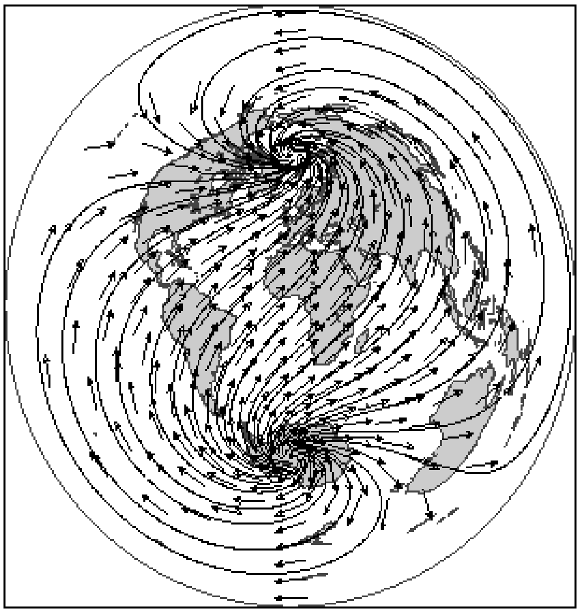


Table of Vectors user entry points

The Vectors utility has nine user entry points. In addition there are two routines not called directly from an application, but which you may modify in order to control coordinate system mapping and vector masking. This module organizes these routines by function.

Initialization routine

VVINIT

Sets up the array space and establishes the basic mapping from grid coordinate space to NDC space. Also sets up color threshold array and determines maximum and minimum vector magnitudes and scalar array values.

Drawing routine

VVECTR
Following a required call to VVINIT, draws a vector field plot.

Parameter routines

VVGETC
Gets character parameters.

VVGETI
Gets integer parameters.

VVGETR
Gets real parameters.

VVRSET
Restores all parameters to default values.

VVSETC
Sets character parameters.

VVSETI
Sets integer parameters.

VVSETR
Sets real parameters.

User-modifiable routines

VVUDMV
Decides whether to draw a vector segment based on the values of its area group identifiers.

VVUMXY
Maps the location and direction of a vector from data coordinate space to NDC space.

Table of Streamlines user entry points

The Streamlines utility has seven user entry point routines. In addition, there are four routines you do not call directly from an application, but that you may modify in order to control coordinate system mapping and streamline masking. This module lists these routines by function.

Initialization routine

STINIT
Sets up the array space and establishes the basic mapping from grid coordinate space to NDC space.

Drawing routine

STREAM
Following a required call to STINIT, draws a streamline field flow representation of a vector field on the plotting surface.

Parameter routines

STGETI
Gets integer parameters.

STGETR
Gets real parameters.

STRSET
Restores all parameters to default values.

STSETI
Sets integer parameters.

STSETR
Sets real parameters.

User-modifiable routines

STUMSL
Draws part of a streamline based on the area group identifiers of the segment.

STUMXY
Maps a point on the streamline from data coordinates to user coordinates.

STUIXY
Maps a point on the streamline from user coordinates to data coordinates.

STUMTA
Returns the directional angle of the streamline at a point with respect to NDC space.

Table of Vectors parameters

For a complete description of the Vectors parameters, see the **vectors_params man** page or the Vectors programmer document.

Parameter	Brief description	Fortran type
AMN	Arrow head MiNimum size	Real
AMX	Arrow head MaXimum size	Real
CLR	Array of GKS CoLoR indices	Integer array
CPM	ComPatibility Mode	Integer
CTV	Color Threshold Value control	Integer
DMN	NDC MiNimum vector size	Real, read-only
DMX	NDC MaXimum vector size	Real, read-only
DPF	Vector label Decimal Point control Flag	Integer
LBC	Vector LaBel Color	Integer
LBL	Vector LaBeL flag	Integer
LBS	Vector LaBel character Size	Real
LWD	Vector Line WiDth	Real
MAP	MAP transformation code	Integer
MNC	MiNimum vector text block Color	Integer
MNP	MiNimum vector text block Positioning mode	Integer
MNS	MiNimum vector text block character Size	Real
MNT	MiNimum vector Text string	Character*36
MNX	MiNimum vector text block X coordinate	Real
MNY	MiNimum vector text block Y coordinate	Real
MSK	MaSK to area map flag	Integer
MXC	MaXimum vector text block Color	Integer
MXP	MaXimum vector text block Positioning mode	Integer
MXS	MaXimum vector text block character Size	Real
MXT	MaXimum vector Text string	Character*36
MXX	MaXimum vector text block X coordinate	Real
MXY	MaXimum vector text block Y coordinate	Real
NLV	Number of colors LeVels	Integer
PAI	Parameter Array Index	Integer
PLR	PoLaR input mode	Integer
PMN	MiNimum scalar array value	Real, read-only
PMX	MaXimum scalar array value	Real
PSV	P array Special Value	Real
SET	SET call flag	Integer
SPC	SPecial Color	Integer
SVF	Special Value Flag	Integer
TRT	TRansformation Type	Integer
TVL	Array of Threshold VaLues	Real array
USV	U array Special Value	Real
VFR	Minimum Vector FRactional length	Real
VHC	Vector High Cutoff value	Real
VLC	Vector Low Cutoff value	Real
VRL	Maximum Vector Realized Length	Real
VMN	MiNimum Vector magnitude	Real, read-only
VMX	MaXimum Vector magnitude	Real, read-only
VPB	ViewPort Bottom	Real
VPL	ViewPort Left	Real
VPO	Vector POsitioning mode	Integer
VPR	ViewPort Right	Real
VPS	ViewPort Shape	Real
VPT	ViewPort Top	Real
VST	Vector STatistics output flag	Integer
VSV	V array Special Value	Real
WDB	WinDow Bottom	Real
WDL	WinDow Left	Real
WDR	WinDow Right	Real
WDT	WinDow Top	Real
XC1	X Coordinate at index 1	Real
XCM	X Coordinate at index M	Real
XIN	X-axis array INcrement (grid)	Integer

YC1	Y Coordinate at index 1	Real
YCN	Y Coordinate at index N	Real
YIN	Y-axis array INcrement (grid)	Integer
ZFC	Zero Field text block Color	Integer
ZFP	Zero Field text block Positioning mode	Integer
ZFS	Zero Field text block character Size	Real
ZFT	Zero Field Text string	Character*36
ZFX	Zero Field text block X coordinate	Real
ZFY	Zero Field text block Y coordinate	Real

Table of Streamlines parameters

For a complete description of the Streamlines parameters, see the **streamlines_params man** page or the Streamlines programmer document.

Parameter	**Brief description**	**Fortran type**
AGD	Arrow GriD increment	Integer
ARL	ARrow head Length	Real
CDS	Critical DiSplacement	Real
CKP	ChecK Progress count	Integer
CKX	ChecK (X) crossover count	Integer
CYK	CYclical data flag	Integer
CPM	ComPatibility Mode	Integer
DFM	DiFferential Magnitude	Real
LWD	Streamline Line WiDth	Real
MAP	MAP transformation code	Integer
MSK	MaSK to area map flag	Integer
PLR	PoLaR input mode	Integer
SET	SET call flag	Integer
SGD	Stream starting GriD	Integer
SSP	Streamline SPacing value	Real
SST	Streamline STatistics output	Integer
SVF	Special Value Flag	Integer
TRP	InTeRPolation method	Integer
TRT	TRansformation Type	Integer
USV	U array Special Value	Real
VNL	Vector NormaLization value	Real
VPB	ViewPort Bottom	Real
VPL	ViewPort Left	Real
VPR	ViewPort Right	Real
VPS	ViewPort Shape	Real
VPT	ViewPort Top	Real
VSV	V array Special Value	Real
WDB	WinDow Bottom	Real
WDL	WinDow Left	Real
WDR	WinDow Right	Real
WDT	WinDow Top	Real
XC1	X Coordinate at index 1	Real
XCM	X Coordinate at index M	Real
YC1	Y Coordinate at index 1	Real
YCN	Y Coordinate at index N	Real

Using parameters in Vectors and Streamlines

The parameter access routines give you control of many aspects of the field flow utilities' behavior, including coordinate mapping, masked drawing, rendering precision, special value processing, and vector coloring. This module lists the calls you have at your disposal.

Synopsis

```
CALL VVSETC (CNM, CVL)
CALL VVSETI (CNM, IVL)
CALL VVSETR (CNM, RVL)
CALL VVSETC (CNM, CVL)
CALL VVGETI (CNM, IVL)
CALL VVGETR (CNM, RVL)
CALL VVRSET
CALL STSETI (CNM, IVL)
CALL STSETR (CNM, RVL)
CALL STGETI (CNM, IVL)
CALL STGETR (CNM, RVL)
CALL STRSET
```

Routines

VVSETC
Gives a character value to a parameter.

VVSETI / STSETI
Gives an integer value to a parameter.

VVSETR / STSETR
Gives a real value to a parameter.

VVGETC
Retrieves a character value for a parameter.

VVGETI / STGETI
Retrieves integer parameter information.

VVGETR / STGETR
Retrieves real parameter information.

VVRSET / STRSET
Resets all parameters to default values.

Arguments

CNM
Character, Input---A character expression that represents a parameter name. The field flow utilities use only the first three characters in the string. Many users include additional characters that describe the parameter to make their code easier to maintain.

CVL

Character, Input or Output---A character value or character variable. Note that Vectors has character type parameters and character getting and setting routines, but Streamlines does not.

IVL

Integer, Input or Output---An integer value or integer variable.

RVL

Real, Input or Output---A real value or real variable.

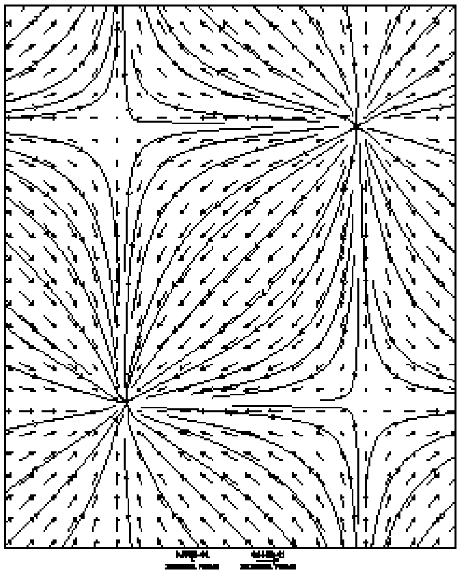
The field flow parameter access routines follow the model established by other NCAR Graphics utilities. You should be aware that Vectors has two parameters that are actually arrays. You access a particular element of the array by first setting the parameter array index, PAI, (itself a parameter) to the index of the element you want to access.

Note: To avoid a common user problem, be sure to double-check your GET and SET calls to ensure that you use the correct routine with the correct variable type. For example, do **not** use STSETI to set a real parameter value.

Creating simple field flow plots

The minimum requirements for creating a plot using either Vectors or Streamlines are much the same. First you must set up the vector field component arrays, then you call the appropriate initialization routine, and finally you call VVECTR or STREAM to draw the plot. Streamlines also requires an uninitialized work array. Using a mathematically generated flow field as an example, this section steps through the code required to render the plot, first with Vectors and then with Streamlines. The image below shows both representations, with streamlines overlaid on vectors.

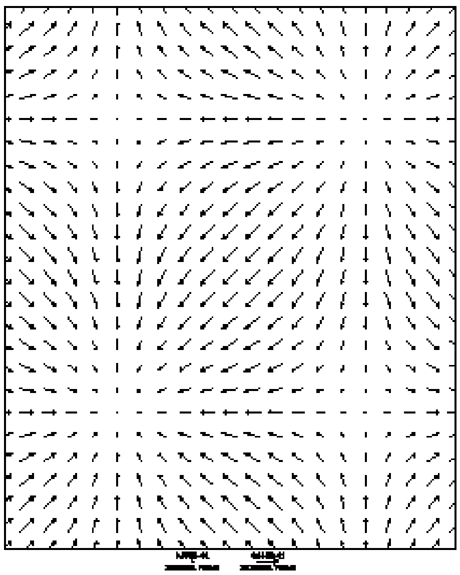
A simple field flow plot: Streamlines over Vectors



A simple plot using Vectors

Once the data arrays have been set up, call VVINIT (initialize Vectors) followed by VVECTR (draw vectors) to draw a vector field plot. By default, the plot will be shaped according to the ratio of the number of array elements along each axis. The default maximum viewport fills the plotting area, less 5% around each edge for margins. If you can dimension the component arrays exactly to the size of the field you want to plot, the VVINIT call becomes even simpler. To draw a border around your plot simply call PERIM(1,0,1,0) any time after VVINIT.

A simple Vector field plot



Code segment from ffex01.f

```
      IDM=0
      RDM=0.0
      CALL VVINIT(U,M,V,M,RDM,IDM,M,N,RDM,IDM)
      CALL VVECTR(U,V,RDM,IDM,IDM,RDM)
      CALL PERIM(1,0,1,0)
      CALL FRAME
```

Synopsis

```
CALL VVINIT (U, LU, V, LV, P, LP, M, N, WRK, LW)
CALL VVECTR (U, V, P, IAM, VVUDMV, WRK)
```

Arguments

U (LU, N)

Real 2-D array, Input---By default, assumed to contain the first dimensional Cartesian components of the vector field. However, if the PLR parameter is nonzero, it is treated as containing the vector magnitudes.

LU

Integer, Input---The first dimension of array U.

V (LV,N)

Real 2-D array, Input---By default, assumed to contain the second dimensional Cartesian components of the vector field. However, if the PLR parameter is nonzero, it is treated as containing the vector angles.

LV

Integer, Input---The first dimension of array V.

P (LP,N)

Real 2-D array, Input---Array of scalar data that may be used to color the vectors. The grid points are assumed to coincide with the grid points of the U and V arrays. This argument is used only if the CTV parameter has an absolute value of 2; otherwise, it is ignored and may be assigned a dummy value.

LP

Integer, Input---The first dimension of array P.

M

Integer, Input---Number of elements along the first dimensional axis containing data to be processed in each of the arrays, U, V, and P (if used).

N

Integer, Input---Number of elements along the second dimensional axis containing data to be processed in each of the arrays, U, V, and P (if used).

IAM(*)

Integer array, Input---Area map array previously established by calls to routines in the Areas utility. Required only if the MSK parameter is set to a nonzero value; otherwise, it is ignored and may be assigned a dummy value.

VVUDMV

External subroutine, Input---User-definable masked drawing subroutine. Required only if MSK is set to a nonzero value; otherwise, it is ignored and may be assigned a dummy value. When used, it must be declared external.

WRK(LW)

Real array, Workspace---For future enhancement. It is currently ignored and may be assigned a dummy value.

LW

Integer, Input---Assumed size of the array WRK. Not currently used, this argument should be assigned the integer value 0.

Discussion

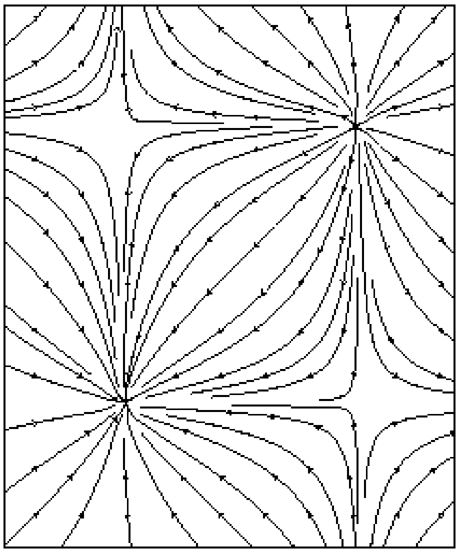
Line 1 of the **ffex01.f** code segment sets the integer dummy variable IDM to 0. Line 2 sets the real dummy variable RDM to 0.0. You could use an integer variable for all arguments given dummy values, since Vectors never examines them when they are not used, but it is probably wise to remain aware of the variable types. Lines 3 and 4, calls to VVINIT and VVECTR, do the work of the Vectors utility. Line 5 draws an outline around the plot, and line 6 advances the frame.

The arguments U, V, P, and WRK refer to the same entities in both VVINIT and VVECTR. Although there are a total of 12 arguments to both routines, if the desired field grid size matches the dimensions of the U and V arrays and you are not drawing masked vectors or using the P scalar array to color the vectors, you can ignore all but six. Of these six remaining arguments, three get the same value---the number of grid elements along the first dimension. Effectively, therefore, you need worry about only four.

A simple plot using Streamlines

The code required to create a simple Streamlines plot is very similar to what you need for a Vectors plot. Just remember than in addition to the data arrays, Streamlines also always requires the work array. Dimension the work array so that there is at least one element for each element used in both the field component arrays, or in other words, two times the input parameter M times the input parameter N. Then call STINIT (initialize Streamlines), followed by a call to STREAM to render the streamlines.

A simple Streamlines field flow plot



Code segment from ffex01.f

```
1        IDM=0
2        RDM=0.0
3        CALL STINIT(U,M,V,M,RDM,IDM,M,N,WRK,IWSIZE)
4        CALL STREAM(U,V,RDM,IDM,IDM,WRK)
5        CALL PERIM(1,0,1,0)
6        CALL FRAME
```

Synopsis

```
CALL STINIT (U, LU, V, LV, P, LP, M, N, WRK, LW)
CALL STREAM (U, V, P, IAM, STUMSL, WRK)
```

Arguments

U (LU,N)
: *Real 2-D array, Input*---By default, assumed to contain the first dimensional Cartesian components of the flow field. However, if the PLR parameter is nonzero, it is treated as containing the vector magnitudes.

LU
: *Integer, Input*---The first dimension of array U.

V (LV,N)
: *Real 2-D array, Input*---By default, assumed to contain the second dimensional Cartesian components of the flow field. However, if the PLR parameter is nonzero, it is treated as containing the vector angles.

LV
: *Integer, Input*---The first dimension of array V.

P (LP,N)
: *Real 2-D array, Input*---For future enhancement. It is currently ignored and may be assigned a dummy value.

LP
: *Integer, Input*---The first dimension of array P. Not currently used, this argument should be assigned the integer value 0.

M
: *Integer, Input*---Number of elements along the first dimensional axis containing data to be processed in each of the arrays, U and V.

N
: *Integer, Input*---Number of elements along the second dimensional axis containing data to be processed in each of the arrays, U and V.

IAM(*)
: *Integer array, Input*---Area map array previously established by calls to routines in the Areas utility. Required only if the MSK parameter is set to a nonzero value; otherwise, it is ignored and may be assigned a dummy value.

STUMSL
: *External subroutine, Input*---User-definable masked drawing subroutine. Required only if MSK is set to a nonzero value; otherwise, it is ignored and may be assigned a dummy value. When used, you must declare the name of the routine as external in the routine that calls STREAM.

WRK(LW)

Real array, Workspace---Work array used to store normalized values of the U and V vector components while the streamlines are calculated. It also holds bit flags indicating whether a grid location is eligible for starting a streamline or placing a directional arrowhead.

LW

Integer, Input---Assumed size of the array WRK. Currently, it must be greater than or equal to 2*M*N.

Discussion

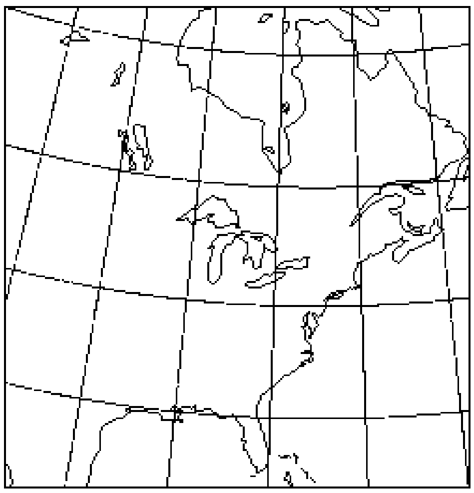
Lines 1 and 2 of the **ffex01.f** code segment assign values to the dummy variables. The calls to STINIT and STREAM in lines 3 and 4 render the streamlines. Line 5 creates a border, and line 6 advances the frame.

Again, note the similarities between Streamlines and Vectors. With the exception of the work array (ignored by Vectors, but required by Streamlines), both plots use the same declarations and field arrays. When drawing a basic plot with Streamlines, you can assign dummy values to four of the parameters.

Exploring the field flow utilities

This section looks at some of the options available to Vectors and Streamlines. Using appropriate parameter settings, you can adjust the appearance of the plot, either to enhance depiction of underlying patterns contained in the data or purely for aesthetic reasons. Other facilities exist to help you combine field flow output with other utilities in NCAR Graphics.The first four modules in this section demonstrate some of the variations obtainable by successively applying options to the same basic plot of actual U.S. National Meteorological Center forecast data displayed over an Ezmap projection. The next module demonstrates polar coordinate mapping, and the final one shows Vectors plotted over a polar stereographic map projection. For information about drawing map backgrounds, please see the *NCAR Graphics Contouring and Mapping Tutorial*.

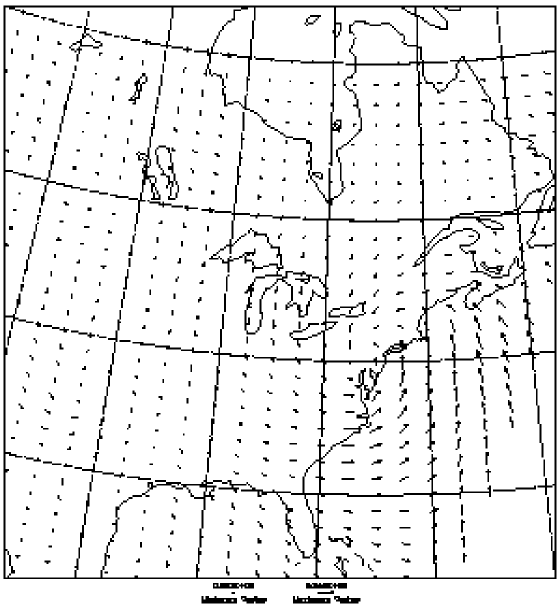
Eastern North America using lambert conical projection



Displaying vectors overlaying an Ezmap projection

In all but the simplest situations, you must know certain basic information about your field data in order to display it correctly. For this example, it is necessary to determine what special values are used to flag grid points with non-valid data. You also need to know the coordinate boundaries of the dataset to map the field correctly into the Ezmap projection. This module shows the parameter-setting calls you must make to display the example National Meteorological Center dataset.

NMC early domestic product set: Severe snowstorm, March 13, 1993



Code segment from ffex02.f

```
      CALL VVSETI('MAP -- Mapping Flag', 1)
      CALL VVSETI('SET -- Set Call Flag', 0)
      CALL VVSETR('XC1 -- Lower X Bound', -140.0)
      CALL VVSETR('XCM -- Upper X Bound', -52.5)
      CALL VVSETR('YC1 -- Lower X Bound', 20.0)
      CALL VVSETR('YCN -- Upper Y Bound', 60.0)
      CALL VVSETR('USV -- U Special Value', -9999.0)
      CALL VVSETR('VSV -- V Special Value', -9999.0)
      CALL VVSETR('PSV -- P Special Value', -9999.0)
      CALL VVSETI('SVF -- Special Value Flag', 3)
      CALL VVSETI('SPC -- P Special Color', 1)
```

Synopsis

```
CALL VVSETI ('MAP', imap)
CALL VVSETI ('SET', iset)
CALL VVSETR ('XCM', rxcm)
CALL VVSETI ('SVF', rsvf)
CALL VVSETR ('USV', rusv)
```

```
CALL VVSETI ('SPC', ispc)
```

Description

MAP

Integer---Specifies the mapping from data coordinates into user coordinate space. There are three pre-defined mappings:

0
: Identity mapping (default).

1
: Ezmap projections.

2
: Polar coordinate mapping.

Any other value greater than 0 implies a user-defined mapping that must be implemented by modifying the source code of VVUMXY. Values less than 0 are reserved.

SET

Integer---Flag specifying whether Vectors should perform its own call to the SET routine.

0
: Vectors should not perform SET call.

1
: Vectors should perform SET call (default).

XC1, XCM, YC1, YCN

Real---These four parameters define the boundaries of the data coordinate system. XC1 and YC1 specify the location of element (1,1) of each of the input data arrays, while XCM and YCN specify the location of element (M,N) of the same arrays (where M and N are the arguments to VVINIT).

SVF

Integer---The special value flag specifies how to handle special values in the U and V input data arrays.

0
: No special value processing (default).

1
: Do not draw the vector at grid points containing a special value in the U array.

2
: Do not draw the vector at grid points containing a special value in the V array.

3
: Do not draw the vector at grid points containing a special value in either the U or the V array (most generally useful).

4
: Do not draw the vector at grid points containing special values in both the U and the V array (least generally useful).

USV, VSV, PSV

Real ---Special values for the input data arrays, U, V, and P respectively (default: 1.0E12).

SPC
Integer---Controls special value processing for the P scalar data array.

<0
No special value processing (default).

0
Do not draw vectors at grid points containing a special value in the P array.

>0
Draw vectors at grid points containing a special value in the P array, using as the special color the GKS color index SPC.

Discussion

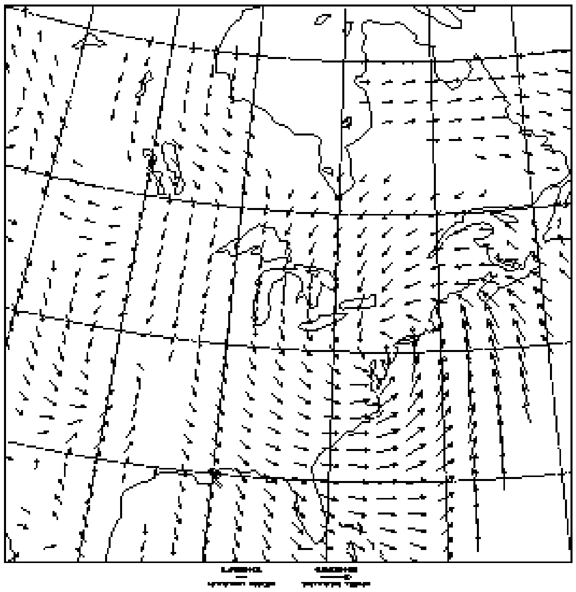
To display a vector field using an Ezmap projection, you must always set the MAP and SET parameters as in lines 1 and 2 of the **ffex02.f** code segment. You must also make calls similar to lines 3 through 6 to inform Vectors of the boundaries of your data. Note that the values you give these parameters depend on the dataset, but not on the particular Ezmap projection you use. Once you have set them correctly, you can vary the Ezmap projection parameters any way you like.

If the data include invalid data flagged by special values, you need to find out what the special values are. Line 7 through 9 set each of the special value parameters USV, VSV, and PSV to -9999.0, the value the NMC data uses to flag invalid data. Line 10 sets up special-value processing for the U and V vector component arrays. The area at the bottom right corner of the plot where no vectors appear is in fact a region where the input arrays contain special values. Special value processing for the P array is also enabled, anticipating use of the P array later.

Modifying the appearance of a Vector plot

The third frame in this example illustrates some of the facilities in the Vectors utility for adjusting the rendering of a vector field plot. The intent is to enhance the depiction of the overall flow characteristics of the field, possibly at the expense of some of the detailed information about flow speeds. Among other things, the plot eliminates low-velocity vectors, magnifies the overall vector length, and adjusts the minimum vector length to one-third the maximum.

Vector field with rendering modifications



Code segment from ffex02.f

```
      CALL VVINIT (U,M,V,M,P,M,M,N,IDM,IDM)
      IF (IFRMNO .EQ. 3) THEN
         CALL VVSETR('AMN -- Arrow Minimum Size',0.025)
         CALL VVSETR('LWD -- Vector Line Width',1.75)
         CALL VVGETR('VMN -- Minimum Vector', VMN)
         CALL VVGETR('VMX -- Maximum Vector', VMX)
         CALL VVSETR('VLC -- Vector Low Cutoff', VMN+(VMX-VMN)/8.0)
         CALL VVGETR('DMX -- Device Maximum Vector Length', DMX)
         CALL GETSET(VL,VR,VB,VT,UL,UR,UB,UT,LL)
         CALL VVSETR('VRL - Vector Realized Length',1.8*DMX/(VR-VL))
         CALL VVSETR('VFR - Vector Fractional Length',0.33)
      END IF
```

Synopsis

```
CALL VVSETR ('AMN', ramn)
CALL VVSETR ('LWD', rlwd)
CALL VVGETR ('VMN', rvmn)
CALL VVSETR ('VLC', rvlc)
CALL VVGETR ('DMX', rdmx)
```

Description

AMN, AMX

Real---Normally the size of the arrowhead on a vector is proportional to the length of the vector; this parameter lets you specify, as a fraction of the viewport width, a minimum size for the arrowhead. The default value of AMN is 0.01. A corresponding parameter, AMX, specifies a maximum size and has a default value of 0.2.

