Chapter 4

Interactive Input Methods and Graphical User Interfaces

4.1 Overview

In order to be able to interact with the graphical image input methods are required. These can be used to just change the location and orientation of the camera, or to change specific settings of the rendering itself.

Different devices are more suitable for changing some settings then others. In this chapter we will specify different types of these devices and discuss their advantages.

4.2 Input methods

Input methods can be classified using the following categories:

- Locator
- Stroke
- String
- Valuator
- Choice
- Pick

4.2 Input methods

Locator

A device that allows the user to specify one coordinate position. Different methods can be used, such as a mouse cursor, where a location is chosen by clicking a button, or a cursor that is moved using different keys on the keyboard. Touch screens can also be used as locators; the user specifies the location by inducing force onto the desired coordinate on the screen.

4.2 Input methods

Stroke

A device that allows the user to specify a set of coordinate positions. The positions can be specified, for example, by dragging the mouse across the screen while a mouse button is kept pressed. On release, a second coordinate can be used to define a rectangular area using the first coordinate in addition.

4.2 Input methods

String

A device that allows the user to specify text input. A text input widget in combination with the keyboard is used to input the text. Also, virtual keyboards displayed on the screen where the characters can be picked using the mouse can be used if keyboards are not available to the application.

4.2 Input methods

Valuator

A device that allows the user to specify a scalar value. Similar to string inputs, numeric values can be specified using the keyboard. Often, up-down-arrows are added to increase or decrease the current value. Rotary devices, such as wheels can also be used for specifying numerical values. Often times, it is useful to limit the range of the numerical value depending on the value.

4.2 Input methods

Choice

A device that allows the user to specify a menu option. Typical choice devices are menus or radio buttons which provide various options the user can choose from. For radio buttons, often only one option can be chosen at a time. Once another option is picked, the previous one gets cleared.

4.2 Input methods

Pick

A device that allows the user to specify a component of a picture. Similar to locator devices, a coordinate is specified using the mouse or other cursor input devices and then back-projected into the scene to determine the selected 3-D object. It is often useful to allow a certain “error tolerance” so that an object is picked even though the user did not exactly onto the object but close enough next to it. Also, highlighting objects within the scene can be used to traverse through a list of objects that fulfill the proximity criterion.

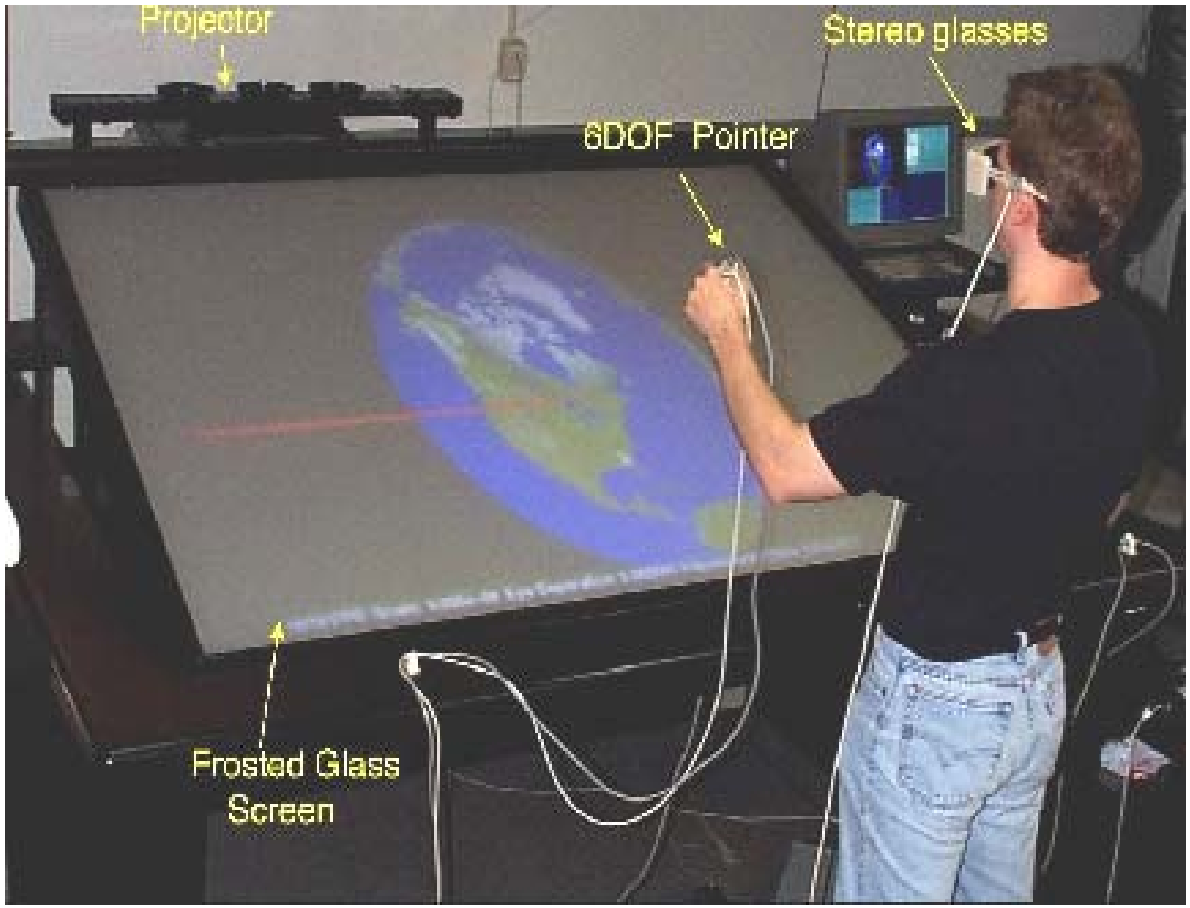
4.2 Input methods

Pick (continued)

Certain applications do not allow the use of mouse or keyboard. In particular, 3-D environments, where the user roams freely within the scene, mouse or keyboard would unnecessarily bind the user to a certain location. Other input methods are required in these cases, such as a wireless gamepad or a 3-D stylus, that is tracked to identify its 3-D location.

4.2 Input methods

Pick (continued)



4.2 Input methods

Pick (continued)

PHANTOM Omni from SensAble

Usually, these devices come with their own API.

