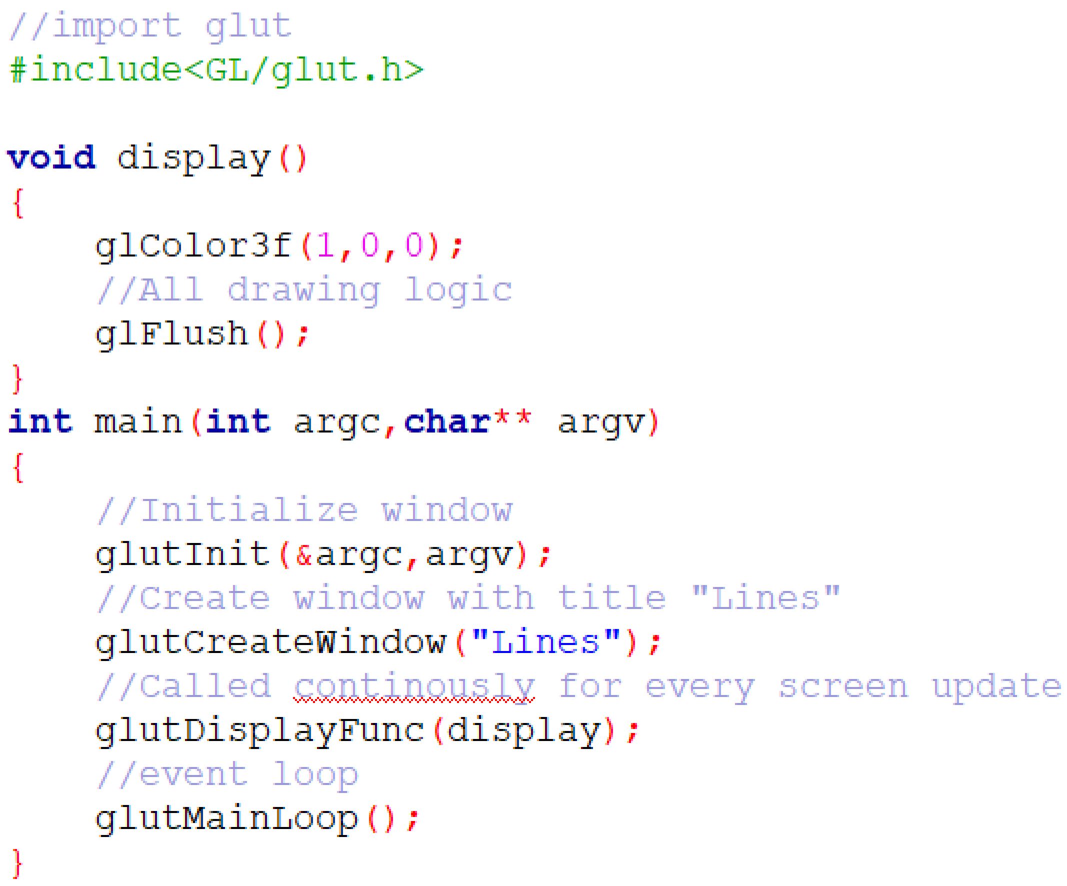
CSC354 – <Lab Manual>

Prepared by: Mr. Aoun Haider

Computer Graphics using OpenGL

Introduction to OpenGL:

Basic Structure:



```
//import glut
#include<GL/glut.h>

void display()
{
    glColor3f(1,0,0);
    //All drawing logic
    glFlush();
}
int main(int argc,char** argv)
{
    //Initialize window
    glutInit(&argc,argv);
    //Create window with title "Lines"
    glutCreateWindow("Lines");
    //Called continously for every screen update
    glutDisplayFunc(display);
    //event loop
    glutMainLoop();
}
```

OpenGL: for modeling, viewing lighting and clipping.

```
gluOrtho(left,right,bottom,top);
```

GLUT: Creation of common objects like spheres, torus, tetrahedron etc.

```
glutCreateWindow(“Computer Graphics Lab”);
```

```
glutDisplayFunction(mydisplay);
```

GL: For windowing, interaction system (Window management, mouse interaction and menus)

All functions which belong to GL are written with the prefix *gl*.

```
glVertex2f(1.0,0.0); //Create a point at (x,y)
```

Primitives and Attributes:

1. **Points**
2. **Lines**
3. **Curves**
4. **Polygons**
5. **Surfaces**

```
glBegin(primType);
```

…

```
glEnd();
```

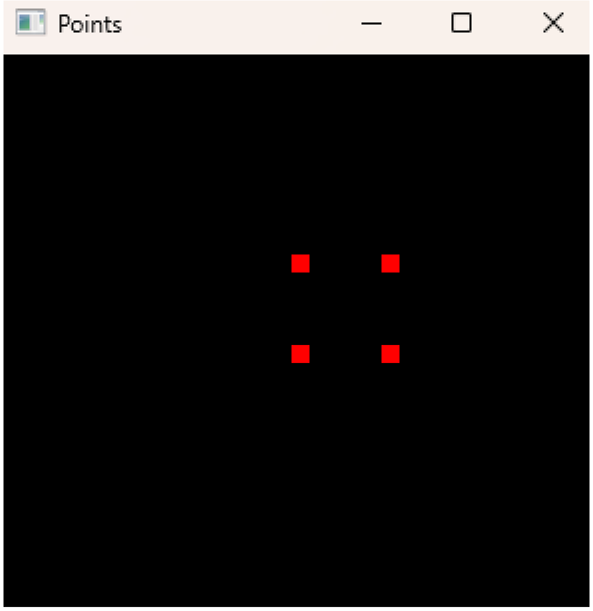
1. **Points**
   ```
   glBegin(GL_POINTS);
   glVertex2f(x,y);
   glEnd();
   ```
2. **Lines**
   ```
   glBegin(GL_LINES);
   glVertex2f(x1,y1);
   glVertex2f(x2,y2);
   glEnd();
   ```
3. **Triangle**
   ```
   glBegin(GL_TRIANGLES);
   glVertex2f(x1,y1);
   glVertex2f(x2,y2);
   glVertex2f(x3,y3);
   glEnd();
   ```

Primitives:

GL_POINTS, GL_LINES, GL_POLYGON, GL_TRIANGLES, GL_QUADS, GL_LINE_STRIP, GL_LINE_LOOP, GL_QUAD_STRIP, Gl_TRIANGLE_STRIP, GL_TRIANGLE_FAN

GL_POINTS:

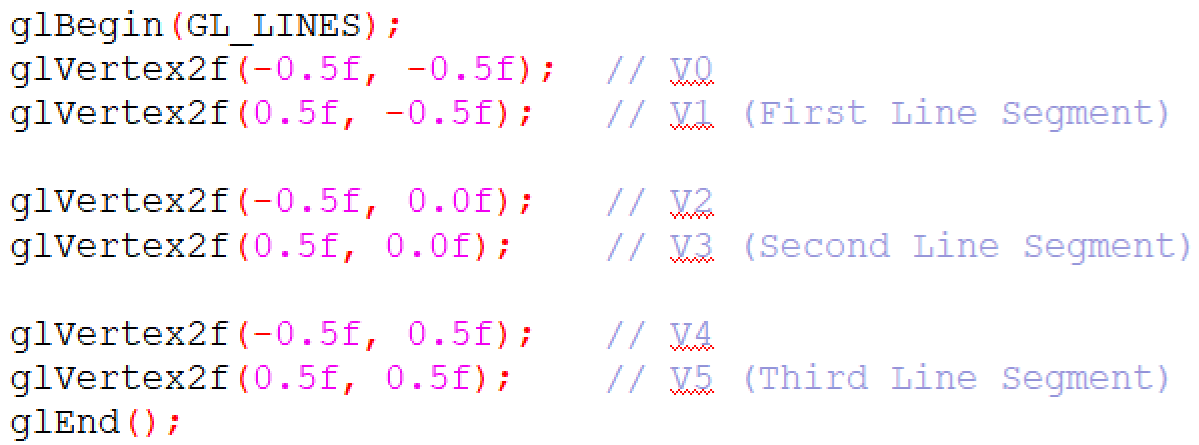
```
glBegin(GL_POINTS);
glVertex2f(0,0);
glVertex2f(0,0.3);
glVertex2f(0.3,0.3);
glVertex2f(0.3,0);
glEnd();
```



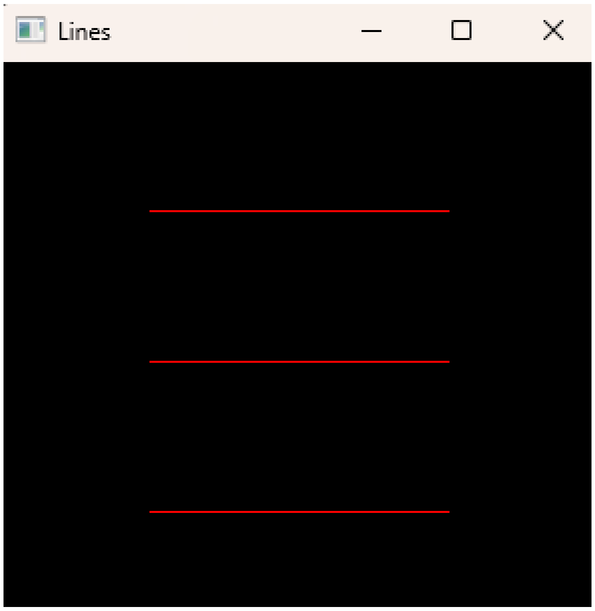
GL_LINES:

Consider a set of vertices V0, V1, V2, V3, V4, V5. Points are not shared.

- First line segment: V0, V1
- Second line segment: V2, V3
- Third line segment: V4, V5



```
glBegin(GL_LINES);
glVertex2f(-0.5f, -0.5f);   // V0
glVertex2f(0.5f, -0.5f);    // V1 (First Line Segment)

glVertex2f(-0.5f, 0.0f);    // V2
glVertex2f(0.5f, 0.0f);     // V3 (Second Line Segment)

glVertex2f(-0.5f, 0.5f);    // V4
glVertex2f(0.5f, 0.5f);     // V5 (Third Line Segment)
glEnd();
```

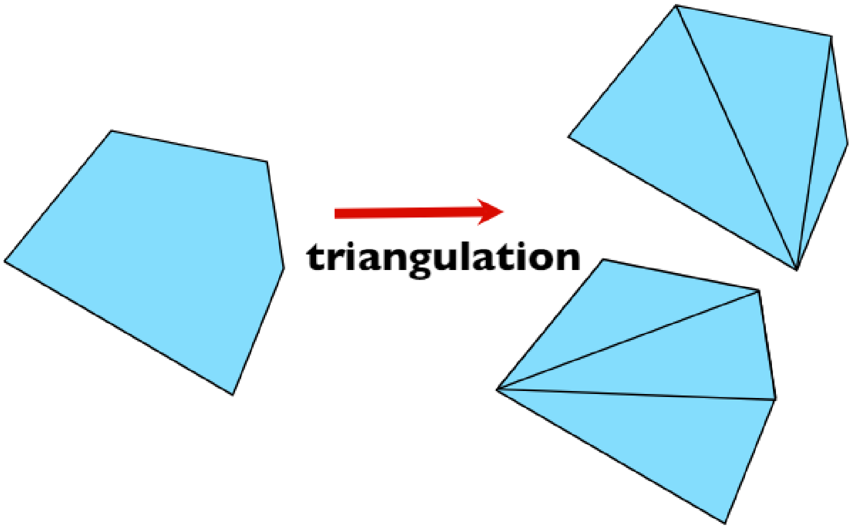


GL_TRIANGLE_STRIP:

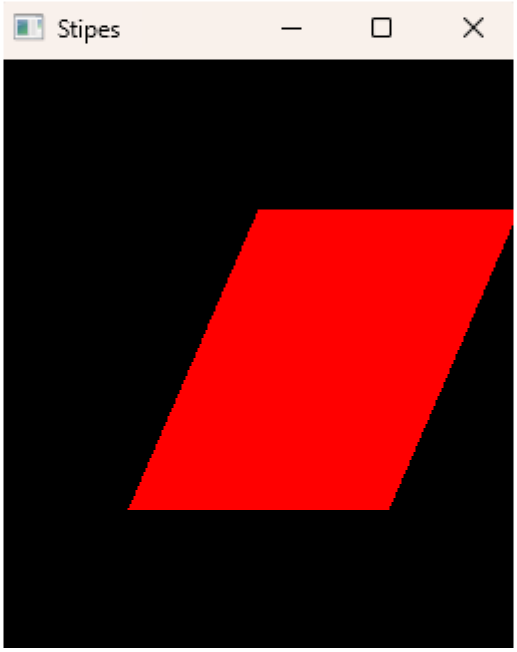
Consider a set of vertices V_0, V_1, V_2, V_3, V_4. Points are shared.

- First triangle: V_0, V_1,V_2
- Second triangle: V_1, V_2,V_3
- Third triangle: V_2, V_3,V_4

We can represent complex shapes using different triangles like below:



```
glBegin(GL_TRIANGLE_STRIP);
glVertex2f(-0.5f, -0.5f);
glVertex2f(0.0f, 0.5f);
glVertex2f(0.5f, -0.5f);
glVertex2f(1.0f, 0.5f);
glEnd();
```



GL_POLYGON:

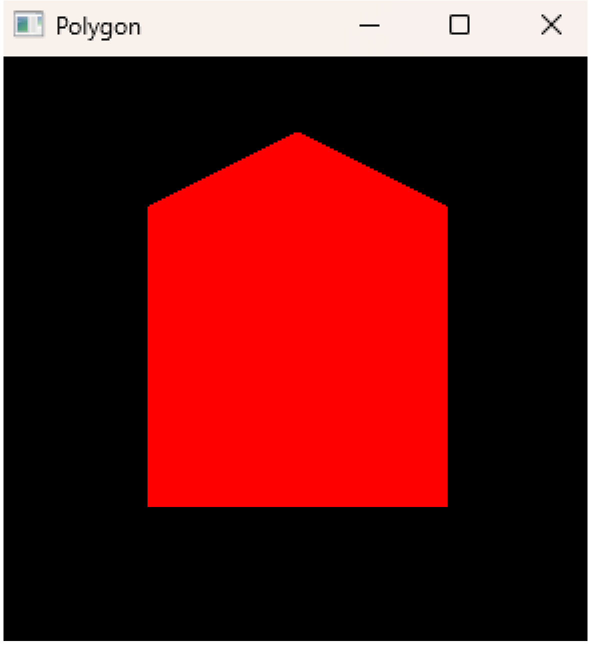
Consider a list of vertices V0, V1, V2, V3, V4.

- The vertices will be connected in the order provided.

Author: Mr. Aoun Haider

- The polygon is closed by connecting V4 back to V0.
- Vertices should be specified in a consistent order.
- Specify the boundary of polygon and dots are connected in a loop fashion.

```
glBegin(GL_POLYGON);
glVertex2f(-0.5f, -0.5f);   // V0
glVertex2f(-0.5f, 0.5f);    // V1
glVertex2f(0.0f, 0.75f);    // V2
glVertex2f(0.5f, 0.5f);     // V3
glVertex2f(0.5f, -0.5f);    // V4
glEnd();
```

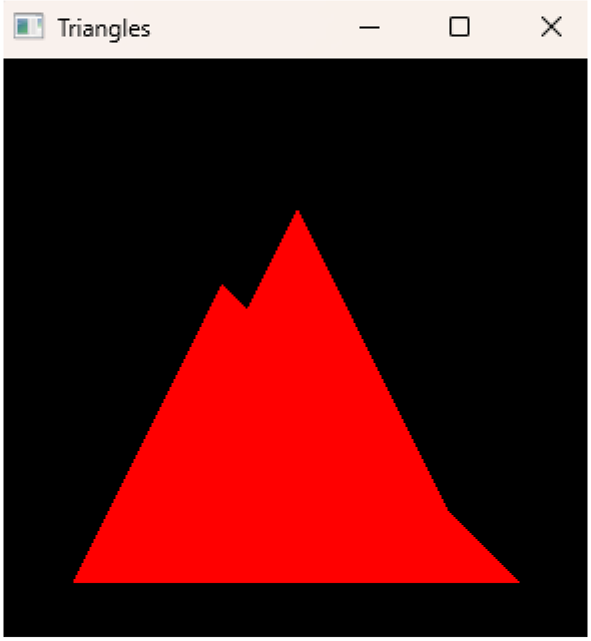


GL_TRIANGLES:

Consider a list of vertices V0, V1, V2, V3, V4,V5. Points are not shared.

- First triangle: V0, V1, V2
- Second triangle: V3, V4,V5

```
glBegin(GL_TRIANGLES);
// First Triangle
glVertex2f(-0.5f, -0.5f);   // V0
glVertex2f(0.0f, 0.5f);     // V1
glVertex2f(0.5f, -0.5f);    // V2

// Second Triangle
glVertex2f(-0.75f, -0.75f);   // V3
glVertex2f(-0.25f, 0.25f);    // V4
glVertex2f(0.75f, -0.75f);    // V5
glEnd();
```

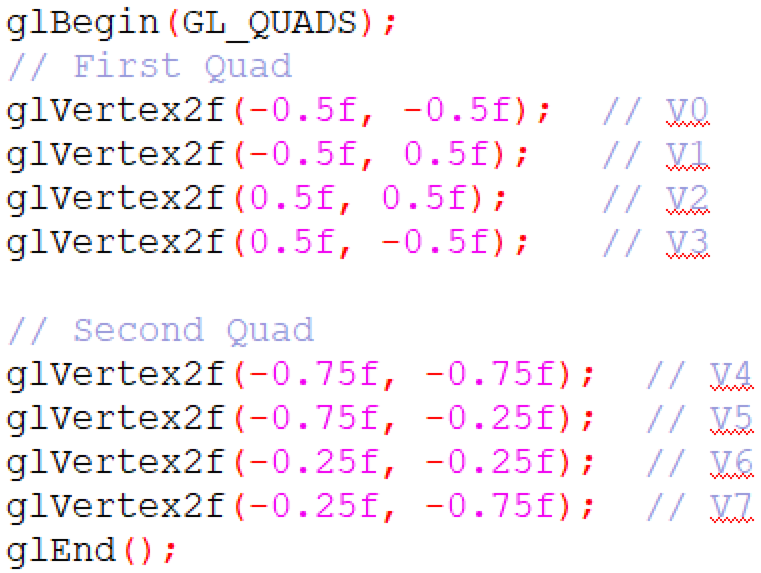


GL_QUADS:

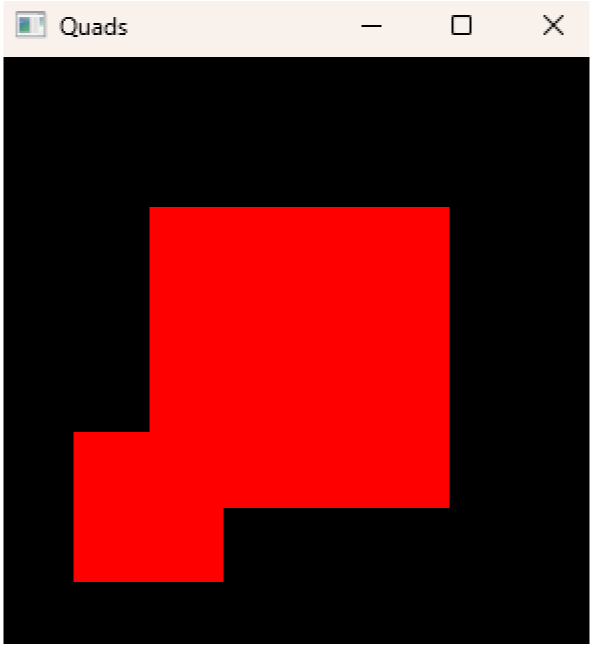
It is used to draw quadrilaterals. Points are not shared.

Consider a list of vertices V0, V1, V2, V3, V4,V5, V6, V7. Points are not shared.

- First triangle: V0, V1, V2, V3
- Second triangle: V4, V5,V6, V7



```
glBegin(GL_QUADS);
// First Quad
glVertex2f(-0.5f, -0.5f);   // V0
glVertex2f(-0.5f, 0.5f);    // V1
glVertex2f(0.5f, 0.5f);     // V2
glVertex2f(0.5f, -0.5f);    // V3

// Second Quad
glVertex2f(-0.75f, -0.75f);   // V4
glVertex2f(-0.75f, -0.25f);   // V5
glVertex2f(-0.25f, -0.25f);   // V6
glVertex2f(-0.25f, -0.75f);   // V7
glEnd();
```



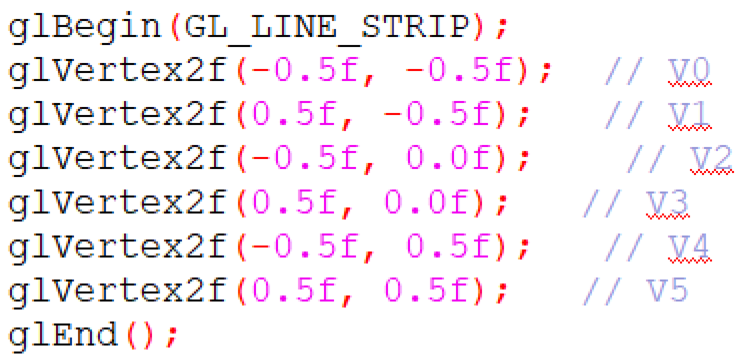
GL_LINE_STRIP:

Consider a list of vertices V0, V1, V2, V3. Points are shared.

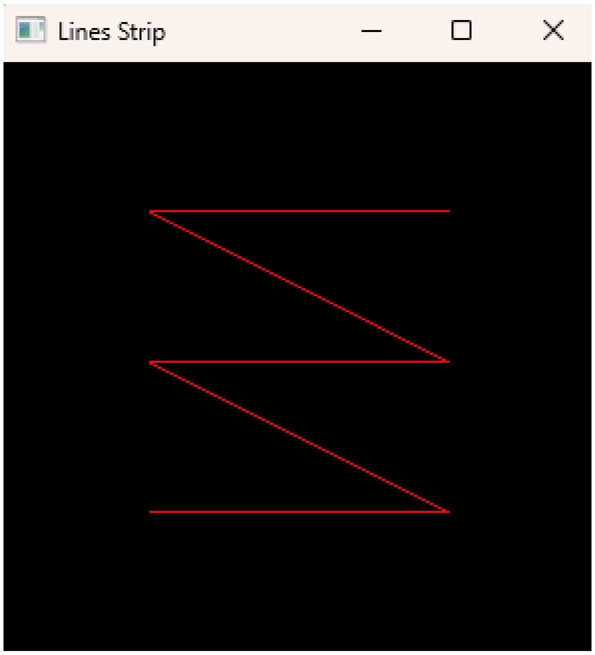
First line segment: V0 to V1

Second line segment: V1 to V2

Third line segment: V2 to V3



```
glBegin(GL_LINE_STRIP);
glVertex2f(-0.5f, -0.5f);  // V0
glVertex2f(0.5f, -0.5f);   // V1
glVertex2f(-0.5f, 0.0f);    // V2
glVertex2f(0.5f, 0.0f);   // V3
glVertex2f(-0.5f, 0.5f);   // V4
glVertex2f(0.5f, 0.5f);   // V5
glEnd();
```



GL_LINE_LOOP:

It is similar to GL_LINE_STRIP with one difference, it automatically connects last point with first creating a loop.

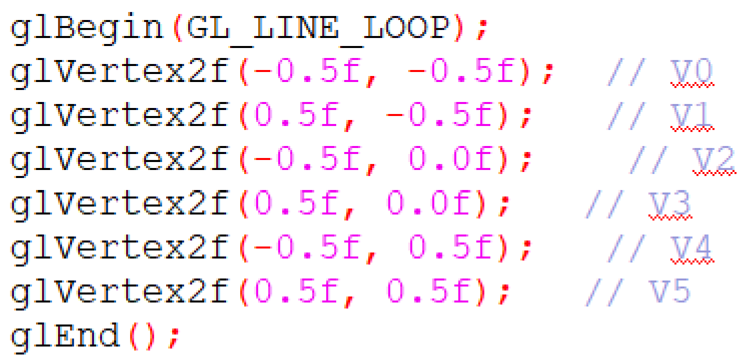
Consider a list of vertices V0, V1, V2, V3. Points are shared.

First line segment: V0 to V1

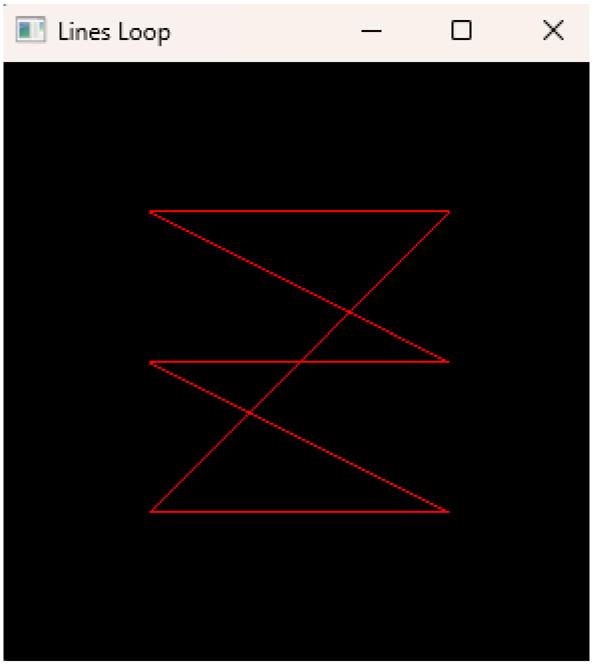
Second line segment: V1 to V2

Third line segment: V2 to V3

Third line segment: V3 to V0



```
glBegin(GL_LINE_LOOP);
glVertex2f(-0.5f, -0.5f);   // V0
glVertex2f(0.5f, -0.5f);    // V1
glVertex2f(-0.5f, 0.0f);     // V2
glVertex2f(0.5f, 0.0f);    // V3
glVertex2f(-0.5f, 0.5f);    // V4
glVertex2f(0.5f, 0.5f);    // V5
glEnd();
```



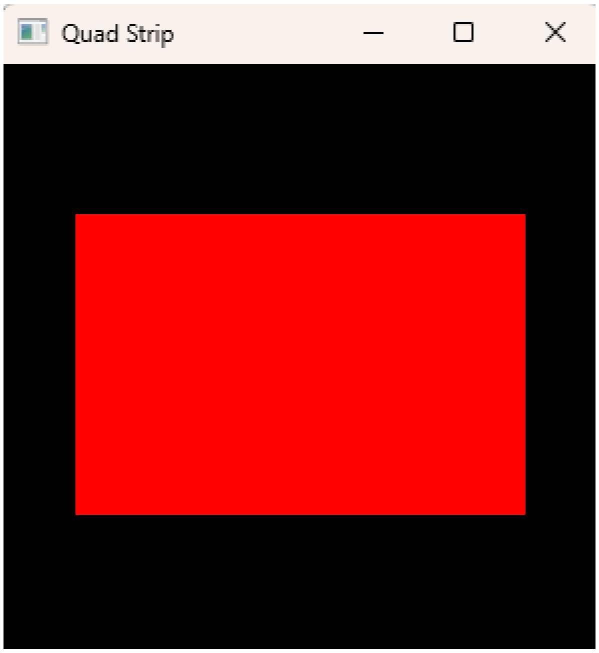
GL_QUAD_STRIP:

It is used to draw quadrilaterals. Points are shared.

Consider a list of vertices V0, V1, V2, V3, V4,V5, V6, V7. Points are not shared.

- First quad: V0, V1, V2, V3
- Second quad: V1, V2,V3, V4
- Third quad: V2, V3,V4, V5
- Fourth quad: V3, V4,V5, V6

```
glBegin(GL_QUAD_STRIP);
glVertex2f(-0.75f, -0.5f);   // V0
glVertex2f(-0.75f, 0.5f);    // V1
glVertex2f(-0.25f, -0.5f);   // V2
glVertex2f(-0.25f, 0.5f);    // V3
glVertex2f(0.25f, -0.5f);    // V4
glVertex2f(0.25f, 0.5f);     // V5
glVertex2f(0.75f, -0.5f);    // V6
glVertex2f(0.75f, 0.5f);     // V7
glEnd();
```

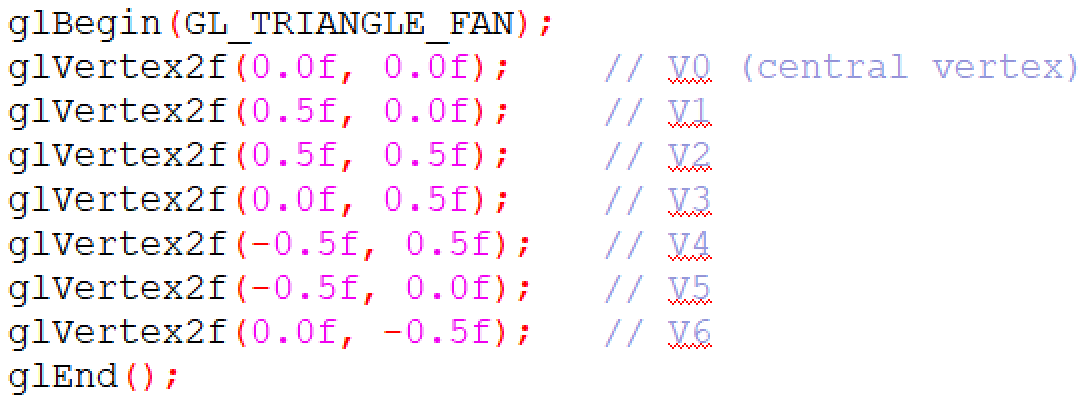


GL_TRIANGLE_FAN:

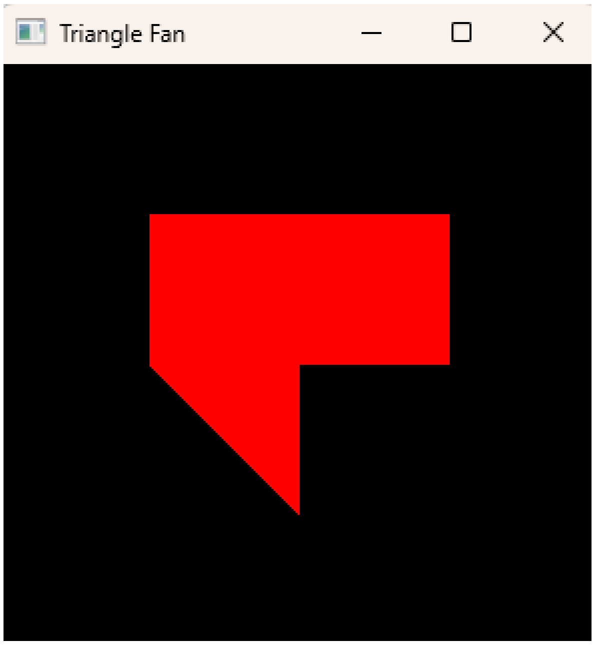
It shares first point as central.

Consider a list of vertices V0, V1, V2, V3, V4.

- First triangle: V0, V1, V2
- Second triangle: V0, V2,V3,
- Third triangle: V0, V3,V4



```
glBegin(GL_TRIANGLE_FAN);
glVertex2f(0.0f, 0.0f);     // V0 (central vertex)
glVertex2f(0.5f, 0.0f);     // V1
glVertex2f(0.5f, 0.5f);     // V2
glVertex2f(0.0f, 0.5f);     // V3
glVertex2f(-0.5f, 0.5f);    // V4
glVertex2f(-0.5f, 0.0f);    // V5
glVertex2f(0.0f, -0.5f);    // V6
glEnd();
```



Other Drawing Attributes:

- Point size: glPointSize()
- Line width: glLineWidth()
- Dashed or dotted line: glLineStipple()
- Polygon Pattern: glPolygonStipple()

glLineStipple(int factor,pattern)

Example-1:

Pattern: 0x00FF (binary: 0000000011111111)

Factor: 1

Result: Dashed pattern with 8 pixels on and 8 pixels off.

Example-2:

Pattern: 0x00FF (binary: 0000000011111111)

Factor: 2

Result: Dashed pattern with 16 pixels on and 16 pixels off.

Example-3:

Pattern: 0x0F0F (binary: 0000111100001111)

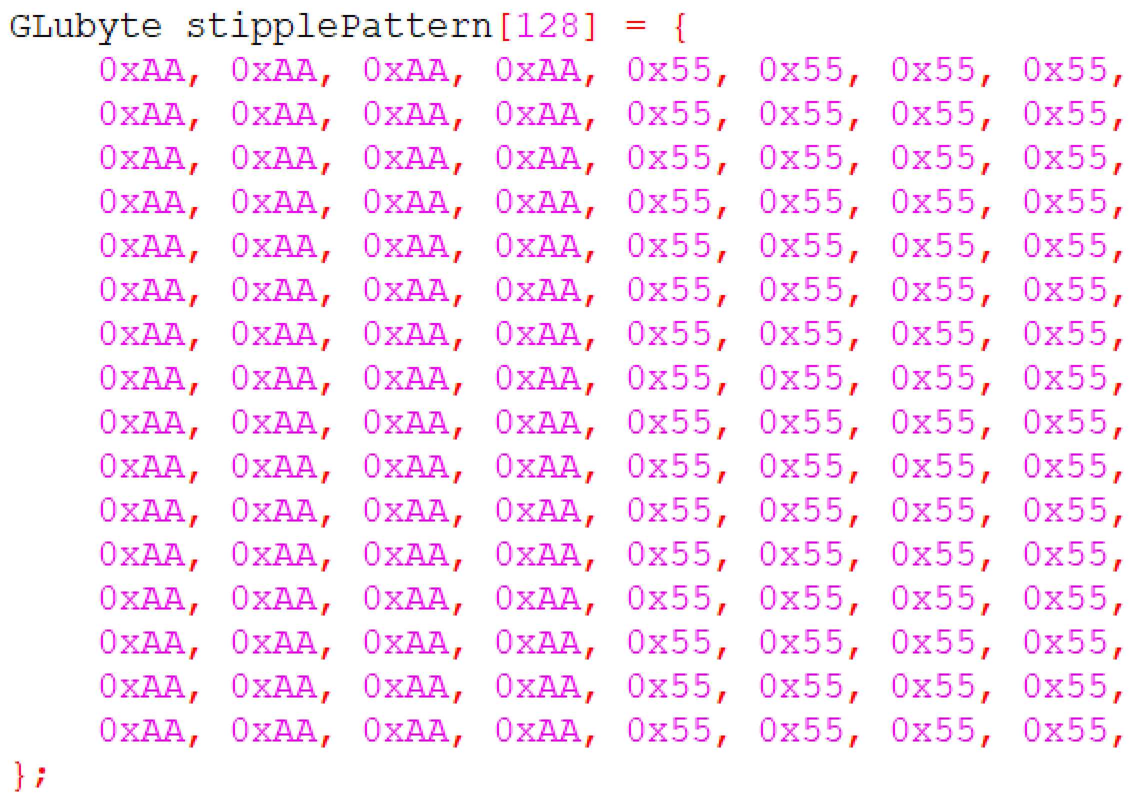
Factor: 1

Result: Alternating dash pattern with 4 pixels on and 4 pixels off.

glPolygonStipple(const GLubyte *mask)

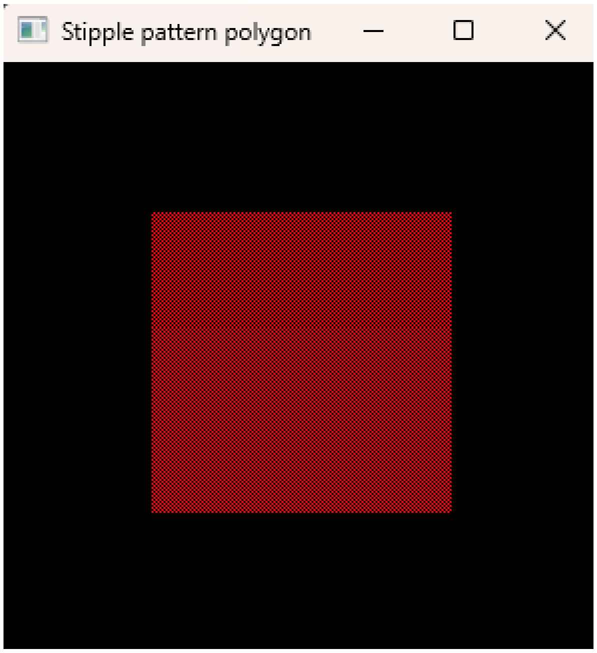
mask: A pointer to a 32x32 array of bits, defining a stipple pattern:

//defining stippling pattern:



```
GLubyte stipplePattern[128] = {
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
};
```

```
glColor3f(1,0,0);
//glPointSize(9);

glEnable(GL_POLYGON_STIPPLE);
glPolygonStipple(stipplePattern);

glBegin(GL_POLYGON);
glVertex2f(-0.5f,-0.5f);
glVertex2f(0.5f,-0.5f);
glVertex2f(0.5f,0.5f);
glVertex2f(-0.5f,0.5f);
glEnd();
```

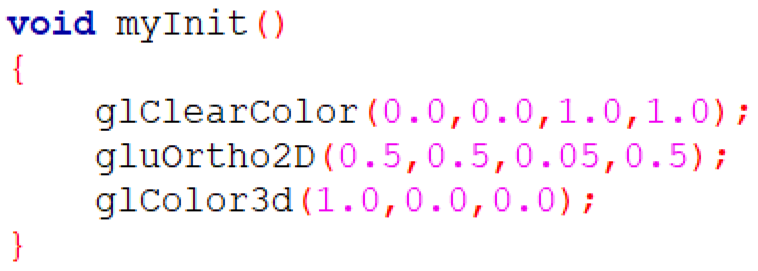


//Disable stipple pattern:

```
glDisable(GL_POLYGON_STIPPLE);
```

2D Viewing:

We can set the area of screen where components must be visible and any component outside that boundary will be clipped.



```
void myInit()
{
    glClearColor(0.0,0.0,1.0,1.0);
    gluOrtho2D(0.5,0.5,0.05,0.5);
    glColor3d(1.0,0.0,0.0);
}
```

```
gluOrtho2D(left, right, bottom, top);
```

3D Viewing:

It contains z-axis as far and near value.

```
gluOrtho2D(left, right, bottom, top, near, far);
```

Matrix Modes:

GL_PROJECTION:

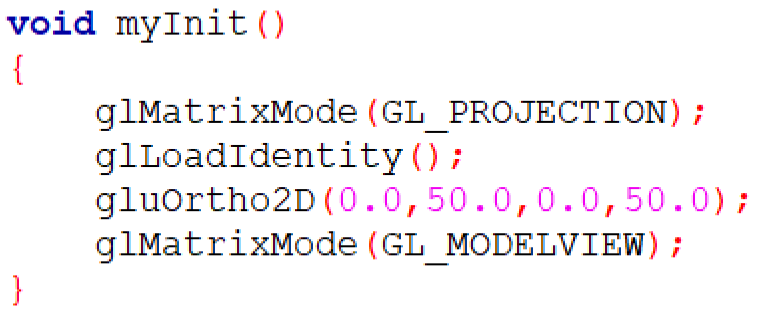
It is enabled before setting orthogonal or frustum.

GL_MODELVIEW:

It is enabled before applying operations on shapes like scaling, rotation, translation.

Resetting all operations:

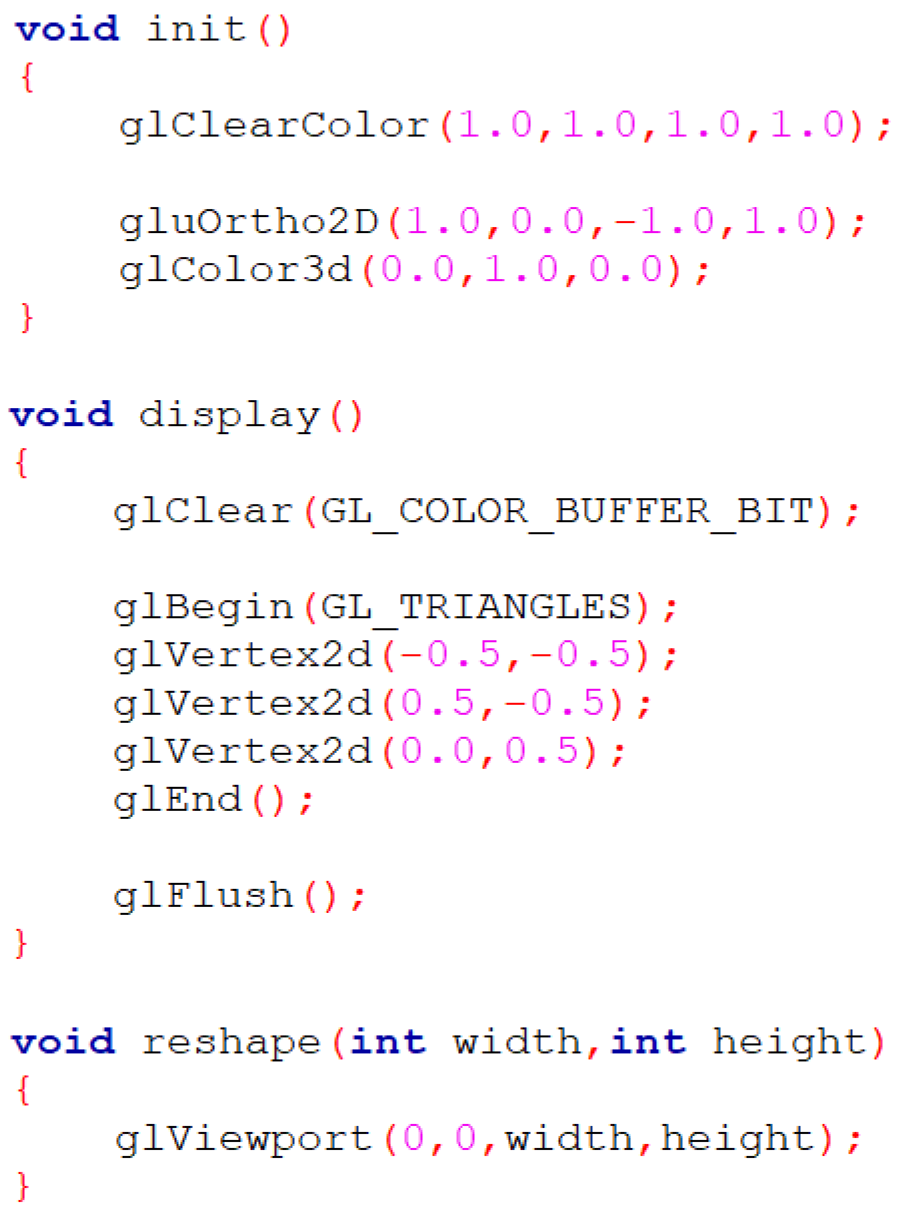
glLoadIdentity() resets all applied operations.



```
void myInit()
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0,50.0,0.0,50.0);
    glMatrixMode(GL_MODELVIEW);
}
```

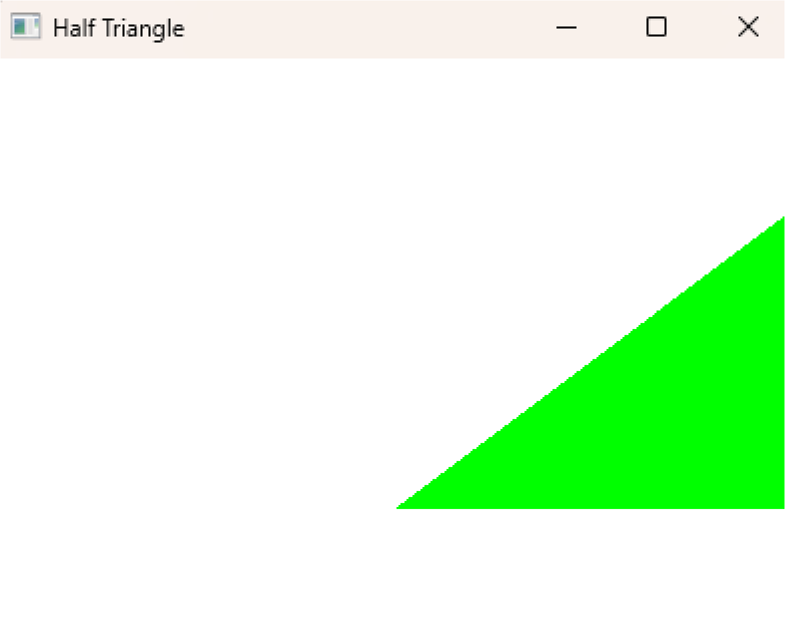
Clipping triangle:

Let say we have drawn complete triangle but only want to see left half. We can use gluOrtho2D() to set the visible area only left most side.



