3D GRAPHICS APPLICATION SYSTEM

3D GRAPHICS APPLICATION SYSTEM

by

Slawomir Nick Werczak, M. Sc. (ME)

A Project

Submitted to the Faculty of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Master of Science

McMaster University

December 1992

Master of Science

McMaster University

Hamilton, Ontario

Title: **3D GRAPHICS APPLICATION SYSTEM**

Author: **SLAWOMIR NICK WERCZAK**

Supervisor: **DR. WILLIAM F. SMYTH**

Number of Pages: vi, 144

ABSTRACT

This project is about a 3D graphics system for engineers and architects. Its goal is twofold: to provide tools for drawing 3D models, and to create a database support system where data related to these models would be stored (at present the first part has been implemented). Costs, materials, calculations of loads, stresses, and other important factors will be available to the user during the whole design process. The system will therefore allow for drawing realistic projects, not just conceptual visual models.

The drawing tools provided by the system include a few drawing techniques and viewing functions. The system's drawing routines operate on sets of points, lines, or components so that they are very fast and efficient. This approach to the method of drawing along with the description of the system are given in the following write-up.

ACKNOWLEDGEMENTS

It has been my privilege to work under the supervision of Dr. W. F. Smyth. To him, I wish to express my thanks for his ideas and inspiration during the course of my research work as well as for the constructive criticism of the various drafts of this thesis.

I would also like to acknowledge my deep appreciation to the many people at McMaster who have shared their friendship.

TABLE OF CONTENTS

CHAPTER ONE

INTRODUCTION

This write-up describes the 3D Graphical Application System (here called 3D GAS) developed to assist an engineer or architect in the design phase of three-dimensional structures such as bridges, buildings, and steel transmission towers. Like other CAD systems 3D GAS eliminates the tedious and inefficient manual work involved in the drafting process and provides an easy way for storing, retrieving, displaying and manipulating the edited models. 3D GAS is, however, not just a replica of existing CAD systems. It is a powerful tool in the design process. Most engineering or architectural structures demonstrate symmetry or they consist of a number of identical components. 3D GAS takes advantage of this fact. It allows for the simultaneous drawing of multiple points, lines or whole structures as well as copying, manipulation and insertion of the components into the edited structure.

The project described here is only a first stage of a larger system and further evolution is expected. In the process of engineering or architectural design many factors should be taken into consideration—obviously the appearance of the final product is not just the pure implementation of the idea of the designer. Other aspects such as loads, stresses, available materials, etc. affect the design process and cannot be considered separately. These data must be available during the whole process of design. As it is now, 3D GAS could be thought of as a tool for drafting, editing and archiving the 3D structures. The anticipated extension to it would consist of the set of data files in which all information relating to the structure would be stored. These

data such as lengths and location of members, directions and intensities of loads, stress resultants from analyses, properties of members selected, fabrication details, etc. will be entered by the designer or calculated by the system. The system environment should allow for data manipulation and modification while progress in the design process occurs.

Considering the future evolution of 3D GAS, a few things had to be thought of at an early stage. To allow some data of the designed structure, such as for example resultant stresses, to be processed by the system, it is obligatory that the dimensions of the structure be real. The structure created on the screen is therefore not just a model but rather a real object in an appropriate scale. Secondly, manipulation of or reference to the data will require some identification. This has been accomplished by introduction of labels for nodes and members.

It should be mentioned that 3D GAS has been influenced by the project EGAS by Hong Zhou [4], designed for the 2D graphics application. Besides the fact that 3D GAS adds one more dimension to the application, there are so many other differences in the implementation, that 3D GAS should not be considered a continuation of EGAS but rather another approach. The main concept of the drawing techniques remains, however, the same.

3D GAS is an X Window System application. It is based on two packages: the XView Toolkit and PHIGS graphics package. XView is one of the few toolkits available for the X Window System. It provides the programmer with prebuilt, reusable user interface components and is therefore responsible for the application's "look and feel". PHIGS is specifically designed to support interactive 2D and 3D graphics applications. Both XView and PHIGS as well as the main concepts of 3D GAS will be discussed in the later sections of this write-up.

1.1. ORGANIZATION OF THIS WRITE-UP

The following chapters are organized as follows:

CHAPTER 1: is the chapter you are reading now.

CHAPTER 2: introduces 3D GAS and its features. In this chapter all drawing routines available in 3D GAS will be presented and illustrated with the examples. Also the internal organization of the created structures will be explained.

CHAPTER 3: discusses the implementation of user interface controls and workstation windows used in the application.

CHAPTER 4: presents on a conceptual level the two packages used in 3D GAS: XView and PHIGS.

CHAPTER 5: discusses the present scope and the future of 3D GAS.

APPENDIX A: is a manual for the installation and compilation of 3D GAS.

APPENDIX B: contains 3D GAS source code.

APPENDIX C: user's manual (help file).

BIBLIOGRAPHY

CHAPTER TWO

3D GAS OVERVIEW

2.1. WHAT IS 3D GAS?

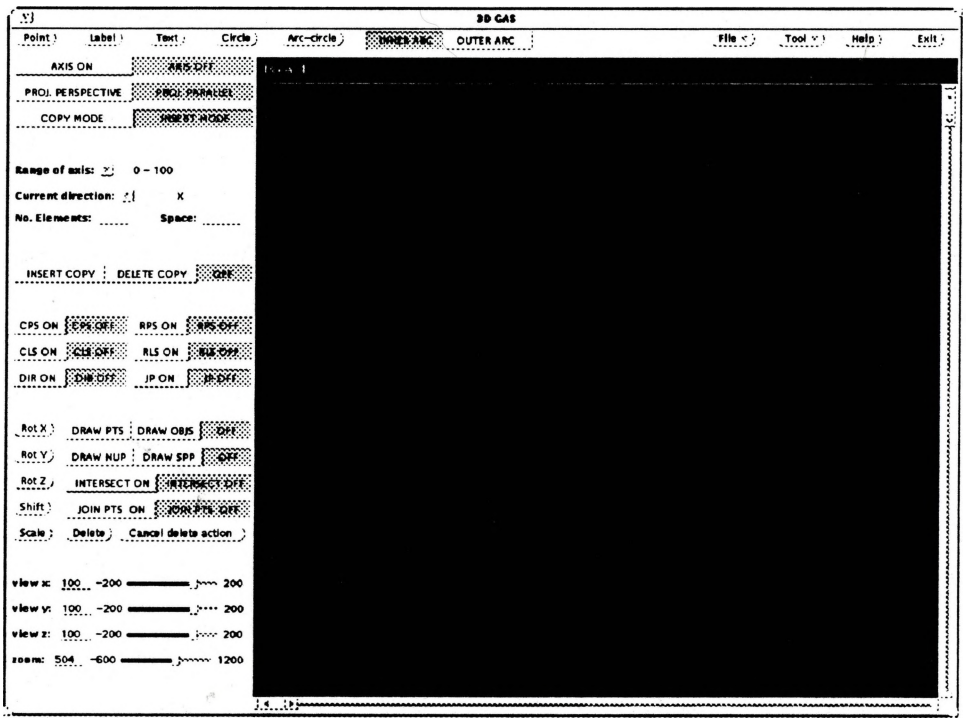

Figure 2-1. 3D GAS application screen.

3D GAS is a menu driven application. As shown in Figure 2-1, the application screen is a configuration of windows and user interface controls. The big window, called the *drawing window*, is the main window of the application. This is an area where the graphics can be displayed and the user's input is handled. The user interface controls, operated by a mouse, offer a wide range of choices and menu buttons whose

purpose is to facilitate the use of the interface.

3D GAS provides several tools which can be used to make the design process easier and more efficient. These tools are described below under the following headings:

- fast drawing techniques based on point and line sets;
- drawing modes;
- transformation and viewing functions;
- archiving and loading facilities.

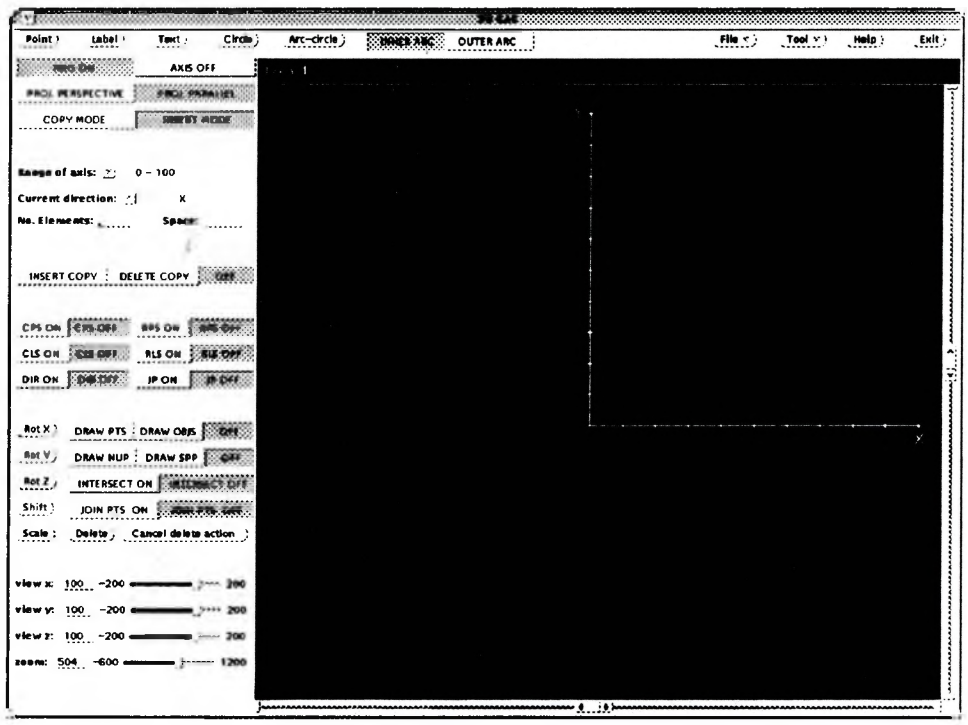

Figure 2-2. The default view.

Drawing in 3D GAS occurs in the world coordinate system. World coordinates are fixed. This has that advantage that all models created in these coordinates occupy fixed positions in the space and are relative to each other. Viewing the models using the viewing functions does not change the coordinate system but only the position of the plane onto which the models are mapped. The default view, shown in Figure 2-2, is an x-y plane with the z axis directed toward the user. Most of the examples presented in the following sections are shown in this view. For transparency, the axes in those examples are not displayed; in cases where the change of the view is required to better illustrate the model, the axes are displayed.

2.2. DRAWING TECHNIQUES BASED ON POINT AND LINE SETS

The application supports several routines for drawing multiple points, lines and whole structures. To achieve this it operates on sets of points and lines in order to produce new sets of points and lines. The input sets, called the *Current Point Set* (*CPS*), *Reference Point Set* (*RPS*), *Current Line Set* (*CLS*) and *Reference Line Set* (*RLS*), are either defined by the user or set by default.

The following part of this section will briefly describe the available drawing routines based on point and line sets. To give the user a better understanding how they work, an algorithm for some of the routines will be given.

Let's assume that the type of CPS and other point sets is:

```
type point3 = record
                x, y, z: real;
              end;
```

```
pointset = record
             pt_num: integer;
             points: array[1..pt_num] of point3;
           end;
var CPS, RPS, CLS, RLS: pointset;
```

The reason that the type of the line sets and the type of the point sets are the same is that in the application the lines are defined by their end points. For points $pt\_num = 1$ while for lines $pt\_num = 2$.

2.2.1. DRAWING A SET OF POINTS IN A GIVEN DIRECTION SEPARATED BY A SPECIFIED DISPLACEMENT

The action of the routine for drawing a set of points depends on the JOIN mode selected. If the JOIN mode is off, the function draws a set of points in a given direction separated by a specified displacement; if the JOIN mode is on—the function additionally joins the end points with lines. This function operates on the Current Point Set and by default produces the new Current Point Set. The Current Point Set, a structure which occurs in many 3D GAS applications, defines here the initial points the drawing will start from.

If n is the number of points to be drawn corresponding to each point of CPS, the following algorithm is used by this routine (for simplicity, it has been assumed that the direction can be only X, Y or Z).

```
procedure draw_point_set(CPS: pointset; displacement: real; n: integer;
                         join: boolean; direction: char)
var line, point: pointset;          /* auxiliary variables used for input in the */
    i, j: integer;                  /* POLYMARKER3 and POLYLINE3 functions */
begin
   line.pt_num := 2;
   point.pt_num := 1;
   for i := 1 to CPS.pt_num do
      for j := 1 to n do begin
         if join then
            line.points[ 1 ] := CPS.points[ i ];
         case direction of                    /* CPS.points is iteratively displaced */
            'X': CPS.points[ i ].x := CPS.points[ i ].x ± displacement;
            'Y': CPS.points[ i ].y := CPS.points[ i ].y ± displacement;
            'Z': CPS.points[ i ].z := CPS.points[ i ].z ± displacement;
         end;
         point.points[ 1 ] := CPS.points[ i ];
         POLYMARKER3(point);     /* PHIGS function used to iteratively draw */
         if join then begin                /* a set of points, see below, section 2.6.1 */
            line.points[ 2 ] := CPS.points[ i ];
            POLYLINE3(line);          /* PHIGS function used to iteratively draw */
         end;                              /* a set of lines, see below, section 2.6.1 */
      end;
end;
```

Figure 2-3 presents a model drawn by the function described in this section. In this example the Current Point Set, which consisted initially of the points 1, 2, 3, 4, 5 and 6, has been iteratively displaced, and after the routine has been executed, it contains the six end points of the lines on the opposite sides of the corresponding points.

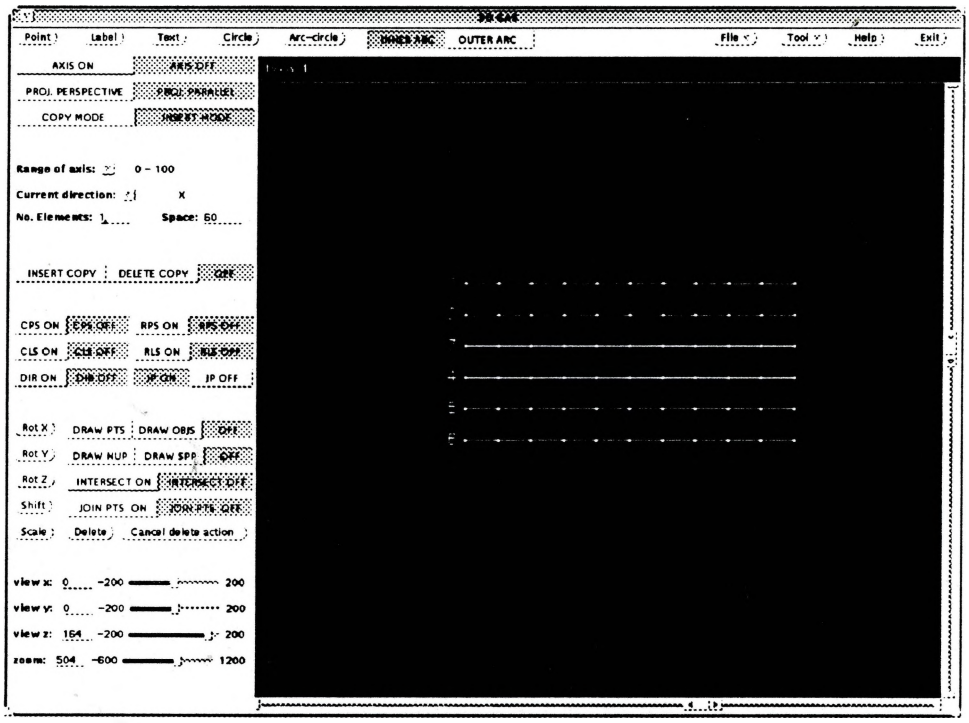

Figure 2-3. Drawing a set of joined points in a given direction separated by a given displacement.

2.2.2. JOINING TWO SETS OF POINTS WITH LINES

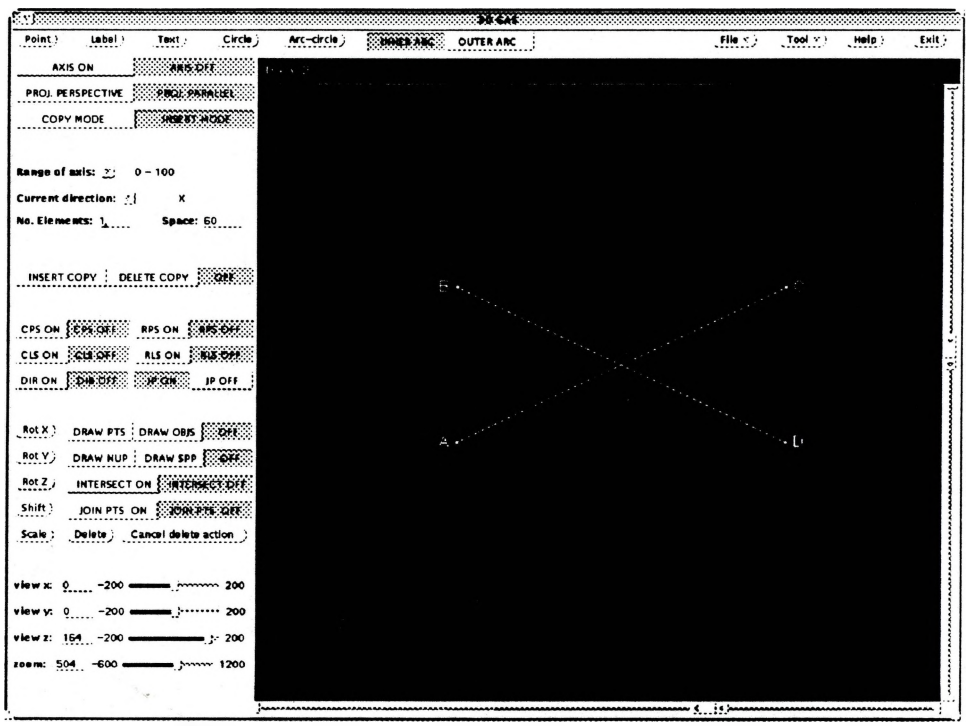

Figure 2-4. Joining two sets of points with lines.

The routine described in this section draws multiple lines which join points of the Current Point Set with corresponding points of the Reference Point Set. As shown in Figure 2-4, the CPS consist of points A and B and the RPS consist of points C and D. If the number of points belonging to the Current Point Set does not equal the number of points belonging to the Reference Point Set, the application ignores the excess points.

```
procedure join(CPS, RPS: pointset)
var line: pointset;
```

```
    i: integer;
begin
   line.pt_num := 2;
   for i := 1 to min(CPS.pt_num, RPS.pt_num) do begin
      line.points[ 1 ] := CPS.points[ i ];
      line.points[ 2 ] := RPS.points[ i ];
      POLYLINE3(line);
   end;
end;
```

2.2.3. DRAWING A SET OF POINTS AT THE INTERSECTION OF TWO LINE SETS

As shown in Figure 2-5 this routine draws a set of points at the intersection of the CLS and the RLS. The CLS consists of lines defined by points A-C and E-G, whereas the RLS consists of lines defined by points B-D and F-H.

The algorithm used by this function is based on two line equations:

$$x = x_1 + v(x_2 - x_1); \qquad x = x_3 + w(x_4 - x_3);$$

$$y = y_1 + v(y_2 - y_1); \qquad y = y_3 + w(y_4 - y_3);$$

$$z = z_1 + v(z_2 - z_1); \qquad z = z_3 + w(z_4 - z_3);$$

where x_1, y_1, z_1, etc. are the coordinates of the given end points of the lines, "v" and "w" are the parameters, and $x_2 - x_1$, $y_2 - y_1$, $z_2 - z_1$, etc. are the slopes of the lines.

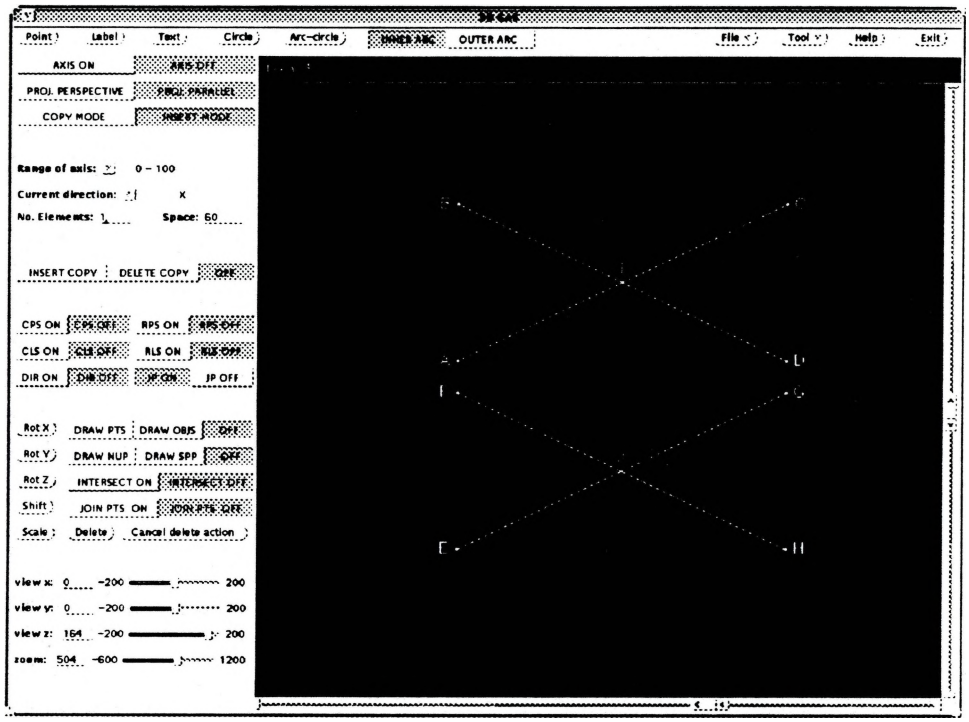

Figure 2-5. Drawing a set of points at the intersection of two line sets.

The intersection of two lines is a point whose coordinates satisfy both equations. The algorithm therefore has the form:

```
procedure intersect(CLS, RLS: pointset)
var point: pointset;
    i: integer;
begin
   point.pt_num := 1;
   for i := 1 to min(CLS.pt_num, RLS.pt_num) do begin
      calculate point.points by solving two line equations;
```

```
      POLYMARKER3(point);
   end;
end;
```

2.2.4. DRAWING A SET OF STRUCTURES IN A GIVEN DIRECTION SEPARATED BY A SPECIFIED DISPLACEMENT

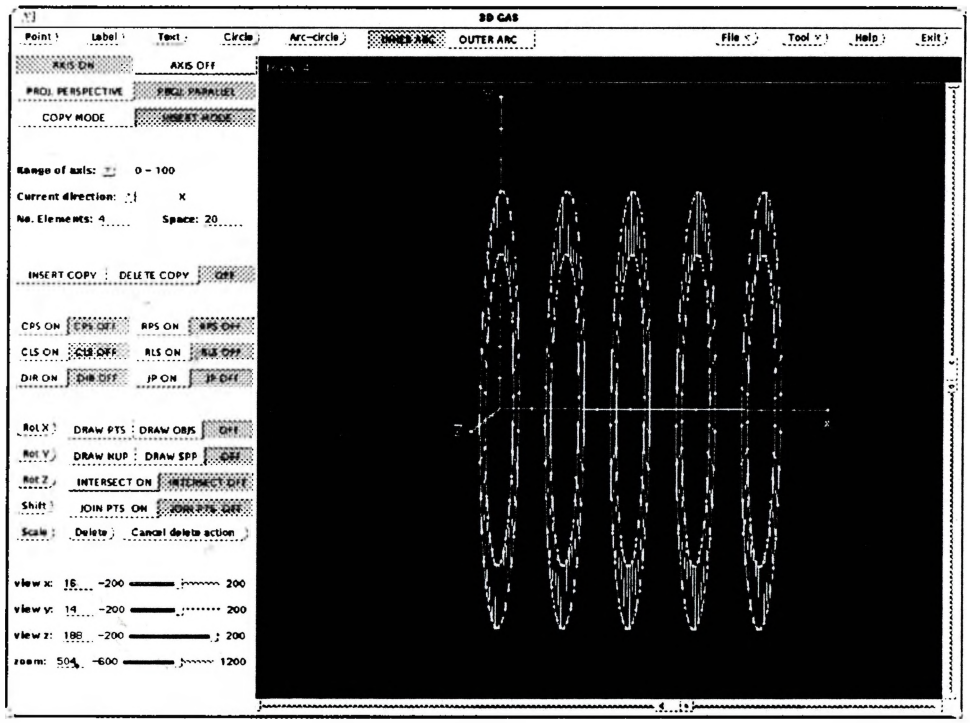

Figure 2-6. Drawing a set of structures.

This routine draws multiple copies of a created structure. Figure 2-6 gives an example of a model consisting of a number of identical components which has been

drawn using this function. The view of the component in the y-z plane is shown in Figure 2-7.

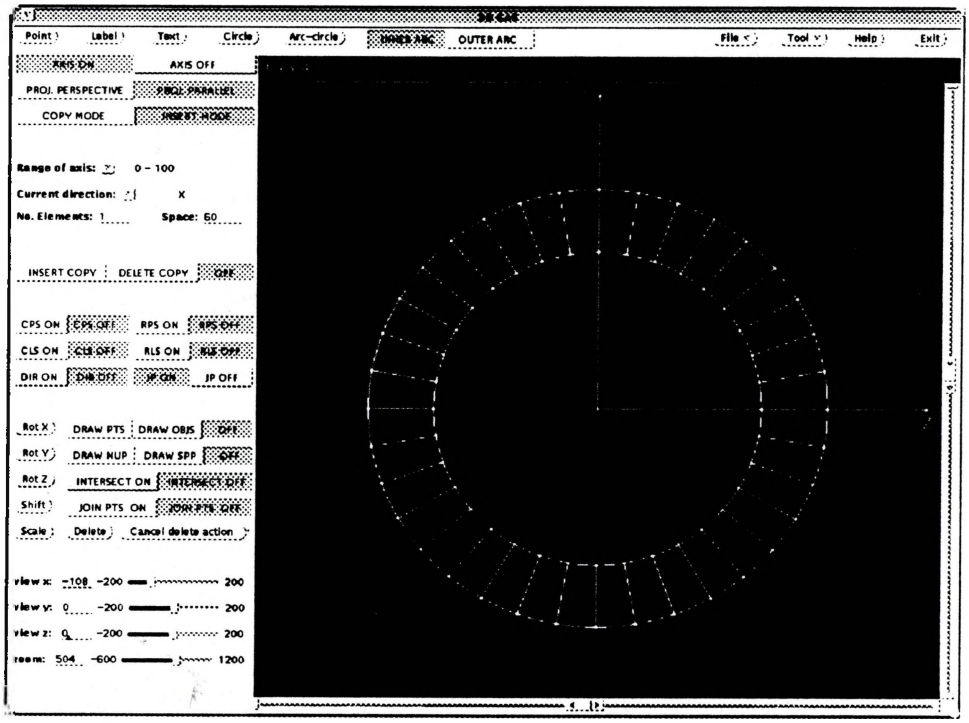

Figure 2-7. A view of the component used in the routine to create a set of structures shown in Figure 2-6.

The algorithm used by this routine traverses the edited structure using the structure pointer to mark the current position during traversal. At this point one remark should be made. Until now the word *structure* has been used to refer to any drawing created on the screen. In the application it might have, however, different meaning. The structure is an entity consisting of a sequence of elements such as output primitives, view selection, primitive attributes, labels, etc. (see sections 2.6.1

and 4.2.1). Additionally, the structure can consist of other structures which are invoked during traversal. Because drawing in 3D GAS creates only one structure at a time, the term *structure* meaning the drawing and the term *structure* meaning the internal representation of the model are used interchangably.

The structure pointer is a pointer to an element of the structure; by default it points to the last element. The value of the structure pointer can be retrieved—this property has been used in the algorithm to determine the end of the structure—or arbitrarily changed so that it points to the required element. The data of the element of the structure pointed to by the structure pointer can be retrieved. In the algorithm, the data relative to such structure elements as points, lines, circles, arcs and labels is iteratively retrieved and transformed to reflect the displacement. Then based on these data a new element of the same type is created and appended to the end of the structure.

If copy_num is the number of copies to be drawn in a given direction, the algorithm used in the application will look like this:

```
procedure make_copies(structure_pointer, copy_num: integer)
var cur_pointer, i: integer;
begin
  for cur_pointer := 1 to structure_pointer do begin
    retrieve data and type of the element pointed to by the cur_pointer;
    for i := 1 to copy_num do begin
      calculate data of a new element (add displacement);
      create new element;
```

```
        append new element;
      end;
    end;
end;
```

2.2.5. DRAWING A NUMBER OF POINTS SEPARATED BY A GIVEN DISPLACEMENT BETWEEN TWO DEFINED POINTS

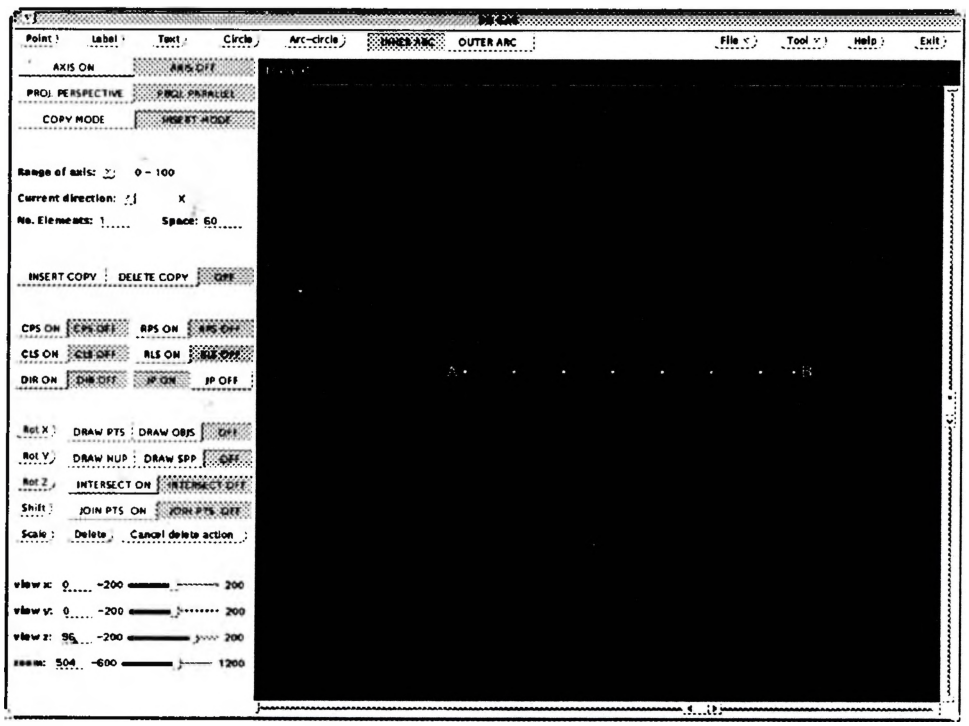

Figure 2-8. Drawing a number of points separated by a given displacement between two defined points (JOIN mode is OFF).

This very useful function frees the designer from tedious calculations. It draws the maximum number of points which are separated by a given displacement between two given points in the CPS. In the example shown in Figure 2-8, the CPS consists of points A and B.

The maximum number of points (max_pt) can be calculated by dividing the distance between two given points by the displacement. If *dist* defines a distance between points A and B and *displacement* defines the value of the displacement entered by the user, max_pt is equal:

$$\text{max\_pt} = \lceil (dist - displacement)/displacement \rceil .$$

The coordinates of the set of points between these two points are calculated iteratively by adding the value of the displacement to the first point in the CPS. The algorithm is presented below:

```
procedure max_points(CPS: pointset; displacement: real)
var point: pointset;
    max_pt: int;
    point3: array [1..max_pt] of point3;     /* used for calculations of coordinates */
begin
   calculate dist;
   calculate max_pt;
   point.pt_num := 1;
   for i := 1 to max_pt do begin
      point3[ i ].x := CPS.points[ 1 ].x + i * displacement.x;
```

```
        point3[ i ].y := CPS.points[ 1 ].y + i * displacement.y;
        point3[ i ].z := CPS.points[ 1 ].z + i * displacement.z;
        point.points := point3[ i ];
        POLYMARKER3(point);
    end;
end;
```

2.2.6. DRAWING A SPECIFIED NUMBER OF POINTS BETWEEN TWO DEFINED POINTS

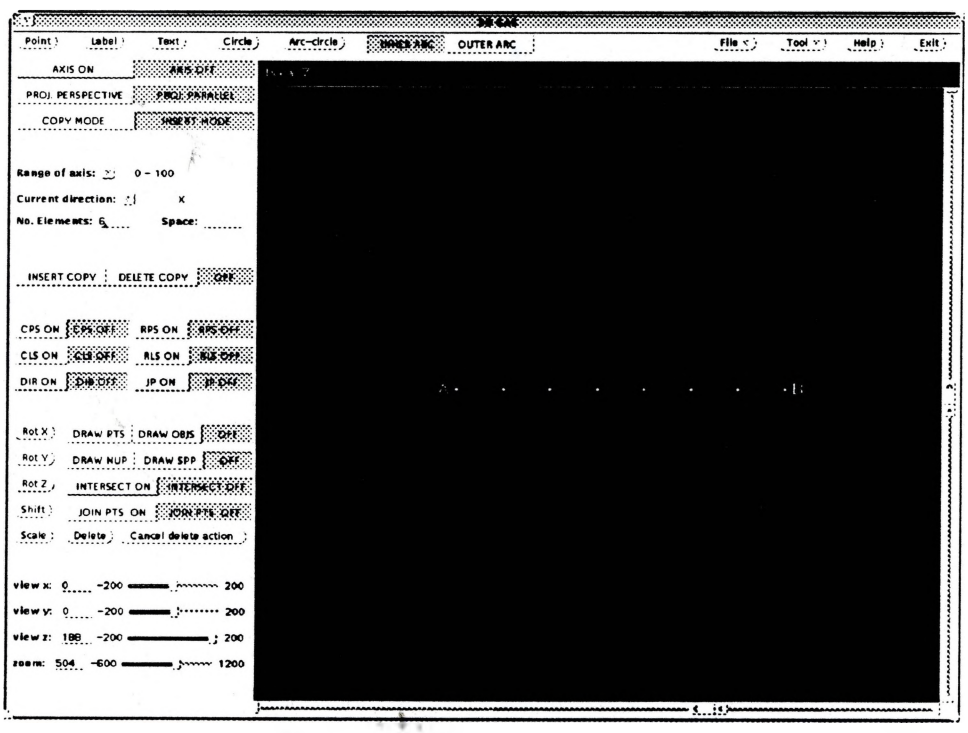

Figure 2-9. Drawing a specified number of points between two defined points.

