Intoduction to OpenGL(continue)

- **A simple program**

```
#include <GL/glut.h>
#define SIZE 500.0

void display (void)
{
 glClear(GL_COLOR_BUFFER_BIT);
}

void init (void)
{
}

int main(int argc, char *argv[])
{

 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
 glutInitWindowSize(SIZE, SIZE);
 glutInitWindowPosition(100, 50);
 glutCreateWindow("My first OpenGL program");
 init();
 glutDisplayFunc(display);
 glutMainLoop();
 return 0;
}
```

- **A simple display() function**

```
void display (void)
{
 glClear(GL_COLOR_BUFFER_BIT);
 glColor3f(0.0, 0.0, 0.0);
 glBegin(GL_LINES);
   glVertex2f(-1.0, 0.0);
   glVertex2f(1.0, 0.0);
 glEnd();
}
```

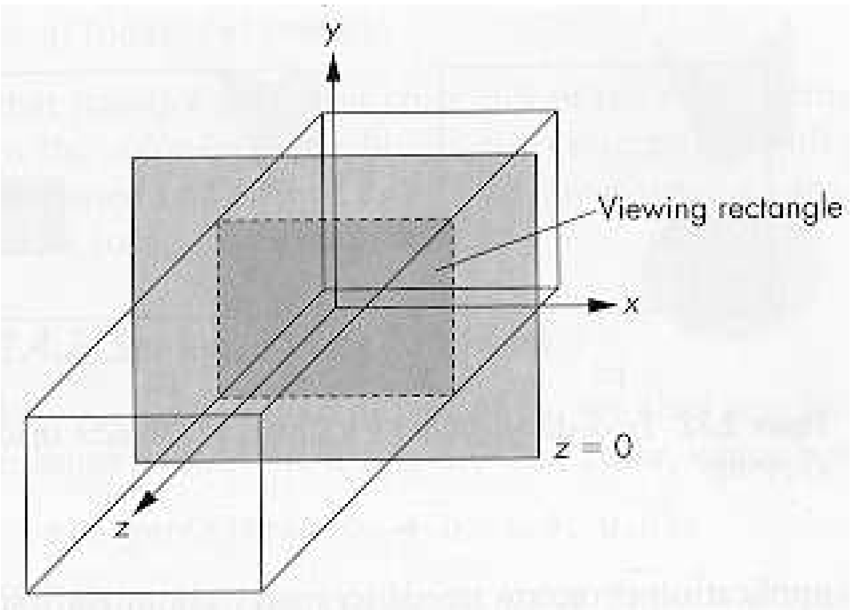
- **Another display() function**

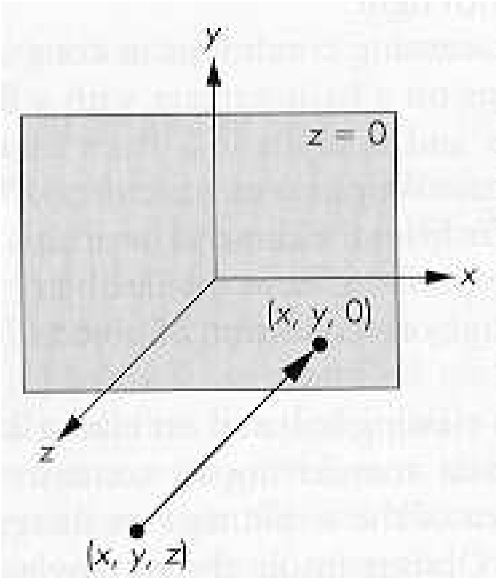
```
void display (void)
{
 int i;

 glClear(GL_COLOR_BUFFER_BIT);
 glColor3f(1.0, 1.0, 0.0);
 glBegin(GL_LINES);
   for(i=-25; i<=25; i++) {
     float x = i / 25.0;
     glVertex2f(x, 0.0);
     glVertex2f(x, 0.0);
   }
 glEnd();
}
```

- **Initialization**

```
void init (void)
{
 glClearColor(1.0, 1.0, 1.0, 0.0);
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 glOrtho(-1, 1, -1, 1, -1, 1);
/* glOrtho(left, right, bottom, top, near, far); */
}
```





- **The Idle function**

  * OpenGL calls the idle function when it has nothing else to do

  * Callback function: *void idle_callback(void)*

  * Registering function: *void glutIdleFunc(idle_callback)*

  * Usually, the idle function is used to provide continuous animation

```
void display (void)
{
 glClear(GL_COLOR_BUFFER_BIT);
 glColor3f(1.0, 0.0, 0.0);
 glMatrixMode(GL_MODELVIEW);
 glLoadIdentity();
 glRotatef(xangle, 1.0, 0.0, 0.0);
 glRotatef(yangle, 0.0, 1.0, 0.0);
 glRotatef(zangle, 0.0, 0.0, 1.0);
 glutWireCube(1.0);
}
```

```
/* idle function */

void spin (void)
{
 xangle += 1.0;
 glutPostRedisplay();
}
```

spin() must be registered: *glutIdleFunc(spin);;*

- **The Keyboard function**

```
void readkeys(unsigned char key, int x, int y)
{
  switch (key)
  {
    case 'x':
        xangle += 10.0;
        glutPostRedisplay();
        break;
    case 'y':
        yangle += 10.0;
        glutPostRedisplay();
        break;
    case 'z':
        zangle += 10.0;
        glutPostRedisplay();
        break;
    default:
        break;
  }
}
```

readkeys() must be registered: *glutKeyboardFunc(readkeys);*

- **Mouse Buttons**

```
void mousepos(int button, int state, int x, int y)
{
 if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
 {
  xangle = x * 360.0 / SIZE;
  glutPostRedisplay();
 }
 else if (button == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN)
 {
  yangle = x * 360.0 / SIZE;
  glutPostRedisplay();
 }
 else if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
 {
  zangle = x * 360.0 / SIZE;
  glutPostRedisplay();
 }
}
```

mousepos() must be registered: *glutMouseFunc(mousepos);*

- **Dragging the Mouse**

```
void drag(int x, int y)
{
 zangle= x * 360.0 / SIZE;
 xangle= y * 360.0 / SIZE;
 glutPostRedisplay();
}
```

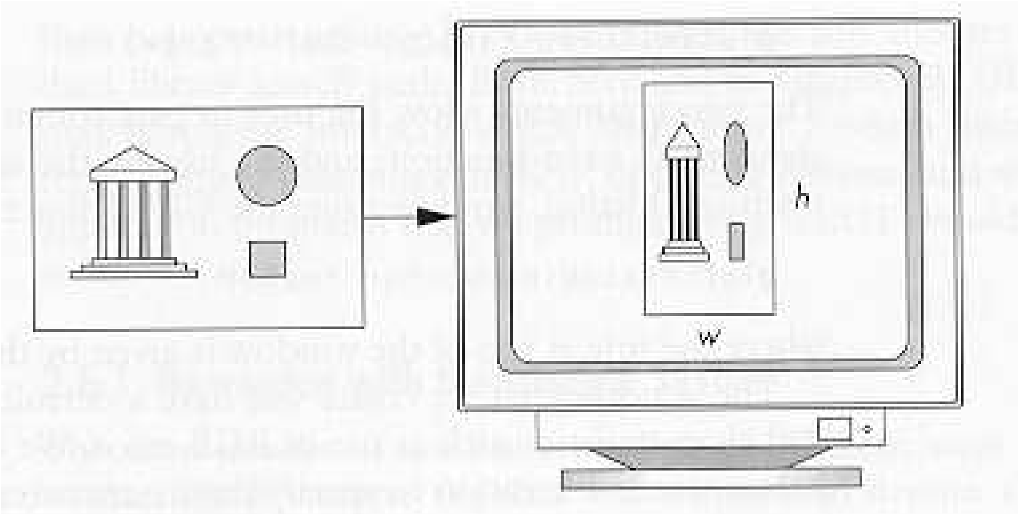
drag() must be registered:

glutMotionFunc(drag); or *glutPassiveMotionFunc(drag);*

- **Resizing the Graphics Window**

  - <u>Aspect ratio of a rectangle:</u> the ratio of rectangle's width to its height

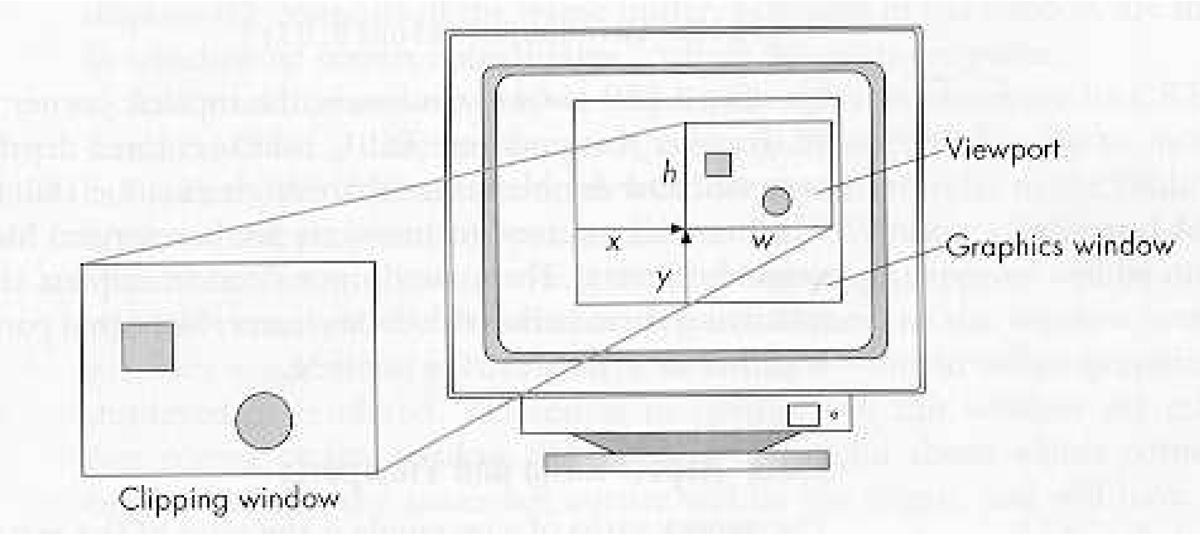
  - The aspect ratio of the viewing rectangle (specified by *glOrtho()*) must be the same with the aspect ratio of the graphics window (specified by *glutInitWindowSize()*)



  - <u>Viewport:</u> a rectangular area of the graphics window where the output is displayed (usually, the same as the graphics window) - see pages 177-179

  - We can avoid distortion if we ensure that the viewing rectangle and the viewport have the same aspect ratio

void glViewport(GLint x, GLint y, GLsizei w, GLsizei h)



```
void reshape (int width, int height)
{
 float aspect_ratio;
 float hvvw;      /* half width of viewing volume */
 float hvvh;      /* half height of viewing volume */

 aspect_ratio = (float) width / (float) height;

 if(width >= height) {
   hvvh = 1.0;
   hvvw = aspect_ratio;
  }
 else {
   hvvh = 1.0/aspect_ratio;
   hvvw = 1.0;
  }

 glViewport(0, 0, width, height);
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 glOrtho(-hvvw, hvvw, -hvvh, hvvh, -1, 1);
}
```

Registering function: *glutReshapeFunc(reshape);*

- **Menus**

```
#define M_RED 20
#define M_GREEN 21
#define M_BLUE 22

void initialize_menu (void)
{
 glutCreateMenu(choose_color);
 glutAddMenuEntry("Red", M_RED);
 glutAddMenuEntry("Green", M_GREEN);
 glutAddMenuEntry("Blue", M_BLUE);
 glutAttachMenu(GLUT_RIGHT_BUTTON);
}
```

```
void choose_color(int sel)
{
 switch(sel) {
  case M_RED:
      glColor3f(1.0, 0.0, 0.0);
      break;
  case M_GREEN:
      glColor3f(0.0, 1.0, 0.0);
      break;
  case M_BLUE:
      glColor3f(0.0, 0.0, 1.0);
      break;
  default:
      break;
 }
}
```

- **Submenus**

- Write code for the main menu after the submenus

```
void initialize_menu (void)
{
 int sub1;

 sub1 = glutCreateMenu(choose_red);
 glutAddMenuEntry("Black", M_BLACK);
 glutAddMenuEntry("Yellow", M_YELLOW);

 glutCreateMenu(choose_color);
 glutAddSubMenu("Red",sub1);
 glutAddMenuEntry("Green", M_GREEN);
 glutAddMenuEntry("Blue", M_BLUE);
 glutAttachMenu(GLUT_RIGHT_BUTTON);

}
```

```
#define M_BLACK 23
#define M_YELLOW 24

void choose_red(int sel)
{
 switch(sel) {
   case M_BLACK:
        glColor3f(0.0, 0.0, 0.0);
        break;
   case M_YELLOW:
        glColor3f(1.0, 1.0, 0.0);
        break;
   default:
        break;
 }
}
```