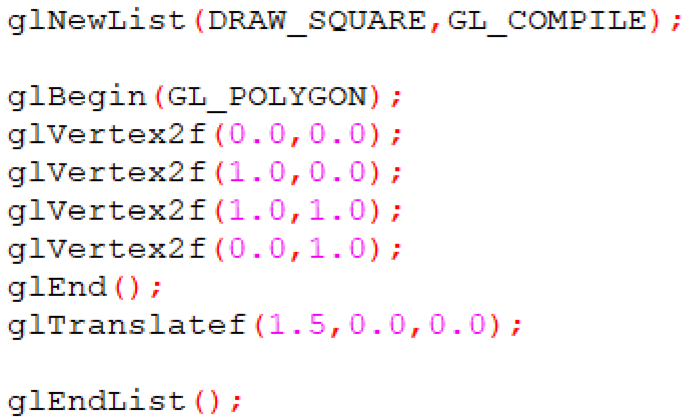
```
void init()
{
    glClearColor(1.0,1.0,1.0,1.0);

    gluOrtho2D(1.0,0.0,-1.0,1.0);
    glColor3d(0.0,1.0,0.0);
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_TRIANGLES);
    glVertex2d(-0.5,-0.5);
    glVertex2d(0.5,-0.5);
    glVertex2d(0.0,0.5);
    glEnd();

    glFlush();
}

void reshape(int width,int height)
{
    glViewport(0,0,width,height);
}
```

```
int main(int argc,char** argv)
{
     glutInit(&argc,argv);
     glutInitWindowSize(400,300);
     glutCreateWindow("Half Triangle");
     glutReshapeFunc(reshape);
     glutDisplayFunc(display);

     init();
     glutMainLoop();
}
```



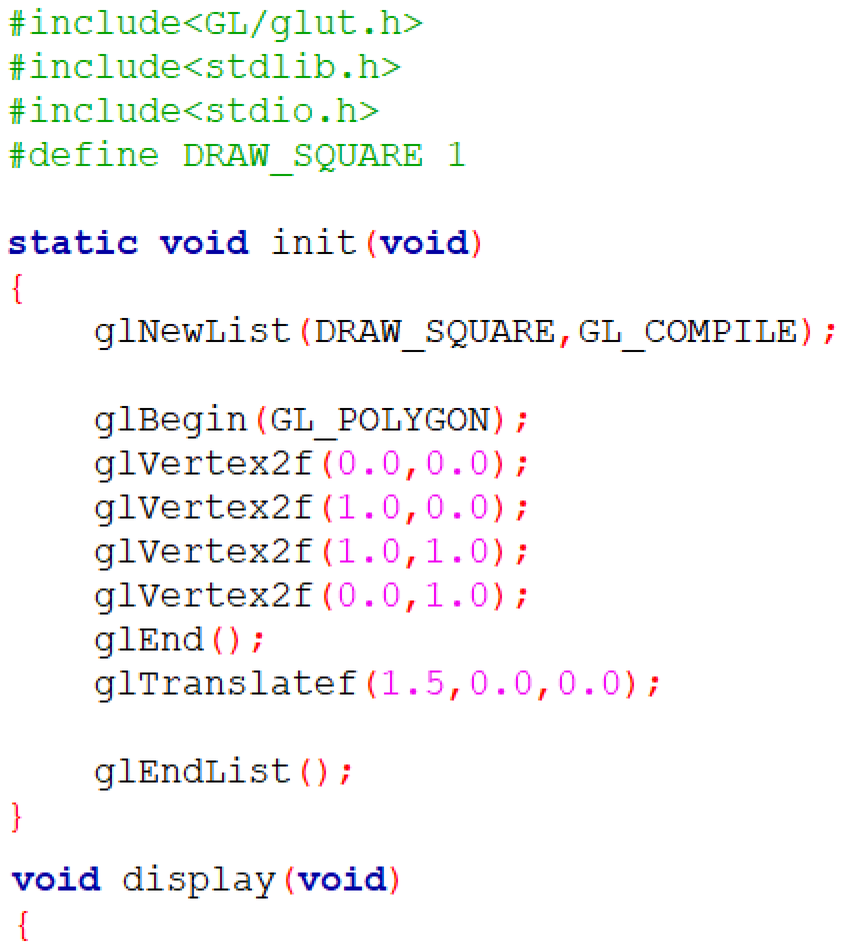
Reusing commands:

We can write a set of commands and store them to disk for later use. This is useful where same kind of operation is required again and again.

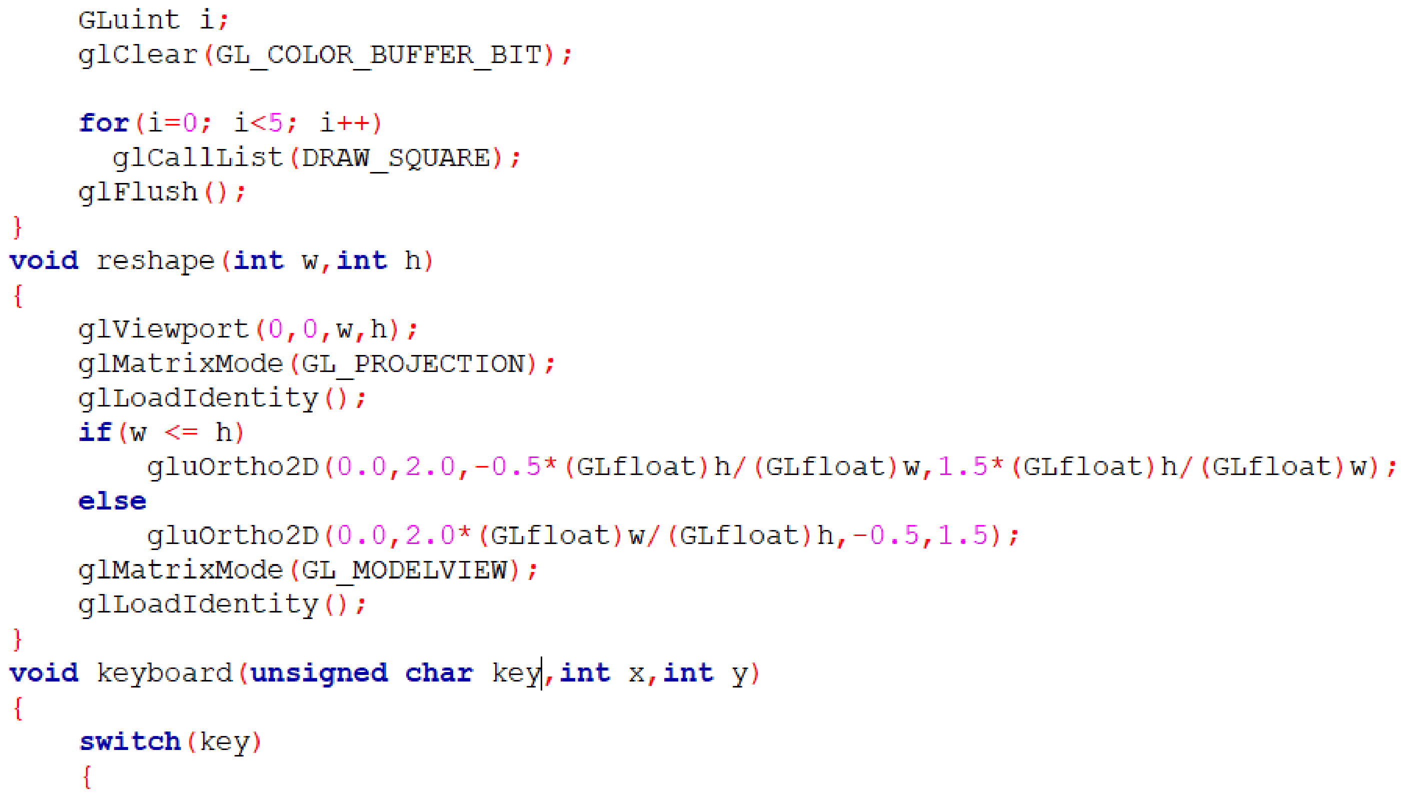


```
glNewList(DRAW_SQUARE,GL_COMPILE);

glBegin(GL_POLYGON);
glVertex2f(0.0,0.0);
glVertex2f(1.0,0.0);
glVertex2f(1.0,1.0);
glVertex2f(0.0,1.0);
glEnd();
glTranslatef(1.5,0.0,0.0);

glEndList();
```

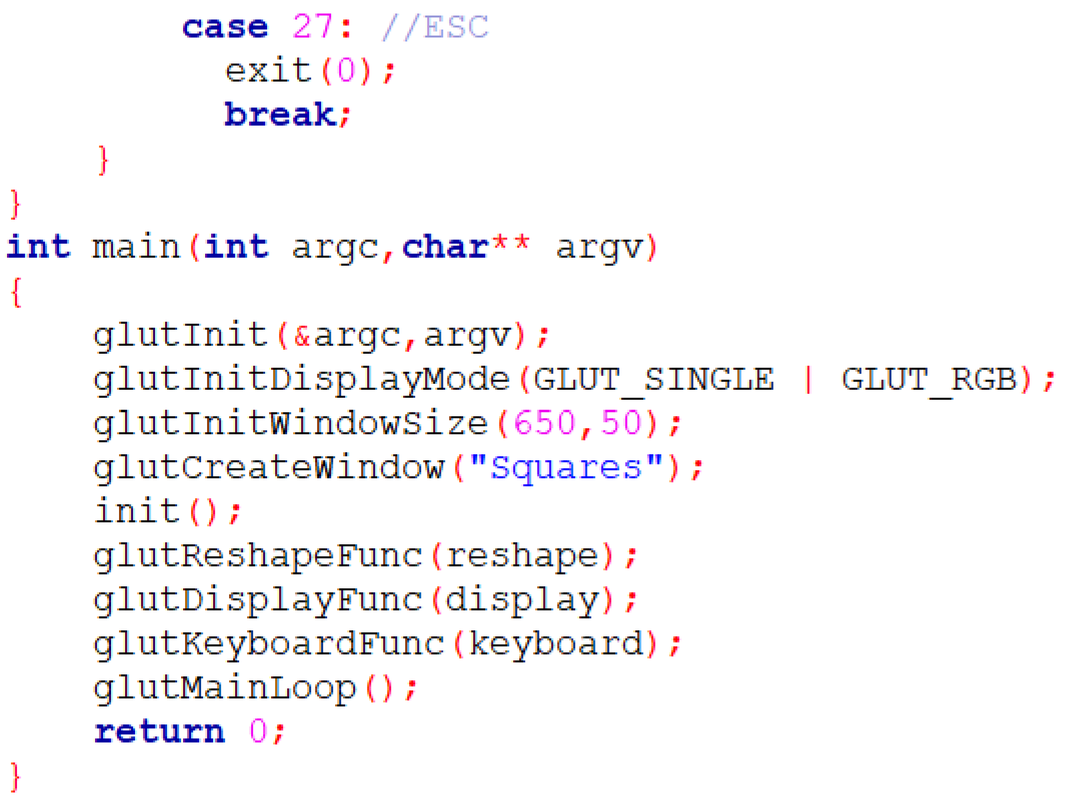
Complete program:



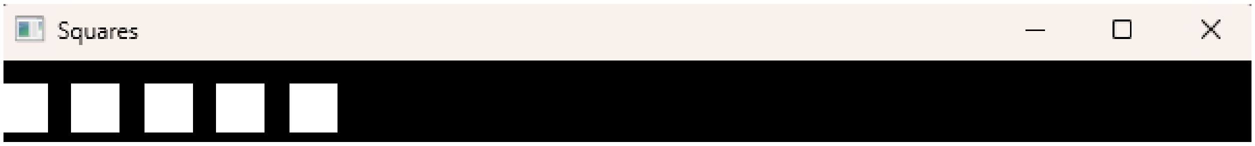
```
#include<GL/glut.h>
#include<stdlib.h>
#include<stdio.h>
#define DRAW_SQUARE 1

static void init(void)
{
    glNewList(DRAW_SQUARE,GL_COMPILE);

    glBegin(GL_POLYGON);
    glVertex2f(0.0,0.0);
    glVertex2f(1.0,0.0);
    glVertex2f(1.0,1.0);
    glVertex2f(0.0,1.0);
    glEnd();
    glTranslatef(1.5,0.0,0.0);

    glEndList();
}
void display(void)
{
```



```
    GLuint i;
    glClear(GL_COLOR_BUFFER_BIT);

    for(i=0; i<5; i++)
      glCallList(DRAW_SQUARE);
    glFlush();
}
void reshape(int w,int h)
{
    glViewport(0,0,w,h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if(w <= h)
        gluOrtho2D(0.0,2.0,-0.5*(GLfloat)h/(GLfloat)w,1.5*(GLfloat)h/(GLfloat)w);
    else
        gluOrtho2D(0.0,2.0*(GLfloat)w/(GLfloat)h,-0.5,1.5);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
void keyboard(unsigned char key,int x,int y)
{
    switch(key)
    {
```



```
        case 27: //ESC
          exit(0);
          break;
      }
}
int main(int argc,char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(650,50);
    glutCreateWindow("Squares");
    init();
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);
    glutMainLoop();
    return 0;
}
```



Lab Task<1>   **Bloom Taxonomy Level:<Applying>**

Draw circle with left click and exit with right click.

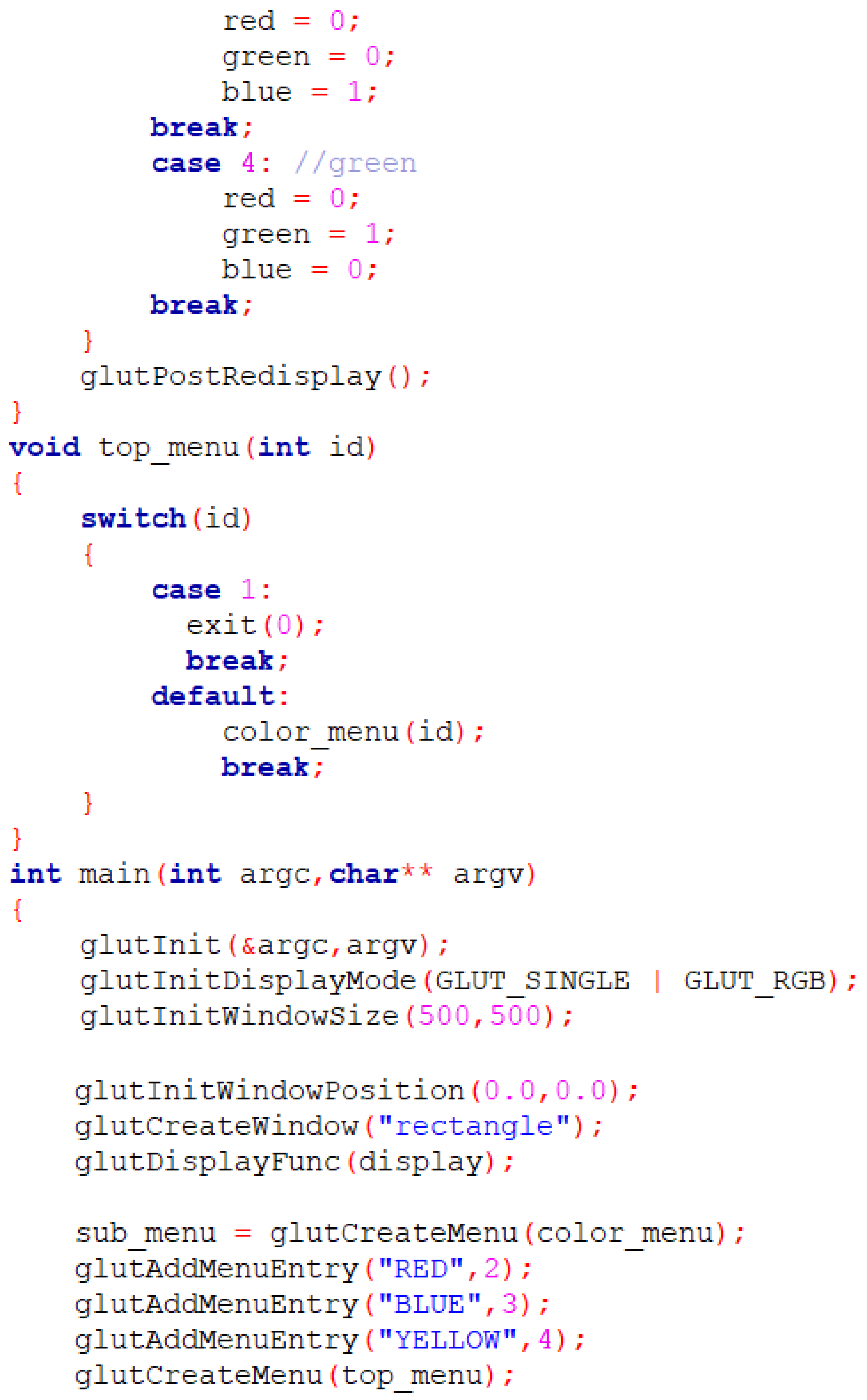
Solution:

```
#include<GL/glut.h>

void drawSquare()
{
    glColor3f(1.0,0.0,0.0);
    glBegin(GL_POLYGON);
    glVertex2f(0.25,0.25);
    glVertex2f(0.25,0.75);
    glVertex2f(0.75,0.75);
    glVertex2f(0.75,0.25);
    glEnd();
    glFlush();
}
void mouse(int button,int state,int x,int y)
{
    if(button == GLUT_LEFT_BUTTON && state ==GLUT_DOWN)
     drawSquare();
    if(button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
     exit(0);
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}

int main(int argc,char** argv)
{
    glutInit(&argc,argv);
    glutInitWindowSize(500,500);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow("square");
    glutMouseFunc(mouse);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Menu:

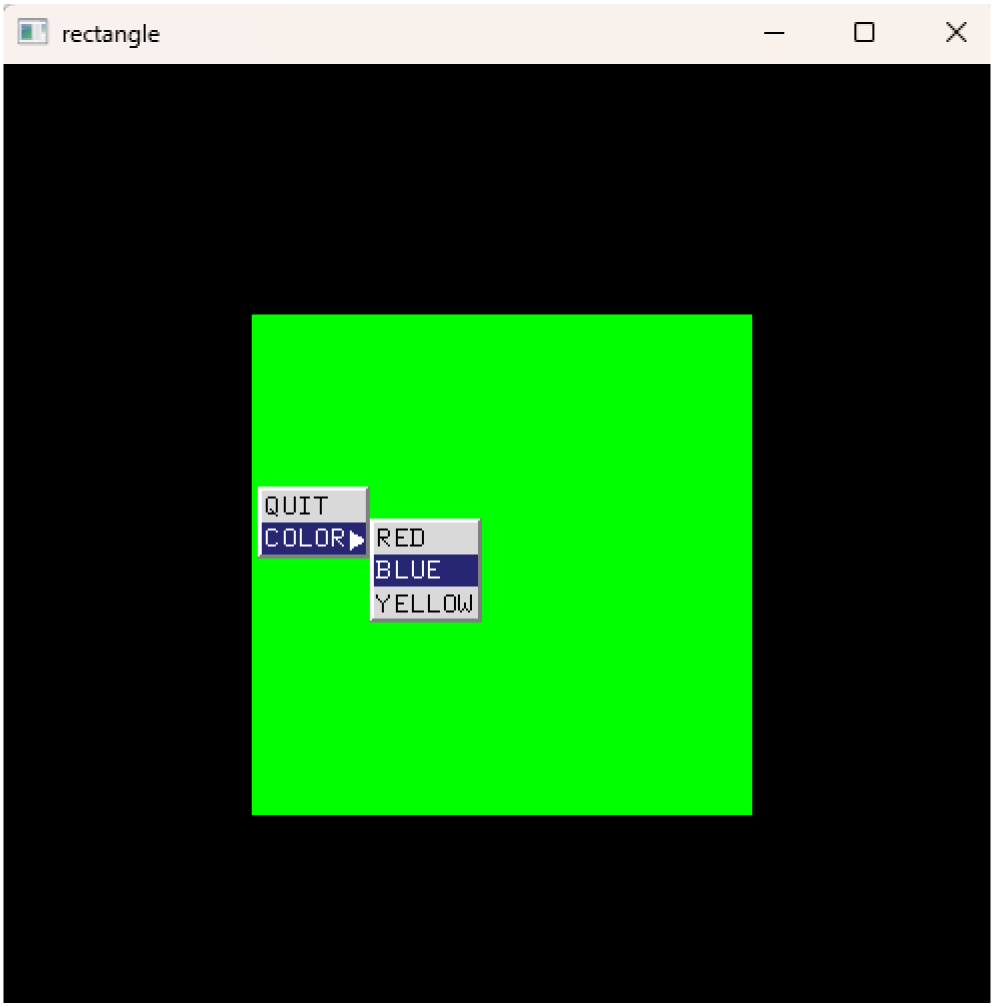
We can also create menu to show a list of options. glutCreateMenu(menuFunc), glutAddMenuEntry(“RED”,2), glutAddSubMenu(“COLOR”,menu);

```
#include<GL/glut.h>
#include<GL/gl.h>

int red=0,green=1,blue=0;

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(red,green,blue);
    glBegin(GL_POLYGON);
    glVertex2f(-0.5,-0.5);
    glVertex2f(-0.5,0.5);
    glVertex2f(0.5,0.5);
    glVertex2f(0.5,-0.5);
    glEnd();
    glFlush();
}
int sub_menu;
void color_menu(int id)
{
    switch(id)
    {
        case 2: //red
            red = 1;
            green = 0;
            blue = 0;
        break;
        case 3: //blue
```



```
            red = 0;
            green = 0;
            blue = 1;
        break;
        case 4: //green
            red = 0;
            green = 1;
            blue = 0;
        break;
    }
    glutPostRedisplay();
}
void top_menu(int id)
{
    switch(id)
    {
        case 1:
          exit(0);
          break;
        default:
            color_menu(id);
            break;
    }
}
int main(int argc,char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500,500);

    glutInitWindowPosition(0.0,0.0);
    glutCreateWindow("rectangle");
    glutDisplayFunc(display);

    sub_menu = glutCreateMenu(color_menu);
    glutAddMenuEntry("RED",2);
    glutAddMenuEntry("BLUE",3);
    glutAddMenuEntry("YELLOW",4);
    glutCreateMenu(top_menu);
```

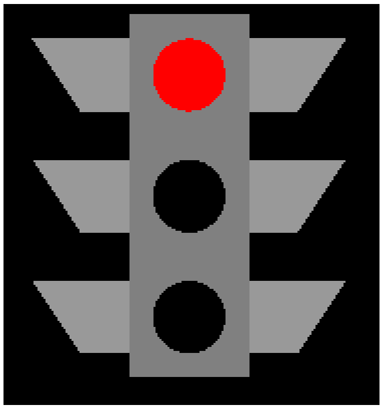
```
glutAddMenuEntry("QUIT",1);
glutAddSubMenu("COLOR",sub_menu);
glutAttachMenu(GLUT_RIGHT_BUTTON);
glutMainLoop();
return 0;

}
```



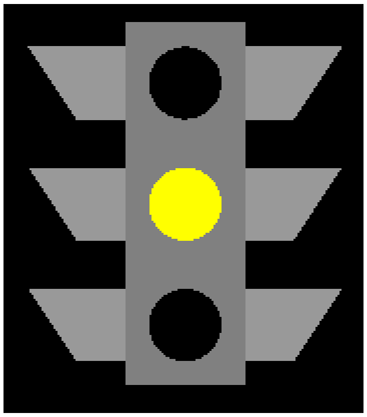
Author: Mr. Aoun Haider

Lab Task<2> **Bloom Taxonomy Level:<Applying>**

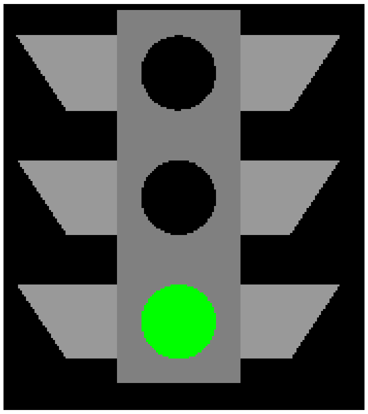
Create traffic signal animation using quadrilaterals and circles with each light that should turn on after 1 second. At start, red light will be turned on, then after 1 second (1000ms), yellow light will be turned on and then green light after 1 second.



0ms



1000ms



2000s

Hint: Use below method for timer functionality, it will invoke the callback function after provided interval.