The following function resembles the previous one with the exception that it calculates the displacement and then draws a given number of points separated by the displacement between the two points defined by the Current Point Set. In the example shown in Figure 2-9, the CPS consists of points A and B and the number of points to be drawn between them is set to 6.

If n defines the number of points to be drawn between two given points, the routine uses the folowing algorithm:

```
procedure n_points(CPS: pointset; n: int)
var point: pointset;
    displacement: real;
    point3: array [1..max_pt] of point3;   /* used for calculations of coordinates */
begin
   calculate dist;
   displacement := dist / (n + 1);
   point.pt_num := 1;
   for i := 1 to n do begin
      point3[ i ].x := CPS.points[ 1 ].x + i * displacement.x;
      point3[ i ].y := CPS.points[ 1 ].y + i * displacement.y;
      point3[ i ].z := CPS.points[ 1 ].z + i * displacement.z;
      point.points := point3[ i ];
      POLYMARKER3(point);
   end;
end;
```

2.3. DRAWING MODES

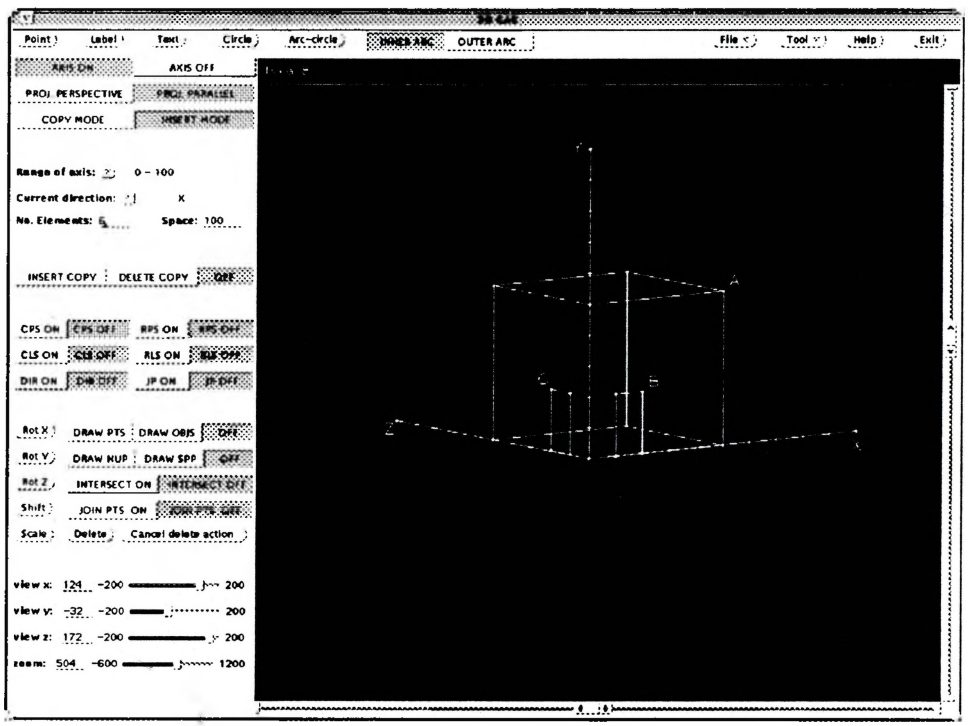

Figure 2-10. Utilizing the *COPY MODE*.

There are two modes, *COPY MODE* and *INSERT MODE*, in which the drawing techniques described in the previous section can be applied.

INSERT MODE, the default, is the main mode the drawing should be drawn in. It creates a main structure which appears as a white wire-frame model on the blue background of the workstation. During the session, at each step of the drawing process, a new element is added/deleted to/from this structure. Only a structure created in this mode can be saved into or loaded from a file by the applicaton.

COPY MODE is an auxiliary mode. It creates an auxiliary structure which

can be inserted into the main structure. The purpose of this mode is twofold: to allow the designer to create a draft of the structure and to create a component of the structure which can be inserted multiple times into the main structure in the appropriate positions. Visually the drawing performed in this mode can be easily distinguished from the main structure because it is displayed in a different color. In the application magenta has been used for structures designed in this mode.

3D GAS provides the choice item *INSERT COPY/DELETE COPY* which allow for the insertion and deletion of the structure created in *COPY MODE*. The example shown in Figure 2-10 demonstrates the use of *COPY MODE*. A component (a small doorway) has been created in this mode, inserted into the main structure A in position B, rotated around the y axis, and finally inserted again in position C.

2.4. TRANSFORMATION AND VIEWING FUNCTIONS

Transformation functions available in 3D GAS perform rotations about x, y, z axes, x, y, z translations and x, y, z scaling. The effect of their use depends, however, on the drawing mode selected. If in *COPY MODE*, these functions replace the values of the coordinates of the elements of the structure in the world coordinate system by the new values calculated in the application. In effect, the transformed structure replaces the structure before transformation. The same functions used in *INSERT MODE* do not replace the structure but append the transformed structure to the existing one. Additionally, if JOIN mode is ON the corresponding points of the structure before the transformation and the points of the transformed structure will

be joined with straight lines. The transformation functions can therefore be thought of as another tool for drawing symmetrical structures. The sphere shown in the Figure 2-11 was drawn using this technique. The drawing took place in the *INSERT MODE* and began with the creation of a circle (see Figure 2-12) which was then rotated around the y axis. The transformation function used in this mode appended the rotated circle(s) to the already existing one(s). This action was repeated until the sphere was closed.

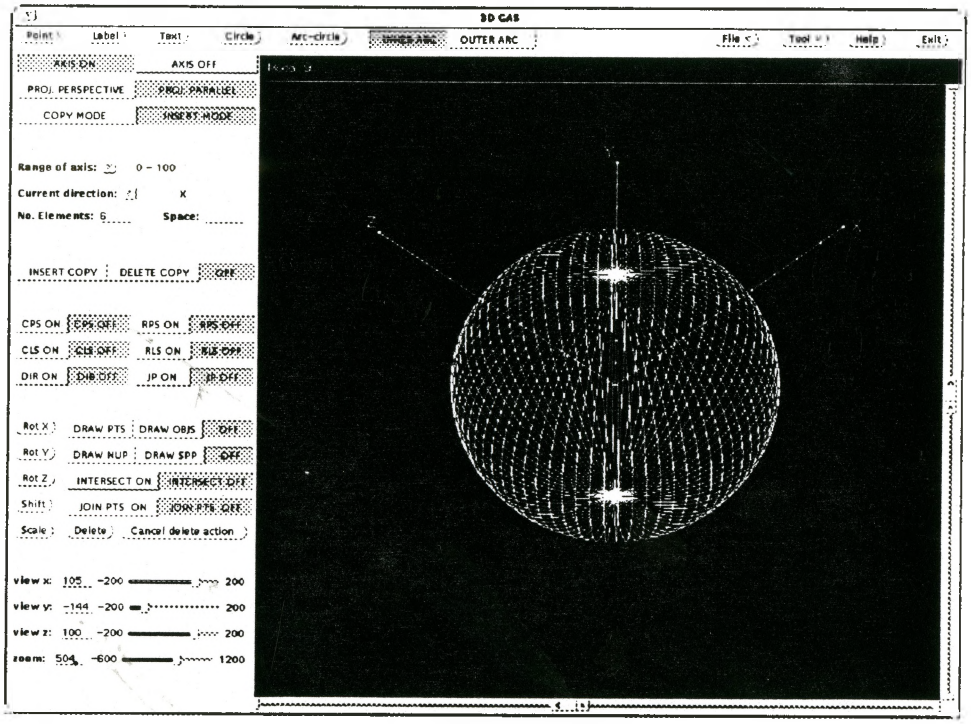

Figure 2-11. A sphere created by rotation of the circle around the y axis.

Using translation functions in *INSERT MODE* with JOIN mode set to ON resembles the technique called *extrusion* used in other CAD systems (see chapter 5). In this technique, the view of the three dimensional object is first created in a plane,

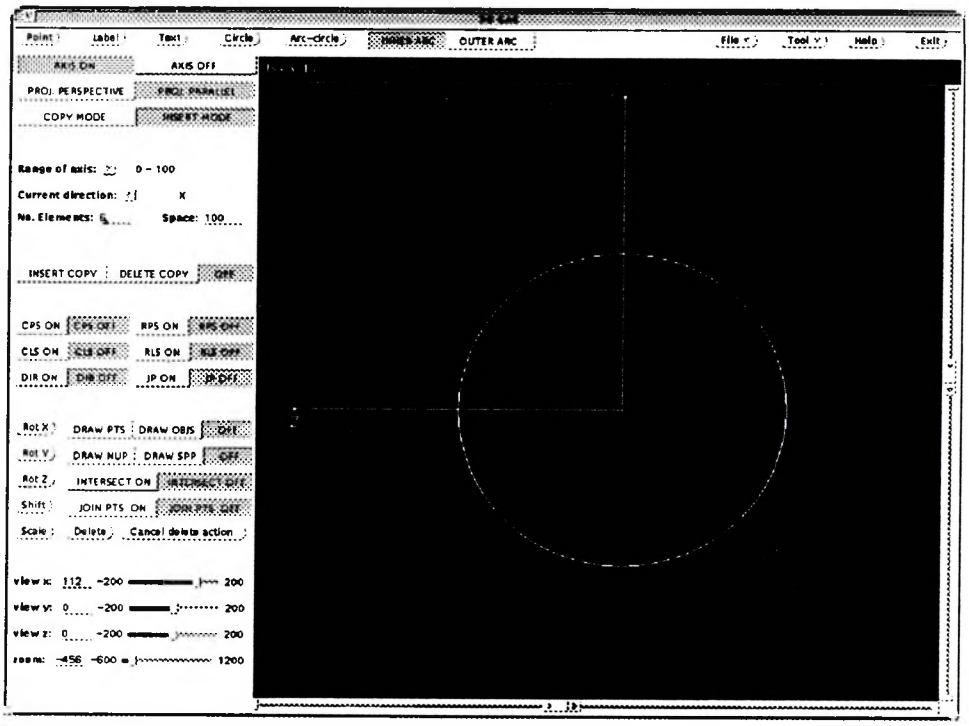

Figure 2-12. A circle used for drawing the sphere from Figure 2-11.

then the copy of this view is made in a plane parallel to the first one (the view is translated) and the corresponding points are joined with lines. This approach is illustrated in Figure 2-13. The design of the cube presented there involved the following steps:

- the point (0, 0, 0) was created (by default it became the CPS);
- JOIN mode was set to ON;
- the point (60, 0, 0) was created using the routine described in section 2.2.1;
- the line defined by the two points was translated in z direction by 60 units;
- the square was translated in y direction by 60 units.

The technique for drawing a sphere as shown in Figure 2-11 is not acceptable if in *COPY MODE*, where the transformation functions are supposed to perform simple transformations to manipulate a component. Therefore, the same sequence of actions as described above in a case of the sphere but performed in the *COPY MODE* would cause the circle to rotate around the y axis.

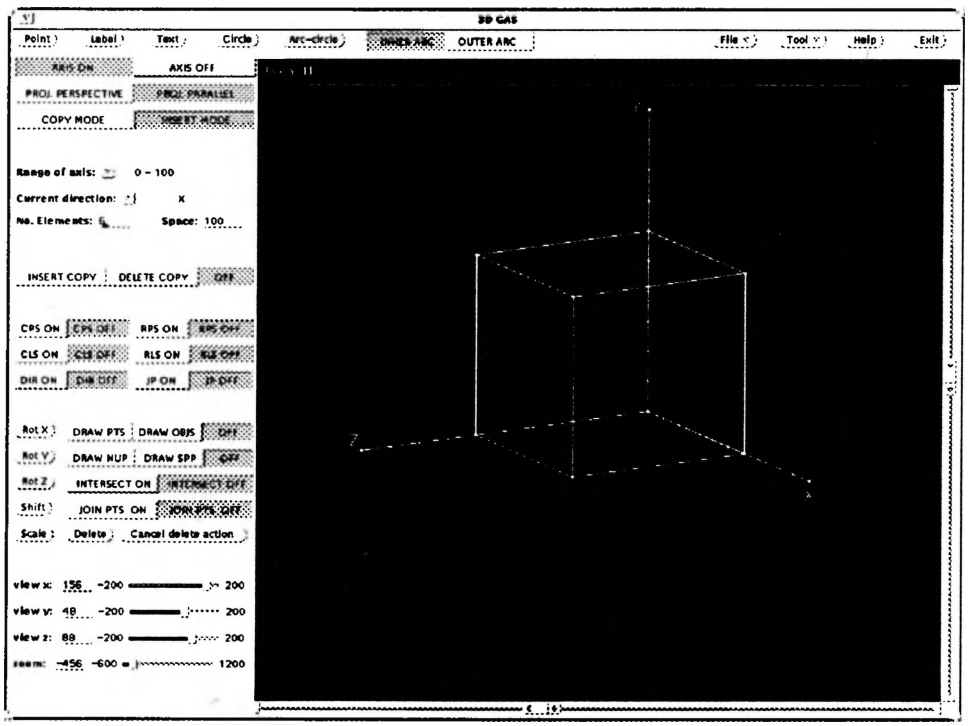

Figure 2-13. A cube created by translation of entities.

The purpose of the viewing functions is to provide a facility for viewing edited models. The visual effect of rotation and zooming using these functions is achieved by changing the viewing parameters, responsible for mapping the model onto the screen, rather than by changing the location of the model within the coordinate system. The viewing functions allow looking at the model from different points arbitrarily selected

in 3D space. They are especially useful at certain stages of the design process. See chapter 4.2.3 for the implementation details.

2.5. ARCHIVING AND LOADING

The edited structure can be archived into a file together with the project description and then loaded again. This feature is very useful when dealing with the large and complicated models which require a large amount of time to be spent on design and drawing.

2.6. ORGANIZATION OF GRAPHIC STRUCTURES

With the exception of the transformation functions, all graphics in 3D GAS are implemented using the PHIGS package; the organization of the graphic structures therefore follows the PHIGS specification described more precisely in chapter 4.2.1.

2.6.1. STRUCTURE ELEMENTS

Graphic objects are represented as a sequence of elements such as output primitives, view selection, primitive attributes, labels, etc. The elements are grouped into entities called *structures*, mentioned before in section 2.2.4, which can be stored and retrieved from the *Central Structure Store* (*CSS*).

Output primitives are the basic graphic elements used to construct an object.

The output primitives used in the application include:

- polymarker
- polyline
- text
- annotation text relative
- generalized drawing primitive (GDP).

A *polymarker* is an element of a set of simple predefined shapes such as a point, a circle, a cross, a square etc. Because the available type "point" is just a small pixel, in 3D GAS, the type "square", used in the appropriate scale, is used to draw points. The polymarker primitive is created by the function POLYMARKER3 (see section 2.2.1 for an example).

A *polyline* defines points for drawing a set of connected straight lines. It is created by the function POLYLINE3 (see section 2.2.1 for an example).

Text is an output primitive containing character string data which serves as the project description. The elements of this type are created by the TEXT function.

Annotation text relative is an output primitive containing character string data. In contradistinction to the text primitive, annotation text relative elements are always mapped parallel to the display screen. They are used in 3D GAS to implement labels.

The *generalized drawing primitive* structure element contains data to generate figures such as circles, arcs and ellipses. It is created by the GENERALIZED DRAWING PRIMITIVE function.

The appearance of the above output primitives on the screen depends on their attributes such as width, color, size, etc. The attributes are set by the special functions—in 3D GAS they are initialized once when the program starts. Each of the output primitives is preceded in the structure by its pick id. The pick id serves as an identification of the output primitive so that it can be distinguished by the application when picked with the mouse. In this report, *clicking* will refer to pressing the mouse button in any context; *picking* will mean the selection of an output primitive in the drawing window.

2.6.2. STRUCTURES OF 3D GAS

In 3D GAS, there are four structures called *AXIS*, *OBJECT*, *COPY OBJECT* and *TEXT*. The structure *AXIS* contains a set of three axes and is fixed (i.e. cannot be interactively modified), whereas the modification of the three others is possible.

OBJECT is any structure created in the *INSERT MODE*. This is the main structure created by the designer which can be saved and loaded.

COPY OBJECT is any structure created in the *COPY MODE*. This is an auxiliary structure which can be inserted into the *OBJECT* structure.

The structure *TEXT* is edited into the text window and contains text which can be a description of the project.

CHAPTER THREE

USER INTERFACE

3D GAS uses the XView package to build the user interface. The application therefore reflects XView features which in turn implement the OPEN LOOK Graphical User Interface (GUI). OPEN LOOK, developed by AT&T, provides the user with user interface reusable components. The shape and appearance of these components is restrained in XView.

The application screen, as shown in Figure 2-1, consists of two workstation windows and user interface controls. The selection of the controls as well as picking in the drawing window is done by mouse clicking. The application does not set the functions of the mouse buttons to operate the controls. This has been done by the XView package. According to this setting, pressing control buttons occurs by clicking with the left mouse button and pressing the menu buttons [1] is performed by the right mouse button (clicking with the left mouse button on the menu button selects the first option from that menu). The middle mouse button does not take part in selecting controls. In the drawing window, however, the mouse buttons are set by the application to be equivalent. Thus picking in this window can be accomplished by the left, middle or right button.

In this chapter, the convention has been adopted that the names of the application controls are written in upper case or lower case *italics* depending on how

[1] Control buttons are buttons which trigger an action; menu buttons are buttons which invoke menus.

they appear in the application screen.

3.1. WORKSTATION WINDOWS

In 3D GAS, two windows, separated by a horizontal white line, are used to display graphics. The window at the top (see Figure 3-1), called the *text window*, serves as a text editor where the description of the graphical structure such as the project title, author, data can be entered and subsequently saved with the drawing.

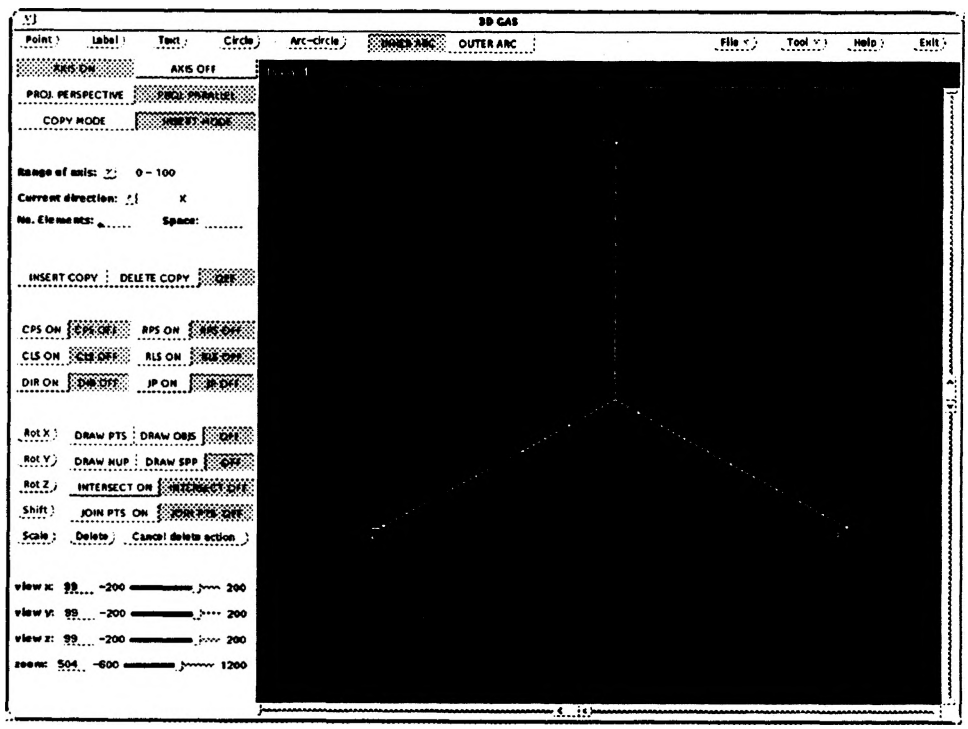

Figure 3-1. Centering of the coordinate system.

The big window below the text window, called the *drawing window* or the *paint window*, is a main window of the application where the drawing can be made

and input received. The application determines what input it is interested in by setting an input mask. In 3D GAS, the only input that can be received in the drawing window is picking. Each output primitive edited in this window can be picked with the mouse and thus made available to subsequent functions such as *delete* or included in the CPS, CLS, etc. Flashing of the picked element is a confirmation of the action of picking. Picking points causes additionally an audible signal.

In 3D GAS, only about one-fourth of the paint window is visible at a time. The viewport has attached horizontal and vertical scrollbars to view the other parts of the drawing if it does not fit entirely in the window limited by the viewport. When the program starts, the upper–left quadrant of the paint window is shown in the viewport. Therefore, before starting the drawing it is necessary to center the coordinate system. This can be done by clicking on the *AXIS ON* choice item to make the set of three axes visible in the window and setting both scrollbars in the middle of their range. The effect of this action is shown in Figure 3-1. The default viewing parameters were changed to show all three axis.

3.2. USER INTERFACE CONTROLS

The user interface controls are operated by the mouse. When the desired control is clicked, the callback routine related to it is invoked and an appropriate action is performed. The clicking on the button *File*, for example, causes a pulldown menu with two options: *Load* and *Save* to appear at the upper left of the screen. The clicking on one of these, in turn, invokes a command frame which prompts a user to enter an appropriate filename to save or load the graphic structure depending on the selection made (see Figure 3-2).

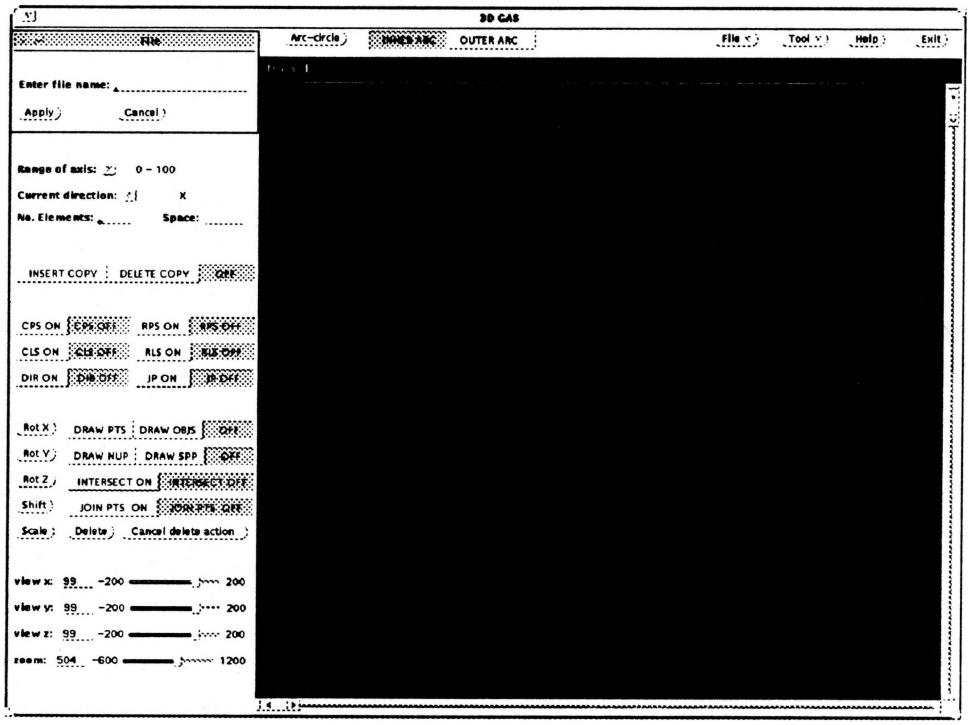

Figure 3-2. Command frame for saving or loading the drawing.

The user interface controls used in 3D GAS can be divided into 5 types:

- button items;
- text fields;
- choice items and abbreviated choice items;
- sliders;
- scrollbars.

Button items are used to trigger an appropriate action when selected. They can either start a chain of menus like the *File* button described above, or invoke a command represented by their labels such as the *Delete* and *Cancel delete action*

buttons.

Text fields allow the entry of a character string. The string can be longer than the specified text field. If this happens, an arrow pointing to the left appears on the left of the text field, indicating that some text is no longer visible. Similarly, the arrow on the right indicates that some right portion of the string is not visible. A caret is used to mark the current text field. If a user enters the value of a string by pressing the Control key, the caret will be advanced to the next text field. Two examples of text fields are the *No. Elements* and *Space* fields in Figure 3-2.

Choice items provides a list of different choices which can be selected by clicking with the mouse. The idea of the abbreviated choice items is similar with the exception that only the selected value of the choice item is displayed. *AXIS ON/AXIS OFF* in Figure 3-1 is an example of a choice item; *Range of axis* in the same figure represents an abbreviated choice item.

Sliders allow the selection of a value within a specified range. Each slider has four components: the label, the current value, the slider bar, and the minimum and maximum value (the range). The value of the slider can be entered either by changing the position of the slider bar or by entering the current value which is displayed in the numeric text field. In the application sliders are used to implement viewing functions. Figure 3-2 demonstrates four sliders: *view_x*, *view_y*, *view_z* and *zoom*.

Scrollbars are used to view the contents of the windows they are attached to. The unit of the scrollbar is the number of pixels it scrolls with one click of the mouse. In 3D GAS this unit is set to ten pixels.

CHAPTER FOUR
IMPLEMENTATION AND SOFTWARE

Although 3D GAS has been implemented in the X Window System, only a small part of it refers directly to the X basic library (Xlib). The main part of this project is based on two other packages, i.e. XView and PHIGS. This chapter discusses, to a limited degree, these two packages.

4.1. XVIEW

XView [1] is one of the few toolkits created for X and one of the two (the Athena widget set was the alternative) available in the Department of Computer Science and Systems at McMaster University where the project has been developed. The reason why XView has been chosen for this project is simple: XView is a reliable graphical user interface for commercial applications. The Athena widget set was developed by MIT as a test and demonstration of Intrinsics, the library of predefined widgets, rather than as an interface package for commercial use.

XView is a descendant of the SunView Toolkit created for the Sun's windowing system called SunWindows. The main difference between SunWindows and X is that SunWindows is not a networked system, i.e. a single computer had to handle applications, display and keyboard. When MIT released X and its popularity grew, Sun accepted it and created XView to make a great number of already existing SunView applications portable to an X environment.

An important feature of XView is that it implements the standard OPEN LOOK GUI mentioned at the beginning of chapter 3. OPEN LOOK, not bound to any operating system, windowing system or graphic display, makes XView easily portable to X.

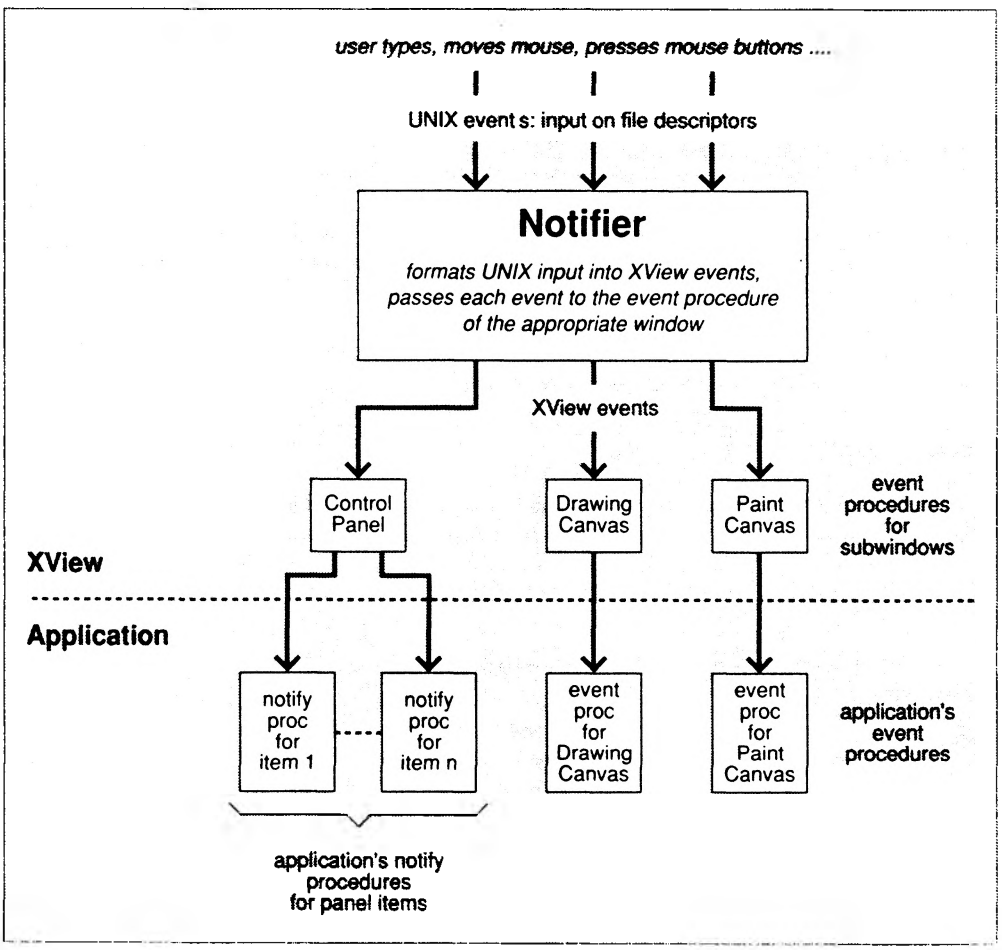

Figure 4-1. Xview event handling mechanism [1].

As an X based system, XView is a package for *event driven* applications and relies on communication with the X server which is accomplished via a network protocol. This type of implementation allows programs to be run on one machine

and displayed on another. An XView application does not receive events directly; they are sent by the X server to the special XView feature called the Notifier which dispatches them to the appropriate object. The event handling mechanism is shown in Figure 4-1.

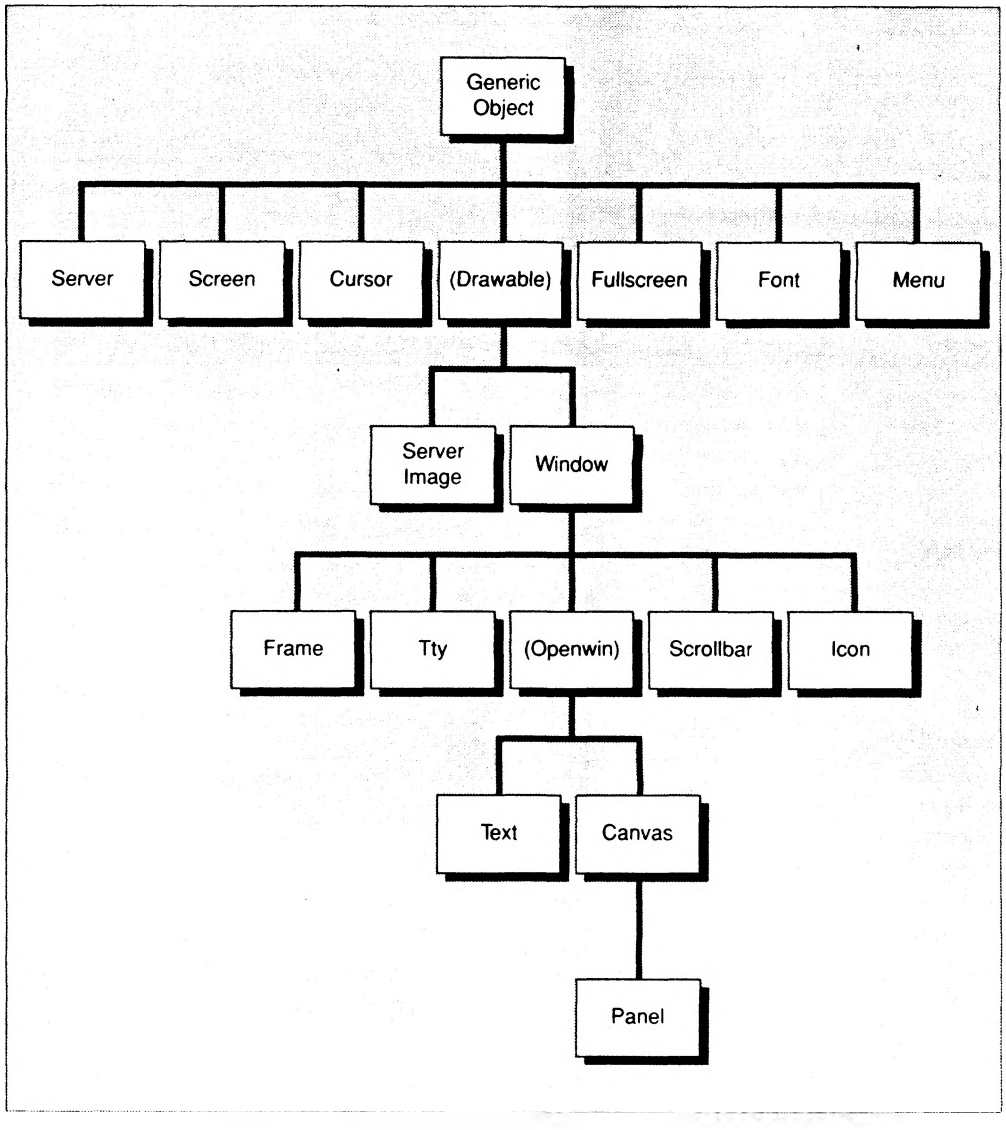

Figure 4-2. XView class hierarchy [1].