LWD

Real---Specifies the line width used to render the vector arrows. Since the appearance on the plot of a particular line width setting depends on the output device, you may need to adjust the value for different devices.

VMN, VMX

Real---After a call to VVINIT, these read-only parameters allow you to find the lowest and highest magnitude values in the vector field. Calling VVECTR causes them to be modified to contain the lowest and highest magnitudes of the vectors that actually appear in the plot.

VLC, VHC

Real---VLC, the vector low cutoff parameter, allows you to specify the minimum magnitude a vector must have to qualify for plotting. You can also specify a maximum magnitude with VHC, the corresponding vector high cutoff parameter.

DMX, DMN

Real---After calling VVINIT, you can use the read-only parameter DMX to determine the length, in NDCs, that Vectors would use by default to render the maximum magnitude vector. It is calculated as half the diagonal length of a grid box, assuming a linear mapping into user coordinates. After the call to VVECTR, DMX contains the NDC length of the longest vector actually plotted. The corresponding parameter, DMN, is only useful after the VVECTR call, when it contains the NDC length of the shortest vector actually plotted. These are the lengths in NDC space of the arrows you see in the maximum and minimum vector text blocks.

VRL

Real---VRL specifies the length (as a fraction of viewport width) you want Vectors to use for rendering the vector with the greatest magnitude. When VRL has its default value of 0.0, Vectors uses the value of DMX to determine the longest vector length. But note that VRL uses different units from DMX.

VFR

Real---Use this parameter to specify the length of the smallest vector arrow as a fraction of the length of the largest. Vectors then scales all remaining vectors proportionally. Its default value is 0.0, specifying that vector length is strictly proportional to the size of the vector.

Discussion

The calls discussed here modify attributes of the vector field rendering. Note that they appear in the code *after* the VVINIT initialization call. Many of the Vectors utility's parameters should not be modified at this stage of the processing. When in doubt, set parameters before the VVINIT call. However, for certain field attribute parameters, in particular VLC and VRL, the advantage of delaying the call is that you can first inquire certain characteristics of the datasets. Then you can use the dynamically obtained results to create more general code.

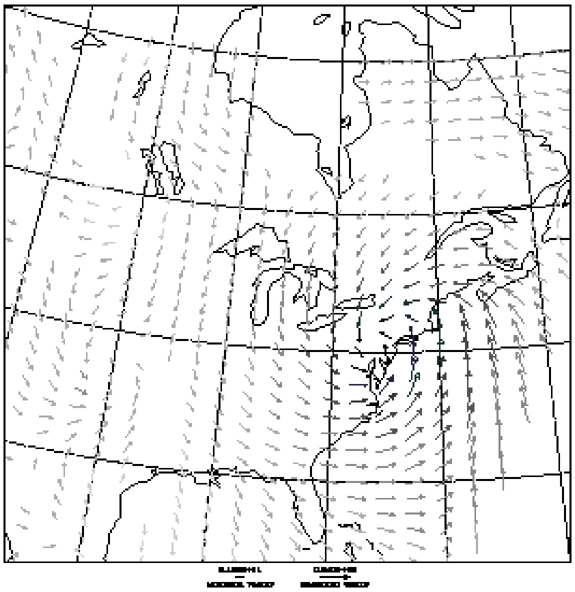
Lines 3 and 4 of the **ffex02.f** code segment modestly increase the minimum arrowhead size and the vector line width. You may notice that the effect of these changes varies depending on the plot scale and the device resolution. Lines 5 and 6 retrieve the maximum and minimum vector magnitudes from the field data. Then, based on a simple calculation using these values, the code in line 7 sets the vector low cutoff parameter, VLC, to eliminate the smallest 12.5% of the vectors from the plot.

Line 8 retrieves DMX, the default maximum vector length, and line 9 calls GETSET to find the current viewport width. Line 10 increases the vector realized length by a factor of 1.8, using a calculation that includes conversion of the default length, DMX, from NDCs to a fraction of the viewport width. Finally, line 11 sets the minimum displayed vector length to one-third the maximum length. The end result is a plot that more obviously displays the directional characteristics of the flow.

Coloring a Vector plot based on scalar array data

You can add another dimension to the information you convey through the use of the field coloring facilities provided by Vectors. The fourth frame of this example shows the same vector field data, but now surface pressure data for the same time period and grid locations is used to determine the color of each vector individually. The online example, displayed on a color workstation, shows cyan for the lowest pressure, progressing to red for the highest. In the grayscale version presented here, lows are black and highs are a light gray.

Vector field colored according to surface pressure data



Code segment from ffex02.f

```
      DATA ICLRIX / 2,16,30,44,59,73,87,102,116,130,144,159,173,187 /
      IF (IFRMNO .EQ. 4) THEN
         CALL VVSETI('CTV -- Color Thresholds Value Control', 2)
         CALL VVSETI('NLV -- Number Of Levels', NCLRS)
         DO 100 I=1,NCLRS,1
            CALL GSCR(1,ICLRIX(I),RGBV(1,I),RGBV(2,I),RGBV(3,I))
            CALL VVSETI('PAI -- Parameter Array Index', I)
            CALL VVSETI('CLR -- GKS Color Index', ICLRIX(I))
 100     CONTINUE
      END IF
      CALL VVINIT (U,MSIZE,V,MSIZE,P,MSIZE,MSIZE,NSIZE,IDM,IDM)
```

Synopsis

```
CALL VVSETI ('CTV', ictv)
CALL VVSETI ('NLV', inlv)
CALL VVSETI ('PAI', ipai)
CALL VVSETI ('CLR', iclr)
CALL VVSETI ('TVL', itvl)
```

Description

CTV

Integer---Use the color threshold value control parameter to enable vector coloring.

-2
: Color vectors based on scalar array data. The Vectors utility will not alter threshold value data in the TVL parameter array.

-1
: Color vectors based on vector magnitude. Vectors will not alter threshold value data.

0
: No individual coloring of vectors. All vectors are given the color specified by the current GKS polyline color index (default).

1
: Color vectors based on vector magnitude. VVINIT assigns values to the first NLV elements of the threshold array TVL.

2
: Color vectors based on scalar array data. VVINIT assigns values to the first NLV elements of the threshold array TVL.

NLV

Integer---Specifies the number of levels used to color the vectors. Whenever the CTV parameter has a nonzero value enabling vector coloring, you need to define NLV elements of the CLR parameter array to appropriate GKS color index values. In addition, if CTV is negative, you are responsible for ensuring that the threshold value array, TVL, contains valid values. The maximum value you can give to NLV currently is 64.

PAI

Integer---Before setting or retrieving a parameter array element, you must first set PAI, the parameter array index parameter, to the array index of the element you want to access. In Vectors, there are currently two parameter arrays, CLR and TVL.

CLR

Integer array---Parameter array containing valid GKS color index values. Unless your output device has a fixed non-writable color table, you are also responsible for calling GSCR to set the GKS color representation for each color index you use. See the "Color tables and color mapping" chapter for more information on setting up color tables.

TVL

Integer array---Contains the threshold values that determine which color is assigned to a particular vector. Vectors looks for the first element of TVL greater than or equal to the vector's associated value (either from the scalar array or a magnitude). It then determines the GKS color index to use from the value stored in the CLR array element with the same index. When CTV is less than zero, you are responsible for setting up the TVL array with NLV values in ascending order starting from the first element of the array.

Discussion

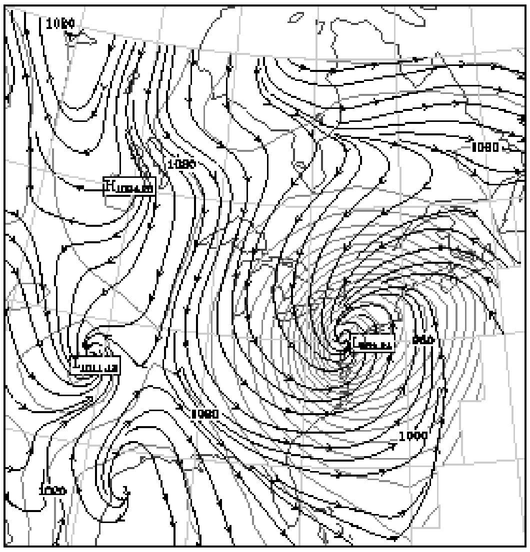
You must set parameters CTV, the color threshold value control, and NLV, the number of levels, as a prerequisite for coloring the vector field. When CTV has a positive value, VVINIT sets up the threshold value array to contain a number (specified by NLV) of linearly spaced values. If CTV is negative, VVINIT does not touch the contents of TVL. If you want to look at more than one set of data using the same color scale, you can choose one dataset, preferably the one with the widest domain, as a reference. You allow VVINIT to choose the threshold values for this set only; then you set CTV negative to tell Vectors to use the same scaling for subsequent datasets. On the other hand, if the default linear threshold value setup is inadequate for your purposes, you can set CTV negative immediately to take complete control of the threshold array values.

However you handle the threshold array, you are always responsible for defining the colors you want to use and setting up the color parameter array, CLR. The DATA statement in line 1 of the **ffex02.f** code segment uses a trick of spacing out the color indices between the values 2 and 187 to allow the same code to work for a workstation with settable colors as well as for the fixed color table grayscale PostScript device. This device has a color range from index 1 for black and 255 for white. The GKS GSCR call for setting the RGB color representation is simply ignored for such a device.

Masked Streamlines over contour lines and Ezmap

This module uses the same data as the previous plots, but presents a different picture. The map projection is identical, but now surface pressure is represented as a contour plot using the Conpack utility and Streamlines draws the flow field. On a color workstation, the plot uses color to distinguish the various kinds of lines. In the gray scale version shown here, shades of gray and line thickness are used. To illustrate the masked drawing capabilities of Streamlines, the example masks the streamlines to the contour line and high/low labels drawn by Conpack.

Streamlines representing wind flow over contour plot of surface pressure



Code segment from ffex03.f

```
      EXTERNAL STUMSL
      CALL CPRECT (P,MSIZE,MSIZE,NSIZE,RWRK,ICPRWL,IWRK,ICPIWL)
      CALL ARINAM (IAM,IAREAL)
      CALL CPLBAM (P,RWRK,IWRK,IAM)
      CALL STSETI ('MSK -- Streamline Masking',1)
      CALL STSETR ('SSP -- Stream Spacing', 0.012)
      CALL STSETR ('DFM -- Differential Magnitude', 0.012)
      CALL STSETI ('SST -- Streamline Statistics', 1)
      CALL STINIT (U,MSIZE,V,MSIZE,IDM,IDM,MSIZE,NSIZE,WRK,ISTWKL)
      CALL STREAM (U,V,IDM,IAM,STUMSL,WRK)
```

Synopsis

```
CALL STSETI ('MSK', imsk)
CALL STSETR ('SSP', rssp)
CALL STSETR ('DFM', rdfm)
CALL STSETI ('SST', isst)
```

Description

MSK

Integer--- Flag indicating whether streamlines should be drawn masked to an area map established by the Areas utility.

0

Do not draw streamlines masked (default).

1

Draw streamlines masked to area map.

SSP

Real---The streamline spacing parameter establishes the minimum distance (as a fraction of the viewport width) a streamline in progress is allowed to approach existing streamlines before being terminated. In general, giving this value a larger number increases the distance between streamlines, and has a tendency to create more, but shorter streamlines. The spacing is only checked at intervals, so streamlines sometimes approach closer than the specified distance. The checking frequency is adjustable using the streamline crossover checking parameter, CKX. The streamline starting grid increment parameter, SGD, also affects the overall streamline density. The default value of SSP is 0.015.

DFM

Real---The streamline differential magnitude parameter specifies, as a fraction of the viewport width, the step size used to calculate each incremental piece of the streamline. As DFM becomes smaller, the resolution of the plot increases, but so does the amount of time Streamlines uses to process the field data. Memory requirements do not change. The default value of DFM is 0.02.

SST

Integer---Flag specifying whether Streamlines should output to the default output unit a message summarizing the streamline statistics. (Vectors has a similar facility accessed through the parameter VST.)

0

Do not output Streamlines statistics (default).

1

Output Streamlines statistics.

Discussion

Line 1 of the **ffex03.f** code segment declares as external the user-definable routine called eventually from an Areas utility routine to do the streamline drawing. Streamlines provides a basic default version of this routine that is adequate for this example. For more complicated masking, you would use identical techniques for modifying the routine as for the analogous routines required by the Ezmap masked grid routine, MAPGRM, or the Conpack masked contour routine, CPCLDM, both of which are discussed in the *NCAR Graphics Contouring and Mapping Tutorial*. Lines 2 through 4 call the Areas and Conpack routines required to create an area map containing information about the contour labels. After these calls, even if the code that draws the labels were omitted, Streamlines would leave holes where the labels belong.

Line 5 sets the parameter MSK to 1, telling Streamlines to perform masking. This call has the effect of causing STREAM to inspect the contents of its IAM area map array argument and return an error if it is judged invalid.

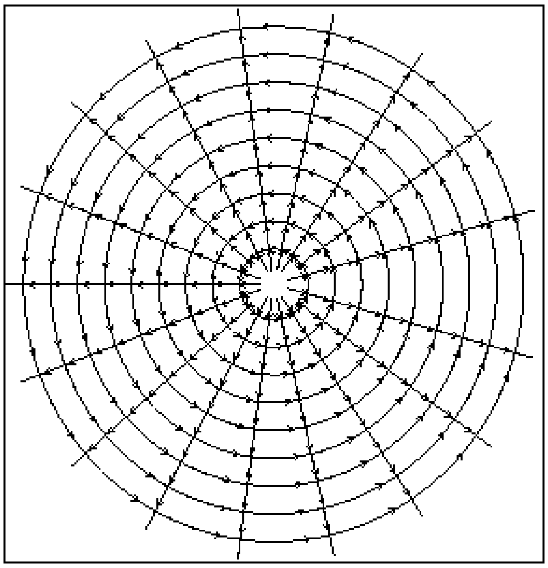
Lines 6, 7, and 8 are not required for this plot, but you should be aware of the effect of these parameters. If the streamlines seem to be broken into too many pieces, a smaller value of SSP, the streamline spacing parameter, may help. If the rendering is not smooth enough, try a smaller value for the differential magnitude, DFM. Turning on the statistics output parameter, SST, results in output showing the total number of differential steps used to compute all the streamlines in the plot. This number may help you determine the best compromise between plot quality and computing time used.

Note that the masking capabilities of Vectors parallel those of Streamlines. Study the Ezmap and Conpack sections of the *NCAR Graphics Contouring and Mapping Tutorial* to learn about area maps and masked drawing of contour plots. The initial parameter-setting calls for this example---MAP, SET, data coordinate boundary parameters, and special values---are very similar to the setup calls for Vectors described in the preceding module "Displaying Vectors overlaying an Ezmap projection," except that the special value flag, SVF, for Streamlines has only two values, off and on. If you look at the online example, compare also the code that performs the equivalent duties for Conpack. Note that you must be sure to set the Conpack out-of-range parameter, ORV, when overlaying using an Ezmap projection.

Mapping to polar coordinates in Streamlines

The graphic presented in this module illustrates how Streamlines performs polar coordinate mapping. Vectors uses the same model. The code calls STREAM twice. The first time the U array components are all set to 1.0, while the V array has all zero values. The second call reverses the situation. The straight radial lines with arrows pointing outwards from the center comprise the first field; the concentric circles with arrows pointing counterclockwise, the second. Therefore, the U array contains components parallel to the radius, while the V array components are perpendicular to the radius.

Uniform orthogonal polar fields plotted using Streamlines



Code segment from ffex04.f

```
      CALL STSETI('MAP -- Mapping Mode', 2)
      CALL STSETR('WDL -- Window Left', -20.0)
      CALL STSETR('WDR -- Window Right', 20.0)
      CALL STSETR('WDB -- Window Bottom', -20.0)
      CALL STSETR('WDT -- Window Top', 20.0)
      CALL STSETR('XC1 -- Lower X Bound', 1.0)
      CALL STSETR('XCM -- Upper X Bound', 20.0)
      CALL STSETR('YC1 -- Lower Y Bound', 0.0)
      CALL STSETR('YCN -- Upper Y Bound', 360.0)
      DO 1000 K=1,0,-1
         CALL STSETI('TRT -- Transform Type', K)
```

Synopsis

```
CALL STSETI ('MAP', imap)
CALL STSETR ('WDL', rwdl)
CALL STSETR ('XC1', rsc1)
CALL STSETI ('TRT', itrt)
```

Description

MAP

Integer---Specifies the mapping from data coordinates into user coordinate space. There are three pre-defined mappings:

0
: Identity mapping (default).

1
: Ezmap projections.

2
: Polar coordinate mapping.

Any other value greater than 0 implies a user-defined mapping that must be implemented by modifying the source code of VVUMXY. Values less than 0 are reserved.

WDL, WDR, WDB, WDT

Real---These four parameters define the boundaries of the window (or user coordinate system). WDL and WDR specify the left and right edges of the window, while WDB and WDT specify the bottom and top edges of the window. They refer to the same entities as arguments 5, 6, 7, and 8 of the SET call.

XC1, XCM, YC1, YCN

Real---These parameters define the boundaries of the data coordinate system. In other words, XC1 and YC1 specify the location of element (1,1) of each of the input data arrays, while XCM and YCN specify the location of element (M,N) of the same arrays (where M and N are the arguments to VVINIT). When the MAP parameter specifies an Ezmap projection, the X values are treated as longitude and the Y values as latitude. For the polar mapping, the X values are treated as radius, and the Y values as angle.

TRT

Integer---If the MAP parameter has a value of 0 or 2, the transformation type parameter qualifies the mapping, as follows:

0
: Only the location of the each grid point is mapped. The directional component of the field arrays is calculated relative to an underlying rectangular coordinate system.

1
: Both the location and the directional component are mapped (default).

Discussion

Line l of the **ffex04.f** code segment sets the mapping mode to 2, specifying polar coordinate mapping. Be sure not to confuse the polar coordinate mapping mode with the polar input mode parameter, PLR.

Lines 2 through 5 define the window (that is, the boundaries of the user coordinate space) by setting each of the four window boundary parameters. For this example, the SET parameter has been left in its default state, with the result that STINIT will do a call to the SET routine. As an alternative, you could accomplish the same goal by calling SET yourself, using the values given to the window boundary parameters as arguments 5 through 8 of the SET call.

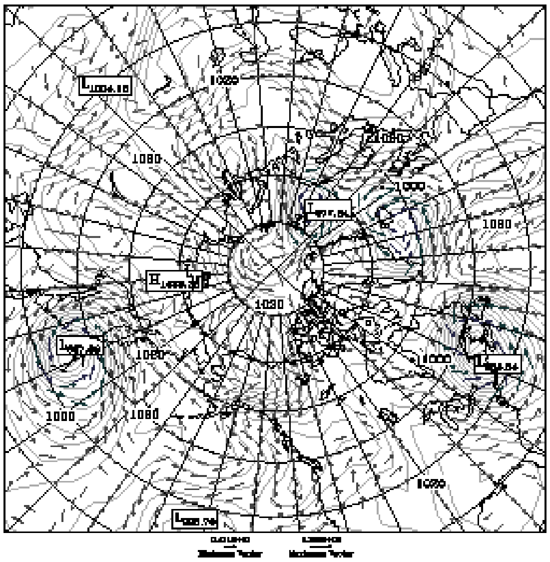
The values you use for the window boundary need to be based on some understanding of the mapping from data coordinates to the user coordinate system. When polar coordinate mapping is in effect, the outer extent of the radius combined with the angle it sweeps determines the amount of user coordinate space required. From looking at lines 6 through 9, you can see that the outer extent of the radius, defined by the parameter XCM, is 20; the angle going from YC1 to YCN is 360.0 to 0.0---in other words a complete circle. Therefore, the user space must extend a minimum of 20 units in every direction from the origin for all the data to be visible. The conclusion is that the window boundaries should go from -20 to 20 along both axes.

The effect of the non-default setting of the TRT parameter is not shown in the graphic displayed here, but you can see what it does by looking at the second frame of the online example. When TRT has a value of 0, only the location of the grid points are mapped into the polar coordinate system. The vector components themselves are processed relative to an underlying rectangular coordinate system. For the uniform fields in the example, the resulting streamlines are series of horizontal and vertical straight lines.

A polar plot using Vectors, Conpack, and Ezmap

The final field flow example shows a vector field overlaying a contour plot on an Ezmap polar stereographic projection. In the lower right corner of the plot appears the same storm system as in previous examples. Here it is seen in the context of higher latitude northern weather around the globe. Although most of the code is very similar to previous examples, there is one new difficulty to resolve---with this kind of projection, the data become very crowded near the poles. Although neither Vectors nor Streamlines has yet implemented a general solution to the problem, the code presented here is a relatively easy workaround for plots like this.

Polar stereographic plot using NMC global product set data, March 13, 1993



Code segment from ffex05.f

```
1        PARAMETER (NROWS = 11)
2        DIMENSION ITHIN(NROWS)
3        DATA ITHIN /90,15,5,5,4,4,3,3,2,2,2/
4        DO 50 J=NSIZE,NSIZE-NROWS+1,-1
5           DO 49 I=1,MSIZE
6               IF (MOD(I,ITHIN(NSIZE-J+1)).NE.0) THEN
7                   U(I,J) = -9999.0
8               ENDIF
9    49     CONTINUE
10   50  CONTINUE
```

Discussion

The code used to generate the **ffex05** example was created for the most part by combining pieces of examples **ffex02** (the Vectors and Ezmap plot) and **ffex03** (the Streamlines, Conpack, and Ezmap plot). Only a few changes are required to place the data properly within the projection. You must set up the map projection and modify the data array sizes and the data coordinate boundaries to accommodate the global dataset. The remaining changes are only to improve the clarity and appearance of the plot. However, note that when a plot has as many levels of overlay as this example, you may need to experiment for a while to avoid obscuring important details, especially when rendering to a grayscale device.

In a polar projection such as this one, the data grid becomes progressively more crowded as the pole is approached. However, unlike some other projections, the crowding is uniform for each latitudinal row of the grid. This fact makes possible a relatively simple method of removing the excess vectors.

Line 1 of the **ffex05.f** code segment sets the parameter, NROWS, to the number of rows in the plot determined by inspection to be too crowded. Line 2 dimensions an array called ITHIN to the value of this parameter, and then line 3 initializes this array with empirically determined data that are later used in the MOD function of line 6 to weed out all but a fraction of the vectors in the row. Knowing that the last row of this array represents 90 degrees north, line 4 begins a backwards loop through the top NROWS rows of the data. The inner loop of lines 5 through 9 sets each component of the U array whose column index is not evenly divisible by the appropriate value from the ITHIN array to the special value associated with this dataset: -9999.0. The result, for the affected rows, is that only a fraction of the vectors appear: if the ITHIN element is 3, for example, only a third of the vectors, evenly spaced along the row, will show up.

Of course this is an ad hoc approach, and will not work for situations where the crowding is not uniform within a row of the grid. You should also be aware that it is not suitable for use with the Streamlines utility. However, in the situations where it can be used, it is easy to implement and only requires a modest amount of experimentation to adjust the thinning array data suitably.

Chapter 19: Drawing 3-D objects

The Threed utility allows you to define a perspective transformation (a projection) from a three-dimensional space to a two-dimensional space and then to draw the perspective images of objects (points, lines, curves, perimeters, and character strings) defined in three dimensions.

Note: Threed has been essentially unchanged since the early 1970s. There are plans to replace it with a new package that has a more consistent and modern interface. This new package will unify the functionality of all three of the utilities Threed, Surface, and Isosurface.

A simple Threed plot

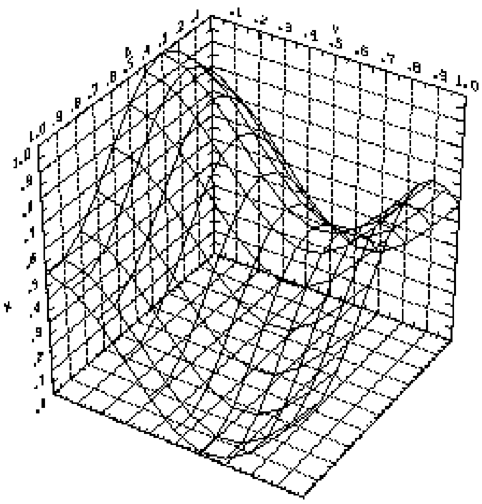


Table of Threed user entry points

The Threed utility has ten user entry points. This module lists the routines in functional groups.

The initialization routine

SET3
: Defines the transformation from three dimensions to two dimensions.

Object-drawing routines

CURVE3
: Draws the projection of a curve defined by a sequence of points in 3-space.

LINE3
: Draws the projection of a straight-line segment.

FENCE3
: Draws the projection of a curve, just as CURVE3 does, but with a "fence" of straight line segments that are parallel to one of the three axes and that extend to a specified plane perpendicular to that axis.

FRST3
: Specifies the first in a sequence of points connected by straight-line segments, the projections of which are to be drawn. (FRST3 is called for the first point in the sequence and VECT3 is called for the rest.)

PERIM3
: Draws the projection of a rectangular perimeter in a plane parallel to one of the three coordinate planes, with inward-pointing tick marks at specified intervals.

POINT3
: Draws the projection of a point.

PWRZT
: Draws the projection of character strings that are positioned in a plane parallel to one of the three coordinate planes.

VECT3
: Defines the second and all subsequent points in a sequence of points connected by straight-line segments, the projections of which are to be drawn. (FRST3 is called for the first point in the sequence and VECT3 is called for the rest.)

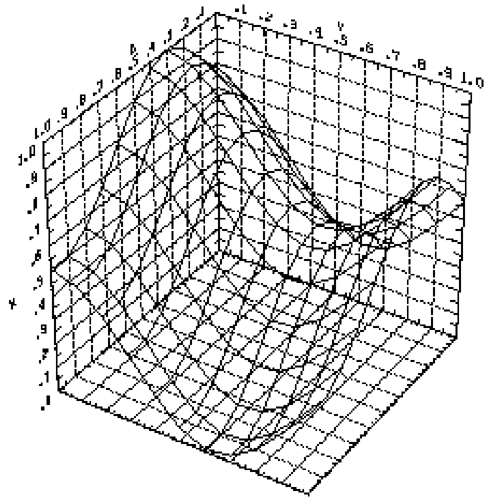
Parameter-access routine

TICK43
: Gives user control of tick mark length in PERIM3.

Defining a 3-D to 2-D transformation

The routine SET3 is called to define the 3-D to 2-D transformation to be used.

A simple Threed plot



Code segment from fthex01.f

```
      CALL SET3 (.1,.9,.1,.9,UMIN,UMAX,VMIN,VMAX,WMIN,WMAX,PEYE)
```

Synopsis

```
      CALL SET3 (XMIN, XMAX, YMIN, YMAX, UMIN, UMAX, VMIN, VMAX, WMIN,
     +          WMAX, PEYE)
```

Arguments

XMIN, XMAX, YMIN, YMAX

Real, Input---Fractional coordinates (in NDCs) defining the portion of the plotter frame (the "viewport") into which 3-D objects are to be projected. Each has a value between 0. and 1., inclusive. Examples: To use the whole plotter frame, set XMIN=0., XMAX=1., YMIN=0., and YMAX=1.; to use the upper left quarter of the plotter frame, set XMIN=0., XMAX=.5, YMIN=.5, and YMAX=1.

UMIN, UMAX, VMIN, VMAX, WMIN, WMAX

Real, Input---The limiting coordinates, in a 3-D coordinate space of the user's choosing, of the rectangular box that is to be projected. This box normally contains all of the objects whose projections are to be drawn.

PEYE

Real array, Input---An array of length three, specifying the coordinates, in the user's 3-D coordinate system, of the eye position. The point (PEYE(1),PEYE(2),PEYE(3)) must be outside the box defined by UMIN, UMAX, VMIN, VMAX, WMIN, and WMAX. Objects will be drawn as they appear from this point.

Discussion

The call to SET3 says what portion of the 3-D coordinate system is to be projected, from what point it is to be projected, and where the resulting picture is to be placed on the plotter frame.

The array PEYE specifies the position of the eye, which may be referred to as the "viewpoint" or "center" of the projection.

The "line of sight" runs from the point

(PEYE(1),PEYE(2),PEYE(3))

(the eye position) to the point

(UCEN, VCEN, WCEN)

(the point at the center of the projected box---UCEN=.5*(UMIN+UMAX), VCEN=.5*(VMIN+VMAX), and WCEN=.5*(VMIN+VMAX))

The "image plane" is perpendicular to the line of sight and may be thought of as passing through the point at the center of the projected box. The projection of an arbitrary point P in 3-space is the point where the straight line from the eye position to P passes through the image plane. The projection of a more complicated object is just the union of the projections of its constituent points.

