Event Handling

Parameterizing Figures

- Setting vertices for every drawing
  - Not flexible
  - Hardwired code
- Parameterized drawings
  - Flexible
  - Pass parameter to the routine and draw families of objects by passing parameter values to the routine.

Mouse Interaction

- Knowing activities of the mouse and reacting to it helps in many graphics oriented tasks and also user interfaces. The mouse interaction can be achieved in two ways:
  - Mouse Buttons
  - Mouse Motion

Mouse Button

- Register a function that will be responsible for handling mouse activities using

  glutMouseFunc(mouseHandler)

- Parameters of the mouse handler should be as follows
- mouseHandler(int button, int state, int x, int y)
- The **button** parameter will receive one of the following values
  - GLUT_LEFT_BUTTON
  - GLUT_MIDDLE_BUTTON
  - GLUT_RIGHT_BUTTON

State parameters

- The **state** parameter will receive one of the following values
  - GLUT_DOWN
  - GLUT_UP
- x and y are screen coordinates of the mouse pointer
- The x value is the number of pixels from the left of the window.
- The y value is the number of pixels *down* from the top of the window.

How to handle handler

- In order to see the effects of some activity of the mouse or keyboard, the mouse or keyboard handler *must* call either myDisplay() or glutPostRedisplay().
- Usually the coded inside the mouse handler will check the button and its state and react accordingly e.g.
  - if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)

```
void myMouse(int button, int state, int x, int y)
{
    #define NUM 20
    static GLintPoint List[NUM];
    static int last = -1;                  // last index used so far

   // test for mouse button as well as for a full array
     if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN && last < (NUM -1))
     {
        List[++last].x = x;          // add new point to list
        List[  last].y = screenHeight - y;
        glClear(GL_COLOR_BUFFER_BIT);      // clear the screen
        glBegin(GL_LINE_STRIP);          // redraw the polyline
          for(int i = 0; i <= last; i++)
            glVertex2i(List[i].x, List[i].y);
        glEnd();
        glFlush();
   }
   else if(button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
     last = -1;              // reset the list to empty
}
```

Mouse motion

- Active mouse motion: Movement of mouse on the window when one of its button is pressed
  - ***handleActiveMouseMotion(int x, int y)***
  - ***glutMotionFunc(handleActiveMouseMotion)***
- Passive mouse motion: Movement of mouse when no button is pressed
  - ***handlePassiveMouseMotion(int x, int y)***
  - ***glutPassiveMotionFunc(handlePassiveMouseMotion)***

Keyboard Interaction

- Keys that are associated to some (single byte) ASCII code.
- Make a function to do something on press of some key. It should receive an unsigned character for the key and the mouse location e.g.
- handleNormalKeys(unsigned char key, int x, int y)
- The function has to be registered with the operating system using
- **glutKeyboardFunc(handleNormalKeys)**

Example…

- Usual contents of the keyboard handler would be to check which key was pressed and react to it.
- For example, having 27 as ASCII code for ESC:
  - if (key == 27)
  - …
  - if (key == 'c')
  - …

```
void myKeyboard(unsigned char theKey, int mouseX, int
    mouseY)
{
      GLint x = mouseX;
      GLint y = screenHeight - mouseY; // flip y value
    switch(theKey)
      {case 'd':
           drawDot(x, y);
           break;
// draw dot at mouse position
      case 'E':
           exit(-1);          //terminate the program
       default: break;        // do nothing
      }
}
```

Special Keys

- Keys that are not represented in the ASCII table e.g. function and arrow keys. These have code of two bytes each.
- Similar to normal keys, make a function to do something on press of the required keys. It should receive an integer for the key and the mouse location e.g.

  handleSpecialKeys(int key, int x, int y)

