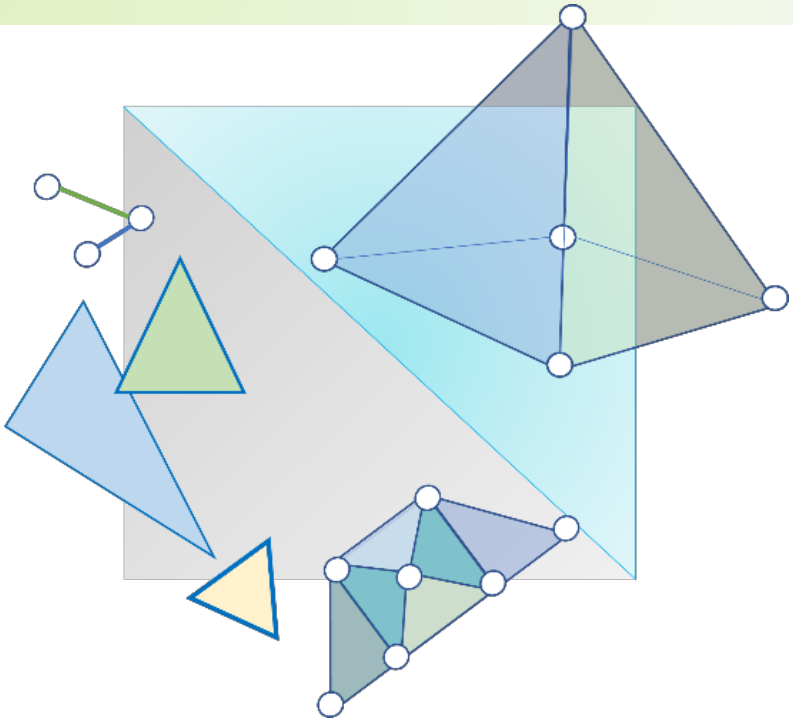
# CITS3003 Graphics & Animation

Lecture 11:
Interactive Programs
with Callbacks and Menus



# Content

- How to build interactive programs using GLUT callbacks
  - Mouse
  - Keyboard
  - Reshape
  - Idle
- Introduction to menus in GLUT

# Using a Pointing Device (Mouse)

- **Mouse Event:** When one of the mouse buttons is depressed or released or the scroll wheel is moved. glutMouseFunc()

- **Motion Event:** When the mouse is moved within the window with one of the buttons depressed. glutMotionFunc()

- **Passive Motion Event:** When the mouse is moved within the window without a button being held down. glutPassiveMotionFunc()

# The Mouse Callback

```
glutMouseFunc(mymouse);

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

- The parameters passed to the function are:

  **button** - which mouse button caused the event
  - **GLUT_LEFT_BUTTON**
  - **GLUT_MIDDLE_BUTTON**
  - **GLUT_RIGHT_BUTTON**
  - Wheel is still a button
    - button == 3 (scroll forward)
    - button == 4 (scroll backwards)

  **state** - state of that button (**GLUT_UP**, **GLUT_DOWN**)

  **x, y** – the mouse click position (in pixels) in the window

# Terminating a Program

- We can use a simple *mouse callback to terminate a program:*

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

- Note that we ignore the x and y parameters in the example above.

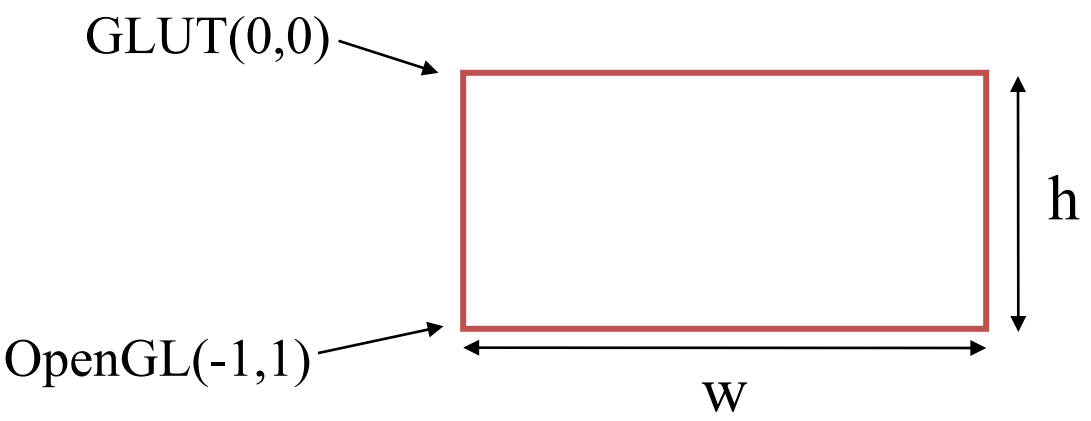
# Mouse Positioning (x, y)

- The mouse position on the screen window is usually measured in pixels with the origin at the top-left corner
  - When the window is refreshed, it is done from top to bottom

- OpenGL uses a world coordinate system with the origin at the center of the window. Thus,
  - you must invert the *y* coordinate passed to your callback function by the height of the window
  - i.e., $y = h - y;$

    $y=(y/(h/2) - 1.0;$

  - x=x/(w/2) – 1.0;



# How to Obtain the Window Size

- To invert the *y* position, we need to know the window height
  - Note that the window height can change during program execution
  - We can use a global variable to keep track of the window height value

```
glutGet(GLUT_WINDOW_WITDH)
glutGet(GLUT_WINDOW_HEIGHT)
```

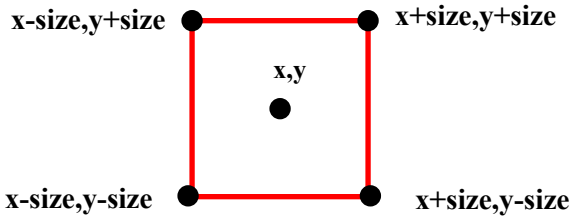
# Using the Mouse Position

- In the next example, we show how to draw a small square at the location of the mouse each time the left mouse button is clicked.

# Example 1: Drawing squares at clicked location of cursor

```
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 = h-y; /* invert y position */

    points[i] = vec2(x+size, y+size);
    points[i+1] = vec2(x-size, y+size);
    points[i+2] = vec2(x-size, y-size);
    points[i+3] = vec2(x+size, y-size);
    i += 4;
}
```




Note that in the drawSquare function, variable h is a global variable storing the height (in pixels) of the window.

**h**, **size**, **points** and **i** are global variables.

# Example 2: Drawing a triangle by specifying 3 vertices

```
int w, h;   //the width and height of window
int count = 0;

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)
    {
       y = h-y; /* invert y position */

        points[count].x = (float) x/(w/2) - 1.0;
        points[count].y = (float) y/(h/2) - 1.0;
        count++;
    }
    if (count == 3)
    {
        glutPostRedisplay();
        count = 0;
    }
}
```

# Using the Motion Callback

- We can draw squares (or anything else) using the motion callback:

```
glutMotionFunc(mymotion);
```

- We can also draw squares without depressing a button using the passive motion callback

```
glutPassiveMotionFunc(mypassivemotion);
```

# Using the Keyboard Callback

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

Parameters passed to the mykey callback function are:

- **key** – the ASCII code of the key depressed
- **x, y** - and mouse location at the time the key was pressed

Example:

```
void mykey(unsigned char key, int x, int y)
{
    if (key == 'Q' | key == 'q')
        exit(0);
}
```

# Special and Modifier Keys

- GLUT defines the special keys in **glut.h**
  - Function key 1: **GLUT_KEY_F1, GLUT_KEY_F12 …**
  - Up arrow key: **GLUT_KEY_UP, GLUT_KEY_RIGHT …**
  - **e.g., if (key == GLUT_KEY_F1 ……**

- Can also check whether one of the modifiers
  - **GLUT_ACTIVE_SHIFT**
  - **GLUT_ACTIVE_CTRL**
  - **GLUT_ACTIVE_ALT**

  is depressed by

  **glutGetModifiers()**

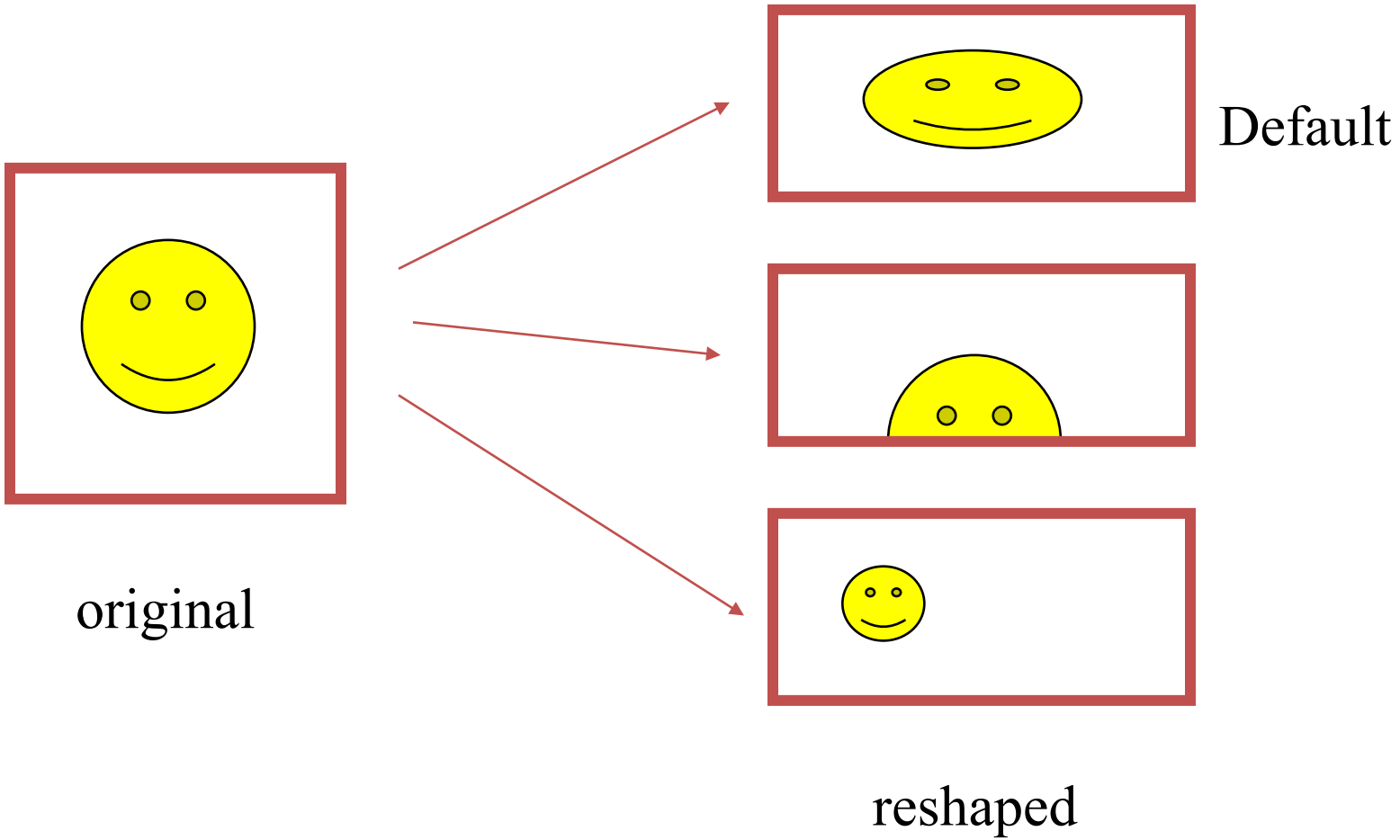

https://en.wikipedia.org/wiki/Function_key#/media/File:Qwerty.svg

# Reshaping the Window

- We can reshape and resize the display window by pulling the corner of the window
- What happens to the display?
  - The window in the application must be redrawn
  - There are three possibilities:
    1. We can display the whole world but force it to fit in the new window (this can alter the aspect ratio).
    2. We can display part of the world.
    3. We can alter the scale the of whole world to fit in the window and keep the aspect ratio.

# Reshape Possibilities

- Three reshape possibilities



# The Reshape Callback