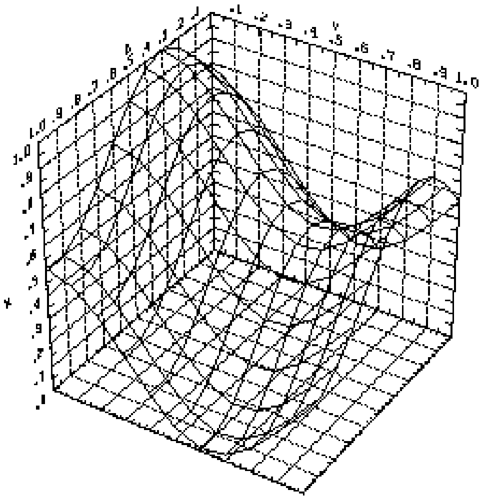
The projection in the image plane of the box specified by the arguments UMIN, UMAX, VMIN, VMAX, WMIN, and WMAX will be scaled to fit in the viewport specified by the arguments XMIN, XMAX, YMIN, and YMAX. The projected box will retain its natural aspect ratio; typically, it will just touch the left and right edges of the viewport, or the top and bottom of the viewport, but not both.

The image of the box will be rotated so that the projection of a unit vector based at the center of the box and pointing in the direction of the positive W axis will be made to point upward in the viewport. However, if the line of sight is very nearly parallel to the W axis, so that the projection of this unit vector shrinks almost to a point, then a unit vector based at the center of the box and pointing in the direction of the positive V axis will be made to point upward in the viewport instead. One implication of this is that, if you are making a movie by letting the eye "fly around" the objects being projected and it gets too close to being directly above or below the center of the box, there may be a sudden disconcerting rotation of the projected image.

Drawing a perimeter

The routine PERIM3 is used to draw a perimeter in a plane perpendicular to one of the three axes in the 3-D coordinate system.

A simple Threed plot



Code segment from fthex01.f

```
      CALL PERIM3 (10,2,10,2,1,0.)
      CALL PERIM3 (10,2,10,2,2,0.)
      CALL PERIM3 (10,4,10,4,3,0.)
```

Synopsis

```
CALL PERIM3 (MAJOR1, MINOR1, MAJOR2, MINOR2, IAXIS, VALUE)
```

Arguments

MAJOR1

Integer, Input---Specifies the number of major divisions of those sides of the perimeter parallel to direction 1. (Directions 1 and 2 are defined by the value of IAXIS.) Major divisions are indicated by tick marks.

MINOR1

Integer, Input---Specifies the number of minor divisions per major division along those sides of the perimeter parallel to direction 1. (Directions 1 and 2 are defined by the value of IAXIS.) Minor divisions are indicated by tick marks that, by default, are smaller than the tick marks indicating major divisions.

MAJOR2

Integer, Input---Specifies the number of major divisions of those sides of the perimeter parallel to direction 2. (Directions 1 and 2 are defined by the value of IAXIS.)

MINOR2

Integer, Input---Specifies the number of minor divisions per major division along those sides of the perimeter parallel to direction 2. (Directions 1 and 2 are defined by the value of IAXIS.)

IAXIS

Integer, Input---Specifies which of the three axes the perimeter is to be perpendicular to and identifies directions 1 and 2:

1
: The perimeter is to be perpendicular to the U axis. Direction 1 is parallel to the V axis and direction 2 is parallel to the W axis.

2
: The perimeter is to be perpendicular to the V axis. Direction 1 is parallel to the U axis and direction 2 is parallel to the W axis.

3
: The perimeter is to be perpendicular to the W axis. Direction 1 is parallel to the U axis and direction 2 is parallel to the V axis.

VALUE
: *Real, Input*---The value at which the plane of the perimeter is to intersect the axis specified by IAXIS. If IAXIS is a 1, VALUE is a U coordinate; if IAXIS is a 2, VALUE is a V coordinate; and, if IAXIS is a 3, VALUE is a W coordinate.

Discussion

Each call to PERIM3 draws a perimeter: a rectangle representing the intersection of a plane perpendicular to one of the three axes with the box specified by the arguments UMIN, UMAX, VMIN, VMAX, WMIN, and WMAX in the last call to SET3. The perimeter can be placed at any desired position along the axis to which it is perpendicular; it can have inward-pointing major and minor tick marks along its edges, spaced as specified by the user. Tick marks along two parallel edges of the perimeter are controlled by one pair of arguments, and tick marks along the other two parallel edges are controlled by the other pair of arguments.

Each of the arguments MAJOR1 and MAJOR2 refers to the number of major divisions (spaces between ticks) rather than to the number of major ticks; counting those at the ends, there will be one more major tick than there are major divisions.

Each of the arguments MINOR1 and MINOR2 refers to the number of minor divisions (spaces between ticks) rather than to the number of minor ticks; in a given major interval, there will be one fewer minor tick than there are minor divisions.

The **fthex01** example shows three different perimeters, one for each coordinate plane; the perimeters are placed on the far sides of the box. The rectangle itself is drawn in the current polyline color, as determined by the last call to the GKS routine GSPLCI; by default, color index 1 is used. Major ticks and minor ticks are drawn in the colors implied by the values of the variables ITHRMJ and ITHRMN in the Threed COMMON block

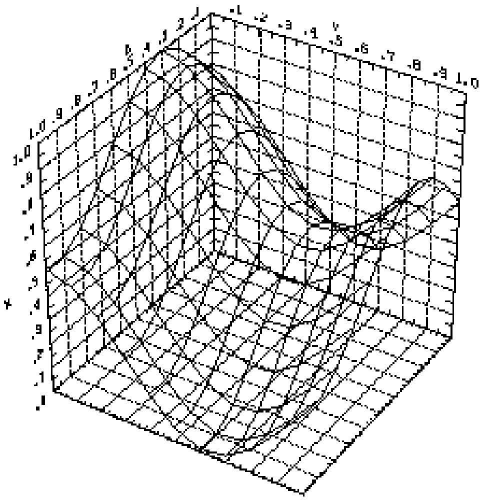
```
COMMON /THRINT/ ITHRMJ,ITHRMN,ITHRTX
```

The default value of both ITHRMJ and ITHRMN is 1; user code may change these values. All lines are drawn using the current GKS line width scale factor, as set by the last call to the GKS routine GSLWSC.

Changing the lengths of tick marks on a perimeter

The routine TICK43 is used to set internal parameters specifying the lengths of the ticks on perimeters drawn by subsequent calls to PERIM3.

A simple Threed plot



Code segment from fthex01.f

```
      CALL TICK43 (12,8,24,16,48,32)
```

Synopsis

```
CALL TICK43 (MAJORU, MINORU, MAJORV, MINORV, MAJORW, MINORW)
```

Arguments

MAJORU
: *Integer, Input*---Specifies the length of major ticks to be used on perimeter sides that are parallel to the U axis.

MINORU
: *Integer, Input*---Specifies the length of minor ticks to be used on perimeter sides that are parallel to the U axis.

MAJORV, MINORV
: *Integer, Input*---Specify the lengths of major and minor ticks for perimeter sides parallel to the V axis.

MAJORW, MINORW
: *Integer, Input*---Specify the lengths of major and minor ticks for perimeter sides parallel to the W axis.

Discussion

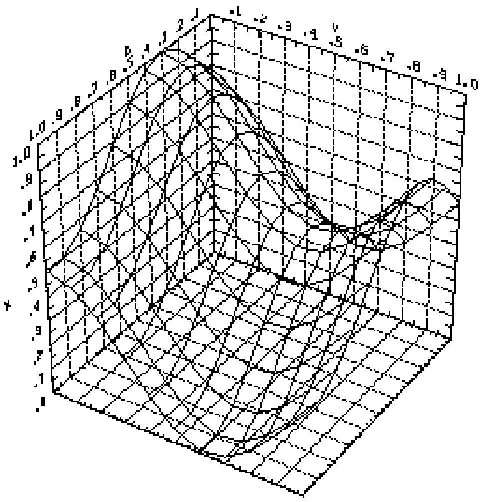
All tick lengths are specified as integers between 0 and 1024. The value *n* produces a tick that is *n*/1024ths as long as the longest side of the box defined by the arguments UMIN, UMAX, VMIN, VMAX, WMIN, and WMAX in the last call to SET3. The default values are 12 for a major tick and 8 for a minor tick.

The **fthex01** example uses default-size tick marks on perimeter sides parallel to the U axis, double-size tick marks on perimeter sides parallel to the V axis, and quadruple-size tick marks on perimeter sides parallel to the W axis.

Drawing points

The routine POINT3 is called to draw the projections of points in 3-space.

A simple Threed plot



Code segment from fthex01.f

```
    DO 108 I=1,11
       PTMP=REAL(I-1)/10.
       DO 107 J=1,101
          QTMP=REAL(J-1)/100.
          CALL POINT3 (PTMP,QTMP,0.)
          CALL POINT3 (QTMP,PTMP,0.)
          CALL POINT3 (PTMP,0.,QTMP)
          CALL POINT3 (QTMP,0.,PTMP)
          CALL POINT3 (0.,PTMP,QTMP)
          CALL POINT3 (0.,QTMP,PTMP)
107    CONTINUE
108 CONTINUE
```

Synopsis

```
      CALL POINT3 (UPOS, VPOS, WPOS)
```

Arguments

UPOS, VPOS, WPOS

Real, Input---The U, V, and W coordinates of a point whose projection is to be drawn.

Discussion

The **fthex01.f** code segment above uses the routine POINT3 to draw grids in each of the three perimeters. While this produces a nice effect on most devices, it does add considerably to the size of the metafile produced and may be rather slow to display on some devices.

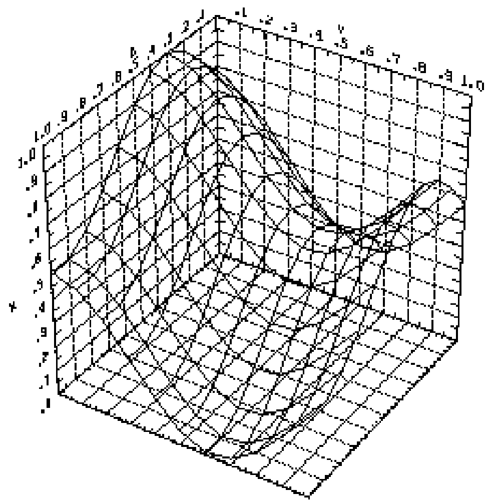
Each point is drawn by calling the GKS routine GPM to draw a polymarker of type 1. Points are therefore drawn in the current polymarker color, as determined by the last call to the GKS routine GSPMCI; by default, color index 1 is the one used.

Calling the GKS routine GSMKSC to change the polymarker size does not normally affect polymarkers of type 1, so there is no good way to make the points bigger; if you want to do this, you should use LINE3 (or perhaps FRST3 and VECT3) instead of POINT3 to draw an object of the desired size.

Drawing straight-line segments and curves with FRST3 and VECT3

The routines FRST3 and VECT3 are used to draw the projections of straight-line segments in 3-space, using a "pen-move" paradigm. A curve may be approximated by drawing a sequence of connected straight-line segments.

A simple Threed plot



Code segment from fthex01.f

```
      WFUN(U,V)=.5+.25*SIN(5.*U)+.25*COS(5.*V)
```

```
2       CALL PLOTIF (0.,0.,2)
3       CALL GSLWSC (2.)
4       DO 110 I=1,11
5          UTMP=REAL(I-1)/10.
6          CALL FRST3 (UTMP,0.,WFUN(UTMP,0.))
7          DO 109 J=2,11
8                 VTMP=REAL(J-1)/10.
9                 CALL VECT3 (UTMP,VTMP,WFUN(UTMP,VTMP))
10 109     CONTINUE
11 110  CONTINUE
12      DO 112 J=1,11
13          VTMP=REAL(J-1)/10.
14          CALL FRST3 (0.,VTMP,WFUN(0.,VTMP))
15          DO 111 I=2,11
16                 UTMP=REAL(I-1)/10.
17                 CALL VECT3 (UTMP,VTMP,WFUN(UTMP,VTMP))
18 111      CONTINUE
19 112  CONTINUE
```

Synopsis

```
      CALL FRST3 (UPOS, VPOS, WPOS)
      CALL VECT3 (UPOS, VPOS, WPOS)
```

Arguments

UPOS, VPOS, WPOS

Real, Input---The U, V, and W coordinates of a point in 3-space.

Discussion

The **fthex01.f** code segment uses FRST3 and VECT3 to draw a simple wire-frame representation of the surface defined by the arithmetic statement function on line 1. Line 2 flushes the SPPS polyline buffer, so that lines waiting in that buffer to be drawn will not be affected by the change in line width that is requested by the GKS call on line 3. Lines 4 through 11 draw slices through the function parallel to the V/W plane. Lines 12 through 19 draw slices through the function parallel to the U/W plane.

Each call to FRST3 moves a conceptual "pen" to the new starting point for a sequence of line draws. Each call to VECT3 draws a line from the current pen position to a new pen position and then makes that the current pen position.

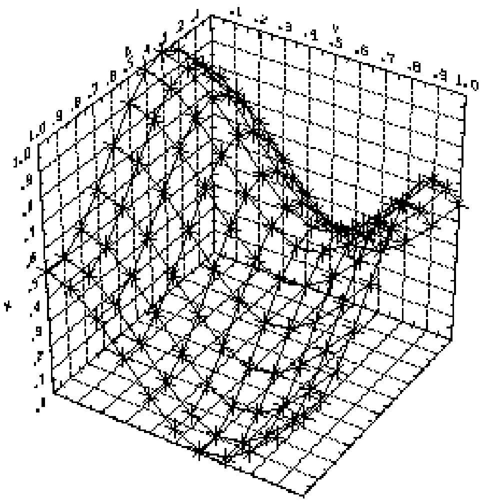
Line segments are drawn in the current polyline color, as determined by the last call to the GKS routine GSPLCI; by default, color index 1 is used. Line width is determined by the last call to the GKS routine GSLWSC; by default, the value of the line width scale factor is 1.

Note that the routines FRST3 and VECT3 do not flush the SPPS polyline buffer. In fact, to have them do so would entirely defeat the purpose of that buffer. Therefore, if your last call was to the routine VECT3 and you are about to call GKS routines to change color or line width, you must first call PLOTIF (as shown on line 2) to flush the buffer; otherwise, the color and line width changes will affect line draws flushed from the buffer later.

Drawing straight-line segments with LINE3

The routine LINE3 may be called to draw the projection of the straight line segment joining two points in 3-space.

A simple Threed plot using LINE3



Code segment from fthex02.f

```
      WFUN(U,V)=.5+.25*SIN(5.*U)+.25*COS(5.*V)
      CALL PLOTIF (0.,0.,2)
      CALL GSLWSC (2.)
      DO 114 I=1,21,2
         UTMP=REAL(I-1)/20.
         DO 113 J=1,21,2
                VTMP=REAL(J-1)/20.
                WTMP=WFUN(UTMP,VTMP)
                CALL LINE3 (UTMP-.04,VTMP,WTMP,UTMP+.04,VTMP,WTMP)
                CALL LINE3 (UTMP,VTMP-.04,WTMP,UTMP,VTMP+.04,WTMP)
                CALL LINE3 (UTMP,VTMP,WTMP-.04,UTMP,VTMP,WTMP+.04)
113      CONTINUE
114   CONTINUE
```

Synopsis

```
      CALL LINE3 (UPOS1, VPOS1, WPOS1, UPOS2, VPOS2, WPOS2)
```

Arguments

UPOS1, VPOS1, WPOS1
: *Real, Input*---The U, V, and W coordinates of a point in 3-space.

UPOS2, VPOS2, WPOS2
: *Real, Input*---The U, V, and W coordinates of another point in 3-space.

Discussion

The **fthex02.f** code segment uses LINE3 to draw a little set of "axes" at each point on a wire-frame representation of the surface defined by the arithmetic statement function on line 1. Line 2 flushes the SPPS polyline buffer, so that lines waiting in that buffer to be drawn will not be affected by the change in line width that is requested by the GKS call on line 3. (**Note**: LINE3 flushes the SPPS polyline buffer after drawing a line, so you needn't worry about its leaving anything in the buffer.) Lines 4 through 13 loop through the points on the grid, drawing the desired object at each of those points.

The statement:

```
      CALL LINE3 (UPOS1, VPOS1, WPOS1, UPOS2, VPOS2, WPOS2)
```

is equivalent to the three statements:

```
      CALL FRST3 (UPOS1, VPOS1, WPOS1)
      CALL VECT3 (UPOS2, VPOS2, WPOS2)
      CALL PLOTIF (0.,0.,2)
```

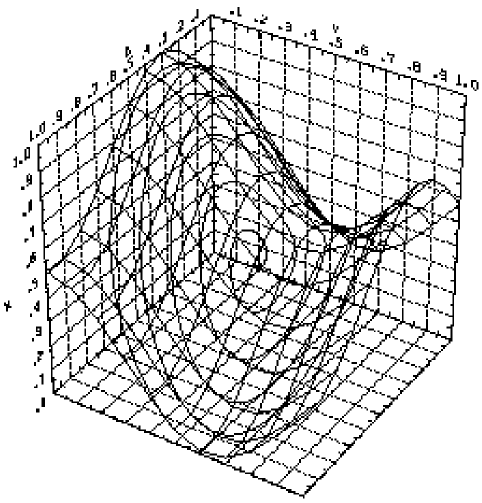
but is slightly more efficient. To approximate a curve defined by three or more points, though, it is not efficient to use LINE3, because the first point of each line segment after the first will be a repeat of the second point of the previous line segment and will therefore be repeated in the metafile. Thus, to approximate a curve, you should use FRST3 and VECT3 (or CURVE3, which is described in the next module).

Straight-line segments drawn by LINE3 are drawn in the current polyline color, as determined by the last call to the GKS routine GSPLCI; by default, color index 1 is used. Line width is determined by the last call to the GKS routine GSLWSC; by default, the line width scale factor is 1.

Drawing curves with CURVE3

The routine CURVE3 may be called to draw the projection of a curve defined by an arbitrary number of points in 3-space.

A simple Threed plot using CURVE3



Code segment from fthex03.f

```
1        WFUN(U,V)=.5+.25*SIN(5.*U)+.25*COS(5.*V)
2        CALL PLOTIF (0.,0.,2)
3        CALL GSLWSC (2.)
4        DO 113 I=1,361
5           TEMP=5.*REAL(I-1)
6           UCRV(I)=.5+TEMP/3600.*COS(.017453292519943*TEMP)
7           VCRV(I)=.5+TEMP/3600.*SIN(.017453292519943*TEMP)
8           WCRV(I)=WFUN(UCRV(I),VCRV(I))
9  113   CONTINUE
10       CALL CURVE3 (UCRV,VCRV,WCRV,361)
```

Synopsis

```
      CALL CURVE3 (UCURVE, VCURVE, WCURVE, NPNTS)
```

Arguments

UCURVE, VCURVE, WCURVE

Real arrays, Input---Arrays of length at least NPNTS, containing the U, V, and W coordinates, respectively, of the points defining the curve whose projection is to be drawn.

NPNTS

Integer, Input---The number of points defining the curve.

Discussion

The **fthex03.f** code segment generates and stores the U, V, and W coordinates of a sequence of points along a curve in 3-space (the curve is the intersection of a sort of spiral cylinder with the surface defined by the arithmetic statement function on line 1); it then calls CURVE3 to draw the curve. Note that no attempt is made to smooth the curve; it is approximated by drawing the straight-line segments defined by adjacent points in the input arrays.

Line 2 flushes the SPPS polyline buffer, so that lines waiting in that buffer to be drawn will not be affected by the change in line width that is requested by the GKS call on line 3. (**Note:** CURVE3 flushes the SPPS polyline buffer after drawing a curve, so you needn't worry about its leaving anything in the buffer.)

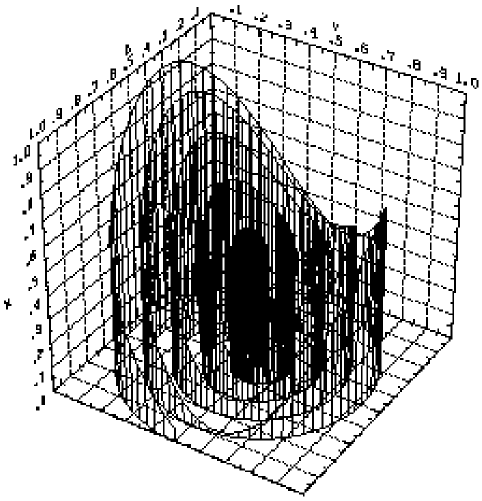
Lines 4 through 9 define the input arrays, and line 10 calls CURVE3.

Curves drawn by CURVE3 are drawn in the current polyline color, as determined by the last call to the GKS routine GSPLCI; by default, color index 1 is used. Line width is determined by the last call to the GKS routine GSLWSC; by default, the line width scale factor is 1.

Drawing curves with fences, using FENCE3

The routine FENCE3 may be called to draw the projection of a curve defined by an arbitrary number of points in 3-space. It also draws a fence joining the curve to a plane perpendicular to one of the three coordinate axes.

A simple Threed plot using FENCE3



Code segment from fthex04.f

```
      WFUN(U,V)=.5+.25*SIN(5.*U)+.25*COS(5.*V)
      CALL PLOTIF (0.,0.,2)
      CALL GSLWSC (2.)
      DO 109 I=1,361
         TEMP=5.*REAL(I-1)
         UCRV(I)=.5+TEMP/3600.*COS(.017453292519943*TEMP)
         VCRV(I)=.5+TEMP/3600.*SIN(.017453292519943*TEMP)
         WCRV(I)=WFUN(UCRV(I),VCRV(I))
  109 CONTINUE
      CALL FENCE3 (UCRV,VCRV,WCRV,361,3,0.)
```

Synopsis

```
      CALL FENCE3 (UCURVE, VCURVE, WCURVE, NPNTS, IOREN, BOTTOM)
```

Arguments

UCURVE, VCURVE, WCURVE

Real arrays, Input---Arrays of length at least NPNTS, containing the U, V, and W coordinates, respectively, of the points defining the curve whose projection is to be drawn.

NPNTS

Integer, Input---The number of points defining the curve.

IOREN

Integer, Input---The direction in which fence lines are to be drawn, as follows:

1
: Parallel to the U axis.

2
: Parallel to the V axis.

3
: Parallel to the W axis.

BOTTOM
: *Real, Input*---Specifies where the bottom of the fence is to be drawn. For example, if fence lines are to be drawn parallel to the W axis and BOTTOM = 2., then the bottom of the fence would be in the plane W = 2.

  The word "bottom" may be misleading here; the entire curve might be below the plane of the "bottom" and the fence might be horizontal, rather than vertical, in the three-dimensional space.

Discussion

The **fthex04.f** code segment stores the U, V, and W coordinates of a series of points along a curve in 3-space (the curve is the intersection of a sort of spiral cylinder with the surface defined by the arithmetic statement function on line 1) and then calls FENCE3 to draw the curve. Note that no attempt is made to smooth the curve; it is approximated by drawing the straight-line segments defined by adjacent points in the input arrays.

The call on line 2 flushes the SPPS polyline buffer, so that lines waiting in that buffer to be drawn will not be affected by the change in line width that is requested by the GKS call on line 3. (**Note**: FENCE3 flushes the SPPS polyline buffer after drawing a curve, so you needn't worry about its leaving anything in the buffer.)

Lines 4 through 9 define the input arrays, and line 10 calls FENCE3.

The curve and the fence are drawn in the colors implied by the values of the variables ITHRMJ and ITHRMN, respectively, in the Threed COMMON block:

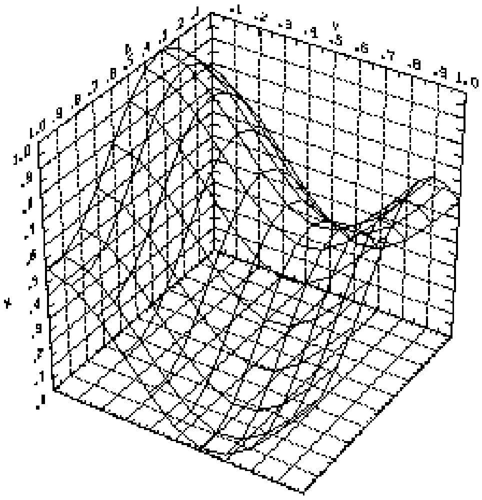
```
COMMON /THRINT/ ITHRMJ,ITHRMN,ITHRTX
```

The default value of both ITHRMJ and ITHRMN is 1; user code may change these values.

Drawing character strings (labels)

The routine PWRZT can be called to draw the projections of character strings that are positioned in any plane perpendicular to one of the three axes.

A simple Threed plot



Code segment from fthex01.f

```
      CHARACTER*8 CHRS
      CALL PWRZT (.5,0.,1.1,’U’,1,3,-1,+3,0)
      DO 102 ILAB=1,10
         UPOS=REAL(ILAB)/10.
         WRITE (CHRS,’(F8.1)’) UPOS
         IBEG=0
         DO 101 ICHR=1,8
            IF (CHRS(ICHR:ICHR).NE.’ ’) THEN
                IF (IBEG.EQ.0) THEN
                    IBEG=ICHR
                END IF
                IEND=ICHR
            END IF
  101    CONTINUE
         IF (CHRS(IBEG:IBEG).EQ.’0’) IBEG=MIN(IBEG+1,IEND)
         CALL PWRZT (UPOS,0.,1.05,CHRS(IBEG:IEND),IEND-IBEG+1,
     +    3,-1,+3,0)
  102 CONTINUE
```

Synopsis

```
CALL PWRZT (UPOS, VPOS, WPOS, CHRS, LCHRS, ISIZE, IDIR, ITOP, ICEN)
```

Arguments

UPOS, VPOS, WPOS

Real, Input---The U, V, and W coordinates of a point in 3-space, relative to which the character string is to be positioned.

CHRS

Character expression, Input---The character string to be written. It may contain uppercase alphabetic characters, the digits 0 through 9, and the characters +, -, *, /, (,), =, blank, comma, and period. Other characters are treated as blanks. (Note especially that lowercase alphabetic characters are not available.)

LCHRS

Integer, Input---The length of the character string CHRS (the number of characters to be written).

ISIZE

Integer, Input---The character width to be used, defined in terms of the width of the plotter frame. Because projected characters vary in width depending on just where they are placed within the box being viewed and the position from which that box is viewed, ISIZE is interpreted as specifying the width of a character when that character is positioned and viewed in such a way as to make it as large as it could possibly be---when the character is on the near side of the box and in a plane perpendicular to the line of sight.

0, 1, 2, or 3

Character width is 1, 1.5, 2, or 3, respectively, times a standard width equal to 1/128th of the width of the plotter frame. For example, when ISIZE =3, the character width is 3/128ths of the width of the plotter frame.

>3

Character width is in units of 1/1024th of the width of the plotter frame.

One third of the "width" referred to here is white space. What ISIZE really specifies is the distance between the centers of adjacent characters in a string. Characters are digitized to be 7/6 * ISIZE units high and 4/6 * ISIZE units wide, excluding white space.

IDIR

Integer, Input---The direction in which the character string is to be written, as follows:

+1 implies +U, -1 implies -U. +2 implies +V, -2 implies -V. +3 implies +W, -3 implies -W. Note that the character string cannot be written at an arbitrary angle; it must be written parallel to one of the 3-D axes.

ITOP

Integer, Input---The direction from the center of the first character to the top of the first character; possible values of ITOP are the same as those of IDIR, above. ABS(ITOP) must not be equal to ABS(IDIR).

ICEN

Integer, Input---The centering option, specifying where (UPOS,VPOS,WPOS) is relative to the string written:

-1

The center of the left edge of the first character.

0

The center of the entire string.

+1

The center of the right edge of the last character.

Discussion

The **fthex01.f** code segment shows how to use PWRZT to label an axis/perimeter. No attempt has been made to generalize the code; it depends heavily on specific numbers having been used in the call to SET3.

