ESV Workstation

Stereo User's Manual

EVANS & SUTHERLAND COMPUTER CORPORATION
Salt Lake City, Utah

DOCUMENTATION WARRANTY:

PURPOSE: This documentation is provided to assist an Evans & Sutherland trained BUYER in using a product purchased from Evans & Sutherland. It may contain errors or omissions that only a trained individual may recognize. Changes may have occurred to the hardware/software, to which this documentation refers, which are not included in this documentation or may be on a separate errata sheet. Use of this documentation in such changed hardware/software could result in damage to hardware/software. User assumes full responsibility of all such results of the use of this data.

WARRANTY: This document is provided, and Buyer accepts such documentation, "AS-IS" and with "ALL FAULTS, ERRORS, AND OMISSIONS." BUYER HEREBY WAIVES ALL IMPLIED AND OTHER WARRANTIES, GUARANTIES, CONDITIONS OR LIABILITIES, EXPRESSED OR IMPLIED ARISING BY LAW OR OTHERWISE, INCLUDING, WITHOUT LIMITATIONS, ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. BUYER FURTHER HOLDS SELLER HARMLESS OF ANY DIRECT OR INDIRECT DAMAGES, INCLUDING CONSEQUENTIAL DAMAGES.

Part Number: 517941-101 AA
April, 1991

Table of Contents

1. Overview

This manual is arranged as follows:

- Chapter 1 (this chapter) describes the Stereo Viewing Option components, stereoscopic graphics, and contains a video overview.
- Chapter 2 describes stereo implementation using PHIGS.
- Chapter 3 describes stereo implementation using ES/PSX.
- Appendix A contains a glossary of stereoscopic terms.
- Appendix B contains a PHIGS stereo example program.

Stereoscopic Graphics

When a monoscopic display of a 3D object is viewed on a graphics screen, several cues suggest to the eye-brain system that the image represents a 3D object:

- The edges of the object meet at angles which make sense only if the brain assumes that the 2D picture is actually a projection of a 3D object.
- The intensity of edges and other lines in the picture can be made to vary, suggesting depth by the dimming of the lines with distance.
- The object can be drawn in perspective, where the portions of the object which are closer to the viewer are drawn larger than those which are farther from the viewer.

Adequate pictures can be produced with these cues which faithfully reproduce an object as it would be viewed from a selected point. However, the brain recognizes that only a picture is being viewed, and not the object itself, because the most important depth cue, *stereopsis*, is missing.

Stereopsis gives viewers the ability to qualitatively and, to some degree, quantitatively, judge the distance to an object by how different the object looks to each of their two eyes. The difference in appearance between the two views is caused by *parallax*, the apparent shifting of a nearby object with respect to a far object as the viewer's point of view shifts. The eye-brain system is carefully trained to observe this change in appearance and uses it to accurately judge the relative distance and size of an object.

To display true 3D, each eye must be made to see a different image, adjusted for the correct parallax. Since the placement of the two-eye system is important, the view must be adjusted for the correct position of the viewer with respect to the monitor screen. In this way, the screen can be made to ap-

pear as a transparent window through which a real 3D object is seen, not as a surface on which a picture is drawn.

Stereo Viewing Option

When viewing a 3D object, each eye views the object from a slightly different perspective because of the *interocular separation*. 2D views of the object, called *stereoscopic fields*, are projected onto the retinae of the eyes. The stereoscopic fields are transmitted to the brain, and the brain fuses them by a process known as *stereopsis* to form a 3D view of the object.

The Stereo Viewing Option provides the hardware and software tools required to create the left-eye and right-eye stereoscopic fields of a 3D object on the ESV Workstation; to alternately display the two stereoscopic fields on a monitor; and to direct the left-eye stereoscopic field to the left eye and the right-eye stereoscopic field to the right eye of the user.

Stereo on the ESV Workstation

The ESV Workstation monitor accommodates both monoscopic and stereoscopic viewing of an image. ESV Workstation stereoscopy is non-interlaced, which means that each time the monitor screen is refreshed, either the right-eye or left-eye stereoscopic field is drawn complete from top to bottom. Two refresh cycles are required to draw the complete stereoscopic picture.

The ESV Workstation monitor has a default resolution of 1280 horizontal pixels by 1024 vertical pixels. The pixels are stored in a special memory called the *frame buffer*. In the stereoscopic viewing mode, the video hardware divides the frame buffer into two equal parts. The first 512 scan lines (lines 0 through 511) are used for the first stereoscopic field (*e.g.*, the left-eye view), and the second 512 scan lines (lines 512 through 1023) are used for the second stereoscopic field (*e.g.*, the right eye).

The first 512 horizontal scanlines of frame buffer memory are used for the left-eye view. These 512 scan lines are displayed across the entire surface of the monitor, so that there is now a resolution of 512 vertical pixels, rather than the original 1024. The image now has pixels that are twice as tall as they used to be. While these 512 scanlines are being displayed, the *stereoscopic viewing device* blanks the right lens so that only the left eye can see the left-eye view on the monitor.

The second 512 horizontal scanlines of frame buffer memory are used for the right-eye view. While the right-eye view is being displayed, the *stereoscopic viewing device* blanks the left lens so that only the right eye can see the right-eye view on the monitor. Once again, the right-eye view's 512 scanlines are displayed across the entire surface of the monitor.

Stereoscopic Viewing Device

The CrystalEyes *stereoscopic viewing device* is an electrochemical apparatus which selects and transmits the stereoscopic field displayed on the monitor to the intended eye while blocking the direct vision of the other eye. The device synchronizes to an inter-field time interval signal and alters its state to pass the subsequent field to the intended eye. The stereoscopic viewing device consists of a *liquid crystal shutter* and a *device controller.*

The *liquid crystal shutter* is an alternate-state, electrochemical, optical filter that encodes each stereoscopic field with a unique polarization which is decoded by a like-polarized lens. The liquid crystal shutter consists of a *sheet polarizer, light modulator* and *polarization analyzer.*

The *light modulator* consists of two liquid crystal cells mounted back-to-back. When light passes through the modulator, the modulator changes the phase of one of the components of light with respect to the other, depending on the potential applied to its electrodes. When bonded to a sheet polarizer, its two states produce circularly-polarized light which is either orthogonal or parallel to the sense of each polarization analyzer.

The *polarization analyzer* is a circular polarizer which decodes the circularly-polarized light passing through it in the following manner. The polarization analyzer transmits light of like-handedness, but blocks light of opposite-handedness. A set of polarization analyzers consists of a right-circular analyzer and a left-circular analyzer, and they are mounted, together with the light modulator, in the apparatus which is worn by the user.

The *device controller* synchronizes to the display signal, which indicates the beginning of the inter-field time interval and changes the light modulator state to transmit the next stereoscopic field to the intended eye. The device controller is placed on the monitor and it transmits the display signal, by means of infrared light, to the apparatus which is worn by the user.

The CrystalEyes device displays the pair of left-eye and right-eye views 60 times per second, which means that each eye view is on the screen for only 1/120th of a second.

Stereoscopic Fields

The stereoscopic fields are created by the software application running on the ESV Workstation. Chapter 2 describes stereo implementation using PHIGS, and chapter 3 describes stereo implementation using ES/PSX.

2. PHIGS Stereo Implementation

The ESV Workstation provides a method for creating stereo images through PHIGS. This method includes two separate "screens" in the X server, one for monoscopic applications and one for stereoscopic applications. Stereo screens are established with command line switches at the time the X server is started (See the man pages for **Xesv**, **XGetScreenInfo**, **XScreenWarpByCursor**, **XWarpToScreen**, and **csm**.)

The stereo application is responsible for setting up the stereoscopic fields in the view table and for opening the X connection to the stereo screen. Everything else is handled by the X server.

Note: If you are using the **mwm** window manager, you must use the **-multiscreen** option on the command line when you start **mwm**.

Interface Overview

The ESV Workstation contains a special screen in the X server to support stereoscopic applications. The stereo application must open a window somewhere in the "stereo screen" (**host:0.*x***) where ***x*** is the screen number of a stereo screen. You toggle between screens by moving the cursor off the left or right side of the screen.

The stereo screen is 512 pixels high. If PHIGS graphics are displayed in this window, the system will traverse the graphics structure twice to display both stereoscopic fields.

Using the PHIGS Viewing Model for Stereo

The stereo application must create the left-eye and right-eye stereoscopic fields in the PHIGS view table. These views are used by the system during traversal. To tell the system which stereoscopic field should be used for each eye, a PHIGS **GSE** is provided, which is an extension of the **SET_VIEW_INDEX** element. Rather than indicating a single index in the **GSE**, the application indicates three indices: one for left-eye, one for right-eye, and one for monoscopic viewing.

The left-eye and right-eye views are established in the View Reference Coordinate (VRC) system in VRC coordinates, as shown in figure 2-1. Left and right are established by shifting the Projection Reference Point (PRP) off the VRC z-axis in the x-direction. Note that the viewing frustum is created by passing planes through the PRP and the corners of the viewing window (which is a rectangle on the viewing plane).

The closer the PRP is to the viewing plane, the more severe the perspective angle. The view window rectangle is specified by the **window** field in the **Pview_map3** structure that is passed to **peval_view_map_matrix3**.

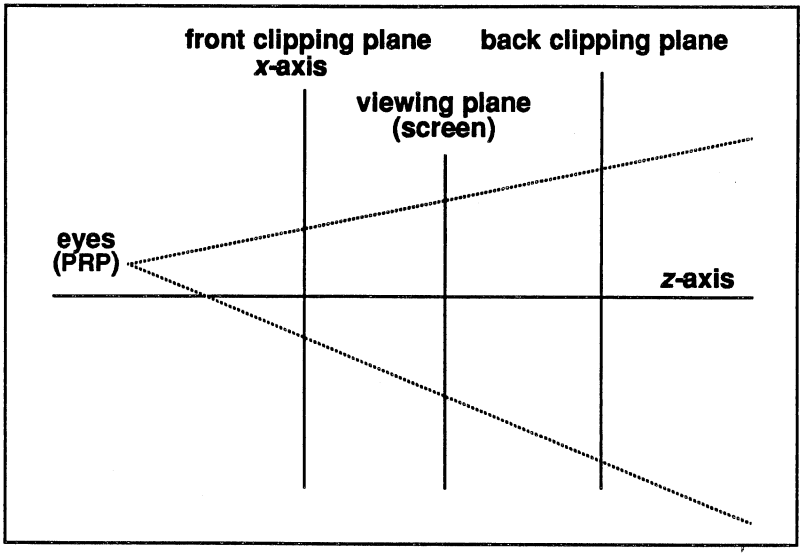


Figure 2-1. View reference coordinate system

- The viewing plane coincides with the surface of the monitor screen, and the viewing window corresponds to the X window on the screen in which the graphics will be displayed. The following parameters should be carefully selected to create the best stereo image:
- PRP *z*-value (the distance between eyes and the monitor screen),
- The size of the X window in which the 3D object is drawn,
- PRP *x*-value (the interocular separation).

In the beginning of the example program in "Appendix B," you will see a constant, **INCHES_TO_VRC**, which allows you to map physical inches to VRC distances. It is critical that real inches are correctly converted to units of VRC space, so that the stereoscopic fields are correctly calculated.

```
/* Stereo params in physical inches */
#define EYE_TO_SCREEN    36.0
#define WINDOW_SIZE      9.0
#define VIEW_RATIO       (EYE_TO_SCREEN / WINDOW_SIZE)
/* Stereo params in VRC units */
```

```
#define VRC_WINDOW_SIZE 4.0
#define INCHES_TO_VRC   (VRC_WINDOW_SIZE / WINDOW_SIZE)
#define VRC_PRP         (EYE_TO_SCREEN * INCHES_TO_VRC)
#define HALF_OCCULAR    (1.25 * INCHES_TO_VRC)
```

Stereo Setup Procedure

1. Create a stereo screen when starting the X server

First, create a stereo screen to hold stereo applications. This is done by using special command line options when the X server is started. There are two options you need to use; the **-nscreens *MxN*** option that creates an array of size ***M*** by ***N*** available screens, and the **-stereoscr *n*** option that indicates which of the screens are stereo screens. For example, if you start the X server directly, you could use

```
% Xesv -nscreens 2x1 -stereoscr 1
```

This creates two screens numbered 0 and 1. Screen number 1 is a stereo screen.

Or, if you start the X server with a call to **xinit**, the command is

```
% xinit -- -nscreens 2x1 -stereoscr 1
```

Many workstations use the **xdm** display manager. In this environment the command line options for the server are set in the **/usr/lib/X11/xdm/Xservers** file. There are many ways this can be done. (See the **xdm** man page for more details.) For example,

```
unix:0.0 local /usr/bin/X11/Xesv -nscreens 2x1 -stereoscr 1
```

2. Start a window manager that supports multiple screens

If you are using the **mwm** window manager, start it using the **-multiscreen** command line option.

```
% mwm -multiscreen &
```

This command should appear in your **.xsession** file if you are using **xdm** or in your **.xinitrc** file if you are using **xinit**. See the **mwm** man page for more details.

3. Start a screen manager to allow you to switch screens

You will probably want to start up a multiscreen manager client that will allow you to switch between the regular screen and the stereo screen if your application doesn't make use of **XScreenWarpByCursor**. (See the **XScreenWarpByCursor** man page for more details.). A simple screen manager is provided with the ESV 2.0 software, called **csm**. (See the **csm** man page for more details.) You may also want to put the following command in your **.xsession** or **.xinitrc** file:

```
% csm
```

4. Open a connection to the stereo screen

The application must first open a connection to the stereo screen. This is done by opening screen **host:0.*x***., where ***x*** is the screen number of a stereo screen. To find out which screens are stereo screens, make a call to the X extensions function **XGetScreensInfo**. The display pointer that is returned with the connection is then used when creating the window that will hold the PHIGS workstation graphics.

```
/* Open the stereo screen for stereo windows. */
    stereoDisplayString = XDisplayString(dpy);
    strptr = rindex(stereoDisplayString, ':');
    XGetScreensInfo(dpy, &screen_info);
    numscreens = screen_info->numofscreens;
    for (i=0; i<numscreens; i++)
    {
if (screen_info->screens[i].screentype == xStereoScreen)
{
      sprintf( (strptr + 1), "0.%d", i);
}
    }

if (!(sdpy = XOpenDisplay(stereoDisplayString) ))
{
      perror("Cannot open display for Stereo screen\n");
      exit(-1);
}
```

5. Open a window on the stereo screen

Do this by creating the window with the stereo screen's display pointer.

6. Open a PHIGS workstation for the stereo window

For example,

```
conn.drawable_id = stereo_win;
popen_ws(stereo_wks, (Pconnid *)
   (&conn),phigs_ws_type_x_drawable);
```

7. Set up the left-eye and right-eye views

Set up the PRPs (one for each eye) in conjunction with the viewing plane. The position of the viewing plane corresponds to the monitor screen. The *z* value of the PRP is the *z*-coordinate of the PRP in VRC space. This distance should directly correspond to the physical distance of your eyes from the screen. The application should translate physical distances into distances in VRC units. Create the left-eye and right-eye viewing matrices and put one each in the view tables of the left-eye and right-eye workstations.

```
/* left eye view */
mapping.proj_ref_point.x = -1.0 * half_occular_dist;
peval_view_map_matrix3(&mapping, &err,
                       view_rep.map_matrix);
pset_view_rep3(stereo_wks, left_view_index, &view_rep);

/* right eye view */
mapping.proj_ref_point.x = half_occular_dist;
peval_view_map_matrix3(&mapping, &err,
                       view_rep.map_matrix);
pset_view_rep3(stereo_wks, right_view_index, &view_rep);
```

5. Put the GSE stereo view index element into the structure

Put the **GSE** stereo view index element into the 3D structure that points to the stereo viewing matrices created in step 4.

```
popen_struct(structure);
pset_view_ind(stereo_view_index);
```

6. Post the structure to the stereo workstation

Post the structure to the stereo workstation to view the 3D object.

```
ppost_struct(stereo_wks, structure, 0.0);
```

3. ES/PSX Stereo Implementation

Selecting a Stereoscopic Model

A stereoscopic viewing model defines a set of 4x4 viewing matrices, one for the left-eye view and one for the right-eye view. The graphics viewing transformations required to produce the proper stereoscopic view for each eye depend on the user's requirements. Most users prefer a "natural" viewing transformation model, where the image is computed as though it were etched on the screen by light rays passing through the screen on the way from a real object to the eye. This "natural" model is preferred because it takes full advantage of the following depth cues to enhance the perception of stereopsis:

- Perspective
- Oblique angle display
- Intensity cueing
- Parallax

The **F:STEREO** function implements the "natural" viewing transformation model described above to take full advantage of the depth cues, and its use is strongly recommended. The **F:STEREO** function provides considerable control by allowing the user to amplify or suppress any of the depth cues. For example, the **F:STEREO** function can produce an orthographic projection for users' applications. Refer to the "Perspective vs. Orthographic Projection" helpful hint for more details.

The left-eye and right-eye 4x4 viewing matrices, generated by the **F:STEREO** function, are computed by sending the desired object viewing parameters to the **F:STEREO** function in a way which includes the viewer and the screen as objects in the scene. These parameters include:

- Viewport size, placement, and aspect ratio
- Location of the viewer with respect to the viewport
- Placement of the viewer's eyes on his face
- Placement of the near and far clipping planes as though they were real objects in the room
- A number to relate the size in data space to the size in room space

The 4x4 viewing matrices can be generated by one of the commands in the following list. Only the **F:STEREO** function, or one of the other commands shown in the list, can be used to create the 4x4 viewing matrices. They must not be mixed in the display structure.

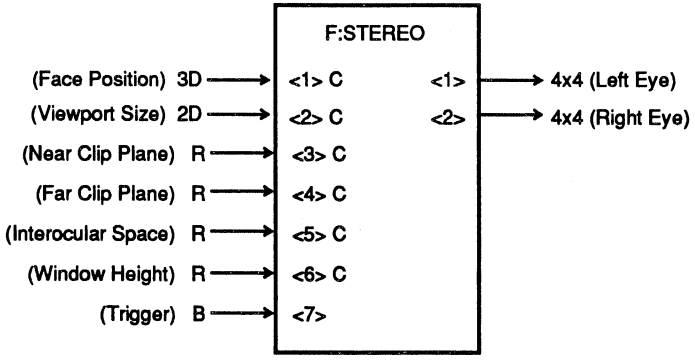
- **F:STEREO** function
- **EYEBACK** command
- **FIELD_OF_VIEW** command
- **MATRIX_4x4** command
- **WINDOW** command

If the **F:STEREO** function is not used, the "Helpful Hints for Implementing Stereoscopic Images" section contains guidelines for creating stereoscopic images.

The F:STEREO Function

Type:

Intrinsic User Function — Miscellaneous



Purpose:

This function produces the left-eye and right-eye matrices for stereoscopic display.

Description:

Inputs

<1> — The (x,y,z) position of the face of the viewer with respect to the origin, which is at the screen in the center of the viewport. z will be negative.

<2> — The (x,y) dimensions of the viewport as measured on the screen.

<3> — The distance from the screen to the near clipping plane. A negative number means the plane is positioned closer to the viewer than the screen.

<4> — The distance from the screen to the far clipping plane. This number can also be negative, but must always be more positive than input <3>.

<5> — The interocular separation of the viewer. This number is typically 2.5 in. (6.35 cm).

<6> — This number represents the vertical height of the viewing pyramid-of-vision, in *data-space* units, at the origin (*i.e.*, window height bottom-to-top).

<7> — This input triggers the function. A Boolean true triggers a perspective stereoscopic view, and a Boolean false triggers an orthographic stereoscopic view.

Outputs

<1> — The 4x4 matrix describing the left-eye view.

<2> — The 4x4 matrix describing the right-eye view.

Notes:

1) Inputs <1> through <5> are specified in room-space coordinates, such as inches or centimeters. Any measuring units are permitted, as long as consistency is maintained. Input <6> uses the coordinate system of the data base.

2) Inputs <1> through <6> are constant input. Messages placed on them are remembered until replaced by new messages. After initial setup is completed, these inputs can be adjusted. The function is then triggered by either a Boolean true for perspective view or a Boolean false for orthographic view on input <7>.

   Orthographic stereoscopic views (see input <7>) are not generally recommended, since the images they produce are in conflict with the stereopsis cues supplied in stereoscopic display. The natural perspective display for the user's viewpoint is computed automatically, based on the other inputs, when input <7> is a Boolean true.

3) The outputs of this function typically connect to nodes of the **MATRIX_4x4** type, which are placed in the left and right branches of a dual display structure. These branches also contain the viewport specification for the left-eye and right-eye viewports. After the eye viewport and viewing matrix are specified, the structure converges to draw the user's scene, being traversed once for each eye.

4) For a single view, output <2> can be disconnected, and the interocular distance of input <5> set to zero.

Example:

The following code is an example of a stereo implementation using the **F:STEREO** function. The **STEREO_TRIGGER** function is used to hold the Boolean true for perspective viewing from one firing to the next.

```
STEREO_EYES := F:STEREO; STEREO_TRIGGER := F:CONSTANT;
CONN STEREO_EYES <1> : <1> Left_eye_object.Mtx;
   { 4x4 left eye view matrix }
CONN STEREO_EYES <2> : <1> Right_eye_object.Mtx;
   { 4x4 right eye view matrix }
CONN STEREO_TRIGGER <1> : <7> STEREO_EYES;
   { sets perspective & triggers}
SEND V3D(0,0,-24) TO <1> STEREO_EYES;
   { initialize face position }
SEND V2D(11,11) TO <2> STEREO_EYES;
   { initialize vp screen size }
SEND -4.0  TO <3> STEREO_EYES;  { near clipping plane }
SEND  4.0  TO <4> STEREO_EYES;  { far clipping plane  }
SEND  2.5  TO <5> STEREO_EYES; {recommended eye spacing }
SEND  2.0  TO <6> STEREO_EYES; { viewing pyramid height }
SEND  TRUE TO <2> STEREO_TRIGGER;
   { arm the trigger function for TRUE perspective }
```

Developing a Stereoscopic Display Structure

The left-eye and right-eye viewing matrices generated by the **F:STEREO** function or some other model must be incorporated into the stereoscopic display structure. To understand this procedure, a discussion of a monoscopic display structure is presented first, followed by a discussion of the corresponding stereoscopic display structure.

Most monoscopic display structures can easily be converted to stereoscopic display structures. Examination of the parts of a monoscopic display structure will help in understanding the creation of a stereoscopic display structure. Figure 3-1 shows the parts of a typical monoscopic image display structure.

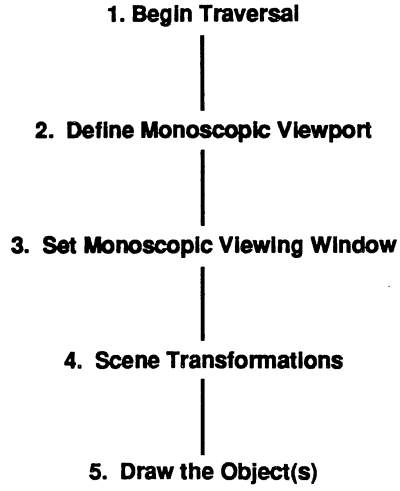


Figure 3-1. Monoscopic image display structure

The five parts of the display structure shown in figure 3-1 have the following purposes:

1) The root of the display structure
2) Define the part of the screen on which the picture will be drawn
3) Place the position of the viewer in the scene at the proper location
4) Move the object(s) around to set the scene
5) Define the shape of the object(s)

Items 4 and 5 can be part of a complex structure, but items 1, 2, and 3 are usually a single node.

The following describes the building of a model to display a cube. The parts of the display structure are outlined. The numbers correspond to figure 3-1. They are in reverse order because the structure is typically defined first at the leaves and finally down to the root.

5. Draw the Object(s)

Initially, the cube is defined at the origin. Two vector lists are created, each defining a square, and translated from the origin by one data-space unit in each direction. The corners are then connected. This is accomplished by the following commands:

```
Cube :=begin_structure
      Translate 0,0,1;
      Vector_list block connected n=5
      1,1 1,-1 -1,-1 -1,1 1,1;
      Translate 0,0,-2;
      Vector_list block connected n=5
      1,1 1,-1 -1,-1 -1,1 1,1;
      Translate 0,0,1;
      Vector_list block separate n=8
      -1,-1,-1 -1,-1,1
      -1, 1,-1 -1, 1,1
      1,-1,-1  1,-1,1
      1, 1,-1  1, 1,1;
End_structure;
```

4. Scene Transformations

The cube defined in the previous step is scaled to quarter-size and randomly rotated. Also, near and far clipping are turned on. The names assigned to the nodes are chosen at random. This is accomplished by the following commands:

```
Rot_x := rotate in x 30 then cube;
Rot_y := rotate in y 10 then rot_x;
Scal := scale 0.25 then rot_y;
Klip := set depth_clipping on then scal;
World := instance of klip;
```

3. Set Monoscopic Viewing Window

The **FIELD_OF_VIEW** command is used to position the viewer for a monoscopic view. This is accomplished by the following command:

```
Mtx := field_of_view 30 front=-1 back=1 then world;
```

2. Define Monoscopic Viewport

For simplicity, a unity viewport is specified. This is accomplished by the following command:

```
Vp := viewport horizontal=-1:1 vertical=-1:1 intensity=0:1
   then mtx;
```

1. Begin Traversal

Displaying the monoscopic image is accomplished by the following command:

```
Display Vp;
```

For the stereoscopic view, a separate view for each eye is required. Figure 3-2 shows the corresponding stereoscopic display structure. Steps 5 and 4 are the same as in the monoscopic view, since the same scene is being viewed with both eyes. The structure is a directed graph, meaning that two or more branches are permitted to grow back together to arrive at the same leaf node.

It is necessary to specify different *y* values for the left and right viewports. The ESV Workstation stereoscopic timing format will cause the left and right viewport images to appear in the same viewing region of the screen.

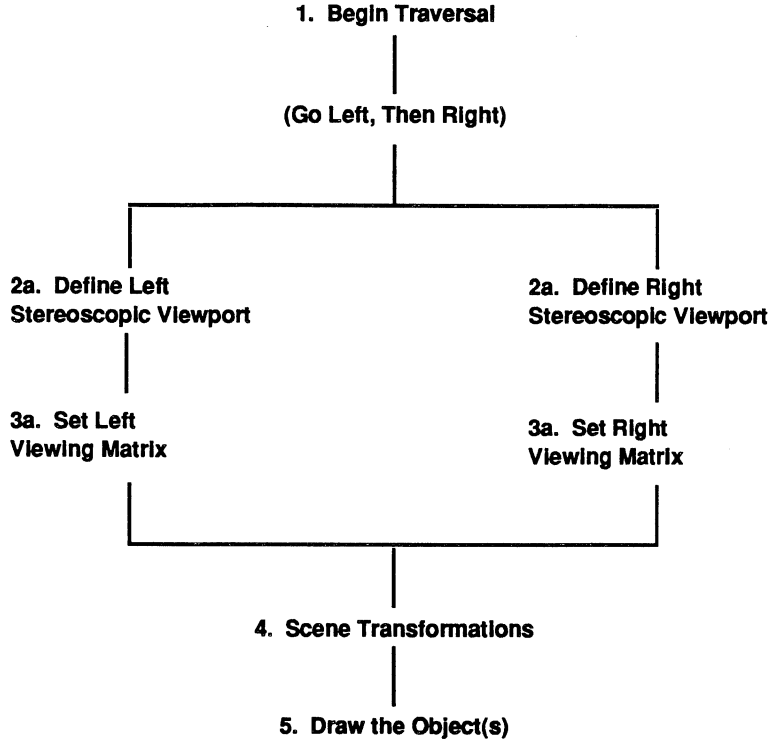


Figure 3-2. Stereoscopic image display structure

5. Draw the Object(s)

Initially, the cube is defined at the origin. Two vector lists are created, each defining a square, and translated from the origin by one data-space unit in each direction. The corners are then connected. This is accomplished by the following commands:

```
Cube :=begin_structure
      Translate 0,0,1;
      Vector_list block connected n=5
      1,1 1,-1 -1,-1 -1,1 1,1;
      Translate 0,0,-2;
      Vector_list block connected n=5
      1,1 1,-1 -1,-1 -1,1 1,1;
      Translate 0,0,1;
      Vector_list block separate n=8
      -1,-1,-1 -1,-1,1
      -1, 1,-1 -1, 1,1
      1,-1,-1  1,-1,1
      1, 1,-1  1, 1,1;
End_structure;
```

4. Scene Transformations

The cube defined in the previous step is scaled to quarter-size and randomly rotated. Also, near and far clipping are turned on. The names assigned to the nodes are chosen at random. This is accomplished by the following commands:

```
Rot_x := rotate in x 30 then cube;
Rot_y := rotate in y 10 then rot_x;
Scal := scale 0.25 then rot_y;
Klip := set depth_clipping on then scal;
World := instance of klip;
```

3a. Set Left Viewing Matrix

Since the left eye needs to be specified as being slightly off-axis from the viewport center, none of the standard eye-position routines (such as **FIELD_OF_VIEW**) can be used with accuracy. Instead, the **F:STEREO** function is used to generate the left-eye matrix. This matrix is supplied to the structure by the following commands:

```
STER := F:STEREO;
CONN STER <1> : <1> Mtx_left;
```

The following values are sent to the **F:STEREO** function, and the function generates a matrix to set the left object viewing area:

- The face of the viewer is 24 inches back from the screen, centered in front of the viewport.
- The viewport size is 14 inches wide by 11 inches high (full-screen).
- The clipping planes are set at 7 inches before and behind the screen.
- The left eye of the viewer is 1.25 inches to the left (*i.e.*, the viewer's interocular separation is 2.5 inches).

- The viewing aperture is 2 data units high at the screen, which also contains the origin at the center of the viewport

All of the above variables are input to the **F:STEREO** function and can be changed, depending on the model. Setting an initial left viewing matrix is accomplished by the following commands:

```
Mtx_left := matrix_4x4
      .60829,  0,       0,       0
      0,       .77419,  0,       0
      -.03168, 0,       -.21544, .17742
      0,       0,       .27419,  .77419   then world;
```

3b. Set Right Viewing Matrix

The right-eye position matrix is the same as left-eye, except that the right eye of the viewer is 1.25 inches to the right. Setting an initial right viewing matrix is accomplished by the following commands:

```
Mtx_right := matrix_4x4
      .60829,  0,       0,       0
      0,       .77419,  0,       0
      .03168,  0,       -.21544, .17742
      0,       0,       .27419,  .77419   then world;
```

The **F:STEREO** function is used to generate the right-eye matrix. This matrix is supplied to the structure by the following command:

```
CONN STER <2> : <1> Mtx_right;
```

2a. and 2b. Define Left and Right Stereoscopic Viewports

Each master stereoscopic viewport provides optimal stereoscopic viewing on the ESV Workstation. The master stereoscopic viewport determines the part of the screen that is available for stereoscopic viewing and the eye that sees it. These viewports are defined by the following commands:

```
left_vp := viewport horizontal=-1:1
      vertical=.00390625:.99609375 intensity=1:1;
right_vp := viewport horizontal=-1:1
      vertical=-.99609375:-.00390625 intensity=1:1;
```

These horizontal, vertical, and intensity values should be used exactly as shown in any stereoscopic application program. Dividing the screen into menu viewports and object display viewports can be done after the master viewports are defined. The ESV Workstation stereoscopic timing format will cause the left and right viewport images to appear in the same region of the screen.

1. Begin Traversal

Displaying the data structure and dividing the trunk into two branches is accomplished by the following commands:

```
Trunk := instance of vp_left,vp_right;
Display trunk;
```

Stereoscopic Viewports

The first element of the left branch of the stereoscopic data structure should be the master stereoscopic viewport for the left-eye view, defined as follows:

```
left_vp := viewport horiz= -1:1
      vert= .00390625 : .99609375 intens = 1:1;
```

Similarly, the first element of the right branch of the stereoscopic data structure should be the master stereoscopic viewport for the right-eye view, defined as follows:

```
right_vp := viewport horiz= -1:1
      vert= -.99609375: -.00390625 intens= 1:1;
```

Once the master stereoscopic viewports for the left-eye and right-eye branches are defined, the object and menu viewports can be defined below them with any shape, size, or intensity. All subsequent viewports can be defined as they would be in a monoscopic structure. Objects which are to be displayed for both eyes must be instanced under both the left-eye and right-eye master viewports.

The following code shows the implementation of viewports for the object display area. Note that the intensity is specified using the **SET INTENSITY** command.

```
SEND 'GO' TO <1> STEREO_CONFIG$;
   { trigger the system stereo config }
....
....
....
Left_stereo_view := begin_s
   { use this vp definition as is }
LEFT_VP := VIEWPORT HORIZ= -1:1
   VERT= .00390625 : .99609375 INTENS = 1:1;
INST STEREO_PICK_CONFIG;
   { stereo pick location and cursor config}
inst Left_eye_object;
   { left eye object display structure }
....
end_s;
Right_stereo_view := begin_s
   { use this vp definition as is }
```

```
RIGHT_VP := VIEWPORT HORIZ= -1:1
   VERT= -.99609375 : -.00390625 INTENS= 1:1;
inst Right_eye_object;
   { right eye object display structure }
....
end_s;
Left_eye_object := begin_s
Clip := set depth_clipping off;
   { fkeys network input point }
viewport horiz = -.8 : .8 vert = -1:1;
   { use any desired values }
set intensity on 0:1;
border_color := set color 120,.4 then border;
{ F:STEREO function input point. Correct stereo values will
   be supplied to the matrix when the STEREO_EYES function
   is triggered }
Mtx := matrix_4x4 1,0,0,0
0,1,0,0
0,0,1,0
0,0,0,1;
....
....
end_s;
Right_eye_object := begin_s
Clip := set depth_clipping off;
   { fkeys network input point }
viewport horiz = -.8 : .8 vert = -1:1;
set intensity on 0:1;
border_color := set color 120,.4 then border;
{ F:STEREO function input point. Correct stereo values will
   be supplied to the matrix when the STEREO_EYES function
   is triggered }
Mtx := matrix_4x4 1,0,0,0
0,1,0,0
0,0,1,0
0,0,0,1;
....
....
end_s;
```

Items such as menu text and borders, which will not be displayed with stereoscopic separation, can be defined in both the left-eye and right-eye branches or, optionally, they can be defined in only one branch. If a menu is defined only in the left branch, it will be visible only to the left eye. This can be a desirable effect, depending on the viewer preference. Items which are to be displayed for both eyes with no stereoscopic separation should be instanced

below the master viewport **LEFT_VP** or **RIGHT_VP** but above the **F:STEREO** matrix. The border in the previous code example is used in this way.

Picking Implementation

Stereoscopic application programs which implement picking must reconfigure the ESV Workstation system pick location and cursor definitions. This is accomplished by issuing the following command when the program enters the stereoscopic viewing mode:

```
send 'go' to <1>stereo_config$;
```

This command reconfigures the ESV Workstation to provide a tablet-driven cursor which picks accurately on both menu items and 3D stereoscopic objects. It also configures the correct system viewport and stereoscopic timing format.

An application program uses the new picking/cursor configuration by declaring a single instance of **STEREO_PICK_CONFIG** immediately after the **LEFT_STEREO_VIEWPORT** definition. Picking accuracy for 3D objects is achieved by restricting the cursor and locations that can be picked to the left branch of the stereoscopic structure. The **PICK IDENTIFIERS** and **SET PICKING ON** nodes in the left branch are the only ones that will ever actually result in a pick. Those on the right branch will have no effect since the cursor can never reach the locations in the right branch. The following example shows picking implementation for the **FKEY** menus.

```
SEND 'GO' TO <1> STEREO_CONFIG$;
   { trigger the system stereo config }
....
....
Left_stereo_view := begin_s
LEFT_VP := VIEWPORT HORIZ= -1:1
   VERT= .00390625 : .99609375 INTENS = 1:1;
INST STEREO_PICK_CONFIG;
   { stereo pick location and cursor config}
inst Left_eye_object;
   { left eye object display structure - pickable}
inst fkey_menu;      { left eye on screen menu - pickable }
....
end_s;
Right_stereo_view := begin_s
RIGHT_VP := VIEWPORT HORIZ= -1:1
   VERT= -.99609375 : -.00390625 INTENS= 1:1;
inst Right_eye_object;
   { right eye object display structure - pickable}
inst fkey_menu;    { right eye on screen menu - not pickable}
```

```
....
end_s;
fkey_menu := begin_s
menu_vp := view horiz= -1.0:-.85 vert= -1:1;
   { can use any values }
character scale .25,.05;
if bit 4 on;                    { don't display if menus off }
inst menu_outline;
set pick identifier= menu_labels;
set picking on;                  { picks on left branch only }
inst fkey_labels;
end_s;
```

Cursor Implementation

The stereoscopic system cursor defined by **STEREO_CONFIG$** is a white **X** with an open green box at its center. Picking is accomplished by aligning the point and the open box. This cursor makes picking easier for stereoscopic objects, particularly objects with a large *z*-translation. Picking on 3D stereoscopic objects will be accurate as long as the point being picked can be visually distinguished from surrounding points.

The shape and color of the stereoscopic cursor can be modified by the application program with some cautions and restrictions. Changes made to the stereoscopic cursor definition will be permanent until the system is rebooted, as long as the names in the system cursor structure are not redefined to instance user's structure names. If a system cursor name is redefined to instance a structure with a user's name, those parts of the cursor will be subject to initialization commands, which can result in some or all of the stereoscopic cursor definition being lost. For example, the stereoscopic system cursor is defined as follows:

```
configure a;
STCURSOR1 := set inten on 1:1 then STCURSOR_PARTS1;
STCURSOR_PARTS1 := inst STCURSOR_CROSS1,STCURSOR_SQUARE1;
STCURSOR_CROSS1 := set color 0,0 then STC_CROSS_VECS1;
STCURSOR_SQUARE1 := set color 240,.8 then STC_SQUARE_VECS1;
STC_SQUARE_VECS1:=vec n=5 -.015,-.015 -.015,.015 .015,.015
   .015,-.015 -.015,-.015;
STC_CROSS_VECS1:=vec sep n=5 -.035,-.035 -.015,-.015
   .015,.015  .035,.035 .035,-.035  .015,-.015 -.015,.015 -
   .035,.035;
SEND V2D(.015,.015) TO <2> STEREO_PICK_LOCATION$1;
finish configuration;
```

An application can safely and permanently change the outline of the square center box to full intensity red by entering the following command:

```
STCURSOR_SQUARE := SET COLOR 120,1 THEN STC_SQUARE_VECS;
```

This change to the stereoscopic system cursor will be retained until it is redefined, or the system is rebooted. This will be true even if the system is switched in and out of the stereoscopic viewing mode.

If an application redefines the stereoscopic cursor color as follows:

```
STCURSOR_SQUARE := INST MY_USER_COLOR;
MY_USER_COLOR := SET COLOR 120,1 THEN STC_SQUARE_VECS;
```

then any initialization command would destroy **MY_USER_COLOR**, leaving:

```
ST_CURSOR_SQUARE := NIL;
```

A third party application that modifies the stereoscopic cursor should recreate its original definition before terminating.

When an application program exits from the stereoscopic viewing mode, the following command should be entered to restore the system to the normal monoscopic cursor and picking:

```
send 'go' to <1>restore_config$;
```

The Stereo Viewing Option supports only the update rate cursor.

Configuring the ESV Workstation for Stereoscopic Viewing

The default stereo network and display structure discussed here are contained in the configuration file **stcfg.dat**, which, if in the ES/PSX configuration directory, are read at start-up time.

The first step in viewing a stereoscopic image is to put the ESV Workstation into the stereoscopic viewing mode. This is accomplished by entering the following command:

```
send 'go' to <1>stereo_config$;
```

This command causes the ESV Workstation system configuration to change from the monoscopic viewing mode to the stereoscopic viewing mode. The system reconfiguration includes the following steps:

- Enlarging the size of the system viewport **VPF1$**, so that the full screen is available for stereoscopic viewing,
- Setting the correct stereoscopic timing format for the **PS390ENV** function,
- Restructuring the system definitions of the pick location and cursor.

In the stereoscopic viewing mode, the text on the terminal emulator will become larger, and it will appear bright only to the right eye when the viewing glasses are worn and the device controller is turned on. If the bright image is in the left eye, the stereoscopic mode must be reversed by toggling the STEREO/PSEUDO switch on the device controller.

To return to the monoscopic viewing mode, enter the following command:

```
send 'go' to <1>restore_config$;
```

Alternative System Configuration

Some applications do not require the stereoscopic cursor and picking. The following set of commands can be used in place of the **STEREO_CONFIG$** and **RESTORE_CONFIG$** functions to set up the system viewport and timing format. To enter the stereoscopic viewing mode, enter the following commands:

```
configure a;
send m2d(-.997,.997 -1,1) to <1>vpf1$;
finish configuration;
send fix(2) to <5>ps390env;
   { "fix(3)" may be used for special low-precision stereo-
   scopic mode. }
send true to <1>ps390env;
```

These commands must be executed as a single file. They set the ESV Workstation system viewport so that the full screen is available for stereoscopic viewing, and they set the ESV Workstation timing format to merge the left-eye and right-eye views into a single, full-screen stereoscopic image. Note the alternative, low-precision format available. This format provides the same resolution in both the *x* and *y* directions.

To return to the monoscopic viewing mode, enter the following commands:

```
configure a;
send m2d(-1,1 -1,1) to <1>vpf1$;
finish configuration;
send fix(0) to <5>ps390env;
send true to <1>ps390env;
```

Helpful Hints for ES/PSX Stereo Implementation

Toggling Between Monoscopic and Stereoscopic Viewing Modes

Enter the following command to go from the stereoscopic viewing mode to the monoscopic viewing mode:

```
send 'go' to <1>restore_config$;
```

Enter the following command to go from the monoscopic viewing mode to the stereoscopic viewing mode:

```
send 'go' to <1>stereo_config$;
```

These commands insure that the proper timing format is sent to input <5> of the **PS390ENV** function, and they also activate the stereoscopic picking and cursor configuration.

Update Rate Cursor

The stereoscopic system cursor is an update rate cursor. The monoscopic system default cursor is also an update rate cursor. The stereoscopic viewing configuration does not support refresh rate cursors, sometimes called hardware cursors.

If an application uses the refresh rate cursor, the system must be returned to the update rate cursor before entering the stereoscopic viewing mode. Enter the following commands to restore the update rate cursor:

```
send fix(0) to <4>ps390env;
send true to <1>ps390env;
```

Line Resolution in Stereo

Images on the ESV Workstation are normally displayed with a screen resolution of 1024 horizontal scanlines, each containing 1280 pixels. This provides a picture with an aspect ratio of 5:4.

In stereoscopic mode, only half the scanlines are seen by the left eye, and the other scanlines are seen by the right eye. This effectively halves the screen resolution in the vertical axis, while maintaining the full resolution in the horizontal axis. Since a pixel is now effectively twice as high as it is wide, lines drawn with angles near the horizontal tend to appear thicker, while near-vertical lines remain crisp.

While most users prefer to maintain the high resolution in the horizontal axis wherever possible, some users require that lines at all angles be drawn with constant thickness, even though this means giving up available horizontal resolution. For these users, a special uniform-resolution stereoscopic format is provided, which consists of only 640 pixels along the horizontal axis. It may be invoked as follows.

1) For stereo commands involving the **STEREO_CONFIG$** function, a variable called **STEREO_KIND$** is provided. It contains a default value of 2, which selects mixed-resolution stereo. It must be modified before invoking the **STEREO_CONFIG$** function, as follows.

   - For uniform-resolution stereo, enter the following commands:

     ```
     store fix(3) in stereo_kind$;
     send 'go' to <1>stereo_config$;
     ```

   - To return to mixed-resolution stereo, enter the following commands:

     ```
     store fix(2) in stereo_kind$;
     send 'go' to <1>stereo_config$;
     ```

   - The user who wishes to change between the two stereo modes must first return to monoscopic mode by entering the following command:

     ```
     send 'go' to <1>restore_config$;
     ```

   Since **STEREO_KIND$** is a variable, it will remember its value. Thus, the above **STORE** commands need only be executed once.

2) For users controlling stereo operations by means of the **PS390ENV** initial function instance (see "Alternative System Configuration"), the following changes apply.

   - For uniform-resolution stereo, enter the following commands:

     ```
     send fix(3) to <5>ps390env;
     send true to <1>ps390env;
     ```

   - For mixed-resolution stereo, enter the following commands:

     ```
     send fix(2) to <5>ps390env;
     send true to <1>ps390env;
     ```

   - To switch between stereo resolutions, first return to the monoscopic mode by entering the following commands:

     ```
     send fix(0) to <5>ps390env;
     send true to <1>ps390env;
     ```

Image Distortion

If an image is distorted in one direction, the incorrect viewport size measurements, clipping plane values, or window height are being sent to the **F:STEREO** function. Refer to the "Clipping Planes" and "Object Viewing Region" helpful hints for information about converting data space coordinates to the room space units needed for input to the **F:STEREO** function.

Depth Cues

Special applications may require adjusting or even deleting one or more of the following depth cues:

- Perspective
- Oblique angle display
- Intensity cueing
- Parallax

For example, in some modeling applications, measurements must be made using the tablet and cursor. Because perspective will foreshorten dimensions with depth, making them invalid for these applications, an orthographic image must be selected while measurements are made, even though the view will be distorted. Sending a Boolean false to input <7> of the **F:STEREO** function will produce an orthographic view.

Adjusting Perspective

Since perspective is determined by the geometry of the screen and the user's eyes, the amount of perspective effect can be changed by multiplying both the inter-eye spacing value of **F:STEREO** input <5> and the elements of the viewer position vector on input <1> by a constant.

For example, doubling both the *z*-distance of the user from the screen and the interocular spacing reduces the perspective of the image to half of its former value.

Perspective vs. Orthographic Projection

Most applications which use orthographic projections for the monoscopic viewing mode should use the **F:STEREO** function perspective view for the stereoscopic viewing mode. Many monoscopic applications use an orthographic projection to prevent distortions in relative object size. The perspective view generated by the **F:STEREO** function does not produce distortions in apparent object size as the object is moved toward the viewer. An orthographic view in the stereoscopic viewing mode can appear distorted to the viewer, since the eye-brain system expects perspective depth cues when looking at 3D objects.

Ghosting

Ghost lines are caused by the blending of the left-eye and right-eye images on the liquid crystal shutter, and they can be a problem to users. The eye-brain system can be trained to ignore ghost lines when they are near enough to the real line that they are not confused with another, unrelated, line. However, in a very dense picture this can be impossible.

To alleviate the problem, the parallax cue can be reduced, causing the ghost lines to move toward their visible counterparts. This is done on input <5> of the **F:STEREO** function by specifying a smaller number for the interocular spacing. The effect is to reduce the stereopsis in the image, so that its volume seems to decrease, while leaving perspective untouched. Although artificial, it can be a good compromise between ghosting and no stereoscopic effect.

Use darker backgrounds to increase the dynamic range. Since ghosting is color dependent, use red whenever possible and avoid using white.

Clipping Planes

Placement of the clipping planes in room space can be made directly as inputs to the **F:STEREO** function. To place these planes in data space, the user should remember that there is a ratio between the viewport height in data space, input <6> of the **F:STEREO** function, and the same measurement in room space, the *y*-component of input <2>. Remembering this ratio permits the conversion of any position value, including clipping planes, between room space and data space.

For example, consider the following data:

- 20 cm = room space viewport height (*y*-component of input <2>)
- 2 units = data space window height (input <6>)
- –1.2 units = data space near plane *z*-position (input <3>)
- 2.5 units = data space far plane *z*-position (input <4>)

First, find the ratio of room space to data units:

$$ratio = \frac{20\ cm_{room\ space}}{2\ units_{data\ space}} = 10\ cm\ of\ room\ space\ per\ data\ space\ unit$$

Next compute the room space clipping plane values which correspond to the -1.2 and 2.5 data space inputs to the **F:STEREO** function:

$$near\ clipping\ plane_{cm\ of\ room\ space} =$$

$$-1.2\ units_{data\ space} * ratio = -12\ cm_{room\ space}$$

$$far\ clipping\ plane_{cm\ of\ room\ space} =$$

$$2.5\ units_{data\ space} * ratio = 25\ cm_{room\ space}$$

When the room space clipping plane coordinates are known, rearrange the equation to compute the correct data space coordinates for the input to the **F:STEREO** function:

$$near\ clipping\ plane_{data\ space} = \frac{near\ clipping\ plane_{room\ space}}{ratio}$$

$$far\ clipping\ plane_{data\ space} = \frac{far\ clipping\ plane_{room\ space}}{ratio}$$

When using the **F:STEREO** viewing model, the near clipping plane should not go beyond the eye point. If the eye point is set at 60 cm from the screen, then the near clipping plane cannot be less than –60 cm (room space).

Object Viewing Region

The visible region of data space produced by the **WINDOW** command can be positioned anywhere with relation to the object, while the visible region of data space produced by the **F:STEREO** function viewing matrix is always centered about the origin. The **F:STEREO** viewing region can be modified by including a translation node in the display structure where it joins after the left-eye and right-eye branches.

The x and y values at the point of the translation should be the average of the left and right (for x) and the bottom and top (for y) boundaries of the corresponding **WINDOW** command.

The values for the translation (x_T, y_T) can be computed from the average of the boundaries for the corresponding **WINDOW** command as follows:

$$y_T = -\frac{x_{W(left)} + x_{W(right)}}{2}$$

$$y_T = -\frac{y_{W(bottom)} + y_{W(top)}}{2}$$

For example, assume that a square window is desired, ranging from 300 to 700 in *x*, and from 0 to 400 in *y*. Input <6> of **F:STEREO** (window height) would be 400, input <2> of **F:STEREO**, the viewport size measured at the screen, would be square, and the translation node at the joining of the left-eye and right-eye branches of the structure would contain the vector (-500,-200,0), where -500 is the negative *x*-average of 300 and 700, and -200 is the negative *y*-average of 0 and 400.

Registration

The left-eye and right-eye views in a stereoscopic model are registered when they appear to be exactly superimposed on each other at the plane of the screen. If the **F:STEREO** function is used with the correct left-eye and right-eye viewports, the views will be registered.

In some stereoscopic models not using the **F:STEREO** function, registration can be a problem. In this case, the left-eye and right-eye views are horizontally shifted with respect to each other. This usually occurs when one of the viewports is moved in *x* with respect to the other, creating the problem of misaligned horizontal clipping. The images involved are usually orthographic rather than perspective.

Stereo in the X Environment

When stereo mode is entered, all other X windows are obscured by the stereo window. ES/PSX will grab the X pointer device to prevent it from remaining within some other window. If the keyboard and any other device focus is following the pointer, those devices will be focused on the stereo window.

If the keyboard and/or other devices have been explicitly focused in another window by a window manager, it is advisable to request ES/PSX to grab these other devices also so that they will not be inadvertently unavailable when entering stereo mode. ES/PSX can be instructed to grab all devices when entering stereo mode by specifying the following in an X defaults file:

```
...main.stereoGrab: on
```

You can also grab all devices by entering **-stereoGrab** on the command line.

Note: The only way that keyboard focus may be forced to remain in a window from which you are typing commands to ES/PSX is to explicitly focus the keyboard within the window before going into stereo mode. However, this will likely make the keyboard unavailable for use within the stereo window.

Emergency Exit from Stereoscopic Viewing Mode

During program development, users may find themselves switching into the stereoscopic viewing mode without providing a means to switch back to the monoscopic viewing mode. This generally happens because the entire screen is allocated for display of the stereoscopic image when switching to the stereoscopic viewing mode, thus obscuring all other windows. The pointer is grabbed by the stereo window rather than remaining in any other window it may have previously been in.

ES/PSX may be forced out of the stereoscopic viewing mode in all cases by the following escape sequence. While holding down the CONTROL and SHIFT keys on the ESV Workstation keyboard, simultaneously press buttons 1, 2, and 3 on the mouse. ES/PSX will switch back to the monoscopic viewing mode, and a value of fix(0) will be sent to input <5> of the function instance **PS390ENV**, to prevent that function from inadvertently entering the stereoscopic viewing mode when activated for other purposes. If the stereoscopic viewing mode was initially entered via the **STEREO_CONFIG$** function, the cursor must still be restored to its proper form and range by the following command:

```
send true to <1>restore_config$;
```

Helpful Hints for Viewing Stereoscopic Images

Lighting

The CrystalEyes stereoscopic viewing device is designed to work best in low light conditions. Fluorescent light is detrimental to the quality of the picture. The frequency of the fluorescent light can cause some flickering that will not be seen when using incandescent light.

Viewing Angle

The stereoscopic effect can be lost if the viewing angle is too great. A large viewing angle will also reduce the dynamic range.

A. Glossary

accommodation. The focusing of the image of an object on the retina by one eye. This process is controlled by a different set of muscles than the process of convergence. *See also* ***convergence***.

convergence. The focusing of both eyes on the same point of an object. This process is controlled by a different set of muscles than the process of accommodation. *See also* ***accommodation***

depth cue. An operation that imparts the illusion of depth to an image. Depth cues allow a user to perceive 3D on a 2D screen.

disparity. The lateral displacement of an object's image on the screen which is produced because the eyes view the object from two different viewpoints. When viewing an object at infinite distance, the disparity is equal to the interocular separation. When viewing an object at the screen, the disparity is equal to zero. *See also* ***parallax***.

dynamic range. A measure of the relative luminance of an information element when transmitted by the stereoscopic viewing device in its transmissive state, and when blocked by the stereoscopic viewing device in its occluding state. The formula and methodology employed to determine the dynamic range is as follows:

$$\textit{dynamic range} = \frac{\textit{intensity through open shutter}}{\textit{intensity through closed shutter}}$$

ghosting. The blending of left-eye and right-eye images on the liquid crystal shutter due to the transition time between the transmissive and occluding states of the shutter.

interocular separation. The separation between the eyes. This distance is approximately 2-1/2 inches, but varies between individuals.

parallax. The apparent change in the direction of an object when viewed from two different viewpoints. When viewing an object at an infinite distance, the parallax is equal to zero. *See also* ***disparity***.

stereoscopic field. A view of a picture from the perspective of one eye.

stereopsis. The process which occurs in the brain where two 2D images are combined to form one 3D image.

stereoscopy. The science of viewing in three dimensions.

viewing angle. The angle at the apex of the viewing pyramid (eye point of the viewer) used to define a perspective viewing area.