- This function has to be registered with the operating system using

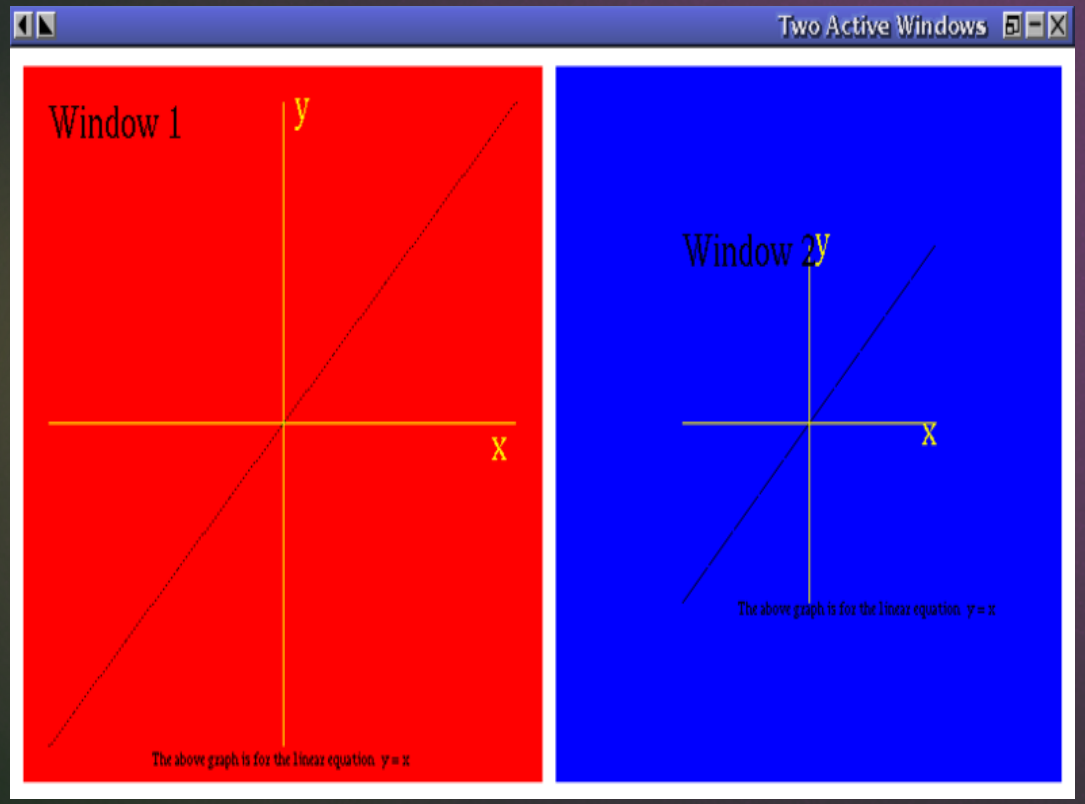
  glutSpecialFunc(handleSpecialKeys)

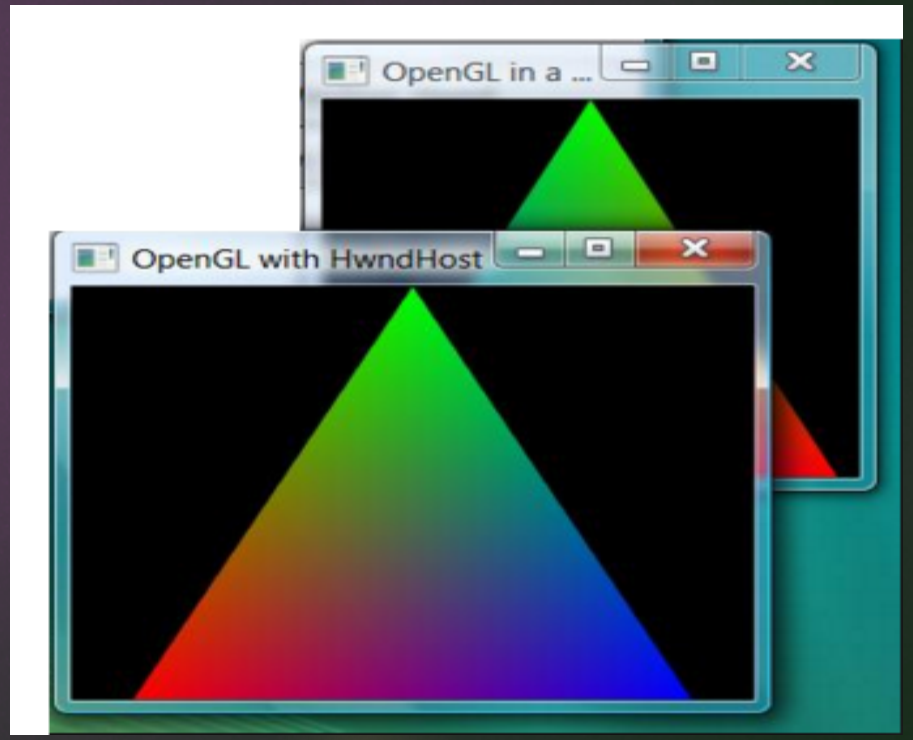
- The key parameter can have one of the following values:
  - GLUT_KEY_F1 to GLUT_KEY_F12 for function keys
  - GLUT_KEY_LEFT, GLUT_KEY_RIGHT, GLUT_KEY_UP, and GLUT_KEY_DOWN
- for arrow keys
  - GLUT_KEY_PAGE_UP, GLUT_KEY_PAGE_DOWN, GLUT_KEY_HOME, GLUT_KEY_END, GLUT_KEY_INSERT
- Additionally status of the special keyboard keys can also be tested using

```
int key = glutGetModifiers( );
```

  - The returned value can be one of
  - GLUT_ACTIVE_SHIFT, GLUT_ACTIVE_CTRL, GLUT_ACTIVE_ALT

Windows and Displays





Window management

- There can exist parallel and sub-windows under the same OpenGL program
- A new window can be created using
  - ***int id = glutCreateWindow("Window Title");***
- A window can be activated using
  - ***glutSetWindow(id)***
- The drawing commands affect the currently active window
- Each window can have different display properties. The display properties

  (using glutInitDisplayMode()) can be set before creating a new window

Behavior of Callback Functions

- Keyboard and mouse events are routed by the event loop to the callbacks registered for the window in which the cursor is located.
- Display events generated for each window separately
  - when the O/S determines that the window must be redisplayed.
- Display events can be user generated using glutPostRedisplay().
  - These events are routed to the display callback for the current window.
- The shared idle() function is executed whenever

Example Using Two Windows


```
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB |GLUT_DEPTH);
glutInitWindowSize(500, 500);

// create the first window
window1 = glutCreateWindow("First Window - Perspective
View");
// register callbacks for first window, which is now current
glutReshapeFunc(window1_reshape);
glutDisplayFunc(window1_display);
glutMouseFunc(window1_mouse);
```

```
//define a window position for second window
glutPositionWindow(520,20);
// register callbacks for second window, which is now current
glutReshapeFunc(window2_reshape);
glutDisplayFunc(window2_display);
glutMouseFunc(window1_mouse); //note we share the mouse function
glutIdleFunc(spinCube); //idle function is not associated with a window
//create the second  window
window2 = glutCreateWindow("Second Window - Top/Down View");
glutMainLoop();
```

Display management

- A display controller function is registered with the operating system using
  - glutDisplayFunc(displayController)
- A function can made that should handle situations when the window gets reshaped. This function is registered using
  - glutReshapeFunc(reshapeHandler)
- A function that should keep doing something during the system waits for any keyboard or mouse action from the user is registered as idle callback using
  - glutIdleFunc(doWhenIdle))

glutPostRedisplay(): Screen Refreshing

- In order to activate the display controller function deliberately glutPostRedisplay() can called any time.
- This helps refreshing the screen for example while showing animations.
- A window ID can be given as parameter to refresh a specific window
- The ultimate effect of this function is to call your Display callback for the current window .

Menu Driven User Interface

Menu Driven User Interface

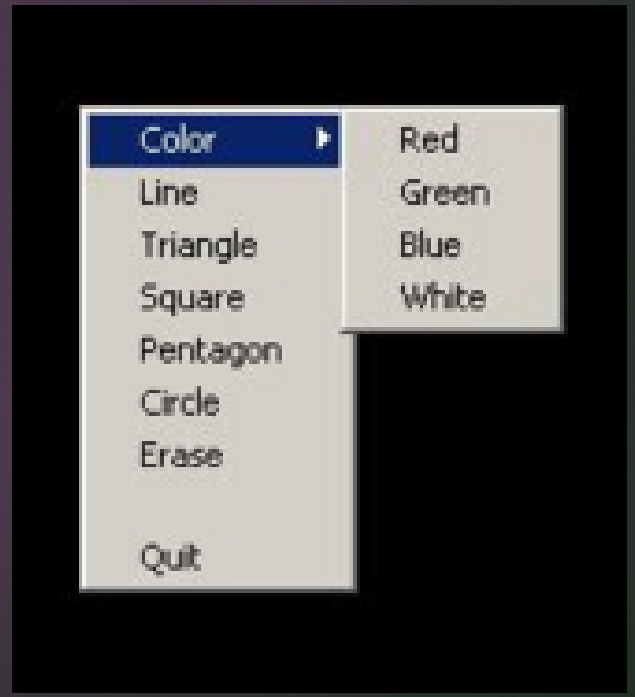
- Creating menus to interact with the user involves the following tasks
  - **Step 1:** Define the actions to be performed on selected of each menu item in a function
  - **Step 2:** Register this function as call back function associated to the menue
  - **Step 3:** Link the menu to a particular mouse button
- There can exist many menus in a program

Example:

- int main_menu =glutCreateMenu(processMenuItems);
  - void glutAddMenuEntry(“WOMEN", 1);
  - void glutAddMenuEntry(“MEN", 2);
  - void glutAddMenuEntry(“KIDS", 3);
  - Void glutAttachMenu(GLUT_RIGHT_BUTTON);

- The menu callback function may look like this:

```
void processMenuItems(int id) {
switch (id) {
case 1 :
WOMEN();
BREAK
. . .
case 2 :
MEN ();
BREAK;
. . .
case 3 :
KIDS();
BERAK
exit(0);
```



Sub menues

- For the same menu function, the menus could be made heirarchical:

```
int submenu = glutCreateMenu(
    processMenuItems );
glutAddMenuEntry("Item-1", 1);
glutAddMenuEntry("Item-2", 2);
int main_menu= glutCreateMenu(
    processMenuItems );
glutAddSubMenu("Items", submenu);
void glutAddMenuEntry("Quit", 3);
void glutAttachMenu(GLUT_RIGHT_BUTTON);
```

Detach Menus

- void glutDetachMenu(int button);

  Parameters: button - the button to detach

- The button parameter takes the same values as for the *glutAttachMenu*