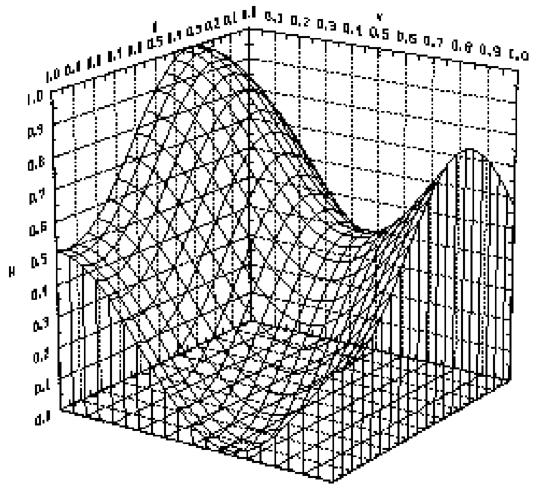
Line 1 defines a character variable in which label strings may be formed. Line 2 writes an appropriate label for the U axis. The loop on Lines 3 through 17 writes 10 numeric labels for the U axis. The inner loop on lines 7 through 14 just finds the start and end of the numeric label in the character variable.

Because characters drawn by PWRZT are stroked using the GKS polyline primitive (so that they can be projected from 3-D to 2-D), they are drawn in the current polyline color, as determined by the last call to the GKS routine GSPLCI; by default, color index 1 is used. Line width is determined by the last call to the GKS routine GSLWSC; by default, the line width scale factor is 1.

Overlaying output from Surface on a Threed plot

It is not particularly easy to overlay output from Surface and Threed. The example described here does it using previously-unadvertised features of both packages. It is hoped that the new 3-D package now in preparation will make this technique obsolete and make it easier to overlay such output.

Overlaid Surface/Threed output



Code segment from fthex05.f

```
      COMMON /TEMPRT/ RZRO
      COMMON /SRFIP1/ IFR,ISTP,IROTS,IDRX,IDRY,IDRZ,IUPPER,ISKIRT,
     + NCLA,THETA,HSKIRT,CHI,CLO,CINC,ISPVAL
      DIMENSION PEYE(6)
      DIMENSION U(21),V(21),W(21,21),RWRK(882)
      IFR=0
      ISKIRT=1
      HSKIRT=0.
      (set U, V, W, UMIN, UMAX, VMIN, VMAX, WMIN, WMAX, and RZRO)
      PEYE(4)=.5*(UMIN+UMAX)
      PEYE(5)=.5*(VMIN+VMAX)
      PEYE(6)=.5*(WMIN+WMAX)
```

```
13       CALL SETR(UMIN,UMAX,VMIN,VMAX,WMIN,WMAX,RZRO)
14       CALL SET3(.0087976,.9902248,.0087976,.9902248,
15       UMIN,UMAX,VMIN,VMAX,WMIN,WMAX,PEYE)
16       CALL PLOTIF (0.,0.,2)
17       CALL GSLWSC (2.)
18       CALL SRFACE (U,V,W,RWRK,21,21,21,PEYE,0.)
```

Discussion

To overlay Threed and Surface output, you must ensure that both packages use the same mapping from 3-space to 2-space. There are three parts to this, as shown in the **fthex05.f** code segment:

Part 1: Line 1 declares a Threed COMMON block in which it is possible to communicate to the package a value RZRO which, when set nonzero, selects a different internal scaling option. The same value is passed to Surface by the call on line 13 and performs the same function in that package. The value of RZRO is that distance from which the box defined by the values of UMIN, UMAX, VMIN, VMAX, WMIN, and WMAX, when viewed from the direction that makes it largest, is to just touch the edges of the viewport in the image plane. Using an eye position closer to the center of the box than RZRO will result in a larger image and using an eye position further from the center of the box than RZRO will result in a smaller image.

Part 2: Line 4 declares an array in which to put the eye position for Threed (elements 1, 2, and 3) and both the eye position (elements 1, 2, and 3) and the position of the point looked at (elements 4, 5, and 6) for Surface. On lines 10, 11, and 12, the position of the point looked at is set to the value that Threed will use by default (the center of the box defined by UMIN, UMAX, VMIN, VMAX, WMIN, and WMAX), so that Surface will do the same thing.

Part 3: The first four arguments in the call to SET3, on lines 14 and 15, have just the right values required to ensure that Threed will use exactly the same viewport that Surface will use.

Lines 2 and 3 declare a Surface COMMON block providing access to internal parameters controlling various aspects of its behavior. Lines 6, 7, and 8 set parameters to turn off frame advances, to turn on the drawing of a "skirt," and to specify the level at the bottom of the "skirt."

Line 5 defines arrays to be used as arguments in the call to SRFACE. Three of these define the surface to be drawn and the final one is a workspace array.

Lines 16 and 17 flush the SPPS polyline buffer and double the GKS line width, so that the surface drawn by the call to SRFACE, on line 18, will be bolder than the rest of the plot.

Chapter 20: Drawing surfaces and isosurfaces

Three-dimensional plots can be drawn using the Surface and Isosurface utilities.

The SRFACE routine draws a perspective picture of a surface represented by a two-dimensional input array. Each value in the array represents the height for a point in the surface.

The ISOSRF routine also draws a perspective picture of a surface; however, it computes the surface from a 3-D "volume" of data and a threshold value defining the surface.

This module discusses how to draw 3-D images and add perspective text to a plot.

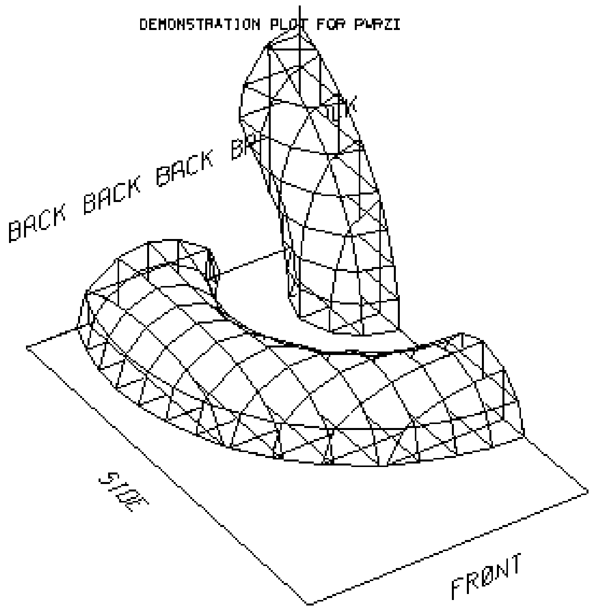


Table of isosurface-drawing user entry points

There are two sets of routines for drawing 3-D images. This table lists the routines in each set: the Isosurface routines and the Surface routines.

Isosurface routines

EZISOS
: Simpler alternative for drawing an isosurface than using the ISOSRF routine.

ISGETI
: Gets isosurface integer parameters.

ISGETR
: Gets isosurface real parameters.

ISOSRF
: Draws an isosurface.

ISSETI
: Sets the isosurface integer parameters.

ISSETR

Sets the isosurface real parameters.

PWRZI

Plots characters in 3-space when using ISOSRF.

Surface routines

EZSRFC

Simpler alternative for drawing a surface than using the SRFACE routine.

PWRZS

Plots characters in 3-space when using the SRFACE routine.

SRFACE

Draws a surface.

SETR

Overlays THREED and SRFACE plots. See the **fthex05.f** code segment in the THREED module for an example that uses this routine. (The SETR routine has not been previously documented.)

Table of Isosurface-drawing parameters

For a complete description of the Isosurface parameters, see the **isosurface_params man** page or the Isosurface programmer document.

Isosurface parameters

```
-------------------------------------------------------------
Parameter  Brief description                     Fortran type
-------------------------------------------------------------
IU         Number of U-axis Interpolation slabs  Integer
IV         Number of V-axis Interpolation slabs  Integer
IW         Number of W-axis Interpolation slabs  Integer
RF         ReFerence plane drawing flag          Integer
RS         Relative Size flag                    Real
SL         Smooth curve segment Length           Real
SM         Screen Model resolution               Integer
ST         Spline Tension                        Real
SV         Special Value flag                    Integer
VB         Viewport Bottom edge                  Real
VL         Viewport Left edge                    Real
VR         Viewport Right edge                   Real
VT         Viewport Top edge                     Real
-------------------------------------------------------------
```

Table of surface-drawing parameters

The Surface parameters are not accessible via parameter retrieving and setting routines at this time. We hope to update the Surface utility and include new functionality as well as parameter-access routines.

Currently, however, internal Surface parameters are only accessible in common blocks. Please see the Surface **man** page for more information about specific routines.

Surface internal parameters

```
-----------------------------------------------------------
Parameter  Brief description                   Fortran type
-----------------------------------------------------------
IFR        Flag for calling FRame              Integer
ISTP       Stereo flag                         Integer
IROTS      Controls plotting direction         Integer
IDRX       DRaw lines of constant X flag       Integer
IDRY       DRaw lines of constant Y flag       Integer
IDRZ       DRaw lines of constant Z flag       Integer
IUPPER     Draw UPPER/lower/both side flag     Integer
ISKIRT     Draw SKIRT around surface flag      Integer
NCLA       Number of Z levels (approximation)  Integer
THETA      Stereo angle in radians             Real
HSKIRT     Height of SKIRT                     Real
CHI        HIghest level of Constant Z         Real
CLO        LOwest level of Constant Z          Real
CINC       INCrement between levels            Real
IOFFP      Special value flag                  Integer
SPVAL      The SPecial VALue                   Real
-----------------------------------------------------------
```

Isosurface parameters: What they do and how to use them

The Isosurface parameters control characteristics such as position, missing data value, interpolation factors, and so on. This module explains how to set and retrieve these Isosurface parameters with parameter-access routines. The internal Surface parameters, however, must be accessed via common blocks.

Synopsis

```
      CALL ISGETI (PNAM, IVAL)
      CALL ISGETR (PNAM, RVAL)
      CALL ISSETI (PNAM, IVAL)
      CALL ISSETR (PNAM, RVAL)
```

Routines

ISGETI, ISGETR

The ISGETI and ISGETR routines are used to retrieve the current value of the parameter, PNAM. The value is returned in the second argument.

ISGETI is used to retrieve integer parameters. ISGETR is used to retrieve real parameters.

ISSETI, ISSETR

The ISSETI and ISSETR routines are used to assign a value (the second argument) to the parameter, PNAM.

ISSETI is used to set integer parameters.ISSETR is used to set real parameters.

Arguments

PNAM

Character, Input---A two-letter string that specifies the name of a parameter. This is one of the two-character mnemonics listed in the parameter table under "Isosurface parameters." Only the first two characters of PNAM are relevant, and they should be uppercase. A longer character string may be used to more clearly document what parameter is being accessed.

IVAL

Integer, Input or Output---An integer value.

RVAL

Real, Input or Output---A real value.

Discussion

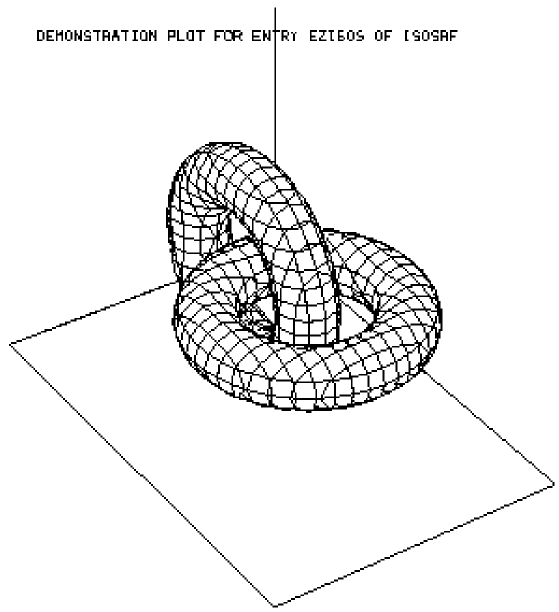
The Isosurface parameters are controlled using the ISGET*x* and ISSET*x* parameter-access routines.

The internal Surface parameters, however, cannot currently be accessed with these types of routines. We hope to add this functionality in the future. Until then, Surface parameters must be accessed via common blocks.

Drawing pictures with the EZISOS routine

If the following three conditions are met, then you can use EZISOS to draw an isosurface. The conditions are when the entire range of the input array is used, when it doesn't matter which constant lines of U, V, and W are used to draw the surface, and when a frame advance is to be done after drawing the surface. If these three conditions are not met, then you must use ISOSRF. This module demonstrates how to use EZISOS.

EZISOS plot



Code segment from fisissrf.f

```
      DIMENSION       T(21,31,19),SLAB(33,33),EYE(3)
      DATA NU,NV,NW/21,31,19/
      DATA TISO/0./

      EYE(1) = 100.
      EYE(2) = 150.
      EYE(3) = 125.
      CALL GSELNT(0)
      CALL EZISOS (T,NU,NV,NW,EYE,SLAB,TISO)
      RETURN
```

Synopsis

```
CALL EZISOS (FINPUT, MU, MV, MW, EYE, SLAB, FISO)
```

Arguments

FINPUT

Real array, Input---A 3-D array of data defining the "volume" of data through which the isosurface, defined by FISO, passes. The entire array is used and cannot be subsampled, unlike when using the ISOSRF routine. The dimensions of FINPUT are MU * MV * MW.

MU, MV, MW

Integer, Input---The first, second and third dimensions of FINPUT, respectively.

EYE

Real array, Input---A three-element array that defines the eye position in the UVW coordinate system. The eye position at (EYE(1), EYE(2), EYE(3)) must be outside the bounds of the box containing the data. The eye looks from the eye position to the center of the data box, and the projection plane is perpendicular to the line of sight. A good initial choice for EYE is:

(5. * REAL(MU), 3.5 * REAL(MV), 2. * REAL(MW))

SLAB

Real array, Workspace---A workspace array dimensioned at least *n* by *n*, where *n* is defined to be the maximum of (MU, MV, MW) + 2.

FISO

Real, Input---The value that defines the isosurface. The isosurface passes through the data values in FINPUT that equal FISO.

Discussion

EZISOS should only be used to draw an isosurface if the following three conditions are met: if all of FINPUT is used, if it doesn't matter which constant lines of U, V, and W are used to draw the surface, and if a frame advance is to be done after the surface is drawn. If these conditions are not met, then use the ISOSRF routine described in the next module.

The **fisissrf.f** code segment demonstrates how to use EZISOS to draw a perspective picture of an isosurface.

Line 1 sets up the dimensions of the input array, workspace array, and eye point. Because of space constraints, the code for filling the input array, T, with a "volume" of data is not shown. Please see the **fisissrf.f** file for this code.

Line 2 initializes the integer variables that define the input array dimensions.

Line 3 sets the threshold value that defines the isosurface. The isosurface will pass through all data points in the input array, T, that equal this value.

Lines 5 through 7 define the position of the eye.

Line 8 selects the identity transformation in which both the window and viewport have the range of 0. to 1. in both dimensions.

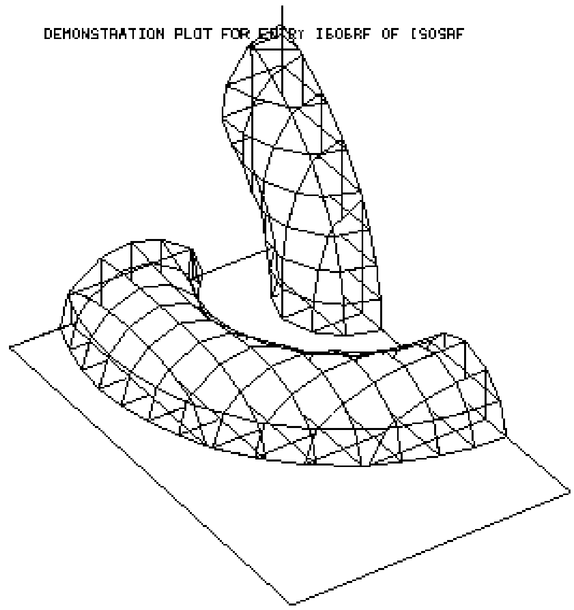
Line 9 calls EZISOS to draw the isosurface.

Line 10 returns from the function call. Note that when using EZISOS, you do not need to advance the frame with the FRAME call.

Drawing pictures with the ISOSRF routine

ISOSRF is a more general-purpose isosurface-drawing routine than EZISOS. Unlike EZISOS, it does not have any predefined conditions that must be met, and ISOSRF supports data subsampling. This module demonstrates how to draw an isosurface with the ISOSRF routine.

ISOSRF plot



Code segment from fisissrf.f

```
      DIMENSION       T(21,31,19),SLAB(33,33),EYE(3)
      DATA NU,NV,NW/21,31,19/
      DATA TISO/0./
      DATA IFLAG/-7/
      EYE(1) = 100.
      EYE(2) = 150.
      EYE(3) = 125.
      CALL GSELNT(0)
      MU=NU/2
      MV=NV/2
      MW=NW/2
      MUVWP2=MAX0(MU,MV,MW)+2
      CALL ISOSRF(T(MU,MV,MW),NU,MU,NV,MV,MW,EYE,MUVWP2,SLAB,TISO,IFLAG)
      CALL FRAME
```

Synopsis

```
      CALL ISOSRF (FINPUT, LU, MU, LV, MV, MW, EYE, MUVWP2, SLAB, FISO,
     +             IFLAG)
```

Arguments

FINPUT

Real array, Input---A 3-D array of data defining the "volume" through which the isosurface, defined by FISO, passes. The dimensions of FINPUT are LU by LV by *m*, where *m* is greater than or equal to MW. Unlike the EZISOS routine, ISOSRF allows you to create an isosurface from a subarray, defined by MU, MV, and MW, of FINPUT.

LU, LV

Integer, Input---Defines the first and second dimensions of the FINPUT array, respectively.

MU, MV, MW

Integer, Input---Defines the range to be used for the first, second and third subscript of the FINPUT array, respectively. Only the portion of the array consisting of elements FINPUT(IU,IV,IW), for IU = 1 to MU, IV = 1 to MV, and IW = I to MW, is used. This may or may not be the entire array.

EYE

Real array, Input---A three-element array that defines the eye position in the UVW coordinate system. They eye position at (EYE(1), EYE(2), EYE(3)) must be outside the bounds of the box containing the data. The eye looks from the eye position to the center of the data box, and the projection plane is perpendicular to the line of sight. A good initial choice for EYE is:

(5. * REAL(MU), 3.5 * REAL(MV), 2.*REAL(MW))

MUVWP2

Integer, Input---The integer value that is the maximum of (MU, MV, and MW) plus 2.

SLAB

Real array, Workspace---A workspace array dimensioned at least MUVWP2 * MUVWP2.

FISO

Real, Input---The value that defines the isosurface. The isosurface passes through the data values in FINPUT that equal FISO.

IFLAG

Integer, Input---IFLAG serves two purposes:

First, the absolute value of IFLAG determines which type of lines (lines of constant U, V, and W) are drawn to approximate the isosurface. The following table lists the types of lines drawn based on the absolute value of IFLAG.

IFLAG	Lines of Constant U	Lines of Constant V	Lines of Constant W
1	no	no	yes
2	no	yes	no
3	no	yes	yes
4	yes	no	no
5	yes	no	yes
6	yes	yes	no
7	yes	yes	yes

Second, the sign of IFLAG determines what is inside and what is outside the solid bounded by the isosurface and thus which lines are visible and what is done at the boundary of the box containing the data. If IFLAG is positive, values greater than FISO are considered to be inside the solid formed by the isosurface. If IFLAG is negative, values less than FISO are considered to be inside the isosurface. If ISOSRF draws a cube, reverse the sign of IFLAG.

Discussion

ISOSRF allows you to draw a perspective view of an isosurface. Contours created by taking slices in any of three directions through the isosurface are drawn with hidden portions of the contour lines removed.

Line 1 of the **fisissrf.f** code segment sets up the dimensions of the input array, workspace array, and eye point array. Because of space constraints, the code for filling the input array, T, with a "volume" of data is not shown. Please see the **fisissrf.f** file for this code.

Line 2 initializes the integer variables that define the input array dimensions.

Line 3 sets the threshold value that defines the isosurface. The isosurface passes through all data points in the input array, T, that equal this value.

Line 4 sets IFLAG to -7. This indicates to the ISOSRF routine to draw lines of constant U, V, and W, and that values less than FISO should be considered inside the isosurface solid.

Lines 5 through 7 define the eye position.

Line 8 selects the identity transformation in which both the window and viewport have the range of 0. to 1. in both dimensions.

Lines 9 through 11 set the subarray size.

Line 12 calculates the maximum of (MU, MV, and MW) plus 2.

Line 13 draws the isosurface.

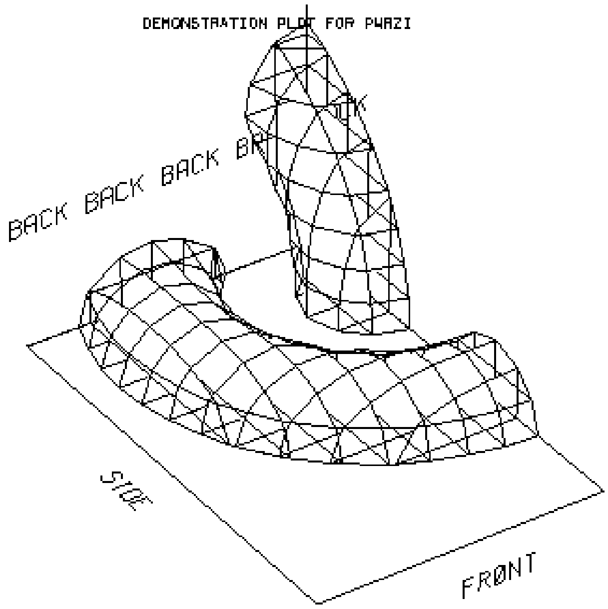
Line 14 advances the frame. FRAME must be called explicitly with the ISOSRF routine. (You do not need to call FRAME when using EZISOS.)

Drawing text in 3-space when using ISOSRF

PWRZI is a character-plotting routine used to draw characters in 3-space when using the ISOSRF routine. The PWRZI routine lets you position text at any position and orientation in the plot.

This module demonstrates how to use PWRZI to draw text when using the ISOSRF routine and how to set the surface and text colors.

Drawing text with the PWRZI routine



Code segment from fispwrzi.f

```
      CALL GSPLCI(4)
      CALL ISOSRF (T(MU,MV,MW),NU,MU,NV,MV,MW,EYE,MUVWP2,SLAB,TISO,
     + IFLAG)
      ISIZE = 35
      CALL GSPLCI(2)
      CALL PWRZI (5.,16.,.5,'FRONT',5,ISIZE,-1,3,0)
      CALL PWRZI (11.,7.5,.5,'SIDE',4,ISIZE,2,-1,0)
      CALL PWRZI (5.,1.,5.,' BACK BACK BACK BACK BACK',25,ISIZE,-1,3,0)
      CALL FRAME
```

Synopsis

```
CALL PWRZI (X, Y, Z, LABEL, N, ISIZE, LINE, ITOP, ICNT)
```

X, Y, Z

Real, Input---The positioning coordinates for the characters to be drawn. These are floating point numbers in the same 3-space coordinate system used in ISOSRF.

LABEL

Character, Input---Specifies the character string to be drawn. PWRZI only draws uppercase characters, so LABEL must be a string of all uppercase characters.

N

Integer, Input---Specifies the number of characters in LABEL.

ISIZE

Integer, Input---Specifies the size of characters to be used.

If ISIZE is between 0 and 3, the value specifies 1., 1.5, 2., or 3. times a standard width equal to

1/128th of the screen width.

If ISIZE is greater than 3, the value specifies a character width in plotter address units (by default, 1 plotter address unit is 1/1024th of the screen width).

LINE
Integer, Input---Specifies the coordinate direction in which the characters are written.

+1
Draws text in the +X direction.
-1
Draws text in the -X direction.
+2
Draws text in the +Y direction.
-2
Draws text in the -Y direction.
+3
Draws text in the +Z direction.
-3
Draws text in the -Z direction.

ITOP
Integer, Input---Specifies the coordinate direction from the center of the first character to the top of the first character. Valid values for ITOP are the same as those for LINE, given above. Note that the absolute value of LINE cannot equal the absolute value of ITOP.

ICNT
Integer, Input---Specifies the centering option.

-1
(X,Y,Z) is the center of the left edge of the first character.
0
(X,Y,Z) is the center of the entire string.
+1
(X,Y,Z) is the center of the right edge of the last character.

Discussion

PWRZI is a limited text-drawing routine for labeling isosurfaces in 3-space. There is only one font, and lowercase characters will not be printed. Character color is controlled by calling the GKS routine GSPLCI to set the polyline color index. The PWRZI routine should only be called after ISOSRF is called.

Line 1 of the **fispwrzi.f** code segment sets the line color by specifying a color index in the GKS color table. The color table was set up prior to entering this code segment by calling the GKS routine GSCR. This effectively sets the color of the lines drawn by ISOSRF in the next line.

Lines 2 and 3 draw an isosurface.

Line 4 sets the ISIZE variable that specifies character width. The value 35 indicates that the characters will be 35 plotter address units in width.

Line 5 sets the line color to a new color table index. This effectively sets the color of the text drawn by the next three PWRZI calls.

Line 6 draws the string 'FRONT' at position (5.,16.,.5). The string is drawn in the negative X direction, and the character is drawn "upright" in the positive Z direction. The coordinate (5.,16.,.5) specifies the center of the string.

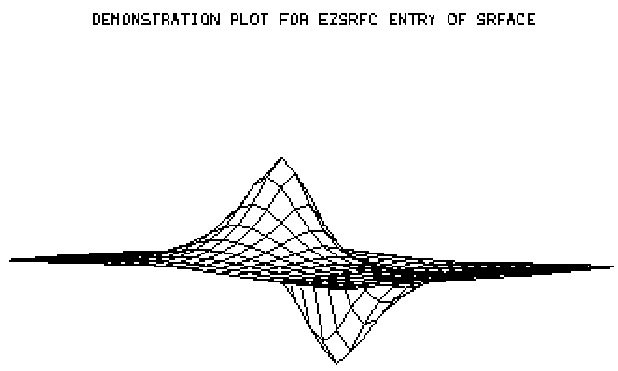
Similarly, lines 7 and 8 draw two more strings on the plot at different positions and different orientations.

Line 9 advances the frame.

Drawing pictures with the EZSRFC routine

EZSRFC is used to draw a picture of a 3-D surface represented by a 2-D input array where each array value represents a height. This module demonstrates how to draw a picture with EZSRFC and how to set the line color used to draw the surface.

EZSRFC plot



Code segment from fsrezsrf.f

```
      REAL             Z(21,25),  WORK(1096)
      DATA  ANGH/45./, ANGV/15./
      CALL GSCR (1,0,1.,1.,1.)
      CALL GSCR (1,1,0.,0.,1.)
      DO  20 I=1,21
         X = .1*FLOAT(I-11)
         DO  10 J=1,25
            Y = .1*FLOAT(J-13)
            Z(I,J) = (X+Y+1./((X-.1)**2+Y**2+.09)-
     +         1./((X+.1)**2+Y**2+.09))*.25
 10      CONTINUE
 20   CONTINUE
```

```
13      CALL GSELNT(0)
14      CALL EZSRFC (Z,21,25,ANGH,ANGV,WORK)
```

Synopsis

```
CALL EZSRFC (ZDATA, M, N, ANGH, ANGV, WORK)
```

Arguments

ZDATA

Real array, Input---The M by N array to be drawn. Each element of the array represents a height of the surface. EZSRFC, unlike the SRFACE routine described in the next module, requires that the entire array be used.

M, N

Integer, Input---The first and second dimensions of ZDATA, respectively.

ANGH

Real, Input---The angle, in degrees, in the X/Y plane to the line of sight (counterclockwise from the plus-X axis).

ANGV

Real, Input---Angle in degrees from the X/Y plane to the line of sight. Positive angles set the line of sight above the surface and negative values below.

WORK

Real array, Workspace---A scratch storage array dimensioned at least 2 * M * N + M + N.

Discussion

EZSRFC should only be used if the entire input array of heights (ZDATA) is to be drawn, the data points are equally spaced in the X/Y plane, there are no stereo pairs (two pictures) to be drawn, and scaling is chosen internally. Otherwise, you must use the SRFACE routine.

Line 1 of the **fsrezsrf.f** code segment declares the data array and workspace arrays.

Line 2 sets the variables that represent the line of sight angles.

Line 3 sets the background color of the plot.

Line 4 sets the foreground color of the plot. EZSRFC uses the color represented by color index 1 in the GKS color table to color the lines it draws.

Lines 5 through 12 initialize the input data array with height values that represent the surface to be drawn.

Line 13 selects the identity transformation in which both the window and viewport have the range of 0. to 1. in both dimensions.

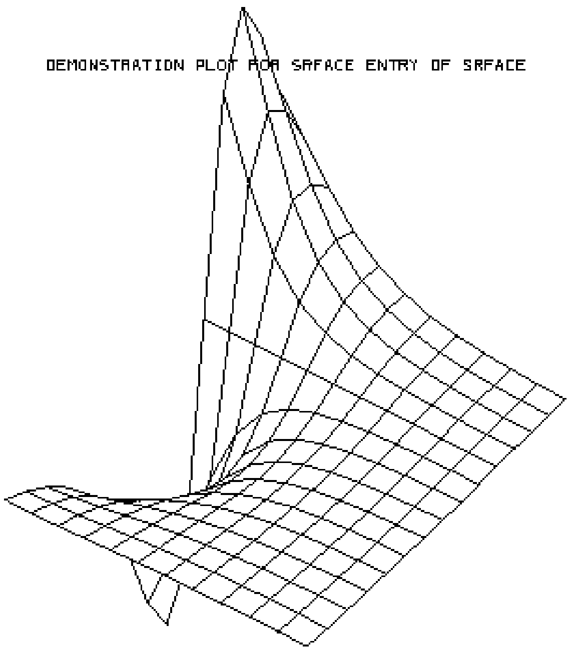
Line 14 draws the surface.

Note that the routine FRAME does not need to be called when using the EZSRFC routine; it is called internally.

Drawing pictures with the SRFACE routine

SRFACE is a more general purpose surface-drawing routine than EZSRFC. SRFACE, for example, supports subsampling data and drawing stereo pairs. This module demonstrates how to draw a surface with the SRFACE routine.

SRFACE plot



Code segment from fsrsrfac.f

```
      REAL XX(21), YY(25), Z(21,25), S(6), WORK(1096)
      DATA S(1), S(2), S(3), S(4), S(5), S(6)
     + /-8.0, -6.0,  3.0,  0.0,  0.0,  0.0/

      CALL GSCR (1,0,1.,1.,1.)
      CALL GSCR (1,1,0.,0.,1.)
      CALL GSELNT(0)
      CALL SRFACE (XX,YY,Z,WORK,21,15,12,S,0.)
```

Synopsis

```
      CALL SRFACE (XCOORD, YCOORD, ZDATA, WORK, MX, NX, NY, EYE, STEREO)
```

Arguments

XCOORD, YCOORD

Real array, Input---Linear arrays of X and Y coordinates, respectively, of the surface. The length of XCOORD is NX, and the length of YCOORD is NY.

ZDATA

Real array, Input---The MX by NY array to be drawn. Each element of the array represents a height of the surface. The units of the array elements are assumed to be in the same units as XCOORD and YCOORD and not wildly different in magnitude. For example, if XCOORD and YCOORD range from 0 to 10,000, and ZDATA ranges from 0 to 1, a flat plane will be drawn. For the best results, the ranges of XCOORD, YCOORD, and ZDATA should be roughly the same.

WORK

Real array, Workspace---A scratch array of at least 2 * NX * NY words long.

MX

Integer, Input---The first dimension of ZDATA.

NX

Integer, Input---The number of data values in the X direction (the first subscript direction) in ZDATA to be plotted. When plotting an entire array, MX=NX.

NY

Integer, Input---The number of data values in the Y direction in ZDATA to be plotted.

EYE

Real array, Input---EYE is a six-element array that defines the line of sight. The units of EYE are assumed to be in the same units as XCOORD and YCOORD. The viewer's eye is at coordinate (EYE(1), EYE(2), EYE(3)), and the point looked at is at coordinate (EYE(4), EYE(5), EYE(6)). The eye should be outside the bounding box of the data that has opposite corners at (XCOORD(1), YCOORD(1), minimum value in ZDATA) and (XCOORD(NX), YCOORD(NY), maximum value in ZDATA).

For a nice perspective effect, the distance between the eye and the point looked at should be 5 to 10 times the size of the bounding box.

STEREO

Real, Input---Flag to indicate if stereo pairs are to be drawn. The value 0.0 means no stereo pair (one picture). Nonzero values mean to create two pictures. The value of STEREO is the relative angle between the eyes. A value of 1.0 produces a standard separation. A negative value of STEREO reverses the left and right figures.

Discussion

The **fsrsrfac.f** code segment demonstrates how to use the SRFACE routine to draw a 3-D surface.

Line 1 declares the dimensions of the XCOORD and YCOORD coordinate arrays, the ZDATA array, the EYE array, and the WORK array, respectively.

Lines 2 and 3 initialize the coordinates of the eyepoint and the point looked at.

Line 5 sets the background color of the plot to white.

Line 6 sets the foreground color of the plot to blue. The default line color of pictures drawn by SRFACE is set to color index 1 (the foreground color). Therefore, the color of surface lines can be changed by setting the foreground color (color table index 1).

Lines 7 selects the identity transformation in which both the window and viewport have the range of 0. to 1. in both dimensions.

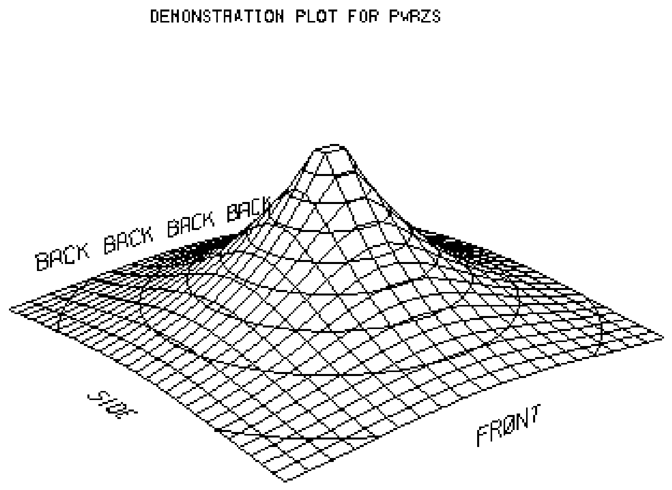
Line 8 draws the surface. In this example, the NX and NY arguments are set to 15 and 12, respectively. The dimensions the ZDATA argument, however, are 21 by 25. This results in subsampling the data.

Note that the routine FRAME was not called to advance the frame. FRAME is called automatically when using the SRFACE routine.

Drawing text in 3-space when using SRFACE

PWRZS is a character-plotting routine used to draw characters in 3-space when using the SRFACE routine. The PWRZS routine lets you position text at any position and orientation in the plot.

This module demonstrates how to use PWRZS to draw text when using the SRFACE routine and how to set the text colors.



Code segment from fsrpwrzs.f

```
      CALL GSELNT (0)
      CALL SRFACE (X,Y,Z,MM,M,M,N,S,0.)
      CALL GSPLCI(2)
      ISIZE = 35
      CALL PWRZS (0.,1.1,0.,'FRONT',5,ISIZE,-1,3,0)
      CALL PWRZS (1.1,0.,0.,'SIDE',4,ISIZE,2,-1,0)
      CALL PWRZS (0.,-1.1,.2,' BACK BACK BACK BACK BACK',25,ISIZE,-1,3,0)
```

Synopsis

```
CALL PWRZS (X, Y, Z, LABEL, N, ISIZE, LINE, ITOP, ICNT)
```

X, Y, Z

Real, Input---The positioning coordinates for the characters to be drawn. These are floating point numbers in the same 3-space coordinate system used in SRFACE.

LABEL

Character, Input---Specifies the character string to be drawn. PWRZS only draws uppercase characters, so LABEL must be a string of all uppercase characters.

N

Integer, Input---Specifies the number of characters in LABEL.

ISIZE

Integer, Input---Specifies the size of characters to be used.

If ISIZE is between 0 and 3, the value specifies 1., 1.5, 2., or 3. times a standard width equal to 1/128th of the screen width.

If ISIZE is greater than 3, the value specifies a character width in plotter address units (by default, 1 plotter address unit is 1/1024th of the screen width).

LINE

Integer, Input---Specifies the coordinate direction in which the characters are written.

+1
Draws text in the +X direction.
-1
Draws text in the -X direction.
+2
Draws text in the +Y direction.
-2
Draws text in the -Y direction.
+3
Draws text in the +Z direction.
-3
Draws text in the -Z direction.

ITOP

Integer, Input---Specifies the coordinate direction from the center of the first character to the top of the first character. Valid values for ITOP are the same as those for LINE, given above. Note that the absolute value of LINE cannot equal the absolute value of ITOP.

ICNT

Integer, Input---Specifies the centering option.

-1
    (X,Y,Z) is the center of the left edge of the first character.
0
    (X,Y,Z) is the center of the entire string.
+1
    (X,Y,Z) is the center of the right edge of the last character.

Discussion

PWRZS is a limited text-drawing routine for labeling surfaces in 3-space. There is only one font, and lowercase characters will not be printed. Character color is controlled by calling the GKS routine GSPLCI to set the polyline color index. The PWRZS routine should be called after SRFACE is called.

Line 1 of the **fsrpwrzs.f** code segment sets the identity transformation in which both the window and viewport have the range of 0. to 1. in both directions.

Line 2 draws a surface. The arguments to this call were set prior to entering this code segment.

Line 3 sets the polyline color index to 2. The characters drawn with PWRZS will use the color set by the GKS routine GSPLCI.

Line 4 sets the ISIZE variable that specifies character width. The value 35 indicates that the characters will be 35 plotter address units in width.

Line 5 draws the string 'FRONT' at position (0.,1.1,0.). The string is drawn in the negative X direction, and the character is drawn "upright" in the positive Z direction. The coordinate (0.,1.1,0.) specifies the center of the string.

Similarly, lines 6 and 7 draw two more strings on the plot at different positions and different orientations.

Chapter 21: Movie titles

The Scrolled_title utility contains routines to produce movie and video titles with a minimum of effort. Titles can be fixed or scrolled. Options exist for fading in and fading out.

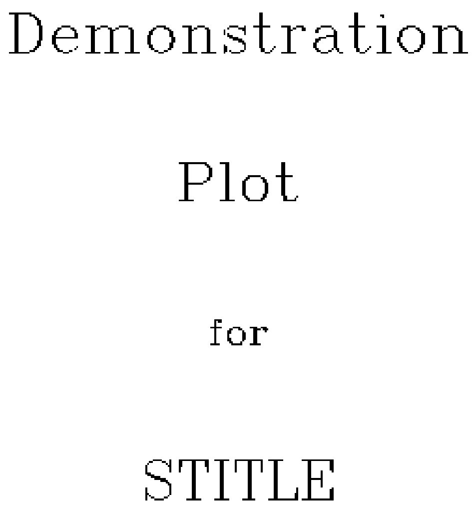


Table of Scrolled_title user entry points

There are six user entry points in this utility. This module organizes them by function.

Action routines

FTITLE
Generates fixed titles.

STITLE
Generates scrolled titles.

Parameter routines

SLGETI
Retrieves integer parameters.

SLGETR
Retrieves real parameters.

SLSETI
Sets integer parameters.

SLSETR
Sets real parameters.

Table of Scrolled_title parameters

You can set an arbitrary number of options through successive calls to any of the parameter-setting routines. A call to FTITLE or STITLE will then generate the desired titles.Please see the **scrolled_title_params man** page for more complete information on these parameters.

Parameter	Fortran type	Brief description	Default value
ALN	Integer	ALigNment frame output flag	0
BGB	Real	BackGround color Blue component	Device default
BGG	Real	BackGround color Green component	Device default
BGR	Real	BackGround color Red component	Device default
FGB	Real	ForeGround color Blue component	Device default
FGG	Real	ForeGround color Green component	Device default
FGR	Real	ForeGround color Bed component	Device default
FIN	Real	Seconds to Fade IN the first title	0.
FOU	Real	Seconds to Fade OUt the last title	0.
GSZ	Real	Value for interline spacing (FTITLE)	40.
ICO	Integer	Centering option (FTITLE)	1
ICU	Integer	Unit number for reading input (FTITLE)	5
INC	Integer	Vertical coordinate spacing between practice frames (STITLE)	300
LOG	Integer	Fortran LOGical unit number for WISS*	4
LX1	Integer	Lower left X value of the viewport	0
LX2	Integer	Upper right X value of the viewport	32767
LY1	Integer	Lower left Y value of the viewport	0
LY2	Integer	Upper right Y value of the viewport	32767
MAP	Integer	Plotchar's parameter 'MA'	100
NXE	Integer	Ends scrolling in the X direction	512
NXS	Integer	Starts scrolling in the X direction	512
ORV	Real	Plotchar's parameter 'OR'	1.E12
PSZ	Real	Default character height (FTITLE)	21.
SBK	Integer	Suppress BackGround color fade in/out	0
SFG	Integer	Suppress ForeGround color fade in/out	0
TM1	Real	Seconds of blank frames before any title frames (FTITLE)	1.
TM2	Real	Seconds of blank frames between sets of title frames (FTITLE)	0.5
WID	Integer	Workstation IDentifier for WISS*	9

*WISS - Workstation Independent Segment Storage

Table of high-quality fonts available for titling

A number of high-quality filled fonts are available for use in the Scrolled_title utility. They are accessed by either name or number using the Plotchar parameter-setting routines PCSETC or PCSETI. The following table describes the type of fonts available and how they are accessed.

Parameter FN can be input as either an integer or a character string. The following table gives the values of FN for various high-quality fonts.

Integer ID	Character ID
-21	helvetica
-22	helvetica-bold
-25	times-roman
-26	times-bold
-29	courier
-30	courier-bold
-33	greek

```
-34         math-symbols
-35         text-symbol
-36         weather1
-37         weather2
---------------------------
```

Example: The following calls are equivalent.

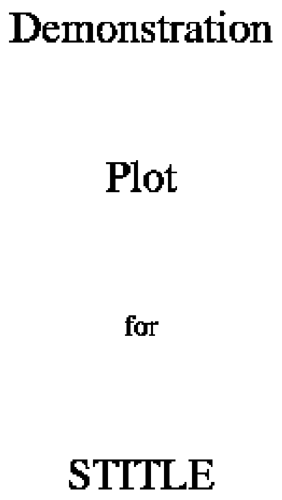
```
      CALL PCSETI ('FN', -25)
      CALL PCSETC ('FN', 'times-roman')
```

The integer form of FN can be plus or minus. The character form of FN can be uppercase or lowercase, but not mixed.

By subtracting 100 from any font in this table, you can add outlines to the text which can be drawn in different colors than the filled interiors of the text. Refer to Chapter 11 "Drawing text and symbols" for details.

Generating high-quality titles

You can obtain high-quality fonts for films and videos using STITLE. The following title is an example of a Times-Roman font.



Code segment from stfont.f

```
        NCARDS = 4
        CARDS(1) = '  512  760    1  1.5Demonstration'
        CARDS(2) = '  512  600    1  1.5Plot'
        CARDS(3) = '  512  440    1  1.0for'
        CARDS(4) = '  512  280    1  1.5STITLE'
        CALL PCSETC('FN','times-roman')
        NYST  = 512
        NYFIN = 512
        TST   = 0.0
        TMV   = 0.0
        TFIN  = 6.0
```

```
      MOVIE =  1
      CALL STITLE (CARDS,NCARDS,NYST,NYFIN,TST,TMV,TFIN,MOVIE)
```

Synopsis

```
CALL STITLE (CARDS, NCARDS, NYST, NYFIN, TST, TMV, TFIN, MV)
```

Arguments

CARDS

Character array, Input---A CHARACTER*n* array dimensioned for NCARDS. You must fill this array prior to calling STITLE, either by internal manipulations or by reading *n*-character "card images." Each element of the array CARDS contains the following:

COLUMNS 1-5

An integer, called MX, which is the X coordinate of this line of text on the scroll, or an indicator that this line of text is a continuation of the previous line. MX is the coordinate of the middle of the line if ICNTR is 1, and the coordinate of the left edge of the first character if ICNTR is 0. See columns 11 through 15 for ICNTR.

The scroll of text is 1024 units wide and any number of units high. MX = -9999 is the continuation card indicator. Multiple continuation cards may be used as long as the total number of characters in a continued line does not exceed 512. Trailing blanks are omitted from the last continuation card.

COLUMNS 6-10

An integer, called MY, which is the Y coordinate of this line of text on the scroll (MY may be outside the 1 to 1024 unit range). On a continuation line, columns 6 to 20 are ignored.

COLUMNS 11-15

The value of ICNTR, which is the centering option:

0

Start the text at MX.

1

Center the text about MX.

2

End the text at MX.

COLUMNS 16-20

The value of SIZE, which is the relative size of characters. This multiplies the Plotchar character height. The recommended range is 1. to 2.5 (Plotchar modifiers can also be used to change sizes).

COLUMNS 21-80

Text for this line, or for continuation of a line when MX = -9999. Plotchar modifiers may be used; see the Plotchar documentation.

NCARDS

Integer, Input---Dimension of the array CARDS, that is, the number of lines in CARDS.

NYST

Integer, Input---The STITLE coordinate that will be at the center of the screen when the text is first displayed.

NYFIN

Integer, Input---The STITLE coordinate that will be at the center of the screen when the text is last displayed.

TST

Real, Input---The time, in seconds, that the scroll will be stationary at NYST. One second is recommended.

TMV

Real, Input---The time, in seconds, for a title line to scroll from NYST to NYFIN. This should be the time required to read the text aloud at slow to normal speed.

TFIN

Real, Input---The time, in seconds, that the scroll will be stationary at NYFIN. One second is recommended.

MV

Integer, Input---Switch to indicate whether this is a practice run or the movie is being made:

0

Movie is being made.

1

Practice run.

Practice runs output outlined representative frames from the scroll with a legend indicating the number of seconds the frame will be shown at the start or finish, or the number of seconds into the total moving time that a particular frame represents. When the movie is being made, this practice output is suppressed.

Discussion

STITLE takes all input through its argument list, and generates graphic output.

All arguments are unchanged on output.

STITLE in effect moves a body of text up through the screen window, outputting frames required to generate a movie sequence of duration specified by the user.

At each frame, STITLE skips plotting lines of text that are completely outside the screen window. Lines of text that are partially in the window are clipped by Plotchar.

Line 1 of the **stfont.f** code segment states that there will be four lines of text generated. Lines 2 through 5 give the text string (X,Y) locations, the text centering option, the Plotchar size modifier, and the text strings.

Line 6 chooses Times-Roman for the title font.

Line 7 shows the STITLE coordinate that will be at the center of the screen when the text is first displayed, and line 8 shows that it will remain at this position at the end of the sequence.

Lines 9 and 10 show that the scroll hold and move times are zero. (This is a fixed title.) Line 11 shows that the fixed title will be held for 6.0 seconds. Line 12 indicates that this is just a test, and line 13 generates the titles.

Note that the output produced (a single frame with no scrolling to appear for 6.0 seconds) could equally well have been produced by FTITLE. We call STITLE in this demo to avoid reading the input lines.

Glossary

This glossary is a brief introduction to terms used in *NCAR Graphics Fundamentals, UNIX Version.* Definitions here are intended to clarify issues without overwhelming new users with precise technical jargon. These definitions primarily represent usages of words in this guide, and may not be complete by industry standards.

ANSI---American National Standards Institute.

ASCII---American Standard Code for Information Interchange.

Aspect ratio---Specifies the height-to-width relationship of characters. For example, characters with an aspect ratio of 2.0 are twice as tall as they are wide.

CGM---Computer Graphics Metafile. A metafile that conforms to American National Standard ANSI X3.122-1986.

Color index---A nonnegative integer assigned a color value in a color table.

Color map--- *See* Color table.

Color model---*See* Color space.

Color palette---A table of colors that are referenced by the pixel values in an indexed color image. Color table, color map, and color palette may be used interchangeably.

Color space---A three-dimensional coordinate system, where coordinates in that system represent colors.

Color table---A list that associates RGB (red, green, blue) triplets with nonnegative integer values called color indices. Each RGB triplet defines a single color and RGB values are normalized between 0. and 1. where each value represents the intensity of the red, green, or blue hue in the color. Color table, color map, and color palette may be used interchangeably.

Color value---A value in a color model.

Dash pattern---The sequence of dashes, dots, spaces, and text used to represent a line. Dash patterns can be selected using the routines GSLN, DASHDB, and DASHDC.

Data coordinates---A coordinate system defining the location of data in an input data array. The mapping routines provided by NCAR Graphics utilities such as Conpack, Streamlines, and Vectors transform data coordinate space to user coordinate space.

Digital image---*See* Raster image.

Fontcap---A file that contains detailed information that can be used to plot characters. Fontcaps can have a human-readable ASCII form or a binary form that is readable by **ctrans**.

Fractional coordinates---The NCAR Graphics coordinate system in which each axis ranges from 0. to 1. NCAR Graphics fractional coordinates are the logical equivalent of GKS normalized device coordinates.

Function codes---Character codes that are embedded in the text string argument passed to the PLCHHQ or PLHIQU routines. These codes control text characteristics such as font, superscripting, subscripting, orientation, and position.

GKS---Graphical Kernel System. American National Standard ANSI X3.128-1985, which standardizes basic graphics functionality.

GKS segment---A collection of graphic display elements that may be manipulated as a whole.

GKS workstation---An abstraction of a physical device that allows for the display of lines, text, closed polygons, and markers on a rectangular surface of fixed resolution.

Graphcap---A file that contains detailed information that can be used to define the capabilities of a specific plotting device. Graphcaps can have a human-readable ASCII form or a binary form that is readable by **ctrans**.

Grid coordinates---A coordinate system coincident with the assumed dimensional boundaries of an input data array. NCAR Graphics utilities such as Conpack, Streamlines, and Vectors provide parameters for mapping the grid coordinate boundaries into a data coordinate space defining the location of the data.

GUI---Graphical User Interface. In NCAR Graphics, all GUIs run on X Window System terminals or workstations.

HDF---Hierarchical Data Format. A file format for storing raster data developed by the National Center for Supercomputing Applications (NCSA).

HLS---Hue, Lightness, and Saturation color model.

HSV---Hue, Saturation, and Value color model.

Identity transformation---A window-to-viewport mapping in which both the window and viewport range from 0. to 1. in both dimensions.

Indexed color imagery---Imagery where the pixel values reference RGB colors contained in a color table. *See* Raster images.

Internal parameters---Variables that control the behavior of NCAR Graphics utilities. These variables can only be accessed via FORTRAN 77 common blocks. See the specific utility's documentation for a description of how to access the utility's parameters.

Isosurface---Perspective representation of 3-D "volume" of data. The isosurface is defined by a threshold value, and it passes through all data points that equal this value. The routines EZISOS and ISOSRF draw isosurfaces. *See* Surface.

LLU---Low Level Utility. Refers to any of the Fortran-callable utilities in NCAR Graphics.

Log scaling---Describes a Cartesian axis in which values range logarithmically from one end to the other.

Markers---Symbols (dots, plus signs, asterisks, circles, crosses, and FORTRAN 77 characters) drawn using routines such as GPM, POINT, and POINTS.

Metafile---Generally synonymous with NCGM in this document. A metafile can be any mechanism for retaining and transporting graphical data and control information. This information contains a device-independent description of one or more pictures.

Mirror imaging---Describes a Cartesian X or Y axis in which values decrease from left to right or top to bottom.

NCGM---NCAR Computer Graphics Metafile. A CGM created by NCAR Graphics, using the NCAR Graphics GKS library. This is an ANSI standard CGM with an NCAR-specific wrapper on it.

NDC---Normalized Device Coordinates. A device-independent GKS coordinate system in which the horizontal and vertical coordinate ranges are both 0. to 1.

Normalization transformation---The transformation that occurs between the world coordinate window and the normalized device coordinates.

Parameters---Variables that control the behavior of NCAR Graphics utilities and are typically accessed via parameter-access routines that can retrieve or set the parameter value.

PAU---Plotter Address Unit. A measure of length. By default, 1 PAU is equal to 1/1024th of the maximum screen size. NCAR Graphics utilities are migrating away from the use of the PAU terminology.

Pixel---Picture element.

Polyline---A curve made up of line segments. Polylines are drawn using the routines GPL, CURVE, and CURVED.

Polymarkers---A series of markers drawn using the GKS routine GPM.

PWRITX database---One of the character databases used by the Plotchar utility. The database contains the complex and duplex character sets.

Raster image---A digital image is a 2-D array of pixels (picture elements), where each pixel either directly or indirectly defines the color of that (X, Y) location. In the case of "indexed-color" imagery, the pixel values are mapped into a color by using the pixel values as indices into a separate color palette. In "true-color" imagery, each pixel value maps directly into the actual pixel color (RGB, HSV, and so on).

RGB---Red, Green, and Blue color model.

Special value---The number that represents missing or special data in a dataset. Typically, this value is outside the range of the "normal" dataset values.

SPPS---An NCAR Graphics utility that draws points, lines, and text, and supplies a few low-level shortcuts, such as opening and closing GKS.

SPPS polyline buffer---An area of memory that contains the commands for polyline segments so that NCGM files can be kept small.

Stereo pairs---Two identical images that are drawn at some offset on a single or separate frame.

Subsampling---A method for reducing the size of a dataset by extracting a subset of the data.

Surface---A perspective representation of 2-D data, where each value of the data represents height in the surface. The routines EZSRFC and SRFACE draw surfaces. See the "Drawing Surfaces and Isosurfaces" chapter.

True color imagery---Imagery where each pixel has a red, green, and blue value. *See* Raster images.

User coordinates---The NCAR Graphics Cartesian coordinate system in which you perform your calculations. This is identical to the GKS world coordinate system when mirror imaging and/or log scaling of an axis is not used. In user coordinates, windows can be defined in which the coordinates decrease from left to right or top to bottom (mirror imaging) or range in a logarithmic scale (log scaling). GKS world coordinates, however, always increase from left to right and bottom to top in a linear fashion. See Chapter 8 "Coordinate systems in NCAR Graphics" and Appendix A "The Use of X/Y Coordinates in NCAR Graphics" for more information.

WC---World Coordinates. The GKS Cartesian coordinate system in which you perform your calculations. See Chapter 8 "Coordinate systems in NCAR Graphics" and Appendix A "The Use of X/Y Coordinates in NCAR Graphics" for more information.

X Window System---A network-based graphics window system that was developed at the Massachusetts Institute of Technology (MIT) in 1984.

YIQ---Broadcast television color model.

Appendix A: The use of X/Y coordinates in NCAR Graphics

Introduction

This appendix is intended to clear up some of the confusion about the way in which X/Y coordinates used in calls to NCAR Graphics routines are mapped to a graphics device and to collect in one place general information about the subject. It will not cover all of the details of every routine mentioned. This is one of the most difficult areas for novice users of NCAR Graphics; even experienced users sometimes have difficulties.

For historical reasons, the situation is a bit complicated (but not as complicated as one might think). Four or five different coordinate systems are mentioned in the documentation (actually, one of these now accounts for almost all of the usage). Some of the higher-level utilities change parameters that affect the mapping of coordinates used in subsequent calls; there are good reasons for this, but the documentation does not always make it clear what is going on.

Required abstractions: Plotter frame, world and user coordinates

Each graphics device uses a highly specific coordinate system; for example, the device on which I am typing now, when used in graphics mode, requires integer X coordinates ranging from 0 at the left edge of the screen to 1152 at the right edge of the screen and integer Y coordinates ranging from 0 at the top of the screen to 900 at the bottom of the screen. It would be most annoying if, every time I drew a picture, I had to use X and Y coordinates in these ranges. Therefore, two simplifying abstractions are introduced.

First, the NCAR Graphics user assumes that he or she is working with an idealized graphics device having a square display area (often referred to in the documentation as the "plotter frame"). This display area may be addressed using what are called "normalized device coordinates" (NDCs); these are X/Y coordinate pairs having the following characteristics: The X coordinate is a real number between zero and one; zero refers to the left edge of the area and one refers to the right edge of the area. The Y coordinate is also a real number between zero and one; zero refers to the bottom edge of the area and one refers to the top edge of the area. The way in which normalized device coordinates are mapped to the actual coordinates required by a given device is outside the scope of this document; typically, it is determined by the translator being used to drive the device and the "graphcap" which describes the graphical capabilities of that device.

Second, although one can work directly with normalized device coordinates (and may sometimes want to do so), this is not usually done. One works instead with what are called "world coordinates" (in the case of a few of the routines in GKS proper) or with what are called "user coordinates" (in the case of almost all other NCAR Graphics routines). Both of these are Cartesian coordinate systems using real numbers.

The way in which world coordinates and user coordinates are mapped into normalized device coordinates is controlled by the user (within limits) and is the subject of this document. The overall effect is to allow the user to work in a natural coordinate system (perhaps, for example, the system in which the data are given); NCAR Graphics routines called transform the coordinates used into the appropriate normalized device coordinates.

