GLUT Event Loop

- **`glutMainLoop();`**

puts the program in an infinite event loop

- In each pass through the event loop, GLUT
  - looks at the events in the queue
  - for each event in the queue, GLUT executes the appropriate callback function if one is defined
  - if no callback is defined for the event, the event is ignored

KENT STATE UNIVERSITY

Event Types

- Window: resize, expose, iconify
- Mouse: click one or more buttons
- Motion: move mouse
- Keyboard: press or release a key
- Idle: nonevent
  - Define what should be done if no other event is in queue

Callbacks

- Programming interface for event-driven input
- Define a *callback function* for each type of event the graphics system recognizes
- This user-supplied function is executed when the event occurs
- GLUT example:
  `glutMouseFunc(mymouse)`

mouse callback function

KENT STATE UNIVERSITY

GLUT callbacks

GLUT recognizes a subset of the events recognized by any particular window system (Windows, X, Macintosh)

- **`glutDisplayFunc`**
- **`glutMouseFunc`**
- **`glutReshapeFunc`**
- **`glutKeyboardFunc`**
- **`glutIdleFunc`**
- **`glutMotionFunc, glutPassiveMotionFunc`**

The Display Callback

- The display callback is executed whenever GLUT determines that the window should be refreshed, for example
  - When the window is first opened
  - When the window is reshaped
  - When a window is exposed
  - When the user program decides it wants to change the display
- In **`main()`**
  - **`glutDisplayFunc(mydisplay)`** identifies the function to be executed
  - Every GLUT program must have a display callback

Posting Redisplays

- Many events may invoke the display callback function
  - Can lead to multiple executions of the display callback on a single pass through the event loop
- We can avoid this problem by instead using

```
glutPostRedisplay();
```

  which sets a flag.
- GLUT checks to see if the flag is set at the end of the event loop
- If set then the display callback function is executed

KENT STATE UNIVERSITY

Animating a Display

- When one redraws the display through the display callback, we usually start by clearing the window
  - **`glClear()`**

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

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 displayed
- Program then requests a double buffer in main.c
  - **`glutInitDisplayMode(GL_RGB | GL_DOUBLE)`**
  - At the end of the display callback buffers are swapped

Double Buffering

```
void mydisplay()
{

glClear(GL_COLOR_BUFFER_BIT|….)
.
/* draw graphics here */
.
    glutSwapBuffers()
}
```

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() {
      glClear();
/* draw something that depends on t */
      glutSwapBuffers();
}
```

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
}
```

KENT STATE UNIVERSITY

The mouse Callback

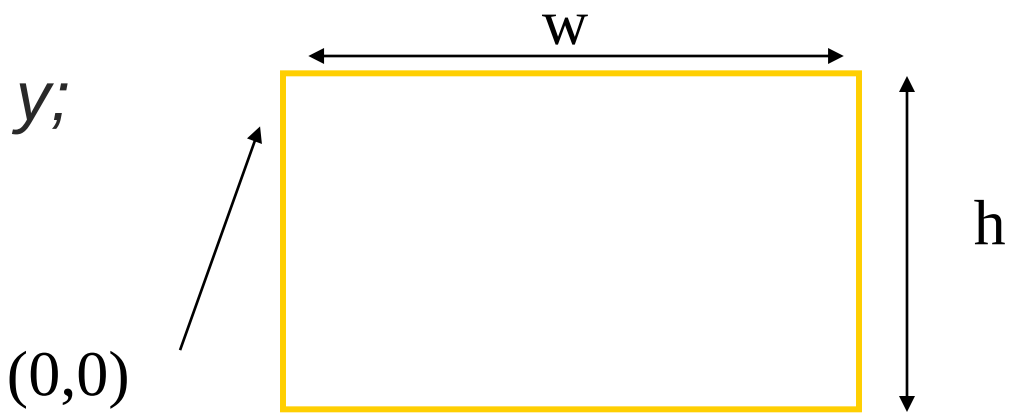
```
glutMouseFunc(mymouse)
void mymouse(GLint button, GLint
  state, GLint x, GLint y)
```

- Returns
  - which button (**GLUT_LEFT_BUTTON**, **GLUT_MIDDLE_BUTTON**, **GLUT_RIGHT_BUTTON**) caused event
  - state of that button (**GLUT_UP**, **GLUT_DOWN**)
  - Position in window

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

$y = h - y;$



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 query functions
    - **`glGetIntv`**
    - **`glGetFloatv`**

to obtain any value that is part of the state

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);
}
```

Using the Keyboard

```
glutKeyboardFunc(mykey)
void mykey(unsigned char key,
      int x, int y)
```

- Returns ASCII code of key depressed and mouse location

```
void mykey()
{
      if(key == ‘Q’ | key == ‘q’)
            exit(0);
}
```

KENT STATE UNIVERSITY

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 if 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

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 a good place to put viewing functions because it is invoked when the window is first opened

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 */
}
```

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

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

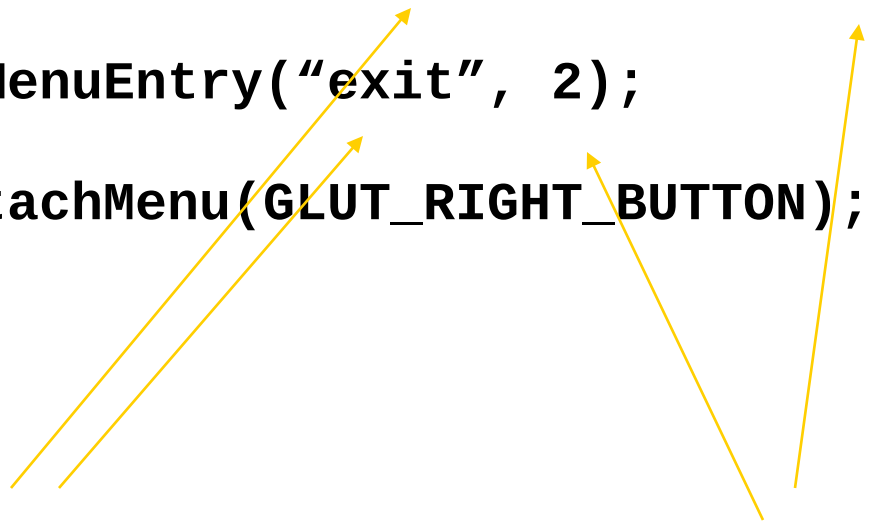
Defining a Simple Menu

- In **`main()`**

```
menu_id = glutCreateMenu(mymenu);
glutAddmenuEntry("clear Screen", 1);

gluAddMenuEntry("exit", 2);

glutAttachMenu(GLUT_RIGHT_BUTTON);
```



clear screen
exit

entries that appear when right button depressed

identifiers

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

Other functions in GLUT

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

Example

- \\samba.cs.kent.edu\ogre\classEnv\Project\BasicDemo
- http://www.cs.kent.edu/~ruttan/GameEngines/lectures/upload/windows/BasicDemo.zip