UCS505
COMPUTER GRAPHICS LAB ASSIGNMENTS

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Submitted To: Ms. Rupali

Submitted By

Vaibhav Malhotra (101803129)

Batch: COE-06

Computer Science and Engineering Department

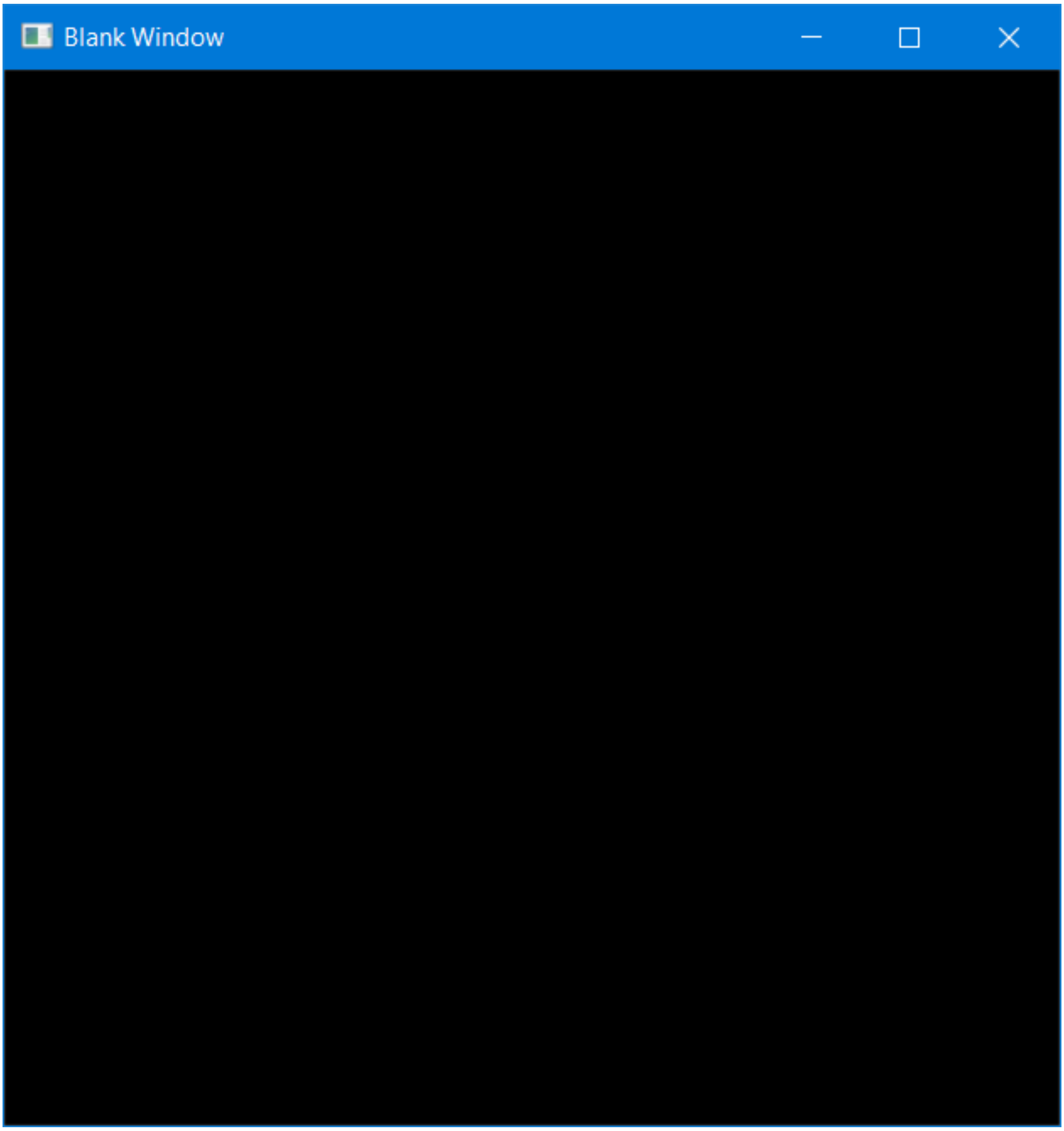
Thapar Institute of Engineering and Technology, Patiala

Table of Contents

1. Create Empty Window

Black Color
(Screen)

```
    #include<GL/
    glut.h>
    void display() {
    glClear(GL_COLOR_BUFFER_BIT); glColor3f(1.0, 0.0,
    0.0);glFlush();
    }
    void myinit()
    { glClearColor(0.0, 0.0, 0.0,
    0.0);
    glColor3f(1.0, 0.0, 0.0); glPointSize(5.0);
    glMatrixMode(GL_PROJECTION); gluOrtho2D(0.0, 499.0, 0.0,
    499.0);
    }
void main(int argc, char** argv) { glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE |
    GLUT_RGB);glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0); glutCreateWindow("Blank
    Window");glutDisplayFunc(display);
    myinit(); glutMainLoop();
    }
```



White Color (Screen)

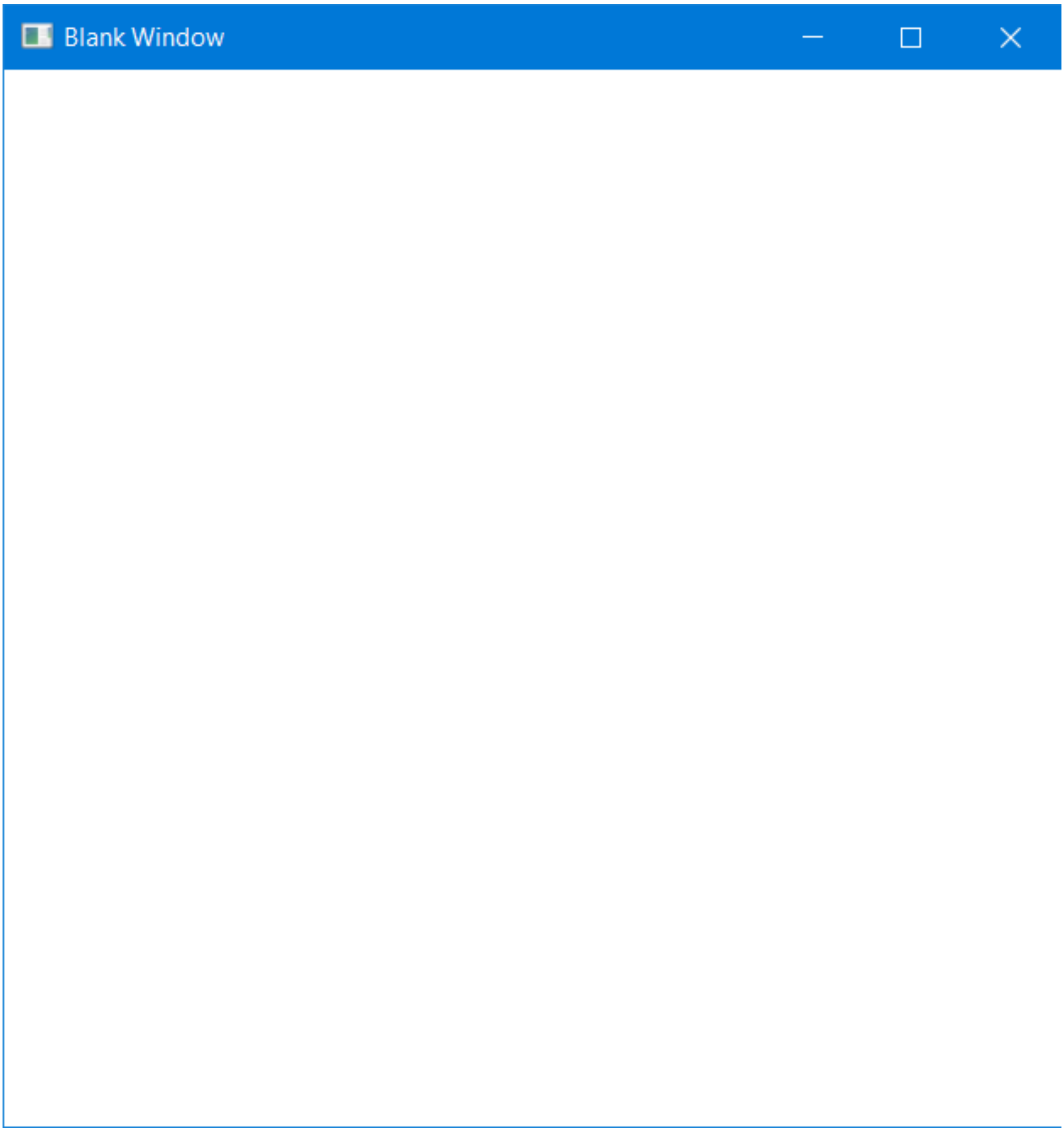
```
#include<GL/glut.h>

void display() {

glClear(GL_COLOR_BUFFER_BIT); glColor3f(1.0, 0.0, 0.0);
```

```
    glFlush();

    }

    void myinit()

    { glClearColor(1.0, 1.0, 1.0,

    1.0);

    glColor3f(1.0, 0.0, 0.0); glPointSize(5.0);
    glMatrixMode(GL_PROJECTION); gluOrtho2D(0.0, 499.0, 0.0,
    499.0);

    }
void main(int argc, char** argv) { glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_SINGLE |
    GLUT_RGB);glutInitWindowSize(500, 500);

    glutInitWindowPosition(0, 0); glutCreateWindow("Blank
    Window");glutDisplayFunc(display);
    myinit(); glutMainLoop();

    }
```

Colored (Screen)

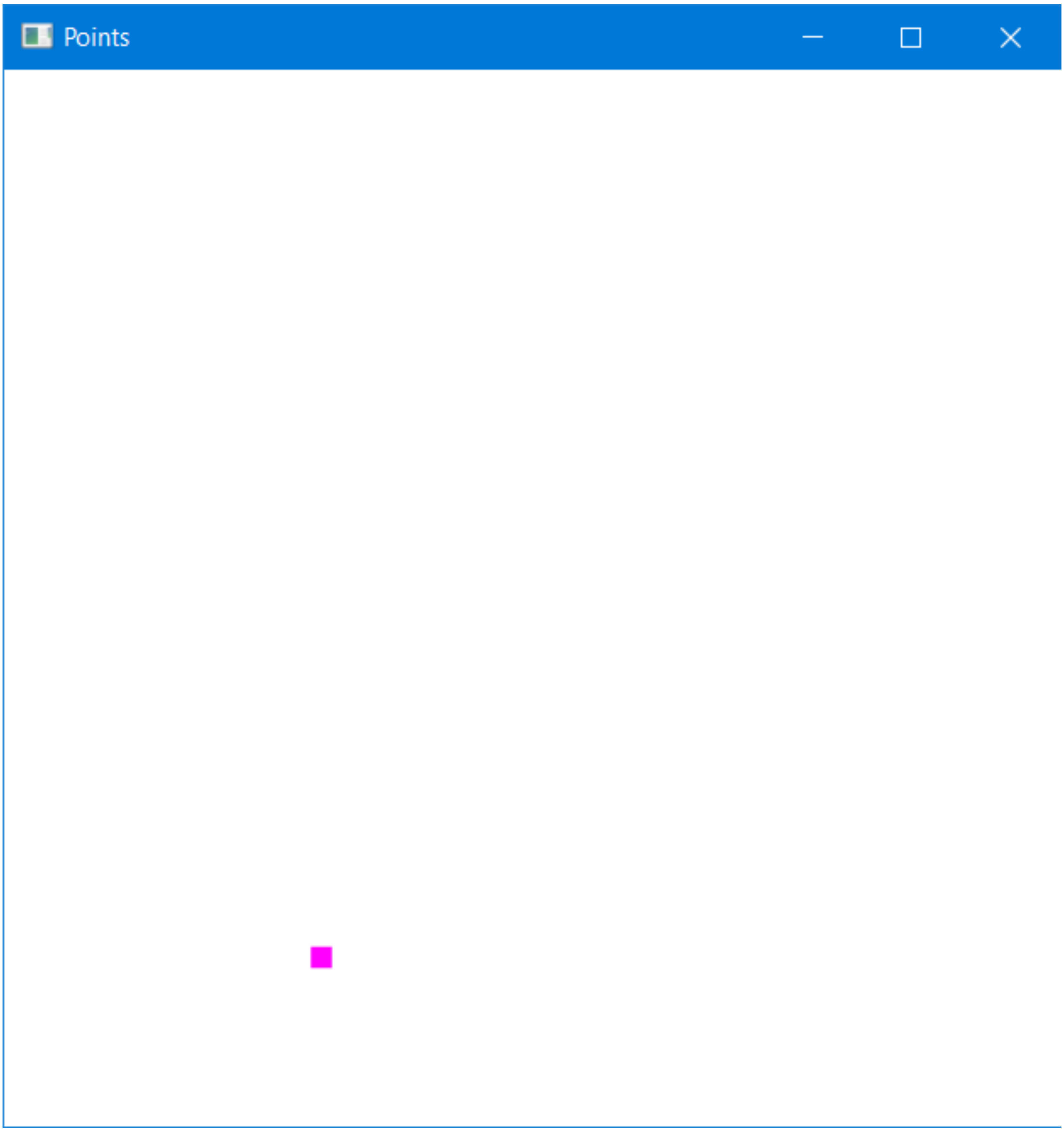
```
#include<GL/
glut.h>
void display() {

glClear(GL_COLOR_BUFFER_BIT); glColor3f(1.0, 0.0,

0.0);glFlush();

}
void myinit()

{ glClearColor(0.0, 1.0, 1.0,

0.0);

glColor3f(1.0, 0.0, 0.0); glPointSize(5.0);
glMatrixMode(GL_PROJECTION); gluOrtho2D(0.0, 499.0, 0.0,
499.0);
```

```
        }

    void main(int argc, char** argv) { glutInit(&argc, argv);

        glutInitDisplayMode(GLUT_SINGLE |
        GLUT_RGB);glutInitWindowSize(500, 500);

        glutInitWindowPosition(0, 0); glutCreateWindow("Blank
        Window");glutDisplayFunc(display);
        myinit(); glutMainLoop();

        }
```



2. Draw a point of width 10 pixel

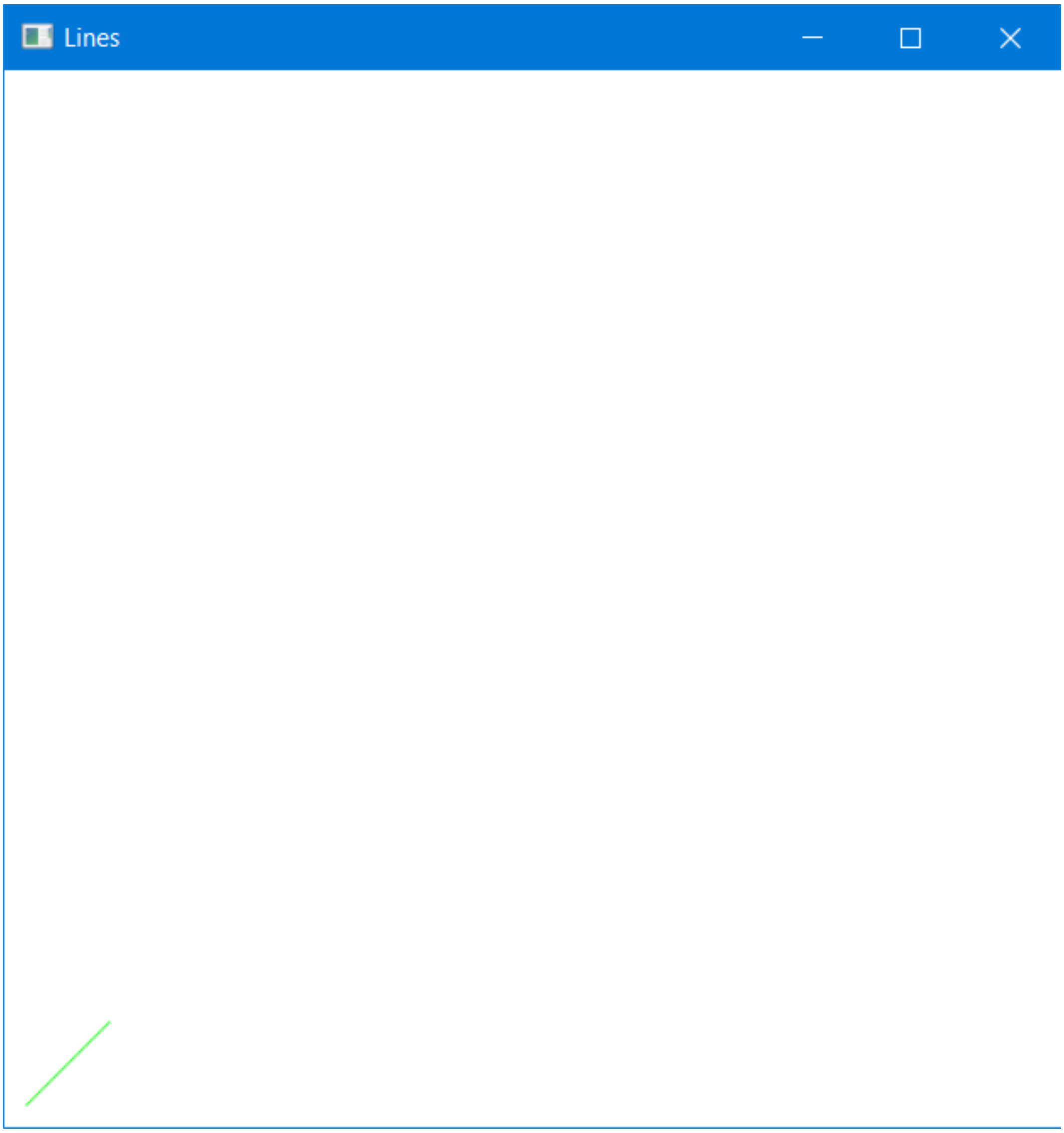
```
          #include<GL/glut.h>
          void display() {

          glClear(GL_COLOR_BUFFER_BIT); glColor3f(1.0, 0.0, 1.0);
          glBegin(GL_POINTS); glVertex2f(150.0, 80.0);

          glEnd();glFlush();

          }
          void myinit()

          { glClearColor(1.0, 1.0, 1.0,

          1.0);

          glColor3f(1.0, 0.0, 0.0); glPointSize(10.0);
          glMatrixMode(GL_PROJECTION); gluOrtho2D(0.0, 499.0, 0.0,
          499.0);

          }
     void main(int argc, char** argv) { glutInit(&argc, argv);

          glutInitDisplayMode(GLUT_SINGLE            |
          GLUT_RGB);glutInitWindowSize(500, 500);

          glutInitWindowPosition(5,        5);
          glutCreateWindow("Points");glutDisplayFunc(display);
          myinit(); glutMainLoop();
}
```



Points

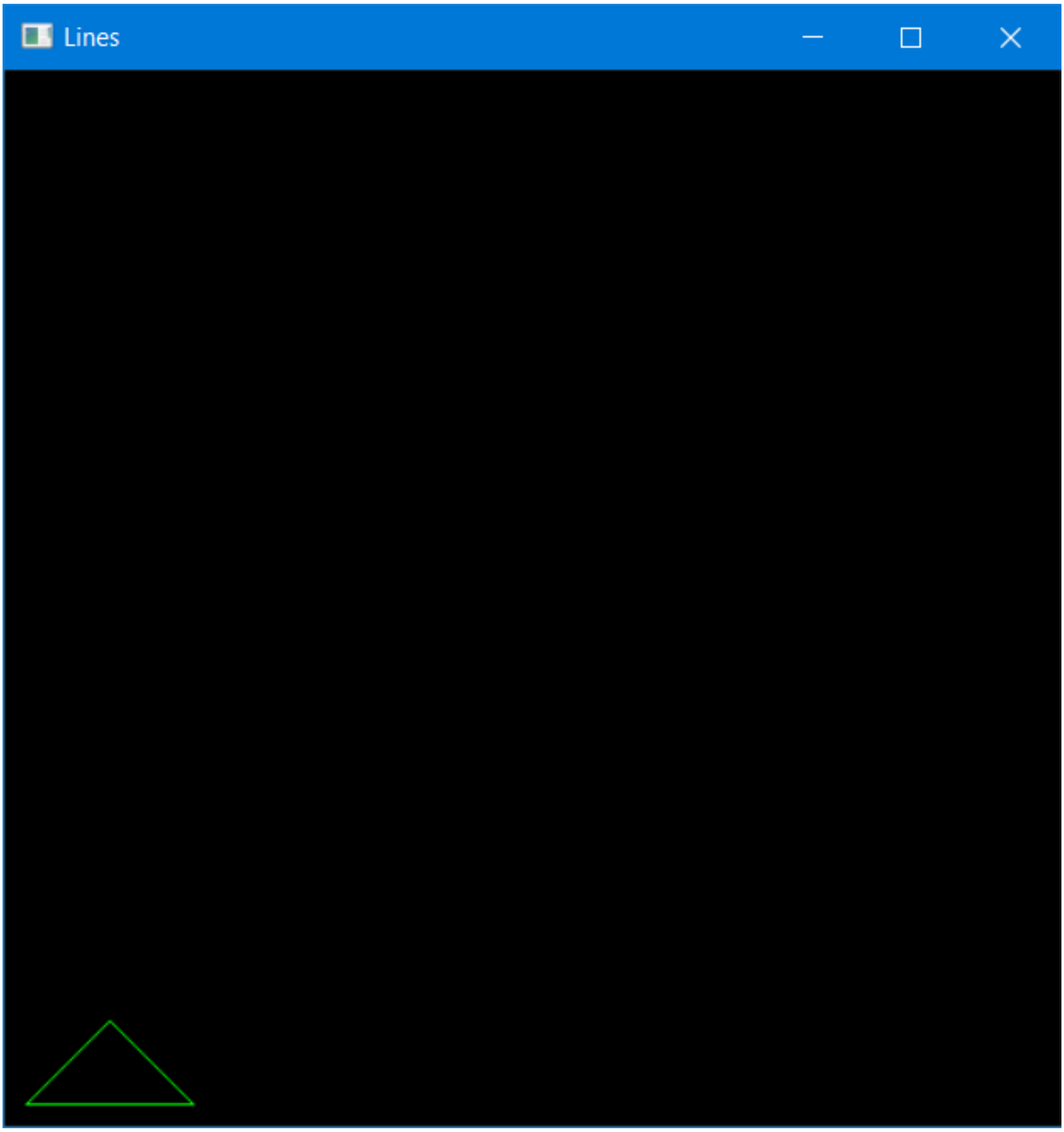
3. Draw a green color line from (10,10) to (50,50)

```
#include<GL/glut.h>
 void display() {

 glClear(GL_COLOR_BUFFER_BIT); glColor3f(0.0, 100.0,
 0.0);
 glBegin(GL_LINES); glVertex2f(10.0, 10.0);

 glVertex2f(50.0, 50.0);

 glEnd();glFlush();


 }

 void myinit()

 { glClearColor(1.0, 1.0, 1.0,

 1.0);

 glColor3f(1.0, 0.0, 0.0); glPointSize(10.0);
 glMatrixMode(GL_PROJECTION); gluOrtho2D(0.0, 499.0, 0.0,
 499.0);

 }
void main(int argc, char** argv) { glutInit(&argc, argv);

 glutInitDisplayMode(GLUT_SINGLE |
 GLUT_RGB);glutInitWindowSize(500, 500);

 glutInitWindowPosition(5, 5);
 glutCreateWindow("Lines");glutDisplayFunc(display);
 myinit(); glutMainLoop();

 }
```


Lines

4. Draw a triangle on a black background

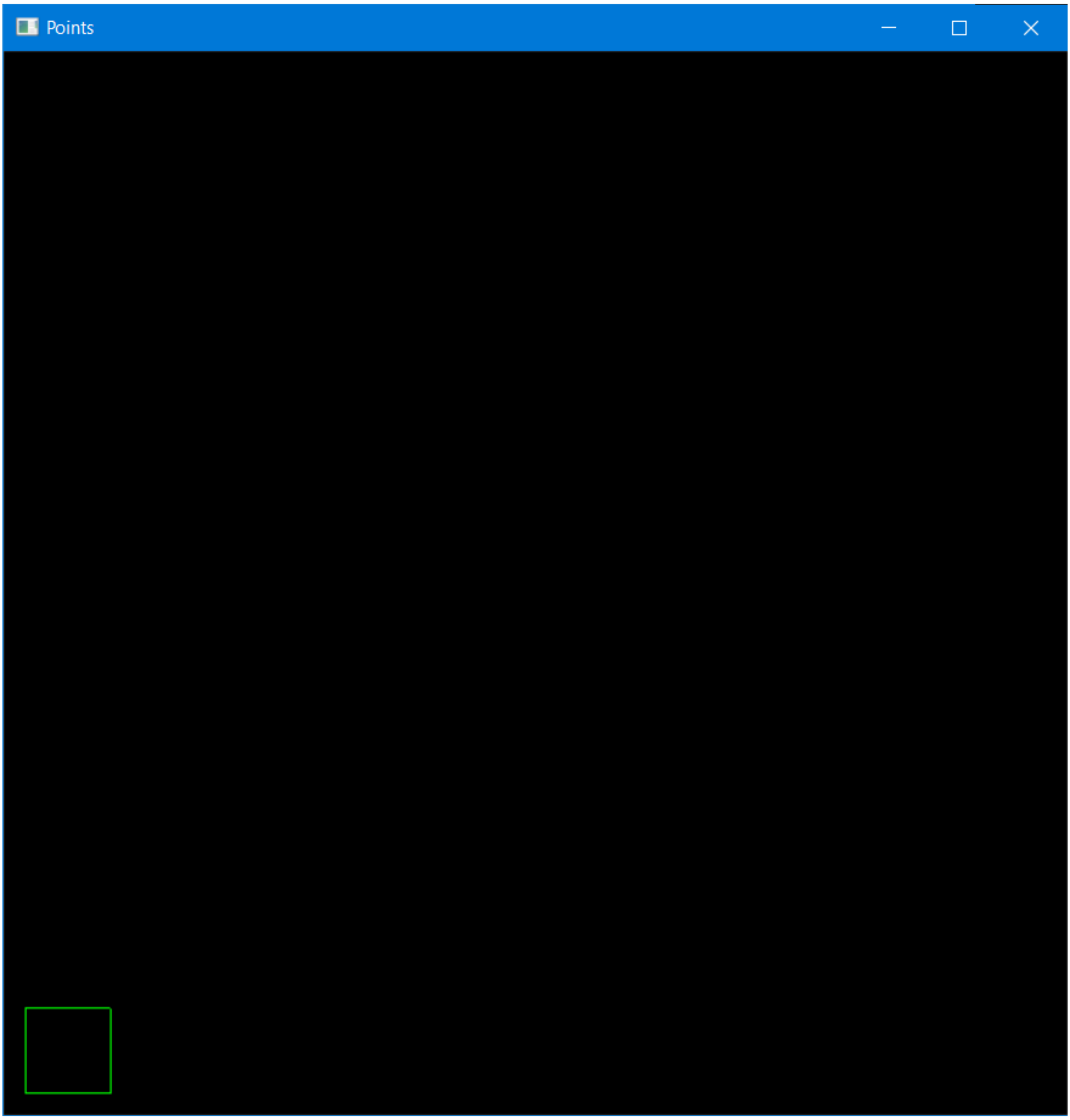
```
    #include<GL/glut.h>
    void display() {

    glClear(GL_COLOR_BUFFER_BIT); glColor3f(0.0, 100.0,
    0.0);
    glBegin(GL_LINES); glVertex2f(10.0, 10.0);

    glVertex2f(50.0, 50.0);

    glVertex2f(50.0, 50.0);

    glVertex2f(90.0, 10.0);

    glVertex2f(10.0, 10.0);

    glVertex2f(90.0, 10.0);

    glEnd();glFlush();

    }
    void myinit()

    { glClearColor(0.0, 0.0, 0.0,

    0.0);

    glColor3f(1.0, 0.0, 0.0); glPointSize(10.0);
    glMatrixMode(GL_PROJECTION); gluOrtho2D(0.0, 499.0, 0.0,
    499.0);
    }
void main(int argc, char** argv) { glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_SINGLE |
    GLUT_RGB);glutInitWindowSize(500, 500);

    glutInitWindowPosition(5, 5);
    glutCreateWindow("Lines");glutDisplayFunc(display);
    myinit(); glutMainLoop();

    }
```


Lines

5. Draw a rectangle on black background

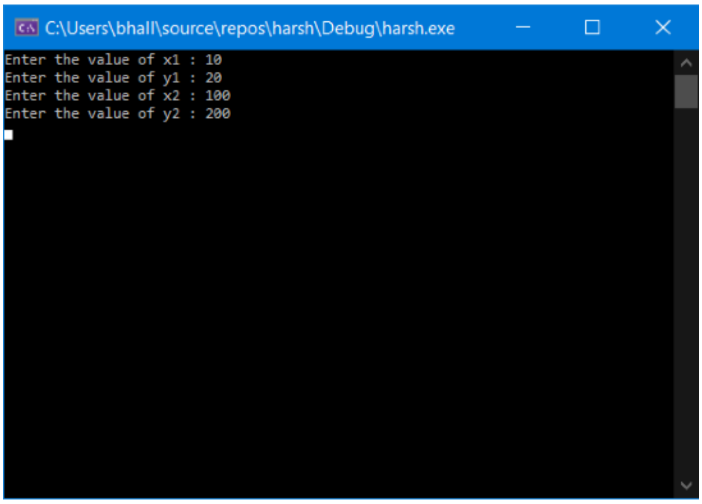
```
#include<GL/glut.h>

void display() {

 glClear(GL_COLOR_BUFFER_BIT);

glColor3f(0.0,0.0,0.0);

glBegin(GL_LINES);
glVertex2f(10.0, 10.0);

glVertex2f(50.0, 10.0);

glVertex2f(50.0, 10.0);

glVertex2f(50.0, 50.0);

glVertex2f(50.0, 50.0);

glVertex2f(10.0, 50.0);

glVertex2f(10.0, 50.0);

glVertex2f(10.0, 10.0);

 glEnd();

glFlush();

}

void myinit()

{ glClearColor(1.0, 1.0, 0.0,

1.0);

glColor3f(1.0, 0.0, 0.0); glPointSize(50.0);
glMatrixMode(GL_PROJECTION); gluOrtho2D(0.0, 499.0, 0.0,
499.0);

}
```

```
void main(int argc, char** argv) { glutInit(&argc,  argv);

          glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
          glutInitWindowSize(700, 700);

          glutInitWindowPosition(5, 5);
          glutCreateWindow("Points");glutDisplayFunc(display);
          myinit();
          glutMainLoop();}
          }
```

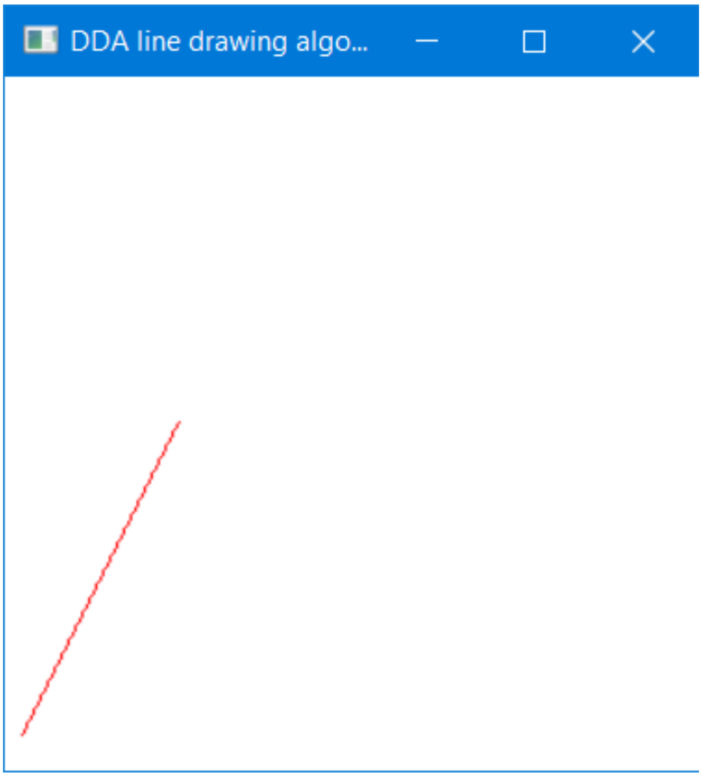


6. Draw a line using DDA algorithm

```
#include using namespace std;
float x1, x2, Y1, y2;
void display(void)
{ glClear(GL_COLOR_BUFFER_BIT);
float dy, dx, step, x, y, k, Xin, Yin;
dx = x2 - x1;
dy = y2 - Y1;
if (abs(dx) > abs(dy))
{ step = abs(dx); }
else step = abs(dy);
Xin = dx / step;
Yin = dy / step; x = x1;
y = Y1;
glBegin(GL_POINTS);
glColor3f(1.0, 0.0, 0.0);
glVertex2i(x, y);
glEnd();
for (k = 1; k <= step; k++)
{ x = x + Xin;
 y = y + Yin;
 glBegin(GL_POINTS);
glVertex2i(x, y);
glEnd(); }
glFlush(); }
void init(void)
{ glClearColor(1, 1, 1, 1);
glMatrixMode(GL_PROJECTION);
gluOrtho2D(0, 400, 0, 400); }
 int main(int argc, char** argv)
 { cout << "Enter the value of x1 : ";
cin >> x1;
 cout << "Enter the value of y1 : “;
 cin >> Y1;
cout <<"Enter the value of x2 : ";
cin >> x2;
cout << "Enter the value of y2 : ";
cin >> y2;
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutCreateWindow("DDA line drawing algorithm");
glutDisplayFunc(display);
init();
glutMainLoop();
return 0; }
```



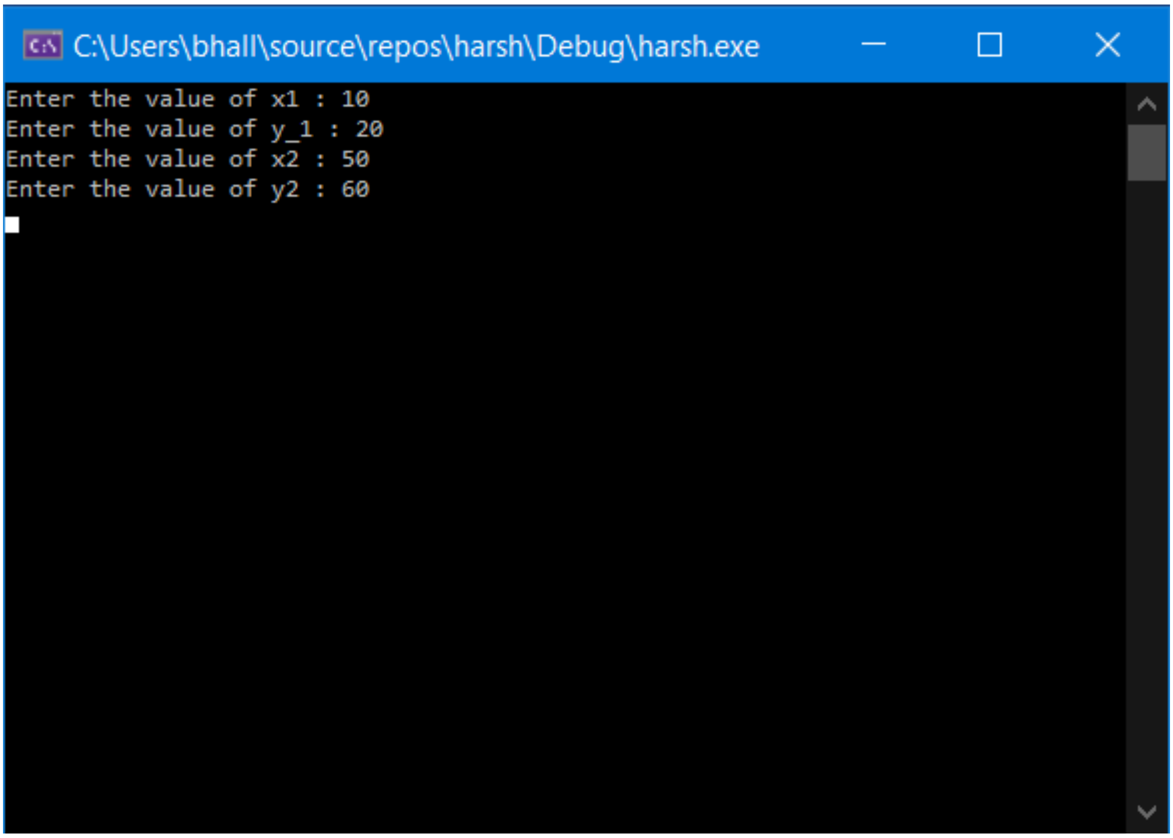
```
Enter the value of x1 : 10
Enter the value of y1 : 20
Enter the value of x2 : 100
Enter the value of y2 : 200
```



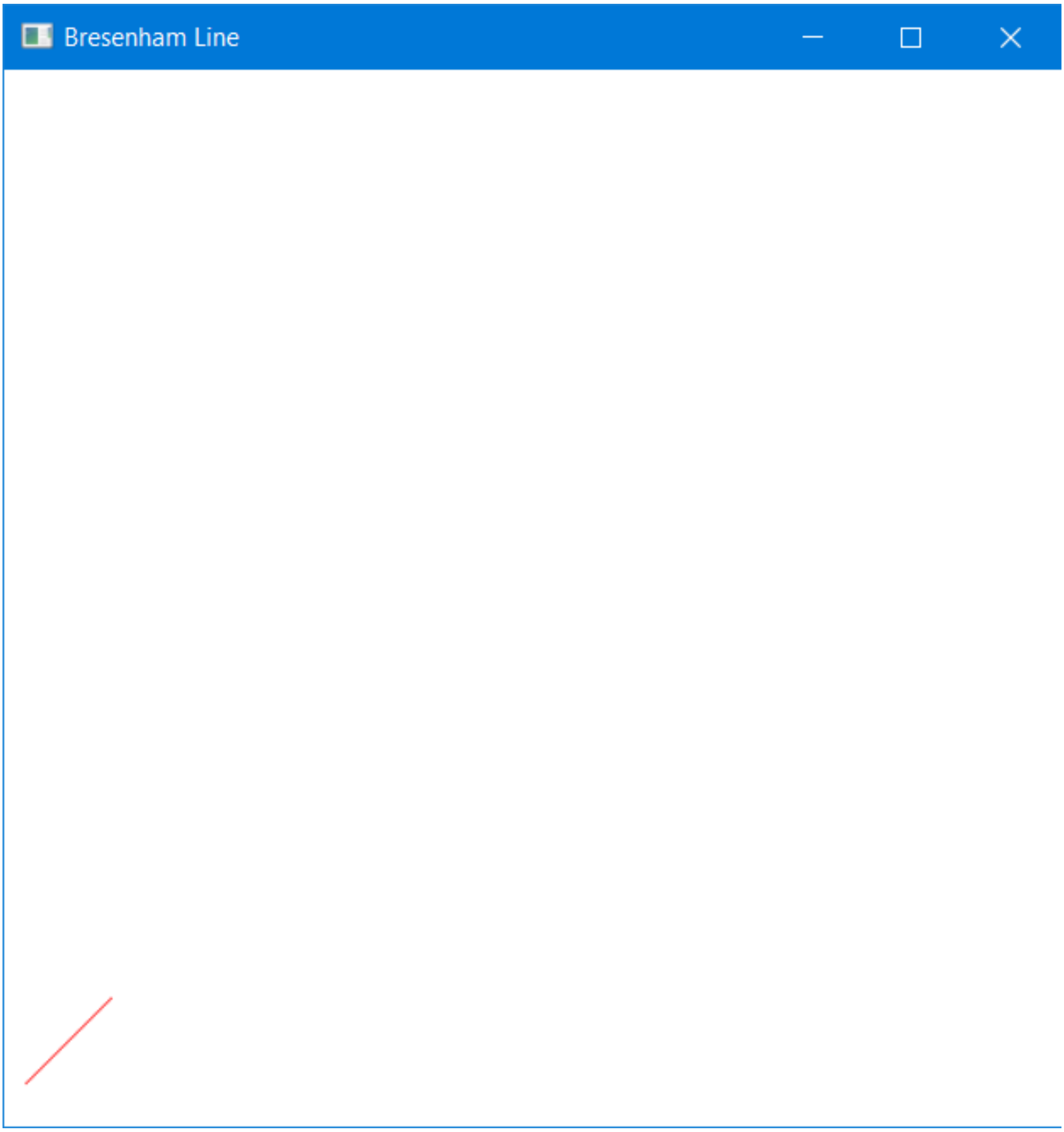
7. Draw a line using Bresenham algorithm

```
//Bresenham Line Drawing algo
#include <GL/glut.h>
#include<iostream>
int x1, y_1, x2, y2, x, y;
void display()
{
  x = x1;
  y = y_1;
  int   dx, dy,           //deltas
     pk,                  //decision parameter
     k, y_inc;            //looping variable

  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1, 0, 0);
  //plot first point x1, y_1
  glBegin(GL_POINTS);
  glVertex2i(x, y);
  glEnd();

  // difference between starting and ending points
  dx = x2 - x1;
  dy = y2 - y_1;
  pk = 2 * dy - dx;

  if (dx >= 0)
     y_inc = 1;
  else
     y_inc = -1;

  for (k = 0; k < abs(dx); k++) {
     if (pk < 0) {
        pk = pk + 2 * dy;                    //calculate next pk
                 //next pixel: (x+1, y )
     }
     else {
        //next pixel: (x+1, y(+,-)1)
        pk = pk + 2 * dy - 2 * dx;           //calculate next pk
        y = y + y_inc;

     }
     x++;
     glBegin(GL_POINTS);
     glVertex2i(x, y);
     glEnd();
  }

  glFlush();
}
void init(void)
{
  glClearColor(1.0, 1.0, 1.0, 0.0);
```

```
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 500.0, 0.0, 500.0);
}

int main(int argc, char** argv) {
    std::cout << "Enter the value of x1 : ";
    std::cin >> x1;
    std::cout << "Enter the value of y_1 : ";
    std::cin >> y_1;
    std::cout << "Enter the value of x2 : ";
    std::cin >> x2;
    std::cout << "Enter the value of y2 : ";
    std::cin >> y2;

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Bresenham Line");
    glutDisplayFunc(display);
    init();
    glutMainLoop();
    return 0;
}
```

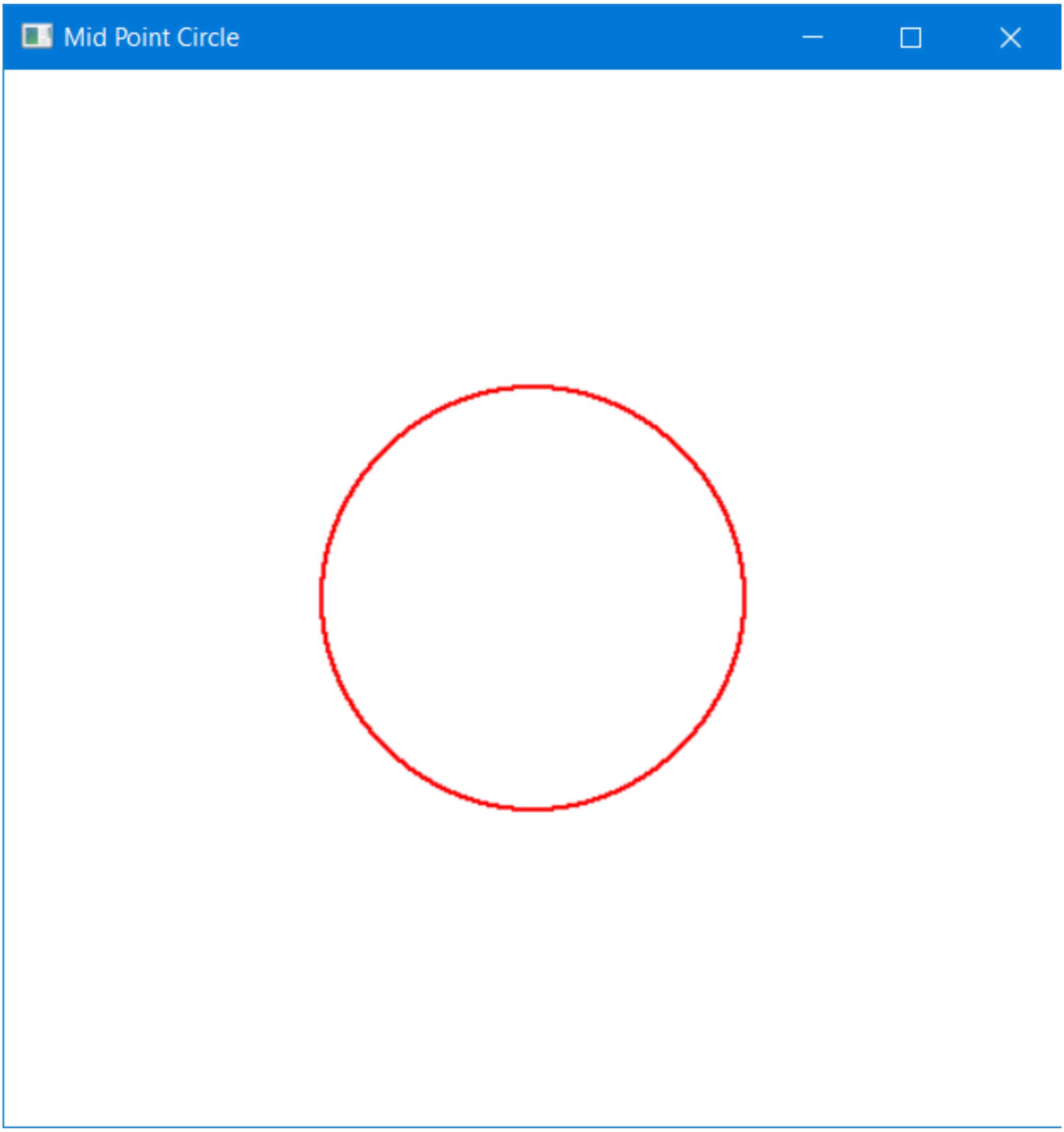


```
Enter the value of x1 : 10
Enter the value of y_1 : 20
Enter the value of x2 : 50
Enter the value of y2 : 60
```


Bresenham Line

8. Draw a circle using midpoint algorithm

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


void circle() { glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0,0.0, 0.0);
//glPointSize(2.0); float r =

100;float x = 0, y = r; float p =

1 - r;


while (x <= y)
{ x
++;
if (p < 0) {
p += 2 * (x + 1) + 1;
}
else
{ y--;
p += 2 * (x + 1) + 1 - 2 * (y - 1);
}
glBegin(GL_POINTS); glVertex2i(x, y); glVertex2i(-x,
y);glVertex2i(x, -y); glVertex2i(-x, -y);


glVertex2i(y, x); glVertex2i(-y, x); glVertex2i(y, -x); glVertex2i(-y,
-x); glEnd();
}


glFlush();
}

void myinit(){ glClearColor(1.0, 1.0, 1.0, 1.0);

glMatrixMode(GL_PROJECTION); gluOrtho2D(-250, 250, -250,
250);


}

int main(int argc, char ** argv) { glutInit(&argc, argv);

glutInitDisplayMode(GLUT_SINGLE |
GLUT_RGB);glutInitWindowSize(500, 500);

glutInitWindowPosition(0, 0); glutCreateWindow("Mid
PointCircle"); glutDisplayFunc(circle);

myinit(); glutMainLoop(); return 0;

}
```

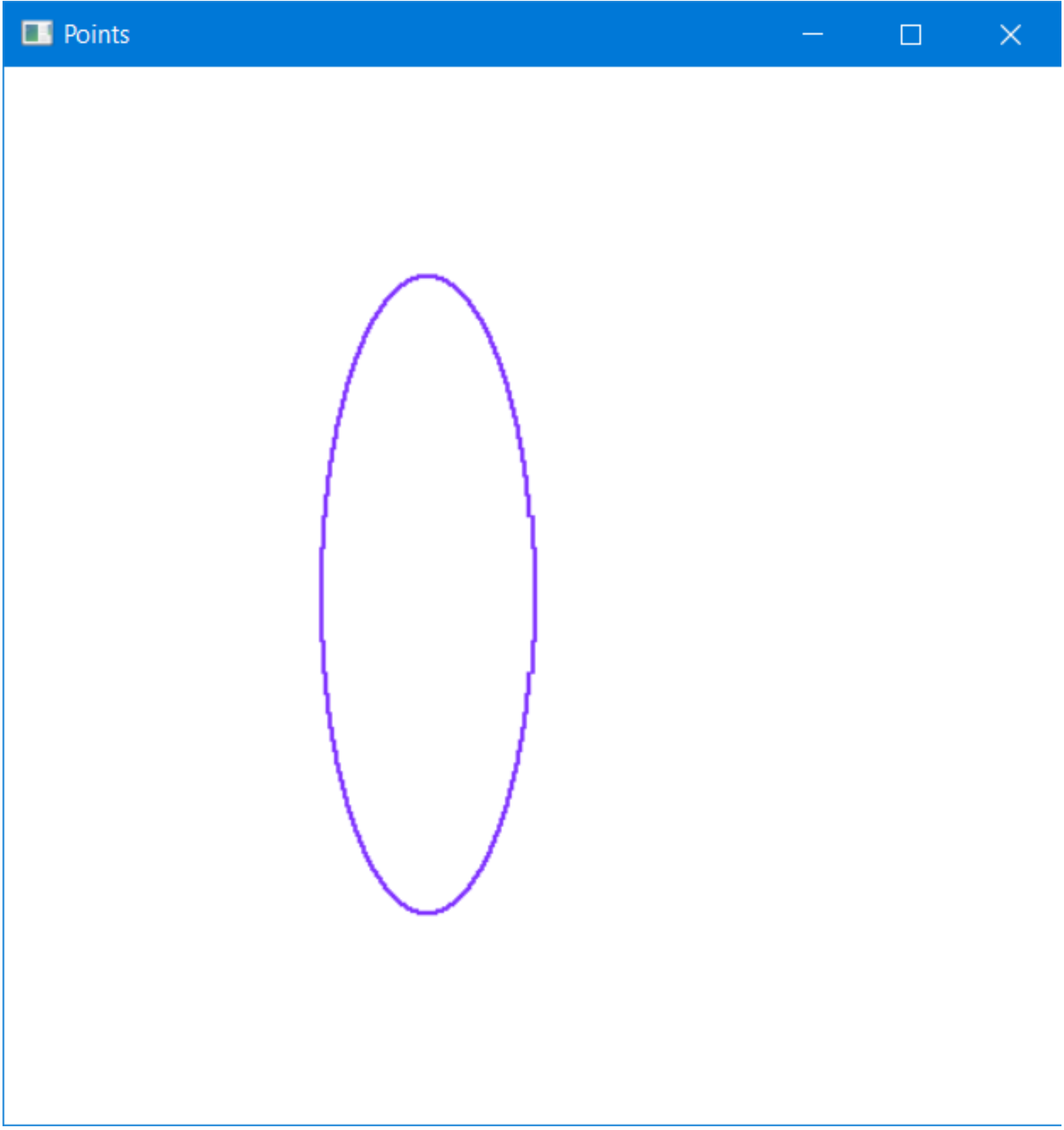


Mid Point Circle

9. Draw an ellipse midpoint ellipse algorithm

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

void ellipse(int rx, int ry, int xc, int yc)
{
        float dx, dy, d1, d2, x, y;
        x = 0;
        y = ry;

        d1 = (ry * ry) - (rx * rx * ry) + (0.25 * rx * rx);
        dx = 2 * ry * ry * x;
        dy = 2 * rx * rx * y;

        while (dx < dy)
        {

                glVertex2f(x + xc, y + yc);
                glVertex2f(-x + xc, y + yc);
                glVertex2f(x + xc, -y + yc);
                glVertex2f(-x + xc, -y + yc);


                if (d1 < 0) {
                        x++;
                        dx = dx + (2 * ry * ry);
                        d1 = d1 + dx + (ry * ry);
                }
                else {
                        x++;
                        y--;
                        dx = dx + (2 * ry * ry);
                        dy = dy - (2 * rx * rx);
                        d1 = d1 + dx - dy + (ry * ry);
                }
        }

        d2 = ((ry * ry) * ((x + 0.5) * (x + 0.5))) + ((rx * rx) * ((y - 1) * (y - 1))) - (rx * rx * ry * ry);

        while (y >= 0) {

                glVertex2f(x+xc, y+yc);
        glVertex2f(-x + xc, y + yc);
        glVertex2f(x + xc, -y + yc);
        glVertex2f(-x + xc, -y + yc);

        if (d2 > 0) {
            y--;
            dy = dy - (2 * rx * rx);
            d2 = d2 + (rx * rx) - dy;
        }
        else {
            y--;
            x++;
            dx = dx + (2 * ry * ry);
            dy = dy - (2 * rx * rx);
            d2 = d2 + dx - dy + (rx * rx);
        }
    }
}
void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.5, 0.2, 1.0);
    glBegin(GL_POINTS);


    ellipse(50, 150, 200, 250);

    glEnd();
    glFlush();
}

void myinit() {
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glColor3f(1.0, 1.0, 1.0);
    glPointSize(2.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 499.0, 0.0, 499.0);
}

void main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Points");
    glutDisplayFunc(display);

    myinit();
```

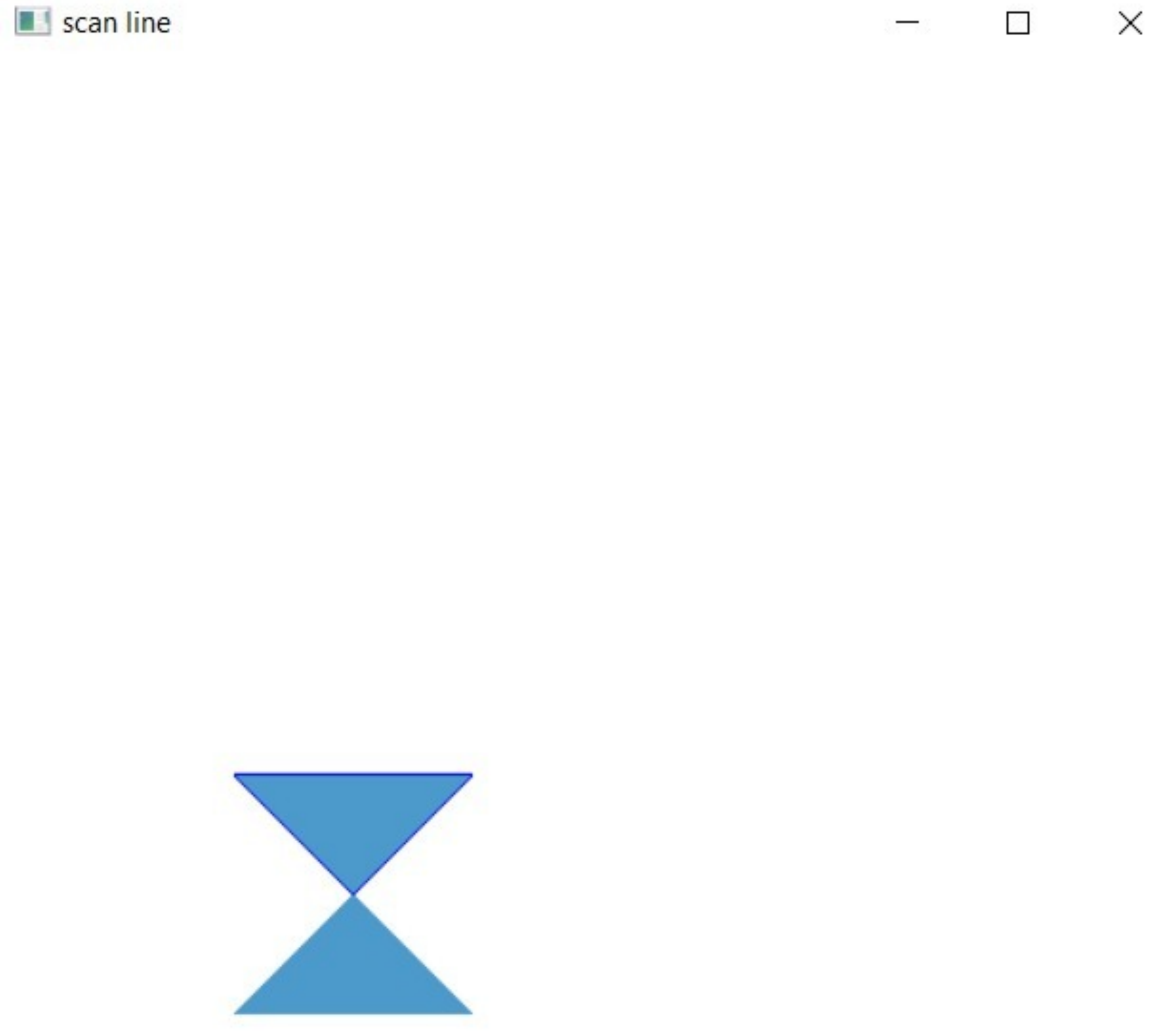
```
        glutMainLoop();
}
```



10. Write a program to fill a polygon using scan line fill algo

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

#include "math.h"
#define DEG2RAD 3.14159/180.0


int le[500], re[500], flag = 0, m;

void edge(int x0, int y0, int x1, int y1)
{
        if (y1 < y0) {
                int tmp;
                tmp = y1;
                y1 = y0;
                y0 = tmp;
                tmp = x1;
                x1 = x0;
                x0 = tmp;
        }

        int x = x0;

        m = (y1 - y0) / (x1 - x0);
        for (int i = y0; i < y1; i++) {
                if (x < le[i])
                        le[i] = x;
                if (x > re[i])
                        re[i] = x;
                x += (1 / m);
        }
}

void display() {


        glClearColor(1, 1, 1, 1);
        glClear(GL_COLOR_BUFFER_BIT);


        glColor3f(0, 0, 0);
        glBegin(GL_LINE_LOOP);

        glVertex2f(100, 200);
        glVertex2f(200, 100);
        glVertex2f(100, 100);
        glVertex2f(200, 200);

        glEnd();
        glFlush();

        for (int i = 0; i < 200; i++) {
                le[i] = 250;
                re[i] = 0;
        }
        edge(100, 200, 200, 100);
        edge(200, 100, 100, 100);
        edge(100, 100, 200, 200);
        edge(200, 200, 100, 200);

        flag = 1;
        if (flag == 1)
        {
                for (int i = 0; i < 200; i++)
                {
                        if (le[i] < re[i])
                        {
                                for (int j = le[i]; j < re[i]; j++)
                                {
                                        glColor3f(0.3, 0.6, 0.8);
                                        glBegin(GL_POINTS);
                                        glVertex2f(j, i);
                                        glEnd();

                                }

                        }
                }
        }
        glFlush();



}

void myinit() {
        glClearColor(1.0, 1.0, 1.0, 1.0);
        glColor3f(1.0, 1.0, 1.0);
        glPointSize(2.0);
        glMatrixMode(GL_RGB);
        glLoadIdentity();
        gluOrtho2D(0.0, 499.0, 0.0, 499.0);
}
```

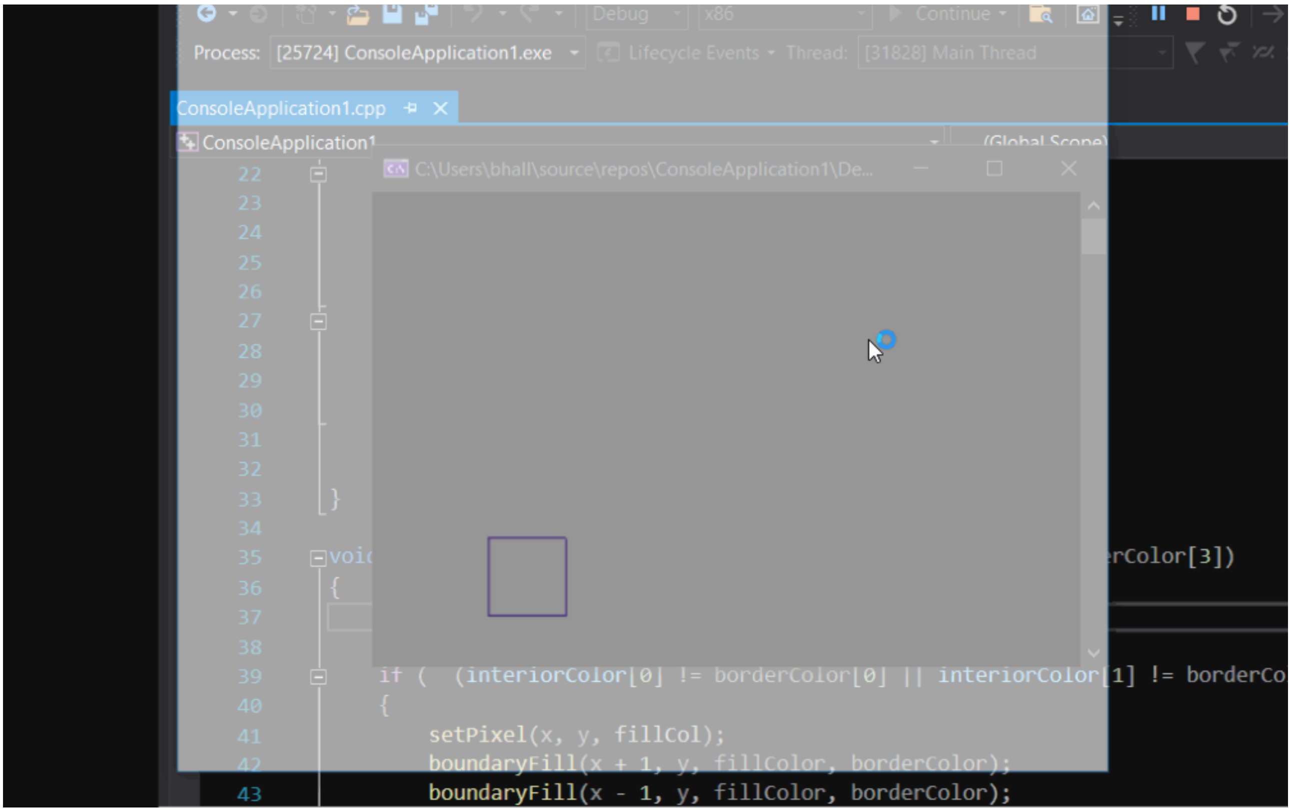
```
void main(int argc, char** argv) {

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(1000, 800);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Points");
    glutDisplayFunc(display);



    myinit();
    glutMainLoop();
}
```



11. Write a program to fill a polygon using boundary fill

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

int ww = 600, wh = 500;
float fillCol[3] = { 0.2,0.1,0.4 };
float borderCol[3] = { 0.4,0.2,0.8 };
void setPixel(int pointx, int pointy, float f[3])
{
        glBegin(GL_POINTS);
        glColor3fv(f);
        glVertex2f(pointx, pointy);
        glEnd();
        glFlush();

}
void getPixel(int x, int y, float pixels[3])
{
        glReadPixels(x, y, 1.0, 1.0, GL_RGB, GL_FLOAT, pixels);
}
void drawPolygon()
{


        for (int i = 0; i < 50; i++)
        {
                glVertex2f(200, 100 + i);
                glVertex2f(250, 100 + i);
        }
        for (int i = 0; i < 50; i++)
        {
                glVertex2f(200 + i, 100);  glVertex2f(200 + i, 150);
        }
        glEnd();
        glFlush();
}

void boundaryFill(int x, int y, float fillColor[3], float borderColor[3])
{
        float interiorColor[3];
        getPixel(x, y, interiorColor);
        if (  (interiorColor[0] != borderColor[0] || interiorColor[1] != borderColor[1] ||
interiorColor[2] != borderColor[2]) && (interiorColor[0] != fillColor[0] && (interiorColor[1]) !=
fillColor[1] && (interiorColor[2]) != fillColor[2]))
        {
                setPixel(x, y, fillCol);
                boundaryFill(x + 1, y, fillColor, borderColor);
                boundaryFill(x - 1, y, fillColor, borderColor);
```

```
			boundaryFill(x, y + 1, fillColor, borderColor);
			boundaryFill(x, y - 1, fillColor, borderColor);
		}
		else
			return;

}

void display()
{
		glClearColor(1.0, 1.0, 1.0, 1.0);
		glClear(GL_COLOR_BUFFER_BIT);
		glColor3fv(borderCol);
		glBegin(GL_POINTS);


		drawPolygon();
		boundaryFill(210, 140, fillCol, borderCol);
		glEnd();
		glFlush();
}

void myinit()
{
		glViewport(0, 0, ww, wh);
		glMatrixMode(GL_PROJECTION);
		glLoadIdentity();
		gluOrtho2D(0.0, (GLdouble)ww, 0.0, (GLdouble)wh);
}
int main(int argc, char** argv)
{
		glutInit(&argc, argv);
		glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
		glutInitWindowSize(ww, wh);
		glutCreateWindow("Bountry-Fill-Recursive");
		glutDisplayFunc(display);
		myinit();

		glutMainLoop();
		return 0;
}
```



Debug
x86
Continue
Process: [25724] ConsoleApplication1.exe
Lifecycle Events
Thread: [31828] Main Thread
ConsoleApplication1.cpp
ConsoleApplication1
(Global Scope)
C:\Users\bhall\source\repos\ConsoleApplication1\De...

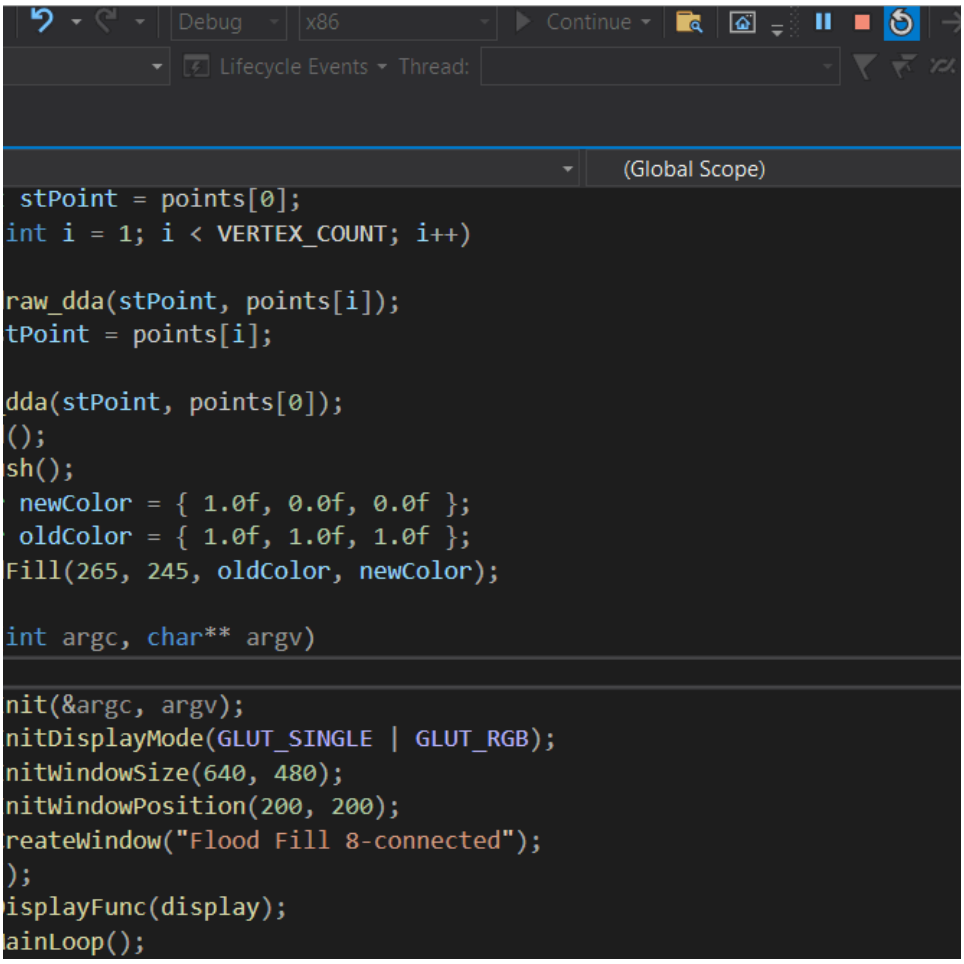
```
}

void                                                    erColor[3])
{


    if (  (interiorColor[0] != borderColor[0] || interiorColor[1] != borderCo
    {
        setPixel(x, y, fillCol);
        boundaryFill(x + 1, y, fillColor, borderColor);
        boundaryFill(x - 1, y, fillColor, borderColor);
```

Boundary Fill 8 Connected

```
#include <cmath>
#include <gl/glut.h>
struct Point
{
   GLint x;
   GLint y;
};
struct Color
{
   GLfloat r;
   GLfloat g;
   GLfloat b;
};
void init()
{
   glClearColor(1.0, 1.0, 1.0, 0.0);
   glColor3f(0.0f, 0.0f, 0.0f);
   gluOrtho2D(0.0, 640.0, 0.0, 480.0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glPointSize(1.0f);
}
void draw_dda(Point p1, Point p2)
{
   GLfloat dx = p2.x - p1.x;
   GLfloat dy = p2.y - p1.y;
   GLfloat x1 = p1.x;
   GLfloat y1 = p1.y;
   GLfloat step = 0;
   if (abs(dx) > abs(dy))
   {
      step = abs(dx);
   }
   else
   {
      step = abs(dy);
   }
   GLfloat xInc = dx / step;
   GLfloat yInc = dy / step;
   for (float i = 1; i <= step; i++)
   {
      glVertex2i(x1, y1);
      x1 += xInc;
      y1 += yInc;
   }
```

```
}
Color getPixelColor(GLint x, GLint y)
{
   Color color;
   glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, &color);
   return color;
}
void setPixelColor(GLint x, GLint y, Color color)
{
   glColor3f(color.r, color.g, color.b);
   glBegin(GL_POINTS);
   glVertex2i(x, y);
   glEnd();
   glFlush();
}
void floodFill(GLint x, GLint y, Color oldColor, Color newColor)
{
   Color color;
   color = getPixelColor(x, y);
   if (color.r == oldColor.r && color.g == oldColor.g && color.b == oldColor.b)
   {
      setPixelColor(x, y, newColor);
      floodFill(x + 1, y, oldColor, newColor);
      floodFill(x, y + 1, oldColor, newColor);
      floodFill(x - 1, y, oldColor, newColor);
      floodFill(x, y - 1, oldColor, newColor);
      floodFill(x + 1, y + 1, oldColor, newColor);
      floodFill(x + 1, y - 1, oldColor, newColor);
      floodFill(x - 1, y + 1, oldColor, newColor);
      floodFill(x - 1, y - 1, oldColor, newColor);
   }
   return;
}
#define VERTEX_COUNT 4
Point points[VERTEX_COUNT] = {
   250, 270,
   270, 270,
   270, 220,
   250, 220 };
void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_POINTS);
   Point stPoint = points[0];
   for (int i = 1; i < VERTEX_COUNT; i++)
   {
      draw_dda(stPoint, points[i]);
      stPoint = points[i];
```

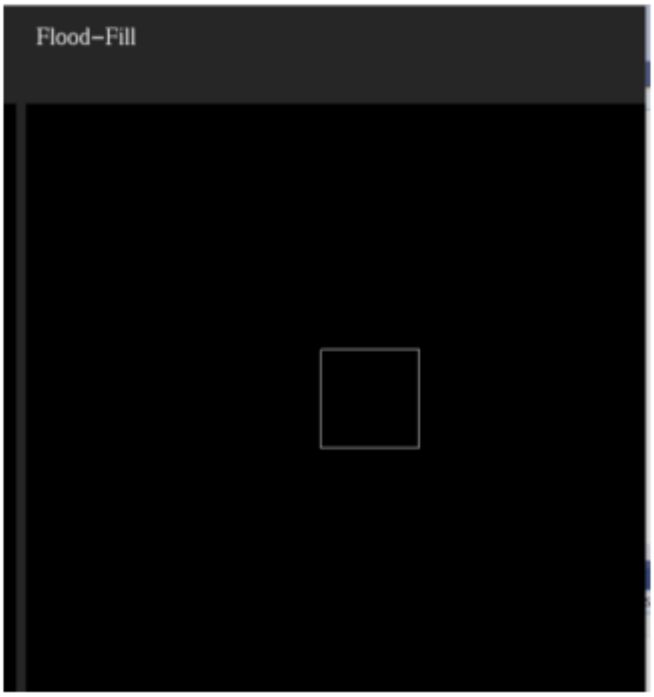
```
    }
    draw_dda(stPoint, points[0]);
    glEnd();
    glFlush();
    Color newColor = { 1.0f, 0.0f, 0.0f };
    Color oldColor = { 1.0f, 1.0f, 1.0f };
    floodFill(265, 245, oldColor, newColor);
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(200, 200);
    glutCreateWindow("Flood Fill 8-connected");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```



```
 stPoint = points[0];
int i = 1; i < VERTEX_COUNT; i++)

raw_dda(stPoint, points[i]);
tPoint = points[i];

dda(stPoint, points[0]);
();
sh();
 newColor = { 1.0f, 0.0f, 0.0f };
 oldColor = { 1.0f, 1.0f, 1.0f };
Fill(265, 245, oldColor, newColor);

int argc, char** argv)

nit(&argc, argv);
nitDisplayMode(GLUT_SINGLE | GLUT_RGB);
nitWindowSize(640, 480);
nitWindowPosition(200, 200);
reateWindow("Flood Fill 8-connected");
);
isplayFunc(display);
ainLoop();
```

12. Write a program to fill a polygon using flood fill

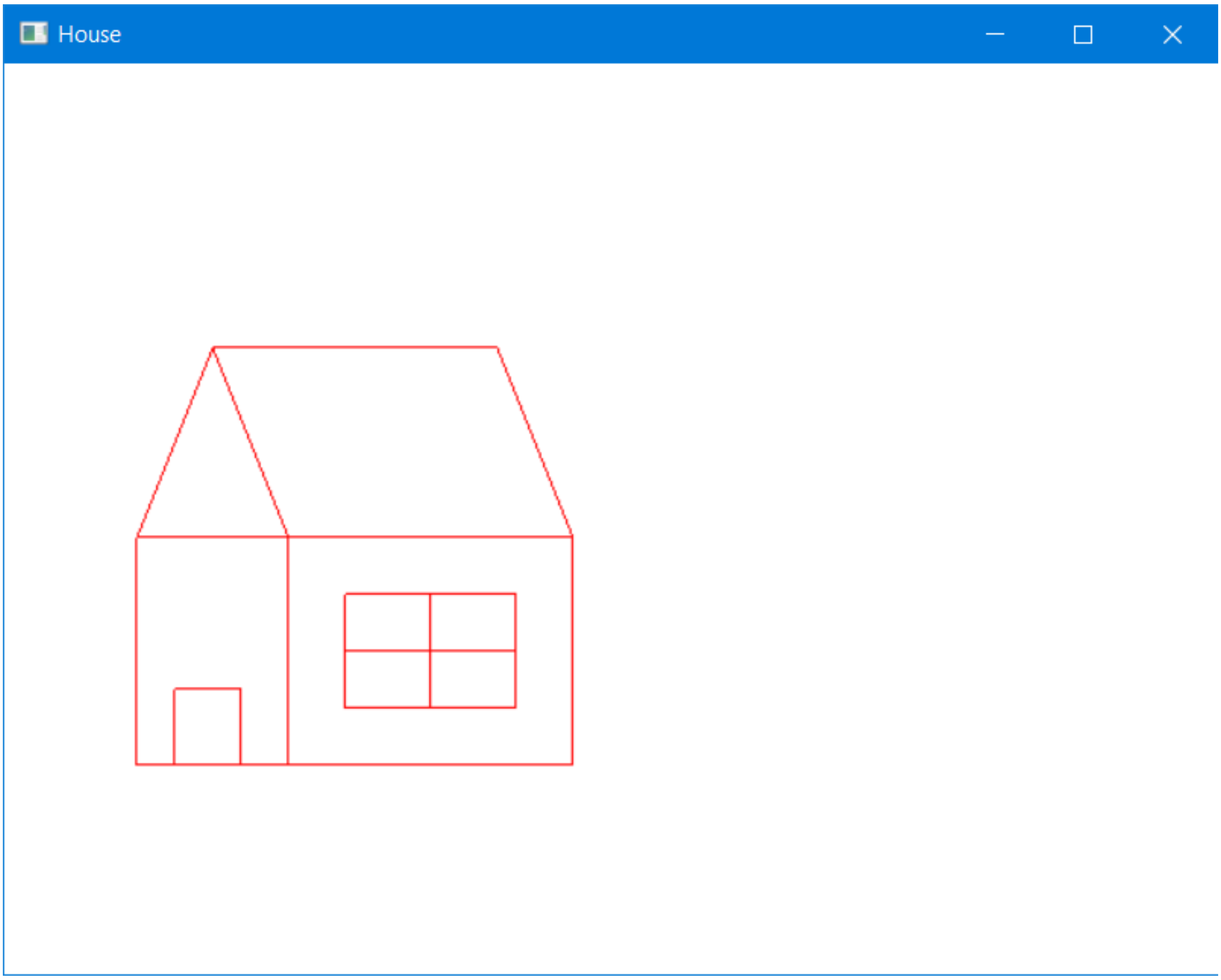
```
#include <stdio.h>
#include <math.h>
#include <GL/glut.h>
#include <cstring>
#include <string>
using namespace std;
int winIdMain;
int winIdSub, winIdSub2;
typedef struct pix
{
 float r, g, b;
}pix;
void printb(string c, int x, int y)
{
 glRasterPos2i(x, y);
 glColor3f(1.0, 1.0, 1.0);
 for (int i = 0; i < c.length(); i++)
 glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, c[i]);
 glFlush();
}
void floodDis()
{ // printb("Flood-Fill",680,670);
 glColor3f(1.0, 1.0, 1.0);
 glBegin(GL_LINE_LOOP);
 glVertex2i(300, 300);
 glVertex2i(300, 400);
 glVertex2i(400, 400);
 glVertex2i(400, 300);
 glEnd();
 glFlush();
}
void boundDis()
{ // printb("Boundary-Fill",10,670);
 glColor3f(1.0, 1.0, 1.0);
 glBegin(GL_LINE_LOOP); glVertex2i(300, 300);
 glVertex2i(300, 400);
 glVertex2i(400, 400);
 glVertex2i(400, 300);
 glEnd();
 glFlush();
}
void FloodFill(int a, int b, pix neww, pix old)
{
 pix tem;
 glReadPixels(a, b, 1, 1, GL_RGB, GL_FLOAT, &tem);
 printf("%f %f %f\n", tem.r, tem.g, tem.b);
 if ((tem.r == old.r) && (tem.g == old.g) && (tem.b == old.b))
 {
 glBegin(GL_POINTS);
 glColor3f(neww.r, neww.g, neww.b);
 glVertex2i(a, b);
 glEnd();
 glFlush();
 FloodFill(a + 1, b, neww, old);
 FloodFill(a - 1, b, neww, old);
 FloodFill(a, b + 1, neww, old);
 FloodFill(a, b - 1, neww, old);
 }
}
void boundFill(int a, int b, pix fil, pix boun)
{
 pix tem;
 glReadPixels(a, b, 1, 1, GL_RGB, GL_FLOAT, &tem);
 //printf("%f %f %f\n",tem.r,tem.g,tem.b);
 if ((tem.r != boun.r) && (tem.g != boun.g) && (tem.b != boun.b) && (tem.r !=
fil.r) && (tem.g != fil.g) && (tem.b != fil.b))
 {
 glBegin(GL_POINTS);
 glColor3f(fil.r, fil.g, fil.b);
 glVertex2i(a, b);
 glEnd();
 glFlush();
 boundFill(a + 1, b, fil, boun);
 boundFill(a - 1, b, fil, boun);
 boundFill(a, b + 1, fil, boun);
 boundFill(a, b - 1, fil, boun);
 }
}
void floo(int button, int state, int x, int y)
{
 printf("fl %d %d\n", x, y);
 pix fi, bo;
 fi.r = 0.1;
 fi.g = 0.1;
 fi.b = 1.0; bo.r = 0.0;
 bo.g = 0.0;
 bo.b = 0.0;
 int xi = x;
 int yi = (660 - y);
 FloodFill(xi, yi, fi, bo);
}
void boun(int button, int state, int x, int y)
{
 printf("bo %d %d\n", x, y);
 pix fi, bo;
 fi.r = 0.8;
 fi.g = 0.001;
 fi.b = 0.8;
 bo.r = 1.0;
 bo.g = 1.0;
 bo.b = 1.0;
 /* if((x>=300 && x<=400) && (y>=300 && y <=400))
 {
 glClear(GL_COLOR_BUFFER_BIT);
 printb("Invalid",200,150);
 }
 else{ */
 int xi = x;
 int yi = (660 - y);
 boundFill(xi, yi, fi, bo);
 // }
}
void clear() {
 glClear(GL_COLOR_BUFFER_BIT);
 glFlush();
}
void mainDis()
{
 glutSetWindow(winIdMain);
 printb("Boundary-Fill", 10, 660);
 printb("Flood-Fill", 680, 660);
}
void init()
{
 glClearColor(0.15, 0.15, 0.15, 1);
 glColor3f(0.5, 0.5, 0.5);
 gluOrtho2D(0, 1300, 0, 700);
 glViewport(0, 0, 1300, 700);
}
int main(int argc, char** argv)
{
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
 glutInitWindowPosition(0, 0); glutInitWindowSize(1300, 700);
 winIdMain = glutCreateWindow("The OpenGL");
 init();
 clear();
 glutDisplayFunc(mainDis);
 winIdSub = glutCreateSubWindow(winIdMain, 10, 100, 650, 650);
 gluOrtho2D(0, 650, 0, 650);
 glClearColor(0.25, 0.25, 0.25, 1);
 glutDisplayFunc(boundDis);
 glutMouseFunc(boun);
 winIdSub2 = glutCreateSubWindow(winIdMain, 670, 100, 650, 650);
 gluOrtho2D(0, 650, 0, 650);
 glClearColor(0.25, 0.25, 0.25, 1);
 glutDisplayFunc(floodDis);
 glutMouseFunc(floo);
 glutMainLoop();
}
```



13. WAP for drawing: (i) House (ii) Car (iii) Fish (iv) Man

DRAW HOUSE

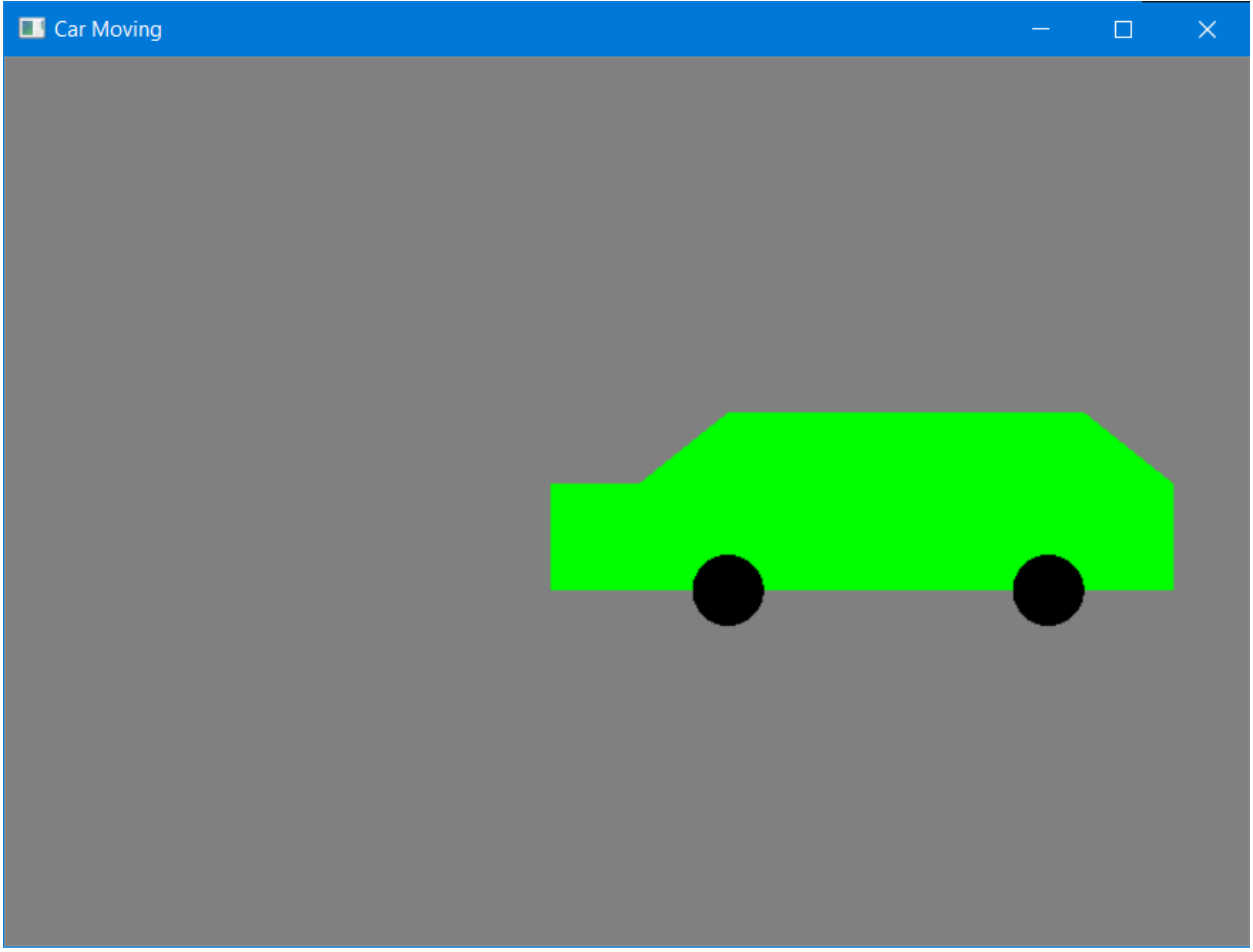
```
#include <windows.h>
#include<GL/glut.h>
void myInit(void)
{
glClearColor(1.0, 1.0, 1.0, 0.0); // set the bg color to a bright white
glColor3f(20.0f, 0.0f, 0.0f); // set the drawing color to black
glPointSize(10.0); //set the point size to 4 by 4 pixels
glMatrixMode(GL_PROJECTION);// set up appropriate matrices- to be explained
glLoadIdentity();// to be explained
gluOrtho2D(0.0, 640.0, 0.0, 480.0);// to be explained
}
void myDisplay(void)
{
glClear(GL_COLOR_BUFFER_BIT);
int y = 100;
glBegin(GL_LINE_LOOP);
glVertex2i(70, 10 + y);
glVertex2i(70, 130 + y);
glVertex2i(150, 130 + y);
glVertex2i(150, 10 + y);
glEnd();
glBegin(GL_LINE_LOOP);
glVertex2i(70, 130 + y);
glVertex2i(110, 230 + y);
glVertex2i(150, 130 + y);
glEnd();
glBegin(GL_LINE_LOOP);
glVertex2i(150, 130 + y);
glVertex2i(300, 130 + y);
glVertex2i(300, 10 + y);
glVertex2i(150, 10 + y);glEnd();
glBegin(GL_LINE_LOOP);
glVertex2i(110, 230 + y);
glVertex2i(260, 230 + y);
glVertex2i(300, 130 + y);
glVertex2i(70, 130 + y);
glEnd();
glBegin(GL_LINE_LOOP);
glVertex2i(180, 40 + y);
glVertex2i(180, 100 + y);
glVertex2i(270, 100 + y);
glVertex2i(270, 40 + y);
glEnd();
glBegin(GL_LINE_LOOP);
glVertex2i(225, 100 + y);
glVertex2i(225, 40 + y);
glEnd();
glBegin(GL_LINE_LOOP);
glVertex2i(180, 70 + y);
glVertex2i(270, 70 + y);
glEnd();
glBegin(GL_LINE_LOOP);
glVertex2i(90, 10 + y);
glVertex2i(90, 50 + y);
glVertex2i(125, 50 + y);
glVertex2i(125, 10 + y);
glEnd();
glFlush();
}
void main(int argc, char** argv)
{
glutInit(&argc, argv); // initialize the toolkit
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); // set the display mode
glutInitWindowSize(640, 480); // set the window size
glutInitWindowPosition(100, 150); // set the window position on the screen
glutCreateWindow("House"); // open the screen window(with its exciting title)
glutDisplayFunc(myDisplay); // register the redraw function
myInit();
glutMainLoop(); // go into a perpetual loop
}
```



Draw Car

```
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif
#include <stdlib.h>
#include <math.h>
GLint b = 300;
float counter = 600.0;
void initOpenGl()
{
 glClearColor(0.5, 0.5, 0.5, 1); //Background Color
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 gluOrtho2D(0, 700, 0, 500);
 glMatrixMode(GL_MODELVIEW);
}void wheel(int x, int y)
{
 float th;
 glBegin(GL_POLYGON);
 glColor3f(0, 0, 0);
 for (int i = 0; i < 360; i++)
 {
 th = i * (3.1416 / 180);
 glVertex2f(x + 20 * cos(th), y + 20 * sin(th));
 }
 glEnd();
}
void car()
{
 //Bottom Part
 glLoadIdentity();
 counter = counter - 0.05;
 glTranslated(counter, 100, 0.0);
 //glScaled(0.1,0.1,0.0);
 glBegin(GL_POLYGON);
 glVertex2f(100, 100);
 glVertex2f(100, 160);
 glVertex2f(450, 160);
 glVertex2f(450, 100);
 //Top Part
 glBegin(GL_POLYGON);
 glVertex2f(150, 160);
 glVertex2f(200, 200);
 glVertex2f(400, 200);
 glVertex2f(450, 160);
 glEnd();
 wheel(200, 100);
 wheel(380, 100);
}
void display()
{
 glClear(GL_COLOR_BUFFER_BIT);
 //Push and pop matrix for separating circle object from Background
 glColor3f(0.0, 1.0, 0.0);
 car();
 glutSwapBuffers();
 glFlush();
}
int main(int argc, char** argv)
{
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
 glutInitWindowSize(700, 500);
 glutInitWindowPosition(0, 0);
 glutCreateWindow("Car Moving");
 initOpenGl();
 glutDisplayFunc(display);
```

```
glutIdleFunc(display); glutMainLoop();
return 0;}
```

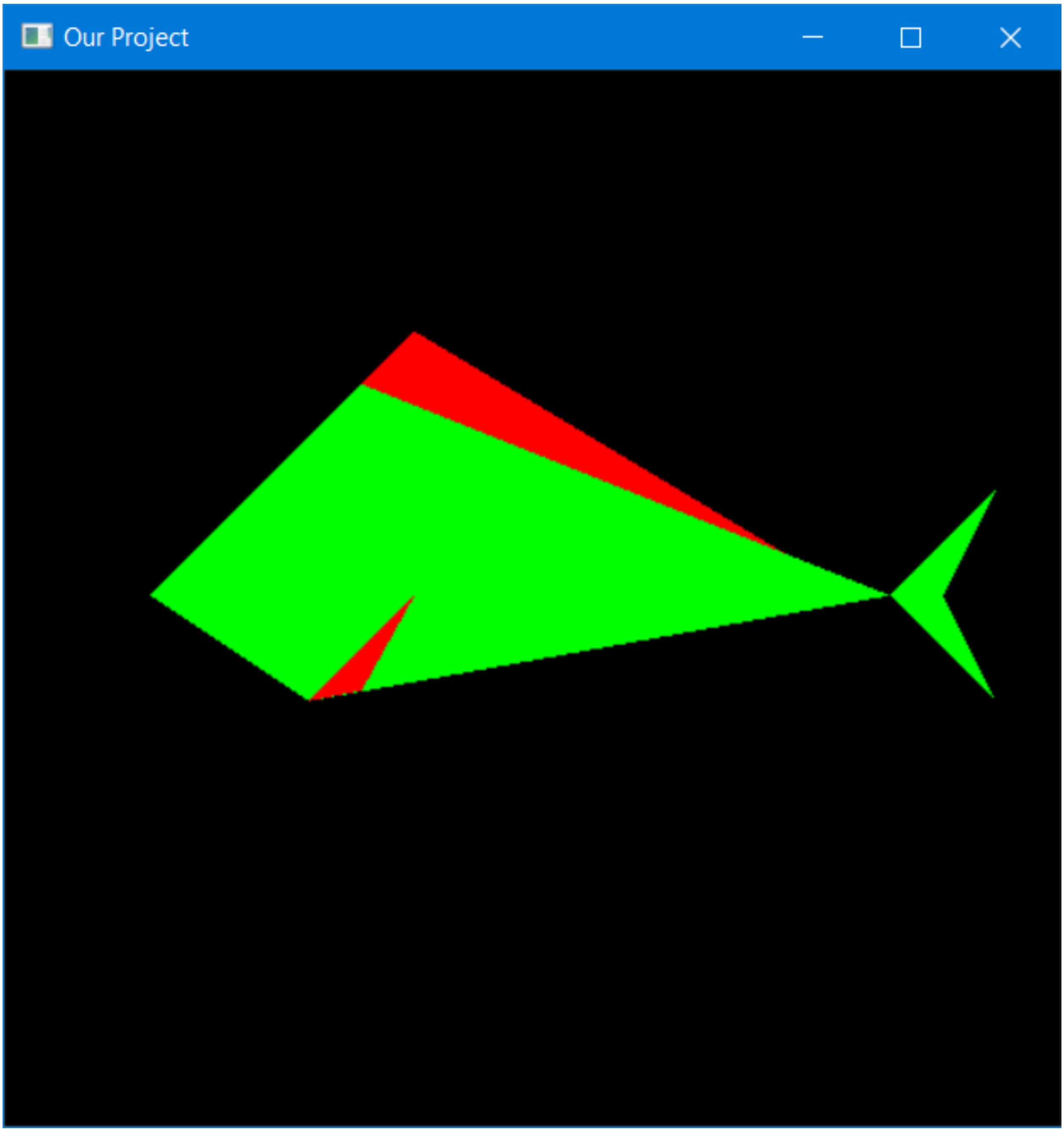


DRAW FISH

```
#include<GL/glut.h>
#include <stdio.h>
float xt=1.0,yt=1.0; // For interactive Keyboard
float x = 1.0,y = 1.0,z=1.0; // For Movement
float angle =0; // For Function animation
float Autorun = 300; // For Movement Autorun
void animation(void)
{
if(angle>=0 && angle<10)
angle = angle+0.5;
else angle = 0;
glutPostRedisplay();}
void Auto(void)
{
if(Autorun<=300 && Autorun>-350)
Autorun = Autorun-0.05;
else Autorun = 300;
glutPostRedisplay();
}
```

```
void settings(void)
{
glClearColor(0.0,0.0,0.0,0.0);
glPointSize(5.0);
glLineWidth(1.0);
glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
gluOrtho2D(0.0,400.0,0.0,400.0);
}
void Display(void)
{
glClear(GL_COLOR_BUFFER_BIT);
glPushMatrix();
glPushMatrix();
glRotatef(angle,0.0,0.0,0.0);
glTranslatef(Autorun,0.0,0.0);
glTranslatef(xt,yt,0.0); //For Move NEWS/QDZC each
Position.
glBegin(GL_POLYGON); // draw body
glColor3f(0.0,1.0,0.0);
glVertex2i(40,200);
glVertex2i(120,280);
glVertex2i(320,200);
glVertex2i(100,160);
glEnd();
glPushMatrix();
glRotatef(angle,0.0,0.0,0.0);
glBegin(GL_POLYGON); //draw tail
glColor3f(0.0,1.0,0.0);
glVertex2i(320,200);
glVertex2i(360,240);
glVertex2i(340,200);
glVertex2i(360,160);
glVertex2i(320,200);
glEnd();
glBegin(GL_POLYGON); //draw Top Key
glColor3f(1.0,0.0,0.0);
glVertex2i(120,280);
glVertex2i(140,300);
glVertex2i(280,216);
glVertex2i(120,280);
glEnd();
glBegin(GL_POLYGON); //draw Buttom Key
glColor3f(1.0,0.0,0.0);
glVertex2i(100,160);
glVertex2i(140,200);
glVertex2i(120,164);
glVertex2i(100,160);
```

```
glEnd();
glPopMatrix();glPopMatrix();
glPopMatrix();
glutSwapBuffers();
glFlush();
}
void keyboard(GLubyte key, int x, int y) // For keyboard interactive
{
switch ( key )
{
case 'd':
xt += 2.0; // Move thuea la 2.0 picel step by
step
glColor3f(0.0,1.0,0.0);
glutPostRedisplay();
break;
case 'a':
xt -= 2.0;
glColor3f(1.0,0.0,0.0);
glutPostRedisplay();
break;
case 's':
yt -= 2.0;
glColor3f(0.0,0.0,0.0);
glutPostRedisplay();
break;
case 'w':
yt += 2.0;
glColor3f(0.0,0.0,1.0);
glutPostRedisplay();
break;
case 'e':
xt += 2.0;
yt += 2.0;
glColor3f(1.0,0.0,1.0);
glutPostRedisplay();
break;
case 'q':
xt -= 2.0;
yt += 2.0;
glColor3f(0.0,1.0,1.0);
glutPostRedisplay();
break;
case 'c':
xt += 2.0;
yt -= 2.0;
glColor3f(1.0,0.0,1.0);
glutPostRedisplay();
break;
case 'z':
xt -= 2.0;
yt -= 2.0;
glColor3f(0.0,0.0,1.0);
glutPostRedisplay();
break;
default:break;
}
}
void main(int a,char ** b)
{
glutInit(&a,b);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowPosition(200,50); //200 Lelt 50 Height
glutInitWindowSize(500,500);
glutCreateWindow("Our Project");
settings();
glutDisplayFunc(Display);
glutIdleFunc(animation);
glutIdleFunc(Auto);
glutKeyboardFunc(keyboard);
glutMainLoop();
}
```


Our Project

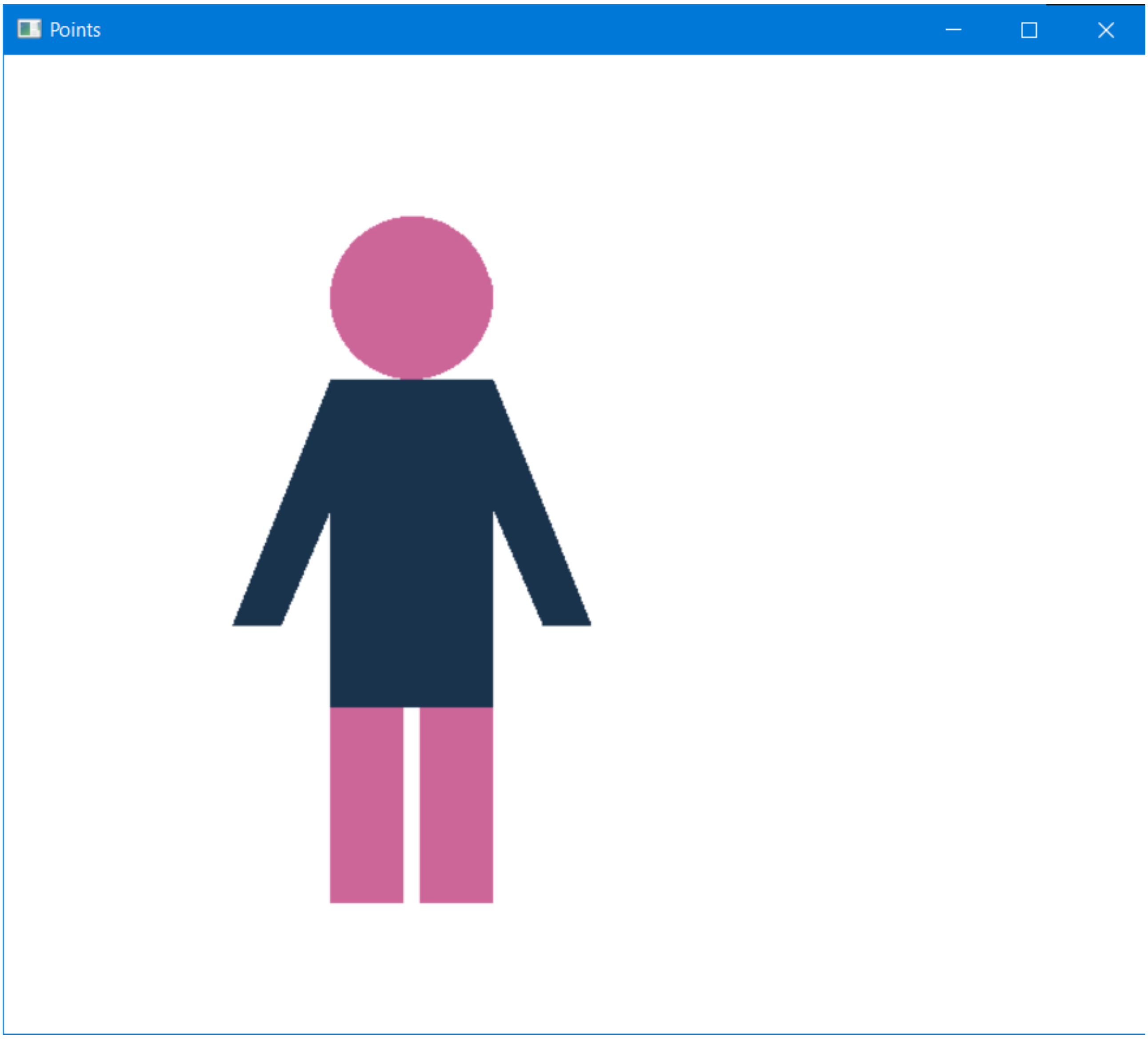
DRAW MAN

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


void draw(int x, int y, int xc, int yc) { glVertex2f(xc + x, yc + y);

        glVertex2f(xc + x, yc - y); glVertex2f(xc - x, yc - y); glVertex2f(xc

        -x, yc + y); glVertex2f(xc + y, yc + x); glVertex2f(xc - y, yc + x);

        glVertex2f(xc - y, yc - x); glVertex2f(xc + y, yc - x);

        }

void circle(int R, int xc, int yc) { glBegin(GL_POLYGON); int x = 0, y =

        R;int P = 3 - 2 * R; while (y >= x) {

        draw(x, y, xc, yc); if (P < 0)

        {P += 4 * x + 6;


        }

        else {

        P += 4 * (x - y) + 10;
        y--;

        } x++;

        }

        glEnd();

        }



void rect(int x1, int y1, int x2, int y2)

        { glBegin(GL_POLYGON); glVertex2f(x1, y1);

        glVertex2f(x2, y1); glVertex2f(x2, y2); glVertex2f(x1,

        y2);

        glEnd();

        }



void arm(int x1, int x2, int x3, int x4, int y1, int y2, int y3, int y4)

        {glBegin(GL_POLYGON);

        glVertex2f(x1,    y1);    glVertex2f(x2,    y2);    glVertex2f(x3,

        y3);glVertex2f(x4, y4); glEnd();

        }
```

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

//body
glColor3f(0.1, 0.2, 0.3);
rect(200, 200, 300, 400);
arm(300, 300, 330, 360, 400, 320, 250, 250);
arm(200, 200, 170, 140, 400, 320, 250, 250);
//legs
glColor3f(0.8, 0.4, 0.6);
rect(200, 80, 245, 200);
rect(255, 80, 300, 200);

//head and legs glColor3f(0.6, 0.5, 0.2);
circle(50, 250, 450);
```

```
        glFlush();

        }



        void myinit() {

        glClearColor(1.0, 1.0, 1.0, 1.0);

        glColor3f(1.0, 0.0, 0.0); glPointSize(5.0);

        glMatrixMode(GL_PROJECTION);

        glLoadIdentity();gluOrtho2D(0.0, 699.0, 0.0,

        599.0);

        }



void main(int argc, char** argv) { glutInit(&argc, argv);

        glutInitDisplayMode(GLUT_SINGLE        |

        GLUT_RGB);glutInitWindowSize(700, 600);

        glutInitWindowPosition(0,     0);

        glutCreateWindow("Points");glutDisplayFunc(display);
```

```
myinit(); glutMainLoop();

}
```



14. WAP to perform basic 2D transformation

Translation

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

float tx, ty, tz=0.0;

void draw_car(){
        //car boundary
        glBegin(GL_POLYGON
        );glVertex2f(100, 100);
        glVertex2f(450, 100);
        glVertex2f(450, 160);
        glVertex2f(400, 200);
        glVertex2f(200, 200);
        glVertex2f(150, 160);
        glVertex2f(100,
        150);glEnd();
//wheel 1
        glBegin(GL_POLYGON
        );glColor3f(0.0, 0.0,
        0.0); int x1=200, y=100,
        r=25;
        for(int i=0;i<360;i++){
                float th=i*3.142/180;
                glVertex2f(x1+r*cos(th),
                    y+r*sin(th));

          }

          glEnd();
//wheel 2

          glBegin(GL_POLYGON

          );glColor3f(0.0, 0.0,

          0.0); int x2=380;

          for(int i=0;i<360;i++){

                    float th=i*3.142/180;

                    glVertex2f(x2+r*cos(th),

                    y+r*sin(th));

          }

          glEnd();


}



void display()

{


 glClear(GL_COLOR_BUFFER_BIT

 );glColor3f(1,0,0);

 draw_car();


 //switch matrix mode to modelview in order to work with
objectcoordinates

 glMatrixMode(GL_MODELVIEW

 );glLoadIdentity();



 // in-built glTranslatef, set tz=0 for 2-d translation
 //glTranslatef(tx, ty, 0.0);
 // without in-built method, multiply modelview matrix with
translatematrix


 GLfloat translate[16]={1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, tx, ty, tz, 1};
//4x4 transformation matrix expressed in column-major order
 glMultMatrixf(translate);                  //multiplies matrix in
argument,with the current matrix and saves the result in current
matrix.
                                                    //Current matrix
here is modelview matrix


 glColor3f(0,0,1
 );draw_car();


 //switch from object coordinates to projection (camera)
 coordinatesglMatrixMode(GL_PROJECTION);
 glLoadIdentity(
 );glFlush();
}

void myInit()
{
 glClearColor(1,1,1,1);
 glColor3f(0,0,1);
 glMatrixMode(GL_PROJECTION
 );glLoadIdentity();
 gluOrtho2D(-100,700,-100, 400);
}

int main(int argc,char** argv)
{
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_SINGLE|
 GLUT_RGB);glutInitWindowSize(800,500);
 std::cout<<"Enter translation in x-axis
 tx";std::cin>>tx;
 std::cout<<"Enter translation in y-axis
 ty";std::cin>>ty;
 glutCreateWindow("2D
 Translation");
 glutDisplayFunc(display);
 myInit();
 glutMainLoop(
 );
}
```

Rotation

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

float a, x=0, y=0;

void draw_car(){
        //car boundary
        glBegin(GL_POLYGON
        );glVertex2f(100, 100);
        glVertex2f(450, 100);
        glVertex2f(450, 160);
```

```
          glVertex2f(400, 200);

          glVertex2f(200, 200);

          glVertex2f(150, 160);
          glVertex2f(100,

          150);glEnd();

//wheel 1

          glBegin(GL_POLYGON

          );glColor3f(0.0, 0.0,

          0.0); int x1=200, y=100,

          r=25;

          for(int i=0;i<360;i++){

                    float th=i*3.142/180;

                    glVertex2f(x1+r*cos(th),

                    y+r*sin(th));

          }

          glEnd();

//wheel 2

          glBegin(GL_POLYGON

          );glColor3f(0.0, 0.0,

          0.0); int x2=380;

          for(int i=0;i<360;i++){

                    float th=i*3.142/180;

                    glVertex2f(x2+r*cos(th),

                    y+r*sin(th));

          }

          glEnd();


}


void draw_triangle(){
        glBegin(GL_TRIANGLES

        );glVertex2i(100, 100);

        glVertex2i(100, 200);

        glVertex2i(200,

        150);glEnd();

}

void display()

{


 glClear(GL_COLOR_BUFFER_BIT

 );glColor3f(1,0,0);

 draw_car();    //or draw_triangle();


 //switch matrix mode to modelview in order to work with
objectcoordinates

 glMatrixMode(GL_MODELVIEW

 );glLoadIdentity();



 // in-built glTranslatef, set tz=0 for 2-d translation

 //glRotatef(a, 0, 0, 1);


 // without in-built method, multiply modelview matrix with
translatematrix


 GLfloat rotate[16]={cos(a), sin(a), 0, 0, -sin(a), cos(a), 0, 0, 0, 0, 1,
0, 0, 0, 0, 1};  //4x4 transformation matrix expressed
incolumn-major order


 //rotation around origin

// glMultMatrixf(rotate);                    //multiplies matrix in
argument,with the current matrix and saves the result in current
matrix.
```

```
                                                        //Current matrix
here is modelview matrix
//rotation around pt (x,y)

 GLfloat translate1[16]={1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, x, y, 0, 1};

 GLfloat translate2[16]={1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, -x, -y, 0, 1};

// First apply translate (-x, -y), then rotate, then

 translate(x,y)glMultMatrixf(translate2);

 glMultMatrixf(rotate);

 glMultMatrixf(translate1

 );


 glColor3f(0,0,1);

 draw_car(); //or draw_triangle();


 //switch from object co-ordinates to projection (camera)

 coordinatesglMatrixMode(GL_PROJECTION);

 glLoadIdentity(

 );glFlush();

}

void myInit()

{

 glClearColor(1,1,1,1);

 glColor3f(0,0,1);

 glMatrixMode(GL_PROJECTION

 );glLoadIdentity();

 gluOrtho2D(-200,600, -100, 700);

}

int main(int argc,char** argv)
```

```
{
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_SINGLE|
 GLUT_RGB);glutInitWindowSize(800,800);
 std::cout<<"Enter angle of rotation in
 degrees";std::cin>>a;
 std::cout<<"Enter x value of point about which to
 rotate";std::cin>>x;
 std::cout<<"Enter y value of point about which to rotate";
 std::cin>>y;
 glutCreateWindow("2D
 Rotation");
 glutDisplayFunc(display);
 myInit();
 glutMainLoop();
}
```

Scaling

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

#include <stdlib.h>

#include <math.h>

void init()

{

   glClearColor(1,
1, 1, 1.0);
```

```
   glPointSize(2.0);

glMatrixMode(GL
_PROJECTION);
   glLoadIdentity();

gluOrtho2D(-500,
500.0, -500.0,
500.0);
}
void
makeTriangle(float
x1, float y1, float
x2, float y2, float
x3, float y3)
{

glClear(GL_COLO
R_BUFFER_BIT);
   glColor3f(1.0,
0.0, 0.0);

glBegin(GL_POIN
TS);
   float m1 =
float(y2 - y1) / (x2
- x1);
   float m2 =
float(y3 - y2) / (x3
- x2);
   float m3 =
float(y1 - y3) / (x1
- x3);
   float c1 = y1 -
m1 * x1;
   float c2 = y2 -
m2 * x2;
   float c3 = y3 -
m3 * x3;
   int i = 0;
   while (x1 + i <=
x2)
   {
      glVertex2f(x1
+ i, m1 * (x1 + i) +
c1);
      i++;
   }
   i = 0;
   while (x2 + i <=
x3)
    {
        glVertex2f(x2
+ i, m2 * (x2 + i) +
c2);
        i++;
    }
    i = 0;
    while (x3 - i >=
x1)
    {
        glVertex2f(x3
- i, m3 * (x3 - i) +
c3);
        i++;
    }
}
void translate(float
x1, float y1, float
x2, float y2, float
x3, float y3, float
xp, float yp)
{
```

```
glClear(GL_COLO
R_BUFFER_BIT);

   x1 = x1 + xp;

   x2 = x2 + xp;

   x3 = x3 + xp;

   y1 = y1 + yp;

   y2 = y2 + yp;

   y3 = y3 + yp;


makeTriangle(x1,
y1, x2, y2, x3, y3);

}

void scaling(float
x1, float y1, float
x2, float y2, float
x3, float y3)

{

   translate(x1, y1,
x2, y2, x3, y3, -x1,
-y1);

   float new_x1 =
(x1 - x1) * 0.75;

   float new_y1 =
(y1 - y1) * 0.5;
```

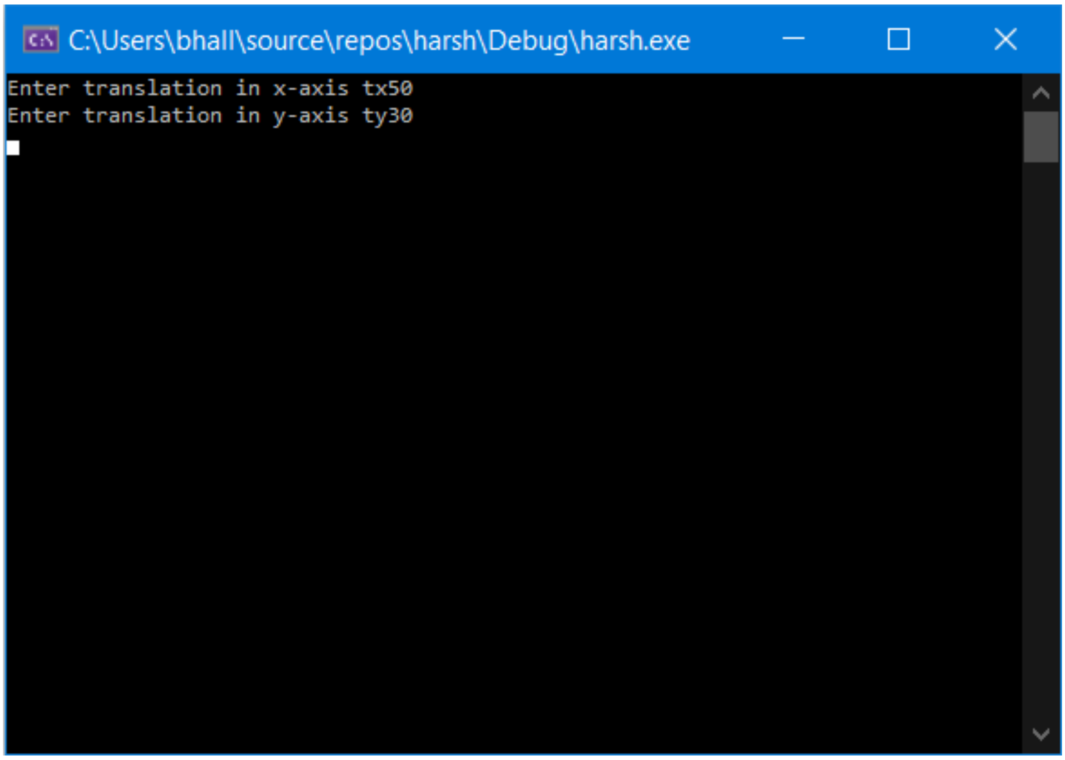
```
    float new_x2 =
(x2 - x1) * 0.75;
    float new_y2 =
(y2 - y1) * 0.5;
    float new_x3 =
(x3 - x1) * 0.75;
    float new_y3 =
(y3 - y1) * 0.5;

makeTriangle(new
_x1, new_y1,
new_x2, new_y2,
new_x3, new_y3);

translate(new_x1,
new_y1, new_x2,
new_y2, new_x3,
new_y3, 100, 100);
    glEnd();
    glColor3f(0, 0.0,
0.0);

glBegin(GL_LINE
S);
    glVertex2f(-500,
0);
```

```
   glVertex2f(500,
0);


glBegin(GL_LINE
S);
   glVertex2f(0,
-500);
   glVertex2f(0,
500);
   glEnd();
   glFlush();
}
void display()
{


makeTriangle(100,
100, 175.0, 175.0,
250.0, 100.0);
   scaling(100,
100, 175.0, 175.0,
250.0, 100.0);
}
int main(int argc,
char** argv)
```

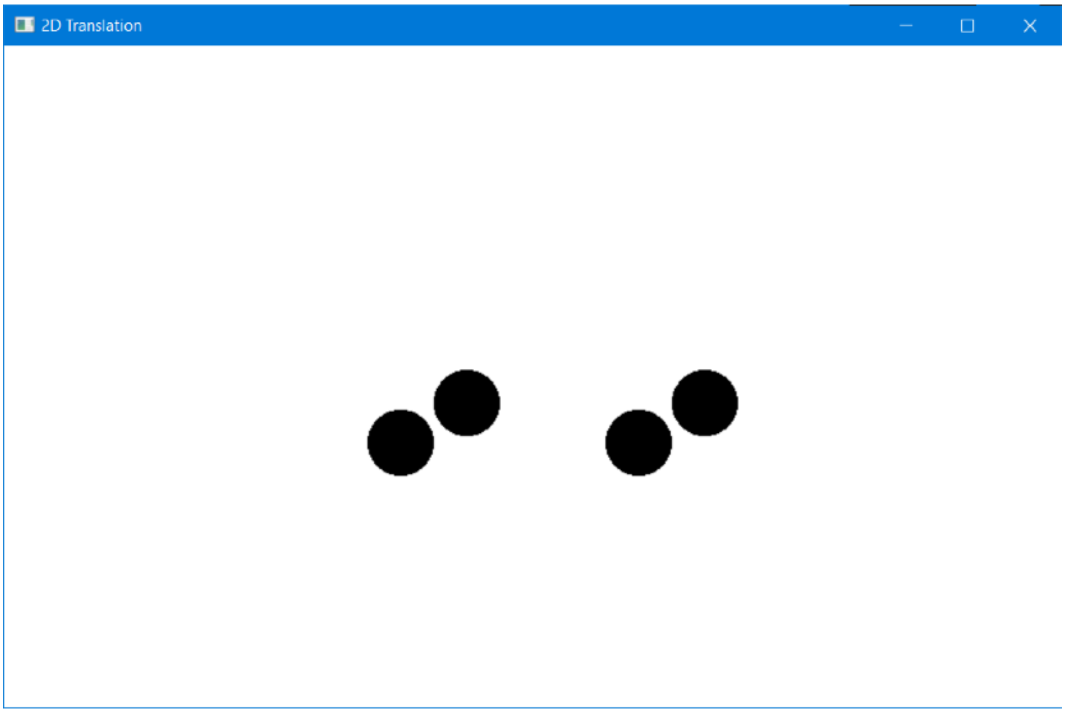
```
{
   glutInit(&argc,
argv);


glutInitDisplayMo
de(GLUT_SINGL
E | GLUT_RGB);


glutInitWindowPos
ition(100, 100);


glutInitWindowSiz
e(800, 600);


glutCreateWindow
("Scaling");
   init();


glutDisplayFunc(di
splay);
   glutMainLoop();
}
```

Output:

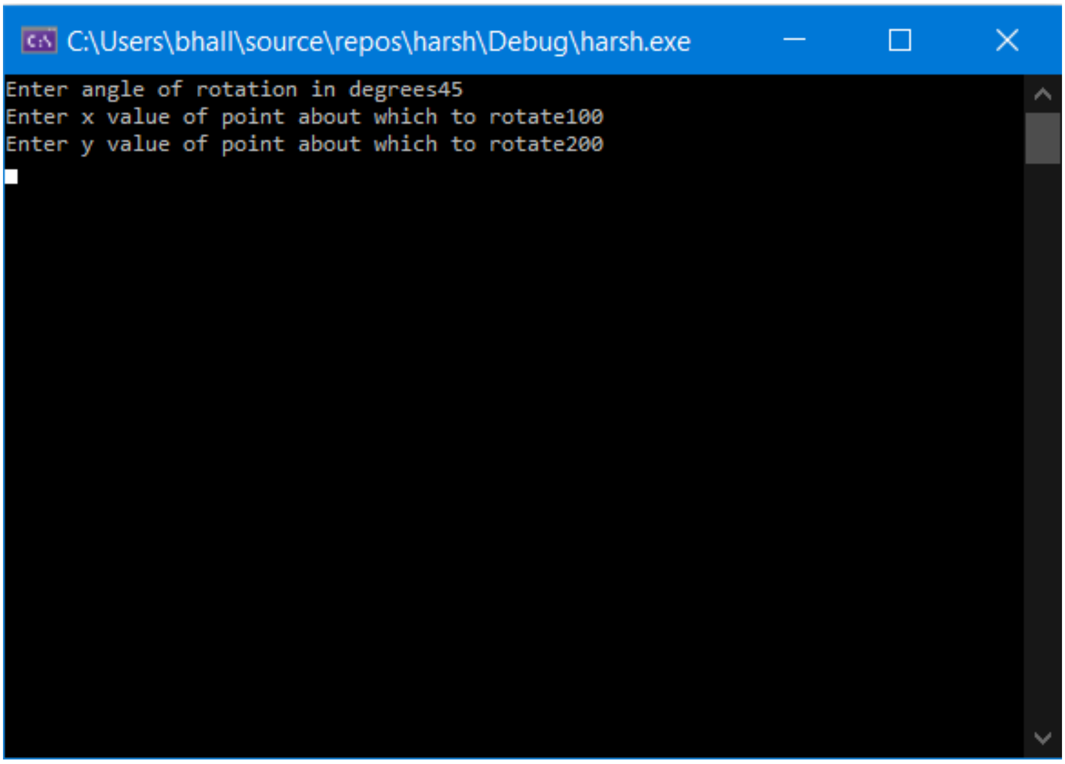
Translation:



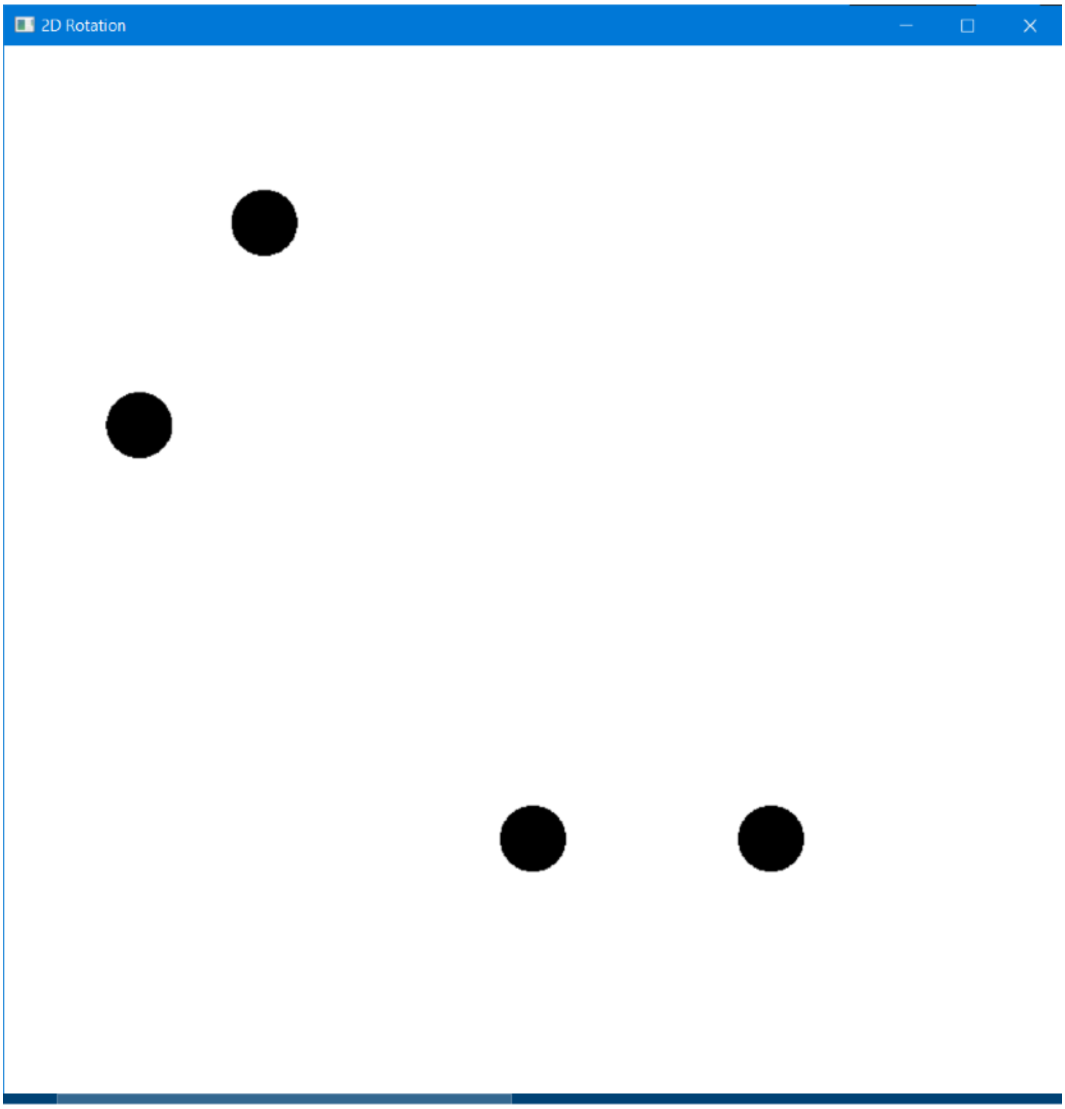
```
Enter translation in x-axis tx50
Enter translation in y-axis ty30
```



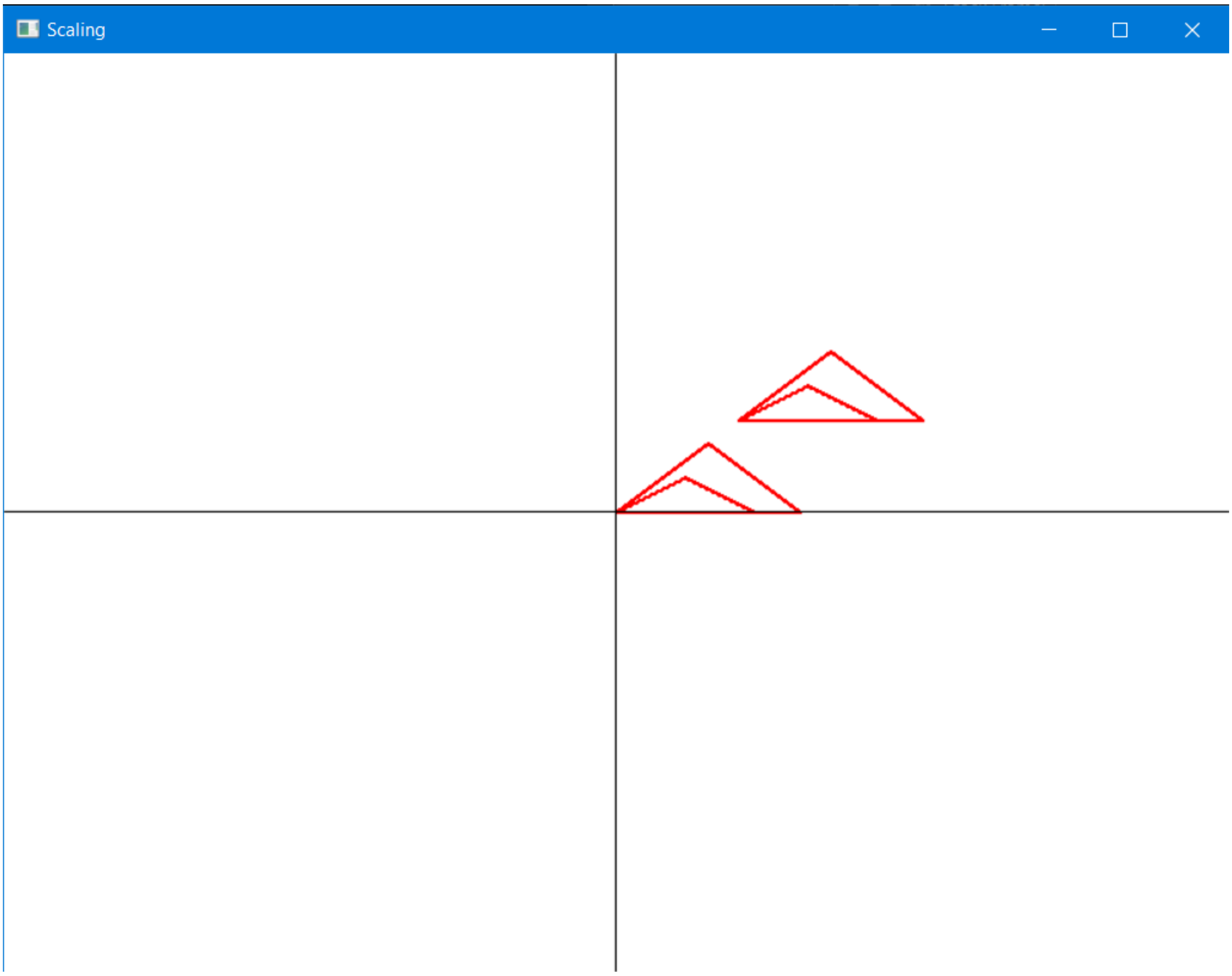
Rotation:



```
Enter angle of rotation in degrees45
Enter x value of point about which to rotate100
Enter y value of point about which to rotate200
```



Scaling:


Scaling

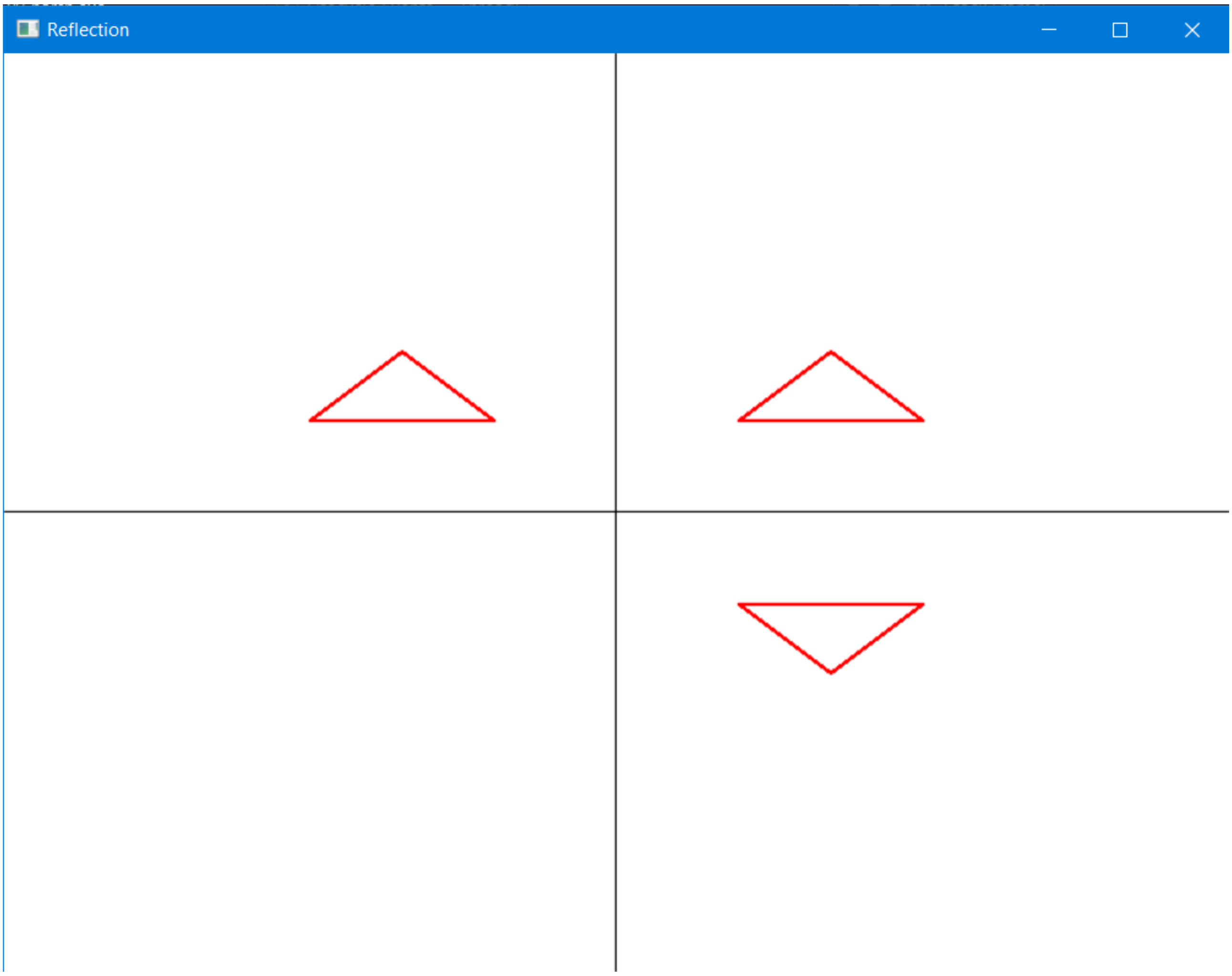
15 a. Reflection about x-axis, y-axis and a line y=x+2

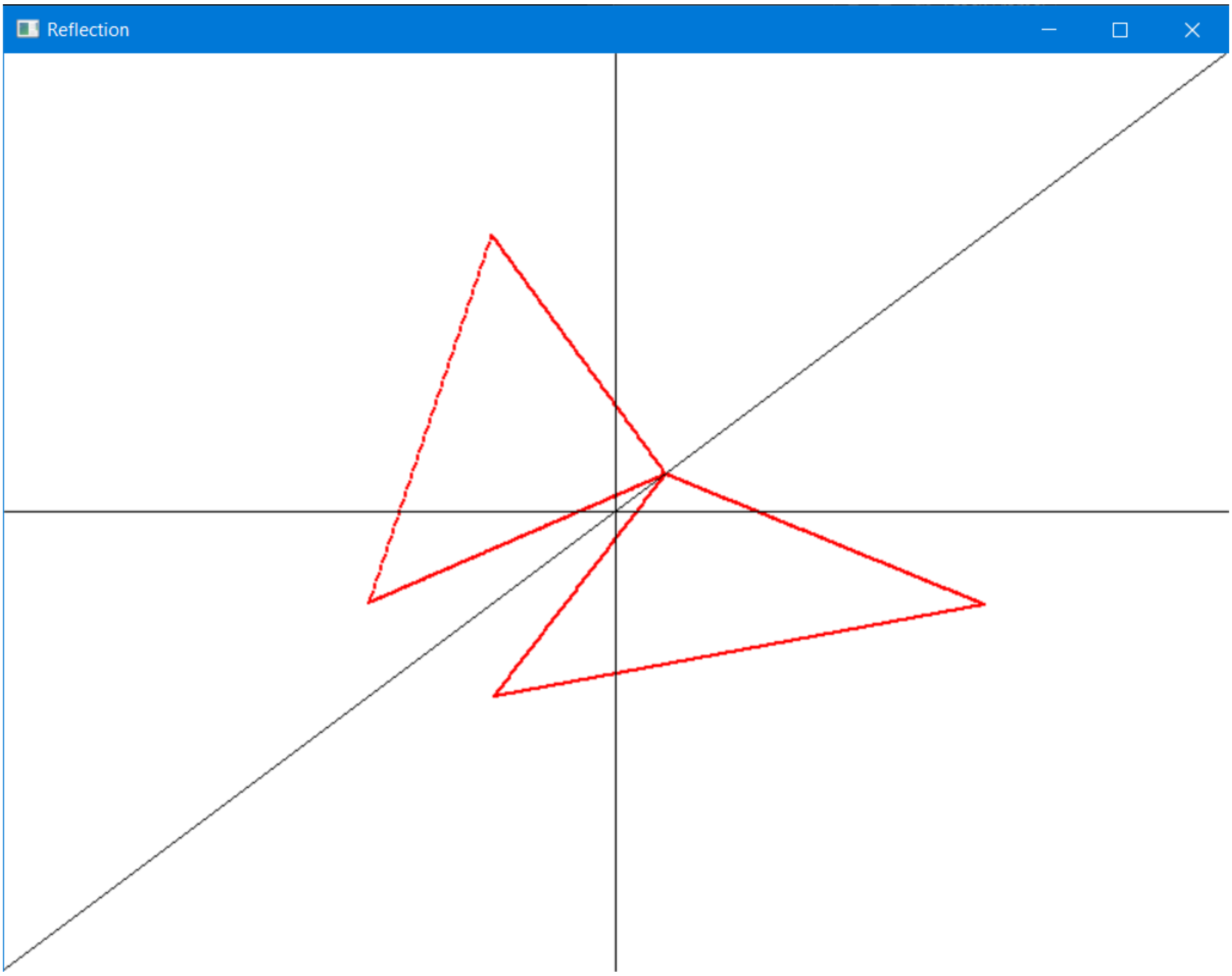
```
#include <GL\glut.h>

#include <stdlib.h>
#include <math.h>
void init()
{
   glClearColor(1, 1, 1, 1.0);
   glPointSize(2.0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(-500, 500.0, -500.0, 500.0);
}
void makeTriangle(float x1, float y1, float x2, float y2, float x3, float y3)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 0.0, 0.0);
   glBegin(GL_POINTS);
   float m1 = float(y2 - y1) / (x2 - x1);
   float m2 = float(y3 - y2) / (x3 - x2);
   float m3 = float(y1 - y3) / (x1 - x3);
   float c1 = y1 - m1 * x1;
   float c2 = y2 - m2 * x2;
   float c3 = y3 - m3 * x3;
   int i = 0;
   while (x1 + i <= x2)
   {
      glVertex2f(x1 + i, m1 * (x1 + i) + c1);
      i++;
   }
   i = 0;
   while (x2 + i <= x3)
   {
      glVertex2f(x2 + i, m2 * (x2 + i) + c2);
      i++;
   }
   i = 0;
   while (x3 - i >= x1)
   {
      glVertex2f(x3 - i, m3 * (x3 - i) + c3);
      i++;
   }
}
void display()
{
   makeTriangle(100, 100, 175.0, 175.0, 250.0, 100.0);
   makeTriangle(-250, 100, -175.0, 175.0, -100.0, 100.0);
   makeTriangle(100, -100, 175.0, -175.0, 250.0, -100.0);
```

```
    glEnd();
    glColor3f(0, 0.0, 0.0);
    glBegin(GL_LINES);
    glVertex2f(-500, 0);
    glVertex2f(500, 0);
    glBegin(GL_LINES);
    glVertex2f(0, -500);
    glVertex2f(0, 500);
    glEnd();
    glFlush();
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Reflection");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
}
```

About y=x+2

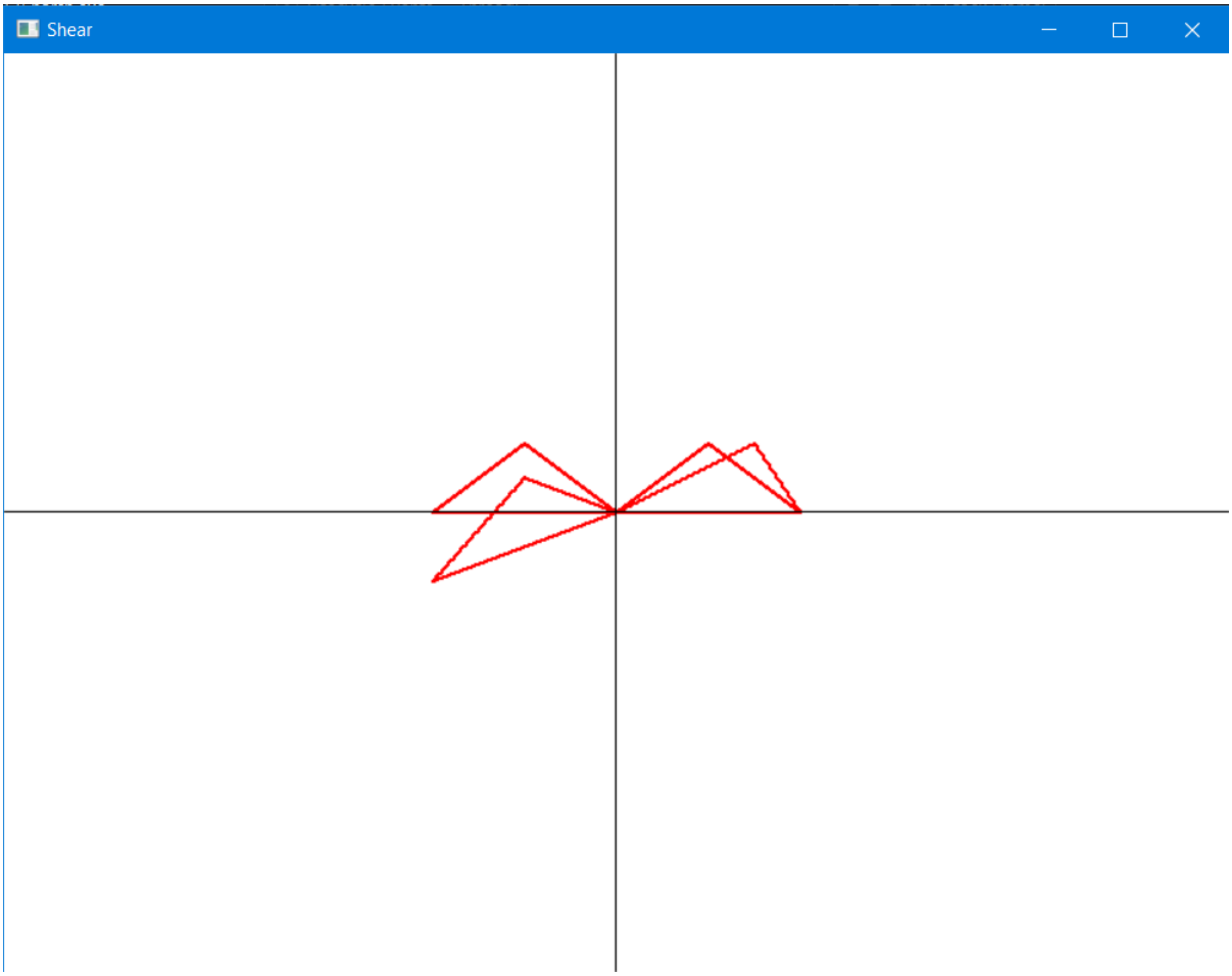
```
#include <GL\glut.h>
#include <stdlib.h>
#include <math.h>
#include <iostream>
using namespace std;
void init()
{
    glClearColor(1, 1, 1, 1.0);
    glPointSize(2.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-500, 500.0, -500.0, 500.0);
}
void makeTriangle(float x1, float y1, float x2, float y2, float x3, float y3)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_POINTS);
    float m1 = float(y2 - y1) / (x2 - x1);
    float m2 = float(y3 - y2) / (x3 - x2);
    float m3 = float(y1 - y3) / (x1 - x3);
    float c1 = y1 - m1 * x1;
    float c2 = y2 - m2 * x2;
    float c3 = y3 - m3 * x3;
    int i = 0;
    while (x1 + i <= x2)
    {
        glVertex2f(x1 + i, m1 * (x1 + i) + c1);
        i++;
    }
    i = 0;
    while (x2 + i <= x3)
    {
        glVertex2f(x2 + i, m2 * (x2 + i) + c2);
        i++;
    }
    i = 0;
    while (x3 - i >= x1)
    {
        glVertex2f(x3 - i, m3 * (x3 - i) + c3);
        i++;
    }
}
void reflection(float x1, float y1, float x2, float y2, float x3, float y3)
{
    float new_x1 = y1 - 2;
    float new_y1 = x1 + 2;
    float new_x2 = y2 - 2;
    float new_y2 = x2 + 2;
    float new_x3 = y3 - 2;
    float new_y3 = x3 + 2;
    cout << new_x1 << " " << new_y1 << " " << new_x2 << " " << new_y2 << " " << new_x3 << "
" << new_y3;
    makeTriangle(new_x1, new_y1, new_x3, new_y3, new_x2, new_y2);
}
void display()
{
    makeTriangle(-100, -200, 40, 42, 300, -100);
    reflection(-100, -200, 40, 42, 300, -100);
    glEnd();
    glColor3f(0, 0.0, 0.0);
    glBegin(GL_LINES);
    glVertex2f(-500, 0);
    glVertex2f(500, 0);
    glBegin(GL_LINES);
    glVertex2f(0, -500);
    glVertex2f(0, 500);
    glBegin(GL_LINES);
    glVertex2f(-500, -498);
    glVertex2f(498, 500);
    glEnd();
    glFlush();
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Reflection");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
}
```




Reflection

15 (b)Shear about x-axis and y-axis

```
#include <GL\glut.h>
#include <stdlib.h>
#include <math.h>
#include <iostream>
using namespace std;
void init()
{
  glClearColor(1, 1, 1, 1.0);
  glPointSize(2.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(-500, 500.0, -500.0, 500.0);
}
void makeTriangle(float x1, float y1, float x2, float y2, float x3, float y3)
{
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 0.0, 0.0);
  glBegin(GL_POINTS);
  float m1 = float(y2 - y1) / (x2 - x1);
  float m2 = float(y3 - y2) / (x3 - x2);
  float m3 = float(y1 - y3) / (x1 - x3);
  float c1 = y1 - m1 * x1;
  float c2 = y2 - m2 * x2;
  float c3 = y3 - m3 * x3;
  int i = 0;
  while (x1 + i <= x2)
  {
    glVertex2f(x1 + i, m1 * (x1 + i) + c1);
    i++;
  }
  i = 0;
  while (x2 + i <= x3)
  {
    glVertex2f(x2 + i, m2 * (x2 + i) + c2);
    i++;
  }
  i = 0;
  while (x3 - i >= x1)
  {
    glVertex2f(x3 - i, m3 * (x3 - i) + c3);
    i++;
  }
}
void shear_x(float x1, float y1, float x2, float y2, float x3, float y3)
{
  glClear(GL_COLOR_BUFFER_BIT);
  x1 = x1 + 0.5 * y1;
  x2 = x2 + 0.5 * y2;
  x3 = x3 + 0.5 * y3;
  makeTriangle(x1, y1, x2, y2, x3, y3);
}
void shear_y(float x1, float y1, float x2, float y2, float x3, float y3)
{
  glClear(GL_COLOR_BUFFER_BIT);
  y1 = y1 + 0.5 * x1;
  y2 = y2 + 0.5 * x2;
  y3 = y3 + 0.5 * x3;
  makeTriangle(x1, y1, x2, y2, x3, y3);
}
void display()
{
  makeTriangle(0, 0, 75.0, 75.0, 150.0, 0);
  shear_x(0, 0, 75.0, 75.0, 150.0, 0);
  makeTriangle(-150, 0, -75.0, 75.0, 0, 0);
  shear_y(-150, 0, -75.0, 75.0, 0, 0);
  glEnd();
  glColor3f(0, 0.0, 0.0);
  glBegin(GL_LINES);
  glVertex2f(-500, 0);
  glVertex2f(500, 0);
  glBegin(GL_LINES);
  glVertex2f(0, -500);
  glVertex2f(0, 500);
  glBegin(GL_LINES);
  glEnd();
  glFlush();
}
int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition(100, 100);
  glutInitWindowSize(800, 600);
  glutCreateWindow("Shear");
  init();
  glutDisplayFunc(display);
  glutMainLoop();
}
```

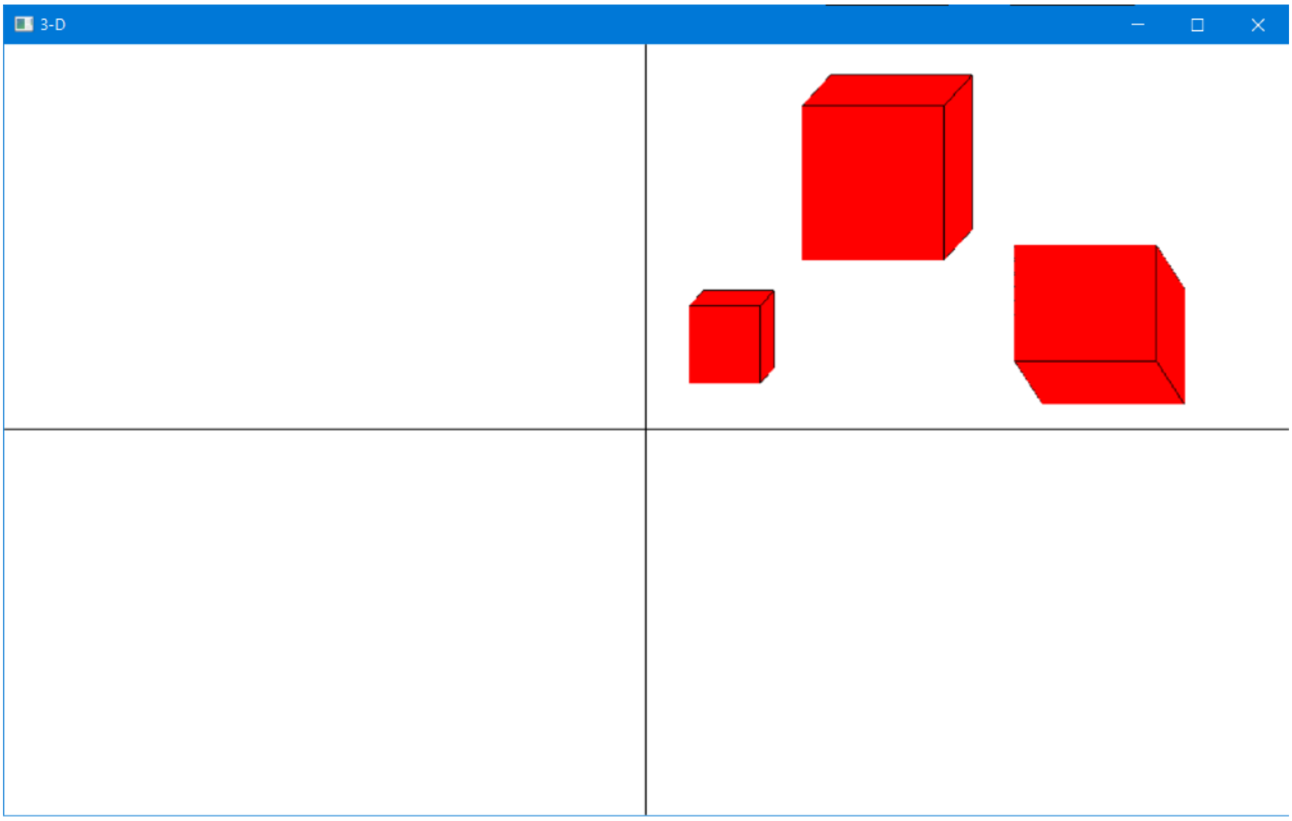

Shear

16. WAP to perform basic 3D transformation

```
#include <GL\glut.h>
#include <stdlib.h>
#include <math.h>
#include <iostream>
using namespace std;
typedef float Matrix4[4][4];
Matrix4 theMatrix;
static GLfloat input[8][3] =
{
   {40, 40, -50}, {90, 40, -50}, {90, 90, -50}, {40, 90, -50}, {30, 30, 0}, {80, 30, 0}, {80, 80, 0},
{30, 80, 0} };
float output[8][3];
void init()
{
   glClearColor(1.0, 1.0, 1.0, 1.0);
   glOrtho(-454.0, 454.0, -250.0, 250.0, -250.0, 250.0);
   glEnable(GL_DEPTH_TEST);
}
void setIdentityM(Matrix4 m)
{
   for (int i = 0; i < 4; i++)
      for (int j = 0; j < 4; j++)
         m[i][j] = (i == j);
}
void Axes(void)
{
   glColor3f(0.0, 0.0, 0.0);
   glBegin(GL_LINES);
   glVertex2s(-1000, 0);
   glVertex2s(1000, 0);
   glEnd();
   glBegin(GL_LINES);
   glVertex2s(0, -1000);
   glVertex2s(0, 1000);
   glEnd();
}
void draw(float a[8][3])
{
   glBegin(GL_QUADS);
   glColor3f(1, 0, 0);
   glVertex3fv(a[0]);
   glVertex3fv(a[1]);
   glVertex3fv(a[2]);
```

```
glVertex3fv(a[3]);
glColor3f(1, 0, 0);
glVertex3fv(a[0]);
glVertex3fv(a[1]);
glVertex3fv(a[5]);
glVertex3fv(a[4]);
glColor3f(1, 0, 0);
glVertex3fv(a[0]);
glVertex3fv(a[4]);
glVertex3fv(a[7]);
glVertex3fv(a[3]);
glColor3f(1, 0, 0);
glVertex3fv(a[1]);
glVertex3fv(a[2]);
glVertex3fv(a[6]);
glVertex3fv(a[5]);
glColor3f(1, 0, 0);
glVertex3fv(a[2]);
glVertex3fv(a[3]);
glVertex3fv(a[7]);
glVertex3fv(a[6]);
glColor3f(1, 0, 0);
glVertex3fv(a[4]);
glVertex3fv(a[5]);
glVertex3fv(a[6]);
glVertex3fv(a[7]);
glEnd();
glColor3f(0, 0, 0);
glBegin(GL_LINE_LOOP);
glVertex3fv(a[0]);
glVertex3fv(a[1]);
glVertex3fv(a[1]);
glVertex3fv(a[2]);
glVertex3fv(a[2]);
glVertex3fv(a[3]);
glEnd();
glColor3f(0, 0, 0);
glBegin(GL_LINE_STRIP);
glVertex3fv(a[0]);
glVertex3fv(a[1]);
glVertex3fv(a[5]);
glVertex3fv(a[4]);
glVertex3fv(a[0]);
glEnd();
glColor3f(0, 0, 0);
glBegin(GL_LINE_STRIP);
glVertex3fv(a[0]);
```

```
    glVertex3fv(a[4]);
    glVertex3fv(a[7]);
    glVertex3fv(a[3]);
    glVertex3fv(a[0]);
    glEnd();
    glColor3f(0, 0, 0);
    glBegin(GL_LINE_STRIP);
    glVertex3fv(a[1]);
    glVertex3fv(a[2]);
    glVertex3fv(a[6]);
    glVertex3fv(a[5]);
    glVertex3fv(a[1]);
    glEnd();
    glColor3f(0, 0, 0);
    glBegin(GL_LINE_STRIP);
    glVertex3fv(a[2]);
    glVertex3fv(a[3]);
    glVertex3fv(a[7]);
    glVertex3fv(a[6]);
    glVertex3fv(a[2]);
    glEnd();
    glColor3f(0, 0, 0);
    glBegin(GL_LINE_STRIP);
    glVertex3fv(a[4]);
    glVertex3fv(a[5]);
    glVertex3fv(a[6]);
    glVertex3fv(a[7]);
    glVertex3fv(a[4]);
    glEnd();
}
void RotateX(float angle) //Parallel to x
{
angle = angle * 3.142 / 180;
theMatrix[1][1] = cos(angle);
theMatrix[1][2] =
-sin(angle);
theMatrix[2][1] = sin(angle);
theMatrix[2][2] = cos(angle);
}
void scale(int sx, int sy, int sz)
{
    theMatrix[0][0] = sx;
    theMatrix[1][1] = sy;
    theMatrix[2][2] = sz;
}
void multiplyM()
{
    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            for (int k = 0; k < 3; k++)
            {
                output[i][j] = output[i][j] + input[i][k] * theMatrix[k][j];
            }
        }
    }
}
void trans(int tx, int ty, int tz)
{
    for (int i = 0; i < 8; i++)
    {
        output[i][0] = input[i][0] + tx;
        output[i][1] = input[i][1] + ty;
        output[i][2] = input[i][2] + tz;
    }
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    Axes();
    glColor3f(1.0, 0.0, 0.0);
    draw(input);
    trans(50, 50, 50);
    scale(1.5, 1.5, 1.5);
    multiplyM();
    draw(output);
    trans(200, 0, 0);
    RotateX(60);
    multiplyM();
    draw(output);
    glFlush();
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(1000, 600);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("3-D");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
}
```



3-D

17. WAP to clip a line using Liang Barsky Algorithm and CohenSutherland

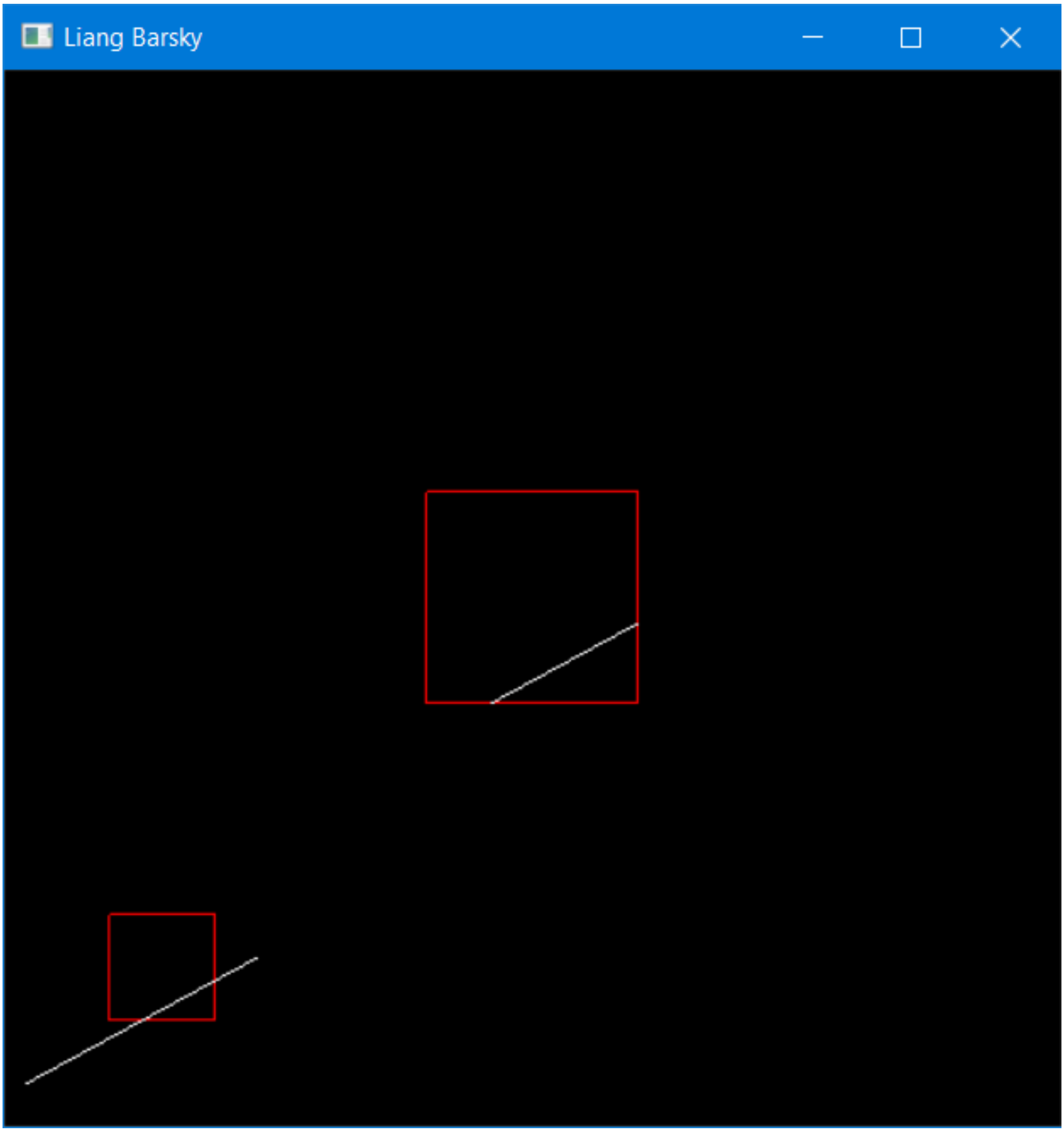
Liang Barsky Algorithm

```
#include <GLUT/GLUT.h>


double y_max = 100, y_min = 50, x_max = 100, x_min = 50; //
Oldviewport

double ny_max = 300, ny_min = 200, nx_max = 300, nx_min = 200;
// New clipped ViewPort

double t1 = 0.0, t2 = 1.0;      // Intial and final time


double x1 = 10, y1 = 20;    // Point

1double x2 = 120, y2 = 80;  //

Point 2



void myDisplay();

void draw_lineAndPort(double x1, double y1, double x2, double
y2,double y_max, double y_min, double x_max, double x_min);

void liangBarsky(double x1, double y1, double x2, double

y2);bool cliptest(double p, double q);



void myInit()

{

   glLoadIdentity();

   glMatrixMode(GL_PROJECTION

   );gluOrtho2D(0, 500, 0, 500);

   glMatrixMode(GL_MODELVIEW);
}
int main(int argc, char **argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE |
   GLUT_RGB);glutInitWindowPosition(0, 0);
   glutInitWindowSize(500, 500);

   glutCreateWindow("Liang Barsky");


   glutDisplayFunc(myDisplay
   );myInit();
   glutMainLoop(
   );return 0;
}




void myDisplay()
{
   glClear(GL_COLOR_BUFFER_BIT);

   draw_lineAndPort(x1, y1, x2, y2, y_max, y_min, x_max,
   x_min);liangBarsky(x1, y1, x2, y2);

   glFlush();
}
```

```
void draw_lineAndPort(double x1, double y1, double x2, double
y2,double y_max, double y_min, double x_max, double x_min)
{
    glColor3d(1, 0, 0);
    glBegin(GL_LINE_LOOP
    );glVertex2d(x_min,
    y_min);
    glVertex2d(x_max,
    y_min);
    glVertex2d(x_max,
    y_max);
    glVertex2d(x_min,
    y_max);glEnd();

    glColor3d(1, 1, 1);
    glBegin(GL_LINES
    );glVertex2d(x1,
    y1); glVertex2d(x2,
    y2); glEnd();
}

bool cliptest(double p, double q)
{
    double t = q / p;

    if (p == 0 && q < 0)    // Line is parallel to viewport and outside
    {
        return false;
    }
    else if (p < 0)
```

```
   {
     if (t > t1)
     t1 = t; if (t > t2)
     return false;
     }
     else if (p > 0)
     {
     if (t < t2)t2 = t;
     if (t < t1) return false;
     }

   return true;

}


void liangBarsky(double x1, double y1, double x2, double y2)

{

   double dx = x2 -

   x1;double dy = y2

   - y1;


   /*

    -t * dx < x1 - x_min        ... [1]

     t * dx < x_max - x1        ... [2]

    -t * dy < y1 - y_min        ... [3]

     t * dy < y_max - y1        ... [4]

    */

   if (cliptest(-dx, x1 - x_min) && cliptest(dx, x_max - x1)
&&cliptest(-dy, y1 - y_min) && cliptest(dy, y_max - y1))

   {

      if (t2 < 1)
```

```
{
   x2 = x1 + t2 *
   dx;y2 = y1 + t2
   * dy;
}
if (t1 > 0)
{
   x1 = x1 + t1 *
   dx;y1 = y1 + t1
   * dy;
}

// Scaling to new View port
double scale_x = (nx_max - nx_min) / (x_max -
x_min);double scale_y = (ny_max - ny_min) / (y_max
- y_min);

// New coordinates of the points
// Point 1
double nx1 = nx_min + (x1 - x_min) *
scale_x;double ny1 = ny_min + (y1 - y_min)
* scale_y;

// Point 2
double nx2 = nx_min + (x2 - x_min) *
scale_x;double ny2 = ny_min + (y2 - y_min)
* scale_y;

// Plotting new Viewport and clipped line
draw_lineAndPort(nx1, ny1, nx2, ny2, ny_max,
        ny_min,
nx_max, nx_min);
    }
}
```

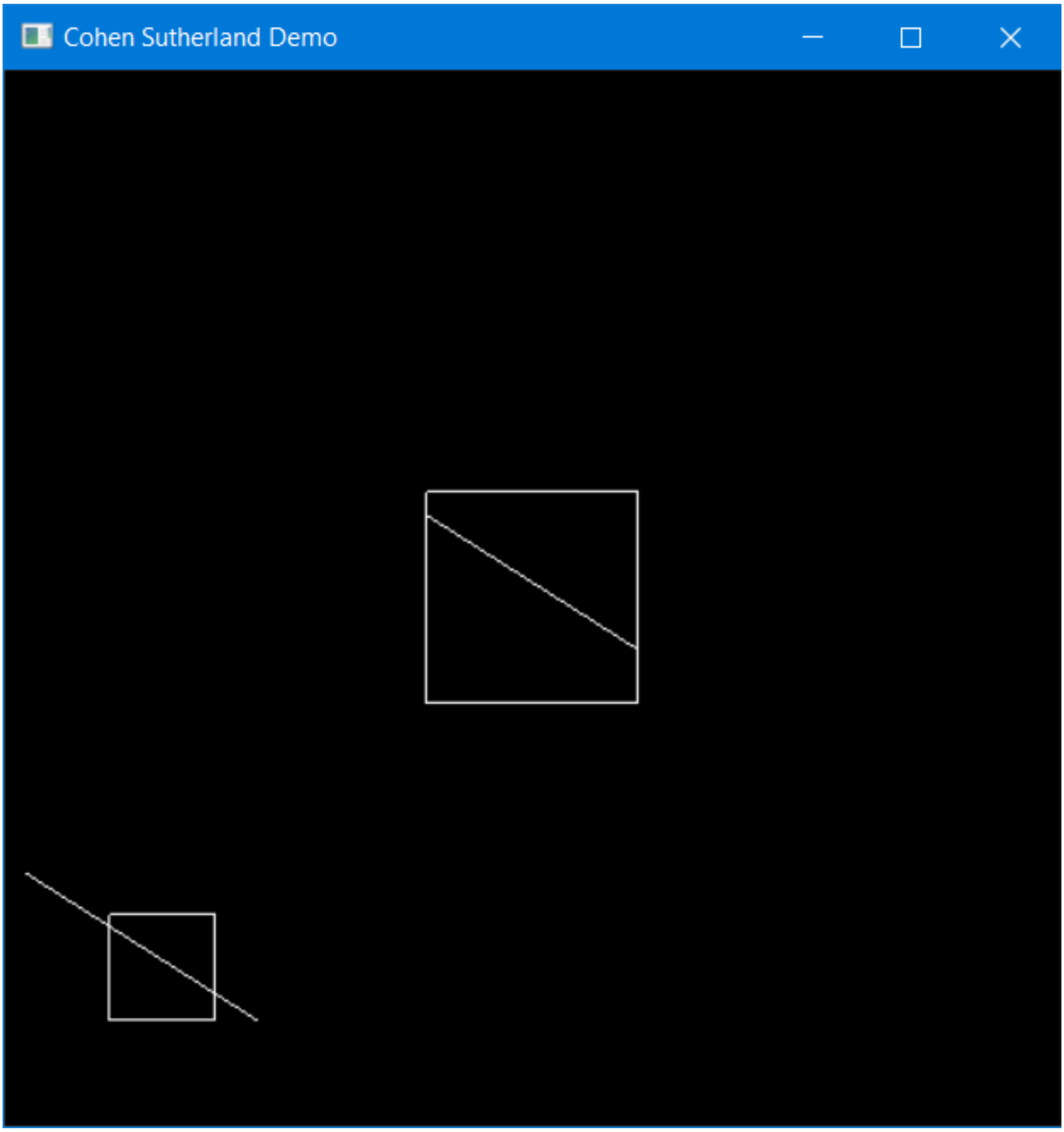


Cohen Sutherland

```
#include <GLUT/

GLUT.h>#include

<stdio.h>


#pragma GCC diagnostic ignored "-Wdeprecated-declarations" //
Remove deprecation warnings


double y_max = 100, y_min = 50, x_max = 100, x_min = 50; //
Oldviewport

double ny_max = 300, ny_min = 200, nx_max = 300, nx_min = 200;
// New clipped ViewPort

int TOP = 8, BOTTOM = 4, RIGHT = 2, LEFT = 1;


double x1 = 10, y1 = 120;    // Point

1double x2 = 120, y2 = 50; // Point

2


void myInit();

void

myDisplay();

void draw_lineAndPort(double x1, double y1, double x2, double
y2,double y_max, double y_min, double x_max, double x_min);

void cohenSutherland(double x1, double y1, double x2, double

y2);int outcode(double x, double y);


int main(int argc, char **argv)

{

   glutInit(&argc, argv);

   glutInitDisplayMode(GLUT_SINGLE |

   GLUT_RGB);glutInitWindowPosition(0, 0);

   glutInitWindowSize(500, 500);
```

```
    glutCreateWindow("Cohen Sutherland Demo");


    glutDisplayFunc(myDisplay
    );myInit();
    glutMainLoop(
    );return 0;
}

void myInit()
{
    glLoadIdentity();
    glMatrixMode(GL_PROJECTION
    );gluOrtho2D(0, 500, 0, 500);
    glMatrixMode(GL_MODELVIEW);
}

void myDisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);
    draw_lineAndPort(x1, y1, x2, y2, y_max, y_min, x_max,
    x_min);cohenSutherland(x1, y1, x2, y2);
    glFlush();
}

void draw_lineAndPort(double x1, double y1, double x2, double
y2,double y_max, double y_min, double x_max, double x_min)
{
      // Viewposrt glColor3d(1, 0, 0);
      glBegin(GL_LINE_LOOP);
      glVertex2d(x_min, y_min);
      glVertex2d(x_max, y_min);
      glVertex2d(x_max, y_max);
      glVertex2d(x_min, y_max);
      glEnd();
```

```
// Line glColor3d(1, 1, 1);
glBegin(GL_LINES);
glVertex2d(x1, y1); glVertex2d(x2, y2); glEnd();
}


int outcode(double x, double y)
{
int outcode = 0;
if (y > y_max)
outcode |= TOP;
else if (y < y_min)
outcode |= BOTTOM;



if (x > x_max) outcode |= RIGHT;
else if (x < x_min) outcode |= LEFT;
return outcode;
}
void cohenSutherland(double x1, double y1, double x2, double y2)
{
int outcode1 = outcode(x1, y1);int outcode2 = outcode(x2, y2);int outcodeOut;
bool accept = false, done = false;


do
{
if ((outcode1 | outcode2) == 0)          // line is completely inside
{
accept = true;done = true;
}
else if ((outcode1 & outcode2) != 0) // line is completely outside
{
done = true;
}
else
{
outcodeOut = (outcode1 != 0)? outcode1 : outcode2;double x, y;
double slope = (y2 - y1) / (x2 - x1);


if (outcodeOut & TOP)
{
y = y_max;
x = x1 + (y - y1) / slope;}
else if (outcodeOut & BOTTOM)
{
y = y_min;
x = x1 + (y - y1) / slope;
}
else if (outcodeOut & RIGHT)
{
x = x_max;
y = y1 + (x - x1) * slope;
}
else
{
x = x_min;
y = y1 + (x - x1) * slope;
}

if (outcodeOut == outcode1)
{
x1 = x;y1 = y;
outcode1 = outcode(x1, y1);
}
else
{
x2 = x;y2 = y;
outcode2 = outcode(x2, y2);
}
}
                    } while (!done);



                    if(accept)

                    {

                      double scale_x = (nx_max - nx_min) / (x_max -

                      x_min);double scale_y = (ny_max - ny_min) / (y_max

                      - y_min);


                      double nx1 = nx_min + (x1 - x_min) *

                      scale_x;double ny1 = ny_min + (y1 - y_min)

                      * scale_y;


                      double nx2 = nx_min + (x2 - x_min) *

                      scale_x;double ny2 = ny_min + (y2 - y_min)

                      * scale_y;


                      draw_lineAndPort(nx1, ny1, nx2, ny2, ny_max,
                  ny_min,nx_max, nx_min);

                    }

                  }
```


Cohen Sutherland Demo

18. WAP to clip a line using Nicholl-Lee-Nicholl line clipping

```
#include <GL\glut.h>
#include <stdlib.h>
#include <math.h>
#include <cstdlib>
using namespace std;
double y_max = 100, y_min = 50, x_max = 100, x_min = 50;       // Old viewport
double ny_max = 300, ny_min = 200, nx_max = 300, nx_min = 200; // New
ViewPort
double t1 = 0.0, t2 = 1.0;
double X1 = 10;
double Y1 = 20;
double X2 = 120;
double Y2 = 80;
double xx1, xx2, yy1, yy2;
void init()
{
   glLoadIdentity();
   glClearColor(1.0, 1.0, 1.0, 1.0);
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D(0, 500, 0, 500);
   glMatrixMode(GL_MODELVIEW);
}
void draw_lineAndPort(double x1, double y1, double x2, double y2, double
y_max, double y_min, double x_max, double x_min)
{
   glColor3d(1, 0, 0);
   glBegin(GL_LINE_LOOP);
   glVertex2d(x_min, y_min);
   glVertex2d(x_max, y_min);
   glVertex2d(x_max, y_max);
   glVertex2d(x_min, y_max);
   glEnd();
   glColor3d(0, 0, 0);
   glBegin(GL_LINES);
   glVertex2d(x1, y1);
   glVertex2d(x2, y2);
   glEnd();
}
void clipline1(int x1, int y1, int x2, int y2)
```

```
{
   int draw = 1;
   float m, m1, m2, m3, m4;
   m = ((float)(y2 - y1)) / (x2 - x1);
   m1 = ((float)(y_min - y1)) / (x_min - x1);
   m2 = ((float)(y_min - y1)) / (x_max - x1);
   m3 = ((float)(y_max - y1)) / (x_max - x1);
   m4 = ((float)(y_max - y1)) / (x_min - x1);
   xx1 = x1;
   yy1 = y1;
   if (((abs(m) >= m1 && x2 < x1) || (abs(m) > abs(m2) && x2 > x1)) && y1 >
y2)
   {
      if (y2 > y_min)
      {
         xx2 = x2;
         yy2 = y2;
      }
      else
      {
         yy2 = y_min;
         xx2 = x1 + (y_min - y1) / m;
      }
   }
   else if (m > m2 && m < m3 && x2 >= x_max)
   {
      if (x2 < x_max)
      {
         xx2 = x2;
         yy2 = y2;
      }
      else
      {
         xx2 = x_max;
         yy2 = y1 + (x_max - x1) * m;
      }
   }
   else if ((abs(m) >= m3 && x2 > x1) || (abs(m) > abs(m4) && x2 < x1))
   {
      if (y2 < y_max)
```

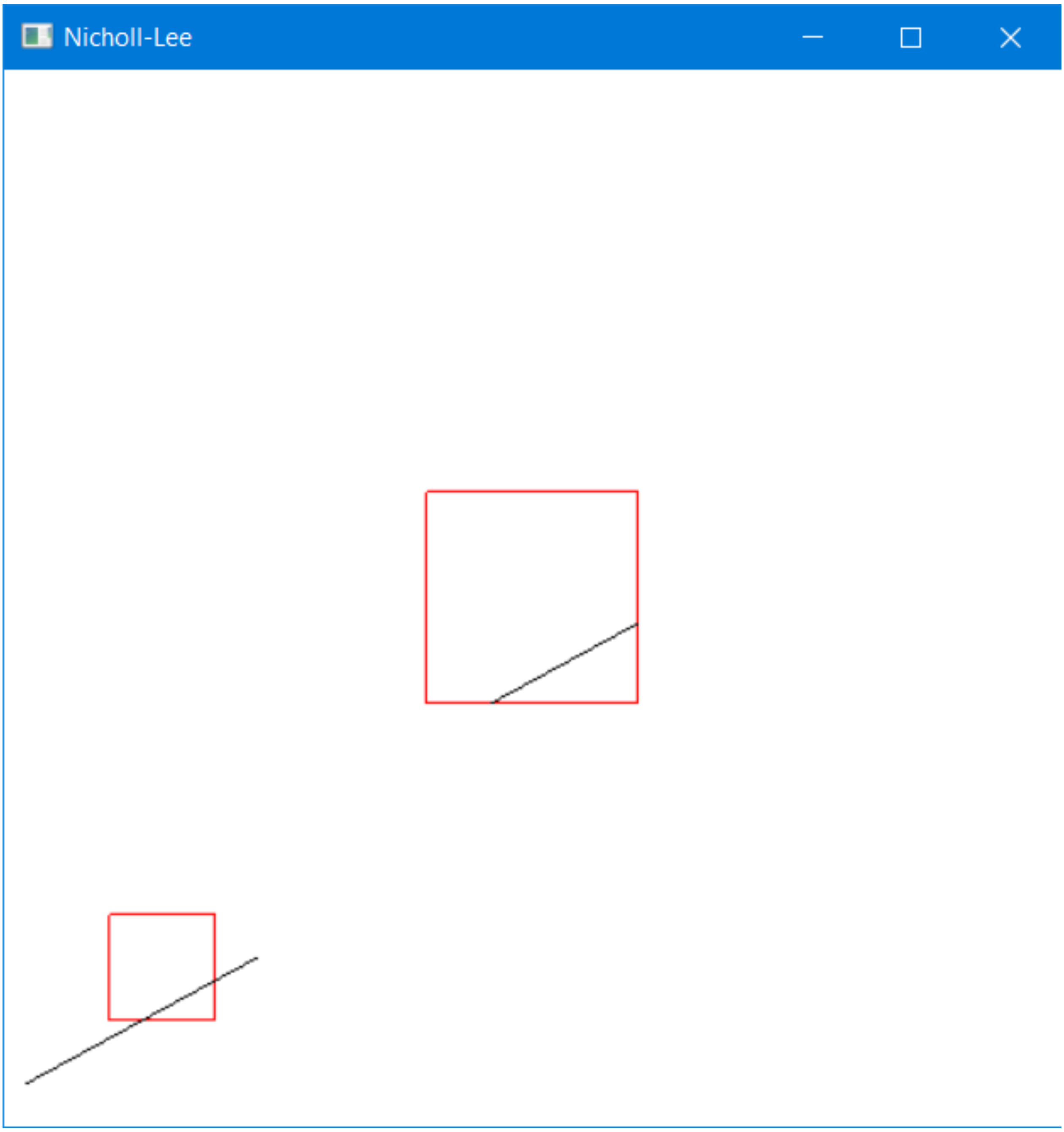
```
        {
            xx2 = x2;
            yy2 = y2;
        }
        else
        {
            yy2 = y_max;
            xx2 = x1 + (y_max - y1) / m;
        }
    }
    else if (m > m4 && m < m1)
    {
        if (x2 > x_min)
        {
            xx2 = x2;
            yy2 = y2;
        }
        else

        {
            xx2 = x_min;
            yy2 = y1 + (x_min - x1) * m;
        }
    }
}
void clipline2(int x1, int y1, int x2, int y2)
{
    int draw = 1;
    float m, m1, m2, m3, m4;
    m = ((float)(y2 - y1)) / (x2 - x1);
    m1 = ((float)(y_min - y1)) / (x_min - x1);
    m2 = ((float)(y_min - y1)) / (x_max - x1);
    m3 = ((float)(y_max - y1)) / (x_max - x1);
    m4 = ((float)(y_max - y1)) / (x_min - x1);
    if (m > m1 && m < m2)

    {
        if (y2 > y_min)

        {
```

```
        xx1 = x_min;
        yy1 = y1 + m * (x_min - x1);
        xx2 = x2;
        yy2 = y2;
    }
    else

    {
        xx1 = x_min;
        yy1 = y1 + m * (x_min - x1);
        yy2 = y_min;
        xx2 = x1 + (y_min - y1) / m;
    }
}
else if (m > m2 && m < m3)

{
    if (x2 < x_max)

    {
        xx1 = x_min;
        yy1 = y1 + m * (x_min - x1);
        xx2 = x2;
        yy2 = y2;
    }
    else

    {
        xx1 = x_min;
        yy1 = y1 + m * (x_min - x1);
        xx2 = x_max;
        yy2 = y1 + (x_max - x1) * m;
    }
}
else if (m > m3 && m < m4)
{
    if (y2 < y_max)
    {
        xx1 = x_min;
        yy1 = y1 + m * (x_min - x1);
```

```
            xx2 = x2;
            yy2 = y2;
        }
        else
        {
            xx1 = x_min;
            yy1 = y1 + m * (x_min - x1);
            yy2 = y_max;
            xx2 = x1 + (y_max - y1) / m;
        }
    }
}
void clipline3(int x1, int y1, int x2, int y2)
{
    int draw = 1;
    float m, m1, m2, m3, m4, tm1, tm2;
    int flag, t;
    tm1 = ((float)(y_min - y1)) / (x_min - x1);
    tm2 = ((float)(y_max - y_min)) / (x_max - x_min); //diagonal slope
    m = ((float)(y2 - y1)) / (x2 - x1);
    m1 = ((float)(y_min - y1)) / (x_max - x1);
    m2 = ((float)(y_max - y1)) / (x_max - x1);
    m3 = ((float)(y_min - y1)) / (x_min - x1);
    m4 = ((float)(y_max - y1)) / (x_min - x1);
    if (tm1 < tm2)
    {
        flag = 2;
        t = m2;
        m2 = m3;
        m3 = t;
    }
    else
        flag = 1;
    if (m > m1 && m < m2)
    {
        if (x2 > x_max && y2 > y_min)
        {
            yy1 = y_min;
            xx1 = x1 + (y_min - y1) / m;
            xx2 = x_max;
        yy2 = y1 + m * (x_max - x1);
    }
    else if (y2 > y_min && x2 < x_max)

    {
        yy1 = y_min;
        xx1 = x1 + (y_min - y1) / m;
        yy2 = y2;
        xx2 = x2;
    }
}
else if (m > m2 && m < m3)

{
    if (flag == 1)

    {
        if (y2 >= y_max)

        {
            yy1 = y_min;
            xx1 = x1 + (y_min - y1) / m;
            xx2 = x1 + (y_max - y1) / m;
            yy2 = y_max;
        }
        else if (y2 >= y_min)

        {
            yy1 = y_min;
            xx1 = x1 + (y_min - y1) / m;
            xx2 = x2;
            yy2 = y2;
        }
    }
    else

    {
        if (x2 >= x_max)

        {
                xx1 = x_min;
                yy1 = y1 + m * (x_min - x1);
                xx2 = x_max;
                yy2 = y1 + m * (x_max - x1);
            }
            else if (x2 >= x_min)

            {
                xx1 = x_min;
                yy1 = y1 + m * (x_min - x1);
                xx2 = x2;
                yy2 = y2;
            }
        }
    }
    else if (m > m3 && m < m4)

    {
        if (y2 >= y_max)

        {
            xx1 = x_min;
            yy1 = y1 + m * (x_min - x1);
            xx2 = x1 + (y_max - y1) / m;
            yy2 = y_max;
        }
        else if (y2 >= y_min)
        {
            xx1 = x_min;
            yy1 = y1 + m * (x_min - x1);
            yy2 = y2;
            xx2 = x2;
        }
    }
}
int first_end_point_region(int x, int y)
{
    if (x >= x_min && x <= x_max && y >= y_min && y <= y_max)
        return 1;
    else if (x < x_min && y >= y_min && y <= y_max)
```

```
        return 2;
    else if (x <= x_min && y <= y_min)
        return 3;
    else
        return 0;
}
void nicholl_lee(double x1, double y1, double x2, double y2)
{
    int ch = first_end_point_region(x1, y1);
    switch (ch)
    {
    case 1:
        clipline1(x1, y1, x2, y2);
        break;
    case 2:
        clipline2(x1, y1, x2, y2);
        break;
    case 3:
        clipline3(x1, y1, x2, y2);
        break;
    }
    // Scaling to new View port
    double scale_x = (nx_max - nx_min) / (x_max - x_min);
    double scale_y = (ny_max - ny_min) / (y_max - y_min);
    // New coordinates of the points
    // Point 1
    double nx1 = nx_min + (xx1 - x_min) * scale_x;
    double ny1 = ny_min + (yy1 - y_min) * scale_y;
    // Point 2
    double nx2 = nx_min + (xx2 - x_min) * scale_x;
    double ny2 = ny_min + (yy2 - y_min) * scale_y;
    draw_lineAndPort(nx1, ny1, nx2, ny2, ny_max, ny_min, nx_max, nx_min);
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    draw_lineAndPort(X1, Y1, X2, Y2, y_max, y_min, x_max, x_min);
    nicholl_lee(X1, Y1, X2, Y2);
    glFlush();
}
```

```
                    int main(int argc, char** argv)
                    {
                       glutInit(&argc, argv);
                       glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
                       glutInitWindowPosition(0, 0);
                       glutInitWindowSize(500, 500);
                       glutCreateWindow("Nicholl-Lee");
                       glutDisplayFunc(display);
                       init();
                       glutMainLoop();
                       return 0;
}
```



19. WAP to clip a polygon using Sutherland Hodgeman and Weiler Atherton Algorithm

Sutherland
Hodgeman

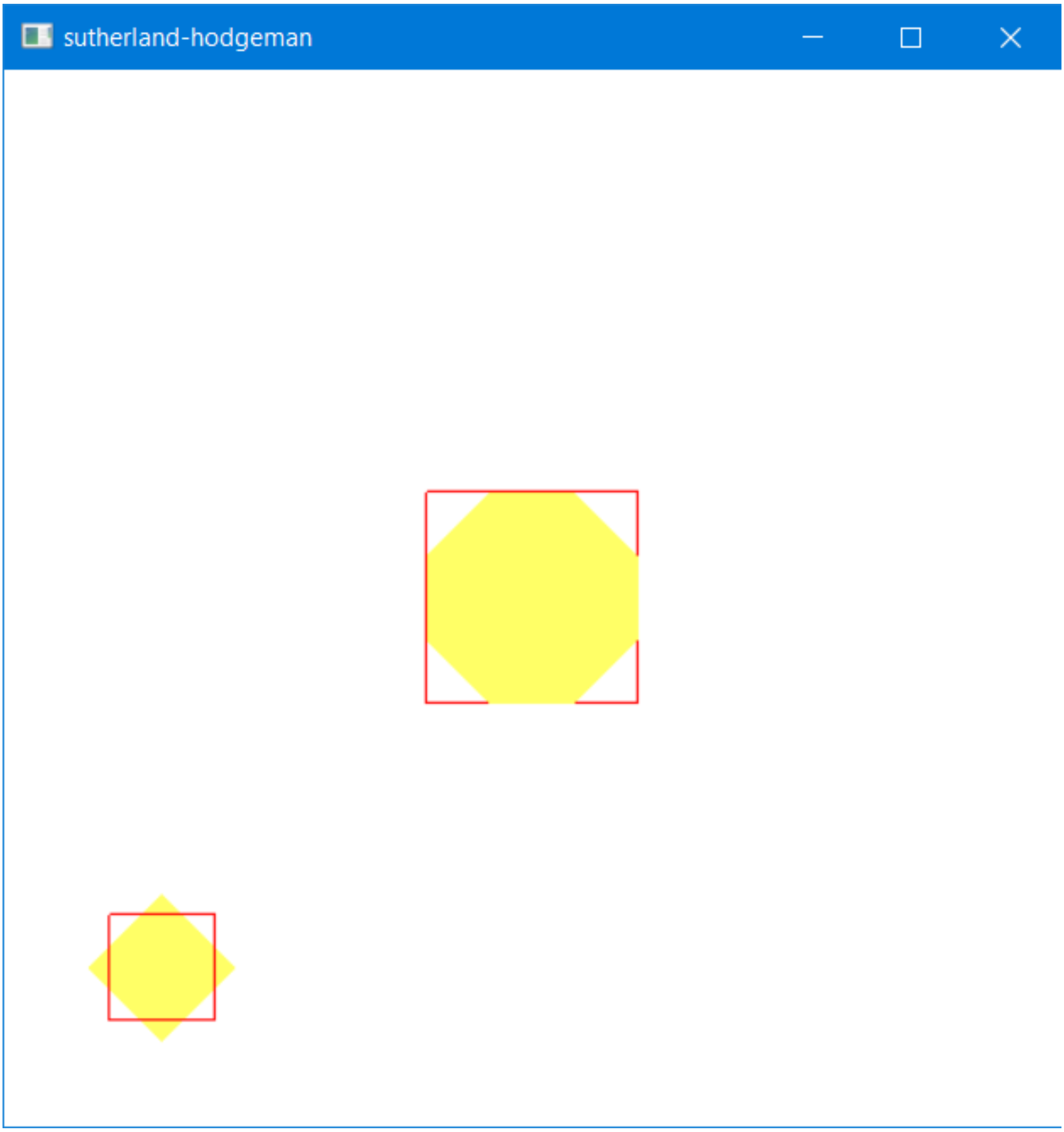
```
#include <GL\glut.h>
#include <stdlib.h>
#include <math.h>
#include <cstdlib>
#include <iostream>
using namespace std;
const int MAX_POINTS = 20;
double y_max = 100, y_min = 50, x_max = 100, x_min = 50;      // Old viewport
double ny_max = 300, ny_min = 200, nx_max = 300, nx_min = 200; // New ViewPort
int n_sp = 4, n_cw = 4;
struct vertex
{
   float x;
   float y;
};
vertex cw[] = { {50, 50}, {50, 100}, {100, 100}, {100, 50} };
vertex sp[] = { {40, 75}, {75, 110}, {110, 75}, {75, 40} };
void init()
{
   glLoadIdentity();
   glClearColor(1.0, 1.0, 1.0, 1.0);
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D(0, 500, 0, 500);
   glMatrixMode(GL_MODELVIEW);
}
void draw_lineAndPort(double y_max, double y_min, double x_max, double x_min)
```

```
{
    glColor3d(1, 0, 0);
    glBegin(GL_LINE_LOOP);
    glVertex2d(x_min, y_min);
    glVertex2d(x_max, y_min);
    glVertex2d(x_max, y_max);
    glVertex2d(x_min, y_max);
    glEnd();
}
void draw_poly(vertex vlist[], int n)
{
    glColor3d(1, 1, 102.0 / 255);
    glBegin(GL_POLYGON);
    for (int i = 0; i < n; i++)
    {
        glVertex2d(vlist[i].x, vlist[i].y);
        glVertex2d(vlist[(i + 1) % n].x, vlist[(i + 1) % n].y);
    }
    glEnd();
}
int x_intersect(int x1, int y1, int x2, int y2,
    int x3, int y3, int x4, int y4)
{
    int num = (x1 * y2 - y1 * x2) * (x3 - x4) -
        (x1 - x2) * (x3 * y4 - y3 * x4);
    int den = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);
    return num / den;
}
int y_intersect(int x1, int y1, int x2, int y2,
    int x3, int y3, int x4, int y4)
{
    int num = (x1 * y2 - y1 * x2) * (y3 - y4) -
```

```
        (y1 - y2) * (x3 * y4 - y3 * x4);
    int den = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);
    return num / den;
}
void clip(vertex poly_points[], int& poly_size, int x1, int y1, int x2, int y2)
{
    int new_points[MAX_POINTS][2], new_poly_size = 0;
    for (int i = 0; i < poly_size; i++)
    {
        // i and k form a line in polygon
        int k = (i + 1) % poly_size;
        int ix = poly_points[i].x, iy = poly_points[i].y;
        int kx = poly_points[k].x, ky = poly_points[k].y;
        // Calculating position of first point
        // w.r.t. clipper line
        int i_pos = (x2 - x1) * (iy - y1) - (y2 - y1) * (ix - x1);
        // Calculating position of second point
        // w.r.t. clipper line
        int k_pos = (x2 - x1) * (ky - y1) - (y2 - y1) * (kx - x1);
        // Case 1 : When both points are inside
        if (i_pos < 0 && k_pos < 0)
        {
            //Only second point is added
            new_points[new_poly_size][0] = kx;
            new_points[new_poly_size][1] = ky;
            new_poly_size++;
        }
        // Case 2: When only first point is outside
        else if (i_pos >= 0 && k_pos < 0)
        {
            // Point of intersection with edge
            // and the second point is added
            new_points[new_poly_size][0] = x_intersect(x1,
                y1, x2, y2, ix, iy, kx, ky);
            new_points[new_poly_size][1] = y_intersect(x1,
                y1, x2, y2, ix, iy, kx, ky);
            new_poly_size++;
            new_points[new_poly_size][0] = kx;
            new_points[new_poly_size][1] = ky;
            new_poly_size++;
        }
        // Case 3: When only second point is outside
        else if (i_pos < 0 && k_pos >= 0)


        {
            //Only point of intersection with edge is added
            new_points[new_poly_size][0] = x_intersect(x1,
                y1, x2, y2, ix, iy, kx, ky);
            new_points[new_poly_size][1] = y_intersect(x1,
                y1, x2, y2, ix, iy, kx, ky);
            new_poly_size++;
        }
        // Case 4: When both points are outside
        else


        {
            //No points are added
        }
    }
    poly_size = new_poly_size;
    for (int i = 0; i < poly_size; i++)


    {
        poly_points[i].x = new_points[i][0];
```

```
        poly_points[i].y = new_points[i][1];
    }
}
void sutherlandhodgeman()
{
    vertex sp1[20];
    for (int i = 0; i < 4; i++)

    {
        int k = (i + 1) % 4;
        clip(sp, n_sp, cw[i].x, cw[i].y, cw[k].x, cw[k].y);
    }
    double scale_x = (nx_max - nx_min) / (x_max - x_min);
    double scale_y = (ny_max - ny_min) / (y_max - y_min);
    for (int i = 0; i < n_sp; i++)
    {
        sp1[i].x = nx_min + (sp[i].x - x_min) * scale_x;
        sp1[i].y = ny_min + (sp[i].y - y_min) * scale_y;
    }
    draw_lineAndPort(ny_max, ny_min, nx_max, nx_min);
    draw_lineAndPort(y_max, y_min, x_max, x_min);
    draw_poly(sp1, n_sp);
    for (int i = 0; i < n_sp; i++)
        cout << '(' << sp[i].x << ", " << sp[i].y << ") ";
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    draw_lineAndPort(y_max, y_min, x_max, x_min);
    draw_poly(sp, 4);
    sutherlandhodgeman();
    glFlush();
```

```
}
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowPosition(0, 0);
   glutInitWindowSize(500, 500);
   glutCreateWindow("sutherland-hodgeman");
   glutDisplayFunc(display);
   init();
   glutMainLoop();
   return 0;
}
```



sutherland-hodgeman

Weiler Atherton

Algorithm

```
#include <iostream>
#include <cstdlib>
#include <vector>
#include <list>
#include <GL/glut.h>
#define Size 500
using namespace std;
typedef float Color[3];
struct Point
{
int x, y;
};
typedef struct IntersectionPoint
{
int pointFlag;
int index0, index1;
Point p;
bool inFlag;
int dis;
} IP;
double y_max = 100, y_min = 50, x_max = 100, x_min = 50;       // Old viewport
double ny_max = 300, ny_min = 200, nx_max = 300, nx_min = 200; // New ViewPort
double scale_x = (nx_max - nx_min) / (x_max - x_min);
double scale_y = (ny_max - ny_min) / (y_max - y_min);
class Pg
{
public:
vector<Point> pts;
Pg(void);
void drawPgLine(Color c);
void drawpoly();
};
Pg::Pg(void)
{
}
void Pg::drawPgLine(Color c)
{
glColor3fv(c);
glLineWidth(2.0);
glBegin(GL_LINE_LOOP);
int size = pts.size();
for (int i = 0; i < size; i++)
glVertex2i(pts[i].x, pts[i].y);
```

```
glEnd();
}
void Pg::drawpoly()
{
glColor3d(1, 1, 102.0 / 255);
glBegin(GL_POLYGON);
int size = pts.size();
for (int i = 0; i < size; i++)
{
glVertex2d(pts[i].x, pts[i].y);
glVertex2d(pts[(i + 1) % size].x, pts[(i + 1) % size].y);
}
glEnd();
}
bool isPointInsidePg(Point p, Pg &py)
{
int cnt = 0, size = py.pts.size();
for (int i = 0; i < size; i++)
{
Point p1 = py.pts[i];
Point p2 = py.pts[(i + 1) % size];
if (p1.y == p2.y)
continue;
if (p.y < min(p1.y, p2.y))
continue;
if (p.y >= max(p1.y, p2.y))
continue;
double x = (double)(p.y - p1.y) * (double)(p2.x - p1.x) / (double)(p2.y - p1.y) + p1.x;
if (x > p.x)
cnt++;
}
return (cnt % 2 == 1);
}
int cross(Point &p0, Point &p1, Point &p2)
{
return ((p2.x - p0.x) * (p1.y - p0.y) - (p1.x - p0.x) * (p2.y - p0.y));
}
bool onSegment(Point &p0, Point &p1, Point &p2)
{
int minx = min(p0.x, p1.x), maxx = max(p0.x, p1.x);
int miny = min(p0.y, p1.y), maxy = max(p0.y, p1.y);
if (p2.x >= minx && p2.x <= maxx && p2.y >= miny && p2.y <= maxy)
return true;
return false;
}
bool segmentsIntersect(Point &p1, Point &p2, Point &p3, Point &p4)
{
int d1 = cross(p3, p4, p1);
int d2 = cross(p3, p4, p2);
int d3 = cross(p1, p2, p3);
int d4 = cross(p1, p2, p4);
if (((d1 > 0 && d2 < 0) || (d1 < 0 && d2 > 0)) &&
((d3 > 0 && d4 < 0) || (d3 < 0 && d4 > 0)))
return true;
if (d1 == 0 && onSegment(p3, p4, p1))
return true;
if (d2 == 0 && onSegment(p3, p4, p2))
return true;
if (d3 == 0 && onSegment(p1, p2, p3))
return true;
if (d4 == 0 && onSegment(p1, p2, p4))
return true;
return false;
}
Point getintersectPoint(Point p1, Point p2, Point p3, Point p4)
{
Point p;
int b1 = (p2.y - p1.y) * p1.x + (p1.x - p2.x) * p1.y;
int b2 = (p4.y - p3.y) * p3.x + (p3.x - p4.x) * p3.y;
int D = (p2.x - p1.x) * (p4.y - p3.y) - (p4.x - p3.x) * (p2.y - p1.y);
int D1 = b2 * (p2.x - p1.x) - b1 * (p4.x - p3.x);
int D2 = b2 * (p2.y - p1.y) - b1 * (p4.y - p3.y);
p.x = D1 / D;
p.y = D2 / D;
return p;
}
void generateIntersectPoints(Pg &pyclip, Pg &py, list<IP> &iplist)
{
int clipSize = pyclip.pts.size(), pySize = py.pts.size();
for (int i = 0; i < clipSize; i++)
{
Point p1 = pyclip.pts[i];
Point p2 = pyclip.pts[(i + 1) % clipSize];
for (int j = 0; j < pySize; j++)
{
Point p3 = py.pts[j];
Point p4 = py.pts[(j + 1) % pySize];
if (segmentsIntersect(p1, p2, p3, p4))
{
IP ip;
ip.index0 = j;
ip.index1 = i;
ip.p = getintersectPoint(p1, p2, p3, p4);
iplist.push_back(ip);
}
}
```

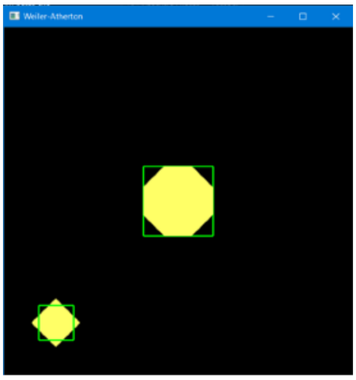
```
}
}
int getDistance(Point &p1, Point &p2)
{
return (p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y);
}
bool distanceComparator(IP &ip1, IP &ip2)
{
return ip1.dis < ip2.dis;
}
void generateList(Pg &py, list<IP> &iplist, list<IP> &comlist, int index)
{
int size = py.pts.size();
list<IP>::iterator it;
for (int i = 0; i < size; i++)
{
Point p1 = py.pts[i];
IP ip;
ip.pointFlag = 0;
ip.p = p1;
comlist.push_back(ip);
list<IP> oneSeg;
for (it = iplist.begin(); it != iplist.end(); it++)
{
if ((index == 0 && i == it->index0) ||
(index == 1 && i == it->index1))
{
it->dis = getDistance(it->p, p1);
it->pointFlag = 1;
oneSeg.push_back(
*it);
}
}
oneSeg.sort(distanceComparator);
for (it = oneSeg.begin(); it != oneSeg.end(); it++)
comlist.push_back(
*it);
}
}
void getPgPointInOut(list<IP> &Pglist, Pg &pyclip)
{
bool inFlag;
list<IP>::iterator it;
for (it = Pglist.begin(); it != Pglist.end(); it++)
{
if (it->pointFlag ==
0)
{
```

```
if (isPointInsidePg(it->p,
pyclip))
inFlag = true;
else
inFlag = false;
}
else
{
inFlag = !inFlag;
it->inFlag = inFlag;
}
}
}
bool operator==(Point &p1, Point &p2)
{
return p1.x == p2.x && p1.y == p2.y;
}
void getClipPointInOut(list<IP> &cliplist, list<IP> &Pglist)
{
list<IP>::iterator it, it1;
for (it = cliplist.begin(); it != cliplist.end(); it++)
{
if (it->pointFlag ==
0)
continue;
for (it1 = Pglist.begin(); it1 != Pglist.end(); it1++)
{
if (it1->pointFlag ==
0)
continue;
if (it->p ==
it1->p)
it->inFlag = it1->inFlag;
}
}
}
void generateClipArea(list<IP> &Pglist, list<IP> &cliplist)
{
list<IP>::iterator it, it1;
Pg py;
Color c = {0.0, 0.0, 1.0};
for (it = Pglist.begin(); it != Pglist.end(); it++)
if (it->pointFlag ==
1 &&
it->inFlag)
break;
py.pts.clear();
while (true)
```

```
{
if (it == Pglist.end())
break;
py.pts.push_back(it->p);
for (; it != Pglist.end(); it++)
{
if (it->pointFlag == 1 && !it->inFlag)
break;
py.pts.push_back(it->p);
}
for (it1 = cliplist.begin(); it1 != cliplist.end(); it1++)
if (it1->p == it->p)
break;
for (; it1 != cliplist.end(); it1++)
{
if (it1->pointFlag == 1 && it1->inFlag)
break;
py.pts.push_back(it1->p);
}
if (py.pts[0] == it1->p)
{
int size = py.pts.size();
for (int i = 0; i < size; ++i)
{
py.pts[i].x = nx_min + (py.pts[i].x - x_min) * scale_x;
py.pts[i].y = ny_min + (py.pts[i].y - y_min) * scale_y;
}
py.drawpoly();
py.pts.clear();
for (; it != Pglist.end(); it++)
if (it->pointFlag == 1 && it->inFlag)
break;
continue;
}
for (; it != Pglist.end(); it++)
if (it->p == it1->p)
break;
}
}
void weilerAtherton(Pg &pyclip, Pg &py)
{
list<IP> iplist, Pglist, cliplist;
generateIntersectPoints(pyclip, py, iplist);
generateList(py, iplist, Pglist, 0);
generateList(pyclip, iplist, cliplist, 1);
getPgPointInOut(Pglist, pyclip);
getClipPointInOut(cliplist, Pglist);
generateClipArea(Pglist, cliplist);
```

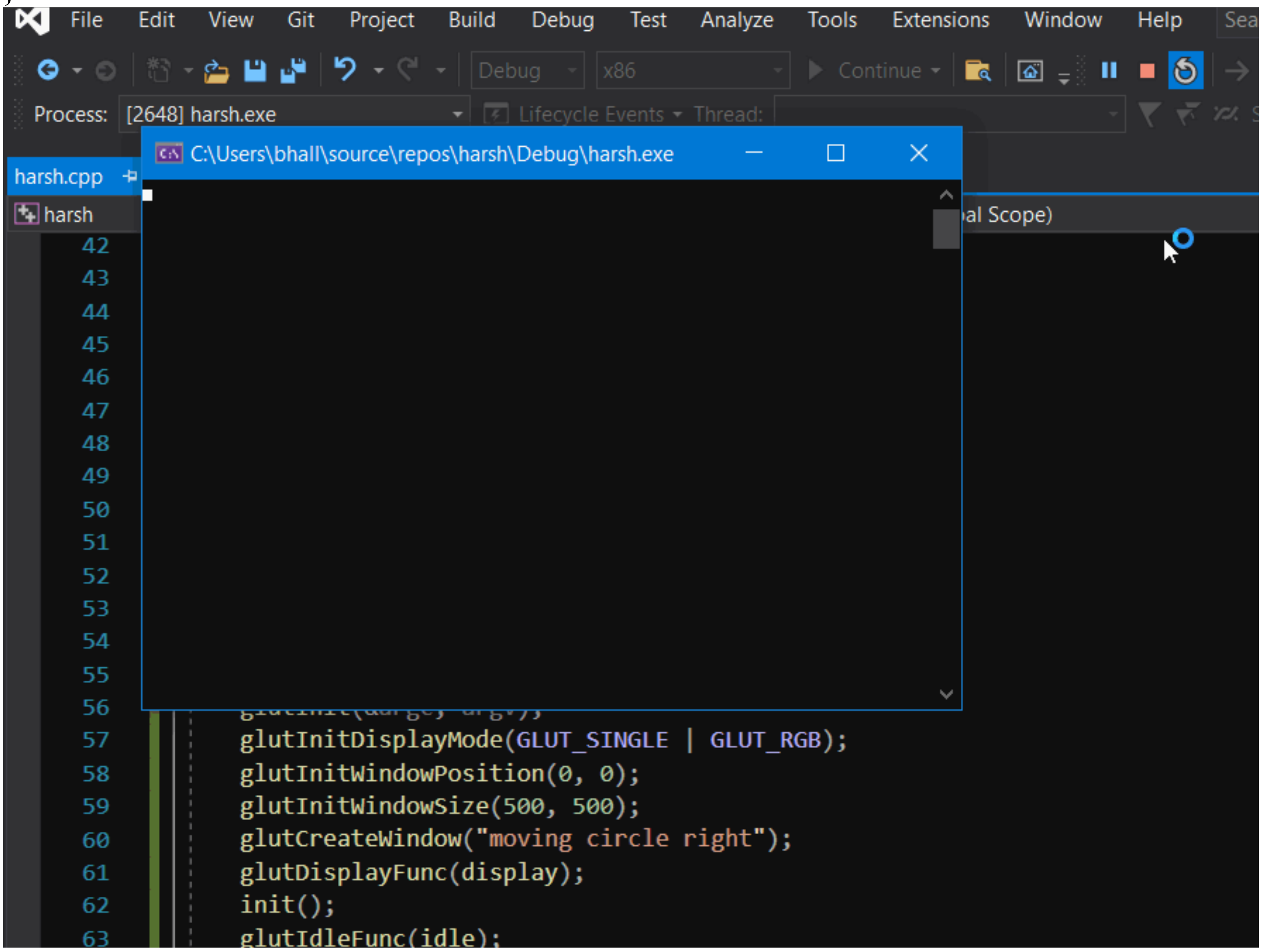
```
}
void init()
{
glClearColor(0.0, 0.0, 0.0, 0.0);
glColor3f(1.0, 0.0, 0.0);
glPointSize(1.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, 500, 0.0, 500);
}
void GenerateRandomSimplePg(Pg &G, int M)
{
Point P;
G.pts.clear();
for (int i = 0; i < M; ++i)
{
bool flag;
do
{
P.x = rand() % Size;
P.y = rand() % Size;
flag = true;
for (int j = 1; j < i - 1; ++j)
if (segmentsIntersect(G.pts[j - 1], G.pts[j], G.pts[i - 1], P))
{
flag = false;
break;
}
if (flag && i == M - 1)
{
for (int j = 2; j < i; ++j)
if (segmentsIntersect(G.pts[j - 1], G.pts[j], P, G.pts[0]))
{
flag = false;
break;
}
}
} while (!flag);
G.pts.push_back(P);
}
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
glEnable(GL_POINT_SMOOTH);
Pg pyclip, py, new_pyclip;
Point p1, p2, p3, p4;
p1.x = 50, p1.y = 50;
```

```
p2.x = 50, p2.y = 100;
p3.x = 100, p3.y = 100;
p4.x = 100, p4.y = 50;
pyclip.pts.push_back(p1);
pyclip.pts.push_back(p2);
pyclip.pts.push_back(p3);
pyclip.pts.push_back(p4);
Point p5, p6, p7, p8;
p5.x = 40, p5.y = 75;
p6.x = 75, p6.y = 110;
p7.x = 110, p7.y = 75;
p8.x = 75, p8.y = 40;
py.pts.push_back(p5);
py.pts.push_back(p6);
py.pts.push_back(p7);
py.pts.push_back(p8);
Point p9, p10, p11, p12;
p9.x = 200, p9.y = 200;
p10.x = 200, p10.y = 300;
p11.x = 300, p11.y = 300;
p12.x = 300, p12.y = 200;
new_pyclip.pts.push_back(p9);
new_pyclip.pts.push_back(p10);
new_pyclip.pts.push_back(p11);
new_pyclip.pts.push_back(p12);
Color a = {1.0, 0.0, 0.0};
Color b = {0.0, 1.0, 0.0};
py.drawpoly();
pyclip.drawPgLine(b);
new_pyclip.drawPgLine(b);
weilerAtherton(pyclip, py);
glFlush();
}
int main(int argc, char **argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(Size, Size);
glutInitWindowPosition(100, 100);
glutCreateWindow("Weiler-Atherton");
glutDisplayFunc(display);
init();
glutMainLoop();
return 0;
}
```


Weiler-Atherton

20 (i) Circle moving from left to right and vice versa

```
#include <GL\glut.h>
#include <stdlib.h>
#include <math.h>
#include <cstdlib>
float current_angle = 0.0f;
float step_angle = 0.2f;
float center_x = 100.0f;
float center_y = 100.0f;
void init()
{
   glLoadIdentity();
   glClearColor(1.0, 1.0, 1.0, 1.0);
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D(0, 500, 0, 500);
   glMatrixMode(GL_MODELVIEW);
}
void circle(int x, int y)
{
   float th;
   glColor3f(0, 0, 1);
   glBegin(GL_POLYGON);
   for (int i = 0; i < 360; i++)
   {
      th = i * (3.1416 / 180);
      glVertex2f(x + 30 * cos(th), y + 30 * sin(th));
   }
}
void drawcircle()
{
   glPushMatrix();
   glTranslated(center_x, center_y, 0);
   glRotatef(current_angle, 0, 0, 1);
   current_angle += step_angle;
   glTranslated(-center_x, -center_y, 0);
   glColor3f(1.0f, 0.0f, 0.0f);
   circle(center_x, center_y);
   center_x += 0.2;
   if (current_angle > 360)
      current_angle = 0;
   glEnd();
   glPopMatrix();
}
void display()
{
   drawcircle();
   glFlush();
   glutSwapBuffers();
   glutPostRedisplay();
}
void idle()
{
   display();
}
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowPosition(0, 0);
   glutInitWindowSize(500, 500);
   glutCreateWindow("moving circle right");
   glutDisplayFunc(display);
   init();
   glutIdleFunc(idle);
   glutMainLoop();
   return 0;
}
```



```
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(500, 500);
    glutCreateWindow("moving circle right");
    glutDisplayFunc(display);
    init();
    glutIdleFunc(idle);
```

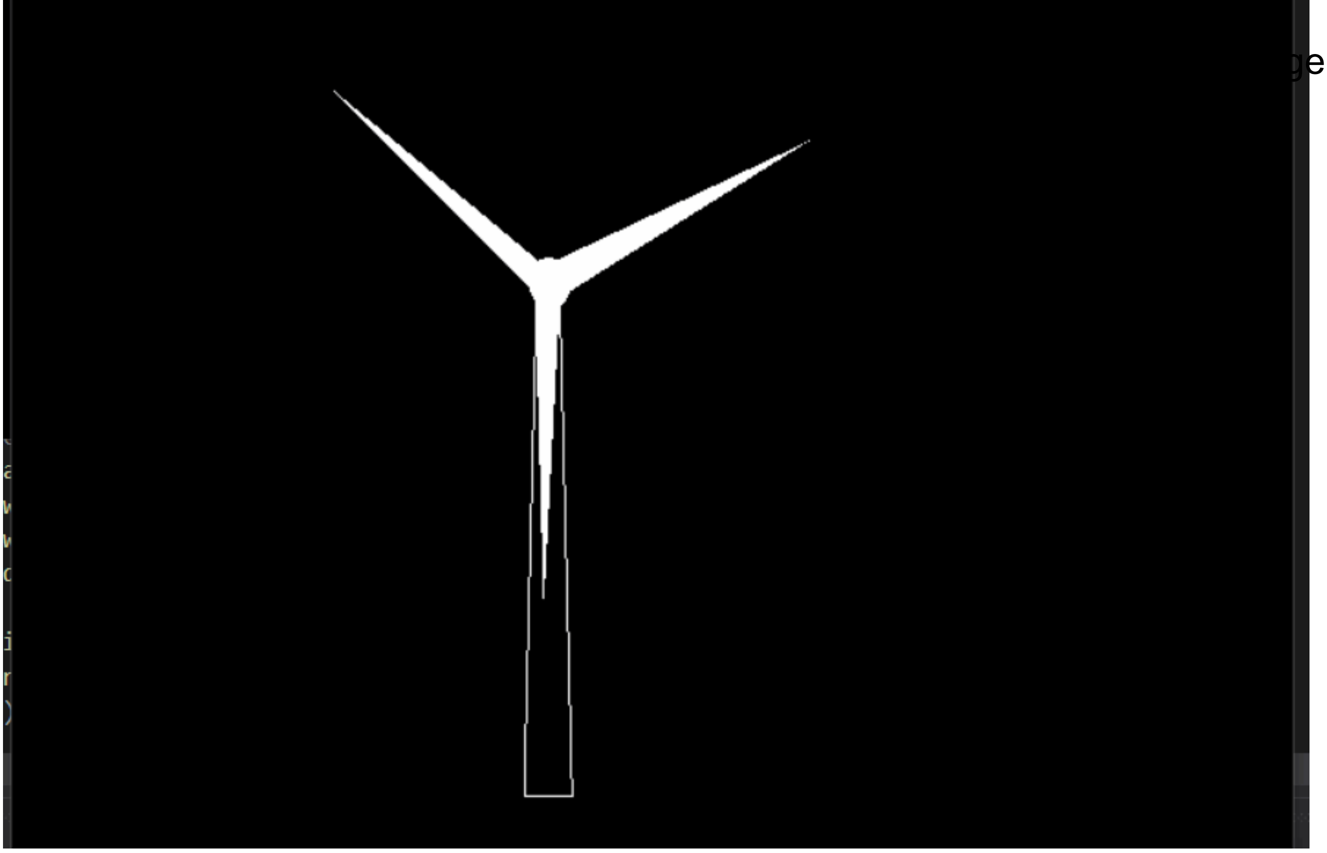
ii) Windmill

Rotation

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

#include <stdlib.h>

#include <math.h>

#include <cstdlib>


float current_angle = 0.0f;

float step_angle = 0.2f;

float center_x = 168.0f;

float center_y = 180.0f;

void init()

{

   glClearColor(0.0, 0.0, 0.0, 0.0);



glMatrixMode(GL_PROJECTION);

   gluOrtho2D(0.0, 400, 0.0, 300.0);

}

void circle(int x, int y)

{

   float th;

   glColor3f(1, 1, 1);

   glBegin(GL_POLYGON);

   for (int i = 0; i < 360; i++)


   {

      th = i * (3.1416 / 180);

      glVertex2f(x + 7 * cos(th), y +

6.5 * sin(th));
```

```
    }
}
void drawTurbine()
{
    glBegin(GL_LINE_LOOP);
    glColor3f(1.0, 1.0, 1.0);
    glVertex2f(164, 180);
    glVertex2f(160, 40);
    glVertex2f(175, 40);
    glVertex2f(171, 180);
    glEnd();
    glPushMatrix();
    glTranslatef(center_x, center_y,
0.0f);
    glRotatef(current_angle, 0, 0, 1);
    current_angle += step_angle;
    glTranslatef(
        -center_x,
        -center_y, 0.0f);
    glBegin(GL_TRIANGLES);
    glColor3f(1.0, 1.0, 1.0);
    glVertex2f(173, 180);
    glVertex2f(163, 180);
    glVertex2f(168, 270);
    glEnd();
    glBegin(GL_TRIANGLES);
    glColor3f(1.0, 1.0, 1.0);
    glVertex2f(170, 174);
    glVertex2f(175, 180);
    glVertex2f(247, 140);
    glEnd();
    glBegin(GL_TRIANGLES);
    glColor3f(1.0, 1.0, 1.0);
    glVertex2f(162, 180);
    glVertex2f(167, 174);
    glVertex2f(88, 140);
    glEnd();
    circle(168, 180);
    glEnd();
    glPopMatrix();
}
void display()
{

glClear(GL_COLOR_BUFFER_BIT);

glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    drawTurbine();
    glFlush();
    glutSwapBuffers();
    glutPostRedisplay();
}
void idle()
{
    display();
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
```

```
glutInitDisplayMode(GLUT_SINGLE
| GLUT_RGB);
   glutInitWindowSize(700, 600);
   glutInitWindowPosition(10, 10);
   glutCreateWindow("Wind
Turbine");
   init();
   glutIdleFunc(idle);
   glutDisplayFunc(display);
   glutMainLoop();
   return 0;
}
```



iii) Football goal

```
#include<windows.
h>#include<GL/
glut.h>
#include<GL/gl.h>
#include<GL/glu.h>
#include<cmath>
#include <iostream>
#include<stdlib.h>
#include
"Camera.h"
#include "Ball.h"
#include "Board.h"
//colors
 /* darkgrey = (0.4, 0.4, 0.4) -> 0.133

red = (1.0, 0.0, 0.0) -> 0.213
 green = (0.0, 1.0, 0.0) -> 0.715
 blue = (0.0, 0.0, 1.0) -> 0.072
 cyan = (0.0, 1.0, 1.0) -> 0.787
 magenta = (1.0, 0.0, 1.0) -> 0.285
 yellow = (1.0, 1.0, 0.0) -> 0.928
 white = (1.0, 1.0, 1.0) -> 1.000
 black = (0.0, 0.0, 0.0) -> 0.000
 darkred = (0.5, 0.0, 0.0) -> 0.046
 darkgreen = (0.0, 0.5, 0.0) -> 0.153
 darkblue = (0.0, 0.0, 0.5) -> 0.015
 darkcyan = (0.0, 0.5, 0.5) -> 0.169
darkmagenta = (0.5, 0.0, 0.5) -> 0.061
```

```
 darkyellow = (0.5, 0.5, 0.0) -> 0.199
 lightgrey = (0.8, 0.8, 0.8) -> 0.604
 */
//Global variables: a camera, a board and a
ballCamera camera;
int W=10,D=8;//for board's width and
depthBoard board(W,D);

int H = 2;//Height of the goal post
//Just initializes the ball
positiondouble rad=0.3;
Ball football= Ball(rad);//radius and coordinates of center of
ball128
double Ynew,Znew; //Gives the final position values for y and
zcoordinates. x-coordinate is
known and x=width-2

//Application-specific initialization: Set up global lighting
parametersand create display lists
void init()
{
glEnable(GL_DEPTH_TEST); //calculate depth value
glLightfv(GL_LIGHT0,GL_DIFFUSE,WHITE); //sets
lightproperties i.e. light no,
parameter(here diffuse light), and color
glLightfv(GL_LIGHT0,GL_SPECULAR,WHITE
);
glMaterialfv(GL_FRONT,GL_SHININESS,RED); //sets material
properties i.e. face
being lit, parameter, color
glMaterialf(GL_FRONT,GL_SHININESS,50); //50 gives
thespecular exponent of
```

```
        material
glEnable(GL_LIGHTING); //enables lighting
glEnable(GL_LIGHT0); //enables particular dynamic light, here0
        board.create();

        }

        //Draws one frame, the play field then the ball from the
        currentcamera position

        void display()

        {

        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT|
        GL_ACCUM_BU

        FFER_BIT); //clears buffers

        glLoadIdentity(); //loads identity

        matrix

        gluLookAt(camera.getX(),camera.getY(),camera.getZ(),board.cent
        erx(),0.0,board.cen

        terz(),0.0,1.0,0.0);

         //creates a viewing

        matrixboard.draw();

        129

         if(football.state == 0)

         {

         football.stop();//Set ball's position to its initial position

         }

         else if (football.state==1)

        {

         football. NewPosSet(Ynew,Znew);//Sets new position (x,y,z)
        withnew values of y and

        z.

         football.motion();//Uses linear equations y on x and z on x to
        giveball a random motion

        }

        else if(football.state==2)
```

```
{
 football.jerkback();//Used to jerk back the ball when it touches
thenet
}
else
 {
 football.gravity(); //bring the ball to the ground
 }
 football.make();//Creates the ball whose center is at a
position(x,y,z) initially
glFlush(); //empties all buffers;forces execution of GL commands
infinite
time
glutSwapBuffers(); //flips back buffer with front buffer
}
//On reshape, constructs a camera that perfectly fits the
windowvoid reshape(GLint w,GLint h)
{ 1
30
glViewport(0,0,w,h); //set the viewport rectangle for the
currentOpenGL
context
glMatrixMode(GL_PROJECTION); //set matrix mode to
projectionglLoadIdentity();
gluPerspective(60.0, GLfloat(w)/GLfloat(h), 0.5 , 200.0); //
initializethe projection
matrix to a perspective projection matrix.
glMatrixMode(GL_MODELVIEW); //st matrix to model view
whichis a
combination of view and model (or world) matrix transformation
}
```

```
//Requests to draw the next
framevoid timer(int v)
{
glutPostRedisplay(); //to tell GLUT that we are ready to
renderanother frame
glutTimerFunc(1000/60,timer,v); //registers a timer callback to
betriggered in
1000/60 milliseconds.
}
//Gives random number between two double variables M and
Ndouble getRand(double M, double N)
{
 return M + (rand() / ( RAND_MAX / (N-M) ) ) ;
}
//Moves the camera according to the key pressed, then ask to
refreshthe display.
void special(int key,int, int)
{
switch(key)
{
case
GLUT_KEY_LEFT:
camera.moveLeft();
break;
case
GLUT_KEY_RIGHT:
camera.moveRight();
break;
case
GLUT_KEY_UP:
camera.moveUp();
break;
case GLUT_KEY_DOWN:
camera.moveDown(
);break;
 case GLUT_KEY_HOME: //Used to give motion to the
 ballfootball.stop();
 football.state=1;
 Ynew = getRand(rad+0.5,H-rad);//get random value between
radiusof the ball and
height of pole-radius (in y axis)
 Znew = getRand(rad+1,D+rad-3);//get random value between
radiusof ball + 1 and
sum of total length of base and radius (in z axis)
//total base length of the goal post is
 depth-3break;
 case GLUT_KEY_END: //Used to bring ball to initial
 positionfootball.stop();
 break;
 ;
}
glutPostRedisplay();
}
//Initialize GLUT and enters the main
loopint main()
{
glutInitDisplayMode(GLUT_DOUBLE|
GLUT_RGB|GLUT_DEPTH); //initialize
the
display mode; GLUT_DOUBLE- window will be double buffered
glutInitWindowPosition(100,100); //initialize the position of
thenewly created
window
```

glutInitWindowSize(1000,800); //initialize the size of the newlycreated window

glutCreateWindow("Penalty Kick"); //create the render windowusing the parameters

we have specified before

//glClearColor(0, 0, 0,1.0); //Change Background color(RGBA)

glutDisplayFunc(display); //register the callbacks for the GULTevent system.

glutReshapeFunc(reshape);

glutSpecialFunc(special); //Here the glutSpecial refers to keyboardfunction of

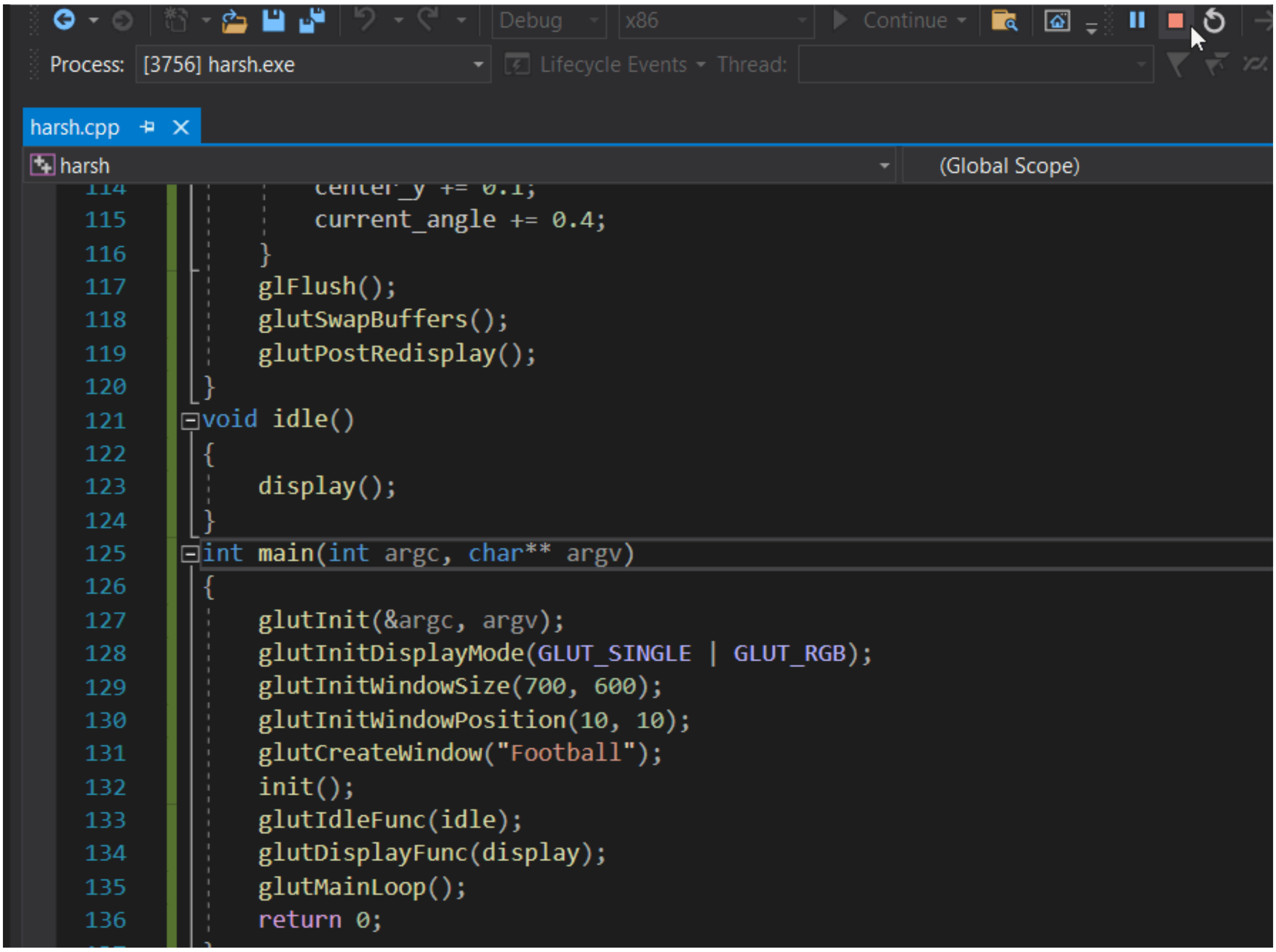
Opengl.

glutTimerFunc(100, timer,

0);init();

glutMainLoop(); //starts the GLUT event processing

loopreturn 0;}



Process: [3756] harsh.exe

```
        center_y += 0.1;
        current_angle += 0.4;
    }
    glFlush();
    glutSwapBuffers();
    glutPostRedisplay();
}
void idle()
{
    display();
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(700, 600);
    glutInitWindowPosition(10, 10);
    glutCreateWindow("Football");
    init();
    glutIdleFunc(idle);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
```