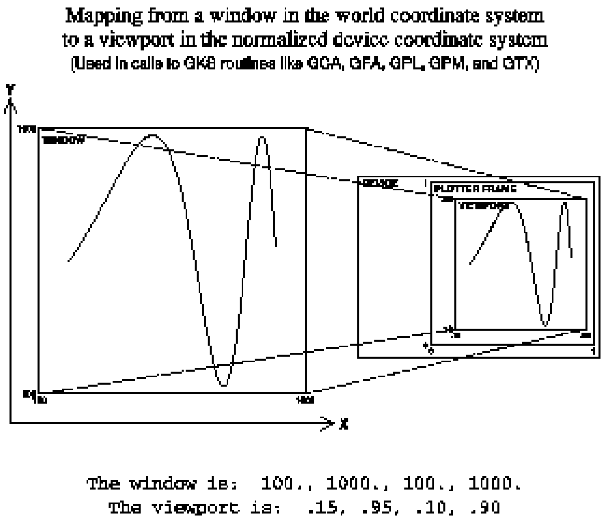
X/Y coordinates in calls to GKS routines (world coordinates)

Figure 1 shows how world coordinates in calls to GKS routines are mapped to the plotter frame. The mapping of world coordinates is defined by the current "normalization transformation." Each normalization transformation is associated with a nonnegative integer; currently, the NCAR Graphics GKS package only allows the use of normalization transformations 0 and 1.

To define the normalization transformation associated with an integer greater than 0, call the GKS routine GSVP to define its "viewport" and the GKS routine GSWN to define its "window." The viewport is defined by four real numbers between 0 and 1, specifying the rectangular portion of the plotter frame that is to be used; the window is defined by four real numbers, specifying a rectangular portion of the "world coordinate system" that is to be linearly mapped to the viewport. Both sets of numbers are in the order "Xmin, Xmax, Ymin, Ymax," where Xmin is less than Xmax and Ymin is less than Ymax.

Normalization transformation 0 cannot be changed. It is the identity transformation, which maps the unit square (in which all X/Y coordinates are normalized device coordinates, between 0 and 1, inclusive) back into the unit square. Its viewport and window are both defined by the four numbers 0., 1., 0., and 1., in that order.

To make a given normalization transformation the "current" one (to "select" it), call the GKS routine GSELNT. To determine the largest usable normalization transformation number, call the GKS routine GQMNTN. To determine which normalization transformation is the current one, call the GKS routine GQCNTN. To determine the window and viewport of a given normalization transformation, call the GKS routine GQNT.



GKS routines which require the use of world coordinates include the following primitives:

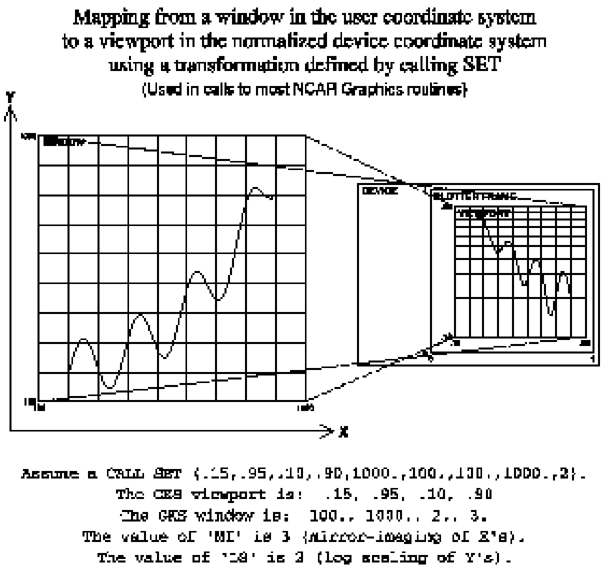
- GCA draws a "cell array"; it colors a rectangular area on the plotter frame, using a rectangular array of color indices provided by the caller.
- GFA draws a "fill area"; it fills a polygon defined by a set of points.
- GPL draws a "polyline" connecting a set of points in a given order.
- GPM draws a "polymarker" marking the position of each point in a set.
- GTX draws a character string at a given position.

These are the only routines the average user is likely to encounter which require world coordinates as arguments.

X/Y coordinates in calls to most NCAR Graphics routines (user coordinates)

Figure 2 shows how user coordinates are mapped to the plotter frame (for the moment, ignore references to the routine SET, which will be discussed in the next section). User coordinates are interpreted using the window and viewport of the current GKS normalization transformation, but the mapping from the window to the viewport is further modified as implied by the current values of two internal parameters, MI and LS, that provide two valuable features of the pre-GKS version of NCAR Graphics:

- The parameter MI allows for "mirror imaging" (mapping increasing X values from right to left, instead of from left to right, in the viewport, or increasing Y values from top to bottom, instead of from bottom to top, in the viewport, or both).
- The parameter LS allows for "log scaling" (substituting the base-10 logarithms of X and/or Y coordinates for the coordinates themselves).



The current values of MI and LS may be set by calling the SPPS routine SETUSV and retrieved by calling the SPPS routine GETUSV. Possible values are as follows:

MI = 1 - X mapping normal, Y mapping normal (the default)
MI = 2 - X mapping normal, Y mapping inverted
MI = 3 - X mapping inverted, Y mapping normal
MI = 4 - X mapping inverted, Y mapping inverted

LS = 1 - X mapping linear, Y mapping linear (the default)
LS = 2 - X mapping linear, Y mapping logarithmic
LS = 3 - X mapping logarithmic, Y mapping linear
LS = 4 - X mapping logarithmic, Y mapping logarithmic

Example: Assume that normalization transformation 1 is the current one, that its viewport is defined by the four numbers 0., .5, .5, and 1. (which refers to the upper left quadrant of the plotter frame), and that its window is defined by the four numbers -1., 1., 0., and 2. Assuming that MI and LS have default values, the statement

```
CALL POINT (0.,1.)
```

will draw a point in the center of the viewport (which is just what the equivalent call to the GKS routine GPM would do). Changing the value of LS to 2 would move the point drawn by POINT to the center of the bottom edge of the viewport (because the change in LS makes the Y mapping logarithmic and the base-10 logarithm of 1. is 0.). Changing the value of MI to 2 would then move the point to the center of the top edge of the viewport (because the change causes mirror-imaging of Y coordinates). A call to GPM would not be affected by the changes in LS and MI.

> **To repeat:** X/Y coordinates in calls to GKS routines are not affected by the values of LS and MI; these routines expect world coordinates (as defined by the GKS standard), not user coordinates. The five GKS routines you are most likely to use are GCA, GFA, GPL, GPM, and GTX.

Note: Even when normalization transformation 0 is the current one, the flags LS and MI affect the interpretation of X/Y coordinates in calls to NCAR graphics routines that expect user coordinates. If your object in using normalization transformation 0 is to use normalized device coordinates in a meaningful way, then you will probably (but not necessarily) want to reset LS and MI to their default values.

The routine SET

The SPPS routine SET provides a way to set the current viewport and window and the values of the parameters MI and LS all at once. (That is to say, it allows one, with a single call, to completely define the user coordinate system.) SET is meant to have the same effect as the routine SET in the pre-GKS version of NCAR Graphics. Consider the statement

```
CALL SET (VPL,VPR,VPB,VPT,WDL,WDR,WDB,WDT,LLF)
```

VPL, VPR, VPB, and VPT are the four numbers defining the desired viewport; each of the values is between 0 and 1, inclusive; the suffixes "L", "R", "B", and "T" refer to the left, right, bottom, and top edges of the viewport, respectively. It must be the case that VPL is less than VPR and that VPB is less than VPT.

WDL, WDR, WDB, and WDT define the desired window in the user coordinate system. The desired value of the mirror-imaging parameter MI is implicit in the ordering of these values. If WDL is less than WDR, then X coordinates are not to be mirror-imaged; otherwise they are. Similarly, if WDB is less than WDT, then Y coordinates are not to be mirror-imaged; otherwise they are. Note that WDL and WDR specify a range of values of X coordinates prior to the taking of logarithms, if any, specified by LLF; similarly, WDB and WDT specify a range of values of Y coordinates prior to the taking of logarithms, if any, specified by LLF.

The final argument, LLF, specifies the desired value of the log-scaling parameter LS; its value must be one of 1, 2, 3, or 4. Note that, if the value of LLF is other than 1, then either the first two, or the last two, or all four of the numbers defining the window of the GKS normalization transformation will be base-10 logarithms of the arguments WDL, WDR, WDB, and WDT.

The routine SET always redefines normalization transformation 1 and makes it the current normalization transformation. The values of the arguments VPL, VPR, VPB, and VPT are used, in that order, to define the viewport of normalization transformation 1. The values of the arguments WPL, WPR, WPB, and WPT (or their base-10 logarithms, depending on the value of LLF) are used to define the window of normalization transformation 1; the order of these values may have been changed.

Examples: Below are shown four SET calls; each is followed by the resulting values of the GKS viewport and window and the values of the SPPS parameters MI and LS.

```
CALL SET ( .1, .9, .3, .7 , -1., 1., 1., 10. , 1 )
```

GKS viewport: .1, .9, .3, .7
GKS window: -1., 1., 1., 10.
Value of MI: 1
Value of LS: 1

```
CALL SET ( .1, .9, .3, .7 , 1., -1., 1., 10. , 1 )
```

GKS viewport: .1, .9, .3, .7
GKS window: -1., 1., 1., 10.
Value of MI: 3
Value of LS: 1

```
CALL SET ( .1, .9, .3, .7 , -1., 1., 1., 10. , 2 )
```

GKS viewport: .1, .9, .3, .7
GKS window: -1., 1., 0., 1.
Value of MI: 1
Value of LS: 2

```
CALL SET ( .1, .9, .3, .7 , 1., -1., 10., 1. , 2 )
```

GKS viewport: .1, .9, .3, .7
GKS window: -1., 1., 0., 1.
Value of MI: 4
Value of LS: 2

Note: You must remember that, when you call SET, it will affect the values of the internal parameters MI and LS. If you then call the GKS routine GSELNT to select a different normalization transformation, the parameters MI and LS will not be reset and will therefore affect the interpretation of user coordinates in subsequent calls. In particular, if you use mirror-imaging and/or log scaling and then call GSELNT to make normalization transformation 0 the current one, the mirror-imaging and/or log scaling will still be in effect unless you also reset MI and/or LS to 1 using SETUSV calls.

See also the example code and plot at the end of this document.

The routine GETSET

The SPPS routine GETSET provides a way to retrieve the nine values that, when used as arguments in a subsequent SET call, will restore the current mapping of user-system X/Y coordinates. It is called using a statement of the form

```
    CALL GETSET (VPL,VPR,VPB,VPT,WDL,WDR,WDB,WDT,LLF)
```

It is important to realize that, if the GETSET call is executed when normalization transformation 0 is the current one, a subsequent SET call using the nine returned values will not exactly restore the underlying GKS parameters; instead, it will redefine normalization transformation 1 to have the same effect as normalization transformation 0.

See also the example code and plot at the end of this document.

Other coordinate systems used in NCAR Graphics

Three other coordinate systems used in NCAR Graphics are described in the "Coordinate systems in NCAR Graphics" section of this guide, and in the *User's Guide for NCAR GKS-0A Graphics* programmer document. Thumbnail descriptions follow. All three of these systems are used to address points on the entire plotter frame.

1. The "fractional" coordinate system uses real X coordinates ranging from 0 at the left edge to 1 at the right edge and real Y coordinates ranging from 0 at the bottom edge to 1 at the top edge. These correspond to what are called "normalized device coordinates" in GKS. This system is used frequently.

2. The "metacode" coordinate system (PAUs) uses integer X coordinates ranging from 0 at the left edge to 32,767 at the right edge and integer Y coordinates ranging from 0 at the bottom edge to 32,767 at the top edge. This coordinate system is a relic which has not been entirely eradicated. Users should avoid using it.

3. The "plotter" coordinate system uses integer X coordinates ranging from 1 at the left edge to 2**MX at the right edge and integer Y coordinates ranging from 1 at the bottom edge to 2**MY at the top edge. MX and MY are the values of internal parameters that can be set using SETUSV and retrieved using GETUSV. The default values of MX and MY are 10 and 10. This coordinate system is a relic which has not been entirely eradicated. Users should avoid using it.

Each of the following SPPS routines uses one of these coordinate systems:

- FL2INT, given an X/Y coordinate pair in the user system, returns an X/Y coordinate pair in the metacode system.

- MXMY returns the metacode coordinates of the current pen position.

- PLOTIF raises or lowers the "pen," as directed, and then moves it to a specified position in the fractional coordinate system.

- PLOTIT raises or lowers the "pen," as directed, and then moves it to a specified position in the metacode coordinate system.

The fractional coordinate system is used in several high-level utilities to specify what portion of the plotter frame to use.

The 3D packages ISOSRF, ISOSRFHR, SRFACE, and THREED of course use three-dimensional coordinate systems.

Routines in the packages EZMAP and EZMAPA use latitude/longitude coordinates.

The SPPS conversion routines

SPPS provides functions to allow for easy mapping of an X or Y coordinate in one coordinate system to the corresponding X or Y coordinate in another coordinate system. A couple of them are very useful, including CFUX, CFUY, CUFX, and CUFY (the remainder are obsolete). Suppose that we wish to label a given point, whose coordinates are given in the user coordinate system, using a two-line label of the form

X*string of information*

where the "X" is to be centered on the point in question and the associated string of information is to be a little to the right of and below the X. Assume that the desired string of information is in a character variable CHRS. The code required would be something like the following:

```
CALL PLCHHQ (X,Y,'X',.02,0.,0.)
CALL PLCHHQ (CFUX(CUFX(X)+.01),CFUY(CUFY(Y)-.01),CHRS,.01,0.,-1.)
```

The high-quality character-writing routine PLCHHQ, from the package Plotchar, is used. In each of the first two arguments in the second call, we convert a coordinate from the user system to the fractional system, add a constant that is meaningful in the fractional system, and then convert the coordinate back into the user system.

Use of SET and GETSET by various utilities

Many of the NCAR Graphics utilities have as arguments X/Y coordinates that are expected to be in the user coordinate system, as defined by the current GKS window and viewport and the SPPS parameters MI and LS (all of which may have been defined by a call to SET, of course).

Many of the utilities themselves call SET. Some of these first save the current definition of the user coordinate system (by a call to GETSET) and then restore that definition (by another call to SET) before returning to the caller; others (intentionally) do not. Even those that do save and restore the original SET call may change the current normalization transformation number and the definition of normalization transformation number 1 (for reasons detailed in the descriptions of the routines SET and GETSET, above).

Below are listed all of the NCAR utility packages. Each is accompanied by a short description of ways in which the package makes use of X/Y coordinates in the user coordinate system and of the routines SET and GETSET. Remember that the user coordinate system may have been defined by a SET call or by the equivalent GKS calls to define the window and viewport of the current normalization transformation and the SPPS calls to define the values of the parameters MI and LS; for the sake of brevity, the phrase "the most recent SET call" will be used instead of the longer "the most recent SET call or equivalent calls to ...".

Areas:

The routines ARDRLN, AREDAM, and ARGTAI have arguments that are X/Y coordinates in the user coordinate system; these arguments are interpreted according to the most recent SET call.

The routine ARDRLN calls the user routine specified by the argument "LPR" (which is expected to examine a polyline and either draw it or ignore it) with X/Y coordinates in the fractional system; prior to each such call, ARDRLN calls SET to define the user system to be identical to the fractional system (so that the X/Y values provided will be usable in calls to any of the routines that draw polylines). Similarly, the routine ARSCAM calls the user routine specified by the argument "APR" (which is expected to examine an area and either fill it or ignore it) with X/Y coordinates in the fractional system. Again, ARSCAM calls SET to define the user system to be identical to the fractional system (so that the X/Y values provided will be usable in calls to any of the routines that do area fill. In both cases, the original SET call is saved (by calling GETSET) before the call to the user routine and restored (by another call to SET) after the call to the user routine.

Autograph:

The routines AGCURV, AGSTUP, EZY, EZXY, EZMY, EZMXY, and IDIOT have arguments that are ultimately used as X/Y coordinates in the user coordinate system. Each of these routines does a call to SET before using these coordinates.

A call to the routine AGSTUP determines the necessary SET call and does it; certain values of Autograph's internal parameter SET. may direct that some or all of the arguments in the SET call are to be obtained by calling GETSET. The routines AGBACK and AGCURV re-call SET, using arguments saved in labelled COMMON. Each of the routines EZY, EZXY, EZMY, EZMXY, and IDIOT calls AGSTUP, so each also indirectly calls SET. In no case is the original SET call saved or restored; this allows a user to draw objects on an Autograph plot using the coordinates of the data points as user coordinates.

The routine AGUTOL, which is called by Autograph and defines the mapping between the user system along a specified axis and the "label coordinate system" on that axis, may be called with an input value that is either an X or a Y coordinate in the user coordinate system. The routine AGPWRT is called by Autograph to draw character strings. User-supplied versions of these two routines may assume that X/Y coordinates defined by their arguments are commensurate with the arguments in the last SET call (which will have been done by the calling routine).

Colconv:

No arguments are X/Y coordinates in the user coordinate system.

SET is not called by Colconv.

Conpack:

Each of the routines CPRECT, CPSPS1 (also known as CPSPRS), and CPSPS2, one of which is called as the first step in drawing a contour plot, determines the SET call that will be required for subsequent drawing of the contour plot and does the necessary call. (The internal parameter SET may be given a value saying that some or all of the SET call arguments are to be obtained from a call to GETSET.) Conpack routines called subsequently that depend on the proper SET call to have been done do it for themselves, using variables saved in Conpack's labelled COMMON blocks; routines that do this are CPBACK, CPCICA, CPCLAM, CPCLDM, CPCLDR, CPCLTR, CPLBAM, CPLBDR, and CPPKLP. In no case is the original SET call saved and restored; this allows a user to draw objects on a Conpack plot using meaningful user coordinates.

X/Y coordinates used by Conpack routines to draw objects on the contour plot are user coordinates and must be commensurate with the SET call. Various desirable effects may be created (for example, the mapping of Conpack output onto an Ezmap background); creating these effects may require setting one or more of the Conpack parameters MAP, SET, VPL, VPR, VPB, VPT, VPS, WDL, WDR, WDB, WDT, XC1, XCM, YC1, and YCN. It may also require providing one's own version of the routine CPMPXY.

The routines CPCNRC and CPEZCT, which are intended to simulate the older routines CONREC and EZCNTR, do save and restore the original SET call. In this respect, they are inconsistent with the rest of Conpack, but consistent with the routines they are meant to replace.

Conran_family:

Each member of this family has as arguments arrays of X and Y coordinates which are ultimately used as X/Y coordinates in the user coordinate system. Normally, no SET call is expected to have been done by the user. Certain values of the internal parameter SCA will require a SET call to have been done.

The parameter SCA determines whether or not SET is to be called and, if so, which of the arguments are obtained by calling GETSET. If SET is called, the original SET call is not restored; this allows a user to draw objects on a plot drawn by one of these routines using meaningful user coordinates.

Dashline:

The routines FRSTD, VECTD, CURVED, LINED, and PWRTM have arguments that are X/Y coordinates in the user coordinate system; they are interpreted according to the last SET call.

No calls to SET are done by any of the routines in these packages.

Ezmap:

No arguments are X/Y coordinates in the user coordinate system.

SET is called directly by MAPINT and indirectly by MAPDRW (when it calls MAPINT). By calling MAPRS, a user may force restoration of the last SET call done by MAPINT. The fact that MAPINT calls SET allows objects to be drawn on an Ezmap background. The routines MAPFST, MAPGRD, MAPIT, MAPIQ, MAPLBL, MAPLOT, and MAPVEC all draw such objects and they all depend on the SET call to have been done by the earlier call to MAPINT. Given the latitude and longitude of a point on the globe, the routine MAPTRN allows one to retrieve an X/Y coordinate pair that is commensurate with that SET call. A call to SET that follows the call to MAPINT and precedes calls to other Ezmap routines is likely to cause problems.
The viewport to be used by Ezmap is specified by a call to MAPPOS. Typically, the map produced will not fill the entire viewport. This is because each projection has a natural aspect ratio and it is unlikely that the viewport chosen will be of exactly the shape required to hold the map specified by the call to MAPSET. To distort the map to fill a specified viewport, you can, after the call to MAPINT, insert the two statements

```
CALL GETSET (XXL,XXR,XXB,XXT,WDL,WDR,WDB,WDT,LLF)

CALL SET (VPL,VPR,VPB,VPT,WDL,WDR,WDB,WDT,LLF)
```

where VPL, VPR, VPB, and VPT define the desired viewport. This forces output from subsequent calls to object-drawing routines in Ezmap to fill that viewport, rather than the one defined by MAPINT.
All of the routines MAPBLA, MAPGRM, MAPIQA, MAPIQM, MAPITA, and MAPITM depend on an earlier call to MAPINT to have done a SET call. A call to SET that follows the call to MAPINT and precedes a call to one of the above routines is likely to cause problems.

Gflash:

No arguments are X/Y coordinates in the user coordinate system.

Gflash does not call SET.

Gridall:

The routines GRIDAL and HALFAX have arguments that are
X/Y coordinates in the current user coordinate system.

No routines in this package call SET. The routine GRIDAL, which may be called directly or indirectly (as a result of a call to any of the routines GRID, GRIDL, HALFAX, PERIM, or PERIML), assumes that SET has been called; it calls GETSET to obtain the values that determine where the grid is to be drawn and what labels, if any, are to be written on it.

Halftone:

No arguments are X/Y coordinates in the user coordinate system.

The argument "NPRM" determines whether or not the HAFTON routine calls SET and, if so, which of the arguments are obtained by calling GETSET. When HAFTON does call SET, it saves and restores the original SET call.

Histogram:

No arguments are X/Y coordinates in the user coordinate system.

No routines in this package call SET.

Isosurface:

No arguments are X/Y coordinates in the user coordinate system.

ISOSRF does not call SET. 3D coordinates are mapped to 2D coordinates in the metacode system and PLOTIT is called to draw the desired objects. The PWRZI routine does not call SET.

Labelbar:

No arguments are X/Y coordinates in the user coordinate system.

LBLBAR calls SET in such a way that it can work completely in the fractional coordinate system; it saves and restores the original SET call.
The routine LBFILL is called repeatedly by LBLBAR to fill the rectangles that make up the label bar; the user may supply a version of this routine to do the fill in the required manner. X/Y coordinates in calls to LBFILL are given in the fractional coordinate system; the required SET call may be assumed to have been done.

Plotchar:

PLCHHQ, PLCHMQ, and PLCHLQ have arguments that are
X/Y coordinates in the user coordinate system.

No calls to SET are done by Plotchar routines.

Scrolled_title:

No arguments are X/Y coordinates in the user coordinate system.

No calls to SET are done by the routines in this package.

Softfill:

SFSGFA and SFWRLD have arguments that are X/Y coordinates in the user coordinate system, as defined by the last SET call. These coordinates are converted to the fractional system and then passed on to the routine SFNORM.

SFNORM calls SET to make the user coordinate system identical to the fractional coordinate system; it saves and restores the original SET call.

SPPS:

The routines CURVE, FL2INT, FRSTPT, LINE, POINT, POINTS, PWRIT, VECTOR, and WTSTR have arguments that are X/Y coordinates in the user coordinate system, as defined by the last SET call.

Under certain conditions, CURVE and POINTS call SET; when they do, they save and restore the original SET call.

Streamlines:

In general, the approach Streamlines uses for coordinate system set-up is consistent with the model established by the Conpack utility.

The Streamlines initialization routine STINIT determines whether to call the SET routine based on the value of the internal parameter SET. When STINIT does the SET call, the user can influence how it is called by setting some or all of the viewport and window control parameters, VPL, VPR, VPB, VPT, VPS,WDL, WDR, WDB, and WDT. The routine that actually draws the streamlines, STREAM, assumes that the proper SET call has been performed, and, as needed, retrieves information about the current coordinate state from variables stored in Streamlines' named common blocks. Except when invoked through STRMLN or EZSTRM, interfaces that have been retained for compatibility with older versions of this utility, Streamlines does not save and restore the original SET call; this allows a user to draw objects on a Streamlines plot using meaningful user coordinates.

By default, Streamlines uses X coordinates between 1. and REAL(M) and Y coordinates between 1. and REAL(N), where M and N are the assumed sizes of the input data arrays. In other words, an identity mapping is established between the array grid boundaries and the user coordinate boundaries. By appropriately setting the parameters, XC1, XCM, YC1, and YCM, however, the user can establish a mapping from the array grid indices to a coordinate system more natural for the data, such as longitude and latitude. Moreover, when the MAP parameter has a non-default value, Streamlines can send the streamlines through a user-definable mapping transformation before arriving at user coordinate space. Two transformations are pre-defined: a mapping through the various Ezmap projections, and a polar coordinate mapping. The user may implement other mapping transformations by modifying the user-modifiable routines STUMXY, STUIXY, and STUMTA.

Internally, Streamlines maps the directional component of the streamlines using normalized device coordinates as a reference coordinate system. This is because, unlike user coordinate space, NDC space is guaranteed to be uniform and rectangular. However, the coordinates output to the drawing routines are effectively always in user coordinates. Whenever the default array grid to user coordinate space mapping is modified, either directly or indirectly, the user needs to ensure that the coordinate system boundaries align across the transformation.

The parameter, TRT, only implemented for certain settings of the MAP parameter, allows the user to control mapping of the directional component of the streamline. Depending on the value of

TRT, the user can choose between a space mapping, where the directional component transforms, and a location mapping, where the mapping simply establishes the position of the grid points on an underlying rectangular coordinate system.

Surface:

No arguments are X/Y coordinates in the user coordinate system. The PWRSZ routine calls SET; it saves and restores the original SET call.

SRFACE calls SET; it saves and restores the original SET call.

Threed:

No arguments are X/Y coordinates in the user coordinate system.

POINT3 calls SET; it saves and restores the original SET call. Other routines in this package do not call SET; 3D coordinates are mapped to 2D coordinates in the metacode system, and PLOTIT is called to draw the desired objects.

Vectors:

In general, the approach Vectors uses for coordinate system set-up is consistent with the model established by the Conpack utility.

The Vectors initialization routine VVINIT determines whether to call the SET routine based on the value of the internal parameter SET. When VVINIT does the SET call, the user can influence how it is called by setting some or all of the viewport and window control parameters, VPL, VPR, VPB, VPT, VPS, WDL, WDR, WDB, and WDT. The routine that actually draws the vectors, VVECTR, assumes that the proper SET call has been performed, and, as needed, retrieves information about the current coordinate state from variables stored in Vectors' named common blocks. Except when invoked through VELVCT, VELVEC, or EZVEC, interfaces that have been retained for compatibility with older versions of this utility, Vectors does not save and restore the original SET call; this allows a user to draw objects on a Vectors plot using meaningful user coordinates.

By default, Vectors uses X coordinates between 1. and REAL(M) and Y coordinates between 1. and REAL(N), where M and N are the assumed sizes of the input data arrays. In other words, an identity mapping is established between the array grid boundaries and the user coordinate boundaries. By appropriately setting the parameters, XC1, XCM, YC1, and YCM, however, the user can establish a mapping from the array grid indices to a coordinate system more natural for the data, such as longitude and latitude. Moreover, when the MAP parameter has a non-default value, Vectors can send the vectors through a user-definable mapping transformation before arriving at user coordinate space. Two transformations are pre-defined: a mapping through the various Ezmap projections, and a polar coordinate mapping. The user may implement other mapping transformations by modifying the user-modifiable routine, VVUMXY.

Internally, Vectors maps the directional component of the vectors using normalized device coordinates as a reference coordinate system. This is because, unlike user coordinate space, NDC space is guaranteed to be uniform and rectangular. However, the coordinates output to the drawing routines are effectively always in user coordinates. Whenever the default array grid to user coordinate space mapping is modified, either directly or indirectly, the user needs to ensure that the coordinate system boundaries align across the transformation.

The parameter TRT, only implemented for certain settings of the MAP parameter, allows the user to control mapping of the directional component of the vectors. Depending on the value of TRT, the user can choose between a space mapping, where the directional component transforms, and a location mapping, where the mapping simply establishes the position of the grid points on an underlying rectangular coordinate system.

An example illustrating the effects of MI and LS

The following example illustrates the effect of MI and LS on the mapping of X/Y coordinates from a window in the user coordinate system to a viewport on the plotter frame. Both the code and the resulting plot are given.

The plot shows a 4x4 array of rectangles, each containing a spiral drawn by a call to the NCAR Graphics routine CURVE. All the spirals are defined by the same set of X/Y coordinates in the user coordinate system. The value of the SPPS parameter MI varies from row to row and the value of the SPPS parameter LS varies from column to column, so that all possible combinations of the two parameters are shown.

