

## Objectives

- Learn to build interactive programs using GLUT callbacks
  - Mouse
  - Keyboard
  - Reshape
- Introduce menus in GLUT
- Picking
  - Select objects from the display
  - Three methods



Slides by Edward Angel © 2002

## Animating a Display

- When we redraw the display through the display callback, we usually start by clearing the window
  - `glClearColor()`
- then draw the altered display
- Problem: the drawing of information in the frame buffer is decoupled from the display of its contents
  - Graphics systems use dual ported memory
- Hence we can see partially drawn display
  - See the program `single_double.c` for an example with a rotating cube



Slides by Edward Angel © 2002

## Double Buffering

- Instead of one color buffer, we use two
  - **Front Buffer**: one that is displayed but not written to
  - **Back Buffer**: one that is written to but not altered
- Program then requests a double buffer in `main.c`
  - `glutInitDisplayMode(GL_RGB | GL_DOUBLE)`
  - At the end of the display callback buffers are swapped

```
void mydisplay() {
    glClearColor();
    /* draw graphics here */
    glutSwapBuffers();
}
```



Slides by Edward Angel © 2002

## Using the idle callback

- The idle callback is executed whenever there are no events in the event queue
  - `glutIdleFunc(myidle)`
  - Useful for animations

```
void myidle() {
    /* change something */
    t += dt;
    glutPostRedisplay();
}
void mydisplay() {
    glClearColor();
    /* draw something that depends on t */
    glutSwapBuffers();
}
```



Slides by Edward Angel © 2002

## Using globals

- The form of all GLUT callbacks is fixed
  - `void mydisplay()`
  - `void mymouse(GLint button, GLint state, GLint x, GLint y)`
- Must use globals to pass information to callbacks

```
float t; /*global */

void mydisplay()
{
    /* draw something that depends on t */
}
```



Slides by Edward Angel © 2002

## The mouse callback

- `glutMouseFunc(mymouse)`
- `void mymouse(GLint button, GLint state, GLint x, GLint y)`
- Returns
  - which button (`GLUT_LEFT_BUTTON`, `GLUT_MIDDLE_BUTTON`, `GLUT_RIGHT_BUTTON`) caused event
  - state of that button (`GL_UP`, `GL_DOWN`)
  - Position in window



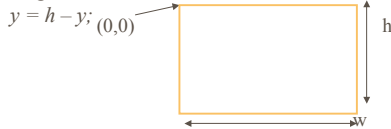
Slides by Edward Angel © 2002

## Positioning

The position in the screen window is usually measured in pixels with the origin at the top-left corner

Consequence of refresh done from top to bottom  
OpenGL uses a world coordinate system with origin at the bottom left

Must invert  $y$  coordinate returned by callback by height of window



Slides by Edward Angel © 2002

## Obtaining the window size

- To invert the  $y$  position we need the window height
    - Height can change during program execution
    - Track with a global variable
    - New height returned to reshape callback that we will look at in detail soon
    - Can also use enquiry functions
      - `glGetIntv`
      - `glGetFloatv`
- to obtain any value that is part of the state



Slides by Edward Angel © 2002

## Terminating a program

- In our original programs, there was no way to terminate them through OpenGL
- We can use the simple mouse callback

```
void mouse(int btn, int state, int x, int y)
{
    if(btn==GLUT_RIGHT_BUTTON && state==GLUT_DOWN)
        exit(0);
}
```



Slides by Edward Angel © 2002

## Using the mouse position

- In the next example, we draw a small square at the location of the mouse each time the left mouse button is clicked
- This example does not use the display callback but one is required by GLUT; We can use the empty display callback function `mydisplay()`



