

---

**UNIVERSITY OF PETROLEUM & ENERGY STUDIES**

**School of Computer Science**

**COMPUTER GRAPHICS**  
**LAB FILE**

**Submitted By:**

Dhruv Singhal

500075346

R177219074

AIML B3

**Submitted To:**

Dr. Niharika Singh

Asst. Professor

SOCS

UPES

# Lab Exercise - 1

## Introduction to OpenGL

### Q 1. What is OpenGL?

OpenGL, short for "Open Graphics Library," is an application programming interface (API) designed for rendering 2D and 3D graphics. It provides a common set of commands that can be used to manage graphics in different applications and on multiple platforms.

By using OpenGL, a developer can use the same code to render graphics on a Mac, PC, or mobile device. Nearly all modern operating systems and hardware devices support OpenGL, making it an easy choice for graphics development.

### Q 2. What is GLU/GLUT?

GLU is the OpenGL Utility Library. This is a set of functions to create texture mipmaps from a base image, map coordinates between screen and object space, and draw quadric surfaces.

GLUT is the OpenGL Utility Toolkit, a window system independent toolkit for writing OpenGL programs. It implements a simple windowing API for OpenGL.

### Q 3. What is OpenGL Architecture?

The architecture of OpenGL is based on a client-server model. An application program written to use the OpenGL API is the "client" and runs on the CPU. The implementation of the OpenGL graphics engine is the "server" and runs on the GPU.

### Q 4. Setting up the environment

I referred to the below Youtube link to setup OpenGL in CodeBlocks.

<https://www.youtube.com/watch?v=14atQ1GTNYg>

First we have to download & install the CodeBlocks

(Includes a drive link in description of youtube video including codeblocks and required files)

Then, we have to Setup the required GLUT header, library & dll files & paste them at their respective locations where CodeBlocks is installed.

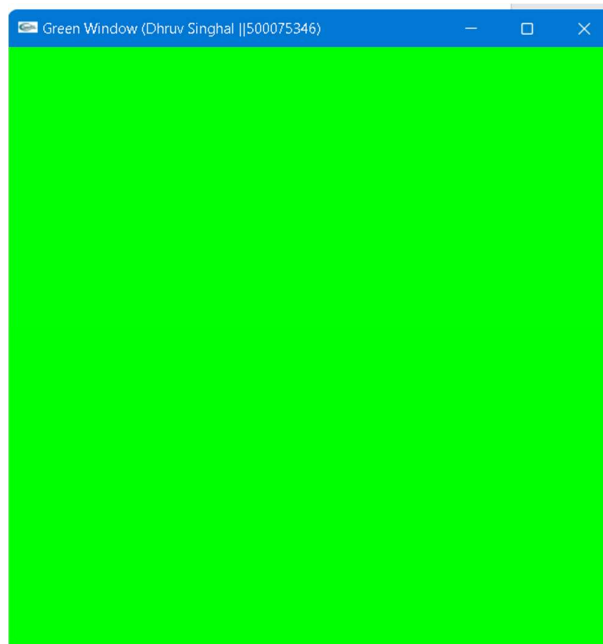
Then we can include the library in the project & use it.

### Q 5. Initialize a window of Green Colour Code

```
#include<windows.h>
#include<GL/glu.h>
#include<GL/glut.h>
#include<iostream>
using namespace std;
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}
void init()
```

```
{  
  
    glClearColor(0.0,1,0.0,1.0);  
    glColor3f(0.0,1.0,0.0);  
}  
  
int main(int argc , char** argv)  
{  
    glutInit(&argc,argv);  
    glutInitDisplayMode(GLUT_RGB);  
    glutInitWindowPosition(200,100);  
    glutInitWindowSize(500,500);  
    glutCreateWindow("Green Window (Dhruv Singhal ||500075346)");  
    glutDisplayFunc(display);  
    init();  
    glutMainLoop();  
    return 0;  
}
```

**OUTPUT:**



## Experiment 2:

# Drawing line using DDA, Bresenham's algorithm

**Experiment 2: Drawing a line** [Usage of Open GL on Linux Environment for Virtual Environment]

- Draw a line using equation of line  $Y=m*X+C$ .

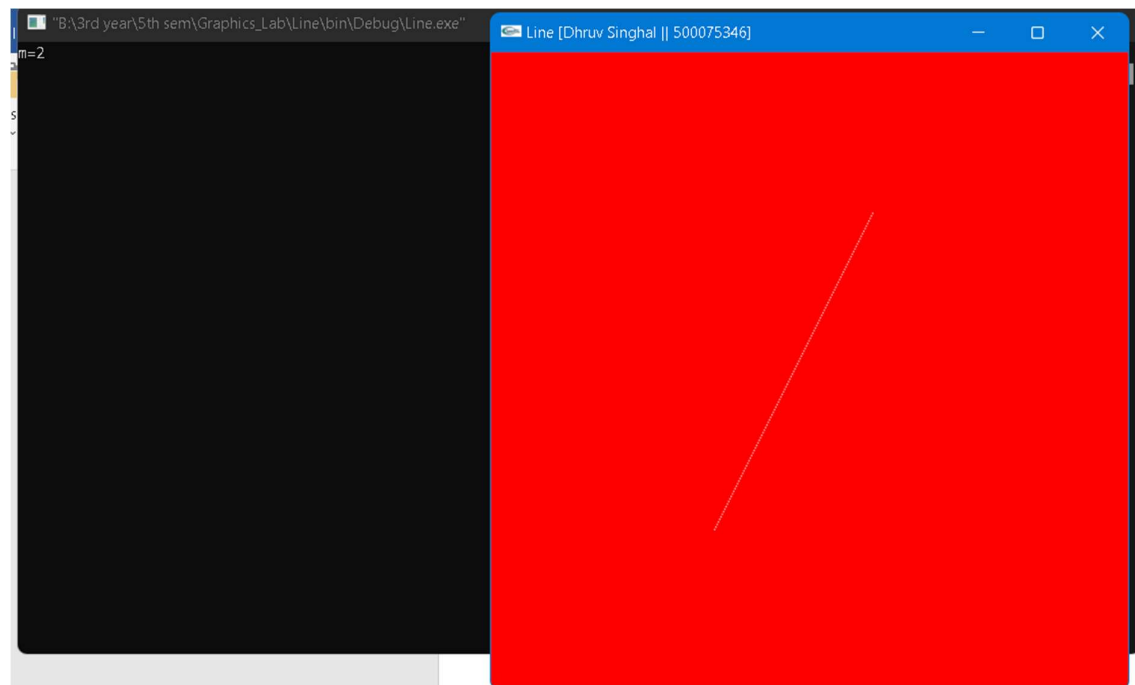
### CODE:

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

void display()
{
    glClearColor(1.0,0.0,0.0,0.0);
    glColor3f(1,1,1);
    glClear(GL_COLOR_BUFFER_BIT);
    float x1=-0.3,y1=-0.5;
    float x2=0.2,y2=0.5;
    float m,c,y;
    m=(y2-y1)/(x2-x1);
    c=y1-m*x1;
    glBegin(GL_LINES);
    for( float x=x1;x<=x2;x=x+0.002){
        y=m*x+c;
        glVertex2f(x,y);
    }
    glEnd();
    cout<<"m="<<m;
```

```
    glFlush();  
}  
  
int main(int argc , char** argv)  
{  
    glutInit(&argc,argv);  
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);  
    glutInitWindowPosition(200,100);  
    glutInitWindowSize(500,500);  
    glutCreateWindow("Line [Dhruv Singhal || 500075346]");  
    glutDisplayFunc(display);  
    glutMainLoop();  
    return 0;  
}
```

### Output:



## Experiment 2: Drawing a line [Usage of Open GL on Linux Environment for Virtual Environment]

Draw a line using DDA algorithm for slope  $m < 1$  and  $m > 1$

### CODE:

```
#include<windows.h>

#include<GL/glu.h>

#include<GL/glut.h>

#include<stdio.h>

float x1,x2,y1,y2;

void display(void)

{

    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1.0, 0.0, 0.0); //Quadrant Plot Graph

    glBegin(GL_LINES);

    glVertex2i(-50, 0);

    glVertex2i(50, 0);

    glVertex2i(0, -50);

    glVertex2i(0, 50);

    glEnd();

    float dy,dx,step,x,y,k,m;

    dx=x2-x1;

    dy=y2-y1;

    m=dy/dx;

    if(abs(dx)> abs(dy))

    {

        step = abs(dx);

    }

    else

    {

        step = abs(dy);

    }

    x=x1;
```

```

y=y1;

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

for (k=1 ; k<=step; k++)

{ // 0.5 factor is added to remove the stair case effect

    if(m<1)

    {

        x= 1*0.5 + x;

        y= m*0.5 + y;

    }

    if(m==1)

    {

        x= 1*0.5 + x;

        y= 1*0.5 + y;

    }

    if(m>1)

    {

        x= (1/m)*0.5 + x;

        y= 1*0.5 + y;

    }

    glBegin(GL_POINTS);

    glColor3f(0.0,0.0,0.0);

    glVertex2f(x,y);

    glEnd();

}

glFlush();
}

void init(void)

{

    glClearColor(1.0,1.0,0.0,0.0);

    glMatrixMode(GL_PROJECTION);

    gluOrtho2D(-50,50,-50,50);

}

int main(int argc, char** argv)

{

```

```

printf("Enter the value of x1 : ");
scanf("%f",&x1);

printf("Enter the value of y1 : ");
scanf("%f",&y1);

printf("Enter the value of x2 : ");
scanf("%f",&x2);

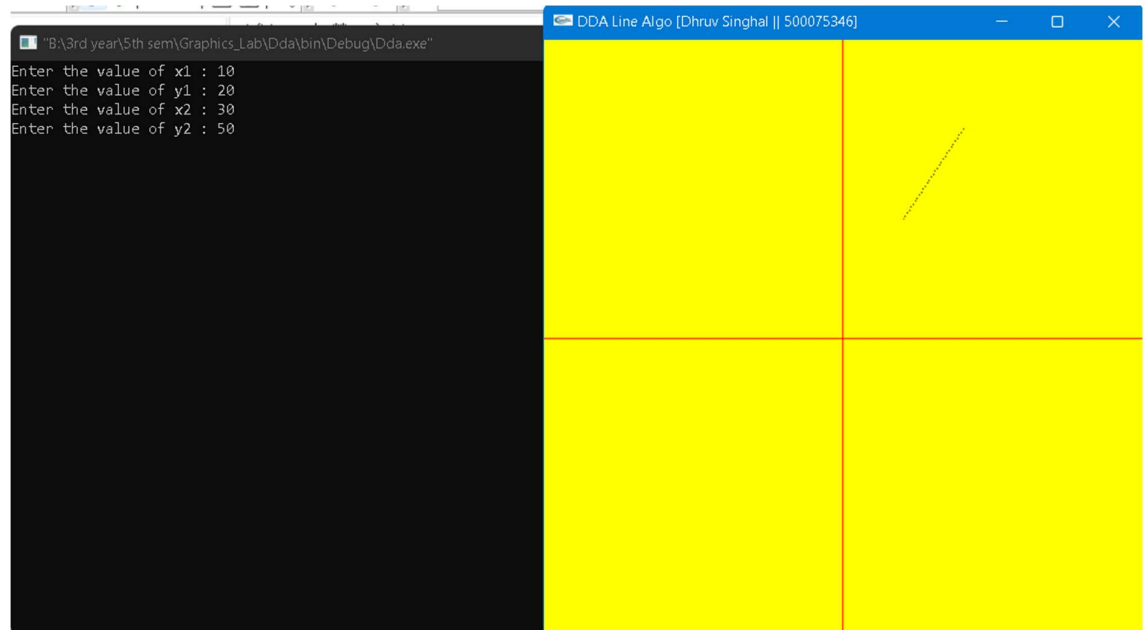
printf("Enter the value of y2 : ");
scanf("%f",&y2);

glutInit(&argc, argv);
glutInitDisplayMode ( GLUT_RGB);
glutInitWindowSize (500, 500);
glutInitWindowPosition (100,100);
glutCreateWindow ("DDA Line Algo [Dhruv Singhal || 500075346]");
init();
glutDisplayFunc(display);
glutMainLoop();

return 0;
}

```

### Output:





## Experiment 2: Drawing a line [Usage of Open GL on Linux Environment for Virtual Environment]

- Draw a line using Bresenham algorithm for slope  $m < 1$  and  $m > 1$ .

### CODE:

```
#include<windows.h>

#include<GL/glut.h>

#include<stdio.h>

int xa,xb,ya,yb;

void display (void)

{

int dx=xb-xa;

int dy=yb-ya;

int x=xa,y=ya;

glClear (GL_COLOR_BUFFER_BIT);

glColor3f (0.0, 1.0, 0.0);

glBegin(GL_POINTS);

glVertex2i(x,y);

int m;

m=dy/dx;

if(m<1){

int p0 = 2*dy - dx;

int p =p0;

int k;

for(k=0;k<dx;k++)

{

if(p<0)

{

x = x+1;

glVertex2i(x,y);

p=p+2*dy;
```

```
printf("%d %d\n",x,y);

}

else

{

x = x+1; y = y+1;

glVertex2i(x,y);

p=p+2*dy-2*dx;

printf("%d %d\n",x,y);

}

}

}

if(m>=1){

int P0 = 2*dx - dy;

int P =P0;

int K;

for(K=0;K<dy;K++)

{

if(P<0)

{

y = y+1;

glVertex2i(x,y);

P=P+(2*dx);

printf("%d %d\n",x,y);

}

else

{

x = x+1; y = y+1;

glVertex2i(x,y);

P=P+2*dx-2*dy;

printf("%d %d\n",x,y);

}

}

}

glEnd();

glFlush();
```

```
}

void init(void)

{

glClearColor (0.0, 0.0, 0.0, 0.0);

glMatrixMode(GL_PROJECTION);

glLoadIdentity();

gluOrtho2D(-100,100,-100,100);

}

int main(int argc, char** argv)

{

printf("Enter the value of x1 : ");

scanf("%d",&xa);

printf("Enter the value of y1 : ");

scanf("%d",&ya);

printf("Enter the value of x2 : ");

scanf("%d",&xb);

printf("Enter the value of y2 : ");

scanf("%d",&yb);


glutInit(&argc, argv);

glutInitDisplayMode (GLUT_RGB);

glutInitWindowSize (500, 500);

glutInitWindowPosition (100, 100);

glutCreateWindow ("Bresenham Line Algorithm [ Dhruv Singhal | | 500075346 ");

init();

glutDisplayFunc(display);

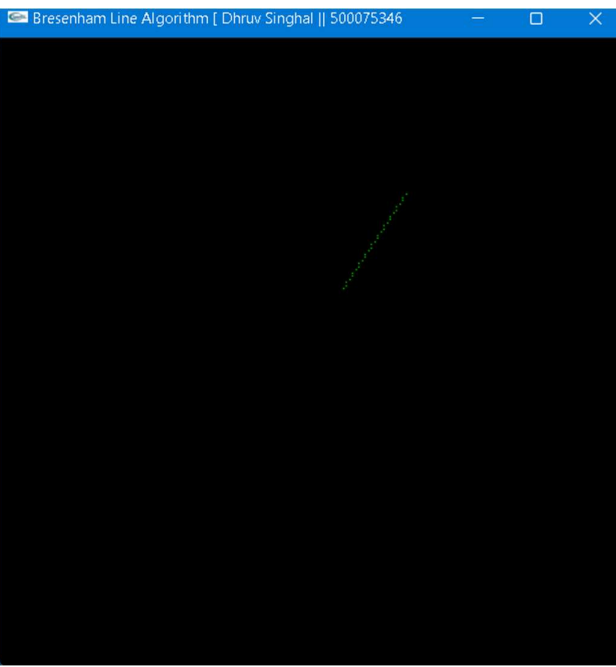
glutMainLoop();

return 0;

}
```

## Output:

```
0 error(s), 0 warning(s), 0 minute(s), 1 second(s)
B:\3rd year\5th sem\Graphics_Lab\Bresenham\bin\Debug\Bresenham
Enter the value of x1 : 10
Enter the value of y1 : 20
Enter the value of x2 : 30
Enter the value of y2 : 50
11 21
11 22
12 23
13 24
13 25
14 26
15 27
15 28
16 29
17 30
17 31
18 32
19 33
19 34
20 35
21 36
21 37
22 38
23 39
23 40
24 41
25 42
25 43
26 44
27 45
27 46
```



# Lab Exercise - 3

## Drawing Circle and Ellipse using Mid-point algorithm

**Experiment 3: Drawing a Circle and an Ellipse** [Done on OpenGL Environment]

- Draw the circle with the help of polar equations

# Take the value of radius, major axis and minor axis as input from the user.

**CODE:**

```
#include<windows.h>
#include<GL\glew.h>
#include<GL\glut.h>
#include <stdio.h>
#include <stdlib.h>
#include<math.h>

int xc, yc, r;

void putpixel(int x, int y)
{
    glPointSize(5.0);
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_POINTS);
    glVertex2i(xc + x, yc + y);
    glVertex2i(xc + x, yc - y);
    glVertex2i(xc + y, yc + x);
    glVertex2i(xc + y, yc - x);
    glVertex2i(xc - x, yc - y);
    glVertex2i(xc - y, yc - x);
    glVertex2i(xc - x, yc + y);
    glVertex2i(xc - y, yc + x);
    glEnd();
}

void display()
```

```

{
float x, y;

x = 0, y = r;

float theta = 0;

float inc = (float)1 / r;

glColor3f(1.0, 0.0, 0.0); //Quadrant Plot Graph

glBegin(GL_LINES);

glVertex2i(-50, 0);

glVertex2i(50, 0);

glVertex2i(0, -50);

glVertex2i(0, 50);

glEnd();

float end = 3.14 / 4;

float C = cos(inc);

float S = sin(inc);

while (theta <= end)

{

float xtemp = x;

x = x * C - y * S;

y = y * C + xtemp * S;

putpixel(x, y);

theta = theta + inc;

}

glFlush();

}

void init()

{

glClearColor(0.7, 0.7, 0.7, 0.7);

glMatrixMode(GL_PROJECTION);

glLoadIdentity();

gluOrtho2D(-50, 50, -50, 50);

}

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

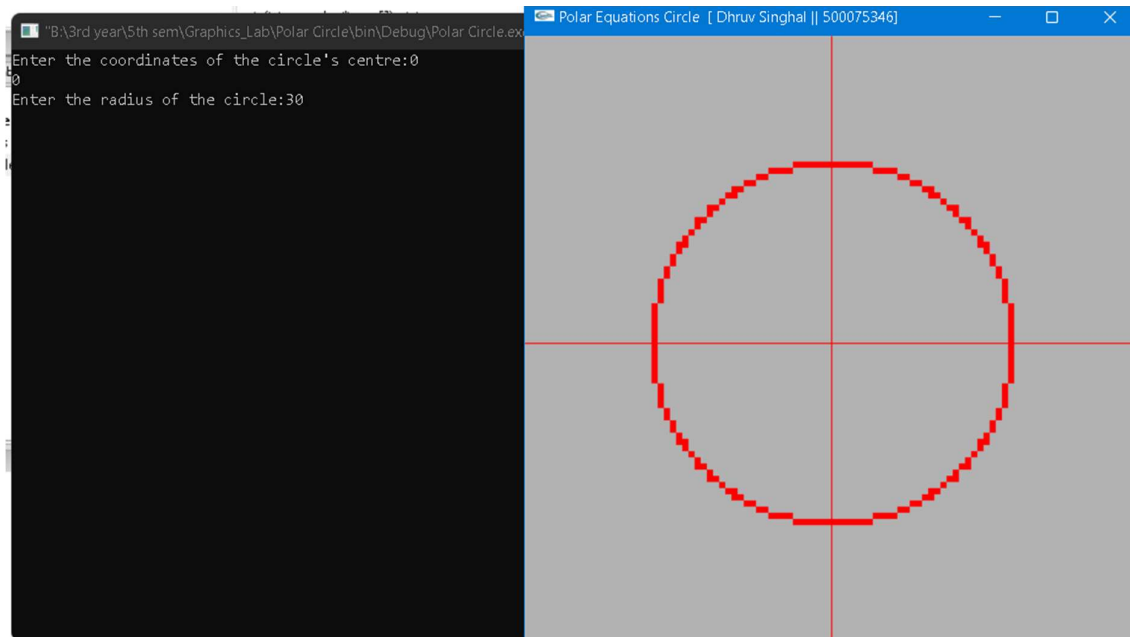
printf("Enter the coordinates of the circle's centre:");

scanf("%d %d",&xc,&yc);

```

```
printf("Enter the radius of the circle:");  
scanf("%d",&r);  
  
glutInit(&argc, argv);  
glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);  
glutInitWindowSize(500, 500);  
glutInitWindowPosition(200, 100);  
glutCreateWindow("Polar Equations Circle [ Dhruv Singhal || 500075346]");  
  
init();  
  
glutDisplayFunc(display);  
  
glutMainLoop();  
}
```

## OUTPUT:



### Experiment 3: Drawing a Circle and an Ellipse [Done on OpenGL Environment]

- Draw the circle with the help of mid-point method.

# Take the value of radius, major axis and minor axis as input from the user.

**CODE:**

```
#include<windows.h>

#include<GL\glew.h>

#include<GL\glut.h>

#include <stdio.h>

int x,y,r,xc,yc;

void putpixel(int x, int y)

{

    glPointSize(8.0);

    glColor3f(1.0, 0.0, 0.0);

    glBegin(GL_POINTS);

    glVertex2i(xc + x, yc + y);

    glVertex2i(xc + x, yc - y);

    glVertex2i(xc + y, yc + x);

    glVertex2i(xc + y, yc - x);

    glVertex2i(xc - x, yc - y);

    glVertex2i(xc - y, yc - x);

    glVertex2i(xc - x, yc + y);

    glVertex2i(xc - y, yc + x);

    glEnd();

}

void display()

{

    glColor3f(1.0, 0.0, 0.0); //Quadrant Plot Graph

    glBegin(GL_LINES);

    glVertex2i(-50, 0);

    glVertex2i(50, 0);

    glVertex2i(0, -50);

    glVertex2i(0, 50);

    glEnd();

    int d[r];

    d[0]=1-r;
```



```

x=0,y=0;

y=r;

if(d[0]<=0)
{
    putpixel(x,y);

    d[1]=d[0]+2*x+1;

    x=x+1;
}
else
{
    putpixel(x,y);

    d[1]=d[0]+2*x+3-2*y;

    x=x+1;

    y=y-1;
}

int i=1;
for(; i<y; i++)
{
    if(d[i]<=0)
    {
        putpixel(x,y);

        d[i+1]=d[i]+2*x+1;

        x=x+1;
    }
    else
    {
        putpixel(x,y);
        d[i+1]=d[i]+2*x+3-2*y;

        x=x+1;

        y=y-1;
    }
}

glEnd();

glFlush();
}

```

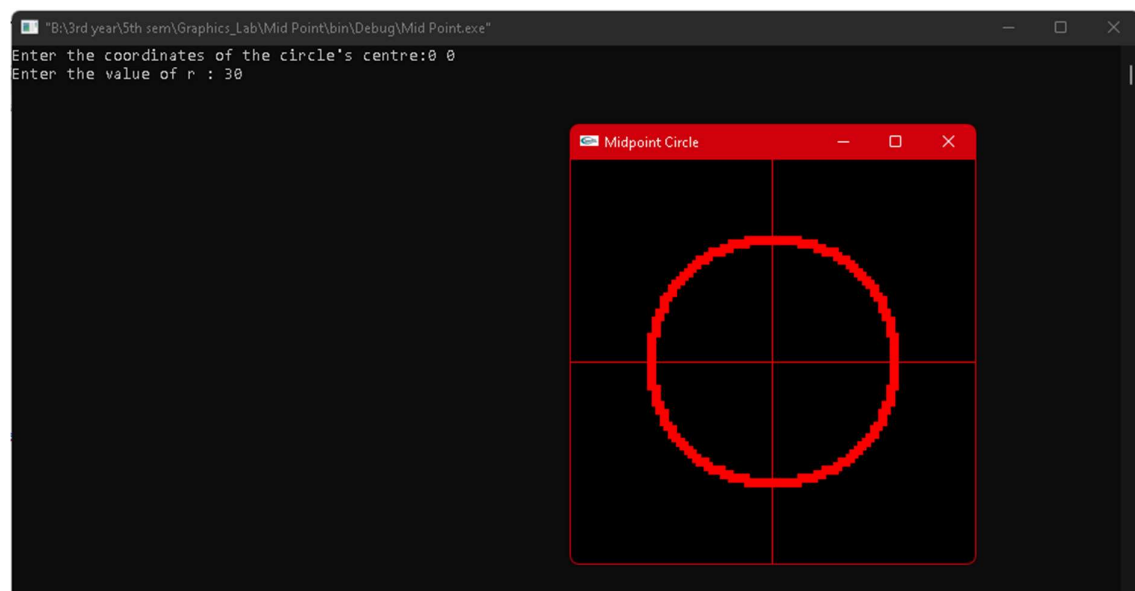
```

void init()
{
    glClearColor(0.7, 0.7, 0.7, 0.7);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-50, 50, -50, 50);
}

int main(int argc, char* argv[])
{
    printf("Enter the coordinates of the circle's centre:");
    scanf("%d %d",&xc,&yc);
    printf("Enter the value of r : ");
    scanf("%d",&r);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutInitWindowSize(350, 350);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Midpoint Circle [Dhruv Singhal | | 500075346]");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
}

```

## OUPUT:



### Experiment 3: Drawing a Circle and an Ellipse [Done on OpenGL Environment]

- Draw the Ellipse with the mid-point method.

# Take the value of radius, major axis and minor axis as input from the user.

**CODE:**

```
#include<windows.h>

#include<GL\glew.h>

#include<GL\glut.h>

#include <stdio.h>

int x,y,rx,ry,xc,yc;

void display()

{

    glColor3f(1.0, 0.0, 0.0); //Quadrant Plot Graph

    glBegin(GL_LINES);

    glVertex2i(-50, 0);

    glVertex2i(50, 0);

    glVertex2i(0, -50);

    glVertex2i(0, 50);

    glEnd();

    glBegin(GL_POINTS);

    glColor3f(0.0,1.0,1.0);

    x=0;

    y=ry;

    float p=ry*ry-rx*rx*(ry-(rx*rx)/4);

    while(2*ry*ry*x <=2*rx*rx*y)

    {

        if(p < 0)

        {

            x++;

            p = p+2*ry*ry*x+ry*ry;

        }

        else

        {

            x++;

            y--;

            p = p+2*ry*ry*x-2*rx*rx*y-ry*ry;

        }

    }

}
```

```

    }

    glVertex2i(xc+x,yc+y);

    glVertex2i(xc+x,yc-y);

    glVertex2i(xc-x,yc+y);

    glVertex2i(xc-x,yc-y);

}

float p2=ry*ry*(x+0.5)*(x+0.5)+rx*rx*(y-1)*(y-1)-rx*rx*ry*ry;

while(y > 0)

{

    if(p2 <= 0)

    {

        x++;

        y--;

        p2 = p2+2*ry*ry*x-2*rx*rx*y+rx*rx;

    }

    else

    {

        y--;

        p2 = p2-2*rx*rx*y+rx*rx;

    }

    glVertex2i(xc+x,yc+y);

    glVertex2i(xc+x,yc-y);

    glVertex2i(xc-x,yc+y);

    glVertex2i(xc-x,yc-y);

}

glEnd();

glFlush();

}

void init()

{

    glClearColor(0.7, 0.7, 0.7, 0.7);

    glMatrixMode(GL_PROJECTION);

    glLoadIdentity();

    gluOrtho2D(-50, 50, -50, 50);

}

```

```

int main(int argc, char* argv[])
{
    printf("Enter the coordinates of the ellipse centre:");

    scanf("%d %d",&xc,&yc);

    printf("Enter the value of rx : ");

    scanf("%d",&rx);

    printf("Enter the value of ry : ");

    scanf("%d",&ry);

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);

    glutInitWindowSize(500, 500);

    glutInitWindowPosition(100, 100);

    glutCreateWindow("Midpoint Ellipse [Dhruv Singhal | 500075343]");

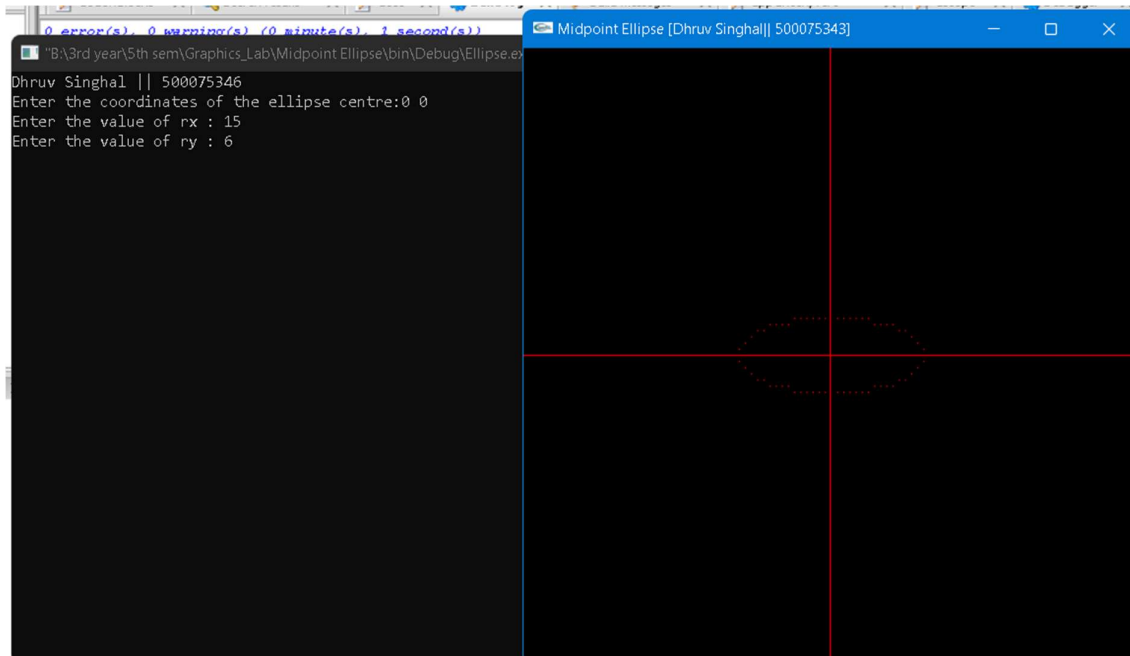
    init();

    glutDisplayFunc(display);

    glutMainLoop();
}

```

## OUPUT:



# Lab Exercise- 4

## Filling – Area

**Experiment 4: Filling –Area** [Small Project will be given for demonstration]

- WAP to fill a region using boundary fill algorithm using 4 or 8 connected approaches.

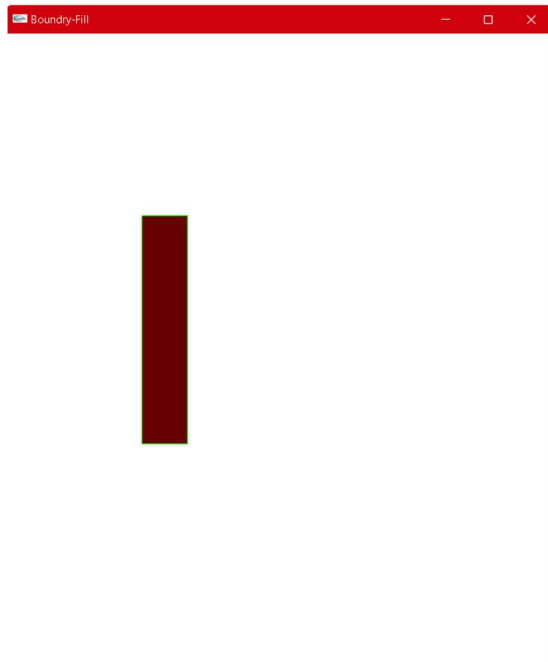
```
#include<windows.h>
#include <GL/glut.h>
int ww = 600, wh = 700; // window width and height
float fillCol[3] = {0.4,0.0,0.0};
float borderCol[3] = {0.0,0.0,0.0};
void setPixel(int pointx, int pointy, float f[3])
{
    glBegin(GL_POINTS);
    glColor3fv(f);
    glVertex2i(pointx,pointy);
    glEnd();
    glFlush();
}
void getPixel(int x, int y, float pixels[3])
{
    glReadPixels(x,y,1.0,1.0,GL_RGB,GL_FLOAT,pixels);
}
void drawPoly(int x1, int y1, int x2, int y2)
{
    glColor3f(0.0,1.0,0.0);
    glBegin(GL_LINES);
    glVertex2i(x1, y1);
    glVertex2i(x1, y2);
    glVertex2i(x2, y1);
    glVertex2i(x2, y2);
    glVertex2i(x1, y1);
    glVertex2i(x2, y1);
    glVertex2i(x1, y2);
    glVertex2i(x2, y2);
    glEnd();
    glFlush();
}
void display()
{
    glClearColor(1.0,1.0,1.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    drawPoly(150,250,200,500);
    glFlush();
}
void boundaryFill4(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,fillColor);
            boundaryFill4(x+1,y,fillColor,borderColor);
            boundaryFill4(x-1,y,fillColor,borderColor);
            boundaryFill4(x,y+1,fillColor,borderColor);
            boundaryFill4(x,y-1,fillColor,borderColor);
        }
    }
}
void mouse(int btn, int state, int x, int y)
{
    if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        int xi = x;
        int yi = (wh-y);
        boundaryFill4(xi,yi,fillCol,borderCol);
    }
}
void init()
{
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0,(GLdouble)ww,0.0,(GLdouble)wh);
    glMatrixMode(GL_PROJECTION);
}
int main(int argc, char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(ww,wh);
    glutCreateWindow("Boundry-Fil [Dhruv Singhal | | 500075346 ] ");
    glutDisplayFunc(display);
    init();
    glutMouseFunc(mouse);
    glutMainLoop();
    return 0;
}

```

**OUTPUT:**



#### Experiment 4: Filling –Area [Small Project will be given for demonstration]

- WAP to fill a region using flood fill algorithm using 4 or 8 connected approaches.

```
#include<windows.h>
#include <gl/glut.h>
#include<math.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(0.0, 0.0, 0.0);
    glPointSize(1.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0, 640, 0, 480);
}

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);
    }
    return;
}

void onMouseClick(int button, int state, int x, int y)
{
    Color newColor = {1.0f, 0.0f, 0.0f};
    Color oldColor = {1.0f, 1.0f, 1.0f};

    floodFill(320, 240, oldColor, newColor);
}

void draw_circle(Point pC, GLfloat radius)
{
    GLfloat step = 1/radius;
    GLfloat x, y;

    for(GLfloat theta = 0; theta <= 360; theta += step)
    {
        x = pC.x + (radius * cos(theta));
        y = pC.y + (radius * sin(theta));
        glVertex2i(x, y);
    }
}

void display(void)
{
    Point pt = {320, 240};
    GLfloat radius = 60;

    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POINTS);
    draw_circle(pt, radius);
    glEnd();
    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(200, 200);
    glutCreateWindow("Circle with flood fill");
    init();
}

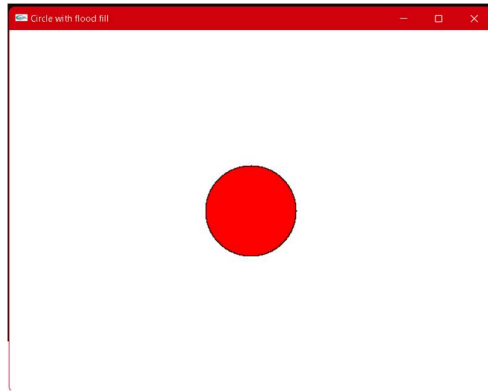
```

```

glutDisplayFunc(display);
glutMouseFunc(onMouseClicked);
glutMainLoop();
return 0;
}

```

## OUTPUT:



# Lab Exercise- 5&6

## Viewing & Clipping

### Experiment 5 & 6: Viewing and Clipping [Geographical Animation for demonstration]

- Write an interactive program for line clipping using Cohen Sutherland line clipping algorithm.
- Write an interactive program for line clipping using Liang-Barsky line clipping algorithm.
- Write an interactive program for polygon clipping using Sutherland – Hodgeman polygon clipping algorithm.

```

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

const int MAX_POINTS = 20;
GLint count = 0;

void init(void)
{
    glClearColor(1.0,1.0,1.0,0.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-1000,1000,-1000,1000);
}

void plotline(float a,float b,float c,float d)
{

```

```

glBegin(GL_LINES);
glVertex2i(a,b);
glVertex2i(c,d);
glEnd();
}

// Returns x-value of point of intersection of two lines
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;
}

// Returns y-value of point of intersection of two lines
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;
}

// This functions clips all the edges w.r.t one clip edge of clipping area
void clip(int poly_points[][2], int &poly_size, int x1, int y1, int x2, int y2)
{
    int new_points[MAX_POINTS][2], new_poly_size = 0;

    // (ix,iy),(kx,ky) are the co-ordinate values of the points
    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][0], iy = poly_points[i][1];
        int kx = poly_points[k][0], ky = poly_points[k][1];

        // 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
    }
}

// Copying new points into original array and changing the no. of vertices
poly_size = new_poly_size;
for (int i = 0; i < poly_size; i++)
{
    poly_points[i][0] = new_points[i][0];
    poly_points[i][1] = new_points[i][1];
}
}

// Implements Sutherland♦Hodgman algorithm
void suthHodgClip(int poly_points[][2], int poly_size, int clipper_points[][2], int clipper_size)
{
    //i and k are two consecutive indexes
    for (int i=0; i<clipper_size; i++)
    {
        int k = (i+1) % clipper_size;

        // We pass the current array of vertices, it's size
        // and the end points of the selected clipper line
        clip(poly_points, poly_size, clipper_points[i][0],
            clipper_points[i][1], clipper_points[k][0],
            clipper_points[k][1]);
    }

    // Printing vertices of clipped polygon
    for (int i=0; i < poly_size; i++)
    {
        glColor3f(0.0,0.0,0.0);
        if(i!=(poly_size-1))
        {
            glBegin(GL_LINES);
            glVertex2i(poly_points[i][0],poly_points[i][1]);
            glVertex2i(poly_points[i+1][0],poly_points[i+1][1]);
            glEnd();
        }
        else
        {
            glBegin(GL_LINES);
            glVertex2i(poly_points[i][0],poly_points[i][1]);
            glVertex2i(poly_points[0][0],poly_points[0][1]);
            glEnd();
        }
    }
}

```

```

    }
}

void mousePtPlot(GLint button, GLint action, GLint xMouse, GLint yMouse)
{
    if(button == GLUT_LEFT_BUTTON && action == GLUT_UP)
    {
        if(!count)
        {
            int poly_size = 8;
            int poly_points[20][2] = {{-450,0},{-450,800},{0,800},{0,500},{-350,700},{-350,200},{-200,200},{-200,0}};

            // Defining clipper polygon vertices in clockwise order
            // 1st Example with square clipper
            int clipper_size = 4;
            int clipper_points[][2] = {{-300,100},{-300,600},{200,600},{200,100}};

            //Calling the clipping function
            suthHodgClip(poly_points, poly_size, clipper_points, clipper_size);
            count++;
            cout<<"Polygon clipped\n";
            glFlush();
        }
    }
    if(button == GLUT_RIGHT_BUTTON && action == GLUT_UP)
    {
        exit(0);
    }
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0,1.0,0.0);

    glBegin(GL_LINE_LOOP);
        glVertex2i(-300,100);
        glVertex2i(200,100);
        glVertex2i(200,600);
        glVertex2i(-300,600);
    glEnd();

    glColor3f(1.0,0.0,0.0);
    glBegin(GL_LINE_LOOP);
        glVertex2i(-450,0);
        glVertex2i(-200,0);
        glVertex2i(-200,200);
        glVertex2i(-350,200);
        glVertex2i(-350,700);
        glVertex2i(0,500);
        glVertex2i(0,800);
        glVertex2i(-450,800);
    glEnd();

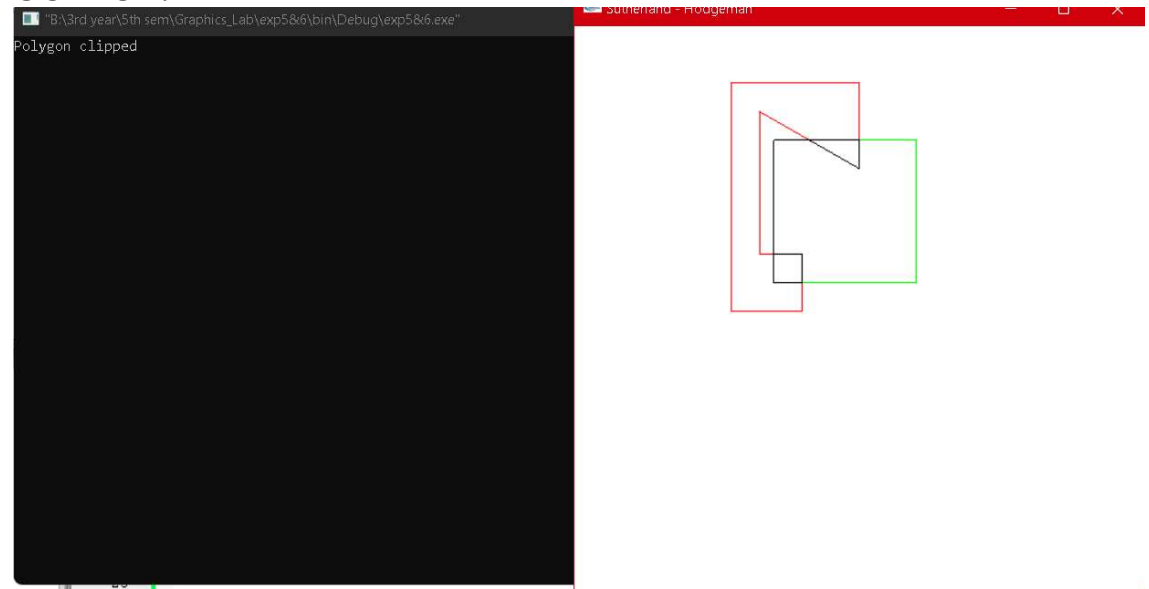
    glFlush();
}

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

```

```
glutInitWindowSize(500,500);  
glutInitWindowPosition(0,0);  
glutCreateWindow("Sutherland - Hodgeman");  
glutMouseFunc(mousePtPlot);  
glutDisplayFunc(display);  
  
init();  
glutMainLoop();  
return 0;  
}
```

## OUTPUT:



# Lab Exercise- 7&8

## Basic Two- & Three-Dimensional Transformations

### Experiment 7 & 8: Basic Two3 & Three Dimensional Transformations

- Write an interactive program for following basic transformation.
- Translation
- Rotation
- Scaling
- Reflection about axis.
- Reflection about a line  $Y=mX+c$  and  $aX+bY+c=0$ .
- Shear about an edge and about a vertex.

### Exp7: 2D Transformation

```
#include<windows.h>
#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);
    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 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);
    gluOrtho2D(-200.0, 200.0, -200.0, 200.0);
}

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

    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 shearing 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;
        }
    }
}

```

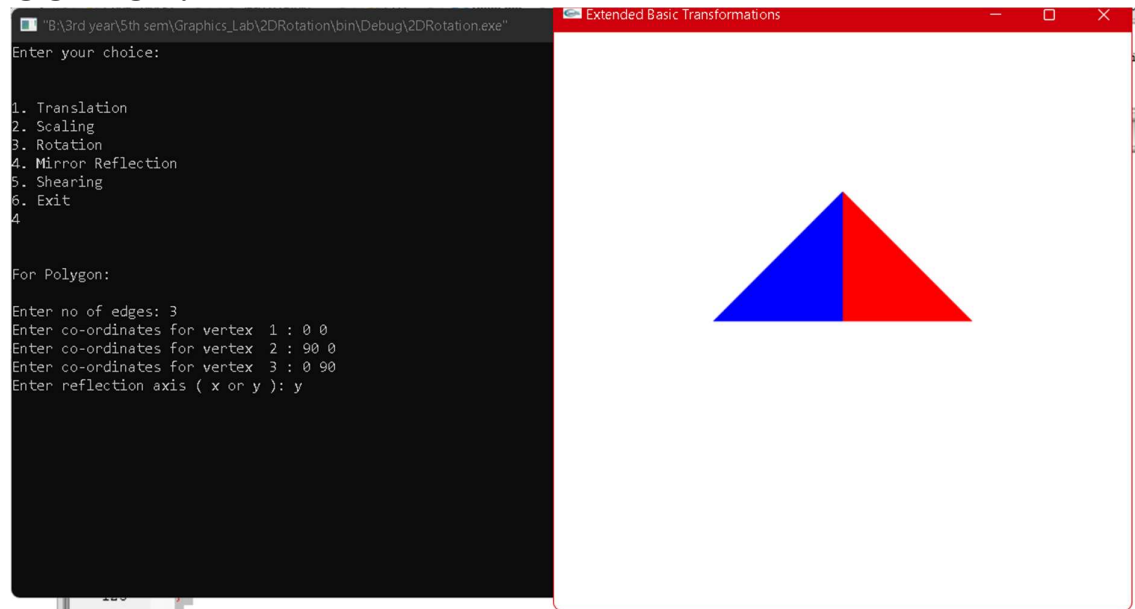
```

    }
    //cout << "\n\nPoints:" << pntX[0] << ", " << pntY[0] << endl;
    //cout << angleRad;

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("Extended Basic Transformations");
    glutDisplayFunc(myDisplay);
    myInit();
    glutMainLoop();
}

```

## OUTPUT:



## Exp 8: 3D transformation

```

#include<iostream>
#include<windows.h>
#include<math.h>
#include<GL/glut.h>
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];
float tx,ty,tz;
float sx,sy,sz;

```

```

float angle;

int choice,choiceRot;

void setIdentityM(Matrix4 m)
{
    for(int i=0;i<4;i++)
        for(int j=0;j<4;j++)
            m[i][j]=(i==j);
}

void translate(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 scale(int sx,int sy,int sz)
{
    theMatrix[0][0]=sx;
    theMatrix[1][1]=sy;
    theMatrix[2][2]=sz;
}

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 RotateY(float angle) //parallel to y
{
    angle = angle*3.14/180;
    theMatrix[0][0] = cos(angle);
    theMatrix[0][2] = -sin(angle);
    theMatrix[2][0] = sin(angle);
    theMatrix[2][2] = cos(angle);
}

void RotateZ(float angle) //parallel to z
{
    angle = angle*3.14/180;
    theMatrix[0][0] = cos(angle);
    theMatrix[0][1] = sin(angle);
    theMatrix[1][0] = -sin(angle);
    theMatrix[1][1] = cos(angle);
}

void multiplyM()
{
    //We Don't require 4th row and column in scaling and rotation
    //[8][3]=[8][3]*[3][3] //4th not used
    for(int i=0;i<8;i++)

```

```

{
    for(int j=0;j<3;j++)
    {
        output[i][j]=0;
        for(int k=0;k<3;k++)
        {
            output[i][j]=output[i][j]+input[i][k]*theMatrix[k][j];
        }
    }
}

}

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 draw(float a[8][3])
{

    glBegin(GL_QUADS);
    glColor3f(0.7,0.4,0.5); //behind
    glVertex3fv(a[0]);
    glVertex3fv(a[1]);
    glVertex3fv(a[2]);
    glVertex3fv(a[3]);

    glColor3f(0.8,0.2,0.4); //bottom
    glVertex3fv(a[0]);
    glVertex3fv(a[1]);
    glVertex3fv(a[5]);
    glVertex3fv(a[4]);

    glColor3f(0.3,0.6,0.7); //left
    glVertex3fv(a[0]);
    glVertex3fv(a[4]);
    glVertex3fv(a[7]);
    glVertex3fv(a[3]);

    glColor3f(0.2,0.8,0.2); //right
    glVertex3fv(a[1]);
    glVertex3fv(a[2]);
    glVertex3fv(a[6]);
    glVertex3fv(a[5]);

    glColor3f(0.7,0.7,0.2); //up
    glVertex3fv(a[2]);
    glVertex3fv(a[3]);
    glVertex3fv(a[7]);
    glVertex3fv(a[6]);

    glColor3f(1.0,0.1,0.1);
    glVertex3fv(a[4]);
    glVertex3fv(a[5]);
    glVertex3fv(a[6]);
}

```