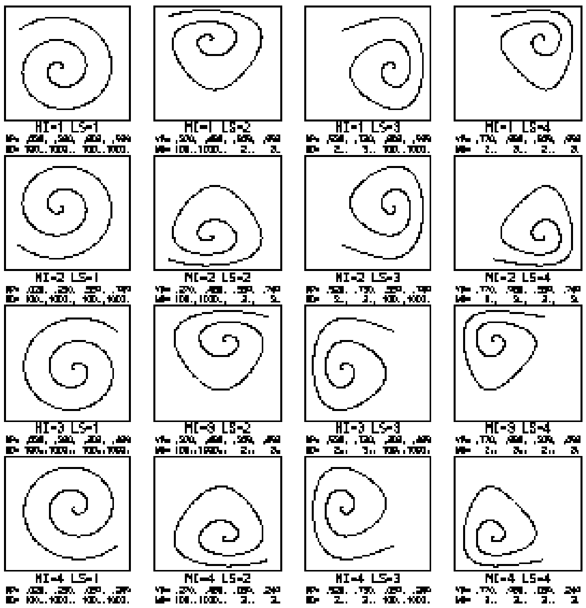
The mapping for each of the sixteen curves is specified by a SET call which defines the desired viewport (in the fractional coordinate system), the desired window (in the user coordinate system), and the desired value of LS; each SET call is followed by a SETUSV call defining the desired value of MI. (To define the desired value of MI using the SET call could have been done by changing the order of arguments 5-8, which define the window; we chose not to do this, but to use the combined effect of a SET call and a SETUSV call.)

Below each curve are shown the values of MI and LS used to draw that curve and the values of the parameters defining the GKS window and viewport, as obtained by a call to the GKS routine GQNT. Note that, although the values of arguments 5-8 in the SET call (which define the window in the user coordinate system) do not change, the values of the GKS parameters which define the window do change, to reflect the current value of LS.

```
               ============ CODE BEGINS ============
  PROGRAM EXMPLE
C
C Define arrays to hold data defining a spiral in the user coordinate
C system.
C
        DIMENSION X(476),Y(476)
C
C Define arrays to hold the numbers defining the viewport and window,
C as retrieved from GKS.
C
        DIMENSION VIEW(4),WIND(4)
C
C Define a character variable in which to construct labels.
C
        CHARACTER*26 CHRS
C
C Open GKS.  Open and activate a single metacode workstation.
C
        CALL OPNGKS
C
C Turn off clipping at the edges of the viewport (which GKS does by
C default).
```

```
C
        CALL GSCLIP (0)
C
C Define the X and Y coordinates of a spiral in the user coordinate
C system.  It lies in a rectangular region bounded by the lines
C "X=100", "X=1000", "Y=100", and "Y=1000".
C
        DO 101 I=1,476
          THETA=.031415926535898*FLOAT(I-1)
          X(I)=500.+.9*FLOAT(I-1)*COS(THETA)
          Y(I)=500.+.9*FLOAT(I-1)*SIN(THETA)
  101   CONTINUE
C
C Loop through the possible values of 'LS'.
C
        DO 103 ILS=1,4
C
C Define the fractional coordinates of the left and right edges of the
C viewport.
C
          VPL=REAL(ILS-1)/4.+.020
          VPR=REAL(ILS  )/4.-.020
C
C For each of the possible values of 'LS', loop through the possible
C values of 'MI'.
C
          DO 102 IMI=1,4
C
C Define the fractional coordinates of the bottom and top edges of the
C viewport.
C
            VPB=REAL(4-IMI)/4.+.059
            VPT=REAL(5-IMI)/4.-.001
C
C Outline the viewport.  PLOTIF expects fractional coordinates.
C
            CALL PLOTIF (VPL,VPB,0)
            CALL PLOTIF (VPR,VPB,1)
            CALL PLOTIF (VPR,VPT,1)
            CALL PLOTIF (VPL,VPT,1)
            CALL PLOTIF (VPL,VPB,1)
C
C Call SET to define the mapping from the user system to the plotter
C frame.  The SET call specifies 'MI' = 1 (since the value of argument
C 5 is less than that of argument 6 and the value of argument 7 is less
C that of argument 8).  The SETUSV call overrides this to obtain the
C desired value.
C
            CALL SET    (VPL,VPR,VPB,VPT,100.,1000.,100.,1000.,ILS)
            CALL SETUSV ('MI (MIRROR IMAGING FLAG)',IMI)
C
C Call the routine CURVE to draw the spiral.
C
            CALL CURVE  (X,Y,476)
C
C Label the curve.  First, write the values of 'MI' and 'LS'.  Note
C the use of CFUX and CFUY to map meaningful fractional coordinates
C to the user coordinates required by PLCHMQ.
C
            WRITE (CHRS,'(''MI='',I1,'' LS='',I1)') IMI,ILS
            CALL PLCHMQ (CFUX(.5*(VPL+VPR)),CFUY(VPB-.0120),
```

```
     +                                CHRS(1:9),.012,0.,0.)
C
C Retrieve the values defining the window and viewport, using GKS
C calls.
C
            CALL GQNT (1,IERR,WIND,VIEW)
C
C Write them out, too.
C
            WRITE (CHRS,'(''VP='',F5.3,3('','',F5.3))') (VIEW(I),I=1,4)
            CHRS( 4: 4)=' '
            CHRS(10:10)=' '
            CHRS(16:16)=' '
            CHRS(22:22)=' '
            CALL PLCHMQ (CFUX(.5*(VPL+VPR)),CFUY(VPB-.0320),
     +                               CHRS(1:26),.008,0.,0.)
            WRITE (CHRS,'(''WD='',F5.0,3('','',F5.0))') (WIND(I),I=1,4)
            CALL PLCHMQ (CFUX(.5*(VPL+VPR)),CFUY(VPB-.0480),
     +                               CHRS(1:26),.008,0.,0.)
C
C End of loop through the values of 'MI'.
C
  102     CONTINUE
C
C End of loop through the values of 'LS'.
C
  103   CONTINUE
C
C Terminate GKS.
C
        CALL CLSGKS
C
C Done.
C
        STOP
C
      END
              ============= CODE ENDS =============
```



Appendix B: UNIX environment variables

Before using NCAR Graphics, you must set either the NCARG_ROOT or the NCARG_BIN, NCARG_LIB, and the NCARG_INCLUDE environment variables. These variables address the directories where parts of NCAR Graphics are installed on your system and allow NCAR Graphics to be moved after it is installed.

NCAR Graphics also offers other environment variables as a shortcut, so you don't have to type the same information over and over again.

DISPLAY---This is really an X Window System environment variable, but it is listed here because it does affect NCAR Graphics output. DISPLAY sets the X Window workstation or X terminal name for displaying graphics.

GRAPHCAP---Specifies the graphics output device type for subsequent metafile translation.

NCARG_BIN---The directory where NCAR Graphics binaries are installed. If set, NCARG_BIN takes precedence over the NCARG_ROOT environment variable.

NCARG_GKS_BUFSIZE---Provides you with control over how CGM output is buffered.

NCARG_GKS_DEBUG---If set, will send debugging information about the GKS I/O subsystem to standard output.

NCARG_GKS_GENCGM---Allows you to request that the NCAR GKS package generate standard CGM rather than NCGM (NCAR-blocked CGM). Note that other NCAR Graphics utilities, such as translators, cannot read in generic CGMs yet.

NCARG_GKS_OUTPUT---Allows you to specify what file or process CGM output is directed to. By default, NCARG_GKS_OUTPUT is the file **gmeta**.

NCARG_INCLUDE---The directory where NCAR Graphics **include** files are installed.

NCARG_LIB---The directory where NCAR Graphics libraries and object files are installed.

NCARG_ROOT---The parent directory of the NCAR Graphics installation for most sites.

Appendix C: User entry points and functions

Note: Typing **man *name***, where ***name*** is any of the following table entries, will display the reference manual page. All **man** page names should be typed in lowercase.

The Areas utility

Areas
A set of routines to create an area map from a set of edges.

Areas_params
Contains a brief listing of the parameters of the Areas utility.

ARDBPA
Produces a picture of that part of the contents of the area map MAP which belongs to the group IGIP. The character string LABL will be used as a label for the picture.

ARDRLN
Draws a polyline that is masked by a given area map.

AREDAM
Adds edges to an area map.

ARGETI
Retrieves an integer parameter in Areas.

ARGTAI
Gets the area identifiers associated with a given point.

ARINAM
Initializes area map.

ARPRAM
Preprocesses an area map that has been initialized by a call to ARINAM and to which edges have been added by calls to AREDAM.

ARSCAM
Scans an area map from left to right, extracting definitions of all the areas in the map.

ARSETI
Sets an integer parameter in Areas.

The Autograph utility

Autograph
Draws graphs, each with a labeled background and each displaying one or more curves.

Autograph_params
Briefly describes all Autograph parameters.

AGBACK
Draws the background specified by the current values of the control parameters: the primary parameters with default values or with values supplied by the user, and the secondary parameters with values computed by AGSTUP.

AGBNCH
Provides an easy way to convert binary dash patterns into character dash patterns.

AGCHAX
Provides a way for the user to change the color, intensity, line style, etc., of various portions of the axes.

AGCHCU
Provides a way for the user to change the color, intensity, line style, etc., of curves drawn by Autograph.

AGCHIL
Provides a way for the user to change the color, intensity, text style, etc., of the informational labels.

AGCHNL
Provides a way for the user to substitute arbitrary character strings for the numeric labels generated by Autograph.

AGCURV
Draws a curve in a manner specified by the current values of the control parameters---the primary parameters with default values or with values supplied by the user, and the secondary parameters with values computed by AGSTUP.

AGDSHN
Provides an easy way to generate the names of parameters in the group DASH/PATTERN, for use in calls to AGSETC and AGGETC.

AGGETC
Allows a user program to retrieve (in effect) the character-string values of certain single parameters.

AGGETI
Allows a user program to retrieve the integer equivalent of the real value of a single parameter.

AGGETP
Allows a user program to get the values of a group of parameters containing one or more elements.

AGGETR
Allows a user program to retrieve the real value of a single parameter.

AGPWRT
Provides a way for the user to change the style of all text strings drawn by Autograph.

AGRSTR

Restores a saved state of Autograph.

AGSAVE

Saves the current state of Autograph for later restoration by AGRSTR.

AGSETC

Allows a user program to (in effect) store a character string as the value of a specified single parameter.

AGSETI

Allows a user program to store the real equivalent of an integer as the value of a single parameter.

AGSETP

Allows a user program to reset the values of a group of parameters containing one or more elements.

AGSETR

Allows a user program to store a real number as the value of a single parameter.

AGSTUP

Performs "set-up" tasks required before AGBACK and AGCURV may be called. Basically, AGSTUP examines the current values of the primary control parameters for errors and computes from them and from its arguments the values of secondary control parameters. The primary and secondary control parameters together determine how the routines AGBACK and AGCURV will behave.

AGUTOL

Provides a way for the user to change the user-system-to-label-system mapping for one or more of the four axes.

ANOTAT

Changes the values of certain primary control parameters, purportedly having to do with "annotation" of a graph.

DISPLA

Changes the values of certain primary control parameters purportedly having to do with the "display" of a graph.

EZMXY

Draws, in a manner determined by the current values of the control parameters, a complete graph of one or more curves, each defined by a set of points (XDRA(I),YDRA(I,J)) or (XDRA(I),YDRA(J,I)) or (XDRA(I,J),YDRA(I,J)) or (XDRA(J,I),YDRA(J,I)), depending on the current value of ROW), for I from 1 to NPTS. The curve number J runs from 1 to MANY. The argument GLAB may be used to specify a "graph label" to be placed at the top of the graph.

EZMY

Draws, in a manner determined by the current values of the control parameters, a complete graph of one or more curves, each defined by a set of points (I,YDRA(I,J)) or (I,YDRA(J,I)), depending on the current value of ROW), for I from 1 to NPTS. The curve number J runs from 1 to MANY. The argument GLAB may be used to specify a "graph label" to be placed at the top of the graph.

EZXY

Draws, in a manner determined by the current values of the control parameters, a complete graph of a single curve through the points (XDRA(I),YDRA(I)), for I from 1 to NPTS. The argument GLAB may be used to specify a "graph label" to be placed at the top of the graph.

EZY

Draws, in a manner determined by the current values of the control parameters, a complete graph of a single curve through the points (I,YDRA(I)), for I from 1 to NPTS. The argument GLAB may be used to specify a "graph label" to be placed at the top of the graph.

The Bivar utility

Bivar

Provides bivariate interpolation and smooth surface fitting for values given at irregularly distributed points. The resulting interpolating function and its first-order partial derivatives are continuous. The method employed is local, i.e., a change in the data in one area of the plane does not affect the interpolating function except in that local area. Also, the method gives exact results when all points lie in a plane.

IDBVIP

Produces interpolated values at points (XI(I), YI(I)), I=1, ..., NIP. This is useful for filling in missing data points on a grid.

IDSFFT

Performs smooth surface fitting when the projections of the data points in the X/Y plane are irregularly distributed in the plane.

The Colconv utility

Colconv

Allows a user to convert among the color models HSV, HLS, and YIQ.

HLSRGB

Converts a color specification given as the Hue, Lightness, and Saturation (HLS) values to Red, Green, and Blue (RGB) intensity values in the RGB color space.

HSVRGB

Converts a color specification given in the Hue, Saturation, and Value (HSV) color space to color values in the Red, Green, Blue (RGB) color space.

RGBHLS

Converts a color specification given in the Red, Green, Blue (RGB) color space to color values in the Hue, Lightness, Saturation (HLS) color space.

RGBHSV

Converts a color specification given in the Red, Green, Blue (RGB) color space to color values in the Hue, Saturation, and Value (HSV) color space.

RGBYIQ

Converts a color specification given in the RGB Red, Green, Blue, (RGB) color space to a color specification in the YIQ color space.

YIQRGB

Converts a color specification given in the YIQ coordinate system to the equivalent color specification in the Red, Green, Blue (RGB) coordinate system.

The Conpack utility

Conpack

Provides a sort of toolkit of Fortran subroutines that can be called in various combinations to draw different kinds of contour plots.

Conpack_params

This document briefly describes all Conpack parameters.

CPBACK

Draws a background for a contour plot.

CPCHCF

Provides user control as a constant-field message is drawn.

CPCHCL

Provides user control as contour lines are drawn.

CPCHHL

Provides user control as high and low labels are drawn.

CPCHIL

Provides user control as the informational label is drawn.

CPCHLL

Provides user control as contour line labels are drawn.

CPCICA

Incorporates into a user's cell array color indices determined by examining where the user's contours lie relative to the cell array.

CPCLAM

Adds contour lines to an area map. This is part of the process of drawing a solid-fill contour plot.

CPCLDM

Draws contour lines masked by an existing area map. The object of this may be simply to avoid drawing contour lines through label boxes, but the routine may be used for more complicated tasks, like limiting the drawing of contour lines to the ocean areas on an Ezmap background.

CPCLDR

Draws contour lines.

CPCLTR

Traces the contour lines at a given level and retrieves them for some sort of user-defined processing.

CPCNRC

Draws black-and-white contours with a single call. Simulates the behavior of the old routine CONREC; it has the same arguments and produces similar output.

CPDRPL

Provides a useful polyline-drawer for the routine CPCLDM.

CPEZCT

Draws black-and-white contours with a single call. Simulates the behavior of the old subroutine EZCNTR in Conrec_family; it has the same arguments and will produce similar output.

CPGETC

Retrieves the current value of a parameter of type CHARACTER.

CPGETI

Retrieves the current value of a parameter of type INTEGER.

CPGETR

Retrieves the current value of a parameter of type REAL.

CPLBAM

Adds label boxes (for the informational label, high and low labels, and contour-line labels) to an area map. The ultimate object of this will usually be to prevent contour lines drawn by CPCLDM from passing through labels or to prevent fill of the label boxes as contour bands are filled.

CPLBDR

Draws labels (an informational label, high and low labels, and line labels).

CPMPXY

Maps Conpack output from a rectangular coordinate system superimposed on the data grid to some other coordinate system.

CPPKCL

Picks a set of contour levels.

CPPKLB

Picks a set of labels for labeled contour levels.

CPRECT
Initializes the contouring of a rectangular array of data.

CPRSET
Resets all parameters to their initial default values.

CPSCAE
Updates a particular element of a user's cell array.

CPSETC
Sets the value of a parameter of type CHARACTER.

CPSETI
Sets the value of a parameter of type INTEGER.

CPSETR
Sets the value of a parameter of type REAL.

CPSPS1
Interpolates from an array of data on a "sparse" rectangular grid which is regularly spaced in X and Y to an array of data on a "dense" rectangular grid and to initialize contouring from the array on the dense grid. (By a "sparse" grid is meant one whose dimensions are smaller than one would like, so that contour lines constructed directly on it are composed of long straight segments.) CPSPS1 may be viewed as a data smoothing routine.

CPSPS2
Interpolates from an array of data on a "sparse" rectangular grid which is irregularly spaced in X and Y to an array of data on a "dense" rectangular grid and to initialize contouring from the array on the dense grid. (By a "sparse" grid is meant one whose dimensions are smaller than one would like, so that contour lines constructed directly on it are composed of long straight segments.) CPSPS2 may be viewed as a data smoothing routine.

The Conran_family utility

Conran_family
Is a set of two-dimensional contour mapping routines for irregularly distributed data.

There are three Conran_family entries, CONRAN, CONRAQ, and CONRAS. CONRAN can be invoked in two ways. One generates smoothed contours, the other does not. CONRAQ is a quick version of the package which has a more limited set of options than the other entries. CONRAS is the super version of the package.

Conran_family_params
Includes a brief description of all Conran_family parameters.

CONOP1

Turns options off and on for the Conran_family utility.

CONOP2
Specifies various INTEGER parameters to be used by the Conran_family utility.

CONOP3
Specifies various REAL parameters to be used by the Conran_family utility.

CONOP4
Specifies various CHARACTER parameters to be used by the Conran_family utility.

CONRAN
Draws a two-dimensional contour map from irregularly distributed data. The contour lines may be smoothed.

CONRAQ
Draws a two-dimensional contour map from irregularly distributed data. This is the small quick version of the Conran_family. It also has the fewest available number of options.

CONRAS
Draws a two-dimensional contour map from irregularly distributed data. This is the super version of the Conran_family.

The Dashline utility

Dashline
A set of line-drawing routines that can be used to generate various dashed line patterns (including solid), can add labels to the lines, can smooth the lines, and can suppress crowding of lines.

Dashline_params
Includes a brief description of all Dashline parameters.

Dashsmth
A smoother for Dashline.

Dashsupr
Super smooth opton for Dashline.

CURVED
Draws a curve through points.

DASHDB
Chooses a dash pattern without labels.

DASHDC
Chooses a dash pattern with labels.

FRSTD
Puts pen down at a point in user coordinates.

LASTD
Completes drawing of a line with interspersed characters generated by FRSTD and VECTD calls.

LINED
Draws a line segment between two points in user coordinates.

RESET
Initializes the Model Picture bit map to zeros.

VECTD
Draws a line segment between the pen position and the current point in user coordinates.

The Ezmap utility

Ezmap
Allows one to plot maps of the earth according to any of ten different projections, with parallels, meridians, and continental, international, and/or U.S. state outlines.

Ezmapa was the name of a supplement to the Ezmap utility that allowed users to produce solid-filled maps of the earth. All Ezmapa routines are now organized under the Ezmap utility.

Ezmap_params
This document briefly describes all Ezmap parameters.

MAPACI
Normally used to pick colors for the areas created by the boundary lines added to an area map by a call to MAPBLA.

MAPBLA
Adds the set of boundary lines to the area map of projected geographical entities, determined by the current state of the parameters of Ezmap.

MAPDRW
Draws the complete map described by the current values of the parameters of Ezmap.

MAPEOD
Called by the Ezmap routines MAPLOT and MAPBLA. The default version does nothing. A user-written version may be supplied to examine each outline-dataset segment and, perhaps, to delete selected ones.

MAPFST
Draws lines on a map used in conjunction with MAPVEC.

MAPGCI
Returns a set of points on the shortest great circle route between two user-specified points on the

globe.

MAPGRD
Draws a latitude/longitude grid.

MAPGRM
Draws lines of latitude and longitude masked against an existing area map.

MAPGTC
Gets the current value of a specified Ezmap parameter of type CHARACTER.

MPGETC is an alternate name for the routine MAPGTC.

MAPGTI
Gets the current value of a specified Ezmap parameter of type INTEGER.

MPGETI is an alternate name for the routine MAPGTI.

MAPGTL
Gets the current value of a specified Ezmap parameter of type LOGICAL.

MPGETL is an alternate name for the routine MAPGTL.

MAPGTR
Gets the current value of a specified EZMAP parameter of type REAL.

MPGETR is an alternate name for the routine MAPGTR.

MAPINT
Initializes Ezmap.

MAPIQ
Terminates a string of calls to MAPIT.

MAPIQA
Terminates a string of calls to MAPITA.

MAPIQM
Terminates a string of calls to the routine MAPITM.

MAPIT
Draws lines on a map.

MAPITA
Adds to an area map the projections of lines defined by a series of latitude/longitude coordinates.

MAPITM
Draws lines on a map, masked by a specified area map.

MAPLBL
Labels the map.

MAPLOT
Draws geographical outlines.

MAPPOS
Positions the map on the plotter frame.

MAPROJ
Sets the map projection to be used.

MAPRS
Re-calls SET.

MAPRST
Restores the state of Ezmap saved by an earlier call to MAPSAV.

MAPSAV
Saves the current state of Ezmap for later restoration by MAPRST.

MAPSET
Specifies the rectangular portion of the u/v plane to be drawn.

MAPSTC
Sets the values of certain Ezmap parameters of type CHARACTER.

MPSETC is an alternate name for the routine MAPSTC.

MAPSTI
Sets the values of certain Ezmap parameters of type INTEGER.

MPSETI is an alternate name for the routine MAPSTI.

MAPSTL
Sets the values of certain Ezmap parameters of type LOGICAL.

MPSETL is an alternate name for the routine MAPSTL.

MAPSTR
Sets the values of certain Ezmap parameters of type REAL.

MPSETR is an alternate name for the routine MAPSTR.

MAPTRA
Projects points.

MAPTRI
Performs inverse transformations.

MAPTRN
Projects points.

SUPCON is an alternate name for the routine MAPTRN.

MAPUSR
Called by Ezmap routines that draw the various parts of the map. The default version does nothing. A user-written version may be supplied to change the appearance of the map.

MAPVEC
Draws lines on a map---used in conjunction with MAPFST.

SUPMAP
Draws a map with a single call.

The Gflash utility

Gflash
Captures and inserts specified portions of graphics instructions in subsequent frames.

GFLAS1
Initiates storage of plotting instructions into a disk file. Instructions subsequent to GFLAS1, but prior to a GFLAS2 call, will be stored on disk rather than inserted into the output metafile.

GFLAS2
Terminates putting plotting instructions to disk and resumes putting plotting instructions to the metafile output. A call to GFLAS2 can only be made after a previous call to GFLAS1.

GFLAS3
Inserts the instructions saved on disk with a previous GFLAS1 identifier IB into the output metafile. GFLAS3 can be called only after a previous GFLAS1 and GFLAS2 sequence or after a call to GFLAS4. GFLAS3 also uses Fortran logical unit IC for its reads.

GFLAS4
Allows the user to access a disk file of plotting instructions generated with a GFLAS1 and GFLAS2 sequence in a previous job for use in a GFLAS3 call.

The GKS-0A Fortran library

GKS
GKS (the Graphical Kernel System) is a set of low-level graphics functions that are standardized by the American National Standards Institute (ANSI) and the International Standards Institute (ISO). The collection of pages in 3NCARG reflects those GKS functions documented in the "User’s Guide for NCAR GKS-0A Graphics" programmer document. This collection does not

contain all of the functions in GKS-0A, but merely the most frequently used ones. NCAR GKS-0A does contain all of the functions required to be implemented by the GKS Standard.

GACTM (Accumulate transformation matrix)
Constructs a GKS segment transformation matrix by starting with an existing matrix and composing it with a shift vector, a rotation angle, and X and Y scale factors to create a new transformation matrix. The rotation and scaling are done with respect to a user-defined fixed point.

GACWK (Activate workstation)
Activates a GKS workstation.

GCA (Cell array)
Draws a cell array starting with a rectangle specified by two corner points; this initial rectangle is subdivided into subrectangles by specifying an arbitrary number of divisions in the X and Y directions; each subrectangle (or cell) is assigned a color and drawn with that color.

GCLKS (Close GKS)
Closes the GKS package.

GCLRWK (Clear workstation)
Clears a workstation.

GCLSG (Close segment)
Closes a GKS segment.

GCLWK (Close workstation)
Closes a workstation.

GCRSG (Create segment)
Creates a GKS segment.

GCSGWK (Copy segment to workstation)
Copies a GKS segment to a GKS workstation.

GDAWK (Deactivate workstation)
Deactivates a workstation.

GDSG (Delete segment)
Deletes a GKS segment.

GESC (Escape)
Provides a standard way of implementing nonstandard GKS features. NCAR GKS has only two user-accessible escape functions defined: one for changing the name of a metafile and another for pausing in an X window.

GEVTM (Evaluate transformation matrix)
Constructs a GKS segment transformation matrix starting from a given point, a shift vector, a rotation angle, and X and Y scale factors.

GFA (Fill area)
Output primitive for filling polygonal areas.

GOPKS (Open GKS)
Opens the GKS package.

GOPWK (Open workstation)
Opens a GKS workstation.

GPL (Polyline)
This output primitive draws line segments connecting a sequence of user-specified coordinate pairs.

GPM (Polymarker)
The polymarker output primitive draws selected symmetric symbols to mark user-specified coordinate positions.

GQASF (Inquire aspect source flags)
Inquires the values for the aspect source flags that determine whether primitive attributes are to be selected from individual settings or from "bundle tables."

GQCHH (Inquire character height)
Retrieves the current character height.

GQCHSP (Inquire character spacing)
Retrieves the current value for character spacing.

GQCHUP (Inquire character up vector)
Retrieves the current value of the character up vector.

GQCHXP (Inquire character expansion factor)
Retrieves the current value of the character expansion factor.

GQCLIP (Inquire clipping indicator)
Retrieves the current value of the clipping indicator as well as the current clipping rectangle.

GQCNTN (Inquire current normalization transformation number)
Retrieves the number of the current normalization transformation.

GQCR (Inquire color representation)
Retrieves the color value associated with a color index on a particular workstation.

GQFACI (Inquire fill area color index)
Retrieves the current fill area color index.

GQFAIS (Inquire fill area interior style)
Retrieves the current value for fill area interior style.

GQFASI (Inquire fill area style index)
Retrieves the current value of the fill area style index.

GQLN (Inquire line type)
Retrieves the current setting for line type.

GQLWSC (Inquire line width scale factor)
Retrieves the current setting of the line width scale factor.

GQMK (Inquire marker type)
Retrieves the current value for the marker type.

GQMKSC (Inquire marker size scale factor)
Retrieves the current value for the polymarker size scale factor.

GQMNTN (Inquire maximum normalization transformation number)
Retrieves the maximum number of supported normalization transformations.

GQNT (Inquire normalization transformation)
Retrieves the window and viewport associated with a given normalization transformation. Users of NCAR Graphics are encouraged to use the SPPS routines SET and GETSET to handle normalization transformations instead of the GKS entries GSWN, GSVP, GSELNT, and GQNT.

GQOPS (Inquire operating state value)
Returns the operating state of GKS.

GQOPSG (Inquire name of open segment)
Returns the name of the currently open segment.

GQPLCI (Inquire polyline color index)
Gets the polyline color index.

GQPMCI (Inquire polymarker color index)
Retrieves the polymarker color index.

GQSGUS (Inquire set of segment names in use)
Retrieves what segment names are currently in use.

GQTXAL (Inquire text alignment)
Retrieves the horizontal and vertical text alignments.

GQTXCI (Inquire text color index)
Retrieves the current text color index.

GQTXFP (Inquire text font and precision)
Retrieves the current setting of the text font and precision values.

GQTXP (Inquire text path)
Retrieves the current text path.

GSASF (Set aspect source flags)
Sets the aspect source flags that determine whether primitive attributes are to be selected from individual settings or from "bundle tables."

GSCHH (Set character height)
Sets the character height for text.

GSCHSP (Set character spacing)
Sets the character spacing for text.

GSCHUP (Set character up vector)
Specifies the angle at which subsequent text is to be drawn with GTX.

GSCHXP (Set character expansion factor)
Sets the width to height ratio of text.

GSCLIP (Set clipping indicator)
Controls whether data are displayed outside the boundaries of the world coordinate window of the current normalization transformation.

