The graPHIGS Programming Interface: Getting Started

SC33-8198-00

IBM

The graPHIGS Programming Interface: Getting Started

SC33-8198-00

Note

Before using this information and the product it supports, read the information in "Notices," on page 27.

Second Edition (September 1992)

This edition applies to the AIXwindows Environment/6000 (1.3) AIXwindows/3D feature, Program Number 5601-257, and to all subsequent releases of this product until otherwise indicated in new editions.

Contents

About This Book

This book guides you through the steps of writing and running your first graPHIGS API program.

Who Should Use This Book

This book is intended for first-time users of the graPHIGS API. It is also a good source of review for application and system programmers.

Highlighting

The following highlighting conventions are used in this book:

Bold	Identifies commands, subroutines, keywords, files, structures, directories, and other items whose names are predefined by the system. Also identifies graphical objects such as buttons, labels, and icons that the user selects.
Italics	Identifies parameters whose actual names or values are to be supplied by the user.
`Monospace`	Identifies examples of specific data values, examples of text similar to what you might see displayed, examples of portions of program code similar to what you might write as a programmer, messages from the system, or information you should actually type.

ISO 9000

ISO 9000 registered quality systems were used in the development and manufacturing of this product.

Related Publications

The following books contain information on graPHIGS API products:

- *The graPHIGS Programming Interface: Customization and Problem Diagnosis*
- *The graPHIGS Programming Interface: ISO PHIGS Subroutine Reference*
- *The graPHIGS Programming Interface: ISO PHIGS Quick Reference*
- *The graPHIGS Programming Interface: Messages and Codes*
- *The graPHIGS Programming Interface: Quick Reference*
- *The graPHIGS Programming Interface: Subroutine Reference*
- *The graPHIGS Programming Interface: Technical Reference*
- *The graPHIGS Programming Interface: Understanding Concepts*

Chapter 1. Introduction

Welcome to the IBM Personal graPHIGS Programming Interface, IBM's implementation of the PHIGS standard graphics programming interface. This guide takes you through the steps of writing and running your first graPHIGS API programs for the IBM RS/6000. This guide assumes you are familiar with the C programming language and with an editor, such as vi. In addition, the graPHIGS API, V2R2.0 or later, must be installed on your system.

This guide is divided into the following sections:

- Creating Your First graPHIGS API Program
- Adding Simple Interaction to a Program
- Creating Your First 3D graPHIGS API Program
- Adding Shading to a Program
- Creating a Very Simple Modeller
- Continuing with the graPHIGS API

Notes:

1. The example programs listed in this guide are available on the installation media for the AIXwindows/3D feature of the AIXwindows Environment/6000. The programs are installed in individual subdirectories under the following directory:
   ```
   /usr/lpp/graPHIGS/samples/gettingstarted
   ```
2. To compile and link all of the samples, enter the following commands on the command line:
   ```
   cd /usr/lpp/graPHIGS/samples/gettingstarted
   make
   ```
   You can then change directory (**cd**) to an individual sample directory and run the sample by entering:
   ```
   ./name
   ```
   where *name* is the name of the sample (such as `square`).
3. An interactive tutorial is also available for the graPHIGS API under the following directory:
   ```
   /usr/lpp/graPHIGS/clients/gPtutor
   ```
4. A sample debugger is also available for the graPHIGS API under the following directory:
   ```
   /usr/lpp/graPHIGS/clients/gPdbg
   ```
   The sample debugger provides the ability to trace, debug and modify the graPHIGS API calls interactively. Additionally, it allows you to look at the Structure Store, View Characteristics and Workstation specific data.
   For details on compiling, linking, and running the debugger, look at the README file in this directory:
   ```
   /usr/lpp/graPHIGS/clients/gPdbg
   ```

Chapter 2. Creating Your First graPHIGS API Program

This section covers the following topics:
- Writing, Compiling, and Running the **square** Program
- Examining the **square** Program
  – Including the afmnc.h File
  – Initializing the graPHIGS API
  – Creating a Geometry Structure
  – Displaying the Structure
  – Exiting the Program

Writing, Compiling, and Running the square Program

Your first program is called **square**, since it puts a red square with a white border on the screen.

1. Enter the following program into a file called **square.c** (e.g., `vi square.c`):

   (Ref #1.)

```
/*
 * COMPONENT_NAME:  graPHIGS API Samples
 *
 * ORIGINS:  27
 *
 * (C) COPYRIGHT International Business Machines Corp. 1990
 * All Rights Reserved
 *
 * Licensed Materials - Property of IBM
 *
 * US Government Users Restricted Rights - Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 */
/*-------------------------------------------------------------*/
/* graPHIGS Start-Up :     our first program                   */
#include <afmnc.h>                      /* graPHIGS include file */

 main() {
  int wsid = 1, viewid = 0, strid = 1;
  int n4[1] = {4};
  int status,choice;
  static float pts[8] = {0.5,0.5, 0.5,-0.5, -0.5,-0.5, -0.5,0.5};

  GPOPPH ("        ",0);         /* open graPHIGS                */
  GPOPWS (wsid,"*","X       ");  /* open a workstation           */

  GPOPST(strid);         /* open structure                       */
  GPEF  (2);             /* set edge on                          */
  GPIS  (2);             /* set interior style solid             */
  GPICI (2);             /* set interior color RED               */
  GPECI (1);             /* set edge color WHITE                 */
  GPPG2 (1,n4,2,pts);    /* add 2D polygon                       */
  GPCLST();              /* close structure                      */

  GPARV (wsid,viewid,strid,1.0);    /* link root to view         */
  GPUPWS(wsid,2);                   /* update workstation        */
  GPRQCH(wsid,2,&status,&choice);   /* wait for mouse hit        */
  GPCLPH();                         /* close graPHIGS            */
}
-------------------------------------------------------------------
```

2. Compile the program using the following command:

```
cc -o square square.c -lgP
```

3. Run the program by entering:

   ```
   ./square
   ```

   A window pops onto the screen. Inside the window is a red square with a white border. To exit the program, place the mouse pointer, or *locator input device*, inside the graPHIGS API window and press any mouse button.

