

**MADHAV INSTITUTE OF TECHNOLOGY AND
SCIENCE, GWALIOR**



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PRACTICAL FILE
OF
COMPUTER GRAPHICS
(150313)

SUBMITTED BY:

ASMITA JAIN

0901EO201017

SUBMITTED TO:

PROF. HEMLATA ARYA

INDEX

SR. NO.	TITLE	PAGE NO.
01	Installation and Introduction to OpenGL basics, graphic functions, commands for compiling and executing an OpenGL Program.	03
02	Write an OpenGL Program to create an output window, to plot a point with given coordinates and other basic demonstrations.	04
03	WAP which displays a triangle in 2D by using OpenGL.	06
04	Write an OpenGL Program to implement DDA Line Drawing Algorithm.	08
05	Write an OpenGL Program to implement Bresenham's Line Algorithm.	12
06	Write an OpenGL Program to implement Mid-Point Circle Algorithm.	19
07	Write an OpenGL Program to implement following 2D transformations: I. Translation of a polygon. ii. Scaling of a polygon. iii. Rotation of a polygon.	23
08	Write an OpenGL Program to implement: i. Flood Filling Algorithm using polygon. ii. Boundary Filling Algorithm using polygon.	36

01) Installation and Introduction to OpenGL basics, graphic functions, commands for compiling and executing an OpenGL Program.

OpenGL is the most widely adopted 2D and 3D graphics API in the industry, bringing thousands of applications to a wide variety of computer platforms. It is window-system and operating-system independent as well as network-transparent. OpenGL enables developers of software for PC, workstation, and supercomputing hardware to create high-performance, visually compelling graphics software applications, in markets such as CAD, content creation, energy, entertainment, game development, manufacturing, medical, and virtual reality. OpenGL exposes all the features of the latest graphics hardware.

02) Write an OpenGL Program to create an output window, to plot a point with given coordinates and other basic demonstrations.

```
# include <GL/glut.h>
void display()
{
    glColor3f(0.7, 0.3, 0.9);
    glPointSize(10.0);
    glBegin(GL_POINTS);
    glColor3f(0.5, 0.8, 0.7);
    glVertex2i(70, 80);
    glVertex2i(20, 70);
    glEnd();
    glFlush();
}

void init(void)
{
    glClearColor(0, 0, 0, 0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-100.0, 100.0, -100.0, 100.0, -1.0, 1.0);
}

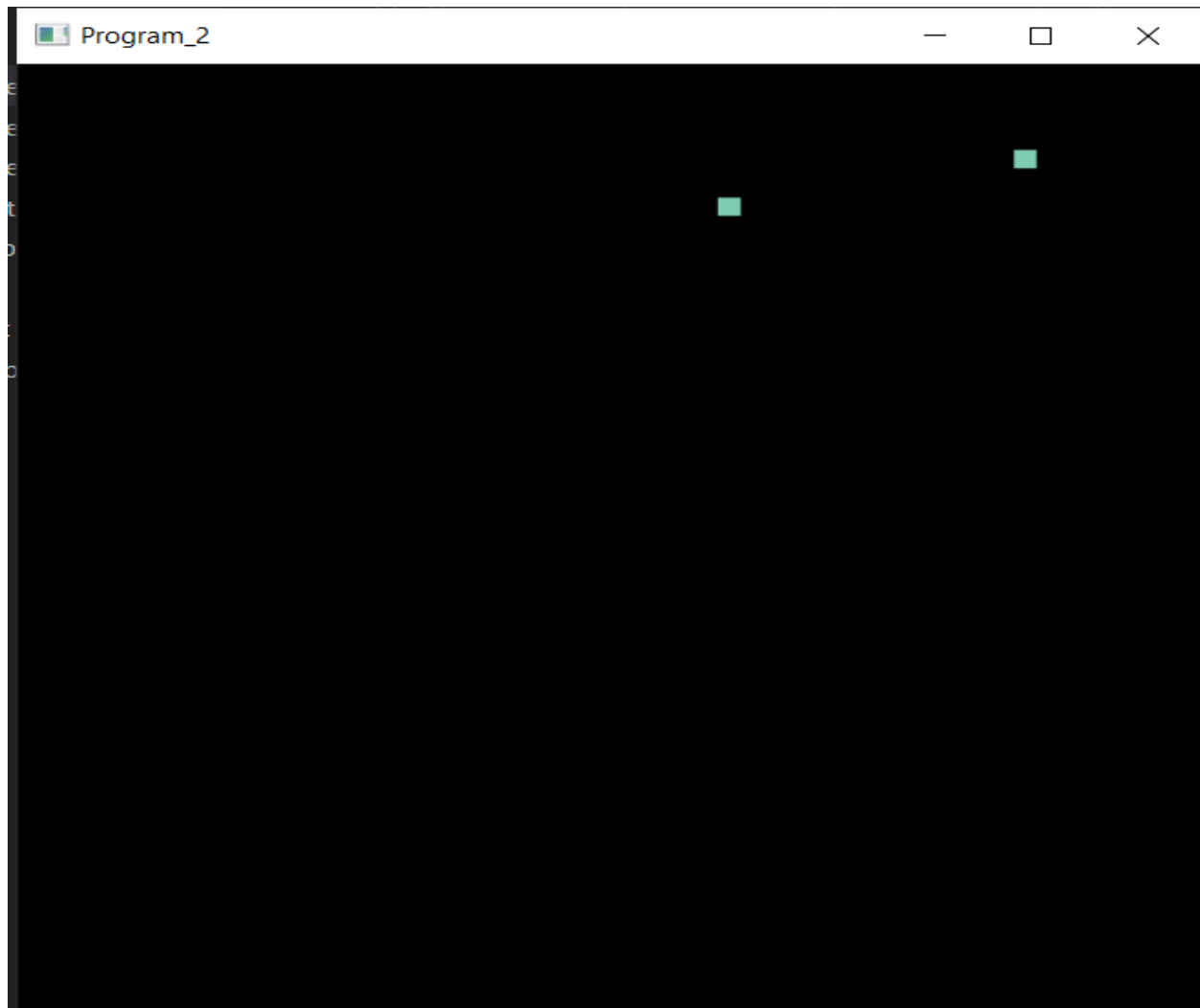
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutInitWindowPosition(121, 121);
    glutInitWindowSize(521, 521);

    glutCreateWindow("Program_2");
    init();
    glutDisplayFunc(display);

    glutMainLoop();

    return 0;
}
```