Slides by Edward Angel © 2002

## Drawing squares at cursor location

```
void mymouse(int btn, int state, int x, int y)
{
    if(btn==GLUT_RIGHT_BUTTON && state==GLUT_DOWN)
        exit(0);
    if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
        drawSquare(x, y);
}
void drawSquare(int x, int y)
{
    y=w-y; /* invert y position */
    glColor3ub( (char) rand()%256, (char) rand()%256,
                (char) rand()%256); /* a random color */
    glBegin(GL_POLYGON);
        glVertex2f(x+size, y+size);
        glVertex2f(x-size, y+size);
        glVertex2f(x-size, y-size);
        glVertex2f(x+size, y-size);
    glEnd();
}
```



Slides by Edward Angel © 2002

## Using the motion callback

- We can draw squares (or anything else) continuously as long as a mouse button is depressed by using the motion callback
  - `glutMotionFunc(drawSquare)`
- We can draw squares without depressing a button using the passive motion callback
  - `glutPassiveMotionFunc(drawSquare)`



Slides by Edward Angel © 2002

## Using the keyboard

- `glutKeyboardFunc(mykey)`
  - `Void mykey(unsigned char key, int x, int y)`
    - Returns ASCII code of key depressed and mouse location
    - Note GLUT does not recognize key release as an event
- ```
void mykey() {  
    if(key == 'Q' | key == 'q')  
        exit(0);  
}
```



Slides by Edward Angel © 2002

## Special and Modifier Keys

- GLUT defines the special keys in `glut.h`
  - Function key 1: `GLUT_KEY_F1`
  - Up arrow key: `GLUT_KEY_UP`
    - `if(key == 'GLUT_KEY_F1' .....`
- Can also check of one of the modifiers
  - `GLUT_ACTIVE_SHIFT`
  - `GLUT_ACTIVE_CTRL`
  - `GLUT_ACTIVE_ALT`is depressed by `glutGetModifiers()`
- Allows emulation of three-button mouse with one- or two-button mice



Slides by Edward Angel © 2002

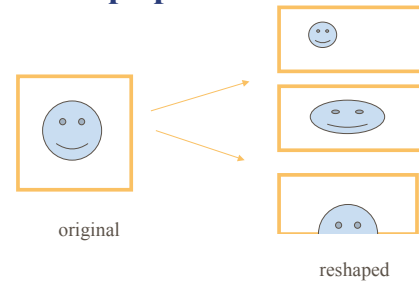
## Reshaping the window

- We can reshape and resize the OpenGL display window by pulling the corner of the window
- What happens to the display?
  - Must redraw from application
  - Two possibilities
    - Display part of world
    - Display whole world but force to fit in new window
      - Can alter aspect ratio



Slides by Edward Angel © 2002

## Reshape possibilities



Slides by Edward Angel © 2002

## The Reshape callback

- `glutReshapeFunc(myreshape)`
- `void myreshape(int w, int h)`
  - Returns width and height of new window (in pixels)
  - A redisplay is posted automatically at end of execution of the callback
  - GLUT has a default reshape callback but you probably want to define your own
- The reshape callback is good place to put camera functions because it is invoked when the window is first opened



Slides by Edward Angel © 2002

## Example Reshape

- This reshape preserves shapes by making the viewport and world window have the same aspect ratio

```
void myReshape(int w, int h)  
{  
    glViewport(0, 0, w, h);  
    glMatrixMode(GL_PROJECTION); /* switch matrix mode */  
    glLoadIdentity();  
    if (w <= h)  
        gluOrtho2D(-2.0, 2.0, -2.0 * (GLfloat) h / (GLfloat) w,  
                  2.0 * (GLfloat) h / (GLfloat) w);  
    else  
        gluOrtho2D(-2.0 * (GLfloat) w / (GLfloat) h, 2.0 *  
                  (GLfloat) w / (GLfloat) h, -2.0, 2.0);  
    glMatrixMode(GL_MODELVIEW); /* return to modelview mode */  
}
```



Slides by Edward Angel © 2002

## Toolkits and Widgets

- Most window systems provide a toolkit or library of functions for building user interfaces that use special types of windows called *widgets*
- Widget sets include tools such as
  - Menus
  - Slidebars
  - Dials
  - Input boxes
- But toolkits tend to be platform dependent
- GLUT provides a few widgets including menus



Slides by Edward Angel © 2002

## Menus

- GLUT supports pop-up menus
  - A menu can have submenus
- Three steps
  - Define entries for the menu
  - Define action for each menu item
    - Action carried out if entry selected
  - Attach menu to a mouse button

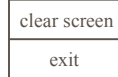


Slides by Edward Angel © 2002

## Defining a simple menu

### ■ In `main.c`

```
menu_id = glutCreateMenu(mymenu);
glutAddMenuEntry("clear Screen", 1);
glutAddMenuEntry("exit", 2);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```



entries that appear when right button depressed

identifiers



Slides by Edward Angel © 2002

## Menu actions

### ■ Menu callback

```
void mymenu(int id)
{
    if(id == 1) glClear();
    if(id == 2) exit(0);
}
```

- Note each menu has an id that is returned when it is created

### ■ Add submenus by

```
glutAddSubMenu(char *submenu_name, submenu id)
```

entry in parent menu



Slides by Edward Angel © 2002

## Other functions in GLUT

- Dynamic Windows
  - Create and destroy during execution
- Subwindows
- Multiple Windows
- Changing callbacks during execution
- Timers
- Portable fonts
  - `glutBitmapCharacter`
  - `glutStrokeCharacter`



Slides by Edward Angel © 2002

## Picking

- Identify a user-defined object on the display
- In principle, it should be simple because the mouse gives the position and we should be able to determine to which object(s) a position corresponds
- Practical difficulties
  - Pipeline architecture is feed forward, hard to go from screen back to world
  - Complicated by screen being 2D, world is 3D
  - How close do we have to come to object to say we selected it?



Slides by Edward Angel © 2002

## Three Approaches

- Hit list
  - Most general approach but most difficult to implement
- Use back or some other buffer to store object ids as the objects are rendered
- Rectangular maps
  - Easy to implement for many applications
  - See paint program in text



Slides by Edward Angel © 2002

## Rendering Modes

- OpenGL can render in one of three modes selected by `glRenderMode(mode)`
  - `GL_RENDER`: normal rendering to the frame buffer (default)
  - `GL_FEEDBACK`: provides list of primitives rendered but no output to the frame buffer
  - `GL_SELECTION`: Each primitive in the view volume generates a *hit record* that is placed in a *name stack* which can be examined later



Slides by Edward Angel © 2002

## Selection Mode Functions

- `glSelectBuffer()`: specifies name buffer
  - `glInitNames()`: initializes name buffer
  - `glPushName(id)`: push id on name buffer
  - `glPopName()`: pop top of name buffer
  - `glLoadName(id)`: replace top name on buffer
- `id` is set by application program to identify objects



Slides by Edward Angel © 2002

## Using Selection Mode

- Initialize name buffer
- Enter selection mode (using mouse)
- Render scene with user-defined identifiers
- Reenter normal render mode
  - This operation returns number of hits
- Examine contents of name buffer (hit records)
  - Hit records include id and depth information



Slides by Edward Angel © 2002

## Selection Mode and Picking

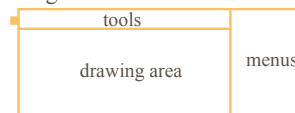
- As we just described it, selection mode won't work for picking because every primitive in the view volume will generate a hit
- Change the viewing parameters so that only those primitives near the cursor are in the altered view volume
  - Use `gluPickMatrix` (see text for details)



Slides by Edward Angel © 2002

## Using Regions of the Screen

- Many applications use a simple rectangular arrangement of the screen



- Easier to look at mouse position and determine which area of screen it is in that using selection mode picking



Slides by Edward Angel © 2002

## Using another buffer and colors for picking

- For a small number of objects, we can assign a unique color (often in color index mode) to each object
- We then render the scene to a color buffer other than the front buffer so the results of the rendering are not visible
- We then get the mouse position and use `glReadPixels ()` to read the color in the buffer we just wrote at the position of the mouse
- The returned color gives the id of the object

