CSC 470 COMPUTER GRAPHICS

- Drawing Polylines Stored in a File
- Parameterizing Figures
- Menus

Display lists

- Display lists provide a way for OpenGL to redraw arbitrary primitives with a single call.
- Display lists compile a set of commands that draw a particular object.
- Only work with completely static geometry.

Display lists

```
int list;

// Some method called during initialization
void initialize_triangle (){
  list = glGenLists( 1 );
  glNewList( list, GL_COMPILE ); // starts the list
     glBegin( GL_TRIANGLE );
           glVertex2f( 0.0, 0.0 );
           glVertex2f( 0.0, 1.0 );
           glVertex2f( 1.0, 1.0 );
     glEnd();
  glEndList(); // finishes the list
}

// Sample drawing function
void display_callback(){
  glCallList( list ); // renders the compiled list
}
```

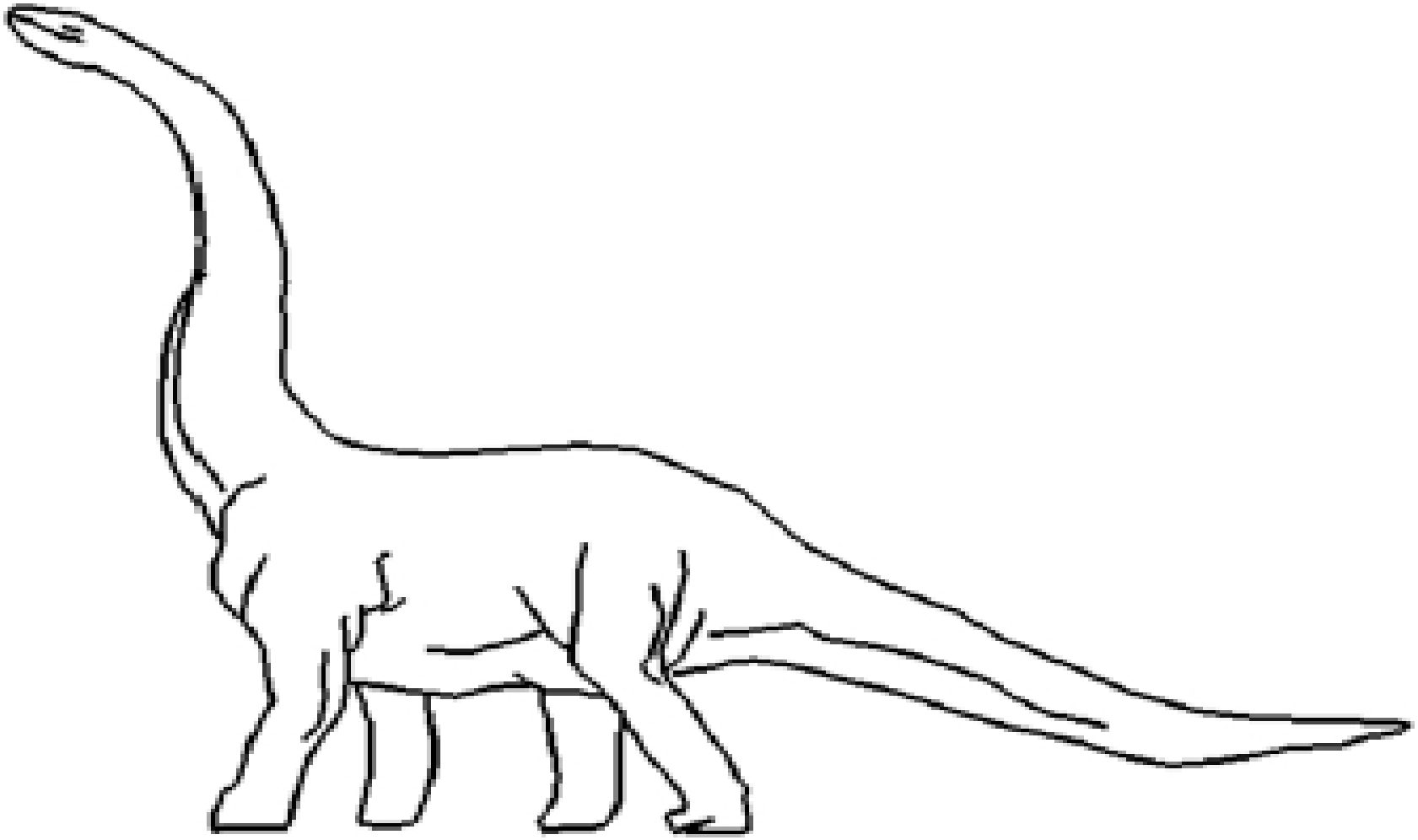
Display lists

- Pros:
  - Very fast, sometimes fastest of any method.
- Cons:
  - Only works with unchanging geometry.
  - Can consume a lot of GPU memory.

Drawing Polylines

- Drawing polyline from vertices in a file
  - # polylines
  - # vertices in first polyline
  - Coordinates of vertices, x y, one pair per line
  - Repeat last 2 lines as necessary
- File for dinosaur available from theWeb site
- Code to draw polylines/polygons in Fig. 2.1.

Example “dino”



Suppose the file dino.dat contains a collection of polylines, in the following format (the comments are not part of the file):

```
21          number of polylines in the file
4           number of points in the first polyline
169 118     first point of first polyline
174 120     second point of first polyline
179 124
178 126
5           number of points in the second polyline
298 86      first point of second polyline
304 92
310 104
314 114
314 119
29
32 435
10 439
. . . etc.
```

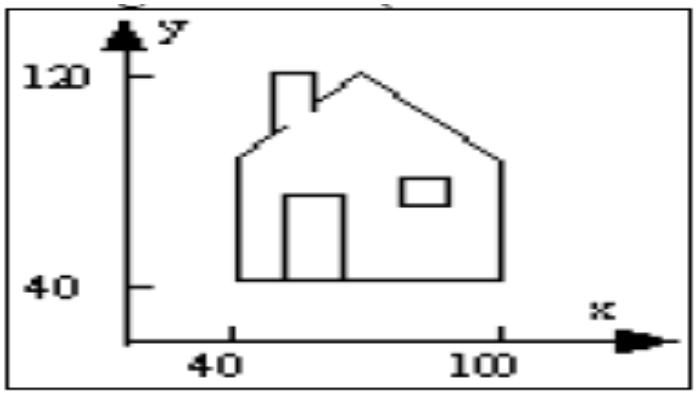
Fig. 2.1 A Function for Drawing polylines stored in a File


```
void drawPolyLineFile(char * fileName)
{
   fstream inStream;
   inStream.open(fileName, ios ::in); // open the file
   if(inStream.fail())
   return;
   glClear(GL_COLOR_BUFFER_BIT); // clear the screen
   GLint numpolys, numLines, x ,y;
   inStream >> numpolys; // read the number of polylines
   for(int j = 0; j < numpolys; j++) // read each polyline
              {
                 inStream >> numLines;
                 glBegin(GL_LINE_STRIP); // draw the next polyline
                 for (int i = 0; i < numLines; i++)
                           {
                              inStream >> x >> y; // read the next x, y pair
                              glVertex2i(x, y);
                           }
                           glEnd();
              }
   glFlush();
   inStream.close();
}
```

Parameterizing Figures

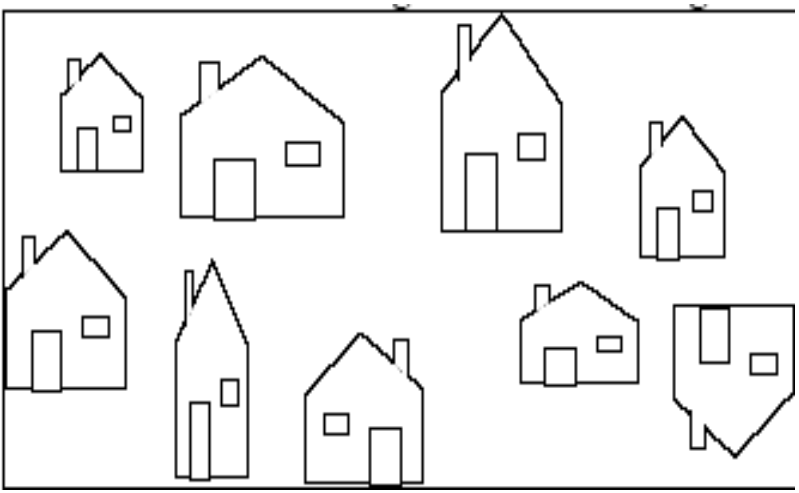
- Parameterizing Drawings: allows making them different sizes and aspect ratios
- Code for a parameterized house is in Fig. 2.2.

Fig. 2.2 Parameterizing Figures




```
void hardwiredHouse(void)
{
   glBegin(GL_LINE_LOOP);
      glVertex2i(40, 40);      // draw the shell of house
      glVertex2i(40, 90);
      glVertex2i(70, 120);
      glVertex2i(100, 90);
      glVertex2i(100, 40);
   glEnd();
   glBegin(GL_LINE_STRIP);
      glVertex2i(50, 100);     // draw the chimney
      glVertex2i(50, 120);
      glVertex2i(60, 120);
      glVertex2i(60, 110);
   glEnd();
      . . . // draw the door
      . . . // draw the window
}
```

Fig. 2.2 Drawing a village by calling parameterizedHouse with different parameters



```
void parameterizedHouse(GLintPoint peak, GLint width, GLint height)
 // the top of house is at the peak; the size of house is given
 //  by height and width
{
   glBegin(GL_LINE_LOOP);
     glVertex2i(peak.x,                peak.y);  // draw shell of house
     glVertex2i(peak.x + width / 2, peak.y - 3 * height /8);
     glVertex2i(peak.x + width / 2  peak.y -     height);
     glVertex2i(peak.x - width / 2, peak.y -     height);
     glVertex2i(peak.x - width / 2, peak.y - 3 * height /8);
   glEnd();
   draw chimney in the same fashion
   draw the door
   draw the window
}
```

Building a Polyline Drawer

```
class GLintPointArray{
   const int MAX_NUM = 100;
   public:
   int num;
   GLintPoint pt[MAX_NUM];
};

void drawPolyLine(GlintPointArray poly, int closed)
{
   glBegin(closed ? GL_LINE_LOOP : GL_LINE_STRIP);
      for(int i = 0; i < poly.num; i++)
            glVertex2i(poly.pt[i].x, poly.pt[i].y);
   glEnd();
   glFlush();
}
```

Relative Line Drawing

- Requires keeping track of current position on screen (CP).
- moveTo(x, y); set *CP* to (x, y)
- lineTo(x, y); draw a line from *CP* to (*x*, *y*), and then update *CP* to (*x*, *y*).
- Code is on the next slide.
- Caution! *CP* is a global variable, and therefore vulnerable to tampering from instructions at other points in your program.

```
class GLintPoint{
public:
GLint x, y;
};
void myInit(void)
{
}
GLintPoint CP;
void moveto(GLint x, GLint y)
{CP.x = x; CP.y = y; // update the CP}

void lineto(GLint x, GLint y)
{glBegin(GL_LINES); // draw the line
   glVertex2i(CP.x, CP.y);
   glVertex2i(x, y);
glEnd();
glFlush();
CP.x = x+10; CP.y = y+10; // update the CP}
void myDisplay(void)
{          GLintPoint temp={100,200};
           GLint x[100], y[100];
           x[0]=30;
           y[0]=30;
           moveto(x[0],y[0]);
           for(int i = 1; i < 10; i++)
           {          x[i]=x[i-1]+60;
                      y[i]=y[i-1]+90;
                      lineto(x[i], y[i]);
           }
           glFlush();
}

void main(int argc, char **argv)
{
}
```

What will change with the red line change?

Using Menus

- Both GLUT and GLUI make menus available.
- GLUT menus are simple, and GLUI menus are more powerful.
- We will build a single menu that will allow the user to change the color of a triangle, which is undulating back and forth as the application proceeds.

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

GLUT Menu Callback Function

- int glutCreateMenu(myMenu); returns menu ID
- void myMenu(int num); //handles choice num
- void glutAddMenuEntry(char* name, int value); // value used in myMenu switch to handle choice
- void glutAttachMenu(int button); // one of GLUT_RIGHT_BUTTON, GLUT_MIDDLE_BUTTON, or GLUT_LEFT_BUTTON
  - Usually GLUT_RIGHT_BUTTON

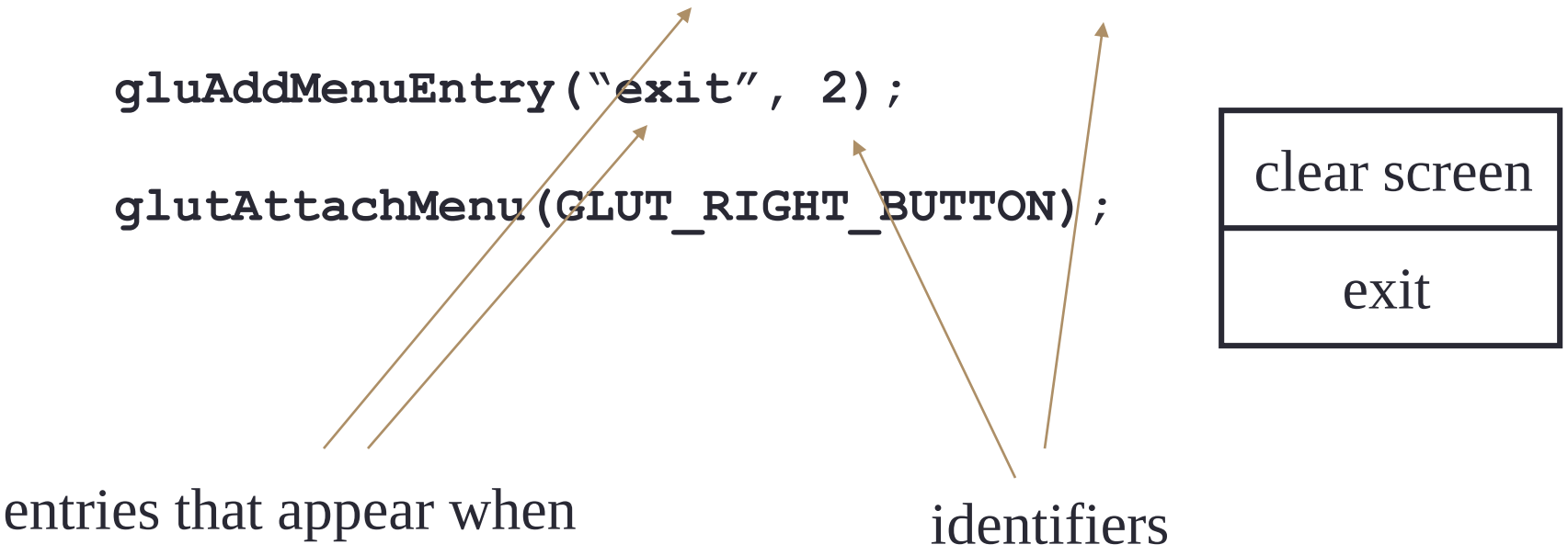
Defining a simple menu

- In **`main.c`**

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

gluAddMenuEntry("exit", 2);

glutAttachMenu(GLUT_RIGHT_BUTTON);
```



GLUT subMenus

- Create a subMenu first, using menu commands, then add it to main menu.
  - A submenu pops up when a main menu item is selected.
- glutAddSubMenu (char* name, int menuID); // menuID is the value returned by glutCreateMenu when the submenu was created
- Complete code for a GLUT Menu application is in Fig. 2.3 (No submenus are used)

Fig. 2.3. Defining a simple menu

```
glutCreateMenu(demo_menu);
glutAddMenuEntry("quit",1);
glutAddMenuEntry("increase square size", 2);
glutAddMenuEntry("decrease square size", 3);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```

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

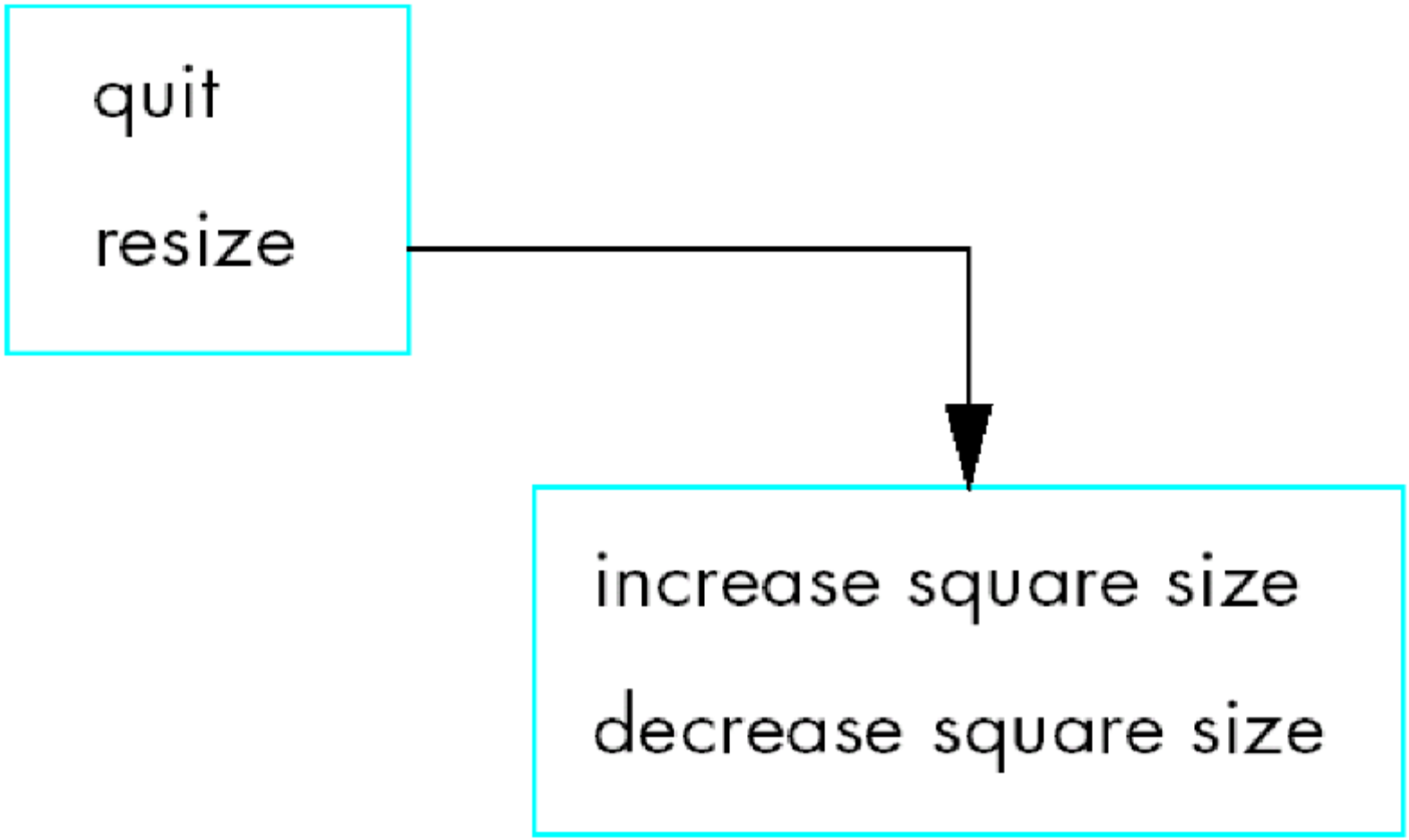
glutAddSubMenu(char *submenu_name, submenu id)

entry in parent menu

Menu Action

```
void demo_menu(int id)
{
      if(id == 1) exit( );
      else if (id == 2) size = 2 * size;
      else if (size > 1) size = size/2;
      glutPostRedisplay( );
}
```

Structure of Hierarchical Menus



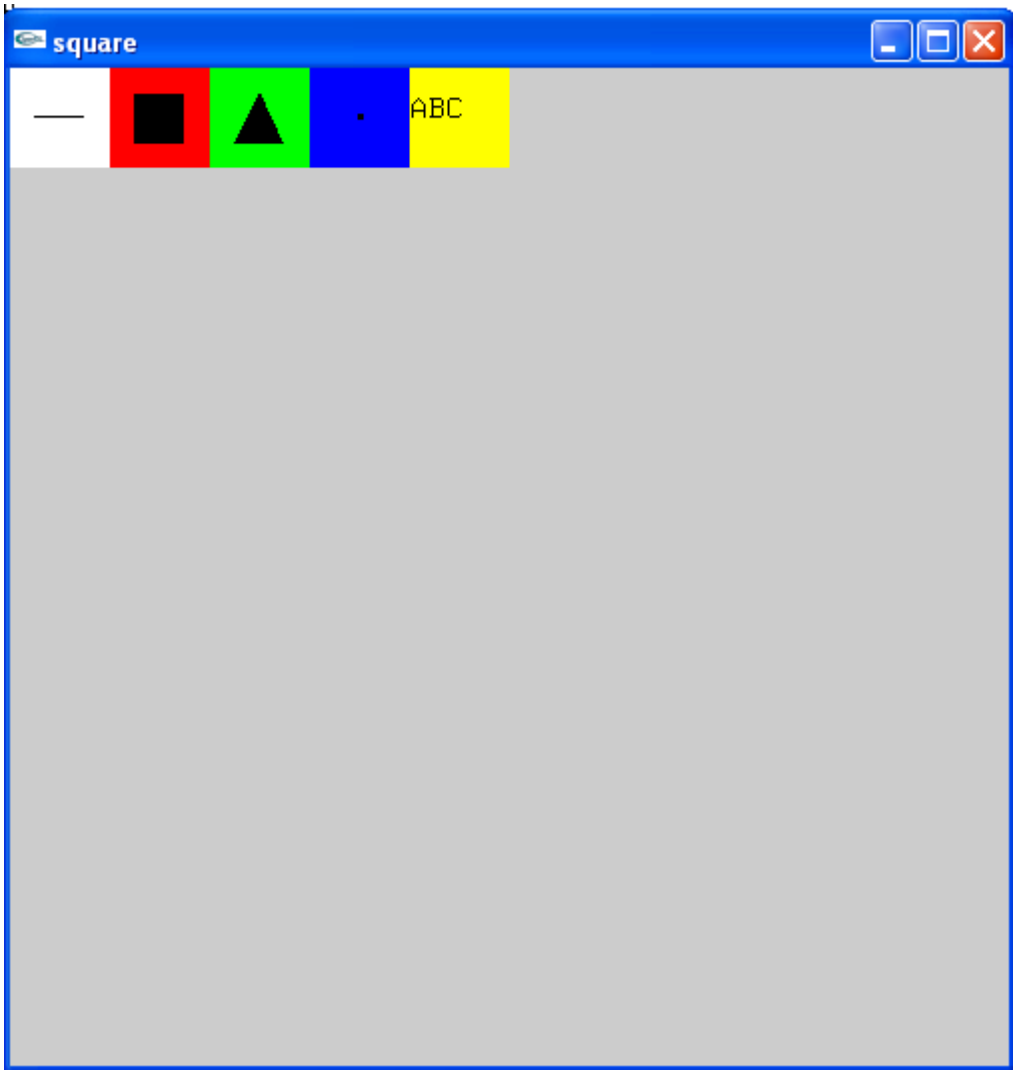
Code for hierarchical menu

```
sub_menu = glutCreateMenu(size_menu);
glutAddMenuEntry("Increase square size", 2);
glutAddMenuEntry("Decrease square size", 3);
glutCreateMenu(top_menu);
glutAddMenuEntry("Quit",1);
glutAddSubMenu("Resize", sub_menu);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```

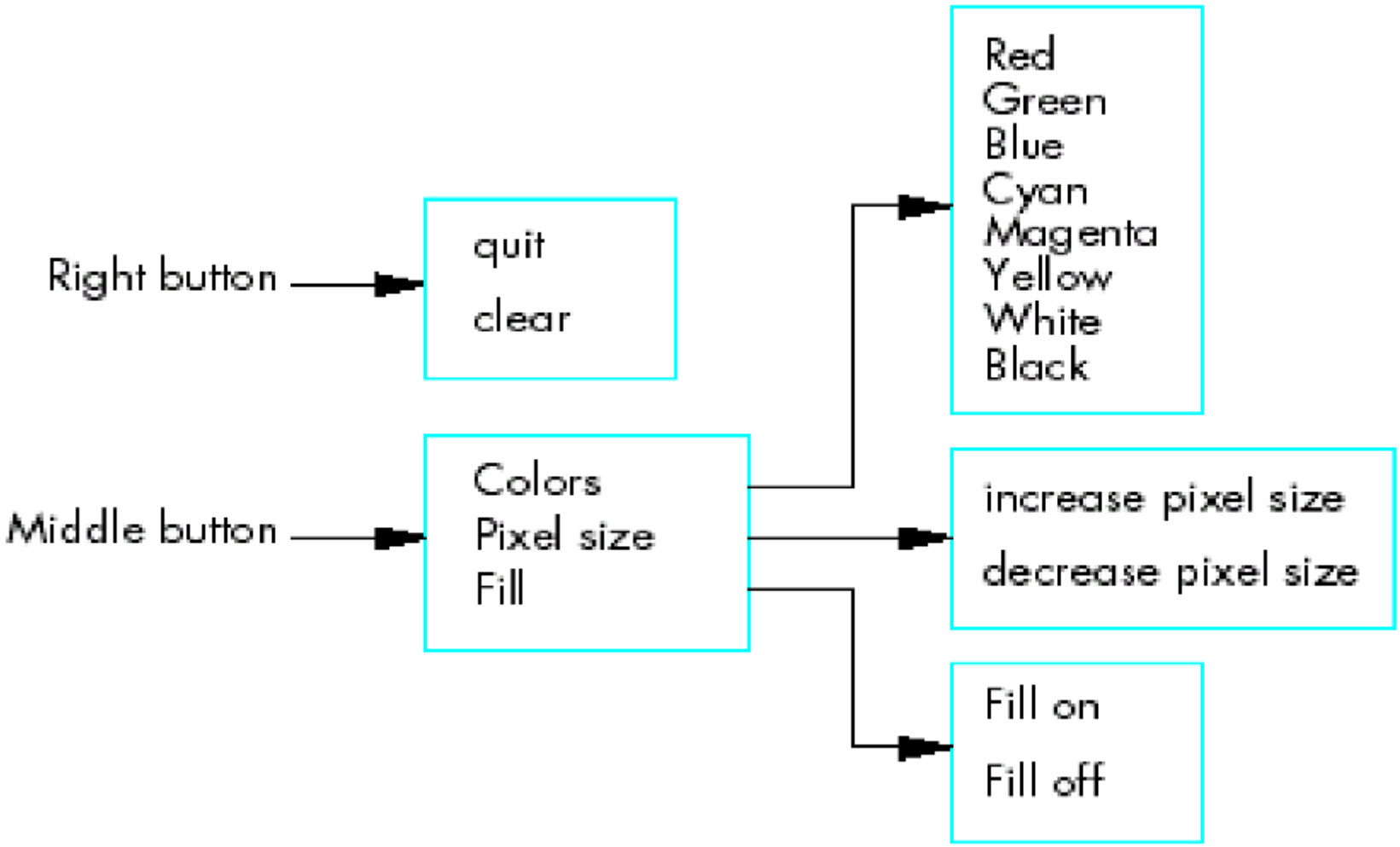
Simple Paint Program

- It should have the ability to work with geometric objects, such as line segments and polygons. Given that geometric objects are defined through vertices, the program should allow us to enter vertices interactively.
- It should have the ability to manipulate pixels, and thus to draw directly into the frame buffer.
- It should provide control of attributes such as color, line type, and fill patterns.
- It should include menus for controlling the application.
- It should behave correctly when the window is moved or resized.

User screen



Menu structure



Functions

```
void mouse(int btn, int state , int x, int y); /* mouse callback */
void key( unsigned char c, int x, int y);      /* keyboard callback
void display(void);                            /* display callback *
void drawSquare(int x, int y);                 /* random--color squa
                                                  function*/
void myReshape(GLsizei, GLsizei);              /* reshape callback *
void myinit(void);                             /* initialization fun
void screen_box(int x, int y, int s);          /* box--drawing
                                                  function */


void right_menu(int id);                       /* menu callbacks */
void middle_menu(int id);
void color_menu(int id);
void pixel_menu(int id);
void fill_menu(int id);
int pick(int x, int y);                        /* mode--selection
                                                  function */
```

Program main

```
int main(int argc, char **argv)
{
    int c_menu, p_menu, f_menu;

    glutInit(&argc,argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB)
    glutCreateWindow("Paint");
    glutDisplayFunc(display);
    c_menu = glutCreateMenu(color_menu);
    glutAddMenuEntry("Red",1);
    glutAddMenuEntry("Green",2);
    glutAddMenuEntry("Blue",3);
    glutAddMenuEntry("Cyan",4);
    glutAddMenuEntry("Magenta",5);
    glutAddMenuEntry("Yellow",6);
    glutAddMenuEntry("White",7);
    glutAddMenuEntry("Black",8);
```

Program main

```
    p_menu = glutCreateMenu(pixel_menu);
    glutAddMenuEntry("increase pixel size", 1);
    glutAddMenuEntry("decrease pixel size", 2);
    f_menu = glutCreateMenu(fill_menu);
    glutAddMenuEntry("fill on", 1);
    glutAddMenuEntry("fill off", 2);
    glutCreateMenu(right_menu);
    glutAddMenuEntry("quit",1);
    glutAddMenuEntry("clear",2);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    glutCreateMenu(middle_menu);
    glutAddSubMenu("Colors", c_menu);
    glutAddSubMenu("Pixel size", p_menu);
    glutAddSubMenu("Fill", f_menu);
    glutAttachMenu(GLUT_MIDDLE_BUTTON);
    myinit ();
    glutReshapeFunc (myReshape);
    glutMouseFunc (mouse);
    glutKeyboardFunc(key);
    glutMainLoop();

}
```

myInit( )

```
void myinit(void)
{
/* set up a font in display list */
        int i;
        base = glGenLists(128);
        for(i=0;i<128;i++)
        {
                glNewList(base+i, GL_COMPILE);
                glutBitmapCharacter(GLUT_BITMAP_9_BY_15, i);
    /* change to a stroke font by substituting next line */
                /*glutStrokeCharacter(GLUT_STROKE_ROMAN,i); */
                glEndList();
        }
        glListBase(base);

        glViewport(0,0,ww,wh);


/* Pick 2D clipping window to match size of X window.
This choice avoids the need to scale object coordinates
each time that the window is resized */
```

myInit( ) cont'd

```
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(0.0, (GLdouble) ww , 0.0, (GLdouble) wh ,
                                    -1.0, 1.0);

/* set clear color to black and clear window */

        glClearColor (0.0, 0.0, 0.0, 1.0);
        glClear(GL_COLOR_BUFFER_BIT);
        glFlush();
}
```

Code in mouse

```
case (TRIANGLE):          /* pick detected click in triangle box */
      switch(count)       /* switch on number of vertices */
      {
        case(0):          /* store first vertex */
          count++;
          xp[0] = x;
          yp[0] = y;
          break;
        case(1):          /* store second vertex */
          count++;
          xp[1] = x;
          yp[1] = y;
          break;
```

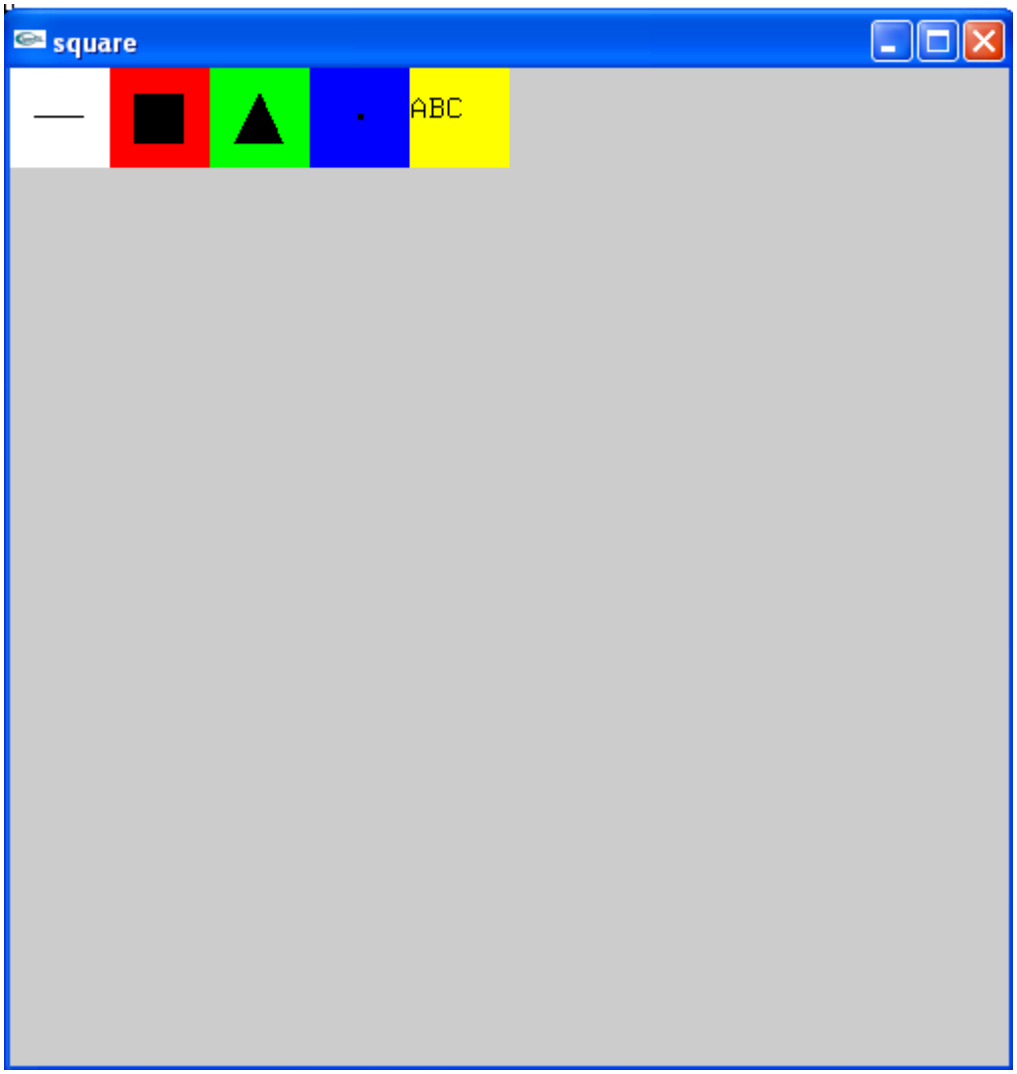
Code in mouse cont'd

```
case(2):          /* third vertex: draw triangle */
  if(fill) glBegin(GL_POLYGON);
  else glBegin(GL_LINE_LOOP);
     glVertex2i(xp[0],wh-yp[0]);
     glVertex2i(xp[1],wh-yp[1]);
     glVertex2i(x,wh-y);
  glEnd();
  draw_mode=0;  /* reset mode */
  count=0;
```

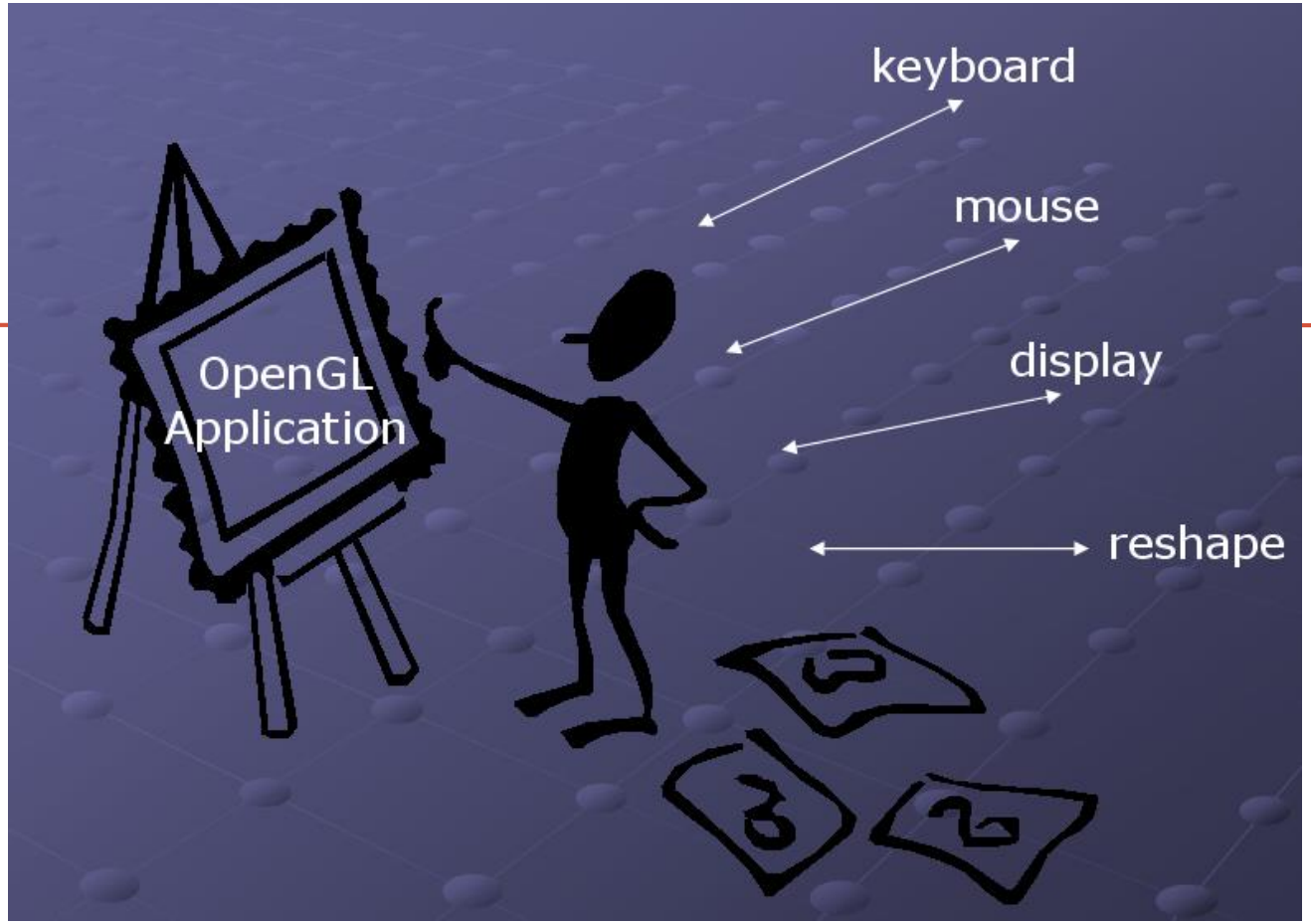
Function key

```
void key(unsigned char k, int xx, int yy)
{
   if(draw_mode!=TEXT) return; /* draw characters until
                                  mode changes */
   glRasterPos2i(rx,ry);
   glCallList(k);  /* Display list for character k */
   rx+=glutBitmapWidth(GLUT_BITMAP_9_BY_15,k);
}
```

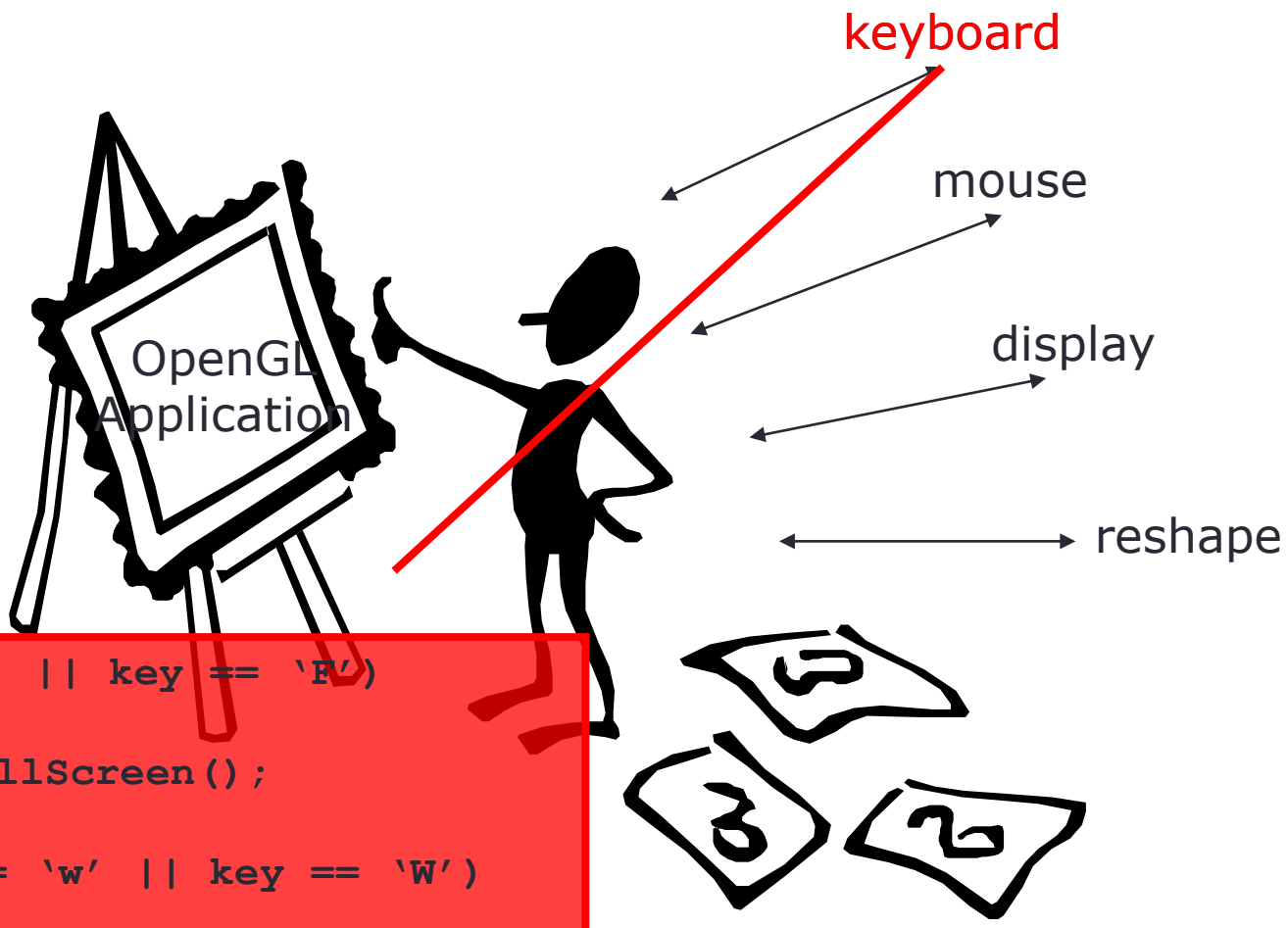
User screen



ANATOMY OF GUI



Anatomy of GLUT




```
if (key == 'f' || key == 'F')
{
        glutFullScreen();
}
else if(key == 'w' || key == 'W')
{
        glutReshapeWindow(640,480);
}
```

Anatomy of GLUT

```
void myKeyboard(unsigned char key, int mouseX, int
  mouseY)
```

Runs whenever a keyboard event occurs (e.g. user presses a key)

```
void myMouse(int button, int state, int x, int y)
```

Runs whenever a mouse event occurs (e.g. user presses a mouse button, mouse moves)

```
void myDisplay(void)
```

Runs whenever the system determines that the window must be redrawn (e.g. window comes to the front, window has been moved)

```
void myReshape(void)
```

Runs whenever a window changes size

Anatomy of GLUT

```
void myKeyboard(unsigned char key, int mouseX, int mouseY)
{
  if (key == 'f' || key == 'F')
  {
    glutFullScreen();
  }
  else if(key == 'w' || key == 'W')
  {
    glutReshapeWindow(640,480);
  }
}
```

Anatomy of GLUT

```
int main()
{
 //initialise things
 //create a screen window
 glutDisplayFunc(myDisplay);
 glutReshapeFunc(myReshape);
 glutMouseFunc(myMouse);
 glutKeyboardFunc(myKeyboard);
 //initialise other things
 glutMainLoop();
}
```

Opening a Window

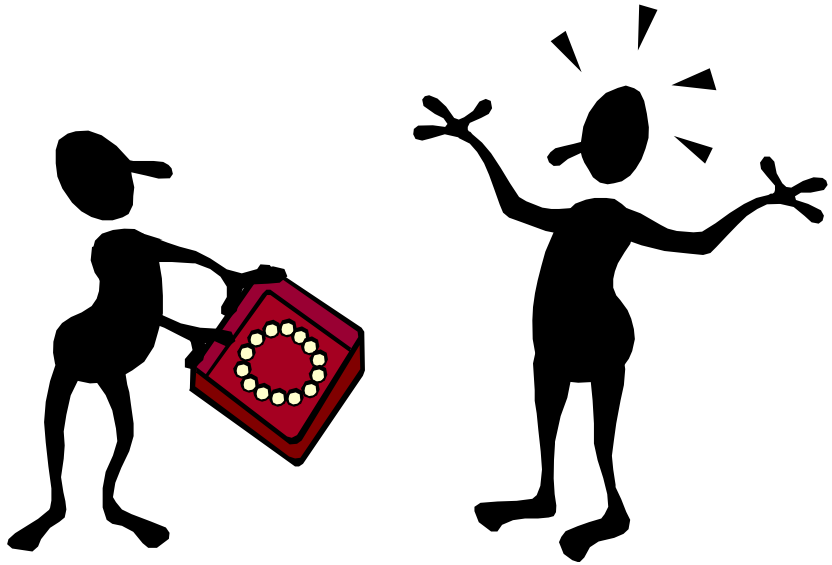
```
int main(int argc, char **argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT RGB);
   glutInitWindowSize(640,480);
   glutInitWindowPosition(100,100);
   glutCreateWindow("My OpenGL Window");

   glutDisplayFunc(myDisplay);
   glutReshapeFunc(myReshape);
   glutMouseFunc(myMouse);
   glutKeyboardFunc(myKeyboard);

   myInit();
   glutMainLoop();
}
```

Example: Mouse Plots

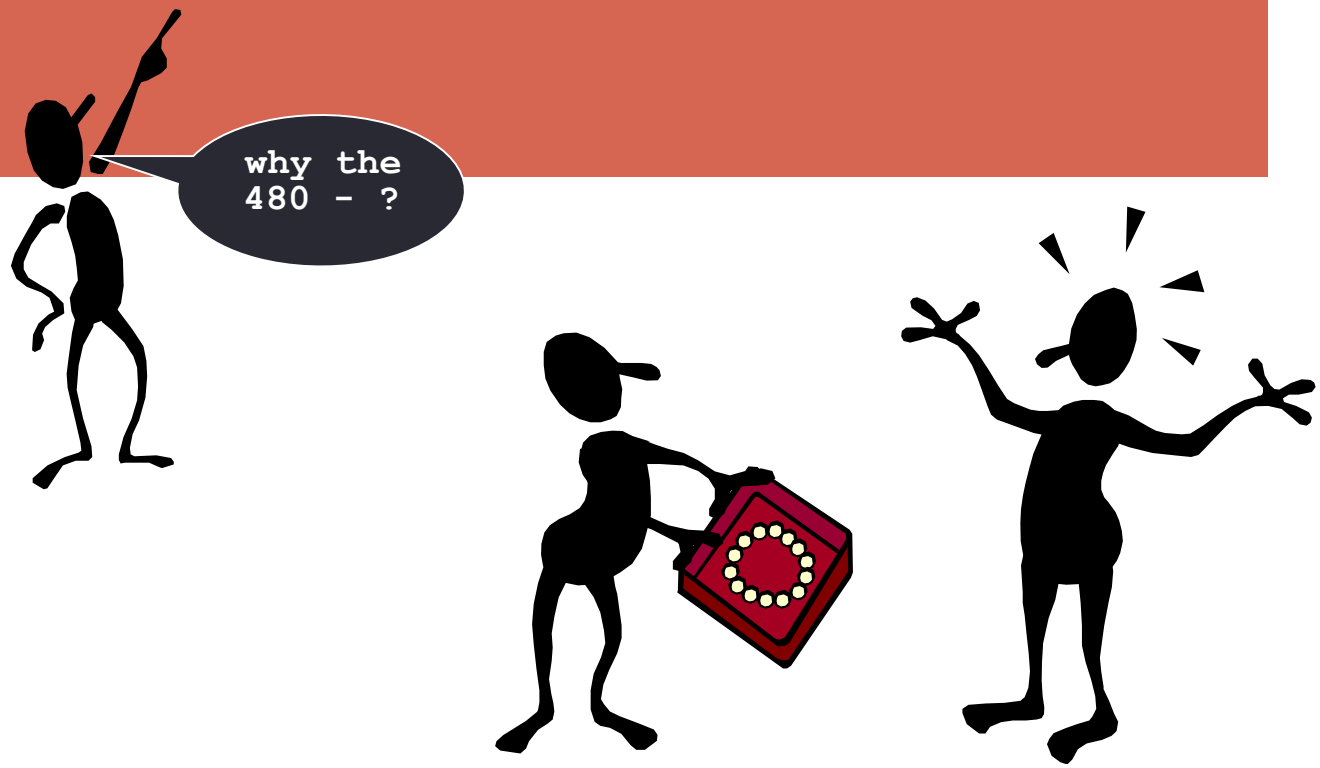
- How do you *draw a dot* in the window at the *mouse location* when the user clicks on the *left mouse button*?



Example: Mouse Plots

```
void myMouse(int button, int state, int x, int y)
{
        if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
        {
            drawDot(x, 480 - y);
        }

}
```



Example: Keyboard Plots

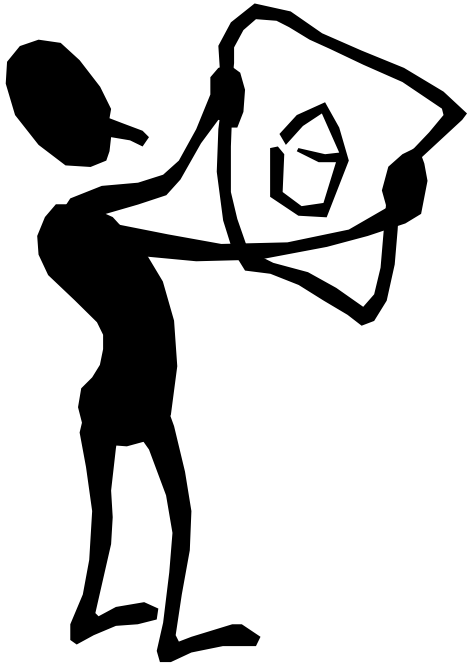
- How do you plot points in the window at the current mouse location when the ‘p’ key is pressed?

Example: Keyboard Plots

```
void myKeyboard(unsigned char key, int mx, int my)
{
        GLint x = mx, y = 480 - my;
        switch(key)
        {
            case 'p':
                drawDot(x,y);
                break;
            case 27:    //ESC
                exit(-1);
            default:
                break;
        }
}
```

Example: Mouse Drawing

- How do you use the mouse to draw freehand in a window?



Example: Mouse Drawing

```
void myMovedMouse(int mouseX, int mouseY)
{
        GLint x = mouseX;
        GLint y = 480 - mouseY;
        GLint brushSize = 20;
        glRecti(x,y, x+ brushSize, y + brushSize);
        glFlush();
}
```