Practical 1

AIM : Introduction to Graphics Programming

The graphical approach to programming allows a computer to process spatial representations in two or more dimensions. In contrast to text-based programming, which uses lines of code, graphical programming replaces text with pictures or symbols of physical things.

Graphical programming provides an approach that's more intuitive and less cumbersome for some programmers. It also can be a more effective way to introduce computer programming to visual learners. For example, researchers at the Lifelong Kindergarten Group have created a program called Scratch, which uses graphical programming to help children learn math and engage in creative thinking.

Graphical programming is often called visual programming or Visual Programming Language (VPL). It's different from Microsoft's Visual Basic, which defines pictures by using text-based language.

Introduction to BGI/OpenGL Library

OpenGL (Open Graphics Library) is a cross-platform, hardware-accelerated, language-independent, industrial standard API for producing 3D (including 2D) graphics. Modern computers have dedicated GPU (Graphics Processing Unit) with its own memory to speed up graphics rendering. OpenGL is the software interface to graphics hardware. In other words, OpenGL graphic rendering commands issued by your applications could be directed to the graphic hardware and accelerated.

We use 3 sets of libraries in our OpenGL programs:
Core OpenGL (GL)
OpenGL Utility Library (GLU)
OpenGL Utilities Toolkit (GLUT)

“graphics.h” is a header file that comes under WinBGIm Graphics Library that allows users to create graphics based programs in C++. Here are some of the predefined functions that allows users to create basic shapes, graphs easily.

1. Circle function

Syntax:

circle(int x, int y, int radius)

Used to draw a circle with center (x,y) and third parameter specifies the radius of the circle.

2. Closegraph function

Syntax:

closegraph()

Closes the graphics mode, deallocates all memory allocated by graphics system and restores the screen to the mode it was in before you called initgraph.

3. Ellipse function

Syntax:

ellipse(int x, int y, int stangle, int endangle, int xradius, int yradius)

Used to draw an ellipse (x,y) are coordinates of center of the ellipse, stangle is the starting angle, end angle is the ending angle, and fifth and sixth parameters specifies the X and Y radius of the ellipse.

4. Floodfill function

Syntax:

floodfill(int x, int y, int border)

Used to fill an enclosed area. Current fill pattern and fill color is used to fill the area.(x, y) is any point on the screen if (x,y) lies inside the area then inside will be filled otherwise outside will be filled,border specifies the color of boundary of area.

5. getmaxx and getmaxy function

Syntax:

getmaxx()

getmaxy()

Returns the maximum X & Y coordinate for current graphics mode and driver.

6. putimage function

Syntax:

putimage(int left, int top, void *ptr, int op)

Puts the bit image previously saved with getimage back onto the screen, with the upper left corner of the image placed at (left, top). ptr points to the area in memory where the source image is stored. The op argument specifies a operator that controls how the color for each destination pixel on screen is computed

7. setfillstyle function

Syntax:

setfillstyle(int pattern, int color)

Sets the current fill pattern and fill color.

8. setlinestyle function

Syntax:

setlinestyle(int linestyle, unsigned upattern, int thickness)

Sets the current line style, pattern and thickness of the line.

9. getx & gety function

Syntax:

getx()
gety()

Returns the X & Y coordinate of the current position.

10. detectgraph function

Syntax:

detectgraph(int *graphicsdriver ,int *graphicsmode)

Used to find out the current graphics driver and mode.

11. fillellipse function

Syntax:

fillellipse(int x, int y, int xradius, int yradius)

x and y are coordinates of center of the ellipse, x-radius and y-radius are x and y radius of ellipse respectively.

12. getcolor function

Syntax:

getcolor()

Returns the current drawing color.

13. getmaxmode function

Syntax:

getmaxmode()

Returns the graphics mode set by initgraph or setgraphmode.

14. initgraph function

Syntax:

initgraph (int *graphdriver, int *graphmode, char *pathtodriver)

Initializes the graphics system by loading a graphics driver from disk (or validating a registered driver), and putting the system into graphics mode.

15. line function

Syntax:

line(int x1, int y1, int x2, int y2)

Used to draw a line from a point(x1,y1) to point(x2,y2) i.e. (x1,y1) and (x2,y2) are end points of the line.

16. putpixel function

Syntax:

putpixel(int x, int y, int color)

Plots a pixel at location (x, y) of specified color.

17. setcolor function

Syntax:

setcolor(int color)

Each color is assigned a number. A total of 16 colors are available. Sets the current color to color indicated by integer.

18. textwidth function

Syntax:

textwidth(char *string)

Returns the width of a string in pixels.

19. drawpoly function

Syntax:

drawpoly(int num, int *polypoints)

Used to draw polygons i.e. triangle, rectangle, pentagon, hexagon etc.

20. getpixel function

Syntax:

getpixel(int x, int y)

Returns the color of pixel present at location(x, y).

21. graphresult function

Syntax:

graphresult()

Returns the error code for the last graphics operation that reported an error

22. rectangle function

Syntax:

rectangle(int left, int top, int right, int bottom)

Used to draw a rectangle. Coordinates of left top and right bottom corner are required to draw the rectangle.

23. fillpoly function

Syntax:

fillpoly(int num, int *polypoints)

Draws and fills a polygon. It require same arguments as drawpoly.

24. sector function

Syntax:

sector(int x, int y, int stangle, int endangle, int xradius, int yradius)

Draws and fills an elliptical pie slice.

Practical 2

AIM : To implement DDA (Digital Differential Analysis) Algorithm

CODE

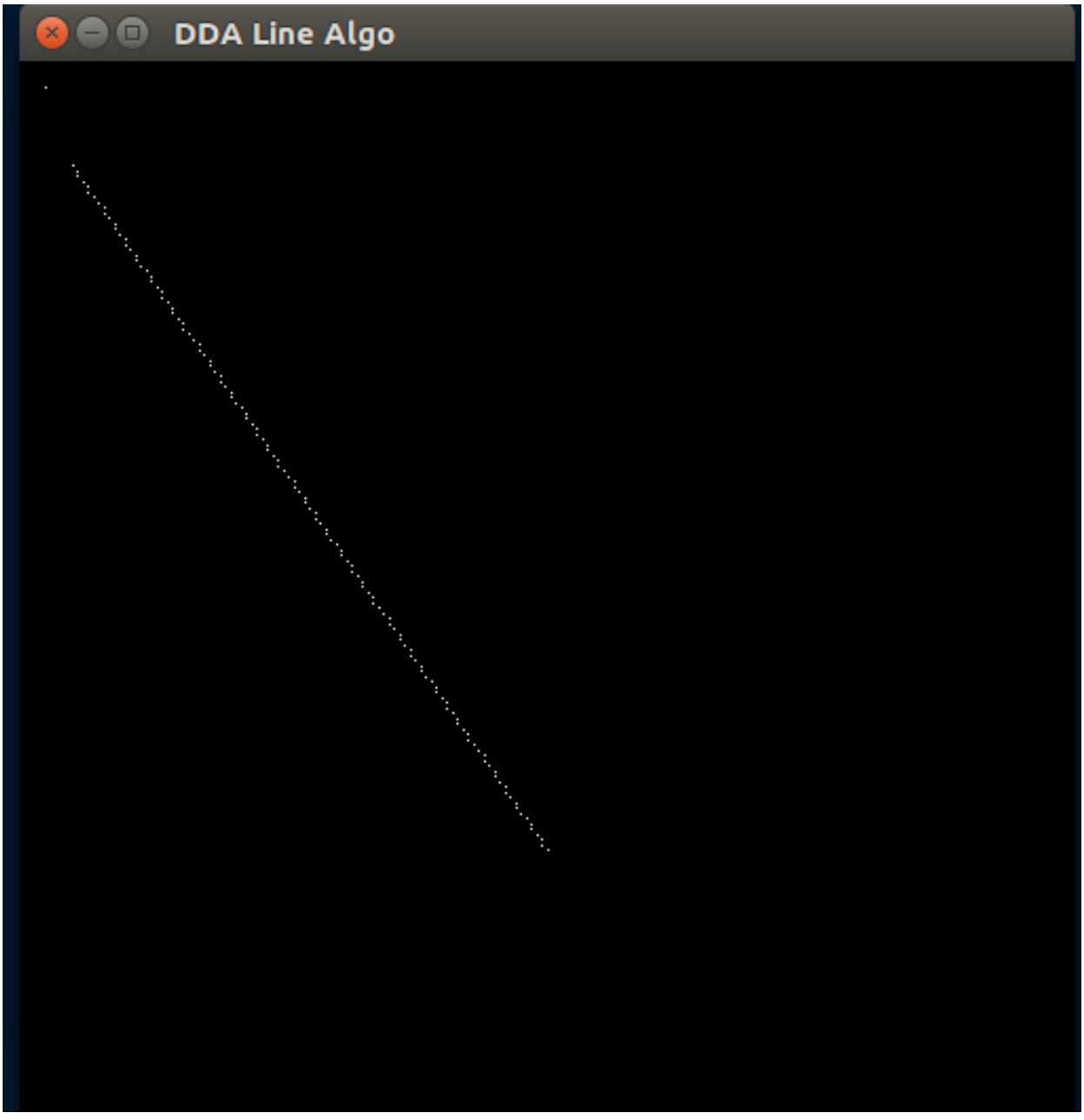
```
#include<GL/glut.h>
#include<iostream>

using namespace std;
float x1,x2,y1,y2;

float round(float x)
{
	if(x-(int)x>=0.5)
		return (float)((int)(x+1));
	else
		return (float)((int)x);
}

void display(void)
{
	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);
	glVertex2i(x,y);
	glEnd();
```

```
	glBegin(GL_POINTS);
	glVertex2i(5,5);
	glEnd();
	float xx,yy;

	for (k=1 ;k<=step;k++)
	{
		x= x + Xin;
		y= y + Yin;
		xx = round(x);
		yy = round(y);
		cout<<xx<<','<<yy<<endl;
		glBegin(GL_POINTS);
		glVertex2i(xx,yy);
		glEnd();
	}

	glFlush();
}

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);
	glutInitWindowSize (500, 500);
	glutInitWindowPosition (100,100);
	glutCreateWindow ("DDA Line Algo");
```

```
	glClearColor(0.0,0.0,0.0,0.0);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(0,200,200,0);

	glutDisplayFunc(display);
	glutMainLoop();

	return 0;
}
```

OUTPUT

```
e Algo
shivank@shivank-Vostro-5568: ~/Documents/CGLAB
shivank@shivank-Vostro-5568:~/Documents/CGLAB$ g++ DDAA.cpp -o abc -lGL -lGLU -lglut
shivank@shivank-Vostro-5568:~/Documents/CGLAB$ ./abc
Enter the value of x1 : 10
Enter the value of y1 : 20
Enter the value of x2 : 100
Enter the value of y2 : 150
11.21
```



Practical 3

AIM : To implement Bresenham's Line Drawing algorithm

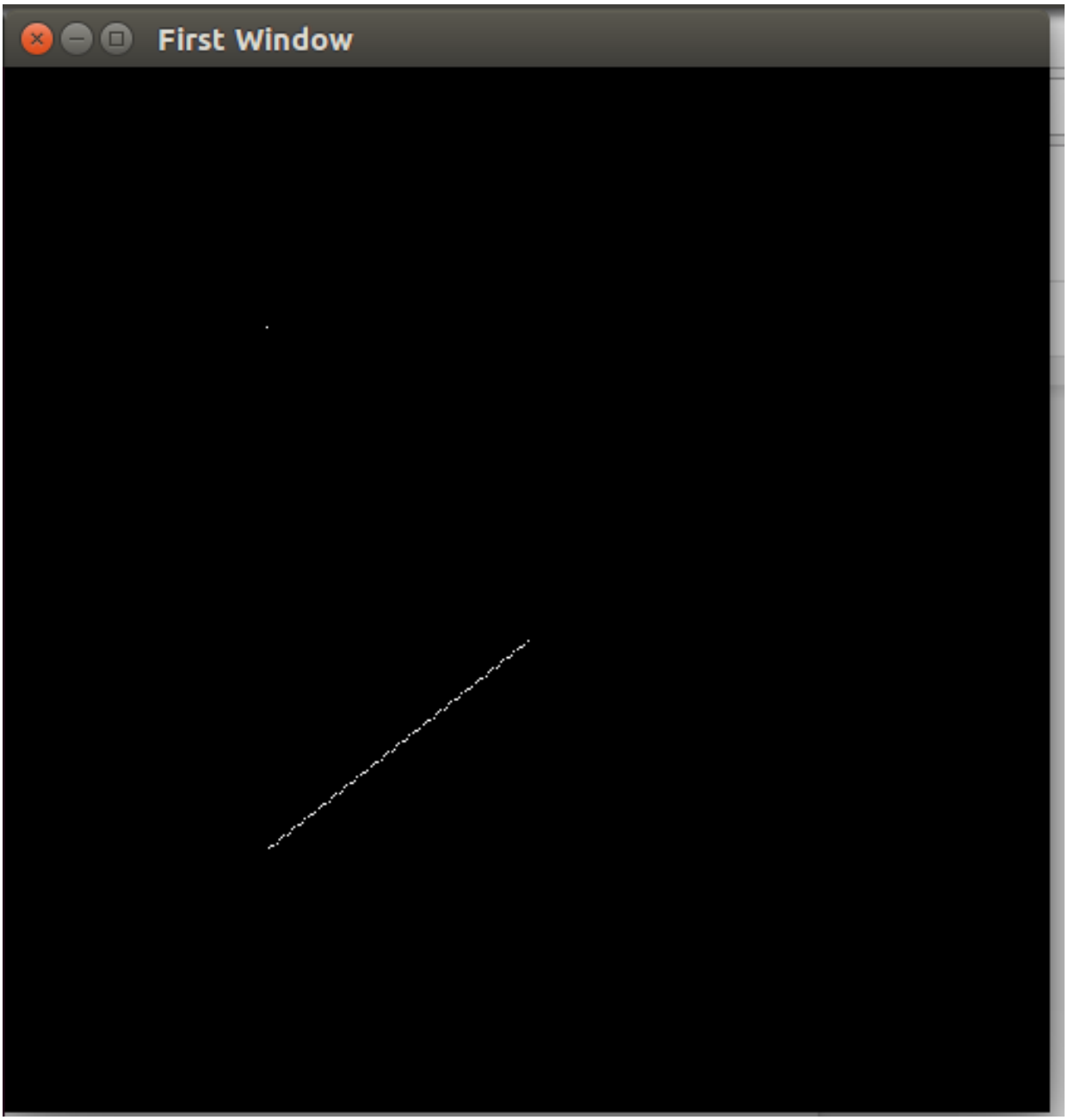
CODE

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

using namespace std;
int X1,Y1,X2,Y2;

void line(void){
	int x=X1,y=Y1,dx,dy,po,pn;
	dx=X2-X1;
	dy=Y2-Y1;
	pn=2*dy-dx;
	glBegin(GL_POINTS);
	glVertex2i(x,y);
	glEnd();
	for(int i=0;i<abs(dx);i++){
		po=pn;
		if(pn<0){
			x=x+1;
			pn=po + 2*dy;
		}
		else{
			x=x+1;
			y=y+1;
			pn = po + 2*dy -2*dx;
		}
		glBegin(GL_POINTS);
		glVertex2i(x,400-y);
		glEnd();
	}

	glFlush();
}

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

```
	cout<<"Enter X1 :";cin>>X1;
	cout<<"Enter Y1 :";cin>>Y1;
	cout<<"Enter X2 :";cin>>X2;
	cout<<"Enter Y2 :";cin>>Y2;

	glutInit(&argc, argv);
	glutInitWindowSize(500,500);
	glutCreateWindow("First Window");
	gluOrtho2D(0,400,400,0);
	glutDisplayFunc(line);
	glutMainLoop();

	return 0;
}
```

OUTPUT

```
shivank@shivank-Vostro-5568: ~/Documents/CGLAB
shivank@shivank-Vostro-5568:~/Documents/CGLAB$ g++ Bresenham.cpp -o abc -lGL -lGLU -lglut
shivank@shivank-Vostro-5568:~/Documents/CGLAB$ ./abc
Enter X1 :100
Enter Y1 :100
Enter X2 :200
Enter Y2 :180
```



Practical 4

AIM : To implement Midpoint Line Drawing Algorithm

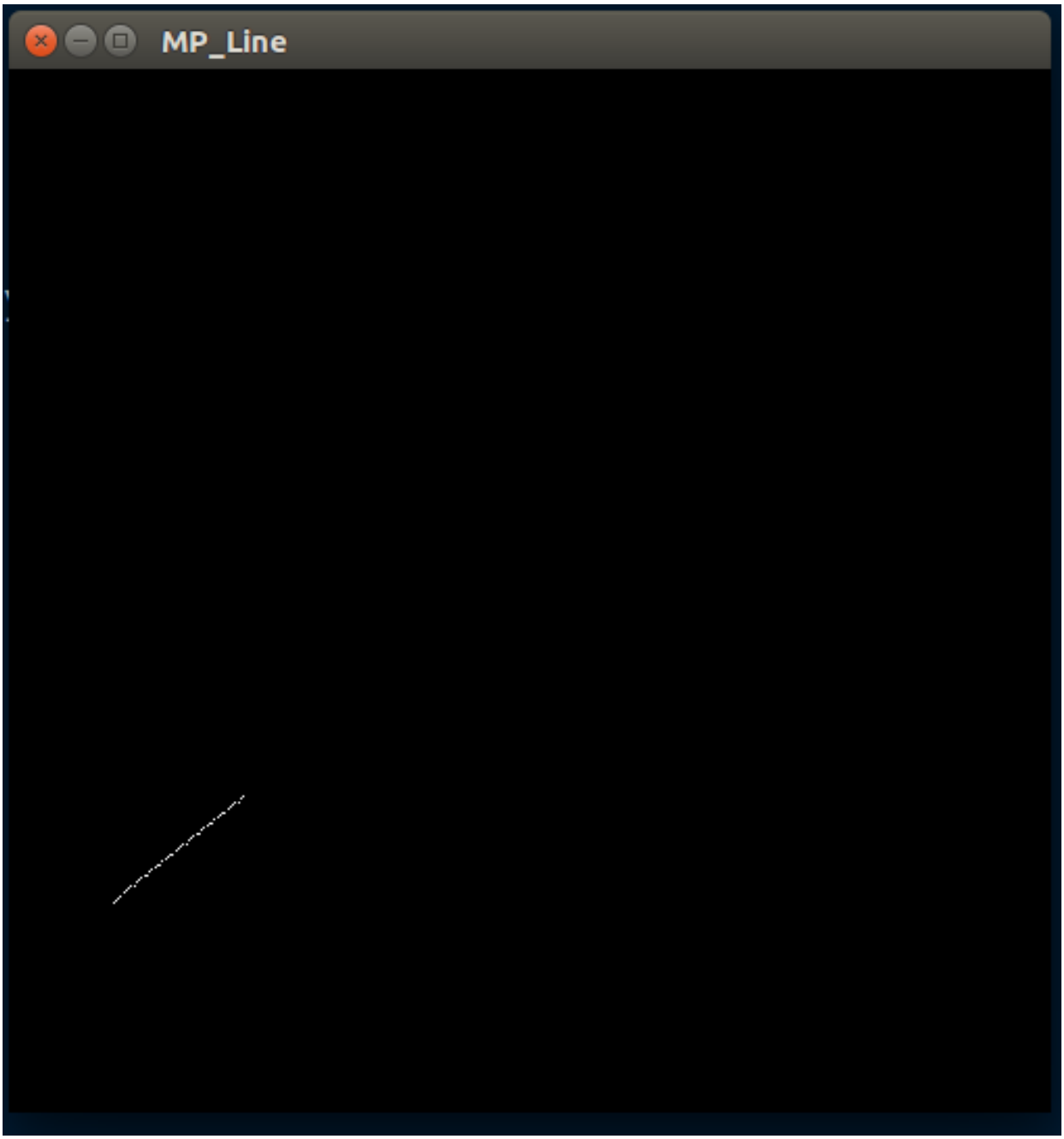
CODE

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

using namespace std;

int X1,Y1,X2,Y2;

void Line(){
	int dx,dy,de,dne,x,y,po,pn;
	dx=X2-X1;
	dy=Y2-Y1;
	x=X1;y=Y1;
	glBegin(GL_POINTS);
		glVertex2i(x,400-y);
	glEnd();
	pn=2*dy + dx;
	do{
		if(pn>0){
			x++;y++;
			po=pn;
			pn = po + 2 * (dy-dx);
		}
		else
		{
			x++;
			po = pn;
			pn = po + 2*dy;
		}

		glBegin(GL_POINTS);
		glVertex2i(x,400-y);
		glEnd();

	}while(x!=X2);
```

```
	glFlush();
}

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

	cout<<"Enter X1 :";cin>>X1;
	cout<<"Enter Y1 :";cin>>Y1;
	cout<<"Enter X2 :";cin>>X2;
	cout<<"Enter Y2 :";cin>>Y2;

	glutInit(&argc, argv);
	glutInitWindowSize(500,500);
	glutCreateWindow("MP_Line");
	gluOrtho2D(0,400,400,0);
	glutDisplayFunc(Line);
	glutMainLoop();
}
```

OUTPUT

```
shivank@shivank-Vostro-5568: ~/Documents/CGLAB
shivank@shivank-Vostro-5568:~/Documents/CGLAB$ g++ Mid_Point_line.cpp -o abc -lGL -lGLU -lglut
shivank@shivank-Vostro-5568:~/Documents/CGLAB$ ./abc
Enter X1 :40
Enter Y1 :80
Enter X2 :90
Enter Y2 :120
```



Practical 5

AIM : To implement Midpoint Circle Drawing Algorithm

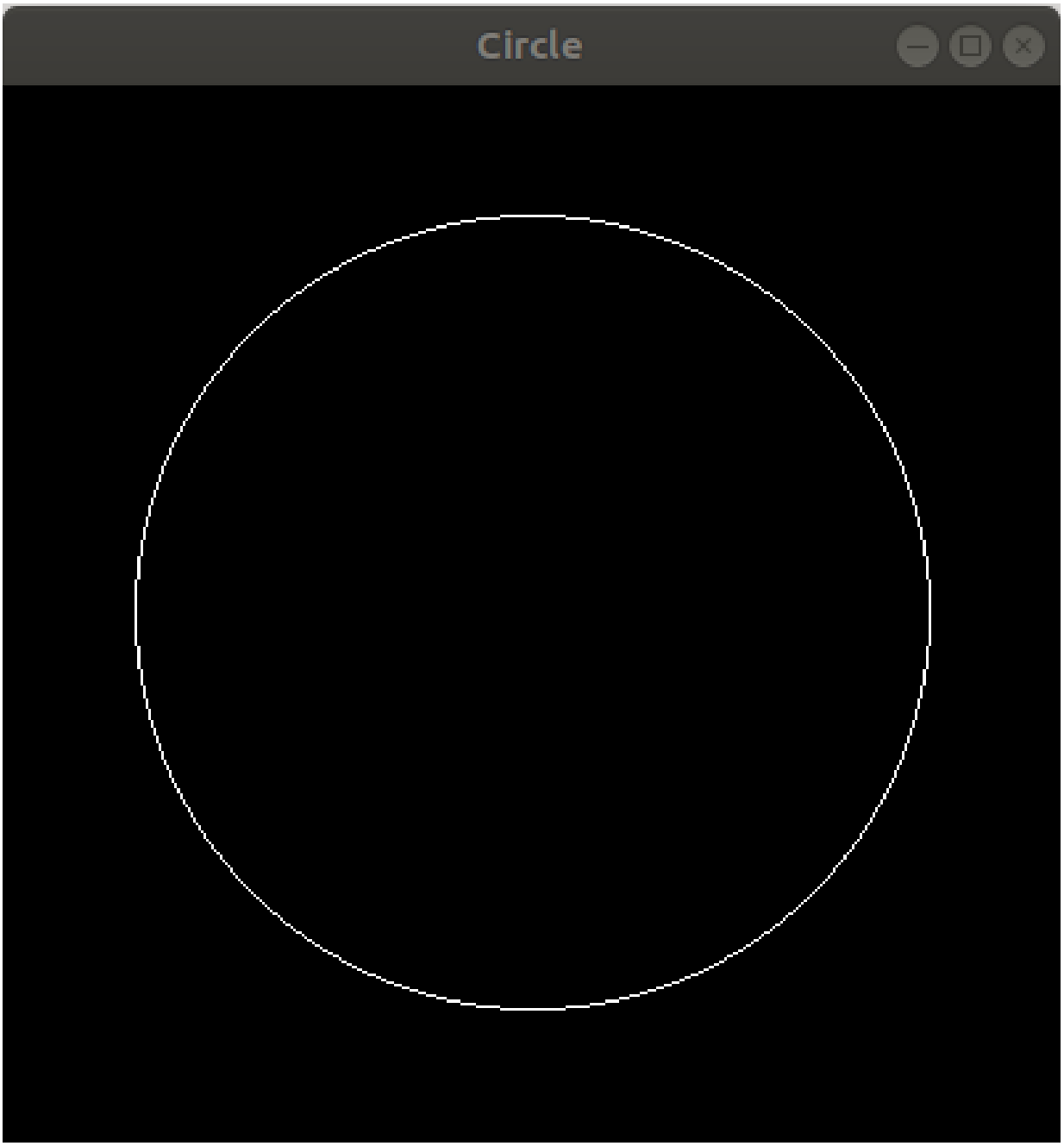
CODE

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

using namespace std;

int xc,yc,r;

void circles(){
	int x,y,P;
	x=r;
	y=0;
	P=1-r;
	while(x>y){

		glBegin(GL_POINTS);
		glVertex2i(x+xc,y+yc);
		glVertex2i(-x+xc,y+yc);
		glVertex2i(x+xc,-y+yc);
		glVertex2i(-x+xc,-y+yc);

		glVertex2i(y+xc,x+yc);
		glVertex2i(-y+xc,x+yc);
		glVertex2i(y+xc,-x+yc);
		glVertex2i(-y+xc,-x+yc);
		glEnd();
		y++;
		if (P <= 0)
			P = P + 2*y + 1;
		else
		{
			x--;
			P = P + 2*y - 2*x + 1;
		}
	}
	glBegin(GL_POINTS);
```

```
        glVertex2i(x+xc,y+yc);
        glVertex2i(-x+xc,y+yc);
        glVertex2i(x+xc,-y+yc);
        glVertex2i(-x+xc,-y+yc);

        glVertex2i(y+xc,x+yc);
        glVertex2i(-y+xc,x+yc);
        glVertex2i(y+xc,-x+yc);
        glVertex2i(-y+xc,-x+yc);
        glEnd();

        glFlush();
}

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

        cout<<"Enter Center position:\nX :";cin>>xc;
        cout<<"Y :";cin>>yc;
        cout<<"Enter radius :";cin>>r;

        glutInit(&argc, argv);
        glutInitWindowSize(400,400);
        glutCreateWindow("Circle");
        gluOrtho2D(-200,200,200,-200);
        glutDisplayFunc(circles);
        glutMainLoop();

}
```

OUTPUT

```
File  Edit  View  Search  Terminal  Help
shivank@shivank-Vostro-5568:~/Documents/CGLAB$ g++ MidPtCircle.cpp -o a -lGL -lGLU -lglut
shivank@shivank-Vostro-5568:~/Documents/CGLAB$ ./a
Enter Center postion:
X :0
Y :0
Enter radius :150
```



Practical 6

AIM : To implement Midpoint Ellipse Drawing Algorithm

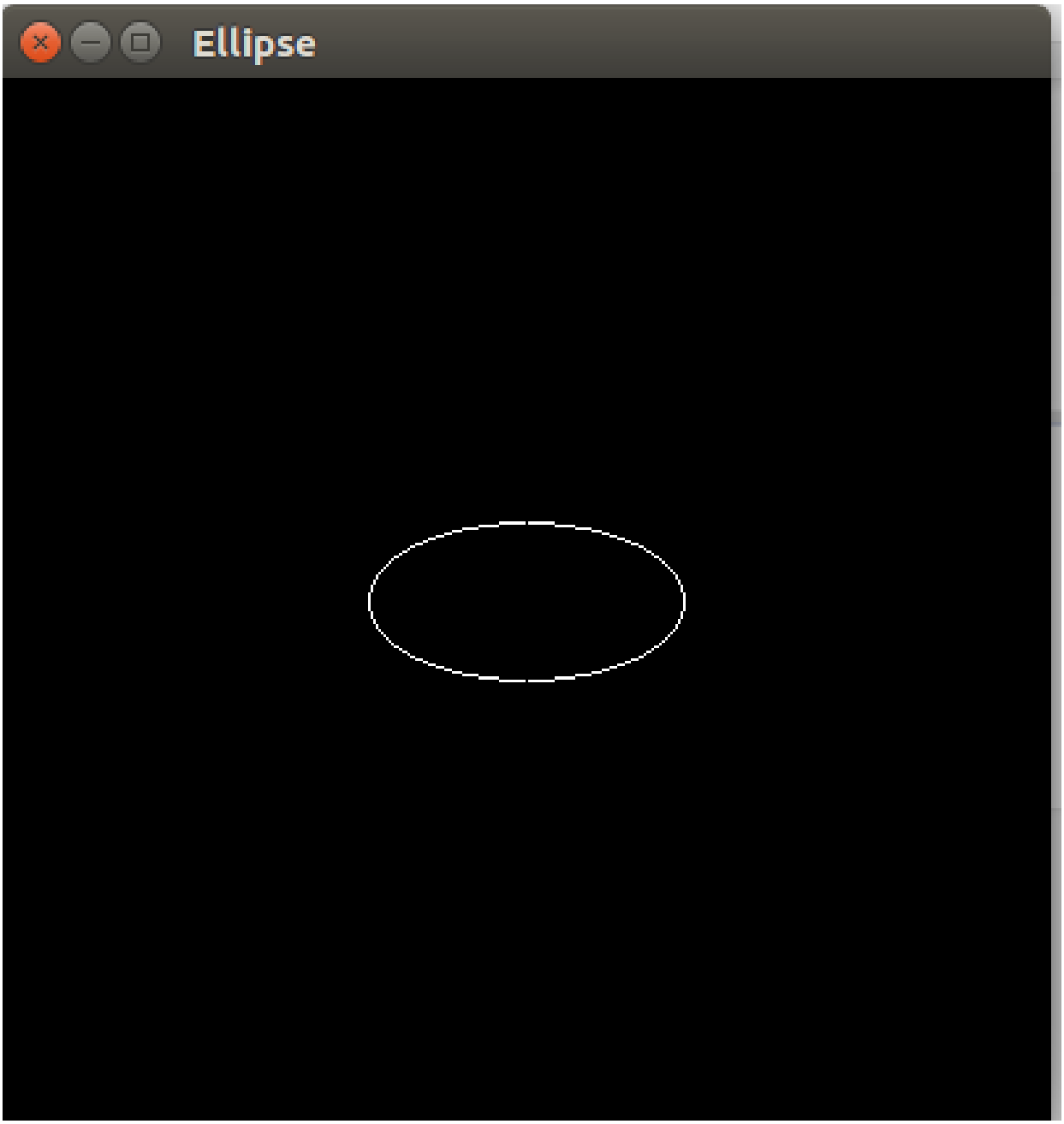
CODE

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

using namespace std;

int xc,yc,x,y;float p;
long rx,ry;

void ellipse(){
	//Region 1
	p=ry*ry-rx*rx*ry+rx*rx/4;
	x=0;y=ry;
	while(2.0*ry*ry*x <= 2.0*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;
		}
		glBegin(GL_POINTS);
		glVertex2i(xc+x,yc+y);
		glVertex2i(xc+x,yc-y);
		glVertex2i(xc-x,yc+y);
		glVertex2i(xc-x,yc-y);
		glEnd();
	}

	//Region 2
	p=ry*ry*(x+0.5)*(x+0.5)+rx*rx*(y-1)*(y-1)-rx*rx*ry*ry;
	while(y > 0)
```

```
	{
		if(p <= 0)
		{
			x++;y--;
			p = p+2*ry*ry*x-2*rx*rx*y+rx*rx;
		}
		else
		{
			y--;
			p = p-2*rx*rx*y+rx*rx;
		}
		glBegin(GL_POINTS);
		glVertex2i(xc+x,yc+y);
		glVertex2i(xc+x,yc-y);
		glVertex2i(xc-x,yc+y);
		glVertex2i(xc-x,yc-y);
		glEnd();
	}

	glFlush();
}

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

	cout<<"Enter coordinates of centre : ";
	cin>>xc>>yc;
	cout<<"Enter x,y radius of ellipse: ";
	cin>>rx>>ry;

	glutInit(&argc, argv);
	glutInitWindowSize(400,400);
	glutCreateWindow("Circle");
	gluOrtho2D(-200,200,200,-200);
	glClear(GL_COLOR_BUFFER_BIT);
	glutDisplayFunc(ellipse);
	glutMainLoop();

}
```

OUTPUT

```
shivank@shivank-Vostro-5568:~/Documents/CGLAB$ ./abc
Enter coordinates of centre : 0 0
Enter x,y radius of ellipse: 60 30
```



Practical 7-8

AIM : To implement 2-D transformations on graphics

(a)

1. Translation
2. Scaling
3. Rotation

(b)

1. Reflection
2. Shearing

CODE

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

using namespace std;
int x[4]={150,150,-150,-150},y[4]={100,-100,-100,100};

double round(double d)
{
	return floor(d + 0.5);
}

void Trans(double X, double Y)
{
	glClear(GL_COLOR_BUFFER_BIT);
	glBegin(GL_LINE_LOOP);
	for (int i = 0; i < 4; i++)
	{
		glVertex2i(round(x[i] + X), round(y[i] + Y));
	}
	glEnd();
	glFlush();
}

void Scale(double X, double Y)
{
	glClear(GL_COLOR_BUFFER_BIT);
```

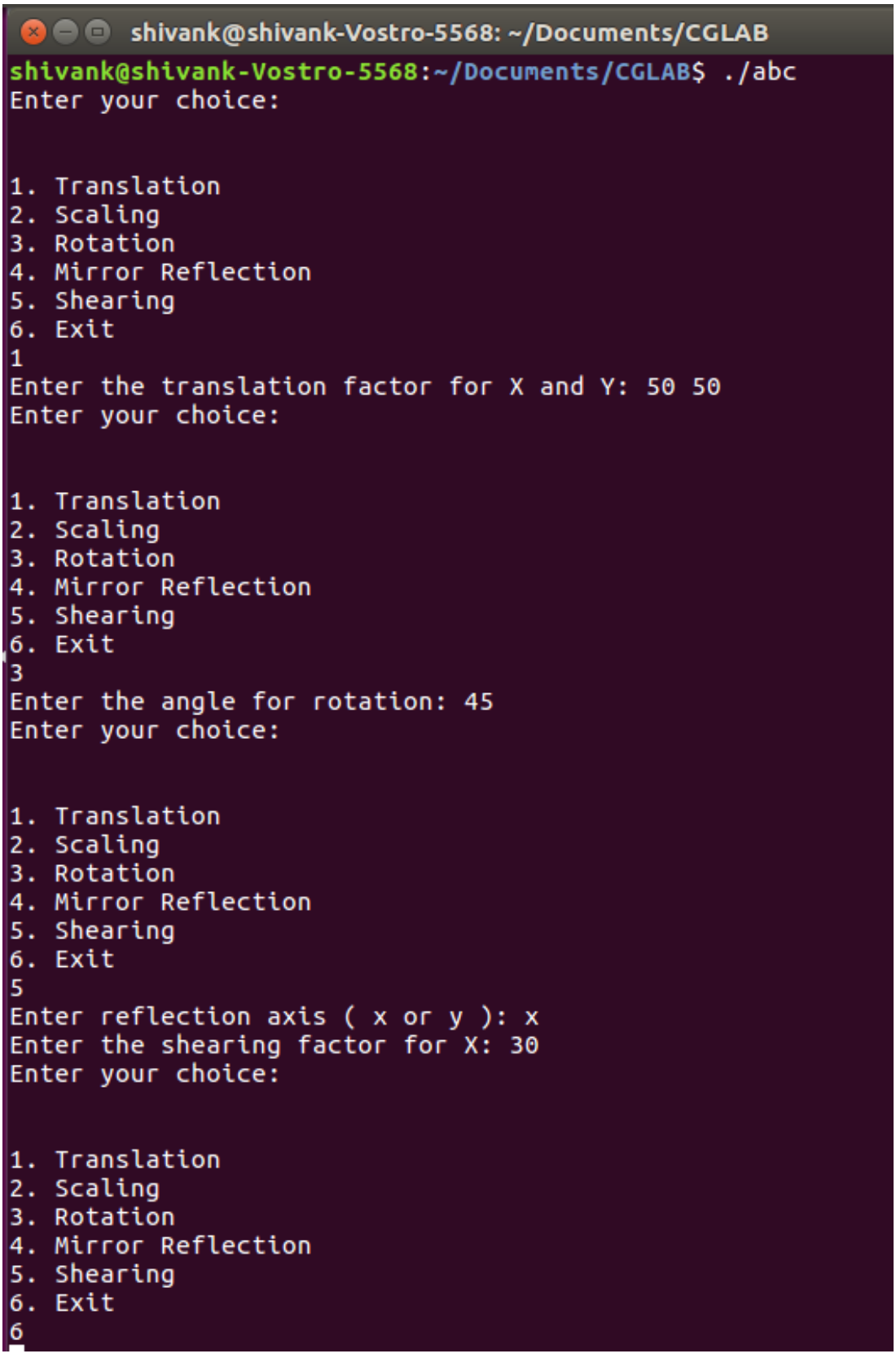
```
	glBegin(GL_LINE_LOOP);
	for (int i = 0; i < 4; i++)
	{
		glVertex2i(round(x[i] * X), round(y[i] * Y));
	}
	glEnd();
	glFlush();
}

void Rotate(double angleRad)
{
	glClear(GL_COLOR_BUFFER_BIT);
	glBegin(GL_LINE_LOOP);
	for (int i = 0; i < 4; i++)
	{
		glVertex2i(round((x[i] * cos(angleRad)) - (y[i] * sin(angleRad))), round((x[i] *
sin(angleRad)) + (y[i] * cos(angleRad))));
	}
	glEnd();
	glFlush();
}

void Reflect(char reflectionAxis)
{
	glClear(GL_COLOR_BUFFER_BIT);
	glBegin(GL_LINE_LOOP);

	if (reflectionAxis == 'x' || reflectionAxis == 'X')
	{
		for (int i = 0; i < 4; i++)
		{
			glVertex2i(round(x[i]), round(y[i] * -1));

		}
	}
	else if (reflectionAxis == 'y' || reflectionAxis == 'Y')
	{
		for (int i = 0; i < 4; i++)
		{
			glVertex2i(round(x[i] * -1), round(y[i]));
```

```
                }
        }
        glEnd();
        glFlush();
}

void Shear(char shearingAxis,double X,double Y)
{
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_LINE_LOOP);

        if (shearingAxis == 'x' || shearingAxis == 'X')
        {
                glVertex2i(x[0], y[0]);

                glVertex2i(x[1] + X, y[1]);
                glVertex2i(x[2] + X, y[2]);

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

                glVertex2i(x[2], y[2] + Y);
                glVertex2i(x[3], y[3] + Y);
        }
        glEnd();
        glFlush();
}

void transformation(void){
        double X,Y;
        double angle, angleRad;
        char Axis;
        glBegin(GL_LINE_LOOP);
                glVertex2i(150,100);
                glVertex2i(150,-100);
                glVertex2i(-150,-100);
```

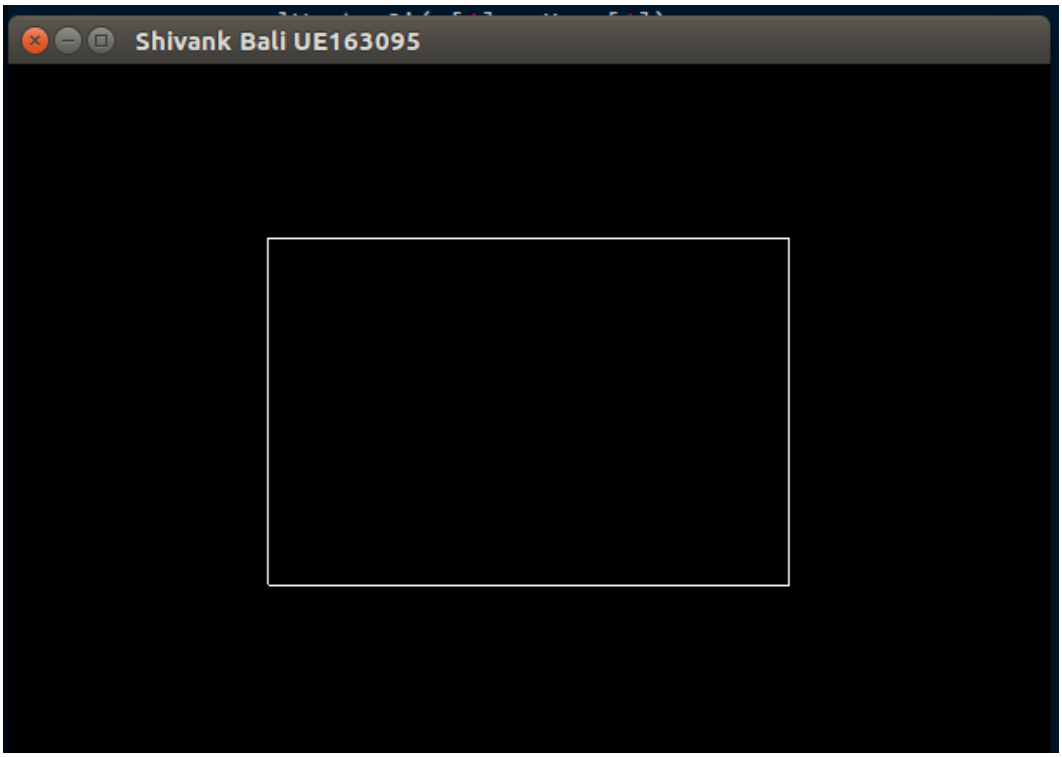
```
        glVertex2i(-150,100);
glEnd();
glFlush();
int choice;
do{
        cout << "Enter your choice:\n\n" << endl;
        cout << "1. Translation" << endl;
        cout << "2. Scaling" << endl;
        cout << "3. Rotation" << endl;
        cout << "4. Mirror Reflection" << endl;
        cout << "5. Shearing" << endl;
        cout << "6. Exit" << endl;

        cin >> choice;


        if (choice == 1)
        {
                cout << "Enter the translation factor for X and Y: "; cin >> X >> Y;
                Trans(X,Y);
        }
        else if (choice == 2)
        {
                cout << "Enter the scaling factor for X and Y: "; cin >> X >> Y;
                Scale(X,Y);
        }
        else if (choice == 3)
        {
                cout << "Enter the angle for rotation: "; cin >> angle;
                angleRad = angle * 3.1416 / 180;
                Rotate(angleRad);
        }
        else if (choice == 4)
        {
                cout << "Enter reflection axis ( x or y ): "; cin >> Axis;
                Reflect(Axis);
        }
        else if (choice == 5)
        {
                cout << "Enter shearing axis ( x or y ): "; cin >> Axis;
```

```
                        if (Axis == 'x' || Axis == 'X')
                        {
                                cout << "Enter the shearing factor for X: "; cin >> X;
                        }
                        else
                        {
                                cout << "Enter the shearing factor for Y: "; cin >> Y;
                        }
                        Shear(Axis,X,Y);
                }
        }while(choice<6&&choice>0);
}

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

        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(600,400);
        glutInitWindowPosition(50,50);
        glutCreateWindow("Shivank Bali UE163095");

        gluOrtho2D(-300,300,-200,200);
        glClear(GL_COLOR_BUFFER_BIT);
        glutDisplayFunc(transformation);
        glutMainLoop();

        return 0;
}
```

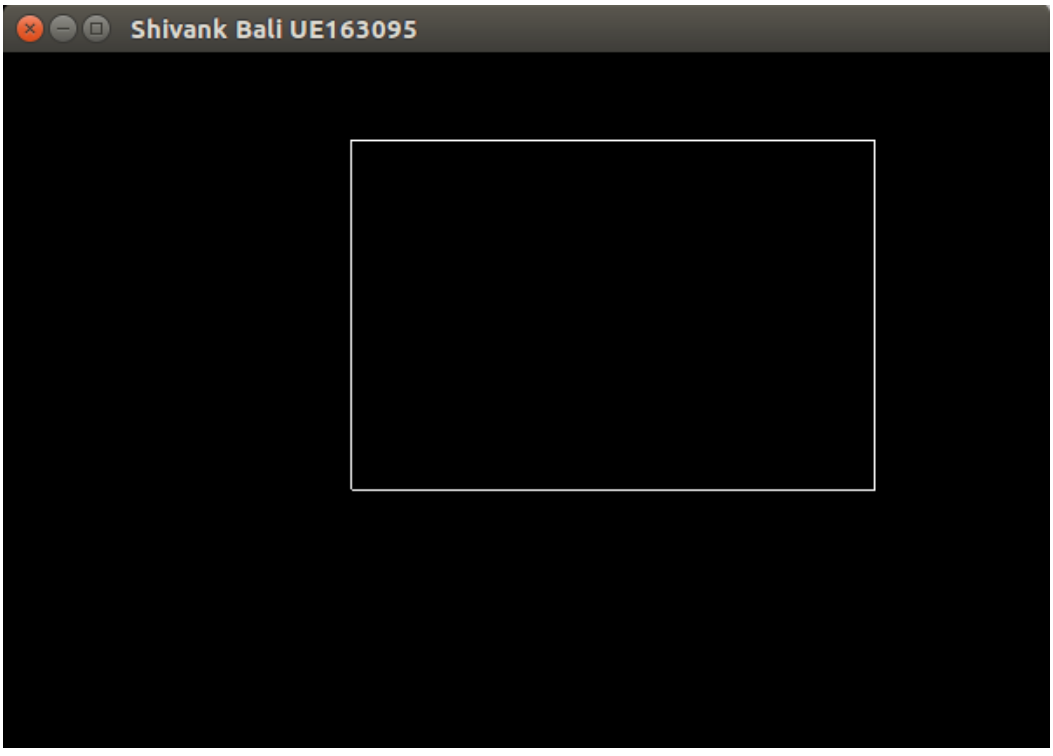
OUTPUT


shivank@shivank-Vostro-5568: ~/Documents/CGLAB
shivank@shivank-Vostro-5568:~/Documents/CGLAB$ ./abc
Enter your choice:
1. Translation
2. Scaling
3. Rotation
4. Mirror Reflection
5. Shearing
6. Exit
1
Enter the translation factor for X and Y: 50 50
Enter your choice:
1. Translation
2. Scaling
3. Rotation
4. Mirror Reflection
5. Shearing
6. Exit
3
Enter the angle for rotation: 45
Enter your choice:
1. Translation
2. Scaling
3. Rotation
4. Mirror Reflection
5. Shearing
6. Exit
5
Enter reflection axis (x or y): x
Enter the shearing factor for X: 30
Enter your choice:
1. Translation
2. Scaling
3. Rotation
4. Mirror Reflection
5. Shearing
6. Exit
6

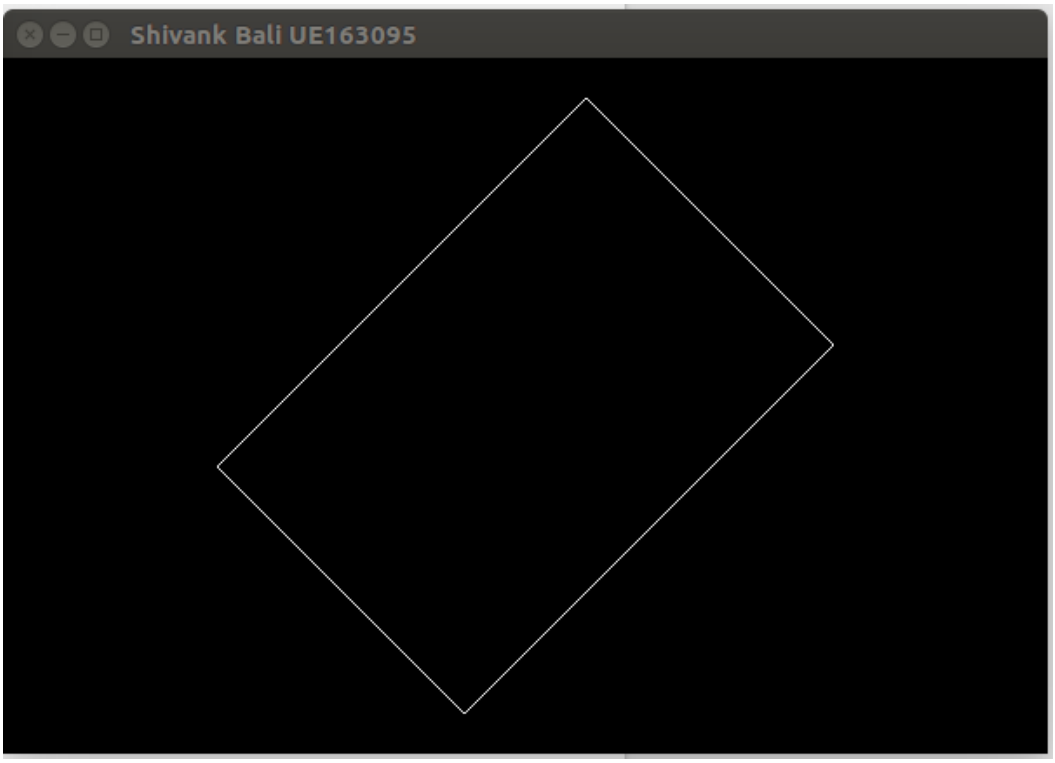
BEFORE TRANSFORMATION



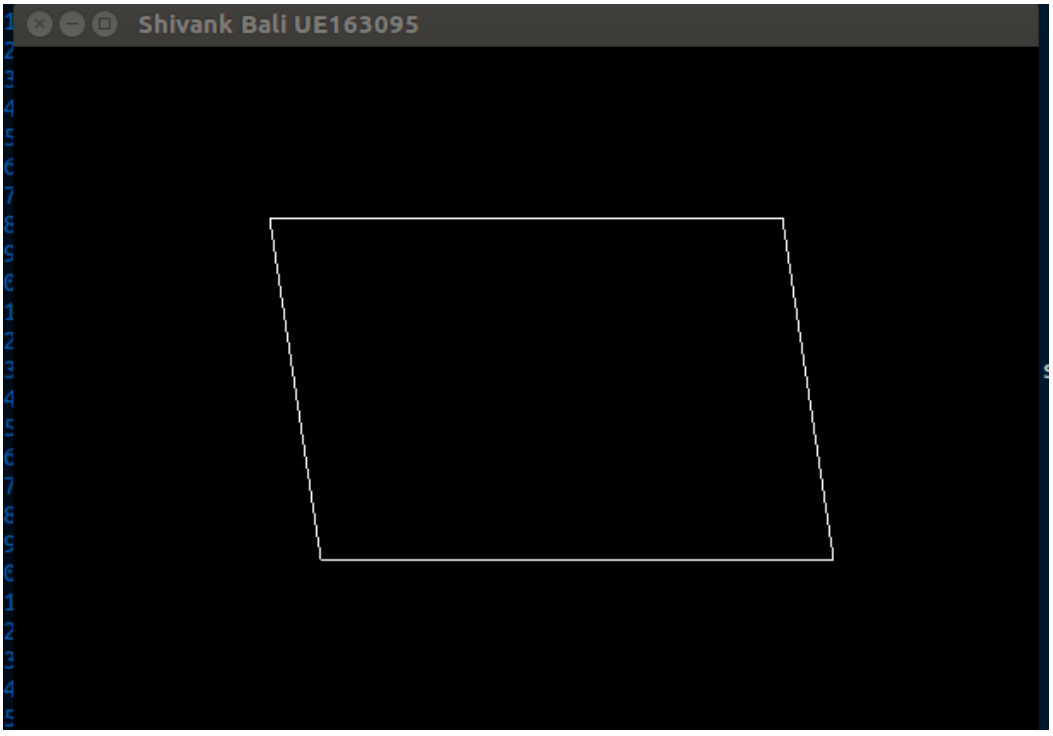
AFTER TRANSLATION



AFTER ROTATION



AFTER SHEARING



Practical 9-10

AIM : To implement 3-D transformations on graphics

(a)

1. Translation
2. Scaling
3. Rotation

(b)

1. Reflection
2. Shearing

CODE

```
#include<bits/stdc++.h>
using namespace std;

double  X[4]={3,4,5,6},
        Y[4]={4,3,6,2},
        Z[4]={7,8,3,1};

double round(double d)
{
        return floor(d + 0.5);
}

void Display(){
        cout<<"\tX\t\tY\t\tZ\n";
        for(int i=0;i<3;i++){
                cout<<"\t"<<X[i]<<"\t";
                cout<<"\t"<<Y[i]<<"\t";
                cout<<"\t"<<Z[i]<<"\n";
        }
}


void Trans(double x, double y, double z)
{
        for (int i = 0; i < 4; i++)
        {
                X[i]+=x;
```

```
			Y[i]+=y;
			Z[i]+=z;
		}
		Display();
}

void Scale(double x, double y,double z)
{
		for (int i = 0; i < 4; i++)
		{
			X[i]*=x;
			Y[i]*=y;
			Z[i]*=z;
		}
		Display();
}

void Rotate(double angleRad)
{
		for (int i = 0; i < 4; i++)
		{
			X[i] = round((X[i] * cos(angleRad)) - (Y[i] * sin(angleRad)));
			Y[i] = round((X[i] * sin(angleRad)) + (Y[i] * cos(angleRad)));

		}
		Display();
}

void Reflect(char reflectionAxis)
{
		if (reflectionAxis == 'x' || reflectionAxis == 'X')
		{
			for (int i = 0; i < 4; i++)
			{
				X[i]*=-1;

			}
		}
		else if (reflectionAxis == 'y' || reflectionAxis == 'Y')
		{
```

```
		for (int i = 0; i < 4; i++)
		{
			Y[i]*=-1;
		}
	}
	else if (reflectionAxis == 'z' || reflectionAxis == 'Y')
	{
		for (int i = 0; i < 4; i++)
		{
			Z[i]*=-1;
		}
	}
	Display();
}

void Shear(char shearingAxis,double x,double y)
{
	for(int i=0;i<4;i++){
		if (shearingAxis == 'x' || shearingAxis == 'X')
		{
			X[i] = X[i] + x*Y[i] +y*Z[i];
		}
		else if (shearingAxis == 'y' || shearingAxis == 'Y')
		{
			Y[i] = Y[i] + x*X[i] +y*Z[i];
		}
		else if (shearingAxis == 'z' || shearingAxis == 'Z')
		{
			Z[i] = Z[i] + x*X[i] +y*Y[i];
		}
	}
	Display();
}



void transformation(){
	double x,y,z;
	double angle, angleRad;
	char Axis;
```

```
int choice;
Display();
do{
	cout << "\nEnter your choice:\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 == 1)
	{
		cout << "Enter the translation factor for X, Y and Z: "; cin >> x >> y >> z;
		Trans(x,y,z);
	}
	else if (choice == 2)
	{
		cout << "Enter the scaling factor for X, Y and Z: "; cin >> x >> y >> z;
		Scale(x,y,z);
	}
	else if (choice == 3)
	{
		cout << "Enter the angle for rotation about z axis: "; cin >> angle;
		angleRad = angle * 3.1416 / 180;
		Rotate(angleRad);
	}
	else if (choice == 4)
	{
		cout << "Enter reflection axis ( X = 0 or Y = 0 or Z = 0)(Enter x,y,z): ";
cin >> Axis;
		Reflect(Axis);
	}
	else if (choice == 5)
	{
		cout << "Enter Shearing Plane about ( x or y or z): "; cin >> Axis;
		if (Axis == 'x' || Axis == 'X')
```

```
                        {
                                cout << "Enter the shearing factor Sy and Sz : "; cin >> x >> y;
                        }
                        else if(Axis == 'y' || Axis == 'Y')
                        {
                                cout << "Enter the shearing factor Sx and Sz : "; cin >> x >> y;
                        }
                        else
                        {
                                cout << "Enter the shearing factor Sx and Sy : "; cin >> x >> y;
                        }
                        Shear(Axis,x,y);
                }
        }while(choice<6&&choice>0);
}

int main(){
        transformation();
        return 0;
}
```

OUTPUT 9

```
shivank@shivank-Vostro-5568: ~/Documents/CGLAB
shivank@shivank-Vostro-5568:~/Documents/CGLAB$ ./3dtrans
        X               Y               Z
        3               4               7
        4               3               8
        5               6               3

Enter your choice:

1. Translation
2. Scaling
3. Rotation
4. Mirror Reflection
5. Shearing
6. Exit
1
Enter the translation factor for X, Y and Z: 5 0 0
        X               Y               Z
        8               4               7
        9               3               8
        10              6               3

Enter your choice:

1. Translation
2. Scaling
3. Rotation
4. Mirror Reflection
5. Shearing
6. Exit
2
Enter the scaling factor for X, Y and Z: 0.5 0.5 1
        X               Y               Z
        4               2               7
        4.5             1.5             8
        5               3               3

Enter your choice:

1. Translation
2. Scaling
3. Rotation
4. Mirror Reflection
5. Shearing
6. Exit
3
Enter the angle for rotation about z axis: 50
        X               Y               Z
        1               2               7
        2               2               8
        1               3               3
```

OUTPUT 10

```
Enter your choice:

1. Translation
2. Scaling
3. Rotation
4. Mirror Reflection
5. Shearing
6. Exit
4
Enter reflection axis ( X = 0 or Y = 0 or Z = 0)(Enter x,y,z): y
        X               Y               Z
        1               -2              7
        2               -2              8
        1               -3              3

Enter your choice:

1. Translation
2. Scaling
3. Rotation
4. Mirror Reflection
5. Shearing
6. Exit
5
Enter Shearing Plane about ( x or y or z): y
Enter the shearing factor Sx and Sz : 4 6
        X               Y               Z
        1               44              7
        2               54              8
        1               19              3

Enter your choice:

1. Translation
2. Scaling
3. Rotation
4. Mirror Reflection
5. Shearing
6. Exit
6
shivank@shivank-Vostro-5568:~/Documents/CGLAB$
```

Practical 11

AIM : To implement line Clipping in graphics

CODE

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

using namespace std;

void line(void){
	int x1,y1,x2,y2;

	cout<<"Enter the point :";
	cin>>x2>>y2;

	cout<<"Enter the point :";
	cin>>x1>>y1;

	glBegin(GL_LINE_LOOP);
		glVertex2i(-150,100);
		glVertex2i(150,100);
		glVertex2i(150,-100);
		glVertex2i(-150,-100);
	glEnd();
	glBegin(GL_LINES);
		glVertex2i(x1,y1);
		glVertex2i(x2,y2);
	glEnd();
	glFlush();

	char ch;
	cout<<"Clip line out of region :(y/n)";
	cin>>ch;
	if(ch=='y'||ch=='Y'){
		cout<<y1+(y2-y1)*(150-x1)/(x2-x1);
		if(x1>150){
			y1=y1+(y2-y1)*(150-x1)/(x2-x1);
			x1=150;
		}
```

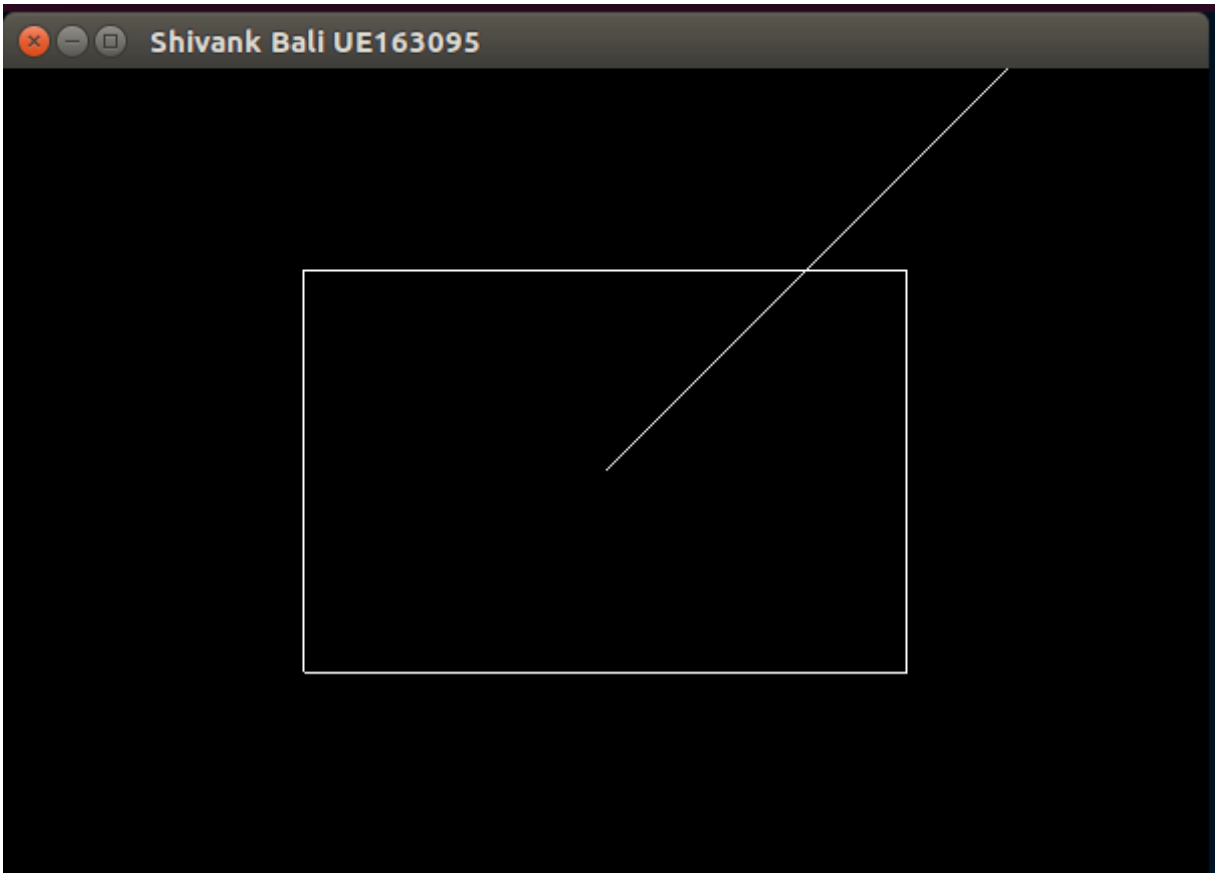
```
        if(y1>100){
                x1=x1+(x2-x1)*(100-y1)/(y2-y1);
                y1=100;
        }
        if(x1<-150){
                y1=y1+(y2-y1)*(-150-x1)/(x2-x1);
                x1=-150;
        }
        if(y1<-100){
                x1=x1+(x2-x1)*(-100-y1)/(y2-y1);
                y1=-100;
        }

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

        if(x1>150){
                y1=y1+(y2-y1)*(150-x1)/(x2-x1);
                x1=150;
        }
        if(y1>100){
                x1=x1+(x2-x1)*(100-y1)/(y2-y1);
                y1=100;
        }
        if(x1<-150){
                y1=y1+(y2-y1)*(-150-x1)/(x2-x1);
                x1=-150;
        }
        if(y1<-100){
                x1=x1+(x2-x1)*(-100-y1)/(y2-y1);
                y1=-100;
        }
}

glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_LINE_LOOP);
        glVertex2i(-150,100);
        glVertex2i(150,100);
        glVertex2i(150,-100);
        glVertex2i(-150,-100);
```

```
	glEnd();
	glBegin(GL_LINES);
		glVertex2i(x1,y1);
		glVertex2i(x2,y2);
	glEnd();

	glFlush();
}

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

	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
	glutInitWindowSize(600,400);
	glutInitWindowPosition(50,50);
	glutCreateWindow("Shivank Bali UE163095");

	gluOrtho2D(-300,300,-200,200);
	glClear(GL_COLOR_BUFFER_BIT);
	glutDisplayFunc(line);
	glutMainLoop();

	return 0;
}
```

OUTPUT

```
shivank@shivank-Vostro-5568:~/Documents/CGLAB$ ./abc
Enter the point :0 0
Enter the point :200 200
Clip line out of region :(y/n)
```

BEFORE CLIPPING



AFTER CLIPPING