sun
microsystems

Getting Started with SunPHIGS

Part Number: 800-3061-10
Revision A of 30 August 1989

Getting Started with SunPHIGS

Part No: 800-3061-10
Revision A of 30 August 1989
SunPHIGS Release 1.1

Contents

Chapter 1 An Overview of *PHIGS* **3**

1.1. Introduction 3

1.2. *PHIGS*:
Programmer's Hierarchical Interactive Graphics System 3

Structure Elements 4

Traversal-time Binding 5

Central Structure Store 5

Workstation 6

Input Devices 6

State Variables 7

Description Tables 7

State Lists 7

Chapter 2 Programming in *PHIGS* **13**

2.1. Creating a *PHIGS* Program 13

2.2. Skeleton *PHIGS* Programs 13

2.3. Multiple Libraries 14

2.4. Compiling and Linking *SunPHIGS* Programs 14

C Program Examples 14

Sun Tool workstation 15

Sun Canvas workstation 15

FORTRAN Program Examples 15

Sun Tool workstation 15

Sun Canvas workstation 15

Running a *SunPHIGS* application 15

2.5. SunPHIGS `lint` Library 15

`cfigsl.c` 16

`ffigsl.f` 16

2.6. Creating a *SunPHIGS* Program 17

Sun Tool Workstation Configuration 17

Workstation Type Create 18

Workstation Type Set 19

Workstation Type Get 20

Workstation Type Destroy 20

The *Valulator* Window 20

2.7. Sun Canvas Workstation Configuration 21

The *Message* Window 21

The *SunPHIGS/SunView* Interface in C 22

The *SunPHIGS/SunView* Interface with FORTRAN 22

2.8. Creating Structures 23

Structure Elements 24

Output Primitives 24

`cfigs2.c` 25

`ffigs2.f` 26

Primitive Attributes 26

Polyline Attributes 26

`cfigs3.c` 27

`ffigs3.f` 27

Polymarker Attributes 28

Text Attributes 28

Fill Area Attributes 28

Fill Area Set Attributes 29

Individual Attributes 29

Bundled Attributes 29

`ffigs4.f` 30

2.9. Structure Traversal and Invocation 32

Executing a Child Structure 33

A Complex Structure Hierarchy 33

2.10. Manipulating Structures 35

Change Structure 35

Copy Structure 35

The Structure Element Pointer 35

Set Edit Mode 35

Delete Element 36

Delete Structure 36

Archiving and Retrieving Structures 37

Open Archive File 37

Close Archive File 37

Retrieve Structures 37

2.11. Workstation State List 37

The Display Update State 37

Deferral Modes 37

Modification Modes 38

(WAIT and NIVE) 38

Window Damage 38

pickjet.c 38

2.12. Inquiries 39

Chapter 3 The Transformation Pipeline **43**

3.1. Modeling Transformations 43

Local Modeling Transformation 43

Global Modeling Transformation 45

`fourview.c:` 45

3.2. Viewing 46

Another Look at `fourview.c` 47

Workstation Transformations 48

Chapter 4 Input **53**

4.1. Six Input Classes 53

Sample Uses 53

Locator Devices 54

Stroke Devices 54

Valuator Devices 55

Choice Devices 55

String Devices 55

Pick Devices 55

4.2. SunPHIGS Input Devices 57

4.3. Prompt/Echo Types (PETs) 58

Input Data Records 59

FORTRAN Packed Data Record 59

4.4. Three Operating Modes 60

Request Mode 61

Event Mode 61

Sample Mode 62

Appendix A Examples **65**

A.1. C Examples 65

`axes.c` 65

`canvasattrs.c` 67

`canvasid.c` 69

`cpolygons.c` 70

`ex1.c` 73

`ex2.c` 78

`fourview.c` 83

`fourview_cvs.c` 92

`non_square.c` 101

`pickit.c` 104

`rspheres.c` 113

`spheres.c` 124

`txattrs.c` 128

A.2. FORTRAN Examples 140

`fbundles.f` 140

`ffillset.f` 142

flines.f 145

fmarkers.f 148

fpolygons.f 150

ftext.f 152

ftextall.f 154

toolattrs.f 156

Appendix B Tutorial Examples **161**

B.1. C Examples 161

cfigs1.c 161

cfigs2.c 162

cfigs3.c 163

loc.c 165

pickjet.c 167

B.2. FORTRAN Examples 171

ffigs1.f 171

ffigs2.f 172

ffigs3.f 173

ffigs4.f 174

figscanval.f 177

figstoolval.f 179

Appendix C Viewing **183**

C.1. View Orientation 183

C.2. View Mapping 184

Tables

Table 2-1 SunPHIGS Default Workstation Types 18

Table 2-2 Workstation Type Functions 18

Table 2-3 Traversal Order for Simple Hierarchical Structure Network 32

Table 2-4 Order of Element Execution During Traversal 35

Table 4-1 *PHIGS* Physical Devices 58

Figures

Figure 2-1 SunPHIGS Workstation with Several Primitives 24

Figure 2-2 Structure built by programs `cfigs3.c` and `ffigs3.f` 28

Figure 2-3 Structure using Bundled Edge Flag Attribute 31

Figure 2-4 Simple Hierarchical Structure Network 32

Figure 2-5 Structure Network with Both Structures Posted (to Different Workstations) 33

Figure 2-6 Structure Hierarchy for Robot Arm 34

Figure 2-7 Creating Elements in *Insert* Edit Mode 36

Figure 2-8 Creating Elements in *Replace* Edit Mode 36

Figure 3-1 Structure Network for Robot Arm 44

Figure 4-1 A Plane Structure Hierarchy from which Engine may be Picked 56

Figure 4-2 Pick Path from Selecting Left Engine 56

Figure C-1 View Reference Coordinate System 184

Figure C-2 The Parallel Viewing Model 186

Figure C-3 The Perspective Viewing Model 186

1

An Overview of *PHIGS*

An Overview of *PHIGS* **3**

1.1. Introduction 3

1.2. *PHIGS*:
Programmer's Hierarchical Interactive Graphics System 3

Structure Elements 4

Traversal-time Binding 5

Central Structure Store 5

Workstation 6

Input Devices 6

State Variables 7

Description Tables 7

State Lists 7

1

An Overview of *PHIGS*

1.1. Introduction

SunPHIGS is an implementation of *PHIGS*, the *Programmer's Hierarchical Interactive Graphics System*, designed to run on the complete family of Sun systems and hardware accelerators. *SunPHIGS* diligently adheres to the *PHIGS* standard. The functionality described herein is all part of that standard unless otherwise stated to be *SunPHIGS* implementation dependent.

This tutorial, *Getting Started with SunPHIGS*, is designed to familiarize the user with the concepts of the *PHIGS* standard and the use of the *SunPHIGS* product.

This chapter is an overview of *PHIGS*, and will acquaint the reader with the *PHIGS* standard terminology. Each topic mentioned in the overview will be dealt with, in greater depth, in the later chapters of this document.

Throughout this tutorial, the *PHIGS* standard function names will appear in all capital letters. This is to key the reader that a complete description of the function can be looked up in the *SunPHIGS Reference Manual.* Code fragments, which are excerpts from the full example programs located in the Appendices as well as on the *SunPHIGS1.1* Distribution Tape are provided to enhance the understanding of the dialogue. These will appear in `listing font.` Refer to the complete example program for a better understanding of the functionalities discussed.

1.2. *PHIGS*: Programmer's Hierarchical Interactive Graphics System

PHIGS, the standard, is a hierarchical, interactive system for the definition, modification and display of 3-D and 2-D data.

SunPHIGS is an implementation of the *PHIGS* standard, and hence is a library of functions that support graphics applications on Sun's Workstations. *SunPHIGS* is integrated with Sun's window system environment, and designed to transparently take advantage of hardware acceleration on Sun's CXP models.

PHIGS provides structured organization of graphical data in a central database called the *Central Structure Store* (CSS).

- Graphics objects are defined by a sequence of *elements* (e.g., output primitives, attributes, etc.). These elements are grouped into *structures.* A structure can be thought of as a contiguous linear array of graphical data. Relationships between structures are *hierarchical.*
- The Central Structure Store is a collection of structures in hierarchical (tree-like) networks. Each structure is a *node* in the network. A *root* structure, or

posted structure is the node from which other nodes originate. A *parent* structure is one which references other structures. A *child* structure is one which is referenced by other structures. *Ancestor* and *descendent* structures are those along the traversal path between the posted structure and the last structure in the network. Structures can reference, or execute other structures (non-recursively).

PHIGS provides modern concepts in a graphics standard such as structures, structure editing, traversal-time attribute binding of attributes, and control over detectability, highlighting and visibility of output primitives.

PHIGS separates the definition of graphical data from the actions which display the data. Graphics information is stored in the CSS in the form of structures. These structures may be edited by inserting and deleting structure elements. New structures can be created by copying graphical data from other structures. Images can share component objects. *Objects*, as used in this document, refer to portions of the graphical image to be displayed. Structures can be invoked (executed) from within other structures. All of these features reduce repetition and connectivity problems through repeated use of component objects and the relationships between them. Both construction of image data and user interaction are facilitated.

- Data display is independent of the creation and modification of graphical data. The application program has the ability to define the graphical data stored in the CSS and to control when and how these images are to be viewed on selected workstations.
- Attributes are bound to output primitives during structure network traversal (execution). This allows a structure (*child*) executed from within another structure (*parent*), to "inherit" the attributes of the structure referencing it. Modification of primitive attributes is also permitted dynamically through structure editing.
- Detectability, highlighting and visibility of primitives as groups are provided for in the *name set* functionality.
- Modeling transformations and view indices provide dynamic capabilities which can be stored and edited as structure elements.
- *PHIGS* supports both parallel and perspective projection viewing.

Structure Elements

The basic building blocks of structures are *structure elements*:

- *Output Primitives* are the basic geometric graphics available for display. These are: POLYLINE, POLYMARKER, TEXT, FILL AREA (polygon), FILL AREA SET (polygon with multiple bounds), CELL ARRAY and GENERALIZED DRAWING PRIMITIVES (GDPs) which are implementation dependent primitives. (*SunPHIGS* provides one GDP, the *Polyline Set*).
- *Attributes* control the appearance of output primitives. Examples of functions which set attributes are: SET LINETYPE, SET LINEWIDTH SCALE FACTOR and SET POLYLINE COLOUR INDEX.

- *Modeling Transformations* are 4 × 4 matrices used to map modeling coordinates to world coordinates. The two modeling transformation functions are: SET LOCAL TRANSFORMATION and SET GLOBAL TRANSFORMATION.
- *View Selections* define the orientation of the displayed image relative to the viewer's point-of-view. SET VIEW INDEX is the element used to select a view's use. The SET VIEW REPRESENTATION function is used to define the view on a specific workstation.
- *Structure Invocations* are the most notable feature of the *PHIGS* hierarchical model. EXECUTE STRUCTURE invokes one structure from another.
- The *Name Set* functions control the visibility, highlighting and detectability of groups of primitives in conjunction with workstation *filters*. The two name set functions are *Add Names to Set* and *Remove Names From Set.*
- *Pick Identifiers* associate application supplied indentitifiers with output primitives and are one part of the pick path produced by a pick input device. SET PICK IDENTIFIER, creates a structure element containing a value for the *current pick identifier* which applies to all output primitives following in the structure network.
- *Labels* are used as delimiters to facilitate structure editing. The LABEL function creates a label to mark locations within a structure.
- *Application Data* embeds non-executable data in the database for use by the application only. This data might include comments or part numbers. The APPLICATION DATA function provides this capability.
- GENERALIZED STRUCTURE ELEMENTS (GSEs) are implementation defined and enable access, during traversal, to special control and attribute related actions that are beyond the scope of the structure elements defined by *PHIGS*. *SunPHIGS* provides one GSE, *Set Highlight Colour Index.*

Traversal-time Binding

Traversal-time binding (during structure execution) provides an inheritance capability.

A Structure *inherits* the attributes of its parent (the structure invoking it), much as a subroutine would inherit the values passed in the parameter list. A child structure may modify its own attributes, without affecting the parent structure attributes. Thus the same structure invoked from different parents might inherit different attribute values. The posted structure of a structure network inherits its attributes from the *PHIGS description table* which contains standard system default values.

Central Structure Store

The Central Structure Store is the central graphical database which holds display lists. All data is available to all *workstations*. This provides device independence as there is no workstation-dependent structure storage.

Workstation

A workstation is the logical interface through which the application controls a physical device. There are five types of workstations available in *PHIGS*: the *input-only* workstation which has at least one logical input device; the *output-only* workstation which has only a display area (no input devices); the *input-output* workstation which combines the capabilities of both the input and output only workstations; and the *metafile input* and *metafile output* workstation types. *SunPHIGS* provides two *input-output* workstation types and one *metafile output* workstation type. The *input-output* workstations are described in the following chapter. For information on the *metafile output* workstation, see OPEN WORKSTATION in the *SunPHIGS Reference Manual.*

PHIGS allows for addressing specific workstation capabilities through description tables and state lists.

- The *workstation description table* describes the capabilities and characteristics of a specific device.
- The *workstation state list* contains the workstation values modifiable by an application during a *PHIGS* session and workstation state maintained for you by *PHIGS*. There is one workstation state list for each open workstation.

Input Devices

There are six logical input device classes supported by *PHIGS*:

- A *Locator* device returns a point in world coordinates. INITIALIZE LOCATOR, SET LOCATOR MODE and GET LOCATOR are examples of functions which pemit handling of the locator class of devices.
- A *Stroke* device returns a sequence of points in world coordinates. INITIALIZE STROKE, SET STROKE MODE and GET STROKE are examples of functions which permit handling of the stroke class of devices.
- A *Valuator* device returns a real number. INITIALIZE VALUATOR, SET VALUATOR MODE and GET VALUATOR are examples of functions which permit handling of the valuator class of devices.
- A *Choice* device returns a non-negative integer. INTIALIZE CHOICE, SET CHOICE MODE and GET CHOICE are examples of functions which permit handling of the choice class of devices.
- A *Pick* device returns a pick path which identifies a picked primitive. INITIALIZE PICK, SET PICK MODE and GET PICK are examples of functions which permit handling of the pick class of devices.
- A *String* device returns a character string. INITIALIZE STRING, SET STRING MODE and GET STRING are examples of functions which permit handling of the string class of devices.

Each logical input device can be operated in one of three *operating modes*:

- In *Request* mode, the application prompts for and waits for user input. REQUEST <*device class*> is the function which requests the current measure of a device of that class.

- In *Sample* mode, the application calls for the current value of a device without waiting for action from the user. SAMPLE <*device class*> is the function which samples the current measure of the specified device.
- In *Event* mode the operator enters input into a central input queue and the application reads from the queue. AWAIT EVENT moves the event from the input queue to the current event report and the GET <*device class*> function retrieves the device measure from the current event report.

State Variables

The value of four state variables define the current *PHIGS* operating state. Individual *PHIGS* functions can be executed only in the proper state.

- The *PHIGS system state variable* reflects whether *PHIGS* is open (PHOP) or closed (PHCL).
- *Workstation state variable* indicates whether any workstation is open (WSOP) or all are closed (WSCL).
- The *structure state variable* designates whether any structure is open (STOP) or all are closed (STCL).
- The *archive state variable* specifies whether any archive file is open (AROP) or all are closed (ARCL).

Description Tables

Tables contain data which *PHIGS* uses to display structures.

Description Tables contain the initial values and system limits. The data in these tables are initialized to standard and implementation dependent values when *PHIGS* is opened. Values in description tables may be inquired but not modified by the application.

- The *PHIGS description table* contains the system parameters and limits.
- The *workstation description tables* contain initial state values and predefined attribute bundle definitions, for each available workstation type.

State Lists

State Lists contain current application-dependent values. These tables are dynamically modifiable during application execution.

- The *PHIGS state list* contains the system state values.
- The *workstation state lists*, one for each open workstation, contain the workstation values active during a *PHIGS* session.
- The *structure state lists*, one for each defined structure, contain all the structure elements and the list of workstations to which the structure is posted.
- The *error state list* contains error information.

2

Programming in *PHIGS*

Programming in *PHIGS* **13**

2.1. Creating a *PHIGS* Program 13

2.2. Skeleton *PHIGS* Programs 13

2.3. Multiple Libraries 14

2.4. Compiling and Linking *SunPHIGS* Programs 14

C Program Examples 14

Sun Tool workstation 15

Sun Canvas workstation 15

FORTRAN Program Examples 15

Sun Tool workstation 15

Sun Canvas workstation 15

Running a *SunPHIGS* application 15

2.5. SunPHIGS `lint` Library 15

`cfigsl.c` 16

`ffigsl.f` 16

2.6. Creating a *SunPHIGS* Program 17

Sun Tool Workstation Configuration 17

Workstation Type Create 18

Workstation Type Set 19

Workstation Type Get 20

Workstation Type Destroy 20

The *Valulator* Window 20

2.7. Sun Canvas Workstation Configuration 21

The *Message* Window 21
The *SunPHIGS/SunView* Interface in C 22
The *SunPHIGS/SunView* Interface with FORTRAN 22
2.8. Creating Structures 23
Structure Elements 24
Output Primitives 24
`cfigs2.c` 25
`ffigs2.f` 26
Primitive Attributes 26
Polyline Attributes 26
`cfigs3.c` 27
`ffigs3.f` 27
Polymarker Attributes 28
Text Attributes 28
Fill Area Attributes 28
Fill Area Set Attributes 29
Individual Attributes 29
Bundled Attributes 29
`ffigs4.f` 30
2.9. Structure Traversal and Invocation 32
Executing a Child Structure 33
A Complex Structure Hierarchy 33
2.10. Manipulating Structures 35
Change Structure 35
Copy Structure 35
The Structure Element Pointer 35
Set Edit Mode 35
Delete Element 36
Delete Structure 36
Archiving and Retrieving Structures 37
Open Archive File 37
Close Archive File 37
Retrieve Structures 37
2.11. Workstation State List 37
The Display Update State 37

Deferral Modes 37
Modification Modes 38
(WAIT and NIVE) 38
Window Damage 38
pickjet.c 38
2.12. Inquiries 39

2

Programming in *PHIGS*

2.1. Creating a *PHIGS* Program

Now that we have had a look at An Overview of *PHIGS*, there is no time like the present to begin programming in *PHIGS*. This section is designed to teach the basics of programming with *PHIGS*. *SunPHIGS* -specific and more complex programs will follow in this chapter and in the appendices. Since *SunPHIGS* provides both a C and FORTRAN binding, different topics will incorporate code fragments in one or both languages. The *PHIGS* standard function name will appear throughout this text in all capital letters. The function language binding names will appear within the example code.

The *tutorial* directory within the *examples* directory on the *SunPHIGS 1.1* distribution tape includes some simple programs and the programs described in this chapter. The programs can typically be found in Appendix B of this manual and with the following pathname:

```
/usr/lib/phigs1.1/examples/tutorial
```

More complex examples (as well as a makefile for compiling them) are provided in the *examples* directory, typically found in Appendix A of this manual and with the following pathname:

```
/usr/lib/phigs1.1/examples
```

2.2. Skeleton *PHIGS* Programs

Two example programs, one in C and the other in FORTRAN, are provided in this chapter. They contain basic skeleton *PHIGS* programs. A brief explanation of the parameter list is provided. The *SunPHIGS Reference Manual* should be consulted for a full explanation of each individual function. The proper header file must be included in the program in order to open a workstation or access *SunPHIGS* constant definitions.

□ *PHIGS* must be opened (OPEN PHIGS) prior to calling any other *PHIGS* function. This function will initialize the *PHIGS* environment and enable access to all other *PHIGS* functions. The first parameter is the error message file and the second parameter the amount of memory units, which can be ignored for now.

□ Once *PHIGS* has been opened, it is possible to open a workstation (OPEN WORKSTATION). A *workstation identifier* is passed as the first parameter and its value will be used to identify the workstation in subsequent *PHIGS* function calls. The second parameter is the *connection identifier* of the workstation. This value is used when interfacing the *SunPHIGS* workstation with

SunView subwindows. This will be described in a later section devoted to *SunPHIGS* Workstation Configuration. The last parameter is the type of workstation to open. The available workstation types for *SunPHIGS* are the *Sun Tool*, the *Sun Canvas*, and the *CGM Output* workstation. We will begin with the *SunPHIGS Sun Tool* workstation type.

- The UNIX `sleep` command is called in order to pause long enough to view the opened workstation. We then close the workstation, (CLOSE WORKSTATION) and lastly, we close *PHIGS* (CLOSE PHIGS).

2.3. Multiple Libraries

There are several libraries relating to *SunPHIGS*:

`-lphigs` The basic *SunPHIGS* library. At *run time*, this library will switch operation to the most efficient floating point support available.

`-lphigs77` The FORTRAN compatibility library, `-lphigs77`, must be linked to FORTRAN-based *SunPHIGS* applications **before any other *SunPHIGS* libraries.**

`-llphigs` If a program uses a *Sun Canvas* workstation, it must be linked with the `-llphigs` (notice the extra 'l' before `phigs`) library **ahead of the** *SunPHIGS* library, `-lphigs`.

Note, `-l`*library* is the UNIX convention for the library files which reside in `/usr/lib` (e.g. `-lphigs` is equivalent to `/usr/lib/libphigs.a`). Additional *SunPHIGS* libraries are replacements for `-lphigs` that support a specific floating point option. These libraries have names starting with `-lphigs`. To prepare an application with one of these floating point specific libraries, you must also use the appropriate compiler switch when compiling and linking. The instructions given in this section prepare programs to select floating point support at run time.

2.4. Compiling and Linking *SunPHIGS* Programs

This section illustrates procedures to compile a *PHIGS* program and link it to the appropriate *SunPHIGS* library. More complete instructions may be found in the *SunPHIGS Software Installation Guide*. Other Sun libraries and the mathematics library must also be linked to any *SunPHIGS* application, as shown. The order in which the libraries is linked is critical.

The *Sun Canvas* workstation type, introduced later, requires special linking instructions, which are given here.

The files named `Makefile` in the *examples* and *tutorial* directories are `make(1)` description files for compiling and linking to *SunPHIGS*. The makefile in each directory will compile the programs provided in that directory.

C Program Examples

Sun Tool workstation

A C application that does not use *Sun Canvas* workstations may be compiled and linked in one step, as follows:

```
hostname% cc ex1.c -o ex1 -lphigs \
  -lsuntool -lsunwindow -lpixrect -lm
```

Sun Canvas workstation

A C application that uses a *Sun Canvas* workstation also requires the `-llphigs` library. Such a program may be compiled and linked in one step, as follows:

```
hostname% cc ex2.c -o ex2 -llphigs -lphigs \
  -lsuntool -lsunwindow -lpixrect -lm
```

FORTRAN Program Examples

Sun Tool workstation

A FORTRAN application that does not use *Sun Canvas* workstations may be compiled and linked in one step, as follows:

```
hostname% f77 flines.f -o flines -lphigs77 -lphigs \
  -lsuntool -lsunwindow -lpixrect -lm
```

Sun Canvas workstation

A FORTRAN application that uses a *Sun Canvas* workstation also requires the `-llphigs` library. Such a program may be compiled and linked in one step, as follows:

```
hostname% cc fmarkers.f -o fmarkers canvasid.o -lphigs77 \
  -llphigs -lphigs -lsuntool -lsunwindow -lpixrect -lm
```

Running a *SunPHIGS* application

To run the program, merely use its name. For example:

```
hostname% ex1
```

2.5. SunPHIGS `lint` Library

For C programmers, *SunPHIGS* provides a `lint` library which provides type checking beyond the capabilities of the C compiler. For example, to use the *SunPHIGS* `lint` library to check a program `ex1.c` against the *SunPHIGS* function calling sequences, a command like the following is used:

```
hostname% lint ex1.c -lphigs
```

Note that the error messages `lint` generates are mostly warnings, and may not have an effect on the operation of the program. For a detailed explanation of `lint`, see the `lint` chapter in the SunOS 3.2 manual, *Programming Utilities for the Sun Workstation*, or the 4.0 manual, *Programming Utilities and Libraries*.

cfigs1.c

cfigs1.c is a simple skeleton *PHIGS* program in C which demonstrates how to open and close a *SunPHIGS Sun Tool* workstation.

Note: It is necessary to include the declarations in phigs.h in order to open a workstation or access *SunPHIGS* constants.

```
#include <phigs/phigs.h>

main()
{
```

Open *PHIGS*.

```
popenphigs((Pchar*)NULL, PDEFAULT_MEM_SIZE);
```

Open a workstation numbered 1 and pause for 5 seconds.

```
popenws(1,(Pconnid)NULL,phigs_ws_type_sun_tool);
sleep(5);
```

Close workstation 1 and close *PHIGS*.

```
pclosews(1);
pclosephigs();
}
```

ffigs1.f

ffigs1.f is a simple skeleton *PHIGS* program in FORTRAN which demonstrates how to open and close a *SunPHIGS Sun Tool* workstation.

Note: It is necessary to include the declarations in phigs77.h in order to open a workstation or access *SunPHIGS* constants.

```
include '/usr/include/phigs/phigs77.h'
```

Open *PHIGS* using logical unit number 6 for the *SunPHIGS* error file.

```
call popph(6, 0)
```

Open a workstation numbered 1 and pause for 5 seconds.

```
call popwk(1, 0, phigswsttool)
call sleep(5)
```

Close workstation 1 and close *PHIGS*.

```
call pclwk(1)
call pclph

stop
end
```

2.6. Creating a *SunPHIGS* Program

SunPHIGS integrates graphics into the *Sun View* windowing environment. *SunView* is a system which supports interactive, graphics-based applications running within windows. In this section we will learn how to configure *SunPHIGS* workstations and how to integrate them with the *SunView* window management system. For more information on *SunView*, see the *SunView Programmer's Guide*. *SunPHIGS* has two types of *input/output* workstation types:

- The *Sun Tool* workstation - a *SunPHIGS* generated graphics display window and *message* text subwindow.
- The *Sun Canvas* workstation - a *SunView* generated graphics display with optional message text subwindow.

These two workstation types will be discussed separately. For the *SunPHIGS* user, new to the Sun window system environment, *SunPHIGS* provides a default workstation tool (*Sun Tool*) which consists of a *SunView* frame and subwindows. No knowledge of *SunView* is necessary to generate this type of workstation.

For the experienced *SunView* user, *SunPHIGS* provides the *Sun Canvas* workstation type. Here the application creates its own *SunView* frame and subwindows and *SunPHIGS* uses these as *PHIGS* workstations. This allows the programmer to incorporate *SunPHIGS* workstations into a *SunView* application.

The two programs above create *PHIGS* workstations of the *Sun Tool* workstation type. This generates a *SunView* frame with two subwindows: a *graphics* window and a *message* text subwindow. The message window will appear as the default for the *Sun Tool* workstation, but can be suppressed. An optional *Valuator* device window is also available with the *Sun Tool* workstation.

With the *Sun Canvas* type workstations the message text subwindow is optional with the default being no text subwindow. An optional *Valuator* device window is also available with the *Sun Canvas* workstation.

All output primitives are displayed in the graphics window. Messages from the MESSAGE function appear in the message window. Messages will display on the graphics window if the message window is not available. The message window is also used as the *String* device for workstations of category OUTIN. The *Valuator* window contains the *Valuator* devices. This *Valuator* window is only visible if a *Valuator* device is active. The code fragments below demonstrate the creation and modification of the *Sun Tool* and *Sun Canvas* workstation types.

Sun Tool Workstation Configuration

The OPEN WORKSTATION function requires a *workstation type* parameter. The predefined workstation types for a C program are:

Table 2-1 *SunPHIGS Default Workstation Types*

Type	C Name	FORTRAN Name
Sun Tool	`phigs_ws_type_sun_tool`	`phigswsttool`
Sun Canvas	`phigs_ws_type_sun_canvas`	`phigswstcanvas`
CGM Output	`phigs_ws_type_cgm_out`	`phigswstcgmout`

As shown in the examples `cfigs1.c` and `ffigs1.f` above, these *base workstation types* can be used directly with the OPEN WORKSTATION call, or they can be used to create new workstation types.

Table 2-2 *Workstation Type Functions*

FORTRAN Subroutine	*C Function*
`phigswstcreate (basewst, newwst)`	`phigs_ws_type_create`
`phigswstdestroy (wst)`	`phigs_ws_type_destroy`
`phigswstset (wst, attribute, value)`	`phigs_wst_type_set`
`phigswstget (wst, attribute, value)`	`phigs_wst_type_get`

The *SunPHIGS* WORKSTATION TYPE CREATE function is used to copy an existing workstation type. It can also be modified by this function in a C program, or subsequently modified by the *SunPHIGS* WORKSTATION TYPE SET function in both C and FORTRAN. This new workstation type can then be passed to OPEN WORKSTATION.

Workstation Type Create

We will begin by using the *Sun Tool* workstation type and creating a new workstation with a minor modification.

```
Pwstype WStype;
popenphigs((Pchar*)NULL, PDEFAULT_MEM_SIZE);

WStype = phigs_ws_type_create(phigs_ws_type_sun_tool,
    PHIGS_TOOL_LABEL, "SunPHIGS Tool Workstation",
    0);

popenws(WS1, (Pconnid)NULL, WStype);
```

Each workstation type has a *workstation description table* associated with it. This table contains the *PHIGS*-specific data describing workstation capabilities and some *SunPHIGS*-specific data. The *SunPHIGS*-specific data, as well as some of the *PHIGS* data, can be changed by the application before opening the workstation. The fields in the workstation description table are modified by calling either WORKSTATION TYPE CREATE or WORKSTATION TYPE SET and specifying *attribute-value* pairs. The *attribute* specifies the field to change and the *value* is the value to assign to the field. Both the C and FORTRAN constants for the workstation description table attributes are described in the WORKSTATION TYPE SET man page. The C constants are enumerated types defined in `phigs.h`. The FORTRAN constants are defined with *Parameter* statements in `phigs77.h`.

After *PHIGS* has been opened, a new workstation of type *Sun Tool* is created and the tool label (or window namestripe) is modified. The default label for a *Sun Tool* workstation is "PHIGS Workstation". Applications might want to label each workstation differently. Here the `PHIGS_TOOL_LABEL` *attribute* is used to change the label to read "SunPHIGS Tool Workstation". Once the new workstation type, `WStype`, has been created, this new type is passed to the OPEN WORKSTATION function. Note that the original `phigs_ws_type_sun_tool` type workstation cannot be modified and if opened would still contain the default label. `WStype` is a modified *copy* of the original *Sun Tool* type. Also, `WStype` can now be used as the base workstation type by a future call to WORKSTATION TYPE CREATE.

In C example above, only one attribute-value pair was used, `PHIGS_TOOL_LABEL`, "SunPHIGS Tool Workstation". With C programs, attribute-value pairs are contained in a 0 terminated list of workstation type attributes and corresponding values. This *attribute-value-list* can be used to modify workstation description table fields with either WORKSTATION TYPE CREATE or WORKSTATION TYPE SET.

The code fragment that follows uses this list to configure a *SunPHIGS* workstation to be 400 × 600 pixels and to implicitly define the aspect ratio of the workstation device space so that the image is rendered on the full display surface. For the complete program which uses the code fragments below see `non_square.c` in the *examples* directory.

```
Pwstype         wst;
float           xmax = 400.0, ymax = 600.0;

wst = phigs_ws_type_create(phigs_ws_type_sun_tool,
    PHIGS_TOOL_WIDTH,  (int)xmax,
    PHIGS_TOOL_HEIGHT, (int)ymax,
    PHIGS_DEVICE_COORD_XMAX_PTR, &xmax,
    PHIGS_DEVICE_COORD_YMAX_PTR, &ymax,
    0);
```

Workstation Type Set

Although the WORKSTATION TYPE CREATE function is also used to create new workstation types with FORTRAN programs, separate calls to WORKSTATION TYPE SET must be used to modify the workstation attribute values before opening a new workstation.

Below is an example of workstation type creation and modification using the FORTRAN binding. This code fragment configures a *SunPHIGS* workstation to be 600 × 600 pixels, disables the message window so that only the graphics window is displayed, and changes the tool label to read "FORTRAN phigswttool" . For the complete program which uses the code fragments below see `toolattrs.f` in the *examples* directory.

```
integer wkid, strid, wstooltype, labellen
label = 'FORTRAN phigswsttool'

call phigswstcreate(phigswsttool, wstooltype)
call phigswstset(wstooltype, PHIGSTOOLHEIGHT, 600)
call phigswstset(wstooltype, PHIGSTOOLWIDTH, 600)
call phigswstset(wstooltype, PHIGSTEXTSW, PHIGSNONE)
call phigswstset(wstooltype, PHIGSTOOLLABEL, label(1:20))
```

Workstation Type Get

In addition to the workstation configuration functions discussed above there are also the WORKSTATION TYPE GET and WORKSTATION TYPE DESTROY functions. WORKSTATION TYPE GET is used to retrieve the value of a specified workstation type field.

From the same program example used above, `toolattrs.f`, WORKSTATION TYPE GET is used to retrieve the current value of the tool label and display it with the *Text* primitive in the graphics window.

```
call phigswstget(wstooltype, PHIGSTOOLLABEL, labellen, label
call pschh(.02)
call ptx(.1, .4, label)
```

Here the retrieved value is returned as an output parameter. In FORTRAN, the label length is also returned when the workstation attribute value is a string. In C, this function returns the requested information as the value of the function. This return value must be cast to the appropriate type when assigning it to a variable.

In `txattrs.c`, from the *examples* directory, WORKSTATION TYPE GET is used to retrieve the base type of the workstation type.

```
Phigs_base_name  n;

n = (Phigs_base_name)phigs_ws_type_get( *wst, PHIGS_BASE_NAM
```

Workstation Type Destroy

WORKSTATION TYPE DESTROY deallocates any memory used by the workstation type. After a workstation type has been destroyed, it can no longer be used.

The *Valulator* Window

The *Valuator* device window contains the *Valuator* devices, and is only visible if a *Valulator* device is active. Workstation attribute values may be set to control the position of the *Valulator* window on the display.

The code fragment below is from `figstoolval.f` in the *tutorial* directory. This program demonstrates the creation of the *Valuator* window, but does not utilize the *Valuator* device. See `rspheres.c` in the *examples* directory for a complete program which uses a *Valuator* device and the section on Input for a discussion on setting up logical input devices.

Create a new workstation of type `phigswsttool`; set the attribute values for positioning the *Valuator* window; open the workstation.

```
call phigswstcreate(phigswsttool, toolval)
call phigswstset(toolval, PHIGSVALPANELX, 650)
call phigswstset(toolval, PHIGSVALPANELY, 25)
call popwk(wkid, 0, toolval)
```

INITIALIZE VALUATOR and SET VALUATOR MODE to activate the device.

```
call pinvl(wkid, valdev, 0.5, pet, 0.0,1.0,0.0,1.0, -1.0,1.0, ldr,rec)
call psvlm(wkid, valdev, PEVENT, PECHO)
```

The man page for WORKSTATION TYPE SET, which describes all the workstation attribute fields, states that PHIGS_VAL_PANEL_X[Y] and PHIGSVALPANELX[Y] specify the desired location, in screen coordinates, of the *Valuator* panel. These coordinates are in relation to the top left corner of the workstation's window. Therefore, in the above example, the *Valuator* window will be seen 50 pixels to the right of the workstatation window and 25 pixels down.

2.7. Sun Canvas Workstation Configuration

As pointed out in the Creating a *SunPHIGS* Program section, an application may want to integrate *SunPHIGS* workstations with a *SunView* application. This would allow the programmer to have *Sun View* windows and *SunPHIGS* windows (workstations) running together in the same application.

In this section we will demonstrate different workstation configurations using a *Sun Canvas* workstation type.

To use a *Sun Canvas* workstation, first a *SunView* canvas subwindow must be created and the Canvas *handle* passes to OPEN WORKSTATION as the *connection identifier*. To learn more about creating *Sunview* subwindows, see the *SunView Programmers Guide*. Canvasattrs.c and fmarkers.f, in the *examples* directory, are complete programs which use *SunView* to produce a *Canvas* subwindow. The code fragments below are from canvasattrs.c and demonstrate creating both a *SunView Canvas* and a *Text* subwindow handle, which are then passed to *SunPHIGS*.

The *Message* Window

The default for a *Sun Tool* workstation type is to display the *message* text subwindow. The default for the *Sun Canvas* workstation is not to display the message window. In order to bring up the *text* subwindow, a new workstation type must be created, and the attribute-value pair (PHIGS_TEXTSW, *text handle*) used to enable the text subwindow display. The effect of many *Sun Tool* workstation attributes can be created for *Sun Canvas* workstations by using *SunView* attributes. An example is the frame label, shown below.

The *SunPHIGS/SunView* Interface in C

Obtain *SunView canvas* and *textsw* handles.

```
    frame = window_create(NULL, FRAME,
        FRAME_LABEL,        "SunPHIGS Canvas Workstation",
        0);

    canvas = window_create(frame, CANVAS,
        WIN_PERCENT_HEIGHT, 90,
        0);

    textsw = (Textsw) window_create(frame, TEXTSW,
        WIN_PERCENT_HEIGHT, 10,
        TEXTSW_IGNORE_LIMIT,TEXTSW_INFINITY,
        0);
```

Open *PHIGS* and create a *Sun Canvas* workstation with a *Message* window. The *SunView textsw* handle is passed as the value for the PHIGS_TEXTSW workstation attribute. Open the modified *Sun Canvas* workstation.

```
popenphigs((Pchar*)NULL, PDEFAULT_MEM_SIZE);

canvaswst = phigs_ws_type_create(phigs_ws_type_sun_canvas,
    PHIGS_TEXTSW, textsw,
    0);

popenws(WS, (Pconnid)canvas, canvaswst);
```

The *SunPHIGS/SunView* Interface with FORTRAN

To create a *Sun Canvas* workstation using the FORTRAN binding, a small C function is used as the interface. `fmarkers.f` calls `canvasid`, a C function which creates the *SunView* canvas subwindow and returns the *Canvas* handle. *SunView* functions cannot be called directly from a FORTRAN application. Included in the `canvasid.c` file is another function, `display`, which activates the *SunView Notifier*, needed to display the windows on the screen.