Examining the square Program

This section describes the purpose of various parts of the program:

Including the afmnc.h File

Every graPHIGS API program written in the C programming language must include the file **afmnc.h**. This file contains the entry point definitions for all graPHIGS API functions.

```
#include <afmnc.h>             /* graPHIGS include file */
```

Initializing the graPHIGS API

The following two lines are what actually initialize the graPHIGS API. The Open graPHIGS (**GPOPPH**) subroutine puts the graPHIGS API into the open state, and Open Workstation subroutine opens one graphics workstation with the specified workstation identifier, *wsid*.

```
-----------------------------------------------------------------
GPOPPH ("        ",0);         /* open graPHIGS              */
GPOPWS (wsid,"*","X       ");  /* open a workstation         */
-----------------------------------------------------------------
```

Creating a Geometry Structure

The next section of code creates a geometry structure with the specified structure identifier, *strid*. The actual geometry and attributes are placed inside this structure. In this case, polygon edge attributes and polygon interior attributes are inserted into the structure, followed by a 2D polygon.

```
-----------------------------------------------------------------
GPOPST(strid);         /* open structure                      */
GPEF  (2);             /* set edge on                         */
GPIS  (2);             /* set interior style solid            */
GPICI (2);             /* set interior color RED              */
GPECI (1);             /* set edge color WHITE                */
GPPG2 (1,n4,2,pts);    /* add 2D polygon                      */
GPCLST();              /* close structure                     */
-----------------------------------------------------------------
```

Displaying the Structure

The next two lines of code display the geometry on the screen. The geometry structure you just created must be associated with a workstation view before it can be displayed. The first line of code associates the structure (called a *root structure*) with the default graPHIGS API view 0, specified by the view identifier *viewid*. The second line of code tells the workstation to update itself using the Update Workstation (**GPUPWS**) subroutine.

```
-----------------------------------------------------------------
GPARV (wsid,viewid,strid,1.0);   /* link root to view        */
GPUPWS(wsid,2);                  /* update workstation       */
-----------------------------------------------------------------
```

Exiting the Program

The last two lines of code allow you to exit the program gracefully. The Request Choice (**GPRQCH**) subroutine tells the graPHIGS API to wait until a mouse button is pressed. Then, the Close graPHIGS (**GPCLPH**) subroutine closes the graPHIGS API and the program exits.

```
-----------------------------------------------------------------
GPRQCH(wsid,2,&status,&choice);  /* wait for mouse hit         */
GPCLPH();                        /* close graPHIGS             */
-----------------------------------------------------------------
```

Chapter 3. Adding Simple Interaction to a Program

This section covers the following topics:

- Writing, Compiling, and Running the **flex** Program
- Examining the **flex** Program
  - Adding Label Elements to the Geometry Structure
  - Using an Input Loop

Writing, Compiling, and Running the flex Program

The second program is called **flex>**, for *flexible square*. It uses the **square** program as a base, and adds some simple user interaction. The size of the red square can now be interactively controlled by using the mouse pointer.

1. Enter the following program into a file called **flex.c**

```
-----------------------------------------------------------------
/*
 * COMPONENT_NAME:  graPHIGS API Samples
 *
 * ORIGINS:  27
 *
 * (C) COPYRIGHT International Business Machines Corp. 1990
 * All Rights Reserved
 *
 * Licensed Materials - Property of IBM
 *
 * US Government Users Restricted Rights - Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 */
/*---------------------------------------------------------------*/
/* graPHIGS Start-Up : adding simple input                       */
#include [afmnc.h]                     /* graPHIGS include file */

main() {
  int wsid = 1, viewid = 0, strid = 1;
  int n4[1] = {4};
  int lcview,major,class,minor;
  float matrix[9],lcpos[3];
  static float oldpos[2] = {1.0,1.0};
  static float pts[8] = {1.0,1.0, 1.0,-1.0, -1.0,-1.0,
-1.0,1.0};

  GPOPPH ("        ",0);         /* open graPHIGS              */
  GPOPWS (wsid,"*","X       ");  /* open a workstation         */

  GPOPST(strid);         /* open structure                     */
  GPINLB(1);             /* insert label 1                     */
  GPINLB(2);             /* insert label 2                     */
  GPEF  (2);             /* set edge on                        */
  GPIS  (2);             /* set interior style solid           */
  GPICI (2);             /* set interior color RED             */
  GPECI (1);             /* set edge color WHITE               */
  GPPG2 (1,n4,2,pts);    /* add 2D polygon                     */
  GPCLST();              /* close structure                    */

  GPARV (wsid,viewid,strid,1.0);   /* link root to view         */
  GPUPWS(wsid,2);                  /* update workstation        */
  GPCHMO(wsid,2,3,2);              /* choice: event mode        */
  GPLCMO(wsid,1,2,2);              /* locator: sample mode      */

  do  {                            /*loop until mouse hit       */
     GPSMLC (wsid, 1, &lcview,lcpos); /* get mouse position     */
```

```
        if  (lcpos[0] != oldpos [0]  ||
             lcpos[1] != oldpos [1]  )
        {
          oldpos[0] = lcpos [0] ;
          oldpos[1] = lcpos [1] ;
      GPSC2  (lcpos, matrix);          /* calculate scale matrix   */
      GPOPST (strid);                  /* open structure for edit  */
      GPDELB (1,2);                    /* delete between labels    */
      GPMLX2  (matrix, 3);             /* insert new model matrix  */
      GPCLST ();                       /* close structure          */
      GPUPWS (wsid,2):                 /* update workstation       */
     }
     GPAWEV (0.0,&major,&class,&minor);    /* check for any events */
    } while (class != 4);
    GPCLPH();
   }
```

2. Compile the program using the following command:

   ```
   cc -o flex flex.c -lgP
   ```

3. Run the program by entering:

   ```
   ./flex
   ```

   A window pops onto the screen. Inside the window is a red rectangle with a white border. The size of the rectangle can be controlled by moving the mouse pointer while it is inside the graPHIGS API window. To exit the program, position the mouse pointer in the graPHIGS API window and press any button.