GSCR (Set color representation)
Associates a color value with a color index.

GSELNT (Select normalization transformation)
Selects a predefined or user-defined transformation that maps world coordinates to normalized device coordinates.

GSFACI (Set fill area color index)
Sets the fill area color index.

GSFAIS (Set fill area interior style)
Sets the fill style of polygons drawn with GFA.

GSFASI (Set fill are style index)
Sets the fill area style index.

GSLN (Set line type)
Sets the line type to solid or various dashed patterns.

GSLWSC (Set linewidth scale factor)
Sets the linewidth scale factor, or relative thickness of a polyline.

GSMK (Set marker type)
Sets the type of polymarker to be used in subsequent GPM calls.

GSMKSC (Set marker size scale factor)
Sets the polymarker size scale factor --- the relative size of the marker.

GSPLCI (Polyline color index)
Sets the polyline color index.

GSPMCI (Set polymarker color index)
Sets the polymarker color index.

GSSGT (Set segment transformation)
Associates a segment transformation with a named segment.

GSTXAL (Set text alignment)
Sets the text alignment.

GSTXCI (Set text color index)
Sets the text color index.

GSTXFP (Set text font and precision)
Sets the text font and precision.

GSTXP (Set text path)
Sets the text paths or directions in which text is to be drawn.

GSVP (Set viewport)
Establishes a rectangular subspace of normalized device coordinates space. Calls to GSVP are discouraged with NCAR Graphics. Instead, use the SET subroutine because SET also supports mirror imaging and log scaling of axes.

GSWN (Set window)
Establishes a window, or rectangular subspace, of world coordinates to be plotted. Calls to GSWN are discouraged with NCAR Graphics. Instead, use the SET subroutine because SET also supports mirror imaging and log scaling of axes.

GTX (Text)
The basic output primitive for drawing text. A number of attribute-setting routines are available for selecting the size, font, precision, orientation, color, character spacing, etc.

GUWK (Update workstation)
Ensures that the workstation is current and reflects all requests that have been made.

The GKS-0A C library

gaccum_tran_matrix (Accumulate transformation matrix)
Constructs a GKS segment transformation matrix by starting with an existing matrix and composing it with a shift vector, a rotation angle, and X and Y scale factors to create a new transformation matrix. The rotation and scaling are done with respect to a user-defined fixed point.

gactivate_ws (Activate workstation)
Activates a GKS workstation.

gcell_array (Cell array)
Draws a cell array starting with a rectangle specified by two corner points; this initial rectangle is subdivided into subrectangles by specifying an arbitrary number of divisions in the X and Y directions; each subrectangle (or cell) is assigned a color and drawn with that color.

gclear_ws (Clear workstation)
Clears a workstation.

gclose_gks (Close GSK)
Closes the GKS package.

gclose_seg (Close segment)
Closes a GKS segment.

gclose_ws (Close workstation)
Closes a workstation.

gcopy_seg_ws (Copy segment to workstation)
Copies a GKS segment to a GKS workstation.

gcreate_seg (Create segment)
Creates a GKS segment.

gdeactivate_ws (Deactivate workstation)
Deactivates a workstation.

gdel_seg (Delete segment)
Deletes a GKS segment.

gescape (Escape)
Provides a standard way of implementing nonstandard GKS features. NCAR GKS has only two user-accessible escape functions defined: one for changing the name of a metafile and another for pausing in an X window.

geval_tran_matrix (Evaluate transformation matrix)
Constructs a GKS segment transformation matrix starting from a given point, a shift vector, a rotation angle, and X and Y scale factors.

gfill_area (Fill area)
Output primitive for filling polygonal areas.

ginq_asfs (Inquire aspect source flags)
Inquires the values for the aspect source flags that determine whether primitive attributes are to be selected from individual settings or from "bundle tables."

ginq_char_expan (Inquire character expansion factor)
Retrieves the current value of the character expansion factor.

ginq_char_ht (Inquire character height)
Retrieves the current character height.

ginq_char_space (Inquire character spacing)
Retrieves the current value for character spacing.

ginq_char_up_vec (Inquire character up vector)
Retrieves the current value of the character up vector.

ginq_clip (Inquire clipping indicator)
Retrieves the current value of the clipping indicator as well as the current clipping rectangle.

ginq_colr_rep (Inquire color representation)
Retrieves the color value associated with a color index on a particular workstation.

ginq_cur_norm_tran_num (Inquire current normalization transformation number)
Retrieves the number of the current normalization transformation.

ginq_fill_colr_ind (Inquire fill area color index)
Retrieves the current fill area color index.

ginq_fill_int_style (Inquire fill area interior style)
Retrieves the current value for fill area interior style.

ginq_fill_style_ind (Inquire fill area style index)
Retrieves the current value of the fill area style index.

ginq_line_colr_ind (Inquire polyline color index)
Gets the polyline color index.

ginq_linetype (Inquire line type)
Retrieves the current setting for line type.

ginq_linewidth (Inquire line width scale factor)
Retrieves the current setting line width scale factor.

ginq_marker_colr_ind (Inquire polymarker color index)
Retrieves the polymarker color index.

ginq_marker_size (Inquire marker size scale factor)
Retrieves the current value for the polymarker size scale factor.

ginq_marker_type (Inquire marker type)

Retrieves the current value for the marker type.

ginq_max_norm_tran_num (Inquire maximum normalization transformation

number)
Retrieves the maximum number of supported normalization transformations.

ginq_name_open_seg (Inquire name of open segment)
Returns the name of the currently open segment.

ginq_norm_tran (Inquire normalization transformation)
Retrieves the window and viewport associated with a given normalization transformation. Users of NCAR Graphics are encouraged to use the SPPS routines c_set and c_getset to handle normalization transformations instead of the GKS entries gset_win, gset_vp, gsel_norm_tran, and ginq_norm_tran.

ginq_op_st (Inquire operating state value)
Returns the operating state of GKS.

ginq_set_seg_names (Inquire set of segment names in use)
Retrieves what segment names are currently in use.

ginq_text_align (Inquire text alignment)
Retrieves the horizontal and vertical text alignments.

ginq_text_colr_ind (Inquire text color index)
Retrieves the current text color index.

ginq_text_font_prec (Inquire text font and precision)
Retrieves the current settings of the text font and precision values.

ginq_text_path (Inquire text path)
Retrieves the current text path.

gopen_gks (Open GKS)
Opens the GKS package.

gopen_ws (Open workstation)
Opens a GKS workstation.

gpolyline (Polyline)
This output primitive draws line segments connecting a sequence of user-specified coordinate pairs.

gpolymarker (Polymarker)
This polymarker output primitive draws selected symmetric symbols to mark user-specified coordinate positions.

gsel_norm_tran (Select normalization transformation)
Selects a predefined or user-defined transformation that maps world coordinates to normalized device coordinates.

gset_asfs (Set aspect source flags)
Sets the aspect source flags that determine whether primitive attributes are to be selected from individual settings or from "bundle tables."

gset_char_expan (Set character expansion factor)
Sets the width-to-height ratio of text.

gset_char_ht (Set character height)
Sets the character height for text.

gset_char_space (Set character spacing)
Sets the character spacing for text.

gset_char_up_vec (Set character up vector)
Specifies the angle at which subsequent text is to be drawn with gtext.

gset_clip_ind (Set clipping indicator)
Controls whether data are displayed outside the boundaries of the world coordinate window of the current normalization transformation.

gset_colr_rep (Set color representation)
Associates a color value with a color index.

gset_fill_colr_ind (Set fill area color index)
Sets the fill area color index.

gset_fill_int_style (Set fill area interior style)
Sets the fill style of polygons drawn with gfill_area.

gset_fill_style_ind (Set fill are style index)
Sets the fill area style index.

gset_line_colr_ind (Polyline color index)
Sets the polyline color index.

gset_linetype (Set line type)
Sets the line type to solid or various dashed patterns.

gset_linewidth (Set line width scale factor)
Sets the line width scale factor, or relative thickness of a polyline.

gset_marker_colr_ind (Set polymarker color index)
Sets the polymarker color index.

gset_marker_size (Set marker size scale factor)
Sets the polymarker size scale factor---the relative size of the marker.

gset_marker_type (Set marker type)
Sets the type of polymarker to be used in subsequent GPM calls.

gset_seg_tran (Set segment transformation)
Associates a segment transformation with a named segment.

gset_text_align (Set text alignment)
Sets the text alignment.

gset_text_colr_ind (Set text color index)
Sets the text color index.

gset_text_font_prec (Set text font and precision)
Sets the text font and precision.

gset_text_path (Set text path)
Sets the text paths or directions in which text is to be drawn.

gset_vp (Set viewport)
Establishes a rectangular subspace of normalized device coordinates space. Calls to gset_vp are discouraged with NCAR Graphics. Instead, use the c_set routine because c_set also supports mirror imaging and log scaling of axes.

gset_win (Set window)
Establishes a window, or rectangular subspace, of world coordinates to be plotted. Calls to gset_win are discouraged with NCAR Graphics. Instead, use the c_set routine because c_set also supports mirror imaging and log scaling of axes.

gtext (Text)
The basic output primitive for drawing text. A number of attribute setting routines are available for selecting the size, font, precision, orientation, color, character spacing, etc.

gupd_ws (Update workstation)
Ensures that the workstation is current and reflects all requests that have been made.

The Gridall utility

Gridall
Allows one to draw backgrounds for X/Y plots. Included are routines for drawing grids, perimeters, and pairs of axes. Gridall parameters control such things as the color indices and line width scale factors used to draw various parts of the background, the formats to be used for labels, the sizes of the characters to be used for labels, the amount by which labels are to be offset from the axes, and the routine to be called to draw the labels (the SPPS routine WTSTR or the Plotchar routine PLCHHQ).

Gridall_params
This document briefly describes all Gridall parameters.

GACOLR
Sets the values of parameters determining the color of various parts of the background.

GAGETC
Retrieves the value of a specified parameter of type CHARACTER.

GAGETI
Retrieves the value of a specified parameter of type INTEGER.

GAGETR
Retrieves the value of a specified parameter of type REAL.

GASETC
Gives a new value to a specified parameter of type CHARACTER.

GASETI
Gives a new value to a specified parameter of type INTEGER.

GASETR
Gives a new value to a specified parameter of type REAL.

GRID
Draws an unlabeled grid.

GRIDAL
Draws any of the supported types of backgrounds. Each of the other background-drawing routines is implemented by a call to GRIDAL.

GRIDL
Draws a labeled grid with each major division labeled with its numeric value.

HALFAX
Draws orthogonal axes intersecting at a specified point and with a specified set of labels.

LABMOD
Presets parameters controlling the appearance of labels drawn by GRIDAL, GRIDL, ... et al. LABMOD itself does no plotting and, in order to have any effect, must be called prior to the background-drawing routines for which it is presetting parameters.

PERIM
Draws an unlabeled perimeter with inward-pointing tick marks. The directions and lengths of tick marks may be changed by calling TICKS and/or TICK4.

PERIML
Draws a labeled perimeter with inward-pointing tick marks. The directions and lengths of tick

marks may be changed by calling TICK4.

TICK4
Allows for program control of tick mark length and direction.

The Histogram utility

Histogram
This utility is used to generate histograms. Histograms or bar charts are used to show the distribution of values in a like-sample dataset. The individual values are sorted by size into a number of value ranges called class intervals. The number of samples out of the total which fall into a class interval are represented as a histogram bar height. The height of each histogram bar is proportional to the number of samples in that class interval.

Histogram_params
This document briefly describes all histogram parameters.

HISTGR
Plots a histogram with various options including specification of class values, spacing between histogram bars, shading of bars, windowing (i.e. scaling), specification of color, labels, titles, etc. Data values are partitioned into classes; histogram bars represent either number of occurrences within each class, or a Y-value associated with that class (user choice). Options are set by calls to subroutines HSTOPL, HSTOPR, HSTOPC, and HSTOPI before the call to HISTGR.

HSTOPC
Specifies various CHARACTER variables to be used by the Histogram utility.

HSTOPI
Specifies various INTEGER parameters to be used by the Histogram utility.

HSTOPL
Turns parameter options off and on for the Histogram utility.

HSTOPR
Specifies various REAL arrays to be used by the the Histogram utility.

The Isosurface utility

Isosurface
Allows the user to draw a perspective view of the surface defined by the equation $f(u,v,w)=fiso$, where the function f is approximated by a 3-dimensional array of data and *fiso* is a user-specified "isovalue." Contours created by taking slices in any of three directions through the surface are drawn with hidden portions of the contour lines removed.

Isosurface_params
This document briefly describes all Isosurface parameters.

EZISOS
Draws an isosurface.

ISGETI
Gets the current values of parameters of type INTEGER.

ISGETR
Gets the current values of parameters of type REAL.

ISOSRF
Draws an isosurface.

ISSETI
Resets the current values of parameters of type INTEGER.

ISSETR
Resets the current values of parameters of type REAL.

PWRZI
Plots characters in 3-space when using Isosurface. For a large class of possible positions, the hidden character problem is solved.

The Labelbar utility

Labelbar
Creates a labeled, filled, rectangular bar to serve as a key for a filled plot.

Labelbar_params
This document briefly describes all Labelbar parameters.

LBFILL
A routine that is called by LBLBAR (when the argument IFTP is nonzero) to fill portions of the bar. The default version of this routine does color fill by calling GSFACI and GFA.

LBGETI
Retrieves the current value of parameters of type INTEGER in Labelbar.

LBGETR
Retrieves a parameter of type REAL in Labelbar.

LBLBAR
Draws a complete label bar.

LBSETI
Sets the value of parameters of type INTEGER in Labelbar.

LBSETR

Gives new values to parameters of type REAL in Labelbar.

The Ngmisc utility

Ngmisc
Is a miscellaneous collection of useful functions that are task-specific and not complex enough to be packages in their own right.

NGDOTS
Draws filled circular dots at coordinate positions (X(I),Y(I),I=1,NUM) at size SIZE with color given by the color index ICOLOR.

NGPICT
Effects a break in the picture drawing sequence in a Fortran code using GKS. The actions taken depend on whether the designated workstation is a metafile or an output/input workstation. An option is provided for prompting the user when an output/input workstation is ready and waiting after a pause.

NGWSYM
Draws a symbol from the standard WMO/NOAA meteorological fonts by reference to the font name and symbol number within that font.

The Plotchar utility

Plotchar
Allows a user to draw characters of high, medium, or low quality.

Plotchar_params
This document briefly describes all Plotchar parameters.

PCDLSC
Defines the default list of "special colors" used by PCHIQU in drawing certain characters from the filled fonts.

PCGETC
Retrieves the value of a specified parameter of type CHARACTER.

PCGETI
Retrieves the value of a specified parameter of type INTEGER.

PCGETR
Retrieves the value of a specified parameter of type REAL.

PCHIQU
Draws high-quality characters. By default, it uses the same database as the old NCAR Graphics routine PWRITX, but it can also use characters from any of the fontcap-defined databases, it has an improved interface, and it has many more capabilities than PWRITX.

PLCHHQ is an alternate name for the routine PCHIQU.

PCLOQU
Draws characters of "low quality" by calling the GKS character-drawing routines. No function codes may be used. Using PCLOQU to draw a given string of characters will create a smaller metafile than if PCHIQU or PCMEQU were used; the results will depend on capabilities of the translator.

PLCHLQ is an alternate name for the routine PCLOQU.

PCMEQU
Draws characters of "medium quality." It does this by drawing lines, just as PCHIQU does, but it does not produce quite such fancy characters. No function codes may be used. Using PCMEQU to draw a given string of characters will create a larger metafile than if PCLOQU were used, which may be a disadvantage. However, it may also be more dependable, in that it does not depend on capabilities the translator may or may not have.

PLCHMQ is an alternate name for the routine PCMEQU.

PCMPXY
This routine is normally not called directly by the user (though it can be). It is called by PCHIQU and by PCMEQU when the user has set the mapping flag MA nonzero to request mapping of characters from one X/Y coordinate system to another.

PCPNWI
Is a character function.

PCSETC
Gives a new value to a specified parameter of type CHARACTER.

PCSETI
Gives a new value to a specified parameter of type INTEGER.

PCSETR
Gives a new value to a specified parameter of type REAL.

The Scrolled_title utility

Scrolled_title
Creates movie titles. The titles can be scrolled, faded in, and/or faded out. Foreground and background colors can be specified.

Scrolled_title_params
This document briefly describes all Scrolled_title parameters.

FTITLE
Reads, from standard input, the information necessary to define the desired title frames, and then calls STITLE to create those frames. Used for unscrolled movie titles.

SLGETI

Gets the current values of parameters of type INTEGER that affect the behavior of STITLE and FTITLE.

SLGETR

Gets the current values of parameters of type REAL that affect the behavior of STITLE and FTITLE.

SLSETI

Sets the values of parameters of type INTEGER that affect the behavior of STITLE and FTITLE.

SLSETR

Sets the values of parameters of type REAL that affect the behavior of STITLE and FTITLE.

STITLE

Creates scrolled movie titles. It receives all input through the argument list.

The Softfill utility

Softfill

Fills a polygonal subset of the plotter frame.

Softfill_params

This document briefly describes all Softfill parameters.

SFGETC

Retrieves the current value of a specified parameter.

SFGETI

Retrieves the current integer value of a specified parameter.

SFGETP

Retrieves the current dot pattern.

SFGETR

Retrieves the current real value of a specified parameter.

SFNORM

The subroutines SFWRLD and SFNORM are used to fill that portion of the plotter frame inside the area defined by a given polygonal boundary.

SFSETC

Sets the character value of a specified parameter.

SFSETI

Sets the current integer value of a parameter.

SFSETP

Defines a new dot pattern.

SFSETR

Sets the current real value of a specified parameter.

SFSGFA (SOFTFILL Simulate GFA)

Fills, in one of various ways, an area of the plotter frame defined by a given set of points; it is intended to provide a way to use the GKS fill-area routine, if it works (as is the case in the version of GKS distributed with NCAR Graphics), or a suitable pattern-fill substitute, otherwise. Doing all area fills with SFSGFA has the advantage that the way in which they are done can then be changed by modifying the value of a single parameter of Softfill, named TY.

SFWRLD

The subroutines SFWRLD and SFNORM are used to fill that portion of the plotter frame inside the area defined by a given polygonal boundary.

The SPPS library

SPPS

Is a collection of routines for defining and managing NCAR Graphics coordinate systems, for drawing lines, markers, and text in those coordinate systems, and for implementing efficiency considerations such as polyline segment buffering.

SPPS_converters

A set of functions which transform coordinates between the various NCAR Graphics coordinate systems.

SPPS_params

This document briefly describes all SPPS parameters.

CFUX

Converts from fractional coordinates to user coordinates.

CFUY

Converts from fractional coordinates to user coordinates.

CLSGKS

Deactivates workstation 1, closes workstation 1, and closes GKS.

CUFX

Converts from user coordinates to fractional coordinates.

CUFY

Converts from user coordinates to fractional coordinates.

CURVE

Draws a curve defined by the series of points (PX(I),PY(I), I = 1,NP), in the user coordinate system. The pen is left at the location of the last point in the curve.

FRAME

Advances to the next picture in the case of CGM output, and pauses in the window of most recent creation for X11 output. A mouse or key click in the window on pause will cause all active workstations to be cleared.

FRSTPT

Generates a pen-up move to the point (PX, PY) in the user coordinate system. It is used in conjunction with routine VECTOR to draw lines.

GETSET

Returns a set of values which, if used as arguments in a call to SET, will cause normalization transformation 1, axis scaling (argument LS), and axis mirror imaging to be defined in such a way as to duplicate the combined effects of the current normalization transformation and axis scaling and mirror imaging.

GETUSV

Gets the value of one of the SPPS parameters.

LINE

Draws a line from the point (X1, Y1) to the point (X2, Y2).

OPNGKS

Opens GKS, opens workstation 1 to receive metacode output and activates workstation 1.

PLOTIF

Moves the plotter pen to a designated position in fractional coordinates, or causes an SPPS polyline buffer flush.

POINT

Draws a point at the location (PX, PY) in the user coordinate system.

POINTS

Draws a series of markers at the locations (PX(I),PY(I),I=1,NP). The markers can also be connected by drawing lines between them.

SET

Sets the parameters that define the mapping back and forth between fractional and user coordinates.

SETI

Sets the plotter address unit range exponents.

SETUSV

Sets the value of one of the SPPS parameters.

SFLUSH

Flushes the SPPS polyline buffer accumulated through calls to routines PLOTIF.

VECTOR
Generates a pen-down move to the point (PX, PY) in the user coordinate system. It is used in conjunction with routine FRSTPT to draw lines.

The Streamlines utility

Streamlines
Allows a user to plot a streamline representation of field flow given two arrays containing the components of the field vectors on a uniform grid in a user-defined coordinate system.

Streamlines_params
This document briefly describes all Streamlines parameters.

STGETI
Gets the current value of a parameter of type INTEGER.

STGETR
Gets the current value of a parameter of type REAL.

STINIT
Performs initialization tasks required before STREAM may be called to create a streamline field flow plot, including copying array size information into internal common block variables and establishing the basic mapping from grid coordinates to data coordinates and from user coordinates to NDC space.

STREAM
Outputs the streamline representation of the flow field, according to the specifications established by the parameter-setting routines and the initialization routine, STINIT.

STRSET
Resets all parameters to their initial default values.

STSETI
Sets the value of a parameter of type INTEGER.

STSETR
Sets the value of a parameter of type REAL.

STUIXY
Inversely maps a single point on the streamline from user to data coordinate space. The argument list for STUIXY parallels that of the internal Streamlines routine, STIMXY, used for the predefined mappings employed when the MAP parameter has a value between 0 and 2.

STUMSL
This routine is the user-definable external subroutine used to draw masked streamlines. The default version of the routine draws any polyline all of whose area identifiers are greater than or

equal to zero.

STUMTA

Given the coordinates of a point on the streamline in data, user, and NDC space, and the interpolated, normalized components of the vector at the point relative to data coordinate space, the user-modifiable routine STUMTA finds the directional angle of the streamline relative to NDC space at the point.

STUMXY

A user-modifiable routine that maps a single point on the streamline from data to user coordinate space. The argument list for STUMXY parallels that of the internal Streamlines routine, STMPXY, used for the predefined mappings employed when the MAP parameter has a value between 0 and 2.

The Surface utility

Surface

Draws a perspective picture of a function of two variables with hidden lines removed. The function is approximated by a two-dimensional array of heights.

Surface_params

This document briefly describes the Surface internal parameters.

SRFACE

Draws a perspective picture of a function of two variables with hidden lines removed. The function is approximated by a two-dimensional array of heights.

EZSRFC

Draws a perspective picture of a function of two variables with hidden lines removed. The function is approximated by a two-dimensional array of heights. Use EZSRFC only if the entire array is to be drawn, the data points are equally spaced in the X/Y plane, there are no stereo pairs, and scaling is chosen internally.

PWRZS

A character-plotting routine for plotting characters in 3-space when using SRFACE.

SETR

Sets constants prior to SRFACE calls so that picture size will change according to the viewer distance from the object. It can be used in creating a movie of an object evolving in time.

The Threed utility

Threed

A 3-D line-drawing package.

CURVE3

Draws a curve through points.

FENCE3

Used to draw a line in 3-space as well as a "fence" between the line and the plane normal to one of the coordinate axes.

FRST3

Positions the pen on a point in 3-space.

LINE3

Draws a line between two points in 3-space.

PERIM3

Draws a perimeter with tick marks.

POINT3

Draws a point in 3-space.

PWRZT

Plots characters in 3-space when using THREED.

SET3

Set up 3-D equivalents of viewport and window, as well as perspective angle.

TICK43

Allows conrol of tick mark length.

VECT3

Draws a line between the current pen position and the point (UA, VA, WA). The current pen position becomes (UA, VA, WA). Note that a curve can be drawn by using a FRST3 call followed by a sequence of VECT3 calls.

The Vectors utility

Vectors

Allows a user to plot vector fields given two arrays containing the components of the vectors on a uniform grid in a user-defined coordinate system. In addition, the user may pass in another array of scalar data defined over the same grid space independent of, but presumably related in some fashion, to the vector data. The utility will then color each vector based on the scalar value at the same grid point. Alternatively, the user can choose to color the vectors based on the vector magnitude at the grid point. Vectors replaces the Velvct utility.

Vectors_params

This document briefly describes all Vectors parameters.

VVECTR

Manages the coordinate system mapping, color setting, auxiliary text output, and drawing of the vector field plot, according to the specifications established by the parameter setting routines and the initialization routine, VVINIT.

VVGETC
Gets the current value of a Vectors parameter of type CHARACTER.

VVGETI
Gets the current value of a Vectors parameter of type INTEGER.

VVGETR
Gets the current value of a Vectors parameter of type REAL.

VVINIT
Performs initialization tasks required before VVECTR may be called to plot a vector field, including copying array size information into internal common block variables, establishing the basic mapping from grid coordinates to data coordinates and from user coordinates to NDC space, determining the maximum and minimum vector magnitudes and scalar array values, and, if required, setting up the color threshold value array.

VVRSET
Resets all parameters to their initial default values.

VVSETC
Sets the value of a Vectors parameter of type CHARACTER.

VVSETI
Sets the value of a Vectors parameter of type INTEGER.

VVSETR
Sets the value of a Vectors parameter of type REAL.

VVUDMV
This routine is the user-definable external subroutine used to draw masked vectors. The default version of the routine draws any polyline all of whose area identifiers are greater than or equal to zero.

VVUMXY
The user may modify this routine to define a custom mapping of vectors from a data coordinate system aligned with the natural boundaries of the vector field to the uniform normalized device coordinate (NDC) system suitable for generating a plot on an output device. It has same parameters as the internal Vectors routine, VVMPXY, used for the predefined mappings employed when the MAP parameter has a value between 0 and 2.

Man page entries for NCAR Graphics commands and auxiliary support programs

cgm2ncgm
Filters NCAR CGM from vanilla CGM.

cgm
Discusses various CGM tools, including CGM_open, CGM_close, CGM_lseek, CGM_read, CGM_write, CGM_directory,CGM_freeDirectory, CGM_printDirectory,

CGM_getInstr,CGM_flushGetInstr, CGM_putInstr, CGM_flushOutputInstr,CGM_initMetaEdit, CGM_termMetaEdit, CGM_copyFrames, CGM_deleteFrames,CGM_mergeFrames, CGM_moveFrames, CGM_readFrames, CGM_valid,CGM_writeFile, CGM_writeFrames, CGM_appendFrames --- Computer Graphics Metafile operations

ctrans
A Computer Graphics Metafile CGM translator.

fcaps
Reports available fontcaps.

findg
Locates calls to all obsolete entries in pre-GKS NCAR Graphics.

fontc
Fontcap preprocessor for NCAR Graphics.

fontcap
NCAR Graphics font definition file.

gcaps
Reports available graphcaps.

graphc
Graphcap preprocessor for NCAR Graphics.

graphcap
NCAR Graphics graphic device definition file.

ictrans
Interactive NCAR CGM translator **ctrans**.

idt
X Window interactive image display tool.

med
NCAR CGM metafile frame editor.

ncarg_cbind
Description of how to use the NCAR Graphics C-binding.

ncarg_gks
NCAR Graphics GKS environment variables.

ncarg_gks_cbind
Description of how to use the NCAR Graphics GKS C-binding.

ncargcc

Command for compiling C code that uses NCAR Graphics.

ncargex
NCAR Graphics C and Fortran examples and tests.

ncargf77
Command for compiling Fortran code that uses NCAR Graphics.

ncargintro
NCAR Graphics overview.

ncargmv
Moves an installed version of NCAR Graphics.

ncargrun
Runs a user program and redirects the metafile output.

ncargsrc
Retrieves NCAR Graphics source.

ncargtar
Creates **tar** file for NCAR Graphics.

ncargversion
Displays NCAR Graphics version number.

ncarv_spool
Spooled device configuration table for interactive metafile translator **ictrans**.

ncgm2cgm
Filters NCAR CGM from vanilla CGM.

pre2ncgm
Converts a pre-CGM NCAR metafile to an NCAR CGM.

psblack
PostScript filter to convert the background to black.

pswhite
PostScript filter to convert the background to white.

rascat
Concatenates, converts raster files.

ras_formats
Information on raster formats supported by NCAR Graphics.

ras_palette

Formats descriptions for palette files.

rasgetpal
Extracts the color palette of a raster file and writes it to standard output.

rasls
Lists information on raster files.

rassplit
Splits a multiframe raster file into single-frame raster files.

rasview
Raster file previewer for the X Window System.

INDEX

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Numerics

A

D

E

F

G

M

N

O

P

Q

R

S

Zooming in