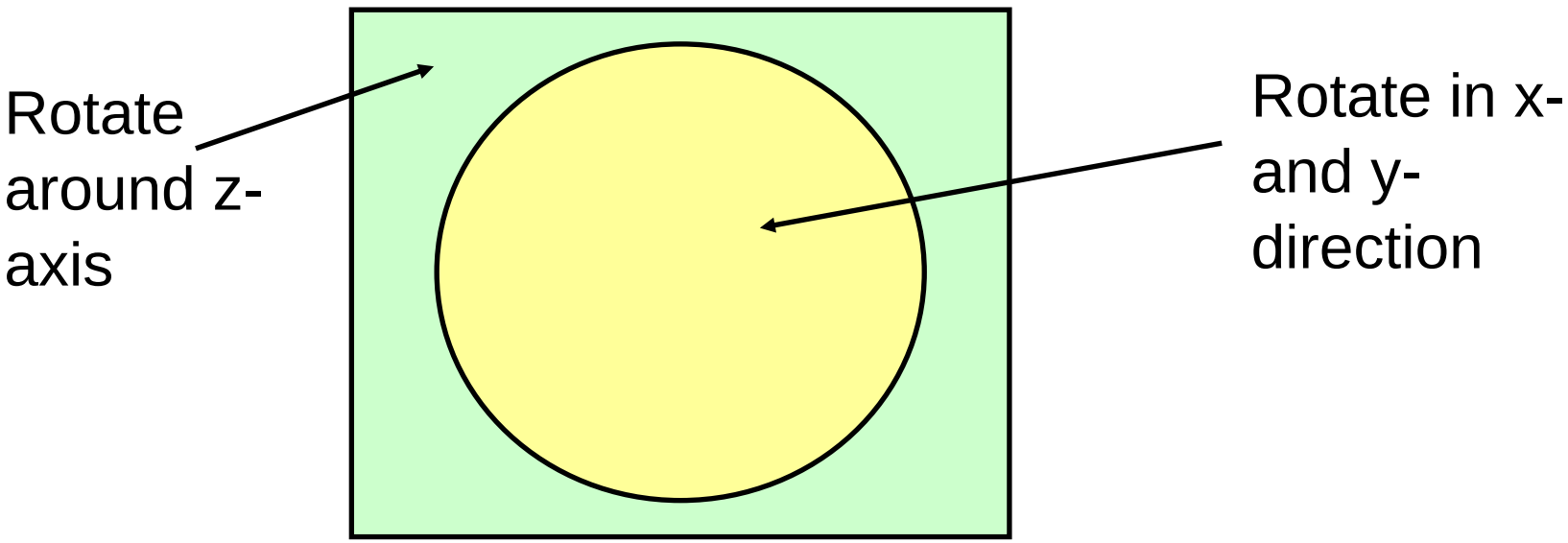
4.3 Mouse Control

On a desktop computer, the mouse can be used to manipulate the viewing direction or pick objects. By clicking a mouse button followed by dragging, the camera can be moved closer or further away from the scene (zooming) or moved within a plane parallel to the scene (panning). The rotation of the camera can be mapped directly to the mouse movement resulting in a rotation in x- and y-direction with respect to the local viewing coordinate system. Better results for rotating the camera can be achieved by using a virtual trackball.

4.3 Mouse Control

Virtual trackball

The idea is to project mouse movement on an hypothetical sphere filling the 3D window and to apply the rotation resulting from the sphere being manipulated when the mouse button is pressed to the object.



4.3 Mouse Control

GLUT

Mouse input can be implemented using GLUT. Callback functions are used by GLUT to allow application dependent input.

The following callback function indicates a pressed mouse button:

```
void mouseFcn (Glint button, Glint action,
               Glint x, Glint y);

glutMouseFunc (mouseFcn);
```

Valid mouse buttons (`action`) are `GLUT_LEFT_BUTTON`, `GLUT_RIGHT_BUTTON`, and `GLUT_MIDDLE_BUTTON`.

4.3 Mouse Control

GLUT (continued)

Mouse motion can be tracked while a mouse button is pressed by registering the following callback:

```
void fcnDoSomething (Glint x, Glint y);

glutMotionFunc (fcnDoSomething);
```

If mouse movement is to be tracked without any mouse button pressed the following callback can be used:

```
void fcnDoSomethingElse(Glint x, Glint y);

glutPassiveMotionFunc(fcnDoSomeThingElse);
```

Note: OpenGL counts from the bottom line so that the y coordinate is reversed!

4.4 Keyboard Control

In GLUT, a callback function can be registered that is called whenever a key is pressed on the keyboard. For standard keys, the following can be used:

```
void keyFcn(GLubyte key, Glint x, Glint y)

glutKeyboardFunc(keyFcn);
```

In addition to the pressed key, the current mouse position in screen coordinates is given.

The parameter `key` is assigned a character value or the corresponding ASCII code.

4.4 Keyboard Control

For special keys, such as cursor or function keys, a different callback has to be registered:

```
void specialKeyFcn (Glint specialKey,
                    Glint x, Glint y);
glutSpecialFunc (specialKeyFcn);
```

The parameter `specialKey` is assigned symbolic constants defined by GLUT depending on the pressed key. Valid constants, for example, are:

`GLUT_KEY_F1` through `GLUT_KEY_F12`, `GLUT_KEY_UP`, `GLUT_KEY_RIGHT`, `GLUT_KEY_PAGE_DOWN`, `GLUT_KEY_HOME`

4.5 OpenGL Picking

In OpenGL, we can interactively select objects by pointing to screen positions. However, the picking operations in OpenGL are not straightforward. Basically, picking is performed by using a designated pick window to form a revised view volume and then previously defined identifiers for those objects that intersect the revised view volume are stored in a pick-buffer array.