```
glutTimerFunc(timeInMilliSeconds, callback, callbackParams);
```

To create quad and circle, use the below methods:

```
    //Quad code
    glBegin(GL_QUADS);
    glVertex3f(0.0f,0.6f,0.0f);
    glVertex3f(0.2f,0.6f,0.0f);
    glVertex3f(0.4f,0.9f,0.0f);
    glVertex3f(0.0f,0.9f,0.0f);
    glEnd();
    //Circle code
    glutSolidSphere(0.3,200,2);
```

Author: Mr. Aoun Haider

Solutions:

```
#include<GL/glut.h>

void update(int);
void update1(int);
void update2(int);
void drawScene();
void keyboard(unsigned char key,int x,int y);
void handleResize(int w,int h);

GLfloat a1=0.0,b1=0.0,c1=0.0;
GLfloat a2=0.0,b2=0.0,c2=0.0;
GLfloat a3=0.0,b3=0.0,c3=0.0;



int main(int argc,char** argv)
{
   glutInit(&argc,argv);
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB |
GLUT_DEPTH);
   glutInitWindowSize(500,500);
   glutCreateWindow("Traffic signal");
   glutDisplayFunc(drawScene);
```

```
    glutReshapeFunc(handleResize);
    glutTimerFunc(1000,update,0);
    glutKeyboardFunc(keyboard);
    glutKeyboardFunc(keyboard);
    glutMainLoop();

    return 0;
}
void handleResize(int w,int h)
{
    glViewport(0,0,w,h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0,(double)w/(double)h,1.0,200.0);
}
void drawScene()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    glTranslatef(0.0f,0.0f,-5.0f);
    glPushMatrix();
```

```
glTranslatef(0.0f,-1.0f,0.0f);
glColor3f(0.5f,0.5f,0.5f);
//Middle Rectangle
glBegin(GL_QUADS);
glVertex3f(-0.5f,0.5f,0.0f);
glVertex3f(0.0f,0.5f,0.0f);
glVertex3f(0.0f,2.0f,0.0f);
glVertex3f(-0.5f,2.0f,0.0f);
glEnd();

glColor3f(0.6f,0.6f,0.6f);
//Top-Right Quad
glBegin(GL_QUADS);
glVertex3f(0.0f,1.6f,0.0f);
glVertex3f(0.2f,1.6f,0.0f);
glVertex3f(0.4f,1.9f,0.0f);
glVertex3f(0.0f,1.9f,0.0f);
glEnd();

//Middle right Quad
glBegin(GL_QUADS);
glVertex3f(0.0f,1.1f,0.0f);
```

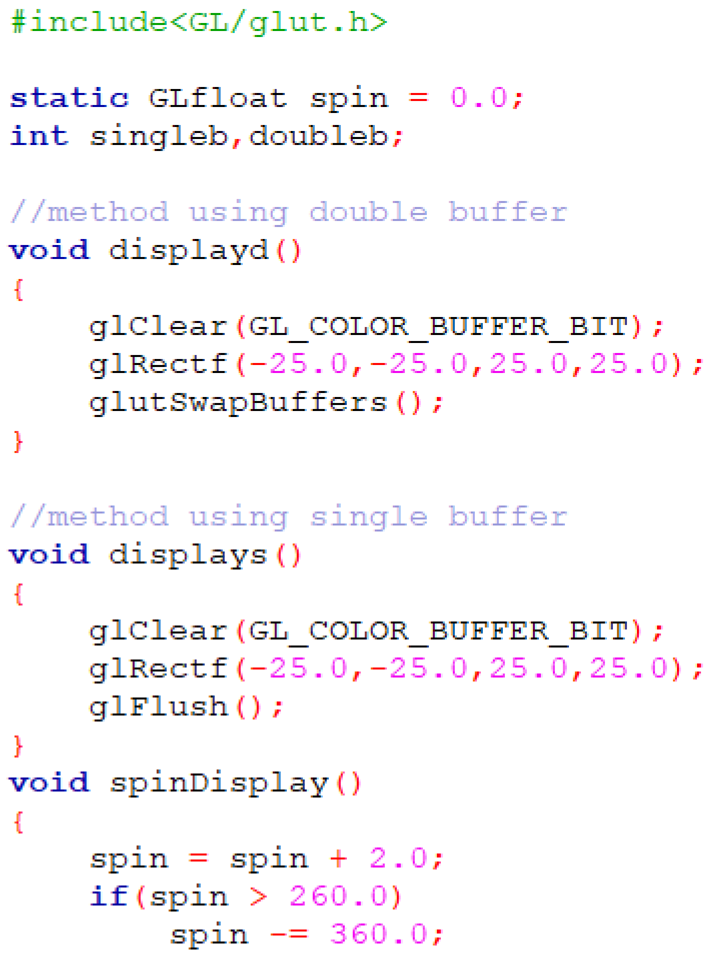
```
glVertex3f(0.2f,1.1f,0.0f);
glVertex3f(0.4f,1.4f,0.0f);
glVertex3f(0.0f,1.4f,0.0f);
glEnd();

//Bottom right Quad
glBegin(GL_QUADS);
glVertex3f(0.0f,0.6f,0.0f);
glVertex3f(0.2f,0.6f,0.0f);
glVertex3f(0.4f,0.9f,0.0f);
glVertex3f(0.0f,0.9f,0.0f);
glEnd();

//Top Left Quad
glBegin(GL_QUADS);
glVertex3f(-0.5f,1.6f,0.0f);
glVertex3f(-0.7f,1.6f,0.0f);
glVertex3f(-0.9f,1.9f,0.0f);
glVertex3f(-0.5f,1.9f,0.0f);
glEnd();

//Middle left Quad
glBegin(GL_QUADS);
```

```
glVertex3f(-0.5f,1.1f,0.0f);
glVertex3f(-0.7f,1.1f,0.0f);
glVertex3f(-0.9f,1.4f,0.0f);
glVertex3f(-0.5f,1.4f,0.0f);
glEnd();

//Bottom left Quad
glBegin(GL_QUADS);
glVertex3f(-0.5f,0.6f,0.0f);
glVertex3f(-0.7f,0.6f,0.0f);
glVertex3f(-0.9f,0.9f,0.0f);
glVertex3f(-0.5f,0.9f,0.0f);
glEnd();

//Red Light
glPushMatrix();
glTranslatef(-0.5f,2.5f,-5.0f);
glColor3f(a1,b1,c1);
glutSolidSphere(0.3,200,2);
glPopMatrix();
glPopMatrix();

//Yellow Light
```

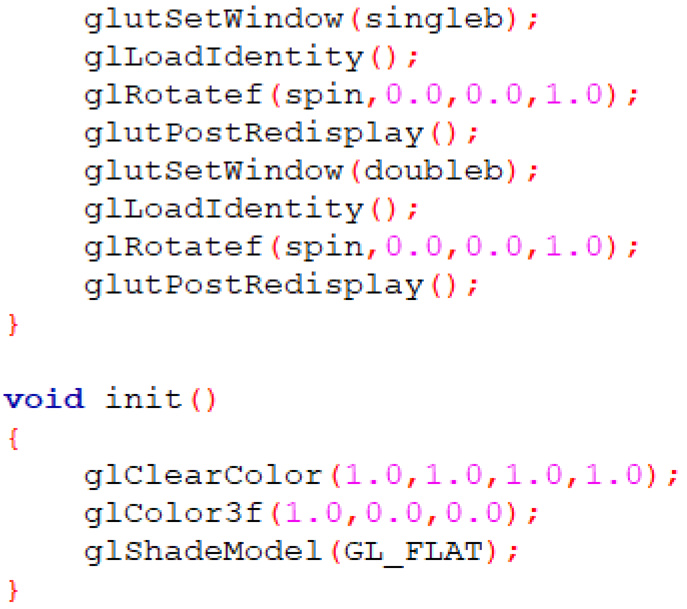
```
    glPushMatrix();
    glTranslatef(-0.5f,0.5f,-5.0f);
    glColor3f(a2,b2,c2);
    glutSolidSphere(0.3,200,2);
    glPopMatrix();

    //Green Light
    glPushMatrix();
    glTranslatef(-0.5f,-0.5f,-5.0f);
    glColor3f(a3,b3,c3);
    glutSolidSphere(0.3,200,2);
    glPopMatrix();
    glPopMatrix();

    glutSwapBuffers();
}
void update(int value)
{
    a1=1.0;
    b1=0.0;
    c1=0.0;
    a2=0.0;
    b2=0.0;
    c2=0.0;
    a3=0.0;
    b3=0.0;
    c3=0.0;
    drawScene();
    glutPostRedisplay();
    glutTimerFunc(1000,update1,0);
}
void update1(int value)
{
    a1=0.0;
    b1=0.0;
    c1=0.0;
    a2=1.0;
    b2=1.0;
    c2=0.0;
    a3=0.0;
    b3=0.0;
    c3=0.0;
    drawScene();
    glutPostRedisplay();
    glutTimerFunc(1000,update2,0);
}
```

```
void update2(int value)
{
    a1=0.0;
    b1=0.0;
    c1=0.0;
    a2=0.0;
    b2=0.0;
    c2=0.0;
    a3=0.0;
    b3=1.0;
    c3=0.0;
    drawScene();
    glutPostRedisplay();
    glutTimerFunc(1000,update,0);
}
void keyboard(unsigned char key,int x,int y)
{
    switch(key)
    {
        case 27:
          exit(0);
        break;
    }}
```

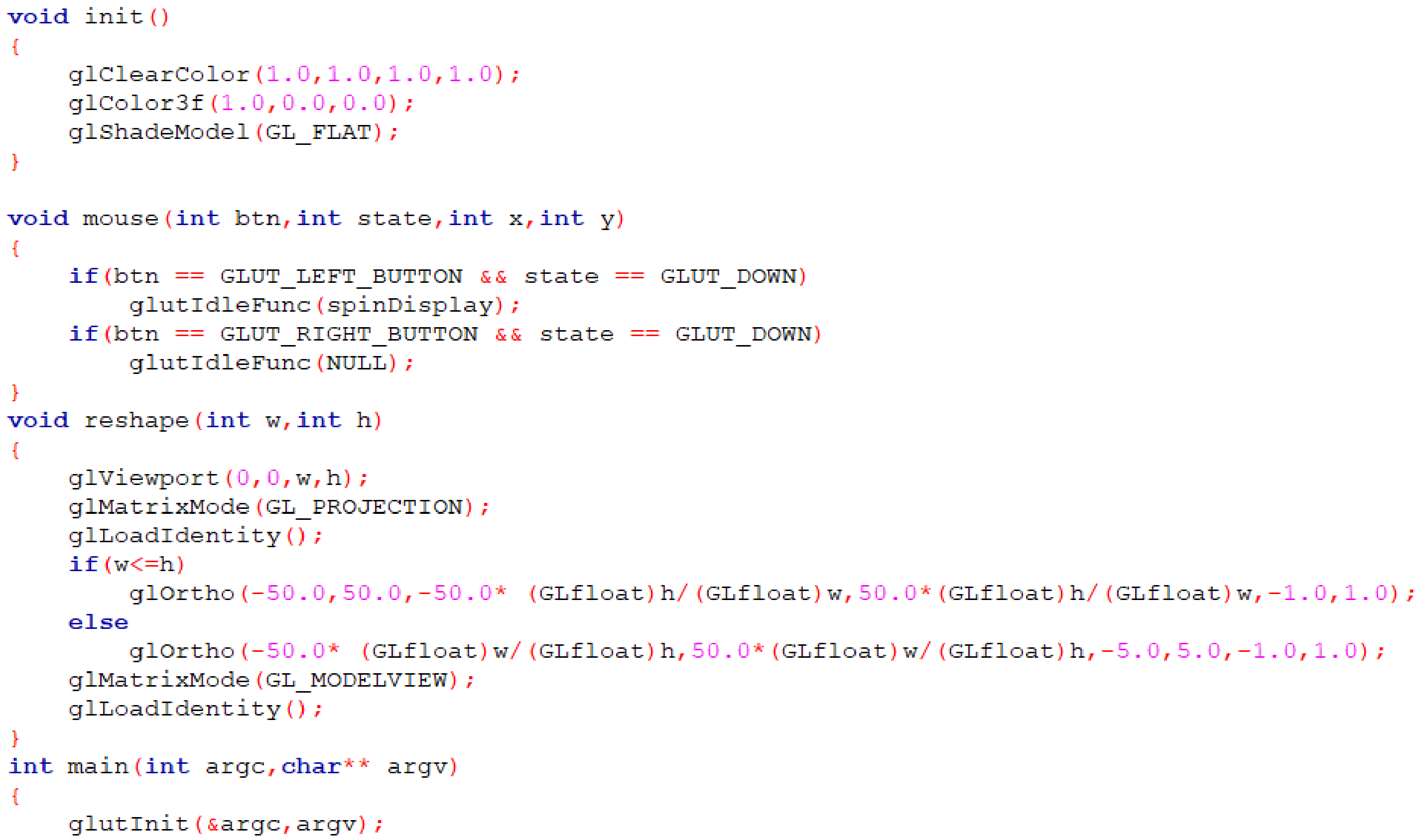
Creating two windows of square rotation with left click:



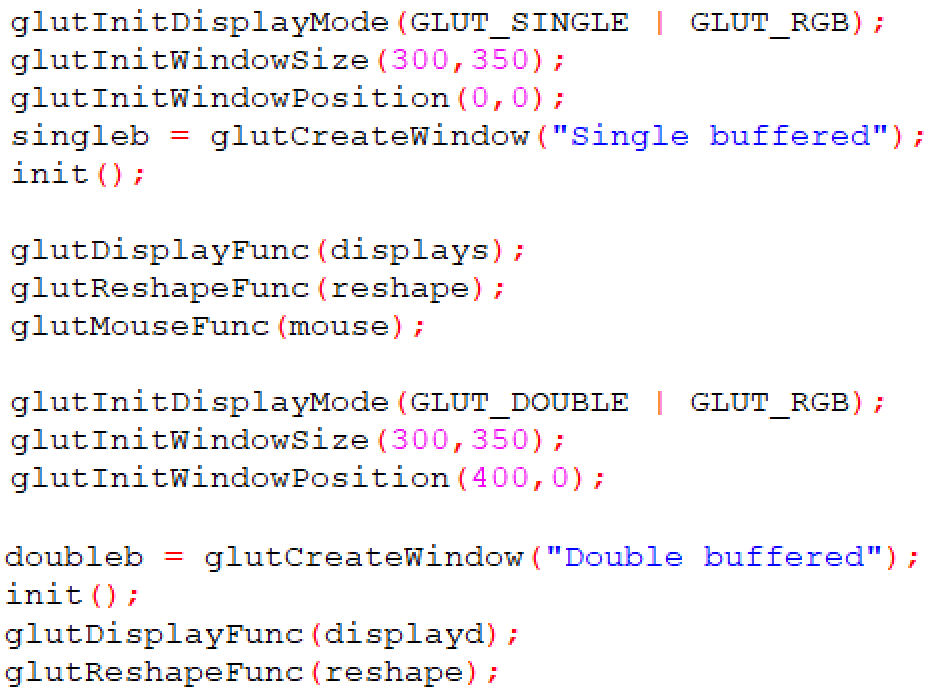
```
#include<GL/glut.h>

static GLfloat spin = 0.0;
int singleb,doubleb;

//method using double buffer
void displayd()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glRectf(-25.0,-25.0,25.0,25.0);
    glutSwapBuffers();
}

//method using single buffer
void displays()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glRectf(-25.0,-25.0,25.0,25.0);
    glFlush();
}
void spinDisplay()
{
    spin = spin + 2.0;
    if(spin > 260.0)
        spin -= 360.0;
```



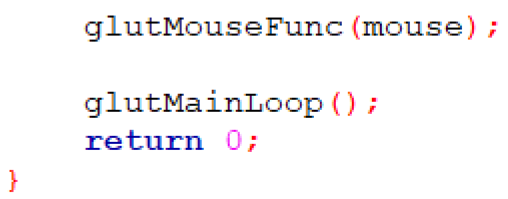
```
    glutSetWindow(singleb);
    glLoadIdentity();
    glRotatef(spin,0.0,0.0,1.0);
    glutPostRedisplay();
    glutSetWindow(doubleb);
    glLoadIdentity();
    glRotatef(spin,0.0,0.0,1.0);
    glutPostRedisplay();
}

void init()
{
    glClearColor(1.0,1.0,1.0,1.0);
    glColor3f(1.0,0.0,0.0);
    glShadeModel(GL_FLAT);
}
```



```
void init()
{
    glClearColor(1.0,1.0,1.0,1.0);
    glColor3f(1.0,0.0,0.0);
    glShadeModel(GL_FLAT);
}

void mouse(int btn,int state,int x,int y)
{
    if(btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
        glutIdleFunc(spinDisplay);
    if(btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
        glutIdleFunc(NULL);
}
void reshape(int w,int h)
{
    glViewport(0,0,w,h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if(w<=h)
        glOrtho(-50.0,50.0,-50.0* (GLfloat)h/(GLfloat)w,50.0*(GLfloat)h/(GLfloat)w,-1.0,1.0);
    else
        glOrtho(-50.0* (GLfloat)w/(GLfloat)h,50.0*(GLfloat)w/(GLfloat)h,-5.0,5.0,-1.0,1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
int main(int argc,char** argv)
{
    glutInit(&argc,argv);
```



```
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(300,350);
glutInitWindowPosition(0,0);
singleb = glutCreateWindow("Single buffered");
init();

glutDisplayFunc(displays);
glutReshapeFunc(reshape);
glutMouseFunc(mouse);

glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
glutInitWindowSize(300,350);
glutInitWindowPosition(400,0);

doubleb = glutCreateWindow("Double buffered");
init();
glutDisplayFunc(displayd);
glutReshapeFunc(reshape);
```



```
    glutMouseFunc(mouse);

    glutMainLoop();
    return 0;
}
```

Picking and Memorizing objects:

To make the computer able to recognize object we need to follow steps:

1. Naming the object
2. Memorizing the object
3. Identifying which object were displayed

1. Enter selection mode, put OpenGL into selection mode:

   ```
   glRender(GL_SELECT);
   ```

2. Draw the scene as usual but also give each object an id

   ```
   glInitNames();
   glPushName(0);
   //Set viewing, color or other settings
   before drawing shapes
   glLoadName(1);
   //draw object 1
   glLoadName(2);
   //draw object 2
   glPopMatrix();
   ```

3. Leave selection mode and return to render mode. At this point, information indicating which object where selected i.e. *Memorizing the object*.

   Tell OpenGL about the buffer, because it will use the buffer to memorize the object with the below command:

   ```
   glSelectBuffer(int bufferSize, unsigned int*
   buffer);
   ```

Lighting:

There are different kinds of lights depending upon the requirement.

1. **Ambient Light:** Light affects overall object equally, regardless of their position and orientation.
2. **Diffuse Light:** Light that strikes surfaces and scatters evenly. It depends upon the angle between the light and the surface normal.
3. **Specular Light:** Reflects off shiny surfaces creating highlights. It depends on the angle between the light direction, surface normal and viewer.

Light source properties are:

- **Position:** From were the light originates.
- **Color:** Defined by its component: ambient, diffuse, specular

Material properties are:

- **Ambient:** How the material reflects ambient light.
- **Diffuse:** How the material reflects ambient light.
- **Specular:** How the material reflects ambient light.
- **Shininess:** How the material reflects ambient light.

Setting up lighting in OpenGL:

1. Enable lighting
2. Define lighting properties
3. Define material properties
4. Enable color tracing (optional)
5. Render the scene

Step-1: Enable Lighting

```
glEnable(GL_LIGHTING); //Enable lighting in general

glEnable(GL_LIGHT0); //Enable a specific light source 0
```

Step-2: Define lighting properties

```
GLfloat ambientLight[] = {0.2f,0.2f,0.2f,1.0f};

GLfloat diffuseLight[] = {0.8f,0.8f,0.8f,1.0f};

GLfloat specularLight[] = {1.0f,1.0f,1.0f,1.0f};
GLfloat lightPosition[] = {50.0f,50.0f,50.0f,1.0f};
//positional light

//Apply these properties to GL_LIGHT0

glLightfv(GL_LIGHT0, GL_AMBIENT, ambientLight);

glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuseLight);

glLightfv(GL_LIGHT0, GL_SPECULAR, specularLight);

glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);
```

Step-3: Define material properties

```
GLfloat materialAmbient[] = {0.2f,0.2f,0.2f,1.0f};

GLfloat materialDiffuse[] = {0.8f,0.8f,0.8f,1.0f};

GLfloat materialSpecular[] = {1.0f,1.0f,1.0f,1.0f};

GLfloat materialShininess[] = {50.0f}; //0 to 128

//Apply these properties to material

glMaterialfv(GL_FRONT, GL_AMBIENT, materialAmbient);

glMaterialfv(GL_FRONT, GL_DIFFUSE, materialDiffuse);

glMaterialfv(GL_FRONT, GL_SPECULAR, materialSpecular);

glMaterialfv(GL_FRONT, GL_POSITION, materialShininess);
```

Step-4: Enable color tracing

```
glEnable(GL_COLOR_MATERIAL);

glColorMaterial(GL_FRONT,GL_AMBIENT_AND_DIFFUSE);
```

Step-5: Render the scene

```
void RenderScene(void) {

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glutSwapBuffers();

}
```

Types of light sources:

1. ***Directional light:***
    - Mimics a light source infinitely far away (like the sun).
    - Only direction is considered, not position.

```
GLfloat lightDirection[] = {0.0f,-1.0f,-1.0f,0.0f};
glLightfv(GL_LIGHT0,GL_POSITION,lightDirection);
```

2. ***Point light:***
    - Emits light equally in all direction from a specific point.
    - Position is a point in space.

```
GLfloat lightPosition[] = {1.0f,2.0f,3.0f,1.0f};
glLightfv(GL_LIGHT0,GL_POSITION,lightPosition);
```

3. ***Spot light:***

```
GLfloat spotDirection[] = {0.0f,-1.0f,0.0f};
glLightfv(GL_LIGHT0,GL_SPOT_DIRECTION,spotDirection
);
glLightfv(GL_LIGHT0, GL_SPOT_CUTOFF,45.0f);
```

Note: There are 8 lights allowed using *GL_LIGHT0* to *GL_LIGHT1*

Rubber Band operations:

When we create a shape on black screen, same buffer values are updated at corresponding position where we are drawing shape and if color of shape is red, only the area where shape is displayed, color black which was of previous screen will be replaced by current red while remaining screen will be still black. This is the default behavior of OpenGL API. We can customize it to apply different logical operations like AND, OR, XOR, NAND and so on. For example, we want not to replace the new color with previous one but XOR of both. For example, if previous color at the position where we are drawing triangle is red and new color is white, XORing will result in cyan color instead of white color. We can also apply other logical operations in the same way. XOR can also be used to remove the object from screen because XORing two times will result in same result as was previous:

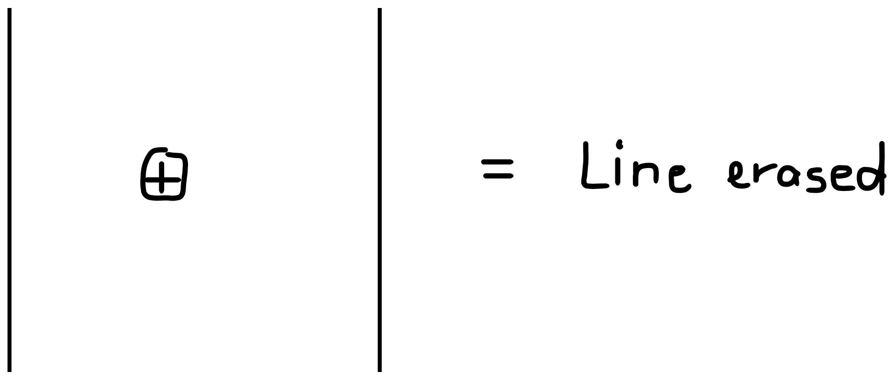
X ^ Y ^ Y = X

We can observe from above equation that XORing two times result in removal of that element. If we XOR same object drawing at same position, it will be removed.

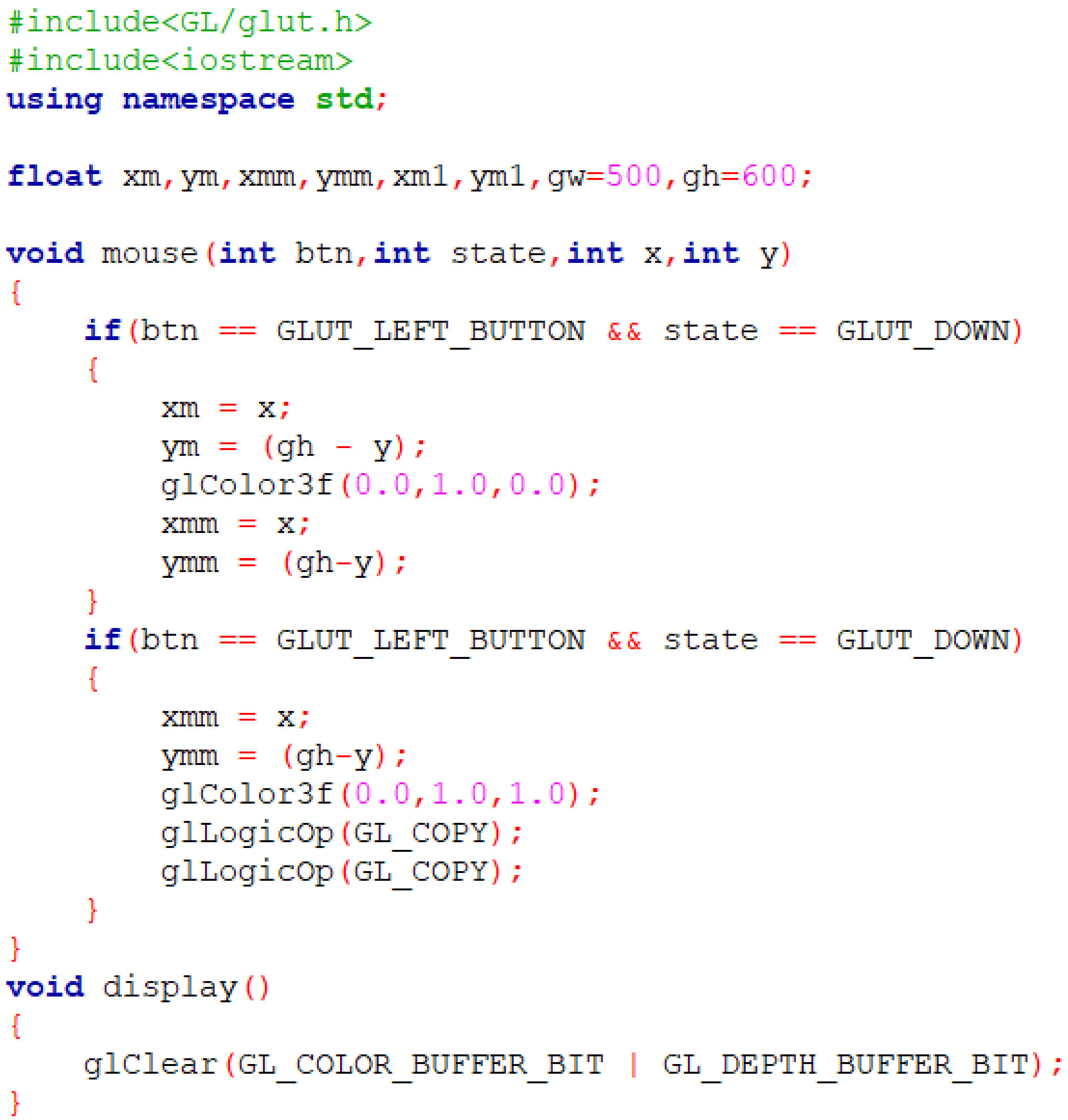
Steps to apply rubber band operations:

1. Enable logical operation:
   glEnable(GL_COLOR_LOGIC_OP);
2. Choose logical operation:
   glLogicOp(GL_COPY); //default
3. Disable logical operation:
   glDisable(GL_COLOR_LOGIC_OP);

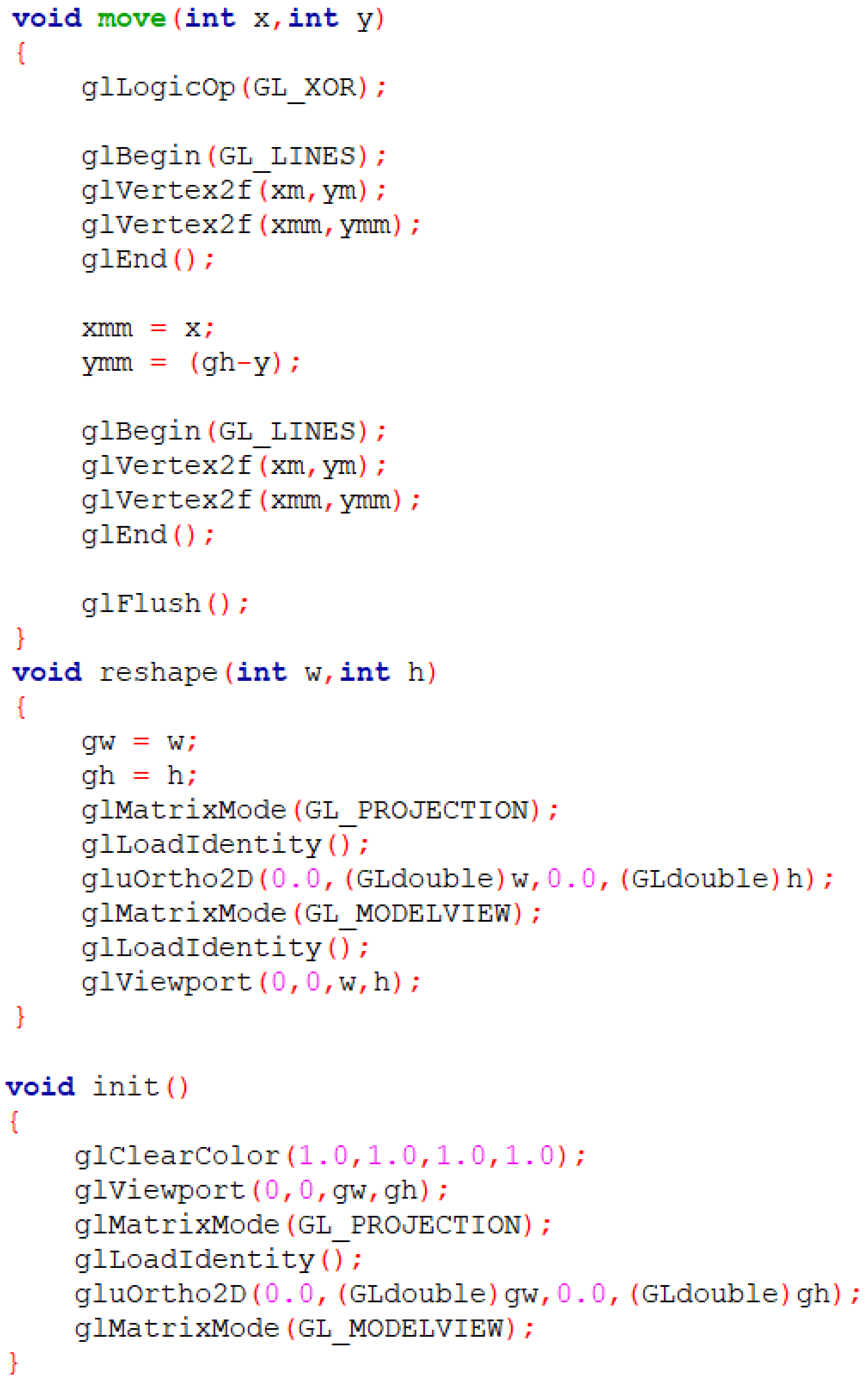
opcode	**Logical operation**
GL_AND	s & d
GL_NAND	~(s & d)
GL_OR	s | d
GL_NOR	~(s | d)
GL_XOR	s ^ d
GL_EQUIV	~(s ^ d)
GL_AND_REVERSE	s & ~d
GL_AND_INVERTED	~s & d
GL_OR_REVERSE	s | ~d
GL_OR_INVERTED	~s | d



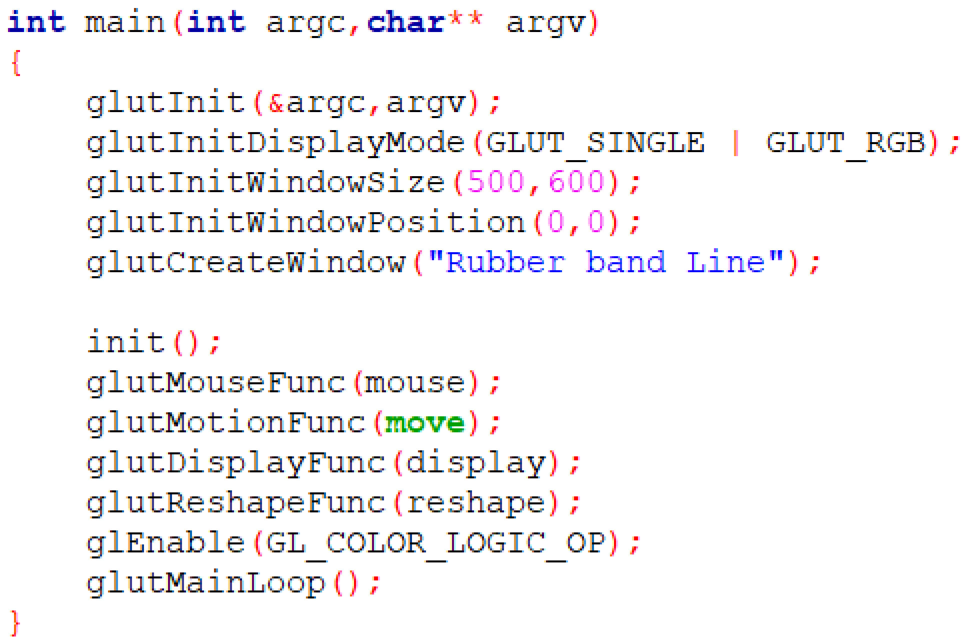
Line rubber banding example:



```
#include<GL/glut.h>
#include<iostream>
using namespace std;

float xm,ym,xmm,ymm,xm1,ym1,gw=500,gh=600;

void mouse(int btn,int state,int x,int y)
{
    if(btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        xm = x;
        ym = (gh - y);
        glColor3f(0.0,1.0,0.0);
        xmm = x;
        ymm = (gh-y);
    }
    if(btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        xmm = x;
        ymm = (gh-y);
        glColor3f(0.0,1.0,1.0);
        glLogicOp(GL_COPY);
        glLogicOp(GL_COPY);
    }
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
}
```



```
void move(int x,int y)
{
    glLogicOp(GL_XOR);

    glBegin(GL_LINES);
    glVertex2f(xm,ym);
    glVertex2f(xmm,ymm);
    glEnd();

    xmm = x;
    ymm = (gh-y);

    glBegin(GL_LINES);
    glVertex2f(xm,ym);
    glVertex2f(xmm,ymm);
    glEnd();

    glFlush();
}
void reshape(int w,int h)
{
    gw = w;
    gh = h;
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0,(GLdouble)w,0.0,(GLdouble)h);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glViewport(0,0,w,h);
}

void init()
{
    glClearColor(1.0,1.0,1.0,1.0);
    glViewport(0,0,gw,gh);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0,(GLdouble)gw,0.0,(GLdouble)gh);
    glMatrixMode(GL_MODELVIEW);
}
```

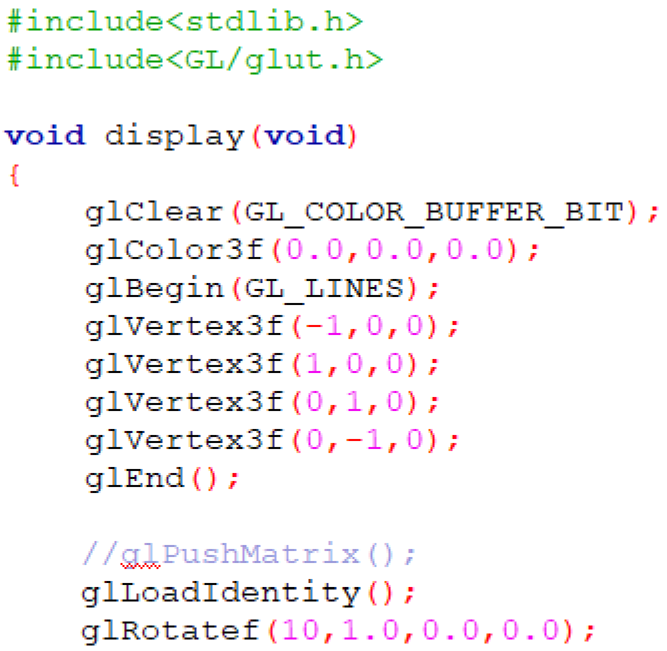


```
int main(int argc,char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500,600);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Rubber band Line");

    init();
    glutMouseFunc(mouse);
    glutMotionFunc(move);
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glEnable(GL_COLOR_LOGIC_OP);
    glutMainLoop();
}
```

Push and Pop Matrix:

When we display some shape, its matrix values are overridden in model matrix, we need to go back to previous version of matrix after drawing the shape otherwise shape will be displayed, and next transformations will be continued with existing one.

Try out the below example with push, pop and load identity method commenting out and see the differences:



```
#include<stdlib.h>
#include<GL/glut.h>

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0,0.0,0.0);
    glBegin(GL_LINES);
    glVertex3f(-1,0,0);
    glVertex3f(1,0,0);
    glVertex3f(0,1,0);
    glVertex3f(0,-1,0);
    glEnd();

    //glPushMatrix();
    glLoadIdentity();
    glRotatef(10,1.0,0.0,0.0);
    glRotatef(10,0.0,1.0,0.0);
    glTranslatef(0,0,0);
    glColor3f(1.0,0.0,0.0);
    glutWireCube(0.2);
    //glPopMatrix();

    //glPushMatrix();
    //glLoadIdentity();
    glRotatef(10,1.0,0.0,0.0);
    glRotatef(10,0.0,1.0,0.0);

    glTranslatef(-.3,0,0);
    glColor3f(0.0,0.0,1.0);
    glutWireCube(0.2);
    //glPopMatrix();
    glFlush();
}
int main(int argc,char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Push & Pop");
    glLineWidth(2);
    glClearColor(1.0,1.0,1.0,0.0);
    glutDisplayFunc(display);
    glutMainLoop();
}
```

Lab Task<3> **Bloom Taxonomy Level:<Applying>**

Create two solid torus and a teapot above both. Also create a single torus with sphere above it. Handle keyboard inputs to accomplish:

if keypress = 27: *escape*

if keypress = 'd': rotate *downward*

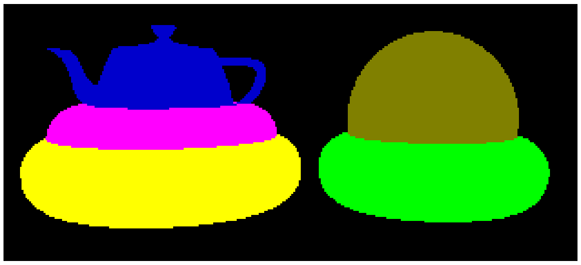
if keypress = 'u': rotate *upward*

if keypress = 'l': rotate *leftward*

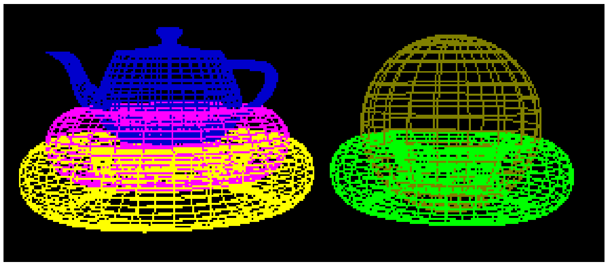
if keypress = 'r': rotate *rightward*

if keypress = 's': show *solid version of shape*

if keypress = 'w': show *wired version of shape*



Solid shapes



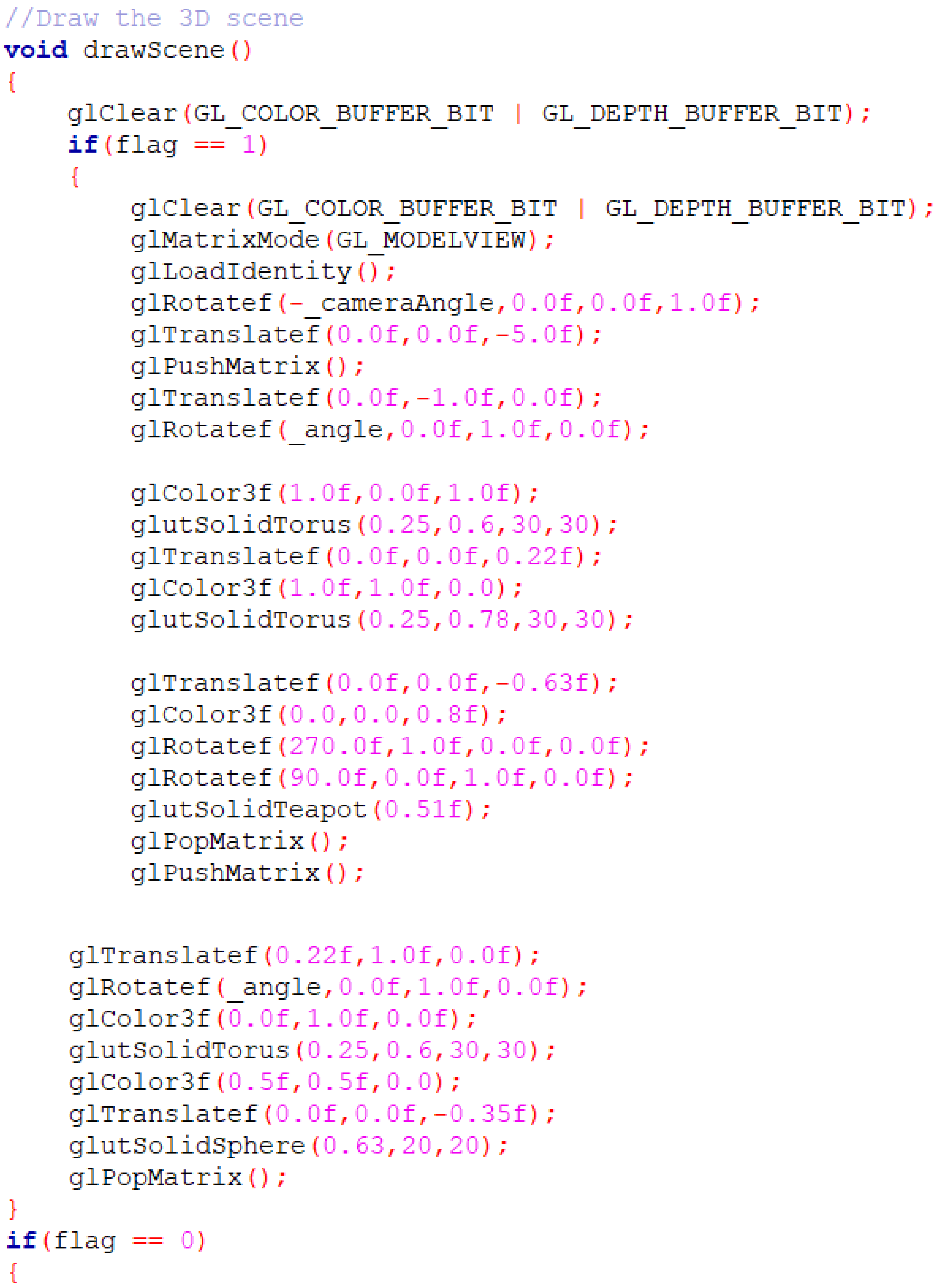
Wired shapes

Author: Mr. Aoun Haider

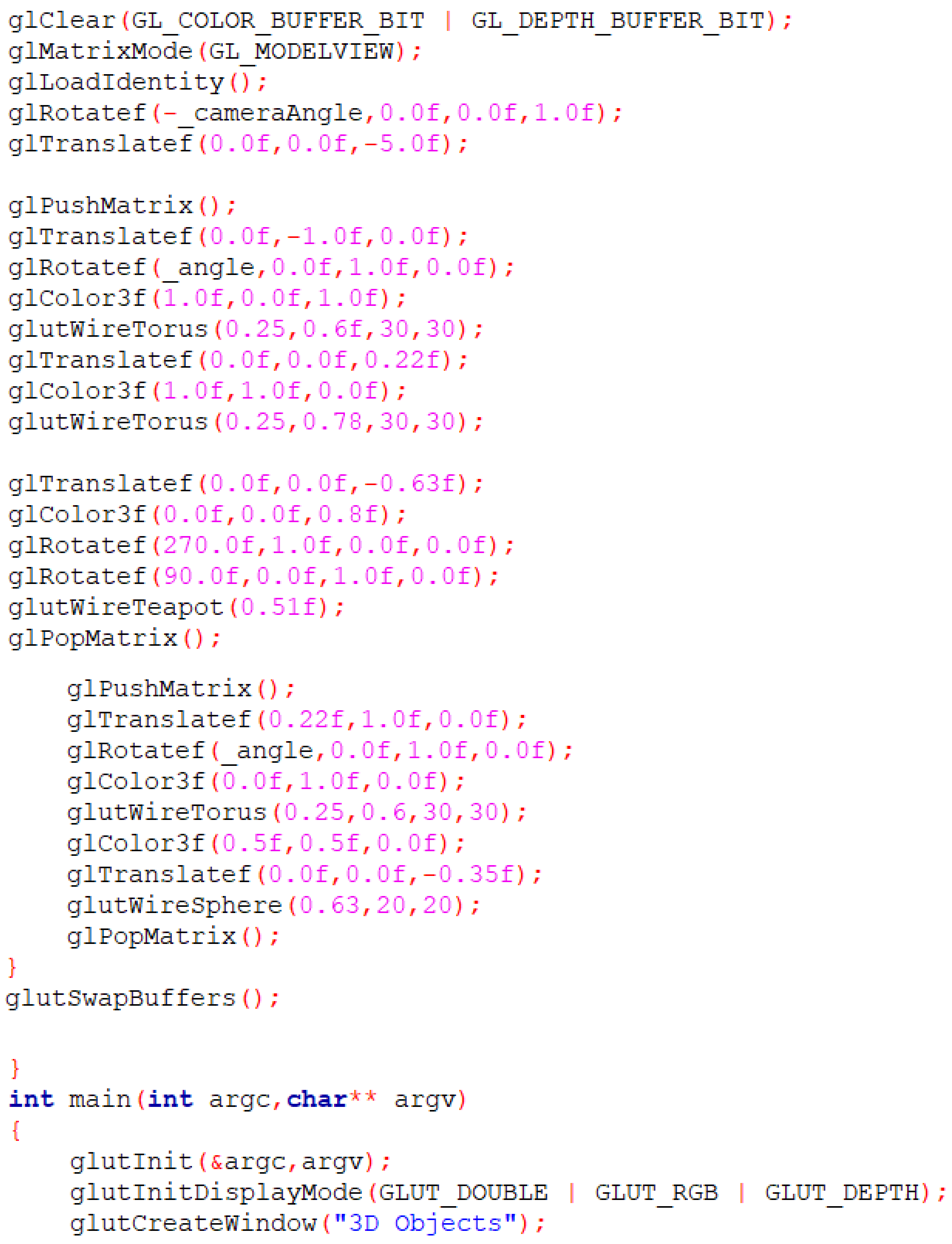
Solutions:

```
#include<GL/glut.h>
#include<stdio.h>
#include<iostream>
using namespace std;

float _angle = 30.0f;
float _cameraAngle = 90.0f;
int flag = 1;
void rotatedown()
{
    _angle += 2.0f;
    if(_angle > 360)
        _angle -= 360;
    glutPostRedisplay();
}
void rotateup()
{
    _angle -= 2.0f;
    if(_angle > 360)
        _angle -= 360;
    glutPostRedisplay();
}
void rotateleft()
{
    _cameraAngle -= 2.0f;
    if(_cameraAngle > 360)
        _cameraAngle -= 360;

    glutPostRedisplay();

}
void rotateright()
{
    _cameraAngle += 2.0f;
    if(_cameraAngle > 360)
        _cameraAngle -= 360;
    glutPostRedisplay();
}
```

```
//Called when a key is pressed
void handleKeyPress(unsigned char key,int x,int y)
{
    switch(key)
    {
        case 27: //ESC key
            exit(0);
        break;
        case 'd': rotatedown();
        break;
        case 'u': rotateup();
        break;
        case 'l': rotateleft();
        break;
        case 'r': rotateright();
        break;
        case 's': flag = 1;

        printf("\n coming in s");
        glutPostRedisplay();
        break;
        case 'w': flag = 0;
        printf("\n coming in w");
        glutPostRedisplay();
        break;
    }
}

void initRendering()
{

    glEnable(GL_DEPTH_TEST);
    glEnable(GL_COLOR_MATERIAL);
    glClearColor(0.0f,0.0f,0.0f,0.0f);
    //change the background to black
}
void handleResize(int w,int h)
{
    glViewport(0,0,w,h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0,(double)w/(double)h,1.0,200.0);
}
```



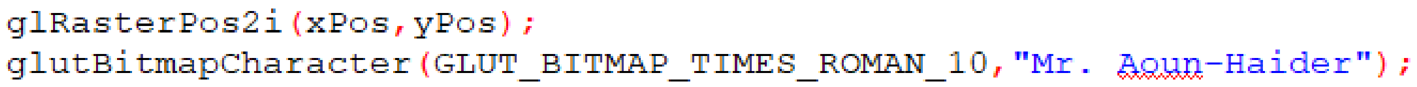
```
//Draw the 3D scene
void drawScene()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    if(flag == 1)
    {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glRotatef(-_cameraAngle,0.0f,0.0f,1.0f);
        glTranslatef(0.0f,0.0f,-5.0f);
        glPushMatrix();
        glTranslatef(0.0f,-1.0f,0.0f);
        glRotatef(_angle,0.0f,1.0f,0.0f);

        glColor3f(1.0f,0.0f,1.0f);
        glutSolidTorus(0.25,0.6,30,30);
        glTranslatef(0.0f,0.0f,0.22f);
        glColor3f(1.0f,1.0f,0.0);
        glutSolidTorus(0.25,0.78,30,30);

        glTranslatef(0.0f,0.0f,-0.63f);
        glColor3f(0.0,0.0,0.8f);
        glRotatef(270.0f,1.0f,0.0f,0.0f);
        glRotatef(90.0f,0.0f,1.0f,0.0f);
        glutSolidTeapot(0.51f);
        glPopMatrix();
        glPushMatrix();


    glTranslatef(0.22f,1.0f,0.0f);
    glRotatef(_angle,0.0f,1.0f,0.0f);
    glColor3f(0.0f,1.0f,0.0f);
    glutSolidTorus(0.25,0.6,30,30);
    glColor3f(0.5f,0.5f,0.0);
    glTranslatef(0.0f,0.0f,-0.35f);
    glutSolidSphere(0.63,20,20);
    glPopMatrix();
}
if(flag == 0)
{
```



```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glRotatef(-_cameraAngle,0.0f,0.0f,1.0f);
glTranslatef(0.0f,0.0f,-5.0f);

glPushMatrix();
glTranslatef(0.0f,-1.0f,0.0f);
glRotatef(_angle,0.0f,1.0f,0.0f);
glColor3f(1.0f,0.0f,1.0f);
glutWireTorus(0.25,0.6f,30,30);
glTranslatef(0.0f,0.0f,0.22f);
glColor3f(1.0f,1.0f,0.0f);
glutWireTorus(0.25,0.78,30,30);

glTranslatef(0.0f,0.0f,-0.63f);
glColor3f(0.0f,0.0f,0.8f);
glRotatef(270.0f,1.0f,0.0f,0.0f);
glRotatef(90.0f,0.0f,1.0f,0.0f);
glutWireTeapot(0.51f);
glPopMatrix();

    glPushMatrix();
    glTranslatef(0.22f,1.0f,0.0f);
    glRotatef(_angle,0.0f,1.0f,0.0f);
    glColor3f(0.0f,1.0f,0.0f);
    glutWireTorus(0.25,0.6,30,30);
    glColor3f(0.5f,0.5f,0.0f);
    glTranslatef(0.0f,0.0f,-0.35f);
    glutWireSphere(0.63,20,20);
    glPopMatrix();
}
glutSwapBuffers();

}
int main(int argc,char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutCreateWindow("3D Objects");
    initRendering();
    //glutFullScreen();
    glutDisplayFunc(drawScene);
    glutKeyboardFunc(handleKeyPress);
    glutReshapeFunc(handleResize);
    glutMainLoop();
}
```

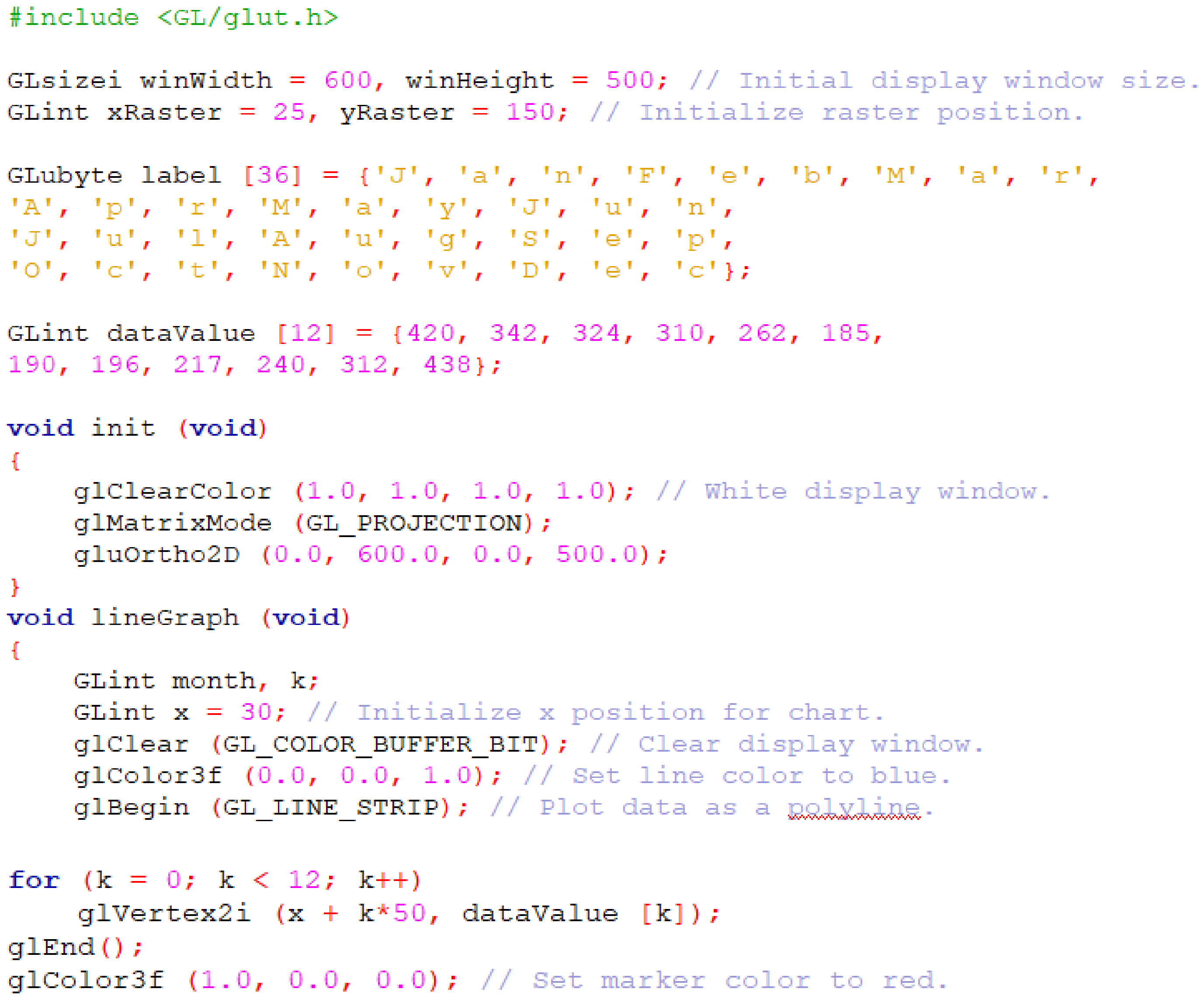
Displaying text:

We can also display text at specified screen position. We need to first set the raster position where character will be placed and then the font style and then the actual text.

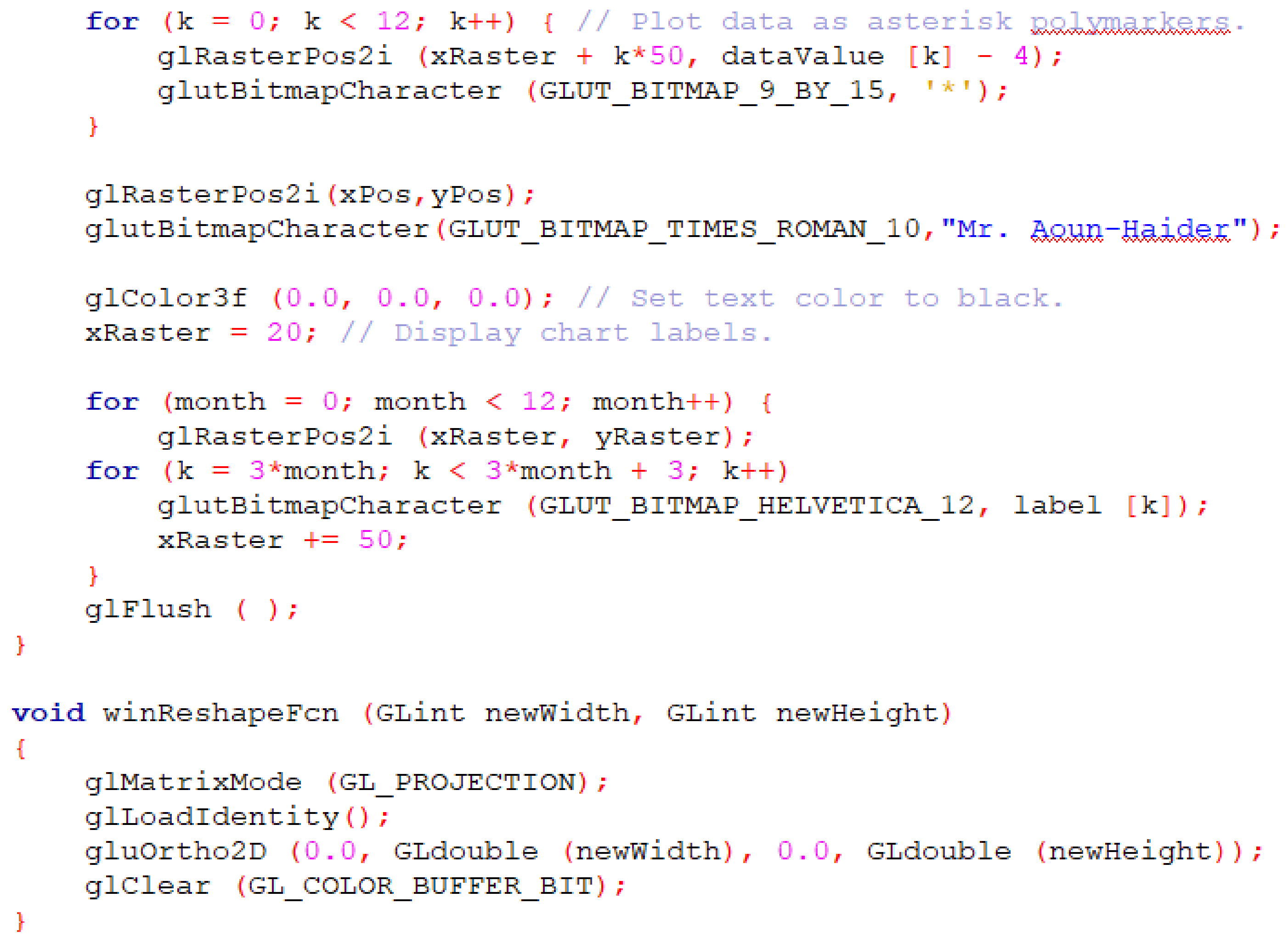


```
glRasterPos2i(xPos,yPos);
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_10,"Mr. Aoun-Haider");
```

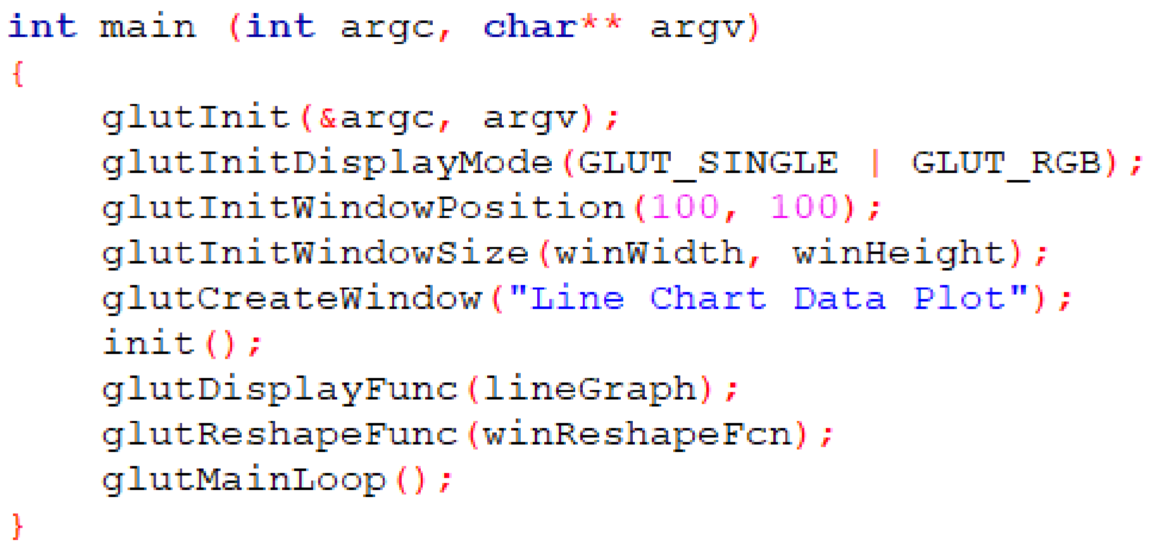
Let display a monthly income chart with asterisks placed at the specific point using bitmap character.



```
#include <GL/glut.h>

GLsizei winWidth = 600, winHeight = 500; // Initial display window size.
GLint xRaster = 25, yRaster = 150; // Initialize raster position.

GLubyte label [36] = {'J', 'a', 'n', 'F', 'e', 'b', 'M', 'a', 'r',
'A', 'p', 'r', 'M', 'a', 'y', 'J', 'u', 'n',
'J', 'u', 'l', 'A', 'u', 'g', 'S', 'e', 'p',
'O', 'c', 't', 'N', 'o', 'v', 'D', 'e', 'c'};

GLint dataValue [12] = {420, 342, 324, 310, 262, 185,
190, 196, 217, 240, 312, 438};

void init (void)
{
    glClearColor (1.0, 1.0, 1.0, 1.0); // White display window.
    glMatrixMode (GL_PROJECTION);
    gluOrtho2D (0.0, 600.0, 0.0, 500.0);
}
void lineGraph (void)
{
    GLint month, k;
    GLint x = 30; // Initialize x position for chart.
    glClear (GL_COLOR_BUFFER_BIT); // Clear display window.
    glColor3f (0.0, 0.0, 1.0); // Set line color to blue.
    glBegin (GL_LINE_STRIP); // Plot data as a polyline.

for (k = 0; k < 12; k++)
    glVertex2i (x + k*50, dataValue [k]);
glEnd();
glColor3f (1.0, 0.0, 0.0); // Set marker color to red.
```



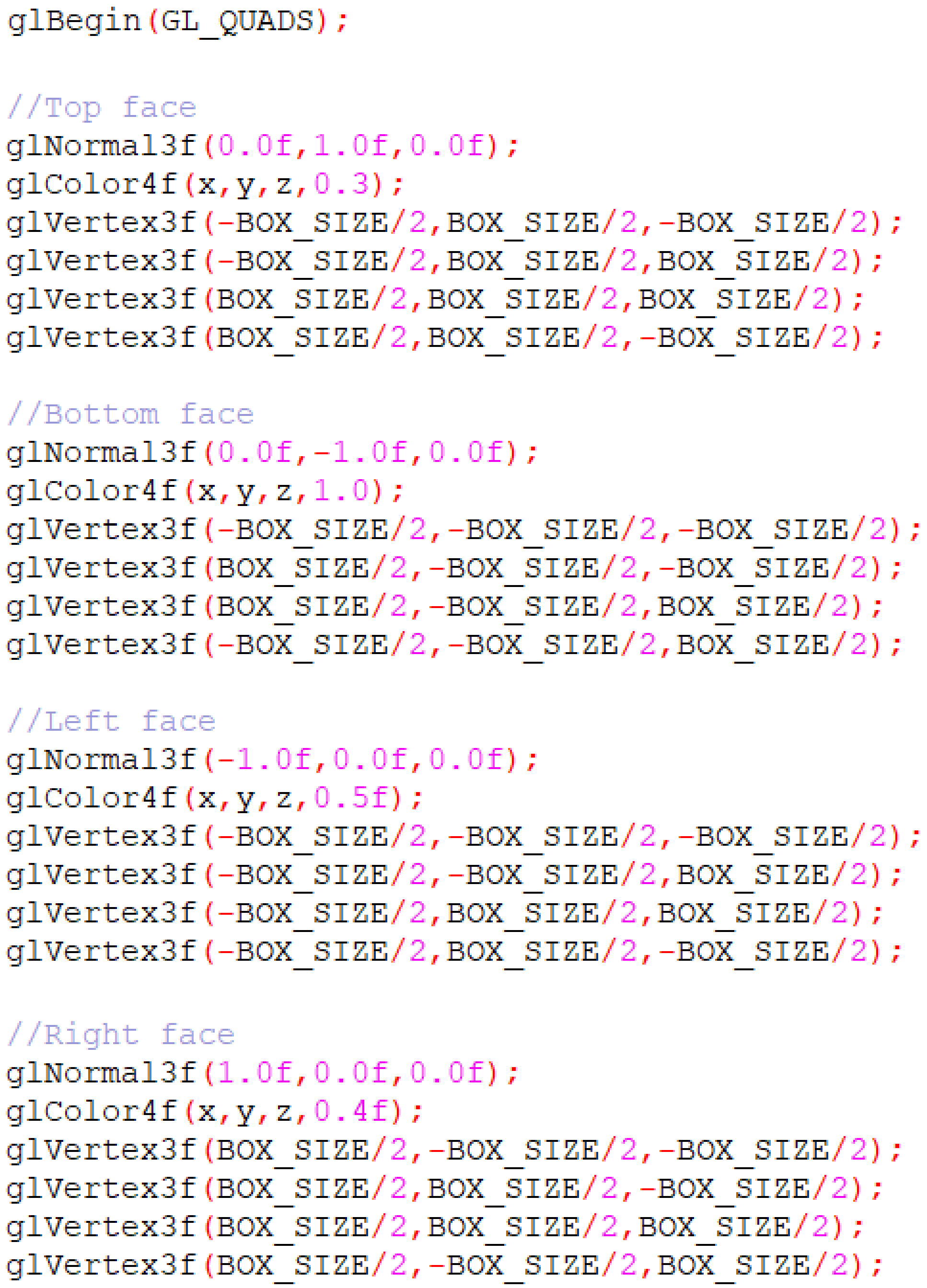
```
    for (k = 0; k < 12; k++) { // Plot data as asterisk polymarkers.
        glRasterPos2i (xRaster + k*50, dataValue [k] - 4);
        glutBitmapCharacter (GLUT_BITMAP_9_BY_15, '*');
    }

    glRasterPos2i(xPos,yPos);
    glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_10,"Mr. Aoun-Haider");

    glColor3f (0.0, 0.0, 0.0); // Set text color to black.
    xRaster = 20; // Display chart labels.

    for (month = 0; month < 12; month++) {
        glRasterPos2i (xRaster, yRaster);
    for (k = 3*month; k < 3*month + 3; k++)
        glutBitmapCharacter (GLUT_BITMAP_HELVETICA_12, label [k]);
        xRaster += 50;
    }
    glFlush ( );
}

void winReshapeFcn (GLint newWidth, GLint newHeight)
{
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D (0.0, GLdouble (newWidth), 0.0, GLdouble (newHeight));
    glClear (GL_COLOR_BUFFER_BIT);
}
```



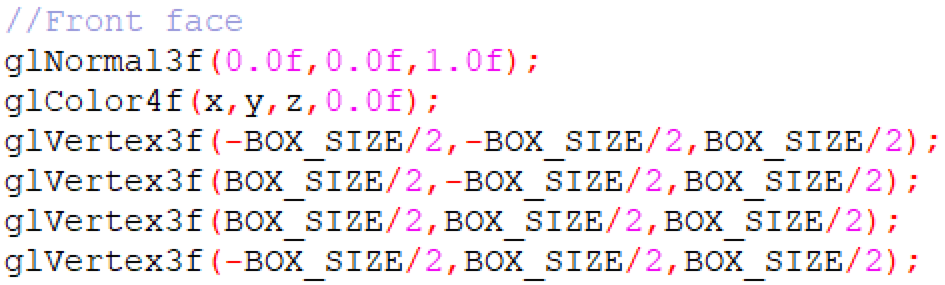
```
int main (int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(winWidth, winHeight);
    glutCreateWindow("Line Chart Data Plot");
    init();
    glutDisplayFunc(lineGraph);
    glutReshapeFunc(winReshapeFcn);
    glutMainLoop();
}
```

Author: Mr. Aoun Haider

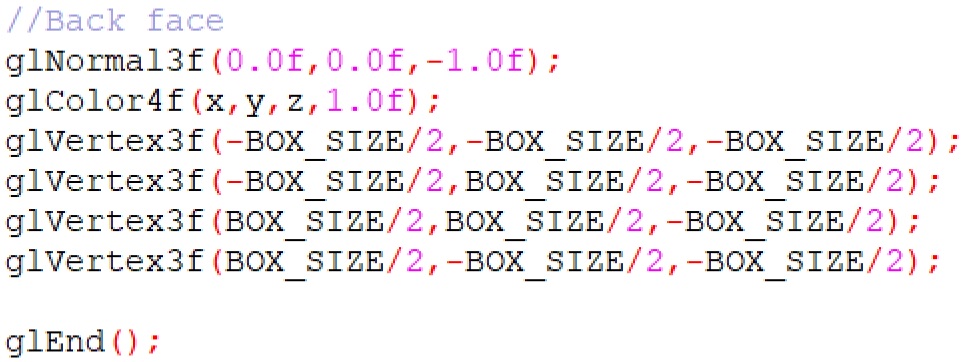
Creating a completely confined cube:



```
glBegin(GL_QUADS);

//Top face
glNormal3f(0.0f,1.0f,0.0f);
glColor4f(x,y,z,0.3);
glVertex3f(-BOX_SIZE/2,BOX_SIZE/2,-BOX_SIZE/2);
glVertex3f(-BOX_SIZE/2,BOX_SIZE/2,BOX_SIZE/2);
glVertex3f(BOX_SIZE/2,BOX_SIZE/2,BOX_SIZE/2);
glVertex3f(BOX_SIZE/2,BOX_SIZE/2,-BOX_SIZE/2);

//Bottom face
glNormal3f(0.0f,-1.0f,0.0f);
glColor4f(x,y,z,1.0);
glVertex3f(-BOX_SIZE/2,-BOX_SIZE/2,-BOX_SIZE/2);
glVertex3f(BOX_SIZE/2,-BOX_SIZE/2,-BOX_SIZE/2);
glVertex3f(BOX_SIZE/2,-BOX_SIZE/2,BOX_SIZE/2);
glVertex3f(-BOX_SIZE/2,-BOX_SIZE/2,BOX_SIZE/2);

//Left face
glNormal3f(-1.0f,0.0f,0.0f);
glColor4f(x,y,z,0.5f);
glVertex3f(-BOX_SIZE/2,-BOX_SIZE/2,-BOX_SIZE/2);
glVertex3f(-BOX_SIZE/2,-BOX_SIZE/2,BOX_SIZE/2);
glVertex3f(-BOX_SIZE/2,BOX_SIZE/2,BOX_SIZE/2);
glVertex3f(-BOX_SIZE/2,BOX_SIZE/2,-BOX_SIZE/2);

//Right face
glNormal3f(1.0f,0.0f,0.0f);
glColor4f(x,y,z,0.4f);
glVertex3f(BOX_SIZE/2,-BOX_SIZE/2,-BOX_SIZE/2);
glVertex3f(BOX_SIZE/2,BOX_SIZE/2,-BOX_SIZE/2);
glVertex3f(BOX_SIZE/2,BOX_SIZE/2,BOX_SIZE/2);
glVertex3f(BOX_SIZE/2,-BOX_SIZE/2,BOX_SIZE/2);
```



```
//Front face
glNormal3f(0.0f,0.0f,1.0f);
glColor4f(x,y,z,0.0f);
glVertex3f(-BOX_SIZE/2,-BOX_SIZE/2,BOX_SIZE/2);
glVertex3f(BOX_SIZE/2,-BOX_SIZE/2,BOX_SIZE/2);
glVertex3f(BOX_SIZE/2,BOX_SIZE/2,BOX_SIZE/2);
glVertex3f(-BOX_SIZE/2,BOX_SIZE/2,BOX_SIZE/2);
```



```
//Back face
glNormal3f(0.0f,0.0f,-1.0f);
glColor4f(x,y,z,1.0f);
glVertex3f(-BOX_SIZE/2,-BOX_SIZE/2,-BOX_SIZE/2);
glVertex3f(-BOX_SIZE/2,BOX_SIZE/2,-BOX_SIZE/2);
glVertex3f(BOX_SIZE/2,BOX_SIZE/2,-BOX_SIZE/2);
glVertex3f(BOX_SIZE/2,-BOX_SIZE/2,-BOX_SIZE/2);

glEnd();
```

//Try to visualize the cube, how it is drawn!!

Normal is just perpendicular to the surface to the plane it is point to.

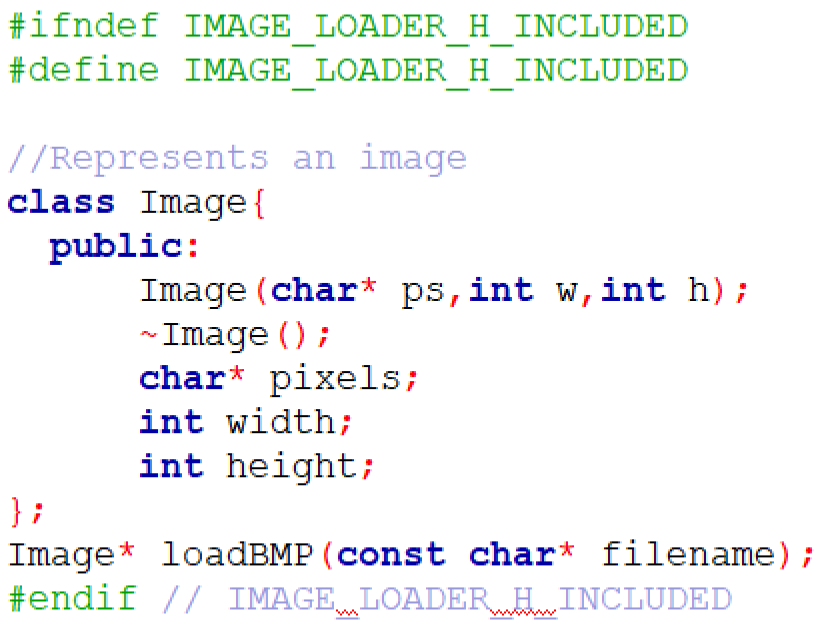
Texture mapping:

To map an image to graphical object, texture mapping can be used. In openGl, it is quite painful to code texture mapping but yet another creative task. First, we have to load the image using an online image loader library, then enable the texture mapping, then bind the texture to the object, then specify filtering technique (magnification, minification, repeat and many more) and then provide the reference coordinate to map the object pixel to texture pixel or texel.

Download imageloader.h from github and include them in the project as imageloader.cpp not .h extension.

https://github.com/anurag173/Modelling-Software-using-OpenGL/blob/master/imageloader.h

//imageloader.cpp



```
#ifndef IMAGE_LOADER_H_INCLUDED
#define IMAGE_LOADER_H_INCLUDED

//Represents an image
class Image{
  public:
      Image(char* ps,int w,int h);
      ~Image();
      char* pixels;
      int width;
      int height;
};
Image* loadBMP(const char* filename);
#endif // IMAGE_LOADER_H_INCLUDED
```

//Example of texture mapping:

#include <iostream>

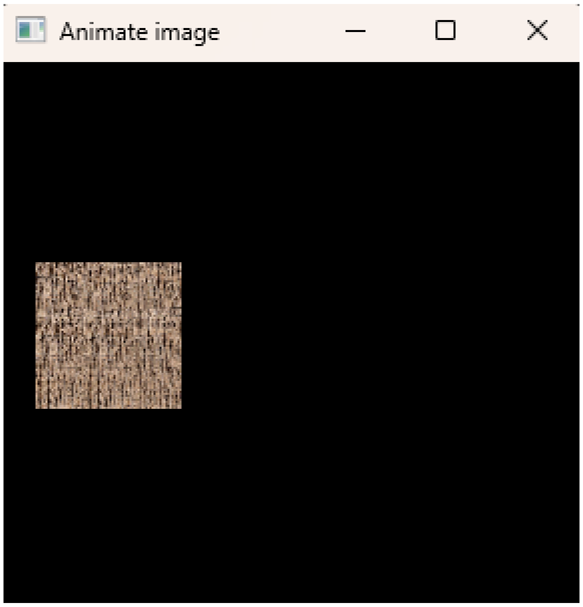
#include <stdlib.h>

#include <GL/glut.h>

#include "imageloader.h"

```
const float BOX_SIZE = 4.0f;
GLfloat xval = 0.0f;
GLuint _textureId;

// Function to load texture from image
GLuint loadTexture(Image* image) {
    GLuint textureId;
    glGenTextures(1, &textureId);
    glBindTexture(GL_TEXTURE_2D, textureId);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image->width, image->height, 0, GL_RGB, GL_UNSIGNED_BYTE, image->pixels);

    // Set texture parameters
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    //glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    //glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

    return textureId;
}

void initRendering() {
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_NORMALIZE);
    glEnable(GL_COLOR_MATERIAL);

    // Load the image for texture
    Image* image =
loadBMP("E:\\OpenGL_Lab\\Texture_Mapping\\sample.bmp");
    if (!image) {
        std::cerr << "Image loading failed" << std::endl;
        exit(1);
    }
    _textureId = loadTexture(image);
    delete image;
}

void handleResize(int w, int h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, (float)w / (float)h, 1.0, 200.0);
}

void drawScene() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(-5.0f + xval, 0.0f, -20.0f);

    // Enable 2D texturing
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, _textureId);

    glBegin(GL_QUADS);
    // Draw front face of the box and map the texture
    glNormal3f(0.0f, 0.0f, 1.0f);
    glTexCoord2f(0.0f, 0.0f);
    glVertex3f(-BOX_SIZE / 2, -BOX_SIZE / 2, BOX_SIZE / 2);
    glTexCoord2f(1.0f, 0.0f);
    glVertex3f(BOX_SIZE / 2, -BOX_SIZE / 2, BOX_SIZE / 2);
    glTexCoord2f(1.0f, 1.0f);
    glVertex3f(BOX_SIZE / 2, BOX_SIZE / 2, BOX_SIZE / 2);
    glTexCoord2f(0.0f, 1.0f);
    glVertex3f(-BOX_SIZE / 2, BOX_SIZE / 2, BOX_SIZE / 2);
    glEnd();

    glDisable(GL_TEXTURE_2D);
    glutSwapBuffers();
    glutPostRedisplay();
}
```

```
void animate() {
    xval += 0.1f;
    if (xval >= 10.0f)
        xval = 0;
    glutPostRedisplay();
}

void mouse(int btn, int state, int x, int y) {
    if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
        glutIdleFunc(animate);
    if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
        glutIdleFunc(NULL);
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(800, 800);
    glutCreateWindow("Animate image");
    initRendering();
    glutDisplayFunc(drawScene);
    glutMouseFunc(mouse);
    glutReshapeFunc(handleResize);
    glutMainLoop();
    return 0;
}
```

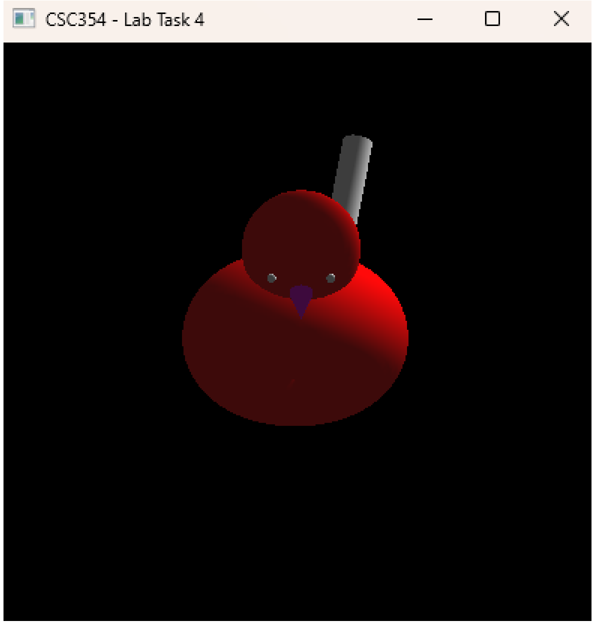


Note: *Change the image path with your image*

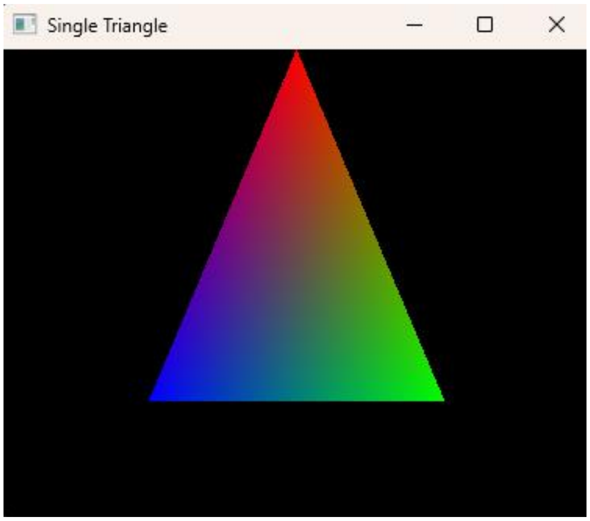
Lab Task<4>

Bloom Taxonomy Level:<Applying>

1. Create a bird using OpenGL primitives, you can also create custom shapes using triangular meshes. When user right click, bird should move from left to right infinitely, if press right click, bird should stop moving at the point where right click is down.



2. Create simple triangle of colors of your own choice with bilinear interpolation implemented:



Solutions:

```
//task-1.cpp
//author: mr. aoun_haider
#include<stdio.h>
#include<GL/glut.h>
GLfloat x = 0;
void init(GLdouble a,GLdouble b,GLdouble c,GLdouble d)
{
    GLfloat light_ambient[] = {a,b,c,d};
    GLfloat light_diffuse[] = {a,b,c,d};
    GLfloat light_specular[] = {1.0,1.0,1.0,1.0};
    GLfloat light_position[] = {1.0,1.0,1.0,0.5};
    glLightfv(GL_LIGHT0,GL_AMBIENT,light_ambient);
    glLightfv(GL_LIGHT0,GL_DIFFUSE,light_diffuse);
    glLightfv(GL_LIGHT0,GL_SPECULAR,light_specular);
    glLightfv(GL_LIGHT0,GL_POSITION,light_position);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);
}
void animate()
{
    x += 0.01;
    if(x >= 1)
        x = 0;
    glutPostRedisplay();
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();

    //Code for drawing lower sphere
    glPushMatrix();
    glTranslatef(-0.5+x,0,0);
    glRotatef(-60,1,0,0);
    glColor3f(1.0,0.0,0.0);
    init(1,0,0,0);
    glutSolidTorus(0.22,0.16,50,50);
    glPopMatrix();

    //Code for drawing upper sphere
    glPushMatrix();
    init(1,0,0,0);
    glTranslatef(-0.48+x,0.3,-0.25);
    glutSolidSphere(0.2,50,20);
    glPopMatrix();

    //Code for drawing beak of bird
    glPushMatrix();
```

```
init(1,0,1,0);
glTranslatef(-0.48+x,0.15,-0.5);
glRotatef(45,1,0,0);
glutSolidCone(0.04,0.12,100,100);
glPopMatrix();

//Code for left eye of bird
glPushMatrix();
glTranslatef(-0.58+x,0.2,-0.5);
init(1,1,1,1);
glutSolidSphere(0.015,50,20);
glPopMatrix();

//Code for right eye of bird
glPushMatrix();
glTranslatef(-0.38+x,0.2,-0.5);
init(1,1,1,1);
glutSolidSphere(0.015,50,20);
glPopMatrix();

//Code for drawing tail of bird
glPushMatrix();
glTranslatef(-0.42+x,0,0.25);
glRotatef(-64,1,0,0);
glRotatef(10,0,1,0);
```

```
    glColor3f(1.0,1.0,0.0);
    GLUquadric *cyll = gluNewQuadric();
    gluCylinder(cyll,0.05,0.05,0.75,100,100);
    glPopMatrix();
    glFlush();
    glutSwapBuffers();
}
void mouse(int btn,int state,int x,int y)
{
    if(btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
        glutIdleFunc(animate);
    if(btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
        glutIdleFunc(NULL);
}
int main(int argc,char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500,500);
    glutCreateWindow("CSC354 - Lab Task 4");
    init(1,1,0,0);
    glutMouseFunc(mouse);
    glutDisplayFunc(display);
    glutMainLoop();
}
```

```
//task-2.cpp
//author: mr. aoun_haider
#include<GL/glut.h>
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
    glColor3f(0.0,0.0,1.0); //blue
    glVertex2f(-0.5,-0.5);
    glColor3f(0.0,1.0,0.0); //green
    glVertex2f(0.5,-0.5);
    glColor3f(1.0,0.0,0.0); //red
    glVertex2f(0.0,1.0);
    glEnd();
    glFlush(); //single buffer, so need a flush
}
int main(int argc,char** argv)
{
    glutInit(&argc,argv);
    glutCreateWindow("Single Triangle");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

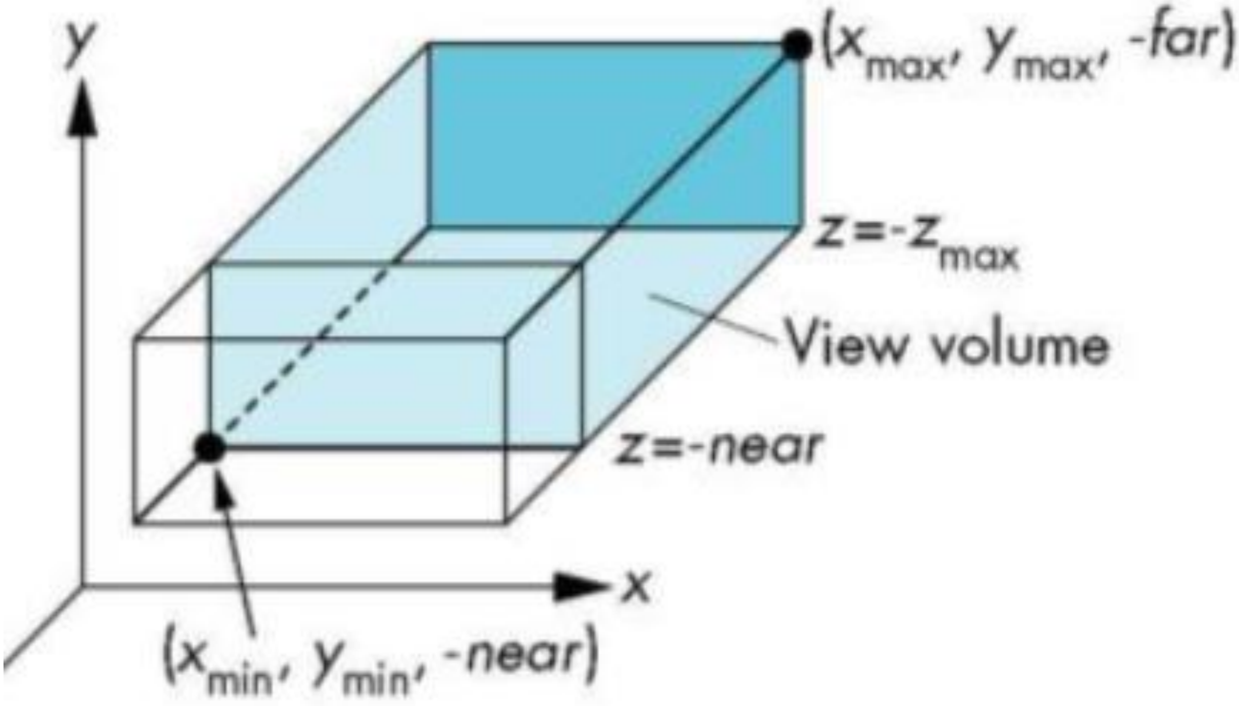
Perspective projection:

OpenGL maintains two matrices one for projection and other for model view. We can change the camera orientation to look some distant object far closer or from other different angles. In 3D scenes, camera acts like player a move in a room which gives us feel as player is going inside the room. There are a lot of methods to set the viewport and camera position.

1. ***gluLookAt(***

   ```
           GLdouble eyeX,
           GLdouble eyeY,
           GLdouble eyeZ, //position at which camera will see object
           GLdouble lookatX,
           GLdouble lookatY,
           GLdouble lookatZ, //where object to be seen by camera is located
           GLdouble upX,
           GLdouble upY,
           GLdouble upZ, //up direction of camera
       );
   ```

2. ***gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top);***
3. ***gluOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);***
4. ***glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble nearVal, GLdouble farVal);***
5. ***gluPerspective(GLdouble fieldOfViewAngle, GLdouble aspectRatio, GLdouble zNear, GLdouble zFar);***

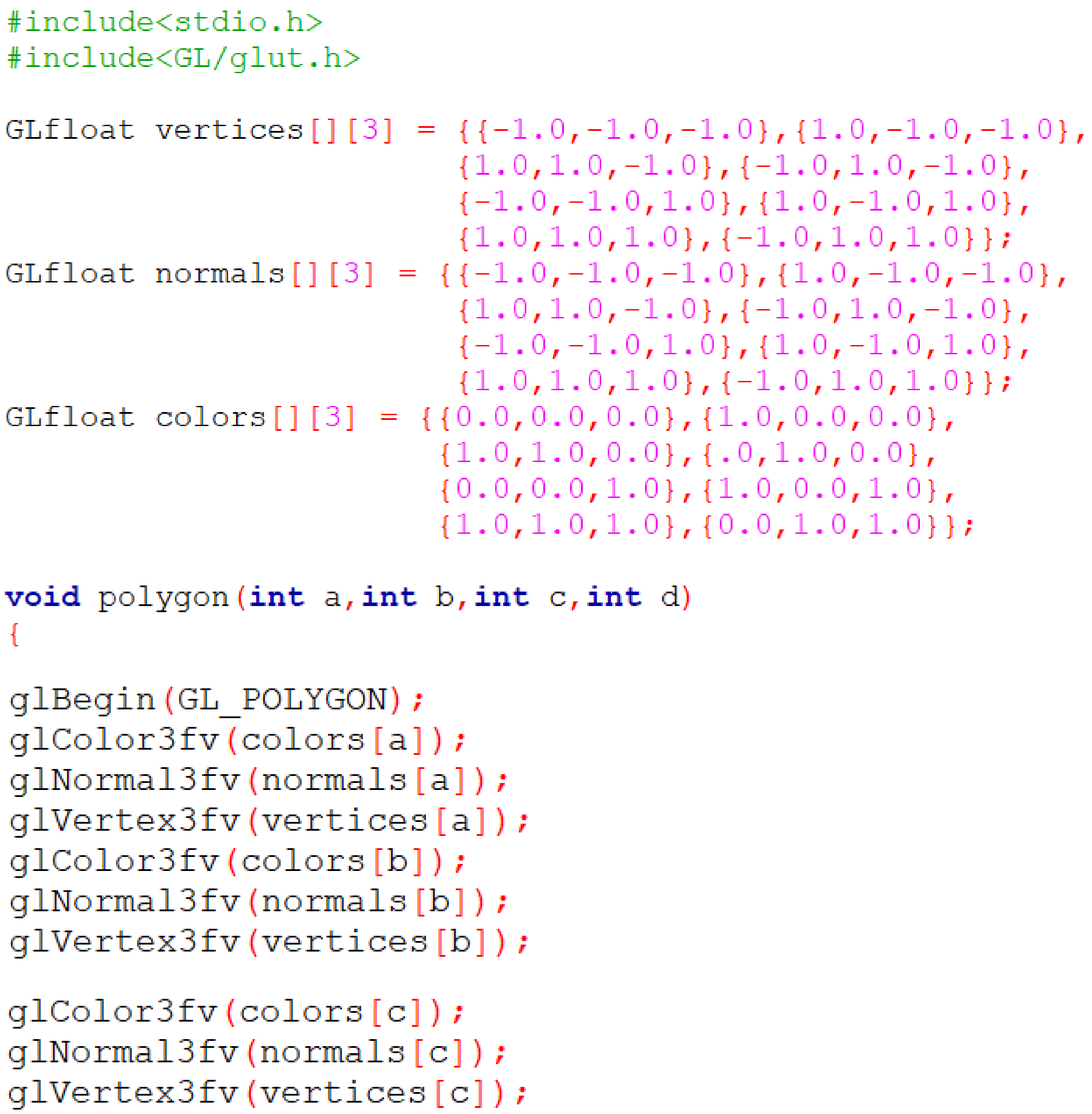


Near value: distance b/w object & camera

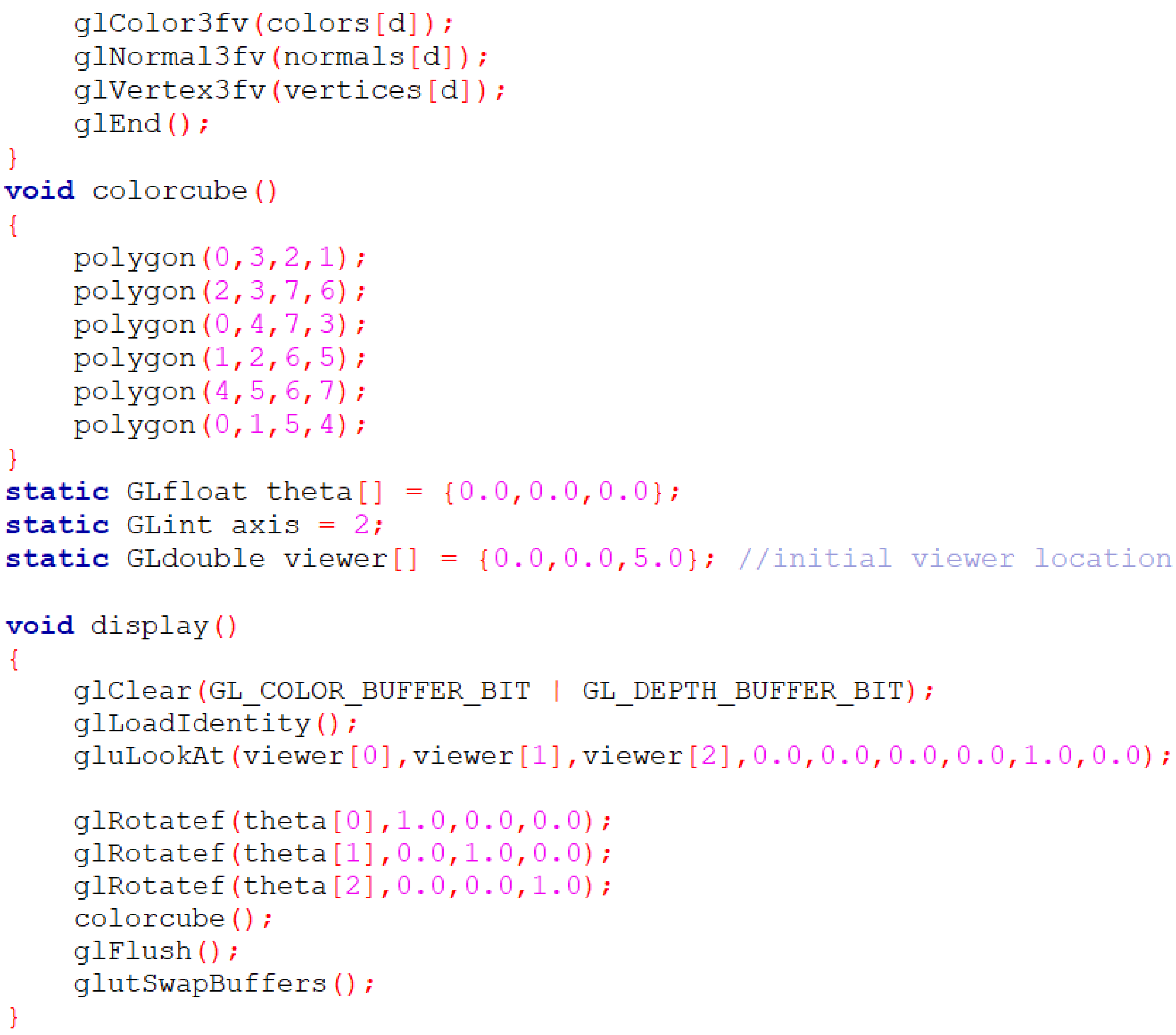
Far value: maximum distance b/w object & camera after which object will be invisible.

Example program:

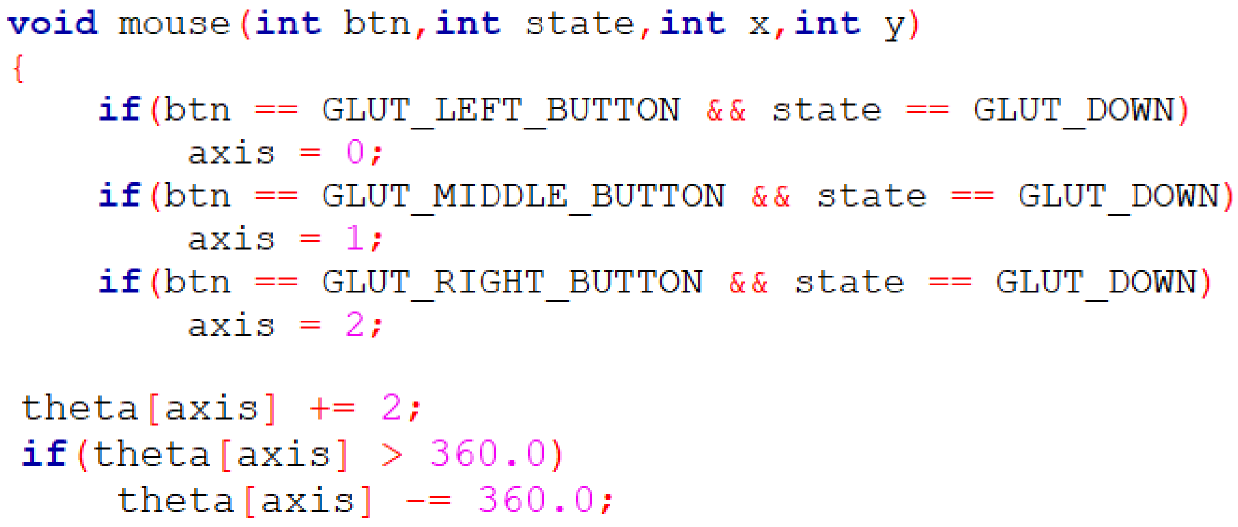
Let's take the previous example of cube rotating around x-axis on left click, around y-axis on middle mouse button click and around z-axis on right click. When user press 'x', x-axis distance between camera and object (cube in this case) will be decreased by 1 unit. If pressed 'X', x-axis distance is increased by 1 unit. Same for all other axis as well.



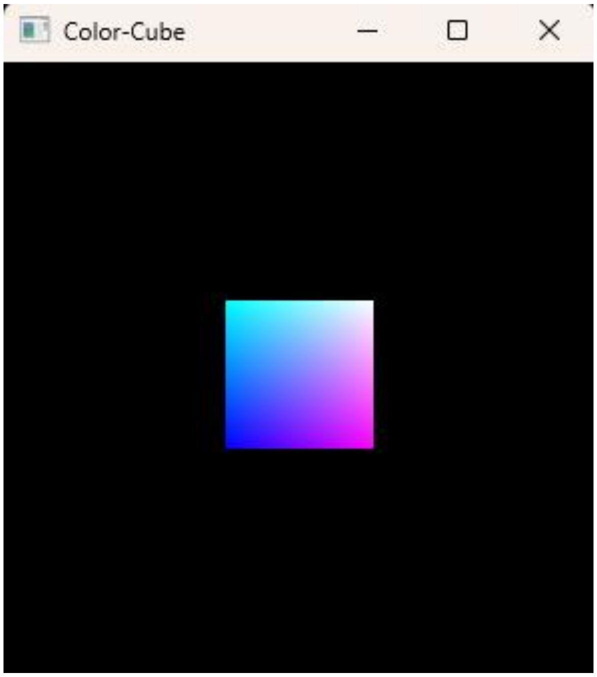
```
#include<stdio.h>
#include<GL/glut.h>

GLfloat vertices[][3] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},
                        {1.0,1.0,-1.0},{-1.0,1.0,-1.0},
                        {-1.0,-1.0,1.0},{1.0,-1.0,1.0},
                        {1.0,1.0,1.0},{-1.0,1.0,1.0}};
GLfloat normals[][3] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},
                        {1.0,1.0,-1.0},{-1.0,1.0,-1.0},
                        {-1.0,-1.0,1.0},{1.0,-1.0,1.0},
                        {1.0,1.0,1.0},{-1.0,1.0,1.0}};
GLfloat colors[][3] = {{0.0,0.0,0.0},{1.0,0.0,0.0},
                       {1.0,1.0,0.0},{.0,1.0,0.0},
                       {0.0,0.0,1.0},{1.0,0.0,1.0},
                       {1.0,1.0,1.0},{0.0,1.0,1.0}};

void polygon(int a,int b,int c,int d)
{

glBegin(GL_POLYGON);
glColor3fv(colors[a]);
glNormal3fv(normals[a]);
glVertex3fv(vertices[a]);
glColor3fv(colors[b]);
glNormal3fv(normals[b]);
glVertex3fv(vertices[b]);

glColor3fv(colors[c]);
glNormal3fv(normals[c]);
glVertex3fv(vertices[c]);
```

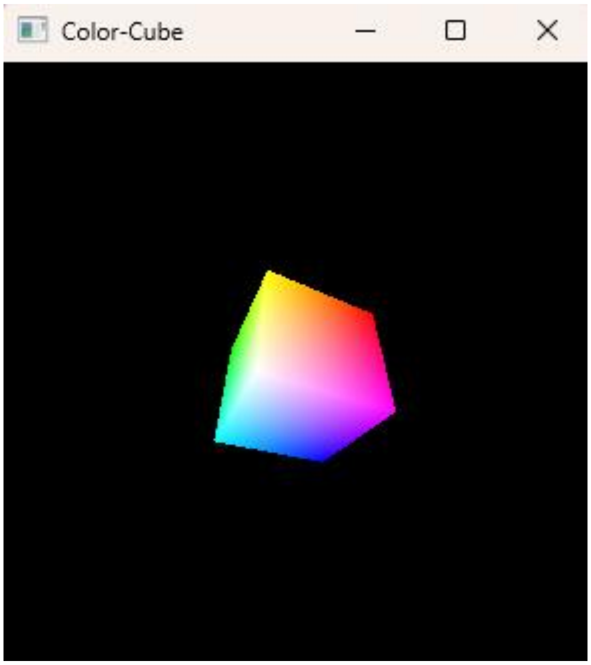


```
    glColor3fv(colors[d]);
    glNormal3fv(normals[d]);
    glVertex3fv(vertices[d]);
    glEnd();
}
void colorcube()
{
    polygon(0,3,2,1);
    polygon(2,3,7,6);
    polygon(0,4,7,3);
    polygon(1,2,6,5);
    polygon(4,5,6,7);
    polygon(0,1,5,4);
}
static GLfloat theta[] = {0.0,0.0,0.0};
static GLint axis = 2;
static GLdouble viewer[] = {0.0,0.0,5.0}; //initial viewer location

void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    gluLookAt(viewer[0],viewer[1],viewer[2],0.0,0.0,0.0,0.0,1.0,0.0);

    glRotatef(theta[0],1.0,0.0,0.0);
    glRotatef(theta[1],0.0,1.0,0.0);
    glRotatef(theta[2],0.0,0.0,1.0);
    colorcube();
    glFlush();
    glutSwapBuffers();
}
```



```
void mouse(int btn,int state,int x,int y)
{
    if(btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
        axis = 0;
    if(btn == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN)
        axis = 1;
    if(btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
        axis = 2;

 theta[axis] += 2;
 if(theta[axis] > 360.0)
     theta[axis] -= 360.0;
    display();
}

void keys(unsigned char key,int x,int y)
{
    switch(key)
    {
        case 'x':
            viewer[0] -= 1.0;
        break;

        case 'X':
            viewer[0] += 1.0;
        break;

        case 'y':
            viewer[1] -= 1.0;
        break;

        case 'Y':

        viewer[1] += 1.0;
    break;

    case 'z':
        viewer[2] -= 1.0;
    break;

    case 'Z':
        viewer[2] += 1.0;
    break;
}
display();

}
void reshape(int w,int h)
{
    glViewport(0,0,w,h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    if(w <= h)
        glFrustum(-2.0,2.0,-2.0*(GLfloat)h/(GLfloat)w,2.0*(GLfloat)h/(GLfloat)w,2.0,20.0);
    else
        glFrustum(-2.0,2.0,-2.0*(GLfloat)w/(GLfloat)h,2.0*(GLfloat)w/(GLfloat)h,2.0,20.0);

    //gluPerspective(45.0,w/h,-10.0,10.0);
    glMatrixMode(GL_MODELVIEW);
}

int main(int argc,char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500,500);
    glutCreateWindow("Color-Cube");
    glutReshapeFunc(reshape);
    glutMouseFunc(mouse);
    glutDisplayFunc(display);
    glutKeyboardFunc(keys);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
}
```





Try update viewer array to see how size of object changes with gluLookAt(); Increasing z-value will maximizes the object size, y- will move camera upward.

Author: Mr. Aoun Haider