```
glutReshapeFunc(myreshape);
void myreshape(int w, int h)
```

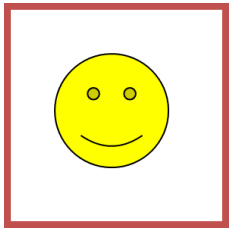
- Parameters passed to the function:

  w, h – the width and height of new window (in pixels)

- What happens when the window is resized:
  - 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

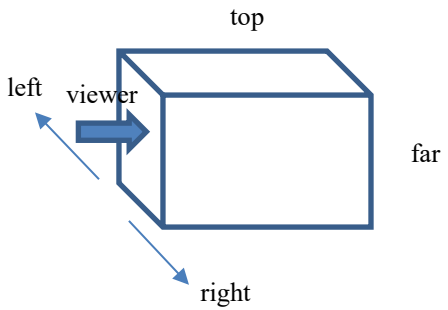
# The Reshape Callback (cont.)

Suppose that the original display window is 500 (width) x 500 (height) pixels, and the clipping volume is: left=-0.2, right=0.2, bottom=-0.2, top=0.2, near=2.0, far=20.0.



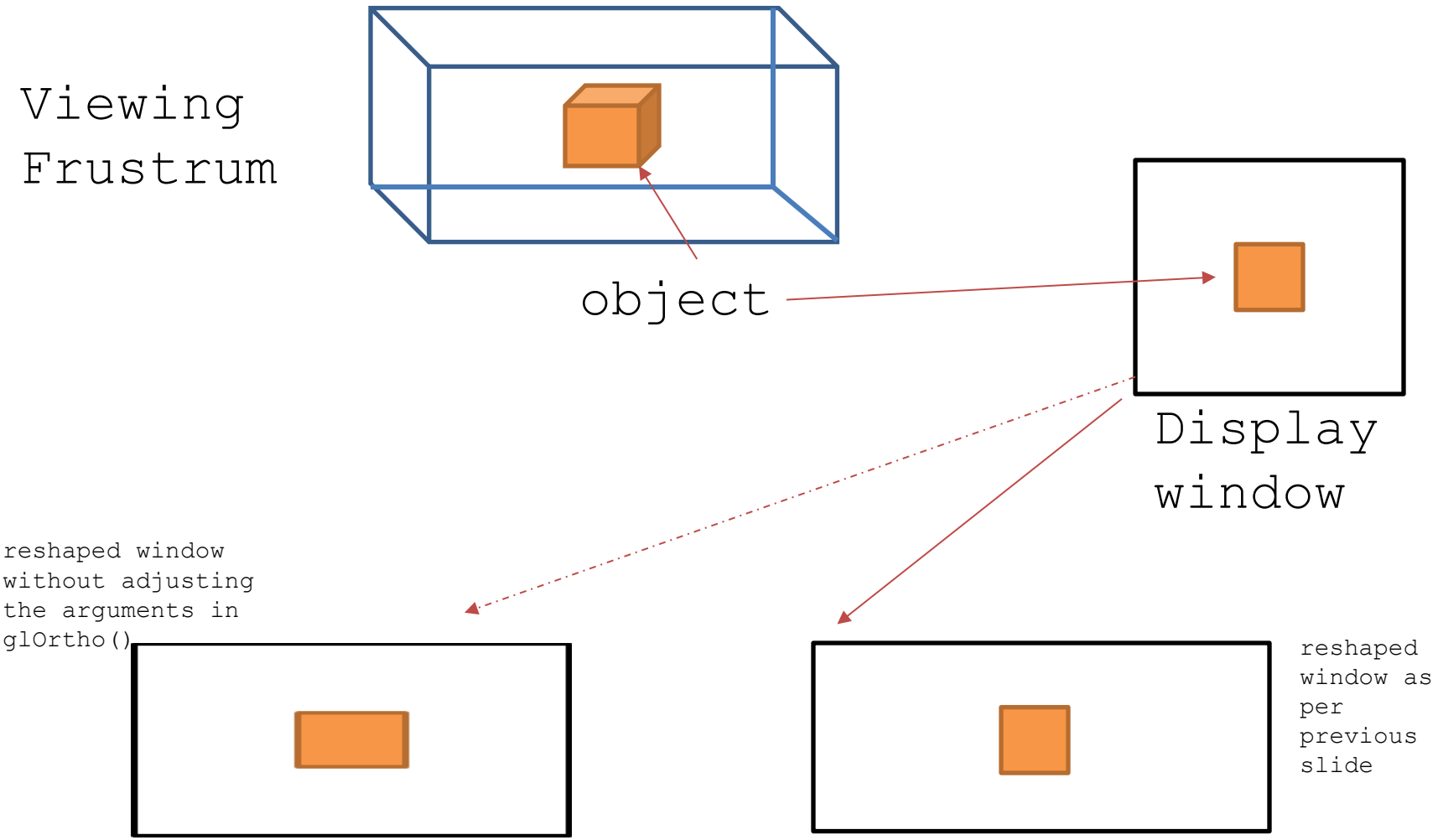
```
void reshape(int w, int h)
{
  glViewport(0, 0, w, h);