The code fragments below are from `fmarkers.f` and demonstrate the creation of a *Sun Canvas* workstation from a FORTRAN program. See `canvasid.c`, also in the *examples* directory for the complete C interface function.

```
    integer canvas, canvasid
```

SunView Setup `canvasid` is a C function which returns the connection identifier *canvas* for a *SunPHIGS* canvas workstation.

```
    canvas = canvasid()
```

Open *PHIGS* and open the *Sun Canvas* workstation.

```
    call popph(6, 0)
    call popwk(1, canvas, phigswstcanvas)
```

Call the C/*SunView* interface function to display the *SunView* canvas window.

```
    call display
```

The *Valuator* device window for a *Sun Canvas* workstation type is used in the same way as for a *Sun Tool* workstation. Workstation attribute values are specified using WORKSTATION TYPE SET to control the location for display of the *Valuator* window. The code fragment below is from `figscanval.f` in the *tutorial* directory. This program demonstrates the creation of the *Valuator* window, but does not utilize the *Valuator* device. See `rspheres.c` in the *examples* directory for a complete program which uses a *Valuator* device and the section on *Input* for a discussion on setting up logical input devices.

```
    canvas = canvasid()

    call popph(6, 0)
```

Create a new workstation of type *phigswstcanvas*; set the attribute values for positioning the *Valuator* window; open the workstation.

```
 call phigswstcreate(phigswstcanvas, canvaswst)
 call phigswstset(canvaswst, PHIGSVALPANELX, 625)
 call phigswstset(canvaswst, PHIGSVALPANELY, 25)
 call popwk(wkid, canvas, canvaswst)
```

INITIALIZE VALUATOR and SET VALUATOR MODE to activate the device.

```
call pinvl(wkid, valdev, 0.5, pet, 0.0,1.0,0.0,1.0, -1.0,1.0, ldr,rec)
call psvlm(wkid, 1, PEVENT, PECHO)

call display
```

The WORKSTATION TYPE GET and WORKSTATION TYPE DESTROY functions for the *Sun Canvas* workstation type are used in the same way as with a *Sun Tool* type workstation.

2.8. Creating Structures

Now that we are able to bring up a *SunPHIGS* workstation it is time to begin displaying objects. As previously discussed, graphical objects are ''defined'' by a sequence of elements within structures. However, the functions which create the structure elements to describe an object do not display the object. Structure traversal is the mechanism for displaying objects on a workstation.

First a structure must be created. Any functions which ''reference'' a structure, such as OPEN STRUCTURE, POST STRUCTURE, or EXECUTE STRUCTURE will implicitly create an empty structure (one with no structure elements) if no structure exists with the specified structure identifier. A structure must be open before the application can insert new elements or make changes to existing elements.

Structure Elements

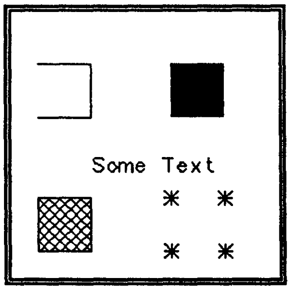
Structure elements are the building blocks of a graphical image. The different structure elements which contribute to the display of a graphical image are:

- *output primitives* - the basic graphic element used to construct an object
- *primitive attributes* - affect the appearance of the output primitives
- *structure invocations* - execute one structure from another
- *modeling transformations* - affect the placement of a graphical object
- *view selections* - control the viewing angle
- *name sets* - control visibility, highlighting and detectability of groups of primitives
- *pick identifiers* - associate names with output primitives
- *labels* - used by the application to mark locations within a structure
- *application data* used by the application to store non-executable data into the CSS
- *generalized structure elements* - implementation defined structure elements which enable access to special control and attribute related actions beyond the scope of the standardized *PHIGS* structure elements

Output Primitives

Output primitives are the basic graphic elements used to construct an object. The POLYLINE function creates a structure element which defines points for drawing a set of connected straight lines. The POLYMARKER function creates an element containing a set of coordinates at which to draw the same symbol. TEXT creates character string data. FILL AREA and FILL AREA SET elements are used to generate polygons. The *PHIGS* standard also defines the CELL ARRAY primitive and the GENERALIZED DRAWING PRIMITIVE (GDP). *SunPHIGS* currently implements one GDP, the *Polyline Set*

Figure 2-1 *SunPHIGS Workstation with Several Primitives*



We will take our original *PHIGS* program and use it as a model to build upon in order to generate a basic output primitive. Since we will be inserting a structure element that uses parameter data, we begin with defining the data.

After opening *PHIGS* and a workstation, a structure is opened. The only parameter to OPEN STRUCTURE is a *structure identifier*, which is used to specify the structure within the CSS to be opened. If no structure exists for the identifier specified, one is created.

Once the structure is opened, any structure elements can be inserted. We will use the POLYLINE function to insert a polyline output primitive structure element. We will then close the structure (CLOSE STRUCTURE) and post it (POST STRUCTURE). It is not until the structure is posted that the polyline will actually be displayed on the workstation.

Note that it is not necessary to close the structure before posting it to the workstation; however, it is a good idea for reasons discussed in the section on deferral modes.

cfigs2.c

cfigs2.c demonstrates the use of structures and structure elements in displaying a simple polyline.

```
#include <phigs/phigs.h>

main()
{
```

Define a series of points for a polyline to be displayed. Ppoint is a structure containing the *x* and *y* coordinates. The array is static so the C language will allow its initialization.

```
static Ppoint xypoints[] = { {0.1, 0.5}, {0.9, 0.5} } ;
```

Open *PHIGS*, open a workstation and open structure 1.

```
popenphigs((Pchar*)NULL, PDEFAULT_MEM_SIZE;
popenws(1, (Pconnid)NULL, phigs_ws_type_sun_tool);
popenstruct(1);
```

Insert the polyline structure element into the open structure. The first parameter is the number of points defining the polyline. The second parameter is a pointer to an array of structures containing the *x-y* coordinates. Close the stucture.

```
ppolyline(2, xypoints);
pclosestruct();
```

Post the structure to the workstation in order to have the structure's contents displayed; pause.

```
ppoststruct(1,1,0.);
sleep(5);
```

Close the workstation and close *PHIGS*.

```
pclosews(1);
pclosephigs();

}
```

ffigs2.f

`ffigs2.f` demonstrates the use of structures and structure elements in displaying a simple polyline.

```
include '/usr/include/phigs/phigs77.h'
```

Define a series of points for a polyline to be displayed.

```
real xpoints(2), ypoints(2)
data xpoints /0.1, 0.9/
data ypoints /0.5, 0.5/
```

Open *PHIGS*, open a workstation and open structure 1.

```
call popph(6, 0)
call popwk(1, 0, phigswsttool)
call popst(1)
```

Insert the polyline structure element into the open structure. The first parameter is the number of points defining the polyline. The other parameters are the arrays containing the *x-y* coordinates. Close the stucture.

```
call ppl(2, xpoints, ypoints)
call pclst
```

Post the structure to the workstation in order to have the structure's contents displayed; pause.

```
call ppost(1, 1, 0.)
call sleep(5)
```

Close the workstation and close *PHIGS*.

```
call pclwk(1)
call pclph
```

Primitive Attributes

The examples above cover the basics of displaying a graphical object on a *SunPHIGS* workstation. However, there are many aspects that we will want to control in order to display the object exactly as we want it to appear (e.g., its color, linetype, etc.). The code fragments below demonstrate the use of output primitive attributes. These functions create structure elements that define the appearance of primitives. For the complete program to which these example lines belong see `cfigs3.c` and `ffigs3.f` in the *tutorial* directory.

Polyline Attributes

The SET <attribute> function creates a structure element which defines the appearance of primitives by assigning an individual attribute value to them.

- For a *polyline*, like the one we just generated, the attribute functions are SET LINETYPE (e.g., solid, dashed, dotted, dash-dotted, etc.), SET LINEWIDTH SCALE FACTOR, which controls the width of the line, and SET POLYLINE COLOUR INDEX. *PHIGS* uses indexed color entries from a color table, therefore this attribute is an index into the workstation's color table.

cfigs3.c

cfigs3.c demonstrates the definition of output primitive attributes to change the appearance of a simple polyline.

```
static Pfloat linewidth = 5.0;
static Pinty colorindex = 5;
```

Create structure elements to set the linetype, the linewidth scale factor and the polyline color index; move the *x-y* coordinates and redraw the polyline.

```
psetlinetype(PLN_DOTDASHDOT);
psetlinewidth(linewidth);
psetlinecolourind(colorindex);


xypoints[0].y += .2;
xypoints[1].y += .2;
ppolyline(2, xypoints);
```

ffigs3.f

ffigs3.f demonstrates the definition of output primitive attributes to change the appearance of a simple polyline.

```
linewidth  = 5.0
colorindex = 5
```

Create structure elements to set the linetype, the linewidth scale factor and the polyline color index; move the *x-y* coordinates and redraw the polyline.

```
call psln(PLNDOTDASHDOT)
call pslwsc(linewidth)
call psplci(colorindex)

ypoints(1) = ypoints(1) + .2
ypoints(2) = ypoints(2) + .2
call ppl(2, xpoints, ypoints)
```

Both programs cfigs3.c and ffigs3.f build a structure which is shown in Figure 2-2. The first polyline appears across the center of the window, and uses default attributes. The second polyline appears above it, and uses attributes set by intervening elements.

Figure 2-2 *Structure built by programs* `cfigs3.c` *and* `ffigs3.f`

Structure 1

POLYLINE (0.1,0.5), (0.9,0.5)
SET LINETYPE *dot-dash-dot-dotted*
SET LINEWIDTH SCALE FACTOR *5.0*
SET POLYLINE COLOUR INDEX *5 (YELLOW)*
POLYLINE (0.1,0.7), (0.9,0.7)

For a complete programming example that displays all polyline linetypes available with this implementation of *PHIGS*, see `flines.f` in the *examples* directory. C examples using polylines can also be found in the *examples* directory.

Polymarker Attributes

The attribute functions for *polymarkers* are very similar to the polyline attribute functions. They are SET MARKER TYPE, SET MARKER SIZE SCALE FACTOR and SET POLYMARKER COLOUR INDEX. The mechanism for creating polymarker attribute elements is the same as for polylines. For a programming example that displays all marker types available with this implementation, see `fmarkers.f` in the *examples* directory. C examples using polymarkers can also be found in the *examples* directory.

Text Attributes

The attribute functions for the *text* output primitive are more extensive.

□ SET TEXT FONT, SET TEXT PRECISION, SET TEXT COLOUR INDEX, SET CHARACTER HEIGHT, SET CHARACTER SPACING, and SET CHARACTER EXPANSION FACTOR all affect the appearance of the text string.

□ SET TEXT PATH, SET TEXT ALIGNMENT, and SET CHARACTER UP VECTOR affect where the text is placed relative to the workstation display.

Several programs in the *examples* directory demonstrate the use of text attributes. `Ftext.f` displays strings in all fonts supported by *SunPHIGS* and the text precision attribute. `Ftextall.f` displays text strings using different character spacing and expansion factors as well as various text paths and character up vectors. `Txattrs.c` in the *examples* directory demonstrates the use of text primitive attributes using the C binding.

Fill Area Attributes

Attribute functions for *fill areas* (polygons) are SET INTERIOR STYLE, SET INTERIOR STYLE INDEX and SET INTERIOR COLOUR INDEX. The interior style attribute determines whether a fill area is to be hollow, solid filled, hatch filled or empty. Hollow fills are represented by a line drawn around the interior. Empty fill areas are invisible on the display. The interior style index attribute is an index into the workstation's hatch table (see Default Tables) selecting which hatch is to be used. The interior colour index is, like the other color index attributes, an index into the color table.

`Fpolygons.f` demonstrates the use of different interior styles. `Cpolygons.c` demonstrates the use of the interior style index. Both of these programs use the *fill area* function to create the structure elements and display polygons in various colors and styles.

Fill Area Set Attributes

Fill area sets have the same attributes as the fill area primitives. In addition, because fill area sets have edges (fill areas do not), there are additional fill area set attribute functions which affect the appearance of the edge itself.

- □ SET EDGE FLAG determines whether or not a fill area set will display an edge.
- □ SET EDGE TYPE, SET EDGEWIDTH SCALE FACTOR and SET EDGE COLOUR INDEX affect the appearance of the edge in the same way the corresponding primitive attributes affect the appearance of a polyline.

In the *examples* directory, `ffillset.f` demonstrates the creation of fill area sets using the FORTRAN binding and `fourview.c`, the C binding.

Individual Attributes

So far, we have been creating structure elements in order to change the appearance of output primitives. These are called *individual* attributes. Creating or editing structure elements in the CSS causes a workstation independent, global change. Primitives which *follow* in the structure network will reflect their new appearance on all open workstations. By default, only individual attributes are used.

Bundled Attributes

Another technique allows for the same primitive to appear differently on different workstations. This is called *Bundled Attributes*. The mechanism for defining workstation dependent attribute values involves selecting attributes from a workstation bundle table. The SET <attribute> REPRESENTATION functions define each bundle table entry.

The bundled attributes for each primitive are as follows:

- □ *polyline* - line type, line width scale factor and polyline color index;
- □ *polymarker* - marker type, marker size scale factor and polymarker color index;
- □ *text* - text font, text precision, text color index, character expansion factor and character spacing.

For fill areas and fill area sets:

- □ *interior* - interior style, interior style index and interior color index.

For fill area sets:

- □ *edge* - edge flag, edge color index, edge line type and edge width scale factor.

Predefined bundle table entries exist for all of the primitives described above. By using the predefined attribute values in a workstation's bundle tables, or modifying the bundle table attributes, the application may use the same primitive data on multiple workstations with different attribute values. The workstation description table contains the default bundle table entries (see *PHIGS Workstation Description Table* (7P)).

Each bundle table entry is associated with an index. The bundle table index is used when defining new bundled attribute values using the SET <attribute> REPRESENTATION functions. The SET <primitive> INDEX function creates a structure element containing this index to select an entry from a workstation's bundle table.

The default source for primitive attributes are the *individual* attribute values. To notify *PHIGS* that the application wants to use the bundle table attribute values, the SET INDIVIDUAL ASF (Aspect Source Flag) function is used.

In the `ffillset.f` example program, a second structure was created in order to display the same fill area set with and without edges on two separate workstations. This could be done just as effectively by changing an attribute value in one workstation's bundle table. We will modify the `ffillset.f` program to use a workstation bundle table entry to display the edge of the fill area set on one workstation but not on the other. Below are code fragments which show this process. For the complete program containing these code fragements see `ffigs4.f` in the *tutorial* directory. For a complete program demonstrating bundle table definition see `fbundles.f` in the *examples* directory.

`ffigs4.f`

`ffigs4.f` demonstrates how to create a fill area set with and without edges using the workstation bundle tables.

```
integer edgeindex, edgetype, edgecolor
real edgewidth

edgeindex = 1
edgetype = 1
edgewidth = 1.0
edgecolor = 1

call popph(6, 0)
call popwk(WS1, 0, tool1)
call popwk(WS2, 0, tool2)
```

Use workstation 1's bundle table to turn the SET EDGE FLAG off. The workstation's bundle table edge flag attribute will need to be changed from the default value (on) to off. See the *PHIGS Workstation Description Table* (*7P*) manual page for workstation default values.

The SET EDGE REPRESENTATION function defines an edge attribute bundle table entry for the specified workstation. This bundle contains the edgetype, width and color as well as whether the edge is displayed. The function below uses the edge flag value `POFF` to specify that the edge is not to be displayed on `WS1`.

```
call psedr(WS1, edgeindex, POFF, edgetype, edgewidth, edgecolor)
```

Open a structure and begin inserting elements. The SET EDGE INDEX function creates a structure element containing an edge index value which selects an entry from the workstation's edge bundle table.

```
call popst(strid)
call psedi(edgeindex)
```

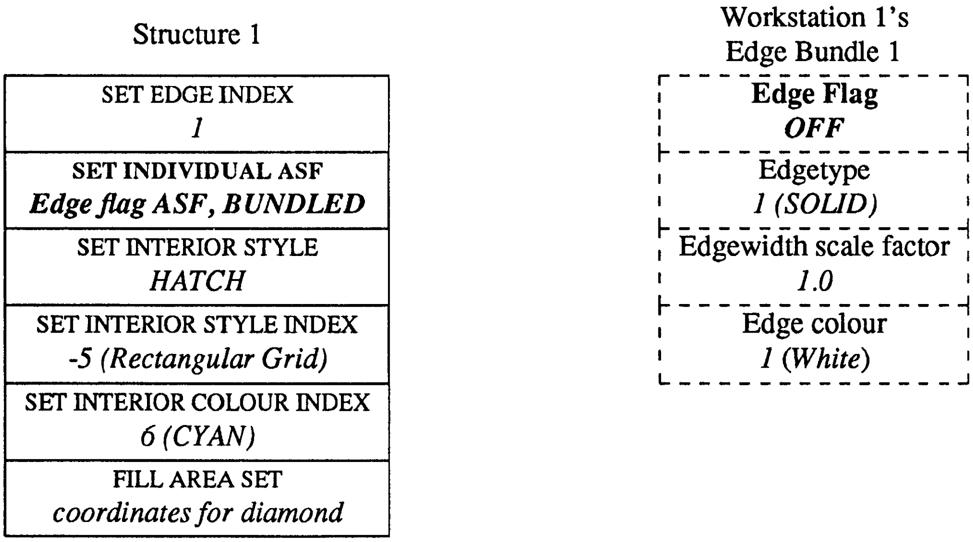
Use the SET INDIVIDUAL ASF function to insert a structure element containing the Aspect Source Flag value; this determines whether the primitive's individual attribute value or the workstation's bundle table attribute value will be used for the specified attribute. Here we designate the bundled attribute for the edge flag. All other edge attributes will use their *individual* values.

```
call psiasf(PEDFG, PBUNDL)
```

Create the individual attribute value elements for the fill area set using SET INTERIOR STYLE, SET INTERIOR STYLE INDEX and SET INTERIOR COLOUR INDEX. Insert the fill area set element into the structure and close it.

```
call psis(PHATCH)
call psisi(hatchindex)
call psici(colourindex)
call pfas(3, boundaries, fasxarr, fasyarr)
call pclst
```

Figure 2-3 *Structure using Bundled Edge Flag Attribute*



Structure 1

SET EDGE INDEX *1*
SET INDIVIDUAL ASF ***Edge flag ASF, BUNDLED***
SET INTERIOR STYLE *HATCH*
SET INTERIOR STYLE INDEX *-5 (Rectangular Grid)*
SET INTERIOR COLOUR INDEX *6 (CYAN)*
FILL AREA SET *coordinates for diamond*

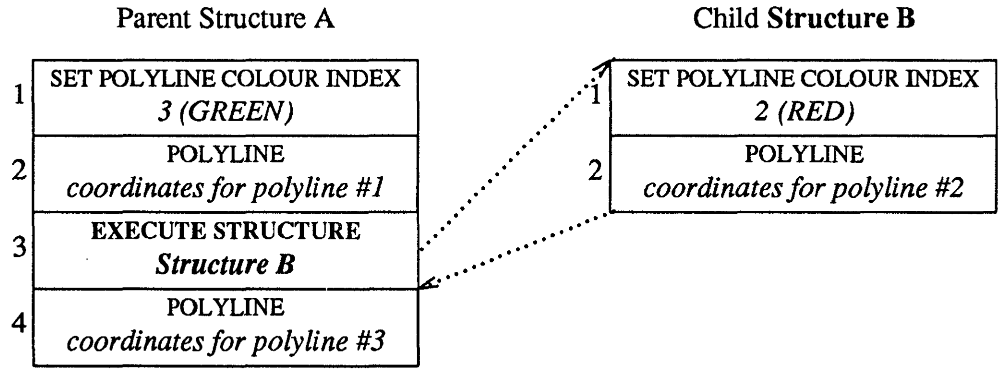
Workstation 1's
Edge Bundle 1

Edge Flag ***OFF***
Edgetype *1 (SOLID)*
Edgewidth scale factor *1.0*
Edge colour *1 (White)*

2.9. Structure Traversal and Invocation

Traversal, the execution of a structure, begins when the structure is posted (POST STRUCTURE). Structures are executed sequentially. Structures can be invoked or executed, from other structures (EXECUTE STRUCTURE). A *referenced* structure (one which is executed from another) is known as a *child* structure. Structures which reference or execute child structures are called *parent* structures. During *traversal*, (the execution of a posted structure) the attribute values are bound to the output primitives. These values are then passed on, or *inherited* by child structures executed after the attribute values have been set. A referenced structure can create its own attribute values using either of the methods discussed above. As shown on Figure 2-4, these values will be bound to any primitives following attribute definition in the child structure and its descendants, but will have no effect upon the primitives in the parent structure.

Figure 2-4 *Simple Hierarchical Structure Network*


Parent Structure A
Child Structure B
1
SET POLYLINE COLOUR INDEX
3 (GREEN)
2
POLYLINE
coordinates for polyline #1
3
EXECUTE STRUCTURE
Structure B
4
POLYLINE
coordinates for polyline #3
1
SET POLYLINE COLOUR INDEX
2 (RED)
2
POLYLINE
coordinates for polyline #2

During traversal, the elements in structure A are executed sequentially until the EXECUTE STRUCTURE, when structure B is executed until its completion. The exectution of structure A is then resumed. Polyline 1 will be green. Polyline 2 will be red. Polyline 3 will be green.

Table 2-3 *Traversal Order for Simple Hierarchical Structure Network*

Structure Name	Element Number	Element Type	Element Data
Structure A	1	SET POLYLINE COLOUR INDEX	*3 (Green)*
Structure A	2	POLYLINE	*coordinates for polyline #1*
Structure A	3	EXECUTE STRUCTURE	*Structure B*
Structure B	1	SET POLYLINE COLOUR INDEX	*2 (Red)*
Structure B	2	POLYLINE	*coordinates for polyline #2*
Structure A	4	POLYLINE	*coordinates for polyline #3*

Below are code fragments which demonstrate the execution of a child structure from within another structure. For the complete program containing these code fragments see `ffillset.f` in the *examples* directory. Most of the C programs in the *examples* directory use the EXECUTE STRUCTURE structure element.

Executing a Child Structure

Open structure A and insert the structure element SET EDGE FLAG with a value of on (the default is off). Also insert an EXECUTE STRUCTURE element to create the empty structure B. Close structure A.

```
call popst(structA)
call psedfg(PON)
call pexst(structB)
call pclst
```

Open structure B and create the fill area set attribute elements SET INTERIOR STYLE, SET INTERIOR STYLE INDEX and SET INTERIOR COLOUR INDEX. Create a fill area set element with 3 boundaries and close structure B.

```
call popst(structB)
call psis(PHATCH)
call psisi(hatchindex)
call psici(colourindex)
call pfas(3, boundaries, fasxarr, fasyarr)
call pclst
```

Post structure A to workstation 1, so the edge flag is ON. Post structure B to workstation 2. It will inherit default attributes, including an edge flag of OFF.

```
call ppost(WS1, structA, 0.)
call ppost(WS2, structB, 0.)
```

Figure 2-5 *Structure Network with Both Structures Posted (to Different Workstations)*

Structure A (Posted to Workstation 1)
SET EDGE FLAG *ON*
EXECUTE STRUCTURE *Structure B*

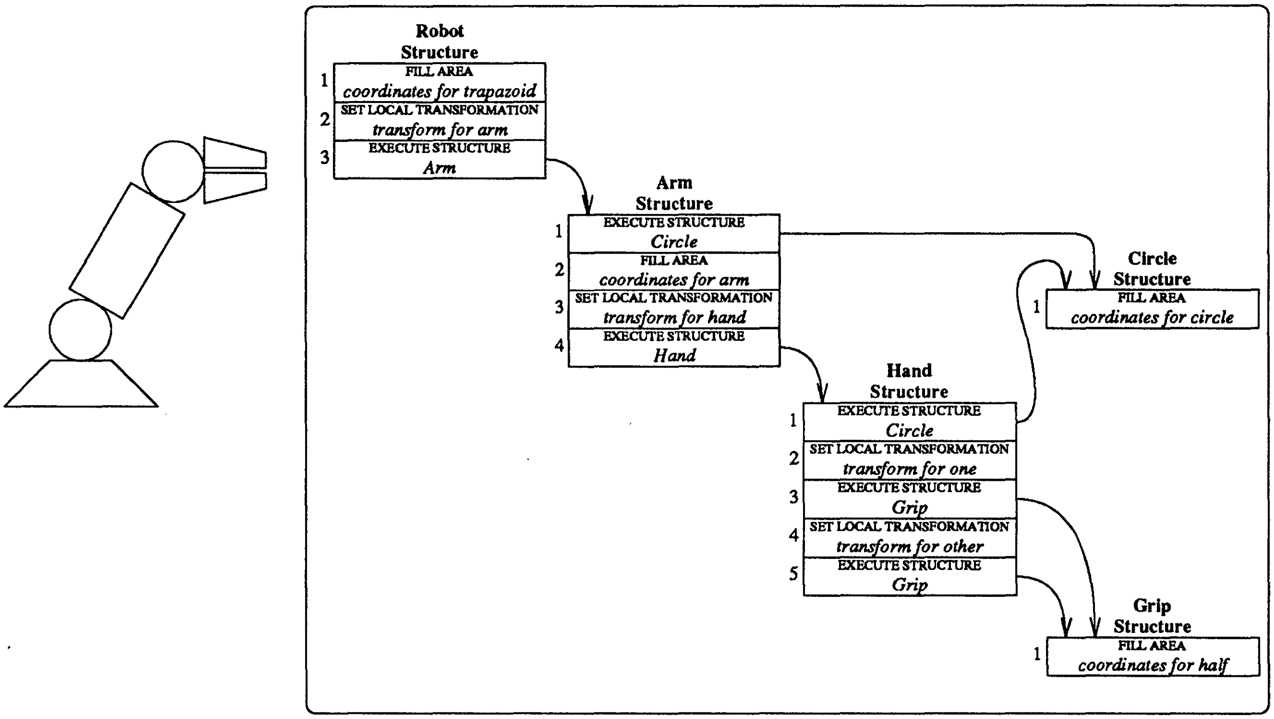
Structure B (Posted to Workstation 2)
SET INTERIOR STYLE *HATCH*
SET INTERIOR STYLE INDEX *-5 (Rectangular Grid)*
SET INTERIOR COLOUR INDEX *6 (CYAN)*
FILL AREA SET *coordinates for diamond*

Note that only workstation 1 will display the edge of the fill area set as structure B is executed from within structure A, the posted structure, and inherits the "edge on". Structure B is posted directly to workstation 2 and executed without inheriting the "edge on".

A Complex Structure Hierarchy

The following robot arm example demonstrates the application of a hierarchical structure network. The placement of the robot hand always starts where the robot arm leaves off. Positioning by use of modeling transformations is discussed in Chapter 3, using this example.

Figure 2-6 *Structure Hierarchy for Robot Arm*


Robot Structure
1 FILL AREA coordinates for trapazoid
2 SET LOCAL TRANSFORMATION transform for arm
3 EXECUTE STRUCTURE Arm
Arm Structure
1 EXECUTE STRUCTURE Circle
2 FILL AREA coordinates for arm
3 SET LOCAL TRANSFORMATION transform for hand
4 EXECUTE STRUCTURE Hand
Circle Structure
1 FILL AREA coordinates for circle
Hand Structure
1 EXECUTE STRUCTURE Circle
2 SET LOCAL TRANSFORMATION transform for one
3 EXECUTE STRUCTURE Grip
4 SET LOCAL TRANSFORMATION transform for other
5 EXECUTE STRUCTURE Grip
Grip Structure
1 FILL AREA coordinates for half

Now we consider the traversal of this structure network, when the Robot structure is posted to a workstation. Each element is executed in succession. When an EXECUTE STRUCTURE element is encountered, the child structure is traversed, and then execution continues at the element following the EXECUTE STRUCTURE element. Because the Grip and Circle structures are referenced more than once, their elements are executed more than once, inheriting different attributes. This causes the two appearances of the circle on the workstation to have different locations.

Table 2-4 *Order of Element Execution During Traversal*

Structure Name	Element Number	Element Type	Element Data
Robot	1	FILL AREA	*coordinates for trapazoid*
Robot	2	SET LOCAL TRANSFORMATION	*transform for arm*
Robot	3	EXECUTE STRUCTURE	*Arm*
Arm	1	EXECUTE STRUCTURE	*Circle*
Circle	1	FILL AREA	*coordinates for circle*
Arm	2	FILL AREA	*coordinates for arm*
Arm	3	SET LOCAL TRANSFORMATION	*transform for hand*
Arm	4	EXECUTE STRUCTURE	*Hand*
Hand	1	STRUCTURE	*Circle*
Circle	1	FILL AREA	*coordinates for circle*
Hand	2	SET LOCAL TRANSFORMATION	*transform for one*
Hand	3	EXECUTE STRUCTURE	*Grip*
Grip	1	FILL AREA	*coordinates for half*
Hand	4	SET LOCAL TRANSFORMATION	*transform for other*
Hand	5	EXECUTE STRUCTURE	*Grip*
Grip	1	FILL AREA	*coordinates for half*

2.10. Manipulating Structures

PHIGS provides many functions for the manipulation of structures.

Change Structure

The CHANGE STRUCTURE IDENTIFIER AND REFERENCES function changes the identifier of the specified structure and changes all references to the original identifier to now reference the new identifier. References may be both EXECUTE STRUCTURE elements and workstation postings (POST STRUCTURES).

Copy Structure

COPY ALL ELEMENTS FROM STRUCTURE copies the elements from a specified *Structure Identifier* and inserts them into the currently open structure after the element pointed to by the element pointer.

The Structure Element Pointer

The SET ELEMENT POINTER function sets the current element pointer, which is the number of the element in the structure currently open at which operations on the structure begin.

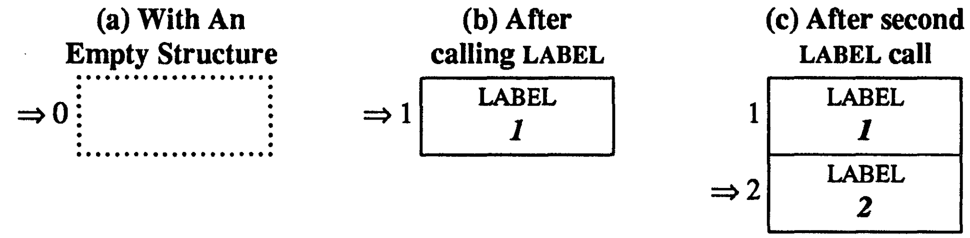
Set Edit Mode

The SET EDIT MODE function sets the edit mode in the *PHIGS* state list to *insert* or *replace*. This value controls how the *PHIGS* functions which create new structure elements add the new element to the currently open structure.

- When the edit mode is *insert* (the default), new structure elements are inserted into the open structure following the element pointed to by the current element pointer. Then the element pointer is advanced to the new element.

As an example, consider opening an (empty) structure and calling the LABEL function twice, with argument 1, and then 2. Figure 2-6 shows that afterward both elements are present and the element pointer points to the last element inserted.

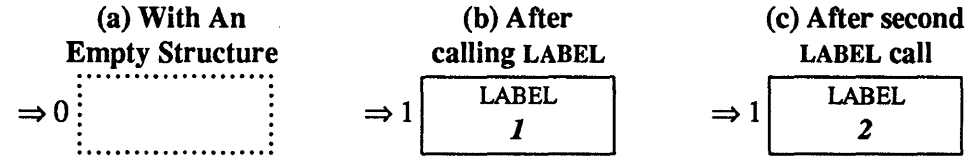
Figure 2-7 *Creating Elements in Insert Edit Mode*



- While the edit mode is *replace*, new structure elements replace the element in the open structure pointed to by the current element pointer.

Considering the same example in Replace edit mode shows that the first element is inserted, but element creation to a nonempty structure replaces the element at the element pointer without moving the pointer. Again, the LABEL function is called twice, with argument 1, and then 2. Figure 2-7 shows that afterward only the last element created is present and the element pointer points to it.

Figure 2-8 *Creating Elements in Replace Edit Mode*



Note: Attempting to insert elements into an open structure while the edit mode is *replace* is difficult. In such a case, each new element will overwrite the existing element at the current element position. In *replace* mode, the element pointer is *not* implicitly incremented after creating an element, as it is in *insert* mode.

Delete Element

DELETE ELEMENT removes the structure element at the element pointer in the open structure and renumbers the remaining elements in the structure.

Delete Structure

The DELETE STRUCTURE function can work one of two ways.

- If the *structure identifier* is not the currently open structure, DELETE STRUCTURE removes the specified structure from the Central Structure Store. The function deletes the structure identifier, the structure contents, and all references to the structure identifier contained in other structures. If the structure is posted to any workstations, it is unposted.
- If the *structure identifier* is the currently open structure, DELETE STRUCTURE replaces the open structure with an empty, unreferenced structure.

Archiving and Retrieving Structures

The ARCHIVE STRUCTURES function copies a list of specified structures from the CSS to the specified open archive file. *SunPHIGS* supports the *PHIGS* archive functionality with a private, *binary* archive format.

Open Archive File

The OPEN ARCHIVE FILE function takes as an argument specifying the UNIX file to use for a given archive identifier. If this file does not exist, it will be created. Once an archive file has been opened, it may be both read from (RETRIEVE STRUCTURES) and written to (ARCHIVE STRUCTURES).

Close Archive File

When CLOSE ARCHIVE FILE is called, any archived structures will be written to the UNIX file. Archives produced by *SunPHIGS* will be upwardly compatible and will be transportable between machine architectures.

Retrieve Structures

RETRIEVE STRUCTURES copies a list of structures from the specified open archive file into the Central Structure Store.

2.11. Workstation State List

The display image is affected by the information contained in the workstation state lists as well as the *CSS*. Workstation attribute bundle tables are an example of this. The application controls when and how changes to the display surface take place by modifying certain values in the workstation state lists.

The Display Update State

SET DISPLAY UPDATE STATE sets the *deferral mode* and *modification mode* entries in the specified workstation's state list. These modes control the degree to which the display must reflect the state of the Centralized Structure Store and the workstation tables. Selection of deferral and modification modes can heavily influences both visual results and performance.

Deferral Modes

The deferral mode controls when the display is updated (i.e., made to match the CSS and workstation tables).

- The default deferral mode is *As Soon As Possible* (ASAP). This requests that *SunPHIGS* keep the screen consistent with the Centralized Structure Store (CSS) and the workstation state list at all times. This typically causes a regeneration for every change to a structure appearing on a workstation and for every change to the workstation's state list.
- "Before the Next Interaction Globally" (BNIG) behaves like ASAP when any input device is active on *any* workstation.
- "Before the Next Interaction Locally" (BNIL) behaves like ASAP when any input device is active on the specified workstation.
- "At Some Time" (ASTI) causes the display to be updated at the discretion of *PHIGS*. With *SunPHIGS*, "At some time" causes the screen to be updated only if the application calls *Close Structure* or if a window system event (e.g., ''damage'' or resize) causes a repaint.
- "When the Application Requests It" (WAIT) will not allow any updates to the display until the application requests it.

REDRAW ALL STRUCTURES and UPDATE WORKSTATION are the two functions which will cause the workstation to be updated when the deferral mode is set to WAIT.

Except for the ASAP deferral mode, there are times in between display updates that the image can be out-of-date, i.e., not reflecting the current state of the CSS and workstation state lists.

Modification Modes

The modification mode controls changes to the display between updates.

- "No Immediate Visual Effects" (NIVE) indicates that the only changes to the display are those which are in accordance with the deferral mode.
- "Update Without Regeneration" (UWOR) allows all the updates that can be realized immediately without regenerating the entire display.
- "Quick Update Method" (UQUM) allows the use of workstation dependent *simulations* of changes that cannot be performed immediately unless the display is regenerated.

(WAIT and NIVE)

"When the Application Requests It" and "No Immediate Visual Effects" (WAIT/NIVE) together prohibit any change to the display in response to changes in the CSS or the workstation state lists.

`Pickjet.c`, in the *tutorial* directory, demonstrates the use of the "At Some Time" and "No Immediate Visual Effects" combination of modes.

Window Damage

With *SunPHIGS*, ASTI causes the screen to be updated only if the application calls CLOSE STRUCTURE or if a window system event (e.g., "damage" or resize) causes a repaint.

SunPHIGS regenerates the image from the CSS when the window is damaged by window system events. However, only the damaged portion of the *SunPHIGS* canvas is repainted. Therefore, this area of the display may not be regenerated with the same information that generated the older, out-of-date, but not "damaged" portion of the display.

pickjet.c

In the `open_phigs` function of `pickjet.c`, *PHIGS* is opened, a workstation is opened and the SET DISPLAY UPDATE MODE function used to set the deferral and modification modes to ASTI/NIVE.

```
popenphigs( (Pchar *)NULL, PDEFAULT_MEM_SIZE);
popenws( ws, (Pconnid)NULL, phigs_ws_type_sun_tool);

psetdisplayupdatest( ws, PASTI, PNIVE); /* turn off auto updates  */
ppoststruct( ws, structure, priority);  /* post before edit is ok */
```

Note that POST STRUCTURE adds a *structure identifier* to a table of posted structures on the specified workstation. Since the structure does not yet exist it will be created as an empty structure and posted to the workstation. However, because the update state is ASTI/NIVE, the display will not be generated until the structure is closed. The `build_css` function opens the empty structure, fills it with the data to display the jet and closes the completed structure.

```
popenstruct( structure);

     ...

pclosestruct();

pupdatews( ws, PPERFORM);          /* update display */
```

The UPDATE WORKSTATION function is used to execute any deferred workstation actions and optionally correct the display if necessary.

There are several programs in the *examples* directory which demonstrate the use of other deferral and modification mode combinations (ASAP/NIVE, WAIT/NIVE and WAIT/UQUM).

2.12. Inquiries

Inquiry functions return values from the various description tables and state lists such as the workstation state list described above.

The inquiry functions provide information concerning:

- the operating state values
- *PHIGS* description table values
- *PHIGS* state list values
- the workstation description tables
- the workstation state list
- the structure state lists
- structure content
- the error state list

Errors detected by inquiry functions are reported through an error indicator parameter. The normal error reporting mechanism is not used for the inquiry functions.

There are over one hundred inquiry functions available in *SunPHIGS.* Several programs in the *examples* directory demonstrate the use of these functions. See the *SunPHIGS Reference Manual* for a complete description of each inquiry function.