Examining the flex Program

This section looks at the program in more detail:

Adding Label Elements to the Geometry Structure

Creating the geometry structure is essentially the same as for the **square** program. The only additions are two label elements, added with the Insert Label (**GPINLB**) subroutine. These labels will allow you to later insert and delete elements between the two labels.

```
---------------------------------------------------------------
GPOPST(strid);         /* open structure                     */
GPINLB(1);             /* insert label 1                     */
GPINLB(2);             /* insert label 2                     */
GPEF  (2);             /* set edge on                        */
GPIS  (2);             /* set interior style solid           */
GPICI (2);             /* set interior color RED             */
GPECI (1);             /* set edge color WHITE               */
GPPG2 (1,n4,2,pts);    /* add 2D polygon                     */
GPCLST();              /* close structure                    */
---------------------------------------------------------------
```

Using an Input Loop

This program also adds a loop to handle input. Inside the loop, the Sample Locator (**GPSMLC**) subroutine samples the position of the mouse pointer (or *locator device*). The locator position is returned as x,y,z coordinates in the variable *lcpos*. If the new and old locator positions are different, the shape of the square is modified. This is done by editing the geometry structure containing the square.

The Scale 2D (**GPSC2**) subroutine is used to create a new 2D scale matrix which is then inserted into the geometry structure using the Set Modeling Transformation 2D (**GPMLX2**) subroutine.

Also inside the input loop, the graPHIGS API event queue is checked using the Await Event (**GPAWEV**) subroutine. The loop is exited if the value for the returned class is 4 (e.g., the mouse button is pressed).

```
----------------------------------------------------------------
do {                             /* loop until mouse hit      */
  GPSMLC(wsid,1,&lcview,lcpos);  /* get the mouse position    */
  if (lcpos[0] != oldpos[0] ||
      lcpos[1] != oldpos[1])
  {
    oldpos[0] = lcpos[0];
    oldpos[1] = lcpos[1];
    GPSC2 (lcpos,matrix);        /* calculate scale matrix    */
    GPOPST(strid);               /* open structure for edit   */
    GPDELB(1,2);                 /* delete between labels     */
    GPMLX2 (matrix,3);           /* insert new model matrix   */
    GPCLST();                    /* close structure           */
    GPUPWS(wsid,2);              /* update workstation        */
  }
  GPAWEV(0.0,&major,&class,&minor);  /* check for any events  */
} while (class != 4);
----------------------------------------------------------------
```

Chapter 4. Creating Your First 3D graPHIGS API Program

This section covers the following topics:

- Writing, Compiling, and Running the **using3d** Program
- Examining the **using3d** Program
  - Opening and Reading a File
  - Creating the Geometry Structure
  - Defining a View
  - Using the Input Loop and Setting a New View Orientation

Writing, Compiling, and Running the using3d Program

The next program, **using3d**, takes you into the realm of 3D. This program reads a wireframe geometry data file and creates a 3D wireframe model from the file's contents. A 3D perspective window is also defined for displaying the geometry.

1. Enter the following program into a file called **using3d.c**

```
--------------------------------------------------------------------
/*
 * COMPONENT_NAME:  graPHIGS API Samples
 *
 * ORIGINS:  27
 *
 * (C) COPYRIGHT International Business Machines Corp. 1990
 * All Rights Reserved
 *
 * Licensed Materials - Property of IBM
 *
 * US Government Users Restricted Rights - Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 */
/*---------------------------------------------------------------*/
/* graPHIGS Start-Up : Using 3D                                  */
#include <afmnc.h>                  /* graPHIGS include file   */
#include <stdio.h>                   /* C standard IO          */

main() {
  static float window[4] = {-0.8,0.8,-0.8,0.8};
  static float viewpt[6] = {0.0,1.0,0.0,1.0,0.0,1.0};
  static float prp   [3] = {0.0,0.0,2.4};
  static float oldpos[2] = {5.0,5.0};
  FILE *fp;
  int wsid=1, viewid=1, strid=1, np=0;
  int lcview, major, class, minor, md[3000];
  float mata[16], matb[16], matc[16], lcpos[3],
pts[3000][3];

  fp = fopen("USING3D.WF","r");
  while ( fscanf(fp,"%d %f %f %f", &md[np],
               &pts[np][0],
               &pts[np][1],
               &pts[np][2]) != -1) {np++;}

  GPOPPH ("        ",0);         /* open graPHIGS                */
  GPOPWS (wsid,"*","X       ");  /* open a workstation           */

  GPOPST(strid);         /* open structure                      */
  GPPLCI(3);             /* set polyline color GREEN            */
  GPDPL3(np,3,pts,md);   /* add 3D disjoint polyline            */
  GPCLST();              /* close structure                     */
  GPVCH (wsid,viewid,2,2,2,  1,0,  2,1,  2);/* activate view     */
-------------------------------------------------------------------
```

2. Create a file called **USING3D.WF** using the following information to define a square:

   ```
   2   -0.5   -0.5    0.0
   2   -0.5    0.5    0.0
   2    0.5    0.5    0.0
   2    0.5   -0.5    0.0
   1   -0.5   -0.5    0.0
   ```

   This information is found in:

   `/usr/lpp/graPHIGS/samples/gettingstarted/using3D/USING3D.WF`

3. Compile the program using the following command:

   ```
   cc -o using3d using3d.c -lgP
   ```

4. Run the program by entering:

   ```
   ./using3d
   ```

   A window pops up on the screen. Inside the window is the green wireframe geometry. The orientation of the viewer can be changed by moving the mouse pointer inside the graPHIGS API window. To exit the program, position the mouse pointer in the graPHIGS API window and press any button.

Examining the using3d Program

This section looks at the program in more detail, and describes the functions of various parts of the code.

Opening and Reading a File

The following lines of code open and read the wireframe geometry file. The data is stored in the **pts[ ]** and **md[ ]** arrays.

```
-----------------------------------------------------------------
fp = fopen("USING3D.WF","r");
while ( fscanf(fp,"%d %f %f %f",
               &md[np],
               &pts[np][0],
               &pts[np][1],
               &pts[np][2]) != -1) {np++;}
-----------------------------------------------------------------
```

Creating the Geometry Structure

The geometry creation code is similar to that of the **square** and **flex** programs. This program specifies a line color attribute, and then inserts a disjoint polyline primitive into the structure.

```
-----------------------------------------------------------------
GPOPST(strid);         /* open structure                        */
GPPLCI(3);             /* set polyline color GREEN              */
GPDPL3(np,3,pts,md);   /* add 3D disjoint polyline              */
GPCLST();              /* close structure                       */
-----------------------------------------------------------------
```

Defining a View

This is the first program which does not use the default graPHIGS API view 0. Instead, this program uses the Set View Characteristics (**GPVCH**) subroutine to activate the view specified by the view identifier, *viewid*. It also uses the Set View Mapping 3D (**GPVMP3**) subroutine to specify the set of parameters which define the view, such as:

- window size
- viewport
- projection type (parallel/perspective)
- view plane

- near clipping plane
- far clipping plane.

Finally, the Set View Input Priority (**GPVIP**) subroutine ensures that input comes from the default view 0.

```
-----------------------------------------------------------------
GPVCH (wsid,viewid,2,2,2,  1,0,  2,1,  2);/* activate view     */
GPVMP3(wsid,viewid,window,viewpt,         /* set view mapping  */
  2,prp,  0.0,  1.6,-1.6);                /* perspective       */
GPVIP (wsid,0,viewid,1);/* set view input priority 0>1         */
-----------------------------------------------------------------
```

Using the Input Loop and Setting a New View Orientation

The last section of code is essentially the same as that of the **flex** program. The only difference is the animation of the model. This program sets the new view orientation into the view matrix using the Set View Matrix 3D (**GPVMT3**) subroutine.

```
-----------------------------------------------------------------
do {                               /* loop until mouse hit       */
  GPSMLC(wsid,1,&lcview,lcpos);  /* get the mouse position     */
  if (lcview == 0 &&
      (lcpos[0] != oldpos[0] ||
       lcpos[1] != oldpos[1]))
  {
    oldpos[0] = lcpos[0];
    oldpos[1] = lcpos[1];
    GPROTX(lcpos[0]*3.14,mata);  /* calculate X rot matrix     */
    GPROTY(lcpos[1]*3.14,matb);  /* calculate Y rot matrix     */
    GPCMT3(mata,matb,matc);      /* multiply matrices          */
    GPVMT3(wsid,viewid,matc);    /* set view matrix, move view */
    GPUPWS(wsid,2);              /* update workstation         */
  }
  GPAWEV(0.0,&major,&class,&minor);  /* check for any events   */
} while (class != 4);
-----------------------------------------------------------------
```

Chapter 5. Adding Shading to a Program

This section covers the following topics:

- Writing, Compiling, and Running the **shade** Program
- Examining the **shade** Program
  - Checking for Shading Support
  - Creating the Geometry Structure
  - Defining Light Sources
  - Setting Viewing Parameters
  - Using the Input Loop and Setting a New Model Orientation

Writing, Compiling, and Running the shade Program

The **shade** program introduces you to the areas of shading, lighting, and hidden surface removal. A shaded version of the geometry used in the **using3d** program is read and displayed in a 3D window.

1. Enter the following program into a file called **shade.c**

```
------------------------------------------------------------------
/*
 * COMPONENT_NAME:  graPHIGS API Samples
 *
 * ORIGINS:  27
 *
 * (C) COPYRIGHT International Business Machines Corp. 1990
 * All Rights Reserved
 *
 * Licensed Materials - Property of IBM
 *
 * US Government Users Restricted Rights - Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 */
/*---------------------------------------------------------------*/
/* graPHIGS Start-Up : shading and lighting                      */
#include <afmnc.h>                     /* graPHIGS include file */
#include <stdio.h>                     /* C standard IO         */

main() {
  static float window[4] = {-0.8,0.8,-0.8,0.8};
  static float viewpt[6] = {0.0,1.0,0.0,1.0,0.0,1.0};
  static float prp   [3] = {0.0,0.0,2.4};
  static float oldpos[2] = {5.0,5.0};
  static float ltdir [3] = {0.1,0.1,-1.0};
  static int   inll  [2] = {1,2};
  FILE *fp;
  int wsid=1, viewid=1, strid=1, nvert=0, pdata=1, n2=2;
  int endflag, lcview, major, class, minor;
  float mata[16], matb[16], matc[16], lcpos[3],
pts[10][6];
  struct {int mode; float r,g,b;} ltcol  = {2,1.0,1.0,1.0};

  GPOPPH ("        ",0);         /* open graPHIGS               */
  GPOPWS (wsid,"*","X       ");  /* open a workstation          */

  if (shadingQ(wsid) == 0) {     /* is shading supported?       */
    printf("No shading support, exiting program.\n");
    GPCLPH ();                   /* close graPHIGS              */
    exit(1);
  }

  fp = fopen("USING3D.SH","r");
```

```
      GPOPST(strid);         /* open structure                     */
      GPINLB(1);             /* insert label 1                     */
      GPINLB(2);             /* insert label 2                     */
   ----------------------------------------------------------------
```

2. Create a file called **USING3D.SH** containing the following information, which defines a square:

   ```
      0   0.0    -0.5   -0.5    1.0    0.0    0.0
      0   0.0    -0.5    0.5    1.0    0.0    0.0
      0   0.0     0.5    0.5    1.0    0.0    0.0
      1   0.0     0.5   -0.5    1.0    0.0    0.0
      0   0.0    -0.5   -0.5    1.0    0.0    0.0
      0   0.0     0.5   -0.5    1.0    0.0    0.0
      0   0.0     0.5    0.5    1.0    0.0    0.0
      1   0.0    -0.5    0.5    1.0    0.0    0.0
      -1  0.0     0.0    0.0    0.0    0.0    0.0
   ```

   This information is found in:

   /usr/lpp/graPHIGS/samples/gettingstarted/shade/USING3D.SH
3. Compile the program using the following command:

   ```
   cc -o shade shade.c -lgP
   ```

   This program introduces a new concept, the graPHIGS API *PROFILE*. The PROFILE is an external file which is read by the graPHIGS API every time a program is executed. The PROFILE is used to alter certain graPHIGS API default values and to override parameters sent to graPHIGS API control functions. The instructions below set up a direct-color color table instead of the default color table.
4. Enter the following into a file called **PROFILE**:

   ```
   * graPHIGS Start-Up - PROFILE for shade.c sample
   *
   AFMMNICK PROCOPT=((DIRCOLOR))
   ```
5. Run the program by entering:

   ```
   ./shade
   ```

   A window pops onto the screen. Inside the window is the shaded 3D geometry. The orientation of the viewer can be changed by moving the mouse pointer while it is inside the graPHIGS API window. To exit the program, position the mouse pointer in the graPHIGS API window and press any button.

Examining the shade Program

This section describes the purpose of various sets of instructions in the **shade** program.

Checking for Shading Support

After opening the graPHIGS API and a graPHIGS API workstation, the program checks to see if the workstation supports shading. This check has been separated into a utility function called **shadingQ** (*shading query*). This utility function uses the graPHIGS API Inquire Light Source Facilities (**GPQLSF**) subroutine to check for shading support. If shading is supported, a value of 1 is returned. Otherwise a value of 0 is returned.

```
----------------------------------------------------------------
if (shadingQ(wsid) == 0) {     /* is shading supported?        */
  printf("No shading support, exiting program.\n");
  GPCLPH ();                   /* close graPHIGS               */
  exit(1);
}

/* utility function to check for shading support */
int shadingQ(wsid)
{
  int status,conlen,errind,maxe,totnum,ltype,maxa,npred;
  char wstype[10],connid[255]

  GPQRCT(wsid,255,&status,&conlen,connid,wstype);
```

```
GPQLSF(wstype,1,0,&errind,&maxe,&totnum,;&ltype,&maxa,&n
pred);
  if (maxe == 0) return 0;
  return 1;
}
-----------------------------------------------------------------
```

Creating the Geometry Structure

This program uses several new subroutines in the geometry structure creation code:

GPICD	**Set Interior Color Direct** - Used in the program to set the color of the model. Allows you to pass in an array containing the red, green and blue color values defining the interior color of the geometry.
GPICD	**Set Interior Color Direct** - Used in the program to set the color of the model. Allows you to pass in an array containing the red, green and blue color values defining the interior color of the geometry
GPSCD	**Set Specular Color Direct** - Used to specify the specular color of the model.
GPLMO	**Set Lighting Mode** - Used to activate lighting.
GPLSS	**Set Light Source State** - Used to activate lighting.
GPHID	**Set HLHSR (Hidden Line and Hidden Surface Removal)** - Used to activate hidden surface removal.
GPPGD3	**Polygon with Data 3D** - Used to create the actual geometry. Allows you to define 3D polygons with vertex normal vectors to simulate the curvature of the polygon.

```
-----------------------------------------------------------------
fp = fopen("USING3D.SH","r");
GPOPST(strid);         /* open structure                      */
GPINLB(1);             /* insert label 1:                     */
GPINLB(2);             /* insert label 2                      */
GPICD (&ltcol.r);      /* set interior color WHITE            */
GPSCD (&ltcol.r);      /* set specular color WHITE            */
GPSPR (0.3,0.4,0.4,9.0,1.0);    /* set surface properties     */
GPLMO (3);             /* set lighting to gouraud             */
GPIS  (2);             /* set interior style solid            */
GPHID (1);             /* activate hidden surface removal     */
GPPGC (2);             /* set polygon culling on              */
GPLSS (2,inll,0,inll); /* activate 2 light sources            */
do {                   /* loop to read X29 shade file         */
  fscanf(fp,"d %f %f  %f %f %f %f",&endflag,
        &pts[nvert][0],
        &pts[nvert][1],
        &pts[nvert][2],
        &pts[nvert][3],
        &pts[nvert][4],
        &pts[nvert][5]);
  nvert++;
  if (endflag == 1) {
    GPPGD3(0,&pdata,0,1,&nvert,1,6,pts); /* create polygon     */
    nvert=0;
  }
} while (endflag != -1);
GPCLST();              /* close structure                     */
-----------------------------------------------------------------
```

Defining Light Sources

The following two lines of code define the light sources. The two Set Light Source Representation (**GPLSR**) subroutines below define one ambient light source and one directional light source.

```
-----------------------------------------------------------------
GPLSR(wsid,1,1,&ltcol,0);     /* define light source 1        */
GPLSR(wsid,2,2,&ltcol,ltdir); /* define light source 2        */
-----------------------------------------------------------------
```

Setting Viewing Parameters

The only remaining new subroutine is Set Extended View Representation (**GPXVR**). This subroutine allows you to set viewing parameters individually and is sometimes used in place of the Set View Mapping 3D (**GPVMP3**) subroutine. This program uses GPXVR to activate hidden surface removal for the view specified by the view identifier *viewid*.

```
-----------------------------------------------------------------
GPXVR (wsid,viewid,10,&n2);  /* activate hidden surface       */
-----------------------------------------------------------------
```

Using the Input Loop and Setting a New Model Orientation

The last section of code is essentially the same as that of the **using3d** program. The only difference is that the **using3d** program sets a new *view* orientation while the **shade** program sets a new *model* orientation. This program sets the new model orientation (transformation matrix) into the open structure using the Set Modeling Transformation 3D (**GPMLX3**) subroutine.

```
------------------------------------------------------------------
  do {                             /* loop until mouse hit      */
    GPSMLC(wsid,1,&lcview,lcpos);  /* get the mouse position    */
    if (lcview == 0 &&
        (lcpos[0] != oldpos[0] ||
         lcpos[1] != oldpos[1]))
    {
      oldpos[0] = lcpos[0];
      oldpos[1] = lcpos[1];
      GPROTX(lcpos[0]*3.14,mata);  /* calculate X rot matrix    */
      GPROTY(lcpos[1]*3.14,matb);  /* calculate Y rot matrix    */
      GPCMT3(mata,matb,matc);      /* multiply matrices         */
      GPOPST(strid);               /* open structure for edit   */
      GPDELB(1,2);                 /* delete between labels     */
      GPMLX3(matc,3);              /* insert new model matrix   */
      GPCLST();                    /* close structure           */
      GPUPWS(wsid,2);              /* update workstation        */
    }
    GPAWEV(0.0,&major,&class,&minor);  /* check for any events  */
  } while (class != 4);
------------------------------------------------------------------
```

Chapter 6. Creating a Very Simple Modeller

This section covers the following topics:
- Writing, Compiling, and Running the **model** Program″
- Examining the **model** Program
  – Initializing the graPHIGS API
  – Using Structure Hierarchy
  – Creating Menus and Adding Pick Identifiers
  – Picking Menu Items
  – Using the Whole Window
  – Using the Input Loop

Writing, Compiling, and Running the model Program

The **model** program allows you to build a 2D model using squares and triangles. Using the menu, you may add squares and triangles, delete all squares or all triangles, or exit the program.

Like the **using3d** program, this program requires a **graPHIGS API PROFILE**.

1. Enter the following program into a file called **model.c**:

```
-----------------------------------------------------------------
/*
 * COMPONENT_NAME:  graPHIGS API Samples
 *
 * ORIGINS:  27
 *
 * (C) COPYRIGHT International Business Machines Corp. 1990
 * All Rights Reserved
 *
 * Licensed Materials - Property of IBM
 *
 * US Government Users Restricted Rights - Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 */
/*-------------------------------------------------------------*/
/* graPHIGS Start-Up :  a simple modeller                      */
#include <afmnc.h>                     /* graPHIGS include file */

main() {
  static float window1[4] = {0.0,70.0,0.0,100.0};
  static float viewpt1[4] = {0.3,1.0,0.0,1.0};
  static float window2[4] = {0.0,9.0,0.0,30.0};
  static float viewpt2[4] = {0.0,0.3,0.0,1.0};
  static float geopts[8] = {0.0,0.0, 5.0,0.0, 5.0,5.0,
0.0,5.0};
  static float boxpts[8] = {0.0,0.0, 9.0,0.0, 9.0,1.0,
0.0,1.0};
  static float boxoff[2] = {0.0,1.0};
  static float txtpos[2] = {4.5,0.5};
  static char  *menus[9] = {"Exit",
                            "",
                            "Remove squares",
                            "Add squares",
                            "",
                            "Remove triangles",
                            "Add triangles"};
  float lcpos[2],mat[9],dydx;
  int wsid=1, ssid=1, ncid=1, viewid1=1,viewid2=2;
  int mode=1, topstrid=1, tristrid=2, sqstrid=3, menustrid=4;
  int depth,lcview,i,n,major,class,minor;
```

```
struct {int strid, pkid, elnum;} pickpath;

GPOPPH ("        ",0);         /* open graPHIGS                */
GPCNC  (ncid,1,0,0);           /* connect to graPHIGS nucleus */
GPCRSS (ssid,ncid,1,0);        /* create structure store      */
GPSSS  (ssid);         /* select structure store      */
GPCRWS (wsid,ncid,1,"*","X         ",0); /*create workstation */
GPASSW (wsid,ssid);            /* link structure store to ws  */

n=1;
GPOPST(topstrid);       /* open geometry structure            */
GPEF  (2);              /* set edge on                        */
GPIS  (2);              /* set interior style solid           */
GPECI (7);              /* set edge color WHITE               */
GPICI (2);              /* set interior color RED             */
GPEXST(tristrid);       /* execute triangle structure         */
GPICI (4);              /* set interior color RED             */
GPEXST(sqstrid);        /* execute square structure           */
GPCLST();               /* close geometry structure           */

GPTRL2(boxoff,mat);     /* calculate translation matrix       */
GPOPST(menustrid);      /* open menu structure                */
GPEF  (2); GPIS  (2);   GPECI (7); GPICI (4);  /* interior    */
GPTXCI(7); GPTXAL(3,4); GPTXPR(3); GPCHH (0.6);/* edge attr   */
n=1;GPADCN(1,&n);       /* add class to set, for picking      */
n=4;
for (i=0;i<7;i++) {
  GPPKID(i);            /* insert pick id                     */
  GPPG2 (1,&n,2,boxpts);/* insert menu item, polygon          */
  GPTX2(txtpos,strlen(menus[i]),menus[i]);   /*   text        */
  GPMLX2(mat,2);        /* translate to next menu item        */
}
GPCLST();    /* close menu structure               */

wholescreen(wsid,&dydx);  /* use whole screen, set WSX        */
window1[3] *= dydx;     /* change window and viewport defs    */
window2[3] *= dydx;     /*   to match screen aspect ratio     */
viewpt1[3] *= dydx;                                 */
viewpt2[3] *= dydx;
GPARV (wsid,viewid1,topstrid,1.0;    /* link root to view     */
GPARV (wsid,viewid2,menustrid,1.0;   /* link root to view     */
GPVMP2(wsid,viewid1,window1,viewpt1);/* set view mapping      */
GPVMP2(wsid,viewid2,window2,viewpt2);/* set view mapping      */
GPVCH (wsid,viewid1,2,2,2,  1,0,  2,1,  2);/* activate view   */
GPVCH (wsid,viewid2,2,2,2,  1,0,  1,1,  2);/* activate view   */
GPVIP (wsid,0,viewid1,2);     /* set view input priority 0>1  */

n=1;
GPPKF (wsid,1,1,&n,0,&n);     /* set pick filter, same class  */
GPPKMO(wsid,1,3,2);           /* pick: event mode             */
GPLCMO(wsid,1,3,2);           /* locator: sample mode         */
GPUPWS(wsid,2);               /* update workstation           */

while (mode != 0) {           /* loop to look for events      */
  GPAWEV(100.0&major,&class,&minor); /*  wait for event       */
  if (class == 5 & minor == 1) {    /*  pick event?          */
    GPGTPK(1,&depth,&pickpath);      /*  get pick information */
    if (pickpath.pkid == 6) mode = 1;/*  square mode          */
    if (pickpath.pkid == 3) mode = 2;/*  triangle mode        */
    if (pickpath.pkid == 0) mode = 0;/*  exit                 */
    if (pickpath.pkid == 5) GPEST(tristrid); /* erase tris    */
    if (pickpath.pkid == 2) GPEST(sqstrid);  /* erase squares */
  }
  if (class == 1 & minor == 1) {
    GPGTLC(&lcview,lcpos);            /* get mouse position   */
    if (lcview == viewid1) {          /* open proper structure*/
      if (mode == 1) { GPOPST(tristrid); n=3;}
```

```
        if (mode == 2) { GPOPST(sqstrid);  n=4;}
        GPTRL2(lcpos,mat);      /* calculate translation matrix */
        GPMLX2(mat,3);          /* insert model transform       */
        GPPG2(1,&n,2,geopts);   /* insert square/triangle       */
        GPCLIST();              /* close structure              */
      }
    }
    GPUPWS(wsid,2);             /* update structure             */
  }
  GPCLPH();   /* close graPHIGS              */
}

wholescreen(WSID,dydx)
  int wsid;
  float *dydx;
{
  float dcsize[3] ;
  int status, units, rsize[3] ;
  static float window[6] = {0.0,1.0,0.0,1.0,0.0,1.0};
  static float viewpt[6] = {0.0,1.0,0.0,1.0,0.0,1.0};

  GPQADS(wsid,&status,&units,dcsize,rsize) ;
  *dydx = window[3] = dcsize[1]/dcsize[0];
  viewpt[1] = dcsize[0];
  viewpt[3] = dcsize[1];
  viewpt[5] = dcsize[2];
  GPWSX3(wsid,window,viewpt) ;
}
-----------------------------------------------------------------
```

2. Compile the program using the following command:

```
cc -o model model.c -lgP
```

3. Enter the following into a file called **PROFILE**:

```
* graPHIGS Start-Up - PROFILE for model.c sample
*
AFMMDFT DEFNUC=(0,)
```

4. Run the program by entering:

```
./model
```

A window pops onto the screen. Inside the window is a blue menu on the left, and a blank drawing area on the right. By clicking button 1 on the mouse while the mouse pointer is inside the drawing area, you may add triangles and squares to the model. By selecting a menu item, you may change between square and triangle mode, erase squares and triangles, or exit the program.

Examining the model Program

This section looks at the program in more detail, describing the purpose of various sets of instructions.

Initializing the graPHIGS API

The graPHIGS API has an architecture (high-level design) very similar to that of the X Window System**. With the X Window System, an application (called a *client*) is linked to *XLIB* which can send and receive the *X protocol* (data stream) to and from an *X server*. Analogous to the X concepts of XLIB and an X server are the graPHIGS API concepts of a *shell* and *nucleus* respectively. In the graPHIGS API, any shell/nucleus combination can communicate in one of the following two ways:

1. The application is linked with the graPHIGS API shell which then communicates with a graPHIGS API nucleus running as a separate process either on the same node (machine) or on another node but on the same network. When using this communication method, the nucleus is called a *remote nucleus* and can be shared by multiple graPHIGS API applications.
2. The application is linked with the graPHIGS API shell which is linked directly to the graPHIGS API nucleus so that the application and the graPHIGS API run as a single process. When using this communication method, the nucleus is called a *private nucleus*.

In order to take advantage of the architecture of the graPHIGS API, the following graPHIGS API subroutines must be used to initialize the graPHIGS API and create a workstation. (This program uses a private nucleus.)

```
-----------------------------------------------------------------
GPOPPH ("        ",0);         /* open graPHIGS                */
GPCNC  (ncid,1,0,0);           /* connect to graPHIGS nucleus  */
GPCRSS (ssid,ncid,1,0);        /* create structure store       */
GPSSS  (ssid);                 /* select structure store       */
GPCRWS (wsid, ncid,1,"*","X        ",0); /* create workstation */
GPASSW (wsid,ssid);            /* link structure store to ws   */
-----------------------------------------------------------------
```

Using Structure Hierarchy

Another new concept is the use of structure hierarchy. The following code is used to create the main geometry structure. Inside this structure, you execute two sub-structures; one to contain the squares, and one to contain the triangles. The capability to arrange structures into a hierarchy is very useful when you wish to:

- Separate your data into logical groups (e.g., menus vs. geometry)
- Mimic the hierarchy of a real world object (e.g., links in a robot)
- Instance the same geometry multiple times (e.g., wheels on a car).

```
-----------------------------------------------------------------
GPOPST(topstrid);      /* open geometry structure             */
GPEF  (2);             /* set edge on                         */
GPIS  (2);             /* set interior style solid            */
GPECI (7);             /* set edge color WHITE                */
GPICI (2);             /* set interior color RED              */
GPEXST(tristrid);      /* execute triangle structure          */
GPICI (4);             /* set interior color RED              */
GPEXST(sqstrid);       /* execute square structure            */
GPCLST();              /* close geometry structure            */
-----------------------------------------------------------------
```

Creating Menus and Adding Pick Identifiers

The next section of code creates a simple graPHIGS API menu. It defines a list of menu items (text with background 2D polygons) using a separate graPHIGS API stucture. With each menu item, you also add a pick identifier which is returned by the graPHIGS API whenever the corresponding menu item is picked. Prior to defining the menu items, the Add Class Name to Set (**GPADCN**) subroutine must be used to make the menu items *pickable*.

```
-----------------------------------------------------------------
GPTRL2(boxoff,mat);    /* calculate translation matrix        */
GPOPST(menustrid);     /* open menu structure                 */
GPEF  (2); GPIS  (2);   GPECI (7); GPICI (4);  /* interior    */
GPTXCI(7); GPTXAL(3,4); GPTXPR(3); GPCHH (0.6);/* edge attr   */
n=1;GPADCN(1,&n);      /* add class to set, for picking       */
n=4;
for (i=0;i<7;i++) {
  GPPKID(i);           /* insert pick id                      */
  GPPG2 (1,&n,2,boxpts);  /* insert menu item, polygon        */
  GPTX2(txtpos,strlen(menus[i]),menus[i]);   /* text
*/
  GPMLX2(mat,2);       /* translate to next menu item         */
}
GPCLST();              /* close menu structure                */
-----------------------------------------------------------------
```

Picking Menu Items

In order to be able to pick the menu items just created, you must first activate a pick device by using the Set Pick Mode (**GPPKMO**) subroutine. You must also set a pick filter to match the class name added to the menu structure by using the Set Pick Filter (**GPPKF**) subroutine.

```
------------------------------------------------------------------
n=1;
GPPKF (wsid,1,1,&n,0,;&n);  /* set pick filter, same class     */
GPPKMO(wsid,1,3,2);        /* pick: event mode                 */
GPLCMO(wsid,1,3,2);        /* locator: sample mode             */
GPUPWS(wsid,2);            /* update workstation               */
------------------------------------------------------------------
```

Using the Whole Window

In the previous programs (**square**, **flex**, **using3d** and **shade**), only a single view has been used. This program defines two views; one for the geometry, and one for the menu.