  //glOrtho(left,right,bottom,top,near,far)

  glOrtho(-0.2*(float)w/(float)h,
           0.2*(float)w/(float)h,
          -0.2, 0.2, 2.0, 20.0);
}
```

Note that *near* and *far* clipping planes should be positive; otherwise the clipping volume would be taken as behind the camera.




No need to call **glutPostRedisplay( )** here (see previous slide)

# The Reshape Callback (cont.)

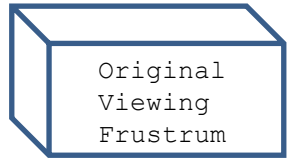


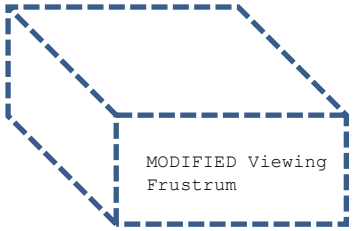
# The Reshape callback (cont.)

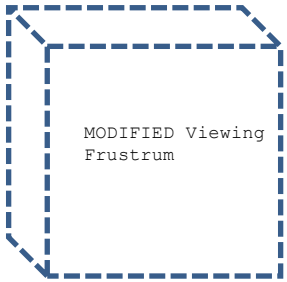
Same setting as described on the previous slide. What does the following reshape callback function do?

```
void reshape(int w, int h) {
  glViewport(0, 0, w, h);

  /*glOrtho(left,right,bottom,top,near,far)*/
  if (w > h)  /* aspect >1 */
    glOrtho(-0.2*(float)w/(float)h,
            0.2*(float)w/(float)h,
            -0.2, 0.2, 2.0, 20.0);

  else  /* aspect<=1 */
    glOrtho(-0.2, 0.2, -0.2*(float)h/(float)w,
            0.2*(float)h/(float)w, 2.0, 20.0);
}
```

aspect = width /height



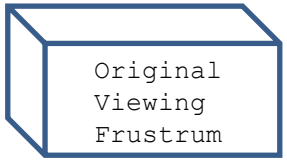

https://www.youtube.com/watch?v=V87lLvKscIY

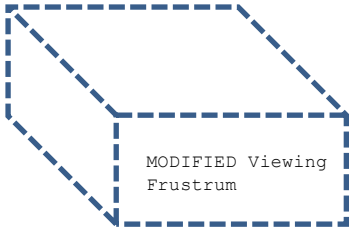
# The Reshape callback (cont.)

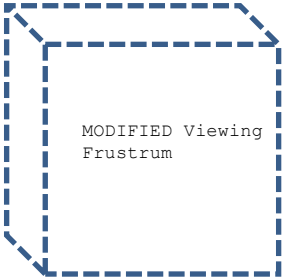
Same setting as described on the previous slide. What does the following reshape callback function do?