3

The Transformation Pipeline

The Transformation Pipeline 43

3.1. Modeling Transformations 43

Local Modeling Transformation 43

Global Modeling Transformation 45

`fourview.c:` 45

3.2. Viewing 46

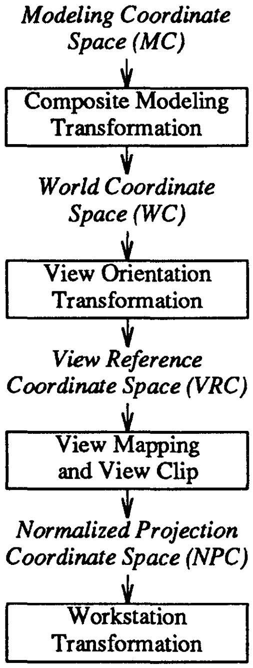
Another Look at `fourview.c` 47

Workstation Transformations 48

3

The Transformation Pipeline

3.1. Modeling Transformations


Modeling Coordinate Space (MC)
Composite Modeling Transformation
World Coordinate Space (WC)
View Orientation Transformation
View Reference Coordinate Space (VRC)
View Mapping and View Clip
Normalized Projection Coordinate Space (NPC)
Workstation Transformation

A *transformation pipeline* is the set of transformations applied to a primitive from its creation to its display. *PHIGS* transformations allow the application to define objects in whatever coordinate space is convenient. The application may then separately describe how to compose the objects into an image, and to place the image onto the display.

Pictured at left is the PHIGS transformation pipeline.

The first step in this pipeline is the *composite modeling transformation*. This is used to map primitives from *Modeling Coordinates* to *World Coordinates*. All *PHIGS* coordinate systems are 3D, right-handed coordinate systems.

In *PHIGS*, the application programmer can compose a graphical picture from separate parts, each of which can be defined within its own modeling coordinate system. *Modeling Coordinate Space* (MC) is used to describe the graphical data. The relative positioning of the separate parts is achieved by having a single *World Coordinate Space* (WC) onto which all the defined Modeling Coordinate systems are mapped using a composite of the modeling transformations. The World Coordinate space can be thought of as a workstation independent *virtual* viewing space and is used to describe objects in relation to each other.

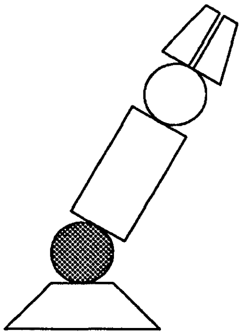
Local Modeling Transformation

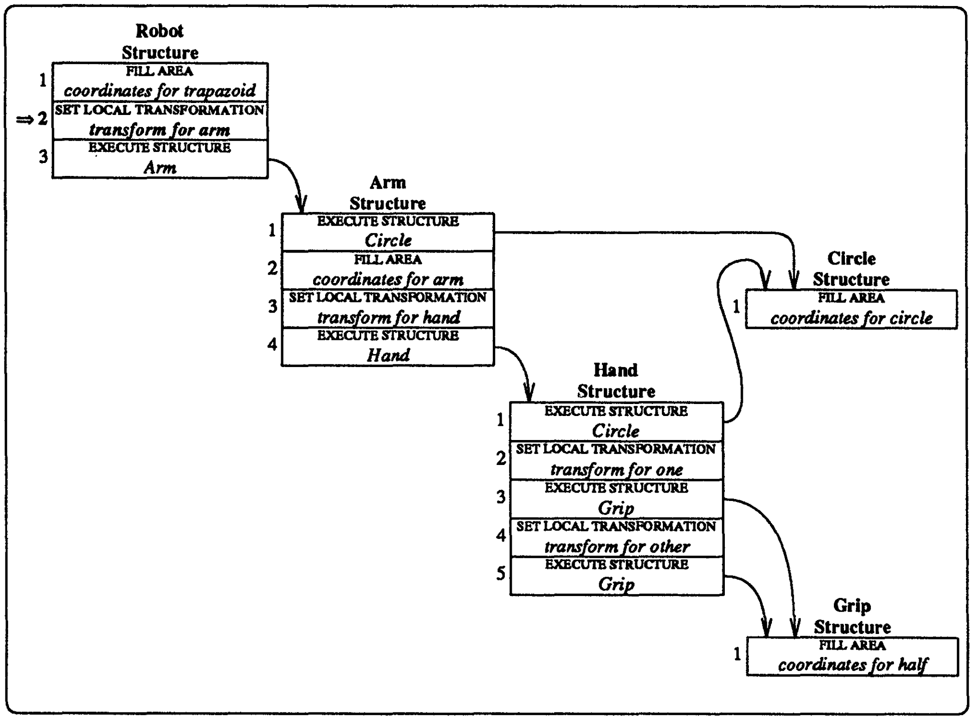
A modeling transformation is stored as a structure element in the CSS. SET LOCAL TRANSFORMATION is used to create such a structure element by specifying a 4 × 4 or 3 × 3 transformation matrix and a composition type. The composite modeling transformation is formed from the hierarchy of component modeling transformations in the current structure path.

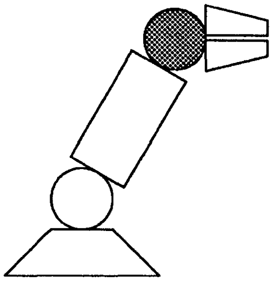
When traversal of a structure begins, the initial *local modeling transformation* and *global modeling transformation* are both the 3-D, 4 × 4 identity matrices. The *composite modeling transformation* within a structure traversal is formed by the matrix multiplication of the *current local modeling transformation* and the *current global modeling transformation*. When an EXECUTE STRUCTURE element is encountered, the current local and global transformations are saved. The child structure inherits the parent's composite modeling transformation as its global modeling transformation. The child inherits an *identity* local modeling transformation which maps every point onto itself unchanged. Therefore, the child's composite modeling transformation (local and global composite) initially equals the parent's. Thereafter, the child can change its own local modeling transformation. After the referenced structure network has been traversed, the parent's transformations are restored.

The robot arm example in Figure 3-1 demonstrates the use of local transformations. The structure network is repeated here, just as in Chapter 2, but with the current element pointer positioned at the Robot structure's element 2, the SET LOCAL TRANSFORMATION with the *transform for arm.* When a transformation with a different rotation replaces this element, subsequent elements – including the Arm structure *and all its descendents* – are rotated, as shown at the left of Figure 3-1.

Figure 3-1 *Structure Network for Robot Arm*




Robot Structure
1 FILL AREA coordinates for trapazoid
⇒2 SET LOCAL TRANSFORMATION transform for arm
3 EXECUTE STRUCTURE Arm
Arm Structure
1 EXECUTE STRUCTURE Circle
2 FILL AREA coordinates for arm
3 SET LOCAL TRANSFORMATION transform for hand
4 EXECUTE STRUCTURE Hand
Circle Structure
1 FILL AREA coordinates for circle
Hand Structure
1 EXECUTE STRUCTURE Circle
2 SET LOCAL TRANSFORMATION transform for one
3 EXECUTE STRUCTURE Grip
4 SET LOCAL TRANSFORMATION transform for other
5 EXECUTE STRUCTURE Grip
Grip Structure
1 FILL AREA coordinates for half



When the Arm's structure's *transformation for hand* (element 3) is replaced with a transformation with a different rotation, the Hand structure's elements and its descendents are rotated, as shown at left.

Global Modeling Transformation

SET GLOBAL TRANSFORMATION can be used to replace the global modeling transformation. During structure traversal, modifications to the global transformation only affect the structure in which they are encountered and in its descendents; the parent structures are not affected.

PHIGS provides a comprehensive set of utility functions to compute transformations for use in SET LOCAL TRANSFORMATION or SET GLOBAL TRANSFORMATION. Included are utilities to produce matrices which independently SCALE, TRANSLATE or ROTATE objects and to composite transformations. The BUILD TRANSFORMATION MATRIX function generates a transformation matrix to perform a transformation specified by a shift vector, rotation angle and scale factors relative to a specified fixed point.

fourview.c:

Fourview.c, in the *examples* directory uses BUILD TRANSFORMATION MATRIX and SET LOCAL TRANSFORMATION MATRIX to rotate an object in a specified view. Below is a code fragment from fourview.c demonstrating the use of these functions. Although four views of the same object are provided, we will look at only one to see how the transformation functions are used. Refer to the complete program for a more complete picture of the modeling transformation process.

First a structure is opened, a view index is set (viewing will be discussed in the next section), and a LABEL element is inserted to *mark* the position within the structure where the transformation element resides. SET LOCAL TRANSFORMATION is called to set the initial *local modeling transformation.* The child structure, OBJECT, is then executed and the structure closed. OBJECT contains the data for the object which is to be rotated.

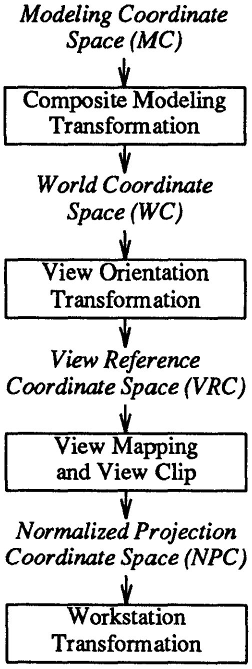
```
#define VIEW_1           1
#define OBJECT_TRANSFORM 1

    static Pmatrix3     identity = { 1.0, 0.0, 0.0, 0.0,
                                     0.0, 1.0, 0.0, 0.0,
                                     0.0, 0.0, 1.0, 0.0,
                                     0.0, 0.0, 0.0, 1.0};

    popenstruct( VIEW_1 );
        psetviewind( VIEW_1 );
        plabel( OBJECT_TRANSFORM );
        psetlocaltran3( identity, PREPLACE );
        pexecutestruct( OBJECT );
    pclosestruct();
```

At a later point in the program, the same structure VIEW_1 (view) is reopened, the element pointer is set to the LABEL element which marks the position of the transformation element, and then the element pointer is advanced by one to point to the element containing the local transformation. BUILD TRANSFORMATION MATRIX is used to generate a new transformation matrix and SET LOCAL TRANSFORMATION MATRIX called again to replace the value of the current local modeling transformation. This calculation and installation of a replacement local modeling transformation is repeated in a loop to rotate the object. Finally, the structure is again closed.

```
popenstruct( view );
    psetelemptr(0);
    psetelemptrlabel( OBJECT_TRANSFORM );
    poffsetelemptr( 1 );
    for ( i = 0; i < 100; i++ ) {
        pbuildtran3( &pt, &pt,
            (Pfloat)i/30., (Pfloat)i/40., (Pfloat)i/50.,
            &scale, &err, transform);
        psetlocaltran3( transform, PREPLACE );
    }
    for ( i = 99; i >= 0; i-- ) {
        pbuildtran3( &pt, &pt,
            (Pfloat)i/30., (Pfloat)i/40., (Pfloat)i/50.,
            &scale, &err, transform);
        psetlocaltran3( transform, PREPLACE );
    }
pclosestruct();
```

3.2. Viewing

Viewing is the mechanism whereby coordinates in the *World Coordinate* (WC) system are transformed to *Normalized Projection Coordinates* (NPC). SET VIEW INDEX creates a structure element which contains a *view index* attribute. The view index is used to select a view representation entry in a workstation view table.


Modeling Coordinate Space (MC)
Composite Modeling Transformation
World Coordinate Space (WC)
View Orientation Transformation
View Reference Coordinate Space (VRC)
View Mapping and View Clip
Normalized Projection Coordinate Space (NPC)
Workstation Transformation

The function SET VIEW REPRESENTATION is used to define a view representation entry on a workstation. A view representation controls the viewing stage of the transformation pipeline which transforms the World Coordinates into Normalized Projection Coordinates and optionally clips to the limits of the NPC space. This view table entry specifies a view orientation matrix, a view mapping matrix, view clipping limits in NPC space and a clipping indicator.

The purpose of the view orientation matrix is to transform World Coordinates relative to the view reference coordinate system. The axes of the VRC system are U, V, N.

The purpose of the view mapping matrix is to transform points in VRC to points in the Normalized Projection Coordinate system. This transformation can be either a parallel or perspective transformation.

The purpose of the view clipping limits is to specify the region of NPC space in which visible data may appear, provided clipping is turned on by the clipping indicator.

EVALUATE VIEW ORIENTATION MATRIX and EVALUATE VIEW MAPPING MATRIX are used to calculate the matrices used by the SET VIEW REPRESENTATION function. EVALUATE VIEW ORIENTATION MATRIX is used to calculate a transformation matrix that tranforms World Coordinates into *View Reference Coordinates* (VRC). The input parameters to this function establish the view reference point and the view up vector. The view orientation matrix is output from this function. EVALUATE VIEW MAPPING MATRIX is used to calculate the mapping matrix for SET VIEW REPRESENTATION. Both EVALUATE VIEW ORIENTATION MATRIX and EVALUATE VIEW MAPPING MATRIX are described in detail in Appendix C.

A predefined number of views can be stored in each workstation. They are numbered consecutively starting with zero. In each view table `view 0` is a special identity view entry that cannot be modified. The predefined view table entries are initialized to values in the workstation description table (see *PHIGS Workstation Description Table* (7P)). The other view table entries are initialized to the same values as view table entry 0.

Another Look at `fourview.c`

In demonstrating the functions described above, we will again use the example program `fourview.c` from the *examples* directory. `Fourview.c` provides four different views of the same object. We will use only one view to see how the viewing functions interact.

After the `OBJECT` data structure is built, a structure for each view is created and a SET VIEW INDEX element inserted for that view.

```
#define VIEW_1                  1

    popenstruct( VIEW_1 );
        psetviewind( VIEW_1 );
        plabel( OBJECT_TRANSFORM );
        psetlocaltran3( identity, PREPLACE );
        pexecutestruct( OBJECT );
    pclosestruct();
```

Next, the view representation is set up by defining the clipping flags and the view mapping matrix EVALUATE VIEW ORIENTATION MATRIX is used to generate the orientation matrix. EVALUATE VIEW MAPPING MATRIX is used to generate the mapping matrix.
Once the representation has been set up, SET VIEW REPRESENTATION is called to define the view representation entry in the workstation view table. Refer to the `man` page entry for each of these functions for a complete description of all of the parameters used below.

```
/* All views use the same reference point. */
vrp.x = 0.0; vrp.y = 0.0; vrp.z = 0.0;

map.proj = PPARALLEL;
rep.clip_xy = rep.clip_back = rep.clip_front = PCLIP;
map.window.xmin = -2.0; map.window.xmax = 2.0;
map.window.ymin = -2.0; map.window.ymax = 2.0;
map.back_plane = -2.0;
map.front_plane = 2.0;
map.view_plane =  1.8;
map.prp.x = (map.window.xmin + map.window.xmax) / 2.0;
map.prp.y = (map.window.ymin + map.window.ymax) / 2.0;
map.prp.z = 10.0;
map.viewport.zmin = 0.0; map.viewport.zmax = 1.0;

/* View 1 -- top view */
vup.x =  0.0; vup.y =  0.0; vup.z = -1.0;
vpn.x =  0.0; vpn.y =  1.0; vpn.z =  0.0;
pevalvieworientationmatrix3( &vrp, &vpn, &vup, &err,
rep.orientation_matrix);
map.viewport.xmin = 0.05; map.viewport.xmax = 0.45;
map.viewport.ymin = 0.55; map.viewport.ymax = 0.95;
pevalviewmappingmatrix3( &map, &err, rep.mapping_matrix);
rep.clip_limit = map.viewport;
psetviewrep3( WSID, VIEW_1, &rep );
```

For a more extensive took at the *PHIGS* viewing model see Appendix C.

Workstation Transformations

The *Normalized Projection Coordinate Space* can be regarded as a workstation dependent *abstract* image composition space. A part of the workstation's NPC space can be selected to be displayed somewhere on the workstation's physical display space. A workstation transformation is a mapping from NPC space to *Device Coordinate Space* (DC) for a particular workstation.

The workstation transformation is a uniform mapping from NPC onto DC for *x* and *y* and thus performs translation and *equal* scaling with a positive scale factor for these two axes. The workstation transformation allows different aspects of the composed picture to be viewed on different workstations.

A workstation transformation is specified by defining the limits of a volume within NPC space within the range 0 to 1 which is to be mapped onto a specified volume in DC space. Workstation transformations can be specified using SET WORKSTATION WINDOW and SET WORKSTATION VIEWPORT.

SET WORKSTATION WINDOW defines the area in NPC space to be displayed on the specified workstation. The workstation window is a rectangular box in NPC space, which is mapped to the *workstation viewport*, defined in *Device Coordinate Space.* The *workstation window* defines *what* within NPC space will be displayed.

SET WORKSTATION VIEWPORT defines the area in DC space onto which the *workstation window* will be mapped. The *workstation viewport* defines *where* the image will be displayed in DC space.

Together, the *workstation window* and the *workstation viewport* define the *workstation transformation* that converts the image from *Normalized Projection Coordinate Space* to the device coordinates of the workstation's physical display surface.

4

Input

Input **53**

4.1. Six Input Classes 53

Sample Uses 53

Locator Devices 54

Stroke Devices 54

Valuator Devices 55

Choice Devices 55

String Devices 55

Pick Devices 55

4.2. SunPHIGS Input Devices 57

4.3. Prompt/Echo Types (PETs) 58

Input Data Records 59

FORTRAN Packed Data Record 59

4.4. Three Operating Modes 60

Request Mode 61

Event Mode 61

Sample Mode 62

4

Input

PHIGS uses the *concept* of a *logical input device* to obtain graphical input. This logical interface provides portablility of applications shielding the application programmer from the physical devices. In *SunPHIGS* the operator enters input using the mouse and keyboard. The *PHIGS* standard uses the term *operator* to refer to the person handling the input device, as distinguished from the *PHIGS* programmer who writes the application.

4.1. Six Input Classes

There are six *classes* of logical input devices distinguished by the *measure*, or value, they produce. For example, a locator device reports the *x*, *y* and *z* position at which the mouse button is pressed. The table below shows the six *device* classes, and the value each produces.

Device Class	Measure Produced
Locator	*x*, *y*, *z* position in World Coordinate (WC) space
Stroke	Sequence of *x*, *y*, *z* positions in WC space
Valuator	Floating-point number
Choice	Integer choice between 1 and *N*
String	String of characters
Pick	A primitive selected from a structure network

Sample Uses

Locator — A locator device may be used by the application to place a graphical object at an *x*-*y*-*z* position interactively selected by the operator at runtime.

Stroke — A stroke device allows the operator to input a sequence of points which the application may use to form a spline.

Valuator — A valuator device allows the operator to input a floating-point number which the application may use in a transformation, as a scale factor or a rotation angle.

Choice — A choice device may be a simple implementation of a menu, selecting the next action to be performed from a set offered by the application.

String — A string device allows an operator to enter text such as filenames or other information requested by the application.

Pick A pick device may select a primitive, or group of primitives, to be edited, deleted, copied, etc.

A logical input device is associated with a particular open workstation. It is therefore uniquely identified by the workstation identifier, device class, and a device number.

Locator Devices

A locator device's measure is reported in World Coordinates. In this description we will assume a *z* value of 0. By default, the WC space ranges from 0 to 1 in the *x*, *y* and *z* directions. If the locator device is triggered by the operator pressing a mouse button while the mouse is in the lower left corner of the workstation, a value near (0,0) would be returned as the locator measure. If the locator device is triggered while in the upper right corner of the workstation, a value near (1,1) would be returned as the locator measure.

Stroke Devices

A stroke device's measure is also reported in World Coordinates. A stroke logical device behaves in the same fashion as the locator device only returning a sequence of *x-y-z* values as the stroke measure.

A locator or a stroke device's measure also includes the view index of the view that was used to map the operator-chosen position(s), in Device Coordinate (DC) space, back into WC's. By default, the only view used for input is view index 0. To override the default view index SET VIEW TRANSFORMATION INPUT PRIORITY is used.

The following code fragments are from `loc.c` in the *tutorial* directory and demonstrate the initialization and setting of a locator device operating mode and echoing state. For a more complex demonstration of using the locator device, see `fourview.c` in the *examples* directory. In the `initialize_input` function of `loc.c`, an initialized locator device is enabled using SET LOCATOR MODE which sets the operating mode to Request and the echo switch to *echo*. for the locator device. In the `request_locator` function, REQUEST LOCATOR is used to request the current measure of the locator device.

```
initialize_input()

        static Ploc       init_location = { 0, {0., 0.} };
        static Plocrec    record;     /* not used, but must be present */
        Plimit            area;
        Pint              pet = 1;    /* data record not used for PET 1 */

        area.xmin = area.ymin = 0.0;
        area.xmax = area.ymax = 1.0;

        pinitloc( ws, devid, &init_location, pet, &area, &record);
        psetlocmode( ws, devid, PREQUEST, PES_ECHO);

request_locator()

        Pqloc             locator;

        preqloc( ws, devid, &locator);
```

Valuator Devices

A valuator device returns the value selected as its measure and a status indicating whether a selection was made or not. `figstoolval.f`, previously discussed in the Workstation Configuration section, demonstrates the initialization and mode setting functions for a valuator device. For a more complex demonstration of using the valuator device, see `rspheres.c` in the *examples* directory.

Choice Devices

A choice device returns the choice selected as its measure and a status indicating whether a selection was made or not. For demonstration of initialization and mode setting for a Choice device, see `rspheres.c` in the *examples* directory.

String Devices

A string device returns a string of characters as its measure. *SunPHIGS* uses the string logical input device to return operators input from the message text subwindow.

Pick Devices

Because a *PHIGS posted* structure network is hierarchical, a single output primitive element may appear on a workstation multiple times with different attribute values and at varying locations. Therefore, a pick measure actually consists of:

> a status indicating whether any primitive was selected;
>
> a *pick path*, the list of triples (structure identifier, pick identifier, and element number) which uniquely describe the path through the structure network's EXECUTE STRUCTURE elements from the posted structure to the primitive element selected; and
>
> the *pick depth*, or number of levels in the pick path.

By default, no primitives are initially selectable by pick input devices. The application must use both the *name set* elements (ADD NAMES TO SET) and *pick filter* (SET PICK FILTER) to allow primitives of interest to be eligible for picking.

A *pick path* and corresponding structure network follow.

Figure 4-1 *A Plane Structure Hierarchy from which Engine may be Picked*

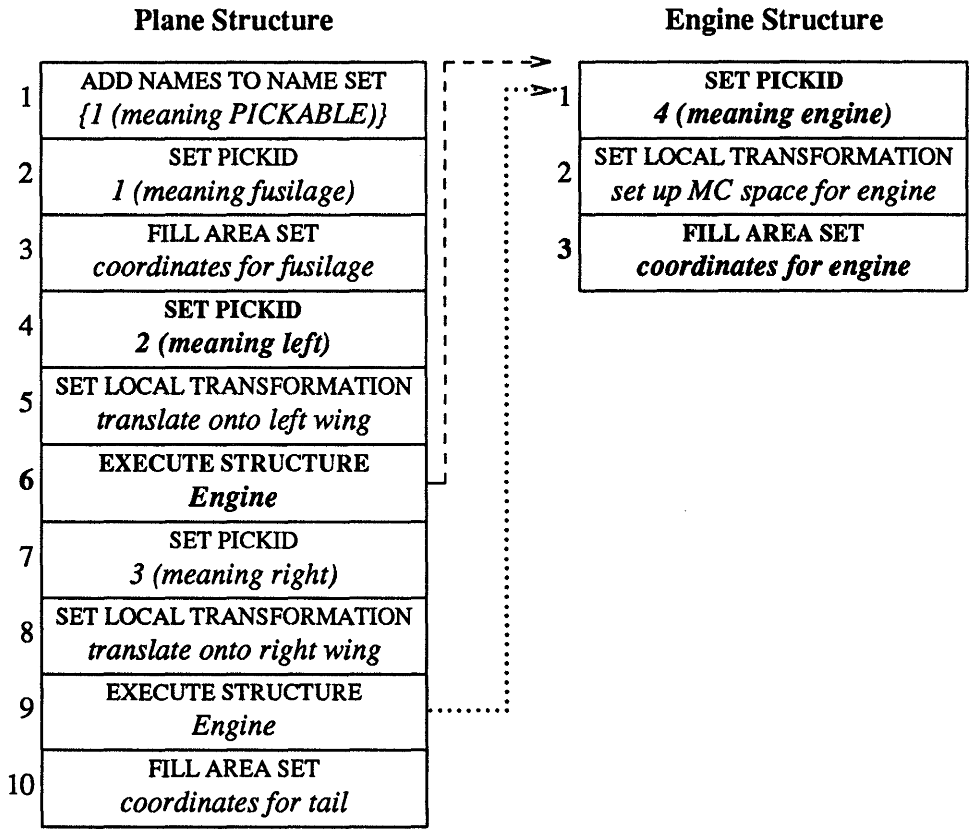

Plane Structure
Engine Structure
1 ADD NAMES TO NAME SET {1 (meaning PICKABLE)}
2 SET PICKID 1 (meaning fusilage)
3 FILL AREA SET coordinates for fusilage
4 SET PICKID 2 (meaning left)
5 SET LOCAL TRANSFORMATION translate onto left wing
6 EXECUTE STRUCTURE Engine
7 SET PICKID 3 (meaning right)
8 SET LOCAL TRANSFORMATION translate onto right wing
9 EXECUTE STRUCTURE Engine
10 FILL AREA SET coordinates for tail
1 SET PICKID 4 (meaning engine)
2 SET LOCAL TRANSFORMATION set up MC space for engine
3 FILL AREA SET coordinates for engine

Figure 4-2 *Pick Path from Selecting Left Engine*

Structure Identifier	**Pick Identifier**	**Element Number**
Plane Structure	2 (meaning left)	6 (EXECUTE STRUCTURE *Engine*)
Engine Structure	4 (meaning engine)	3 (FILL AREA SET *for engine*)
Pick Path Depth is 2		

The following code fragments are from `pickjet.c` in the *tutorial* directory and demonstrate a programming example of setting up primitives to be "pickable". The functions used are ADD NAMES TO SET and SET PICK FILTER. For a more complex picking example, see `pickit.c` in the *examples* directory.

In the `build_css` function, a number of "names" and a list of "names" are initialized. Each name in the *name set* list is a small positive integer. In this instance, only one "name" is used. During traversal of the posted structure network, the ADD NAMES TO SET attribute is bound to the output primitives which follow.

In the `initialize_input` function the list of ''names'' in the *pick inclusion filter* is then initialized with the same data and SET PICK FILTER called to set the PICK input device's pick filter to complete the actions required to make the primitives *pickable*.

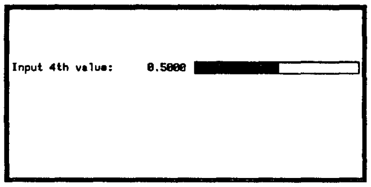
```
#define PICKABLE        1

build_css()

        Pintlst         names_to_add;
        static Pint     name[1] = { PICKABLE };
        Pint            part = 1;

        names_to_add.number = 1;
        names_to_add.integers = name;
        paddnameset( &names_to_add);

        psetintstyle( PSOLID);

        psetintcolourind( BLUE);   /* jet body is blue */
        psetpickid( part);         /* pick this as part #1 */
        pfillarea( 3, jetbody);

initialize_input()

        Pintlst         infilt, exfilt;
        infilt.number = 1;
        infilt.integers = name;
        exfilt.number = 0;

        psetpickfilter( ws, devid, &infilt, &exfilt);
```

4.2. SunPHIGS Input Devices

SunPHIGS supports all six input device classes on *Sun Tool* and *Sun Canvas* workstations. The following table gives the physical devices provided by *PHIGS*. For complete information, see INITIALIZE <*device class*>, in the *SunPHIGS Reference Manual*.

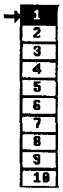
Table 4-1 *PHIGS Physical Devices*

Class	Device Numbers	*SunPHIGS* Implementation Method
Locator	1	Left mouse button and cursor
	2	Middle mouse button and cursor
	3	Right mouse button and cursor
	4	Mouse movement (no button down) and cursor
	5	Mouse dragging (any button down) and cursor
Stroke	1	Left mouse button and cursor
	2	Middle mouse button and cursor
	3	Right mouse button and cursor
Pick	1	Left mouse button and cursor
	2	Middle mouse button and cursor
	3	Right mouse button and cursor
Valuator	1—10	Left mouse button and a *SunView* slider
Choice	1	*SunView* pop-up menu from Left mouse button
	2	*SunView* pop-up menu from Middle mouse button
	3	*SunView* pop-up menu from Right mouse button
	10	Any of mouse buttons
	11	Any ASCII key from keyboard
	12	Any Left function key from keyboard
	13	Any Top function key from keyboard
	14	Any Right function key from keyboard
String	1	Text subwindow input, RETURN terminated



Input 4th value: 0.5000

Valuator Window with Slider



Window with Choice Pop-up

4.3. Prompt/Echo Types (PETs)

The logical input device supports certain prompt/echo types, which determine the details of operator interaction for the input transaction. A *prompt* shows that the device is available to receive input. *SunPHIGS* prompts include changing the mouse cursor displayed or making a *SunView* panel item visible. When no input device is active, a NULL (⃠) cursor is displayed. An *echo* is the "feedback" notification to the operator of the present value of the input device. An example locator prompt/echo type (PET) is a rubber-band line from an initial point to the point the operator is selecting. The default PICK PET for *SunPHIGS* is to blink the selected primitive by changing its color.

Certain PETs are defined by the *PHIGS* standard, but they are not necessarily supported by any particular implementation or input device. The PETs a *PHIGS* implementation supports may vary, even among devices in a single class. For a description of the PETs supported by each available input device, see INITIALIZE <*device class*>, in the *SunPHIGS Reference Manual*.

Input Data Records

Certain prompt echo types (PETs) require additional parameters (such as appearance attributes) to completely specify the input interaction with the operator. These parameters are grouped into a PET-specific input *data record*. For PETs defined by the *PHIGS* standard, some of these parameters are specified as well, but in general this additional information is implementation and device-dependent.

C Programs

In C, the data record is a union. The application initializes the member that matches the PET to be used, and passes the data record to INITIALIZE <*device class*>. The example program, `pickjet.c` uses implementation-dependent PICK PET -1, so it initializes the member `upickpet1_datarec`, which is of type `Pupickpet1_datarec`. The 'u' means the PET is an unregistered (implementation-defined) PET.

FORTRAN Programs

In FORTRAN, the information for the data record is loaded into integer, real, and character arrays, and is *packed* into elements of a CHARACTER*80 array, using PACK DATA RECORD. The data record array is then provided to INITIALIZE <*device class*>.

FORTRAN Packed Data Record

In `figstoolval.f` from the *tutorial* directory, a dummy data record was passed as a prompt/echo type of 1 was used and the data record parameters for this PET are passed as function arguments. There is no data record required, therefore, PACK DATA RECORD was not called. If it is desired to specify a label for the valuator device, a PET of -1 is used and the label, format string and slider length are specified in a data record. The following is an example of packing the data record to be passed to INITIALIZE VALUATOR with the information needed for a PET of -1. The parameters to PACK DATA RECORD are:

il the number of integers (one)

ia an array of integer(s) (contains the length of the slider in pixels)

rl the number of real values (zero)

ra an array of real values (no real values are required, therefore, this is a dummy parameter)

sl the number of character strings (two)

lstr an array containing the lengths of the strings (the label and format strings)

str an array containing the strings (the length and format strings).

mldr the number of 80-character array elements the data record has been dimensioned to

errind an output parameter containing the error indicator (zero if no error)

ldr an output parameter containing the *actual* number of 80-character array elements the data record has used

datrec the packed data record to be passed to the inialization function

```
integer ia(1), lstr(2), errind, ldr
real ra(1)
character *15 str(2)
character *80 datrec(10)

ia(1)   = 200
lstr(1) = 12
lstr(2) = 5
str(1)  = "Slider value"
str(2)  = "%4.1f"

call pprec(1, ia, 0, ra, 2, lstr, str, 10, errind, ldr, datrec)
```

Full descriptions of all supported PETs and their data records may be found on the INITIALIZE <*device class*> pages of the *SunPHIGS Reference Manual*.

The following code fragment from the *tutorial* directory `pickjet.c` demonstrates setting up a data record in C. The function used to initialize the pick input device with the data record is INITIALIZE PICK.

```
static Ppickpath init_pick_path = { 0, (Ppickpathel *)NULL };
static Pint         name[1] = { PICKABLE };
Ppickrec            record;
Plimit              area;
Pint                pet = -1;

area.xmin = area.ymin = 0.0;
area.xmax = area.ymax = 1.0;

record.upickpet1_datarec.highlight_colour = BLACK;
record.upickpet1_datarec.highlight_count = 3;
record.upickpet1_datarec.highlight_duration = 0.1;
/* The pick aperture is defined in NPC, which ranges from 0 to 1.
 * The default size of the window is 600 pixels.
 * To set the aperture to a square 3 pixels on a side,
 * centered around the cursor's "hot spot", we calculate:
 * 3/600. is 0.005.
 */
record.upickpet1_datarec.aperture_size.x = 0.005;
record.upickpet1_datarec.aperture_size.y = 0.005;
record.upickpet1_datarec.aperture_size.z = 0.005;

pinitpick( ws, devid, PP_NOPICK, &init_pick_path, pet, &area,
   &record, PTOP_FIRST);
```

4.4. Three Operating Modes

A *PHIGS* program may select the flow of control between the application program and the logical input device. The application program may use the logical input device in one of the three *operating modes*: *Request, Event* and *Sample*. The default is Request mode. Each input device can operate in any of the three operating modes in a *Sun Tool* workstation, and in Event and Sample modes in the *Sun Canvas* workstation.

Request Mode

In Request mode, when the application requests input from a single logical input device, the application *blocks* (i.e., waits) until the request is satisfied. *PHIGS* prompts for the input and echos the current measure. When the operator triggers the device (e.g., by pressing a mouse button), *PHIGS* returns the measure to the application.

The only other way the application can regain control without the operator triggering the input device is for the operator to use the *break action* to indicate a refusal to provide the input. The *SunPHIGS* break action for any device in Request mode is CTRL-D. The application can determine the break has occurred from status returned from the REQUEST <*device class*> function.

Input devices associated with a *Sun Canvas* workstation do not support Request mode. Demanding input from a particular device before continuing is foreign to *SunView*, which offers the operator multiple actions supported by a handler, called the *notifier*, which reacts to the operation selected.

The following code fragments from the tutorial program `loc.c` demonstrate the use of the locator device in Request mode, using REQUEST LOCATOR.

```
Pint         ws           = 1;
Pint         devid        = 1;  /*locator 1 is LEFT mouse button*/
Pqloc        locator;

preqloc( ws, devid, &locator);
if (locator.status == PSTAT_OK)
        printf( "Locator: x pos=%fy pos=%f0,
                locator.loc.position.x, locator.loc.position.y);
else if (locator.status == PSTAT_NONE)
        printf( "Operator did CTRL-D break action.0);
return( locator.status);
```

Event Mode

After one or more input devices are placed in Event mode using SET <*device class*> MODE, the application still has the flow of control. Simultaneously with execution of the application program, *PHIGS* prompts for the input, and echos the current measure. Whenever the operator triggers the device (e.g., by pressing a mouse button) an input *event* is appended to a central *input queue*, a first-in first-out list of input events. The event contains the class and number of the logical input device and the measure at that instant.

At the application's convenience, AWAIT EVENT is used to determine if the queue has any input events, to move the event at the head of the input queue to the *current event report*, and to obtain that event's device class and device number. If there are no events in the input queue, AWAIT EVENT will optionally wait for an event, until a timeout period has expired. The GET <*device class*> function then returns the measure from the current event report. For example programs which demonstrate using logical input devices in Event mode with both the *Sun Tool* and *Sun Canvas* workstations, see the `fourview` examples and `pickit.c` in the *examples* directory.

Sample Mode

In Sample mode, the application (*not* the operator) selects the instants at which the input device's measure should be obtained. This is performed by the application polling the device for its current input measure. When the application places one or more input devices into sample mode using SET *<device class>* MODE, the application still has the flow of control. Simultaneously with execution of the application program, *PHIGS* prompts for the input, and optionally echos the current measure. The operator changes the current measure (e.g., by moving the mouse) and *PHIGS* updates the echo to reflect the measure. At the program's convenience, it obtains the device's current measure using SAMPLE *<device class>*.

A typical use of Sample mode is to implement an application-specific prompt/echo type. The application would first disable *PHIGS* echoing using SET *<device class>* MODE with the echo switch turned off. Then the application would enter a loop, sampling the current measure and using it to modify the display, perhaps scaling or rotating a multi-primitive object. When the application leaves the loop, SET *<device class>* MODE is again called to return the input devices to Request mode, in which the device is inactive until a request is initiated by the program.

A

Examples

Examples 65

A.1. C Examples 65

axes.c 65

canvasattrs.c 67

canvasid.c 69

cpolygons.c 70

ex1.c 73

ex2.c 78

fourview.c 83

fourview_cvs.c 92

non_square.c 101

pickit.c 104

rspheres.c 113

spheres.c 124

txattrs.c 128

A.2. FORTRAN Examples 140

fbundles.f 140

ffillset.f 142

flines.f 145

fmarkers.f 148

fpolygons.f 150

ftext.f 152

ftextall.f 154

A

Examples

Appendix A contains example programs from the /usr/lib/phigs1.1/examples directory.

A.1. C Examples

The following example programs use the *SunPHIGS* C binding.

axes.c

Called by rspheres.c and used to insert axes into a structure.

```
#ifndef lint
static char sccsid[] = "@(#)axes.c 2.1 88/06/02 Copyr 1988 Sun Micro";
#endif

/*
 * Copyright (c) 1988 by Sun Microsystems, Inc.
 */

/* Insert axes in the currently open structure. */

#include <phigs/phigs.h>

void
axes( origin, length, color )
    Ppoint3 *origin;
    Ppoint3 *length;
    Pint    color[3];
{
    Ppoint3 axis[2];

    axis[0] = axis[1] = *origin;

    axis[1].x = length->x;
    psetlinecolourind( color[0]);
    ppolyline3( 2, axis);

    axis[1].x = origin->x;
    axis[1].y = length->y;
    psetlinecolourind( color[1]);
    ppolyline3( 2, axis);

    axis[1].y = origin->y;
    axis[1].z = length->z;
```

```
        psetlinecolourind( color[2]);
        ppolyline3( 2, axis);
}
```

canvasattrs.c

Demonstrates *Sun Canvas* workstation configurations.

```
#ifndef lint
static  char sccsid[] = "@(#)canvasattrs.c 2.1 88/06/02 SMI";
#endif

/*  canvasattrs.c - This program demonstrates one way to create a new
        workstation of type phigs_ws_type_sun_canvas and modify the
        values in the workstation description table.
        Canvas and text subwindows are generated in SunView and their
        handles are passed to SunPHIGS.

Note:   It is necessry to include the declarations in phigs.h in order to
        open a workstation or access SunPHIGS constants.
*/

#include <phigs/phigs.h>
#include <suntool/sunview.h>
#include <suntool/canvas.h>
#include <suntool/textsw.h>

#define WS 1
#define ST 1

main()
{
        static Frame  frame;
        static Canvas  canvas;
        static Textsw  textsw;

        static Pwstype canvaswst;
        static Ppoint textpts = {0.1, 0.5};
        static Pchar buf[80];

/*  Obtain SunView canvas and textsw handles.
*/
        frame = window_create(NULL, FRAME,
            FRAME_LABEL,        "SunPHIGS Canvas Workstation",
            0);

        canvas = window_create(frame, CANVAS,
            WIN_PERCENT_HEIGHT, 90,
            0);

        textsw = (Textsw) window_create(frame, TEXTSW,
            WIN_PERCENT_HEIGHT, 10,
            TEXTSW_IGNORE_LIMIT,TEXTSW_INFINITY,
            0);

/*  Open PHIGS and create a sun_canvas workstation with a MESSAGE window.
    The SunView textsw handle is passed as the value for the PHIGS_TEXTSW
    workstation attribute.
*/
        popenphigs((Pchar*)NULL, PDEFAULT_MEM_SIZE);

        canvaswst = phigs_ws_type_create(phigs_ws_type_sun_canvas,
            PHIGS_TEXTSW, textsw,
            0);

/*  Open the sun_canvas workstation and a structure to contain text elements.
    The SunView canvas handle is passed as the connection identifier for the
    open workstation function.  The PHIGS workstation handle, canvaswst is
    passed as the workstation type.
*/
        popenws(WS, (Pconnid)canvas, canvaswst);
        popenstruct(ST);

        psetcharheight(.02);
        ptext(&textpts, "This is a phigswstcanvas workstation.");

/*  Post the structure to the workstation and display a message in the textsw.
*/
        ppoststruct(WS, ST, 0.);
        sprintf(buf, "This is the MESSAGE window.");
        pmessage(WS, buf);

/*  Call SunView's notifier.
*/
        window_main_loop(frame);

/*  Close the structure, close the workstation and close PHIGS.
*/
        pclosestruct();
        pclosews(WS);
        pclosephigs();
}
```

canvasid.c

Called by fmarkers.f as a FORTRAN/*SunView* interface.

```
#ifndef lint
static  char sccsid[] = "@(#)canvasid.c 2.2 88/07/08 SMI";
#endif

/* canvasid.c - a C program to setup a SunView Canvas subwindow for SunPHIGS */

#include <suntool/sunview.h>
#include <suntool/canvas.h>
#include <errno.h>

static  Frame   frame;

int
canvasid_()
{
        Canvas  canvas;
        frame = window_create(NULL, FRAME,
                FRAME_LABEL,       "SunPHIGS Canvas Workstation",
                0);
        canvas = window_create(frame, CANVAS,
                WIN_WIDTH,  600,
                WIN_HEIGHT, 600,
                0);
        window_fit(frame);
        return((int)canvas);
}

int
display_()
{
        window_main_loop(frame);
}
```

cpolygons.c

Demonstrates *SunPHIGS* hatch styles.

```
#ifndef lint
static  char sccsid[] = "@(#)cpolygons.c 2.1 88/06/02 SMI";
#endif

/*  cpolygons.c - This program draws fill areas in all available hatch styles.

Note:  It is necessry to include the declarations in phigs.h in order to
       open a workstation or access SunPHIGS constants.  The #defines for
       workstation 1 (WS1) and the structures ST1 and ST2 are for clarity.
*/
#include <phigs/phigs.h>
#define WS1 1
#define ST1 1
#define ST2 2
#define ROWS 4          /* Number of rows of hatch-filled fill areas. */
#define POINTS 4        /* Number of points in each fill area primitive. */
#define HATCHES 6       /* Number of SunPHIGS "basic" hatch styles. */

main()
{
        int xaxis, yaxis, row, width, transparency, index;
        Pwstype WStype;
        static Pchar buf[80];
        static Ppoint fapoints[POINTS];
        static Ppoint orig_xy_pts[POINTS] = { {.07,.775}, {.195,.775},
                                              {.195, .975}, {.07,.975} };
        static Pint hatches[HATCHES] =
                            { PHATCH_HORIZ, PHATCH_VERT, PHATCH_DIAG_45,
                              PHATCH_DIAG_135, PHATCH_GRID_R, PHATCH_GRID_D };

/*  Open PHIGS and create a sun_tool workstation.
*/
        popenphigs((Pchar*)NULL, PDEFAULT_MEM_SIZE);

        WStype = phigs_ws_type_create(phigs_ws_type_sun_tool,
        PHIGS_TOOL_LABEL, "SunPHIGS Tool Workstation",
        0);

/*  Open workstation 1, then fill structure 2 with background elements.
*/
        popenws(WS1, (Pconnid)NULL, WStype);

        fill_background();

/*  Open structure 1 and fill with elements to be displayed during traversal.
*/
        popenstruct(ST1);

/*  Create a structure element containing structure 2's identifier,
    to be invoked during traversal of structure 1.
*/
        pexecutestruct(ST2);

/*  Add a SET INTERIOR STYLE attribute element to structure 1.
```

```
*/
    psetintstyle(PHATCH);

/* Initialize a copy of the "original" point data, which we'll change.
*/
    for (xaxis = 0; xaxis < POINTS; xaxis++) {
        fapoints[xaxis].x = orig_xy_pts[xaxis].x;
    }
    for (yaxis = 0; yaxis < ROWS; yaxis++) {
        fapoints[yaxis].y = orig_xy_pts[yaxis].y;
    }

    for (row = 1; row <= ROWS; row++) {
        switch(row) {
          case 1: width = 0; transparency = 0;
                  break;
          case 2: width = PHIGS_HATCH_DBL_WIDTH;
                  transparency = 0;
                  break;
          case 3: width = 0;
                  transparency = PHIGS_HATCH_TRANSPARENT;
                  break;
          case 4: width = PHIGS_HATCH_DBL_WIDTH;
                  transparency = PHIGS_HATCH_TRANSPARENT;
        }

/*  Add the SET INTERIOR STYLE INDEX and SET INTERIOR COLOUR INDEX.
    Add the FILL AREA output primitive element to the structure.
*/
      for (index = 0; index < HATCHES; index++) {
        psetintcolourind(index+2);
        psetintstyleind(hatches[index] - width - transparency);
        pfillarea(POINTS, fapoints);

/*  Move the x coordinates to the right. */
        for (xaxis = 0; xaxis < POINTS; xaxis++) {
          fapoints[xaxis].x += .15;
        }
      }

/* Finished with the row.
   Reinitialize the original x-axis data and move the y-axis down.
*/
      for (xaxis = 0; xaxis < POINTS; xaxis++) {
        fapoints[xaxis].x = orig_xy_pts[xaxis].x;
      }
      for (yaxis = 0; yaxis < ROWS; yaxis++) {
        fapoints[yaxis].y -= .25;
      }
    }

/*  Post structure ST1 to WS1 to have the structure's contents displayed.
*/
    ppoststruct(WS1, ST1, 0.);
    sprintf(buf, "Displays supported hatch styles.  Exits in 10 seconds."
    pmessage(WS1, buf);
    sleep(10);
/*  Close the structure, close the workstation and close PHIGS.
*/
        pclosestruct();
        pclosews(WS1);
        pclosephigs();
}


/*  fill_background - fills structure 2 with two solid background fill areas.
*/

fill_background()
{
    static Ppoint background1[4]  = { {.05,  .1}, {.97,  .1},
                                      {.97,  .4}, {.05,  .4} };
    static Ppoint background2[4]  = { {.05,  .6}, {.97,  .6},
                                      {.97,  .9}, {.05,  .9} };
    popenstruct(ST2);
    psetintstyle(PSOLID);
    pfillarea(4, background1);
    pfillarea(4, background2);
    pclosestruct();
}
```

ex1.c

Demonstrates 3-D transformations using a *Sun Tool* workstation.

```
#ifndef lint
static  char sccsid[] = "@(#)ex1.c 2.1 88/06/02 SMI";
#endif

/*
 * Copyright (c) 1987 by Sun Microsystems, Inc.
 */

/* ex1.c */

/*
 * Example program that draws the Sun Cube which is made up of the
 * Sun logo. The only geometry information is the points which
 * make up one "u" in the logo.
 */

#include <phigs/phigs.h>

#define WS1 1

#define SUNPURPLE   1   /* colour table index for "sun purple" */
#define WHITE   8   /* colour index 1 used at traversal if monochrome */

/* convert degrees to radians */
#define DEG_TO_RAD(D)   ((3.14159265358 / 180.0) * (D))

Ppoint3 u_pts[] = {
    {   1.0, 0.0, 0.0},
    {   4.0, 0.0, 0.0},
    {   5.0, 1.0, 0.0},
    {   5.0, 11.0, 0.0},
    {   3.0, 11.0, 0.0},
    {   3.0, 2.0, 0.0},
    {   2.0, 2.0, 0.0},
    {   2.0, 11.0, 0.0},
    {   0.0, 11.0, 0.0},
    {   0.0, 1.0, 0.0},
    {   1.0, 0.0, 0.0}
};

#define NUM_U_PTS (sizeof(u_pts)/sizeof(Ppoint3))

u_struct()
{
    popenstruct(1);
    /* "U" points upward. is in * the lower left of a face.  */
        ppolyline3(NUM_U_PTS, u_pts);
    pclosestruct();
}

un_struct()
{
    Pmatrix3    transl, rotz, trans2, composite;
    Pmatrix3    t1;
    Pvector3    p;
    Pint    err;

    popenstruct(2);
    psetlinecolourind(WHITE);
        pexecutestruct(1);

        psetlinecolourind(SUNPURPLE);
        p.x = -2.5;
        p.y = 0.0;
    p.z = 0.0;
    ptranslate3(&p, &err, transl);
    protatez(DEG_TO_RAD(180.0), &err, rotz);
    p.x = 8.5;
    p.y = 11.0;
    p.z = 0.0;
    ptranslate3(&p, &err, trans2);
    pcomposematrix3(rotz, transl, &err, tl);
    pcomposematrix3(trans2, tl, &err, composite);
    psetlocaltran3(composite, PREPLACE);
    /* transform U to point down (kinda looks like an N)
     * and sit on the right of the U in structure 1
     */
    pexecutestruct(1);
    pclosestruct();
}

logo()
{
    Pmatrix3    transl, rotz, trans2, composite;
    Pmatrix3    tl;
    Pvector3    p;
    Pint    err;

    popenstruct(3);
        pexecutestruct(2);  /* lower left UN */
    pclosestruct();

    /* translate the UN to its center */
    p.x = -5.5;
    p.y = -5.5;
    p.z = 0.0;
    ptranslate3(&p, &err, transl);
    popenstruct(3);
    protatez(DEG_TO_RAD(90.0), &err, rotz);
    p.x = 5.5;
    p.y = 17.5;
    p.z = 0.0;
    ptranslate3(&p, &err, trans2);
    pcomposematrix3(rotz, transl, &err, tl);
    pcomposematrix3(trans2, tl, &err, composite);
    psetlocaltran3(composite, PREPLACE);
    pexecutestruct(2);  /* upper left S */
    pclosestruct();

    popenstruct(3);
    protatez(DEG_TO_RAD(180.0), &err, rotz);
    p.x = 17.5;
    p.y = 17.5;
    p.z = 0.0;
    ptranslate3(&p, &err, trans2);
    pcomposematrix3(rotz, trans1, &err, t1);
    pcomposematrix3(trans2, t1, &err, composite);
    psetlocaltran3(composite, PREPLACE);
    pexecutestruct(2);  /* upper right UN */
    pclosestruct();

    popenstruct(3);
    protatez(DEG_TO_RAD(270.0), &err, rotz);
    p.x = 17.5;
    p.y = 5.5;
    p.z = 0.0;
    ptranslate3(&p, &err, trans2);
    pcomposematrix3(rotz, trans1, &err, t1);
    pcomposematrix3(trans2, t1, &err, composite);
    psetlocaltran3(composite, PREPLACE);
    pexecutestruct(2);  /* lower right S */
    pclosestruct();

}

face()
{
    popenstruct(4);
    psetviewind(1);
    pexecutestruct(3);
    pclosestruct();
}


cube2()
{
    Pmatrix3    rot, trans;
    Pmatrix3    composite1, composite2;
    Pvector3    p;
    Pint    err;

    protatey(DEG_TO_RAD(90.0), &err, rot);
    p.x = 23.0;
    p.y = 0.0;
    p.z = 0.0;
    ptranslate3(&p, &err, trans);
    pcomposematrix3(trans, rot, &err, composite1);/* right face */

    protatex(-DEG_TO_RAD(90.0), &err, rot);
    p.x = 0.0;
    p.y = 23.0;
    p.z = 0.0;
    ptranslate3(&p, &err, trans);
    pcomposematrix3(trans, rot, &err, composite2);/* top face */

    popenstruct(4);
        psetlocaltran3(composite1, PREPLACE);
        pexecutestruct(3);
    psetlocaltran3(composite2, PREPLACE);
    pexecutestruct(3);
    pclosestruct();
}

Pviewmapping3 view_map = {
    { -50.0, 50.0, -50.0, 50.0},         /* window */
    { 0.0, 1.0, 0.0, 1.0, 0.0, 1.0},      /* viewport */
    PPARALLEL,                   /* viewport type */
    { 0.5, 0.5, 1000.0},             /* prp */
    0.0,                     /* view plane dist */
    -50.0,                   /* back plane dist */
    50.0                     /* front plane dist */
};

set_up_view(ws_id)
Pint    ws_id;
{
    Ppoint3 vrp;
    Pvector3    vpn, vup;
    Pint    err;
    Pviewrep3   vrep;

    vrp.x = 23.0; vrp.y = 23.0; vrp.z = 0.0;
    vpn.x = 1.0; vpn.y = 1.0; vpn.z = 1.0;
    vup.x = 0.0; vup.y = 1.0; vup.z = 0.0;
    pevalvieworientationmatrix3(&vrp, &vpn, &vup, &err,
    vrep.orientation_matrix);
    if (err) {
    fprintf(stderr, "error from eval orientation %d\n", err);
    exit(4);
    }
    pevalviewmappingmatrix3(&view_map, &err, vrep.mapping_matrix);
    if (err) {
    fprintf(stderr, "error from eval mapping %d\n", err);
    exit(5);
    }
    vrep.clip_limit = view_map.viewport;
    vrep.clip_xy = vrep.clip_back = vrep.clip_front = PNOCLIP;
    psetviewrep3(ws_id, 1, &vrep);
}

Pcobundl sunpurple = {175.0/255.0, 125.0/255.0, 255.0/255.0};

main(argc, argv)
int argc;
char    *argv[];
{
    Pwstype wst;
    unsigned    etime = 10;
    Pchar   buf[100];

    popenphigs((Pchar*)NULL, PDEFAULT_MEM_SIZE);
    {
    Psystemstate    sys_state;

    pinqsystemst(&sys_state);
    if (sys_state != PPHOP)
        exit(1);
    }
    wst = phigs_ws_type_create( phigs_ws_type_sun_tool,
    PHIGS_TOOL_LABEL, "SunPHIGS Tool Workstation",
    0);
    if ( !wst ) {
    pclosephigs();
    exit(1);
    }

    popenws(WS1, (Pconnid)NULL, wst);
    {
    Pwsstate    ws_state;

    pinqwsst(&ws_state);
    if (ws_state != PWSOP)
        exit(3);
    }

    set_up_view(WS1);
    psetdisplayupdatest(WS1, PASAP, PNIVE);
    /* make a "sun purple" */
    psetcolourrep(WS1, SUNPURPLE, &sunpurple);

    u_struct();
    un_struct();
    logo();
    face();
    cube2();
    ppoststruct(WS1, 4, 1.0);
    sprintf( buf, "Sun Logo.  Program will exit in %u seconds.", etime);
    pmessage(WS1, buf);
    sleep(etime);
    pclosews(WS1);
    pclosephigs();
}
```

ex2.c

Demonstrates 3-D transformations using a *Sun Canvas* workstation.

```
#ifndef lint
#static  char sccsid[] = "@(#)ex2.c 2.1 88/06/02 SMI"
#endif

/*
 * Copyright (c) 1988 by Sun Microsystems, Inc.
 */

/*
#include <phigs/phigs.h>
#include <suntool/canvas.h>

#define WS1 1

#define SUNPURPLE   1   /* colour table index for "sun purple" */
#define WHITE   8   /* colour index 1 used at traversal if monochrome */

/* convert degrees to radians */
#define DEG_TO_RAD(D)   ((3.14159265358 / 180.0) * (D))

Ppoint3 u_pts[] = {
    {   1.0, 0.0, 0.0},
    {   4.0, 0.0, 0.0},
    {   5.0, 1.0, 0.0},
    {   5.0, 11.0, 0.0},
    {   3.0, 11.0, 0.0},
    {   3.0, 2.0, 0.0},
    {   2.0, 2.0, 0.0},
    {   2.0, 11.0, 0.0},
    {   0.0, 11.0, 0.0},
    {   0.0, 1.0, 0.0},
    {   1.0, 0.0, 0.0}
};

#define NUM_U_PTS (sizeof(u_pts)/sizeof(Ppoint3))

u_struct()
{
    popenstruct(1);
    /* "U" points upward. is in * the lower left of a face.  */
        ppolyline3(NUM_U_PTS, u_pts);
    pclosestruct();
}

un_struct()
{
    Pmatrix3    transl, rotz, trans2, composite;
    Pmatrix3    t1;
    Pvector3    p;
    Pint    err;

    popenstruct(2);
    psetlinecolourind(WHITE);
        pexecutestruct(1);
        psetlinecolourind(SUNPURPLE);
        p.x = -2.5;
        p.y = 0.0;
    p.z = 0.0;
    ptranslate3(&p, &err, trans1);
    protatez(DEG_TO_RAD(180.0), &err, rotz);
    p.x = 8.5;
    p.y = 11.0;
    p.z = 0.0;
    ptranslate3(&p, &err, trans2);
    pcomposematrix3(rotz, trans1, &err, t1);
    pcomposematrix3(trans2, t1, &err, composite);
    psetlocaltran3(composite, PREPLACE);
    /* transform U to point down (kinda looks like an N)
     * and sit on the right of the U in structure 1
     */
    pexecutestruct(1);
    pclosestruct();
}

logo()
{
    Pmatrix3    trans1, rotz, trans2, composite;
    Pmatrix3    t1;
    Pvector3    p;
    Pint    err;

    popenstruct(3);
        pexecutestruct(2);  /* lower left UN */
    pclosestruct();

    /* translate the UN to its center */
    p.x = -5.5;
    p.y = -5.5;
    p.z = 0.0;
    ptranslate3(&p, &err, trans1);
    popenstruct(3);
    protatez(DEG_TO_RAD(90.0), &err, rotz);
    p.x = 5.5;
    p.y = 17.5;
    p.z = 0.0;
    ptranslate3(&p, &err, trans2);
    pcomposematrix3(rotz, trans1, &err, t1);
    pcomposematrix3(trans2, t1, &err, composite);
    psetlocaltran3(composite, PREPLACE);
    pexecutestruct(2);  /* upper left S */
    pclosestruct();

    popenstruct(3);
    protatez(DEG_TO_RAD(180.0), &err, rotz);
    p.x = 17.5;
    p.y = 17.5;
    p.z = 0.0;
    ptranslate3(&p, &err, trans2);
    pcomposematrix3(rotz, trans1, &err, t1);
    pcomposematrix3(trans2, t1, &err, composite);
    psetlocaltran3(composite, PREPLACE);
    pexecutestruct(2);  /* upper right UN */
    pclosestruct();

    popenstruct(3),
    protatez(DEG_TO_RAD(270.0), &err, rotz);
    p.x = 17.5;
    p.y = 5.5;
    p.z = 0.0;
    ptranslate3(&p, &err, trans2);
    pcomposematrix3(rotz, trans1, &err, t1);
    pcomposematrix3(trans2, t1, &err, composite);
    psetlocaltran3(composite, PREPLACE);
    pexecutestruct(2);  /* lower right S */
    pclosestruct();

}

face()
{
    popenstruct(4);
    psetviewind(1);
    pexecutestruct(3);
    pclosestruct();
}


cube2()
{
    Pmatrix3    rot, trans;
    Pmatrix3    composite1, composite2;
    Pvector3    p;
    Pint    err;

    protatey(DEG_TO_RAD(90.0), &err, rot);
    p.x = 23.0;
    p.y = 0.0;
    p.z = 0.0;
    ptranslate3(&p, &err, trans);
    pcomposematrix3(trans, rot, &err, composite1);/* right face */

    protatex(-DEG_TO_RAD(90.0), &err, rot);
    p.x = 0.0;
    p.y = 23.0;
    p.z = 0.0;
    ptranslate3(&p, &err, trans);
    pcomposematrix3(trans, rot, &err, composite2);/* top face */

    popenstruct(4);
        psetlocaltran3(composite1, PREPLACE);
        pexecutestruct(3);
    psetlocaltran3(composite2, PREPLACE);
    pexecutestruct(3);
    pclosestruct();
}

Pviewmapping3 view_map = {
    { -50.0, 50.0, -50.0, 50.0},        /* window */
    { 0.0, 1.0, 0.0, 1.0, 0.0, 1.0},     /* viewport */
    PPARALLEL,                  /* viewport type */
    { 0.5, 0.5, 1000.0},             /* prp */
    0.0,                    /* view plane dist */
    -50.0,                  /* back plane dist */
    50.0                    /* front plane dist */
};

set_up_view(ws_id)
Pint    ws_id;
{
    Ppoint3 vrp;
    Pvector3    vpn, vup;
    Pint    err;
    Pviewrep3   vrep;

    vrp.x = 23.0; vrp.y = 23.0; vrp.z = 0.0;
    vpn.x = 1.0; vpn.y = 1.0; vpn.z = 1.0;
    vup.x = 0.0; vup.y = 1.0; vup.z = 0.0;
    pevalvieworientationmatrix3(&vrp, &vpn, &vup, &err,
    vrep.orientation_matrix);
    if (err) {
    fprintf(stderr, "error from eval orientation %d\n", err);
    exit(4);
    }
    pevalviewmappingmatrix3(&view_map, &err, vrep.mapping_matrix);
    if (err) {
    fprintf(stderr, "error from eval mapping %d\n", err);
    exit(5);
    }
    vrep.clip_limit = view_map.viewport;
    vrep.clip_xy = vrep.clip_back = vrep.clip_front = PNOCLIP;
    psetviewrep3(ws_id, 1, &vrep);
}

Pcobundl sunpurple = {175.0/255.0, 125.0/255.0, 255.0/255.0};

main(argc, argv)
int argc;
char    *argv[];
{
    Frame   frame;
    Canvas  canvas;

    frame = window_create(NULL, FRAME,
    FRAME_LABEL, "SunPHIGS Canvas Workstation",
    0);
    if ( !frame )
    exit(1);
    canvas = window_create(frame, CANVAS,
       WIN_WIDTH, 600,
       WIN_HEIGHT,  600,
       0);
    if ( ! canvas )
       exit(1);
    window_fit(frame);

    popenphigs((Pchar*)NULL, PDEFAULT_MEM_SIZE);
    {
```

```
Psystemstate    sys_state;

pinqsystemst(&sys_state);
if (sys_state != PPHOP)
    exit(1);
}

popenws(WS1, (Pconnid)canvas, phigs_ws_type_sun_canvas);
{
Pwsstate    ws_state;

pinqwsst(&ws_state);
if (ws_state != PWSOP)
    exit(3);
}

set_up_view(WS1);
psetdisplayupdatest(WS1, PASAP, PNIVE);
/* make a "sun purple" */
psetcolourrep(WS1, SUNPURPLE, &sunpurple);

u_struct();
un_struct();
logo();
face();
cube2();
ppoststruct(WS1, 4, 1.0);
window_main_loop(frame);
pclosews(WS1);
pclosephigs();
```

fourview.c

Demonstrates the 3-D viewing model using a *Sun Tool* workstation.

```
#ifndef lint
static char sccsid[] = "@(#)fourview.c 2.2 89/02/16 Copyr 1988 Sun Micro";
#endif

/*
 * Copyright (c) 1988,1989 by Sun Microsystems, Inc.
 */

/*
   This program is an example of how to set up four views of the same
   object and interact with it in two ways:

       1) Change the viewing parameters of only one view.
       2) Rotate the object only in one view.

   The second operation is supported with a quick-update-method (QUM) and
   can be done without any redraw of the other three views. The first
   operation is currently not supported by a QUM, thus it causes the entire
   workstation to be redrawn when any one view is changed (but with
   colour map double buffering turned on, this is not visible to the user).

   main() is at the bottom of this file.
 */

#include <phigs/phigs.h>

/* Device ids */
#define WSID        1

/* Workstation size and location. */
#define WS_X        100
#define WS_Y        150
#define WS_WIDTH    200
#define WS_HEIGHT   200

/* Colors */
#define BLACK       0
#define WHITE       1
#define RED         2
#define GREEN       3
#define BLUE        4
#define YELLOW      5
#define CYAN        6
#define MAGENTA     7

/* Structure names. */
#define OBJECT      10

#define VIEW_1      1
#define VIEW_2      2
#define VIEW_3      3
#define VIEW_4      4

/* Labels */
```

```
#define OBJECT_TRANSFORM    1


static void
build_css()
{
    /* Build a cube with fill areas for sides. */

    static Ppoint3  fll = { -1.0,-1.0, 1.0};
    static Ppoint3  flr = {  1.0,-1.0, 1.0};
    static Ppoint3  fur = {  1.0, 1.0, 1.0};
    static Ppoint3  ful = { -1.0, 1.0, 1.0};

    static Ppoint3  bll = { -1.0,-1.0,-1.0};
    static Ppoint3  blr = {  1.0,-1.0,-1.0};
    static Ppoint3  bur = {  1.0, 1.0,-1.0};
    static Ppoint3  bul = { -1.0, 1.0,-1.0};

    static Pmatrix3 identity = { 1.0, 0.0, 0.0, 0.0,
                     0.0, 1.0, 0.0, 0.0,
                     0.0, 0.0, 1.0, 0.0,
                     0.0, 0.0, 0.0, 1.0};

    Ppoint3 points[4];
    Ppointlst3  side;

    side.number = 4;
    side.points = points;

    /* Build the object.  Set the colors so that the red, blue and green
     * sides intersect the positive x, y and z axes, respectively.
     */
    popenstruct( OBJECT );
    psetintstyle( PSOLID );
    psethlhsrid( PHIGS_HLHSR_ID_ZBUFF );

    /* front */
    points[0] = fll; points[1] = flr; points[2] = fur; points[3] = ful;
    psetintcolourind( BLUE );
    pfillareaset3( 1, &side );

    /* back */
    points[0] = bll; points[1] = blr; points[2] = bur; points[3] = bul;
    psetintcolourind( CYAN );
    pfillareaset3( 1, &side );

    /* top */
    points[0] = ful; points[1] = fur; points[2] = bur; points[3] = bul;
    psetintcolourind( GREEN );
    pfillareaset3( 1, &side );

    /* bottom */
    points[0] = fll; points[1] = flr; points[2] = blr; points[3] = bll;
    psetintcolourind( MAGENTA );
    pfillareaset3( 1, &side );

    /* right */
    points[0] = flr; points[1] = blr; points[2] = bur; points[3] = fur;
    psetintcolourind( RED );
    pfillareaset3( 1, &side );

    /* left */
    points[0] = bll; points[1] = fll; points[2] = ful; points[3] = bul;
    psetintcolourind( YELLOW );
    pfillareaset3( 1, &side );
    pclosestruct();

    /* Build the hierarchical "views."  Each of these structures will be
     * posted separately but will all reference the same object.  They all
     * set the view index and then execute the object.  The label and
     * transformation elements are added so that the object can be rotated
     * independently in each view (see the function rotate_object_in view()
     * to see how this is done).
     */

    popenstruct( VIEW_1 );
    psetviewind( VIEW_1 );
    plabel( OBJECT_TRANSFORM );
    psetlocaltran3( identity, PREPLACE );
    pexecutestruct( OBJECT );
    pclosestruct();

    popenstruct( VIEW_2 );
    psetviewind( VIEW_2 );
    plabel( OBJECT_TRANSFORM );
    psetlocaltran3( identity, PREPLACE );
    pexecutestruct( OBJECT );
    pclosestruct();

    popenstruct( VIEW_3 );
    psetviewind( VIEW_3 );
    plabel( OBJECT_TRANSFORM );
    psetlocaltran3( identity, PREPLACE );
    pexecutestruct( OBJECT );
    pclosestruct();

    popenstruct( VIEW_4 );
    psetviewind( VIEW_4 );
    plabel( OBJECT_TRANSFORM );
    psetlocaltran3( identity, PREPLACE );
    pexecutestruct( OBJECT );
    pclosestruct();
}

static void
set_up_views()
{
    Pviewrep3       rep;
    Pviewmapping3   map;
    Ppoint3     vrp;
    Pvector3        vup;
    Pvector3        vpn;
    Pint        err;

    /* All views use the same reference point. */
    vrp.x = 0.0; vrp.y = 0.0; vrp.z = 0.0;
```

```
map.proj = PPARALLEL;
rep.clip_xy = rep.clip_back = rep.clip_front = PCLIP;
map.window.xmin = -2.0; map.window.xmax = 2.0;
map.window.ymin = -2.0; map.window.ymax = 2.0;
map.back_plane = -2.0;
map.front_plane = 2.0;
map.view_plane =  1.8;
map.prp.x = (map.window.xmin + map.window.xmax) / 2.0;
map.prp.y = (map.window.ymin + map.window.ymax) / 2.0;
map.prp.z = 10.0;
map.viewport.zmin = 0.0; map.viewport.zmax = 1.0;

/* View 1 -- top view */
vup.x =  0.0; vup.y =  0.0; vup.z = -1.0;
vpn.x =  0.0; vpn.y =  1.0; vpn.z =  0.0;
pevalvieworientationmatrix3( &vrp, &vpn, &vup, &err,
rep.orientation_matrix);
map.viewport.xmin = 0.05; map.viewport.xmax = 0.45;
map.viewport.ymin = 0.55; map.viewport.ymax = 0.95;
pevalviewmappingmatrix3( &map, &err, rep.mapping_matrix);
rep.clip_limit = map.viewport;
psetviewrep3( WSID, VIEW_1, &rep );

/* View 2 -- off axis view */
vup.x =  0.0; vup.y =  1.0; vup.z =  0.0;
vpn.x =  1.0; vpn.y =  1.0; vpn.z =  1.0;
pevalvieworientationmatrix3( &vrp, &vpn, &vup, &err,
rep.orientation_matrix);
map.viewport.xmin = 0.55; map.viewport.xmax = 0.95;
map.viewport.ymin = 0.55; map.viewport.ymax = 0.95;
pevalviewmappingmatrix3( &map, &err, rep.mapping_matrix);
rep.clip_limit = map.viewport;
psetviewrep3( WSID, VIEW_2, &rep );

/* View 3 -- front view */
vup.x =  0.0; vup.y =  1.0; vup.z =  0.0;
vpn.x =  0.0; vpn.y =  0.0; vpn.z =  1.0;
pevalvieworientationmatrix3( &vrp, &vpn, &vup, &err,
rep.orientation_matrix);
map.viewport.xmin = 0.05; map.viewport.xmax = 0.45;
map.viewport.ymin = 0.05; map.viewport.ymax = 0.45;
pevalviewmappingmatrix3( &map, &err, rep.mapping_matrix);
rep.clip_limit = map.viewport;
psetviewrep3( WSID, VIEW_3, &rep );

/* View 4 -- right side view */
vup.x =  0.0; vup.y =  1.0; vup.z =  0.0;
vpn.x =  1.0; vpn.y =  0.0; vpn.z =  0.0;
pevalvieworientationmatrix3( &vrp, &vpn, &vup, &err,
rep.orientation_matrix);
map.viewport.xmin = 0.55; map.viewport.xmax = 0.95;
map.viewport.ymin = 0.05; map.viewport.ymax = 0.45;
pevalviewmappingmatrix3( &map, &err, rep.mapping_matrix);
rep.clip_limit = map.viewport;
psetviewrep3( WSID, VIEW_4, &rep );

/* Set all these priorities to be higher than view 0. */
psetviewtraninputpri( WSID, 0, VIEW_1, PLOWER );
```

```
    psetviewtraninputpri( WSID, 0, VIEW_2, PLOWER );
    psetviewtraninputpri( WSID, 0, VIEW_3, PLOWER );
    psetviewtraninputpri( WSID, 0, VIEW_4, PLOWER );
}

static void
rotate_view( view )
    Pint    view;
{
    Pviewrep3   rep, req_rep;
    Pupdatest   update_state;
    Ppoint3 vrp;
    Pvector3    vup;
    Pvector3    vpn;
    Pint    err;
    Pfloat  *var_dim;
    int     i;

    /* Change the viewing angle (view plane normal) of the specified view. */

    switch ( view ) {
    case VIEW_1:
        vup.x =  0.0; vup.y =  0.0; vup.z = -1.0;
        vpn.x =  0.0; vpn.y =  1.0; vpn.z =  0.0;
        var_dim = &vpn.x;
        break;
    case VIEW_2:
        vup.x =  0.0; vup.y =  1.0; vup.z =  0.0;
        vpn.x =  1.0; vpn.y =  1.0; vpn.z =  1.0;
        var_dim = &vpn.x;
        break;
    case VIEW_3:
        vup.x =  0.0; vup.y =  1.0; vup.z =  0.0;
        vpn.x =  0.0; vpn.y =  0.0; vpn.z =  1.0;
        var_dim = &vpn.x;
        break;
    case VIEW_4:
        vup.x =  0.0; vup.y =  1.0; vup.z =  0.0;
        vpn.x =  1.0; vpn.y =  0.0; vpn.z =  0.0;
        var_dim = &vpn.z;
        break;
    default:
        return;
    }

    /* Inquire the existing view representation and just change
     * its orientation matrix
     */
    pinqviewrep( WSID, view, &err, &update_state, &rep, &req_rep );
    vrp.x = 0.0; vrp.y = 0.0; vrp.z = 0.0;

    /* Set the view deferral state to display changes immediately. */
    psetdisplayupdatest( WSID, PASAP, PNIVE );

    for ( i = 1; i <= 10; i++ ) {
    *var_dim -= 0.10;
    pevalvieworientationmatrix3( &vrp, &vpn, &vup, &err,
        rep.orientation_matrix);
    psetviewrep3( WSID, view, &rep );
    }
    for ( i = 1; i <= 20; i++ ) {
    *var_dim += 0.10;
    pevalvieworientationmatrix3( &vrp, &vpn, &vup, &err,
        rep.orientation_matrix);
    psetviewrep3( WSID, view, &rep );
    }
    for ( i = 1; i <= 10; i++ ) {
    *var_dim -= 0.10;
    pevalvieworientationmatrix3( &vrp, &vpn, &vup, &err,
        rep.orientation_matrix);
    psetviewrep3( WSID, view, &rep );
    }
}

static void
rotate_object_in_view( view )
    Pint    view;
{
    static Ppoint3  pt = {0.0,0.0,0.0};
    static Pvector3 scale = {1.0,1.0,1.0};

    Pmatrix3    transform;
    Pint    err;
    Peditmode   edit_mode;
    int     i;

    if (!view)
    /* User picked outside of the four views - nothing to do. */
    return;

    /* Rotate the object only in the specified view.  This is done by
     * changing the modelling transform in the "view" structure that
     * executes the object.
     */

    /* There is a QUM to support replacement of modelling transforms.  This
     * QUM draws the object in the background color before replacing the
     * transform, then replaces the transform and draws the structure in
     * the correct colors.
     */
    psetdisplayupdatest( WSID, PWAIT, PUQUM );

    /* Save and set the edit mode.  We want REPLACE for this function
     * since we will be replacing the existing transform.
     */
    pinqeditmode( &err, &edit_mode );
    pseteditmode( PEDIT_REPLACE );

    /* Open the structure, position the element pointer to the transform
     * element and replace it.  Each time the transformation element is
     * replaced, the SunPHIGS QUM simulation will make the new transformation
     * visible.
     */
    popenstruct( view );
    psetelemptr(0);
    psetelemptrlabel( OBJECT_TRANSFORM );
    poffsetelemptr( 1 );
    for ( i = 0; i < 100; i++ ) {
        pbuildtran3( &pt, &pt,
        (Pfloat)i/30., (Pfloat)i/40., (Pfloat)i/50.,
        &scale, &err, transform);
        psetlocaltran3( transform, PREPLACE );
    }
    for ( i = 99; i >= 0; i-- ) {
        pbuildtran3( &pt, &pt,
        (Pfloat)i/30., (Pfloat)i/40., (Pfloat)i/50.,
        &scale, &err, transform);
        psetlocaltran3( transform, PREPLACE );
    }

    pclosestruct();
    /* The workstation has visual representation "simulated."
     * Make the workstation entirely correct by updating.
     */
    pupdatews( WSID, PPERFORM );

    /* Restore the edit mode. */
    pseteditmode( edit_mode );
}

static void
input_loop()
{
    int     done = 0;
    Pevent  event;

    /* Enable the input devices. */
    psetlocmode( WSID, 1, PEVENT, PES_ECHO);
    psetlocmode( WSID, 2, PEVENT, PES_ECHO);
    psetlocmode( WSID, 3, PEVENT, PES_ECHO);

    /* Get and process input.  The operator is always selecting views.
     * The operation to perform is based on the button used to
     * select the view:
        Left mouse button:  change viewing paramters.
        Middle mouse button:    rotate the object.
        Right mouse button: terminate the program.
     */
    do {
    pawaitevent( 60.0, &event );
    switch ( event.class ) {
        case PI_LOCATOR: {
        Ploc3   location;

        pgetloc3( &location );
        switch ( event.dev ) {
            case 1:
            rotate_view( location.view_index );
            break;
            case 2:
            rotate_object_in_view( location.view_index );
            break;
            case 3:
            done = 1;
            break;
        }
        break;
        }
    }
    } while ( !done );
}

main( argc, argv )
    int     argc;
    char    *argv[];
{
    Pwstype     wst;

    popenphigs( (Pchar*)0, PDEFAULT_MEM_SIZE );

    /* Create the display hierarchy. */
    build_css();

    /* Create a workstation type with the description table values we want
     * and open a workstation.  Decommission the workstation type when
     * done with it to free up resources.  The specific workstation type is
     * saved (internally to SunPHIGS) with the workstation.
     *
     * Turn on color map double buffering to make updates smoother.
     */
    wst = phigs_ws_type_create( phigs_ws_type_sun_tool,
    PHIGS_TOOL_WIDTH, WS_WIDTH, PHIGS_TOOL_HEIGHT, WS_HEIGHT,
    PHIGS_TOOL_X, WS_X, PHIGS_TOOL_Y, WS_Y,
    PHIGS_TEXTSW, PHIGS_NONE,
    PHIGS_TOOL_LABEL, "Viewing Example",
    PHIGS_COLOR_TABLE_SIZE, 8,
    PHIGS_DOUBLE_BUFFER, PHIGS_DBL_CMAP,
    0);
    popenws( WSID, (Pconnid)0, wst );
    phigs_ws_type_destroy( wst );

    /* Enable Z-buffering. */
    psethlhsrmode( WSID, PHIGS_HLHSR_MODE_ZBUFF );

    /* Set up the views we'll be using. */
    set_up_views();

    /* Set the deferral state so that the screen is not updated when the
     * hierarchy is posted, then post the hierarchy.  The structures posted
     * don't overlap when displayed so posting priority is unimportant.
     */
    psetdisplayupdatest( WSID, PWAIT, PNIVE );
    ppoststruct( WSID, VIEW_1, 1.0 );
    ppoststruct( WSID, VIEW_2, 1.0 );
    ppoststruct( WSID, VIEW_3, 1.0 );
    ppoststruct( WSID, VIEW_4, 1.0 );

    /* Now display the hierarchy and start looking for input. */
    pupdatews( WSID, PPERFORM );
    input_loop();

    /* No more input, close everything. */
    pclosews( WSID );
    pclosephigs();
}
```