Output:



03) WAP which displays a triangle in 2D by using OpenGL.

```
#ifdef __APPLE_CC__
#include <GLUT/glut.h>
#include <GL/glut.h>

void display() {

    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
        glColor3f(1, 0, 0); glVertex3f(-0.6, -0.75, 0.5);
        glColor3f(0, 1, 0); glVertex3f(0.6, -0.75, 0);
        glColor3f(0, 0, 1); glVertex3f(0, 0.75, 0);
    glEnd();

    glFlush();
}

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

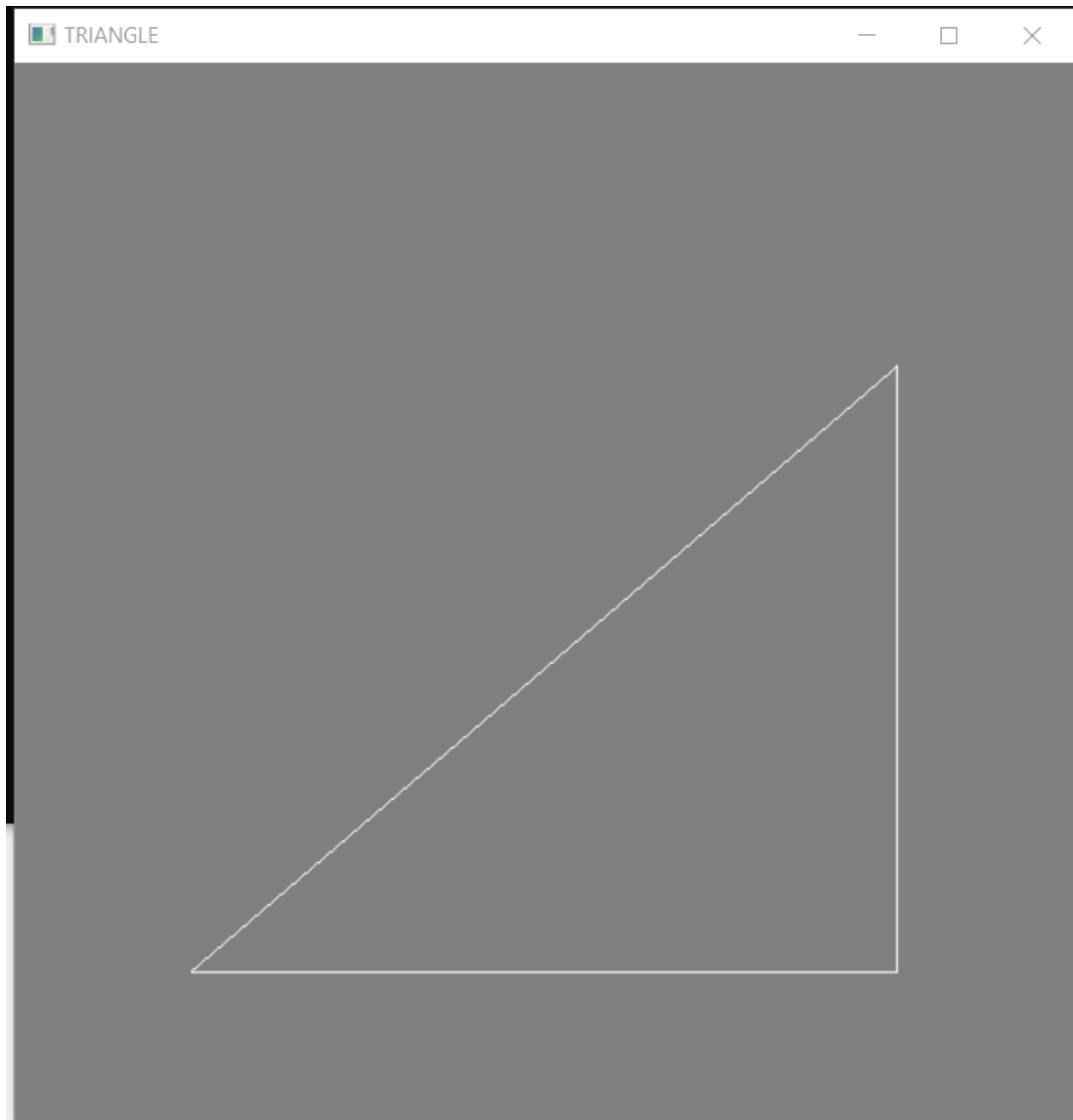
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

    glutInitWindowPosition(80, 80);
    glutInitWindowSize(400, 300);
    glutCreateWindow("A Simple Triangle");

    glutDisplayFunc(display);

    glutMainLoop();
}
```

Output:



04) Write an OpenGL Program to implement DDA Line Drawing Algorithm.

```
#include <math.h>
#include <GL/glut.h>
#include<stdlib.h>
#include<stdio.h>
double X1, Y1, X2, Y2;
float round_value(float v)
{ return floor(v + 0.5); }
void lineDDA(void)
{
    double dx=(X2-X1);
    double dy=(Y2-Y1);
    double steps;
    float xInc,yInc,x=X1,y=Y1;
    steps=(abs(dx)>abs(dy))?(abs(dx)):(abs(dy));
    xInc=dx/(float)steps;
    yInc=dy/(float)steps;
    glClear(GL_COLOR_BUFFER_BIT);

    /* Plot the points */
    glBegin(GL_POINTS);
```



```

/* Plot the first point */
glVertex2d(x,y);

int k;

/* For every step, find an intermediate vertex */
for(k=0;k<steps;k++)
{
    x+=xInc;
    y+=yInc;
    /* printf("%0.6lf %0.6lf\n",floor(x), floor(y)); */
    glVertex2d(round_value(x), round_value(y));
}

glEnd();
glFlush();
}

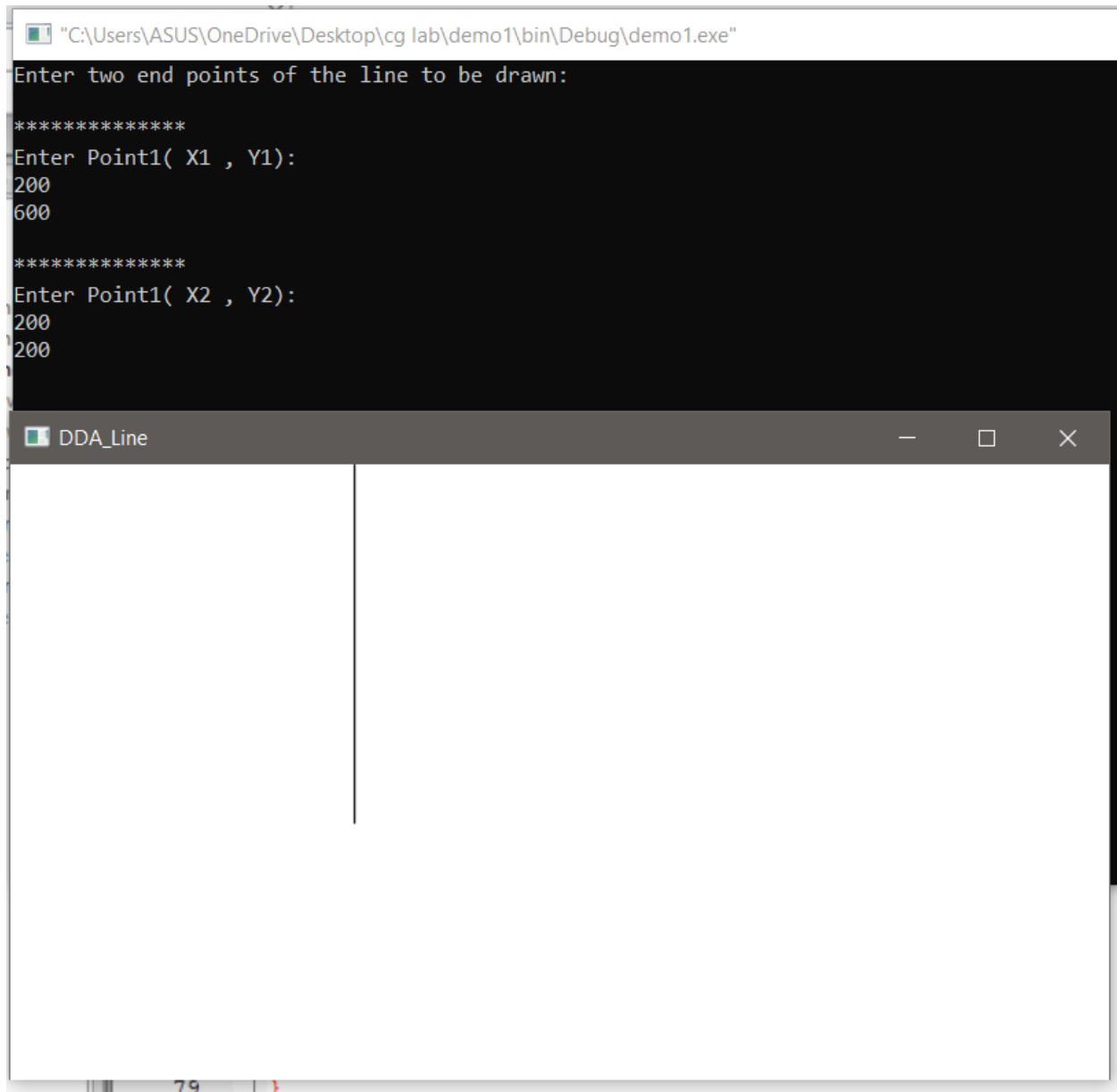
void Init()
{
    glClearColor(1.0,1.0,1.0,0);
    glColor3f(0.0,0.0,0.0);
    gluOrtho2D(0 , 640 , 0 , 480);
}

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

```

```
printf("Enter two end points of the line to be drawn:\n");
printf("\n*****");
printf("\nEnter Point1( X1 , Y1):\n");
scanf("%lf%lf",&X1,&Y1);
printf("\n*****");
printf("\nEnter Point1( X2 , Y2):\n");
scanf("%lf%lf",&X2,&Y2);
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowPosition(0,0);
glutInitWindowSize(640,480);
/* Create the window with title "DDA_Line" */
glutCreateWindow("DDA_Line");
/* Initialize drawing colors */
Init();
/* Call the displaying function */
glutDisplayFunc(lineDDA);
/* Keep displaying untill the program is closed */
glutMainLoop();
}
```

Output:



The screenshot displays two overlapping windows. The top window, titled "C:\Users\ASUS\OneDrive\Desktop\cg lab\demo1\bin\Debug\demo1.exe", is a command prompt with a black background and white text. It prompts the user to "Enter two end points of the line to be drawn:". After a separator line of asterisks, it asks for "Enter Point1(X1 , Y1):" and receives the input "200" followed by "600". Another separator line of asterisks follows, and it then asks for "Enter Point1(X2 , Y2):" with the input "200" followed by "200". The bottom window, titled "DDA_Line", is a graphics application with a white background. It shows a single vertical line drawn on the left side of the canvas, representing the line segment from (200, 600) to (200, 200).

```
"C:\Users\ASUS\OneDrive\Desktop\cg lab\demo1\bin\Debug\demo1.exe"
Enter two end points of the line to be drawn:

*****
Enter Point1( X1 , Y1):
200
600

*****
Enter Point1( X2 , Y2):
200
200
```

DDA_Line

05) Write an OpenGL Program to implement Bresenham's Line Algorithm.