```

glVertex3fv(a[7]);

glEnd();
}

void init()
{
    glClearColor(1.0,1.0,1.0,1.0); //set backgrnd color to white
    glOrtho(-250.0,250.0,-250.0,250.0,-250.0,250.0);
    // Set the no. of Co-ordinates along X & Y axes and their gappings
    glEnable(GL_DEPTH_TEST);
    // To Render the surfaces Properly according to their depths
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    Axes();
    glColor3f(1.0,0.0,0.0);
    draw(input);
    setIdentityM(theMatrix);
    switch(choice)
    {
    case 1:
        translate(tx,ty,tz);
        break;
    case 2:
        scale(sx,sy,sz);
        multiplyM();
        break;
    case 3:
        switch (choiceRot) {
        case 1:
            RotateX(angle);
            break;
        case 2: RotateY(angle);
            break;
        case 3:
            RotateZ(angle);
            break;
        default:
            break;
        }
        multiplyM();
        break;
    }

    draw(output);
    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("3D TRANSFORMATIONS");
    init();
    cout<<"Enter your choice number:\n1.Translation\n2.Scaling\n3.Rotation\nn=>";
    cin>>choice;
    switch (choice) {
    case 1:

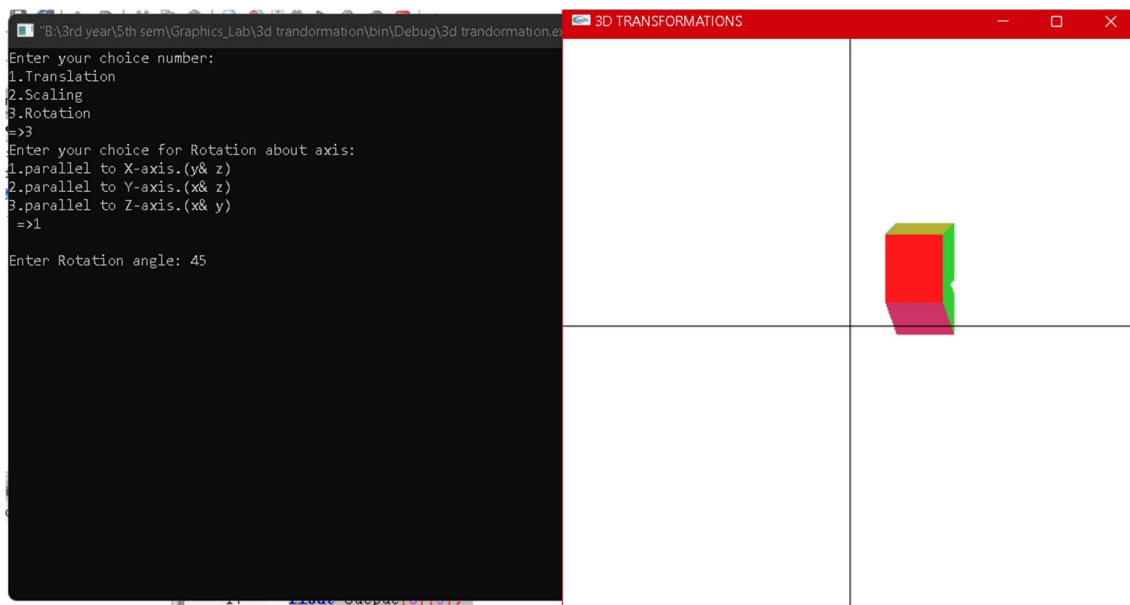
```

```

        cout<<"\nEnter Tx,Ty &Tz: \n";
        cin>>tx>>ty>>tz;
        break;
    case 2:
        cout<<"\nEnter Sx,Sy & Sz: \n";
        cin>>sx>>sy>>sz;
        break;
    case 3:
        cout<<"Enter your choice for Rotation about axis:\n1.parallel to X-axis."
            <<"(y& z)\n2.parallel to Y-axis.(x& z)\n3.parallel to Z-axis."
            <<"(x& y)\n=>";
        cin>>choiceRot;
        switch (choiceRot) {
            case 1:
                cout<<"\nEnter Rotation angle: ";
                cin>>angle;
                break;
            case 2:
                cout<<"\nEnter Rotation angle: ";
                cin>>angle;
                break;
            case 3:
                cout<<"\nEnter Rotation angle: ";
                cin>>angle;
                break;
            default:
                break;
        }
        break;
    default:
        break;
    }
    glutDisplayFunc(display);
    glutMainLoop();
return 0;
}

```

## OUTPUT:



# LAB EXERCISE 9:

# Drawing Bezier curves

## **Experiment 9: Drawing Bezier curves. [ Virtual GLUT based demonstration]**

- Write a program to draw a cubic spline.

```
#include<windows.h>

#include <stdlib.h>

#include <GL/glut.h>

float Points[4][3] = {

    { 10,10,0 },

    { 5,10,2 },

    { -5,0,0 },

    {-10,5,-2}

};

unsigned int LOD=20;

void OnReshape(int w, int h)

{

    if (h==0)

        h=1;

    // set the drawable region of the window

    glViewport(0,0,w,h);

    // set up the projection matrix

    glMatrixMode(GL_PROJECTION);

    glLoadIdentity();

    // just use a perspective projection
```



```

gluPerspective(45,(float)w/h,0.1,100);

// go back to modelview matrix so we can move the objects about

glMatrixMode(GL_MODELVIEW);
}

//----- OnDraw()

void OnDraw() {

    // clear the screen & depth buffer

    glClear(GL_DEPTH_BUFFER_BIT|GL_COLOR_BUFFER_BIT);


    // clear the previous transform

    glLoadIdentity();

    // set the camera position

    gluLookAt( 1,10,30, //      eye pos

               0,0,0,   //      aim point

               0,1,0); //      up direction

    glColor3f(1,1,0);

    glBegin(GL_LINE_STRIP);

    // use the parametric time value 0 to 1

    for(int i=0;i!=LOD;++i) {

        float t = (float)i/(LOD-1);

        // the t value inverted

        float it = 1.0f-t;

        // calculate blending functions

        float b0 = it*it*it/6.0f;

        float b1 = (3*t*t*t - 6*t*t + 4)/6.0f;

        float b2 = (-3*t*t*t + 3*t*t + 3*t + 1)/6.0f;

        float b3 = t*t*t/6.0f;

        // sum the control points multiplied by their respective blending functions

        float x = b0*Points[0][0] +

                  b1*Points[1][0] +

                  b2*Points[2][0] +

                  b3*Points[3][0] ;

        float y = b0*Points[0][1] +

                  b1*Points[1][1] +

                  b2*Points[2][1] +

```

```

        b3*Points[3][1] ;

float z = b0*Points[0][2] +

        b1*Points[1][2] +

        b2*Points[2][2] +

        b3*Points[3][2] ;

    // specify the point
    glVertex3f( x,y,z );
}

glEnd();

// draw the control points
glColor3f(0,1,0);
glPointSize(3);
glBegin(GL_POINTS);
for(int i=0;i!=4;++i) {
    glVertex3fv( Points[i] );
}

glEnd();

// draw the hull of the curve
glColor3f(0,1,1);
glBegin(GL_LINE_STRIP);
for(int i=0;i!=4;++i) {
    glVertex3fv( Points[i] );
}

glEnd();

// currently we've been drawing to the back buffer, we need
// to swap the back buffer with the front one to make the image visible
glutSwapBuffers();
}

//----- OnInit()
void OnInit() {
    // enable depth testing
    glEnable(GL_DEPTH_TEST);
}

//----- OnExit()

```

```

void OnExit() {
}

//----- OnKeyPress()

void OnKeyPress(unsigned char key,int,int) {

    switch(key) {

        // increase the LOD

        case '+':

            ++LOD;

            break;

        // decrease the LOD

        case '-':

            --LOD;

            // have a minimum LOD value

            if (LOD<3)

                LOD=3;

            break;

        default:

            break;

    }

    // ask glut to redraw the screen for us...

    glutPostRedisplay();
}

//----- main()

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

    // initialise glut

    glutInit(&argc,argv);

    // request a depth buffer, RGBA display mode, and we want double buffering

    glutInitDisplayMode(GLUT_DEPTH|GLUT_RGBA|GLUT_DOUBLE);

    // set the initial window size

    glutInitWindowSize(640,480);

    // create the window

    glutCreateWindow("Cubic spline");

    // set the function to use to draw our scene

    glutDisplayFunc(OnDraw);
}

```

```

// set the function to handle changes in screen size
glutReshapeFunc(OnReshape);

// set the function for the key presses
glutKeyboardFunc(OnKeyPress);

// run our custom initialisation
OnInit();

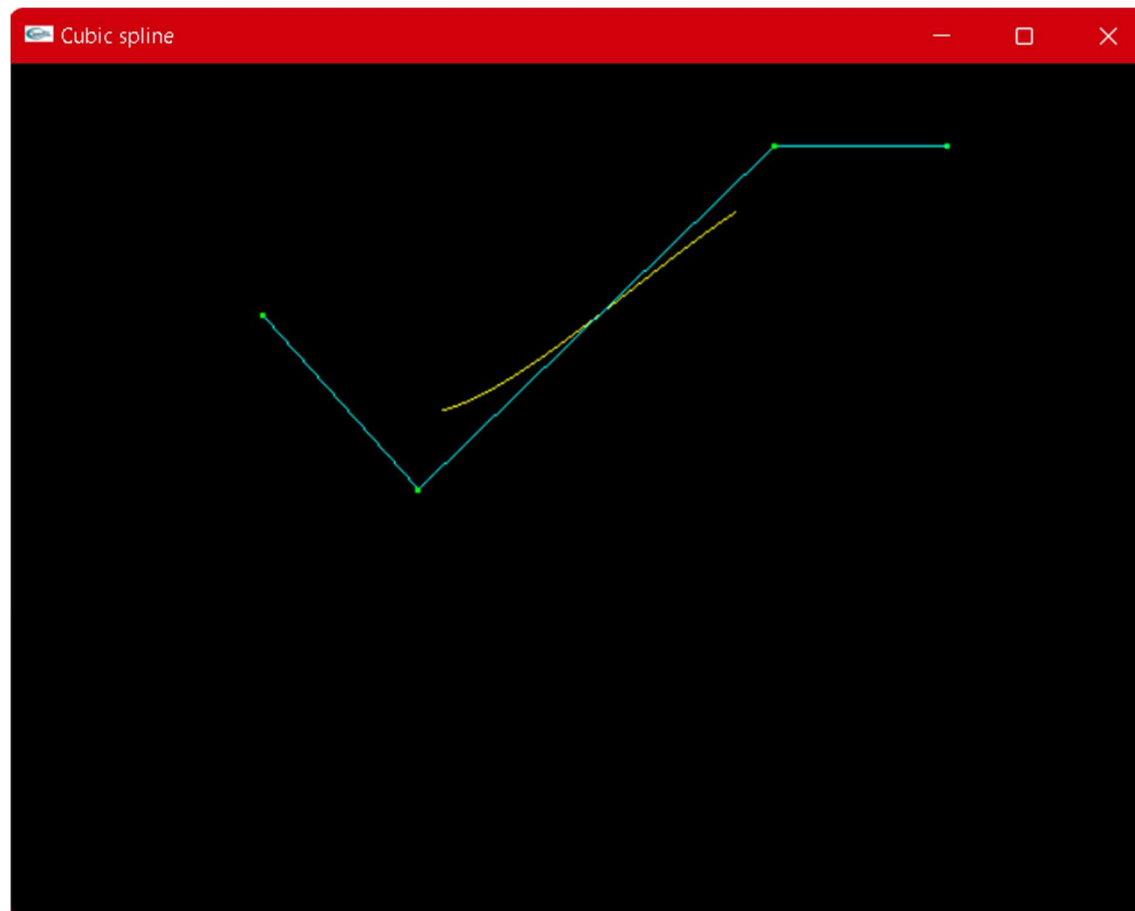
// set the function to be called when we exit
atexit(OnExit);

// this function runs a while loop to keep the program running.
glutMainLoop();

return 0;
}

```

## OUTPUT:



### **Experiment 9: Drawing Bezier curves. [ Virtual GLUT based demonstration]**

- WAP to draw a Bezier curve.

```
#include<windows.h>
```

```

#include <iostream>

#include <stdlib.h>

#include <GL/glut.h>

#include <math.h>

using namespace std;

class Point { //Point class for taking the points

public:

float x, y;

void setxy(float x2, float y2) {

x = x2; y = y2;

}

//operator overloading for '=' sign

const Point& operator=(const Point& rPoint) {

x = rPoint.x;

y = rPoint.y;

return *this;

}

};

int factorial(int n) {

if (n <= 1)

return(1);

else

n = n * factorial(n - 1);

return n;

}

float binomial_coff(float n, float k) {

float ans;

ans = factorial(n) / (factorial(k) * factorial(n - k));

return ans;

}

Point abc[20];

int SCREEN_HEIGHT = 500;

int points = 0;

int clicks = 4;

void myInit() {

```

```

glClearColor(1.0, 1.0, 1.0, 0.0);

glColor3f(0.0, 0.0, 0.0);

glPointSize(3);

glMatrixMode(GL_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0, 640.0, 0.0, 500.0);

}

void drawDot(int x, int y) {

glBegin(GL_POINTS);

glVertex2i(x, y);

glEnd();

glFlush();

}

void drawLine(Point p1, Point p2) {

glBegin(GL_LINES);

glVertex2f(p1.x, p1.y);

glVertex2f(p2.x, p2.y);

glEnd();

glFlush();

}

//Calculate the bezier point

Point drawBezier(Point PT[], double t) {

Point P;

P.x = pow((1 - t), 3) * PT[0].x + 3 * t * pow((1 - t), 2) * PT[1].x + 3 * (1 - t) * pow(t, 2) * PT[2].x + pow(t, 3) * PT[3].x;

P.y = pow((1 - t), 3) * PT[0].y + 3 * t * pow((1 - t), 2) * PT[1].y + 3 * (1 - t) * pow(t, 2) * PT[2].y + pow(t, 3) * PT[3].y;

return P;

}

//Calculate the bezier point [generalized]

Point drawBezierGeneralized(Point PT[], double t) {

Point P;

P.x = 0; P.y = 0;

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

P.x = P.x + binomial_coff((float)(clicks - 1), (float)i) * pow(t, (double)i) * pow((1 - t), (clicks - 1 - i)) * PT[i].x;

P.y = P.y + binomial_coff((float)(clicks - 1), (float)i) * pow(t, (double)i) * pow((1 - t), (clicks - 1 - i)) * PT[i].y;

}

}

```

```

return P;

}

void MouseClickFunc(int button, int state, int x, int y) {

// If left button was clicked

if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {

// Store where mouse was clicked, Y is backwards.

abc[points].setxy((float)x, (float)(SCREEN_HEIGHT - y));

points++;

// Draw the red dot.

drawDot(x, SCREEN_HEIGHT - y);

// If (click-amout) points are drawn do the curve.

if (points == clicks){

glColor3f(0.2, 1.0, 0.0);

// Drawing the control lines

for (int k = 0; k < clicks - 1; k++)

drawLine(abc[k], abc[k + 1]);

Point p1 = abc[0];

/* Draw each segment of the curve.Make t increment in smaller amounts for a more detailed curve.*/

for (double t = 0.0; t <= 1.0; t += 0.02) {

Point p2 = drawBezierGeneralized(abc, t);

cout << p1.x << " , " << p1.y << endl;

cout << p2.x << " , " << p2.y << endl;

cout << endl;

drawLine(p1, p2);

p1 = p2;

}

glColor3f(0.0, 0.0, 0.0);

points = 0;

}

}

}

void DisplayFunc() {

glClear(GL_COLOR_BUFFER_BIT);

glFlush();

}

```

```
int main(int argc, char* argv[]) {  
    glutInit(&argc, argv);  
  
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);  
    glutInitWindowSize(640, 500);  
    glutInitWindowPosition(100, 150);  
    glutCreateWindow("Bezier Curve");  
    glutMouseFunc(MouseClickFunc);  
    glutDisplayFunc(DisplayFunc);  
  
    myInit();  
    glutMainLoop();  
    return 0;  
}
```

## OUTPUT:

