19941106 (this version 19950101)
Bo Nordin (bosse@cb.uu.se)
Centre for Image Analysis
018-183470

Brief PHIGS Reference

Introduction

This document is essentially a listing of the two files *phigsextrn.c* (functions) and *phigs.h* (datatypes, enumerations, and constants) but also an introduction to PHIGS. There are many more PHIGS functions available than those listed here. I have removed all functions dealing with archives, gse's, application data, and Store objects, as well as many gdp functions. I have also removed a few cases of C++ specialisations. This document consists of two parts: the first is a list of PHIGS functions, ordered after function, the second is a list of PHIGS datatypes, in alphabetical order.

In order to find out how the arguments to a particular function should be given, look it up in the datatype section. Some Pints are not really integers, but rather enumerations; such constants can be found in the section PHIGS constants after PHIGS datatypes. I have identified most of them (I think) in the function prototypes; I have called them __xxx in the comments.

NOTE: there are two versions of the document: one full version and one version without the function and datatypes description.

PHIGS Overview

This section is a very brief version of *PHIGS Programming Manual* by Tom Gaskins. What I'm describing here is PHIGS+ with some SUN extensions. PHIGS+ features are generally distinguished by either of the words plus or extended.

How to draw with PHIGS

PHIGS is a tool for *visualising models*. You describe your model and display it on a workstation (a workstation in PHIGS is *not* identical to a SUN workstation, for instance; it is a window on a workstation rather). There are four steps in this:

1. define the model in the PHIGS database, the *CSS (central structure storage)*
2. create a PHIGS workstation to display the model
3. link the model's description to the workstation
4. tell the workstation to display the model.

This is done by the following PHIGS program (OK, it's only a skeleton of a program):

```
popen_phigs()   /* open phigs, parameter are "fixed" */
popen_struct()
...
/* create a structure with attributes and primitives, i.e. 1. above */
...
pclose_struct()
```

```
popen_ws();  /* 2: open a workstation */
ppost_struct(); /* 3. post the model to the workstation */
pupd_ws(); /* 4. update the workstation */
.....
punpost_struct();
pclose_ws();
pclose_phigs();
```

Of course, there's a lot more to drawing in PHIGS but these are the basic steps.

PHIGS datatypes and functions

The file phigs.h contain all PHIGS' datatypes and phigsextrn.h all its functions. In PHIGS, the names of datatypes all begin with a capital "P" (Ppoint3, for instance) while function names begin with the lower-case letter "p" (ppolyline, for instance).

In addition to datatypes and functions, PHIGS also defines mnemonic names for many constants. These identifiers are all upper-case letters, as PIN_PICK, for instance.

Structures and traversal

Structures are opened with popen_struct which also gives each structure a unique name (which you select). You close a structure with pclose_struct. Only one structure may be opened at any time. Once you have created a structure it must be associated with a workstation and displayed. You associate it with a workstation with ppost_struct. It tells PHIGS that a structure is to be displayed on a particular workstation. It's not displayed until the workstation is updated, though. Either you update the workstation explicitly by calling pupd_ws (or predraw_all_structs) or you direct the workstation to automatically update with every change by calling pset_disp_upd_st.

In fact, what you are doing when adding primitives or attributes (*elements*) to an open structure is creating a small program. When PHIGS displays the structure, it steps through the elements of the structure. If the element is an attribute PHIGS sets that attribute and if it is a primitive PHIGS draws it (using the current attributes). This process is called *traversal* and is what happens when a workstation is updated. Elements may be other things than attributes and primitives, as we shall see.

One important element type is the invocation of another structure. This is called executing another structure and is done with pexec_struct.

Primitives

PHIGS can draw the following primitives:

Line primitives:
  polylines (ppolyline, ppolyline3, ppolyline_set3_data),
  B-spline curve (pnuni_bsp_curv)
Area primitives:
  fill area (pfill_area, fill_area3, pfill_area_set, pfill_area_set3_data,
    ptri_strip3_data, pquad_mesh3_data, pnuni_bsp_surf)
Text primitives:

general: (ptext, ptext3)
annotation (panno_text_rel, panno_text_rel3)
Other primitives:
polymarker (ppolymarker, ppolymarker3)
cell array (pcell_array, pcell_array3, cell_array3_plus)
generalized drawing primitive (pgdp, pgdp3)

The difference between text and annotation text is that annotation text is always displayed in the plane of the screen and thus remain legible regardless of how the model is rotated or scaled.

Colour

Colour is probably the most important attribute and is therefore treated separately. I here assume that PHIGS runs a colour workstation. In PHIGS you specify a colour with a *colour type* and a *colour value*. These are passed to PHIGS in a Pgcolr structure:

```
typedef struct { Pint type;  union { Pint ind; struct { Pfloat x, y, z } general; } val; } Pgcolr;
```

The type field is for the colour type and the val union is the value. In this document I will assume that type is PMODEL_RGB or PINDIRECT. The difference between the two is in how a colour is defined. If type is PMODEL_RGB the val union member general contains the red (x), green (y), and blue (z) components of the colour. If type is PINDIRECT the union member ind is an index in a colour table. You set entries in the colour table with pset_colr_rep. As all indexed attributes, colour indices, too, are workstation dependent. In the lists of attribute setting functions below the suffix _ind indicates a colour index (pset_line_colr thus takes a full colour definition as argument while pset_line_colr_ind takes a colour index).

It is important to understand the difference between direct and indirect colour: the same type of indirection is used for bundled attributes and for views.

Bundled attributes

In PHIGS, attributes can be set either individually or in groups (or *bundles*). Bundling provides a way to control attributes. There are separate bundles for each workstation for line, interior (area), edge, text, and marker attributes. You select bundles with bundle index structure elements. There is a separate element for each bundle. You define bundles with one function and select (i.e. create an element in a structure) them with another

Many of the bundles have a shortform for setting them in addition to the full form. Line attributes for instance have the two forms pset_line_rep_plus (full form with all attributes) and pset_line_rep (shortform with only some attributes). The defined bundle is then selected with pset_line_ind.

Bundles contain most of the attributes relevant to their corresponding primitive type. But you can select which of these attributes are taken from the bundle and which are taken from the currently defined values in the traversal list. You make this selection with aspect source flags (aspect is just another name for attribute). Use the function pset_indiv_asf for this.

Bundles are workstation-specific and there exist a number of predefined bundles on each workstation.

Line attributes

Lines have the following attributes:

linetype	pset_linetype
linewidth	pset_linewidth,
linecolour	pset_line_colr, pset_line_colr_ind
line shading method:	pset_line_shad_meth
curve appr. criteria:	pset_curve_approx (b-spline curves only)

The bundled form is selected with pset_line_ind and define with pset_line_rep or pset_line_rep_plus.

Area attributes

Area have the following attributes (this is only a selection):

edge colour:	pset_edge_colr, pset_edge_colr_ind
edge flag:	pset_edge_flag (controls whether edges are displayed or not)
edgetype:	pset_edgetype
edgewidth:	pset_edgewidth
face culling mode:	pset_face_cull_mode
interior colour:	pset_int_colr, pset_int_colr_ind
interior style:	pset_int_style
pattern ref point:	pset_pat_ref_point
pattern reference:	pset_pat_ref_point_vecs
pattern size:	pset_pat_size
pattern:	pset_pat_rep, pset_pat_rep_plus (workstation dependent)

There are two bundles of area primitives. These are, with selection and definition functions:

interior:	pset_int_int	pset_int_rep, pset_int_rep_plus.
edge:	pset_edge_int	pset_edge_rep, pset_edge_rep_plus

The edge bundle contains the first four area attributes while the interior bundle contains colour and style plus many more.

Marker attributes

Marker has the following attributes:

type:	pset_marker_type
size:	pset_marker_size
colour:	pset_marker_colr, pset_marker_colr_ind

The bundled form is selected with pset_marker_ind and defined with pset_marker_rep or pset_marker_rep_plus.

Text attributes

Text has the following attributes:

font:	pset_text_font
character height:	pset_char_ht
character spacing:	pset_char_space
character expansion:	pset_char_expan
character up vector:	pset_char_up_vec
path:	pset_text_path
alignment:	pset_text_align
precision:	pset_text_prec
colour:	pset_text_colr, pset_text_colr_ind

The bundled form is selected with pset_text_ind and defined with pset_text_rep or pset_text_rep_plus.

Annotation text has essentially the same functionality. One additional attribute is annotation style (pset_anno_style).

Modelling

The modelling stage is the creation of graphical objects in their own coordinate system and then transforming them to the world coordinate system. The local transform is the work-horse of the modelling stage. You use to position, scale, and rotate primitives into place. It can be regarded as just another attribute. You can create a local transform with pset_local_tran(3). There are help functions to create transformation matrices:

protate_x, protate_y, protate_z, pscale(3), ptranslate(3), protate, pbuild_tran_matrix(3)

Matrices can be concatenated with pcompose_matrix(3).

Building hierarchies is an important part of modelling. This is done by invoking other structures (pexec_struct). If you build a street with houses, for instance, you would start by building a window struct, a door struct, and a chimney struct in their own coordinate system, then build a house in its coordinate system. In the street struct you would then set a local transformation matrix before each invocation of house. In the house struct you would select a different local transformation for the chimney, each window, and the door. The two local transformations will be autmatically concatenated.

Viewing

Once a model is assembled, we'd like to see it from various directions. If modelling is the transformation of the model (objects are moved), viewing is the transformation of the viewer (you, the viewer, moves). PHIGS uses the model described in Foley, chapter 6, for viewing. The view reference coordinate system defines the *view orientation* and the choice of projection, the view volume and the projection to a viewport define the *view mapping*.

You evaluate the view orientation matrix with peval_view_ori_matrix(3) and the view

mapping matrix with peval_view_map_matrix(3). You can then define your view with pset_view_rep3.
This last function defines a view representation index much like colour indices or bundle indices. You select the view index for a view with pset_view_ind. You cannot change view index 0 (this is also the default).

CSS, Structure editing, and inheritance rules

One important feature of PHIGS is the possibility to edit structures after they have been created. But first a little about inheritance rules: how do executed structures inherit attributes, transformations, etc.

Executed structures inherit the attributes, transformations, names (see Input), aspect source flags, bundle indices, etc. from their ancestors. Changes by children do not affect ancestors, i.e. attributes, etc. are restored.

When you call popen_struct the element pointer is set to point after the last element. PHIGS has a number of functions for resetting the pointer, deleting elements, replacing elements (by changing the edit mode). In particular, you can insert a special kind of element (label) which you can use to position the pointer. In SUN PHIGS you can also move the element pointer to a pick element (see Input). Because elements that we want to remove often are made pickable this last function largely removes the need for labels.

Input

PHIGS input isn't as sophisticated as it could be. Gaskins suggests that we should use X input whenever possible. But here I will concentrate on PHIGS' input functions.

There are six types of logical input devices in PHIGS:
locator *(loc)*, *stroke*, *pick*, *choice*, valuator *(val)*, and *string*
All devices may be used in three different modes:
request, *sample*, or *event*.
You set the mode with pset_xxx_mode where xxx is one of the italicised items above. Unless you want to change default parameters for a device you don't have to initialise it explicitly.

Request mode functions are preq_xxx, sample mode function are psample_xxx. where xxx is one of the six modes.

In event mode we use pawait_event in the event loop. One of the output parameters from this function tells us what device triggered the event. You use this information to select one of the functions pget_xxx.

For locator input, PHIGS automatically converts device coordinate points to world coordinate points.

Pick devices are very useful but also hard to use. This is because of the hierarchical model that PHIGS use and also because of the (I must admit) overly complicated approach PHIGS takes. In order for a object (primitive, structure, or whatever) to be pickable it must be given a name and a pick id. A name is set with padd_names_set and a pick id with pset_pick_id. Both these are modal attributes so if we want everything to be pickable we just set a name first in the main structure. This will then be inherited by all substructures. Now everything would be fine if it weren't for the pick filter. This is filter which determines which names are pickable: in other words, we must include the name (assuming we use the simple model outlined above) in the pick filter with pset_pick_filter. Any picked primitive is identified by the pick path: this path contains the structure id, pick id, and element number, for the picked primitive and all its ancestors. A variable in the returned pick path tells us the depth of the hierarchy.

If events occurred where views overlap, PHIGS returns an event in the view with the highest priority. By default this is view 0 but you can change that with pset_view_tran_in_pri.

Other functions

PHIGS has many inquiry functions. You can ask PHIGS about the state of PHIGS, workstation capabilities, elements in structures, bundles, input functionality, and more...

Other functions include hidden line and surface removal and the possibility to control in detail the rendering pipeline.

Relation between SPHIGS and PHIGS

This section contains a list of some SPHIGS functions (those used in Foley et al, chapter 7) and their counterparts in PHIGS. The list is in alphabetical order with the SPHIGS name as sort key. Together with chapter 7 in Foley et al and the on-line manual answerbook, this list can be used as a reference manual.

SPH_closeStructure	pclose_struct
SPH_composeMatrix	pcompose_matrix, pcompose_matrix3
SPH_deleteElement	pdel_elem
SPH_evaluateViewOrientationMatrix	peval_view_ori_matrix3
SPH_evaluarteViewMappingMatrix	peval_view_map_matrix3
SPH_executeStructure	pexec_struct
SPH_fillArea	pfill_area, pfill_area3
SPH_label	plabel
SPH_moveElementPointerToLabel	pset_elem_ptr_label
SPH_offsetElementPointer	poffset_elem_ptr
SPH_openStructure	popen_struct
SPH_pickCorrelate	----------
SPH_polyhedron	pset_of_fill_area3, and more...
SPH_polyLine	ppolyline, ppolyline3
SPH_polyMarker	polymarker, ppolymarker3
SPH_postRoot	ppost_struct
SPH_regenerateScreen	pupd_ws, predraw_all_structs
SPH_rotateX	protate_x, protate
SPH_rotateY	protate_y, protate
SPH_rotateZ	protate_z
SPH_scale	pscale, pscale3
SPH_setEdgeColor	pset_edge_colr, pset_edge_colr_ind
SPH_setEdgeFlag	pset_edge_flag

SPH_setEdgeStyle	pset_edgetype
SPH_setEdgeWidthScaleFactor	pset_edgewidth
SPH_setElementPointer	pset_elem_ptr
SPH_setImplicitRegenerationMode	pset_disp_upd_st
SPH_setInteriorColor	pset_int_colr, pset_int_colr_ind
SPH_setLineColor	pset_line_colr, pset_line_colr_ind
SPH_setLineStyle	pset_linetype
SPH_setLineWidthScaleFactor	pset_linewidth
SPH_setLocalTransformation	pset_local_tran, pset_local_tran3
SPH_setMarkerColor	pset_,marker_colr, pset_marker_colr_ind
SPH_setMarkerSizeScaleFactor	pset_marker_size
SPH_setMarkerStyle	pset_marker_type
SPH_setPickIdentifier	pset_pick_id
SPH_setRenderingMode	
SPH_setTextColor	pset_text_colr, pset_text_colr_ind
SPH_setTextFont	pset_text_font
SPH_setViewRepresentation	pset_view_rep3
SPH_text	ptext, ptext3, panno_text_rel, panno_text_rel3
SPH_translate	ptranslate, ptranslate3
SPH_unpostRoot	punpost_struct

An example program

This example program draws a polyline. The purpose of the program is to show what a very simple PHIGS program must contain in terms of opening PHIGS, opening workstations, etc.

```
/* Copyright 1992 O'Reilly and Associates, Inc. Permission to use, copy,
 * and modify this program is hereby granted, as long as this copyright
 * notice appears in each copy of the program source code.
 */

#include <phigs/phigs..h>
#define WORKSTATION_ID      (Pint) 1
#define STRUCTURE_ID        (Pint) 1

main() {
 Ppoint_list3     point_list;

 static Ppoint3 points() = {{.2,.2,0},{.4,.7,0},{.6,.3,0},{.8,.8,0}};

 point_list.num_points = 4;
 point_list.points = points;

 popen_phigs( PDEF_ERR_FILE, PDEF_MEM_SIZE );
 popen_struct( STRUCTURE_ID );
 pset_linetype( PLINE_DASH );
 ppolyline3( &point_list );
 pclose_struct();

 popen_ws( WORKSTATION_ID, NULL, phigs_ws_type_x_tool );
 ppost_struct( WORKSTATION_ID, STRUCTURE_ID, (Pfloat) 1.0 );
 pupd_ws( WORKSTATION_ID, PFLAG_PERFORM );

 sleep( 10 );   /* let's show it for 10 seconds */
 pclose_ws( WORKSTATION_ID );
 pclose_phigs();
}
```