fourview_cvs.c

Demonstrates the 3-D viewing model using a *Sun Canvas* workstation.

```
#ifndef lint
static char sccsid[] = "@(#)fourview_cvs.c 2.1 88/06/02 Copyr 1988 Sun Micro";
#endif

/*
 * Copyright (c) 1988 by Sun Microsystems, Inc.
 */

/*
   This program is an example of how to set up four views of the same
   object and interact with it in two ways:

   1) Change the viewing parameters of only one view.
   2) Rotate the object only in one view.

   Four sun_canvas workstations are used, one for each view.  This allows
   each "view" to be updated completely independently of the other three,
   and does not cause all of the "views" to be redrawn when one of them
   changes.
 */

#include <phigs/phigs.h>
#include <suntool/canvas.h>

/* Workstation size and location. */
#define WS_X        100
#define WS_Y        150
#define WS_WIDTH    200
#define WS_HEIGHT   200

/* Colors */
#define BLACK       0
#define WHITE       1
#define RED     2
#define GREEN       3
#define BLUE        4
#define YELLOW      5
#define CYAN        6
#define MAGENTA     7

/* Structure names. */
#define OBJECT      10

/* These are really workstation id's corresponding to the four "views." */
#define VIEW_1      1
#define VIEW_2      2
#define VIEW_3      3
#define VIEW_4      4

/* Labels */
#define OBJECT_TRANSFORM    1


static void
```

```
build_css( triangles )
    int     triangles;
{
    /* Build a cube with fill areas for sides. */

    static Ppoint3  fll = { -1.0,-1.0, 1.0};
    static Ppoint3  flr = {  1.0,-1.0, 1.0};
    static Ppoint3  fur = {  1.0, 1.0, 1.0};
    static Ppoint3  ful = { -1.0, 1.0, 1.0};

    static Ppoint3  bll = { -1.0,-1.0,-1.0};
    static Ppoint3  blr = {  1.0,-1.0,-1.0};
    static Ppoint3  bur = {  1.0, 1.0,-1.0};
    static Ppoint3  bul = { -1.0, 1.0,-1.0};

    static Pmatrix3 identity = { 1.0, 0.0, 0.0, 0.0,
                     0.0, 1.0, 0.0, 0.0,
                     0.0, 0.0, 1.0, 0.0,
                     0.0, 0.0, 0.0, 1.0};

    Ppoint3 points[4];
    Ppointlst3  side;

    side.number = triangles ? 3 : 4;
    side.points = points;

    /* Build the object.  Set the colors so that the red, blue and green
     * sides intersect the positive x, y and z axes, respectively.
     */
    popenstruct( OBJECT );
    psetintstyle( PSOLID );
    psetedgeflag( PEDGE_OFF);
    psethlhsrid( PHIGS_HLHSR_ID_ZBUFF );

    /* front */
    psetintcolourind( BLUE );
    if ( !triangles ) {
        points[0] = fll; points[1] = flr; points[2] = fur; points[3] = ful;
        pfillareaset3( 1, &side );
    } else {
        points[0] = fll; points[1] = flr; points[2] = fur;
        pfillareaset3( 1, &side );
        points[0] = fll; points[1] = fur; points[2] = ful;
        pfillareaset3( 1, &side );
    }

    /* back */
    psetintcolourind( CYAN );
    if ( !triangles ) {
        points[0] = bll; points[1] = blr; points[2] = bur; points[3] = bul;
        pfillareaset3( 1, &side );
    } else {
        points[0] = bll; points[1] = blr; points[2] = bur;
        pfillareaset3( 1, &side );
        points[0] = bll; points[1] = bur; points[2] = bul;
        pfillareaset3( 1, &side );
    }
```

```
/* top */
psetintcolourind( GREEN );
if ( !triangles ) {
    points[0] = ful; points[1] = fur; points[2] = bur; points[3] = bul;
    pfillareaset3( 1, &side );
} else {
    points[0] = ful; points[1] = fur; points[2] = bur;
    pfillareaset3( 1, &side );
    points[0] = ful; points[1] = bur; points[2] = bul;
    pfillareaset3( 1, &side );
}

/* bottom */
psetintcolourind( MAGENTA );
if ( !triangles ) {
    points[0] = fll; points[1] = flr; points[2] = blr; points[3] = bll;
    pfillareaset3( 1, &side );
} else {
    points[0] = fll; points[1] = flr; points[2] = blr;
    pfillareaset3( 1, &side );
    points[0] = fll; points[1] = blr; points[2] = bll;
    pfillareaset3( 1, &side );
}

/* right */
psetintcolourind( RED );
if ( !triangles ) {
    points[0] = flr; points[1] = blr; points[2] = bur; points[3] = fur;
    pfillareaset3( 1, &side );
} else {
    points[0] = flr; points[1] = blr; points[2] = bur;
    pfillareaset3( 1, &side );
    points[0] = flr; points[1] = bur; points[2] = fur;
    pfillareaset3( 1, &side );
}

/* left */
psetintcolourind( YELLOW );
if ( !triangles ) {
    points[0] = bll; points[1] = fll; points[2] = ful; points[3] = bul;
    pfillareaset3( 1, &side );
} else {
    points[0] = bll; points[1] = fll; points[2] = ful;
    pfillareaset3( 1, &side );
    points[0] = bll; points[1] = ful; points[2] = bul;
    pfillareaset3( 1, &side );
}
pclosestruct();

/* Build the hierarchical "views."  Each of these structures will be
 * posted separately but will all reference the same object.  They all
 * set the view index and then execute the object.  The label and
 * transformation elements are added so that the object can be rotated
 * independently in each view (see the function rotate_object_in view()
 * to see how this is done).
 */

popenstruct( VIEW_1 );
```

```
    psetviewind( 1 );
    plabel( OBJECT_TRANSFORM );
    psetlocaltran3( identity, PREPLACE );
    pexecutestruct( OBJECT );
    pclosestruct();

    popenstruct( VIEW_2 );
    psetviewind( 1 );
    plabel( OBJECT_TRANSFORM );
    psetlocaltran3( identity, PREPLACE );
    pexecutestruct( OBJECT );
    pclosestruct();

    popenstruct( VIEW_3 );
    psetviewind( 1 );
    plabel( OBJECT_TRANSFORM );
    psetlocaltran3( identity, PREPLACE );
    pexecutestruct( OBJECT );
    pclosestruct();

    popenstruct( VIEW_4 );
    psetviewind( 1 );
    plabel( OBJECT_TRANSFORM );
    psetlocaltran3( identity, PREPLACE );
    pexecutestruct( OBJECT );
    pclosestruct();
}

static void
set_up_views()
{
    Pviewrep3       rep;
    Pviewmapping3   map;
    Ppoint3     vrp;
    Pvector3        vup;
    Pvector3        vpn;
    Pint        err;

    /* All views use the same reference point. */
    vrp.x = 0.0; vrp.y = 0.0; vrp.z = 0.0;

    map.proj = PPARALLEL;
    rep.clip_xy = rep.clip_back = rep.clip_front = PCLIP;
    map.window.xmin = -2.0; map.window.xmax = 2.0;
    map.window.ymin = -2.0; map.window.ymax = 2.0;
    map.back_plane = -2.0;
    map.front_plane = 2.0;
    map.view_plane =  1.8;
    map.viewport.xmin = 0.0; map.viewport.xmax = 1.0;
    map.viewport.ymin = 0.0; map.viewport.ymax = 1.0;
    map.viewport.zmin = 0.0; map.viewport.zmax = 1.0;
    rep.clip_limit = map.viewport;
    map.prp.x = (map.window.xmin + map.window.xmax) / 2.0;
    map.prp.y = (map.window.ymin + map.window.ymax) / 2.0;
    map.prp.z = 10.0;

    /* View 1 -- top view */
    vup.x =  0.0; vup.y =  0.0; vup.z = -1.0;
    vpn.x =  0.0; vpn.y =  1.0; vpn.z =  0.0;
    pevalvieworientationmatrix3( &vrp, &vpn, &vup, &err,
    rep.orientation_matrix);
    pevalviewmappingmatrix3( &map, &err, rep.mapping_matrix);
    psetviewrep3( VIEW_1, 1, &rep );

    /* View 2 -- off-axis view */
    vup.x =  0.0; vup.y =  1.0; vup.z =  0.0;
    vpn.x =  1.0; vpn.y =  1.0; vpn.z =  1.0;
    pevalvieworientationmatrix3( &vrp, &vpn, &vup, &err,
    rep.orientation_matrix);
    pevalviewmappingmatrix3( &map, &err, rep.mapping_matrix);
    psetviewrep3( VIEW_2, 1, &rep );

    /* View 3 -- front view */
    vup.x =  0.0; vup.y =  1.0; vup.z =  0.0;
    vpn.x =  0.0; vpn.y =  0.0; vpn.z =  1.0;
    pevalvieworientationmatrix3( &vrp, &vpn, &vup, &err,
    rep.orientation_matrix);
    pevalviewmappingmatrix3( &map, &err, rep.mapping_matrix);
    psetviewrep3( VIEW_3, 1, &rep );

    /* View 4 -- right side view */
    vup.x =  0.0; vup.y =  1.0; vup.z =  0.0;
    vpn.x =  1.0; vpn.y =  0.0; vpn.z =  0.0;
    pevalvieworientationmatrix3( &vrp, &vpn, &vup, &err,
    rep.orientation_matrix);
    pevalviewmappingmatrix3( &map, &err, rep.mapping_matrix);
    psetviewrep3( VIEW_4, 1, &rep );
}

static void
rotate_view( view )
    Pint    view;
{
    Pviewrep3   rep, req_rep;
    Pupdatest   update_state;
    Ppoint3 vrp;
    Pvector3    vup;
    Pvector3    vpn;
    Pint    err;
    Pfloat  *var_dim;
    int     i;

    /* Change the viewing angle (view plane normal) of the specified view. */

    switch ( view ) {
    case VIEW_1:
        vup.x =  0.0; vup.y =  0.0; vup   =  1.0;
        vpn.x =  0.0; vpn.y =  1.0; vpn.z =  0.0;
        var_dim = &vpn.x;
        break;
    case VIEW_2:
        vup.x =  0.0; vup.y =  1.0; vup.z =  0.0;
        vpn.x =  1.0; vpn.y =  1.0; vpn.z =  1.0;
        var_dim = &vpn.x;
        break;
    case VIEW_3:
            vup.x =  0.0; vup.y =  1.0; vup.z =  0.0;
            vpn.x =  0.0; vpn.y =  0.0; vpn.z =  1.0;
            var_dim = &vpn.x;
            break;
        case VIEW_4:
            vup.x =  0.0; vup.y =  1.0; vup.z =  0.0;
            vpn.x =  1.0; vpn.y =  0.0; vpn.z =  0.0;
            var_dim = &vpn.z;
            break;
        default:
            return;
        }

        /* Inquire the existing view representation and just change
         * its orientation matrix
         */
        pinqviewrep( view, 1, &err, &update_state, &rep, &req_rep );
        vrp.x = 0.0; vrp.y = 0.0; vrp.z = 0.0;
        psetdisplayupdatest( view, PASAP, PNIVE );

        for ( i = 1; i <= 10; i++ ) {
        *var_dim -= 0.10;
        pevalvieworientationmatrix3( &vrp, &vpn, &vup, &err,
            rep.orientation_matrix);
        psetviewrep3( view, 1, &rep );
        }
        for ( i = 1; i <= 20; i++ ) {
        *var_dim += 0.10;
        pevalvieworientationmatrix3( &vrp, &vpn, &vup, &err,
            rep.orientation_matrix);
        psetviewrep3( view, 1, &rep );
        }
        for ( i = 1; i <= 10; i++ ) {
        *var_dim -= 0.10;
        pevalvieworientationmatrix3( &vrp, &vpn, &vup, &err,
            rep.orientation_matrix);
        psetviewrep3( view, 1, &rep );
        }
    }

    static void
    rotate_object_in_view( view )
        Pint    view;
    {
        static Ppoint3  pt = {0.0,0.0,0.0};
        static Pvector3 scale = {1.0,1.0,1.0};

        Pmatrix3    transform;
        Pint    err;
        Peditmode   edit_mode;
        int     i;

        /* Rotate the object only in the specified view.  This is done by
         * changing the modelling transform in the "view" structure that
         * executes the object.
         */
        psetdisplayupdatest( view, PASAP, PNIVE );
    /* Save and set the edit mode.  We want REPLACE since we will be
     * replacing the existing transform, not adding a new one.
     */
    pinqeditmode( &err, &edit_mode );
    pseteditmode( PEDIT_REPLACE );

    /* Open the structure, position the element pointer to the transform
     * element and replace it.
     */
    popenstruct( view );
    psetelemptr(0);
    psetelemptrlabel( OBJECT_TRANSFORM );
    poffsetelemptr( 1 );
    for ( i = 0; i < 100; i++ ) {
        pbuildtran3( &pt, &pt,
        (Pfloat)i/30., (Pfloat)i/40., (Pfloat)i/50.,
        &scale, &err, transform);
        psetlocaltran3( transform, PREPLACE );
    }
    for ( i = 99; i >= 0; i-- ) {
        pbuildtran3( &pt, &pt,
        (Pfloat)i/30., (Pfloat)i/40., (Pfloat)i/50.,
        &scale, &err, transform);
        psetlocaltran3( transform, PREPLACE );
    }

    pclosestruct();

    /* Restore the edit mode. */
    pseteditmode( edit_mode );
}

static void
input_handler()
{
    int     done = 0;
    Pevent  event;

    /* Get and process input.  This function is called by PHIGS when an
     * input event is added to the input queue.  See the set-up for this
     * (the ESCAPE function call) in main().
     * The operation to perform is based on the button used to
     * select the view:
        Left mouse button:  change viewing paramters.
        Middle mouse button:    rotate the object.
     */

    do {
    pawaitevent( 0.0, &event );
    switch ( event.class ) {
        case PI_LOCATOR:
        switch ( event.dev ) {
            case 1:
            rotate_view( event.ws );
            break;
            case 2:
            rotate_object_in_view( event.ws );
            break;
        }
        break;

        case PI_NONE:
        done = 1;
        break;
    }
    } while ( !done );
}

main( argc, argv )
    int     argc;
    char    *argv[];
{
    Pescapein       esc_rec;
    Pwstype     wst;
    Pint        wsid, device;
    Frame       frame;
    Canvas      canvas;

    popenphigs( (Pchar*)0, PDEFAULT_MEM_SIZE );

    /* Specify the input notification function.  This function will be
     * called by PHIGS whenever an input event is added to the input
     * queue.
     */
    esc_rec.uesc2_idatarec.notify_proc = input_handler;
    pescape( PUESC_INPUT_NOTIFY_PROC, &esc_rec, (Pescapeout*)NULL);

    /* Create the display hierarchy. */
    if ( argc > 1 )
    build_css(0);
    else
    build_css(1);

    /* Create a workstation type with the description table values we want,
     * then open four workstations, each one corresponding to a different
     * "view" of the object.
     */
    wst = phigs_ws_type_create( phigs_ws_type_sun_canvas,
    PHIGS_COLOR_TABLE_SIZE, 8,
    0);

    frame = window_create( NULL, FRAME,
    WIN_X, WS_X, WIN_Y, WS_Y,
    WIN_SHOW, TRUE,
    0);
    canvas = window_create( frame, CANVAS,
    WIN_X, 0, WIN_Y, 0,
    WIN_WIDTH, WS_WIDTH, WIN_HEIGHT, WS_HEIGHT,
    0);
    popenws( VIEW_1, (Pconnid)canvas, wst );

    canvas = window_create( frame, CANVAS,
    WIN_RIGHT_OF, canvas,
    WIN_WIDTH, WS_WIDTH, WIN_HEIGHT, WS_HEIGHT,
    0);
    /* Turn on double buffering for the off-axis view.  The color
```

```
     * table isn't big enough to double buffer all workstations this way.
     */
    phigs_ws_type_set( wst, PHIGS_DOUBLE_BUFFER, PHIGS_DBL_CMAP, 0 );
    popenws( VIEW_2, (Pconnid)canvas, wst );
    phigs_ws_type_set( wst, PHIGS_DOUBLE_BUFFER, PHIGS_DBL_NONE, 0 );

    canvas = window_create( frame, CANVAS,
    WIN_X, 0, WIN_BELOW, canvas,
    WIN_WIDTH, WS_WIDTH, WIN_HEIGHT, WS_HEIGHT,
    0);
    popenws( VIEW_3, (Pconnid)canvas, wst );

    canvas = window_create( frame, CANVAS,
    WIN_RIGHT_OF, canvas,
    WIN_WIDTH, WS_WIDTH, WIN_HEIGHT, WS_HEIGHT,
    0);
    popenws( VIEW_4, (Pconnid)canvas, wst );

    window_fit( frame );

    /* Enable Z-buffering on all workstations. */
    for ( wsid = 1; wsid <= 4; wsid++ )
    psethlhsrmode( wsid, PHIGS_HLHSR_MODE_ZBUFF );

    /* Decommission the workstation type when done with it to free
     * resources.  The specific workstation type is saved (internally
     * to SunPHIGS) with the workstation.
     */
    phigs_ws_type_destroy( wst );

    /* Set up the viewing paramaters of all the views we use. */
    set_up_views();

    /* Set the deferral states and post the hierarchy.  Posting priority
     * is unimportant in this case since only one structure is posted on
     * each workstation.
     */
    for ( wsid = 1; wsid <= 4; wsid++ ) {
    psetdisplayupdatest( wsid, PASAP, PNIVE );
    ppoststruct( wsid, wsid, 1.0 );
    }

    /* Enable the input devices. */
    for ( wsid = 1; wsid <= 4; wsid++ ) {
    for ( device = 1; device <= 3; device++ )
        psetlocmode( wsid, device, PEVENT, PES_ECHO);
    }

    /* Start accepting input. */
    notify_start();

    /* Close everything. */
    for ( wsid = 1; wsid <= 4; wsid++ )
    pclosews( wsid );
    pclosephigs();
}
```

non_square.c

Demonstrates aspect ratio definition for the workstation device space.

```
#ifndef lint
static       char sccsid[] = "@(#)non_square.c 2.1 88/06/02 SMI";
#endif not lint

/*
 * Copyright (c) 1988 by Sun Microsystems, Inc.
*/

/* non-square.c - This program demonstrates the use of the PHIGS
   device coordinate attributes:  PHIGS_DEVICE_COORD_XMAX_PTR and
   PHIGS_DEVICE_COORD_YMAX_PTR which implicitly define the aspect ratio
   of the workstation device space.  This allows the display to fill the
   complete device space of the workstation.
*/

#include <phigs/phigs.h>

#define WS1       1

main(argc, argv)
int       argc;
char      *argv[];
{
    Pwstype       wst;
    Psystemstate  sys_state;
    Pwsstate      ws_state;
    Plimit        wswin;
    float         xmax = 400.0, ymax = 600.0;

/* Open PHIGS */

    popenphigs((Pchar*)NULL, PDEFAULT_MEM_SIZE);
    pinqsystemst(&sys_state);
    if (sys_state != PPHOP)
        exit(1);

/* Configure the PHIGS workstation display size and device coordinate space.
*/
    wst = phigs_ws_type_create(phigs_ws_type_sun_tool,
        PHIGS_TOOL_WIDTH,  (int)xmax,
        PHIGS_TOOL_HEIGHT, (int)ymax,
        PHIGS_DEVICE_COORD_XMAX_PTR, &xmax,
        PHIGS_DEVICE_COORD_YMAX_PTR, &ymax,
        0);

/* Open the workstation */

    popenws(WS1, (Pconnid)NULL, wst);
    pinqwsst(&ws_state);
    if (ws_state != PWSOP)
        exit(3);

/* Set the workstation window to have the same aspect ratio
   as the display space.
```

```
*/
    wswin.xmin = wswin.ymin = 0.0;
    wswin.xmax = xmax / ymax;
    wswin.ymax = 1.0;
    psetwswindow(WS1, &wswin);

/* Open a structure and insert polyline primitives and attribute elements.
*/
    popenstruct(1);
    {
        Ppoint3   pts[5];

    /* Initialize point data for the bottom box. */
        pts[0].x = pts[0].y = pts[0].z = 0.0;
        pts[1].x = 0.0;          pts[1].y = xmax / ymax;  pts[1].z = 0.0;
        pts[2].x = xmax / ymax;  pts[2].y = xmax / ymax;  pts[2].z = 0.0;
        pts[3].x = xmax / ymax;  pts[3].y = 0.0        ;  pts[3].z = 0.0;
        pts[4].x = 0.0        ;  pts[4].y = 0.0        ;  pts[4].z = 0.0;

    /* SET POLYLINE COLOUR INDEX to red and add the POLYLINE. */
        psetlinecolourind(2);
        ppolyline3(5, pts);

    /* Reset the point data and insert more POLYLINE elements. */
        pts[0].x = 0.0        ;  pts[0].y = 0.0        ;  pts[0].z = 0.0;
        pts[1].x = xmax / ymax;  pts[1].y = xmax / ymax;  pts[1].z = 0.0;
        ppolyline3(2, pts);

        pts[0].x = 0.0        ;  pts[0].y = xmax / ymax;  pts[0].z = 0.0;
        pts[1].x = xmax / ymax;  pts[1].y = 0.0        ;  pts[1].z = 0.0;
        ppolyline3(2, pts);

    /* Initialize point data for the top box. */
        pts[0].x = 0.0        ;  pts[0].y = xmax / ymax;  pts[0].z = 0.0;
        pts[1].x = 0.0        ;  pts[1].y = 1.0        ;  pts[1].z = 0.0;
        pts[2].x = xmax / ymax;  pts[2].y = 1.0        ;  pts[2].z = 0.0;
        pts[3].x = xmax / ymax;  pts[3].y = xmax / ymax;  pts[3].z = 0.0;
        pts[4].x = 0.0        ;  pts[4].y = xmax / ymax;  pts[4].z = 0.0;

    /* SET POLYLINE COLOUR INDEX to green and add the POLYLINE. */
        psetlinecolourind(3);
        ppolyline3(5, pts);

    /* Reset the point data and insert more POLYLINE elements. */
        pts[0].x = 0.0        ;  pts[0].y = xmax / ymax;  pts[0].z = 0.0;
        pts[1].x = xmax / ymax;  pts[1].y = 1.0        ;  pts[1].z = 0.0;
        ppolyline3(2, pts);

        pts[0].x = 0.0        ;  pts[0].y = 1.0        ;  pts[0].z = 0.0;
        pts[1].x = xmax / ymax;  pts[1].y = xmax / ymax;  pts[1].z = 0.0;
        ppolyline3(2, pts);
    }

/* Close the structure and post it to the workstation for display. */

    pclosestruct();
    ppoststruct(WS1, 1, 1.0);
    sleep(5);
```

```
/* Close the workstation and PHIGS */

    pclosews(WS1);
    pclosephigs();
}
```

pickit.c

A complex picking example program.

```
#ifndef lint
static char sccsid[] = "@(#)pickit.c 2.1 88/06/02 Copyr 1987 Sun Micro";
#endif

/*
 * Copyright (c) 1988 by Sun Microsystems, Inc.
 */

/* Example program showing how to use a pick input device.  This program
   demonstrates:

    - How to initialize and enable a pick input device.
    - How to collect event mode PHIGS input using either sun_canvas
      or sun_tool workstation types.
    - How to set a pick device's detectability filter.
    - How to use name sets to allow selection and highlighting of
      picked output primitives.
    - How to use SunView and PHIGS together.
        - With both canvas and tool workstation types.
    - How to use the SunPHIGS input event notification for canvas
      workstation types.
    - How to write a program that determines the workstation model
      selected at link time (tool or canvas) and opens the correct
      workstation type.

    This program can be linked using either the tool or the canvas model,
    i.e, either with or without the library lphigs.a.  It determines the
    model used and acts accordingly.

    The output primitives are all in one structure.  Each primitive has a
    separate pick id associated with it; this pick id is also used as the
    name set id.  Each primitive's name is added to the name set before
    the primitive is defined and removed immediately afterwards.  This
    limits the scope of that name to only the elements associated with that
    primitive.

    To collect input, the canvas model registers an input event notification
    function with SunPHIGS.  This function is called by SunPHIGS whenever
    new input is placed on the SunPHIGS input event queue.  The tool model
    uses a timer function that it registers with the SunView notifier.  (This
    is necessary since the application (this program) is using the SunView
    event driven model and needs the flow of control passed back to it to
    query the SunPHIGS event queue.)  When either of these call-back
    functions is called, the SunPHIGS functions AWAIT EVENT and GET PICK are
    used to retrieve any pick input from the SunPHIGS input queue.

    MAIN is at the bottom of the file.
*/

#include <malloc.h>
#include <phigs/phigs.h>
#include <suntool/canvas.h>
#include <suntool/panel.h>
```

```
#define FRAME_X     100
#define FRAME_Y     50

#define WS_X        100
#define WS_Y        150
#define WS_WIDTH    200
#define WS_HEIGHT   200

#define OUTPUT_WS   1

#define NUM_NAMES   20

#define BLACK       0
#define WHITE       1
#define RED     2
#define GREEN       3
#define BLUE        4
#define YELLOW      5
#define CYAN        6
#define MAGENTA     7

/* Primitive pick and name set ids. */
#define LINE        1
#define FILL_AREA   2
#define FILL_AREA_SET   3
#define TEXT        4
#define MARKERS     5

static Pwstype      cur_wst;    /* for tool or canvas */
static Panel        cmd_panel;

/* Control buttons for the Button Panel. */
typedef struct {
    char    *name;
    void    (*func)();
} Control_button;

extern void redraw_all();
extern void clear_highlight();

static Control_button   ctl_btns[] = {
    "redraw",       redraw_all,
    "clear",        clear_highlight,
    NULL,       NULL
};


static void
redraw_all( item )
    Panel_item  item;
{
    predrawallstruct( OUTPUT_WS, PALWAYS);
}

static void
clear_highlight( item )
    Panel_item  item;
{
    Pintlst infilt, exfilt;

    exfilt.number = 0;
    infilt.number = 0;
    /* All highlighted primitives will be redrawn in their original colours. */
    psethilightfilter( OUTPUT_WS, &infilt, &exfilt );
}

static void
select_primitive( path )
    Ppickpath   *path;
{
    Pint    innames[NUM_NAMES], exnames[NUM_NAMES];
    Pint    err, insize, exsize;
    Pintlst infilt, exfilt;

    /* Get the current highlight filter. */
    infilt.integers = innames;
    exfilt.integers = exnames;
    pinqhilightfilter( OUTPUT_WS, NUM_NAMES, 0, NUM_NAMES, 0, &err,
    &infilt, &insize, &exfilt, &exsize );

    /* Add this primitive to the highlight filter's inclusion set. */
    infilt.integers[infilt.number++] = path->pick_path[0].pick_id;
    psethilightfilter( OUTPUT_WS, &infilt, &exfilt );
}

static void
await_event()
{
    Pevent      event;
    Ppickpathel     path[1];
    Ppick       pick;

    do {
    pawaitevent( 0.0, &event);
    switch ( event.class ) {
        case PI_PICK:
        /* Since the device was initialized to return paths
         * BOTTOM_FIRST, we only need to look at the first element
         * of the returned path to determine the primitive picked.
         */
        pick.pick_path.pick_path = path;
        pgetpick( 1, &pick);
        if ( pick.status == PP_OK )
            /* Highlight picked primitive in
             * gse-defined highlight colour.
             */
            select_primitive( &pick.pick_path );
        break;
    }
    } while ( event.class != PI_NONE );
}

static Panel
create_cmd_panel( frame )
    Frame   frame;
{
    Panel           panel;
    register Control_button *btn;
    register int        max_length = 0;

    panel = window_create( frame, PANEL,
    PANEL_LABEL_BOLD, FALSE,
        0);

    for ( btn = ctl_btns; btn->name; btn++) {
       if ( strlen(btn->name) > max_length)
       max_length = strlen(btn->name);
    }

    for ( btn = ctl_btns; btn->name; btn++) {
    panel_create_item( panel, PANEL_BUTTON,
            PANEL_LABEL_IMAGE,
            panel_button_image( panel, btn->name, max_length, NULL),
            PANEL_NOTIFY_PROC, btn->func,
        0);
    }

    window_fit( panel );
    return panel;
}

static void
build_css()
{
    /* Define all the primitives about the origin then position them with
     * local transforms.
     */
    static Ppoint3  square[] = {    0.0,0.0,0.0,
                    0.2,0.0,0.0,
                    0.2,0.2,0.0,
                    0.0,0.2,0.0,
                    0.0,0.0,0.0
                    };
    static Ppointlst3   fas = { 4, square };
    static Ppoint3  tpt = {0.0,0.0,0.0};
    static Pvector3 tdir[2] = { {1.0,0.0,0.0}, {0.0,1.0,0.0} };

    Pint    names[20], err;
    Pintlst nset;
    Pvector3    vtrans; /* location of lower left corner of square. */
    Pmatrix3    trans;
    Pgserec gserec;

    nset.number = 1;
    nset.integers = names;

    /* Add the primitives and bracket each with add/remove name from set.
     * The pick ids correspond to the primitive's name in the set.
     */
    popenstruct( 1 );
    /* Colour to be used to highlight primitives in select_primitive */
    gserec.ugsel_datarec.highlight_colour = YELLOW;
    pgse( PUGSE_HIGHLIGHT_COLOUR_INDEX, &gserec );
```

```
names[0] = LINE;
paddnameset( &nset );
vtrans.x = 0.1; vtrans.y = 0.6; vtrans.z = 0.0;
ptranslate3( &vtrans, &err, trans );
psetlocaltran3( trans, PREPLACE );
psetlinecolourind( WHITE );
psetpickid( LINE );
ppolyline3( 5, square );
premovenameset( &nset );

/* The following fill area and fill area set primitives use the same
 * data as the polyline primitive, but only need the first four points,
 * since fill areas are implicitly closed.
 */
names[0] = FILL_AREA;
paddnameset( &nset );
vtrans.x = 0.6; vtrans.y = 0.6; vtrans.z = 0.0;
ptranslate3( &vtrans, &err, trans );
psetlocaltran3( trans, PREPLACE );
psetintstyle( PSOLID );
psetintcolourind( MAGENTA );
psetpickid( FILL_AREA );
pfillarea3( 4, square );
premovenameset( &nset );

names[0] = FILL_AREA_SET;
paddnameset( &nset );
vtrans.x = 0.1; vtrans.y = 0.1; vtrans.z = 0.0;
ptranslate3( &vtrans, &err, trans );
psetlocaltran3( trans, PREPLACE );
psetintstyle( PSOLID );
psetedgeflag( PEDGE_ON );
psetedgecolourind( GREEN );
psetintcolourind( RED );
psetpickid( FILL_AREA_SET );
pfillareaset3( 1, &fas );
premovenameset( &nset );

names[0] = TEXT;
paddnameset( &nset );
vtrans.x = 0.3; vtrans.y = 0.4; vtrans.z = 0.0;
ptranslate3( &vtrans, &err, trans );
psetlocaltran3( trans, PREPLACE );
psettextcolourind( CYAN );
psetcharheight( 0.05 );
psetpickid( TEXT );
ptext3( &tpt, tdir, "Some Text" );
premovenameset( &nset );

names[0] = MARKERS;
paddnameset( &nset );
vtrans.x = 0.6; vtrans.y = 0.1; vtrans.z = 0.0;
ptranslate3( &vtrans, &err, trans );
psetlocaltran3( trans, PREPLACE );
psetmarkercolourind( BLUE );
psetpickid( MARKERS );
ppolymarker3( 4, square );
premovenameset( &nset );
```

```
    pclosestruct();
}

static void
open_phigs()
{
    char           *buf = NULL;
    Pint            i, err, size;
    Pwstypelst          list;
    Pwstype        *wst;
    Phigs_base_name n;
    Pescapein           esc_rec;

    static struct itimerval itval = { 1, 0, 1, 0};

    /* Open PHIGS, and use the file "my_errors" for the SunPHIGS error file. */
    popenphigs( "my_errors", PDEFAULT_MEM_SIZE);

    /* Report errors synchronously. */
    esc_rec.uescl_idatarec.sync_on = PERRSYNC_ON;
    pescape( PUESC_ERRSYNC, &esc_rec, (Pescapeout*)NULL);

    /* The list of available workstation types indicates which model was
     * selected at link time.  We look through the list to see what types
     * of workstations we can use from this application.  As soon as we
     * see one that tells us how the application was linked, we create one
     * of that type and return.
     */

    /* Get the buffer size needed. */
    pinqwstypes( 0, &err, (Pchar*)NULL, &list, &size);
    if ( !err) {
    if ( size > 0 )
        buf = malloc( (unsigned)size);
    pinqwstypes( size, &err, buf, &list, &size);
    for ( i = 0, wst = list.ws_types; i < list.number; i++, wst++) {
        n = (Phigs_base_name)phigs_ws_type_get( *wst, PHIGS_BASE_NAME);

        if ( n == PHIGS_SUN_TOOL) {
        cur_wst = phigs_ws_type_create( phigs_ws_type_sun_tool,
            PHIGS_TOOL_WIDTH, WS_WIDTH, PHIGS_TOOL_HEIGHT, WS_HEIGHT,
            PHIGS_TOOL_X, WS_X, PHIGS_TOOL_Y, WS_Y,
            PHIGS_TEXTSW, PHIGS_NONE,
            PHIGS_TOOL_SHOW_LABEL, FALSE,
            0);

        /* Set up a timer so we can poll the event queue from our
         * SunView application that uses a PHIGS_SUN_TOOL workstation.
         */
        notify_set_itimer_func( &cur_wst, await_event, ITIMER_REAL,
           &itval, NULL);
        break;

        } else if ( n == PHIGS_SUN_CANVAS) {
        cur_wst = phigs_ws_type_create( phigs_ws_type_sun_canvas,
            PHIGS_TEXTSW, PHIGS_NONE,
            0);
        /* Register our SunPHIGS event callback.  (This feature
         * is only available for PHIGS_SUN_CANVAS workstation
         * types.)
         */
        esc_rec.uesc2_idatarec.notify_proc = await_event;
        pescape( PUESC_INPUT_NOTIFY_PROC, &esc_rec, (Pescapeout*)NULL);
        break;
        }
    }
    if ( buf )
        free(buf);
    }
}

void
open_workstation( ctl_frame )
    Frame   ctl_frame;
{
    Canvas  canvas;
    Frame   pframe;
    Pint    wsid = OUTPUT_WS;

    switch ( (int)phigs_ws_type_get( cur_wst, PHIGS_BASE_NAME)) {
    case PHIGS_SUN_TOOL:
        window_set( ctl_frame, FRAME_LABEL, "Tool Workstation", 0);
        popenws( wsid, (Pconnid)NULL, cur_wst);
        break;

    case PHIGS_SUN_CANVAS:
        window_set( ctl_frame, FRAME_LABEL, "Canvas Workstation", 0);
        pframe = window_create( ctl_frame, FRAME,
        WIN_SHOW, TRUE,
        WIN_X, WS_X - FRAME_X, WIN_Y, WS_X - FRAME_Y,
        FRAME_SHOW_LABEL, FALSE,
        0);
        canvas = window_create( pframe, CANVAS,
        WIN_X, 0,
        WIN_WIDTH, WS_WIDTH, WIN_HEIGHT, WS_HEIGHT,
        0);
        popenws( wsid, (Pconnid)canvas, cur_wst);
        window_fit(pframe);
    }
}

static void
init_pick_device( dev_id )
    Pint    dev_id;
{
    Pint    names[NUM_NAMES];
    Pintlst infilt, exfilt;
    Ppickrec    rec;
    Plimit3 ev; /* echo volume */
    Ppoint3 *aperture;
    Ppick   init_pick;

    /* Make everything detectable by this device. */
    infilt.number = 5;
    infilt.integers = names;
    names[0] = LINE;
    names[1] = FILL_AREA;
    names[2] = FILL_AREA_SET;
    names[3] = TEXT;
    names[4] = MARKERS;
    exfilt.number = 0;
    psetpickfilter( OUTPUT_WS, dev_id, &infilt, &exfilt );

    /* Initialize and enable the device.  Use prompt/echo type 2 (highlight
     * all prims with same pick id of picked primitive) and no initial pick.
     * Set the path order to BOTTOM_FIRST so that we only have to look at
     * the first element in a returned pick path.  The bottom-most element
     * is the one that contains the primitive's pick id (in this application).
     */

    /* Two 0.1 second blinks in yellow. */
    rec.pickpet2_datarec.highlight_colour = YELLOW;
    rec.pickpet2_datarec.highlight_count = 2;
    rec.pickpet2_datarec.highlight_duration = 0.1;

    /* Set echo volume to the entire ws viewport. */
    ev.xmin = ev.ymin = ev.zmin = 0.0;
    ev.xmax = ev.ymax = ev.zmax = 1.0;

    /* Set the pick aperture to 5% of NPC in all dimensions. */
    aperture = &rec.pickpet2_datarec.aperture_size;
    aperture->x = aperture->y = aperture->z = 0.05;

    init_pick.status = PP_NOPICK;
    init_pick.pick_path.depth = 0;
    pinitpick3( OUTPUT_WS, dev_id, init_pick.status, &init_pick.pick_path,
    2, &ev, &rec, PBOTTOM_FIRST);
    psetpickmode( OUTPUT_WS, dev_id, PEVENT, PES_ECHO );
}


main( argc, argv )
    int     argc;
    char    *argv[];
{
    Frame   base_frame;

    /* Open PHIGS and create a structure containing the primitives that
     * will be used to demonstrate picking.
     */
    open_phigs();
    build_css();

    /* Create the control frame. */
    base_frame = window_create( NULL, FRAME,
    WIN_X, FRAME_X, WIN_Y, FRAME_Y,
    FRAME_NO_CONFIRM, TRUE,
    0);
    cmd_panel = create_cmd_panel( base_frame );

    /* Open a workstation of the type appropriate for the model selected
     * at link time, initialize pick device 1 (cursor and left mouse button),
     * and post (display the contents of) the structure created by build_css.
     */
    open_workstation( base_frame );
    init_pick_device( 1 );
    ppoststruct( OUTPUT_WS, 1, 1.0 );

    /* Pass control to SunView. */
    window_fit( base_frame );
    window_main_loop( base_frame );

    /* Cleanup. */
    pclosews( OUTPUT_WS );
    pclosephigs();
}
```

rspheres.c

Demonstrates the transformation pipeline and logical input devices.

```
#ifndef lint
static char sccsid[] = "@(#)rspheres.c 2.3 89/01/04 Copyr 1988 Sun Micro";
#endif

/*
 * Copyright (c) 1988 by Sun Microsystems, Inc.
 */


/* Example program allowing a user to interactively modify viewing
 * parameters using a SunPHIGS valuator device, and see the effect
 * on the displayed image (a sphere bouncing inside a cube).
 *
 * Also demonstrates the use of a SunPHIGS choice device, allowing the
 * user to add additional spheres, modify their attributes, etc.
 */

#include <phigs/phigs.h>
#include <ctype.h>
#include <math.h>

extern int  facet_sphere();

#define MAX_SPHERES 100
#define SPHERE      1
#define LOCATIONS   2
#define ROOT        3

#define LINTSTYLE   1
#define LEDGEFLAG   2

#define BLACK       0
#define WHITE       1
#define RED         2
#define GREEN       3
#define BLUE        4
#define YELLOW      5

#define QUIT        -1

#define WC_MIN      -5.0
#define WC_MAX       5.0

#define TIME_INC    1.0
#define VELOCITY    0.2

#define MAX_RAND    2147483647

#ifndef MIN
#define MIN( a, b)  (((a) < (b)) ? (a) : (b))
#endif
#ifndef MAX
#define MAX( a, b)  (((a) > (b)) ? (a) : (b))
#endif
```

```
static Pchar        *strings[] = {
        "EMPTY",
        "HOLLOW",
        "SOLID",
        "PATTERN",
        "HATCH",
        " ",
        "EDGE_ON",
        "EDGE_OFF",
        " ",
        "PARALLEL",
        "PERSPECTIVE",
        " ",
        "ADD SPHERE",
        "REMOVE SPHERE",
        " ",
        "HIDDEN SURF ON",
        "HIDDEN SURF OFF",
        " ",
        "QUIT"
};

static num_strings = sizeof(strings)/sizeof(Pchar*);

static Pmatrix3     identity = {    1.0, 0.0, 0.0, 0.0,
                    0.0, 1.0, 0.0, 0.0,
                    0.0, 0.0, 1.0, 0.0,
                    0.0, 0.0, 0.0, 1.0
            };

/* Sphere velocity and location data. */
typedef struct {
    Pvector3        velocity;
    Pmatrix3        position;
} Sphere_data;

static Sphere_data  sphere_data[MAX_SPHERES];
static int      sphere_count = 0;

/* Radius of spheres. */
static double       radius = 1.0;

/* Viewing parameters */
static Pviewrep3    rep;
static Pviewmapping3    map;
static Ppoint3      vrp = { 0.0, 0.0, 0.0};
static Pvector3     vup = { 0.0, 0.0, 1.0};
static Pvector3     vpn = { 0.866, -0.5, 0.5};


static void
usage( name )
    char    *name;
{
    fprintf( stderr,
    "usage: %s [count] [e on|off] [i e|h|s] [h on|off] [d c|h|n] [?|1]0,
    name);
}

static void
random_velocity( v )
    Pvector3    *v;
{
    extern long random();

    long    Xr, Yr;
    Pfloat  Xv, Yv, Zv, XYv, V2;

    Xv = Yv = Zv = 0.0;
    V2 = VELOCITY * VELOCITY;

    Xr = random();
    Xv = ((float)Xr/(float)MAX_RAND) * (VELOCITY/sqrt(3.0));
    if ( Xr % 2 )
    Xv = - Xv;

    Yr = random();
    Yv = ((float)Yr/(float)MAX_RAND) * sqrt(V2 - Xv * Xv);
    if ( Yr % 2 )
    Yv = - Yv;

    XYv = sqrt(Xv * Xv + Yv * Yv);
    if ( XYv < VELOCITY ) {
    Zv = sqrt(V2 - XYv * XYv);
    if ( random() % 2 )
        Zv = - Zv;
    }

    v->x = Xv;
    v->y = Yv;
    v->z = Zv;
}

static void
add_sphere()
{
    Sphere_data    *data;


    if ( sphere_count < MAX_SPHERES ) {
    data = &sphere_data[sphere_count++];
    memcpy( data->position, identity, sizeof(Pmatrix3));
    random_velocity( &data->velocity );

    pseteditmode( PEDIT_INSERT);
    popenstruct( LOCATIONS);
        psetlocaltran3( data->position, PREPLACE);
        pexecutestruct( SPHERE);
    pclosestruct();
    pseteditmode( PEDIT_REPLACE);
    }
}

static void
remove_sphere()
```

```
{
    if ( sphere_count > 0 ) {
    popenstruct( LOCATIONS);
        pdelelem();
        pdelelem();
    pclosestruct();
    --sphere_count;
    }
}

static void
init_view_mapping()
{
    map.proj = PPERSPECTIVE;
    map.prp.z = 6.0 * WC_MAX;
    map.window.xmin = WC_MIN; map.window.xmax = WC_MAX;
    map.window.ymin = WC_MIN; map.window.ymax = WC_MAX;
    map.front_plane = 2.0 * WC_MAX;
    map.back_plane =  2.0 * WC_MIN;
    map.view_plane =  0.4 * map.prp.z;
    map.prp.x = (map.window.xmin + map.window.xmax) / 2.0;
    map.prp.y = (map.window.ymin + map.window.ymax) / 2.0;
    map.viewport.xmin = 0.0; map.viewport.xmax = 1.0;
    map.viewport.ymin = 0.0; map.viewport.ymax = 1.0;
    map.viewport.zmin = 0.0; map.viewport.zmax = 1.0;
}

static void
eval_view_rep( rep)
    Pviewrep3       *rep;
{
    Pint        err;

    pevalviewmappingmatrix3( &map, &err, rep->mapping_matrix);
    if ( err) {
    fprintf( stderr, "Error from eval mapping %d0, err);
    }

    pevalvieworientationmatrix3( &vrp, &vpn, &vup, &err,
    rep->orientation_matrix);
    if ( err) {
    fprintf( stderr, "Error from eval orientation %d0, err);
    }
}

static void
set_view_rep()
{
    eval_view_rep( &rep);
    rep.clip_limit = map.viewport;
    rep.clip_xy = rep.clip_back = rep.clip_front = PNOCLIP;
    psetviewrep3( 1, 1, &rep);
}

static void
build_sphere()
{
    Ppointlst3      *facets;
    int         num_facets, num_lat = 6, num_long = 8;
    register int    i;

    num_facets = facet_sphere( radius, num_lat, num_long, &facets);
    popenstruct( SPHERE);
    for ( i = 0; i < num_facets; i++ ) {
    psetintcolourind( (i % 6) + 2 );
    pfillareaset3( 1, &facets[i]);
    }
    pclosestruct();
    free(facets);
}

static void
build_box()
{
    Ppoint3 pts[5];

    psetlinecolourind(RED);
    /* top */
    pts[0].z = pts[1].z = pts[2].z = pts[3].z = pts[4].z = WC_MIN;
    pts[0].x = WC_MIN; pts[0].y = WC_MIN;
    pts[1].x = WC_MAX; pts[1].y = WC_MIN;
    pts[2].x = WC_MAX; pts[2].y = WC_MAX;
    pts[3].x = WC_MIN; pts[3].y = WC_MAX;
    pts[4].x = WC_MIN; pts[4].y = WC_MIN;
    ppolyline3( 5, pts);
    /*  bottom */
    pts[0].z = pts[1].z = pts[2].z = pts[3].z = pts[4].z = WC_MAX;
    ppolyline3( 5, pts);
    /* corners */
    pts[0].z = WC_MIN; pts[1].z = WC_MAX;
    pts[0].x = pts[1].x = WC_MIN; pts[0].y = pts[1].y = WC_MIN;
    ppolyline3( 2, pts);
    pts[0].x = pts[1].x = WC_MAX; pts[0].y = pts[1].y = WC_MIN;
    ppolyline3( 2, pts);
    pts[0].x = pts[1].x = WC_MAX; pts[0].y = pts[1].y = WC_MAX;
    ppolyline3( 2, pts);
    pts[0].x = pts[1].x = WC_MIN; pts[0].y = pts[1].y = WC_MAX;
    ppolyline3( 2, pts);
}

static void
build_css( edge_flag, int_style )
    Pedgef  edge_flag;
    Pinterstyle int_style;
{
    Ppoint3 axes_origin, axes_length;
    Pint    axes_color[3];

    axes_origin.x = axes_origin.y = axes_origin.z = 0.0;
    axes_length.x = axes_length.y = axes_length.z = 0.5;
    axes_color[0] = RED; axes_color[1] = GREEN; axes_color[2] = BLUE;

    popenstruct( ROOT);
    psetviewind( 1);
    psethlhsrid(PHIGS_HLHSR_ID_ZBUFF);
    axes( &axes_origin, &axes_length, axes_color);
    psetintcolourind(YELLOW);
    psetedgecolourind(GREEN);
    plabel(LEDGEFLAG);
    psetedgeflag( edge_flag);
    plabel(LINTSTYLE);
    psetintstyle( int_style);
    pexecutestruct( LOCATIONS);
    build_box();
    pclosestruct();
    build_sphere();
}

#define NEW_RI( _v, _dt, _r )
    {
    (_r) += (_v) * (_dt);
    if ( (_r) >= WC_MAX ) {
        (_r) = 2.0 * WC_MAX - (_r);
        (_v) = -(_v);
    } else if ( (_r) <= WC_MIN ) {
        (_r) = 2.0 * WC_MIN - (_r);
        (_v) = -(_v);
    }
    }

static void
move_spheres()
{
    register Sphere_data    *data;
    register int        i;

    popenstruct( LOCATIONS);
    psetelemptr(1);
    for ( i = 0; i < sphere_count; i++ ) {
    data = &sphere_data[i];
    NEW_RI( data->velocity.x, TIME_INC, data->position[0][3] )
    NEW_RI( data->velocity.y, TIME_INC, data->position[1][3] )
    NEW_RI( data->velocity.z, TIME_INC, data->position[2][3] )
    psetlocaltran3( data->position, PREPLACE);
    poffsetelemptr(2);
    }
    pclosestruct();
}

static void
init_input()
{
    Pchoicerec      crec;
    Pvalrec     vrec;
    Plimit      ea;

    crec.choicepet3_datarec.number = num_strings;
    crec.choicepet3_datarec.strings = strings;
    ea.xmin = 0.0; ea.xmax = 1.0; ea.ymin = 0.0; ea.ymax = 1.0;
    pinitchoice( 1, 3, PCH_OK, 13, 3, &ea, &crec);

    vrec.uvalpet1_datarec.low = -1.0;
    vrec.uvalpet1_datarec.high = 1.0;
    vrec.uvalpet1_datarec.length = 200.0;
    strcpy( vrec.uvalpet1_datarec.format, "%8.3f");
    strcpy( vrec.uvalpet1_datarec.label, "vpn.x     ");
    pinitval( 1, 1, vpn.x, -1, &ea, &vrec);
    strcpy( vrec.uvalpet1_datarec.label, "vpn.y     ");
    pinitval( 1, 2, vpn.y, -1, &ea, &vrec);
    strcpy( vrec.uvalpet1_datarec.label, "vpn.z     ");
    pinitval( 1, 3, vpn.z, -1, &ea, &vrec);
    vrec.uvalpet1_datarec.low = -map.prp.z;
    vrec.uvalpet1_datarec.high = map.prp.z;
    strcpy( vrec.uvalpet1_datarec.label, "view plane");
    pinitval( 1, 4, map.view_plane, -1, &ea, &vrec);

    psetchoicemode( 1, 3, PEVENT, PES_ECHO);
    psetvalmode( 1, 1, PEVENT, PES_ECHO);
    psetvalmode( 1, 2, PEVENT, PES_ECHO);
    psetvalmode( 1, 3, PEVENT, PES_ECHO);
    psetvalmode( 1, 4, PEVENT, PES_ECHO);
}

static void
valuator_event( ev, val )
    Pevent  *ev;
    Pfloat  val;
{
    switch ( ev->dev ) {
    case 1: vpn.x = val; break;
    case 2: vpn.y = val; break;
    case 3: vpn.z = val; break;
    case 4: map.view_plane = val; break;

    default: /* ? */
        fprintf( stderr, "Unknown valuator device %d0, ev->dev);
        return;
    }
    set_view_rep();
}

static int
choice_event( ev, ch )
    Pevent  *ev;
    Pchoice *ch;
{
    int     status = 0;
    Pinterstyle int_style;

    switch ( ch->choice ) {
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
        switch ( ch->choice ) {
        case 1: int_style = PEMPTY; break;
        case 2: int_style = PHOLLOW; break;
        case 3: int_style = PSOLID; break;
        case 4: int_style = PPATTERN; break;
        case 5: int_style = PHATCH; break;
        }
        popenstruct(ROOT);
        psetelemptr(0);
        psetelemptrlabel(LINTSTYLE);
        poffsetelemptr(1);
        psetintstyle(int_style);
        pclosestruct();
        break;

    case 7:
    case 8:
        popenstruct(ROOT);
        psetelemptr(0);
        psetelemptrlabel(LEDGEFLAG);
        poffsetelemptr(1);
        psetedgeflag( ch->choice == 7 ? PEDGE_ON : PEDGE_OFF);
        pclosestruct();
        break;

    case 10:
        map.proj = PPARALLEL;
        set_view_rep();
        break;

    case 11:
        map.proj = PPERSPECTIVE;
        set_view_rep();
        break;

    case 13:
        add_sphere();
        break;
    case 14:
        remove_sphere();
        break;
    case 16:
        psethlhsrmode(1, PHIGS_HLHSR_MODE_ZBUFF);
        break;
    case 17:
        psethlhsrmode(1, PHIGS_HLHSR_MODE_NONE);
        break;
    case 19:
        status = QUIT;
        break;
    }

    return status;
}

static int
check_input()
{
    int     status = 0;
    Pevent  event;
    Pchoice choice;
    Pfloat  val;

    do {
    pawaitevent( 0.02, &event);
    switch ( event.class ) {
        case PI_CHOICE:
        pgetchoice( &choice);
        if ( choice.status == PCH_OK)
            status = choice_event( &event, &choice);
        break;

        case PI_VALUATOR:
        pgetval( &val);
        valuator_event( &event, val);
        break;
    }
    } while ( event.class != PI_NONE );

    return status;
}

main( argc, argv )
    int     argc;
    char    *argv[];
{
    int         i, num_spheres = 1;
    char        *arg, *progname = argv[0];
    Pedgef      edge_flag = PEDGE_OFF;
    Pinterstyle     int_style = PSOLID;
    Pwstype     wst;
    Pint        hidden_surf = PHIGS_HLHSR_MODE_NONE;
    Phigs_dbl_buff  dbl_buff = PHIGS_DBL_CMAP;

    while ( arg = *++argv ) {
    if ( isdigit(*arg) )
        num_spheres = MIN( MAX_SPHERES, atoi(arg));

    else {
        while ( *arg ) {
        switch ( *arg++ ) {
            case 'e':
            edge_flag =
                strcmp( *++argv, "on") ? PEDGE_OFF : PEDGE_ON;
                break;
            case 'h':
            hidden_surf = strcmp( *++argv, "on")
                ? PHIGS_HLHSR_MODE_NONE : PHIGS_HLHSR_MODE_ZBUFF;
            break;
            case 'i':
            switch ( *(*++argv) ) {
                case 'e':
                int_style = PEMPTY; break;
                case 'h':
                int_style = PHOLLOW; break;
                case 's':
                int_style = PSOLID; break;
            }
            break;
            case 'd':
            switch ( *(*++argv) ) {
                case 'c':
                dbl_buff = PHIGS_DBL_CMAP; break;
            case 'h':
                dbl_buff = PHIGS_DBL_HW; break;
            case 'n':
                dbl_buff = PHIGS_DBL_NONE; break;
        }
        break;

        case '-':
        /* Ignore unneeded flag denotation. */
        break;
        case '?':
        case 'l':
        usage( progname);
        exit(2);
        break;
        default:
        fprintf( stderr, "%s: Unrecognized option.0,
            progname);
        usage( progname);
        exit(2);
        break;
    }
    }

}
}

/* Set up initial viewing parameters. */
init_view_mapping();

/* Open PHIGS and create the display hierarchy. */
popenphigs((Pchar*)NULL, PDEFAULT_MEM_SIZE);
build_css( edge_flag, int_style);

/* Create a workstation type with the description table values we want
 * and open a workstation.
 * Use colour map double buffering unless told otherwise on the command
 * line (-d argument).
 */
wst = phigs_ws_type_create( phigs_ws_type_sun_tool,
PHIGS_COLOR_TABLE_SIZE, 8,
PHIGS_DOUBLE_BUFFER, dbl_buff,
PHIGS_TOOL_X, 20, PHIGS_TOOL_Y, 20,
PHIGS_TOOL_HEIGHT, 600, PHIGS_TOOL_WIDTH, 600,
PHIGS_VAL_PANEL_X, 650, PHIGS_VAL_PANEL_Y, 25,
PHIGS_TOOL_SHOW_LABEL, FALSE,
PHIGS_TEXTSW, PHIGS_NONE,
0);
popenws( 1, (Pconnid)NULL, wst);

/* Set the deferral state so that the screen is not updated when
 * the hierarchy is posted, then post the hierarchy.
 */
psetdisplayupdatest(1, PWAIT, PNIVE);
psethlhsrmode(1, hidden_surf);
set_view_rep();
init_input();
for ( i = 0; i < num_spheres; i++ )
```

```
    add_sphere();
    ppoststruct( 1, ROOT, 1.0);
    /* Now display the hierarchy. */
    psetdisplayupdatest(1, PASAP, PNIVE);

    /* Set edit mode to REPLACE to allow replacement of primitive attribute
     * elements as they are changed by the user. Watch for input, and
     * redisplay the image after making the appropriate changes to primitive
     * attributes, viewing parameters, etc.
     */
    pseteditmode( PEDIT_REPLACE);
    for (;;) {
    psetdisplayupdatest(1, PWAIT, PNIVE);
    if ( check_input() != QUIT ) {
        move_spheres();
        psetdisplayupdatest(1, PASAP, PNIVE);
    } else {
        pclosews(1);
        pclosephigs();
        break;
    }
    }
}
```

spheres.c

Called by rspheres.c to form a sphere with triangles.

```
#ifndef lint
static char sccsid[] = "@(#)spheres.c 2.1 88/06/02 Copyr 1988 Sun Micro";
#endif

/*
 * Copyright (c) 1988 by Sun Microsystems, Inc.
 */

/* tessellate a sphere with triangles

    method: divide into latitute and longitude lines.  This gives
    quadrilateral tesellation, then slash each quad to give triangles.
    Exception: first and last latitude rows are already triangles, with
    the N or S pole as a vertex

*/

#include <phigs/phigs.h>
#include <math.h>
#include <malloc.h>

#define MAXLAT 100  /*max number of horiz and vert. divisions of sphere*/
#define MAXLONG 100

#define BIGNUM 999.0
#define PI 3.14159265358979
#define TORAD(x)    ((x)*PI/180.0)

typedef struct {
    Ppoint3 upleft, upright, downleft, downright;
} Quad;


static int
tesselate_sphere( radius, nlat, nlong, facets )
    double  radius;
    int     nlat, nlong;    /*number of horiz, vert quads*/
    Ppoint3 facets[][3];
{
    int     i, j, lat1, lat2, nfacets = 0;
    double  theta, deltay, deltatheta;
    Ppoint3 p1, p2;
    Ppoint3 npole, spole;
    Quad    (*q)[MAXLONG];

    if ( nlat < 2 || 0 != nlat % 2 ) {
    fprintf( stderr, "num lats must be even and >=2\n");
    return -1;
    }
    if( nlat >= MAXLAT ) {
    fprintf( stderr, "current num lat limit is %d\n");
    return -2;
    }
```

```
if( nlong < 3 ) {
fprintf( stderr, "num long must be >= 3\n");
return -3;
}
if( nlong >= MAXLONG ) {
fprintf( stderr, "current num long limit is %d\n");
return -4;
}

q = (Quad(*)[MAXLONG])malloc( nlat * MAXLONG * sizeof(Quad));
if ( !q )
return 0;

npole.x = 0.0; npole.y = 1.0; npole.z = 0.0;
spole.x = 0.0; spole.y = -1.0; spole.z = 0.0;
deltay = 2.0/nlat;  /*y size of horiz. slices*/
deltatheta = 2.0*PI/nlong;

for(j=0; j<nlong; ++j){
q[0][j].upleft =  npole;
q[0][j].upright.x = BIGNUM; /*this is a triangle, not a quad*/
q[nlat-1][j].downleft = spole;
q[nlat-1][j].downright.x = BIGNUM;
}

p1.x = p2.x = 1.0;
p1.y = p2.y = p1.z = p2.z = 0.0;
lat2 = nlat/2;
lat1 = lat2-1;
theta = 0.0;

for(j=0; j<nlong; ++j){
q[lat1][j].downright = q[lat2][j].upright = p1;
if(j>0)
    q[lat1][j-1].downleft = q[lat2][j-1].upleft = p1;
theta += deltatheta;
p1.x = radius*cos(theta);
p1.z = radius*sin(theta);
}
q[lat1][nlong-1].downleft = q[lat2][nlong-1].upleft = q[lat1][0].downright;

for(;;) {
if( lat1 == 0 )
    break;
--lat1; ++lat2;
p1.y += deltay; p2.y -= deltay;
radius = sqrt(1.0 - p1.y*p1.y);
p1.x = p2.x = radius;
p1.z = p2.z = 0.0;
theta = 0.0;

for(j=0; j<nlong; ++j){
    q[lat1][j].downright = q[lat1+1][j].upright = p1;
    q[lat2][j].upright = q[lat2-1][j].downright = p2;
    if(j>0){
    q[lat1][j-1].downleft = q[lat1+1][j-1].upleft= p1;
    q[lat2][j-1].upleft = q[lat2-1][j-1].downleft= p2;
    }
```

```
        theta += deltatheta;
        p1.x = p2.x = radius*cos(theta);
        p1.z = p2.z = radius*sin(theta);
    }
    q[lat1][nlong-1].downleft
        = q[lat1+1][nlong-1].upleft = q[lat1][0].downright;
    q[lat2][nlong-1].upleft
        = q[lat2-1][nlong-1].downleft = q[lat2][0].upright;
    }

    for( i=0; i < nlat; ++i ) {
    for( j = 0; j < nlong; ++j ) {
        if( q[i][j].upright.x == BIGNUM ) {
        facets[nfacets][0] = q[i][j].downleft;
        facets[nfacets][1] = q[i][j].downright;
        facets[nfacets][2] = q[i][j].upleft;
        ++nfacets;
        } else if( q[i][j].downright.x == BIGNUM ) {
        facets[nfacets][0] = q[i][j].downleft;
        facets[nfacets][1] = q[i][j].upleft;
        facets[nfacets][2] = q[i][j].upright;
        ++nfacets;
        } else {
        facets[nfacets][0] = q[i][j].downleft;
        facets[nfacets][1] = q[i][j].downright;
        facets[nfacets][2] = q[i][j].upleft;
        ++nfacets;
        facets[nfacets][0] = q[i][j].downright;
        facets[nfacets][1] = q[i][j].upleft;
        facets[nfacets][2] = q[i][j].upright;
        ++nfacets;
        }
    }
    }

    free(q);
    return nfacets;
}

int
facet_sphere( radius, num_lat, num_long, facetlist )
    float       radius;
    int         num_lat;
    int         num_long;
    Ppointlst3      **facetlist;
{
    Ppoint3     (*facets)[3];
    int         num_facets, farray_size, flist_size;
    char        *buf;
    register int    i;
    register Ppointlst3 *cfp;

    num_facets = (2 + 2 * (num_lat - 2)) * num_long;
    farray_size = num_facets * 3 * sizeof(Ppoint3);
    flist_size = num_facets * sizeof(Ppointlst3);
    buf = malloc(flist_size + farray_size);
    if ( buf ) {
    *facetlist = (Ppointlst3*)buf;
    facets = (Ppoint3(*)[3])(buf + flist_size);
    num_facets = tesselate_sphere( radius, num_lat, num_long, facets);
    for ( i = 0, cfp = *facetlist; i < num_facets; i++, cfp++ ) {
        cfp->number = 3;
        cfp->points = facets[i];
    }
    }

    return num_facets;
}

#ifdef STANDALONE

main( argc, argv)
    int     argc;
    char    *argv[];
{
    Ppointlst3      *facets;
    int         num_facets, num_lat = 8, num_long = 8;
    double      radius = 1.0;
    register int    i;

    num_facets = facet_sphere( radius, num_lat, num_long, &facets);
    for ( i = 0; i < num_facets; i++ ) {
    Ppoint3 *pts;

    pts = facets[i].points;
    fprintf( stdout,
        "%d: \t(%f, %f, %f) \n\t(%f, %f, %f)\n\t(%f, %f, %f)\n",
        i,
        pts[0].x, pts[0].y, pts[0].z,
        pts[1].x, pts[1].y, pts[1].z,
        pts[2].x, pts[2].y, pts[2].z);
    }
    free(facets);
}
#endif
```

txattrs.c

Demonstrates various text functions in C.

```
#ifndef lint
static char sccsid[] = "@(#)txattrs.c 2.1 88/06/02 Copyr 1988 Sun Micro";
#endif

/*
 * Copyright (c) 1988 by Sun Microsystems, Inc.
 */

/* Example program showing the application of certain text attributes
 * and the text extent function. The user may interactively modify
 * text attributes and see how they affect the appearance of the text.
 *
 * This program also demonstrates the use of colour map double buffering.
 *
 * This program can be linked using either the tool or the canvas model,
 * i.e, either with or without the library lphigs.a.  It determines the
 * model used and acts accordingly.
 */

#include <phigs/phigs.h>
#include <suntool/canvas.h>
#include <suntool/panel.h>
#include <malloc.h>

/* Worksatation size and position constants. */
#define WS_SIZE 600
#define WS_X    25
#define WS_Y    200

/* Colors */
#define CINDEX_GRAY 1
#define CINDEX_RED  2
#define CINDEX_GREEN    3
#define CINDEX_YELLOW   5
#define CINDEX_CYAN 6

/* World coordinate limits. */
#define WC_MIN  -100.0
#define WC_MAX   100.0

/* Structure element labels */
#define DUMMY_ELEMENT       0
#define TEXT_ELEMENT        1
#define FONT_ELEMENT        2
#define HEIGHT_ELEMENT      3
#define EXPANSION_ELEMENT   4
#define SPACING_ELEMENT     5
#define UP_ELEMENT      6
#define PATH_ELEMENT        7
#define ALIGN_ELEMENT       8
#define EXTENT_ELEMENT      9
#define CATPOINT_ELEMENT    10

/* Definition of the text local coordinate system. */
```

```
static Ppoint3      loc = { 0.0, 0.0, 0.0};
static Pvector3     dir[2]= { 1.0, 0.0, 0.0,  0.0, 1.0, 0.0};

/* The workstation type used. */
static Pwstype      cur_wst;

/* Current attribute values and default assignments. */
static Pchar        cur_text[100] = "SunPHIGS";
static Pfloat       cur_expansion = 1.0;
static Pfloat       cur_spacing = 0.0;
static Pvector      cur_up = { 0.0, 1.0};
static Pfloat       cur_height = 10.0;
static Pint     cur_font = 1;
static Ptxpath      cur_path = PTP_RIGHT;
static Ptxalign     cur_align = { PAH_NORMAL, PAV_NORMAL};

/* Color rep */
static Pcobundl     gray = { 0.5, 0.5, 0.5};

static void
init_view_mapping()
{
    /* Set the view mapping to display all of WC. */

    Pviewmapping3   map;
    Pviewrep3       rep;
    static Ppoint3  vrp = { 0.0, 0.0, 0.0};
    static Pvector3 vup = { 0.0, 1.0, 0.0};
    static Pvector3 vpn = { 0.0, 0.0, 1.0};
    Pint        err;

    map.proj = PPARALLEL;
    map.prp.z = 10.0;
    map.window.xmin = WC_MIN; map.window.xmax = WC_MAX;
    map.window.ymin = WC_MIN; map.window.ymax = WC_MAX;
    map.front_plane = WC_MAX;
    map.back_plane =  WC_MIN;
    map.view_plane =  0.0;
    map.prp.x = (map.window.xmin + map.window.xmax) / 2.0;
    map.prp.y = (map.window.ymin + map.window.ymax) / 2.0;
    map.viewport.xmin = 0.0; map.viewport.xmax = 1.0;
    map.viewport.ymin = 0.0; map.viewport.ymax = 1.0;
    map.viewport.zmin = 0.0; map.viewport.zmax = 1.0;

    pevalviewmappingmatrix3( &map, &err, rep.mapping_matrix);
    if ( err) {
    fprintf( stderr, "Error %d from eval mapping\n", err);
    }

    pevalvieworientationmatrix3( &vrp, &vpn, &vup, &err,
    rep.orientation_matrix);
    if ( err) {
    fprintf( stderr, "Error %d from eval orientation\n", err);
    }

    rep.clip_limit = map.viewport;
    rep.clip_xy = rep.clip_back = rep.clip_front = PNOCLIP;
    psetviewrep3( 1, 1, &rep);
```

```
}

static void
set_elem_ptr_to( label )
    Pint    label;
{
    /* Find the specified label in the open structure and set the element
     * pointer to the structure element immediately following the label.
     */
    psetelemptr(0);
    psetelemptrlabel(label);
    poffsetelemptr(1);
}

static void
show_extent()
{
    Pint    err;
    Ppoint  cat_pt, box[5];
    Prect   extent;

    /* Inquire the extent of the sample text on the screen and replace
     * the structure element data defining the text extent box and the
     * concatentation point with the new data.
     */
    pinqtextextent( cur_wst, cur_font, cur_expansion, cur_spacing, cur_height,
        cur_path, cur_align.hor, cur_align.ver, cur_text,
    &err, &extent, &cat_pt);
    if ( !err ) {
    box[0] = box[4] = extent.ll;
    box[1].x = extent.ll.x; box[1].y = extent.ur.y;
    box[2] = extent.ur;
    box[3].x = extent.ur.x; box[3].y = extent.ll.y;
    set_elem_ptr_to( EXTENT_ELEMENT);
    ppolyline( 5, box);

    set_elem_ptr_to( CATPOINT_ELEMENT);
    ppolymarker( 1, &cat_pt);

    } else
    fprintf( stderr, "Error %d from Inquire Text Extent\n", err);

    /* Update the ws to show the changed text structure (this initiates
     * a traversal of all posted structures, rendered into the undisplayed
     * buffer, at the end of which the displayed and undisplayed
     * buffers will be swapped).
     */
    pupdatews( 1, PPERFORM);
}

static void
set_text_string( item )
    Panel_item  item;
{
    /* Replace the structure element defining the text string with the
     * new string entered by the user, update the text extent box, and
     * display the changed image.
     */
    set_elem_ptr_to( TEXT_ELEMENT);
    strcpy( cur_text, (Pchar*)panel_get_value( item));
    ptext3( &loc, dir, cur_text);
    show_extent();
}

static void
set_font( item, val )
    Panel_item      item;
    int         val;
{
    /* Replace the structure element setting the current font with the
     * new font selected by the user, update the text extent box, and
     * display the changed image.
     */
    switch ( val ) {
    case 0: cur_font = PFONT_ROMAN_MONO; break;
    case 1: cur_font = PFONT_ROMAN_SIMPLEX; break;
    case 2: cur_font = PFONT_ROMAN_DUPLEX; break;
    case 3: cur_font = PFONT_ROMAN_COMPLEX; break;
    case 4: cur_font = PFONT_ROMAN_TRIPLEX; break;
    case 5: cur_font = PFONT_ITALIC_COMPLEX; break;
    case 6: cur_font = PFONT_ITALIC_TRIPLEX; break;
    case 7: cur_font = PFONT_GREEK_SIMPLEX; break;
    case 8: cur_font = PFONT_GREEK_COMPLEX; break;
    case 9: cur_font = PFONT_SCRIPT_SIMPLEX; break;
    case 10: cur_font = PFONT_SCRIPT_COMPLEX; break;
    case 11: cur_font = PFONT_CARTOGRAPHIC; break;
    case 12: cur_font = PFONT_SYMBOL; break;
    /* See if error conditions handled correctly (Bad Font menu item). */
    default: cur_font = -999; break;
    }

    set_elem_ptr_to( FONT_ELEMENT);
    psettextfont( cur_font);
    show_extent();
}

static void
set_expansion( item, val )
    Panel_item  item;
    int     val;
{
    /* Replace the structure element defining the character expansion with
     * the new value entered by the user, update the text extent box, and
     * display the changed image.
     */
    cur_expansion = 0.01 * (float)val;
    set_elem_ptr_to( EXPANSION_ELEMENT);
    psetcharexpan( cur_expansion);
    show_extent();
}

static void
set_spacing( item, val )
    Panel_item  item;
    int     val;
{
    /* Replace the structure element defining the character spacing with
     * the new value entered by the user, update the text extent box, and
     * display the changed image.
     */
    cur_spacing = 0.01 * (float)val;
    set_elem_ptr_to( SPACING_ELEMENT);
    psetcharspace( cur_spacing);
    show_extent();
}

static void
set_height( item, val )
    Panel_item  item;
    int     val;
{
    /* Replace the structure element defining the character height with
     * the new value entered by the user, update the text extent box, and
     * display the changed image.
     */
    cur_height = (Pfloat)val;
    set_elem_ptr_to( HEIGHT_ELEMENT);
    psetcharheight( cur_height);
    show_extent();
}

static void
set_x_up( item, val )
    Panel_item  item;
    int     val;
{
    /* Replace the structure element defining the x component of the character
     * up vector with the new value entered by the user, update the text
     * extent box, and display the changed image.
     */
    cur_up.x = (Pfloat)val;
    set_elem_ptr_to( UP_ELEMENT);
    psetcharup( &cur_up);
    show_extent();
}

static void
set_y_up( item, val )
    Panel_item  item;
    int     val;
{
    /* Replace the structure element defining the y component of the character
     * up vector with the new value entered by the user, update the text
     * extent box, and display the changed image.
     */
    cur_up.y = (Pfloat)val;
    set_elem_ptr_to( UP_ELEMENT);
    psetcharup( &cur_up);
    show_extent();
}

static void
set_path( item, val )
    Panel_item  item;
    int     val;
{
    /* Replace the structure element defining the text path with the
     * new value entered by the user, update the text extent box, and
     * display the changed image.
     */
    switch ( val ) {
    case 0: cur_path = PTP_RIGHT; break;
    case 1: cur_path = PTP_LEFT; break;
    case 2: cur_path = PTP_UP; break;
    case 3: cur_path = PTP_DOWN; break;
    }

    set_elem_ptr_to( PATH_ELEMENT);
    psettextpath( cur_path);
    show_extent();
}

set_vert_align( item, val )
    Panel_item  item;
    int     val;
{
    /* Replace the structure element defining the text vertical alignment
     * with the new value entered by the user, update the text extent box,
     * and display the changed image.
     */
    switch ( val ) {
    case 0: cur_align.ver = PAV_NORMAL; break;
    case 1: cur_align.ver = PAV_TOP; break;
    case 2: cur_align.ver = PAV_CAP; break;
    case 3: cur_align.ver = PAV_HALF; break;
    case 4: cur_align.ver = PAV_BASE; break;
    case 5: cur_align.ver = PAV_BOTTOM; break;
    }

    set_elem_ptr_to( ALIGN_ELEMENT);
    psettextalign( &cur_align);
    show_extent();
}

set_horiz_align( item, val )
    Panel_item  item;
    int     val;
{
    /* Replace the structure element defining the text horizontal alignment
     * with the new value entered by the user, update the text extent box,
     * and display the changed image.
     */
    switch ( val ) {
    case 0: cur_align.hor = PAH_NORMAL; break;
    case 1: cur_align.hor = PAH_LEFT; break;
    case 2: cur_align.hor = PAH_CENTRE; break;
    case 3: cur_align.hor = PAH_RIGHT; break;
    }

    set_elem_ptr_to( ALIGN_ELEMENT);
    psettextalign( &cur_align);
    show_extent();
```

```
}

static Panel
create_panel( frame)
    Frame   frame;
{
    Panel   panel;

    /* Create the panel with which the user may set text attribute values. */
    panel = window_create( frame, PANEL, PANEL_LABEL_BOLD, TRUE, 0);
    panel_create_item( panel, PANEL_TEXT,
    PANEL_ITEM_Y, ATTR_ROW(0), PANEL_ITEM_X, ATTR_COL(1),
    PANEL_VALUE, cur_text,
    PANEL_LABEL_STRING, "Text string: ",
    PANEL_NOTIFY_PROC, set_text_string,
    PANEL_VALUE_DISPLAY_LENGTH, 30,
    0);
    panel_create_item( panel, PANEL_CYCLE,
    PANEL_ITEM_Y, ATTR_ROW(0), PANEL_ITEM_X, ATTR_COL(50),
    PANEL_LABEL_STRING, "Font:",
    PANEL_CHOICE_STRINGS,
        "Roman Mono",
        "Roman Simplex",
        "Roman Duplex",
        "Roman Complex",
        "Roman Triplex",
        "Italic Complex",
        "Italic Triplex",
        "Greek Simplex",
        "Greek Complex",
        "Script Simplex",
        "Script Complex",
        "Cartographic",
        "Symbol",
        "Bad Font",
        0,
    PANEL_NOTIFY_PROC, set_font,
    0);
    panel_create_item( panel, PANEL_CYCLE,
    PANEL_ITEM_Y, ATTR_ROW(1), PANEL_ITEM_X, ATTR_COL(1),
    PANEL_LABEL_STRING, "Path: ",
    PANEL_CHOICE_STRINGS, "RIGHT", "LEFT", "UP", "DOWN", 0,
    PANEL_NOTIFY_PROC, set_path,
    0);
    panel_create_item( panel, PANEL_CYCLE,
    PANEL_ITEM_Y, ATTR_ROW(1), PANEL_ITEM_X, ATTR_COL(20),
    PANEL_LABEL_STRING, "Horiz Align: ",
    PANEL_CHOICE_STRINGS, "NORMAL", "LEFT", "CENTRE", "RIGHT", 0,
    PANEL_NOTIFY_PROC, set_horiz_align,
    0);
    panel_create_item( panel, PANEL_CYCLE,
    PANEL_ITEM_Y, ATTR_ROW(1), PANEL_ITEM_X, ATTR_COL(50),
    PANEL_LABEL_STRING, "Vert Align: ",
    PANEL_CHOICE_STRINGS,
        "NORMAL", "TOP", "CAP", "HALF", "BASE", "BOTTOM", 0,
    PANEL_NOTIFY_PROC, set_vert_align,
    0);
    panel_create_item( panel, PANEL_SLIDER,
    PANEL_ITEM_Y, ATTR_ROW(2), PANEL_ITEM_X, ATTR_COL(1),
    PANEL_VALUE_X, ATTR_COL(20),
    PANEL_LABEL_X, ATTR_COL(1),
    PANEL_LABEL_STRING, "Expansion (%): ",
    PANEL_MIN_VALUE, 0, PANEL_MAX_VALUE, 200,
    PANEL_SLIDER_WIDTH, 200,
    PANEL_NOTIFY_PROC, set_expansion,
    PANEL_NOTIFY_LEVEL, PANEL_ALL,
    PANEL_VALUE, (int)(cur_expansion * 100.0),
    0);
    panel_create_item( panel, PANEL_SLIDER,
    PANEL_ITEM_Y, ATTR_ROW(3), PANEL_ITEM_X, ATTR_COL(1),
    PANEL_VALUE_X, ATTR_COL(20),
    PANEL_LABEL_X, ATTR_COL(1),
    PANEL_LABEL_STRING, "Spacing (%): ",
    PANEL_MIN_VALUE, -200, PANEL_MAX_VALUE, 200,
    PANEL_SLIDER_WIDTH, 200,
    PANEL_NOTIFY_PROC, set_spacing,
    PANEL_NOTIFY_LEVEL, PANEL_ALL,
    PANEL_VALUE, (int)(cur_spacing * 100.0),
    0);
    panel_create_item( panel, PANEL_SLIDER,
    PANEL_ITEM_Y, ATTR_ROW(4), PANEL_ITEM_X, ATTR_COL(1),
    PANEL_VALUE_X, ATTR_COL(20),
    PANEL_LABEL_X, ATTR_COL(1),
    PANEL_LABEL_STRING, "Height: ",
    PANEL_MIN_VALUE, 0, PANEL_MAX_VALUE, 200,
    PANEL_SLIDER_WIDTH, 200,
    PANEL_NOTIFY_PROC, set_height,
    PANEL_NOTIFY_LEVEL, PANEL_ALL,
    PANEL_VALUE, (int)cur_height,
    0);
    panel_create_item( panel, PANEL_SLIDER,
    PANEL_ITEM_Y, ATTR_ROW(5), PANEL_ITEM_X, ATTR_COL(1),
    PANEL_VALUE_X, ATTR_COL(20),
    PANEL_LABEL_X, ATTR_COL(1),
    PANEL_LABEL_STRING, "Up.x : ",
    PANEL_MIN_VALUE, -10, PANEL_MAX_VALUE, 10,
    PANEL_SLIDER_WIDTH, 40,
    PANEL_NOTIFY_PROC, set_x_up,
    PANEL_NOTIFY_LEVEL, PANEL_ALL,
    PANEL_VALUE, (int)cur_up.x,
    0);
    panel_create_item( panel, PANEL_SLIDER,
    PANEL_ITEM_Y, ATTR_ROW(5), PANEL_ITEM_X, ATTR_COL(40),
    PANEL_VALUE_X, ATTR_COL(50),
    PANEL_LABEL_STRING, "Up.y : ",
    PANEL_MIN_VALUE, -10, PANEL_MAX_VALUE, 10,
    PANEL_SLIDER_WIDTH, 40,
    PANEL_NOTIFY_PROC, set_y_up,
    PANEL_NOTIFY_LEVEL, PANEL_ALL,
    PANEL_VALUE, (int)cur_up.y,
    0);

    return panel;
}

static void
```

```
initialize_text_struct( id )
    Pint    id;
{
    /* Create the structure used to display the text, its extent rectangle,
     * and its concatenation point, using the default text attribute values
     * defined at the top of this file.
     */
    popenstruct( id);
    psettextcolourind(CINDEX_GREEN);
    plabel( FONT_ELEMENT);
    psettextfont( cur_font);
    plabel( HEIGHT_ELEMENT);
    psetcharheight( cur_height);
    plabel( EXPANSION_ELEMENT);
    psetcharexpan( cur_expansion);
    plabel( SPACING_ELEMENT);
    psetcharspace( cur_spacing);
    plabel( UP_ELEMENT);
    psetcharup( &cur_up);
    plabel( PATH_ELEMENT);
    psettextpath( cur_path);
    plabel( ALIGN_ELEMENT);
    psettextalign( &cur_align);
    plabel( TEXT_ELEMENT);
    ptext3( &loc, dir, cur_text);

    psetlinecolourind( CINDEX_CYAN);
    plabel( EXTENT_ELEMENT);
    /* placeholder for polyline text extent box calculated in show_extent */
    plabel( DUMMY_ELEMENT);

    psetmarkercolourind( CINDEX_YELLOW);
    plabel( CATPOINT_ELEMENT);
    /* placeholder for concatentation point marker calculated in show_extent */
    plabel( DUMMY_ELEMENT);

    pclosestruct();

}

static void
grid2( bnds, nx, ny, color )
    Plimit  *bnds;
    int     nx, ny;
    Pint    color;
{
    Ppoint  ln[2];
    int     i;
    float   inc;

    /* Create structure elements defining a grid
     * over the entire display surface.
     */
    psetlinecolourind(color);
    inc =  ny > 1 ? (bnds->ymax - bnds->ymin) / (ny - 1) : 0;
    ln[0].x = bnds->xmin; ln[0].y = bnds->ymin;
    ln[1].x = bnds->xmax; ln[1].y = bnds->ymin;
    for ( i = 0; i < ny; i++ ) {
```

```
    ppolyline( 2, ln);
    ln[0].y += inc;
    ln[1].y += inc;
    }
    inc = nx > 1 ? (bnds->xmax - bnds->xmin) / (nx - 1) : 0;
    ln[0].x = bnds->xmin; ln[0].y = bnds->ymin;
    ln[1].x = bnds->xmin; ln[1].y = bnds->ymax;
    for ( i = 0; i < ny; i++ ) {
    ppolyline( 2, ln);
    ln[0].x += inc;
    ln[1].x += inc;
    }
}

static void
init_ws()
{
    /* Set the workstation to perform no updates until the application
     * explicitly requests one, define a colour table entry for gray,
     * and display the text with default attributes, its extent box,
     * the grid, etc.
     */
    psetdisplayupdatest( 1, PWAIT, PNIVE);
    init_view_mapping();
    psetcolourrep( 1, CINDEX_GRAY, &gray);
    ppoststruct( 1, 1, 1.0);
    show_extent();
}

build_structs()
{
    Ppoint  pt;
    Plimit  bnds;

    /* Create the structures used to display the text, its points
     * of origin and concatenation, and its extent rectangle.
     */
    popenstruct(1);
    psetviewind(1);
    bnds.xmin = bnds.ymin = WC_MIN;
    bnds.xmax = bnds.ymax = WC_MAX;
    grid2( &bnds, 21, 21, CINDEX_GRAY);
    pexecutestruct(2);

    /* origin marker */
    psetmarkercolourind( CINDEX_RED);
    psetmarkertype( PMK_PLUS);
    pt.x = pt.y = 0.0;
    ppolymarker( 1, &pt);

    pclosestruct();

    initialize_text_struct( 2);
}

static void
open_phigs()
{
    char        *buf = NULL;
    Pint        i, err, size;
    Pwstypelst      list;
    Pwstype     *wst;
    Phigs_base_name n;

    popenphigs( (Pchar*)0, PDEFAULT_MEM_SIZE);

    /* The list of available workstation types indicates which model was
     * selected at link time.  We look through the list to see what types
     * of workstations we can use from this application.  As soon as we
     * see one that tells us how the application was linked, we create one
     * of that type and return.
     */

    /* Get the buffer size needed. */
    pinqwstypes( 0, &err, (Pchar *)NULL, &list, &size);
    if ( !err) {
    if (size > 0)
        buf = malloc((unsigned)size);
    pinqwstypes( size, &err, buf, &list, &size);
    for ( i = 0, wst = list.ws_types; i < list.number; i++, wst++) {
        n = (Phigs_base_name)phigs_ws_type_get( *wst, PHIGS_BASE_NAME);
        if ( n == PHIGS_SUN_TOOL) {
        cur_wst = phigs_ws_type_create( phigs_ws_type_sun_tool,
            PHIGS_TOOL_X, WS_X, PHIGS_TOOL_Y, WS_Y,
            PHIGS_TOOL_WIDTH, WS_SIZE, PHIGS_TOOL_HEIGHT, WS_SIZE,
            PHIGS_COLOR_TABLE_SIZE, 8,
            PHIGS_DOUBLE_BUFFER, PHIGS_DBL_CMAP,
            PHIGS_TOOL_LABEL, "SunPHIGS Workstation",
            PHIGS_TOOL_SHOW_LABEL, FALSE,
            PHIGS_TEXTSW, PHIGS_NONE,
            0);
        break;
        } else if ( n == PHIGS_SUN_CANVAS) {
        cur_wst = phigs_ws_type_sun_canvas;
        break;
        }
    }
    if (buf)
        free(buf);
    }
}

main( argc, argv )
    int         argc;
    char        *argv[];
{
    Frame   frame, pframe;
    Canvas  canvas;
    Panel   panel;

    /* Open PHIGS and create the panel with which the user may set
     * text attribute values.
     */
    open_phigs();
    frame = window_create( NULL, FRAME,
    WIN_X, 25, WIN_Y, 20,
    FRAME_NO_CONFIRM, TRUE,
    0);
    panel = create_panel( frame);
    window_fit(panel);
    window_fit(frame);

    /* Create the structures used to display the text, its points
     * of origin and concatenation, its extent rectangle, and a grid.
     * Open the structure containing the text string, concatentation
     * point, and extent rectangle, and set the edit mode to REPLACE
     * to allow replacement of the text attribute elements as they
     * are changed by the user.
     */
    build_structs();
    popenstruct(2);
    pseteditmode(PEDIT_REPLACE);

    /* Open a workstation of the type appropriate for the model selected
     * at link time and display the text on it.
     */
    switch ( (int)phigs_ws_type_get( cur_wst, PHIGS_BASE_NAME)) {
    case PHIGS_SUN_TOOL:
        window_set( frame, FRAME_LABEL, "Tool Ws", 0);
        popenws( 1, (Pconnid)NULL, cur_wst);
        break;

    case PHIGS_SUN_CANVAS:
        pframe = window_create( frame, FRAME,
        WIN_X, WS_X, WIN_Y, WS_Y,
        WIN_SHOW, TRUE,
        0);
        window_set( frame, FRAME_LABEL, "Canvas Ws", 0);
        canvas = window_create( pframe, CANVAS,
        WIN_X, 0,
        WIN_WIDTH, WS_SIZE, WIN_HEIGHT, WS_SIZE,
        0);
        popenws( 1, (Pconnid)canvas, cur_wst);
        window_fit(pframe);
    }
    init_ws();

    /* Wait for user input and modify the text attributes accordingly. */
    window_main_loop( frame);

    /* Cleanup. */
    pclosestruct();
    pclosews(1);
    pclosephigs();
}
```

A.2. FORTRAN Examples

The following example programs use the *SunPHIGS* FORTRAN binding.

fbundles.f

Demonstrates the use of workstation attribute bundles.

```
c   Note:  It is necessary to include the declarations in phigs77.h
c      in order to open a workstation or access SunPHIGS constants.

      include '/usr/include/phigs/phigs77.h'

c   Turn off implicit typing of possible SunPHIGS names.
c   This will cause the compiler to report undeclared usage of such names.

      implicit undefined (P,p,E,e)
      integer wkid, strid, bundlindex, linetype, colorindex, i, j, k
      real xstar(6), xstar2(6), ystar(6), linewidth
      data xstar /0.2, 0.3, 0.4, 0.1, 0.5, 0.2 /
      data ystar /0.2, 0.5, 0.2, 0.4, 0.4, 0.2 /
      wkid  = 1
      strid = 1

c   Open PHIGS, open a SunPHIGS phigswsttool workstation and open a structure.

      call popph(6,0)
      call popwk(wkid, 0, phigswsttool)
      call popst(strid)

c   Define different polyline attributes in 3 workstation bundle tables.

      do 25 i = 2,4
        bundlindex = i
        linetype   = i
        linewidth  = float(i)
        colorindex = i
        call psplr(wkid, bundlindex, linetype, linewidth, colorindex)
25    continue

c   Set the Aspect Source Flag to bundled for each polyline attribute.

      call psiasf(PLN, PBUNDL)
      call psiasf(PLWSC, PBUNDL)
      call psiasf(PPLCI, PBUNDL)

c   Create a polyline star using the default workstation bundle table (index 1).

      call ppl(6, xstar, ystar)

c   Create more polyline stars using polyline bundle indices 2, 3 and 4

      do 50 j = 1,6
        xstar2(j) = xstar(j) + 0.45
50    continue

      call pspli(2)
      call ppl(6, xstar2, ystar)
      do 100 k = 1,6
        ystar(k) = ystar(k) + 0.45
100   continue

      call pspli(3)
      call ppl(6, xstar, ystar)

      call pspli(4)
      call ppl(6, xstar2, ystar)

c  Close the structure and post it to the workstation for display.

      call pclst
      call ppost(wkid, strid, 0.)
      call sleep(5)

c  Close the workstation and close PHIGS.

      call pclwk(wkid)
      call pclph

      stop
      end
```

ffillset.f

Demonstrates how to create a fill area set.

```
c  @(#)ffillset.f 2.1 88/06/02 SMI

c  ffillset.f - This program demonstrates how to create a fill area set
c  with and without edges using the SET EDGE FLAG attribute.
c  This program also demonstrates the creation of two modified workstations
c  and the traversing of a child structure using the EXECUTE STRUCTURE ELEMENT.

c  Note:  It is necessry to include the declarations in phigs77.h
c    in order to open a workstation or access SunPHIGS constants.

      include '/usr/include/phigs/phigs77.h'

c  Turn off implicit typing of possible SunPHIGS names.
c  This will cause the compiler to report undeclared usage of such names.

      implicit undefined (P,p,E,e)

      integer i, WS1, WS2, tool1, tool2, structA, structB
      integer boundaries, hatchindex, colourindex
      real fasxarr(12), fasyarr(12), diamond1x(4), diamond1y(4)
      real diamond2x(4), diamond2y(4), diamond3x(4), diamond3y(4)
      dimension boundaries(3)

c  Fill the boundaries array elements with the cumulative number of
c  array elements in each set of points (i.e., 4 + 4 = 8 + 4 = 12)

      data boundaries /4, 8, 12/
      data hatchindex /-5/
      data colourindex /6/

c  Fill each point set array that will be included in the fill area set arrays.

      data diamond1x /0.0, 0.5, 1.0, 0.5/
      data diamond1y /0.5, 0.0, 0.5, 1.0/
      data diamond2x /0.2, 0.5, 0.8, 0.5/
      data diamond2y /0.5, 0.2, 0.5, 0.8/
      data diamond3x /0.4, 0.5, 0.6, 0.5/
      data diamond3y /0.5, 0.4, 0.5, 0.6/

c  Fill the fill area set arrays with the data from each point set array.

      do 50 i = 1,4
            fasxarr(i)    = diamond1x(i)
            fasyarr(i)    = diamond1y(i)
            fasxarr(i+4)  = diamond2x(i)
            fasyarr(i+4)  = diamond2y(i)
            fasxarr(i+8) = diamond3x(i)
            fasyarr(i+8) = diamond3y(i)
50    continue

      WS1 = 1
      WS2 = 2
      structA = 1
      structB = 2
```

```
c  PHIGS must be opened prior to calling any other PHIGS function.

      call popph(6, 0)

c  Create workstations with modified sizes and x-y positions on the
c  display surface so that both workstations are visible at the same time.
c  Note:  with a FORTRAN program, these attributes are set individually
c  rather than in an attribute value list.

      call phigswstcreate(phigswsttool, tool1)
      call phigswstset(tool1, PHIGSTOOLWIDTH, 500)
      call phigswstset(tool1, PHIGSTOOLHEIGHT, 500)
      call phigswstset(tool1, PHIGSTOOLX, 50)
      call phigswstset(tool1, PHIGSTOOLY, 50)
      call phigswstcreate(tool1, tool2)
      call phigswstset(tool2, PHIGSTOOLX, 600)
      call phigswstset(tool2, PHIGSTOOLY, 300)

c  Open the modified phigswsttool workstations using the workstation handles
c  returned from phigswstcreate.

      call popwk(WS1, 0, tool1)
      call popwk(WS2, 0, tool2)

c  Open structure A and create the structure element SET EDGE FLAG with a
c  value of on (default is off).  Also create an EXECUTE STRUCTURE element
c  which will create the empty structure B.  This structure will be
c  open and filled next.  Close structure A.

      call popst(structA)
      call psedfg(PON)
      call pexst(structB)
      call pclst

c  Open structure B and create the fill area set attribute elements SET
c  INTERIOR STYLE, SET INTERIOR STYLE INDEX and SET INTERIOR COLOUR INDEX.
c  Create a fill area set element with 3 boundaries and close structure B.

      call popst(structB)
      call psis(PHATCH)
      call psisi(hatchindex)
      call psici(colourindex)
      call pfas(3, boundaries, fasxarr, fasyarr)
      call pclst

c  Post structure A to workstation 1 displaying the fill area set with edge

      call ppost(WS1, structA, 0.)
      call pmsg(WS1, 'Displays a fill area set with the edge flag on.')

c  Post structure B to workstation 2, which displays the same fill area set
c  but without edges as the edge flag element resides in structure A only.

      call ppost(WS2, structB, 0.)
      call pmsg(WS2, 'Displays a fill area set with the edge flag off.')
      call sleep(10)

c  Close both workstations and PHIGS.
```

```
call pclwk(WS1)
call pclwk(WS2)
call pclph

stop
end
```

flines.f

Demonstrates the various POLYLINE attributes.

```
c  @(#)flines.f 2.1 88/06/02 SMI

c  flines.f - This program displays all of the PHIGS standard
c             polyline types available in SunPHIGS.

c  Note:  It is necessry to include the declarations in phigs77.h
c         in order to open a workstation or access SunPHIGS constants.

      include '/usr/include/phigs/phigs77.h'

c  Turn off implicit typing of possible SunPHIGS names.
c  This will cause the compiler to report undeclared usage of such names.

      implicit undefined (P,p,E,e)
      integer colourindex
      real linewidth
      real xendpoints, yendpoints
      common colourindex, linewidth, xendpoints(2), yendpoints(2)

c  Initialize the colour index, line width and end-point coordinate data.

      data colourindex /2/
      data linewidth /2.0/
      data xendpoints /0.1, 0.9/
      data yendpoints /0.15, 0.15/

c  Open PHIGS, open workstation 1, open structure 1.
c  Use logical unit number 6 for the SunPHIGS error file.

      call popph(6, 0)
      call popwk(1, 0, phigswsttool)
      call popst(1)

c  Add set character height (default is .01)

      call pschh(.015)

c  Add set linetype, set polyline colour index and set linewidth scale factor
c  elements to the open structure, then add the polyline.

      call psln(PLNDOTDASHDOT)
      call psplci(colourindex)
      call pslwsc(linewidth)
      call ppl(2, xendpoints, yendpoints)

c  Add set text font and set text colour index elements, then add text.

      call pstxfn(PFONTROMANSMPLX)
      call pstxci(colourindex)
      call ptx(xendpoints(1), yendpoints(1) - .035, 'PLNDOTDASHDOT')

c  Add new linetypes and text fonts, then add new lines followed with text.

      call psln(PLNLONGDASH)
      call pstxfn(PFONTROMANDPLX)
      call newline
      call ptx(xendpoints(1), yendpoints(1) - .035, 'PLNLONGDASH')

      call psln(PLSOLI)
      call pstxfn(PFONTROMANCMPLX)
      call newline
      call ptx(xendpoints(1), yendpoints(1) - .035, 'PLSOLI')

      call psln(PLDASH)
      call pstxfn(PFONTROMANTRPLX)
      call newline
      call ptx(xendpoints(1), yendpoints(1) - .035, 'PLDASH')

      call psln(PLDOT)
      call pstxfn(PFONTITALICCMPLX)
      call newline
      call ptx(xendpoints(1), yendpoints(1) - .035, 'PLDOT')

      call psln(PLDASD)
      call pstxfn(PFONTITALICTRPLX)
      call newline
      call ptx(xendpoints(1), yendpoints(1) - .035, 'PLDASD')

c  Post structure 1 to WS 1 to have the structure's contents displayed; pause.

      call ppost(1, 1, 0.)
      call pmsg(1, 'Displays all polyline types.  Exits in 10 seconds.')
      call sleep(10)

c  Close the structure, close the WS, and close PHIGS.

      call pclst
      call pclwk(1)
      call pclph

      stop
      end

c  ==newline==

      subroutine newline

c  Add new polyline and text colour indices, increased linewidth scale factor
c  and end-point coordinate data to the open structure, then add new polyline.

      implicit undefined (P,p,E,e)
      integer colourindex
      real linewidth
      real xendpoints, yendpoints
      common colourindex, linewidth, xendpoints(2), yendpoints(2)

      colourindex = colourindex + 1
      linewidth = linewidth + 1.0
      yendpoints(1) = yendpoints(1) + .15
      yendpoints(2) = yendpoints(2) + .15

      call psplci(colourindex)
call pstxci(colourindex)
call pslwsc(linewidth)
call ppl(2, xendpoints, yendpoints)

return
end
```

fmarkers.f

Demonstrates the POLYMARKER function.

```
c  @(#)fmarkers.f 2.1 88/06/02 SMI

c  fmarkers.f - This program displays all of the PHIGS standard
c               marker types using a SunView canvas subwindow.

c  Note:  It is necessry to include the declarations in phigs77.h
c         in order to open a workstation or access SunPHIGS constants.

      include '/usr/include/phigs/phigs77.h'

c  Turn off implicit typing of possible SunPHIGS names.
c  This will cause the compiler to report undeclared usage of such names.

      implicit undefined (P,p,E,e)
      integer canvas, canvasid, i, colour(5), marker(5)
      character*6 text(5)
      real xpoint, ypoint(5)

c  Initialize the colour index, marker type and text arrays.
c  Initialize the x-y coordinates.

c  Default colours:  1=white; 2=red; 3=green; 4=blue; 5=yellow.

      data colour /1, 2, 3, 4, 5/
      data marker /PPOINT, PPLUS, PAST, POMARK, PXMARK/
      data text /'PPOINT', 'PPLUS ', 'PAST  ', 'POMARK', 'PXMARK'/
      data xpoint /.65/
      data ypoint /.1, .3, .5, .7, .9/

c   SunView Setup - canvasid is a C function which returns the
c   connection identifier 'canvas' for a SunPHIGS canvas workstation.

      canvas = canvasid()

c  Open PHIGS.  Use logical unit number 6 for the SunPHIGS error file.

      call popph(6, 0)

c  Open workstation 1 and open structure 1.

      call popwk(1, canvas, phigswstcanvas)
      call popst(1)

c  Add set marker scale factor element to the open structure.

      call psmksc (5.)

c  Add set marker type and set polymarker colour index, then add polymarker
c  and text.

      do 50 i = 1,5
            call psmk(marker(i))
            call pspmci(colour(i))
            call ppm(1,xpoint, ypoint(i))
            call pschh(.015)
            call ptx(xpoint-.2, ypoint(i), text(i))
50    continue

c  Post structure 1 to WS 1 to have structure's contents displayed on Canvas.

      call ppost(1, 1, 0.)

c  Call SunView's notifier to have SunView canvas window displayed.

      call display

c  Close the structure, close the WS, and close PHIGS.

      call pclst
      call pclwk(1)
      call pclph

      stop
      end
```

fpolygons.f

Demonstrates interior styles for fill areas.

```
c  @(#)fpolygons.f 2.1 88/06/02 SMI

c  fpolygons.f - This program draws fill areas with different interior styles.

c  Note:  It is necessry to include the declarations in phigs77.h
c     in order to open a workstation or access SunPHIGS constants.

      include '/usr/include/phigs/phigs77.h'

c  Turn off implicit typing of possible SunPHIGS names.
c  This will cause the compiler to report undeclared usage of such names.

      implicit undefined (P,p,E,e)

      integer row, x, y, colourindex, hatchindex, newstyle
      real fapointx(4), fapointy(4), savepts(4)
      data savepts  /.025, .225, .225, .025/
      data fapointx /.025, .225, .225, .025/
      data fapointy /.75, .75, .95, .95/
      data colourindex /2/
      data hatchindex /-1/
      data newstyle /1/

c  Open PHIGS, open a workstation(1) and open a structure(1).
c  Use logical unit number 6 for the SunPHIGS error file.

      call popph(6, 0)
      call popwk(1, 0, phigswsttool)
      call popst(1)

c  Add the SET INTERIOR STYLE (hollow) element to the open structure.

      call psis(PHOLLO)

c  Add a SET INTERIOR COLOUR INDEX attribute element and a row of fill areas.
c  Note:  set interior style index for hatch will not be implemented
c         until the 'hatched' row is displayed.

25        do 75 row = 1,4
            call psici(colourindex)
            call psisi(hatchindex)
            call pfa(4, fapointx, fapointy)

c  Move x-axis over.
            do 50 x = 1,4
                  fapointx(x) = fapointx(x) + .25
50                continue
            colourindex = colourindex + 1
            hatchindex = hatchindex - 1
75      continue

c  Move y-axis down.

      do 100 y = 1,4
```

```
            fapointx(y) = savepts(y)
            fapointy(y) = fapointy(y) - .25
100       continue

      goto (125, 150, 175) newstyle

c  Add a another SET INTERIOR STYLE (solid) and the next row of fill areas.

125   call psis(PSOLID)
      colourindex = 4
      newstyle = newstyle + 1
      goto 25

c  Add a another SET INTERIOR STYLE (hatch) and the next row of fill areas.

150   call psis(PHATCH)
      colourindex = 2
      hatchindex = -1
      newstyle = newstyle + 1
      goto 25

c  Add the "empty" interior style to show that it is invisible for fill areas

175   call psis(PISEMP)
      call pfa(5, fapointx, fapointy)
      call pschh(.015)
      call ptx(fapointx(1), fapointy(1) + .1,
&    'Interior style Empty is invisible.  See FILL AREA SET.')

c  Close the structure and post it to the workstation.
c  This will cause the structure's contents to be displayed.

      call pclst
      call ppost(1, 1, 0.)
      call pmsg(1, 'Displays sets of polygons.  Exits in 10 seconds.')
      call sleep(10)

c  Close the the workstation and close PHIGS.

      call pclwk(1)
      call pclph

      stop
      end
```

ftext.f

Demonstrates the text fonts available with *SunPHIGS*.

```
c  @(#)ftext.f 2.1 88/06/02 SMI

c  ftext.f - This program displays all of the fonts available in SunPHIGS.

c  Note:  It is necessry to include the declarations in phigs77.h
c         in order to open a workstation or access SunPHIGS constants.

        include '/usr/include/phigs/phigs77.h'

c  Turn off implicit typing of possible SunPHIGS names.
c  This will cause the compiler to report undeclared usage of such names.

      implicit undefined (P,p,E,e)
      integer font, index, colour(6)
      character*22 text(13)
      real xpoint, ypoint

c  Initialize the index into the text array.  Note, all but the default font
c  are in negative numbers.  Initialize the arrays and the x-y coordinates.

      data font /2/
      data colour /2, 3, 4, 5, 6, 7/

      data text/'PFONTROMANMONO = 1    ',
     1 'PFONTROMANSMPLX = -2  ', 'PFONTROMANDPLX = -3   ',
     2 'PFONTROMANCMPLX = -4  ', 'PFONTROMANTRPLX = -5  ',
     3 'PFONTITALICCMPLX = -6 ', 'PFONTITALICTRPLX = -7 ',
     4 'PFONTGREEKSMPLX = -8  ', 'PFONTGREEKCMPLX = -9  ',
     5 'PFONTSCRIPTSMPLX = -10', 'PFONTSCRIPTCMPLX = -11',
     6 'PFONTCARTO = -12      ', 'PFONTSYMBOL = -13     '/

      data xpoint /.1/
      data ypoint /.8/

c  Open PHIGS, open workstation 1, open structure 1.
c  Use logical unit number 6 for the SunPHIGS error file.

        call popph(6, 0)
        call popwk(1, 0, phigswsttool)
        call popst(1)

c  Add set character height (default is .01)

      call pschh(.015)

c  Add text in the default font (PFONTROMANMONO) and default String Precision.

      call ptx (xpoint, ypoint,
     1 'These are all of the fonts supported by SunPHIGS.')

c  Change set text precision to Character Precision and add to the structure.

      call pstxpr(PCHARP)
```

```
c  Add set text colour index and set text font to the open structure,
c  then add the text until all fonts are displayed.

      do 50 index = 1,6
            ypoint = ypoint - .1
            call pstxci(colour(index))
            call pstxfn(-font)
            call ptx(xpoint, ypoint, text(font))
            font = font + 1
50    continue

c  Add set text precision to Stroke Precision and add new x-y coordinates.

      call pstxpr(PSTRKP)
      xpoint = .5
      ypoint = .8
      do 100 index = 1,6
            ypoint = ypoint - .1
            call pstxci(colour(index))
            call pstxfn(-font)
            call ptx(xpoint, ypoint, text(font))
            font = font + 1
100   continue

c  Post structure 1 to WS 1 to have the structure's contents displayed; pause.

      call ppost(1, 1, 0.)
      call pmsg(1, 'Displays all available fonts.  Exits in 10 seconds.')
      call sleep(10)

c  Close the structure, close the WS, and close PHIGS.

      call pclst
      call pclwk(1)
      call pclph

      stop
      end
```

ftextall.f

Displays text in various directions.

```
c  @(#)ftextall.f 2.2 89/03/31 SMI

c  ftxtall.f - This program demonstrates the use of text attributes
c  to display text in many different fonts, sizes and directions.
c  SET CHARACTER HEIGHT, SET TEXT COLOUR INDEX, SET TEXT FONT, SET
c  CHARACTER SPACING, SET CHARACTER EXPANSION FACTOR, SET TEXT PATH,
c  and SET CHARACTER UP VECTOR are all used to manipulate the character data.

c  Note:  It is necessry to include the declarations in phigs77.h
c     in order to open a workstation or access SunPHIGS constants.

      include '/usr/include/phigs/phigs77.h'

c  Turn off implicit typing of possible SunPHIGS names.
c  This will cause the compiler to report undeclared usage of such names.

        implicit undefined (P,p,E,e)
        integer wkid, strid, i, font, color
        real x, x1, y, height, space, expfactor
        data x /.01/
        data x1 /.2/
        data y /.95/
        data font /-2/
        data color /2/
        data height /.02/
        data space /.5/
        wkid  = 1
        strid = 1

c  Open PHIGS, open a SunPHIGS phigswsttool workstation and open a structure.

        call popph(6,0)
        call popwk(wkid, 0, phigswsttool)
        call popst(strid)

c  Create attribute elements for setting text font and colour, character height,
c  spacing and expansion factor.  Then create the text primitive element.

        do 50 i = 1,3
              call pstxfn(font)
              call pstxci(color)
              call pschh(height)
              call ptx(x, y, 'Welcome to ')
              call pschsp(space)
          expfactor = float(i) - .5
              call pschxp(expfactor)
              call ptx(x1, y, 'PHIGS!')

c  Change x-y coordinates and attribute parameter values.

              x1 = x1 + (i * .1)
              y  = y - .1
              font   = font - 1
              color  = color + 1
              height = height + .01
              space  = space - .25

c  Reset to default values for character spacing and expansion factor.
              call pschsp(0.)
              call pschxp(1.)
50    continue

c  Set different paths (default PRIGHT) and up vectors (default 0.,1.)
c  Note:  default colour indices are: 0=black; 1=white; 2=red; 3=green;
c         4=blue; 5=yellow; 6=cyan; 7=magenta; > 7 = white.

        call pstxfn(font)
        call pstxci(color)
        call pschh(.03)
        call pstxp(PLEFT)
        call ptx(.65, .625, '?dnuora denruT')
        call pstxci(6)
        call pschh(.02)
        call pstxp(PUP)
        call ptx(.75, .05, 'Things are looking up!')
        call pstxci(7)
        call pstxp(PDOWN)
        call ptx(.9, .6, 'Feeling down?')
        call pstxp(PRIGHT)

        call ptx(.2, .4, '   {0., 1.}')
        call pschup(1., 0.)
        call ptx(.2, .4, '   {1., 0.}')
        call pschup(0., -1.)
        call ptx(.2, .4, '   {0., -1.}')
        call pschup(-1., 0.)
        call ptx(.2, .4, '   {-1., 0.}')

        call pstxci(6)
        call pschh(.03)
        call pschup(-1., 1.)
        call ptx(.4, .2, 'Full tilt boogie!')
        call pstxci(3)
        call pschup(1., -1.)
        call ptx(.6, .35, "I'm upside down!")

c  Close the structure and post it to the workstation for display.

      call pclst
      call ppost(wkid, strid, 0.)
      call sleep(10)

c  Close the workstation and close PHIGS.

        call pclwk(wkid)
        call pclph

        stop
        end
```

toolattrs.f

Demonstrates *Sun Tool* workstation configuration.

```
c  @(#)toolattrs.f 2.1 88/06/02 SMI

c  toolattrs.f - This program demonstrates how to create and modify
c         a new workstation of type phigswsttool and how to set and
c         retrieve the values stored in the workstation description table.

c  Note:  It is necessry to include the declarations in phigs77.h
c         in order to open a workstation or access SunPHIGS constants.

      include '/usr/include/phigs/phigs77.h'

c  Turn off implicit typing of possible SunPHIGS names.
c  This will cause the compiler to report undeclared usage of such names.

      implicit undefined (P,p,E,e)
      integer wkid, strid, wstooltype, labellen
      character label*80

c  Initialize data.

      wkid = 1
      strid = 1
      label = 'FORTRAN phigswsttool'

c  Open PHIGS, use logical unit number 6 for the SunPHIGS error file.

      call popph(6, 0)

c  Create a new workstation type and modify the workstation attribute values.
c  Note:  with a FORTRAN program, these attributes are set individually
c  rather than in an attribute value list.

      call phigswstcreate(phigswsttool, wstooltype)
      call phigswstset(wstooltype, PHIGSTOOLHEIGHT, 600)
      call phigswstset(wstooltype, PHIGSTOOLWIDTH, 600)
      call phigswstset(wstooltype, PHIGSTEXTSW, PHIGSNONE)
      call phigswstset(wstooltype, PHIGSTOOLLABEL, label(1:20))

c  Open the modified phigswsttool workstation using the workstation handle
c  returned from phigswstcreate.  Open a structure for text primitive display.

      call popwk(wkid, 0, wstooltype)
      call popst(strid)

      label = 'This is the current contents of the label string. '
      call ptx(.1, .7, label)

c  Retrieve a workstation attribute value from the workstation description
c  table and display its contents with the PHIGS text primitive.

      call phigswstget(wstooltype, PHIGSTOOLLABEL, labellen, label)
      call pschh(.02)
      call ptx(.1, .4, label)
```

```
c  Post the structure to the workstation to display the structure's contents.

      call ppost(wkid, strid, 0.)
      call sleep(10)

c  Close the structure, close the workstation, and close PHIGS.

      call pclst
      call pclwk(wkid)
      call pclph

      stop
      end
```

B

Tutorial Examples

Tutorial Examples **161**

B.1. C Examples 161

cfigs1.c 161

cfigs2.c 162

cfigs3.c 163

loc.c 165

pickjet.c 167

B.2. FORTRAN Examples 171

ffigs1.f 171

ffigs2.f 172

ffigs3.f 173

ffigs4.f 174

figscanval.f 177

figstoolval.f 179

B

Tutorial Examples

Appendix B contains example programs from the /usr/lib/phigs1.1/examples/tutorial.

B.1. C Examples

The following example programs use the *SunPHIGS* C binding.

cfigs1.c

A skeleton *PHIGS* program.

```
#ifndef lint
static  char sccsid[] = "@(#)cfigs1.c 2.1 88/06/02 SMI";
#endif

/*  cfigs1.c - A simple skeleton PHIGS program in C
    which demonstrates how to open and close a SunPHIGS workstation.

Note:  It is necessry to include the declarations in phigs.h
in order to open a workstation or access the SunPHIGS constants.  */

#include <phigs/phigs.h>

main()
{

/*  Open PHIGS  */

        popenphigs((Pchar*)NULL, PDEFAULT_MEM_SIZE);

/*  Open a workstation with a value of 1 and pause for 5 seconds.  */

        popenws(1, (Pconnid)NULL, phigs_ws_type_sun_tool);
        sleep(5);

/*  Close workstation 1 and close PHIGS.  */

        pclosews(1);
        pclosephigs();
}
```

cfigs2.c

Demonstrates the use of structures.

```
#ifndef lint
static  char sccsid[] = "@(#)cfigs2.c 2.1 88/06/02 SMI";
#endif

/*  cfigs2.c - Demonstrates the use of structures
    and structure elements in displaying a simple polyline.  */

#include <phigs/phigs.h>

main()
{
/*  Define a series of points for a polyline to be displayed.
    Ppoint is a structure containing the x and y coordinates. */

        static Ppoint xypoints[] = { {0.1, 0.5}, {0.9, 0.5} };

/*  Open PHIGS, open a workstation and open structure 1.  */

        popenphigs((Pchar*)NULL, PDEFAULT_MEM_SIZE);
        popenws(1, (Pconnid)NULL, phigs_ws_type_sun_tool);
        popenstruct(1);

/*  Insert the polyline structure element into the open structure.
    The first parameter is the number of points to be used to define
    the polyline.  The second parameter is the array of structures
    which contain the x-y coordinates.  Close the structure.  */

        ppolyline(2, xypoints);
        pclosestruct();

/*  Post the structure to the workstation in order to
    have the structure's contents displayed; pause.  */

        ppoststruct(1,1,0.);
        sleep(5);

/*  Close the workstation and close PHIGS.  */

        pclosews(1);
        pclosephigs();
}
```

cfigs3.c

Demonstrates primitive attributes.

```
#ifndef lint
static  char sccsid[] = "@(#)cfigs3.c 2.1 88/06/02 SMI";
#endif

/*  cfigs3.c - Demonstrates defining output primitives attributes
    to change the appearance of a simple polyline.  */

#include <phigs/phigs.h>

main()
{
/*  Define a series of points for a polyline to be displayed.
    Ppoint is a structure containing the x and y coordinates. */

        static Ppoint xypoints[] = { {0.1, 0.5}, {0.9, 0.5} };
    static Pfloat linewidth  = 5.0;
    static Pint colorindex   = 5;

/*  Open PHIGS, open a workstation and open structure 1.  */

        popenphigs((Pchar*)NULL, PDEFAULT_MEM_SIZE);
        popenws(1, (Pconnid)NULL, phigs_ws_type_sun_tool);
        popenstruct(1);

/*  Insert the polyline structure element into the open structure.
    The first parameter is the number of points to be used to define
    the polyline.  The second parameter is the array of structures
    which contain the x-y coordinates.  */

        ppolyline(2, xypoints);

/*  Create structure elements to set the linetype, the linewidth scale facto
    and the polyline color index; move the y-axis coordinates and redraw the
    polyline.   Close the structure after all elements have been inserted.

        psetlinetype(PLN_DOTDASHDOT);
        psetlinewidth(linewidth);
        psetlinecolourind(colorindex);

        xypoints[0].y += .2;
        xypoints[1].y += .2;
        ppolyline(2, xypoints);

        pclosestruct();

/*  Post the structure to the workstation in order to
    have the structure's contents displayed; pause.  */

        ppoststruct(1,1,0.);
        sleep(5);
/*  Close the workstation and close PHIGS.  */

        pclosews(1);
        pclosephigs();
}
```

loc.c

A logical locator input device example.

```
#ifndef lint
static char sccsid[] = "@(#)loc.c 2.1 88/06/02 SMI";
#endif

/*
 *  loc.c  - Demonstrates use of locator input device.
 *  Coordinate system is from 0.0 to 1.0 in both x and y
 *  directions.
 */

#include    <phigs/phigs.h>
#include    <suntool/canvas.h>

Pint        ws      = 1;
Pint        devid       = 1;    /* locator 1 is LEFT mouse button */

Pistatus request_locator();

main( argc, argv)
    int argc;
    char    *argv[];
{
    Pistatus status;

    open_phigs();
    initialize_input();

    pmessage( ws,
        "Use the LEFT mouse button to select cursor locations.\n");
    pmessage( ws,
        "Use CTRL-D (input break action) to terminate program.\n");

    do {
        status = request_locator();
    } while (status == PSTAT_OK);

    close_phigs();
    exit(0);
}


/*
 *  Initialization routine.  Initialize PHIGS and the SunPHIGS
 *  Sun Tool workstation.
 *
 *  The workstation is never updated since nothing is posted to it.
 */
open_phigs()
{

    popenphigs( (Pchar *)NULL, PDEFAULT_MEM_SIZE);
    popenws( ws, (Pconnid)NULL, phigs_ws_type_sun_tool);
}
```

```
/*
 *  Initialize the locator device.
 */
initialize_input()
{

    static Ploc init_location = { 0, {0., 0.} };
    static Plocrec  record;     /* not used, but must be present */
    Plimit      area;
    Pint        pet = 1;    /* data record not used for PET 1 */

    area.xmin = area.ymin = 0.0;
    area.xmax = area.ymax = 1.0;

    pinitloc( ws, devid, &init_location, pet, &area, &record);
        psetlocmode( ws, devid, PREQUEST, PES_ECHO);
}


/*
 *  Use request mode to request the input from the locator device.
 *  preqloc will return an X,Y location and the view index used,
 *  which will be 0 because we haven't changed the view input priorities.
 */
Pistatus
request_locator()
{
        Pqloc       locator;

        preqloc( ws, devid, &locator);
        if (locator.status == PSTAT_OK)
                printf( "Locator: x pos=%f\ty pos=%f\n",
            locator.loc.position.x, locator.loc.position.y);
        else if (locator.status == PSTAT_NONE)
                printf( "Operator did CTRL-D break action.\n");
    return( locator.status);
}


/*
 *  Close SunPHIGS workstation and PHIGS.
 */
close_phigs()
{
    pclosews(ws);
    pclosephigs();
}
```

pickjet.c

A simple picking example.

```
#ifndef lint
static char sccsid[] = "@(#)pickjet.c 2.1 88/06/02 SMI";
#endif

/*
 *  pickjet.c - Demonstrates use of pick input device.
 *
 *  Draw jet as three parts: jetbody, right wing, left wing.
 *  The user may "pick" a jet part, by placing the mouse over
 *  the part and pressing the left mouse button.
 *
 *  This program has sets the "pick aperture" and the pick filter
 *  of the pick device.
 */

#include    <phigs/phigs.h>
#include    <suntool/canvas.h>

#define     PICKABLE    1

#define     BLACK       0
#define     WHITE       1
#define     RED     2
#define     GREEN       3
#define     BLUE        4
#define     YELLOW      5
#define     CYAN        6
#define     MAGENTA     7

static  Ppoint  jetbody[3]   = { {0.1, 0.6}, {0.9, 0.6}, {0.2, 0.5} };
static  Ppoint  rightwing[3] = { {0.4, .53}, {.55, .55}, {.36, .46} };
static  Ppoint  leftwing[3]  = { {0.4, 0.6}, {.55, 0.6}, {.36, .65} };

Pint        ws      = 1;
Pint        structure   = 1;
Pfloat      priority    = 1.0;
Pint        devid       = 1;    /* PICK device 1 is LEFT mouse button */

Pqpickstatus request_pick();

main( argc, argv)
    int argc;
    char    *argv[];
{
    Pqpickstatus status;

    open_phigs();
    build_css();
    initialize_input();
    pmessage( ws,
        "Use the LEFT mouse button to PICK the part under cursor.\n");
    pmessage( ws,
        "Use CTRL-D (input break action) to terminate program.\n");
    do {
        status = request_pick();
    } while( status != PQP_NONE );

    close_phigs();
    exit(0);
}


/*
 *  Initialization routine.  Initialize PHIGS and the SunPHIGS
 *  Sun Tool workstation.
 *
 *  The display will be updated only when the user requests it to be.
 */
open_phigs()
{

    popenphigs( (Pchar *)NULL, PDEFAULT_MEM_SIZE);

    popenws( ws, (Pconnid)NULL, phigs_ws_type_sun_tool);

    psetdisplayupdatest( ws, PASTI, PNIVE);/* turn off auto updates  */
    ppoststruct( ws, structure, priority);/* post before edit is ok */
    sleep(1);
}


/*
 *  Build the jet structure out of three parts:
 *  jet body, right wing and left wing.
 *  The jet is a solid object filled with a different
 *  color for each part.  Each part is pickable seperately.
 */
build_css()
{
    Pintlst names_to_add;
    static Pint name[1] = { PICKABLE };
    Pint    part = 1;

    popenstruct( structure);

    names_to_add.number = 1;
    names_to_add.integers = name;
    paddnameset( &names_to_add);

    psetintstyle( PSOLID);

    psetintcolourind( BLUE);    /* jet body is blue */
    psetpickid( part);      /* pick this as part #1 */
    pfillarea( 3, jetbody);

    psetintcolourind( GREEN);   /* right wing is green */
    psetpickid( ++part);        /* pick this as part #2 */
    pfillarea( 3, rightwing);

    psetintcolourind( RED);     /* left wing is red */
    psetpickid( ++part);        /* pick this as part #3 */
    pfillarea( 3, leftwing);

    pclosestruct();

    pupdatews( ws, PPERFORM);       /* update display */
    sleep(1);
}


/*
 *  Initialize the pick device.
 */
initialize_input()
{

    static Ppickpath init_pick_path = { 0, (Ppickpathel *)NULL };
    static Pint name[1] = { PICKABLE };
    Ppickrec    record;
    Plimit      area;
    Pintlst     infilt, exfilt;
    Pint        pet = -1;

    area.xmin = area.ymin = 0.0;
    area.xmax = area.ymax = 1.0;

    record.upickpet1_datarec.highlight_colour = BLACK;
    record.upickpet1_datarec.highlight_count = 3;
    record.upickpet1_datarec.highlight_duration = 0.1;
    /* The pick aperture is defined in NPC, which ranges from 0 to 1.
     * The default size of the window is 600 pixels.
     * To set the aperture to a square 3 pixels on a side,
     * centered around the cursor's "hot spot", we calculate:
     * 3/600. is 0.005.
     */
    record.upickpet1_datarec.aperture_size.x = 0.005;
    record.upickpet1_datarec.aperture_size.y = 0.005;
    record.upickpet1_datarec.aperture_size.z = 0.005;

    pinitpick( ws, devid, PP_NOPICK, &init_pick_path, pet, &area,
           &record, PTOP_FIRST);

    infilt.number = 1;
    infilt.integers = name;
    exfilt.number = 0;
    psetpickfilter( ws, devid, &infilt, &exfilt);
}

/*
 *  Use request mode picking routine to pick jet parts.
 *  preqpick will return the pickid which is the part number,
 *  the element number, and the depth from posted structure to
 *  the structure containing the picked element.  Here, they are
 *  the same structure, so the depth will be 1.
 */
Pqpickstatus
request_pick()
{
    Pqpick  pick;
    Ppickpathel element[1];
    pick.pick.pick_path = element;
    preqpick( ws, devid, 1, &pick);
    if (pick.status == PQP_OK) {
        printf( "Pick: element number=%d\n", element[0].el_num);
        printf( "Pick: pick id is the part number: %d\n",
                element[0].pick_id);
    } else if (pick.status == PQP_NOPICK)
        printf( "Nothing under cursor\n");
    else if (pick.status == PQP_NONE) {
        printf( "Operator did CTRL-D break action.\n");
    } else {
        printf( "Should never happen.\n");
        pemergencyclosephigs();
        exit(1);
    }
    return( pick.status);
}


/*
 *  Close down workstation and PHIGS.
 */
close_phigs()
{
    pclosews(ws);
    pclosephigs();
}
```

B.2. FORTRAN Examples

The following example programs use the *SunPHIGS* FORTRAN binding.

ffigs1.f

A skeleton *PHIGS* program.

```
c  @(#)ffigs1.f 2.1 88/06/02 SMI

c  ffigs1.f - A simple skeleton PHIGS program in FORTRAN
c  which demonstrates how to open and close a SunPHIGS workstation.

c  Note:  It is necessry to include the declarations in phigs77.h
c  in order to open a workstation or access the :SunPHIGS constants.

      include '/usr/include/phigs/phigs77.h'

c  Open PHIGS using logical unit number 6 for the SunPHIGS error file.

      call popph(6, 0)

c  Open a workstation with a value of 1 and pause for 5 seconds.

      call popwk(1, 0, phigswsttool)
      call sleep (5)

c  Close workstation 1 and close PHIGS.

      call pclwk(1)
      call pclph

      stop
      end
```

ffigs2.f

Demonstrates the use of structures.

```
c  @(#)ffigs2.f 2.1 88/06/02 SMI

c  ffigs2.f - Demonstrates the use of structures
c  and structure elements in displaying a simple polyline.

        include '/usr/include/phigs/phigs77.h'

c  Define a series of points in the x-y axes for a polyline to be displayed.

        real xpoints(2), ypoints(2)
        data xpoints /0.1, 0.9/
        data ypoints /0.5, 0.5/

c  Open PHIGS, open a workstation and open structure 1.

        call popph(6, 0)
        call popwk(1, 0, phigswsttool)
        call popst(1)

c  Insert the polyline structure element into the open structure.
c  The first parameter is the number of points to be used to define
c  the polyline.  The other parameters are the arrays which contain
c  the x-y coordinates.  Close the stucture.

        call ppl(2, xpoints, ypoints)
        call pclst

c  Post the structure to the workstation in order to
c  have the structure's contents displayed; pause.

        call ppost(1, 1, 0.)
        call sleep(5)

c  Close the workstation and close PHIGS.

        call pclwk(1)
        call pclph
```

ffigs3.f

Demonstrates primitive attributes.

```
c  @(#)ffigs3.f 2.1 88/06/02 SMI

c  ffigs3.f - Demonstrates defining output primitives attributes
c  to change the appearance of a simple polyline.

        include '/usr/include/phigs/phigs77.h'

c  Define a series of points in the x-y axes for a polyline to be displayed.

        integer colorindex
        real xpoints(2), ypoints(2), linewidth
        data xpoints /0.1, 0.9/
        data ypoints /0.5, 0.5/
        linewidth  = 5.0
        colorindex = 5

c  Open PHIGS, open a workstation and open structure 1.

        call popph(6, 0)
        call popwk(1, 0, phigswsttool)
        call popst(1)

c  Insert the polyline structure element into the open structure.
c  The first parameter is the number of points to be used to define
c  the polyline.  The other parameters are the arrays which contain
c  the x-y coordinates.  Close the stucture.

        call ppl(2, xpoints, ypoints)

c  Create structure elements to set the linetype, the linewidth scale factor
c  and the polyline color index; move the x-y coordinates and redraw the
c  polyline.   Close the structure after all elements have been inserted.

        call psln(PLNDOTDASHDOT)
        call pslwsc(linewidth)
        call psplci(colorindex)

        ypoints(1) = ypoints(1) + .2
        ypoints(2) = ypoints(2) + .2
        call ppl(2, xpoints, ypoints)
        call pclst

c  Post the structure to the workstation in order to
c  have the structure's contents displayed; pause.

        call ppost(1, 1, 0.)
        call sleep(5)

c  Close the workstation and close PHIGS.

        call pclwk(1)
        call pclph
```

ffigs4.f

Demonstrates bundled and primitive attributes.

```
c  @(#)ffigs4.f 2.1 88/06/02 SMI

c  ffigs4.f - This program demonstrates how to create a fillarea set
c  with and without edges using the workstation's bundle table.

      include '/usr/include/phigs/phigs77.h'

      implicit undefined (P,p,E,e)

      integer i, WS1, WS2, tool1, tool2, strid
      integer edgeindex, edgetype, edgecolor
      integer boundaries, hatchindex, colourindex
      real edgewidth
      real fasxarr(12), fasyarr(12), diamond1x(4), diamond1y(4)
      real diamond2x(4), diamond2y(4), diamond3x(4), diamond3y(4)
      dimension boundaries(3)

c  Fill the boundaries array elements with the cumulative number of
c  array elements in each set of points (i.e., 4 + 4 = 8 + 4 = 12)

      data boundaries /4, 8, 12/
      data hatchindex /-5/
      data colourindex /6/

c  Fill each point set array that will be included in the fill area set arrays.

      data diamond1x /0.0, 0.5, 1.0, 0.5/
      data diamond1y /0.5, 0.0, 0.5, 1.0/
      data diamond2x /0.2, 0.5, 0.8, 0.5/
      data diamond2y /0.5, 0.2, 0.5, 0.8/
      data diamond3x /0.4, 0.5, 0.6, 0.5/
      data diamond3y /0.5, 0.4, 0.5, 0.6/

c  Fill the fill area set arrays with the data from each point set array.

      do 50 i = 1,4
            fasxarr(i)    = diamond1x(i)
            fasyarr(i)    = diamond1y(i)
            fasxarr(i+4)  = diamond2x(i)
            fasyarr(i+4)  = diamond2y(i)
            fasxarr(i+8) = diamond3x(i)
            fasyarr(i+8) = diamond3y(i)
50        continue

c  Initialize values for the workstation and structure identifiers
c  and the parameters for setting the edge bundled attribute values.

      WS1 = 1
      WS2 = 2
      strid = 1
      edgeindex = 1
      edgetype = 1
      edgewidth = 1.0
      edgecolor = 1
```

```
c  PHIGS must be opened prior to calling any other PHIGS function.

      call popph(6, 0)

c  Create two sun_tool workstations and open them.

      call phigswstcreate(phigswsttool, tool1)
      call phigswstset(tool1, PHIGSTOOLWIDTH, 500)
      call phigswstset(tool1, PHIGSTOOLHEIGHT, 500)
      call phigswstset(tool1, PHIGSTOOLX, 50)
      call phigswstset(tool1, PHIGSTOOLY, 50)
      call phigswstcreate(tool1, tool2)
      call phigswstset(tool2, PHIGSTOOLX, 600)
      call phigswstset(tool2, PHIGSTOOLY, 300)
      call popwk(WS1, 0, tool1)
      call popwk(WS2, 0, tool2)

c  Use workstation 1's bundle table to turn the SET EDGE FLAG off.
c  The workstation's bundle table edge flag attribute will need to be
c  changed from the default value (ON) to off.  See the PHIGS WORKSTATION
c  DESCRIPTION TABLE (7P) manual page for workstation default values.

c  The SET EDGE REPRESENTATION defines an edge attribute bundle on the
c  specified workstation.  This determines the edgetype, width and color
c  as well as whether or not the edge is displayed.  The function below uses
c  the edge flag value POFF to specify the edge is not to be displayed on WS1

      call psedr(WS1, edgeindex, POFF, edgetype, edgewidth, edgecolor)

c  Open a structure and begin inserting elements.  The SET EDGE INDEX functic
c  creates a structure element containing the an edge index value which selec
c  an entry from the workstation's edge bundle table.

      call popst(strid)
      call psedi(edgeindex)

c  Use the SET INDIVIDUAL ASF function to insert a structure element
c  containing the Aspect Source Flag value which determines whether the
c  primitive's individual attribute value or the workstation's bundle table
c  attribute value will be used.  Here we designate the bundled attribute
c  for the edge flag.

      call psiasf(PEDFG, PBUNDL)

c  Create the individual attribute value elements for the fill area set usinc
c  SET INTERIOR STYLE, SET INTERIOR STYLE INDEX and SET INTERIOR COLOUR INDEX
c  Insert the fill area set element into the structure and close it.

      call psis(PHATCH)
      call psisi(hatchindex)
      call psici(colourindex)
      call pfas(3, boundaries, fasxarr, fasyarr)
      call pclst

c  Post the structure to both workstations to display the fill area set data.

      call ppost(WS1, strid, 0.)
      call pmsg(WS1, 'Displays a fill area set with the edge flag off.')
      call ppost(WS2, strid, 0.)
      call pmsg(WS2, 'Displays a fill area set with the edge flag on.')
      call sleep(10)

c  Close both workstations and PHIGS.

      call pclwk(WS1)
      call pclwk(WS2)
      call pclph

      stop
      end
```

figscanval.f

Demonstrates the *Valuator* window using a *Sun Canvas* workstation.

```
c  @(#)figscanval.f 2.2 88/07/08 SMI

c  figscanval.f - Create a phigswstcanvas workstation type and set attributes.

c  Note:  It is necessry to include the declarations in phigs77.h
c         in order to open a workstation or access SunPHIGS constants.

        include '/usr/include/phigs/phigs77.h'

c  Turn off implicit typing of possible SunPHIGS names.
c  This will cause the compiler to report undeclared usage of such names.

        implicit undefined (P,p,E,e)
        character *80 rec(1)
        integer wkid, strid, valdev, pet, canvas, canvasid, canvaswst, ldr

c  Initialize variables for the workstation identifier & structure indentifer
c  and the valuator device and prompt/echo type (for INITIALIZE VALUATOR).

        wkid = 1
        strid = 1
        valdev = 1
        pet = 1
        ldr = 0

c   SunView Setup - canvasid is a C function which returns the
c   connection identifier 'canvas' for a SunPHIGS canvas workstation.

         canvas = canvasid()

c  Open PHIGS, use logical unit number 6 for the SunPHIGS error file.

        call popph(6, 0)

c  Create a new workstation of type phigswstcanvas; set the attribute values
c  for positioning the VALUATOR window; open the workstation.

        call phigswstcreate(phigswstcanvas, canvaswst)
        call phigswstset(canvaswst, PHIGSVALPANELX, 625)
        call phigswstset(canvaswst, PHIGSVALPANELY, 25)
        call popwk(wkid, canvas, canvaswst)

c  INITIALIZE VALUATOR and SET VALUATOR MODE to activate the device.

        call pinvl(wkid, valdev, 0.5, pet, 0.0,1.0,0.0,1.0, -1.0,1.0, ldr,rec)
        call psvlm(wkid, 1, PEVENT, PECHO)

c  Open a structure and insert a SET CHARACTER HEIGHT and TEXT element.

        call popst(strid)
        call pschh (.015)
        call ptx(.01, .5,
&       "This is a phigswstcanvas workstation with a VALUATOR window.")
```

```
c  Post structure to the workstation to display the structure's contents.

        call ppost(wkid, strid, 0.)

c  Call SunView's notifier from canvasid() to update the SunView canvas.

        call display

c  Close the structure, close the workstation, and close PHIGS.

        call pclst
        call pclwk(wkid)
        call pclph

        stop
        end
```

figstoolval.f

Demonstrates the *Valulator* window using a *Sun Tool* workstation.

```
c  @(#)figstoolval.f 2.1 88/06/02 SMI

c  figstoolval.f - Create a phigswsttool with a VALUATOR window.

c  Note:  It is necessry to include the declarations in phigs77.h
c         in order to open a workstation or access SunPHIGS constants.

      include '/usr/include/phigs/phigs77.h'

c  Turn off implicit typing of possible SunPHIGS names.
c  This will cause the compiler to report undeclared usage of such names.

      implicit undefined (P,p,E,e)
      character *80 rec(1)
      integer wkid, strid, valdev, pet, toolval, ldr

c  Initialize variables for the workstation identifier & structure indentifer
c  and the valuator device and prompt/echo type (for INITIALIZE VALUATOR).

      wkid = 1
      strid = 1
      valdev = 1
      pet = 1
      ldr = 0

c  Open PHIGS, use logical unit number 6 for the SunPHIGS error file.

      call popph(6, 0)

c  Create a new workstation of type phigswstool; set the attribute values
c  for positioning the VALUATOR window; open the workstation.

      call phigswstcreate(phigswsttool, toolval)
      call phigswstset(toolval, PHIGSVALPANELX, 650)
      call phigswstset(toolval, PHIGSVALPANELY, 25)
      call popwk(wkid, 0, toolval)

c  INITIALIZE VALUATOR and SET VALUATOR MODE to activate the device.

      call pinvl(wkid, valdev, 0.5, pet, 0.0,1.0,0.0,1.0, -1.0,1.0, ldr,rec
      call psvlm(wkid, valdev, PEVENT, PECHO)

c  Open a structure and insert a SET CHARACTER HEIGHT and TEXT element.

      call popst(strid)
      call pschh (.015)
      call ptx(.01, .5,
&     "This is a phigswsttool workstation with a VALUATOR window.")

c  Post structure to the workstation to display the structure's contents.

      call ppost(wkid, strid, 0.)
      call sleep(10)
```

```
c  Close the structure, close the workstation, and close PHIGS.

      call pclst
      call pclwk(wkid)
      call pclph

      stop
      end
```

C

Viewing

Viewing **183**

C.1. View Orientation 183

C.2. View Mapping 184

C

Viewing

PHIGS provides utility functions which compute 4 x 4 and 3 x 3 matrices suitable for use as view orientation matrices and view mapping matrices. The parameters for these utility functions are based on 3-D and 2-D models of orientation and projection. These models explain how certain parameters are used to determine 4 x 4 matrices. *PHIGS* supports only certain models. Other models are possible and can be utilized by an application by creating its own matrices in accordance with the desired model.

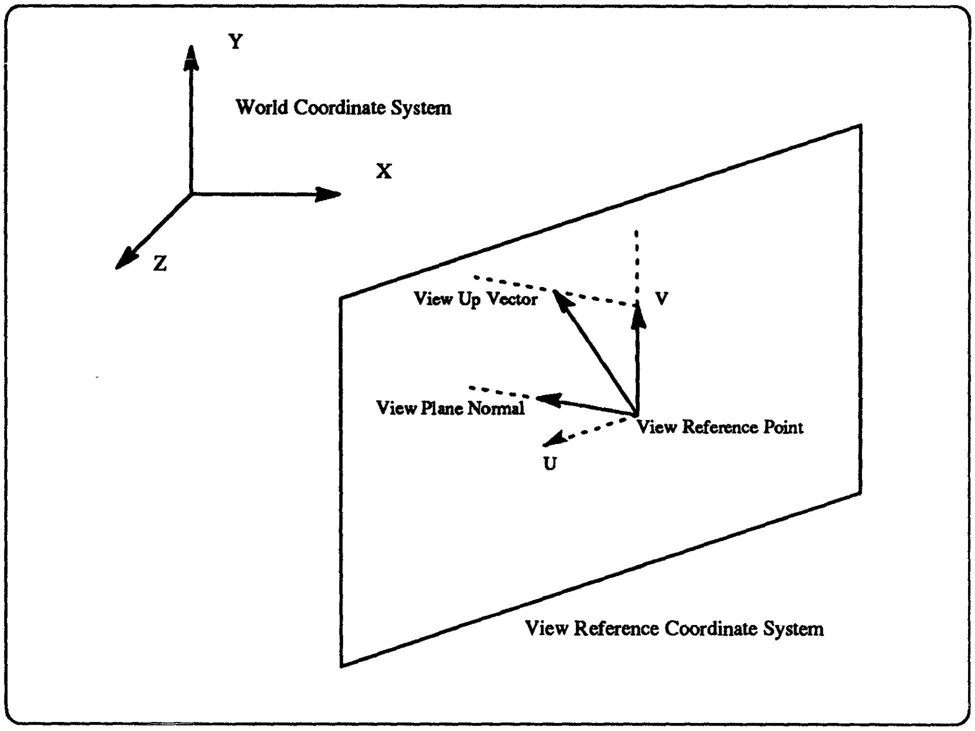
The EVALUATE VIEW ORIENTATION MATRIX and EVALUATE VIEW MAPPING utility functions have two output parameters, a matrix and an error indicator. The matrix is a 4 x 4 or 3 x 3 depending on whether the utility function is 3-D or 2-D. The error indicator will be set to indicate the cause of the error if the input parameters are not well-defined or are inconsistent.

C.1. View Orientation

The model for view orientation provides for positioning and orientation of the *View Reference Coordinate* (VRC) system with respect to the *World Coordinate* (WC) system. The 3-D model is supported by the function VALUATE VIEW ORIENTATION MATRIX 3. The input parameters of the function are:

- □ *view reference point* - in WC, defines the origin of the VRC. The view reference point is typically a point on or near the object to be viewed.
- □ *view plane normal* - a vector relative to the view reference point that defines the N axis of the VRC system. The axes of the View Reference Coordinate system are the UVN axes as described in Figure C-1 below.
- □ *view up vector* - a vector relative to the view reference point which is projected onto the view reference plane via a projection parallel to the view plane normal. The projection of the view up vector onto the view reference plane determines the V axis of the VRC system. The U axis is determined such that the UVN axes form a right-handed coordinate system.

Figure C-1 *View Reference Coordinate System*



The 2-D model for view orientation provides for positioning and orientation of the VRC system in the x-y plane of the WC system. The 2-D model is supported by the function EVALUATE VIEW ORIENTATION MATRIX and its input parameters are:

- *view reference point* - a 2-D point that defines a point in the WC Z=0 plane. This point becomes the origin of the View Reference Coordinate system.
- *view up vector* - a 2-D vector relative to the view reference point that indicates a direction in the WC Z= 0 plane. This direction becomes the V axis of the VRC. The N axis of the VRC system is parallel to the Z axis of the WC system. The U axis of the VRC system is determined such that the UVN axes from a right-handed coordinate system.

C.2. View Mapping

The 3D model for view mapping provides for parallel and perspective transformation of the *View Reference Coordinate* (VRC) system to the *Normalized Projection Coordinate* (NPC) system. This model is supported by the function EVALUATE VIEW MAPPING MATRIX 3. This function maps a volume in VRC called the *view volume*, to a volume in NPC bounded by *projection viewport limits*. EVALUATE VIEW MAPPING MATRIX 3 returns the 4 x 4 matrix which performs this mapping. Figures C-2 and C-3 below show typical view volumes associated with parallel and perspective views. The input parameters for this function are:

- *window limits* - the limits of a rectangular region on the view plane with sides parallel to the U and V axes. The view window is specified as U and V coordinate values. The window limits parameter consists of four values, usually referred to as UMIN, UMAX, VMIN and VMAX. By definition the UMIN edge is the left edge of the view window, the UMAX edge is the right edge, the VMIN edge is its bottom edge and the VMAX edge is its top edge. The left, right, bottom and top edges of the view window, together with the projectors through those edges, define the left, right, bottom and top surfaces of the view volume, respectively. In this model for view mapping, the term *projector* refers to an infinite line in VRC that is mapped (via the view mapping) to an infinite line in NPC of the form X=constant, Y=constant, Z arbitrary.

- □ *projection viewport limits* - the limits of a rectangular parallelpiped in NPC space with edges parallel to the NPC axes. Normalized Projection Coordinate space *conceptually* extends beyond [0,1] x [0,1] x [0,1], however, the part of NPC in which the view clipping limits shall be located is the closed unit cube [0,1] x [0,1] x [0,1]. The six NPC values defining the projection viewport are referred to as XMIN, XMAX, YMIN, YMAX, ZMIN and ZMAX. Typically the projection viewport limits and the view clipping limits will be set to the same values, however, this is not mandatory.

- □ *projection type* - an enumerated type with the values PARALLEL and PERSPECTIVE.

- □ *projection reference point* (PRP) - a position in VRC space which serves to orient the projectors defining the surfaces of the view volume. If the *projection type* is PARALLEL, the projectors are all parallel to the vector joining the projection reference point and the center of the view window. The view volume, therefore, is, a parallelpiped. If the *projection type* is PERSPECTIVE, the projectors all pass through the projection reference point. In the case of the latter, the projectors passing through the view window lie on the surface of a pair of infinite rectangular cones having their common vertex at the PRP. The view volume, therefore, is a portion of this double cone.

- □ *view plane distance, back plane distance* and *front plane distance* - N coordinate values which specify the planes parallel to the UV plane of the View Reference Coordinate system. The front plane and back plane contain the front and back of the view volume. Conceptually, the VRC space is *oriented* (since View Reference Coordinates result from the *view orientation transformation*). Since the front plane should not be behind the back plane, the front plane distance should not be less than the back plane distance.

Figure C-2 *The Parallel Viewing Model*

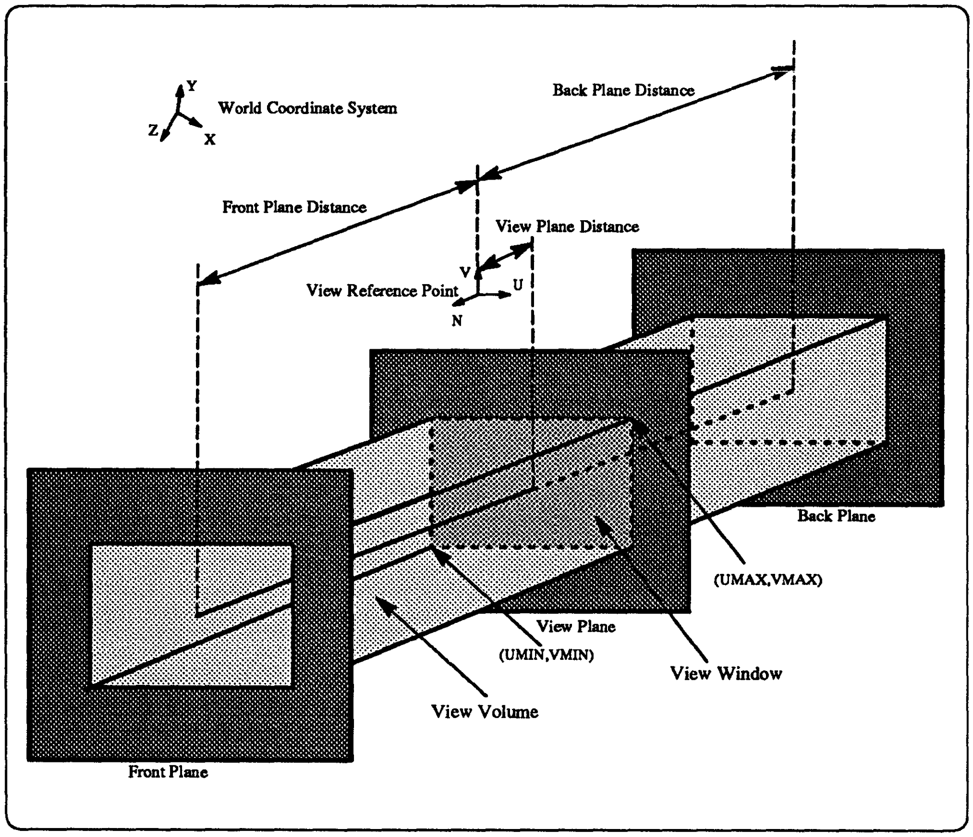
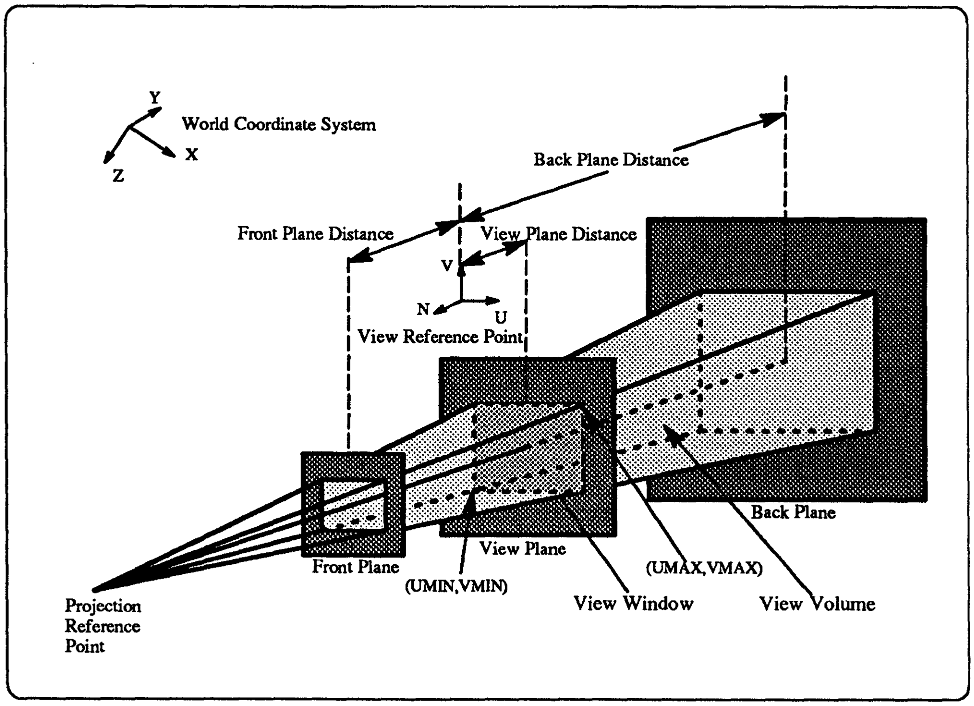


Figure C-3 *The Perspective Viewing Model*



The 2-D model for view mapping provides for parallel transformation of the VRC system to the NPC system. The EVALUATE VIEW MAPPING MATRIX function supports this model which permits the application to specify a rectangle on the N=0 plane in VRC, and to specify a rectangle on the Z=0 plane in NPC. EVALUATE VIEW MAPPING MATRIX returns the 3 x 3 matrix which is used to map the VRC rectangle to the NPC rectangle. The input parameters for this function are:

□ *window limits* - the limits of a rectangular region in VRC which is upright (sides parallel to the U and V axes) and located on the N=0 plane. The view window is specified as U and V coordinate values. The window limits parameter consists of four values called UMIN, UMAX, VMIN and VMAX.

□ *projection viewport limits* - the limits of a rectangle in NPC which is upright (sides parallel to the X and Y axes) and located on the Z=0 plane. Although the NPC *conceptually* extends beyond [0,1] in the X and Y axis, the NPC rectangle is located in the closed unit square [0,1] x [0,1]. The four values defining the NPC rectangle are XMIN, XMAX, YMIN and YMAX.

Notes

Notes

Systems for Open Computing™

Corporate Headquarters
Sun Microsystems, Inc.
2550 Garcia Avenue
Mountain View, CA 94043
415 960-1300
FAX 415 969-9131

For U.S. Sales Office locations, call:
800 821-4643
In CA: 800 821-4642

European Headquarters
Sun Microsystems Europe, Inc.
Bagshot Manor, Green Lane
Bagshot, Surrey GU19 5NL
England
0276 51440
TLX 859017

Australia: (02) 413 2666
Belgium: 32-2-759 5925
Canada: 416 477-6745
France: (1) 40 94 80 00
Germany: (089) 95094-0
Hong Kong: 852 5-8651688
Italy: (39) 6056337
Japan: (03) 221-7021
Korea: 2-563-8700
New Zealand: (04) 499 2344
Nordic Countries: +46 (0)8 7647810
PRC: 1-8315568
Singapore: 224 3388
Spain: (1) 5551648
Switzerland: (1) 8289555
The Netherlands: 033 501234
Taiwan: 2-7213257
UK: 0276 62111

Europe, Middle East, and Africa, call European Headquarters:
0276 51440

Elsewhere in the world, call Corporate Headquarters:
415 960-1300
Intercontinental Sales