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

int x1, y1, x2, y2;

void myInit()
{

glClear(GL_COLOR_BUFFER_BIT);
glClearColor (0.0, 1.0, 1.0,1.0);
//glClearColor(0.0, 0.0, 0.0, 1.0);
glMatrixMode(GL_PROJECTION);
gluOrtho2D(0, 500, 0, 500);

}
```

```
void draw_pixel(int x, int y)

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

void draw_line(int x1, int x2, int y1, int y2)
{
    int dx, dy, i, e;
    int incx, incy, inc1, inc2;
    int x,y;
    dx = x2-x1;
    dy = y2-y1;
    if (dx < 0) dx = -dx;
    if (dy < 0) dy = -dy;
    incx = 1;
    if (x2 < x1) incx = -1;
    incy = 1;
```

```
if (y2 < y1) incy = -1;
```

```
x = x1; y = y1;
```

```
if (dx > dy)
```

```
{
```

```
draw_pixel(x, y);
```

```
e = 2 * dy-dx;
```

```
inc1 = 2*(dy-dx);
```

```
inc2 = 2*dy;
```

```
for (i=0; i<dx; i++)
```

```
{
```

```
if (e >= 0)
```

```
{
```

```
y += incy;
```

```
e += inc1;
```

```
}
```

```
else
```

```
e += inc2;
```

```
x += incx;
```

```
draw_pixel(x, y);
```

```
}
```

```
}
```

```
else
```

```
{
```

```
draw_pixel(x, y);
```

```
e = 2*dx-dy;
```

```
inc1 = 2*(dx-dy);
```

```
inc2 = 2*dx;
```

```
for (i=0; i<dy; i++)
```

```
{
```

```
if (e >= 0)
```

```
{
```

```
x += incx;
```

```
e += inc1;
```

```
}
```

```
else
```

```
e += inc2;
```

```
y += incy;
```

```

draw_pixel(x, y);
}
}
}

void myDisplay()
{
glClear(GL_COLOR_BUFFER_BIT);
glClearColor (0.0, 1.0, 1.0,1.0); // for window's color
glColor3f(1,0,0); //for line's color
glLineWidth(10);
draw_line(x1, x2, y1, y2);
glFlush();
}

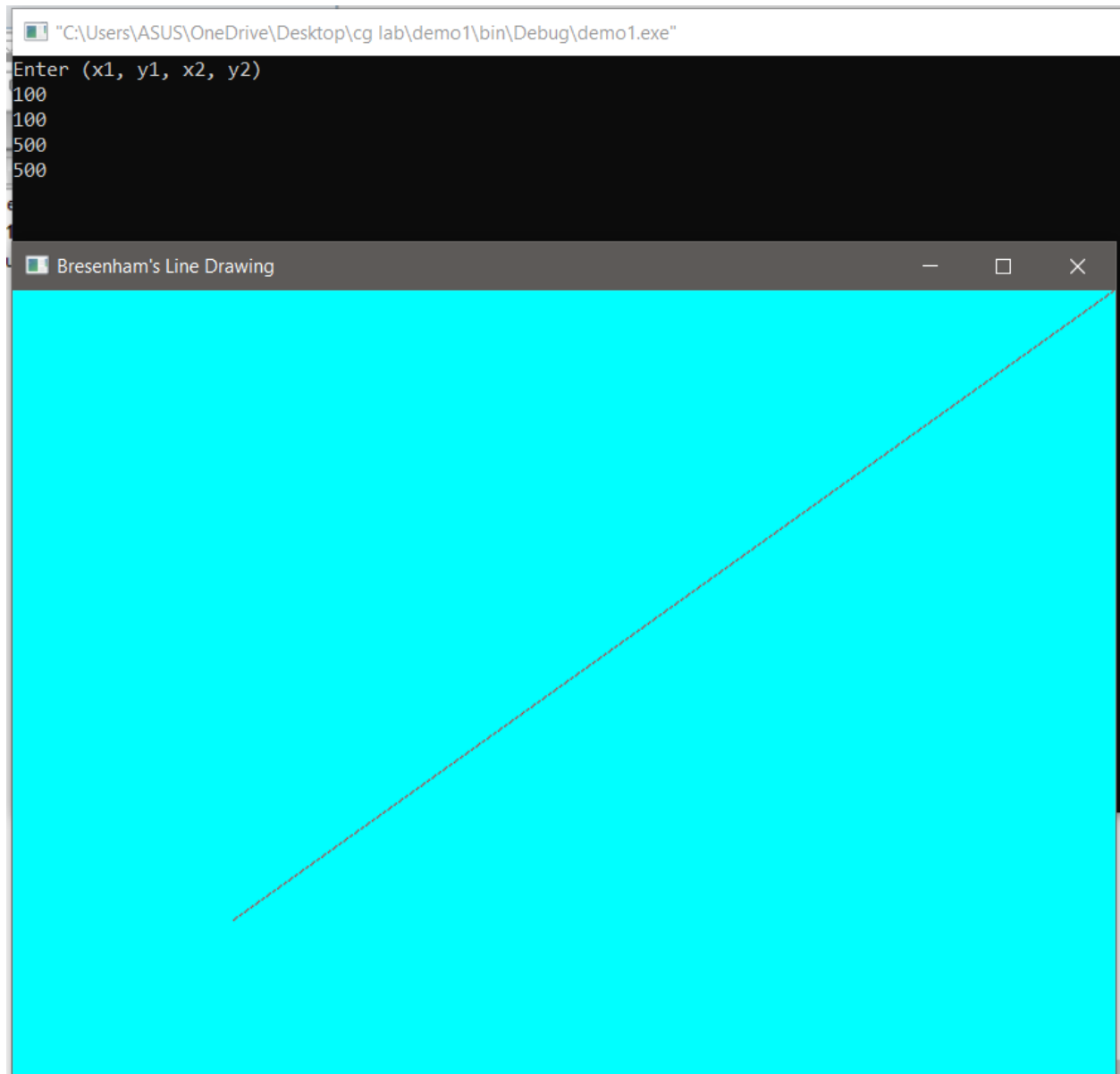
int main(int argc, char **argv)
{
printf( "Enter (x1, y1, x2, y2)\n");
scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);

```



```
glutInitWindowSize(700, 500);  
glutInitWindowPosition(0, 0);  
glutCreateWindow("Bresenham's Line Drawing");  
myInit();  
glutDisplayFunc(myDisplay);  
glutMainLoop();  
  
return 0;  
}
```

Output:



06) Write an OpenGL Program to implement Mid-Point Circle Algorithm.

```
#include <gl/glut.h>
#include <Windows.h>
#include <iostream>
using namespace std;
/*
cOded by (C) Ajith Kp (C) (R) TERMINAL_CODERS (R)
*/
void circle() {
    glColor3f(1.0, 0.0, 0.0);
    glPointSize(2.0);
    float r = 100;
    float x = 0, y = r;
    float p = 1 - r;
    glBegin(GL_POINTS);
    while (x != y)
    {
        x++;
        if (p < 0) {
            p += 2 * (x + 1) + 1;
```

```

    }
    else {
        y--;
        p += 2 * (x + 1) + 1 - 2 * (y - 1);
    }

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

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

}

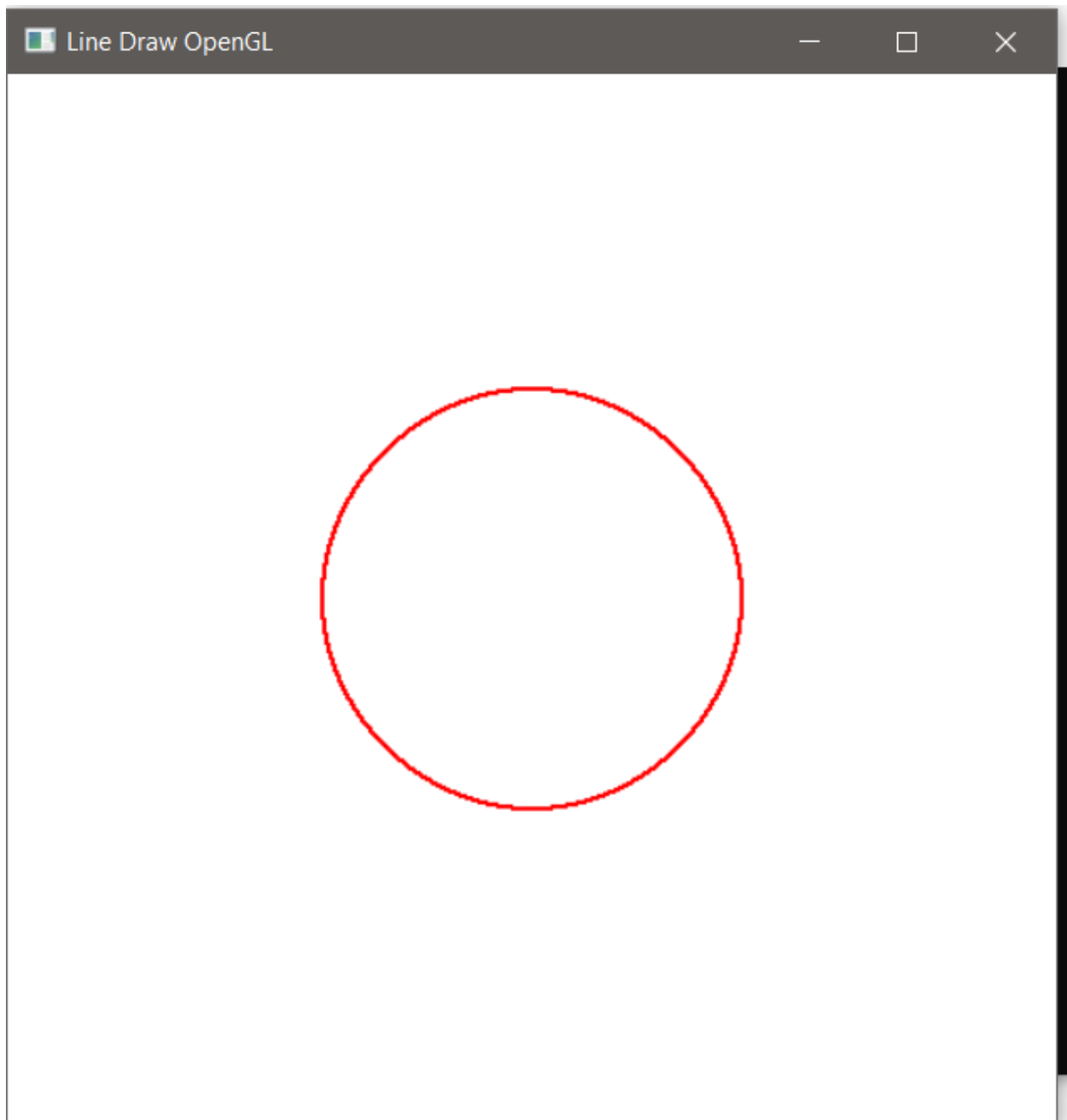
glEnd();
glFlush();
}

int main(int argc, char ** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

```

```
glutInitWindowSize(500, 500);  
glutInitWindowPosition(100, 100);  
glutCreateWindow("Line Draw OpenGL");  
  
glClearColor(1.0, 1.0, 1.0, 1.0);  
glClear(GL_COLOR_BUFFER_BIT);  
gluOrtho2D(-250, 250, -250, 250);  
glMatrixMode(GL_PROJECTION);  
glViewport(0, 0, 500, 500);  
  
glutDisplayFunc(circle);  
glutMainLoop();  
return 0;  
}
```

Output:



07) Write an OpenGL Program to implement following 2D transformations:

i. Translation of a polygon.

ii. Scaling of a polygon.

iii. Rotation of a polygon.

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

int pntX1, pntY1, choice = 0, edges;
vector<int> pntX;
vector<int> pntY;
int transX, transY;
double scaleX, scaleY;
double angle, angleRad;
char reflectionAxis, shearingAxis;
int shearingX, shearingY;
```

```
double round(double d)
```

```
{
```

```
    return floor(d + 0.5);
```

```
}
```

```
void drawPolygon()
```

```
{
```

```
    glBegin(GL_POLYGON);glLineWidth(9);
```

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

```
    for (int i = 0; i < edges; i++)
```

```
    {
```

```
        glVertex2i(pntX[i], pntY[i]);
```

```
    }
```

```
    glEnd();
```

```
}
```

```
void drawPolygonTrans(int x, int y)
```

```
{
```

```
    glBegin(GL_POLYGON);
```

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

```
    for (int i = 0; i < edges; i++)
```

```
    {
```



```

        glVertex2i(pntX[i] + x, pntY[i] + y);
    }
    glEnd();
}

void Axes(void)
{
    glColor3f (0.0, 0.0, 0.0);        // Set the color to BLACK
    glBegin(GL_LINES);                // Plotting X-Axis
    glVertex2s(-1000 ,0);
    glVertex2s( 1000 ,0);
    glEnd();
    glBegin(GL_LINES);                // Plotting Y-Axis
    glVertex2s(0 ,-1000);
    glVertex2s(0 , 1000);
    glEnd();
}

void drawPolygonScale(double x, double y)
{
    glBegin(GL_POLYGON);
    glColor3f(0.0, 0.0, 1.0);
    for (int i = 0; i < edges; i++)
    {

```

```

        glVertex2i(round(pntX[i] * x), round(pntY[i] * y));
    }
    glEnd();
}

void drawPolygonRotation(double angleRad)
{
    glBegin(GL_POLYGON);
    glColor3f(0.0, 0.0, 1.0);
    for (int i = 0; i < edges; i++)
    {
        glVertex2i(round((pntX[i] * cos(angleRad)) - (pntY[i] *
sin(angleRad))), round((pntX[i] * sin(angleRad)) + (pntY[i] *
cos(angleRad))));
    }
    glEnd();
}

void drawPolygonMirrorReflection(char reflectionAxis)
{
    glBegin(GL_POLYGON);
    glColor3f(0.0, 0.0, 1.0);

    if (reflectionAxis == 'x' || reflectionAxis == 'X')

```

```

    {
        for (int i = 0; i < edges; i++)
        {
            glVertex2i(round(pntX[i]), round(pntY[i] * -1));

        }
    }
    else if (reflectionAxis == 'y' || reflectionAxis == 'Y')
    {
        for (int i = 0; i < edges; i++)
        {
            glVertex2i(round(pntX[i] * -1), round(pntY[i]));

        }
    }
    glEnd();
}

void drawPolygonShearing()
{
    glBegin(GL_POLYGON);
    glColor3f(0.0, 0.0, 1.0);

    if (shearingAxis == 'x' || shearingAxis == 'X')

```

```

    {
        glVertex2i(pntX[0], pntY[0]);

        glVertex2i(pntX[1] + shearingX, pntY[1]);
        glVertex2i(pntX[2] + shearingX, pntY[2]);

        glVertex2i(pntX[3], pntY[3]);
    }
    else if (shearingAxis == 'y' || shearingAxis == 'Y')
    {
        glVertex2i(pntX[0], pntY[0]);
        glVertex2i(pntX[1], pntY[1]);

        glVertex2i(pntX[2], pntY[2] + shearingY);
        glVertex2i(pntX[3], pntY[3] + shearingY);
    }
    glEnd();
}

void myInit(void)
{
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glColor3f(0.0f, 0.0f, 0.0f);

```

```
    glPointSize(4.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-640.0, 640.0, -480.0, 480.0);
}

void myDisplay(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0);
    Axes();

    if (choice == 1)
    {
        drawPolygon();
        drawPolygonTrans(transX, transY);
    }
    else if (choice == 2)
    {
        drawPolygon();
        drawPolygonScale(scaleX, scaleY);
    }
    else if (choice == 3)
```

```

        {
            drawPolygon();
            drawPolygonRotation(angleRad);
        }
    else if (choice == 4)
    {
        drawPolygon();
        drawPolygonMirrorReflection(reflectionAxis);
    }
    else if (choice == 5)
    {
        drawPolygon();
        drawPolygonShearing();
    }

    glFlush();
}

int main(int argc, char** argv)
{
    cout << "Enter your choice:\n\n" << endl;

    cout << "1. Translation" << endl;

```

```

cout << "2. Scaling" << endl;
cout << "3. Rotation" << endl;
cout << "4. Mirror Reflection" << endl;
cout << "5. Shearing" << endl;
cout << "6. Exit" << endl;
cin >> choice;
if (choice == 6) {
    return 0;
}
cout << "\n\nFor Polygon:\n" << endl;
cout << "Enter no of edges: "; cin >> edges;

for (int i = 0; i < edges; i++)
{
    cout << "Enter co-ordinates for vertex " << i + 1 << " : "; cin
>> pntX1 >> pntY1;
    pntX.push_back(pntX1);
    pntY.push_back(pntY1);
}
if (choice == 1)
{

```

```

        cout << "Enter the translation factor for X and Y: "; cin >>
transX >> transY;
    }
    else if (choice == 2)
    {
        cout << "Enter the scaling factor for X and Y: "; cin >> scaleX
>> scaleY;
    }
    else if (choice == 3)
    {
        cout << "Enter the angle for rotation: "; cin >> angle;
        angleRad = angle * 3.1416 / 180;
    }
    else if (choice == 4)
    {
        cout << "Enter reflection axis ( x or y ): "; cin >>
reflectionAxis;
    }
    else if (choice == 5)
    {
        cout << "Enter reflection axis ( x or y ): "; cin >> shearingAxis;
        if (shearingAxis == 'x' || shearingAxis == 'X')
        {

```



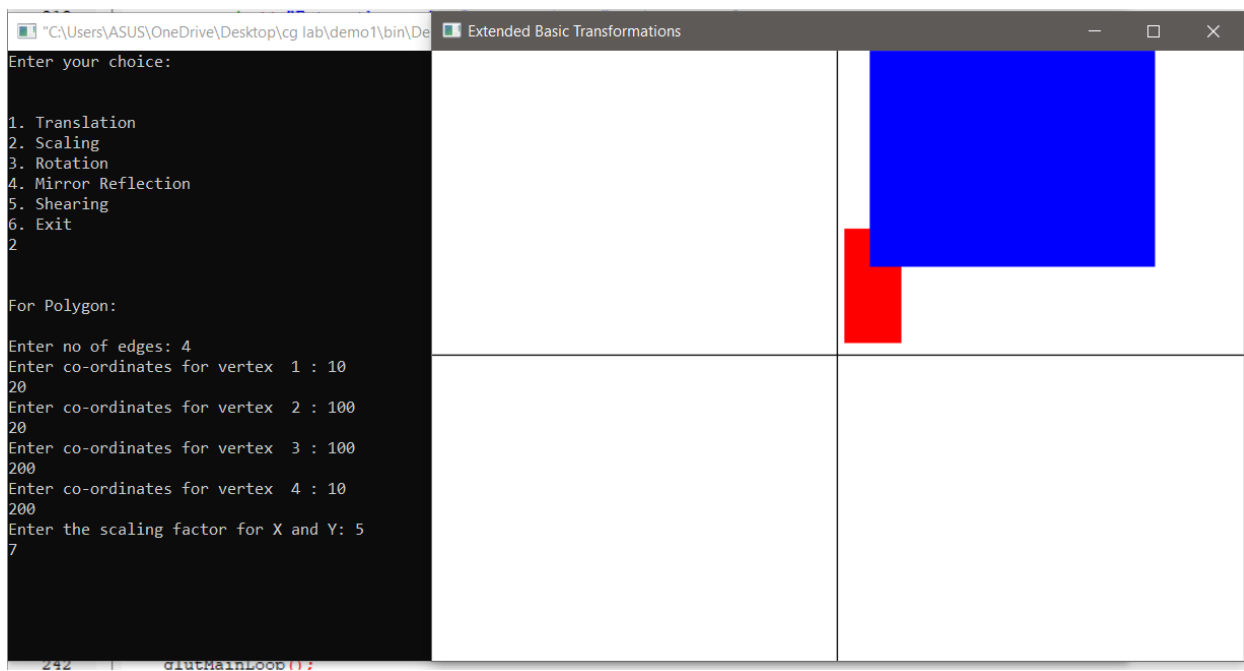
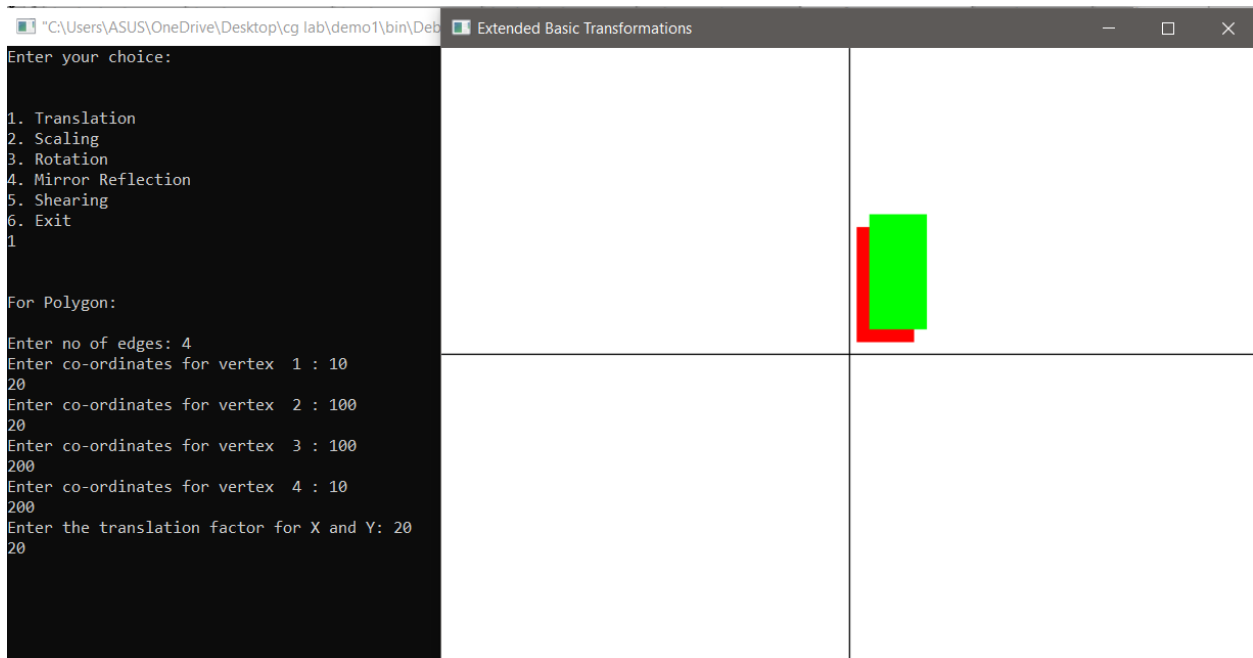
```

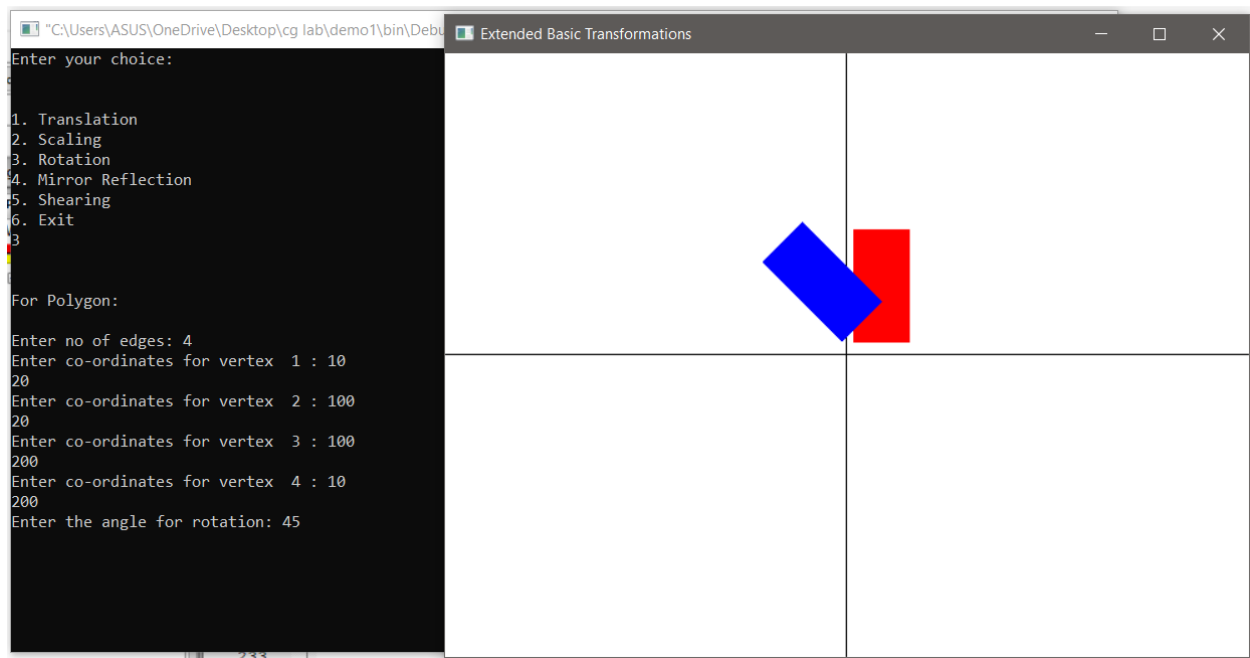
        cout << "Enter the shearing factor for X: "; cin >>
shearingX;
    }
    else
    {
        cout << "Enter the shearing factor for Y: "; cin >>
shearingY;
    }
}

glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(640, 480);
glutInitWindowPosition(100, 150);
glutCreateWindow("Extended Basic Transformations");
glutDisplayFunc(myDisplay);
myInit();
glutMainLoop();
}

```

Output:





08) Write an OpenGL Program to implement:

i. Flood Filling Algorithm using polygon.

ii. Boundary Filling Algorithm using polygon.

i)

```
#include<stdio.h>
```

```
#include<conio.h>
```

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

```
#include<dos.h>
```

```
void flood(int,int,int,int);
```

```
void main()
```

```
{
```

```
    intgd=DETECT,gm;
```

```
    initgraph(&gd,&gm,"C:/TURBOC3/bgi");
```

```
    rectangle(50,50,250,250);
```

```
    flood(55,55,10,0);
```

```
    getch();
```

```
}
```

```
void flood(intx,inty,intfillColor, intdefaultColor)
```

```
{
```

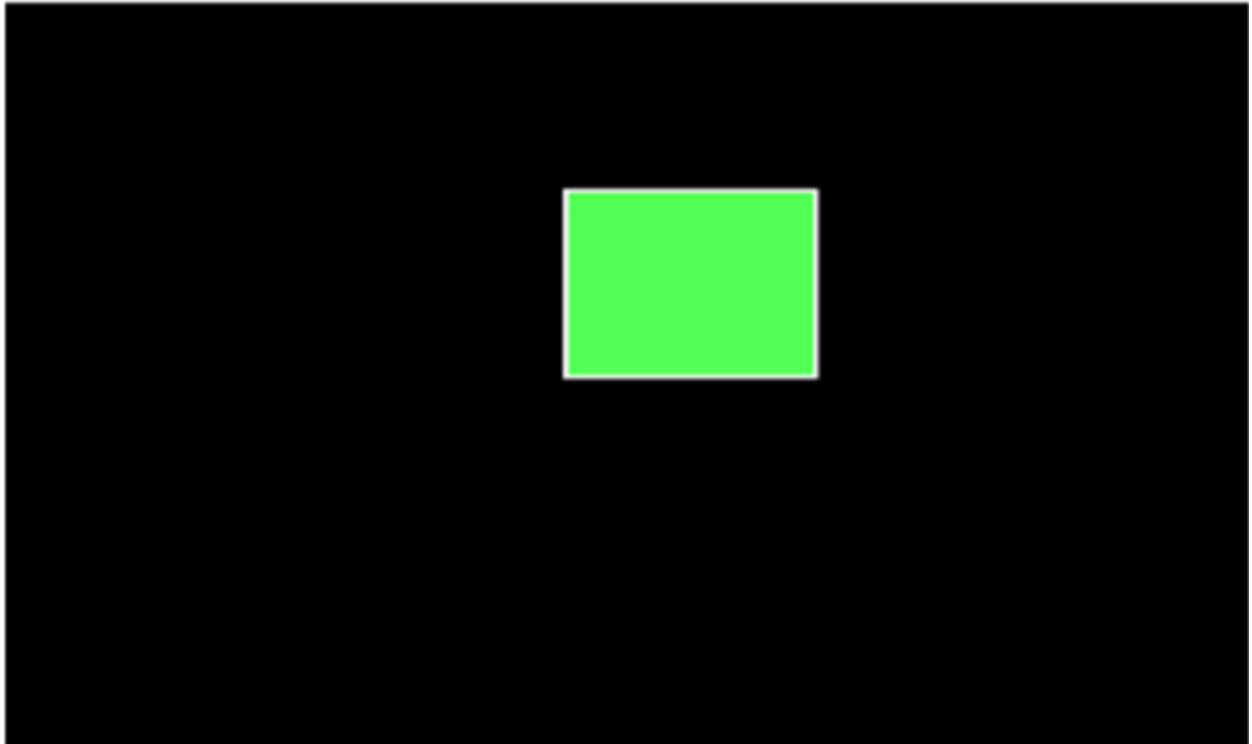
```
    if(getpixel(x,y)==defaultColor)
```

```
    {
```

```
        delay(1);
```

```
    putpixel(x,y,fillColor);  
    flood(x+1,y,fillColor,defaultColor);  
    flood(x-1,y,fillColor,defaultColor);  
    flood(x,y+1,fillColor,defaultColor);  
    flood(x,y-1,fillColor,defaultColor);  
}  
}
```

Output:



ii)

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

void boundaryFill8(int x, int y, int fill_color,int boundary_color)
{
    if(getpixel(x, y) != boundary_color &&
        getpixel(x, y) != fill_color)
    {
        putpixel(x, y, fill_color);
        boundaryFill8(x + 1, y, fill_color, boundary_color);
        boundaryFill8(x, y + 1, fill_color, boundary_color);
        boundaryFill8(x - 1, y, fill_color, boundary_color);
        boundaryFill8(x, y - 1, fill_color, boundary_color);
        boundaryFill8(x - 1, y - 1, fill_color, boundary_color);
        boundaryFill8(x - 1, y + 1, fill_color, boundary_color);
        boundaryFill8(x + 1, y - 1, fill_color, boundary_color);
        boundaryFill8(x + 1, y + 1, fill_color, boundary_color);
    }
}

int main()
{
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");
```

```
rectangle(50, 50, 100, 100);  
boundaryFill8(55, 55, 4, 15);  
delay(10000);  
getch();  
closegraph();  
return 0;  
}
```

Output:

