Software Libraries for 3D Binocular Applications

A Reference Manual

Version 0

July 28, 1994

Scott A. Safier

Robotics Institute

School of Computer Science

Carnegie Mellon University

Table of Contents

Introduction

This document is a user's reference manual for libraries of software that enable viewing 3D stereoscopic (binocular) images. The document is primarily a reference manual, providing detailed documentation on each function and data structure. There are also sample programs in the appendices that demonstrate applications of the software documented here.

The software described was written for X11R4 on a Silicon Graphics workstation running IRIX 4.0.5. The reader is assumed to have knowledge of Silicon Graphics' graphics library (GL), X11, the X Intrinsics (Xt) and mixed model programming.

The document is formatted similar to The Definitive Guides to the X Window System published by O'Reilly & Associates. The next section defines the files necessary to use the software. The third section describes the global data structures and the X Intrinsics widgets that are defined for 3D stereoscopic viewing. The fourth section describes the functions that are available in each of the software libraries. The appendices contain two example programs that use this software. The first example demonstrates creating a specialization of a widget used for 3D stereoscopic visualization. The second, simpler example demonstrates using a widget with 3D stereoscopic functionality in an application. A glossary is also provided as an appendix to define common terms used in this document.

Files

Public Header Files

Public header files define the variables and data structures necessary to use a widget.

• StWindow.h is the public header for the `StereoWindowWidget`. It defines the resources for this widget, the class pointer and the instance variables that an application needs to use this widget.

• St3D.h is the public header file for the `Stereo3DWidget`. It defines all the resources from StWindow.h, and adds new resources for this widget. It also defines the class pointer and the instance variables that an application needs to use this widget.

Private Header Files

Private header files define the variables and data structures to access the internals of a widget. Applications that define a specialization of a particular widget must `#include` the private header file to access its class and instance data structures.

• StWindowP.h is the private header for the `StereoWindowWidget`. It defines all the data structures necessary for this widget's class and instance records.

• SIWP.h is the private header for the `StereoWindowWidget`. It defines all the data structures necessary for this widget's class and instance records. This includes all the data structures defined by StWindowP.h.

Library Archives

A library archive is a collection of software object files that can be shared between applications. This reference manual describes the functions and procedure that can be found in either of two software libraries. Each library assumes a particular hardware configuration for viewing 3D stereoscopic images. All the functions behave identically regardless of the library selected by the application, unless otherwise noted.

• libXstereo.a -- the library for StereoGraphics Inc.'s hardware configuration

• libXSIW.a -- the library for stereo in a window

Data Structures

αRGB

Silicon graphics represents the color of a pixel as a 32-bit integer. The high 8-bits specify the degree of transparency associated with this color value. This is referred to as the α-value. The next 8-bits represent the red component of the color, followed by 8-bits for green, and the lowest 8-bits for the blue component.

stereo_image

The `stereo_image` structure represents the left and right fields of a precomputed 3D stereoscopic image. Each pixel in the image is represented by an αRGB integer. The data structure also has fields for the `width` and `height` of the image.

```
struct stereo_image {
 unsigned long *left, *right;
 int width, height;
};

typedef struct stereo_image *StereoImage;
```

StWindowCallbackStruct

The `StWindowCallbackStruct is widget-specific data structure used as a parameter to any StereoWindowWidget` or `Stereo3DWidget` callback resource. It provides the following widget-specific state information to callback procedure.

- `field` a value of either `StLeft` or `StRight`, for the field being displayed or to display;
- `window` the X structure that is the window for this field;
- `background` the αRGB value for the color of the background;
- `event` a pointer to the XEvent that caused this callback to be executed.

```
typedef struct {
  short    field;
  Window   window;
  long     background;
  XEvent * event;
} StWindowCallbackStruct;
```

Widgets

Name StereoWindowWidget

3D Stereoscopic (binocular) widget -- a widget that creates an X/Motif window for displaying 3D stereoscopic images

Synopsis

public headers: StWindow.h
class pointer: stereoWindowWidgetClass
instantiation: *widget* = XtCreateWidget(*name*, stereoWindowWidgetClass,...)

Description

The StereoWindowWidget is the most primitive widget for implementing a 3D stereoscopic window using a particular hardware configuration. The window created for this widget has both a left and a right field to display a 3D stereoscopic image. The application specifies a callback function to redisplay either the left or the right field of the image.

New Resources

name	*type*	*default*	*description*
StNname	String	NULL	A unique string to name this widget
StNredrawCallback	XtCallbackList		A function that is called to redraw one field of the image. The StCallbackStruct indicates whether the left or the right field should be redrawn.
StNpointer	GL_Object		a GL object for the pointer
StNpointerStyle	int	St2DPointer	one of *StNoPointer*, *St2DPointer*, *St3DPointer*

Translations and Actions

The widget defines translations and actions associated with using a pointer in the window. A two-dimensional pointer is overlaid on the 3D stereoscopic image whenever the cursor enters the window. An application must inherit the translations (by specifying `XtInheritTranslations` as the value of the `translations` field in the core part of the widget's class structure) from this widget to use this pointer functionality. The mouse functionality is triggered by `EnterNotify`, `LeaveNotify`, and `MotionNotify` events.

Callback resources

Redraw callback

An `XtCallbackProc` that is called twice to redraw the stereoscopic image. The procedure must execute anything necessary to generate either the left or the right field, as specified by the state information provided in the `StCallbackStruct`. The callback procedure should not call `swapbuffers()` or `mswapbuffers()`.

Internal structures

Mouse_device

A structure that maintains the current state of the 3D pointing device. The structure is defined as follows in any private definition for stereo 3D graphics windows.

```
struct mouse_device {
 long             cursor_in_window, cursor_mode;
 int              cursor_color, cursor_planes;
 unsigned char ** colors;
 XDevice *        xdevice;
 int *            location;
 int *            delta;
 short            flags;
};
```

See also

`define_cursor_color, get_cursor, initialize_stereo_pointer, overlay_window, set_3d_cursors, set_cursor`

Hardware implementations

This section is devoted to the implementation details specific to two hardware configurations: one for equipment produced for StereoGraphics's Inc., the other for a technique we call *stereo in a window.* If an application does not create widgets that are a specialization a `StereoWindowWidget`, the application developer need not be concerned with the details in this section beyond any resources that are defined for the hardware configurations.

The name of each hardware configuration labels the section that describes its `StereoWindowWidget`. In parenthesis after the name is a compiler declaration. **This compiler declaration must be defined** during the compilation of all application files.

<u>Stereographics (*X_SG*)</u>

Synopsis

private header: StWindowP.h
class name: StereoWindow

Class Hierarchy

Core→XmPrimitive→StereoWindowWidget

Description

Stereographics Inc. provides a technology for viewing 3D stereoscopic images that require binocular images to be stacked vertically. Their hardware drives a monitor at a higher speed (possibly by inserting an extra vertical synch pulse). LCD goggles are synchronized with the refresh of the screen, and one lens is darkened to obscure the image from the inappropriate eye. The implementation of the `StereoWindowWidget` for stereographics renders both the left and the right images on the screen, and, if possible, sends a signal to the monitor to refresh at 120Hz.

Class Structure

```
typedef struct {int dummy;} StereoWindowClassPart;

typedef struct _StereoWindowClassRec {
 CoreClassPart         core_class;
 XmPrimitiveClassPart  primitive_class;
 StereoWindowClassPart stereoWindow_class;
} StereoWindowClassRec;
```

Instance Structure

```
typedef struct {
 char *                 name;
 int                    pointer_type;
 XtCallbackList         redraw_callback;
 Widget                 framel, framer, left, right;
 struct mouse_device *  pointer;
 GL_Object              two_d_cursor;
 struct {
    GLXconfig *      config;
    XVisualInfo *    visualInfo;
    Boolean          override_colormap;
    XtCallbackList   ginit_callback;
    XtCallbackList   input_callback;
    Boolean          overlay_exists;
    Boolean          underlay_exists;
    Boolean          popup_exists;
 } glxpart;
 unsigned long          background;
} StereoWindowPart;
```

```
typedef struct _StereoWindowRec {
  CorePart             core;
  XmPrimitivePart      primitive;
  StereoWindowPart     stereowindow;
} StereoWindowRec;
```

Stereo in a window (*#define X_SIW*)

Synopsis

private header: SIWP.h
class name: StereoInaWindow

Class Hierarchy

Core→XmPrimitive→GlxMDraw→StereoWindowWidget

Description

Stereo in a window is achieved by synchronizing a set of liquid crystal goggles to the current image on the screen. A signal is sent to a serial port indicating which field is being displayed, and some special hardware interprets this signal and drives the liquid crystal goggles. The `StereoWindowWidget` renders the images to be displayed in each buffer of a graphics system configured for double buffering. After each screen refresh, the images and state of the goggles are toggled.

Hardware Specific Resources

name	*type*	*default*	*description*
StNappContext	Pointer	null	the current application context -- this resource is defined for all Stereo3DWidgets, and is described in more detail there. This resource is required for all widgets that use this hardware configuration.
StNportNumber	int	1	the port connected to the LCD goggles. This port will be sent *high* (1) for left, and *low* (0) for right.

Class Structure

```
typedef struct {int dummy;} StereoWindowClassPart;

typedef struct _SIWClassRec {
  CoreClassPart          core_class;
  XmPrimitiveClassPart   primitive_class;
  GlxDrawClassPart       glxDraw_class;
  StereoWindowClassPart  stereoWindow_class;
} StereoWindowClassRec;
```

Instance Structure

```
typedef struct {
  char *                  name;
  int                     pointer_type, port_number;
  short                   state;
  XtIntervalID            current_timer;
  XtCallbackList          redraw_callback;
  struct mouse_device *   pointer;
  unsigned long           background;
  GL_Object               two_d_cursor;
  XtAppContext *          context;
  File *                  port;
} StereoWindowPart;

typedef struct _SIWRec {
  CorePart          core;
  XmPrimitivePart   primitive;
  GlxDrawPart       glxDraw;
  StereoWindowPart  stereowindow;
} StereoWindowRec;
```

Name Stereo3DWidget
3D Stereoscopic (binocular) Widget with support for 3D GL graphics

Synopsis
public headers: St3D.h
class pointer: stereo3DWidgetClass
instantiation: *widget* = XtCreateWidget(*name*, stereo3DWidgetClass,...)

Description

The `Stereo3DWidget` provides all the functionality of the `StereoWindowWidget`, and augments it with capabilities to render 3D stereoscopic images. The widget encapsulates information about the geometry for computing images based upon application specific parameters, and applies this information automatically during rendering. It also provides a primitive capability for managing Silicon Graphic's lighting and material property models.

New Resources

name	*type*	*default*	*description*
StNtrainingDelta	int	40	The number of incremental steps to adjust eye separation from 0 (a flat image) to the current interoccular distance.
StNtraining	int	0	When non-zero, eye-training is occurring.
StNeyeSeperation	float	0.0	The current eye-separation.
StNappContext	Pointer	NULL	The current application context. It is needed for to generate timed events, such as for eye-training. *The application context may be necessary for some StereoWindowWidgets..*

Inherited Resources

name	*inherited from*
StNname	`StereoWindowWidget`
StNredrawCallback	`StereoWindowWidget`
StNpointer	`StereoWindowWidget`
StNpointerStyle	`StereoWindowWidget`

Translations and Actions

The widget defines translations and actions associated with using a pointer in the window. A two-dimensional pointer is overlaid on the 3D stereoscopic image whenever the cursor enters the window. An application must inherit the translations (by specifying `XtInheritTranslations` as the value of the translations field in the core part of the widget's class structure) from this widget to use this pointer functionality. The mouse functionality is triggered by EnterNotify, LeaveNotify, and MotionNotify events.

Internal Structures

```
struct inferred_perspective {
 int      fovy;
 float    aspect, near, far, conv, eye;
 double   zoomFactor, thetaYNoZoom;
 short    inferPerspective;
 double   screenDistance, screenHeight;
};

struct perspective_information {
 double real_near, screen_distance, real_per_virtual,
        eye_seperation;
 double screen_width, screen_height;
 struct inferred_perspective *result;
};

struct coord_system {
 float xmax, ymax, zmax, xmin, ymin, zmin;
};
typedef struct coord_system *WindowCoord;

typedef struct light_map {
 int index;
 int num;
} LIGHT_MAP;
```

See also

change_interoccular_distance, current_viewpoint, define_cursor_color, get_cursor, initialize_stereo_pointer, overlay_window, set_3d_cursors, set_cursor, set_cursor_depth, SetLight, SetLightModel, SetMaterial, set_projection, set_viewpoint, StAppMainLoop, st_lookat, st_rotate_eye, st_rotate_viewpoint, st_translate_eye, st_translate_viewpoint, st_zoom

Hardware implementations

This section is devoted to the implementation details specific to two hardware configurations: one for equipment produced for StereoGraphics's Inc., the other for a technique we call *stereo in a window.* If an application does not create widgets that are a specialization a StereoWindowWidget, the application developer need not be concerned with the details in this section beyond any resources that are defined for the hardware configurations.

The name of each hardware configuration labels the section that describes its StereoWindowWidget. In parenthesis after the name is a compiler declaration. **This compiler declaration must be defined** during the compilation of all application files.

Stereographics (*X_SG*)

Synopsis

private header: St3DP.h
class name: Stereo3DWindow

Class Hierarchy

Core→XmPrimitive→StereoWindowWidget→Stereo3DWidget

Description

Stereographics Inc. provides a technology for viewing 3D stereoscopic images that require binocular images to be stacked vertically. Their hardware drives a monitor at a higher speed (possibly by inserting an extra vertical synch pulse). LCD goggles are synchronized with the refresh of the screen, and one lens is darkened to obscure the image from the inappropriate eye. The implementation of the `StereoWindowWidget` for stereographics renders both the left and the right images on the screen, and, if possible, sends a signal to the monitor to refresh at 120Hz.

Class Structure

```
typedef struct {int dummy;} Stereo3DClassPart;

typedef struct _Stereo3DClassRec {
  CoreClassPart           core_class;
  XmPrimitiveClassPart    primitive_class;
  StereoWindowClassPart   stereoWindow_class;
  Stereo3DClassPart       stereo3D_class;
} Stereo3DClassRec;
```

Instance Structure

```
typedef struct {
 int              training, training_delta;
 double *         new_eye_seperation;
 short            type, lights_set_p;
 union {
    struct coord_system *             system;
    struct perspective_information *  iperspective;
 } description;
 Matrix           left_proj, right_proj, left_model,
                  right_model;
 Matrix           left_model_noview, right_model_noview;
 float            lookfrom[3], lookto[3], twist;
 double           training_eye_seperation;
 GL_Object        rview, lview, three_d_cursor;
 Coord            cursor[3];
 int              material, light_model;
 LIGHT_MAP        curlights[NUM_LIGHTS];
 XtAppContext *   context;
} Stereo3DPart;
typedef struct _Stereo3DRec {
```

```
    CorePart         core;
    XmPrimitivePart  primitive;
    StereoWindowPart stereowindow;
    Stereo3DPart     stereo3D;
 } Stereo3DRec;
```

Stereo in a window (*X_SIW*)

Synopsis

private header: S3DIWP.h
class name: Stereo3DInaWindow

Class Hierarchy

Core→XmPrimitive→GlxMDraw→StereoWindowWidget→Stereo3DWidget

Description

Stereo in a window is achieved by synchronizing a set of liquid crystal goggles to the current image on the screen. A signal is sent to a serial port indicating which field is being displayed, and some special hardware interprets this signal and drives the liquid crystal goggles. The `Stereo3DWidget` renders the images to be displayed in each buffer of a graphics system configured for double buffering. The images and state of the goggles are toggled after each refresh of the screen.

Class Structure

```
typedef struct {int dummy;} Stereo3DClassPart;

typedef struct _Stereo3DClassRec {
  CoreClassPart          core_class;
  XmPrimitiveClassPart   primitive_class;
  GlxDrawClassPart       glxDraw_class;
  StereoWindowClassPart  stereoWindow_class;
  Stereo3DClassPart      stereo3D_class;
 } Stereo3DClassRec;
```

Inherited Resources

name	*inherited from*	*hardware*
StNportNumber	`StWindowWidget`	Stereo in a window

Instance Structure

```
typedef struct {
 int            training, training_delta;
 double *       new_eye_seperation;
 short          type, lights_set_p;
 union {
   struct coord_system *            system;
   struct perspective_information * iperspective;
 } description;
 Matrix         left_proj, right_proj, left_model, right_model;
 Matrix         left_model_noview, right_model_noview;
 float          lookfrom[3], lookto[3], twist;
 double         training_eye_seperation;
 GL_Object      rview, lview, three_d_cursor;
 Coord          cursor[3];
 int            material, light_model;
 LIGHT_MAP      curlights[NUM_LIGHTS];
} Stereo3DPart;

typedef struct _Stereo3DRec {
 CorePart core;
 XmPrimitivePart primitive;
 GlxDrawPart glxDraw;
 StereoWindowPart stereowindow;
 Stereo3DPart stereo3D;
} Stereo3DRec;
```

Functions

Name change_interoccular_distance
specify the distance between the synthetic eyes.

Synopsis

```
void change_interoccular_distance(Stereo3DWidget w,
                                  double new_seperation)
```

inputs

`w` The widget where the interoccular distance is being changed

`new_seperation`
The new interoccular distance specified in world coordinates

Description

`change_interoccular_distance` replaces the distance separating the synthetic eyes with the new value, and recomputes the perspective transformations for the left and right fields of the 3D stereoscopic image.

`new_seperation` is the interoccular distance, and must be a positive number that is measured in the world coordinate system.

Usage

`change_interoccular_distance` is intended to be used to make the 3D image comfortable for viewing by changing the assumed separation between the left and right eyes.

Name current_viewpoint

widget query for current line of sight

Synopsis

```
void current_viewpoint(float *eye, float *view, float *twist,
                       Stereo3DWidget w)
```

inputs

`eye` an array of three floats that is the position of the syntehtic eyes in the world coordinate system

`view` an array of three floats that is the position in the synthetic world being viewed

`twist` the angle of rotation about the line of sight

`w` the widget being query

Description

`current_viewpoint` queries the specified widget for the current viewing parameters. Its output is 7 floating point values representing the position of the synthetic eyes and the point in the synthetic world that defines the line of sight. The final parameter, `twist,` is the angular rotation of the line of sight.

See Also

`st_lookat`

Name define_cursor_color

establish a color in the color map used to display the cursor.

Synopsis

```
int define_cursor_color(StereoWindowWidget w, int id,
                        unsigned char red,
                        unsigned char green,
                        unsigned char blue)
```

inputs

`w`	the stereo widget whose parameters are being changed
`id`	the index in the color map for the color being defined
`red`	the red component of the color being defined
`green`	the green component of the color being defined
`blue`	the blue component of the color being defined

Description

`define_cursor_color` sets up the color map used to display the pointer on the screen. Its parameters are the widget whose color map is being effected, the index in the color map, and the red, green and blue components of the color.

Usage

`define_cursor_color` is an initialization routine that should be called after a widget is created to define the color map for the cursor.

See Also

`set_cursor_color`

Name get_cursor

the current position of the pointer in the synthetic world

Synopsis

```
void get_cursor(Coord *x, Coord *y, Coord *z,
                Stereo3DWidget w)
```

inputs

`x` the x-coordinate of the cursor in the synthetic world

`y` the y-coordinate of the cursor in the synthetic world

`z` the z-coordinate of the cursor in the synthetic world

`w` the widget being queried

Description

Query the specified widget for the current position of the cursor. Its parameters are pointers to coordinates for the cursor's position in 3-space.

See Also

`set_cursor`

Name initialize_stereo_pointer
setup routine for the stereo cursor

Synopsis

```
void initialize_stereo_pointer(Widget background,
                               XtAppContext context)
```

inputs

`background`
a widget that encloses the stereo window, normally a constraint or a composite widget

`context`
the current application context

Description

Sets up the routines to manage the stereo pointer.

Hardware specific usage

StereoGraphics: This initialization routine should be called before any stereo windows are created or realized. It should be called after the `background` widget is realized.

Stereo in a window: no special usage requirements.

Algorithm

Create an empty cursor for the hardware cursor when inside a stereo window.

Set the current cursor for the `background` widget and its children to the *gumby* cursor font.

Name overlayWindow

return the X window structure that is the overlay window of the widget.

Synopsis

`Window overlayWindow(Widget w)`

inputs

`w` a `GlxDrawWidget` with an overlay plane

Description

Returns the X `Window` structure for the overlay window.

Usage

This routine is from Silicon Graphics 4Dgifts directory. GLX uses multiple X Window structures for the normal drawing plane, and the underlay and overlay planes. This function returns the overlay plane of a `GlxDraw` widget, if it has one.

Name read_lr_images

parse a 3D stereoscopic image that has seperate files for each field.

Synopsis

```
StereoImage read_lr_images(char *lname, char *rname)
```

inputs

`lname` a character string that names the file containing the image for the left field

`rname` a character string that names the file containing the image for the right field

Description

Read the 3D stereoscopic image from the files, and return it in a `StereoImage` structure. The files may be formatted as either jpeg, compuserve gif, or one of the pbm family.

See also

`read_stereo_image`
`Data Structures` `stereo_image`

Name read_stereo_image
parse a stereo image stored in a single file

Synopsis

```
StereoImage read_stereo_image(char *fname)
```

inputs

`fname` a character string that names the file containing a 3D stereoscopic image.

Description

Read the 3D stereoscopic image from the file, and return it in a `StereoImage` structure. The image is divided in half horizontally to create individual left and right images. The file may be formatted as either jpeg, compuserve gif, or one of the pbm family.

See also

`read_lr_images`
`Data Structures` `stereo_image`

Name set_3d_cursors

define the graphic objects that will be displayed as the 2D screen cursor and the 3D cursor for the synthetic world.

Synopsis

```
void set_3d_cursors(GL_Object two_d, GL_Object three_d,
                    StereoWindowWidget w)
```

inputs

`two_d`	a display list for the 2D cursor
`three_d`	a display list for the 3D cursor
`w`	the widget where these cursors are defined

Description

This function sets the graphic objects that are displayed for the 2D and the 3D cursor. The 2D cursor is displayed whenever the pointer is flagged as `CURSOR_2D` or when the 3D pointer is not viewable in the current window (e.g. the cursor is not in the line of sight of the synthetic eyes). The 3D cursor is in the synthetic world and may interact with objects in this world.

Usage

This routine should be used to initialize the pointer objects, and whenever it is desirable to have the cursor change shapes.

Bugs

Cursors are drawn in the overdraw plane. Because of hardware limitations, the 3D cursor cannot be occluded by objects in the synthetic world.

See also

`define_cursor_color`, `get_cursor`, `set_cursor`, `set_cursor_color`, `set_cursor_depth`

Name set_cursor

place the cursor in the synthetic world

Synopsis

```
void set_cursor(Coord x, Coord y, Coord z,
                StereoWindowWidget w)
```

inputs

`x`	the new x coordinate for the cursor in the synthetic world
`y`	the new y coordinate for the cursor in the synthetic world
`z`	the new z coordinate for the cursor in the synthetic world
`w`	the widget where this cursor is being positioned

Description

Set the position of a 3D cursor in the synthetic world.

See also

`define_cursor_color, get_cursor, set_3d_cursors, set_cursor_color, set_cursor_depth`

Name set_cursor_color

change the color of the pointer

Synopsis

```
int set_cursor_color(StereoWindowWidget w, int id)
```

inputs

`w` the widget being effected

`id` the id of the color in the cursor's colormap

Description

change the color of the cursor

Example Usage

This code fragment changes the color of the cursor based upon the value of pointer's position relative to the Y axis.

```
{  Coord x, y, z;
   get_cursor(&x, &y, &z, w);
   if (y < 0)
      set_cursor_color(w,1);
   else
      set_cursor_color(w,2);
}
```

See also

`define_cursor_color`

Name set_cursor_depth

change the depth of the pointer in the synthetic world

Synopsis

void set_cursor_depth(Coord z, Stereo3DWidget w)

inputs

z	the designated depth value for the cursor
w	the widget being effected

Description

Change the position of the cursor along the Z axis.

Usage

This routine is used to initialize the depth of the cursor. It is also used in user-interface routines to control the depth of the 3D cursor when other position information is bound to a 2D device, such as a standard mouse.

See also

`set_cursor`

Name SetLight

turn on and off lights in the synthetic world

Synopsis

```
void SetLight(int new_light, Stereo3DWidget w)
```

inputs

`new_light`
: The index of the light to turn on, as defined by `lmbind`.

`w`
: The widget that contains this light.

Description

Turn on a light that has not already been turned on.

Usage

This function initializes lights for the lighting model. It checks if the specified light has been turned on, if not, it turns the light on.

Algorithm

This function originally was provided in Silicon Graphic's 4Dgifts library, and has been converted to work with widgets.

Structures

The lightmap data structure is used internal to widgets.

```
typedef struct light_map {
    int index;
    int num;
} LIGHT_MAP;
```

See Also

`lmbind`(3G), `lmdef`(3G), `SetLightModel`, `SetMaterial`

Name SetLightModel

change the graphic engine's lighting model for the scene being rendered.

Synopsis

```
void SetLightModel(int new_model, Stereo3DWidget w)
```

inputs

`new_model`
: A lighting model assigned by `lmbind`.

`w`
: The widget where the model is being defined.

Description

Define the lighting model to use when rendering images.

Usage

This routine is used to initialize the lighting model in the graphics database.

Algorithm

This function originally was provided in Silicon Graphic's 4Dgifts library, and has been converted to work with widgets.

See also

`lmbind`(3G), `lmdef`(3G), `SetLight`, `SetMaterial`

Name SetMaterial

change the default material properties

Synopsis

```
void SetMaterial(int new_mat, Stereo3DWidget w)
```

inputs

`new_mat`
: the material specification as defined to `lmbind`.

`w`
: the widget being effected.

Description

Assert that the objects being rendered will obey the properties specified for this material.

Algorithm

This function originally was provided in Silicon Graphic's 4Dgifts library, and has been converted to work with widgets.

See also

`lmbind`(3G), `lmdef`(3G), `SetLight`, `SetLightModel`

Name set_projection

establish the initial viewing parameters for the synthetic world

Synopsis

```
void set_projection(Stereo3DWidget w, double far,
                    double near, double height, double width,
                    double factor, double distance,
                    double separation, double aspect)
```

inputs

`w` the 3D stereoscopic window whose perspective geometry is being specified

`far` distance to the far clipping plane in world coordinate system

`near` distance to the near clipping plane in the eye coordinate system

`height` the height of the screen in real world coordinates

`width` the width of the screen in real world coordinates

`factor` the multiple to convert real world coordinates to virtual coordinates

`distance` the distance the viewer is assumed to be from the screen, in real world coordinates

`separation` distance between the synthetic eyes in the world coordinate system

`aspect` width to height ratio of the screen

Description

This routine establishes the geometry for the perspective transformation to render a realistic 3D stereoscopic scene

Usage

`set_projection` should be used to initialize the perspective for the window. Once this perspective is initialized, functions such as change_interoccular_distance or st_zoom should be used to change the parameters associated with the perspective geometry.

See Also

`change_interoccular_distance, st_zoom, st_lookat`

Name set_viewpoint

change the line of sight for the synthetic eyes.

Synopsis

void set_viewpoint(float *eye, float *view, float twist, Stereo3DWidget w)

inputs

`eye`	an array of length 3 that is the position of the eye
`view`	an array of length 3 that is the position of the viewpoint
`twist`	the angular rotation about the line of sight
`w`	the widget being effected

Description

Specify a new line of sight by asserting a new position for the synthetic eyes and the point being viewed in the synthetic world.

Usage

similar to `st_lookat`, but has parameters compatible with `get_viewpoint`.

See also

`get_viewpoint`

Name StAppMainLoop

continually process events in an application with stereo windows.

Synopsis

void StAppMainLoop(XtAppContext app)

inputs

`app` the current application context

Description

an infinite loop that continually processes X Events. It is similar to `XtAppMainLoop`, but may perform actions specific to 3D stereoscopic visualization.

See also

`XtAppContext`

Name st_lookat

change the position of the viewing vehicle and the position in the synthetic world being viewed.

Synopsis

```
void st_lookat(float vx, float vy, float vz, float px, float py, float pz,
               Angle twist, Stereo3DWidget w)
```

inputs

`vx`	the x position of the synthetic eyes in the synthetic world
`vy`	the y position of the synthetic eyes in the synthetic world
`vz`	the z position of the synthetic eyes in the synthetic world
`px`	the x position being viewed in the synthetic world
`py`	the y position being viewed in the synthetic world
`pz`	the z position being viewed in the synthetic world
`twist`	angle of rotation about the line of sight
`w`	the widget being effected

Description

Specify a new line of sight by setting the position of the synthetic eyes and the position being viewed in the synthetic world.

See also

`lookat`(3G), `set_viewpoint`, `get_viewpoint`

Name st_rotate_eye

Define a new line of sight by pivoting the old line of sight about the position being viewed in the synthetic world.

Synopsis

void st_rotate_eye(char axis, double theta, Stereo3DWidget w)

inputs

`axis`	the axis of rotation, either `X`, `Y` or `Z`
`theta`	the number of degrees to rotate, in radians
`w`	the widget

Description

Keeping constant the distance between the synthetic eyes and the point in the synthetic world, and keeping the point being viewed constant, change the position of the synthetic eyes by rotating about the viewpoint.

Name st_rotate_viewpoint

Define a new line of sight by pivoting the old line of sight about the synthetic eyes.

Synopsis

void st_rotate_viewpoint(char axis, double theta, Stereo3DWidget w)

inputs

`axis`	the axis of rotation, either `X`, `Y` or `Z`
`theta`	the number of degrees to rotate, in radians
`w`	the widget

Description

Keeping constant the distance between the synthetic eyes and the point in the synthetic world, and keeping the position of the eyes constant, change the position being viewed by rotating the eyes about an axis.

Name st_translate_eye

Define a new line of sight by moving the position of the synthetic eyes along a vector.

Synopsis

void st_translate_eye(float dx, float dy, float dz, Stereo3DWidget w)

inputs

`dx`	distance to translate in the x dimension
`dy`	distance to translate in the y dimension
`dz`	distance to translate in the z dimension
`w`	the widget where the translation is occurring

Description

The parameters of the function specify offsets to move the position of the synthetic eyes in each of the three dimensions.

Algorithm

$$\begin{bmatrix} \text{position}_x \\ \text{position}_y \\ \text{position}_z \end{bmatrix} = \begin{bmatrix} \text{position}_x \\ \text{position}_y \\ \text{position}_z \end{bmatrix} + \begin{bmatrix} \Delta_x \\ \Delta_y \\ \Delta_z \end{bmatrix}$$

Name st_translate_viewpoint

Define a new line of sight by moving the position of the point being viewed in the synthetic world along a vector.

Synopsis

void st_translate_viewpoint(float dx, float dy, float dz, Stereo3DWidget w)

inputs

`dx`	distance to translate in the x dimension
`dy`	distance to translate in the y dimension
`dz`	distance to translate in the z dimension
`w`	the widget where the translation is occurring

Description

The parameters of the function specify offsets to move the position being viewed in the synthetic world in each of the three dimensions.

Algorithm

$$\begin{bmatrix} \text{position}_x \\ \text{position}_y \\ \text{position}_z \end{bmatrix} = \begin{bmatrix} \text{position}_x \\ \text{position}_y \\ \text{position}_z \end{bmatrix} + \begin{bmatrix} \Delta_x \\ \Delta_y \\ \Delta_z \end{bmatrix}$$

Name st_zoom

change the focal parameter of the optical viewing system

Synopsis

```
void st_zoom(Stereo3DWidget w, double factor)
```

inputs

`w` the widget being effected

`factor`
the magnification to apply to the image

Description

Increases or decreases the magnification being applied to the image.

Name WidgetBackgroundToGlRgb

convert the X color that is the background to the RGB representation used by GL.

Synopsis

```
int WidgetBackgroundToGlRgb(Widget w)
```

inputs

`w` the widget whose background is being queried

Description

Convert the background color of a widget from an X Color to the αRGB representation used by GL.

Algorithm

$$color_{\alpha RGB} = (color_{red} \rightarrow 8) + ((color_{green} \rightarrow 8) \leftarrow 8) + ((color_{blue} \rightarrow 8) \leftarrow 16)$$

Example: StereoWindowWidget

This section provides an example program for using the `StereoWindowWidget`. The example program reads in a stereo image that has been stored in a file or a pair of files. The program is structured to encapsulate the image specific code in a widget. This program also serves as an example of how to write an application specific widget that inherits properties from the `StereoWindowWidget`. The next section provides an example program for the `Stereo3DWidget` that is structured not to use an application specific widget. An application writer may find either style appropriate.

The example program is named *viewmaster* because of its similarity to the viewmaster product that allows a user to view images that are stereo pairs. The program is divided into four files.

- viewmaster.c the main program that creates all the X windows and calls the user interface loop.
- ViewMaster.c the application specific widget.
- ViewMaster.h the public header file for the widget. This header file is included in all application programs that use the ViewMaster widget but do not access the internals of the widget.
- ViewMasterP.h the private header file for the widget. This header file contains the data structure definitions for the widget. Applications that create widgets that are children of the ViewMaster widget include this header file.

viewmaster.c

This file is the main program for viewmaster.

First, we process the include files

```
#include <math.h>
#include <stdio.h>
#include <malloc.h>
```

The X include files. Intrinsic is the header file for Xt, the X intrinsics. The file Shell.h defines the shell widget.

```
#include <X11/Intrinsic.h>
#include <X11/Xatom.h>
#include <X11/StringDefs.h>
#include <X11/Shell.h>
#include "ViewMasterP.h"
#include <gl/get.h>
```

Motif include files. BulletinB.h defines the bulletin board widget. PushB.h defines push buttons. Cascade.h defines cascaded menus. MessageB.h is used in menus. RowColumn.h defines the menu bar. The version of the program that uses stereo in a window also defines the frame widget in Frame.h.

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

#include <Xm/BulletinB.h>
#include <Xm/PushB.h>
#include <Xm/CascadeB.h>
#include <Xm/MessageB.h>
#include <Xm/RowColumn.h>
#ifdef X_SIW
#include <Xm/Frame.h>
#endif
```

Process the include files for stereo windows.

```
#include "StWindow.h"
#include "stereo.h"
#include <X11/Xirisw/GlxMDraw.h>
```

This procedure is the callback method required by `StereoWindowWidgets` to redraw the window.

```
static void redraw_images(ViewMasterWidget w, caddr_t client_data,
                          StWindowCallbackStruct *call_data)
{
  register int factor, hoffset, voffset;
```

Compute the user specified offset.

```
  factor = (call_data->mode == StLeft) ? -1 : 1;
  hoffset = w->viewmaster.horizontal_offset/2 * factor;
  voffset = w->viewmaster.vertical_offset/2 * factor;
```

Clear the background.

```
   cpack(WidgetBackgroundToGlRgb((Widget)w));
  clear();
```

Write the image to the window.

```
  lrectwrite(hoffset, voffset,
             w->core.width + hoffset,
             w->core.height + voffset,
             (call_data->mode == StRight) ?
             w->viewmaster.lrimgs->right :
             w->viewmaster.lrimgs->left);
}
```

The exit function is attached to a menu item. When this function is selected, the widgets on the screen are destroyed, and the program exits. The stereo in a window version of the program need not worry about changing the state of the monitor by calling `stereo_off`.

```
static void Exit (Widget w, XtPointer client_data,
                  XtPointer call_data)
{
#ifndef X_SIW
   stereo_off();
#endif
   XtDestroyWidget((Widget) client_data);
   exit(0);
}
```

The stereographics version of the program also provides control over whether the screen is in stereo mode (120 Hz) or monocular mode (60Hz). This procedure toggles that state.

```
#ifndef X_SIW
static void ToggleStereo(Widget w, XtPointer client_data,
XtPointer call_data)
{
  static int mode = 0;
  if (!mode) {
    stereo_on(1);
    mode = 1;
  } else {
    stereo_off();
    mode = 0;
  }
}
#endif
```

The graphics library widget requires an initialization function that sets up the graphic engine's database.

```
static void
initCB(Widget w, caddr_t client_data,GlxDrawCallbackStruct
*call_data)
{
     GLXwinset(XtDisplay(w), call_data->window);
     dither(DT_OFF); RGBmode();  doublebuffer();
     zbuffer(TRUE);
     if (getenv("ANTIALIAS")) {
       subpixel(TRUE);
       pntsmooth(SMP_ON | SMP_SMOOTHER);
       linesmooth(SML_ON | SML_SMOOTHER | SML_END_CORRECT);
       blendfunction(BF_ONE,BF_SA);
       polysmooth(PYSM_ON);
     }

     GLXwinset(XtDisplay(w), overlayWindow(w));
     doublebuffer(); cmode();

     gflush();
}
```

The main program.

```
void main(argc, argv)
     int argc;
     char **argv;
{
```

Variables for the screen size and whether to display for NTSC.

```
   long xmaxscreen, ymaxscreen;
   int NTSCmodep;
```

Xt widget variables.

```
   Widget shell, background, viewmaster, stwin,
           menubar, filebutton, controls_button,
          filemenu, quit,
           controls_menu, stereo_mode, frame;
   XtAppContext app_context;
```

Variables for Xt's argument passing methodology.

```
   Arg args[20];
   int n;
```

set up Xt's method of parsing command line arguments.

```
   XrmOptionDescRec options[] = {
      {"-hoffset","ViewMaster*hoffset",  XrmoptionSepArg,  NULL},
      {"-voffset","*voffset",            XrmoptionSepArg,  NULL},
      {"-reverse","*reverse",            XrmoptionNoArg,   "True"},
      {"-stdout", "*stdout",             XrmoptionNoArg,   "True"},
   };
```

Fallback resources are defaults. The first entry is null to allow its value to be computed later in the program.

```
   String fallback_resources[] = {
     NULL,
#ifdef X_SIW
      "*frame*shadowType: SHADOW_IN",
#endif
      "*background: black",
     NULL};
```

The parameters for configure the graphics library.

```
   /* The GLX configuration parameter:
    *      Double buffering
    * RGB mode
    *    Zbuffer
    * overlay
    * nothing else special
    */
   GLXconfig glxConfig [] = {
     { GLX_NORMAL, GLX_DOUBLE, TRUE },
     { GLX_NORMAL, GLX_RGB, TRUE },
     { GLX_NORMAL,GLX_ZSIZE, GLX_NOCONFIG },
     { GLX_OVERLAY, GLX_BUFSIZE, 4},
    { 0, 0, 0 },
```

```
};
extern int cursor_planes;
```

Set up the geometry by checking for an environment variable that signals the program should use NTSC protocols. Next, set the first element in the fallback resources to be the geometry of the window.

```
NTSCmodep = getenv("NTSC") ? 1 : 0;
if (!NTSCmodep) {
  char temp[100];
  bzero(temp, 100);
   xmaxscreen = getgdesc(GD_XPMAX);
   ymaxscreen = getgdesc(GD_YPMAX);
   sprintf(temp, "*geometry: %dx%d+0+0", xmaxscreen, ymaxscreen);
   fallback_resources[0] = strdup(temp);
} else {
  xmaxscreen = 646;
  ymaxscreen = 486;
   fallback_resources[0] = "*geometry: 646x486+0+0";
   system("/usr/gfx/setmon NTSC");
}
```

Set up the overlay planes for the cursor.

```
if (cursor_planes < 0) {
  glxConfig[3].arg = 0;
} else {
  if (cursor_planes > 4) {
    cursor_planes = 4;
     fprintf(stderr,"Error: number of cursor planes (%d) must be
less than 4.\n");
    }
   glxConfig[3].arg = cursor_planes;
}
```

Initalize Xt and create the shell widget.

```
 shell = XtVaAppInitialize(&app_context,
                           "ViewMaster",
                           options, XtNumber(options),
                           &argc, argv,
                           fallback_resources,
                           NULL);
```

If compiling for stereo in a window, then create a frame widget.

```
#ifdef X_SIW
     frame = XmCreateFrame (shell, "frame", args, n);
     XtManageChild (frame);
#endif
```

Create a bulletin board widget for displaying the graphics window.

```
n = 0;
 XtSetArg(args[0], XmNallowOverlap, FALSE); n++;
 XtSetArg(args[n], XmNmarginHeight, 0); n++;
```

```
   XtSetArg(args[n], XmNmarginWidth, 0); n++;
   XtSetArg(args[n], XmNnoResize, TRUE); n++;
   XtSetArg(args[n], XmNresizePolicy, XmRESIZE_NONE); n++;
   background = XtCreateManagedWidget("ViewMaster-bb",
                            xmBulletinBoardWidgetClass,
#ifdef X_SIW
                            frame,
#else
                            shell,
#endif
                            args, n);
```

Create a menu bar.

```
   menubar = XmCreateMenuBar(background,
                             "menuBar",
                             NULL,
                             0);
   XtManageChild(menubar);
```

Create the buttons on the menu bar.

```
   filebutton = XtVaCreateManagedWidget("File",
                                        xmCascadeButtonWidgetClass,
                                        menubar,
                                        NULL,
                                        0);
```

The menubar for stereo in a window does not include a button labeled **controls**.

```
#ifndef X_SIW
   controls_button = XtVaCreateManagedWidget("Controls",
                                        xmCascadeButtonWidgetClass,
                                        menubar,
                                        NULL,
                                        0);
#endif
```

Create the menu that is displayed when the file button is selected. The menu has one item to exit the program.

```
   filemenu = XmCreatePulldownMenu(menubar,
                                   "fileMenu",
                                   NULL, 0);
   quit = XtVaCreateManagedWidget("quit",
                                   xmPushButtonWidgetClass,
                                   filemenu,
                                   NULL);
   XtVaSetValues(filebutton, XmNsubMenuId, filemenu, NULL);
```

If the program is compiled for stereographics hardware, create a second button on the menu bar to toggle the hardware state of the monitor between 60 Hz and 120 Hz.

```
#ifndef X_SIW
   controls_menu = XmCreatePulldownMenu(menubar,
                                        "controlsMenu",
                                        NULL, 0);
   stereo_mode = XtVaCreateManagedWidget("toggle stereo",
                                          xmPushButtonWidgetClass,
                                          controls_menu,
                                          NULL);
   XtAddCallback(stereo_mode, XmNactivateCallback, ToggleStereo,
NULL);
   XtVaSetValues(controls_button, XmNsubMenuId, controls_menu,
NULL);
#endif
```

Realize the widget and initialize the stereo pointer.

```
   XtRealizeWidget(shell);
   initialize_stereo_pointer(background, app_context);
```

Create the viewmaster application widget.

```
   n = 0;
   XtSetArg(args[n], VmNprogramname,  *argv); argv++; argc--; n++;
   XtSetArg(args[n], VmNfilecount, argc); n++;
   XtSetArg(args[n], VmNfilenames, argv); n++;

   XtSetArg(args[n], StNname, "viewmaster-st"); n++;
   XtSetArg(args[n], GlxNglxConfig,glxConfig); n++;
   XtSetArg(args[n], GlxNoverrideColormap, TRUE); n++;
   XtSetArg(args[n], GlxNuseOverlay, TRUE); n++;
#ifdef X_SIW
   XtSetArg(args[n], StNappContext, &app_context); n++;
#endif

   viewmaster = XtCreateManagedWidget("viewmaster",
                                      viewMasterWidgetClass,
                                      background,
                                      args,
                                      n);
```

If the program is compiled for stereo in a window hardware, resize the shell window from fullscreen to the size of the image file, with room for the menu bar.

```
#ifdef X_SIW
   { Dimension height, width;
     n = 0;
      XtSetArg(args[n], XtNheight, &height); n++;
      XtSetArg(args[n], XtNwidth, &width); n++;
      XtGetValues(viewmaster,args,n);
     n = 0;
      XtSetArg(args[n], XtNheight, height+30); n++;
      XtSetArg(args[n], XtNwidth, width+6); n++;
      XtSetValues(shell,args,2);
      XtResizeWindow(shell);
   }
#endif
```

Add the callback functions to the application widget. Add the callback function for the **Quit** button in the menubar, passing the application widget as an argument.

```
   XtAddCallback(viewmaster, GlxNginitCallback, initCB, 0);
   XtAddCallback(viewmaster, StNredrawCallback, redraw_images, 0);
   XtAddCallback(quit, XmNactivateCallback, Exit, viewmaster);

   XtRealizeWidget(viewmaster);
```

Call the user interface mainloop.

```
   StAppMainLoop(app_context);
}
```

ViewMaster.h

This is the public header file for the viewmaster widget. Any application that wishes to use the viewmaster widget must include this header file. By contrast, the private header file, ViewMasterP.h is used by applications that wish to extend the viewmaster widget and must access its inner parts.

```
/*
* Xm header file for ViewMaster widget
* Copyright (C) 1994 Carnegie Mellon University
*   Scott Safier
*/
```

It is considered good style to wrap a header file in an `#ifdef`. This makes certain the header file is only loaded once.

```
#ifndef _XmViewmaster_h
#define _XmViewmaster_h

/*****************************************************************
*
 * ViewMaster Widget
 *

*****************************************************************/

/* Parameters:

 Name                 Class          RepType        Default Value
 ----                 -----          -------        -------------
 hoffset              Hoffset           int              0
 voffset              Voffset           int              0
 reverse              Reverse           Boolean          False
 stdout               Stdout            Boolean          False
 files                Files             Char **          NULL
 filesCount           FilesCount        int              0
 program              Program           Char *           NULL
*/
```

Define the resources.

```
#define VmNhoffset "hoffset"
#define VmChoffset "Hoffset"
#define VmNvoffset "voffset"
#define VmCvoffset "Voffset"
#define VmNreverse "reverse"
#define VmCreverse "Reverse"
#define VmNstandardout "stdout"
#define VmCstandardout "Stdout"
#define VmNfilenames "files"
#define VmCfilenames "Files"
#define VmNfilecount "filesCount"
#define VmCfilecount "FilesCount"
#define VmNprogramname "program"
#define VmCProgramname "program"
```

Declare types and external variables.

```
typedef struct _ViewMasterRec *ViewMasterWidget;
typedef struct _ViewMasterClassRec *ViewMasterWidgetClass;
extern WidgetClass viewMasterWidgetClass;

#endif /* _XmViewmaster_h */
/* DON'T ADD STUFF AFTER THIS #endif */
```

ViewMasterP.h

This is the private header file for the viewmaster widget. Private header files are used by applications that extend the functionality of the widget, or by applications that must access the internal components of the widget.

```
/* Viewmaster Xt Widget File
* Copyright (C) 1994 Carnegie Mellon University
*    Scott A. Safier
*/

#ifndef _ViewMasterP_h
#define _ViewMasterP_h

#include <Xm/XmP.h>

#ifdef X_SIW
#include "SIWP.h"
#else
#include "StWindowP.h"
#endif

#include "ViewMaster.h"
```

The viewmaster-specific structure created for each instantiation of the viewmaster widget.

```
/* New fields for the ViewMaster widget instance record */
typedef struct {
  /* the files where the images are stored */
  char **files, *program_name;
  int num_files;
```

```
   /* offsets and other resources */
   int horizontal_offset, vertical_offset;
   Boolean reverse_images;
   Boolean standard_outputp;
   Boolean savep;

   /* images */
   struct stereo_image *lrimgs;

} ViewMasterPart;
```

The structure created for the viewmaster widget. It includes references to each structure above viewmaster in the widget hierarchy. The widget uses Xt's Core widget, and Motif's primitive widget. The stereo in a window paradigm uses the GL widget. Viewmaster is an application of the StereoWindowWidget, and lastly, the viewmaster part of the widget is included.

```
/* Full instance record declaration */
typedef struct _ViewMasterRec {
    CorePart core;
    XmPrimitivePart primitive;
#ifdef X_SIW
    GlxDrawPart glxDraw;
#endif
    StereoWindowPart stereowindow;
    ViewMasterPart viewmaster;
    } ViewMasterRec;
```

A widget class is a structure that is created once, automatically, by the X Intrinsics. It maintains static information about the widget. Neither the viewmaster widget nor the StereoWindowWidget define any specializations in their class, but must define a dummy field to keep the C compiler happy.

```
/* New fields for the ViewMaster widget class record */
typedef struct {int dummy;} ViewMasterClassPart;

/* Full class record declaration. */
typedef struct _ViewMasterClassRec {
    CoreClassPart core_class;
    XmPrimitiveClassPart primitive_class;
#ifdef X_SIW
    GlxDrawClassPart   glxDraw_class;
#endif
    StereoWindowClassPart stereowindow_class;
    ViewMasterClassPart ViewMaster_class;
    } ViewMasterClassRec;

/* Class pointer. */
extern ViewMasterClassRec viewMasterClassRec;

#endif /* _ViewMasterP_h */
```

ViewMaster.c

This is the executable code to implement the viewmaster widget.

```
/*
 * ViewMaster.c
 *
 * a widget which follows the mouse around
 */

# include <X11/Xos.h>
# include <stdio.h>
# include <X11/IntrinsicP.h>
# include <X11/StringDefs.h>
# include <X11/Xmu/Converters.h>
# include "StWindow.h"
# include "ViewMasterP.h"
# define X_STEREO
# include "stereo.h"
# include <X11/CoreP.h>
# include <X11/keysymdef.h>
```

Two macros to aid in the definition of the widget's resources

```
#define offset(field) XtOffset(ViewMasterWidget,viewmaster.field)
#define goffset(field) XtOffset(StereoWindowWidget,stereowindow.field)
```

Define the translation and action tables -- any key stroke will call the ShiftImage procedure.

```
static char defaultTranslations[] =
     "<Key>: ShiftImage()";

static void Exit(), ShiftImage();

static XtActionsRec actions[] = {
     { "ShiftImage", ShiftImage },
  };
```

The widget's resources. The syntax of this table is: resource name, resource class, resource type, resource size, where to store the value in the widget's data structure, the type of the default value, and the default value.