Each XView object (e.g. a panel button or a canvas) registers *notify procedures* or *callback procedures* with the Notifier, and so becomes its client. The client can determine what events it is interested in by setting the *input mask*. The Notifier detects events in which its clients have expressed an interest and dispatches them to the proper clients. After the client has consumed the event, control is returned to the Notifier.

To the programmer, XView is an *object-oriented* toolkit. As shown in Figure 4-2, the organization of the objects is hierarchical with the Generic Object as a root. The objects are grouped into classes, often called packages, defining a set of related functional elements. As an object-oriented model, XView uses subclassing and chained inheritance, i.e. all objects of a particular class inherit properties from its parent class. The Generic Object, for example, contains some basic properties which all classes share. The default setting of some properties such as window dimension, location on the screen, border thickness, etc., can be changed in the application. This setting, however, might not necessarily be honored by the window manager.

In XView, there is a set of six basic functions that allow the programmer to create, destroy or manipulate any object. When an object is created, the XView function returns a *handle* for that object as a kind of identification. It is the handle which is passed to the appropriate function if the programmer wishes to manipulate the data of that object. The reliance on handles in communication with objects enables *data-encapsulation* and is a way of *information-hiding*, typical of object-oriented programming.

The objects used in the application are:

- Generic Object;

- Cursors;
- Fonts;
- Menus;
- Windows;
- Frames;
- Openwins;
- Scrollbars;
- Text Windows;
- Canvases;
- Panels.

The reminder of this section will describe on the conceptual level some of the objects that XView offers. The objects will be presented in the bottom up order shown in Figure 4-2. For this presentation those objects have been chosen whose importance for the understanding of 3D GAS is the greatest.

4.1.1. PANELS

Panels provide the application with the control area where panel items such as menus, buttons, sliders, etc., reside. The main function of panels is to manage a variety of panel items and to distribute events to those items. Panels set up the event mask for their items; the application, however, may specify its own event mask and register callback routines. It is the responsibility of the PANEL package to handle all events occuring in the panel's window and dispatch events to the proper item.

4.1.2. CANVASES

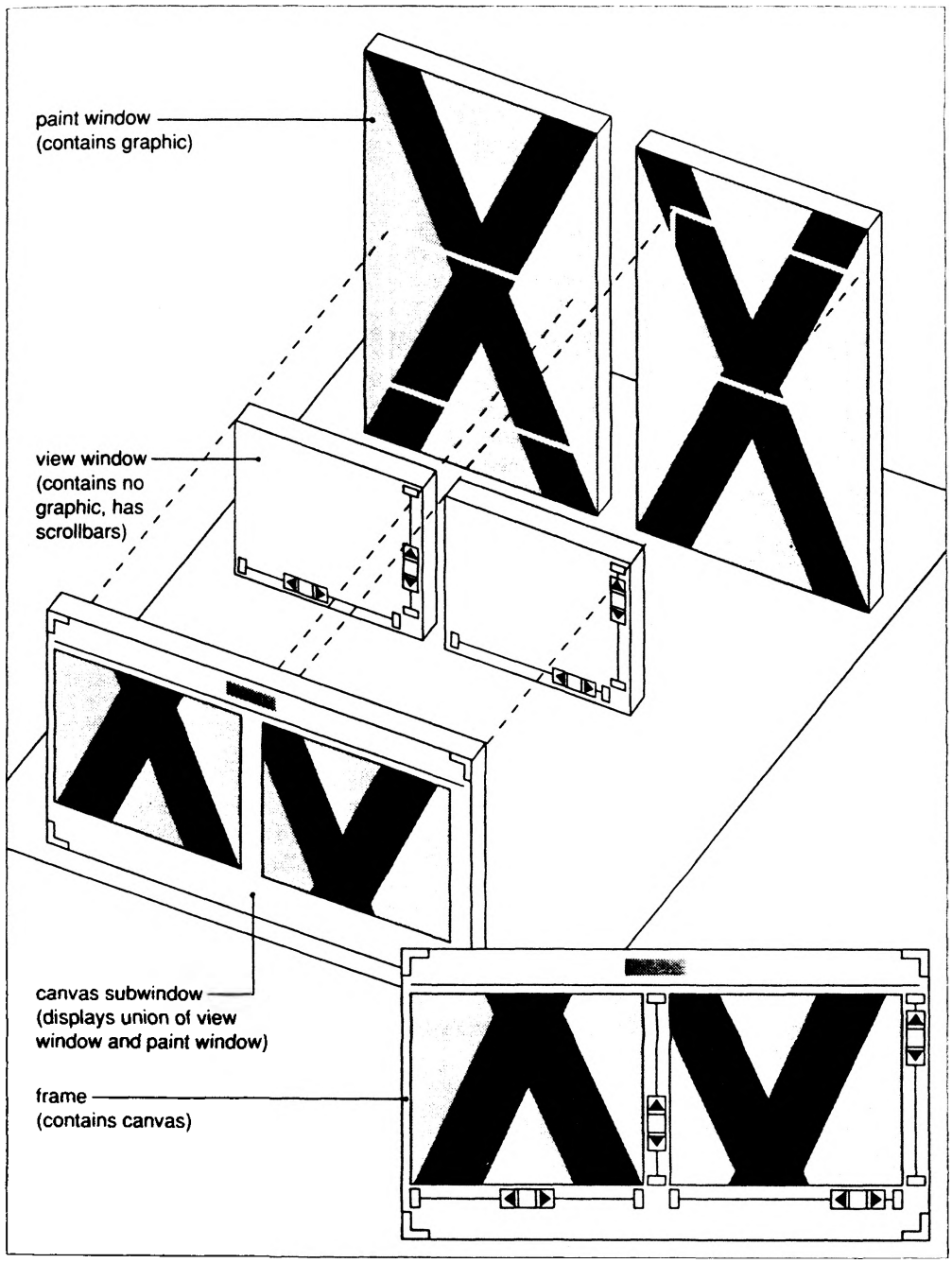

Figure 4-3. Elements of the canvas object [1].

As shown in Figure 4-3, a canvas object consists of three elements: a *paint window*, a *view window* and a *canvas subwindow*.

A canvas subwindow also inherits properties from the FRAME package (see below, section 4.1.4) and may be a parent for one or more view windows. A view window, also called a viewport, through attached vertical or horizontal scrollbars provides mechanisms for viewing the paint window in a particular view.

A paint window is a window which contains graphics and handles input. This window is exclusively associated with the view window which determines what part of it is visible to the user. If the view is split by the scrollbar, an identical paint window is created, i.e. each view handles its own paint window and is independent of others.

Canvases specify events the application should be interested in. They also register an event handler which is invoked whenever one of the defined events takes place. The event handler looks at the type of event it has received and determines the appropriate action. Additionally, it is a responsibility of the application to repaint its canvas at any time. It is done by the *repaint procedure* which is called whenever:

- the canvas has been resized;
- the window has been obscured by other windows;
- the user scrolls the paint window.

4.1.3. TEXT WINDOWS

Text windows are the XView objects for displaying and editing a sequence of ASCII characters. The contents of a text window are stored in a file or in memory.

The TEXT package provides basic editing features such as inserting text into a file, searching and matching a pattern, etc. The text window contains a vertical scrollbar to view the contents of the text.

4.1.4. FRAMES

Visually frames are a kind of a container for other windows. In addition, frames play a bridging role between an application and the window manager. The FRAME package does not manage the appearance of the objects it contains. Headers (title bars), resize corners and colors of those objects are the property of the window manager. The application may give hints to the window manager about some of these attribute settings; they might, however, not be honored and depend on which window manager the application is run.

There are two kinds of frames:

- Base Frames
- Pop-up Frames

Although the base frame is often called the application frame, more than one base frame may be used in one application. Additionally, a base frame may be a parent of another frame called a subframe.

Pop-up frames are the subframes owned by the base frame. Typically, they are used to perform one or more functions and after their role is completed, they go away. The *Enter filename* command frame shown in Figure 3-2 is an example of a pop-up frame.

4.1.5. SCROLLBARS

Scrollbars enable subwindows to be viewed. In a canvas subwindow, scrollbars can be used to see different portions of the paint window which can be larger than the canvas subwindow. There are two kinds of scrollbars: horizontal, placed to the right of the window, and vertical, placed at the bottom.

4.1.6. WINDOWS

The objects of this class are never explicitly created. Rather they are created as parents of most visual objects such as frames, scrollbars or icons, but not panel items. Visual objects are so called because they contain X windows in order to display themselves or handle events. Conversely, the nonvisual objects such as fonts do not contain and are not a subclass of the WINDOWS class.

4.1.7. MENUS

Menus are objects which are not a subclass of the WINDOWS class. This means they do not contain any window; they rather are bound to a window when the menu is selected by the user, avoiding the creation of multiple X windows for each menu.

In XView there are three types of menus:

- pop-up menus which are displayed when the menu button is pressed;
- pulldown menus which are displayed below a menu button when it is pressed;

• pullright menus which are displayed as a menu to the right of a parent menu.

An example of the pulldown menu and the pullright menu is shown in Figure 4-4. In 3D GAS, two pulldown menus are used. One is attached to the *File* button, the other to the *Tool* button. When the user presses the *Tool* button, the choices *Refresh*, *Delete Drawing*, *Undisplay Drawing* and *Display Drawing* appear below the button. The pullright menus are not used in the application.

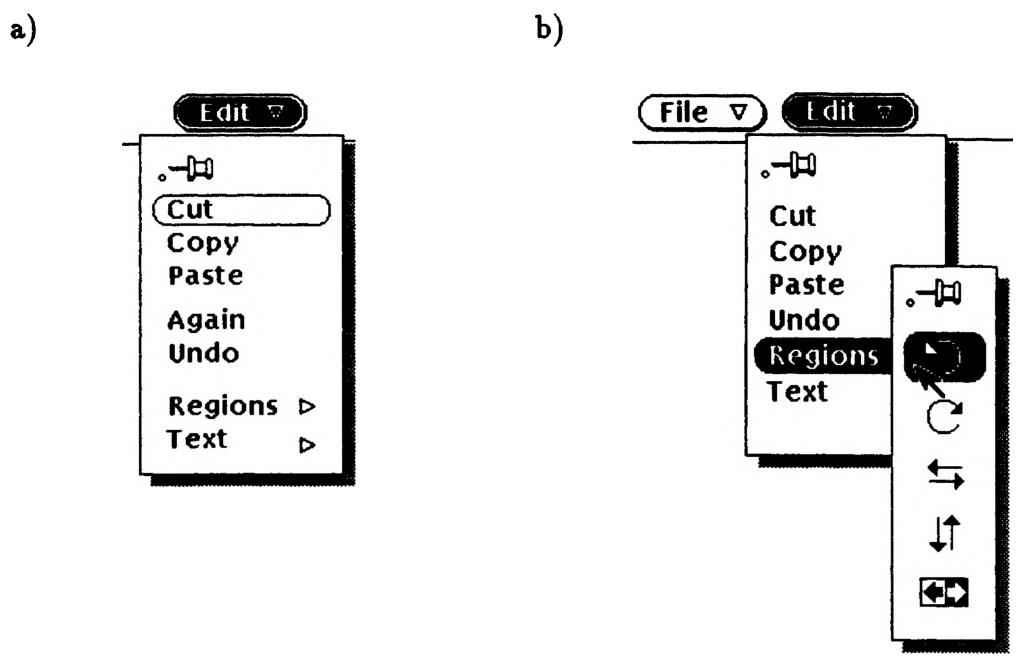

Figure 4-4. An example of the a) pulldown menu, b) pullright menu [1].

4.1.8. GENERIC OBJECT

The Generic Object is a root in the class hierarchy. Because in XView the object creation is top-down, it is created automatically if any of its subclasses is

created. Generic Object has no function and therefore one never creates an instance of that class.

4.2. PHIGS

PHIGS is an acronym for the *Programmer's Hierarchical Graphics System*. Developed by the working group of the American National Standards Institute (ANSI), PHIGS was accepted by ISO in 1985 as an international standard for 2D and 3D graphics.

PHIGS separates the act of creation of the data structure from the data display. The application can therefore control when and how the structure stored in the database, called the *Central Structure Store* (*CSS*), is to be viewed on the workstation. The viewing pipeline as well as the main concepts of PHIGS will be described in this section. For more details refer to [2] and [3].

4.2.1. GRAPHICAL ORGANIZATION

In PHIGS, the graphics data is organized into a tree-like hierarchy of structures. Each structure is thought of as a node in the hierarchy and can be referenced by other structures; each structure can call another using the EXECUTE STRUCTURE element. This implementation allows for the creation of each structure component only once, and for its use during tree traversal as many times as needed. A *root* structure is the node from which other structures originate; a *parent* structure is one that references other structures.

A PHIGS structure is a collection of elements of the following types:

- *Output Primitives* are the basic graphics available for display. They include: POLYLINE, POLYMARKER, TEXT, ANNOTATION TEXT RELATIVE, FILL AREA, FILL AREA SET, CELL ARRAY, GENERALIZED DRAWING PRIMITIVE.

- *Attributes* control the appearance of output primitives. PHIGS provides functions that allow individual specification of attributes as well as groups of attributes called *bundles* for the specific types of primitives.

- *Modelling Transformations* are transformation matrices to convert structures edited in modelling coordinates into world coordinates.

- *View Selections* bind the display to the user specified view.

- *Structure Invocations* are elements mentioned above for referencing other structures.

- *Name Set* defines the group of primitives available for picking, highlighting and invisibility.

- *Pick Identifiers* are the elements which define the pick ids of all following elements in the structure network.

- *Labels* mark an element position within the structure to facilitate structure editing.

- *Application Data* provides the capability of storing non-executable data such as comments in the database.

- *Generalized Structure Element* provides the capability for accessing the implementation-dependent, workstation-dependent, or device-dependent data of the system.

4.2.2. GRAPHICAL DISPLAY

In PHIGS, the term *workstation* means the display device controlled by the application. From the user's point of view, workstations are the windows on the screen where graphics are displayed. An application can open multiple workstations which, in turn, can be split into multiple views. The same or different structures may be *posted* or assigned to each view or to a workstation. Thus, the application may control them independently.

4.2.3. TRANSFORMATIONS PIPELINE

As mentioned in the previous section, a structure can consist of a number of substructures which may be invoked by a parent structure as many times as required. The purpose of transformations is to allow the application to define each of these substructures in a convenient coordinate system. They can be then composed into the main structure and placed into the appropriate position onto the display. The mechanism which performs this, called the *transformation pipeline*, is a series of transformations. As shown in Figure 4-5, the transformation pipeline consists of four transformations: *composite modelling transformation*, *view orientation transformation*, *view mapping transformation* and *workstation transformation*.

The composite modelling transformation maps individual structures into one coordinate system—the world coordinate system (WC). It is composed of two modelling transformations: local and global modelling transformations. The difference between them is that the local modelling transformation affects only the current

structure whereas the global modelling transformation is also applied to all descendents in the structure network. PHIGS provides transformation functions, such as x, y, z rotation, translation and scaling, to calculate the transformation matrices which can then be inserted into a structure by the SET LOCAL TRANSFORMATION or SET GLOBAL TRANSFORMATION functions.

The purpose of the view orientation transformation is to transform the world coordinate space to the view reference coordinate system (VRC). The input parameters to the function EVALUATE VIEW ORIENTATION MATRIX, which produces the view orientation matrix, are: a view reference point, a view plane normal and a view up vector. The view reference point—the origin of the viewing space—, the view plane normal and the view up vector define the *View Plane*, the plane onto which the object is mapped. The view plane normal can be arbitrarily changed causing the effect of rotation of the coordinate system.

The view mapping transformation provides parallel and perspective transformation from VRC space to the normalized projection coordinate system (NPC). It determines the appearance of the object mapped onto the View Plane by setting the *view window* limits and the *projection reference point*. If the projection type is PERSPECTIVE, changing the location of the view reference point causes the effect of zooming.

The workstation transformation performs mapping from normalized projection coordinate space into device coordinate space. The application has control over what is to be displayed and its location by specifying the workstation window and workstation viewport.

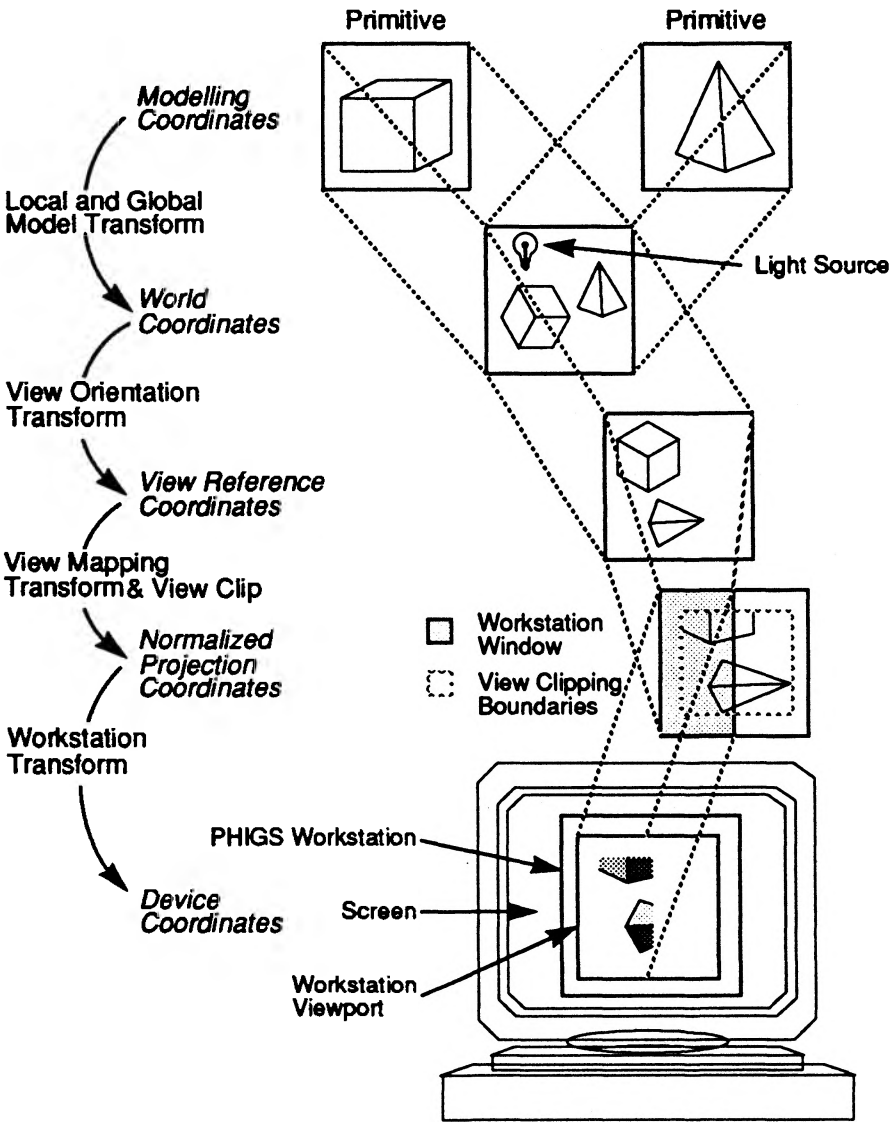

Figure 4-5. The transformation pipeline [2].

4.2.4. BASIC PROGRAMMING PRINCIPLES

Programming in PHIGS involves the following steps:

• Before calling any PHIGS function, PHIGS must be opened. The function OPEN PHIGS initializes PHIGS environment.

• Once PHIGS has been opened, it is possible to open the workstation through the OPEN WORKSTATION call. This function takes three parameters: the workstation id, the connection id and the workstation type. The first parameter is important if the application controls many workstations. The second one bounds the workstation with the proper X window if the workstation type is *X Drawable*. There are the few types of workstations which can be opened; the type of the workstation is the third parameter of the OPEN WORKSTATION function.

• How the graphics will be presented to the user depends on the viewing parameters. The viewing transformation is defined by the SET VIEW REPRESENTATION function. The view representation may specify parallel or perspective projection.

• After PHIGS and the workstation have been opened, a structure can be opened. The only parameter to the OPEN STRUCTURE function is a *structure identifier*, which defines the structure within the CSS to be opened. If no structure with this id exists, one is created. Once the structure has been opened, any structure element can be inserted. This includes also the elements referencing other structures.

• The structure to be displayed on the workstation must be posted to this workstation. It is not necessary to close the structure before posting. Therefore the structure can be interactively created or modified; any changes to it may be visible on the screen.

• CLOSE STRUCTURE, CLOSE WORKSTATION and CLOSE PHIGS are the last three functions executed prior to closing the application.

CHAPTER FIVE

DISCUSSION

This chapter consists of two sections. The first section discusses the features of 3D GAS and compares the approach to the design in 3D space implemented in 3D GAS with other CAD systems such as AutoCAD [1]and CADKEY [2]. In the second section some suggestions for future work are made.

5.1. PRESENT SCOPE OF 3D GAS

Generally, the features of any CAD system can be grouped as follows:

- the features needed to create a model;
- the features needed to view a model.

The features needed to create 3D models available in 3D GAS have been presented in chapter 2. They may be categorized into three groups:

- drawing routines based on point and line sets;
- drawing routines based on transformations in *INSERT MODE*;
- drawing routines utilizing *COPY MODE*.

Practically, design of any structure in 3D GAS can involve all three techniques. The first category is the main drawing technique of 3D GAS. Any structure created

[1] AutoCAD is a registered trademark of Autodesk, Inc
[2] CADKEY is a registered trademark of Micro Control Systems, Inc.

by the application may be drawn using this technique. Since the second and third categories operate on already existing models, they are the auxiliary drawing techniques.

The drawing techniques based on point and line sets were introduced in section 2.2. The routines presented there are the implementation of the approach which was invented to speed up the design process of some engineering and architectural projects. These routines allow for fast creation of sets of points and lines eliminating or greatly reducing the use of the keyboard as an input device. The application offers instead a wide range of user interface controls and a picking device represented by a mouse.

Drawing in 3D GAS is not bound to any plane and can occur in any direction. This is considered an advantage over the AutoCAD and CADKEY systems which are mostly based on working in planes and the extrusion of entities. In AutoCAD, these planes are called the *elevations*; they define the distance between the elevation (and so, all structures created in the elevation) and some base plane. The value of this distance is entered by the user and may be negative which means that the elevation is below the base plane. Creating 3D models in AutoCAD initially resembles a drawing in 2D. After an elevation is specified, the user finds her/himself in a plane and draws a view of a model using ordinary techniques available for 2D. The transition to 3D occurs by specifying the third dimension, called a *height* or a *thickness*, for each component of the model. The view created in the elevation is then translated by the value of the thickness and the corresponding points of the view and its copy are joined with straight lines. A similar drawing idea is used in CADKEY; the plane in which the first stage of the design occurs is defined here by a *current working depth.* In both cases, picking points on the screen returns their x and y coordinates; the third

dimension is defined by the working depth (CADKEY) or the thickness (AutoCAD).

Refraining from using a plane in 3D GAS as an indirect means of drawing and picking in 3D space makes the process of design very clear and simple. The fundamental principle in 3D GAS is that drawing "occurs in 3D space", i.e. the current stage of the drawing is always presented to the user in the three dimensional space. How it is presented depends on the view parameters selected. Secondly, any visible element of the edited structure is accessible for picking at any time during the drawing process. Picking elements of a structure returns their actual x, y, z coordinates in the world coordinate system as they are stored in the database.

Viewing models in 3D GAS does not change the coordinate system as it does in CADKEY. When working with CADKEY, the user is always in one of two coordinate systems: the world coordinate system or the view coordinate system. View coordinates in CADKEY are fixed to the screen, x horizontal, y vertical, z toward the user, and have the same orientation for all views.

5.2. FUTURE OF 3D GAS

The purpose of 3D GAS is to support an engineer or an architect in the design of three dimensional objects. As mentioned in the introduction, the system is supposed not only to provide a reliable tool for drawing and modelling 3D structures, but also to support the designer with databases related to the created models. These databases will be either built from data entered by the user or data calculated by the system. The present scope of the application reflects a first step in its evolution: providing a tool for drawing in 3D space. Some preparations, however, have already

been made for future work. These include the introduction of labels and the storage of the real dimensions of the created models.

Saving models into the file is implemented in 3D GAS by the PHIGS archive function. The file created by this function is the main database of 3D GAS. At present it is exclusively used for retrieving the structure for display—in the future, it might also be used for the extraction of some data required by the user. This can be achieved by specifying the labels of the elements in which the user has shown interest. Because in the application the labels are related to the points (nodes) of the structure, entering the value of the label would extract the coordinates of the associated point. Retrieving the data of lines (edges) would require the specification of two labels. Based on the database described above, other databases of the system can be created (for example databases for stresses, materials, costs, etc.).

Useful features which are not implemented in 3D GAS yet but whose implementation might be desirable are:

- grouping of the elements;
- adding colors to the drawing;
- introducing units and tolerance.

The purpose of grouping is to allow identification of a set of elements that defines a substructure of the structure. A group might be then treated as an entity for deletion, transformation or copying. The addition of colors to the drawing would allow for presentation of each of these groups in different color and make these groups easily distinguishable from each other.

The introduction of units and tolerance should not only focus on the units of

measure (inches, millimeters) but also on the notation in which they are typed in by the user, for example 1'-3.50" or 1'-3 1/2. The specification of tolerance will involve a similar problem.

APPENDIX A

COMPILATION

Before compiling 3D GAS make sure all libraries specified in the makefile are installed (the makefile is listed below). If the paths defined there differ from yours, define new paths for required libraries. To compile the program type: make.

RUNNING

To run the program type: gas. The first window you will see on the screen is the poster of the application. It reveals the title of the application and the name of its author. This poster will be displayed for a few seconds.

MAKEFILE

```
CC = cc
OBJ = gas.c
OPENWINHOME  = /u2/share/Openwin3
XGLHOME = /u2/share/SunPHIGS-2.0/lib/phigs2.0
PHIGSHOME = /u2/share/SunPHIGS-2.0
LIBS = -llphigs -lphigs -lxgl -lxview -lolgx -lX11 -lm
IPATH = -I/u2/share/Openwin3/include -I$(PHIGSHOME)/include
LPATH = -L$(OPENWINHOME)/lib -L$(PHIGSHOME)/lib
      -L$(XGLHOME)/lib
CFLAGS = -g -o


gas : $(OBJ)
$(CC) $(OBJ) $(CFLAGS) gas $(IPATH) $(LPATH) $(LIBS)
```

APPENDIX B

gas.c

```
#include "gas.h"

/*
 * Initialize a set of names for picking.
 */

static void
init_names_set()
{
    Pint        names[10];
    Pint_list   nset;

    names[0] = POINT;
    names[1] = LINE;
    names[2] = LABEL;
    names[3] = CIRCLE;
    names[4] = ARC_CIRCLE;
    names[5] = COPY_POINT;
    names[6] = COPY_LINE;
    names[7] = COPY_LABEL;
    names[8] = COPY_CIRCLE;
    names[9] = COPY_ARC_CIRCLE;

    nset.num_ints = 10;
    nset.ints = names;
    padd_names_set(&nset);
}

/*
 * Callback routine for the initialization of the Current Point Set.
 */

static void
cps_proc(item, val)
Panel_item item;
int        val;
{
    Pint       i;

    xv_set(item, PANEL_VALUE, 1, NULL);
    free(CPS.points);
    CPS.points = (Ppoint3 *)calloc(NP, sizeof(Ppoint3));
    for (i = 0; i < NP; i++)
    {
        CPS.points[i] = PS.points[i];
    }
    CPS.num_points = NP;
    NP = 0;
    NL = 0;
    NELEM = 0;
    elem_pos_list.num_ints = 0;
}
```

```
/*
 * Callback routine for the initialization of the Current Line Set.
 */

static void
cls_proc(item, val)
Panel_item item;
int        val;
{
    Pint       i;

    xv_set(item, PANEL_VALUE, 1, NULL);
    free(CLS.points);
    CLS.points = (Ppoint3 *)calloc(NL, sizeof(Ppoint3));
    for (i = 0; i < NL; i++)
    {
        CLS.points[i] = LS.points[i];
    }
    CLS.num_points = NL;
    NL = 0;
    NP = 0;
    NELEM = 0;
    elem_pos_list.num_ints = 0;
}

/*
 * Callback routine for the initialization of the Reference Point Set.
 */

static void
rps_proc(item, val)
Panel_item item;
int        val;
{
    Pint       i;

    xv_set(item, PANEL_VALUE, 1, NULL);
    free(RPS.points);
    RPS.points = (Ppoint3 *)calloc(NP, sizeof(Ppoint3));
    for (i = 0; i < NP; i++)
    {
        RPS.points[i] = PS.points[i];
    }
    RPS.num_points = NP;
    NP = 0;
    NL = 0;
    NELEM = 0;
    elem_pos_list.num_ints = 0;
}

/*
 * Callback routine for the initialization of the Reference Line Set.
 */

static void
rls_proc(item, val)
```

```
Panel_item item;
int       val;
{
    Pint       i;

    xv_set(item, PANEL_VALUE, 1, NULL);
    free(RLS.points);
    RLS.points = (Ppoint3 *)calloc(NL, sizeof(Ppoint3));
    for (i = 0; i < NL; i++)
    {
        RLS.points[i] = LS.points[i];
    }
    RLS.num_points = NL;
    NL = 0;
    NP = 0;
    NELEM = 0;
    elem_pos_list.num_ints = 0;
}

/*
 * Callback routine for
 * 1. the initialization of the drawing mode: INSERT or COPY
 * 2. the insertion of the COPY_OBJECT into the OBJECT structure
 * 3. the deletion of the COPY_OBJECT structure
 * Unfortunately, there is a bug in the PHIGS package. The pcopy_elem_range
 * function does not work for the GDP3 elements; copying of those elements
 * had to be done by hand.
 */

static void
delete_copy_proc(item, val)
Panel_item item;
int       val;
{
    Pint            err, elem_ptr, elem_ptr1, elem_ptr2;
    Pelem_type      type;
    size_t          size;
    Pelem_data      *gdp;
    Pstore          store;
    Pelem_type_list incl, excl;
    Pelem_type      in_el_type, ex_el_type;
    Psearch_status  search_st;
    Pstruct_st      struct_st;
    Pint            found_el, *int_data, ob_ptr_start, ob_ptr_end;

    switch ((int)xv_get(item, PANEL_CLIENT_DATA)) {
    case 1: switch(val ){
            case 0: MODE = COPY;
              pset_edit_mode(PEDIT_REPLACE);
              pinq_struct_st(&struct_st);
              if (struct_st == PSTRUCT_ST_STOP)
                pclose_struct();
              popen_struct(COPY_OBJECT);
              break;
            case 1: MODE = INSERT;
              pset_edit_mode(PEDIT_INSERT);
              pinq_struct_st(&struct_st);
```

```
        if (struct_st == PSTRUCT_ST_STOP)
          pclose_struct();
        popen_struct(OBJECT);
      break;
    }
case 2: switch(val ){
    case 0:
    if (MODE == COPY)
    {
      pinq_elem_ptr(&err, &elem_ptr1);
      if (elem_ptr1 != start_ptr - 1)
      {
        pcreate_store(&err, &store);
        pclose_struct();
        popen_struct(OBJECT);

        pinq_elem_ptr(&err, &ob_ptr_start);

        in_el_type = PELEM_GDP3;
        incl.num_elem_types = 1;
        incl.elem_types = &in_el_type;
        ex_el_type = PELEM_NIL;
        excl.num_elem_types = 1;
        excl.elem_types = &ex_el_type;

        pset_edit_mode(PEDIT_INSERT);
        elem_ptr2 = start_ptr;
        do
        {
          pelem_search(COPY_OBJECT, elem_ptr2, PDIR_FORWARD,
                    &incl, &excl, &err, &search_st, &found_el);
          if (search_st == PSEARCH_STATUS_SUCCESS)
          {
            pcopy_elem_range(COPY_OBJECT, elem_ptr2, found_el - 1);
            pinq_elem_content(COPY_OBJECT, found_el, store, &err,
                        &gdp);
            pgdp3(gdp->gdp3.point_list, gdp->gdp3.id,
                 gdp->gdp3.data);
            elem_ptr2 = found_el + 1;
          }
          else if (search_st == PSEARCH_STATUS_FAILURE)
            pcopy_elem_range(COPY_OBJECT, elem_ptr2, MAX_INT);

        }while (search_st == PSEARCH_STATUS_SUCCESS &&
              elem_ptr2 <= elem_ptr1);

        pset_edit_mode(PEDIT_REPLACE);

        pinq_elem_ptr(&err, &ob_ptr_end);
        pset_elem_ptr(ob_ptr_start);
        do
        {
          poffset_elem_ptr(1);
          pinq_elem_ptr(&err, &elem_ptr2);
          pinq_cur_elem_type_size(&err, &type, &size);
          if (type == PELEM_PICK_ID)
          {
```

```
                    pinq_cur_elem_content(store, &err, &int_data);
                    switch (*int_data){
                    case 21: pset_pick_id(POINT); break;
                    case 22: pset_pick_id(LINE); break;
                    case 23: pset_pick_id(LABEL); break;
                    case 24: pset_pick_id(CIRCLE); break;
                    case 25: pset_pick_id(ARC_CIRCLE); break;
                    }
                  }
                }while (elem_ptr2 < ob_ptr_end);

                pdel_store(store);
                pclose_struct();
                predraw_all_structs(WS, PFLAG_COND);
                popen_struct(COPY_OBJECT);
              }
            }
            xv_set(item, PANEL_VALUE, 2, NULL);
            break;
            case 1:
            if (MODE == COPY)
            {
                pinq_elem_ptr(&err, &elem_ptr);
                if (elem_ptr != start_ptr - 1)
                {
                  pdel_elem_range(start_ptr, MAX_INT);
                  predraw_all_structs(WS, PFLAG_COND);
                }
                xv_set(item, PANEL_VALUE, 0, NULL);
            }
            xv_set(item, PANEL_VALUE, 2, NULL);
            break;
            case 2: break;
            }
    }
}

/*
 * Callback routine for the addition of the element containig a Label
 */

static void
label_setting_proc(item, val)
Panel_item   item;
int          val;
{
    static char      label[20];
    static Ppoint3   anno_offset = {0.005, -0.003, 0.004};

    if (strcmp((char *)xv_get(item, PANEL_LABEL_STRING), "Label apply"))
    {
       switch((int)xv_get(item, PANEL_CLIENT_DATA)){
          case 1: strcpy(label, (char *)xv_get(item, PANEL_VALUE));
                  xv_set(item, PANEL_VALUE, "", NULL);
                  break;
          case 2: break;
          case 3: switch (val) {
```

```
                    case 0: anno_offset.x =  0.005;
                            anno_offset.y = -0.003;
                            anno_offset.z =  0.004;
                            break;
                    case 1: anno_offset.x =  0.004;
                            anno_offset.y = -0.010;
                            anno_offset.z =  0.004;
                            break;
                    case 2: anno_offset.x = -0.004;
                            anno_offset.y = -0.014;
                            anno_offset.z =  0.004;
                            break;
                    case 3: anno_offset.x = -0.010;
                            anno_offset.y = -0.010;
                            anno_offset.z =  0.004;
                            break;
                    case 4: anno_offset.x = -0.014;
                            anno_offset.y = -0.003;
                            anno_offset.z =  0.004;
                            break;
                    case 5: anno_offset.x = -0.010;
                            anno_offset.y =  0.004;
                            anno_offset.z =  0.004;
                            break;
                    case 6: anno_offset.x = -0.005;
                            anno_offset.y =  0.007;
                            anno_offset.z =  0.004;
                            break;
                    case 7: anno_offset.x =  0.004;
                            anno_offset.y =  0.004;
                            anno_offset.z =  0.004;
                            break;
                }
                break;
            }
        }
        else
        {
            if (MODE == COPY)
              pset_edit_mode(PEDIT_INSERT);
            if (MODE == INSERT)
              pset_pick_id(LABEL);
            else if (MODE == COPY)
              pset_pick_id(COPY_LABEL);
            panno_text_rel3(CPS.points[CPS.num_points-1], &anno_offset, label);
            if (MODE == COPY)
              pset_edit_mode(PEDIT_REPLACE);
        }
}

/*
 * Callback routine for entering a text in the text window.
 */

static void
text_setting_proc(item, val)
Panel_item   item;
```

```
int         val;
{
    static char     text[80];
    static Ppoint   ref_pt = {1.0, 98.0};
    Pstruct_st      struct_st;

    switch((int)xv_get(item, PANEL_CLIENT_DATA)){
    case 1: strcpy(text, (char *)xv_get(item, PANEL_VALUE));
            xv_set(item, PANEL_VALUE, "", NULL);
            break;
    case 2: pinq_struct_st(&struct_st);
            if (struct_st == PSTRUCT_ST_STOP)
              pclose_struct();
            if (MODE == INSERT)
              pset_edit_mode(PEDIT_REPLACE);
            popen_struct(TEXT_OBJECT);
            ptext(&ref_pt, text);
            pclose_struct();
            if (MODE == COPY)
            {
              popen_struct(COPY_OBJECT);
            }
            else if (MODE == INSERT)
            {
              pset_edit_mode(PEDIT_INSERT);
              popen_struct(OBJECT);
            }
            break;
    case 3: break;
    }
}

/*
 * Callback routine for the x, y and z rotations.
 * If mode is COPY the rotated structure replaces the existing one; in a case
 * of INSERT mode the structure is appended to the existing one.
 * This routine performs the calculations of the new coordinates of the
 * elements of the structure rather than using the transformation functions
 * available in PHIGS. The reason is that picking on the element should give
 * the actual coordinates whereas the PHIGS transformation functions don't
 * calculate them but just add the transformation matrices.
 * Unfortunately, there is a bug in the PHIGS package. The pcopy_elem_range
 * function does not work for the GDP3 elements; copying of those elements
 * had to be done by hand.
 */

static void
rotation_proc(item, val)
Panel_item item;
Pint       val;
{
    char            str[20];
    static Pfloat   rad;
    Pint            err;
    Pint            elem_ptr, ob_ptr;
    Pint            cur_elem_ptr;
    Pelem_type      type;
```

```
Ppoint_list3     *pt_list, *ln_list;
Ppoint3          dup_point[2];
Ppoint_list3     dup_list;
static Ppoint    rot_point = {0.0, 0.0};
Pstore           store;
Pfloat           temp;
size_t           size;
Pelem_data       *gdp;
Pelem_type_list  incl, excl;
Pelem_type       in_el_type, ex_el_type;
Psearch_status   search_st;
Pint             found_el, *int_data;

if (strcmp((char *)xv_get(item, PANEL_LABEL_STRING), "Rotate"))
{
   switch((int)xv_get(item, PANEL_CLIENT_DATA)){
   case 1: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
           xv_set(item, PANEL_VALUE, "", NULL);
           panel_advance_caret((Panel)xv_get(item,
                               PANEL_PARENT_PANEL));
           rad = atof(str);
           rad = DEG_TO_RAD(rad);
           break;
   case 2: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
           xv_set(item, PANEL_VALUE, "", NULL);
           rot_point.x = atof(str);
           panel_advance_caret((Panel)xv_get(item,
                               PANEL_PARENT_PANEL));
           break;
   case 3: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
           xv_set(item, PANEL_VALUE, "", NULL);
           rot_point.y = atof(str);
           panel_advance_caret((Panel)xv_get(item,
                               PANEL_PARENT_PANEL));
           break;
   case 4: break;
   }
}
else
{
   pcreate_store(&err, &store);
   if (MODE == COPY)
   {
     pinq_elem_ptr(&err, &elem_ptr);
     if (elem_ptr == start_ptr - 1)
     {
       pclose_struct();
       popen_struct(OBJECT);
       pinq_elem_ptr(&err, &ob_ptr);
       pclose_struct();
       popen_struct(COPY_OBJECT);
       if (ob_ptr != start_ptr - 1)
       {
         in_el_type = PELEM_GDP3;
         incl.num_elem_types = 1;
         incl.elem_types = &in_el_type;
         ex_el_type = PELEM_NIL;
```

```
        excl.num_elem_types = 1;
        excl.elem_types = &ex_el_type;

        pset_edit_mode(PEDIT_INSERT);
        elem_ptr = start_ptr;
        do
        {
          pelem_search(OBJECT, elem_ptr, PDIR_FORWARD,
                    &incl, &excl, &err, &search_st, &found_el);
          if (search_st == PSEARCH_STATUS_SUCCESS)
          {
            pcopy_elem_range(OBJECT, elem_ptr, found_el - 1);
            pinq_elem_content(OBJECT, found_el, store, &err,
                        &gdp);
            pgdp3(gdp->gdp3.point_list, gdp->gdp3.id,
                 gdp->gdp3.data);
            elem_ptr = found_el + 1;
          }
          else if (search_st == PSEARCH_STATUS_FAILURE)
            pcopy_elem_range(OBJECT, elem_ptr, MAX_INT);

        }while (search_st == PSEARCH_STATUS_SUCCESS &&
              elem_ptr <= ob_ptr );
        pset_edit_mode(PEDIT_REPLACE);
      }
    }
  }
  pinq_elem_ptr(&err, &elem_ptr);
  pset_elem_ptr(start_ptr - 1);
  do
  {
    poffset_elem_ptr(1);
    pinq_elem_ptr(&err, &cur_elem_ptr);
    pinq_cur_elem_type_size(&err, &type, &size);
    if (type == PELEM_PICK_ID && MODE == COPY)
    {
       pinq_cur_elem_content(store, &err, &int_data);
       switch (*int_data){
       case 11: pset_pick_id(COPY_POINT); break;
       case 12: pset_pick_id(COPY_LINE); break;
       case 13: pset_pick_id(COPY_LABEL); break;
       case 14: pset_pick_id(COPY_CIRCLE); break;
       case 15: pset_pick_id(COPY_ARC_CIRCLE); break;
       }
    }
    else if (type == PELEM_POLYMARKER3)
    {
       pinq_cur_elem_content(store, &err, &pt_list);
       dup_point[0] = pt_list->points[0];
       switch ((int)xv_get(item, PANEL_CLIENT_DATA)){
       case 1:
       temp = pt_list->points[0].y;
       pt_list->points[0].y = rot_point.x + (pt_list->points[0].y -
         rot_point.x) * cos(rad) - (pt_list->points[0].z -
         rot_point.y) * sin(rad);

       pt_list->points[0].z = rot_point.y + (pt_list->points[0].z -
```

```
      rot_point.y) * cos(rad) + (temp - rot_point.x) * sin(rad);
    break;
    case 2:
    temp = pt_list->points[0].x;
    pt_list->points[0].x = rot_point.x + (pt_list->points[0].x -
      rot_point.x) * cos(rad) - (pt_list->points[0].z -
      rot_point.y) * sin(rad);

    pt_list->points[0].z = rot_point.y + (pt_list->points[0].z -
      rot_point.y) * cos(rad) + (temp - rot_point.x) * sin(rad);
    break;
    case 3:
    temp = pt_list->points[0].x;
    pt_list->points[0].x = rot_point.x + (pt_list->points[0].x -
      rot_point.x) * cos(rad) - (pt_list->points[0].y -
      rot_point.y) * sin(rad);

    pt_list->points[0].y = rot_point.y + (pt_list->points[0].y -
      rot_point.y) * cos(rad) + (temp - rot_point.x) * sin(rad);
    break;
    }
    if (MODE == INSERT)
    {
     pset_elem_ptr(MAX_INT);
     if (join)
     {
      dup_point[1] = pt_list->points[0];
      dup_list.num_points = 2;
      dup_list.points = dup_point;
      pset_pick_id(LINE);
      ppolyline3(&dup_list);
     }
     pset_pick_id(POINT);

    }
    ppolymarker3(pt_list);
}
else if (type == PELEM_POLYLINE3)
{
    pinq_cur_elem_content(store, &err, &ln_list);
    switch ((int)xv_get(item, PANEL_CLIENT_DATA)){
    case 1:
    temp = ln_list->points[0].y;
    ln_list->points[0].y = rot_point.x + (ln_list->points[0].y -
      rot_point.x) * cos(rad) - (ln_list->points[0].z -
      rot_point.y) * sin(rad);

    ln_list->points[0].z = rot_point.y + (ln_list->points[0].z -
      rot_point.y) * cos(rad) + (temp - rot_point.x) * sin(rad);
    temp = ln_list->points[1].y;
    ln_list->points[1].y = rot_point.x + (ln_list->points[1].y -
      rot_point.x) * cos(rad) - (ln_list->points[1].z -
      rot_point.y) * sin(rad);

    ln_list->points[1].z = rot_point.y + (ln_list->points[1].z -
      rot_point.y) * cos(rad) + (temp - rot_point.x) * sin(rad);
    break;
```

```
    case 2:
    temp = ln_list->points[0].x;
    ln_list->points[0].x = rot_point.x + (ln_list->points[0].x -
      rot_point.x) * cos(rad) - (ln_list->points[0].z -
      rot_point.y) * sin(rad);

    ln_list->points[0].z = rot_point.y + (ln_list->points[0].z -
      rot_point.y) * cos(rad) + (temp - rot_point.x) * sin(rad);
    temp = ln_list->points[1].x;
    ln_list->points[1].x = rot_point.x + (ln_list->points[1].x -
      rot_point.x) * cos(rad) - (ln_list->points[1].z -
      rot_point.y) * sin(rad);

    ln_list->points[1].z = rot_point.y + (ln_list->points[1].z -
      rot_point.y) * cos(rad) + (temp - rot_point.x) * sin(rad);
    break;
    case 3:
    temp = ln_list->points[0].x;
    ln_list->points[0].x = rot_point.x + (ln_list->points[0].x -
      rot_point.x) * cos(rad) - (ln_list->points[0].y -
      rot_point.y) * sin(rad);

    ln_list->points[0].y = rot_point.y + (ln_list->points[0].y -
      rot_point.y) * cos(rad) + (temp - rot_point.x) * sin(rad);
    temp = ln_list->points[1].x;
    ln_list->points[1].x = rot_point.x + (ln_list->points[1].x -
      rot_point.x) * cos(rad) - (ln_list->points[1].y -
      rot_point.y) * sin(rad);

    ln_list->points[1].y = rot_point.y + (ln_list->points[1].y -
      rot_point.y) * cos(rad) + (temp - rot_point.x) * sin(rad);
    break;
    }
    if (MODE == INSERT)
    {
      pset_elem_ptr(MAX_INT);
      pset_pick_id(LINE);
    }
    ppolyline3(ln_list);
}
else if (type == PELEM_GDP3)
{
    pinq_cur_elem_content(store, &err, &gdp);
    if (gdp->gdp3.id == -2)
    {
    switch ((int)xv_get(item, PANEL_CLIENT_DATA)){
    case 1:
    temp = gdp->gdp3.data.gdp3_u2.center.y;
    gdp->gdp3.data.gdp3_u2.center.y = rot_point.y +
      (gdp->gdp3.data.gdp3_u2.center.y -
      rot_point.x) * cos(rad)-(gdp->gdp3.data.gdp3_u2.center.z -
      rot_point.y) * sin(rad);
    gdp->gdp3.data.gdp3_u2.center.z = rot_point.y +
      (gdp->gdp3.data.gdp3_u2.center.z -
      rot_point.y) * cos(rad) + (temp - rot_point.x) * sin(rad);
    temp = gdp->gdp3.data.gdp3_u2.dir[0].delta_y;
    gdp->gdp3.data.gdp3_u2.dir[0].delta_y =
```

```
    gdp->gdp3.data.gdp3_u2.dir[0].delta_y * cos(rad) -
    gdp->gdp3.data.gdp3_u2.dir[0].delta_z * sin(rad);
gdp->gdp3.data.gdp3_u2.dir[0].delta_z =
    gdp->gdp3.data.gdp3_u2.dir[0].delta_z * cos(rad) +
    temp * sin(rad);
temp = gdp->gdp3.data.gdp3_u2.dir[1].delta_y;
gdp->gdp3.data.gdp3_u2.dir[1].delta_y =
    gdp->gdp3.data.gdp3_u2.dir[1].delta_y * cos(rad) -
    gdp->gdp3.data.gdp3_u2.dir[1].delta_z * sin(rad);
gdp->gdp3.data.gdp3_u2.dir[1].delta_z =
    gdp->gdp3.data.gdp3_u2.dir[1].delta_z * cos(rad) +
    temp * sin(rad);
break;
case 2:
temp = gdp->gdp3.data.gdp3_u2.center.x;
gdp->gdp3.data.gdp3_u2.center.x = rot_point.x +
    (gdp->gdp3.data.gdp3_u2.center.x -
    rot_point.x) * cos(rad)-(gdp->gdp3.data.gdp3_u2.center.z -
    rot_point.y) * sin(rad);
gdp->gdp3.data.gdp3_u2.center.z = rot_point.y +
    (gdp->gdp3.data.gdp3_u2.center.z -
    rot_point.y) * cos(rad) + (temp - rot_point.x) * sin(rad);
temp = gdp->gdp3.data.gdp3_u2.dir[0].delta_x;
gdp->gdp3.data.gdp3_u2.dir[0].delta_x =
    gdp->gdp3.data.gdp3_u2.dir[0].delta_x * cos(rad) -
    gdp->gdp3.data.gdp3_u2.dir[0].delta_z * sin(rad);
gdp->gdp3.data.gdp3_u2.dir[0].delta_z =
    gdp->gdp3.data.gdp3_u2.dir[0].delta_z * cos(rad) +
    temp * sin(rad);
temp = gdp->gdp3.data.gdp3_u2.dir[1].delta_x;
gdp->gdp3.data.gdp3_u2.dir[1].delta_x =
    gdp->gdp3.data.gdp3_u2.dir[1].delta_x * cos(rad) -
    gdp->gdp3.data.gdp3_u2.dir[1].delta_z * sin(rad);
gdp->gdp3.data.gdp3_u2.dir[1].delta_z =
    gdp->gdp3.data.gdp3_u2.dir[1].delta_z * cos(rad) +
    temp * sin(rad);
break;
case 3:
temp = gdp->gdp3.data.gdp3_u2.center.x;
gdp->gdp3.data.gdp3_u2.center.x = rot_point.x +
    (gdp->gdp3.data.gdp3_u2.center.x-rot_point.x) * cos(rad) -
    (gdp->gdp3.data.gdp3_u2.center.y-rot_point.y) * sin(rad);
gdp->gdp3.data.gdp3_u2.center.y = rot_point.y +
    (gdp->gdp3.data.gdp3_u2.center.y-rot_point.y) * cos(rad) +
    (temp - rot_point.x) * sin(rad);
temp = gdp->gdp3.data.gdp3_u2.dir[0].delta_x;
gdp->gdp3.data.gdp3_u2.dir[0].delta_x =
    gdp->gdp3.data.gdp3_u2.dir[0].delta_x * cos(rad) -
    gdp->gdp3.data.gdp3_u2.dir[0].delta_y * sin(rad);
gdp->gdp3.data.gdp3_u2.dir[0].delta_y =
    gdp->gdp3.data.gdp3_u2.dir[0].delta_y * cos(rad) +
    temp * sin(rad);
temp = gdp->gdp3.data.gdp3_u2.dir[1].delta_x;
gdp->gdp3.data.gdp3_u2.dir[1].delta_x =
    gdp->gdp3.data.gdp3_u2.dir[1].delta_x * cos(rad) -
    gdp->gdp3.data.gdp3_u2.dir[1].delta_y * sin(rad);
gdp->gdp3.data.gdp3_u2.dir[1].delta_y =
```

```
    gdp->gdp3.data.gdp3_u2.dir[1].delta_y * cos(rad) +
    temp * sin(rad);
break;
}
if (MODE == INSERT)
{
  pset_elem_ptr(MAX_INT);
  pset_pick_id(CIRCLE);
}
pgdp3(pt_list, PUGDP_CIRCLE3, gdp->gdp3.data);
}
else if (gdp->gdp3.id == -3)
{
switch ((int)xv_get(item, PANEL_CLIENT_DATA)){
case 1:
temp = gdp->gdp3.data.gdp3_u3.center.y;
gdp->gdp3.data.gdp3_u3.center.y = rot_point.y +
    (gdp->gdp3.data.gdp3_u3.center.y -
    rot_point.x) * cos(rad)-(gdp->gdp3.data.gdp3_u3.center.z -
    rot_point.y) * sin(rad);
gdp->gdp3.data.gdp3_u3.center.z = rot_point.y +
    (gdp->gdp3.data.gdp3_u3.center.z -
    rot_point.y) * cos(rad) + (temp - rot_point.x) * sin(rad);
temp = gdp->gdp3.data.gdp3_u3.dir[0].delta_y;
gdp->gdp3.data.gdp3_u3.dir[0].delta_y =
    gdp->gdp3.data.gdp3_u3.dir[0].delta_y * cos(rad) -
    gdp->gdp3.data.gdp3_u3.dir[0].delta_z * sin(rad);
gdp->gdp3.data.gdp3_u3.dir[0].delta_z =
    gdp->gdp3.data.gdp3_u3.dir[0].delta_z * cos(rad) +
    temp * sin(rad);
temp = gdp->gdp3.data.gdp3_u3.dir[1].delta_y;
gdp->gdp3.data.gdp3_u3.dir[1].delta_y =
    gdp->gdp3.data.gdp3_u3.dir[1].delta_y * cos(rad) -
    gdp->gdp3.data.gdp3_u3.dir[1].delta_z * sin(rad);
gdp->gdp3.data.gdp3_u3.dir[1].delta_z =
    gdp->gdp3.data.gdp3_u3.dir[1].delta_z * cos(rad) +
    temp * sin(rad);
break;
case 2:
temp = gdp->gdp3.data.gdp3_u3.center.x;
gdp->gdp3.data.gdp3_u3.center.x = rot_point.x +
    (gdp->gdp3.data.gdp3_u3.center.x -
    rot_point.x) * cos(rad)-(gdp->gdp3.data.gdp3_u3.center.z -
    rot_point.y) * sin(rad);
gdp->gdp3.data.gdp3_u3.center.z = rot_point.y +
    (gdp->gdp3.data.gdp3_u3.center.z-rot_point.y) * cos(rad) +
    (temp - rot_point.x) * sin(rad);
temp = gdp->gdp3.data.gdp3_u3.dir[0].delta_x;
gdp->gdp3.data.gdp3_u3.dir[0].delta_x =
    gdp->gdp3.data.gdp3_u3.dir[0].delta_x * cos(rad) -
    gdp->gdp3.data.gdp3_u3.dir[0].delta_z * sin(rad);
gdp->gdp3.data.gdp3_u3.dir[0].delta_z =
    gdp->gdp3.data.gdp3_u3.dir[0].delta_z * cos(rad) +
    temp * sin(rad);
temp = gdp->gdp3.data.gdp3_u3.dir[1].delta_x;
gdp->gdp3.data.gdp3_u3.dir[1].delta_x =
    gdp->gdp3.data.gdp3_u3.dir[1].delta_x * cos(rad) -
```

```
                gdp->gdp3.data.gdp3_u3.dir[1].delta_z * sin(rad);
              gdp->gdp3.data.gdp3_u3.dir[1].delta_z =
                gdp->gdp3.data.gdp3_u3.dir[1].delta_z * cos(rad) +
                temp * sin(rad);
              break;
              case 3:
              temp = gdp->gdp3.data.gdp3_u3.center.x;
              gdp->gdp3.data.gdp3_u3.center.x = rot_point.x +
                (gdp->gdp3.data.gdp3_u3.center.x-rot_point.x) * cos(rad) -
                (gdp->gdp3.data.gdp3_u3.center.y-rot_point.y) * sin(rad);
              gdp->gdp3.data.gdp3_u3.center.y = rot_point.y +
                (gdp->gdp3.data.gdp3_u3.center.y-rot_point.y) * cos(rad) +
                (temp - rot_point.x) * sin(rad);
              temp = gdp->gdp3.data.gdp3_u3.dir[0].delta_x;
              gdp->gdp3.data.gdp3_u3.dir[0].delta_x =
                gdp->gdp3.data.gdp3_u3.dir[0].delta_x * cos(rad) -
                gdp->gdp3.data.gdp3_u3.dir[0].delta_y * sin(rad);
              gdp->gdp3.data.gdp3_u3.dir[0].delta_y =
                gdp->gdp3.data.gdp3_u3.dir[0].delta_y * cos(rad) +
                temp * sin(rad);
              temp = gdp->gdp3.data.gdp3_u3.dir[1].delta_x;
              gdp->gdp3.data.gdp3_u3.dir[1].delta_x =
                gdp->gdp3.data.gdp3_u3.dir[1].delta_x * cos(rad) -
                gdp->gdp3.data.gdp3_u3.dir[1].delta_y * sin(rad);
              gdp->gdp3.data.gdp3_u3.dir[1].delta_y =
                gdp->gdp3.data.gdp3_u3.dir[1].delta_y * cos(rad) +
                temp * sin(rad);
              break;
              }
              if (MODE == INSERT)
              {
                pset_elem_ptr(MAX_INT);
                pset_pick_id(ARC_CIRCLE);
              }
              pgdp3(pt_list, PUGDP_CIRC_ARC3, gdp->gdp3.data);
              }
            }
            if (MODE == INSERT)
              pset_elem_ptr(cur_elem_ptr);
          }while (cur_elem_ptr != elem_ptr);
          pdel_store(store);
          if (MODE == INSERT)
          {
            pset_elem_ptr(MAX_INT);
          }
          else if (MODE == COPY)
          {
            predraw_all_structs(WS, PFLAG_COND);
          }
        }
}

/*
 * Callback routine for the x, y and z translations.
 * If mode is COPY the rotated structure replaces the existing one; in a case
 * of INSERT mode the structure is appended to the existing one.
 * This routine performs the calculations of the new coordinates of the
```

```
 * elements of the structure rather than using the transformation functions
 * available in PHIGS. The reason is that picking on the element should give
 * the actual coordinates whereas the PHIGS transformation functions don't
 * calculate them but just add the transformation matrices.
 * Unfortunately, there is a bug in the PHIGS package. The pcopy_elem_range
 * function does not work for the GDP3 elements; copying of those elements
 * had to be done by hand.
 */

static void
shift_proc(item, val)
Panel_item item;
Pint       val;
{
    char            str[20];
    Pint            err;
    Pint            elem_ptr, ob_ptr;
    Pint            cur_elem_ptr;
    Pelem_type      type;
    Ppoint_list3    *pt_list, *ln_list;
    Ppoint3         dup_point[2];
    Ppoint_list3    dup_list;
    Pstore          store;
    Pfloat          temp;
    size_t          size;
    static Pvec3    vec;
    Pelem_data      *gdp;
    Pelem_type_list incl, excl;
    Pelem_type      in_el_type, ex_el_type;
    Psearch_status  search_st;
    Pint            found_el, *int_data;

    if (strcmp((char *)xv_get(item, PANEL_LABEL_STRING), "Translate"))
    {
    switch((int)xv_get(item, PANEL_CLIENT_DATA)){
       case 1: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
               vec.delta_x = atof(str);
               vec.delta_x = vec.delta_x * scale;
               xv_set(item, PANEL_VALUE, "", NULL);
               panel_advance_caret((Panel)xv_get(item,
                                   PANEL_PARENT_PANEL));
               break;
       case 2: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
               vec.delta_y = atof(str);
               vec.delta_y = vec.delta_y * scale;
               xv_set(item, PANEL_VALUE, "", NULL);
               panel_advance_caret((Panel)xv_get(item,
                                   PANEL_PARENT_PANEL));
               break;
       case 3: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
               vec.delta_z = atof(str);
               vec.delta_z = vec.delta_z * scale;
               xv_set(item, PANEL_VALUE, "", NULL);
               panel_advance_caret((Panel)xv_get(item,
                                   PANEL_PARENT_PANEL));
               break;
       case 4: break;
```

```
}
} else
{
   pcreate_store(&err, &store);
   if (MODE == COPY)
   {
     pinq_elem_ptr(&err, &elem_ptr);
     if (elem_ptr == start_ptr - 1)
     {
       pclose_struct();
       popen_struct(OBJECT);
       pinq_elem_ptr(&err, &ob_ptr);
       pclose_struct();
       popen_struct(COPY_OBJECT);
       if (ob_ptr != start_ptr - 1)
       {
         in_el_type = PELEM_GDP3;
         incl.num_elem_types = 1;
         incl.elem_types = &in_el_type;
         ex_el_type = PELEM_NIL;
         excl.num_elem_types = 1;
         excl.elem_types = &ex_el_type;

         pset_edit_mode(PEDIT_INSERT);
         elem_ptr = start_ptr;
         do
         {
           pelem_search(OBJECT, elem_ptr, PDIR_FORWARD,
                    &incl, &excl, &err, &search_st, &found_el);
           if (search_st == PSEARCH_STATUS_SUCCESS)
           {
             pcopy_elem_range(OBJECT, elem_ptr, found_el - 1);
             pinq_elem_content(OBJECT, found_el, store, &err,
                          &gdp);
             pgdp3(gdp->gdp3.point_list, gdp->gdp3.id,
                  gdp->gdp3.data);
             elem_ptr = found_el + 1;
           }
           else if (search_st == PSEARCH_STATUS_FAILURE)
             pcopy_elem_range(OBJECT, elem_ptr, MAX_INT);

         }while (search_st == PSEARCH_STATUS_SUCCESS &&
               elem_ptr <= ob_ptr );
         pset_edit_mode(PEDIT_REPLACE);
       }
     }
   }
   pinq_elem_ptr(&err, &elem_ptr);
   pset_elem_ptr(start_ptr - 1);
   do
   {
     poffset_elem_ptr(1);
     pinq_elem_ptr(&err, &cur_elem_ptr);
     pinq_cur_elem_type_size(&err, &type, &size);
     if (type == PELEM_PICK_ID && MODE == COPY)
     {
       pinq_cur_elem_content(store, &err, &int_data);
```

```
    switch (*int_data){
    case 11: pset_pick_id(COPY_POINT); break;
    case 12: pset_pick_id(COPY_LINE); break;
    case 13: pset_pick_id(COPY_LABEL); break;
    case 14: pset_pick_id(COPY_CIRCLE); break;
    case 15: pset_pick_id(COPY_ARC_CIRCLE); break;
    }
}
else if (type == PELEM_POLYMARKER3)
{
    pinq_cur_elem_content(store, &err, &pt_list);
    dup_point[0] = pt_list->points[0];
    pt_list->points[0].x = pt_list->points[0].x + vec.delta_x;
    pt_list->points[0].y = pt_list->points[0].y + vec.delta_y;
    pt_list->points[0].z = pt_list->points[0].z + vec.delta_z;

    if (MODE == INSERT)
    {
      pset_elem_ptr(MAX_INT);
      if (join)
      {
        dup_point[1] = pt_list->points[0];
        dup_list.num_points = 2;
        dup_list.points = dup_point;
        pset_pick_id(LINE);
        ppolyline3(&dup_list);
      }
      pset_pick_id(POINT);
    }
    ppolymarker3(pt_list);
}
else if (type == PELEM_POLYLINE3)
{
    pinq_cur_elem_content(store, &err, &ln_list);
    ln_list->points[0].x = ln_list->points[0].x + vec.delta_x;
    ln_list->points[0].y = ln_list->points[0].y + vec.delta_y;
    ln_list->points[0].z = ln_list->points[0].z + vec.delta_z;

    ln_list->points[1].x = ln_list->points[1].x + vec.delta_x;
    ln_list->points[1].y = ln_list->points[1].y + vec.delta_y;
    ln_list->points[1].z = ln_list->points[1].z + vec.delta_z;

    if (MODE == INSERT)
    {
      pset_elem_ptr(MAX_INT);
      pset_pick_id(LINE);
    }
    ppolyline3(ln_list);
}
else if (type == PELEM_GDP3)
{
    pinq_cur_elem_content(store, &err, &gdp);
    if (gdp->gdp3.id == -2)
    {
      gdp->gdp3.data.gdp3_u2.center.x =
      gdp->gdp3.data.gdp3_u2.center.x + vec.delta_x;
      gdp->gdp3.data.gdp3_u2.center.y =
```

```
                gdp->gdp3.data.gdp3_u2.center.y + vec.delta_y;
                gdp->gdp3.data.gdp3_u2.center.z =
                gdp->gdp3.data.gdp3_u2.center.z + vec.delta_z;

                if (MODE == INSERT)
                {
                  pset_elem_ptr(MAX_INT);
                  pset_pick_id(CIRCLE);
                }
                pgdp3(pt_list, PUGDP_CIRCLE3, gdp->gdp3.data);
              }
              else if (gdp->gdp3.id == -3)
              {
                gdp->gdp3.data.gdp3_u3.center.x =
                gdp->gdp3.data.gdp3_u3.center.x + vec.delta_x;
                gdp->gdp3.data.gdp3_u3.center.y =
                gdp->gdp3.data.gdp3_u3.center.y + vec.delta_y;
                gdp->gdp3.data.gdp3_u3.center.z =
                gdp->gdp3.data.gdp3_u3.center.z + vec.delta_z;

                if (MODE == INSERT)
                {
                  pset_elem_ptr(MAX_INT);
                  pset_pick_id(ARC_CIRCLE);
                }
                pgdp3(pt_list, PUGDP_CIRC_ARC3, gdp->gdp3.data);
              }
            }
            if (MODE == INSERT)
              pset_elem_ptr(cur_elem_ptr);
          }while (cur_elem_ptr != elem_ptr);
          pdel_store(store);
          if (MODE == INSERT)
          {
            pset_elem_ptr(MAX_INT);
          }
          else if (MODE == COPY)
          {
            predraw_all_structs(WS, PFLAG_COND);
          }
        }
}

/*
 * Callback routine for the x, y and z scaling.
 * This routine performs the calculations of the new coordinates of the
 * elements of the structure rather than using the transformation functions
 * available in PHIGS. The reason is that picking on the element should give
 * the actual coordinates whereas the PHIGS transformation functions don't
 * calculate them but just add the transformation matrices.
 * Unfortunately, there is a bug in the PHIGS package. The pcopy_elem_range
 * function does not work for the GDP3 elements; copying of those elements
 * had to be done by hand.
 */

static void
scale_proc(item, val)
```

```
Panel_item item;
Pint       val;
{
    char            str[20];
    Pint            err;
    Pint            elem_ptr, ob_ptr;
    Pint            cur_elem_ptr;
    Pelem_type      type;
    Ppoint_list3    *pt_list, *ln_list;
    Ppoint_list3    line_list[2];
    Pstore          store;
    Pfloat          temp;
    size_t          size;
    static Pvec3    vec = {1.0, 1.0, 1.0};
    Pelem_data      *gdp;
    Pelem_type_list incl, excl;
    Pelem_type      in_el_type, ex_el_type;
    Psearch_status  search_st;
    Pint            found_el, *int_data;

    if (strcmp((char *)xv_get(item, PANEL_LABEL_STRING), "Scale"))
    {
    switch((int)xv_get(item, PANEL_CLIENT_DATA)){
       case 1: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
               vec.delta_x = atof(str);
               if (vec.delta_x == 0)
                 vec.delta_x = 1;
               xv_set(item, PANEL_VALUE, "", NULL);
               panel_advance_caret((Panel)xv_get(item,
                                   PANEL_PARENT_PANEL));
               break;
       case 2: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
               vec.delta_y = atof(str);
               if (vec.delta_y == 0)
                 vec.delta_y = 1;
               xv_set(item, PANEL_VALUE, "", NULL);
               panel_advance_caret((Panel)xv_get(item,
                                   PANEL_PARENT_PANEL));
               break;
       case 3: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
               vec.delta_z = atof(str);
               if (vec.delta_z == 0)
                 vec.delta_z = 1;
               xv_set(item, PANEL_VALUE, "", NULL);
               panel_advance_caret((Panel)xv_get(item,
                                   PANEL_PARENT_PANEL));
               break;
       case 4: break;
    }
    } else
    {
       if (MODE == INSERT)
         pset_edit_mode(PEDIT_REPLACE);
       pcreate_store(&err, &store);
       if (MODE == COPY)
       {
         pinq_elem_ptr(&err, &elem_ptr);
```

```
  if (elem_ptr == start_ptr - 1)
  {
    pclose_struct();
    popen_struct(OBJECT);
    pinq_elem_ptr(&err, &ob_ptr);
    pclose_struct();
    popen_struct(COPY_OBJECT);
    if (ob_ptr != start_ptr - 1)
    {
      in_el_type = PELEM_GDP3;
      incl.num_elem_types = 1;
      incl.elem_types = &in_el_type;
      ex_el_type = PELEM_NIL;
      excl.num_elem_types = 1;
      excl.elem_types = &ex_el_type;

      pset_edit_mode(PEDIT_INSERT);
      elem_ptr = start_ptr;
      do
      {
        pelem_search(OBJECT, elem_ptr, PDIR_FORWARD,
                  &incl, &excl, &err, &search_st, &found_el);
        if (search_st == PSEARCH_STATUS_SUCCESS)
        {
          pcopy_elem_range(OBJECT, elem_ptr, found_el - 1);
          pinq_elem_content(OBJECT, found_el, store, &err,
                       &gdp);
          pgdp3(gdp->gdp3.point_list, gdp->gdp3.id,
               gdp->gdp3.data);
          elem_ptr = found_el + 1;
        }
        else if (search_st == PSEARCH_STATUS_FAILURE)
          pcopy_elem_range(OBJECT, elem_ptr, MAX_INT);

      }while (search_st == PSEARCH_STATUS_SUCCESS &&
            elem_ptr <= ob_ptr );
      pset_edit_mode(PEDIT_REPLACE);
    }
  }
}
pinq_elem_ptr(&err, &elem_ptr);
pset_elem_ptr(start_ptr - 1);
do
{
  poffset_elem_ptr(1);
  pinq_elem_ptr(&err, &cur_elem_ptr);
  pinq_cur_elem_type_size(&err, &type, &size);
  if (type == PELEM_PICK_ID && MODE == COPY)
  {
    pinq_cur_elem_content(store, &err, &int_data);
    switch (*int_data){
    case 11: pset_pick_id(COPY_POINT); break;
    case 12: pset_pick_id(COPY_LINE); break;
    case 13: pset_pick_id(COPY_LABEL); break;
    case 14: pset_pick_id(COPY_CIRCLE); break;
    case 15: pset_pick_id(COPY_ARC_CIRCLE); break;
    }
```

```
    }
    else if (type == PELEM_POLYMARKER3)
    {
        pinq_cur_elem_content(store, &err, &pt_list);
        pt_list->points[0].x = pt_list->points[0].x * vec.delta_x;
        pt_list->points[0].y = pt_list->points[0].y * vec.delta_y;
        pt_list->points[0].z = pt_list->points[0].z * vec.delta_z;

        ppolymarker3(pt_list);
    }
    else if (type == PELEM_POLYLINE3)
    {
        pinq_cur_elem_content(store, &err, &ln_list);
        ln_list->points[0].x = ln_list->points[0].x * vec.delta_x;
        ln_list->points[0].y = ln_list->points[0].y * vec.delta_y;
        ln_list->points[0].z = ln_list->points[0].z * vec.delta_z;

        ln_list->points[1].x = ln_list->points[1].x * vec.delta_x;
        ln_list->points[1].y = ln_list->points[1].y * vec.delta_y;
        ln_list->points[1].z = ln_list->points[1].z * vec.delta_z;

        ppolyline3(ln_list);
    }
    else if (type == PELEM_GDP3)
    {
        pinq_cur_elem_content(store, &err, &gdp);
        if (gdp->gdp3.id == -2)
        {
          gdp->gdp3.data.gdp3_n2.radius =
          gdp->gdp3.data.gdp3_n2.radius * vec.delta_x;
          gdp->gdp3.data.gdp3_n2.center.x =
          gdp->gdp3.data.gdp3_n2.center.x * vec.delta_x;
          gdp->gdp3.data.gdp3_n2.center.y =
          gdp->gdp3.data.gdp3_n2.center.y * vec.delta_y;
          gdp->gdp3.data.gdp3_n2.center.z =
          gdp->gdp3.data.gdp3_n2.center.z * vec.delta_z;
        }
        else if (gdp->gdp3.id == -3)
        {
          gdp->gdp3.data.gdp3_n3.radius =
          gdp->gdp3.data.gdp3_n3.radius * vec.delta_x;
          gdp->gdp3.data.gdp3_n3.center.x =
          gdp->gdp3.data.gdp3_n3.center.x * vec.delta_x;
          gdp->gdp3.data.gdp3_n3.center.y =
          gdp->gdp3.data.gdp3_n3.center.y * vec.delta_y;
          gdp->gdp3.data.gdp3_n3.center.z =
          gdp->gdp3.data.gdp3_n3.center.z * vec.delta_z;
        }
        pgdp3(gdp->gdp3.point_list, gdp->gdp3.id, gdp->gdp3.data);
    }
}while (cur_elem_ptr != elem_ptr);
pdel_store(store);
if (MODE == INSERT)
{
  pset_edit_mode(PEDIT_INSERT);
}
else if (MODE == COPY)
```

```
        {
            predraw_all_structs(WS, PFLAG_COND);
        }
    }
}

/*
 * Callback routine for sliders to change the view representation.
 */

static void
view_proc(item, value)
Panel_item item;
Pint       value;
{
    switch((int)xv_get(item, PANEL_CLIENT_DATA)){
        case 1: vpn.delta_x = (int)xv_get(item, PANEL_VALUE)/100.0;
                break;
        case 2: vpn.delta_y = (int)xv_get(item, PANEL_VALUE)/100.0;
                break;
        case 3: vpn.delta_z = (int)xv_get(item, PANEL_VALUE)/100.0;
                break;
        case 4: map.view_plane = (int)xv_get(item, PANEL_VALUE);
                break;
        case 5: switch(value){
                    case 0: map.proj_type = PTYPE_PERSPECT; break;
                    case 1: map.proj_type = PTYPE_PARAL; break;
                }
                break;
    }
    pset_disp_upd_st(WS, PDEFER_ASAP, PMODE_NIVE);
    pset_edit_mode(PEDIT_REPLACE);
    set_view_rep();
    pset_edit_mode(PEDIT_INSERT);
    pset_disp_upd_st(WS, PDEFER_WAIT, PMODE_UQUM);
}

/*
 * Callback routine for drawing a point.
 */

static void
create_point_subpanel(panel)
Panel panel;
{
    Pint       row = 0, col = 0;

    xv_create(panel, PANEL_TEXT,
        PANEL_LABEL_X,      5,
        PANEL_LABEL_Y,      xv_row(panel, row),
        PANEL_LABEL_STRING, "Point.x:",
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + YELLOW,
        PANEL_NOTIFY_PROC,  drawing_proc,
        PANEL_VALUE_DISPLAY_LENGTH, 6,
        PANEL_CLIENT_DATA,  9,
        NULL);
    row = 1;
```

```
    xv_create(panel, PANEL_TEXT,
        PANEL_LABEL_X,      5,
        PANEL_LABEL_Y,      xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + YELLOW,
        PANEL_LABEL_STRING, "Point.y:",
        PANEL_NOTIFY_PROC,  drawing_proc,
        PANEL_VALUE_DISPLAY_LENGTH, 6,
        PANEL_CLIENT_DATA,  10,
        NULL);
    row = 2;
    xv_create(panel, PANEL_TEXT,
        PANEL_LABEL_X,      5,
        PANEL_LABEL_Y,      xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + YELLOW,
        PANEL_LABEL_STRING, "Point.z:",
        PANEL_NOTIFY_PROC,  drawing_proc,
        PANEL_VALUE_DISPLAY_LENGTH, 6,
        PANEL_CLIENT_DATA,  11,
        NULL);
    row = 4;
    xv_create(panel, PANEL_BUTTON,
        XV_X,               5,
        XV_Y,               xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + YELLOW,
        PANEL_LABEL_STRING, "Draw point",
        PANEL_NOTIFY_PROC,  drawing_proc,
        PANEL_CLIENT_DATA,  13,
        NULL);
    row = 4;
    col = 15;
    xv_create(panel, PANEL_BUTTON,
        XV_X,               xv_col(panel, col),
        XV_Y,               xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + YELLOW,
        PANEL_LABEL_STRING, "Cancel",
        PANEL_NOTIFY_PROC,  drawing_proc,
        PANEL_CLIENT_DATA,  12,
        NULL);
}

/*
 * This routine creates a panel for labels.
 */

static void
create_label_subpanel(panel)
Panel panel;
{
    Pint        row = 0, col = 0;

    row = 0;
    xv_create(panel, PANEL_TEXT,
        PANEL_LABEL_X,      5,
        PANEL_LABEL_Y,      xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + YELLOW,
        PANEL_LABEL_STRING, "Enter label value:",
        PANEL_NOTIFY_PROC,  label_setting_proc,
```

```
        PANEL_VALUE_DISPLAY_LENGTH, 6,
        PANEL_CLIENT_DATA, 1,
        NULL);
    row = 1;
    col = 0;
    xv_create( panel, PANEL_CYCLE,
        XV_Y,  xv_row(panel, row),
        XV_X,  xv_col(panel, col),
        PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + YELLOW,
        PANEL_LABEL_STRING, "Label layout:",
        PANEL_CHOICE_STRINGS, "  EAST ",
                        "  SOUTH-EAST ",
                        "  SOUTH ",
                        "  SOUTH-WEST ",
                        "  WEST ",
                        "  NORTH-WEST ",
                        "  NORTH ",
                        "  NORTH-EAST ",
                        NULL,
        PANEL_NOTIFY_PROC, label_setting_proc,
        PANEL_CLIENT_DATA,  3,
        NULL);
    row = 4;
    col = 0;
    xv_create(panel, PANEL_BUTTON,
        XV_X,           xv_col(panel, col),
        XV_Y,           xv_row(panel, row),
        PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + YELLOW,
        PANEL_LABEL_STRING, "Label apply",
        PANEL_NOTIFY_PROC,  label_setting_proc,
        NULL);
    row = 4;
    col = 15;
    xv_create(panel, PANEL_BUTTON,
        XV_X,           xv_col(panel, col),
        XV_Y,           xv_row(panel, row),
        PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + YELLOW,
        PANEL_LABEL_STRING, "Cancel",
        PANEL_NOTIFY_PROC,  label_setting_proc,
        PANEL_CLIENT_DATA,  2,
        NULL);
}

/*
 * This routine creates a panel for a text displayed in the text window.
 */

static void
create_text_subpanel(panel)
Panel panel;
{
    Pint        row = 0, col = 0;

    row = 1;
    xv_create(panel, PANEL_TEXT,
        PANEL_LABEL_X,      5,
        PANEL_LABEL_Y,      xv_row(panel, row),
```

```
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + YELLOW,
        PANEL_LABEL_STRING, "Enter text:",
        PANEL_NOTIFY_PROC,  text_setting_proc,
        PANEL_VALUE_DISPLAY_LENGTH, 20,
        PANEL_CLIENT_DATA,  1,
        NULL);
    row = 2;
    col = 0;
    xv_create(panel, PANEL_BUTTON,
        XV_X,               xv_col(panel, col),
        XV_Y,               xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + YELLOW,
        PANEL_LABEL_STRING, "Draw text",
        PANEL_NOTIFY_PROC,  text_setting_proc,
        PANEL_CLIENT_DATA,  2,
        NULL);
    row = 2;
    col = 15;
    xv_create(panel, PANEL_BUTTON,
        XV_X,               xv_col(panel, col),
        XV_Y,               xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + YELLOW,
        PANEL_LABEL_STRING, "Cancel",
        PANEL_NOTIFY_PROC,  text_setting_proc,
        PANEL_CLIENT_DATA,  3,
        NULL);
}

/*
 * This routine creates a panel for the x, y and z rotations.
 */

static void
create_rotate_subpanel(panel, mode)
Panel panel;
Pint  mode;
{
    Pint      row = 0, col = 0;
    static char rot_x[] = "Enter rotation point.x:";
    static char rot_y[] = "Enter rotation point.y:";
    static char rot_z[] = "Enter rotation point.z:";

    row = 0;
    col = 0;
    xv_create(panel, PANEL_TEXT,
        PANEL_LABEL_X,      5,
        PANEL_LABEL_Y,      xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
        PANEL_LABEL_STRING, "Enter rotation angle:",
        PANEL_NOTIFY_PROC,  rotation_proc,
        PANEL_VALUE_DISPLAY_LENGTH, 6,
        PANEL_CLIENT_DATA,  1,
        NULL);
    if (mode == 1)
    {
      row = 1;
      col = 0;
```

```
  xv_create(panel, PANEL_TEXT,
      PANEL_LABEL_X,      5,
      PANEL_LABEL_Y,      xv_row(panel, row),
      PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
      PANEL_LABEL_STRING, rot_y,
      PANEL_NOTIFY_PROC,  rotation_proc,
      PANEL_VALUE_DISPLAY_LENGTH, 6,
      PANEL_CLIENT_DATA,  2,
      NULL);
  row = 2;
  col = 0;
  xv_create(panel, PANEL_TEXT,
      PANEL_LABEL_X,      5,
      PANEL_LABEL_Y,      xv_row(panel, row),
      PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
      PANEL_LABEL_STRING, rot_z,
      PANEL_NOTIFY_PROC,  rotation_proc,
      PANEL_VALUE_DISPLAY_LENGTH, 6,
      PANEL_CLIENT_DATA,  3,
      NULL);
}
else if (mode == 2)
{
  row = 1;
  col = 0;
  xv_create(panel, PANEL_TEXT,
      PANEL_LABEL_X,      5,
      PANEL_LABEL_Y,      xv_row(panel, row),
      PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
      PANEL_LABEL_STRING, rot_x,
      PANEL_NOTIFY_PROC,  rotation_proc,
      PANEL_VALUE_DISPLAY_LENGTH, 6,
      PANEL_CLIENT_DATA,  2,
      NULL);
  row = 2;
  col = 0;
  xv_create(panel, PANEL_TEXT,
      PANEL_LABEL_X,      5,
      PANEL_LABEL_Y,      xv_row(panel, row),
      PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
      PANEL_LABEL_STRING, rot_z,
      PANEL_NOTIFY_PROC,  rotation_proc,
      PANEL_VALUE_DISPLAY_LENGTH, 6,
      PANEL_CLIENT_DATA,  3,
      NULL);
}
else if (mode == 3)
{
  row = 1;
  col = 0;
  xv_create(panel, PANEL_TEXT,
      PANEL_LABEL_X,      5,
      PANEL_LABEL_Y,      xv_row(panel, row),
      PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
      PANEL_LABEL_STRING, rot_x,
      PANEL_NOTIFY_PROC,  rotation_proc,
      PANEL_VALUE_DISPLAY_LENGTH, 6,
```

```
            PANEL_CLIENT_DATA, 2,
            NULL);
        row = 2;
        col = 0;
        xv_create(panel, PANEL_TEXT,
            PANEL_LABEL_X,      5,
            PANEL_LABEL_Y,      xv_row(panel, row),
            PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
            PANEL_LABEL_STRING, rot_y,
            PANEL_NOTIFY_PROC,  rotation_proc,
            PANEL_VALUE_DISPLAY_LENGTH, 6,
            PANEL_CLIENT_DATA,  3,
            NULL);
    }
    row = 4;
    col = 0;
    xv_create(panel, PANEL_BUTTON,
        XV_X,               xv_col(panel, col),
        XV_Y,               xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
        PANEL_LABEL_STRING, "Rotate",
        PANEL_NOTIFY_PROC,  rotation_proc,
        PANEL_CLIENT_DATA,  mode,
        NULL);
    row = 4;
    col = 15;
    xv_create(panel, PANEL_BUTTON,
        XV_X,               xv_col(panel, col),
        XV_Y,               xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
        PANEL_LABEL_STRING, "Cancel",
        PANEL_NOTIFY_PROC,  rotation_proc,
        PANEL_CLIENT_DATA,  4,
        NULL);
}

/*
 * This routine creates a panel for the translations.
 */

static void
create_shift_subpanel(panel)
Panel panel;
{
    Pint    row = 0, col = 0;

    row = 0;
    col = 0;
    xv_create(panel, PANEL_TEXT,
        PANEL_LABEL_X,      5,
        PANEL_LABEL_Y,      xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
        PANEL_LABEL_STRING, "Enter distance in X direction:",
        PANEL_NOTIFY_PROC,  shift_proc,
        PANEL_VALUE_DISPLAY_LENGTH, 6,
        PANEL_CLIENT_DATA,  1,
        NULL);
```

```
    row = 1;
    col = 0;
    xv_create(panel, PANEL_TEXT,
        PANEL_LABEL_X,      5,
        PANEL_LABEL_Y,      xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
        PANEL_LABEL_STRING, "Enter distance in Y direction:",
        PANEL_NOTIFY_PROC,  shift_proc,
        PANEL_VALUE_DISPLAY_LENGTH, 6,
        PANEL_CLIENT_DATA,  2,
        NULL);
    row = 2;
    col = 0;
    xv_create(panel, PANEL_TEXT,
        PANEL_LABEL_X,      5,
        PANEL_LABEL_Y,      xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
        PANEL_LABEL_STRING, "Enter distance in Z direction:",
        PANEL_NOTIFY_PROC,  shift_proc,
        PANEL_VALUE_DISPLAY_LENGTH, 6,
        PANEL_CLIENT_DATA,  3,
        NULL);
    row = 4;
    col = 0;
    xv_create(panel, PANEL_BUTTON,
        XV_X,               xv_col(panel, col),
        XV_Y,               xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
        PANEL_LABEL_STRING, "Translate",
        PANEL_NOTIFY_PROC,  shift_proc,
        NULL);
    row = 4;
    col = 15;
    xv_create(panel, PANEL_BUTTON,
        XV_X,               xv_col(panel, col),
        XV_Y,               xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
        PANEL_LABEL_STRING, "Cancel",
        PANEL_NOTIFY_PROC,  shift_proc,
        PANEL_CLIENT_DATA,  4,
        NULL);
}

/*
 * This routine creates a panel for scaling.
 */

static void
create_scale_subpanel(panel)
Panel panel;
{
    Pint      row = 0, col = 0;

    row = 0;
    col = 0;
    xv_create(panel, PANEL_TEXT,
        PANEL_LABEL_X,      5,
```

```
        PANEL_LABEL_Y,      xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
        PANEL_LABEL_STRING, "Enter scale factor in X direction:",
        PANEL_NOTIFY_PROC,  scale_proc,
        PANEL_VALUE_DISPLAY_LENGTH, 6,
        PANEL_CLIENT_DATA,  1,
        NULL);
    row = 1;
    col = 0;
    xv_create(panel, PANEL_TEXT,
        PANEL_LABEL_X,      5,
        PANEL_LABEL_Y,      xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
        PANEL_LABEL_STRING, "Enter scale factor in Y direction:",
        PANEL_NOTIFY_PROC,  scale_proc,
        PANEL_VALUE_DISPLAY_LENGTH, 6,
        PANEL_CLIENT_DATA,  2,
        NULL);
    row = 2;
    col = 0;
    xv_create(panel, PANEL_TEXT,
        PANEL_LABEL_X,      5,
        PANEL_LABEL_Y,      xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
        PANEL_LABEL_STRING, "Enter scale factor in Z direction:",
        PANEL_NOTIFY_PROC,  scale_proc,
        PANEL_VALUE_DISPLAY_LENGTH, 6,
        PANEL_CLIENT_DATA,  3,
        NULL);
    row = 4;
    col = 0;
    xv_create(panel, PANEL_BUTTON,
        XV_X,               xv_col(panel, col),
        XV_Y,               xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
        PANEL_LABEL_STRING, "Scale",
        PANEL_NOTIFY_PROC,  scale_proc,
        PANEL_CLIENT_DATA,  4,
        NULL);
    row = 4;
    col = 15;
    xv_create(panel, PANEL_BUTTON,
        XV_X,               xv_col(panel, col),
        XV_Y,               xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
        PANEL_LABEL_STRING, "Cancel",
        PANEL_NOTIFY_PROC,  scale_proc,
        PANEL_CLIENT_DATA,  5,
        NULL);
}

/*
 * This routine creates a panel for circles.
 */

static void
create_circle_subpanel(panel)
```

```
Panel panel;
{
    Pint      row = 0, col = 0;

    row = 0;
    col = 0;
    xv_create(panel, PANEL_TEXT,
        PANEL_LABEL_X,      xv_col(panel, col),
        PANEL_LABEL_Y,      xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + YELLOW,
        PANEL_LABEL_STRING, "Center.x:",
        PANEL_NOTIFY_PROC,  circle_proc,
        PANEL_VALUE_DISPLAY_LENGTH, 6,
        PANEL_CLIENT_DATA,  1,
        NULL);
    row = 1;
    col = 0;
    xv_create(panel, PANEL_TEXT,
        PANEL_LABEL_X,      xv_col(panel, col),
        PANEL_LABEL_Y,      xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + YELLOW,
        PANEL_LABEL_STRING, "Center.y:",
        PANEL_NOTIFY_PROC,  circle_proc,
        PANEL_VALUE_DISPLAY_LENGTH, 6,
        PANEL_CLIENT_DATA,  2,
        NULL);
    row = 2;
    col = 0;
    xv_create(panel, PANEL_TEXT,
        PANEL_LABEL_X,      xv_col(panel, col),
        PANEL_LABEL_Y,      xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + YELLOW,
        PANEL_LABEL_STRING, "Center.z:",
        PANEL_NOTIFY_PROC,  circle_proc,
        PANEL_VALUE_DISPLAY_LENGTH, 6,
        PANEL_CLIENT_DATA,  3,
        NULL);
    row = 3;
    col = 0;
    xv_create(panel, PANEL_TEXT,
        PANEL_LABEL_X,      xv_col(panel, col),
        PANEL_LABEL_Y,      xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + YELLOW,
        PANEL_LABEL_STRING, "Dir_vec1.x:",
        PANEL_NOTIFY_PROC,  circle_proc,
        PANEL_VALUE_DISPLAY_LENGTH, 6,
        PANEL_CLIENT_DATA,  4,
        NULL);
    row = 4;
    col = 0;
    xv_create(panel, PANEL_TEXT,
        PANEL_LABEL_X,      xv_col(panel, col),
        PANEL_LABEL_Y,      xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + YELLOW,
        PANEL_LABEL_STRING, "Dir_vec1.y:",
        PANEL_NOTIFY_PROC,  circle_proc,
        PANEL_VALUE_DISPLAY_LENGTH, 6,
```

```
    PANEL_CLIENT_DATA, 5,
    NULL);
row = 5;
col = 0;
xv_create(panel, PANEL_TEXT,
    PANEL_LABEL_X,      xv_col(panel, col),
    PANEL_LABEL_Y,      xv_row(panel, row),
    PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + YELLOW,
    PANEL_LABEL_STRING, "Dir_vec1.z:",
    PANEL_NOTIFY_PROC,  circle_proc,
    PANEL_VALUE_DISPLAY_LENGTH, 6,
    PANEL_CLIENT_DATA,  6,
    NULL);
row = 3;
col = 18;
xv_create(panel, PANEL_TEXT,
    PANEL_LABEL_X,      xv_col(panel, col),
    PANEL_LABEL_Y,      xv_row(panel, row),
    PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + YELLOW,
    PANEL_LABEL_STRING, "Dir_vec2.x:",
    PANEL_NOTIFY_PROC,  circle_proc,
    PANEL_VALUE_DISPLAY_LENGTH, 6,
    PANEL_CLIENT_DATA,  7,
    NULL);
row = 4;
col = 18;
xv_create(panel, PANEL_TEXT,
    PANEL_LABEL_X,      xv_col(panel, col),
    PANEL_LABEL_Y,      xv_row(panel, row),
    PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + YELLOW,
    PANEL_LABEL_STRING, "Dir_vec2.y:",
    PANEL_NOTIFY_PROC,  circle_proc,
    PANEL_VALUE_DISPLAY_LENGTH, 6,
    PANEL_CLIENT_DATA,  8,
    NULL);
row = 5;
col = 18;
xv_create(panel, PANEL_TEXT,
    PANEL_LABEL_X,      xv_col(panel, col),
    PANEL_LABEL_Y,      xv_row(panel, row),
    PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + YELLOW,
    PANEL_LABEL_STRING, "Dir_vec2.z:",
    PANEL_NOTIFY_PROC,  circle_proc,
    PANEL_VALUE_DISPLAY_LENGTH, 6,
    PANEL_CLIENT_DATA,  9,
    NULL);
row = 0;
col = 18;
xv_create(panel, PANEL_TEXT,
    PANEL_LABEL_X,      xv_col(panel, col),
    PANEL_LABEL_Y,      xv_row(panel, row),
    PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + YELLOW,
    PANEL_LABEL_STRING, "Radius:",
    PANEL_NOTIFY_PROC,  circle_proc,
    PANEL_VALUE_DISPLAY_LENGTH, 6,
    PANEL_CLIENT_DATA,  10,
    NULL);
```

```
    row = 7;
    col = 0;
    xv_create(panel, PANEL_BUTTON,
        XV_X,              xv_col(panel, col),
        XV_Y,              xv_row(panel, row),
        PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + YELLOW,
        PANEL_LABEL_STRING, "Draw circle",
        PANEL_NOTIFY_PROC, circle_proc,
        PANEL_CLIENT_DATA, 11,
        NULL);
    row = 7;
    col = 18;
    xv_create(panel, PANEL_BUTTON,
        XV_X,              xv_col(panel, col),
        XV_Y,              xv_row(panel, row),
        PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + YELLOW,
        PANEL_LABEL_STRING, "Cancel",
        PANEL_NOTIFY_PROC, circle_proc,
        PANEL_CLIENT_DATA, 12,
        NULL);
}

/*
 * This routine creates a panel for saving and loading the structure to/from
 * a file.
 */

static void
create_file_menu(panel, frame, cms)
Panel panel;
Frame frame;
Cms   cms;
{
    Frame           file_frame;
    Panel           file_panel;
    Menu            file_menu;
    Pint            row = 0, col = 0;

    file_frame = (Frame)xv_create(frame, FRAME_CMD,
        XV_X,          0,
        XV_Y,          30,
        XV_WIDTH,      290,
        XV_HEIGHT,     100,
        FRAME_LABEL,   "File",
        NULL);
    file_panel = (Panel)xv_get(file_frame, FRAME_CMD_PANEL);
    xv_set(file_panel, WIN_CMS, cms, NULL);
    row = 1;
    xv_create(file_panel, PANEL_TEXT,
        PANEL_LABEL_X,     5,
        PANEL_LABEL_Y,     xv_row(panel, row),
        PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + CYAN,
        PANEL_LABEL_STRING, "Enter file name:",
        PANEL_NOTIFY_PROC, file_menu_proc,
        PANEL_VALUE_DISPLAY_LENGTH, 20,
        PANEL_CLIENT_DATA, 1,
        NULL);
```

```
    row = 2;
    col = 0;
    xv_create(file_panel, PANEL_BUTTON,
        XV_X,              xv_col(panel, col),
        XV_Y,              xv_row(panel, row),
        PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + CYAN,
        PANEL_LABEL_STRING, "Apply",
        PANEL_NOTIFY_PROC, file_menu_proc,
        PANEL_CLIENT_DATA, 2,
        NULL);
    row = 2;
    col = 15;
    xv_create(file_panel, PANEL_BUTTON,
        XV_X,              xv_col(panel, col),
        XV_Y,              xv_row(panel, row),
        PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + CYAN,
        PANEL_LABEL_STRING, "Cancel",
        PANEL_NOTIFY_PROC, file_menu_proc,
        PANEL_CLIENT_DATA, 3,
        NULL);
    file_menu = (Menu)xv_create(NULL, MENU,
        MENU_NOTIFY_PROC, file_frame_notify_proc,
        MENU_STRINGS,     "Load", "Save", NULL,
        MENU_CLIENT_DATA, file_frame,
        NULL);
    row = 0;
    col = 105;
    xv_create (panel, PANEL_BUTTON,
        XV_X,              xv_col(panel, col),
        XV_Y,              xv_row(panel, row),
        PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + YELLOW,
        PANEL_LABEL_STRING, "File",
        XV_HELP_DATA,      "gas:file",
        PANEL_ITEM_MENU,   file_menu,
        NULL);
}

/*
 * This routine creates a Tool menu.
 */

static void
create_tool_menu(panel)
Panel panel;
{
    Menu            tool_menu;
    Pint            row = 0, col = 0;

    tool_menu = (Menu)xv_create(NULL, MENU,
        MENU_NOTIFY_PROC,  tool_menu_proc,
        MENU_STRINGS,      "Refresh", "Delete Drawing",
                           "Undisplay Drawing", "Display Drawing",
                           NULL,
        NULL);
    col = 115;
    xv_create(panel, PANEL_BUTTON,
        XV_X,              xv_col(panel, col),
```

```
        XV_Y,           xv_row(panel, row),
        PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + YELLOW,
        PANEL_LABEL_STRING, "Tool",
        XV_HELP_DATA,      "gas:tool",
        PANEL_ITEM_MENU,   tool_menu,
        NULL);
}

/*
 * This routine creates a help frame with a help message which appears
 * when the Help button is pressed.
 */

static void
create_help_menu(panel, frame, cms)
Panel panel;
Frame frame;
Cms   cms;
{
    Frame           text_frame;
    Panel           text_panel;
    Textsw          textsw;
    Pint            row = 0, col = 0;

    text_frame = (Frame)xv_create(frame, FRAME_CMD,
        XV_X,           0,
        XV_Y,           0,
        XV_WIDTH,       580,
        XV_HEIGHT,      200,
        FRAME_LABEL,    "Help",
        NULL);
    text_panel = (Panel)xv_get(text_frame, FRAME_CMD_PANEL);
    xv_set(text_panel, WIN_CMS, cms, NULL);

    textsw = (Textsw)xv_create(text_frame, TEXTSW,
            WIN_X,          0,
            WIN_Y,          0,
            WIN_ROWS,       10,
            WIN_COLUMNS,    80,
            NULL);
    row = 5;
    col = 0;
    xv_create(text_panel, PANEL_BUTTON,
        XV_X,           xv_col(text_panel, col),
        XV_Y,           xv_row(text_panel, row),
        PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + CYAN,
        PANEL_LABEL_STRING, "Quit",
        PANEL_NOTIFY_PROC, exit_help,
        PANEL_CLIENT_DATA, textsw,
        NULL);

    row = 0;
    col = 125;
    xv_create(panel, PANEL_BUTTON,
        XV_X,           xv_col(panel, col),
        XV_Y,           xv_row(panel, row),
        PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + YELLOW,
```

```
        PANEL_LABEL_STRING, "Help",
        XV_HELP_DATA,       "gas:help",
        PANEL_NOTIFY_PROC,  show_help,
        PANEL_CLIENT_DATA,  textsw,
        NULL);
}

/*
 * This routine creates two buttons for deletion of the element or canceling
 * of the delete  action.
 */

static void
create_delete_menu(panel)
Panel panel;
{
    Pint            row = 0, col = 0;

    row = 19;
    col = 8;
    xv_create(panel, PANEL_BUTTON,
        XV_X,               xv_col(panel, col),
        XV_Y,               xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + WHITE,
        PANEL_LABEL_STRING, "Delete",
        XV_HELP_DATA,       "gas:delete",
        PANEL_NOTIFY_PROC,  delete_proc,
        NULL);
    row = 19;
    col = 16;
    xv_create(panel, PANEL_BUTTON,
        XV_X,               xv_col(panel, col),
        XV_Y,               xv_row(panel, row),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + WHITE,
        PANEL_LABEL_STRING, " Cancel delete action  ",
        XV_HELP_DATA,       "gas:canceldel",
        PANEL_NOTIFY_PROC,  cancel_delete_proc,
        NULL);
}

/*
 * This routine creates controls used in the drawing functions.
 */

static void
create_drawing_menu(panel)
Panel panel;
{
    Pint            row = 0, col = 0;

    row = 6;
    col = 0;
    xv_create( panel, PANEL_CYCLE,
        XV_Y,   xv_row(panel, row),
        XV_X,   xv_col(panel, col),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + SPRING_GREEN,
        PANEL_LABEL_STRING, "Current direction:",
```

```
    PANEL_CHOICE_STRINGS, "       X       ",
                    "      -X       ",
                    "       Y       ",
                    "      -Y       ",
                    "       Z       ",
                    "      -Z       ",
                    NULL,
    XV_HELP_DATA,       "gas:curdir",
    PANEL_NOTIFY_PROC, drawing_proc,
    PANEL_CLIENT_DATA, 1,
    NULL);
row = 13;
col = 0;
xv_create( panel, PANEL_CHOICE,
    XV_Y,  xv_row(panel, row),
    XV_X,  xv_col(panel, col),
    PANEL_LABEL_STRING, "",
    PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + SPRING_GREEN,
    PANEL_CHOICE_STRINGS, "DIR ON ",
                    " DIR OFF ",
                    NULL,
    XV_HELP_DATA,       "gas:dir",
    PANEL_LAYOUT,     PANEL_HORIZONTAL,
    PANEL_NOTIFY_PROC, drawing_proc,
    PANEL_VALUE,      1,
    PANEL_CLIENT_DATA, 2,
    NULL);
row = 15;
col = 8;
xv_create( panel, PANEL_CHOICE,
    XV_Y,  xv_row(panel, row),
    XV_X,  xv_col(panel, col),
    PANEL_LABEL_STRING, "",
    PANEL_CHOICE_STRINGS, "DRAW PTS",
                    "DRAW OBJS",
                    "  OFF  ",
                    NULL,
    XV_HELP_DATA,       "gas:drawpo",
    PANEL_LAYOUT,     PANEL_HORIZONTAL,
    PANEL_VALUE,      2,
    PANEL_NOTIFY_PROC, drawing_proc,
    PANEL_CLIENT_DATA, 3,
    NULL);
row = 16;
col = 8;
xv_create( panel, PANEL_CHOICE,
    XV_Y,  xv_row(panel, row),
    XV_X,  xv_col(panel, col),
    PANEL_LABEL_STRING, "",
    PANEL_CHOICE_STRINGS, "DRAW NUP",
                    " DRAW SPP",
                    "  OFF  ",
                    NULL,
    XV_HELP_DATA,       "gas:drawns",
    PANEL_LAYOUT,     PANEL_HORIZONTAL,
    PANEL_VALUE,      2,
    PANEL_NOTIFY_PROC, drawing_proc,
```

```
    PANEL_CLIENT_DATA, 14,
    NULL);
row = 17;
col = 8;
xv_create( panel, PANEL_CHOICE,
    XV_Y,  xv_row(panel, row),
    XV_X,  xv_col(panel, col),
    PANEL_LABEL_STRING, "",
    PANEL_CHOICE_STRINGS, " INTERSECT ON",
                " INTERSECT OFF",
                NULL,
    XV_HELP_DATA,       "gas:intersect",
    PANEL_LAYOUT,     PANEL_HORIZONTAL,
    PANEL_VALUE,      1,
    PANEL_NOTIFY_PROC, drawing_proc,
    PANEL_CLIENT_DATA, 7,
    NULL);
row = 18;
col = 8;
xv_create( panel, PANEL_CHOICE,
    XV_Y,  xv_row(panel, row),
    XV_X,  xv_col(panel, col),
    PANEL_LABEL_STRING, "",
    PANEL_CHOICE_STRINGS, "  JOIN PTS  ON ",
                " JOIN PTS  OFF ",
                NULL,
    XV_HELP_DATA,       "gas:join",
    PANEL_LAYOUT,     PANEL_HORIZONTAL,
    PANEL_VALUE,      1,
    PANEL_NOTIFY_PROC, drawing_proc,
    PANEL_CLIENT_DATA, 8,
    NULL);
row = 7;
col = 0;
xv_create(panel, PANEL_TEXT,
    PANEL_LABEL_X,      xv_col(panel, col),
    PANEL_LABEL_Y,      xv_row(panel, row),
    PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + SPRING_GREEN,
    PANEL_LABEL_STRING, "No. Elements:",
    XV_HELP_DATA,       "gas:noelem",
    PANEL_NOTIFY_PROC,  drawing_proc,
    PANEL_VALUE_DISPLAY_LENGTH, 5,
    PANEL_VALUE,        "",
    PANEL_CLIENT_DATA,  4,
    NULL);
row = 7;
col = 22;
xv_create(panel, PANEL_TEXT,
    PANEL_LABEL_X,      xv_col(panel, col),
    PANEL_LABEL_Y,      xv_row(panel, row),
    PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + SPRING_GREEN,
    PANEL_LABEL_STRING, "Space:",
    XV_HELP_DATA,       "gas:space",
    PANEL_NOTIFY_PROC,  drawing_proc,
    PANEL_VALUE_DISPLAY_LENGTH, 6,
    PANEL_VALUE,        "",
    PANEL_CLIENT_DATA,  5,
```

```
        NULL);
    row = 13;
    col = 18;
    xv_create( panel, PANEL_CHOICE,
        XV_Y,   xv_row(panel, row),
        XV_X,   xv_col(panel, col),
        PANEL_LABEL_STRING, "",
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + SPRING_GREEN,
        PANEL_CHOICE_STRINGS, "  JP ON  ",
                        "  JP OFF  ",
                        NULL,
        XV_HELP_DATA,       "gas:jp",
        PANEL_LAYOUT,     PANEL_HORIZONTAL,
        PANEL_VALUE,      1,
        PANEL_NOTIFY_PROC, drawing_proc,
        PANEL_CLIENT_DATA, 6,
        NULL);
    row = 5;
    col = 0;
    xv_create( panel, PANEL_CYCLE,
        XV_Y,   xv_row(panel, row),
        XV_X,   xv_col(panel, col),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + SPRING_GREEN,
        PANEL_LABEL_STRING, "Range of axis:",
        PANEL_CHOICE_STRINGS, "    0 - 1    ",
                        "    0 - 5    ",
                        "    0 - 10   ",
                        "    0 - 50   ",
                        "   0 - 100   ",
                        "   0 - 500   ",
                        "   0 - 1000  ",
                        NULL,
        XV_HELP_DATA,       "gas:range",
        PANEL_VALUE,      4,
        PANEL_NOTIFY_PROC, drawing_proc,
        PANEL_CLIENT_DATA, 0,
        NULL);
}

/*
 * This function creates controls for changing the viewing representation.
 */

static void
create_view_menu(panel)
Panel   panel;
{
    Pint    row, col;

    row = 21;
    col = 0;
    xv_create(panel, PANEL_SLIDER,
        XV_Y,   xv_row(panel, row),
        XV_X,   xv_col(panel, col),
        PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
        PANEL_LABEL_STRING, "view x:  ",
        XV_HELP_DATA,       "gas:view",
```

```
    PANEL_MIN_VALUE,   -200,
    PANEL_MAX_VALUE,   200,
    PANEL_VALUE,       100,
    PANEL_NOTIFY_PROC, view_proc,
    PANEL_CLIENT_DATA, 1,
    NULL);
row = 22;
col = 0;
xv_create(panel, PANEL_SLIDER,
    XV_Y,   xv_row(panel, row),
    XV_X,   xv_col(panel, col),
    PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
    PANEL_LABEL_STRING, "view y:  ",
    XV_HELP_DATA,       "gas:view",
    PANEL_MIN_VALUE,   -200,
    PANEL_MAX_VALUE,   200,
    PANEL_VALUE,       100,
    PANEL_NOTIFY_PROC, view_proc,
    PANEL_CLIENT_DATA, 2,
    NULL);
row = 23;
col = 0;
xv_create(panel, PANEL_SLIDER,
    XV_Y,   xv_row(panel, row),
    XV_X,   xv_col(panel, col),
    PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
    PANEL_LABEL_STRING, "view z:  ",
    XV_HELP_DATA,       "gas:view",
    PANEL_MIN_VALUE,   -200,
    PANEL_MAX_VALUE,   200,
    PANEL_VALUE,       100,
    PANEL_NOTIFY_PROC, view_proc,
    PANEL_CLIENT_DATA, 3,
    NULL);
row = 24;
col = 0;
xv_create(panel, PANEL_SLIDER,
    XV_Y,   xv_row(panel, row),
    XV_X,   xv_col(panel, col),
    PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS + CYAN,
    PANEL_LABEL_STRING, "zoom:  ",
    XV_HELP_DATA,       "gas:zoom",
    PANEL_MIN_VALUE,   -600,
    PANEL_MAX_VALUE,   1200,
    PANEL_VALUE,       504,
    PANEL_NOTIFY_PROC, view_proc,
    PANEL_CLIENT_DATA, 4,
    NULL);
row = 2;
col = 0;
xv_create( panel, PANEL_CHOICE,
    XV_Y,   xv_row(panel, row),
    XV_X,   xv_col(panel, col),
    PANEL_LABEL_STRING, "",
    PANEL_CHOICE_STRINGS, " PROJ. PERSPECTIVE ",
                          "   PROJ. PARALLEL   ",
                          NULL,
```

```
        XV_HELP_DATA,     "gas:projection",
        PANEL_LAYOUT,     PANEL_HORIZONTAL,
        PANEL_VALUE,      1,
        PANEL_NOTIFY_PROC, view_proc,
        PANEL_CLIENT_DATA, 5,
        NULL);
    row = 1;
    col = 0;
    xv_create( panel, PANEL_CHOICE,
        XV_Y,  xv_row(panel, row),
        XV_X,  xv_col(panel, col),
        PANEL_LABEL_STRING, "",
        PANEL_CHOICE_STRINGS, "       AXIS ON       ",
                        "       AXIS OFF       ",
                        NULL,
        XV_HELP_DATA,     "gas:axis",
        PANEL_LAYOUT,     PANEL_HORIZONTAL,
        PANEL_VALUE,      1,
        PANEL_NOTIFY_PROC, axis_notify_proc,
        NULL);
    row = 3;
    col = 0;
    xv_create( panel, PANEL_CHOICE,
        XV_Y,  xv_row(panel, row),
        XV_X,  xv_col(panel, col),
        PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + YELLOW,
        PANEL_LABEL_STRING, "",
        PANEL_CHOICE_STRINGS, "     COPY MODE      ",
                        "     INSERT MODE    ",
                        NULL,
        XV_HELP_DATA,       "gas:mode",
        PANEL_LAYOUT,     PANEL_HORIZONTAL,
        PANEL_NOTIFY_PROC, delete_copy_proc,
        PANEL_VALUE,      1,
        PANEL_CLIENT_DATA, 1,
        NULL);
}

/*
 * This routine creates main panel controls.
 */

static void
create_side_menu(panel)
Panel panel;
{
    Pint            row = 0, col = 0;

    row = 11;
    col = 0;
    xv_create( panel, PANEL_CHOICE,
        XV_Y,  xv_row(panel, row),
        XV_X,  xv_col(panel, col),
        PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + SPRING_GREEN,
        PANEL_LABEL_STRING, "",
        PANEL_CHOICE_STRINGS, "CPS ON",
                        "CPS OFF  ",
```

```
                    NULL,
        XV_HELP_DATA,       "gas:cps",
        PANEL_LAYOUT,     PANEL_HORIZONTAL,
        PANEL_VALUE,      1,
        PANEL_NOTIFY_PROC, cps_proc,
        NULL);
    row = 11;
    col = 18;
    xv_create( panel, PANEL_CHOICE,
        XV_Y,  xv_row(panel, row),
        XV_X,  xv_col(panel, col),
        PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + SPRING_GREEN,
        PANEL_LABEL_STRING, "",
        PANEL_CHOICE_STRINGS, "RPS ON ",
                    " RPS OFF  ",
                    NULL,
        XV_HELP_DATA,       "gas:rps",
        PANEL_LAYOUT,     PANEL_HORIZONTAL,
        PANEL_VALUE,      1,
        PANEL_NOTIFY_PROC, rps_proc,
        NULL);
    row = 12;
    col = 0;
    xv_create( panel, PANEL_CHOICE,
        XV_Y,  xv_row(panel, row),
        XV_X,  xv_col(panel, col),
        PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + SPRING_GREEN,
        PANEL_LABEL_STRING, "",
        PANEL_CHOICE_STRINGS, "CLS ON ",
                    "CLS OFF  ",
                    NULL,
        XV_HELP_DATA,       "gas:cls",
        PANEL_LAYOUT,     PANEL_HORIZONTAL,
        PANEL_VALUE,      1,
        PANEL_NOTIFY_PROC, cls_proc,
        NULL);
    row = 12;
    col = 18;
    xv_create( panel, PANEL_CHOICE,
        XV_Y,  xv_row(panel, row),
        XV_X,  xv_col(panel, col),
        PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + SPRING_GREEN,
        PANEL_LABEL_STRING, "",
        PANEL_CHOICE_STRINGS, " RLS ON ",
                    " RLS OFF  ",
                    NULL,
        XV_HELP_DATA,       "gas:rls",
        PANEL_LAYOUT,     PANEL_HORIZONTAL,
        PANEL_VALUE,      1,
        PANEL_NOTIFY_PROC, rls_proc,
        NULL);
    create_view_menu(panel);
}

/*
 * This function creates main panel controls and popup frames and their
 * panels.
```

```
*/

static Panel
create_panel(frame, cms)
Frame frame;
Cms   cms;
{
    Frame           point_frame, label_frame, rotate_x_frame,
                    rotate_y_frame, rotate_z_frame, shift_frame,
                    scale_frame, circle_frame, text_frame;
    Panel           panel, point_panel, label_panel, rotate_panel,
                    shift_panel, scale_panel, circle_panel,
                    text_panel;
    Pint            row = 0, col = 0, rot_mode;


    panel = xv_create(frame, PANEL,
        WIN_CMS,        cms,
        NULL);
    row = 30;
    col = 0;
    point_frame = (Frame)xv_create(frame, FRAME_CMD,
        XV_X,           col,
        XV_Y,           row,
        XV_WIDTH,       290,
        XV_HEIGHT,      180,
        FRAME_LABEL,    "Point",
        NULL);
    label_frame = (Frame)xv_create(frame, FRAME_CMD,
        XV_X,           col,
        XV_Y,           row,
        XV_WIDTH,       290,
        XV_HEIGHT,      180,
        FRAME_LABEL,    "Label",
        NULL);
    text_frame = (Frame)xv_create(frame, FRAME_CMD,
        XV_X,           col,
        XV_Y,           row,
        XV_WIDTH,       290,
        XV_HEIGHT,      100,
        FRAME_LABEL,    "Text",
        NULL);
    col = 0;
    row = 90;
    rotate_x_frame = (Frame)xv_create(frame, FRAME_CMD,
        XV_X,           col,
        XV_Y,           row,
        XV_WIDTH,       290,
        XV_HEIGHT,      180,
        FRAME_LABEL,    "X Rotation",
        NULL);
    rotate_y_frame = (Frame)xv_create(frame, FRAME_CMD,
        XV_X,           col,
        XV_Y,           row,
        XV_WIDTH,       290,
        XV_HEIGHT,      180,
        FRAME_LABEL,    "Y Rotation",
```

```
    NULL);
rotate_x_frame = (Frame)xv_create(frame, FRAME_CMD,
    XV_X,           col,
    XV_Y,           row,
    XV_WIDTH,       290,
    XV_HEIGHT,      180,
    FRAME_LABEL,    "Z Rotation",
    NULL);
shift_frame = (Frame)xv_create(frame, FRAME_CMD,
    XV_X,           col,
    XV_Y,           row,
    XV_WIDTH,       290,
    XV_HEIGHT,      180,
    FRAME_LABEL,    "Translation",
    NULL);
scale_frame = (Frame)xv_create(frame, FRAME_CMD,
    XV_X,           col,
    XV_Y,           row,
    XV_WIDTH,       290,
    XV_HEIGHT,      180,
    FRAME_LABEL,    "Scaling",
    NULL);
circle_frame = (Frame)xv_create(frame, FRAME_CMD,
    XV_X,           col,
    XV_Y,           row,
    XV_WIDTH,       290,
    XV_HEIGHT,      280,
    FRAME_LABEL,    "Circle",
    NULL);
point_panel = (Panel)xv_get(point_frame, FRAME_CMD_PANEL);
xv_set(point_panel, WIN_CMS, cms, NULL);
label_panel = (Panel)xv_get(label_frame, FRAME_CMD_PANEL);
xv_set(label_panel, WIN_CMS, cms, NULL);
text_panel = (Panel)xv_get(text_frame, FRAME_CMD_PANEL);
xv_set(text_panel, WIN_CMS, cms, NULL);
create_point_subpanel(point_panel);
create_label_subpanel(label_panel);
create_text_subpanel(text_panel);
rotate_panel = (Panel)xv_get(rotate_x_frame, FRAME_CMD_PANEL);
xv_set(rotate_panel, WIN_CMS, cms, NULL);
rot_mode = 1;
create_rotate_subpanel(rotate_panel, rot_mode);
rotate_panel = (Panel)xv_get(rotate_y_frame, FRAME_CMD_PANEL);
xv_set(rotate_panel, WIN_CMS, cms, NULL);
rot_mode = 2;
create_rotate_subpanel(rotate_panel, rot_mode);
rotate_panel = (Panel)xv_get(rotate_z_frame, FRAME_CMD_PANEL);
xv_set(rotate_panel, WIN_CMS, cms, NULL);
rot_mode = 3;
create_rotate_subpanel(rotate_panel, rot_mode);
shift_panel = (Panel)xv_get(shift_frame, FRAME_CMD_PANEL);
xv_set(shift_panel, WIN_CMS, cms, NULL);
scale_panel = (Panel)xv_get(scale_frame, FRAME_CMD_PANEL);
xv_set(scale_panel, WIN_CMS, cms, NULL);
create_shift_subpanel(shift_panel);
create_scale_subpanel(scale_panel);
circle_panel = (Panel)xv_get(circle_frame, FRAME_CMD_PANEL);
```

```
xv_set(circle_panel, WIN_CMS, cms, NULL);
create_circle_subpanel(circle_panel);
col = 0;
row = 0;
xv_create (panel, PANEL_BUTTON,
    XV_X,           xv_col(panel, col),
    XV_Y,           xv_row(panel, row),
    PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + CYAN,
    PANEL_LABEL_STRING, "Point",
    XV_HELP_DATA,      "gas:point",
    PANEL_NOTIFY_PROC, frame_notify_proc,
    PANEL_CLIENT_DATA, point_frame,
    NULL);
col = 10;
row = 0;
xv_create (panel, PANEL_BUTTON,
    XV_X,           xv_col(panel, col),
    XV_Y,           xv_row(panel, row),
    PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + CYAN,
    PANEL_LABEL_STRING, "Label",
    XV_HELP_DATA,      "gas:label",
    PANEL_NOTIFY_PROC, frame_notify_proc,
    PANEL_CLIENT_DATA, label_frame,
    NULL);
col = 20;
row = 0;
xv_create (panel, PANEL_BUTTON,
    XV_X,           xv_col(panel, col),
    XV_Y,           xv_row(panel, row),
    PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + CYAN,
    PANEL_LABEL_STRING, "Text",
    XV_HELP_DATA,      "gas:text",
    PANEL_NOTIFY_PROC, frame_notify_proc,
    PANEL_CLIENT_DATA, text_frame,
    NULL);
col = 0;
row = 15;
xv_create (panel, PANEL_BUTTON,
    XV_X,           xv_col(panel, col),
    XV_Y,           xv_row(panel, row),
    PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + CYAN,
    PANEL_LABEL_STRING, "Rot X",
    XV_HELP_DATA,      "gas:rot",
    PANEL_NOTIFY_PROC, frame_notify_proc,
    PANEL_CLIENT_DATA, rotate_x_frame,
    NULL);
col = 0;
row = 16;
xv_create (panel, PANEL_BUTTON,
    XV_X,           xv_col(panel, col),
    XV_Y,           xv_row(panel, row),
    PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + CYAN,
    PANEL_LABEL_STRING, "Rot Y",
    XV_HELP_DATA,      "gas:rot",
    PANEL_NOTIFY_PROC, frame_notify_proc,
    PANEL_CLIENT_DATA, rotate_y_frame,
    NULL);
```

```
col = 0;
row = 17;
xv_create (panel, PANEL_BUTTON,
    XV_X,           xv_col(panel, col),
    XV_Y,           xv_row(panel, row),
    PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + CYAN,
    PANEL_LABEL_STRING, "Rot Z",
    XV_HELP_DATA,     "gas:rot",
    PANEL_NOTIFY_PROC, frame_notify_proc,
    PANEL_CLIENT_DATA, rotate_z_frame,
    NULL);
col = 0;
row = 18;
xv_create (panel, PANEL_BUTTON,
    XV_X,           xv_col(panel, col),
    XV_Y,           xv_row(panel, row),
    PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + CYAN,
    PANEL_LABEL_STRING, "Shift",
    XV_HELP_DATA,     "gas:shift",
    PANEL_NOTIFY_PROC, frame_notify_proc,
    PANEL_CLIENT_DATA, shift_frame,
    NULL);
col = 0;
row = 19;
xv_create (panel, PANEL_BUTTON,
    XV_X,           xv_col(panel, col),
    XV_Y,           xv_row(panel, row),
    PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + CYAN,
    PANEL_LABEL_STRING, "Scale",
    XV_HELP_DATA,     "gas:scale",
    PANEL_NOTIFY_PROC, frame_notify_proc,
    PANEL_CLIENT_DATA, scale_frame,
    NULL);
col = 40;
row = 0;
xv_create (panel, PANEL_BUTTON,
    XV_X,           xv_col(panel, col),
    XV_Y,           xv_row(panel, row),
    PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + CYAN,
    PANEL_LABEL_STRING, "Arc-circle",
    XV_HELP_DATA,     "gas:arc-circle",
    PANEL_NOTIFY_PROC, arc_circle_proc,
    PANEL_CLIENT_DATA, 2,
    NULL);
row = 0;
col = 53;
xv_create( panel, PANEL_CHOICE,
    XV_Y,  xv_row(panel, row),
    XV_X,  xv_col(panel, col),
    PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + CYAN,
    PANEL_LABEL_STRING, "",
    PANEL_CHOICE_STRINGS, "  INNER ARC ",
                  "  OUTER ARC   ",
                  NULL,
    PANEL_LAYOUT,     PANEL_HORIZONTAL,
    PANEL_VALUE,      0,
    PANEL_NOTIFY_PROC, arc_circle_proc,
```

```
        PANEL_CLIENT_DATA, 1,
        NULL);
    col = 30;
    row = 0;
    xv_create (panel, PANEL_BUTTON,
        XV_X,           xv_col(panel, col),
        XV_Y,           xv_row(panel, row),
        PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + CYAN,
        PANEL_LABEL_STRING, "Circle",
        XV_HELP_DATA,      "gas:circle",
        PANEL_NOTIFY_PROC,  frame_notify_proc,
        PANEL_CLIENT_DATA,  circle_frame,
        NULL);
    create_file_menu(panel, frame, cms);
    create_tool_menu(panel);
    create_help_menu(panel, frame, cms);
    row = 0;
    col = 135;
    xv_create(panel, PANEL_BUTTON,
        XV_X,           xv_col(panel, col),
        XV_Y,           xv_row(panel, row),
        PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + YELLOW,
        PANEL_LABEL_STRING, "Exit",
        XV_HELP_DATA,      "gas:quit",
        PANEL_NOTIFY_PROC,  exit_menu_proc,
        PANEL_CLIENT_DATA,  frame,
        NULL);
    row = 9;
    col = 0;
    xv_create( panel, PANEL_CHOICE,
        XV_Y,  xv_row(panel, row),
        XV_X,  xv_col(panel, col),
        PANEL_ITEM_COLOR,  CMS_CONTROL_COLORS + WHITE,
        PANEL_LABEL_STRING, "",
        PANEL_CHOICE_STRINGS, "  INSERT COPY ",
                    "  DELETE COPY ",
                    "   OFF    ",
                    NULL,
        XV_HELP_DATA,       "gas:insdelcopy",
        PANEL_LAYOUT,      PANEL_HORIZONTAL,
        PANEL_NOTIFY_PROC, delete_copy_proc,
        PANEL_VALUE,       2,
        PANEL_CLIENT_DATA, 2,
        NULL);
    create_drawing_menu(panel);
    create_side_menu(panel);
    create_delete_menu(panel);

    window_fit(panel);
    return panel;
}

/*
 * Callback routine for displaying and undisplaying a set of axis.
 */

static void
```

```
axis_notify_proc(item, val)
Panel_item item;
int        val;
{
    switch (val) {
        case 0: ppost_struct(WS, AXIS, .0);
                break;
        case 1: punpost_struct(WS, AXIS);
                break;
        default: break;
    }
}

/*
 * Notify procedure for displaying the popup subframe.
 */

static void
frame_notify_proc(item, event)
Frame item;
Event *event;
{
    xv_set((Frame)xv_get(item, PANEL_CLIENT_DATA), XV_SHOW, TRUE, NULL);
}

/*
 * Notify procedure for displaying the File menu and setting the SAVE and
 * LOAD global variables.
 */

static void
file_frame_notify_proc(menu, menu_item)
Menu menu;
Menu_item menu_item;
{
    xv_set((Frame)xv_get(menu, MENU_CLIENT_DATA),
          XV_SHOW, TRUE,
          NULL);
    if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "Save"))
      SAVE = TRUE;
    else if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "Load"))
      LOAD = TRUE;
}

/*
 * Callback routine for loading and archiving the structure from/to a file.
 */

static void
file_menu_proc(item, val)
Panel_item item;
Pint       val;
{
    static char       str[20];
    Pint_list         str_list;
    Pint              strct[2];
    Pescape_in_data   esc_in;
```

```
    Pescape_out_data  *esc_out;
    Pstore            store;


    switch((int)xv_get(item, PANEL_CLIENT_DATA)){
       case 1: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
               xv_set(item, PANEL_VALUE, "", NULL);
               break;
       case 2: pset_conf_res(PRES_UPD, PRES_UPD);
               esc_in.escape_in_u15.ar_mode = PHIGS_AR_PEX;
               pescape(PUESC_SET_ARCHIVE_TYPE, &esc_in, store, &esc_out);
               popen_ar_file(1, str);
               strct[0] = OBJECT;
               strct[1] = TEXT_OBJECT;
               str_list.num_ints = 2;
               str_list.ints = strct;
               if (SAVE == TRUE)
               {
                  par_structs(1, &str_list);
                  SAVE = FALSE;
               }
               else if (LOAD == TRUE)
               {
                  pret_structs(1, &str_list);
                  LOAD = FALSE;
               }
               pclose_ar_file(1);
               break;
       case 3: break;
    }
}

/*
 * Routine to compare two integers. Used in delete_proc in qsort.
 */

static int intcompare(i, j)
int *i, *j;
{
    return(*j - *i);
}

/*
 * This routine deletes the picked elements. Because the deletion changes
 * the order of the remaining elements, the picked elements are sorted first.
 */

static void
delete_proc(item, val)
Panel_item   item;
int          val;
{
    Pint       i;

    qsort(elem_pos_list.ints, elem_pos_list.num_ints, sizeof(int),
         intcompare);
    for (i = 0; i < elem_pos_list.num_ints; i++)
```

```
    {
        pset_elem_ptr(elem_pos_list.ints[i]);
        pdel_elem();
        pdel_elem();
    }
    pset_elem_ptr(MAX_INT);
    elem_pos_list.num_ints = 0;
    NELEM = 0;
    NP = 0;
    NL = 0;
}

/*
 * This routine cancels delete action by initializing the variables used.
 */

static void
cancel_delete_proc(item, val)
Panel_item   item;
int          val;
{
    elem_pos_list.num_ints = 0;
    NELEM = 0;
    NP = 0;
    NL = 0;
}

/*
 * Callback routine for the Tool menu.
 */

static void
tool_menu_proc(menu, menu_item)
Menu menu;
Menu_item menu_item;
{
    Pint            err, elem_ptr;

    if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "Refresh"))
        predraw_all_structs(WS, PFLAG_ALWAYS);
    else if
    (!strcmp((char *)xv_get(menu_item, MENU_STRING),"Delete Drawing"))
    {
        if (MODE == INSERT)
        {
          pinq_elem_ptr(&err, &elem_ptr);
          if (elem_ptr != start_ptr - 1)
            pdel_elem_range(start_ptr, MAX_INT);
        }
    } else if
    (!strcmp((char *)xv_get(menu_item, MENU_STRING),"Undisplay Drawing"))
    {
        punpost_struct(WS, OBJECT);
        predraw_all_structs(WS, PFLAG_COND);
    } else if
    (!strcmp((char *)xv_get(menu_item, MENU_STRING),"Display Drawing"))
        ppost_struct(WS, OBJECT, 0.1);
```

```
}

/*
 * This routine sets the display variable for a help subframe to true.
 */

static void
show_help(item, val)
Panel_item item;
Pint       val;
{
    Textsw       textsw;

    textsw = (Textsw)xv_get(item, PANEL_CLIENT_DATA);
    xv_set(textsw, TEXTSW_CONTENTS, help_message,
        NULL);
    xv_set((Frame)xv_get(textsw, XV_OWNER), XV_SHOW, TRUE, NULL);
}

/*
 * This routine sets the display variable for a help subframe to false.
 */

static void
exit_help(item, val)
Panel_item item;
Pint       val;
{
    Textsw       textsw;

    textsw = (Textsw)xv_get(item, PANEL_CLIENT_DATA);
    xv_set((Frame)xv_get(textsw, XV_OWNER), XV_SHOW, FALSE, NULL);
    textsw_reset(textsw, 0, 0);
}

/*
 * This routine is invoked when the Exit button is pressed.
 * It exits the application.
 */

static void
exit_menu_proc(item, val)
Panel_item item;
Pint       val;
{
    Pstruct_st   struct_st;
    Frame        frame;

    pinq_struct_st(&struct_st);
    if (struct_st == PSTRUCT_ST_STOP)
        pclose_struct();
    frame = (Frame)xv_get(item, PANEL_CLIENT_DATA);
    pclose_ws(WS);
    pclose_ws(WS_2);
    pclose_phigs();
    xv_destroy_safe(frame);
```

```
}

/*
 * This routine implements fast drawing routines available in 3D GAS.
 * They include:
 * 1. drawing a set of points in a given direction separated by a
 *    specified displacement (case 3 in main switch);
 * 2. drawing a set of points at the intersection of two line sets
 *    (case 7 in main switch);
 * 3. joining points from the CPS with points from the RPS (case 9 in
 *    main switch);
 * 4. drawing a point at a specified location in 3D (case 13 in main switch);
 * 5. a. drawing a number of points between two given points defined by the
 *       CPS separated by a given displacement;
 *    b. drawing a given number of points between two given points defined
 *       by the CPS;
 *    (case 14 in main switch);
 */

static void
drawing_proc(item, val)
Panel_item item;
int        val;
{
    static Pint          dir;
    static Pint          DIR;
    static Pint          num_pts;
    static Pfloat        space;
    static Ppoint3       ppoint3 = {0.0, 0.0, 0.0};
    Ppoint3              *point;
    Ppoint3              line[2], m1, m2, n1, n2, point3;
    Ppoint_list3         point_list, *pt_list, *ln_list;
    Pint                 i, j, k, m, n, pt_num, dont_draw, err;
    Pint                 elem_ptr, cur_elem_ptr;
    char                 str[20];
    Pstruct_st           struct_st;
    Pstore               store;
    Pelem_type           type;
    size_t               size;
    Pelem_data           *gdp;
    Pfloat               x2_x1, y2_y1, z2_z1, dist, val1, val2, s,
                         s1, s2, s3;

    if (MODE == COPY)
      pset_edit_mode(PEDIT_INSERT);
    switch((int)xv_get(item, PANEL_CLIENT_DATA)){
       case 0: switch(val){
                  case 0: scale = 100.0; break;
                  case 1: scale = 20.0; break;
                  case 2: scale = 10.0; break;
                  case 3: scale = 2.0; break;
                  case 4: scale = 1.0; break;
                  case 5: scale = 0.2; break;
                  case 6: scale = 0.1; break;
               }
```

```
        break;
case 1: switch(val){
            case 0: dir = 0; break;
            case 1: dir = 1; break;
            case 2: dir = 2; break;
            case 3: dir = 3; break;
            case 4: dir = 4; break;
            case 5: dir = 5; break;
        }
        break;
case 2: switch(val){
            case 0: DIR = ON; break;
            case 1: DIR = OFF; break;
        }
        break;
case 3: xv_set(item, PANEL_VALUE, 2, NULL);
        if (val == 0 && DIR == OFF)
        {
        pt_num = CPS.num_points;
        m = pt_num;
        n = num_pts;
        k = 0;
        if (pt_num == 1)
        {
            n = num_pts + 1;
            k = 1;
            point = (Ppoint3 *)calloc(n, sizeof(Ppoint3));
        }
        for (i = 0; i < m; i++) {
          for (j = k; j < n; j++) {
            if (pt_num == 1)
            {
              if (j == k)
                  point[j-1] = CPS.points[j-1];
              point[j] = point[j-1];
              switch (dir) {
                  case 0: point[j].x = point[j].x + space; break;
                  case 1: point[j].x = point[j].x - space; break;
                  case 2: point[j].y = point[j].y + space; break;
                  case 3: point[j].y = point[j].y - space; break;
                  case 4: point[j].z = point[j].z + space; break;
                  case 5: point[j].z = point[j].z - space; break;
              }
              point_list.points = &point[j];
            }
            else if (pt_num > 1)
            {
              if (join && (j == 0))
              {
                  line[0] = CPS.points[i];
              }
              switch (dir) {
                  case 0: CPS.points[i].x = CPS.points[i].x + space;
                          break;
                  case 1: CPS.points[i].x = CPS.points[i].x - space;
                          break;
                  case 2: CPS.points[i].y = CPS.points[i].y + space;
```

```
                break;
            case 3: CPS.points[i].y = CPS.points[i].y - space;
                break;
            case 4: CPS.points[i].z = CPS.points[i].z + space;
                break;
            case 5: CPS.points[i].z = CPS.points[i].z - space;
                break;
        }
        point_list.points = &CPS.points[i];
      }
      point_list.num_points = 1;
      if (MODE == INSERT)
        pset_pick_id(POINT);
      else if (MODE == COPY)
        pset_pick_id(COPY_POINT);
      ppolymarker3(&point_list);
    } /* for */
    if (pt_num == 1)
    {
        if (join)
        {
            point_list.num_points = 2;
            line[0] = point[0];
            line[1] = point[num_pts];
            point_list.points = line;
            if (MODE == INSERT)
              pset_pick_id(LINE);
            else if (MODE == COPY)
              pset_pick_id(COPY_LINE);
            ppolyline3(&point_list);
        }
        CPS.num_points = num_pts + 1;
        free(CPS.points);
        CPS.points = (Ppoint3 *)calloc(n,
                    sizeof(Ppoint3));
        bcopy((char *)point, (char*)CPS.points,
            n * sizeof(Ppoint3));
        free(point);
        NP = 0;
        NL = 0;
        NELEM = 0;
        elem_pos_list.num_ints = 0;
    }
    else if (pt_num > 1 && join)
    {
        line[1] = CPS.points[i];
        point_list.num_points = 2;
        point_list.points = line;
        if (MODE == INSERT)
          pset_pick_id(LINE);
        else if (MODE == COPY)
          pset_pick_id(COPY_LINE);
        ppolyline3(&point_list);
    }
  } /* for */
  } /* if */
else if (val == 0 && DIR == ON)
```

```
{
 if (CPS.num_points >= 2)
 {
  free(point);
  point = (Ppoint3 *)calloc(num_pts + 2, sizeof(Ppoint3));
  point[0] = CPS.points[CPS.num_points - 2];
  point[1] = CPS.points[CPS.num_points - 1];
  x2_x1 = point[1].x - point[0].x;
  y2_y1 = point[1].y - point[0].y;
  z2_z1 = point[1].z - point[0].z;
  dist = sqrt(x2_x1*x2_x1 + y2_y1*y2_y1 + z2_z1*z2_z1);
  for (i = 0; i < num_pts; i++)
  {
   point[i+2].x = point[1].x + (i+1)*space*x2_x1/dist;
   point[i+2].y = point[1].y + (i+1)*space*y2_y1/dist;
   point[i+2].z = point[1].z + (i+1)*space*z2_z1/dist;

   point_list.points = &point[i+2];
   point_list.num_points = 1;
   if (MODE == INSERT)
    pset_pick_id(POINT);
   else if (MODE == COPY)
    pset_pick_id(COPY_POINT);
   ppolymarker3(&point_list);
  }
  if (join)
  {
   point_list.num_points = 2;
   line[0] = point[0];
   line[1] = point[num_pts+1];
   point_list.points = line;
   if (MODE == INSERT)
    pset_pick_id(POINT);
   else if (MODE == COPY)
    pset_pick_id(COPY_POINT);
   ppolyline3(&point_list);
  }
  CPS.num_points = num_pts + 2;
  free(CPS.points);
  CPS.points = (Ppoint3 *)calloc(num_pts + 2,
            sizeof(Ppoint3));
  bcopy((char *)point, (char*)CPS.points,
      (num_pts + 2) * sizeof(Ppoint3));
  free(point);
  NP = 0;
  NL = 0;
  NELEM = 0;
  elem_pos_list.num_ints = 0;
 }
}
else if (val == 1 && DIR == OFF)
{
 pinq_struct_st(&struct_st);
 if (struct_st == PSTRUCT_ST_STOP)
 {
    pcreate_store(&err, &store);
    pinq_elem_ptr(&err, &elem_ptr);
```

```
for (i = 0; i < num_pts; i++)
{
  pset_elem_ptr(0);
  do
  {
    poffset_elem_ptr(1);
    pinq_elem_ptr(&err, &cur_elem_ptr);
    pinq_cur_elem_type_size(&err, &type, &size);
    if (type == PELEM_POLYMARKER3)
    {
      pinq_cur_elem_content(store, &err, &pt_list);
      switch (dir) {
      case 0:
      pt_list->points[0].x = pt_list->points[0].x +
                        (i+1) * space; break;
      case 1:
      pt_list->points[0].x = pt_list->points[0].x -
                        (i+1) * space; break;
      case 2:
      pt_list->points[0].y = pt_list->points[0].y +
                        (i+1) * space; break;
      case 3:
      pt_list->points[0].y = pt_list->points[0].y -
                        (i+1) * space; break;
      case 4:
      pt_list->points[0].z = pt_list->points[0].z +
                        (i+1) * space; break;
      case 5:
      pt_list->points[0].z = pt_list->points[0].z -
                        (i+1) * space; break;
      }
      pset_elem_ptr(MAX_INT);
      if (MODE == INSERT)
        pset_pick_id(POINT);
      else if (MODE == COPY)
        pset_pick_id(COPY_POINT);
      ppolymarker3(pt_list);
    }
    else if (type == PELEM_POLYLINE3)
    {
      pinq_cur_elem_content(store, &err, &ln_list);
      switch (dir) {
      case 0:
      ln_list->points[0].x = ln_list->points[0].x +
                        (i+1) * space;
      ln_list->points[1].x = ln_list->points[1].x +
                        (i+1) * space; break;
      case 1:
      ln_list->points[0].x = ln_list->points[0].x -
                        (i+1) * space;
      ln_list->points[1].x = ln_list->points[1].x -
                        (i+1) * space; break;
      case 2:
      ln_list->points[0].y = ln_list->points[0].y +
                        (i+1) * space;
      ln_list->points[1].y = ln_list->points[1].y +
                        (i+1) * space; break;
```

```
  case 3:
  ln_list->points[0].y = ln_list->points[0].y -
                    (i+1) * space;
  ln_list->points[1].y = ln_list->points[1].y -
                    (i+1) * space; break;
  case 4:
  ln_list->points[0].z = ln_list->points[0].z +
                    (i+1) * space;
  ln_list->points[1].z = ln_list->points[1].z +
                    (i+1) * space; break;
  case 5:
  ln_list->points[0].z = ln_list->points[0].z -
                    (i+1) * space;
  ln_list->points[1].z = ln_list->points[1].z -
                    (i+1) * space; break;
  }
  pset_elem_ptr(MAX_INT);
  if (MODE == INSERT)
    pset_pick_id(POINT);
  else if (MODE == COPY)
    pset_pick_id(COPY_POINT);
  ppolyline3(ln_list);
}
else if (type == PELEM_GDP3)
{
  pinq_cur_elem_content(store, &err, &gdp);
  if (gdp->gdp3.id == -2)
  {
  switch (dir) {
  case 0:
  gdp->gdp3.data.gdp3_u2.center.x =
  gdp->gdp3.data.gdp3_u2.center.x +
  (i+1) * space; break;
  case 1:
  gdp->gdp3.data.gdp3_u2.center.x =
  gdp->gdp3.data.gdp3_u2.center.x -
  (i+1) * space; break;
  case 2:
  gdp->gdp3.data.gdp3_u2.center.y =
  gdp->gdp3.data.gdp3_u2.center.y +
  (i+1) * space; break;
  case 3:
  gdp->gdp3.data.gdp3_u2.center.y =
  gdp->gdp3.data.gdp3_u2.center.y -
  (i+1) * space; break;
  case 4:
  gdp->gdp3.data.gdp3_u2.center.z =
  gdp->gdp3.data.gdp3_u2.center.z +
  (i+1) * space; break;
  case 5:
  gdp->gdp3.data.gdp3_u2.center.z =
  gdp->gdp3.data.gdp3_u2.center.z -
  (i+1) * space; break;
  }
  pset_elem_ptr(MAX_INT);
  if (MODE == INSERT)
    pset_pick_id(CIRCLE);
```

```
            else if (MODE == COPY)
              pset_pick_id(COPY_CIRCLE);
            pgdp3(gdp->gdp3.point_list, gdp->gdp3.id,
                gdp->gdp3.data);
            }
            else if (gdp->gdp3.id == -3)
            {
            switch (dir) {
            case 0:
            gdp->gdp3.data.gdp3_u3.center.x =
            gdp->gdp3.data.gdp3_u3.center.x +
            (i+1) * space; break;
            case 1:
            gdp->gdp3.data.gdp3_u3.center.x =
            gdp->gdp3.data.gdp3_u3.center.x -
            (i+1) * space; break;
            case 2:
            gdp->gdp3.data.gdp3_u3.center.y =
            gdp->gdp3.data.gdp3_u3.center.y +
            (i+1) * space; break;
            case 3:
            gdp->gdp3.data.gdp3_u3.center.y =
            gdp->gdp3.data.gdp3_u3.center.y -
            (i+1) * space; break;
            case 4:
            gdp->gdp3.data.gdp3_u3.center.z =
            gdp->gdp3.data.gdp3_u3.center.z +
            (i+1) * space; break;
            case 5:
            gdp->gdp3.data.gdp3_u3.center.z =
            gdp->gdp3.data.gdp3_u3.center.z -
            (i+1) * space; break;
            }
            pset_elem_ptr(MAX_INT);
            if (MODE == INSERT)
              pset_pick_id(ARC_CIRCLE);
            else if (MODE == COPY)
              pset_pick_id(COPY_ARC_CIRCLE);
            pgdp3(gdp->gdp3.point_list, gdp->gdp3.id,
                gdp->gdp3.data);
            }
          }
          pset_elem_ptr(cur_elem_ptr);
        }while (cur_elem_ptr != elem_ptr);
      }
      pset_elem_ptr(MAX_INT);
      pdel_store(store);
    }
  }
  else if (val == 1 && DIR == ON)
  {
    pinq_struct_st(&struct_st);
    if (CPS.num_points >= 2 && struct_st == PSTRUCT_ST_STOP)
    {
      pcreate_store(&err, &store);
      x2_x1 = CPS.points[CPS.num_points - 1].x -
            CPS.points[CPS.num_points - 2].x;
```

```
y2_y1 = CPS.points[CPS.num_points - 1].y -
        CPS.points[CPS.num_points - 2].y;
z2_z1 = CPS.points[CPS.num_points - 1].z -
        CPS.points[CPS.num_points - 2].z;
dist = sqrt(x2_x1*x2_x1 + y2_y1*y2_y1 + z2_z1*z2_z1);
pinq_elem_ptr(&err, &elem_ptr);
for (i = 0; i < num_pts; i++)
{
  pset_elem_ptr(0);
  do
  {
    poffset_elem_ptr(1);
    pinq_elem_ptr(&err, &cur_elem_ptr);
    pinq_cur_elem_type_size(&err, &type, &size);
    if (type == PELEM_POLYMARKER3)
    {
      pinq_cur_elem_content(store, &err, &pt_list);
      pt_list->points[0].x = pt_list->points[0].x +
                             (i+1) * space*x2_x1/dist;
      pt_list->points[0].y = pt_list->points[0].y +
                             (i+1) * space*y2_y1/dist;
      pt_list->points[0].z = pt_list->points[0].z +
                             (i+1) * space*z2_z1/dist;
      pset_elem_ptr(MAX_INT);
      if (MODE == INSERT)
        pset_pick_id(POINT);
      else if (MODE == COPY)
        pset_pick_id(COPY_POINT);
      ppolymarker3(pt_list);
    }
    else if (type == PELEM_POLYLINE3)
    {
      pinq_cur_elem_content(store, &err, &ln_list);
      ln_list->points[0].x = ln_list->points[0].x +
                             (i+1) * space*x2_x1/dist;
      ln_list->points[1].x = ln_list->points[1].x +
                             (i+1) * space*x2_x1/dist;
      ln_list->points[0].y = ln_list->points[0].y +
                             (i+1) * space*y2_y1/dist;
      ln_list->points[1].y = ln_list->points[1].y +
                             (i+1) * space*y2_y1/dist;
      ln_list->points[0].z = ln_list->points[0].z +
                             (i+1) * space*z2_z1/dist;
      ln_list->points[1].z = ln_list->points[1].z +
                             (i+1) * space*z2_z1/dist;
      pset_elem_ptr(MAX_INT);
      pset_elem_ptr(MAX_INT);
      if (MODE == INSERT)
        pset_pick_id(LINE);
      else if (MODE == COPY)
        pset_pick_id(COPY_LINE);
      ppolyline3(ln_list);
    }
    else if (type == PELEM_GDP3)
    {
      pinq_cur_elem_content(store, &err, &gdp);
      if (gdp->gdp3.id == -2)
```

```
                {
                gdp->gdp3.data.gdp3_u2.center.x =
                gdp->gdp3.data.gdp3_u2.center.x +
                (i+1) * space*x2_x1/dist;
                gdp->gdp3.data.gdp3_u2.center.y =
                gdp->gdp3.data.gdp3_u2.center.y +
                (i+1) * space*y2_y1/dist;
                gdp->gdp3.data.gdp3_u2.center.z =
                gdp->gdp3.data.gdp3_u2.center.z +
                (i+1) * space*z2_z1/dist;
                pset_elem_ptr(MAX_INT);
                if (MODE == INSERT)
                  pset_pick_id(CIRCLE);
                else if (MODE == COPY)
                  pset_pick_id(COPY_CIRCLE);
                pgdp3(gdp->gdp3.point_list, gdp->gdp3.id,
                    gdp->gdp3.data);
                }
                else if (gdp->gdp3.id == -3)
                {
                gdp->gdp3.data.gdp3_u3.center.x =
                gdp->gdp3.data.gdp3_u3.center.x +
                (i+1) * space*x2_x1/dist;
                gdp->gdp3.data.gdp3_u3.center.y =
                gdp->gdp3.data.gdp3_u3.center.y +
                (i+1) * space*y2_y1/dist;
                gdp->gdp3.data.gdp3_u3.center.z =
                gdp->gdp3.data.gdp3_u3.center.z +
                (i+1) * space*z2_z1/dist;
                pset_elem_ptr(MAX_INT);
                if (MODE == INSERT)
                  pset_pick_id(ARC_CIRCLE);
                else if (MODE == COPY)
                  pset_pick_id(COPY_ARC_CIRCLE);
                pgdp3(gdp->gdp3.point_list, gdp->gdp3.id,
                    gdp->gdp3.data);
                }
              }
              pset_elem_ptr(cur_elem_ptr);
            }while (cur_elem_ptr != elem_ptr);
          }
          pset_elem_ptr(MAX_INT);
          pdel_store(store);
        }
      }
      break;
case 4: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
      num_pts = atoi(str);
      panel_advance_caret((Panel)xv_get(item,
                    PANEL_PARENT_PANEL));
      break;
case 5: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
      space = atof(str);
      space = space * scale;
      panel_backup_caret((Panel)xv_get(item, PANEL_PARENT_PANEL));
      break;
case 6: switch((int)xv_get(item, PANEL_VALUE)){
```

```
        case 0: join = 1; break;
        case 1: join = 0; break;
        }
        break;
case 7: xv_set(item, PANEL_VALUE, 1, NULL);
        if (CLS.num_points >= 2 && RLS.num_points >= 2)
        {
          if (CLS.num_points > RLS.num_points)
            pt_num = RLS.num_points;
          else
            pt_num = CLS.num_points;
          for (i = 0; i < pt_num; i = i + 2)
          {
            s = s1 = s2 = s3 = 0.0;
            dont_draw = 0;
            m1 = CLS.points[i];
            m2 = CLS.points[i+1];
            n1 = RLS.points[i];
            n2 = RLS.points[i+1];

            if (!(m1.x == 0.0 && m2.x == 0.0 && n1.x == 0.0 &&
                n2.x == 0.0) && !(m1.y == 0.0 && m2.y == 0.0 &&
                n1.y == 0.0 && n2.y == 0.0))
            {
              val1=(m2.x-m1.x)*(n1.y-m1.y)-(n1.x-m1.x)*(m2.y-m1.y);
              val2=(n2.x-n1.x)*(m2.y-m1.y)-(n2.y-n1.y)*(m2.x-m1.x);
              s1 = val1 / val2;
            }
            else if (!(m1.x == 0.0 && m2.x == 0.0 && n1.x == 0.0 &&
                n2.x == 0.0) && !(m1.z == 0.0 && m2.z == 0.0 &&
                n1.z == 0.0 && n2.z == 0.0))
            {
              val1=(m2.x-m1.x)*(n1.z-m1.z)-(n1.x-m1.x)*(m2.z-m1.z);
              val2=(n2.x-n1.x)*(m2.z-m1.z)-(n2.z-n1.z)*(m2.x-m1.x);
              s2 = val1 / val2;
            }
            else if (!(m1.y == 0.0 && m2.y == 0.0 && n1.y == 0.0 &&
                n2.y == 0.0) && !(m1.z == 0.0 && m2.z == 0.0 &&
                n1.z == 0.0 && n2.z == 0.0))
            {
              val1=(m2.y-m1.y)*(n1.z-m1.z)-(n1.y-m1.y)*(m2.z-m1.z);
              val2=(n2.y-n1.y)*(m2.z-m1.z)-(n2.z-n1.z)*(m2.y-m1.y);
              s3 = val1 / val2;
            }

            if (s1)
                if (abs(m1.z - n1.z) != 0 && abs(m2.z - n2.z) != 0)
                dont_draw = 1;
              else
                s = s1;
            else if (s2)
                if (abs(m1.y - n1.y) != 0 && abs(m2.y - n2.y) != 0)
                dont_draw = 1;
              else
                s = s2;
            else if (s3)
                if (abs(m1.x - n1.x) != 0 && abs(m2.x -n2.x) != 0)
```

```
            dont_draw = 1;
          else
            s = s3;


        if (!dont_draw)
        {
          point3.x = n1.x + s * (n2.x - n1.x);
          point3.y = n1.y + s * (n2.y - n1.y);
          point3.z = n1.z + s * (n2.z - n1.z);

          point_list.num_points = 1;
          point_list.points = &point3;
          if (MODE == INSERT)
            pset_pick_id(POINT);
          else if (MODE == COPY)
            pset_pick_id(COPY_POINT);
          ppolymarker3(&point_list);
        }
      }
    }
    break;
case 8: xv_set(item, PANEL_VALUE, 1, NULL);
    if (CPS.num_points > RPS.num_points)
        pt_num = RPS.num_points;
    else
        pt_num = CPS.num_points;
    if (pt_num >= 1)
      for (i = 0; i < pt_num; i++)
      {
        line[0] = CPS.points[i];
        line[1] = RPS.points[i];
        point_list.num_points = 2;
        point_list.points = line;
        if (MODE == INSERT)
          pset_pick_id(LINE);
        else if (MODE == COPY)
          pset_pick_id(COPY_LINE);
          ppolyline3(&point_list);
      }
    break;
case 9: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
    ppoint3.x = atof(str);
    ppoint3.x = ppoint3.x * scale;
    xv_set(item, PANEL_VALUE, "", NULL);
    panel_advance_caret((Panel)xv_get(item,
                        PANEL_PARENT_PANEL));
    break;
case 10: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
    ppoint3.y = atof(str);
    ppoint3.y = ppoint3.y * scale;
    xv_set(item, PANEL_VALUE, "", NULL);
    panel_advance_caret((Panel)xv_get(item,
                        PANEL_PARENT_PANEL));
    break;
case 11: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
    ppoint3.z = atof(str);
```

```
        ppoint3.z = ppoint3.z * scale;
        xv_set(item, PANEL_VALUE, "", NULL);
        panel_advance_caret((Panel)xv_get(item,
                        PANEL_PARENT_PANEL));
        break;
case 12: break;
case 13: point_list.num_points = 1;
        point_list.points = &ppoint3;
        if (MODE == INSERT)
          pset_pick_id(POINT);
        else if (MODE == COPY)
          pset_pick_id(COPY_POINT);
        ppolymarker3(&point_list);
        CPS = point_list;
        break;
case 14: if (CPS.num_points >= 2)
        {
          switch (val){
          case 0: x2_x1 = CPS.points[CPS.num_points - 1].x -
                        CPS.points[CPS.num_points - 2].x;
                  y2_y1 = CPS.points[CPS.num_points - 1].y -
                        CPS.points[CPS.num_points - 2].y;
                  z2_z1 = CPS.points[CPS.num_points - 1].z -
                        CPS.points[CPS.num_points - 2].z;
                  dist = sqrt(x2_x1*x2_x1 + y2_y1*y2_y1 +
                       z2_z1*z2_z1);
                  s = dist/(num_pts + 1);
                  free(point);
                  point = (Ppoint3 *)calloc(num_pts + 2,
                         sizeof(Ppoint3));
                  point[0] = CPS.points[CPS.num_points - 2];
                  point[num_pts+1] = CPS.points[CPS.num_points - 1];
                  for (i = 0; i < num_pts; i++)
                  {
                    point[i+1].x = point[0].x + (i+1)*s*x2_x1/dist;
                    point[i+1].y = point[0].y + (i+1)*s*y2_y1/dist;
                    point[i+1].z = point[0].z + (i+1)*s*z2_z1/dist;

                    point_list.points = &point[i+1];
                    point_list.num_points = 1;
                    if (MODE == INSERT)
                      pset_pick_id(POINT);
                    else if (MODE == COPY)
                      pset_pick_id(COPY_POINT);
                    ppolymarker3(&point_list);
                  }
                  CPS.num_points = num_pts + 2;
                  free(CPS.points);
                  CPS.points = (Ppoint3 *)calloc(num_pts + 2,
                            sizeof(Ppoint3));
                  bcopy((char *)point, (char*)CPS.points,
                       (num_pts + 2) * sizeof(Ppoint3));
                  free(point);
                  NP = 0;
                  NL = 0;
                  NELEM = 0;
                  elem_pos_list.num_ints = 0;
```

```
                break;
            case 1: x2_x1 = CPS.points[CPS.num_points - 1].x -
                        CPS.points[CPS.num_points - 2].x;
                y2_y1 = CPS.points[CPS.num_points - 1].y -
                        CPS.points[CPS.num_points - 2].y;
                z2_z1 = CPS.points[CPS.num_points - 1].z -
                        CPS.points[CPS.num_points - 2].z;
                dist = sqrt(x2_x1*x2_x1 + y2_y1*y2_y1 +
                      z2_z1*z2_z1);
                k = (Pint)dist/space;
                if (k * space == dist)
                  k = k - 1;
                free(point);
                point = (Ppoint3 *)calloc(k + 2,
                        sizeof(Ppoint3));
                point[0] = CPS.points[CPS.num_points - 2];
                point[k+1] = CPS.points[CPS.num_points - 1];
                for (i = 0; i < k; i++)
                {
                  point[i+1].x=point[0].x+(i+1)*space*x2_x1/dist;
                  point[i+1].y=point[0].y+(i+1)*space*y2_y1/dist;
                  point[i+1].z=point[0].z+(i+1)*space*z2_z1/dist;

                  point_list.points = &point[i+1];
                  point_list.num_points = 1;
                  if (MODE == INSERT)
                    pset_pick_id(POINT);
                  else if (MODE == COPY)
                    pset_pick_id(COPY_POINT);
                  ppolymarker3(&point_list);
                }
                CPS.num_points = k + 2;
                free(CPS.points);
                CPS.points = (Ppoint3 *)calloc(k + 2,
                          sizeof(Ppoint3));
                bcopy((char *)point, (char*)CPS.points,
                     (k + 2) * sizeof(Ppoint3));
                free(point);
                NP = 0;
                NL = 0;
                NELEM = 0;
                elem_pos_list.num_ints = 0;
                break;
            case 2: break;
              }
            }
            xv_set(item, PANEL_VALUE, 2, NULL);
            break;
    }
    if (MODE == COPY)
      pset_edit_mode(PEDIT_REPLACE);
}

/*
 * This routine adds a GPD3 element (circle) to the structure.
 */
```

```
static void
circle_proc(item, val)
Panel_item item;
int        val;
{
    static Pgdp_data3     gdp;
    char                  str[20];
    Ppoint_list3          pt_list;
    Pint                  err;

    if (MODE == COPY)
      pset_edit_mode(PEDIT_INSERT);
    switch((int)xv_get(item, PANEL_CLIENT_DATA)){
       case 1: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
             gdp.gdp3_u2.center.x = atof(str);
             gdp.gdp3_u2.center.x = gdp.gdp3_u2.center.x * scale;
             xv_set(item, PANEL_VALUE, "", NULL);
             panel_advance_caret((Panel)xv_get(item,
                          PANEL_PARENT_PANEL));
             break;
       case 2: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
             gdp.gdp3_u2.center.y = atof(str);
             gdp.gdp3_u2.center.y = gdp.gdp3_u2.center.y * scale;
             xv_set(item, PANEL_VALUE, "", NULL);
             panel_advance_caret((Panel)xv_get(item,
                          PANEL_PARENT_PANEL));
             break;
       case 3: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
             gdp.gdp3_u2.center.z = atof(str);
             gdp.gdp3_u2.center.z = gdp.gdp3_u2.center.z * scale;
             xv_set(item, PANEL_VALUE, "", NULL);
             panel_advance_caret((Panel)xv_get(item,
                          PANEL_PARENT_PANEL));
             break;
       case 4: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
             gdp.gdp3_u2.dir[0].delta_x = atof(str);
             gdp.gdp3_u2.dir[0].delta_x = gdp.gdp3_u2.dir[0].delta_x *
                          scale;
             xv_set(item, PANEL_VALUE, "", NULL);
             panel_advance_caret((Panel)xv_get(item,
                          PANEL_PARENT_PANEL));
             break;
       case 5: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
             gdp.gdp3_u2.dir[0].delta_y = atof(str);
             gdp.gdp3_u2.dir[0].delta_y = gdp.gdp3_u2.dir[0].delta_y *
                          scale;
             xv_set(item, PANEL_VALUE, "", NULL);
             panel_advance_caret((Panel)xv_get(item,
                          PANEL_PARENT_PANEL));
             break;
       case 6: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
             gdp.gdp3_u2.dir[0].delta_z = atof(str);
             gdp.gdp3_u2.dir[0].delta_z = gdp.gdp3_u2.dir[0].delta_z *
                          scale;
             xv_set(item, PANEL_VALUE, "", NULL);
             panel_advance_caret((Panel)xv_get(item,
                          PANEL_PARENT_PANEL));
```

```
                break;
        case 7: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
                gdp.gdp3_u2.dir[1].delta_x = atof(str);
                gdp.gdp3_u2.dir[1].delta_x = gdp.gdp3_u2.dir[1].delta_x *
                                        scale;
                xv_set(item, PANEL_VALUE, "", NULL);
                panel_advance_caret((Panel)xv_get(item,
                                PANEL_PARENT_PANEL));
                break;
        case 8: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
                gdp.gdp3_u2.dir[1].delta_y = atof(str);
                gdp.gdp3_u2.dir[1].delta_y = gdp.gdp3_u2.dir[1].delta_y *
                                        scale;
                xv_set(item, PANEL_VALUE, "", NULL);
                panel_advance_caret((Panel)xv_get(item,
                                PANEL_PARENT_PANEL));
                break;
        case 9: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
                gdp.gdp3_u2.dir[1].delta_z = atof(str);
                gdp.gdp3_u2.dir[1].delta_z = gdp.gdp3_u2.dir[1].delta_z *
                                        scale;
                xv_set(item, PANEL_VALUE, "", NULL);
                panel_advance_caret((Panel)xv_get(item,
                                PANEL_PARENT_PANEL));
                break;
        case 10: strcpy(str, (char *)xv_get(item, PANEL_VALUE));
                gdp.gdp3_u2.radius = atof(str);
                gdp.gdp3_u2.radius = gdp.gdp3_u2.radius * scale;
                xv_set(item, PANEL_VALUE, "", NULL);
                panel_advance_caret((Panel)xv_get(item,
                                PANEL_PARENT_PANEL));
                break;
        case 11: if (MODE == INSERT)
                   pset_pick_id(CIRCLE);
                 else if (MODE == COPY)
                   pset_pick_id(COPY_CIRCLE);
                pgdp3(&pt_list, PUGDP_CIRCLE3, &gdp);
                break;
        case 12: break;
    }
    if (MODE == COPY)
      pset_edit_mode(PEDIT_REPLACE);
}

/*
 * This routine adds a GDP3 element (arc) to the structure.
 */

static void
arc_circle_proc(item, val)
Panel_item item;
int        val;
{
    static Pint         inner = TRUE;
    Pgdp_data3          gdp;
    Ppoint_list3        pt_list;
    Pfloat              valx1x0, valx2x0, valy1y0, valy2y0,
```

```
                valz1z0, valz2z0, val1, val2, fac;
Pfloat          tem1, tem2, tem3, tem4, tem5;

switch ((int)xv_get(item, PANEL_CLIENT_DATA)){
case 1: switch(val){
            case 0: inner = TRUE; break;
            case 1: inner = FALSE; break;
          }
          break;
case 2: if (CPS.num_points >= 3)
      {
        if (MODE == COPY)
          pset_edit_mode(PEDIT_INSERT);

        gdp.gdp3_u3.center = CPS.points[CPS.num_points-3];

        valx1x0=CPS.points[CPS.num_points-2].x-gdp.gdp3_u3.center.x;
        valy1y0=CPS.points[CPS.num_points-2].y-gdp.gdp3_u3.center.y;
        valz1z0=CPS.points[CPS.num_points-2].z-gdp.gdp3_u3.center.z;
        valx2x0=CPS.points[CPS.num_points-1].x-gdp.gdp3_u3.center.x;
        valy2y0=CPS.points[CPS.num_points-1].y-gdp.gdp3_u3.center.y;
        valz2z0=CPS.points[CPS.num_points-1].z-gdp.gdp3_u3.center.z;

        val1 = valx1x0*valx2x0+valy1y0*valy2y0+valz1z0*valz2z0;
        tem1 = valx1x0*valx1x0+valy1y0*valy1y0+valz1z0*valz1z0;
        tem2 = valx2x0*valx2x0+valy2y0*valy2y0+valz2z0*valz2z0;
        tem3 = sqrt(tem1);
        tem4 = sqrt(tem2);

        val2 = tem3 * tem4;
        tem5 = val1 / val2;
        fac = val1 / tem1;

        gdp.gdp3_u3.dir[0].delta_x = valx1x0;
        gdp.gdp3_u3.dir[0].delta_y = valy1y0;
        gdp.gdp3_u3.dir[0].delta_z = valz1z0;
        gdp.gdp3_u3.dir[1].delta_x = valx2x0 - fac * valx1x0;
        gdp.gdp3_u3.dir[1].delta_y = valy2y0 - fac * valy1y0;
        gdp.gdp3_u3.dir[1].delta_z = valz2z0 - fac * valz1z0;

        gdp.gdp3_u3.radius = tem3;

        if (inner == TRUE)
        {
          gdp.gdp3_u3.st_ang = 0.0;
          gdp.gdp3_u3.end_ang = acospi(tem5)*M_PI;
        }
        else if (inner == FALSE)
        {
          gdp.gdp3_u3.st_ang = acospi(tem5)*M_PI;
          gdp.gdp3_u3.end_ang = 2*M_PI;
        }
        if (abs(gdp.gdp3_u3.end_ang - 3.14159274101257324) <= 0.00001)
        {
        gdp.gdp3_u3.dir[0].delta_x = 1.0;
        gdp.gdp3_u3.dir[0].delta_y = 0.0;
        gdp.gdp3_u3.dir[0].delta_z = 0.0;
```

```
            gdp.gdp3_u3.dir[1].delta_x = 0.0;
            gdp.gdp3_u3.dir[1].delta_y = 1.0;
            gdp.gdp3_u3.dir[1].delta_z = 0.0;
            }
            if (MODE == INSERT)
              pset_pick_id(ARC_CIRCLE);
            else if (MODE == COPY)
              pset_pick_id(COPY_ARC_CIRCLE);
            pgdp3(&pt_list, PUGDP_CIRC_ARC3, &gdp);
            if (MODE == COPY)
              pset_edit_mode(PEDIT_REPLACE);
          }
          break;
    }
}

/*
 * This routine sets a filter used for picking. During traversal, this
 * set is compared (set-intersection) with the name set to determine
 * the elements which can be picked.
 */

static void
init_pick_filter()
{

    pick_filter.incl_set.num_ints = 10;
    pick_names[0] = LINE;
    pick_names[1] = POINT;
    pick_names[2] = LABEL;
    pick_names[3] = CIRCLE;
    pick_names[4] = ARC_CIRCLE;
    pick_names[5] = COPY_LINE;
    pick_names[6] = COPY_POINT;
    pick_names[7] = COPY_LABEL;
    pick_names[8] = COPY_CIRCLE;
    pick_names[9] = COPY_ARC_CIRCLE;

    pick_filter.incl_set.ints = pick_names;
    pick_filter.excl_set.num_ints = 0;
}

/*
 * This is an input (event) handler for the events occuring in the drawing
 * window.
 */

static void
input_handler(window, event)
Xv_Window window;
Event     *event;
{
   Pescape_in_data     esc_in;
   Pescape_out_data    *esc_out;
   Pint                err;
   Pelem_data          *point_list3;
   Pint                element;
```

```
Pstore          store;

pcreate_store(&err, &store);
if (event_is_button(event) && event_is_down(event)) {
   esc_in.escape_in_u4.point.x = (float)event_x(event);
   esc_in.escape_in_u4.point.y = (float)event_y(event);
   esc_in.escape_in_u4.ws_id = WS;
   esc_in.escape_in_u4.ap_size = 1.0;
   esc_in.escape_in_u4.order = PORDER_TOP_FIRST;
   esc_in.escape_in_u4.depth = 10;
   esc_in.escape_in_u4.pet = PPICK_DEF;
   esc_in.escape_in_u4.echo_switch = PSWITCH_ECHO;
   esc_in.escape_in_u4.echo_volume = echo_volume;
   esc_in.escape_in_u4.filter = pick_filter;
   pescape(PUESC_DRAWABLE_POINT_TO_PICK, &esc_in, store, &esc_out);
   if (esc_out->escape_out_u4.status == PIN_STATUS_OK)
   {
      NELEM = NELEM + 1;
      element = esc_out->escape_out_u4.pick.path_list[0].elem_pos;
      if (NELEM == 1)
      {
        free(elem_pos_list.ints);
        elem_pos_list.ints = (Pint *)malloc(100 * sizeof(Pint));
      }
      else if (NELEM > 100)
        elem_pos_list.ints=(Pint *)realloc(elem_pos_list.ints,
                        (NP * sizeof(Pint)));

      elem_pos_list.ints[NELEM-1] = element;
      elem_pos_list.num_ints = NELEM;

      if (esc_out->escape_out_u4.pick.path_list[0].pick_id == POINT &&
          MODE == INSERT)
      {
        XBell(dpy, 100);
        NP = NP + 1;
        pinq_elem_content(OBJECT, element, store, &err, &point_list3);
        if (NP == 1)
        {
          free(PS.points);
          PS.points = (Ppoint3 *)malloc(100 * sizeof(Ppoint3));
        }
        else if (NP > 100)
          PS.points=(Ppoint3 *)realloc(PS.points,(NP*sizeof(Ppoint3)));

        PS.points[NP-1]  = *point_list3->point_list3.points;
        PS.num_points = NP;
      }
      else if (esc_out->escape_out_u4.pick.path_list[0].pick_id == LINE
             && MODE == INSERT)
      {
        NL = NL + 2;
        pinq_elem_content(OBJECT, element, store, &err, &point_list3);
        if (NL == 2)
        {
          free(LS.points);
          LS.points=(Ppoint3 *)malloc(100*sizeof(Ppoint3));
```

```
            }
            else if (NL > 100)
              LS.points=(Ppoint3 *)realloc(LS.points,(NL*sizeof(Ppoint3)));

            LS.points[NL-2] = point_list3->point_list3.points[0];
            LS.points[NL-1] = point_list3->point_list3.points[1];
            LS.num_points = NL;
          }
          else if
          (esc_out->escape_out_u4.pick.path_list[0].pick_id == COPY_POINT
           && MODE == COPY)
          {
            XBell(dpy, 100);
            NP = NP + 1;
            pinq_elem_content(COPY_OBJECT, element, store, &err,
                              &point_list3);
            if (NP == 1)
            {
              free(PS.points);
              PS.points = (Ppoint3 *)malloc(100 * sizeof(Ppoint3));
            }
            else if (NP > 100)
              PS.points=(Ppoint3 *)realloc(PS.points,(NP*sizeof(Ppoint3)));

            PS.points[NP-1] = *point_list3->point_list3.points;
            PS.num_points = NP;
          }
          else if
          (esc_out->escape_out_u4.pick.path_list[0].pick_id == COPY_LINE
           && MODE == COPY)
          {
            NL = NL + 2;
            pinq_elem_content(COPY_OBJECT, element, store, &err,
                              &point_list3);
            if (NL == 2)
            {
              free(LS.points);
              LS.points=(Ppoint3 *)malloc(100*sizeof(Ppoint3));
            }
            else if (NL > 100)
              LS.points=(Ppoint3 *)realloc(LS.points,(NL*sizeof(Ppoint3)));

            LS.points[NL-2] = point_list3->point_list3.points[0];
            LS.points[NL-1] = point_list3->point_list3.points[1];
            LS.num_points = NL;
          }
        }
      }
      pdel_store(store);
}

/*
 * This routine initializes viewing parameters.
 */

static void
init_view_mapping()
```

```
{
    map.proj_type = PTYPE_PARAL;
    map.win.x_min = WC_MIN;  map.win.x_max = WC_MAX;
    map.win.y_min = WC_MIN;  map.win.y_max = WC_MAX;
    map.proj_vp.x_min = 0.0; map.proj_vp.x_max = 1.0;
    map.proj_vp.y_min = 0.0; map.proj_vp.y_max = 1.0;
    map.proj_vp.z_min = 0.0; map.proj_vp.z_max = 1.0;
    map.proj_ref_point.x = (map.win.x_min + map.win.x_max) / 2.0;
    map.proj_ref_point.y = (map.win.y_min + map.win.y_max) / 2.0;
    map.proj_ref_point.z = 6.0 * WC_MAX;
    map.front_plane = 2.0 * WC_MAX;
    map.back_plane =  2.0 * WC_MIN;
    map.view_plane =  0.4 * map.proj_ref_point.z;

}

/*
 * This routine sets viewing representation of the drawing window.
 */

static void
set_view_rep()
{
    Pint       err;


    peval_view_map_matrix3(&map, &err, rep.map_matrix);
    peval_view_ori_matrix3(&vrp, &vpn, &vup, &err, rep.ori_matrix);
    rep.clip_limit = map.proj_vp;
    rep.xy_clip = rep.back_clip = rep.front_clip = PIND_NO_CLIP;
    pset_view_rep3(WS, VIEW, &rep);
}

/*
 * This routine sets viewing representation of the text window.
 */

static void
set_view_rep2()
{
    static Pmatrix   identity = {{1.0, 0.0, 0.0},
                                 {0.0, 1.0, 0.0},
                                 {0.0, 0.0, 1.0},};
    static Pview_map view2 = {{0.0, 100.0, 0.0, 100.0},
                              {0.0, 1.0, 0.0, 1.0},};
    Pview_rep        view_rep;
    Pint             err;

    peval_view_map_matrix(&view2, &err, view_rep.map_matrix);
    bcopy((char *)identity, (char*)view_rep.ori_matrix,
        sizeof(Pmatrix));
    view_rep.clip_limit = view2.proj_vp;
    view_rep.xy_clip = PIND_NO_CLIP;
    pset_view_rep(WS_2, TEXT_VIEW, &view_rep);
}

/*
```

```
 * This routine creates a fixed structure containing a set of axis.
 */

static void
build_axis()
{
    Pint           num_points = 2, i;
    Ppoint3        axis_origin, axis[2], scale[2];
    Ppoint_list3   pt_list;


        axis_origin.x = 0.0;
        axis_origin.y = 0.0;
        axis_origin.z = 0.0;

        popen_struct(AXIS);
        pset_view_ind(VIEW);

        axis[0] = axis[1] = axis_origin;
        scale[0] = scale[1] = axis_origin;

        axis[1].x = 100.0;
        pset_line_colr_ind(WHITE); /* RED */
        pt_list.num_points = num_points;
        pt_list.points = axis;
        ppolyline3(&pt_list);
        for (i = 1; i <= 10; i++)
        {
            scale[0].x = scale[1].x = 10 * i;
            scale[0].y = -0.5;
            scale[1].y = 0.5;
            pt_list.points = scale;
            ppolyline3(&pt_list);
        }

        axis[1].x = axis_origin.x;
        axis[1].y = 100.0;
        pset_line_colr_ind(WHITE); /* SPRING_GREEN */
        pt_list.num_points = num_points;
        pt_list.points = axis;
        ppolyline3(&pt_list);
        for (i = 1; i <= 10; i++)
        {
            scale[0].y = scale[1].y = 10 * i;
            scale[0].x = -0.5;
            scale[1].x = 0.5;
            pt_list.points = scale;
            ppolyline3(&pt_list);
        }

        axis[1].y = axis_origin.y;
        axis[1].z = 100.0;
        scale[0].y = scale[1].y = 0.0;
        pset_line_colr_ind(WHITE); /* 5 */
        pt_list.num_points = num_points;
        pt_list.points = axis;
        ppolyline3(&pt_list);
```

```
        for (i = 1; i <= 10; i++)
        {
            scale[0].z = scale[1].z = 10 * i;
            scale[0].x = -0.5;
            scale[1].x = 0.5;
            pt_list.points = scale;
            ppolyline3(&pt_list);
        }

        pclose_struct();
}

/*
 * This is a repaint procedure for the drawing window. It is invoked
 * whenever the window is obscured, resized or scrolled.
 */

static void
canvas_repaint_proc(canvas, paint_window, dpy, xwin, xrects)
Canvas     canvas;
Xv_Window  paint_window;
Display    *dpy;
Window     xwin;
Xv_xrectlist *xrects;
{
    switch((Pint)xv_get(canvas, XV_KEY_DATA, WS_ID_KEY)){
      case WS: predraw_all_structs(WS, PFLAG_ALWAYS); break;
      case WS_2: predraw_all_structs(WS_2, PFLAG_ALWAYS); break;
    }
}

/*
 * This routine creates and displays a folder of the project.
 */

static void
disp_folder(folder_frame)
Frame folder_frame;
{
    Canvas          canvas;
    Window          win;
    Pint            row = 0, col = 0;
    Xv_Window       canvas_pw;
    XGCValues       gcvalues;
    Font            font, font2;
    Display         *dpy;
    Pixmap          pattern;
    GC              gc_f, gc_f2, gc_r, gc_r2;
    XEvent          report;
    Pint            expose = 0;
    int             x, y;
    unsigned int    width, height;

    xv_set(folder_frame,
        XV_X,  0,
        XV_Y,  0,
        XV_WIDTH,       1144,
```

```
    XV_HEIGHT,        868,
    FRAME_LABEL,      "3D GAS",
    NULL);
canvas = (Canvas)xv_create(folder_frame, CANVAS,
    XV_X,  0,
    XV_Y,  0,
    XV_WIDTH,         1144,
    XV_HEIGHT,        868,
    NULL);
canvas_pw = (Xv_Window)canvas_paint_window(canvas);
win = (Window)xv_get(canvas_pw, XV_XID);

window_fit(folder_frame);
dpy = (Display *)xv_get(folder_frame, XV_DISPLAY);
pattern = XCreateBitmapFromData(dpy, win, pattern_bits,
                          pattern_width, pattern_height);
font = XLoadFont(dpy, "charB24");
font2 = XLoadFont(dpy, "9x15");

gc_f = XCreateGC(dpy, RootWindow(dpy, DefaultScreen(dpy)),
      0, &gcvalues);
gc_f2 = XCreateGC(dpy, RootWindow(dpy, DefaultScreen(dpy)),
      0, &gcvalues);
gc_r = XCreateGC(dpy, RootWindow(dpy, DefaultScreen(dpy)),
      0, &gcvalues);
gc_r2 = XCreateGC(dpy, RootWindow(dpy, DefaultScreen(dpy)),
      0, &gcvalues);

XSetFont(dpy, gc_f, font);
XSetFont(dpy, gc_f2, font2);
XSetStipple(dpy, gc_r, pattern);
XSetFillStyle(dpy, gc_r, FillStippled);
XSetForeground(dpy, gc_f, BlackPixel(dpy, DefaultScreen(dpy)));
XSetForeground(dpy, gc_f2, BlackPixel(dpy, DefaultScreen(dpy)));
XSetForeground(dpy, gc_r, BlackPixel(dpy, DefaultScreen(dpy)));
XSetForeground(dpy, gc_r2, WhitePixel(dpy, DefaultScreen(dpy)));

xv_set(folder_frame, XV_SHOW, TRUE, NULL);
XFlush(dpy);
while(!expose)
{
  XNextEvent(dpy, &report);
  switch(report.type) {
  case Expose:
    if(report.xexpose.count == 0)
    {
      expose = 1;
      XFillRectangle(report.xexpose.display, report.xexpose.window,
              gc_r, 0, 0, 1144, 868);
      XFillRectangle(report.xexpose.display, report.xexpose.window,
              gc_r2, 380, 260, 400, 300);
      XDrawString(report.xexpose.display, report.xexpose.window,
              gc_f, 408, 350, "3D GRAPHICS APPLICATION SYSTEM", 30);
      XDrawString(report.xexpose.display, report.xexpose.window,
              gc_f2, 570, 460, "by", 2);
      XDrawString(report.xexpose.display, report.xexpose.window,
              gc_f2, 470, 500, "Slawomir (Nick)  WERCZAK", 24);
```

```
            XFlush(dpy);
          }
          break;
        default: break;
        }
      }
}

/*
 * This routine opens phigs and the workstations and initializes the
 * structures: OBJECT and COPY_OBJECT.
 */

static void
open_phigs()
{
    Frame              frame;
    Frame              folder_frame;
    Panel              panel;
    Cms                cms;
    Canvas             canvas;
    Scrollbar          horiz_scrollbar;
    Scrollbar          vert_scrollbar;
    Xv_Window          canvas_pw;
    Xv_Cursor          cursor;
    Pconnid_x_drawable xconnid;
    Pint               err_ind;
    Pcolr_rep          rep0, rep4;
    Pint               row = 0;
    Pint               col = 0;
    Pint               wst1;
    Ppoint             point[2];
    Ppoint_list        pt_list;
    static char        str[] = "Model 1";
    XEvent             report;
    Pint               expose = 0;


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

/* Open PHIGS and create the workstations. */

    popen_phigs((char*)NULL, PDEF_MEM_SIZE);
    wst1 = phigs_ws_type_create(phigs_ws_type_x_drawable,
        PHIGS_COLOUR_MODE, PHIGS_INDEX_COLOUR,
        PHIGS_COLOUR_TABLE_SIZE, 8,
        NULL);

    cms = (Cms)xv_create(NULL, CMS,
      CMS_CONTROL_CMS,        TRUE,
      CMS_SIZE,               CMS_CONTROL_COLORS + NUM_COLORS,
      CMS_COLORS,             colors,
      NULL);

    cursor = (Xv_Cursor)xv_create(NULL, CURSOR,
      CURSOR_SRC_CHAR,        OLC_NAVIGATION_LEVEL_PTR,
```

```
        NULL);

/* Create the XView frame with a panel subwindow and a canvas subwindow.
   The canvas subwindow will become a SunPHIGS workstation.
*/

    frame = (Frame)xv_create(NULL, FRAME_BASE, NULL);
    xv_set(frame, FRAME_LABEL, "3D GAS", NULL);
    dpy = (Display *)xv_get(frame, XV_DISPLAY);

    folder_frame = (Frame)xv_create(frame, FRAME_CMD, NULL);
    disp_folder(folder_frame);


    panel = create_panel(frame, cms);
    window_fit(frame);

    init_pick_filter();

    row = 1;
    col = 37;
    canvas = (Canvas)xv_create(frame, CANVAS,
       XV_X,              xv_col(frame, col),
       XV_Y,              xv_row(panel, row),
       XV_WIDTH,          WS_WIDTH,
       XV_HEIGHT,         WS_HEIGHT,
       CANVAS_WIDTH,      WS_WIDTH,
       CANVAS_HEIGHT,     WS_HEIGHT+ 50,
       CANVAS_AUTO_EXPAND, FALSE,
       CANVAS_AUTO_SHRINK, FALSE,
       OPENWIN_AUTO_CLEAR, FALSE,
       CANVAS_RETAINED,   TRUE,
       CANVAS_FIXED_IMAGE, FALSE,
       CANVAS_REPAINT_PROC, canvas_repaint_proc,
       CANVAS_X_PAINT_WINDOW, TRUE,
       XV_KEY_DATA, WS_ID_KEY, WS_2,
       NULL);

    canvas_pw = (Xv_Window)canvas_paint_window(canvas);
    xconnid.drawable_id = (Window)xv_get(canvas_pw, XV_XID);
    xconnid.display = (Display *)xv_get(canvas, XV_DISPLAY);

    popen_ws(WS_2, (void *)&xconnid, wst1);

    row = 2;
    col = 37;
    canvas = (Canvas)xv_create(frame, CANVAS,
       XV_X,              xv_col(frame, col),
       XV_Y,              xv_row(panel, row),
       XV_WIDTH,          WS_WIDTH,
       XV_HEIGHT,         WS_HEIGHT,
       CANVAS_WIDTH,      CV_WIDTH,
       CANVAS_HEIGHT,     CV_HEIGHT,
       CANVAS_AUTO_EXPAND, FALSE,
       CANVAS_AUTO_SHRINK, FALSE,
       OPENWIN_AUTO_CLEAR, FALSE,
       CANVAS_RETAINED,   TRUE,
```

```
    CANVAS_FIXED_IMAGE, FALSE,
    CANVAS_REPAINT_PROC, canvas_repaint_proc,
    CANVAS_X_PAINT_WINDOW, TRUE,
    XV_KEY_DATA, WS_ID_KEY, WS,
    NULL);
vert_scrollbar = (Scrollbar)xv_create(canvas, SCROLLBAR,
    SCROLLBAR_DIRECTION,          SCROLLBAR_VERTICAL,
    SCROLLBAR_PIXELS_PER_UNIT,    10,
    SCROLLBAR_OBJECT_LENGTH,      CV_WIDTH,
    SCROLLBAR_VIEW_LENGTH,        WS_WIDTH,
    SCROLLBAR_PAGE_LENGTH,        WS_WIDTH,
    NULL);
horiz_scrollbar = (Scrollbar)xv_create(canvas, SCROLLBAR,
    SCROLLBAR_DIRECTION,          SCROLLBAR_HORIZONTAL,
    SCROLLBAR_PIXELS_PER_UNIT,    10,
    SCROLLBAR_OBJECT_LENGTH,      CV_HEIGHT,
    SCROLLBAR_VIEW_LENGTH,        WS_HEIGHT,
    SCROLLBAR_PAGE_LENGTH,        WS_HEIGHT,
    NULL);
window_fit(frame);

canvas_pw = (Xv_Window)canvas_paint_window(canvas);
xv_set(canvas_pw,
    WIN_CURSOR,         cursor,
    WIN_EVENT_PROC, input_handler,
    WIN_CONSUME_EVENTS, WIN_MOUSE_BUTTONS,
                NULL,
    WIN_CONSUME_X_EVENT_MASK, ButtonPressMask,
                NULL,
    NULL);

xconnid.display = (Display *)xv_get(canvas, XV_DISPLAY);
xconnid.drawable_id = (Window)xv_get(canvas_pw, XV_XID);

popen_ws(WS, (void *)&xconnid, wst1);

pinq_colr_rep(WS, 4, PINQ_REALIZED, &err_ind, &rep4);
pinq_colr_rep(WS, 0, PINQ_REALIZED, &err_ind, &rep0);
pset_colr_rep(WS, 0, (Pcolr_rep *)&rep4);
pset_colr_rep(WS, 4, (Pcolr_rep *)&rep0);

pinq_colr_rep(WS_2, 4, PINQ_REALIZED, &err_ind, &rep4);
pinq_colr_rep(WS_2, 0, PINQ_REALIZED, &err_ind, &rep0);
pset_colr_rep(WS_2, 0, (Pcolr_rep *)&rep4);
pset_colr_rep(WS_2, 4, (Pcolr_rep *)&rep0);

phigs_ws_type_destroy(wst1);

set_view_rep();
set_view_rep2();

build_axis();

pset_disp_upd_st(WS, PDEFER_WAIT, PMODE_UQUM);
pset_disp_upd_st(WS_2, PDEFER_WAIT, PMODE_UQUM);

ppost_struct(WS, OBJECT, 0.1);
```

```
ppost_struct(WS, COPY_OBJECT, 0.05);
ppost_struct(WS_2, TEXT_OBJECT, 1.0);

point[0].x = 0.0;
point[0].y = 96.5;
point[1].x = 185.0;
point[1].y = 96.5;

popen_struct(TEXT_OBJECT);
pset_view_ind(TEXT_VIEW);
pset_text_font(PFONT_SIMPLEX);
pset_text_colr_ind(WHITE);
pset_char_ht (1.2);
pset_linewidth(0.1);
pset_line_colr_ind(WHITE);

pt_list.num_points = 2;
pt_list.points = point;
ppolyline(&pt_list);

point[0].x = 1.0;
point[0].y = 98.0;
ptext(point[0], str);
pclose_struct();

popen_struct(COPY_OBJECT);
pset_view_ind(VIEW);
init_names_set();
pset_marker_size(0.2);
pset_marker_colr_ind(6);
pset_marker_type(PMARKER_SQUARE);
pset_line_colr_ind(6);
pset_int_colr_ind(6);
pset_anno_char_ht (0.008);
pset_anno_style(PANNO_STYLE_UNCONNECTED);
pset_curve_approx(PCURV_CONSTANT_PARAMETRIC_BETWEEN_KNOTS,
                  65.5);

pclose_struct();

popen_struct(OBJECT);
pset_view_ind(VIEW);
init_names_set();
pset_marker_size(0.2);
pset_marker_colr_ind(WHITE);
pset_marker_type(PMARKER_SQUARE);
pset_line_colr_ind(WHITE);
pset_int_colr_ind(WHITE);
pset_anno_char_ht (0.008);
pset_anno_style(PANNO_STYLE_UNCONNECTED);
pset_curve_approx(PCURV_CONSTANT_PARAMETRIC_BETWEEN_KNOTS,
                  65.5);

window_fit(frame);
while(!expose)
{
  XNextEvent(dpy, &report);
```

```
        switch(report.type) {
        case Expose:
          if(report.xexpose.count == 0)
          {
            expose = 1;
            XFlush(dpy);
            xv_set(frame, XV_SHOW, TRUE, NULL);
            xv_set(folder_frame, XV_SHOW, FALSE, NULL);
            xv_destroy_safe(folder_frame);
          }
          break;
        default: break;
        }
      }

      xv_main_loop(frame);

}

/* main program */

main(argc, argv)
int   argc;
char  *argv[];
{
      char *       helppath;
      char         buf[BUFSIZ];
      extern char *getenv();


/* Initialize XView */

      xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);

      sprintf(buf, "HELPPATH=/usr/lib/help:%s:", HELPPATHNAME);
      if (helppath = getenv("HELPPATH"))
        strcat(buf, helppath);
      putenv(buf);

/* Open PHIGS and create the SunPHIGS X Drawable workstations. */

      open_phigs();
      exit(0);
}
```

gas.h

```
#include <phigs/phigs.h>
#include <xview/xview.h>
#include <xview/frame.h>
#include <xview/canvas.h>
#include <xview/panel.h>
#include <xview/scrollbar.h>
#include <xview/cms.h>
#include <xview/font.h>
#include <xview/xv_xrect.h>
#include <xview/cursor.h>
#include <xview/textsw.h>
#include <math.h>

#define HELPPATHNAME  "/u0/grad/werczak/proj"

#define AXIS          1
#define OBJECT        2
#define COPY_OBJECT   3
#define TEXT_OBJECT   4

#define WS            1
#define WS_2          2

#define WC_MIN      -210.0
#define WC_MAX       210.0

#define WS_WIDTH     846
#define WS_HEIGHT    798
#define CV_WIDTH    1692
#define CV_HEIGHT   1664

#define NUM_NAMES     20
#define WS_ID_KEY    100

#define VIEW          1
#define TEXT_VIEW     2

#define MAX_INT    32767

#define DEG_TO_RAD(D)  ((3.14159265358 / 180.0) * (D))

/* Primitive pick and name set ids. */
#define POINT         11
#define LINE          12
#define LABEL         13
#define CIRCLE        14
#define ARC_CIRCLE    15
#define COPY_POINT        21
#define COPY_LINE         22
#define COPY_LABEL        23
#define COPY_CIRCLE       24
#define COPY_ARC_CIRCLE   25
```

```
#define INSERT        0
#define COPY          1

#define ON            1
#define OFF           0

#define start_ptr     11

#define BLACK         0
#define WHITE         1
#define RED           2
#define SPRING_GREEN  3
#define PINK          4
#define YELLOW        7
#define CYAN          6
#define ORANGE        5

#define NUM_COLORS    8

#define pattern_width  8
#define pattern_height 8

static char          pattern_bits[] = { 0x55, 0xaa, 0x55, 0xaa,
                                        0x55, 0xaa, 0x55, 0xaa};

static Display       *dpy;
static Plimit3       echo_volume = {0.0, 825.0, 0.0, 811.0, 0.0, 1.0};
static Pfilter       pick_filter;
static Pint          pick_names[NUM_NAMES];

static Pview_rep3     rep;
static Pview_map3     map;
static Ppoint3        vrp = {0.0, 0.0, 0.0};
static Pvec3          vup = {0.0, 1.0, 0.0};
static Pvec3          vpn = {0.0, 0.0, 1.0};

static Pfloat         scale = 1.0;

static Ppoint_list3   PS, LS, RPS, CPS, RLS, CLS;
static Pint_list      elem_pos_list;
static Pint           NP, NL, NELEM;

static Pint           MODE = INSERT;
static Pint           SAVE = FALSE;
static Pint           LOAD = FALSE;
static Pint           join = 0;

static Xv_singlecolor  colors[] = {
                    {  0,   0,   0}, black
                    { 252, 252, 252}, white
                    { 255,   0,   0}, red
                    {  0, 255, 127}, green
                    { 188, 143, 143}, pink
                    { 204,  50,  50}, orange
                    {  0, 255, 255}, cyan
                    { 255, 255,   0}, yellow
                    };
```

```
    static char*        help_message = "To get help on any item on the panel move the cursor to
that item and press <Help> key.";

    static void         init_names_set();
    static void         init_pick_filter();
    static void         input_handler();
    static void         init_view_mapping();
    static void         build_axis();
    static void         canvas_repaint_proc();
    static void         open_phigs();
    static void         drawing_proc();
    static void         label_setting_proc();
    static void         rotation_proc();
    static void         shift_proc();
    static void         scale_proc();
    static void         view_proc();
    static void         cps_proc();
    static void         rps_proc();
    static void         cls_proc();
    static void         rls_proc();
    static void         frame_notify_proc();
    static void         file_frame_notify_proc();
    static void         axis_notify_proc();
    static void         show_help();
    static void         exit_help();
    static void         exit_menu_proc();
    static void         delete_copy_proc();
    static void         tool_menu_proc();
    static void         delete_proc();
    static void         set_view_rep();
    static void         circle_proc();
    static void         arc_circle_proc();
    static void         file_menu_proc();
    static void         cance_delete_proc();
    static void         create_circle_subpanel();
    static void         create_point_subpanel();
    static void         create_drawing_menu();
    static void         create_a_h_e_c_menu();
    static void         create_delete_menu();
    static void         create_p_l_t_menu();
    static void         create_arc_subpanel();
    static void         cancel_delete_proc();
    static void         create_label_subpanel();
    static void         create_side_menu();
    static void         create_circle_arc_menu();
    static void         create_text_subpanel();
    static void         create_shift_subpanel();
    static void         create_transform_menu();
    static void         create_rotate_subpanel();
    static void         create_shift_subpanel();
    static void         create_file_menu();
    static void         create_tool_menu();
    static void         create_help_menu();
    static void         create_delete_menu();
    static void         create_view_menu();
    static Panel        create_panel();
```

```
static int          intcompare();
static void         create_scale_subpanel();
```

APPENDIX C

:help
'Help' button

Press the Help button to get information about any 3D GAS control.
#
:quit
'Exit' button

Press the Exit button to exit the program.
#
:tool
'Tool' button

Pressing the Tool button gives you the following choices:

1. refreshing the screen;
2. deleting the main structure (only in INSERT MODE);
3. undisplaying the main structure (deletes the structure from display)
4. displaying the main structure (adds the structure to display)

#
:file
'File' button

Pressing the File button gives you the following choices:

1. loading the main structure from a specified file;
2. saving the main structure to a specified file.

#
:point
'Point' button

Pressing the Point button allows the entry of the x, y and z coordinates of a point structure element. If the mode is INSERT, the element will be appended to the main structure; if the mode is COPY, the element will be appended to the copy structure.
#
:label
'Label' button

Pressing the Label button allows the entry of the label of a node and its position the device coordinates relative to the position of the node. The node is defined by the last point structure element of the Current Point Set (see CPS help menu).
#
:text
'Text' button

Pressing the Text button allows the entry of the text displayed in the text window.
#
:circle

'Circle' button

Pressing the Circle button allows the entry of data for a circle structure element. These data consist of:

1. the x, y, z coordinates of the centre of the circle;
2. the x, y, z coordinates of two vectors defining the plane on which the circle will be displayed;
3. the value of the radius.

If the mode is INSERT, the element will be appended to the main structure; if the mode is COPY, the element will be appended to the copy structure.
#
:arc-circle
'Arc-circle' button

Pressing the Arc-circle button draws an arc structure element. This arc is defined by the last three points of the CPS (see CPS help menu) and an INNER ARC/OUTER ARC choice menu. The points should be entered into the CPS in the following order:

1. the centre point of the arc;
2. two points defining the limits of the arc (the order does not matter).

If the INNER ARC is chosen, the arc of smaller angle will be drawn. If the OUTER ARC is chosen, the arc of greater angle will be drawn.

If the mode is INSERT, the element will be appended to the main structure; if the mode is COPY, the element will be appended to the copy structure.
#
:axis
'AXIS ON/AXIS OFF' choice menu

Selection of AXIS ON displays a set of x, y and z axes in the world coordinate system. AXIS OFF deletes the set of x, y, z axis from display. The colors of the axis are as follows:

x: red;
y: green;
z: yellow.
#
:projection

'PROJ. PERSPECTIVE/PROJ. PARALLEL'

Selction of PROJ. PERSPECTIVE sets the perspective projection type.
Selection of PROJ. PARALLEL sets the parallel projection type.
#
:mode
'COPY MODE/INSERT MODE' choice menu

COPY MODE/INSERT MODE defines the drawing mode in which the drawing occurs. If in INSERT MODE, a main structure is created. If in COPY MODE, a copy structure is created.
#
:range
'Range of axis' choice menu

Range of axis sets the length of the axis to represent the value of the range.
#
:curdir
'Current direction' choice menu

If the DIR ON/DIR OFF choice menu is set to DIR OFF, 'Current direction' defines the current direction used in the drawing routines. If DIR ON/DIR OFF is set to DIR ON, 'Current direction' is ignored.
#
:noelem
'No. Elements' text field

'No. Elements' defines the number of elements to be drawn by the drawing routines (see the DRAW PTS/DRAW OBJS and DRAW NUP/DRAW SPP help menus).
#
:space
'Space' text field

'Space' defines the displacement used in the drawing routines (see DRAW PTS/DRAW OBJS and DRAW NUP/DRAW SPP help menus).
"#
:insdelcopy
'INSERT COPY/DELETE COPY' choice menu

INSERT COPY appends the copy structure to the main structure. DELETE COPY deletes the copy structure.
#
:cps
'CPS ON/CPS OFF' choice menu

CPS ON defines any previously picked points as the Current Point Set. Picking a point adds it to the temporary point list. To pick a point move the mouse above the specified point and click with one of the mouse buttons (all buttons are equivalent). An audible signal means that the point has been picked.
#
:rps
'RPS ON/RPS OFF' choice menu

RPS ON defines any previously picked points as the Reference Point Set. Picking a point adds it to the temporary point list.To pick a point move the mouse above the specified point and click with one of the mouse buttons (all buttons are equivalent). An audible signal means that the point has been picked.
:cls
'CLS ON/CLS OFF' choice menu

CLS ON defines any previously picked lines as the Current Line Set. Picking a line adds it to the temporary line list.To pick a line move the mouse above the specified line and click with one of the mouse buttons (all buttons are equivalent). Flashing the line means that the line has been picked.
#
:rls
'RLS ON/RLS OFF' choice menu

RLS ON defines any previously picked lines as the Reference Line Set. Picking the line adds it to the temporary line list. To pick a line move the mouse above the specified line and click with one of the mouse buttons (all buttons

are equivalent). Flashing the line means that the line has been picked.
#
:dir
'DIR ON/DIR OFF' choice menu

DIR ON allows the definition of an arbitrary direction to be used in the drawing routines. The direction is defined by the last two points in the CPS. DIR OFF means that the value of the direction is the actual value of the Current direction (see Current direction help menu).
#
:jp
'JP ON/JP OFF' choice menu

JP ON tells the drawing and transformation routines that the end points of any set of points created by those routines should be joined with lines. JP ON affects the following routines:

1. DRAW PTS;
2. X, Y, Z Rotation (if mode is INSERT);
3. Shift (if mode is INSERT).

#
:rot
'Rot X, Rot Y, Rot Z' button

Rot X, Rot Y and Rot Z perform x, y and z rotations. If the drawing mode is INSERT, the structure resulting from the transformation is appended to the main structure. Additionally, if the join mode is set to JP ON the coresponding points of the main structure and the transformed structure will be joined with straight lines. If the drawing mode is COPY the copy structure is replaced by the transformed one.
#
:shift
'Shift' button

'Shift' performs x, y and z translations. If the drawing mode is INSERT, the structure resulting from the transformation is appended to the main structure. Additionally, if the join mode is set to JP ON, the corresponding points of the main structure and the transformed structure will be joined with straight lines. If the drawing mode is COPY, the copy structure is replaced by the transformed one.
#
:scale
'Scale' button

The 'Scale' button allows for entery of the x, y, and z factors used to scale the structure. If the structure contains a circle or arc elements, scaling using the different scale factors in different directions is not possible. The scale factors for such structures must be therefore the same.
#
:drawpo
'DRAW PTS/DRAW OBJS' choice menu

DRAW PTS creates a set of points which by default becomes the Current Point Set. The input to the DRAW PTS routine consists of:

1. the number of points to be drawn defined by 'No. Elements';
2. the displacement defined by 'Space';

3. the direction defined by 'Current direction' or the CPS if 'DIR ON' is selected;
4. the CPS.

DRAW OBJS creates a set of structures which are copies of the existing structure. The input to this routine consists of:

1. the number of structures to be drawn defined by 'No. Elements";
2. the displacement defined by 'Space';
3. the direction defined either by 'Current direction' or the CPS if 'DIR ON' is selected.

If the mode is INSERT, the element will be appended to the main structure; if the mode is COPY, the element will be appended to the copy structure.
#
:drawns
'DRAWS NUP/DRAW SPP' choice menu

DRAW NUP creates a given number of points between two points defined as the CPS. The input to this routine consists of:

1. the number of points to be drawn defined by 'No. Elements';
2. the CPS.

DRAW SPP creates a set of points separated by a given displacement between two points defined as the CPS. The input to this routine consists of:

1. the displacement defined by 'Space';
2. the CPS.

By default the newly created set of points is appended to the CPS.
#
:join
'JOIN PTS ON/JOIN PTS OFF' choice menu

JOIN PTS ON joins the points from the CPS to the corresponding points from the RPS with the straight lines. The input to this routine consists of:

1. the CPS;
2. the RPS.

#
:intersect
'INTERSECT ON/INTERSECT OFF' choice menu

INTERSECT ON creates a set of points at the intersection of two line sets. The input to this routine consists of:

1. the CLS;
2. the RLS.

#
:delete
'Delete' button

'Delete' deletes a list of picked elements of the structure.
#
:canceldel
'Cancel delete action' button

'Cancel delete action' resets the list of elements of the structure to be deleted.
#
:view
'view x, view y, view z' sliders

'View x', 'view y' and 'view z' sliders allow for changing the viewing paramenters and achieve the effect of the rotation of the coordinate system.
#
:zoom
'zoom' slider

If the projection mode is PERSPECTVE, changing the value of this slider will cause the effect of zooming.
#

BIBLIOGRAPHY

- B. Artwick, Microcomputer displays, graphics and animation, Prentice-Hall, Englewood Cliffs, New Jersey, 1985.

[1] • D. Heller, XView Programming Manual, O'Reilly & Associates, Inc ., Sebastopol, CA, 1990.

- F. J. Johnson, AutoCAD in 3D, McGraw-Hill, Inc., 1991.
- O. Jones, Introduction to the X Window System, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1989.
- H. F. Keedy, An introduction to CAD using CADKEY, PWS-KENT Publishing Company, Boston, 1988.
- A. R. Miller, The ABC's of AutoCAD, SYBEX, Inc., Alameda, CA, 1988.
- A. Nye, Xlib Programming Manual, O'Reilly & Associates, Inc., Sebastopol,

[2] • SunPHIGS 2.0 Programming Guide, Sun Microsystems, Inc., 1991.

[3] • SunPHIGS 2.0 Installation Guide, Getting Started with SunPHIGS, SunPHIGS 2.0 Porting Guide, Sun Microsystems, Inc., 1991.

- Taskmaster Mark II / User's manual for Integrated ComputerSystems in engineering design and construction, CADSYS LIMITED, Edmonton, 1976.
- G. B. Thomas, Jr., R. L. Finney, Calculus and Analytic Geometry, Part II/Sixth Edition, Reading, Mass: Addison-Wesley Publishing Company, 1984.

[4] • H. Zhou, Engineering Graphics Application System, McMaster University, Department of Computer Science and Systems, 1992.