B. PHIGS Stereo Example Program

This program can be found in **/usr/people/fstest/demo/stereo_demo.c**.

```
/*****************************************************************************
 *
 *    stereo_demo.c
 *
 *    A Canonical Stereo Program to prototype stereo on ESV, release 2.0
 *
*****************************************************************************/

#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xatom.h>
#include <phigs/phigs.h>
#include <phigs/esgdp.h>
#include <extensions/XMultiScreen.h>
#include <extensions/XInput.h>

extern int errno;

char *ProgramName;

#define  BLACK     0
#define  WHITE     1
#define  RED       2
#define  GREEN     3
#define  BLUE      4
#define  YELLOW    5
#define  CYAN      6
#define  MAGENTA   7
                        /* Stereo params in physical inches */
#define EYE_TO_SCREEN   36.0
#define WINDOW_SIZE     9.0
#define VIEW_RATIO      (EYE_TO_SCREEN / WINDOW_SIZE)
                        /* Stereo params in VRC units */
#define VRC_WINDOW_SIZE 4.0
#define INCHES_TO_VRC   (VRC_WINDOW_SIZE / WINDOW_SIZE)
#define VRC_PRP         (EYE_TO_SCREEN * INCHES_TO_VRC)
#define HALF_OCCULAR    (1.25 * INCHES_TO_VRC)
```

```
                        /* Dial handling */
#define XROT_DIAL       0
#define YROT_DIAL       1
#define ZROT_DIAL       2
#define SCALE_DIAL      3
#define XTRAN_DIAL      4
#define YTRAN_DIAL      5
#define ZTRAN_DIAL      6
#define DIALSCALE       330.0
#define MAXDIALS        8
#define CHARS_PER_DIAL  8
#define SPACEKEYSYM     32


#define FRONT_BOTTOM_LEFT     {-0.5, -0.5,  0.5}
#define FRONT_BOTTOM_RIGHT    { 0.5, -0.5,  0.5}
#define FRONT_TOP_LEFT        {-0.5,  0.5,  0.5}
#define FRONT_TOP_RIGHT       { 0.5,  0.5,  0.5}
#define BACK_BOTTOM_LEFT      {-0.5, -0.5, -0.5}
#define BACK_BOTTOM_RIGHT     { 0.5, -0.5, -0.5}
#define BACK_TOP_LEFT         {-0.5,  0.5, -0.5}
#define BACK_TOP_RIGHT        { 0.5,  0.5, -0.5}

#define FRONT     { FRONT_TOP_RIGHT,    FRONT_TOP_LEFT, \
                    FRONT_BOTTOM_LEFT,  FRONT_BOTTOM_RIGHT}
#define BACK      { BACK_TOP_LEFT,      BACK_TOP_RIGHT, \
                    BACK_BOTTOM_RIGHT,  BACK_BOTTOM_LEFT}
#define LEFT      { FRONT_TOP_LEFT,     BACK_TOP_LEFT, \
                    BACK_BOTTOM_LEFT,   FRONT_BOTTOM_LEFT}
#define RIGHT     { FRONT_TOP_RIGHT,    FRONT_BOTTOM_RIGHT, \
                    BACK_BOTTOM_RIGHT,  BACK_TOP_RIGHT}
#define TOP       { FRONT_TOP_RIGHT,    BACK_TOP_RIGHT, \
                    BACK_TOP_LEFT,      FRONT_TOP_LEFT}
#define BOTTOM    { FRONT_BOTTOM_RIGHT, FRONT_BOTTOM_LEFT, \
                    BACK_BOTTOM_LEFT,   BACK_BOTTOM_RIGHT}

                    /* Flags for current video state */
#define MONO    0
#define STEREO  1
int             mode_flag   = MONO;

static   Ppoint3  cube[6][4]  = { FRONT, BACK, LEFT, RIGHT, TOP, BOTTOM };
Pint              mono_wks    = 1;
Pint              stereo_wks  = 2;
```

```
Pint        structure          = 1;
Pint        mono_view_index    = 1;
Pint        left_view_index    = 1;
Pint        right_view_index   = 2;
Pfloat      priority           = 1.0;
Pint        rotate_elem, translate_elem;
int         winx, winy, winw, winh;
int         mono_winx, mono_winy, mono_winw, mono_winh;
Window      mono_win, mono_rootwin, stereo_win, stereo_rootwin;
Display     *dpy, *sdpy;
GContext    gc;
GC          gc_black;
int         text_x, text_y;
float       h_to_w_aspect;
Pfloat      half_occular_dist = HALF_OCCULAR;

                        /* Globals for device input stuff */
XDevice     *tablet            = NULL;
XID         tablet_id          = 0;
XDevice     *dials             = NULL;
XID         dials_id           = 0;
XDeviceInfo *devices           = NULL;
int         ndevices           = 0;
int         DeviceMotion       = -1;
int         DevicePress        = -1;
int         DeviceRelease      = -1;
int         change[10];
Pmatrix3    xrotmat, yrotmat, zrotmat, scalemat, concat1, concat2,
            concat3;
Pmatrix3    currmatrix, tranmatrix;
Pview_map3  mapping;
Pview_rep3  view_rep;
int         mono_fd, stereo_fd;

Window XCreateWindow();


/*****************************************************************************
 *
 *    identity_matrix
 *
 *    Make an identity matrix.
 *
 *****************************************************************************/
```

```
void  identity_matrix( matrixit )
      Pmatrix3   matrixit;
{
   int i, j;

   for ( i = 0 ; i < 4 ; i++)
   {
      for ( j = 0 ; j < 4 ; j ++ )
      {
         if (i == j )
            matrixit[i][j] = 1.0;
         else
            matrixit[i][j] = 0.0;
      }
   }
}                        /* End of identity_matrix */


/*********************************************************************
 *
 *    usage
 *
 *********************************************************************/
usage ()
{
   fprintf (stderr, "usage:  %s [-options ...]\n\n", ProgramName);
   fprintf (stderr, "where options include:\n");
   fprintf (stderr, "    -display host:dpy  X server to use\n");
   fprintf (stderr, "    -geometry geom     geometry of window\n");
   fprintf (stderr, "-  b                   use backing store\n");
   fprintf (stderr, "    -fg color          set foreground color\n");
   fprintf (stderr, "    -bg color          set background color\n");
   fprintf (stderr, "    -bd color          set border color\n");
   fprintf (stderr, "    -bw width          set border width\n");
   fprintf (stderr, "\n");
   exit (1);
}

/*********************************************************************
 *
 *    doPhigsStuff
 *
 *********************************************************************/
doPhigsStuff(dpy, sdpy)
```

```
    Display     *dpy;
    Display     *sdpy;
{
  static Ppoint                 xyzpoints[] = {{0.1, 0.1}, {0.9, 0.9}};
  Pxphigs_info                  xinfo;
  Pconnid_x_drawable            conn;
  Ppoint                        text_pt;
  Pint                          part = 1;
  Ppoint3                       rep;

  int                           i, err;
  float                         deg;
  Ppoint3                       tran, vrp;
  Pvec3                         vpn, vup;
  Plimit                        wks_viewport, wks_window;
  Pmatrix3                      rotmat, comp_mat, tran1;
  Pgse_stereo_view_indices      stereo_views;
  Pgse_data                     gse_struct;
  unsigned long                 xinfo_mask;
  Ppoint_list3                  point_list;

  xinfo.display                 = dpy;
  xinfo.rmdb                    = NULL;
  xinfo.appl_id.name            = NULL;
  xinfo.appl_id.class           = NULL;
  xinfo.args.argc_p             = NULL;
  xinfo.args.argv               = NULL;
  xinfo.flags.no_monitor        = 1;
  xinfo.flags.force_client_SS= 0;

  xinfo_mask = PXPHIGS_INFO_DISPLAY | PXPHIGS_INFO_FLAGS_NO_MON;
  popen_xphigs((char*)NULL, PDEF_MEM_SIZE, xinfo_mask, &xinfo);

  conn.display          = sdpy;
  conn.drawable_id      = stereo_win;
  popen_ws(stereo_wks, (Pconnid *) (&conn), phigs_ws_type_x_drawable);
  SensitizeWindow(sdpy, stereo_win);
  XSync(sdpy, 0);
  pset_hlhsr_mode(stereo_wks, PHIGS_HLHSR_MODE_ZBUFF);
  conn.display          = dpy;
  conn.drawable_id      = mono_win;
  popen_ws(mono_wks, (Pconnid *) (&conn), phigs_ws_type_x_drawable);
  SensitizeWindow(dpy, mono_win);
  XSync(dpy, 0);
```

```
pset_hlhsr_mode(mono_wks, PHIGS_HLHSR_MODE_ZBUFF);

wks_viewport.x_min = 0.0;
wks_viewport.x_max = winw;
wks_viewport.y_min = 0.0;
wks_viewport.y_max = winh;
pset_ws_vp(stereo_wks, &wks_viewport);

wks_viewport.x_max = mono_winw;
wks_viewport.y_max = mono_winh;
pset_ws_vp(mono_wks, &wks_viewport);

wks_window.x_min = 0.0;
wks_window.x_max = 1.0;
wks_window.y_min = 0.0;
wks_window.y_max = 1.0;
pset_ws_win(stereo_wks, &wks_window);
pset_ws_win(mono_wks, &wks_window);

                    /* Set Background color; entry 0 of table. */
rep.x = 0.50;
rep.y = 0.50;
rep.z = 0.50;
pset_colr_rep(stereo_wks, 0, &rep );
pset_colr_rep(mono_wks, 0, &rep );

rep.x = 1.0;
rep.y = 0.0;
rep.z = 0.0;
pset_colr_rep(stereo_wks, RED, &rep );
pset_colr_rep(mono_wks, RED, &rep );

rep.x = 0.0;
rep.y = 1.0;
rep.z = 0.0;
pset_colr_rep(stereo_wks, GREEN, &rep );
pset_colr_rep(mono_wks, GREEN, &rep );

rep.x = 0.0;
rep.y = 0.0;
rep.z = 1.0;
pset_colr_rep(stereo_wks, BLUE, &rep );
pset_colr_rep(mono_wks, BLUE, &rep );
```

```
rep.x = 1.0;
rep.y = 1.0;
rep.z = 0.0;
pset_colr_rep(stereo_wks, YELLOW, &rep );
pset_colr_rep(mono_wks, YELLOW, &rep );

rep.x = 0.0;
rep.y = 1.0;
rep.z = 1.0;
pset_colr_rep(stereo_wks, CYAN, &rep );
pset_colr_rep(mono_wks, CYAN, &rep );

rep.x = 1.0;
rep.y = 0.0;
rep.z = 1.0;
pset_colr_rep(stereo_wks, MAGENTA, &rep );
pset_colr_rep(mono_wks, MAGENTA, &rep );

              /* Viewing Stuff Common to Stereo and Mono */
              /* Set up the left and right eye views */
vrp.x         = 0.0;   vrp.y       = 0.0;   vrp.z       = 0.0;
vpn.delta_x   = 0.0;   vpn.delta_y = 0.0;   vpn.delta_z = 1.0;
vup.delta_x   = 0.0;   vup.delta_y = 1.0;   vup.delta_z = 0.0;
peval_view_ori_matrix3(&vrp, &vpn, &vup, &err,
                       view_rep.ori_matrix);
view_rep.clip_limit.x_min  = 0.0;
view_rep.clip_limit.x_max  = 1.0;
view_rep.clip_limit.y_min  = 0.0;
view_rep.clip_limit.y_max  = 1.0;
view_rep.clip_limit.z_min  = 0.0;
view_rep.clip_limit.z_max  = 1.0;
view_rep.xy_clip           = PIND_CLIP;
view_rep.back_clip         = PIND_NO_CLIP;
view_rep.front_clip        = PIND_NO_CLIP;

mapping.win.x_min          = -1.0 * VRC_WINDOW_SIZE/2.0;
mapping.win.x_max          =  VRC_WINDOW_SIZE/2.0;
mapping.win.y_min          = -1.0 * VRC_WINDOW_SIZE/2.0;
mapping.win.y_max          =  VRC_WINDOW_SIZE/2.0;
mapping.vp.z_min           = 0.0;
mapping.vp.z_max           = 1.0;
mapping.proj_type          = PTYPE_PERSPECT;
mapping.proj_ref_point.y   = 0.0;
mapping.proj_ref_point.z   = VRC_PRP;
```

```
    mapping.view_plane    = -0.0;
    mapping.back_plane    = -100.0;