```
static XtResource resources[] = {
     {VmNhoffset, VmChoffset, XmRInt, sizeof(int),
        offset(horizontal_offset), XmRString, "0"},
     {VmNvoffset, VmCvoffset, XmRInt, sizeof(int),
        offset(vertical_offset), XmRString, "0"},
     {VmNreverse, VmCreverse, XmRBoolean, sizeof(Boolean),
        offset(reverse_images), XmRString, "FALSE"},
     {VmNstandardout, VmCstandardout, XmRBoolean, sizeof(Boolean),
        offset(standard_outputp), XmRString, "FALSE"},
     {VmNfilenames,VmCfilenames, XmRPointer, sizeof(char **),
        offset(files),XmRString,"0"},
     {VmNfilecount,VmCfilecount, XmRInt, sizeof(char **),
```

```
        offset(num_files),XmRString,"0"},
     {VmNprogramname,VmCProgramname, XmRString, sizeof(char *),
        offset(program_name), XmRString, 0},
  };

#undef offset
#undef goffset

static void Initialize(), Destroy(), Resize();
static Boolean SetValues();

static void ClassInitialize();
```

The definition of the class instance. First the core structure is defined, then each other widget in the hierarchy until the viewmaster class part is defined.

```
ViewMasterClassRec viewMasterClassRec = {
     { /* core fields */
     /* superclass          */   (WidgetClass) &stereoWindowClassRec,
     /* class_name          */   "ViewMaster",
     /* size                */   sizeof(ViewMasterRec),
     /* class_initialize */      ClassInitialize,
     /* class_part_initialize*/ NULL,
     /* class_inited        */   FALSE,
     /* initialize          */   Initialize,
     /* initialize_hook     */   NULL,
     /* realize             */   XtInheritRealize,
     /* actions             */   actions,
     /* num_actions         */   XtNumber(actions),
     /* resources           */   resources,
     /* num_resources       */   XtNumber(resources),
     /* xrm_class           */   NULL,
     /* compress_motion     */   TRUE,
     /* compress_exposure   */   TRUE,
     /* compress_enterleave */   FALSE,
     /* visible_interest    */   FALSE,
     /* destroy             */   Destroy,
     /* resize              */   Resize,
     /* expose              */   XtInheritExpose,
     /* set_values          */   SetValues,
     /* set_values_hook     */   NULL,
     /* set_values_almost   */   NULL,
     /* get_values_hook     */   NULL,
     /* accept_focus        */   NULL,
     /* version             */   XtVersion,
     /* callback_private    */   NULL,
     /* tm_table            */   XtInheritTranslations,
     /* query_geometry      */   XtInheritQueryGeometry,
     /* display_accelerator*/    XtInheritDisplayAccelerator,
     /* extension           */   NULL,
     },
     { /* XmPrimitive fields */
     /* border_highlight */            _XtInherit,
     /* border_unhighlight */               _XtInherit,
     /* translations */                XtInheritTranslations,
     /* arm_and_activate */                 NULL,
     /* syn resources */                    NULL,
     /* num syn resources */                0,
```

```
    /* extensions */                          NULL,
                              },
#ifdef X_SIW
    { NULL, },
#endif
    { 0, },
    { 0, },
};
```

This function is called to initialize the class

```
static void ClassInitialize()
{
     XtAddConverter( XtRString, XtRBackingStore,
                      XmuCvtStringToBackingStore,
                    NULL, 0 );
}
```

The widget class's external variable declaration.

```
WidgetClass viewMasterWidgetClass = (WidgetClass)
&viewMasterClassRec;
```

The initialization function for widget instances. The purpose of the initialization is to verify that the values of the resources that have been defined in this instantiation are appropriate. The function also sets any private fields in data structure.

```
static void Initialize (Widget greq, ViewMasterWidget w,ArgList
args,
               Cardinal *numargs)
{
  /* read the image files */
  if (w->viewmaster.num_files == 1) {
     fprintf(stdout,"Reading stereo image in file %s...\n", w-
>viewmaster.files[0]);
     w->viewmaster.lrimgs = read_stereo_image(w-
>viewmaster.files[0]);
     fprintf(stdout,"done\n");
  } else if (w->viewmaster.num_files == 2) {
     fprintf(stdout,"Reading stereo image in file %s and %s...\n",
          w->viewmaster.files[0], w->viewmaster.files[1]);
     w->viewmaster.lrimgs = read_lr_images(w->viewmaster.files[0],
w->viewmaster.files[1]);
  } else {
     register int i;
     char *program = w->viewmaster.program_name;
     fprintf(stderr,"Fatal error: unprocessed arguments, expected
one or two file names\n");
     for(i = 0; i < w->viewmaster.num_files; i++)
        fprintf(stderr,"%s ",w->viewmaster.files[i]);
     fprintf(stderr,"\n");
     fprintf(stderr,"%s understands all the basic resource settings
from Xt\n",
         program);
     fprintf(stderr," in addition, these resources may be set for
%s\n",
```

```
          program);
     fprintf(stderr,"  *reverse    - swap the left and right
images\n");
     fprintf(stderr,"  *stdout     - save the image to standard
output on exit\n");
     fprintf(stderr,"  *hoffset # - horizontal offset in
pixels\n");
     fprintf(stderr,"  *voffset # - vertical offset in pixels\n");
     fprintf(stderr, "use -xrm <resource string> to set these
options from the command line\n");
     fprintf(stderr, "usage: %s <options> <file name>\n",program);
     fprintf(stderr, "       %s <options> <left file name> <right
file name>\n",
          program, program);
     exit(1);
  }
  w->viewmaster.horizontal_offset *= 2;
  w->viewmaster.vertical_offset *= 2;
  w->core.height = w->viewmaster.lrimgs->height;
  w->core.width = w->viewmaster.lrimgs->width;

#ifdef X_SIW
  w->core.x = 0;
  w->core.y = 25;
#else
  w->core.x = (1280 - w->core.width)/2;
  w->core.y = (YMAXSTEREO - w->core.height)/2;
#endif
  if (w->viewmaster.reverse_images == TRUE) {
    register unsigned long *temp;
    register StereoImage image = w->viewmaster.lrimgs;

    temp = image->right;
    image->right = image->left;
    image->left = temp;
  }
  InitializeStereoWindow (greq, (Widget) w, args, numargs);
  XtAugmentTranslations(w,
                    XtParseTranslationTable(defaultTranslations));
}
```

This function is called when the widget needs to resize its window.

```
static void Resize (Widget gw)
{
  ViewMasterWidget   w = (ViewMasterWidget) gw;

  w->core.height = w->viewmaster.lrimgs->height;
  w->core.width = w->viewmaster.lrimgs->width;
}
```

This function is called when the widget is to be destroyed.

```
static void Destroy (gw)
     Widget gw;
{
     ViewMasterWidget w = (ViewMasterWidget)gw;
```

```
       free(w->viewmaster.lrimgs->left);
       free(w->viewmaster.lrimgs->right);
       free(w->viewmaster.lrimgs);
       DestroyStereoWidget((StereoWindowWidget) w);
}
```

This function is called whenever a resource value is changed. Like the initialization function, it does not directly change values in the instance data structure. Instead, the function confirms that the values as set are valid, and executes any actions associated with the changed value.

```
static Boolean SetValues (ViewMasterWidget current,
                          ViewMasterWidget request,
                          ViewMasterWidget new)
{
  /* offsets and other resources will need to be set here*/
  if (current->viewmaster.vertical_offset != new-
>viewmaster.vertical_offset) {
      if (abs(new->viewmaster.vertical_offset)>=new-
>viewmaster.lrimgs->height) {
        if (new->viewmaster.vertical_offset < 0)
      new->viewmaster.vertical_offset= -(new->viewmaster.lrimgs-
>height - 1);
       else
      new->viewmaster.vertical_offset = new->viewmaster.lrimgs-
>height - 1;
     }
     if (!(new->viewmaster.vertical_offset % 2))
       return TRUE;
     else
       return FALSE;
  }

  if (current->viewmaster.horizontal_offset !=
        new->viewmaster.horizontal_offset) {
      if (abs(new->viewmaster.horizontal_offset) >=
      new->viewmaster.lrimgs->width) {
        if (new->viewmaster.horizontal_offset < 0)
      new->viewmaster.vertical_offset= -(new->viewmaster.lrimgs-
>width - 1);
       else
      new->viewmaster.horizontal_offset= new->viewmaster.lrimgs-
>width - 1;
     }
     if (!(new->viewmaster.horizontal_offset % 2))
       return TRUE;
     else
       return FALSE;
  }
}

/* User Interface Functions */
```

This is the function referenced in the action table. It is called everytime there is a key event.

```
static void ShiftImage (ViewMasterWidget w, XEvent *event,
                String *params,
                Cardinal *num_params)
{
  char buffer[5];
  KeySym keysym;
  Arg arg;

  bzero(buffer,5);
  XLookupString(event,buffer,20,&keysym,NULL);
  switch (keysym) {
  case XK_space:
    break;
  case XK_Left:
  case XK_Right:
    XtSetArg(arg,VmNhoffset,
          w->viewmaster.horizontal_offset +
          ((keysym == XK_Left) ? 1 : -1));
    XtSetValues(w,&arg,1);
    break;
  case XK_Down:
  case XK_Up:
    XtSetArg(arg,VmNvoffset,
          w->viewmaster.vertical_offset +
          ((keysym == XK_Left) ? 1 : -1));
    XtSetValues(w,&arg,1);
    break;
  }
}
```

Example: Stereo3DWidget

This appendix provides an example of using the `Stereo3DWidget`. It is a simple program that reads a sequence of triangles from a data file and draws the corresponding image on the screen. Unlike the previous example, it does not define its own widget, making this a much simpler program.

```
/* Copyright (C) 1994 Carnegie Mellon University */
/* All Rights Reserved */
/* Written by Scott A. Safier */

/* Simple program to demonstrate stereo viewing of data stored as */
/* a series of triangular meshes */

/* this define must be done if we are using X windows and the
stereo library */
```

First we define the `X_STEREO` marker and indicate the include files. Note that the private header file for the `Stereo3DWidget` is not defined. Because this is an application, it does not need to access the internals of the widget.

```
#ifndef X_STEREO
#define X_STEREO
#endif

#include <stdio.h>
#include <malloc.h>
/* X includes */
#include <X11/Intrinsic.h>
#include <X11/Xatom.h>
#include <X11/StringDefs.h>
#include <X11/Shell.h>

/* motif includes */
#include <Xm/Xm.h>
#include <Xm/BulletinB.h>
#include <Xm/PushB.h>
#include <Xm/CascadeB.h>
#include <Xm/MessageB.h>
#include <Xm/RowColumn.h>

/* stereo window includes */
#include "St3D.h"
#include "stereo.h"
#include <X11/Xirisw/GlxMDraw.h>
#include <gl/get.h>

#include "cyber.h"
#include "light.h"

/****************************************************************/
/*                   APPLICATION SPECIFIC FUNCTIONS
*/
/* parse_vertex_file -- read the data file
*/
```

```
/* generate_triangles -- render the data
*/
/*****************************************************************/

/* this function parses a file of vertices organized as triangular */
/* meshes                                                          */
void parse_vertex_file(char *name, Mesh m)
{
  FILE *file;
  char tmp[100];
  register int i;

  if (((int)(file = fopen(name,"r"))) < 0) {
    perror("fopen");
    return;
  }
  do fscanf(file,"%s:",&tmp); while(strcmp(tmp,"new_mesh:") != 0);
  fscanf(file,"%d %d\nn",&(m->vertex_count),&(m->mesh_size));
  fprintf(stderr,"Reading %d vertices and normals\n",m-
>vertex_count);

  m->points = calloc(sizeof(Vertex),m->vertex_count);
  m->triangles = calloc(sizeof(int *), m->mesh_size);

  for (i = 0; i < m->vertex_count; i++) {
    register Vertex v = (Vertex) malloc(sizeof(struct vertex));
    m->points[i] = v;
    fscanf(file,"vertex: %f %f %f\n",&(v->pnt[0]),&(v-
>pnt[1]),&(v->pnt[2]));
  }

  for (i = 0; i < m->vertex_count; i++) {
    register Vertex v = m->points[i];
    fscanf(file,"normal: %f %f %f\n",
         &(v->normal[0]),&(v->normal[1]),&(v->normal[2]));
  }

  fprintf(stderr,"Reading %d triangles\n",m->mesh_size);
  for (i = 0; i < m->mesh_size; i++) {
    register int *tri = calloc(sizeof(int),3);
    m->triangles[i] = tri;
    fscanf(file,"triangle: %d %d
%d\n",&(tri[0]),&(tri[1]),&(tri[2]));
  }
  fclose(file);
}

/* this function draws the triangles */
void generate_triangles(Mesh m)
{
  register int i;
  for (i = m->mesh_size-1; i >= 0; i--) {
    bgnpolygon();
     n3f(m->points[m->triangles[i][0]]->normal);
     v3f(m->points[m->triangles[i][0]]->pnt);
     n3f(m->points[m->triangles[i][1]]->normal);
     v3f(m->points[m->triangles[i][1]]->pnt);
     n3f(m->points[m->triangles[i][2]]->normal);
```

```
     v3f(m->points[m->triangles[i][2]]->pnt);
    endpolygon();
  }
}

/************************************************************/
/*                    WIDGET METHODS                        */
/* the following functions are invoked internal to widgets  */
/*  redraw_scene - method for StereoWindowWidget            */
/*    This method must redraw an image for a window.  The   */
/*    method is past the StWindowCallbackStruct, that includes */
/*    the X window id, the Gl background color (as opposed to  */
/*    the background color in X format),and whether the StLeft */
/*     image or the StRight image should be displayed       */
/*                                                          */
/*  Exit - method for XmCascadeButtonWidget                 */
/*    this method causes the program to exit                */
/*                                                          */
/*  ToggleStereo - method for XmCascadeButtonWidget         */
/*    this method toggles between mono and stereo mode      */
/*                                                          */
/*  InitCB - method for StereoWindowWidget                  */
/*    this optional method performs any window specific     */
/*    initializations for each of the pair of X windows in the */
/*    StereoWindowWidget                                    */
/************************************************************/

/* redraw_scene
 *  this method is required by StereoWindowWidget.
 *  it redisplays the image for ONE window
 *  the user must establish a perspective transformation,
 *  the viewpoint
 *  and the target point prior to rendering anything
 *  the user can assume that any perspective, eye point, and
 *  target point that had been initialized for this window
 *  (and this widget)
 *  have been re-established prior to this method being called
 *
 *  NOTE: DO NOT CALL swapbuffers() in this routine.  if you do,
 *  the left and right fields may not be synchronized.
 *  swapbuffers() is
 *  called for the left and right fields internal to the widget
 */
static void redraw_scene(Stereo3DWidget w, struct mesh *m,
                StWindowCallbackStruct *call_data)
{
   czclear(call_data->background,getgdesc(GD_ZMAX));
   SetMaterial(MAT_WHITEPLASTIC,w);
   generate_triangles(m);
}

/* this method is invoked when QUIT is selected from the FILE menu
*/
static void Exit (Widget w, XtPointer client_data, XtPointer
call_data)
{
#ifndef X_SIW
   stereo_off();
```

```
#endif
   XtDestroyWidget((Widget) client_data);
   exit(0);
}
```

The stereo in a window paradigm does not permit the software to turn stereo mode on and off, so the toggle stereo method is only useful in the Stereographics paradigm.

```
#ifndef X_SIW
/* this method is invoked when TOGGLE STEREO is selected from the
CONTROLS
 * menu
 */
static void ToggleStereo(Widget w, XtPointer client_data,
XtPointer call_data)
{
  static int mode = 0;
  if (!mode) {
    stereo_on(1);
    mode = 1;
  } else {
    stereo_off();
    mode = 0;
  }
}
#endif

/* InitCB
 *  This is an optional method for a StereoWindowWidget
 *  It should call functions that initialize things for each of the
 *  pair of stereo windows in the widget.  In this example,
 *  we must initialize the lighting parameters for each window,
 *  but we need only set the defaults for the widgets once (in the
 *  main program
 */
static void
initCB(Widget w, caddr_t client_data,StWindowCallbackStruct
*call_data)
{
     GLXwinset(XtDisplay(w), call_data->window);
     mmode(MVIEWING);
     DefineMaterials();
     DefineLights();
     DefineLightModels();
}

/* SetupEyeTraining
 * this routine initializes eye training
 */
static void SetupEyeTraining(Widget bw, caddr_t client_data,
                    XtPointer call_data)
{
  Arg arg;
   StereoWindowWidget w = (StereoWindowWidget) client_data;
   XtSetArg(arg,StNtraining,1);
   XtSetValues(w,&arg,1);
}
```

These next two functions create GL display lists for drawing the two-dimensional and three-dimensional cursor.

```
/* Cursors */
static GL_Object make_2dcursor_object()
{
  float v[4][3];
  GL_Object obj;

  v[0][0] = v[2][0] = v[3][1] = v[2][1] =
    -(v[1][0] = v[3][0] = v[0][1] = v[1][1] = 5);
  v[0][2] = v[1][2] = v[2][2] = v[3][2] = 0 ;

  makeobj(obj = genobj());
  linewidth(3);
  bgnline(); v3f(v[0]); v3f(v[3]); endline();
  bgnline(); v3f(v[2]); v3f(v[1]); endline();
  linewidth(1);
  closeobj();
  return obj;
}

static GL_Object make_3dcursor_object()
{
  float v[4][3];
  GL_Object obj;

  v[0][0] = v[0][1] = v[0][2] = v[1][2] = v[2][2] = v[3][0] =
v[3][2] = 0;
  v[1][0] = v[1][1] = v[2][1] = -(v[2][0] = -1);
  v[3][1] = 4;
  makeobj(obj = genobj());
  linewidth(3);
  bgnline(); v3f(v[1]); v3f(v[0]); v3f(v[2]); endline();
  bgnline(); v3f(v[0]); v3f(v[3]); endline();
  linewidth(1);
  closeobj();
  return obj;
}


/*****************************************************************/
/*                        MAIN PROGRAM                           */
/*****************************************************************/

main(int argc,char **argv)
{
  /* this is the data being rendered */
  struct mesh m;

  /* variables controlling screen size */
  Dimension xmaxscreen, ymaxscreen;
  int NTSCmodep;

  /* All the widgets I need */
  Widget shell, background, cyber, stwin,
         menubar, filebutton, controls_button,
```

```
        filemenu, quit,
        controls_menu, stereo_mode, train_eyes;

  /* stuff for X, Xt and motif */
  XtAppContext app_context;
  XEvent event;
  XExposeEvent expose;
  Arg args[20];
  int n;
  String fallback_resources[] = {
#ifndef X_SIW
    NULL,
#endif
    ".clientDecoration: none",
    "*background: black",
    NULL};

  /* stuff for GL mixed model programming */
  /* The GLX configuration parameter:
   *      Double buffering
   * RGB mode
   *    Zbuffer
   * overlay
   * nothing else special
   */
  GLXconfig glxConfig [] = {
    { GLX_NORMAL, GLX_DOUBLE, TRUE },
    { GLX_NORMAL, GLX_RGB, TRUE },
    { GLX_NORMAL,GLX_ZSIZE, GLX_NOCONFIG },
    { GLX_OVERLAY, GLX_BUFSIZE, 4},
    { 0, 0, 0 },
  };
  extern int cursor_planes;
  extern void input_cb(StereoWindowWidget, caddr_t,
                       GlxDrawCallbackStruct *);
```

The geometry for stereo in a window is set up by the user interacting with the window manager.

```
#ifndef X_SIW
  /* set up geometry of window */
  NTSCmodep = getenv("NTSC") ? 1 : 0;
  if (!NTSCmodep) {
    char temp[100];
    bzero(temp, 100);
    xmaxscreen = getgdesc(GD_XPMAX);
    ymaxscreen = getgdesc(GD_YPMAX);
    sprintf(temp, "*geometry: %dx%d+0+0", xmaxscreen, ymaxscreen);
    fallback_resources[0] = strdup(temp);
  } else {
    xmaxscreen = 646;
    ymaxscreen = 486;
    fallback_resources[0] = "*geometry: 646x486+0+0";
#ifndef DEBUG
    system("/usr/gfx/setmon NTSC");
#endif
  }
#endif
```

```
   /*set up overlay planes for cursor */
   if (cursor_planes < 0) {
     glxConfig[3].arg = 0;
   } else {
     if (cursor_planes > 4) {
       cursor_planes = 4;
       fprintf(stderr,"Error: number of cursor planes (%d) must be
less than 4.\n");
     }
     glxConfig[3].arg = cursor_planes;
   }

   /* create shell window and initialize X, Xt and motif*/
   shell = XtVaAppInitialize(&app_context,
                      "Cyber",
                      NULL, 0,
                      &argc, argv,
                      fallback_resources,
                      NULL);

   /* check argument count */
   if (argc != 2) {
     fprintf(stderr,"wrong number of arguments to %s\n",argv[0]);
     fprintf(stderr,"usage: %s <x options> file\n",argv[0]);
   }

   /* second argument must be file --
      discard program and parse file */
   fprintf(stderr,"%s\n",*argv);
   argv++;
   fprintf(stderr,"%s\n",*argv);
   parse_vertex_file(*argv,&m);
```

The next block of code creates a bulletin board widget. This widget is a composite widget that allows child windows to be placed arbitrarily with the bulletin board.

```
   /* create the geometry manager laying out children */
   n = 0;
   XtSetArg(args[0], XmNallowOverlap, FALSE); n++;
   XtSetArg(args[n], XmNmarginHeight, 0); n++;
   XtSetArg(args[n], XmNmarginWidth, 0); n++;
#ifdef X_SIW
   XtSetArg(args[n], XmNnoResize, FALSE); n++;
   XtSetArg(args[n], XmNresizePolicy, XmRESIZE_ANY); n++;
   XtSetArg(args[n], XmNshadowType, XmSHADOW_ETCHED_OUT); n++;
#else
   XtSetArg(args[n], XmNnoResize, TRUE); n++;
   XtSetArg(args[n], XmNresizePolicy, XmRESIZE_NONE); n++;
#endif
   background = XtCreateManagedWidget("Cyber-bb",
                            xmBulletinBoardWidgetClass,
                            shell,
                            args, n);
```

The next few widgets create a menu bar, and add pulldown menus to the menu bar. Each item in the menu bar, and each item in the menu, must be specified with its own widget.

```
   /* create menu bar */
   menubar = XmCreateMenuBar(background,
                     "menuBar",
                     NULL,
                     0);
   XtManageChild(menubar);

   /* create buttons in menu bar */
   filebutton = XtVaCreateManagedWidget("File",
                              xmCascadeButtonWidgetClass,
                              menubar,
                              NULL,
                              0);
   controls_button = XtVaCreateManagedWidget("Controls",
                                 xmCascadeButtonWidgetClass,
                                 menubar,
                                 NULL,
                                 0);
   /* create menu */
   filemenu = XmCreatePulldownMenu(menubar,
                        "fileMenu",
                        NULL, 0);
   quit = XtVaCreateManagedWidget("quit",
                       xmPushButtonWidgetClass,
                       filemenu,
                       NULL);
   XtVaSetValues(filebutton, XmNsubMenuId, filemenu, NULL);

   controls_menu = XmCreatePulldownMenu(menubar,
                              "controlsMenu",
                              NULL, 0);
```

Toggle stereo is not defined in the stereo in a window paradigm.

```
#ifndef X_SIW
   stereo_mode = XtVaCreateManagedWidget("toggle stereo",
                             xmPushButtonWidgetClass,
                             controls_menu,
                             NULL);
#endif
   train_eyes = XtVaCreateManagedWidget("train eyes",
                             xmPushButtonWidgetClass,
                             controls_menu,
                             NULL);
   XtVaSetValues(controls_button, XmNsubMenuId, controls_menu,
NULL);
#ifndef X_SIW
   XtAddCallback(stereo_mode, XmNactivateCallback, ToggleStereo,
NULL);
#endif

   /* draw the shell window on the screen, initialize mouse
      and cursor */
```

```
   XtRealizeWidget(shell);
   initialize_stereo_pointer(background, app_context);
```

Create the `Stereo3DWidget`.

```
   /* make the stereowindow widget for this application */
   n = 0;
   /* its name -- required */
   XtSetArg(args[n], StNname, "cyber-st");n++;
   /* provide the information to configure for GL */
   XtSetArg(args[n], GlxNglxConfig,glxConfig); n++;
   XtSetArg(args[n], GlxNoverrideColormap, TRUE); n++;
   XtSetArg(args[n], GlxNuseOverlay, TRUE); n++;
   /* set the X and Y locations of the window */
#ifdef X_SIW
   XtSetArg(args[n], XtNheight,ymaxscreen); n++;
   XtSetArg(args[n], XtNwidth, xmaxscreen); n++;
#else
   XtSetArg(args[n], XtNx,(XMAXSCREEN-(YMAXSTEREO*2))/2); n++;
   XtSetArg(args[n], XtNy,0); n++;
   /* set the window height and width */
   XtSetArg(args[n], XtNheight,YMAXSTEREO); n++;
   XtSetArg(args[n], XtNwidth, YMAXSTEREO*2); n++;
#endif
   XtSetArg(args[n], StNappContext, &app_context); n++;
```

We do not create a managed widget here. If the widget were managed, it would automatically be realized. The initialization callback needs to be added to the widget prior to its being realized.

```
   cyber = XtCreateWidget("cyber",
                          stereo3DWidgetClass,
                          background,
                          args, n);

   /* assign the methods to the widgets */
   XtAddCallback(cyber, GlxNginitCallback, initCB, 0);
   XtAddCallback(cyber, StNredrawCallback, redraw_scene, &m);

   /* the Exit method takes a widget to destroy, so pass the */
   /* cyber widget as an argument to the method */
   XtAddCallback(quit, XmNactivateCallback, Exit, cyber);
   XtAddCallback(train_eyes, XmNactivateCallback, SetupEyeTraining,
cyber);

   /* draw the widget on the screen */

   XtManageChild(cyber);
   XtRealizeWidget(cyber);
```

Now that the widget is created, we can initialize the perspective geometry, the position of the synthetic eyes and the position of the point being viewed.

```
   /* set the viewing parameters for the widget  */
   set_cursor_depth(0,(Stereo3DWidget) cyber);
   set_3d_cursors(make_2dcursor_object(), make_3dcursor_object(),
```

```
            (StereoWindowWidget) cyber);

    set_projection((Stereo3DWidget) cyber,1000,1.2857,
                   10.5,13.5,100.0,24.0, 1.16, /* 2.54 */
                   1.2857);

    st_lookat(0.0,-0.075,0.5,0,-0.075,0,0,(Stereo3DWidget) cyber);

    /* set the lighting parameters for the widget */
    SetLight(LIGHT_WHITE1_INF, (Stereo3DWidget) cyber);
    SetLight(LIGHT_WHITE_INF, (Stereo3DWidget) cyber);
    SetLight(LIGHT_WHITE2_INF, (Stereo3DWidget) cyber);
    SetLight(LIGHT_WHITE4_INF, (Stereo3DWidget) cyber);
    SetLightModel(MODEL_INFINITE, (Stereo3DWidget) cyber);

    event.xexpose = expose;
    XSendEvent(XtDisplay(cyber), XtWindow(cyber), FALSE,
            ExposureMask, &event);
```

The callback for the user interface would be added here.

```
    StAppMainLoop(app_context);
}
```

Glossary

projecting - The drawing of straight lines or `rays' according to some particular method through every point of a given figure, so as to fall upon or intersect a surface - the **surface of projection** - and produce upon it a new figure each point of which corresponds to a point of the original figure. Hence, each of such points of the resulting figure, is said to be the **projection** of a point of the original one; the whole resulting figure is said to be the **projection** of the original.

to **project** - to do **projecting**; to create a **projection**.

central projection -- rays are drawn from one point - the **center of projection**.

parallel projection all rays are parallel i. e. have the same direction - the **direction of projection**. It can be considered as a **central projection** when the **center of projection** is at infinity.

planar projection the **surface of projection** is a plane - the **plane of projection**.

orthogonal projection - **planar projection** in which the rays are at right angle to the **surface of projection**. If an **orthogonal projection** is *planar*, it is a special case of **parallel projection**; if an **orthogonal projection** has a sphere as its **surface of projection**, it is a special case of **central projection**.

synthetic objects - objects that do not and did not exist in the real world; one "creates" these objects just for viewing them. The set of **synthetic objects** is called **synthetic world**. The **synthetic world** also may contain **synthetic surface(s) of projection** and the **synthetic center(s) of projection**. The last can also be called **synthetic eye(s)**.

screen - real physical screen of a display, in a cinema theater etc. If a screen is rectangular its sizes are referred as the *width* and *height* of the **screen**.

viewed area of projection - the bounded part of a **synthetic surface of projection** that is mapped onto a **screen**.

projection window - a part of the **screen** on which **viewed area of projection** is mapped. The **projection window** simulates the **viewed area of projection** and so must have the same shape and size. For correct viewing perception of **synthetic world** the position of a viewer's eye relative to the **projection window** must be the same that the position of the **synthetic eye** relative to the **viewed area of projection**.

binocular viewing system -- uses 2 different **synthetic eye**s that correspond to the 2 human eyes. It may use 2 different **viewed areas of projection** - one per **synthetic eye** and correspondingly 2 different **projection window**s - one per human eye. Or **binocular viewing system** may use the same **viewed area of projection** for both **synthetic eye**s and accordingly the same **projection window** for both human eyes. In this case this combined **projection window** is called a **binocular window**. This Silicon Graphics defines the viewed area of projection in terms of the aspect ration between the height and width of the screen, the field of view, and the **line of sight**. The line of sight is a straight line defined by the position of the synthetic eyes and some point in the synthetic world.

image -- *1)* a projection of a synthetic world on a screen; *2)* a precomputed projection of the real world or of a synthetic world.

field -- one half of a 3D stereoscopic image. The **left field** and the **right field** combine to form a stereo pair, where the left field is the image seen by the left eye, and the right field is seen by the right eye.

Index

Name

Synopsis

public headers:
private header:
class name:
class pointer:
instantiation:

Class Hierarchy

Description

Resources

name	*type*	*default*	*description*
foo	bar	baz	bek

Translations and Actions

Callback functions

Name

Synopsis

inputs

Description

Usage

Algorithm

Structures