RECORD OF EXPERIMENTS

Computer Graphics Lab

(CSEG3103)

Submitted By:
Harsh Narain Mathur

V Semester, B-1

Sap ID:500069098

Roll No: R100218020

Submitted To:
Mr.Arjun Arora
Assistant Professor,SOCS

School of Computer Science
UNIVERSITY OF PETROLEUM AND ENERGY STUDIES
Dehradun-248007

2020-21

INDEX

S.No.	**O**bjective of the **E**xperiment	**D**ate of **S**ubmission	**R**emarks
1.	Introduction to OpenGL and initialize a Green color		
2.	a).Drawing a line using DDA Algorithm. b) Drawing a line using Bresenham Algorithm.		
3.	a) Drawing a circle using Circle Generating Algorithm. b) Drawing a ellipse using Ellipse Generating Algorithm.		
4.	Filling the objects using Boundary fill, Flood Fill.		
5.	Performing Clipping operation on line using Cohen Sutherland		
6.	Perform Clipping Operation On Polygon Using Sutherland Hodgeman.		
7.	Perform 2d transformation on a given figure.		
8.	Perform 3d Transformation on a square		
9.	Construct a Bezier Curve		
10.	Construct the following 3d shapes: Cube And Sphere.		

EXPERIMENT-1: Introduction to OpenGL and initialize a Green color

INTRODUCTION TO OPEN GL:

- What is OpenGL?

Answer: Open Graphics Library (OpenGL) is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering.

- What is GLU/GLUT?

Answer: GLUT is the OpenGL Utility Toolkit, a window system independent toolkit for writing OpenGL programs. It implements a simple windowing application programming interface (API) for OpenGL. GLUT makes it considerably easier to learn about and explore OpenGL Programming.

What is OpenGL Architecture?

Answer: CPU-GPU Cooperation

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 (including the GLSL shader programs you will write) is the "server" and runs on the GPU. Geometry and many other types of attributes are stored in buffers called Vertex Buffer Objects (or VBOs). These buffers are allocated on the GPU and filled by your CPU program.

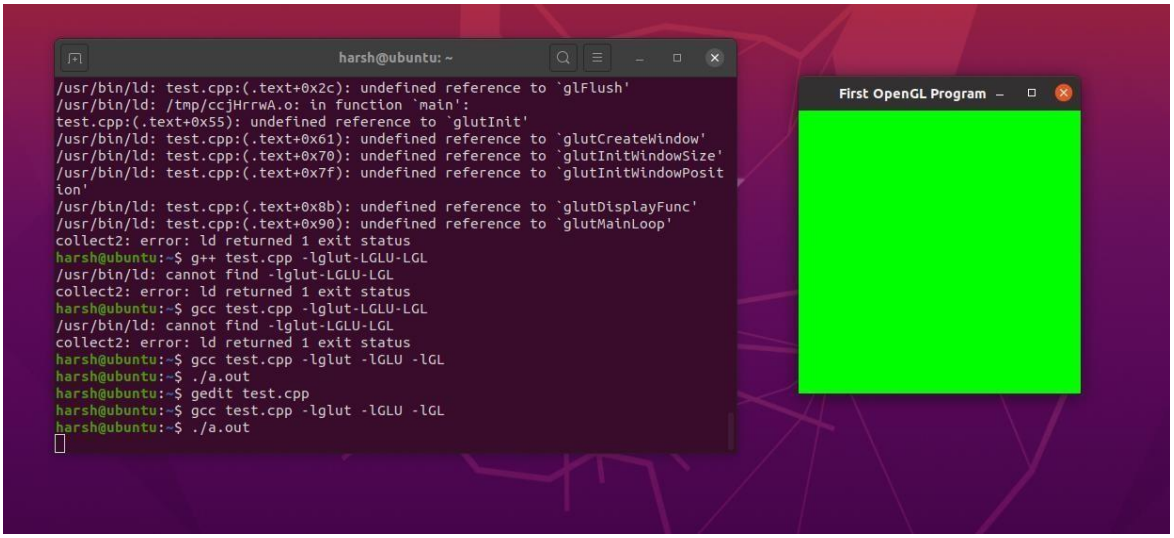
Modeling, rendering, and interaction is very much a cooperative process between the CPU client program and the GPU server programs written in GLSL.

CODE FOR INITILIZE A GREEN COLOUR:

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

void display() {
  glClearColor(0.0, 1.0,0.0,0.0); // Set background color to Green and opaque
  glClear(GL_COLOR_BUFFER_BIT);         // Clear the color buffer (background)
```

```
        glFlush(); // Render now
}
 int main(int argc, char** argv)
{
glutInit(&argc, argv);              // Initialize GLUT
  glutCreateWindow("First OpenGL Program"); // Create a window with the given title
  glutInitWindowSize(320, 320); // Set the window's initial width & height
  glutInitWindowPosition(50, 50); // Initial Position of the window
  glutDisplayFunc(display); // Register display callback handler for window re-paint
  glutMainLoop();       // Enter the event-processing loop
  return 0;
}
```

OUTPUT:



EXPERIMENT-2

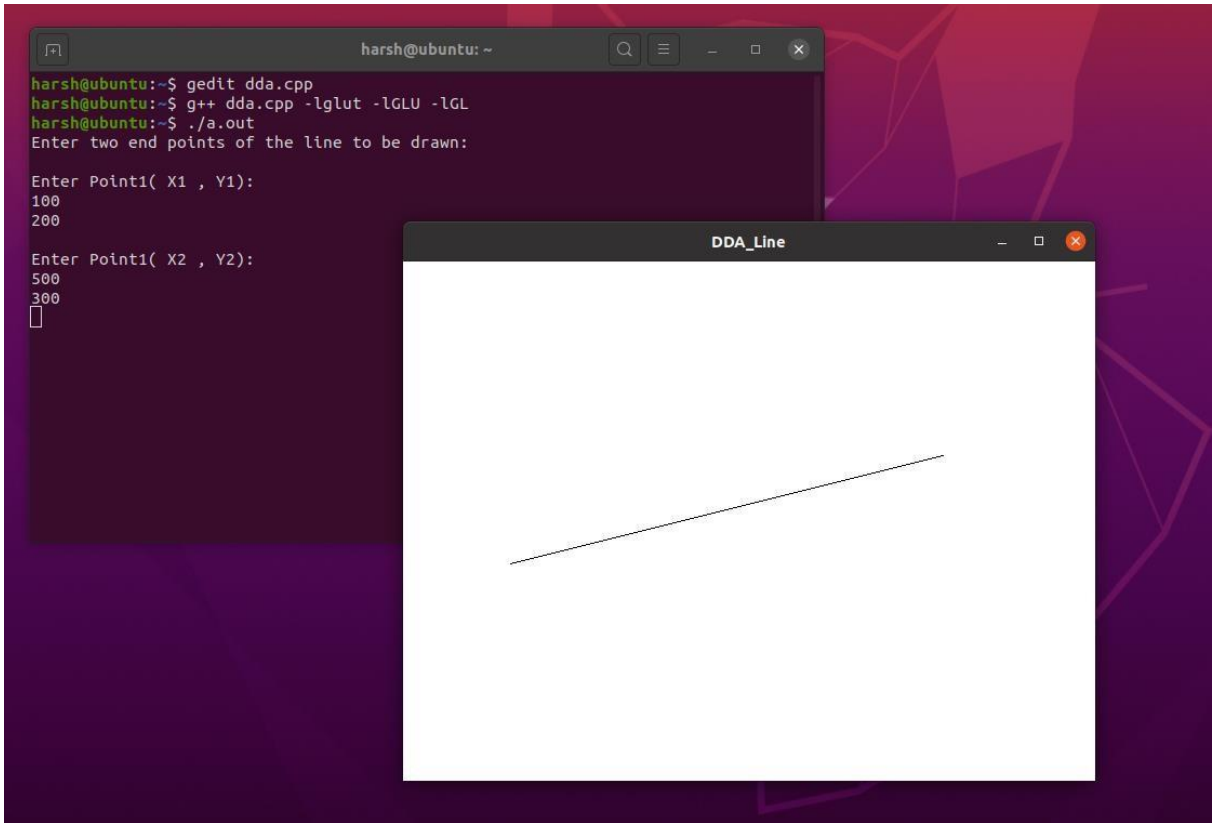
a).Drawing a Line Using DDA Algorithm

CODE:

```
#include <stdio.h>
#include<stdlib.h>
#include <math.h>
#include <GL/glut.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); glBegin(GL_POINTS); glVertex2d(x,y);
int k;

for(k=0;k<steps;k++)
{
x+=xInc; y+=yInc;

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("\nEnter Point1( X1 , Y1):\n"); scanf("%lf%lf",&X1,&Y1); 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);

glutCreateWindow("DDA_Line"); Init();
glutDisplayFunc(LineDDA);

glutMainLoop();
}
```

OUTPUT:



b) Drawing a line using Bresenham Algorithm.

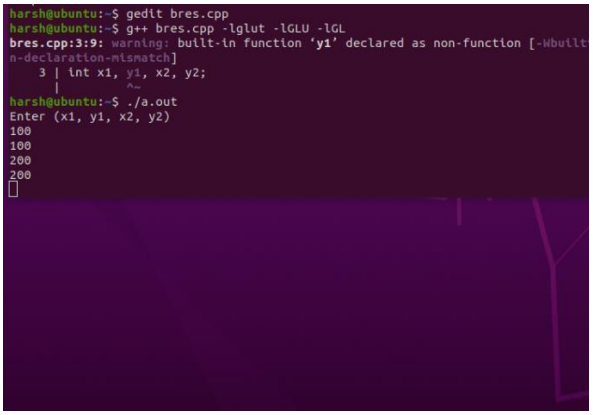
CODE:

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

            int x1, y1, x2, y2;

            void myInit() {
                 glClear(GL_COLOR_BUFFER_BIT);
                 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() {
        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,
```

```
    glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGL
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
   glutCreateWindow("Bresenham's
    myInit();
    glutDisplayFunc(myDisplay);
    glutMainLoop();
}
```

OUTPUT:



EXPERIMENT-3

a) Drawing a circle using Circle Generating Algorithm

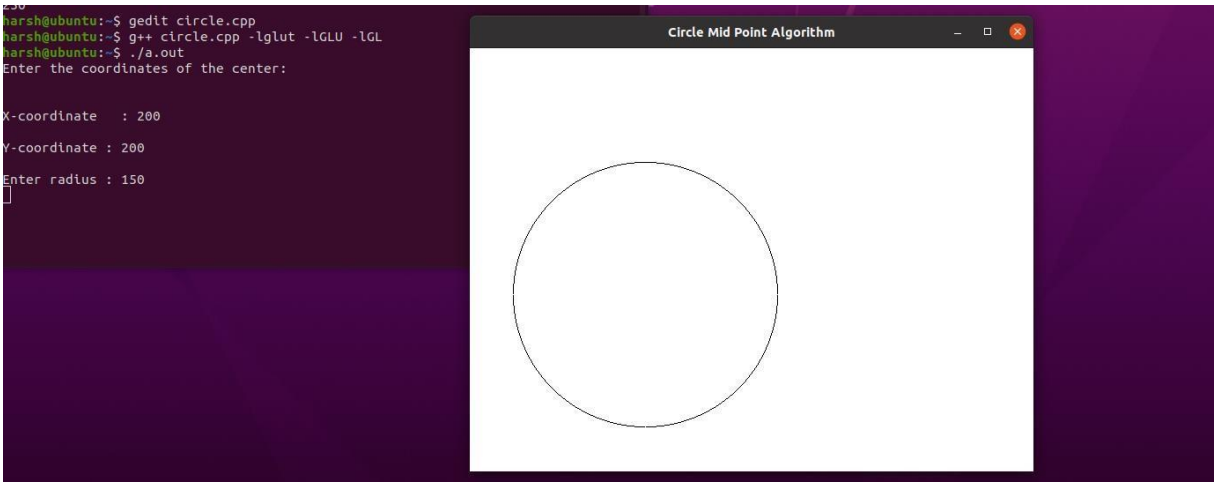
CODE:

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

int pntX1, pntY1, r;

void plot(int x, int y)
{
 glBegin(GL_POINTS);
 glVertex2i(x+pntX1, y+pntY1);
 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(0.0, 640.0, 0.0, 480.0);
}


void midPointCircleAlgo()
{
 int x = 0;
 int y = r;
 float decision = 5/4 - r;
 plot(x, y);

 while (y > x)
 {
 if (decision < 0)
 {
 x++;
 decision += 2*x+1;
 }
 else
 {
 y--;
 x++;
 decision += 2*(x-y)+1;
 }
 plot(x, y);
 plot(x, -y);
 plot(-x, y);
 plot(-x, -y);
 plot(y, x);
 plot(-y, x);
 plot(y, -x);
 plot(-y, -x);
 }

}

void myDisplay(void)
{
 glClear (GL_COLOR_BUFFER_BIT);
 glColor3f (0.0, 0.0, 0.0);
 glPointSize(1.0);

 midPointCircleAlgo();

 glFlush ();
}

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

```
 cout << "Enter the coordinates of the center:\n\n" << endl;

  cout << "X-coordinate : "; cin >> pntX1;
  cout << "\nY-coordinate : "; cin >> pntY1;
  cout << "\nEnter radius : "; cin >> r;


  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);  glutInitWindowSize (640, 480);
  glutInitWindowPosition (100, 150);
  glutCreateWindow ("Circle Mid Point Algorithm");  glutDisplayFunc(myDisplay);
  myInit ();
  glutMainLoop();

}
```

OUTPUT:



b). Drawing a ellipse using Ellipse Generating Algorithm.

CODE:

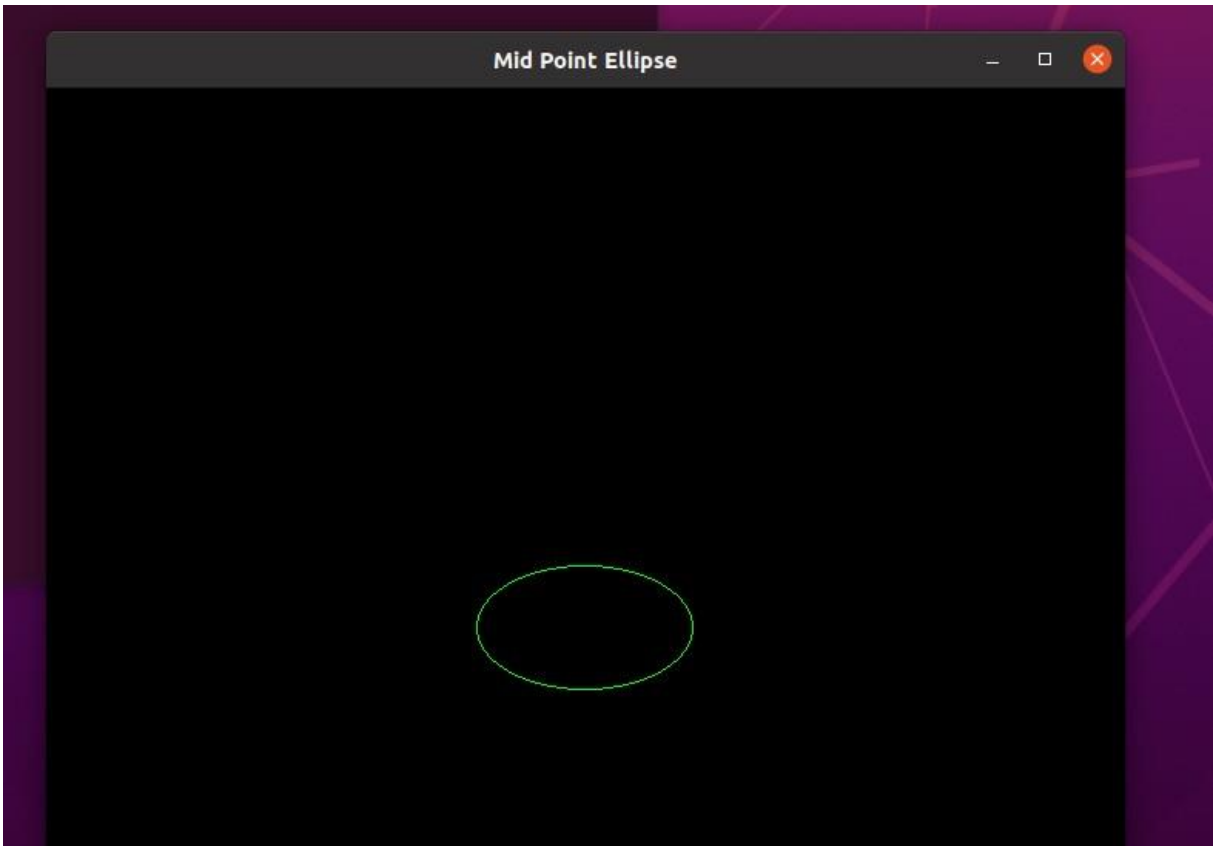
```
#include<GL/glut.h>
#include<GL/gl.h>
#include<iostream>
using namespace std;
int rx,ry;
void init()
{
glClearColor(0.0,0.0,0.0,1.0); //Blue background
glMatrixMode(GL_PROJECTION);
gluOrtho2D(0,700,0,700);
}
void display()
```

```
{
/////////////////
        glClear(GL_COLOR_BUFFER_BIT);
        int c1,c2,x,y,p1,p2,x1,y1,x2,y2;
        c1 = 0;
        x = 0;
x1=x+350;
        y = ry;
y1=y+350;

        p1 = (ry*ry) - (rx*rx)*ry + ((rx*rx)/4);
x2=700-x1;
y2=700-y1;
glColor3f(0,1,0);
glBegin(GL_POINTS);
glVertex2d(x1,y1);
glVertex2d(x1,y2);
glVertex2d(x2,y1);
glVertex2d(x2,y2);
glEnd();
glFlush();
        while((ry*ry*x)<=(rx*rx*y))
        {
                x = x + 1;
x1++;
                if(p1<0)
                {
                        //y remains same
                                p1 = p1 + (ry*ry) + 2*(ry*ry)*x;
                }
                else
                {
                        y = y-1;
                                                p1 = p1 + (ry*ry*(2*x+1)) - 2*(rx*rx)*(y);
y1--;
                }
        x2=700-x1;
y2=700-y1;
glColor3f(0,1,0);
glBegin(GL_POINTS);
glVertex2d(x1,y1);
glVertex2d(x1,y2);
glVertex2d(x2,y1);
glVertex2d(x2,y2);
```

```
glEnd();
glFlush();
        }
        // Starting Region 2
        c2 = 0;
        p2 = (ry*ry)*(x+0.5)*(x+0.5) + (rx*rx)*(y-1)*(y-1) - (rx*rx*ry*ry); x2=700-x1;
y2=700-y1;
glColor3f(0,1,0);
glBegin(GL_POINTS);
glVertex2d(x1,y1);
glVertex2d(x1,y2);
glVertex2d(x2,y1);
glVertex2d(x2,y2);
glEnd();
glFlush();
        while((y>0)&&(x<=rx))
        {
                y = y-1;
y1--;
                if(p2<0)
                {
                        x = x + 1;
x1++;
                                                        p2 = p2 + (rx*rx)*(1-2*y) + 2*(ry*ry)*x;
                }
                else
                {
                        p2 = p2 + (rx*rx)*(1-2*y);
                }
        x2=700-x1;
y2=700-y1;
glColor3f(0,1,0);
glBegin(GL_POINTS);
glVertex2d(x1,y1);
glVertex2d(x1,y2);
glVertex2d(x2,y1);
glVertex2d(x2,y2);
glEnd();
glFlush();
        }
////////////////
}
int main(int argc,char **argv)
{
```

```
        cout<<"Mid point Ellipse Algorithm"<<endl;
        cout<<"Enter rx: ";
        cin>>rx;
        cout<<"Enter ry: ";
        cin>>ry;
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(700,700);
glutCreateWindow("Mid Point Ellipse");
init();
glutDisplayFunc(display);
glutMainLoop();
}
```

OUTPUT:



EXPERIMENT-4

Filling the objects using Boundary fill, Flood Fill

CODE:

A. Boundary Fill

```
#include <math.h>
```

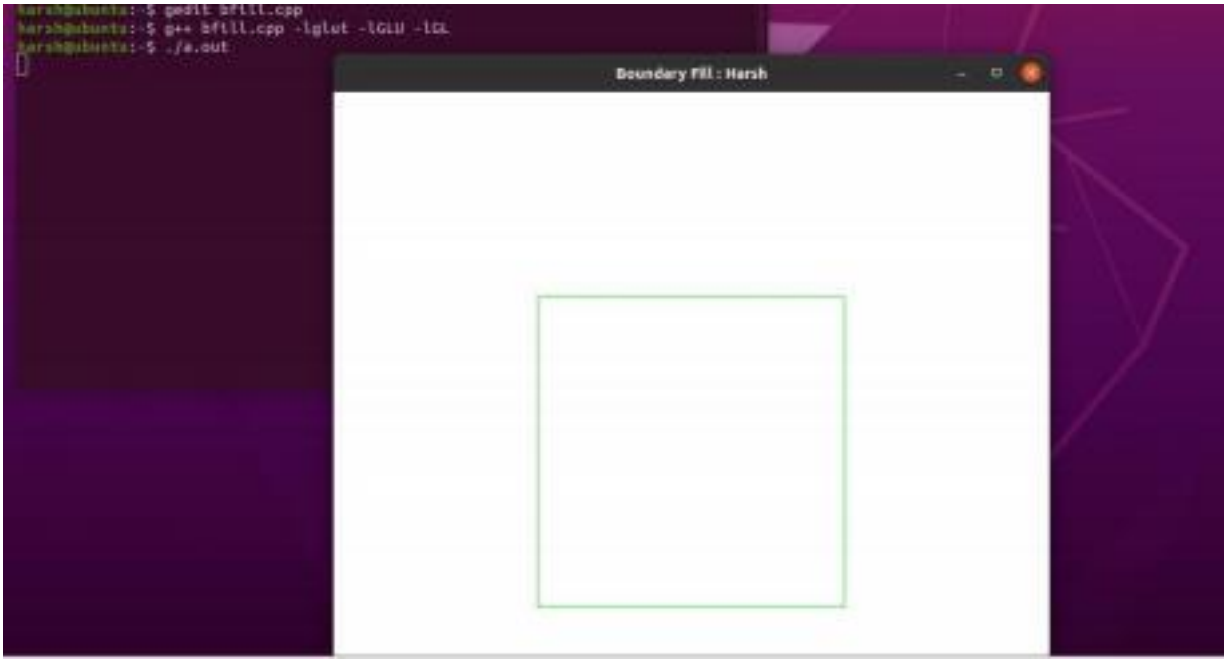
```
#include <GL/glut.h>
#include <GL/gl.h>

const int WIDTH = 700;
const int HEIGHT = 700;
//Structure 1: Point
struct Point
{
	GLint x;
	GLint y;
};

//Structure 2: Color
struct Color {
	GLfloat r;
	GLfloat g;
	GLfloat b;
};

//Function 1: Initialize the OpenGL environment
void init() {
	glClearColor(1.0, 1.0, 1.0, 0.0); // Black Color
	glColor3f(0.0, 0.0, 0.0); // White Color
	glPointSize(1.0); // Specify the point size
	glMatrixMode(GL_PROJECTION); //Transformation
	glLoadIdentity();
	gluOrtho2D(0, 700, 0, 700);
}

// Function of return type Color which is a structure defined above //
Function 2: GetPixelColor
Color getPixelColor(GLint x, GLint y)
{
	Color color; //Color Structure variable
	glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, &color); //Built in functions
	return color;
}

// Function 3: SetPixelColor
void setPixelColor(GLint x, GLint y, Color color)
{
	glColor3f(color.r, color.g, color.b);
	glBegin(GL_POINTS);
	glVertex2i(x, y);
```

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

    // Function 4: Boundary Fill
    void BoundaryFill(int x, int y, Color fillColor, Color boundaryColor){  Color
                        currentColor = getPixelColor(x, y);
        if(currentColor.r != boundaryColor.r && currentColor.g != boundaryColor.g
    &&  currentColor.b != boundaryColor.b) {
                setPixelColor(x, y, fillColor);
                BoundaryFill(x+1, y, fillColor, boundaryColor);
                BoundaryFill(x-1, y, fillColor, boundaryColor);
                BoundaryFill(x, y+1, fillColor, boundaryColor);
                BoundaryFill(x, y-1, fillColor, boundaryColor);
        }
    }

    //Function 5: Mouse Click Function
    void onMouseClick(int button, int state, int x, int y)
    {
        Color fillColor = {1.0f, 0.0f, 0.0f}; // red color will be filled  Color
        boundaryColor = {0.0f, 0.0f, 0.0f}; // black- boundary

        int halfWidth = WIDTH/2;
        if(y > halfWidth)
        {
            y -= (y-halfWidth)*2;
        }
        else
        {
            y += (halfWidth-y)*2;
        }

    BoundaryFill(x, y, fillColor, boundaryColor)
     }
    void display(void) {
    int i;

glClear(GL_COLOR_BUFFER_BIT); //Draw
     square using points //Lower point of
     square
        for(i=200;i<=500;i++)
        {
    glColor3f(0,1,0);
```

```
glBegin(GL_POINTS);
                glVertex2i(i,200);
glEnd();
        glFlush();
        }
        //right boundary
        for(i=200;i<=500;i++)
        {
glColor3f(0,1,0);
glBegin(GL_POINTS);
                glVertex2i(500,i);
glEnd();
        glFlush();
        }
        //Top boundary
        for(i=500;i>=200;i--)
        {
glColor3f(0,1,0);
glBegin(GL_POINTS);
                glVertex2i(i,500);
glEnd();
        glFlush();
        }
        //Left boundary
        for(i=500;i>=200;i--)
        {
glColor3f(0,1,0);
glBegin(GL_POINTS);
                glVertex2i(200,i);
                glEnd();
        glFlush();
        }


}
int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(HEIGHT, WIDTH);
        //glutInitWindowPosition(200, 200);
        glutCreateWindow("Boundary Fill : Harsh");
        init();
        glutDisplayFunc(display);
```

```
        glutMouseFunc(onMouseClick);
        glutMainLoop();
        return 0;
}
```

OUTPUT:



Code:

```
#include <GL/glut.h>
  int ww = 600, wh = 500;
  float bgCol[3] = {0.2, 0.4,0.0};
  float intCol[3] = {1.0,0.0,0.0};
  float fillCol[3] = {0.4,0.0,0.0};
    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 drawPolygon(int x1, int y1, int x2, int y2)
        {
         glColor3f(1.0, 0.0, 0.0);
          glBegin(GL_POLYGON);
```

```
            glVertex2i(x1, y1);
          glVertex2i(x1, y2);
          glVertex2i(x2, y2);
            glVertex2i(x2, y1);
     glEnd();
     glFlush();
     }
void display()
     {
     glClearColor(0.2, 0.4,0.0, 1.0);
     glClear(GL_COLOR_BUFFER_BIT);
     drawPolygon(150,250,200,300);
     glFlush();
     }
void floodfill4(int x,int y,float oldcolor[3],float newcolor[3])
     {
     float color[3];
     getPixel(x,y,color);
     if(color[0]==oldcolor[0] && (color[1])==oldcolor[1] && (color[2])==oldcolor[2]) {
            setPixel(x,y,newcolor);
            floodfill4(x+1,y,oldcolor,newcolor);
            floodfill4(x-1,y,oldcolor,newcolor);
          floodfill4(x,y+1,oldcolor,newcolor);
          floodfill4(x,y-1,oldcolor,newcolor);
      }
     }
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);
            floodfill4(xi,yi,intCol,fillCol);
      }
     }
void myinit()
     {
     glViewport(0,0,ww,wh);
     glMatrixMode(GL_PROJECTION);
     glLoadIdentity();
     gluOrtho2D(0.0,(GLdouble)ww,0.0,(GLdouble)wh);
     glMatrixMode(GL_MODELVIEW);
     }
int main(int argc, char** argv)
```

```
{
glutInit(&argc,argv);
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(ww,wh);
glutCreateWindow("Flood-Fill-Recursive");
glutDisplayFunc(display);
myinit();
glutMouseFunc(mouse);
glutMainLoop();
return 0;
}
```

OUTPUT :

EXPERIMENT-5

PERFORMING CLIPPING OPERATION ON LINE USING COHEN SUTHERLAND

CODE:

```
#include<stdio.h>
#include<GL/glut.h>
#define outcode int
double xmin=50,ymin=50,xmax=100,ymax=100;//
Windows  boundaries
double xvmin=200,yvmin =200, xvmax=300,yvmax=300; //
Viewport  boundaries
const int RIGHT= 8;
```

```
const int LEFT =2;
const int TOP=4;
const int BOTTOM=1;
outcode ComputeOutCode(double x,double y);
void CohenSutherlandLineClipAnddraw(double x0,double y0,double  x1,double y1)
{
   outcode outcode0,outcode1,outcodeOut;
int accept =0,done =0;
outcode0= ComputeOutCode(x0,y0);
outcode1= ComputeOutCode(x1,y1);
do
{
   if(!(outcode0|outcode1))
   { accept=1;
     done=1;
   }
   else
      if(outcode0 & outcode1)
          done=1;
      else
      {
         double x,y;
          outcodeOut=
          outcode0?outcode0:outcode1;  if(outcodeOut & TOP)
  {
x= x0+(x1-x0)*(ymax-y0)/(y1-y0);
             y=ymax;
          }
```

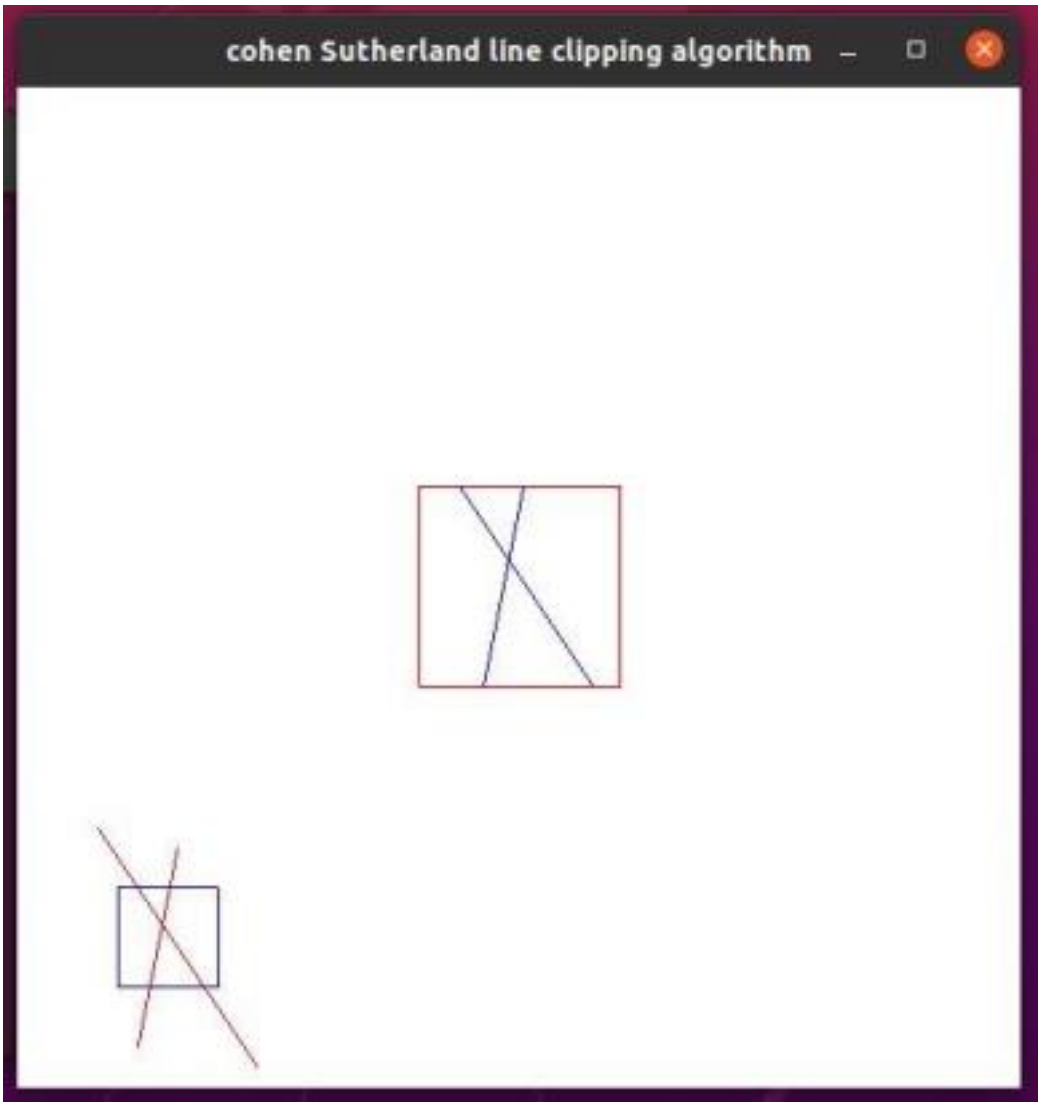
```
        else
if(outcodeOut & BOTTOM)
          {
x= x0+(x1-x0)*(ymin-y0)/(y1-y0);
            y=ymin;
          }
          else
if(outcodeOut & RIGHT)
          {
y= y0+(y1-y0)*(xmax-x0)/(x1-x0);
            x=xmax;
       }
          else
          {
            y= y0+(y1-y0)*(xmin-x0)/(x1-x0);  x=xmin;
          }

     if(outcodeOut == outcode0)
          { x0=x;
           y0=y;
           outcode0 = ComputeOutCode(x0,y0);

          }
       else
          {
           x1=x;
           y1=y;
          outcode1 = ComputeOutCode(x1,y1); }
       }
```

```
}
while(!done);
if(accept)
{
   double sx=(xvmax-xvmin)/(xmax-xmin);  double
   sy=(yvmax-yvmin)/(ymax-ymin);  double vx0 =
   xvmin+(x0-xmin)*sx;  double vy0 = yvmin+(y0-
   ymin)*sy;  double vx1 = xvmin+(x1-
   xmin)*sx;  double vy1 = yvmin+(y1-
   ymin)*sy;  glColor3f(1.0,0.0,0.0);
   glBegin(GL_LINE_LOOP);
   glVertex2f(xvmin,yvmin);
   glVertex2f(xvmax,yvmin);
   glVertex2f(xvmax,yvmax);
   glVertex2f(xvmin,yvmax);
  glEnd();
glColor3f(0.0,0.0,1.0);
glBegin(GL_LINES);
glVertex2d(vx0,vy0);
glVertex2d(vx1,vy1);
glEnd();
}
}
outcode ComputeOutCode(double x,double y) {
   outcode code =0;
    if(y>ymax)
        code |=TOP;
   if(y<ymin)
        code |=BOTTOM;
   if(x>xmax)
        code |=RIGHT;
```

```
    if(x<xmin)
        code |=LEFT;
        return code;
}
void display()
{
    double
    x0=120,y0=10,x1=40,y1=130;  glClear(GL_COL
    OR_BUFFER_BIT);  glColor3f(1.0,0.0,0.0);
    glBegin(GL_LINES);
         glVertex2d(x0,y0);
         glVertex2d(x1,y1);
         glVertex2d(60,20);
         glVertex2d(80,120);
         glEnd();
   glColor3f(0.0,0.0,1.0);
      glBegin(GL_LINE_LOOP);
         glVertex2f(xmin,ymin);
         glVertex2f(xmax,ymin);
         glVertex2f(xmax,ymax);
         glVertex2f(xmin,ymax);
         glEnd();
       CohenSutherlandLineClipAnddraw(x0,y0,x1,y1);
       CohenSutherlandLineClipAnddraw(60,20,80,120);  glFlush();
}
void myinit()
{
    glClearColor(1.0,1.0,1.0,1.0);
    glColor3f(1.0,0.0,0.0);
    glPointSize(1.0);
```

```
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0,499.0,0.0,499.0);
}
int main(int argc, char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);  glutInitWi
    ndowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("cohen Sutherland line clipping
    algorithm");  glutDisplayFunc(display);
    myinit();
    glutMainLoop();
    return 0;
}
```

OUTPUT:



EXPERIMENT-6

PERFORMING CLIPPING OPERATION ON POLYGON USING SUTHERLAND HODGEMAN.

CODE:

```
#include<iostream>
#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 intersectipn 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 intersectipn 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++;
```

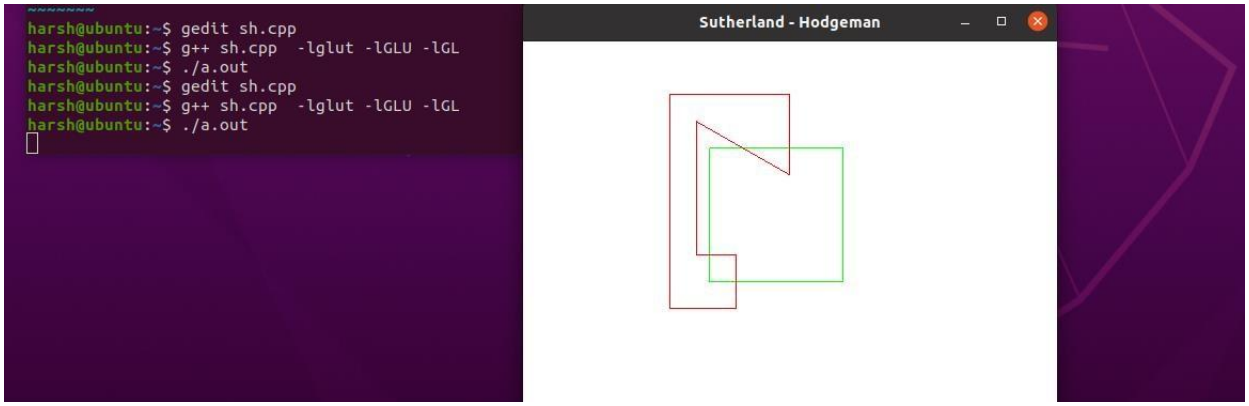
```
}
// Case 2: When only first point is outside
else if (i_pos>= 0 &&k_pos< 0)
{
// Point of intersection with edge
// and the second point is added
new_points[new_poly_size][0] = x_intersect(x1,y1, x2, y2, ix, iy, kx, ky); new_points[new_poly_size][1] = y_intersect(x1,y1, x2, y2, ix, iy, kx, ky); 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++)
{
```

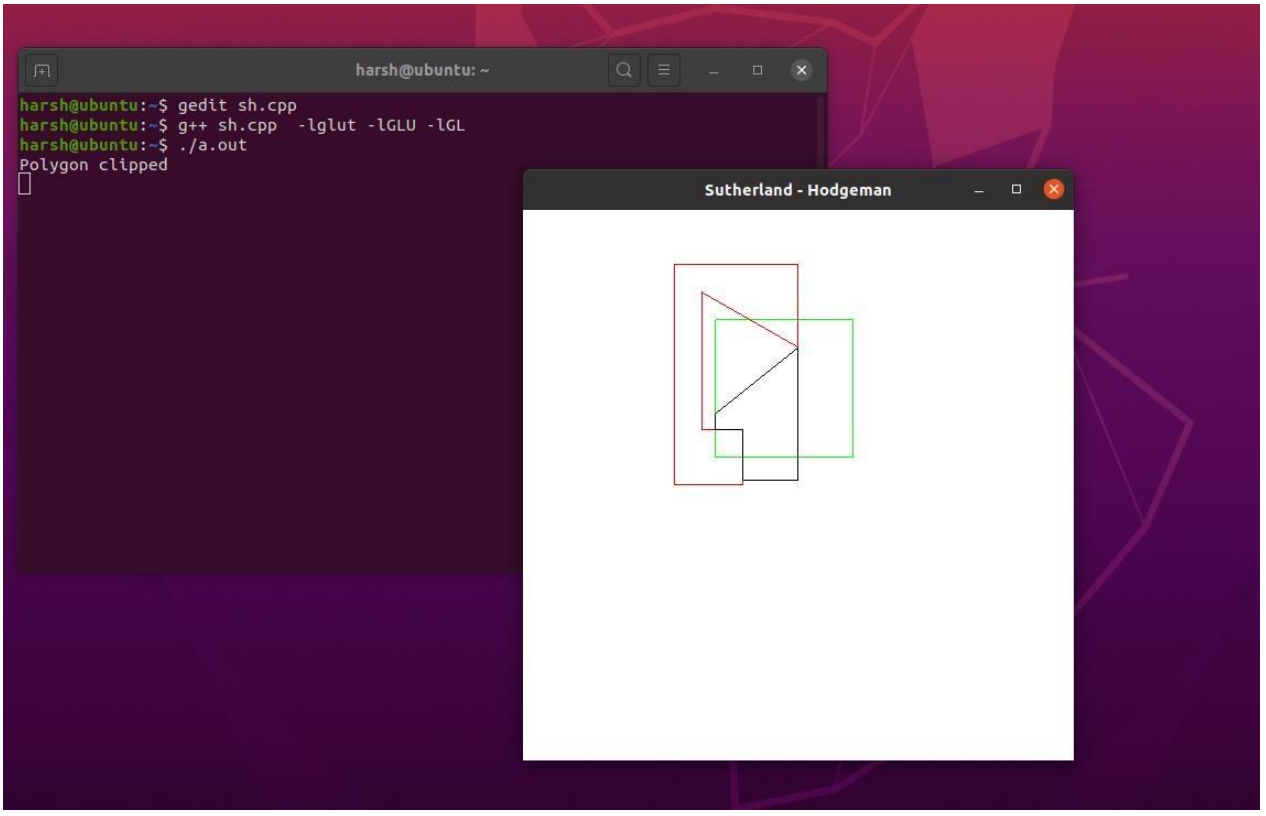
```
int k = (i+1) % clipper_size;
// We pass the current array of vertices, it's size
// and the end points of the selected clipper line
clip(poly_points, poly_size, clipper_points[i][0],
clipper_points[i][1], clipper_points[k][0],
clipper_points[k][1]);
}
// Printing vertices of clipped polygon
for (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 mouse(int button, int action, int x , int y)
{
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++;
printf("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");
```

```
glutDisplayFunc(display);
glutMouseFunc(mouse);
init();
glutMainLoop();
return 0;
}
```

OUTPUT:





POLYGON CLIPPED

EXPERIMENT-7

PERFORM 2D TRANSFORMATION ON A SQUARE

CODE:

```
#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]);
```

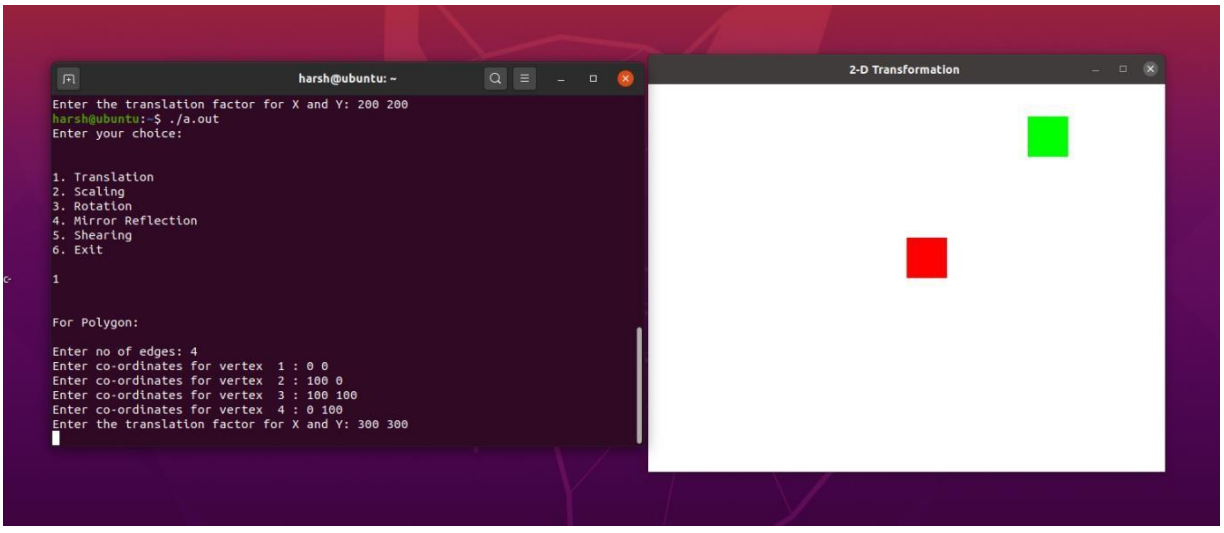
```
            glVertex2i(pntX[1] + shearingX, pntY[1]);
            glVertex2i(pntX[2] + shearingX, pntY[2]);

            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);
        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);

        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\n" << endl;

        cin >> choice;

        if (choice == 6) {
            return choice;
        }

        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)
        {
```

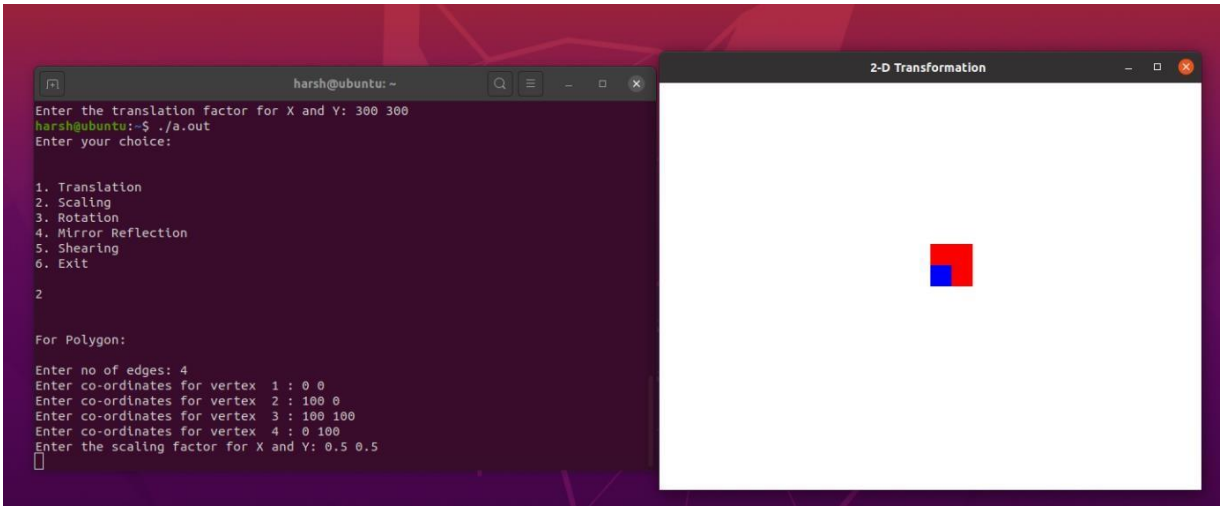
```
          cout << "Enter reflection axis ( x or y ): "; cin
>>  reflectionAxis;
     }
     else if (choice == 5)
     {
          cout << "Enter reflection axis ( x or y ): "; cin
>>  shearingAxis;
           if (shearingAxis == 'x' || shearingAxis == 'X')
          {
                cout << "Enter the shearing factor for X: ";
cin >> shearingX;
          }
          else
          {
                cout << "Enter the shearing factor for Y: ";
cin >> shearingY;
          }
     }
     //cout << "\n\nPoints:" << pntX[0] << ", " << pntY[0]
<<  endl;
     //cout << angleRad;


     glutInit(&argc, argv);
     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
     glutInitWindowSize(640, 480);
     glutInitWindowPosition(100, 150);
     glutCreateWindow("2-D Transformation ");
     glutDisplayFunc(myDisplay);
     myInit();
     glutMainLoop();

}
```

- Output Are As Follows : -

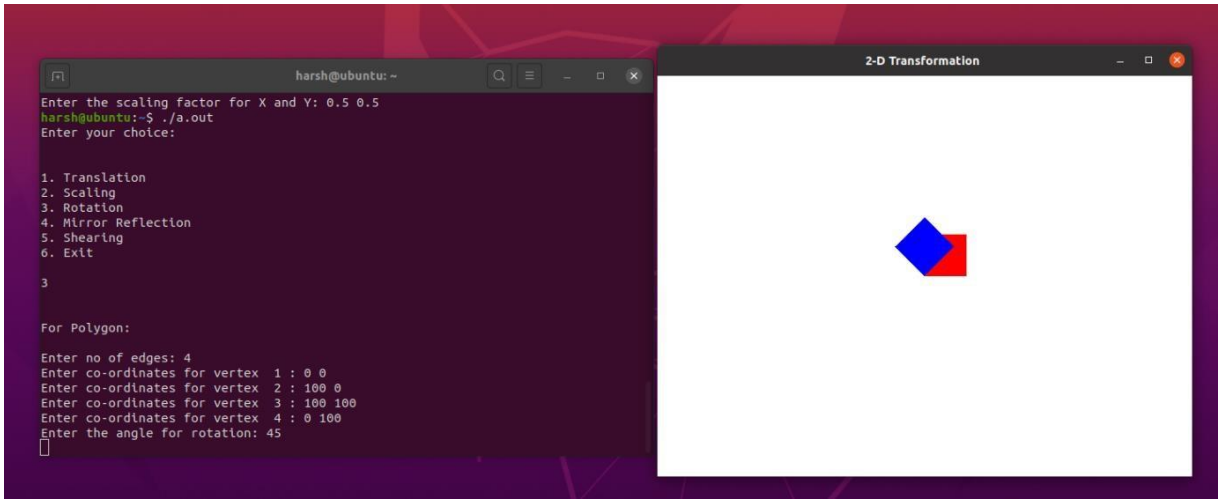
1.)Translation :



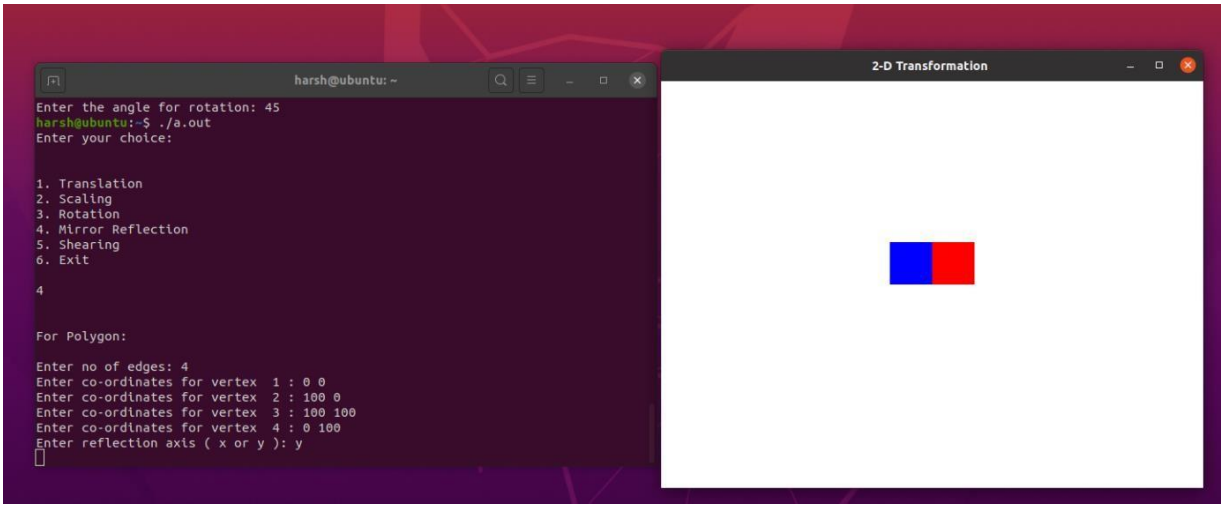
2.)Scaling



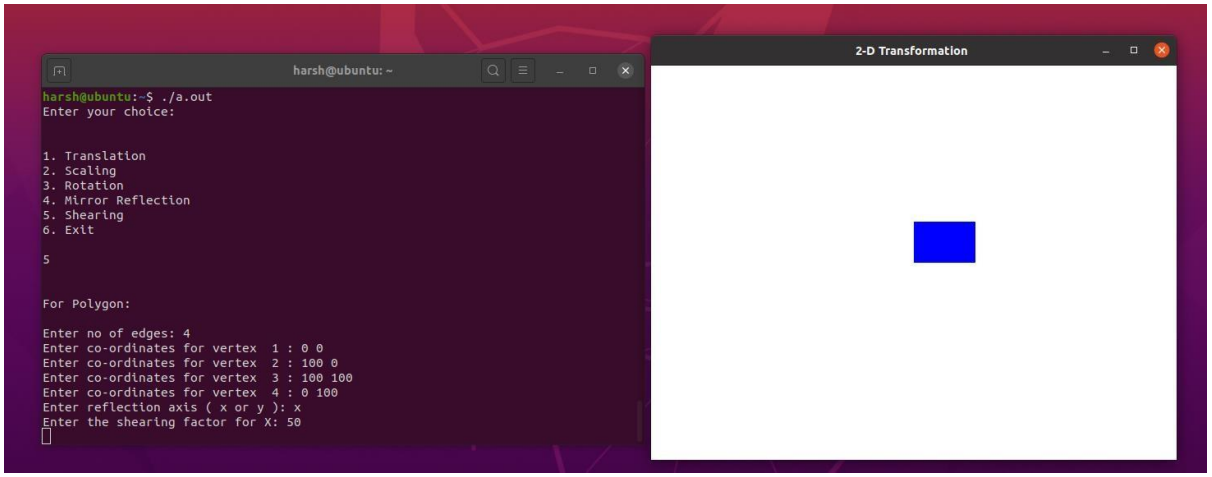
3.)Rotation



4.)Reflection



5.)Shearing



EXPERIMENT-8

PERFORM 3D TRANSFORMATION ON A SQUARE

CODE:

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

typedef float Matrix4x4 [4][4]; Matrix4x4
```

```
theMatrix;

float ptsIni[8][3]={{80,80,-100},{180,80,-100},{180,180,-
100},{80,180,100},{60,60,0},{160,60,0},{160,160,0},{60,160,0}};

//Realign above line while execution

// Initial Co-ordinates of the Cube to be Transformed float

ptsFin[8][3]; float refptX,refptY,refptZ; //Reference points  float
TransDistX,TransDistY,TransDistZ; //Translations along Axes  float
ScaleX,ScaleY,ScaleZ; //Scaling Factors along Axes
float  Alpha,Beta,Gamma,Theta; //Rotation angles about Axes  float
A,B,C; //Arbitrary Line Attributes float
aa,bb,cc; //Arbitrary Line Attributes

float x1,y11,z1,x2,y2,z2; int choice,choiceRot,choiceRef; void

matrixSetIdentity(Matrix4x4 m) // Initialises the matrix as Unit Matrix  { int
i,
j; for

(i=0;

i<4;

i++)
for

(j=0;

j<4;

j++)

m[i][j]

= (i ==

j);

}


void matrixPreMultiply(Matrix4x4 a , Matrix4x4 b)

{// Multiplies matrix a times b, putting result in b

 int i,j;

 Matrix4x4 tmp;

 for (i = 0; i < 4; i++)

for (j = 0; j < 4; j++)

 tmp[i][j]=a[i][0]*b[0][j]+a[i][1]*b[1][j]+a[i][2]*b[2][j]+a[i][3]*b[3][j];
 for (i = 0; i < 4; i++) for (j
```

```
= 0; j < 4; j++)
theMatrix[i][j] = tmp[i][j];
}
void Translate(int tx, int ty, int tz)
{
 Matrix4x4 m;
matrixSetIdentity(m);
m[0][3] = tx; m[1][3]
= ty; m[2][3] = tz;
 matrixPreMultiply(m, theMatrix);
}
void Scale(float sx , float sy ,float sz)
{
 Matrix4x4 m;
matrixSetIdentity(m); m[0][0] =
sx;  m[0][3] = (1 - sx)*refptX; m[1][1] =  sy;
m[1][3] = (1 - sy)*refptY; m[2][2]  = sz;
m[2][3] = (1 -
sy)*refptZ;  matrixPreMultiply(m ,
theMatrix);  }
void RotateX(float angle)
{
 Matrix4x4 m;
matrixSetIdentity(m); angle
=  angle*22/1260; m[1][1] =
cos(angle); m[1][2] = -sin(angle);  m[2][1]
= sin(angle); m[2][2] =  cos(angle);
matrixPreMultiply(m ,  theMatrix);
}
void RotateY(float angle)
{
 Matrix4x4 m;
matrixSetIdentity(m); angle
=  angle*22/1260; m[0][0] =
cos(angle); m[0][2] = sin(angle);  m[2][0]
= -sin(angle); m[2][2] =  cos(angle);
matrixPreMultiply(m ,  theMatrix); }
```

```
void RotateZ(float angle)
{
 Matrix4x4 m;
matrixSetIdentity(m); angle =
angle*22/1260; m[0][0] =
cos(angle); m[0][1] = -sin(angle);
m[1][0] = sin(angle); m[1][1] =
cos(angle); matrixPreMultiply(m ,
theMatrix);
}
void Reflect(void)
{
 Matrix4x4 m;
matrixSetIdentity(m);
switch(choiceRef)
 {
 case 1: m[2][2] = -1;
break; case 2:
m[0][0] = -1; break;
case 3: m[1][1] = -1;
break;
 }
 matrixPreMultiply(m , theMatrix);
}
void DrawRotLine(void)
{
switch(choiceRot)
{
 case 1: glBegin(GL_LINES); glVertex3s(-1000 ,B,C);
glVertex3s( 1000 ,B,C); glEnd(); break; case 2:
glBegin(GL_LINES); glVertex3s(A ,-1000
,C); glVertex3s(A ,1000 ,C); glEnd(); break;
case 3: glBegin(GL_LINES); glVertex3s(A ,B ,-
```

```
1000); glVertex3s(A ,B ,1000); glEnd(); break;
case 4: glBegin(GL_LINES); glVertex3s(x1-
aa*500 ,y11-bb*500 , z1-cc*500);
glVertex3s(x2+aa*500 ,y2+bb*500 , z2+cc*500);
 glEnd();
break;
 }
}
void TransformPoints(void)
{ int
 i,k;
float tmp ; for(k=0
; k<8 ; k++) for (i=0
; i<3 ; i++)

ptsFin[k][i] = theMatrix[i][0]*ptsIni[k][0] + theMatrix[i][1]*ptsIni[k][1]
+  theMatrix[i][2]*ptsIni[k][2] + theMatrix[i][3];
// Realign above line while execution
}
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]) //Display the Figure  { int
i;
 glColor3f (0.7, 0.4, 0.7);
 glBegin(GL_POLYGON);
 glVertex3f(a[0][0],a[0][1],a[0][2]);
```

```
glVertex3f(a[1][0],a[1][1],a[1][2]);

glVertex3f(a[2][0],a[2][1],a[2][2]);

glVertex3f(a[3][0],a[3][1],a[3][2]);

glEnd(); i=0; glColor3f (0.8, 0.6,

0.5);

 glBegin(GL_POLYGON);

 glVertex3s(a[0+i][0],a[0+i][1],a[0+i][2]); glVe
rtex3s(a[1+i][0],a[1+i][1],a[1+i][2]); glVertex
3s(a[5+i][0],a[5+i][1],a[5+i][2]); glVertex3s(a[
4+i][0],a[4+i][1],a[4+i][2]); glEnd(); glColor3f
(0.2, 0.4, 0.7);
 glBegin(GL_POLYGON);

 glVertex3f(a[0][0],a[0][1],a[0][2]);

glVertex3f(a[3][0],a[3][1],a[3][2]);

glVertex3f(a[7][0],a[7][1],a[7][2]);

glVertex3f(a[4][0],a[4][1],a[4][2]);

glEnd(); i=1; glColor3f (0.5, 0.4,

0.3);

 glBegin(GL_POLYGON);

 glVertex3s(a[0+i][0],a[0+i][1],a[0+i][2]); glVe
rtex3s(a[1+i][0],a[1+i][1],a[1+i][2]); glVertex
3s(a[5+i][0],a[5+i][1],a[5+i][2]);
glVertex3s(a[4+i][0],a[4+i][1],a[4+i][2]);

glEnd(); i=2; glColor3f (0.5, 0.6, 0.2);

 glBegin(GL_POLYGON);

 glVertex3s(a[0+i][0],a[0+i][1],a[0+i][2]);

 glVertex3s(a[1+i][0],a[1+i][1],a[1+i][2]);

 glVertex3s(a[5+i][0],a[5+i][1],a[5+i][2]);

 glVertex3s(a[4+i][0],a[4+i][1],a[4+i][2]);

 glEnd(); i=4;

glColor3f (0.7, 0.3, 0.4);

glBegin(GL_POLYGON);

 glVertex3f(a[0+i][0],a[0+i][1],a[0+i][2]);

glVertex3f(a[1+i][0],a[1+i][1],a[1+i][2]);
```

```
glVertex3f(a[2+i][0],a[2+i][1],a[2+i][2]);

glVertex3f(a[3+i][0],a[3+i][1],a[3+i][2]); glEnd();  }


void display(void)

{

 glClear (GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT);  Axes(); glColor3f (1.0, 0.0, 0.0); //
Set the  color to RED Draw(ptsIni);
matrixSetIdentity(theMatrix); switch(choice)  {
 case 1: Translate(TransDistX , TransDistY
,TransDistZ);  break; case 2: Scale(ScaleX, ScaleY, ScaleZ);
break;  case 3: switch(choiceRot)
 {

case 1: DrawRotLine(); Translate(0,-B,-C);
RotateX(Alpha);

Translate(0,B,C); break;


case 2: DrawRotLine();

Translate(-A,0,-C);

 RotateY(Beta);

Translate(A,0,C);

break; case 3:

DrawRotLine();

Translate(-A,-B,0);

 RotateZ(Gamma); Translate(A,B,0); break; case 4: DrawRotLine();  float
MOD =sqrt((x2-x1)*(x2-x1) + (y2-y11)*(y2-y11) + (z2-z1)*(z2-z1));  aa =
(x2-x1)/MOD; bb = (y2-y11)/MOD; cc = (z2-z1)/MOD; Translate(- x1,-y11,-
z1); float ThetaDash;
 ThetaDash = 1260*atan(bb/cc)/22;

 RotateX(ThetaDash);

 RotateY(1260*asin(-aa)/22);

 RotateZ(Theta);

 RotateY(1260*asin(aa)/22);

 RotateX(-ThetaDash);

 Translate(x1,y11,z1); break;

}
```

```
break;
case 4: Reflect();
break;
}
TransformPoints(); Draw(ptsFin);
glFlush();
}
void init(void)
 {
 glClearColor (1.0, 1.0, 1.0, 1.0); // Set the
Background color to WHITE glOrtho(-454.0, 454.0,
-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
 }
int main (int argc, char *argv)
{
 glutInit(&argc, &argv);
 glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowSize (1362, 750); glutInitWindowPosition (0, 0);
glutCreateWindow (" Basic Transformations ");
 init ();
 printf("Enter your choice
number:\n1.Translation\n2.Scaling\n3.Rotation\n4.Reflection\n=>");  sca
nf("%d",&choice); switch(choice)
 {
 case 1:printf("Enter Translation along X, Y & Z\n=>");
scanf("%f%f%f",&TransDistX , &TransDistY , &TransDistZ); break;

case 2:printf("Enter Scaling ratios along X, Y & Z\n=>");
scanf("%f%f%f",&ScaleX , &ScaleY , &ScaleZ); break;
case 3:printf("Enter your choice for Rotation about axis:\n1.parallel to X-axis.(y=B
&  z=C)\n2.parallel to Y-axis.(x=A & z=C)\n3.parallel to Z-axis.(x=A & y=B)\n4.Arbitrary
line
```

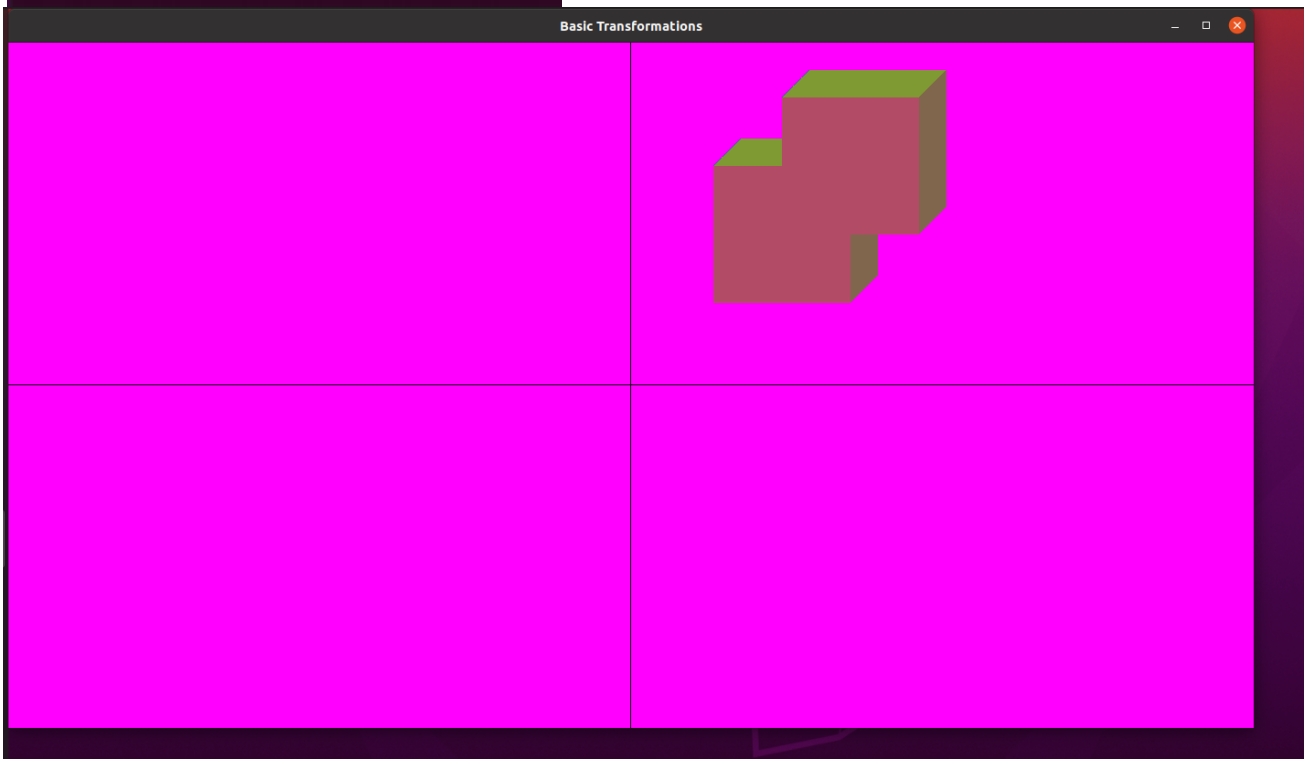
```
passing through (x1,y1,z1) & (x2,y2,z2)\n =>"); //Realign above line while
execution  scanf("%d",&choiceRot); switch(choiceRot)
{
 case 1: printf("Enter B & C: ");
scanf("%f %f",&B,&C); printf("Enter
Rot. Angle Alpha: ");
scanf("%f",&Alpha); break; case 2:
printf("Enter A & C: "); scanf("%f
%f",&A,&C); printf("Enter Rot.
Angle Beta: "); scanf("%f",&Beta);
break; case 3: printf("Enter A & B:
"); scanf("%f %f",&A,&B);
printf("Enter Rot. Angle Gamma: ");
scanf("%f",&Gamma);
 break; case 4: printf("Enter values of x1 ,y1 &
z1:\n"); scanf("%f %f %f",&x1,&y11,&z1);
printf("Enter values of x2 ,y2 & z2:\n");
scanf("%f %f %f",&x2,&y2,&z2); printf("Enter
Rot. Angle Theta: "); scanf("%f",&Theta);
break;
}
break;
 case 4: printf("Enter your choice for reflection about plane:\n1.X-Y\n2.Y
Z\n3.XZ\n=>"); scanf("%d",&choiceRef); break; default: printf("Please enter a  valid
choice!!!\n"); return 0;
 }
 glutDisplayFunc(display);
glutMainLoop();
 return 0;
}
```

OUTPUT:

Output are as Follows ; -

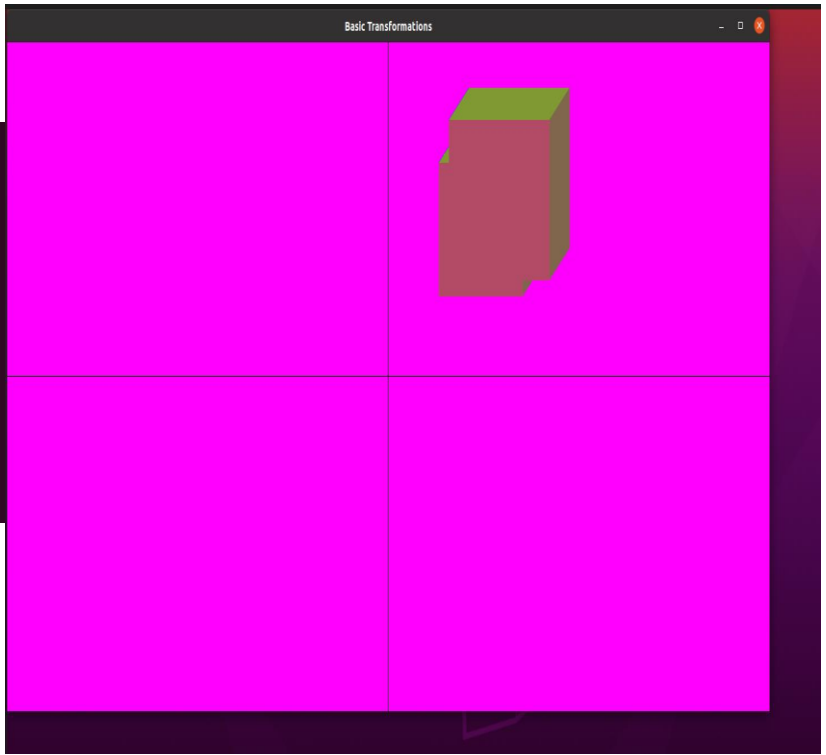
1.) Translation

```
harsh@ubuntu:~$ ./a.out
Enter your choice number:
1.Translation
2.Scaling
3.Rotation
4.Reflection
=>1
Enter Translation along X, Y & Z
=>50
50
50
harsh@ubuntu:~$
```



2.) Scaling

```
harsh@ubuntu:~$ ./a.out
Enter your choice number:
1.Translation
2.Scaling
3.Rotation
4.Reflection
=>2
Enter Scaling ratios along X, Y & Z
=>1.2
1.2
1.2
```

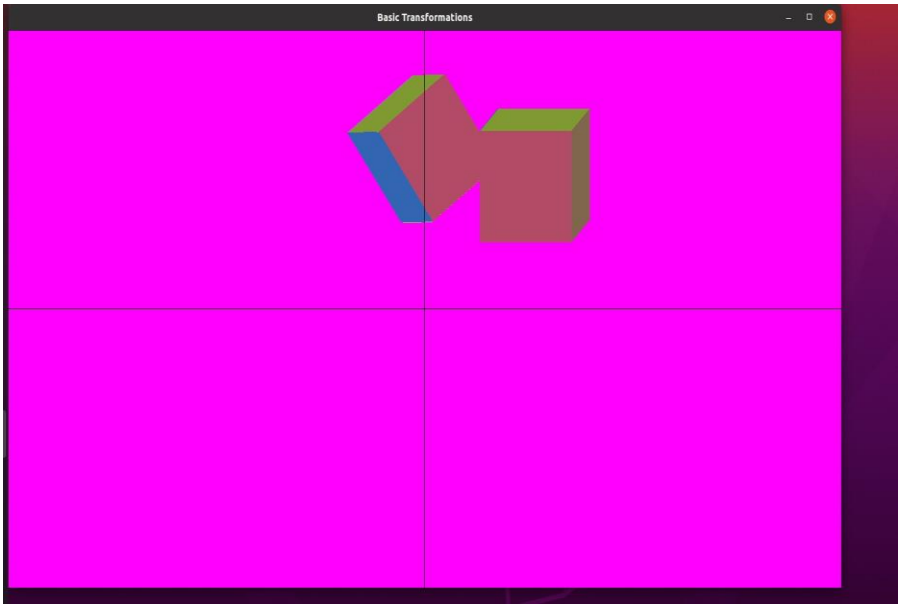


3.) Rotation

```
harsh@ubuntu:~$ ./a.out
Enter your choice number:
1.Translation
2.Scaling
3.Rotation
4.Reflection
=>3
Enter your choice for Rotation about axis:
1.parallel to X-axis.(y=B & z=C)
2.parallel to Y-axis.(x=A & z=C)
3.parallel to         Z-axis.(x=A & y=B)
4.Arbitrary line passing through (x1,y1,z1) &         (x2,y2,z2)
 =>4
Enter values of x1 ,y1 & z1:
2 2 2
Enter values of x2 ,y2 & z2:
1 5 6
Enter Rot. Angle Theta: 45
```

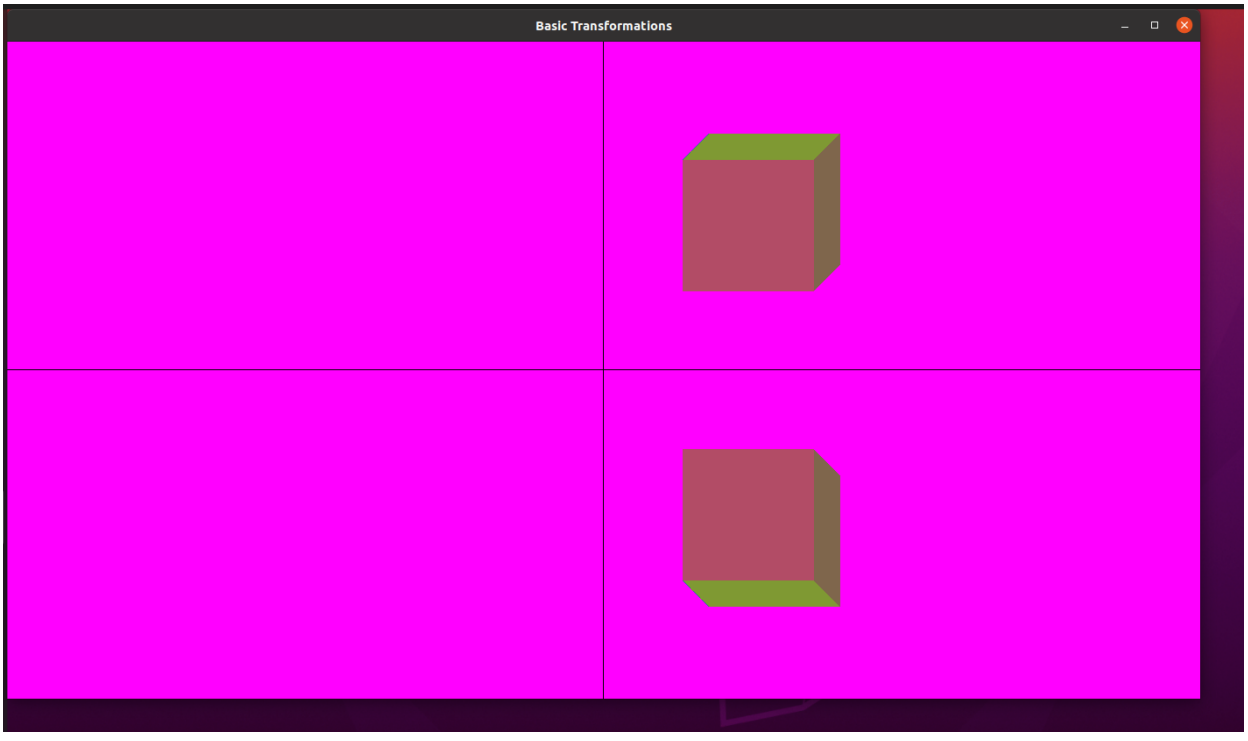
3.) Rotation


```
harsh@ubuntu:~$ ./a.out
Enter your choice number:
1.Translation
2.Scaling
3.Rotation
4.Reflection
=>3
Enter your choice for Rotation about axis:
1.parallel to X-axis.(y=B & z=C)
2.parallel to Y-axis.(x=A & z=C)
3.parallel to         Z-axis.(x=A & y=B)
4.Arbitrary line passing through (x1,y1,z1) &         (x2,y2,z2)
 =>4
Enter values of x1 ,y1 & z1:
2 2 2
Enter values of x2 ,y2 & z2:
1 5 6
Enter Rot. Angle Theta: 45
```

4) Reflection

```
harsh@ubuntu:~$ ./a.out
Enter your choice number:
1.Translation
2.Scaling
3.Rotation
4.Reflection
=>4
Enter your choice for reflection about plane:
1.X-Y
2.Y-Z
3.X-Z
=>3
```



EXPERIMENT-9

CONSTRUCT A BEIZER CURVE

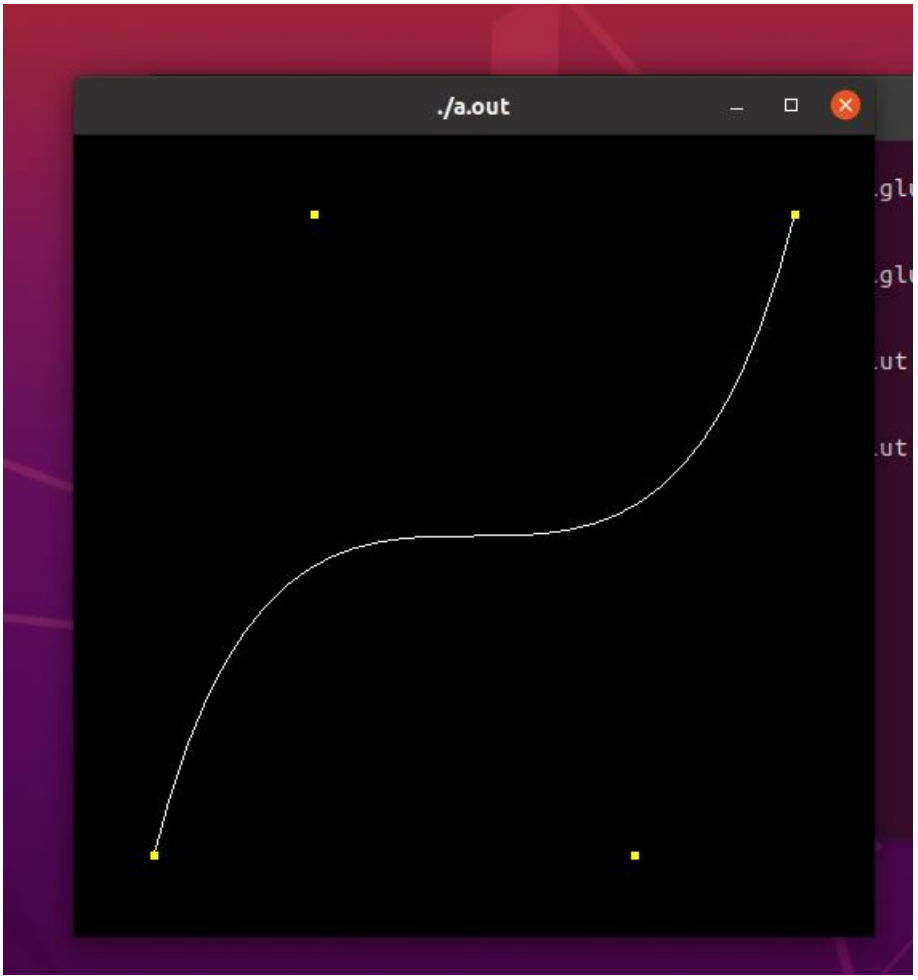
CODE:

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

GLfloat ctrlpoints[4][3] = {
```

```
        { -4.0, -4.0, 0.0}, { -2.0, 4.0, 0.0},
        {2.0, -4.0, 0.0}, {4.0, 4.0, 0.0}};

void init(void)
{
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glShadeModel(GL_FLAT);
   glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, &ctrlpoints[0][0]);
   glEnable(GL_MAP1_VERTEX_3);
}

void display(void)
{
   int i;

   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 1.0, 1.0);
   glBegin(GL_LINE_STRIP);
      for (i = 0; i <= 30; i++)
         glEvalCoord1f((GLfloat) i/30.0);
   glEnd();
   /* The following code displays the control points as dots. */
   glPointSize(5.0);
   glColor3f(1.0, 1.0, 0.0);
   glBegin(GL_POINTS);
      for (i = 0; i < 4; i++)
         glVertex3fv(&ctrlpoints[i][0]);
   glEnd();
   glFlush();
}

void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if (w <= h)
      glOrtho(-5.0, 5.0, -5.0*(GLfloat)h/(GLfloat)w,
              5.0*(GLfloat)h/(GLfloat)w, -5.0, 5.0);
   else
      glOrtho(-5.0*(GLfloat)w/(GLfloat)h,
              5.0*(GLfloat)w/(GLfloat)h, -5.0, 5.0, -5.0, 5.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}

int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (500, 500);
   glutInitWindowPosition (100, 100);
   glutCreateWindow (argv[0]);
   init ();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutMainLoop();
```

```
  return 0;
```

OUTPUT:



EXPERIMENT-10

CONSTRUCT A FOLLOWING 3D SHAPES: CUBE & SPHERE

CODE:

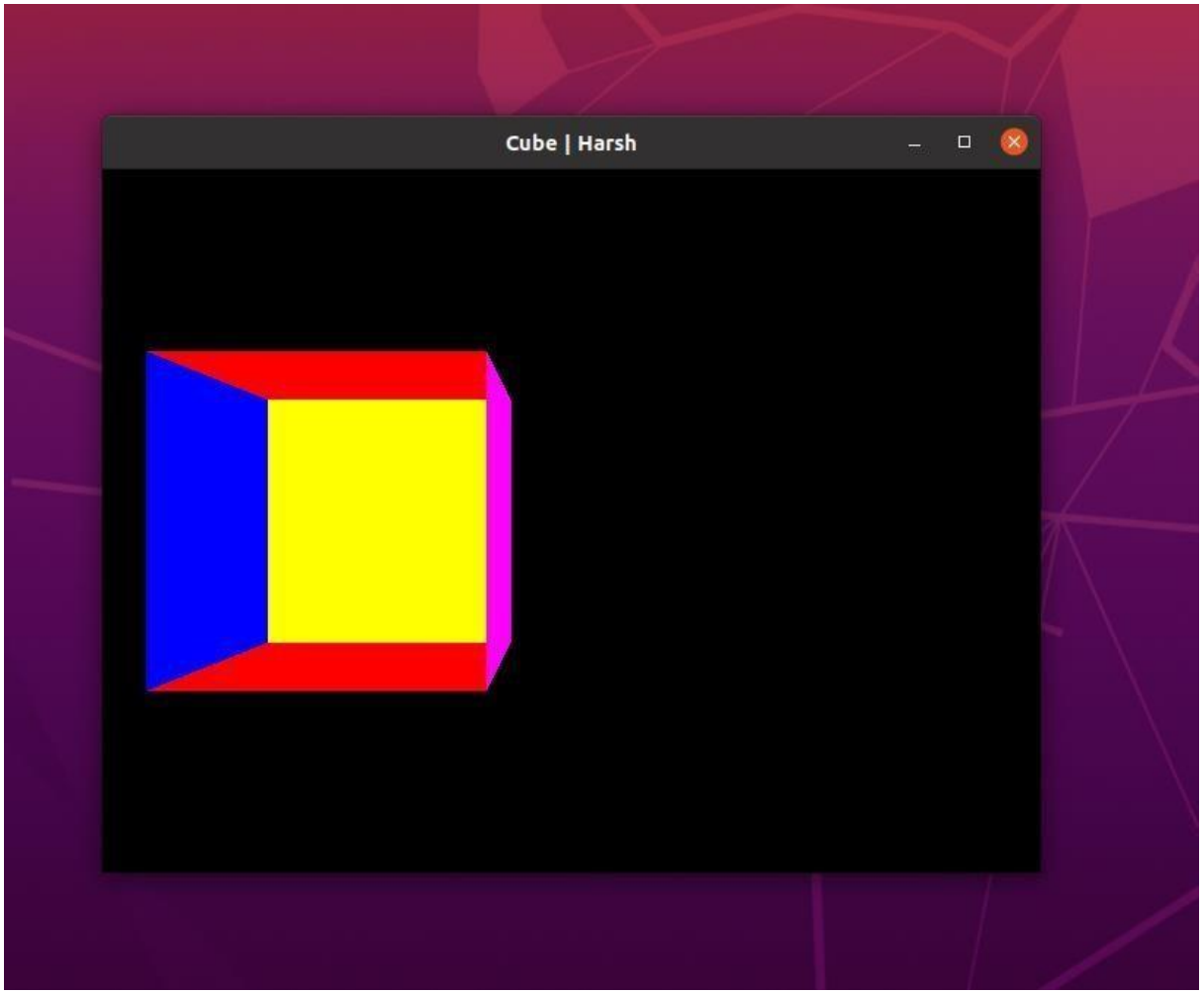
```
  #include<GL/glut.h>
  #include<GL/gl.h>
  char title[] = "3D Shapes"; void
  initGL() {
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glClearDepth(1.0f);
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    glShadeModel(GL_SMOOTH);
    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
  }
```

```
void display() {
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); //
Clear color and depth buffers
   glMatrixMode(GL_MODELVIEW);                    // To operate on model-view matrix

   // Render a color-cube consisting of 6 quads with differentcolors glLoadIdentity(); //
   Reset the model-view matrix glTranslatef(-1.5f, 0.0f, -6.0f); // Move right and into
   thescreen
   glBegin(GL_QUADS);                  // Begin drawing the color cubewith 6 quads
// Top face (y = 1.0f)
      // Define vertices in counter-clockwise (CCW) order with normal pointing out
glColor3f(0.0f, 1.0f, 0.0f);                  // Green
glVertex3f( 1.0f, 1.0f, -1.0f);
glVertex3f(-1.0f, 1.0f, -1.0f);
glVertex3f(-1.0f, 1.0f, 1.0f); glVertex3f( 1.0f,
1.0f, 1.0f);
// Bottom face (y = -1.0f) glColor3f(1.0f, 0.5f, 0.0f);
                                              // Orange
glVertex3f( 1.0f, -1.0f,  1.0f);
glVertex3f(-1.0f, -1.0f,  1.0f);
glVertex3f(-1.0f, -1.0f, -1.0f);
glVertex3f( 1.0f, -1.0f, -1.0f);
// Front face (z = 1.0f) glColor3f(1.0f, 0.0f, 0.0f);
                                              // Red
glVertex3f( 1.0f, 1.0f, 1.0f); glVertex3f(-1.0f, 1.0f,
1.0f);
glVertex3f(-1.0f, -1.0f, 1.0f);
glVertex3f( 1.0f, -1.0f, 1.0f);
// Back face (z = -1.0f)
glColor3f(1.0f, 1.0f, 0.0f);                  // Yellow
```

```
glVertex3f( 1.0f, -1.0f, -1.0f);
glVertex3f(-1.0f, -1.0f, -1.0f);
glVertex3f(-1.0f,  1.0f, -1.0f);
glVertex3f( 1.0f,  1.0f, -1.0f);
// Left face (x = -1.0f) glColor3f(0.0f, 0.0f, 1.0f);
                                                    // Blue
glVertex3f(-1.0f, 1.0f,  1.0f);
glVertex3f(-1.0f,  1.0f, -1.0f);
glVertex3f(-1.0f, -1.0f, -1.0f);
glVertex3f(-1.0f, -1.0f,  1.0f);
// Right face (x = 1.0f)
glColor3f(1.0f, 0.0f, 1.0f);                        // Magenta
glVertex3f(1.0f, 1.0f, -1.0f); glVertex3f(1.0f, 1.0f, 1.0f);
glVertex3f(1.0f, -1.0f,  1.0f);
       glVertex3f(1.0f, -1.0f, -1.0f);
    glEnd();
  glutSwapBuffers();
  }
  void reshape(GLsizei width, GLsizei height) { if (height ==
    0) height = 1;
    GLfloat aspect = (GLfloat)width / (GLfloat)height; glViewport(0,
    0, width, height); glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0f, aspect, 0.1f, 100.0f);
  }
  int main(int argc, char** argv) { glutInit(&argc,
    argv); glutInitDisplayMode(GLUT_DOUBLE);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(50, 50);
    glutCreateWindow("Cube | Harsh");
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    initGL();
    glutMainLoop();
```

```
    return 0;
}
```

OUTPUT:



Ques : Construct the 3d shape

Sphere Code:

```
#include <GL/glut.h>
GLfloat xRotated, yRotated, zRotated; GLdouble
radius=1;


void display(void);
void reshape(int x, int y); void
idle(void)
{
```

```
        xRotated += 0.01;

         zRotated += 0.01;
         display();
}


int main (int argc, char **argv)
{
         glutInit(&argc, argv); glutInitWindowSize(350,350);
         glutCreateWindow("Solid Sphere"); xRotated =
         yRotated = zRotated = 30.0; xRotated=43;
         yRotated=50;

         glutDisplayFunc(display);
        glutReshapeFunc(reshape);
        glutIdleFunc(idle); glutMainLoop();
         return 0;
}

void display(void)
{

         glMatrixMode(GL_MODELVIEW);

         glClear(GL_COLOR_BUFFER_BIT);
         glLoadIdentity();

         glTranslatef(0.0,0.0,-5.0);

         glColor3f(0.9, 0.3, 0.2);

         glRotatef(xRotated,1.0,0.0,0.0);

         glRotatef(yRotated,0.0,1.0,0.0);

         glRotatef(zRotated,0.0,0.0,1.0);

         glScalef(1.0,1.0,1.0);
```

```
        // built-in (glut library) function , draw you a sphere.
        glutSolidSphere(radius,20,20);
        // Flush buffers to screen

        glFlush();

}

void reshape(int x, int y)
{
        if (y == 0 || x == 0) return;
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(39.0,(GLdouble)x/(GLdouble)y,0.6,21.0);
        glMatrixMode(GL_MODELVIEW);
        glViewport(0,0,x,y);
```

- **Output Are As Follows : -**