    mapping.front_plane   = mapping.proj_ref_point.z - 1.0;

                          /* Viewing Stuff for Mono eye view */
    mapping.vp.x_min            = 0.0;
    mapping.vp.x_max            = 1.0;
    mapping.vp.y_min            = 0.0;
    mapping.vp.y_max            = 1.0;
    mapping.proj_ref_point.x    = 0.0;
    peval_view_map_matrix3(&mapping, &err,
                          view_rep.map_matrix);
    pset_view_rep3(mono_wks, mono_view_index, &view_rep);


                          /* Viewing Stuff for Stereo views */
    mapping.vp.x_min = 0.0;
    mapping.vp.x_max = 1.0;
    mapping.vp.y_min = 0.0;
    mapping.vp.y_max = 1.0;

                          /* left eye view */
    mapping.proj_ref_point.x = -1.0 * half_occular_dist;
    peval_view_map_matrix3(&mapping, &err,
                                view_rep.map_matrix);
    pset_view_rep3(stereo_wks, left_view_index, &view_rep);

                          /* right eye view */
    mapping.proj_ref_point.x = half_occular_dist;
    peval_view_map_matrix3(&mapping, &err,
                                view_rep.map_matrix);
    pset_view_rep3(stereo_wks, right_view_index, &view_rep);


/* Build the geometry structure */
/* ---------------------------- */
    popen_struct( structure);

/* Replace the usual setviewind element with the STEREO_VIEW_INDICES GSE */
    gse_struct.unsupp.size      = sizeof(stereo_views);
    gse_struct.unsupp.data      = (char *) &stereo_views;
    stereo_views.mono           = left_view_index;
    stereo_views.left           = left_view_index;
    stereo_views.right          = right_view_index;
```

```
pgse(PES_GSE_STEREO_VIEW_INDICES, &gse_struct);
pset_hlhsr_id(PHIGS_HLHSR_ID_ON);

                    /* Translate node for dials */
tran.x = 0.0;
tran.y = 0.0;
tran.z = 0.0;
translate_elem = 3;
ptranslate3( &tran, &err, tran1 );
pset_local_tran3( tran1, PTYPE_PRECONCAT );

                    /* set initial degree of rotation to 0 */
deg = 0.0;
rotate_elem = 4;
protate_z( deg, &err, comp_mat );
pset_local_tran3( comp_mat, PTYPE_PRECONCAT );

                    /* Translate node to offset object a little */
tran.x = 0.50;
tran.y = 0.0;
tran.z = 0.0;
ptranslate3( &tran, &err, tran1 );
pset_local_tran3( tran1, PTYPE_PRECONCAT );

pset_int_style( PSTYLE_SOLID);

pset_face_cull_mode( PCULL_NONE );
pset_face_disting_mode( PDISTING_NO );


pset_int_colr_ind( BLUE);   /* blue */
point_list.num_points = 4;
point_list.points = cube[0];
pfill_area3(&point_list);

pset_int_colr_ind( GREEN); /* green */
point_list.points = cube[1];
pfill_area3(&point_list);

pset_int_colr_ind( RED);    /* red */
point_list.points = cube[2];
pfill_area3(&point_list);
```

```
    pset_int_colr_ind( YELLOW);/* yellow */
    point_list.points = cube[3];
    pfill_area3(&point_list);

    pset_int_colr_ind( CYAN);  /* cyan */
    point_list.points = cube[4];
    pfill_area3(&point_list);

    pset_int_colr_ind( MAGENTA);/* magenta */
    point_list.points = cube[5];
    pfill_area3(&point_list);

    pclose_struct();

                        /* set edit mode to replace */
    pset_edit_mode(PEDIT_REPLACE);
                        /* set display updates to wait */
    pset_disp_upd_st(stereo_wks, PDEFER_WAIT, PMODE_NIVE);
    pset_disp_upd_st(mono_wks, PDEFER_WAIT, PMODE_NIVE);
                        /* post the structure */
    ppost_struct(stereo_wks, structure, 0.0);
    ppost_struct(mono_wks, structure, 0.0);
                        /* Do the first display */
    pupd_ws( mono_wks, PUPD_PERFORM );

                        /* Go look for dial events, etc. */
    MyMainLoop();

    pclose_ws(stereo_wks);
    pclose_ws(mono_wks);
    pclose_phigs();

}                       /* End of doPhigsStuff */


/*************************************************************************
 *
 *  MyMainLoop
 *
 *      Custom loop so we can catch events
 *      associated with the extensions to X Input and apply them to
 *      PHIGS structures.
 *
 *************************************************************************/
```

```
int MyMainLoop()

{
   int                  i, dials_values[10], error, events_waiting, nfds;
   unsigned long        readfds, writefds, exceptfds;
   unsigned long        monofdmask, stereofdmask;
   static int           rotate_dirty, tran_dirty;
   Pvec3                tranval, scaleval;
   XEvent               event, peek_event;
   XAnyEvent            *any_event;
   XButtonEvent         button_event;
   XDeviceMotionEvent   *dm;
   Display              *current_dpy;

                /* Set up file descriptor masks */
   monofdmask     = 1 << mono_fd;
   stereofdmask   = 1 << stereo_fd;

                /* Initialize the current screen pointer. */
   current_dpy = dpy;

                /* Initialize the rotation-scale accumulation matrix */
   identity_matrix(currmatrix);

                /* Initialize the translation accumulation vector */
   tranval.delta_x = 0.0; tranval.delta_y = 0.0; tranval.delta_z = 0.0;
                /* Initialize the scale accumulation vector */
   scaleval.delta_x = 1.0; scaleval.delta_y = 1.0; scaleval.delta_z = 1.0;

                /* Do forever */
   for (;;)
   {
                /* Use "select" to sleep on input from the two screens */
      readfds = monofdmask | stereofdmask;
      writefds = 0; exceptfds = 0;
      nfds = select(32, &readfds, &writefds, &exceptfds, NULL);
      if (nfds == 0)
         continue;

                /* First get the event */
                /* Check the Mono comm line */
      if ( (readfds & monofdmask) != 0)
      {
         if ( (events_waiting = XPending(dpy)) != 0)
```

```
        {
            current_dpy = dpy;
            XNextEvent(current_dpy, &event);
            any_event = (XAnyEvent *) &event;
            mode_flag = MONO;
        }
    }
                        /* Check the Stereo comm line */
    else if ( (readfds & stereofdmask) != 0)
    {
        if ( (events_waiting = XPending(sdpy)) != 0)
        {
            current_dpy = sdpy;
            XNextEvent(current_dpy, &event);
            any_event = (XAnyEvent *) &event;
            mode_flag = STEREO;
        }
    }
    else
    {
        continue;
    }

    if ( (event.type == MapNotify)    ||
         (event.type == Expose)       ||
         (event.type == ConfigureNotify) )
    {
        if (any_event->window == mono_win)
        {
            predraw_all_structs(mono_wks, PFLAG_ALWAYS);
            XSync(current_dpy, 0);
        }
        else if (any_event->window == stereo_win)
        {
            predraw_all_structs(stereo_wks, PFLAG_ALWAYS);
            XSync(current_dpy, 0);
        }
    }
    else if (event.type == LeaveNotify)
    {
        if (any_event->window == stereo_rootwin)
        {
            current_dpy = dpy;
        }
```

```
    else if (any_event->window == mono_rootwin)
    {
       current_dpy = sdpy;
    }
}

else if (event.type == DeviceMotion)
{
    do
    {
       peek_event.type = 0;
                    /* If this is a tablet event */
       dm = (XDeviceMotionEvent *) &event;
       if (dm->deviceid == tablet_id)
       {
          printf("Tablet Motion\n");
       }
                    /* If this is a dial event */
       else if (dm->deviceid == dials_id)
       {
                    /* Now process the data for all dials. */
          for (i = 0; i < dm->axes_count; i++)
          {
                    /* Copy the new dial value */
             if (dm->axis_data[i] != 0)
             {
                    dials_values[i + dm->first_axis ] =
                    dm->axis_data[i];
                    /* Set the dirty flag */
                    change[i+dm->first_axis] = 1;
             }
          }
          if (XPending(current_dpy) > 0)
          {
             XPeekEvent( current_dpy, &peek_event );
             if ( peek_event.type == DeviceMotion )
                    XNextEvent(current_dpy, &event);
          }

       }            /* End of if this is a dial event */

    } while ( peek_event.type == DeviceMotion );  /* End of do while */

                    /* Now Edit the PHIGS structure */
```

```
                    /* Do the rotation matrices */
    rotate_dirty = FALSE;
    if (change[XROT_DIAL])
    {
       protate_x( (float)dials_values[XROT_DIAL]/DIALSCALE,
                    &error, xrotmat );
       rotate_dirty = TRUE;
    }
    else
       identity_matrix( xrotmat );

    if (change[YROT_DIAL])
    {
       protate_y( (float)dials_values[YROT_DIAL]/DIALSCALE,
                    &error, yrotmat );
       rotate_dirty = TRUE;
    }
    else
       identity_matrix( yrotmat );

    if (change[ZROT_DIAL])
    {
       protate_z( (float)dials_values[ZROT_DIAL]/DIALSCALE,
                    &error, zrotmat );
       rotate_dirty = TRUE;
     }
     else
       identity_matrix( zrotmat );

    if (change[SCALE_DIAL])
    {
       scaleval.delta_x = 1.0 +
          (float)dials_values[SCALE_DIAL]/(10.0*DIALSCALE);
       scaleval.delta_y = scaleval.delta_x;
       scaleval.delta_z = scaleval.delta_x;
       pscale3( &scaleval, &error, scalemat );
       rotate_dirty = TRUE;
    }
    else
       identity_matrix( scalemat );

    if (rotate_dirty == TRUE)
    {
       pcompose_matrix3( currmatrix, xrotmat, &error, concat1 );
```

```
    pcompose_matrix3( concat1, yrotmat, &error, concat2 );
    pcompose_matrix3( concat2, zrotmat, &error, concat3 );
    pcompose_matrix3( concat3, scalemat, &error, currmatrix );
    popen_struct(structure);
    pset_edit_mode(PEDIT_REPLACE);
    pset_elem_ptr(rotate_elem);
    pset_local_tran3( currmatrix,PTYPE_PRECONCAT );
    pclose_struct();
}

                    /* Change Translation Matrix */
tran_dirty = FALSE;
if (change[XTRAN_DIAL])
{
    tranval.delta_x += (float)dials_values[XTRAN_DIAL]/DIALSCALE;
    tran_dirty = TRUE;
}

if (change[YTRAN_DIAL])
{
    tranval.delta_y += (float)dials_values[YTRAN_DIAL]/DIALSCALE;
    tran_dirty = TRUE;
}

if (change[ZTRAN_DIAL])
{
    tranval.delta_z += (float)dials_values[ZTRAN_DIAL]/DIALSCALE;
    tran_dirty = TRUE;
}

if (tran_dirty == TRUE)
{
    ptranslate3(&tranval, &error, tranmatrix);
    popen_struct(structure);
    pset_edit_mode(PEDIT_REPLACE);
    pset_elem_ptr(translate_elem);
    pset_local_tran3( tranmatrix,PTYPE_PRECONCAT );
    pclose_struct();
}


if (mode_flag == STEREO)
{
    pupd_ws(stereo_wks, PUPD_PERFORM);
```

```
        }
        else
         {
            pupd_ws(mono_wks, PUPD_PERFORM);
        }

        for (i=0; i<8; i++)
            change[i] = 0;

    }                       /* End of if event is DeviceMotion */

  }                         /* End of do forever */

}                           /* End of MyMainLoop */


/*****************************************************************************
 *
 *    main
 *
 *****************************************************************************/
main(argc, argv)
     int    argc;
     char   *argv[];
{
    int                     amount, i, j ;
    char                    *geom = NULL;
    register int            xdir, ydir;
    register int            xoff, yoff;
    register int            centerX, centerY;
    XSizeHints              hints;
    XGCValues               xgcv;
    XSetWindowAttributes    xswa;
    XWindowAttributes       winattr;
    Window                  root;
    Colormap                map;
    int                     x, y, w, h;
    int                     mono_button_x, mono_button_y, mono_button_w,
                            mono_button_h;
    int                     mono_button_bw, mono_button_bc,
                            mono_button_bckgrnd;
    Font                    font;
    char                    *fontname;
    XFontStruct             *font_info;
```

```
XGCValues       gcvals;
unsigned        valmask;
unsigned int    d;
unsigned int    bw = 1;
char            *display = NULL;
char            *stereoDisplayString;
Status          status;
char            *strptr;
char            *fg = NULL;
char            *bg = NULL;
char            *bd = NULL;
int             fg_pix, bg_pix, bd_pix;
XColor          fg_def, fg_exact, bg_def, bg_exact, bd_def, bd_exact;
int             bs = NotUseful;
Visual          visual;
int             numscreens;
xScreensInfo    *screen_info;

ProgramName = argv[0];

for (i=1; i < argc; i++)
{
   char *arg = argv[i];

if (arg[0] == '-') {
   switch (arg[1]) {
      case 'd':             /* -display host:dpy */
         if (++i >= argc) usage ();
         display = argv[i];
         continue;
      case 'g':             /* -geometry host:dpy */
         if (++i >= argc) usage ();
         geom = argv[i];
         continue;
      case 'b':             /* -b or -bg or -bd */
         if (!strcmp(argv[i], "-bg")) {
               if (++i >= argc) usage ();
               bg = argv[i];
         } else if (!strcmp(argv[i], "-bd")) {
               if (++i >= argc) usage ();
               bd = argv[i];
         } else if (!strcmp(argv[i], "-bw")) {
               if (++i >= argc) usage ();
               bw = atoi(argv[i]);
```

```
            } else
                  bs = Always;
                  continue;
         case 'f':              /* assume -fg */
            if (++i >= argc) usage ();
            fg = argv[i];
            continue;
         default:
            usage ();
      }
   } else if (argv [i] [0] == '=') /* obsolete */
         geom = argv[i];
   else
       usage ();
}

if (!(dpy = XOpenDisplay(display)))
{
   perror("Cannot open display\n");
   exit(-1);
}

mono_fd = XConnectionNumber(dpy);

                        /* Open the stereo screen for stereo windows. */
stereoDisplayString = XDisplayString(dpy);
strptr = rindex(stereoDisplayString, ':');
XGetScreensInfo(dpy, &screen_info);
numscreens = screen_info->numofscreens;
for (i=0; i<numscreens; i++)
{
   if (screen_info->screens[i].screentype == xStereoScreen)
   {
         sprintf( (strptr + 1), "0.%d", i);
   }
}

if (!(sdpy = XOpenDisplay(stereoDisplayString) ))
{
   perror("Cannot open display for Stereo screen\n");
   exit(-1);
}
stereo_fd = XConnectionNumber(sdpy);
if (fg) {
```

```
   status = XAllocNamedColor(dpy, map, fg, &fg_def, &fg_exact);
   fg_pix = status ? fg_def.pixel : WhitePixel(dpy, DefaultScreen(dpy));
} else
   fg_pix = WhitePixel(dpy, DefaultScreen(dpy));

if (bg) {
   status = XAllocNamedColor(dpy, map, bg, &bg_def, &bg_exact);
   bg_pix = status ? bg_def.pixel : BlackPixel(dpy, DefaultScreen(dpy));
} else
   bg_pix = BlackPixel(dpy, DefaultScreen(dpy));

if (bd) {
   status = XAllocNamedColor(dpy, map, bd, &bd_def, &bd_exact);
   bd_pix = status ? bd_def.pixel : WhitePixel(dpy, DefaultScreen(dpy));
} else
   bd_pix = WhitePixel(dpy, DefaultScreen(dpy));


if (geom)
{
   (void) XParseGeometry(geom, &mono_winx, &mono_winy,
         &mono_winw, &mono_winh);
}

xswa.backing_store      = bs;
xswa.event_mask         = ExposureMask | StructureNotifyMask;
xswa.background_pixel   = bg_pix;
xswa.border_pixel       = bd_pix;
visual.visualid         = CopyFromParent;

                        /* Select leave events on the mono root window */
mono_rootwin = RootWindow(dpy, DefaultScreen(dpy));
XSelectInput(dpy, mono_rootwin, LeaveWindowMask);

                        /* Create Mono window */
if (mono_winh == 0)
   mono_winh = 500;
if (mono_winw == 0)
   mono_winw = 500;
mono_win = XCreateWindow(dpy,
      RootWindow(dpy, DefaultScreen(dpy)),
      mono_winx, mono_winy, mono_winw, mono_winh, bw,
      DefaultDepth(dpy, DefaultScreen(dpy)), InputOutput,
      &visual,
```

```
      CWEventMask | CWBackingStore | CWBorderPixel | CWBackPixel,
      &xswa);
XChangeProperty(dpy, mono_win, XA_WM_NAME, XA_STRING, 8,
      PropModeReplace, "Stereo Demo", sizeof("Stereo Demo") );
XMapWindow(dpy, mono_win);

                        /* Select leave events on the stereo root window */
stereo_rootwin = RootWindow(sdpy, DefaultScreen(sdpy));
XSelectInput(sdpy, stereo_rootwin, LeaveWindowMask);

                        /* Open the stereo window on the Stereo Screen */
winx = 0;
winy = 0;
winh = (XDisplayHeight(sdpy, DefaultScreen(sdpy)))
   - 100 /* Fudge factor for borders */;
winw        = winh;
stereo_win  = XCreateWindow(sdpy,
      RootWindow(sdpy, DefaultScreen(sdpy)),
      winx, winy, winw, winh, bw,
      DefaultDepth(sdpy, DefaultScreen(sdpy)), InputOutput,
      &visual,
      CWEventMask | CWBackingStore | CWBorderPixel | CWBackPixel,
      &xswa);
XSync(sdpy, 0);
XChangeProperty(sdpy, stereo_win, XA_WM_NAME, XA_STRING, 8,
      PropModeReplace, "Stereo Demo", sizeof("Stereo Demo") );
hints.flags = USPosition | USSize ;
hints.x     = winx;
hints.y     = winy;
hints.width = winw;
hints.height= winh;
XSetNormalHints(sdpy, stereo_win, &hints);
XSetTransientForHint(sdpy, RootWindow(sdpy, DefaultScreen(sdpy)),
                        stereo_win );
XSync(sdpy, 0);
XMapWindow(sdpy, stereo_win);
XFlush(sdpy, 0);

doPhigsStuff(dpy, sdpy);

}                       /* End of main */
```

```
/*****************************************************************************
 *
 *    SensitizeWindow
 *
 *    Routine to select input events for PHIGS graphics windows.
 *
 *****************************************************************************/

int SensitizeWindow(my_dpy, my_window)
      Display     *my_dpy;
      Window      my_window;

{
   int                          i, j;
   int                          eventCount = 0;
   XEventClass                  eventClass[100];
   XStringFeedbackControl       strfc;
   KeySym ledstring[MAXDIALS][CHARS_PER_DIAL], blankled[CHARS_PER_DIAL];
   static char                  textstring[MAXDIALS][CHARS_PER_DIAL] =
                               { " X ROT  ", " Y ROT  ", " Z ROT  ", " SCALE  ",
                                 " X TRAN ", " Y TRAN ", " Z TRAN ", "        " };

                           /* Get a list of the available devices */
   devices = XListInputDevices(my_dpy, &ndevices);

                           /* Open the input devices */
   for (i = 0; i < ndevices; i++, devices++)
   {
      if (strcmp("TABLET", devices->name) == 0)
      {
         tablet_id = devices->id;
         tablet = XOpenDevice(my_dpy, tablet_id);
      }

      if (strcmp("KNOB_BOX", devices->name) == 0)
      {
         dials_id = devices->id;
         dials = XOpenDevice(my_dpy, dials_id);
      }
   }
```

```
eventCount = 0;

if (!tablet)
{
    fprintf(stderr, "No TABLET\n");
}
else
{
    DeviceMotionNotify(tablet, DeviceMotion, eventClass[eventCount]);
    eventCount++;

    DeviceButtonPress(tablet, DevicePress, eventClass[eventCount]);
    eventCount++;

    DeviceButtonRelease(tablet, DeviceRelease, eventClass[eventCount]);
    eventCount++;
}
if (!dials)
{
    fprintf(stderr, "No DIALS\n");
}
else
{
    DeviceMotionNotify(dials, DeviceMotion, eventClass[eventCount]);
    eventCount++;
}

if (!dials && !tablet)
{
    exit(0);
}

XFreeDeviceList(devices);

XSelectExtensionEvent(my_dpy, my_window, eventClass, eventCount);

                    /* Set the dial labels */
                    /* Load the keysym arrays */
for (i = 0; i < MAXDIALS; i++)
    for (j = 0; j < CHARS_PER_DIAL; j++)
    {
        ledstring[i][j] = (KeySym) textstring[i][j];   /* Dial labels  */
        blankled[j] = SPACEKEYSYM;                     /* Blank labels */
    }
```

```
    strfc.class           = StringFeedbackClass;
    strfc.length          = sizeof(XStringFeedbackControl);
    strfc.num_keysyms     = CHARS_PER_DIAL;
    for (i=0; i<MAXDIALS; i++)
    {
        strfc.id                = i;
        strfc.syms_to_display   = ledstring[i];
        XChangeFeedbackControl(my_dpy, dials, DvString, &strfc);
    }

    return;

}                       /* End of SensitizeWindow */


                        /* End of stereo_demo.c */
```