Thus, the following steps need to be taken in order to use OpenGL:

- Create and display a scene.
- Pick a screen position and, within the mouse callback function, do the following:

4.5 OpenGL Picking

- Set up a pick buffer.
- Activate the picking operations (selection mode).
- Initialize an ID name stack for object identifiers.
- Save the current projection and model-view matrices.
- Specify a pick window for the mouse input.
- Assign identifiers to objects and reprocess the scene using the revised view volume (pick information is then stored in the pick buffer)
- Restore the original projection and model-view matrices.
- Determine the number of objects that have been picked, and return to the normal rendering mode
- Process the pick information

4.5 OpenGL Picking

Set up a pick buffer

Set up the pick buffer to provide an array where the picked object identifiers will be stored

```
GLint pickBufferSize;
```

```
GLuint pickBuffer[pickBufferSize];
```

```
glSelectBuffer(pickBufferSize,pickBuffer);
```

The `glSelectBuffer` function must be invoked before the OpenGL picking operations (selection mode) are activated.

4.5 OpenGL Picking

Set up a pick buffer (continued)

In the pick buffer, the following information is stored:

- The stack position of the object, which is the number of identifiers in the name stack up to and including the position of the picked object.
- The minimum depth of the picked object
- The maximum depth of the picked object
- The list of identifiers in the name stack from the first (bottom) identifier o the identifier for the picked object.

4.5 OpenGL Picking

Activate the picking operations (selection mode)

The OpenGL picking operations are activated with

```
glRenderMode (GL_SELECT);
```

We can switch back to normal rendering once we are done picking objects using the paramter `GL_RENDER`:

```
nPicks = glRenderMode (GL_RENDER);
```

The function glRenderMode also gives us the number of picked objects as a side effect.

4.5 OpenGL Picking

Initialize an ID name stack for object identifiers

The ID name stack is initialized by issuing the command

```
glInitName ();
```

The ID stack is initially empty, and this stack can only be used in selection mode. To place an unsigned integer value on the stack, thus naming the next object within the scene, we can invoke the following function:

```
GLuint id
glPushName (id);
```

4.5 OpenGL Picking

Specify a pick window for the mouse input

A pick window within a selected viewport is defined using the following GLU function:

```
gluPickMatrix (xPick, yPick,
               widthPick, heightPick,
               vpArray);
```

The parameters `xPick` and `yPick` give the screen-coordinate location for the center of the pick window, while `widthPick` and `heightPick` define its dimensions. The parameter `vpArray` designates an integer array containing the coordinate position and size for the current viewport.

4.5 OpenGL Picking

Process the pick information

The picked objects can be processed by traversing the pick buffer:

```
GLuint objID, *ptr = pickBuffer;
for (int j=0; j<nPicks; j++) {
   objID = *ptr;
   printf (“stack position = %d\n”, objID);
   ptr++
   printf (“min depth = %f, “, (float)*ptr++));
   printf (“max depth = %f\n“, (float)*ptr++));
   printf (“stack IDs are: “);
   for (int k=0; k<objID; k++)
      printf (“ %d “, *ptr++)
   printf (“\n”);
}
```

4.6 User Interfaces

OpenGL Menu Functions

Simple pop-up menus can be created using GLUT directly. Various functions are available for setting up and accessing a variety of menus and associated sub-menus. The GLUT menu commands are placed in the procedure `main` along with other GLUT functions.

GLUT menus use a callback function for one entire menu. To identify the selected menu item, all menu items are provided with an identifier. This identifier is then passed to the callback function.

4.6 User Interfaces

Creating a GLUT Menu

A pop-up menu is created with the statement:

```
void menuFcn (Glint menuItemNumber);
glutCreateMenu (menuFcn);
```

Once the menu is created, we can add menu entries:

```
char *title;
GLint identifier;
glutAddMenuEntry (title, identifier);
```

To attach the menu to a certain mouse button issue:

```
glutAttachMenu (button);
```

4.6 User Interfaces

Creating a GLUT Menu

A pop-up menu is created with the statement:

```
void menuFcn (Glint menuItemNumber);
glutCreateMenu (menuFcn);
```

Once the menu is created, we can add menu entries:

```
char *title;
GLint identifier;
glutAddMenuEntry (title, identifier);
```

To attach the menu to a certain mouse button issue:

```
glutAttachMenu (button);
```

4.6 User Interfaces

Creating and managing multiple GLUT menus

The function `glutCreateMenu` returns an identifier itself that can be used to reference the created menu:

```
GLint menuID = glutCreateMenu (menuFcn);
```

To activate a menu for the current display window, we use the statement:

```
glutSetMenu (menuID);
```

This menu then becomes the current menu, which will pop up in the display window when the mouse button that has been attached to that menu us pressed.

4.6 User Interfaces

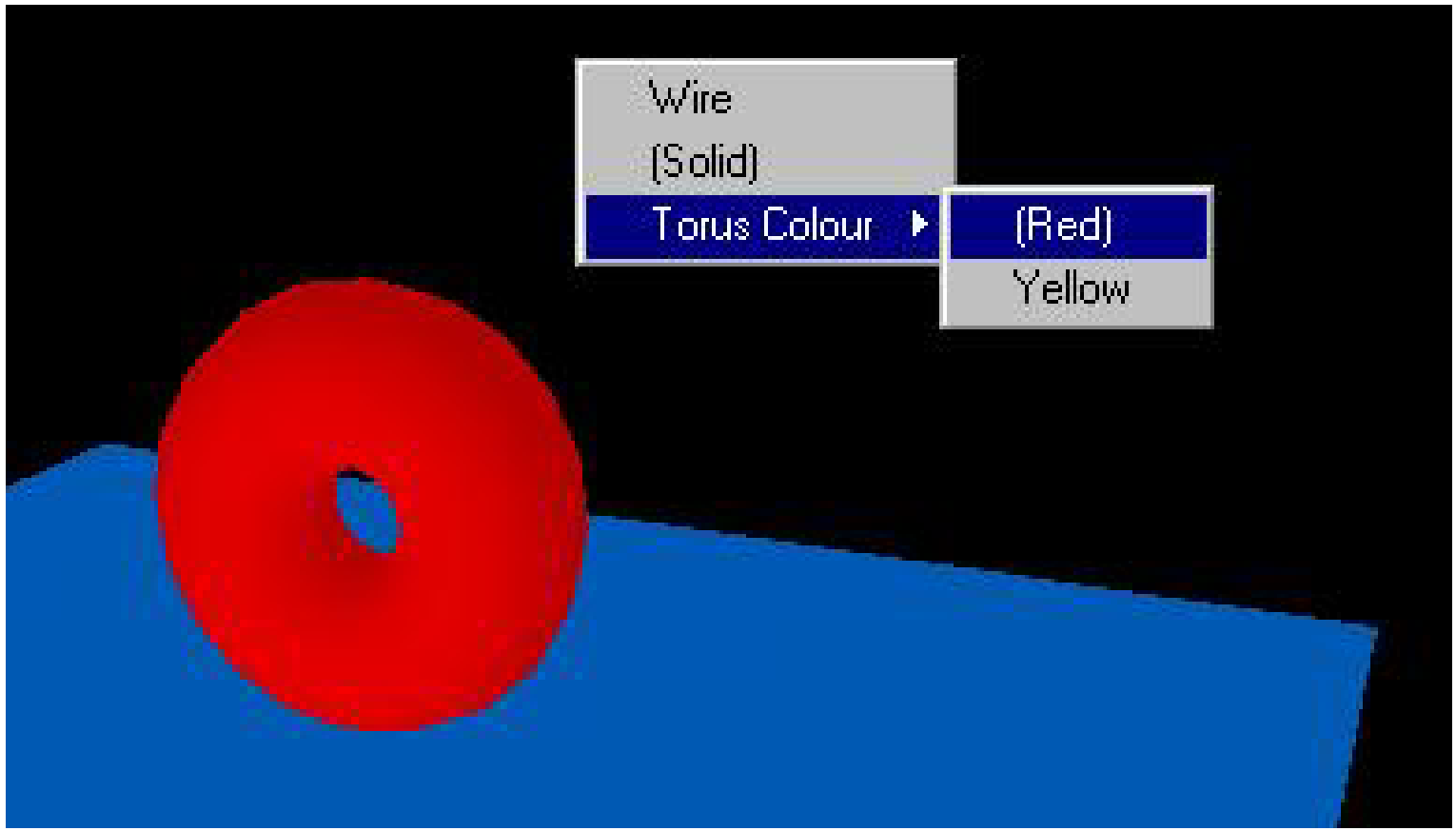
Creating GLUT submenus

A submenu can be associated with a menu just like a regular menu item. First, the submenu is created just like a regular menu and its identifier stored. Then, the submenu can be attached to another menu:

```
GLint submenuID = glutCreateMenu (submenuFcn);
glutAddMenuEntry (“(Red)”, 1);
glutAddMenuEntry (“Yellow”, 2);
glutCreateMenu (menuFcn);
glutAddMenuEntry (“Wire”, 1);
glutAddMenuEntry (“(Solid)”, 2);
glutAddSubMenu (“Torus Colour”, submenuID);
```

4.6 User Interfaces

Example



4.6 User Interfaces

GLUI user interface library (by Paul Rademacher)

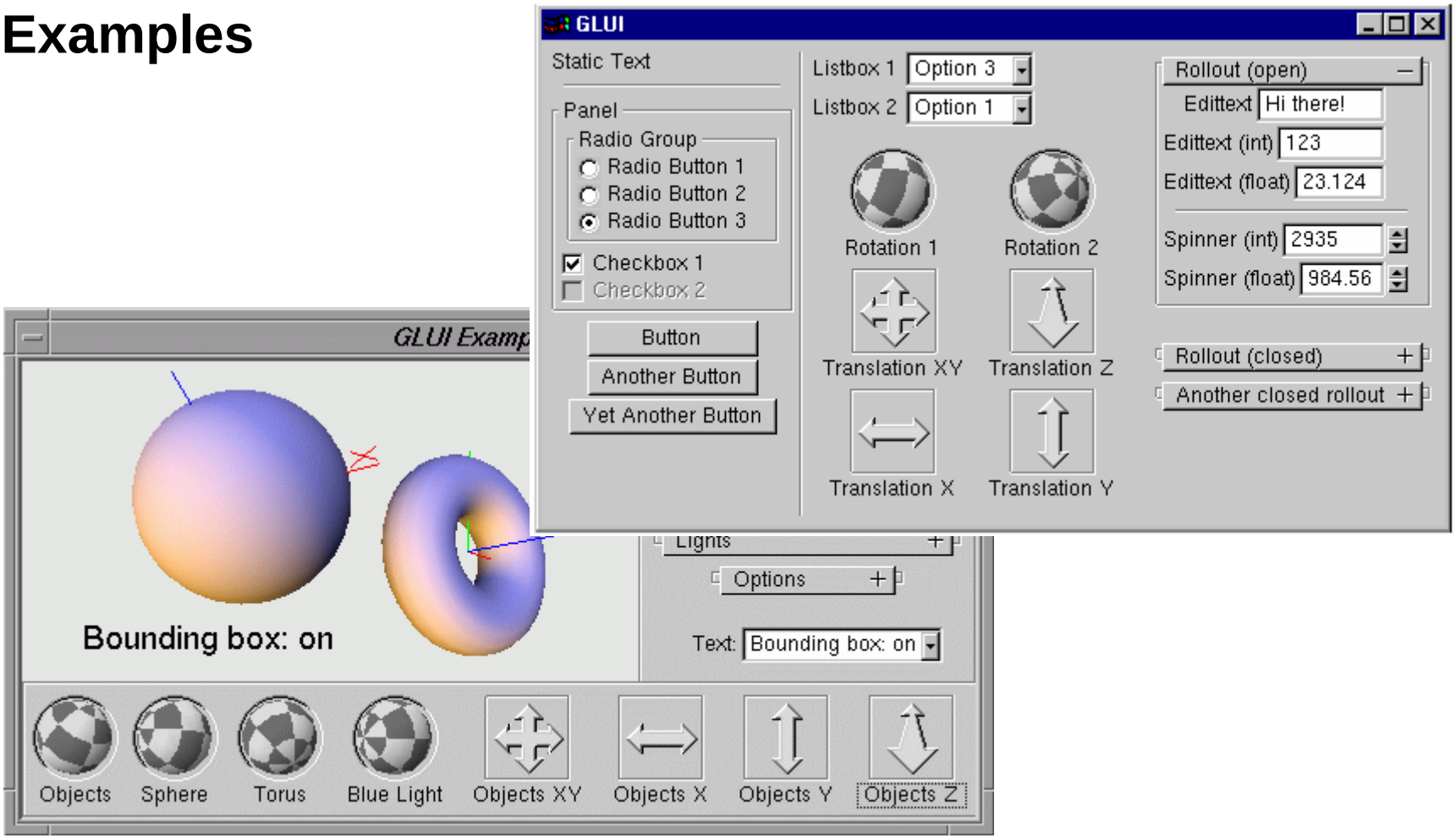
GLUI is a GLUT-based C++ user interface library which provides controls such as buttons, checkboxes, radio buttons, and spinners to OpenGL applications. It is window-system independent, relying on GLUT to handle all system-dependent issues, such as window and mouse management. GLUI can be used to easily add a user interface to an existing OpenGL application.

You can find documentation and the source code here:

http://www.cs.unc.edu/~rademach/glui/

4.6 User Interfaces

Examples



4.6 User Interfaces

Usage for standalone GLUI windows

Integrating GLUI with a new or existing GLUT application is very straightforward. The steps that need to be taken in order to create a user interface are:

1. Add the GLUI library to the link line (e.g., glui32.lib for Windows). The proper order in which to add libraries is: GLUI, GLUT, GLU, OpenGL.
2. Include the file `glui.h` in all sources that will use the GLUI library.
3. Create your regular GLUT windows and popup menus as usual. Make sure to store the window id of your main graphics window, so GLUI windows can later send it redisplay events:

```
int window_id = glutCreateWindow("Main window");
```

4.6 User Interfaces

Usage for standalone GLUI windows (continued)

4. Register your GLUT callbacks as usual (except the Idle callback, discussed below).
5. Register your GLUT idle callback (if any) with GLUI_Master (a global object which is already declared), to enable GLUI windows to take advantage of idle events without interfering with your application's idle events. If you do not have an idle callback, pass in NULL.

```
GLUI_Master.set_glutIdleFunc(myGlutIdle);
```

or

```
GLUI_Master.set_glutIdleFunc(NULL);
```

4.6 User Interfaces

Usage for standalone GLUI windows (continued)

6. In your idle callback, explicitly set the current GLUT window before rendering or posting a redisplay event. Otherwise the redisplay may accidentally be sent to a GLUI window.

```
void myGlutIdle (void) {
   glutSetWindow (main_window);
   glutPostRedisplay ();
}
```

7. Create a new GLUI window using

```
GLUI *glui
  = GLUI_Master.create_glui("name", flags, x, y);
```

   Note that `flags`, `x`, and `y` are optional arguments. If they are not specified, default values will be used. GLUI provides default values for arguments whenever possible.

4.6 User Interfaces

Usage for standalone GLUI windows (continued)

8. Add controls to the GLUI window. For example, we can add a checkbox and a quit button with:

```
glui->add_checkbox("Lighting", &lighting);
glui->add_button("Quit", QUIT_ID, callback_func);
```

9. Let each GLUI window you've created know where its main graphics window is:

```
glui->set_main_gfx_window (window_id);
```

10. Invoke the standard GLUT main event loop, just as in any GLUT application:

```
glutMainLoop ();
```

4.6 User Interfaces

Usage for GLUI subwindows

Adding GLUI subwindows is slightly more complicated than adding standalone GLUI windows. Since the graphics application and GLUI share window space, a little extra work is required to ensure that they cooperate appropriately.

This is particularly the case for existing OpenGL applicartions.

See the GLUI documentation for details:

http://www.cs.unc.edu/~rademach/glui/src/release/glui_manual_v2_beta.pdf