```
void reshape(int w, int h) {
  glViewport(0, 0, w, h);

  /*glOrtho(left,right,bottom,top,near,far)*/
  if (w > h)  /* aspect >1 */
    glOrtho(-0.2*(float)w/(float)h,
            0.2*(float)w/(float)h,
            -0.2, 0.2, 2.0, 20.0);

  else  /* aspect<=1 */
    glOrtho(-0.2, 0.2, -0.2*(float)h/(float)w,
            0.2*(float)h/(float)w, 2.0, 20.0);
}
```

aspect = width /height




https://www.youtube.com/watch?v=V87lLvKscIY

# The Idle Callback

- Invoked when there are no other events. Its default is the null function pointer.
- Uses:
  - continue to generate graphical primitives through a display function while nothing else is happening
  - to produce an animated display.
- In main, we specify an idle callback,
  - glutIdleFunc(idle);

# Example : Idle Callback

```
void display()
{
    // update/recalculate x, y, z locations of vertices
    // based on previous x, y, z locations and/or time
}
void idle()
{
    glutPostRedisplay();
    //sometimes you just have to call the redisplay
    //because what needs to change is already in there
    }
```

# 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 for setting up a menu:
  1) Define entries for the menu
  2) Define action for each menu item
     - Action carried out if an entry is selected
  3) Attach menu to a mouse button

# Defining a Simple Menu

- In the **main** or **init** function:

a unique ID returned by glut

name of the callback function

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

glutAddmenuEntry('exit', 2);

glutAttachMenu(GLUT_RIGHT_BUTTON);
```

| clear screen |
| --- |
| exit |

entries that will appear when right button is pressed

identifiers

# Menu Actions

Example of a simple menu callback function:

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

Note when each menu is created, a unique id is returned by glut

- To add  a submenu, use glutAddSubMenu:

```
glutAddSubMenu(char *submenuName, int submenuId)
```

- To add a menu entry, use glutAddMenuEntry:

```
glutAddMenuEntry(char *entryname, int entryID)
```

- To attach the current menu, use glutAttachMenu.

# Menu – an example

```
// submenu for two light sources
int lightMenuId = glutCreateMenu(lightMenu);
glutAddMenuEntry("Move Light 1", 11);
glutAddMenuEntry("Change RGB of Light 1", 12);
glutAddMenuEntry("Move Light 2", 21);
glutAddMenuEntry("Change RGB of Light 2", 22);

// submenu for camera
int cameraMenuId = glutCreateMenu(cameraMenu);

// add these submenus to the main menu
glutCreateMenu(mainMenu);
glutAddSubMenu("Light sources", lightMenuId);
glutAddSubMenu("Camera", cameraMenuId);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```

Put these lines of code in an *init* function

Callback functions

Always create the submenus before the main menu

# Menu – an example

```
// callback function for the light menu
void lightMenu(int id) {
    switch (id) {
        case 11: // action for moving light 1
        case 12: // action for changing RGB of light 1
        ...
    }
}

// callback function for the camera menu
void cameraMenu(int id) {
    ...
}
```

# Further Reading

"Interactive Computer Graphics – A Top-Down Approach with Shader-Based OpenGL" by Edward Angel and Dave Shreiner, 6th Ed, 2012

- Sec. 2.7 *Control Functions*
- Sec. 2.11 *Adding Interaction* up to
  Sec 2.11.4 *The Idle Callback*
- Sec 2.12 *Menus*
- C++ code in the Chapter04 - Chapter09 folders