Also, you may have noticed in the previous programs that the X window on the IBM RS/6000 was not completely used. By default, the graPHIGS API uses a square display surface within a rectangular X window. This program includes a **wholescreen** utility function to instruct the graPHIGS API to use the entire X window. However, the aspect ratio is now no longer guaranteed to be 1:1. In order for geometry to appear undistorted, you must insure that the aspect ratio for the view's window and viewport values match that of the X window. The **wholescreen** utility function returns the X window aspect ratio which is then used to modify the view's window and viewport values.

```
-------------------------------------------------------------------
  wholescreen(wsid,&dydx);  /* use whole screen, set WSX        */
  window1[3] *= dydx;    /* change window and viewport defs     */
  window2[3] *= dydx;    /*   to match screen aspect ratio      */
  viewpt1[3] *= dydx;
  viewpt2[3] *= dydx;
  GPARV (wsid,viewid1,topstrid,1.0);    /* link root to view    */
  GPARV (wsid,viewid2,menustrid,1.0);   /* link root to view    */
  GPVMP2(wsid,viewid1,window1,viewpt1); /* set view mapping     */
  GPVMP2(wsid,viewid2,window2,viewpt2); /* set view mapping     */
  GPVCH (wsid,viewid1,2,2,2,  1,0,  2,1,  2); /* activate view  */
  GPVCH (wsid,viewid2,2,2,2,  1,0,  1,1,  2); /* activate view  */
  GPVIP (wsid,0,viewid1,2); /* set view input priority 0>1      */
.
.
.
/* utility to use whole screen */
wholescreen(wsid,dydx)
        int wsid;
        float *dydx;
{
        float dcsize[3] ;
        int status, units, rsize[3] ;
        static float window[6] = {0.0,1.0,0.0,1.0,0.0,1.0};
        static float viewpt[6] = {0.0,1.0,0.0,1.0,0.0,1.0};

        GPQADS(wsid,&status,&units,dcsize,rsize) ;
        *dydx = window[3] = dcsize[1]/dcsize[0];
        viewpt[1] = dcsize[0] ;
        viewpt[3] = dcsize[1] ;
        viewpt[5] = dcsize[2] ;
        GPWSX3(wsid,window,viewpt) ;
}
-------------------------------------------------------------------
```

Using the Input Loop

The last section of code contains the input loop. The Await Event (**GPAWEV**) subroutine instructs the graPHIGS API to wait for an event to occur (PICK or LOCATOR). If a pick event occurs, the pick information is retrieved using the Get Pick (**GPGTPK**) subroutine. The program uses the returned pick identifier value to determine which menu item was picked and take the appropriate action. If a locator event occurs, the program adds a square or triangle to the appropriate structure depending on the current mode.

```
-------------------------------------------------------------------
  while (mode != 0) {        /* loop to look for events         */
    GPAWEV(100.0,&major,&class,&minor); /* wait for event       */
    if (class == 5 && minor == 1) {     /* pick event?          */
      GPGTPK(1,&depth,&pickpath);       /* get pick information */
      if (pickpath.pkid == 6) mode = 1; /* square mode          */
      if (pickpath.pkid == 3) mode = 2; /* triangle mode        */
      if (pickpath.pkid == 0) mode = 0; /* exit                 */
      if (pickpath.pkid == 5) GPEST(tristrid); /* erase tris    */
      if (pickpath.pkid == 2) GPEST(sqstrid);  /* erase squares */
    }
    if (class == 1 && minor == 1) {
      GPGTLC(&lcview,lcpos);            /* get mouse position   */
      if (lcview == viewid1) {          /* open proper structure*/
        if (mode == 1) { GPOPST(tristrid); n=3;}
        if (mode == 2) { GPOPST(sqstrid);  n=4;}
        GPTRL2(lcpos,mat);      /* calculate translation matrix */
        GPMLX2(mat,3);          /* insert model transform       */
        GPPG2(1,&n,2,geopts);   /* insert square/triangle       */
        GPCLST();               /* close structure              */
      }
    }
    GPUPWS(wsid,2);             /* update structure             */
  }
-------------------------------------------------------------------
```

Chapter 7. Continuing with the graPHIGS API

With five simple sample programs, you have been introduced to many of the fundamental concepts behind the graPHIGS API. For further information on these graPHIGS API concepts, read the appropriate sections in the following manuals:

- *The graPHIGS Programming Interface: Understanding Concepts*
- *The graPHIGS Programming Interface: Writing Applications*
- *The graPHIGS Programming Interface: Technical Reference*
- *The graPHIGS Programming Interface: Subroutine Reference*
- *The graPHIGS Programming Interface: Quick Reference*

Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Dept. LRAS/Bldg. 003
11400 Burnet Road
Austin, TX 78758-3498
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Trademarks

Readers' Comments — We'd Like to Hear from You

The graPHIGS Programming Interface: Getting Started

Publication No. SC33-8198-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name	Address
Company or Organization	
Phone No.	

Readers' Comments — We'd Like to Hear from You
SC33-8198-00

IBM

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Information Development
Department H6DS-905-6C006
11501 Burnet Road
Austin, TX 78758-3493

Fold and Tape **Please do not staple** Fold and Tape

SC33-8198-00

Cut or Fold
Along Line

Printed in U.S.A.

SC33-8198-00