RECORD OF EXPERIMENTS

Computer Graphics Lab

(CSEG3103)

Submitted By:

KUNAL BHARDWAJ

Professor,SOCS

Submitted To:

Mr.Arjun Arora

Assistant

V Semester, B-1

Sap ID:500069474

Roll No: R100218027

School of Computer Science

UNIVERSITY OF PETROLEUM AND ENERGY STUDIES

Dehradun-248007 2020-21

INDEX

S.No.	Objective of the Experiment	Date of Submission	Remarks
1.	Introduction to OpenGL and initialize a Green color		
2.	Drawing Line using Bresenham's Algorithm		
3.	Drawing Line using DDA Algorithm		
4.	1.Draw an ellipse using ellipse generation algorithm 2.Filling objects using Flood Fill and Boundary Fill		
5.	Perform Clipping Operation On line Using Cohen Sutherland		
6.	Performing Clipping operation on polygon using Sutherland Hodgeman		
7.	Write an interactive program for following basic transformation. ▪ Translation ▪ Rotation ▪ Scaling ▪ Reflection ▪ Shearing		
8.	Write an interactive program for following basic transformation. ▪ Translation ▪ Rotation ▪ Scaling ▪ Reflection		
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 hardwareaccelerated 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 OpenGLArchitecture?

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.

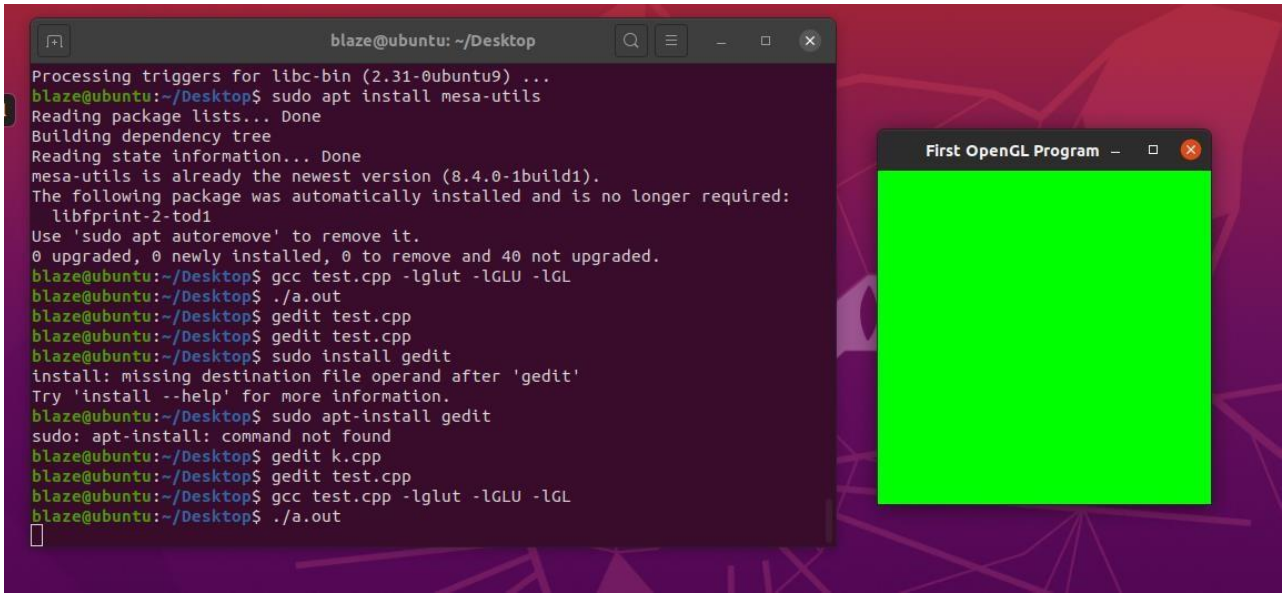
Modeling, rendering, and interaction is very much a cooperative process between the CPU client program and the GPU server programs written in GLSL.

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:



EXP2

AIM: Drawing Line using Bresenham's 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) {

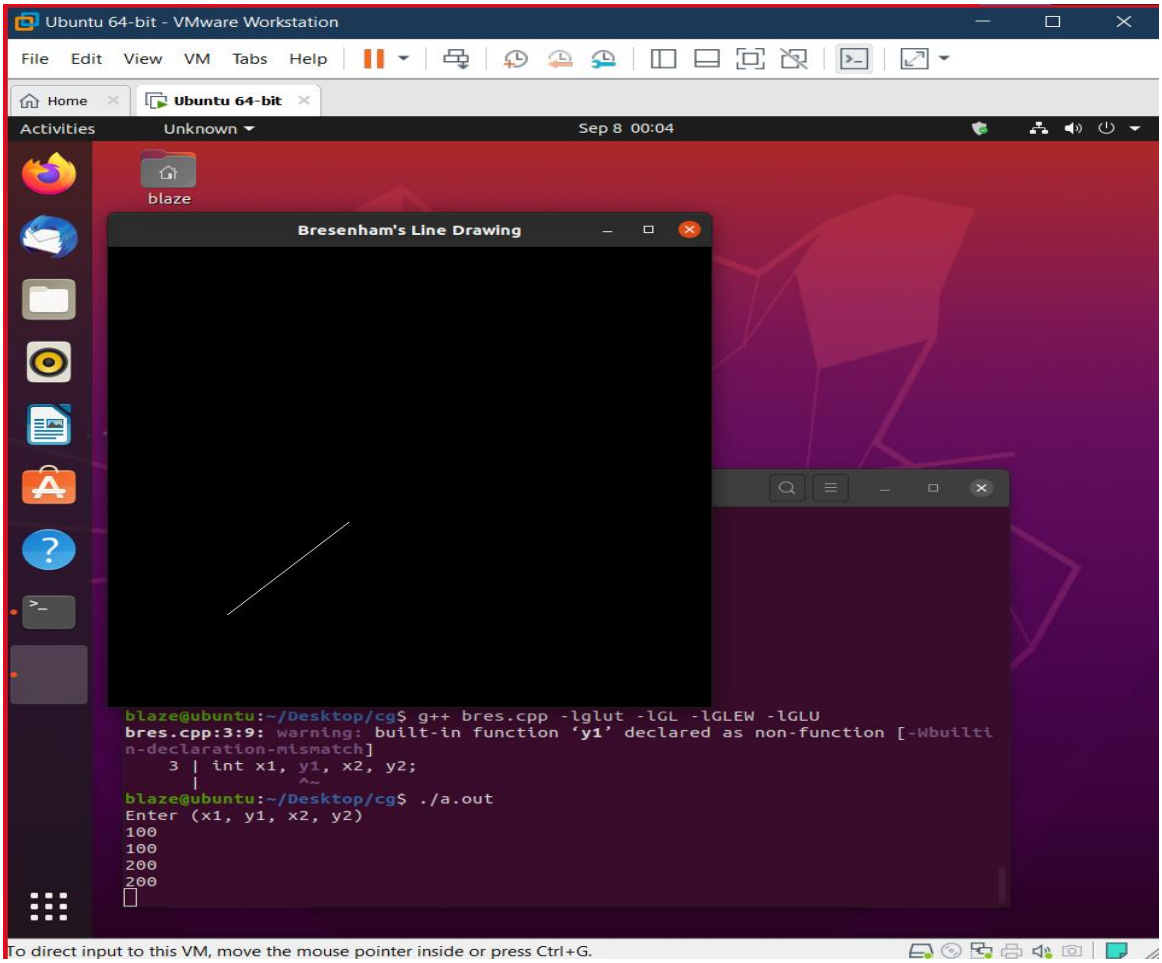
glBegin(GL_POINTS);

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, &x2, &y2);
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500, 500);
glutInitWindowPosition(0, 0);
glutCreateWindow("Bresenham's Line Drawing");
myInit();
glutDisplayFunc(myDisplay);
glutMainLoop();
}
```

OUTPUT:



EXP2

AIM: Drawing Line using DDA Algorithm

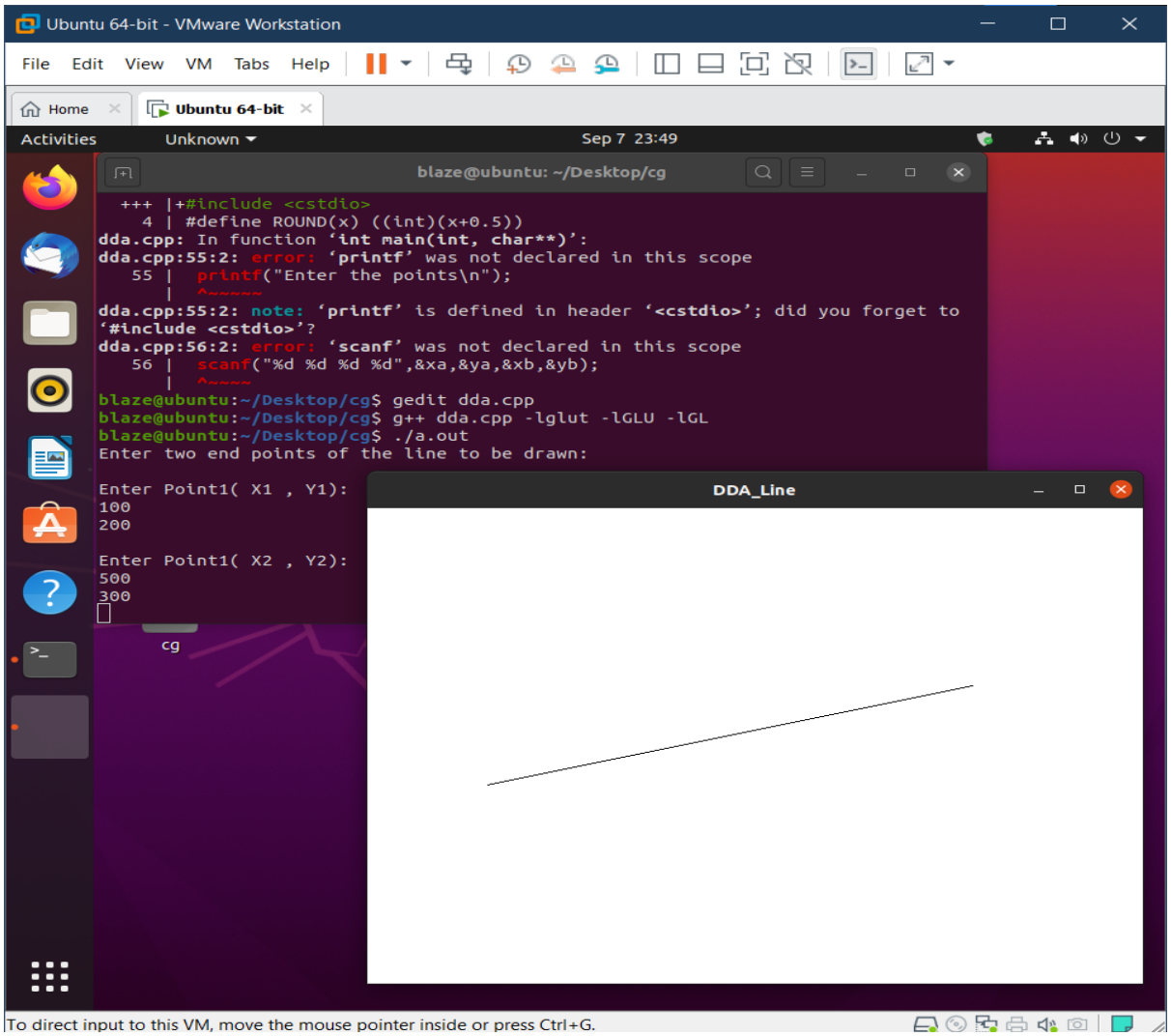
CODE:

```
#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();
  return 0;
}
```

OUTPUT:



EXP3

AIM: Draw an ellipse using ellipse generation algorithm

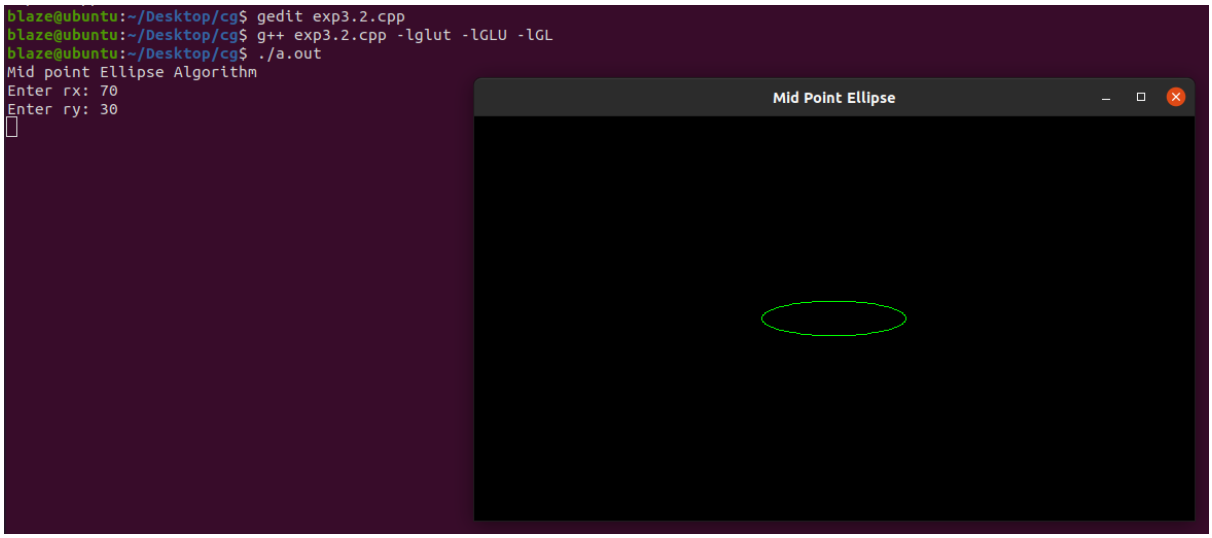
CODE:

```
#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:



EXP3

AIM: Drawing a circle using circle generation algorithm

CODE:

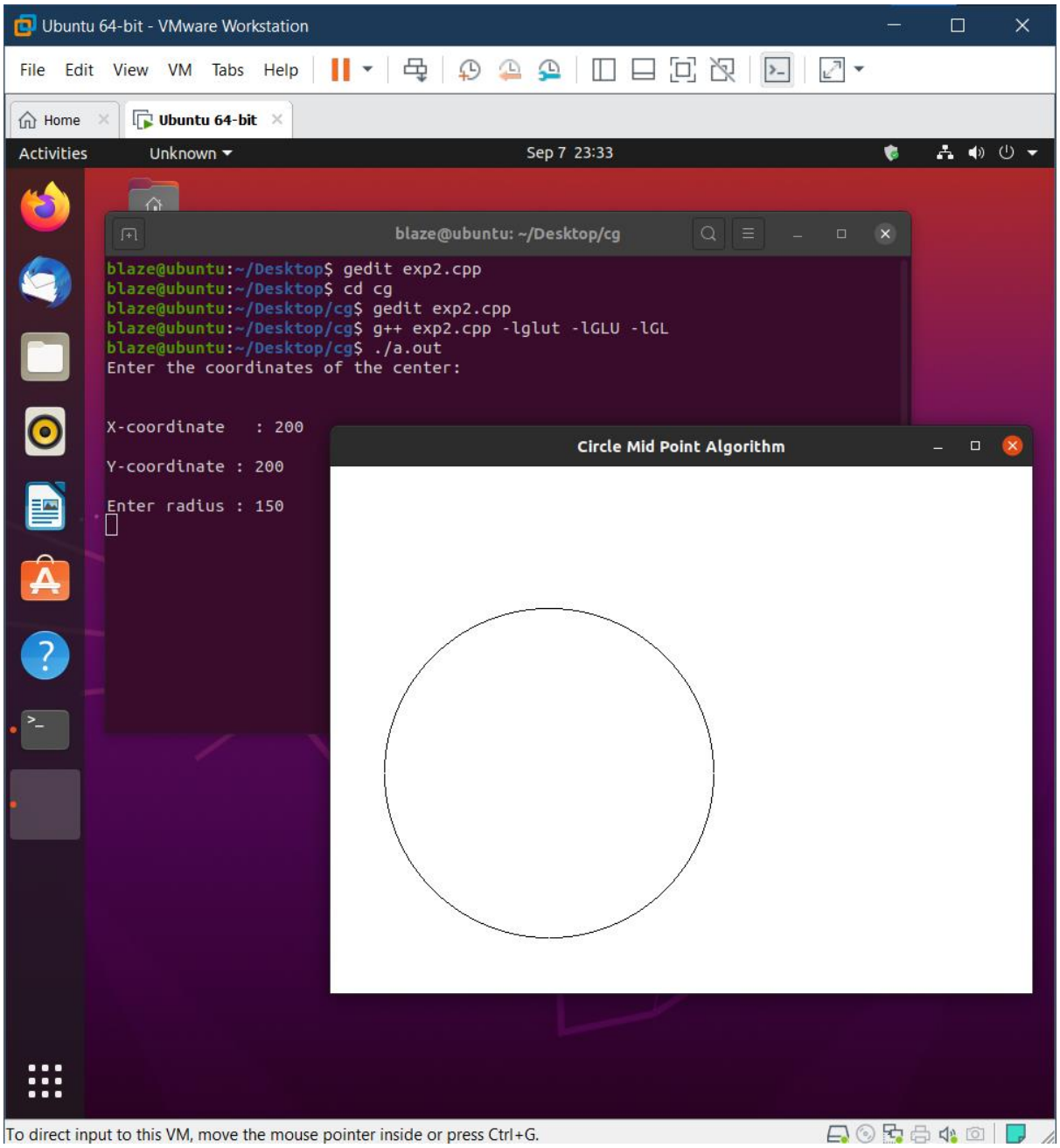
```
#include <stdio.h>

#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:



EXP4

AIM: Filling the objects using flood fill, boundary fill

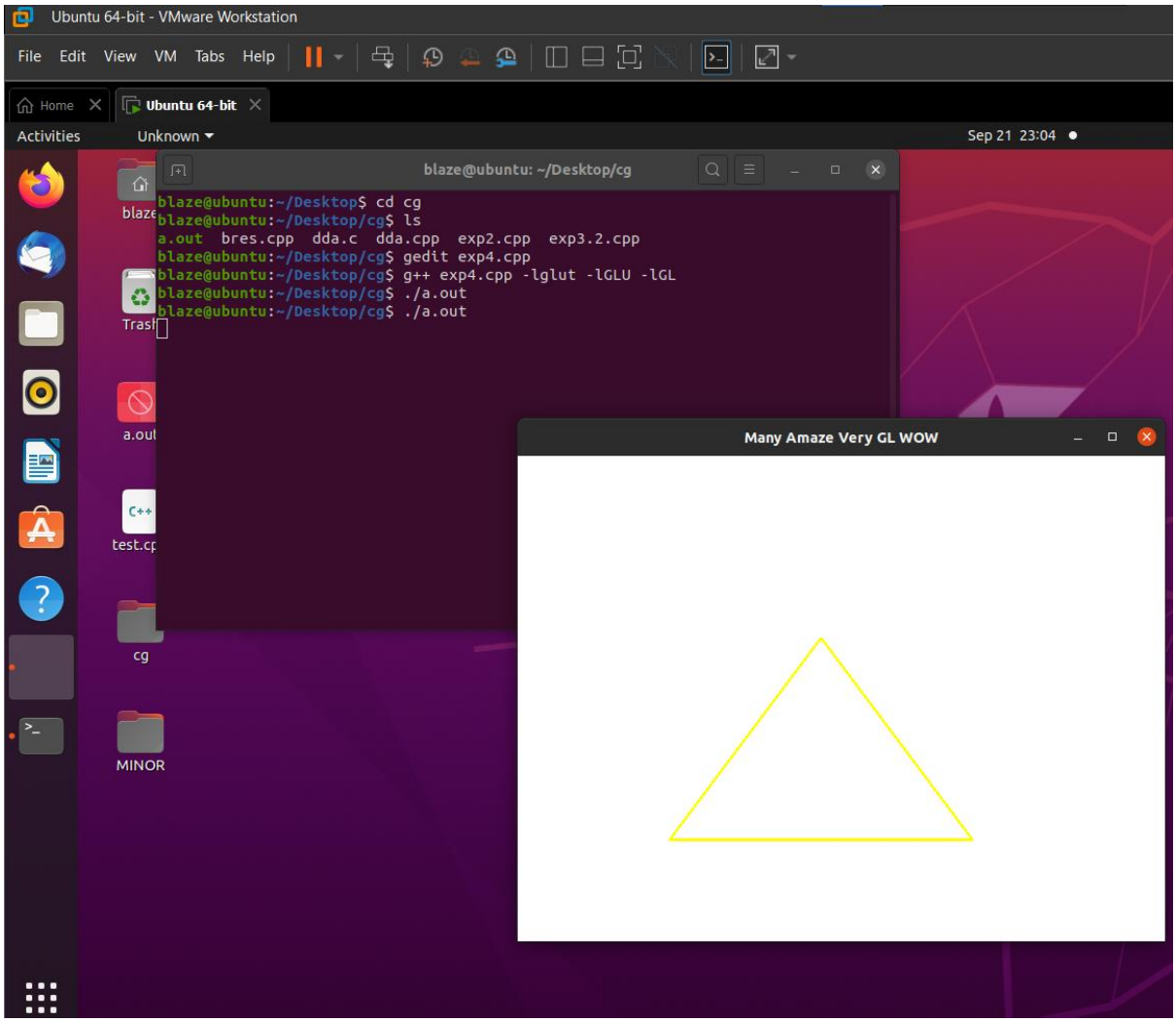
CODE:

```
#include <iostream>

#include <math.h>

#include <time.h>
```

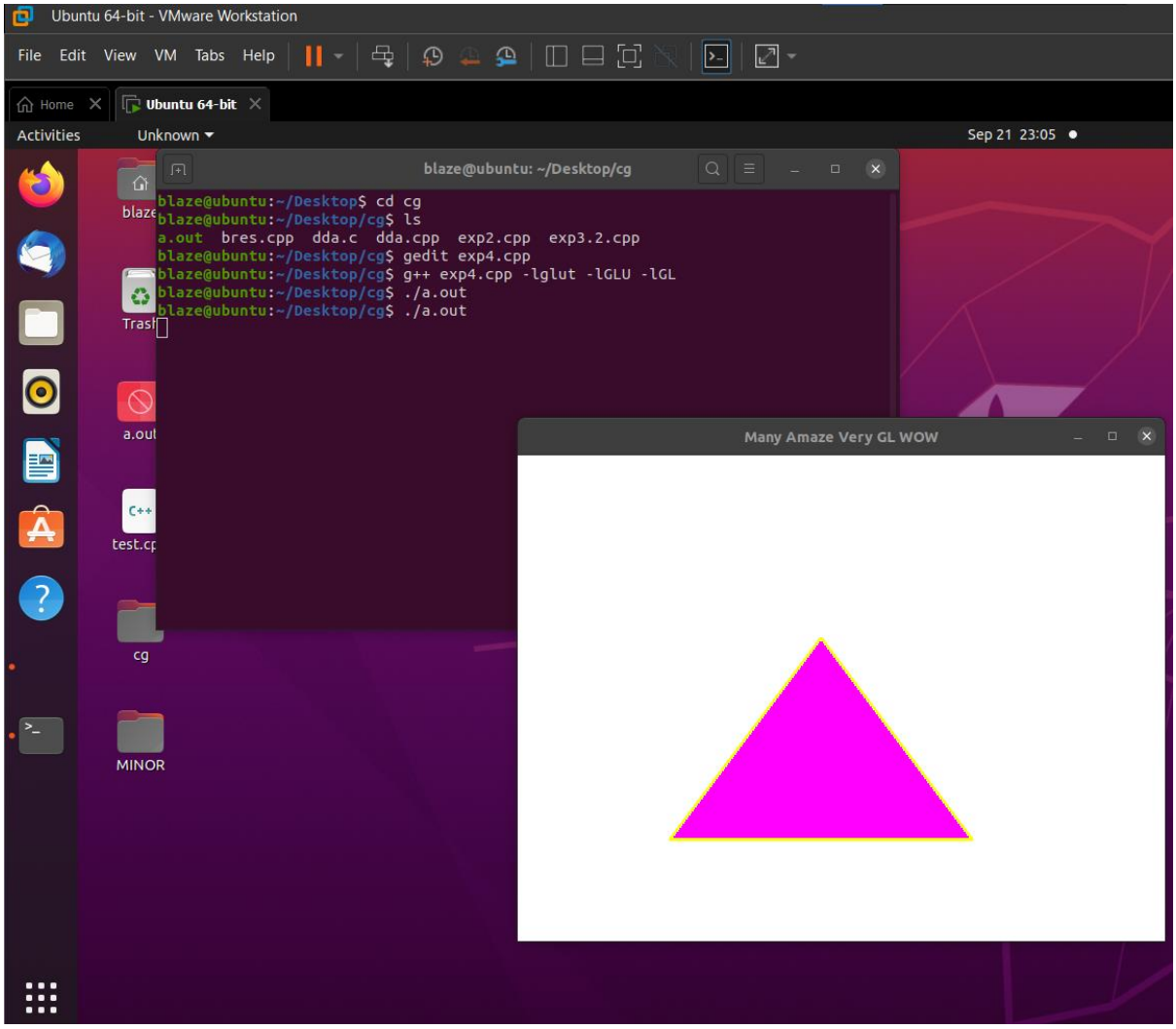
```
#include <GL/glut.h>
using namespace std;
void delay(float ms){
clock_t goal = ms + clock();
while(goal>clock());
}
void init(){
glClearColor(1.0,1.0,1.0,0.0);
glMatrixMode(GL_PROJECTION);
gluOrtho2D(0,640,0,480);
}
void bound_it(int x, int y, float* fillColor, float* bc){
float color[3];
glReadPixels(x,y,1.0,1.0,GL_RGB,GL_FLOAT,color);
if((color[0]!=bc[0] || color[1]!=bc[1] || color[2]!=bc[2])&&(
color[0]!=fillColor[0] || color[1]!=fillColor[1] || color[2]!=fillColor[2])){
glColor3f(fillColor[0],fillColor[1],fillColor[2]);
glBegin(GL_POINTS);
glVertex2i(x,y);
glEnd();
glFlush();
bound_it(x+1,y,fillColor,bc);
bound_it(x-2,y,fillColor,bc);
bound_it(x,y+2,fillColor,bc);
bound_it(x,y-2,fillColor,bc);
}
}
void mouse(int btn, int state, int x, int y){
y = 480-y;
if(btn==GLUT_LEFT_BUTTON)
{
if(state==GLUT_DOWN)
{
```

```
float bCol[] = {1,1,0};
float color[] = {1,0,1};
bound_it(x,y,color,bCol);
}
}
}
void world(){
glLineWidth(3);
glPointSize(2);
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1,1,0);
glBegin(GL_LINE_LOOP);
glVertex2i(150,100);
glVertex2i(300,300);
glVertex2i(450,100);
glEnd();
glFlush();
}
int main(int argc, char** argv){
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(640,480);
glutInitWindowPosition(200,200);
glutCreateWindow("Many Amaze Very GL WOW");
glutDisplayFunc(world);
glutMouseFunc(mouse);
init();
glutMainLoop();
return 0;
}
```

OUTPUT:BOUNDRY FILL



2.FLOODFILL:



EXP 4

TITLE: Filling objects using Flood Fill and Boundary Fill.

Flood Fill Algorithm CODE:

#include <GL/glut.h> int ww = 500, 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])

{

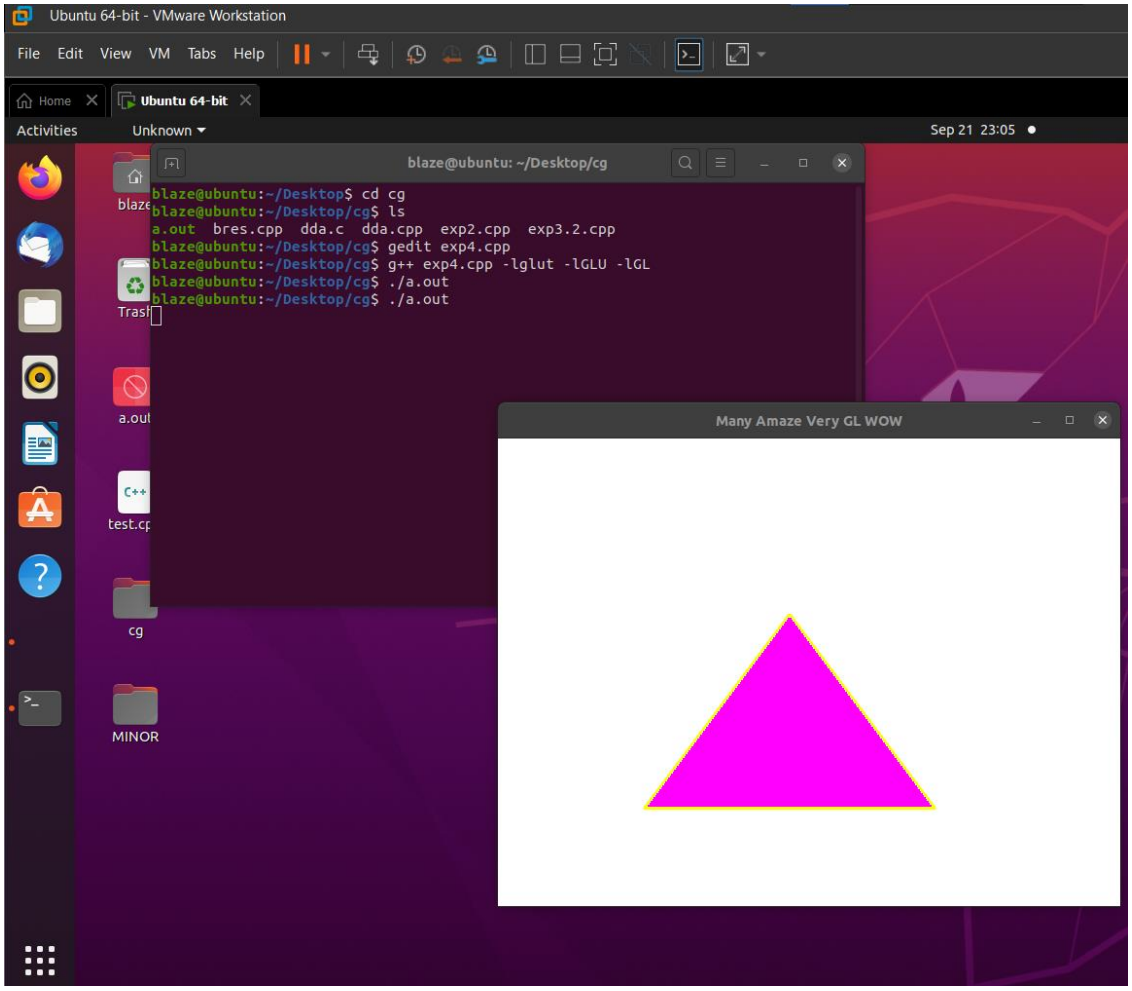
glReadPixels(x,y,1.0,1.0,GL_RGB,GL_FLOAT,pixels);

}

```
void drawPolygon(int x1, int y1, int x2, int y2)
{
        glColor3f(1.0, 1.0, 1.0);          glBegin(GL_POLYGON);  glVertex2i(x1,
y1);     glVertex2i(x1, y2);      glVertex2i(x2, y2);      glVertex2i(x2, y1);
glEnd();          glFlush();
}
void display()
{
        glClearColor(0.0, 0.0 , 0.0 , 0.0);
glClear(GL_COLOR_BUFFER_BIT);   drawPolygon(150,400,350,200);
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("Filling an object using Flood-Fill Algorithm"); glutDisplayFunc(display);
        myinit();
        glutMouseFunc(mouse);
        glutMainLoop();          return 0;
}
```

OUTPUT:



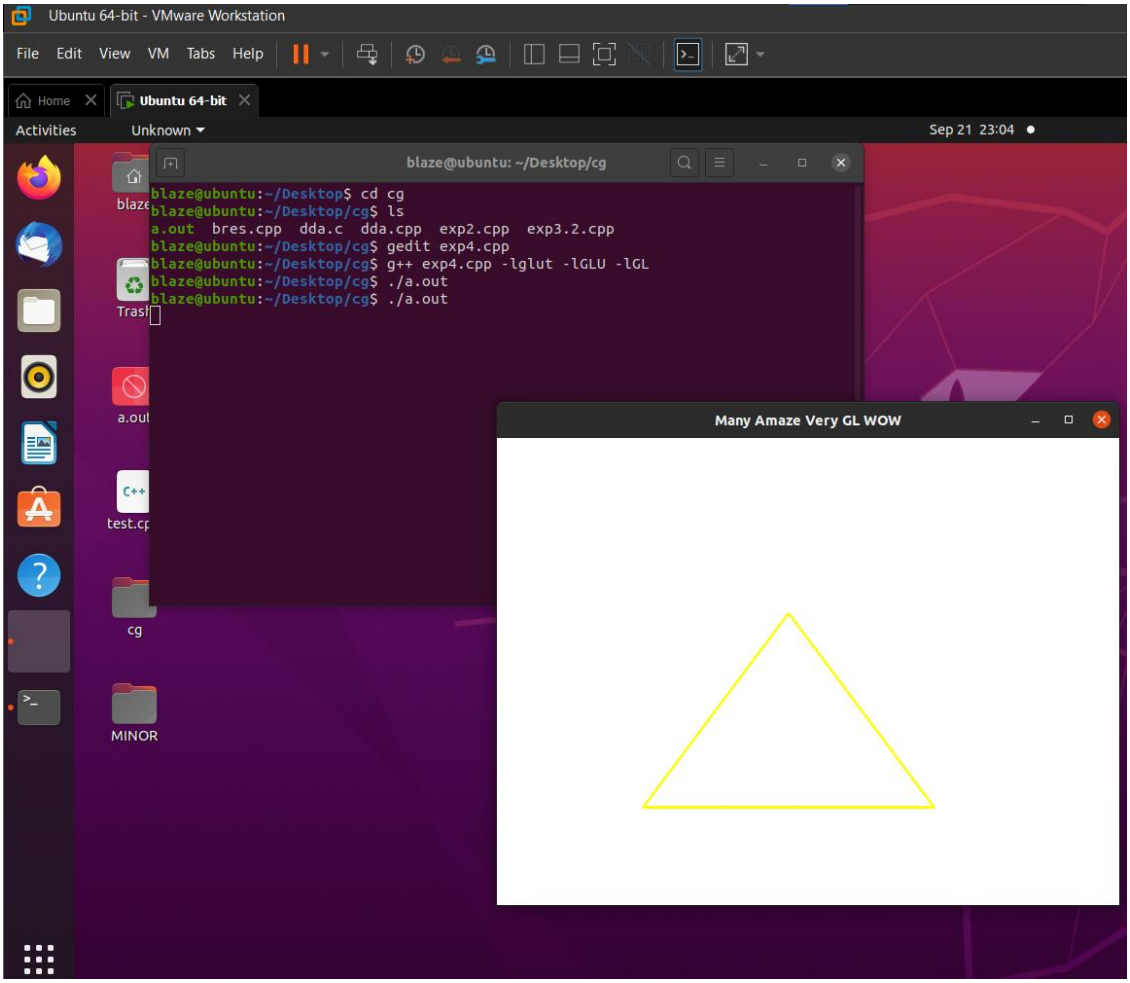
Boundary Fill Algorithm CODE:

```
#include <math.h> #include <GL/glut.h> struct
Point
{
        GLint x;
        GLint y;
};
struct Color
{
        GLfloat r;
        GLfloat g;
        GLfloat b;
}; void init()
{
        glClearColor(1.0, 1.0, 1.0, 0.0);        glColor3f(0.0,
0.0, 0.0);        glPointSize(1.0);
glMatrixMode(GL_PROJECTION);    glLoadIdentity();
gluOrtho2D(0, 500, 0, 500);
}
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 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);
        }
}
void onMouseClick(int button, int state, int x, int y)
{
        Color fillColor = {1.0f, 0.0f, 1.0f};
        Color boundaryColor = {0.0f, 0.0f, 0.0f};
        Point p = {51, 301}; //
        BoundaryFill(p.x, p.y, fillColor, boundaryColor);
}
void draw_dda(Point p1, Point p2)
{
        GLfloat dx = p2.x - p1.x;
        GLfloat dy = p2.y - p1.y;
        GLfloat x1 = p1.x;
      GLfloat y1 = p1.y;      GLfloat step = 0;
if(abs(dx) > abs(dy))
        {
          step = abs(dx);
        }
        else
        {
                step = abs(dy);
        }
        GLfloat xInc = dx/step;          GLfloat yInc =
dy/step;          for(float i = 1; i <= step; i++)
        {
                glVertex2i(x1, y1);              x1 += xInc;      y1 += yInc;
```

```
        }
}
void draw_square(Point a, GLint length)
{
        Point b = {a.x + length, a.y},      c =
{b.x,b.y+length},          d = {c.x-length, c.y};
draw_dda(a, b);            draw_dda(b, c);
draw_dda(c, d);            draw_dda(d, a);
}
void display(void)
{
        Point pt = {50, 300};     GLfloat length = 150;
glClear(GL_COLOR_BUFFER_BIT);   glBegin(GL_POINTS);
draw_square(pt, length);          glEnd();          glFlush();
}
int main(int argc, char** argv)
{
           glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);     glutInitWindowSize(500, 500);
glutInitWindowPosition(200, 200);
        glutCreateWindow("Filling an object with Boundary Fill Algorithm");     init();
        glutDisplayFunc(display);
glutMouseFunc(onMouseClick);          glutMainLoop();
return 0;
}
```

OUTPUT:



EXP – 5

Perform Clipping Operation On line Using Cohen Sutherland.

CODE :

```
#include<GL/glut.h>
#include<math.h>
#include<stdio.h>
#include<iostream>
void display();
using namespace std;
float xmin=-100;
float ymin=-100;
float xmax=100;
float ymax=100;
float xd1,yd1,xd2,yd2;
void init(void)
```

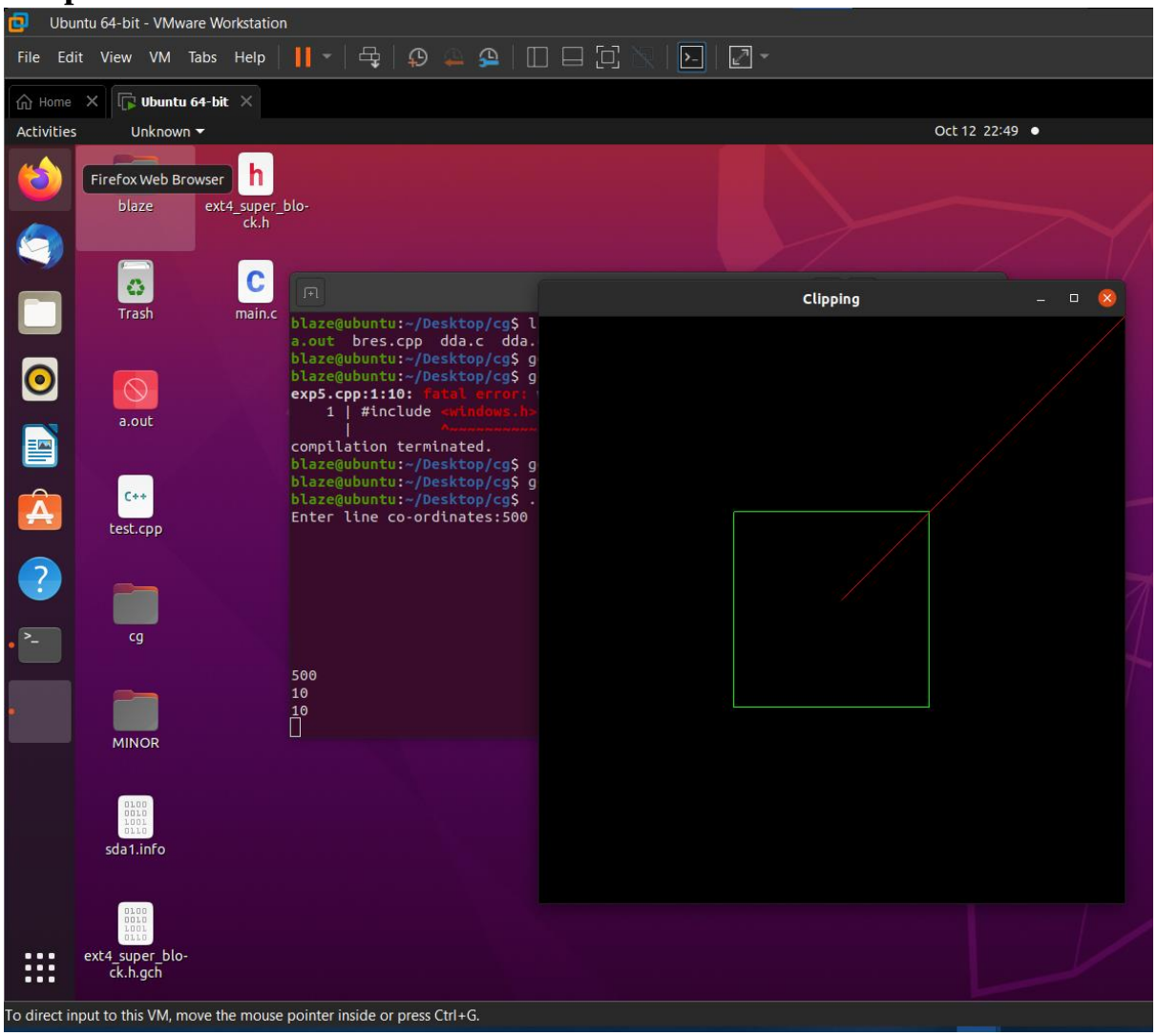
```
{
 glClearColor(0.0,0,0,0);
 glMatrixMode(GL_PROJECTION);
 gluOrtho2D(-300,300,-300,300);
}
int code(float x,float y)
{
 int c=0;
 if(y>ymax)c=8;
 if(y<ymin)c=4;
 if(x>xmax)c=c|2;
 if(x<xmin)c=c|1;
 return c;
}
void cohen_Line(float x1,float y1,float x2,float y2)
{
 int c1=code(x1,y1);
 int c2=code(x2,y2);
 float m=(y2-y1)/(x2-x1);
 while((c1|c2)>0)
 {
 if((c1 & c2)>0)
 {
 exit(0);
 }
 float xi=x1;float yi=y1;
 int c=c1;
 if(c==0)
 {
 c=c2;
 xi=x2;
 yi=y2;
 }
```

```
float x,y;
if((c & 8)>0)
{
y=ymax;
x=xi+ 1.0/m*(ymax-yi);
}
else
if((c & 4)>0)
{
y=ymin;
x=xi+1.0/m*(ymin-yi);
}
else
if((c & 2)>0)
{
x=xmax;
y=yi+m*(xmax-xi);
}
else
if((c & 1)>0)
{
x=xmin;
y=yi+m*(xmin-xi);
}
if(c==c1)
{
xd1=x;
yd1=y;
c1=code(xd1,yd1);
}
if(c==c2)
{
xd2=x;
```

```
 yd2=y;
 c2=code(xd2,yd2);
 }
}
display();
}
void mykey(unsigned char key,int x,int y)
{
 if(key=='c')
 { cout<<"Hello";
 cohen_Line(xd1,yd1,xd2,yd2);
 glFlush();
 }
}
void display()
{
 glClear(GL_COLOR_BUFFER_BIT);
 glColor3f(1.0,1.0,1.0);
 glBegin(GL_LINE_LOOP);
 glVertex2i(xmin,ymin);
 glVertex2i(xmin,ymax);
 glVertex2i(xmax,ymax);
 glVertex2i(xmax,ymin);
 glEnd();
 glColor3f(1.0,0.0,0.0);
 glBegin(GL_LINES);
 glVertex2i(xd1,yd1);
 glVertex2i(xd2,yd2);
 glEnd();
 glFlush();
}
int main(int argc,char** argv)
{
```

```
printf("Enter line co-ordinates:");
cin>>xd1>>yd1>>xd2>>yd2;
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(600,600);
glutInitWindowPosition(0,0);
glutCreateWindow("Cohen Sutherland Algorithm");
glutDisplayFunc(display);
glutKeyboardFunc(mykey);
init();
glutMainLoop();
return 0;
}
```

Output:



EXP-6

AIM: 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
```

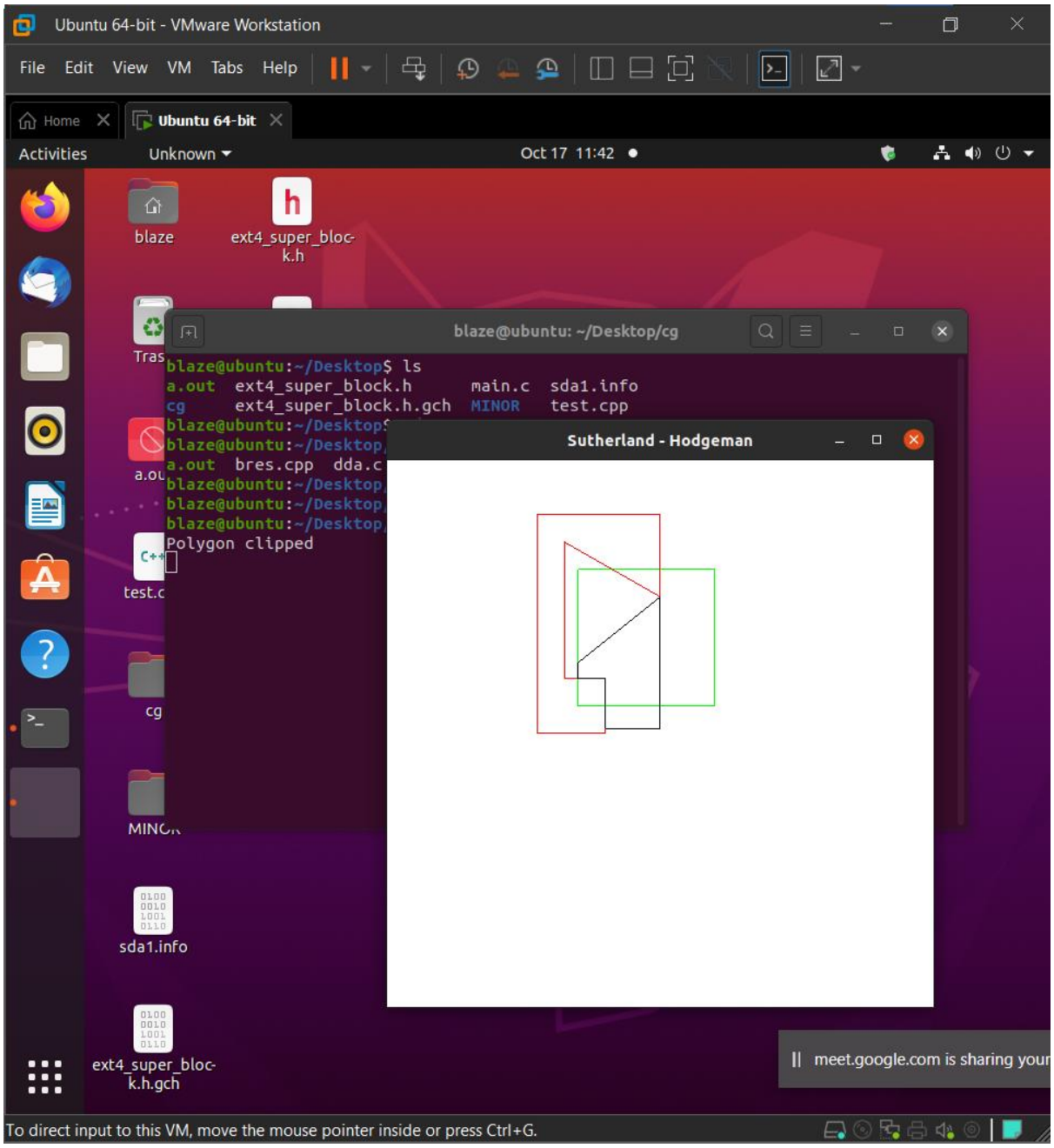
```
// and the second point is added
new_points[new_poly_size][0] = x_intersect(x1,y1, x2, y2, ix, iy, kx, ky);
new_points[new_poly_size][1] = y_intersect(x1,y1, x2, y2, ix, iy, kx, ky);
new_poly_size++;
new_points[new_poly_size][0] = kx;
new_points[new_poly_size][1] = ky;
new_poly_size++;
}
// Case 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
```

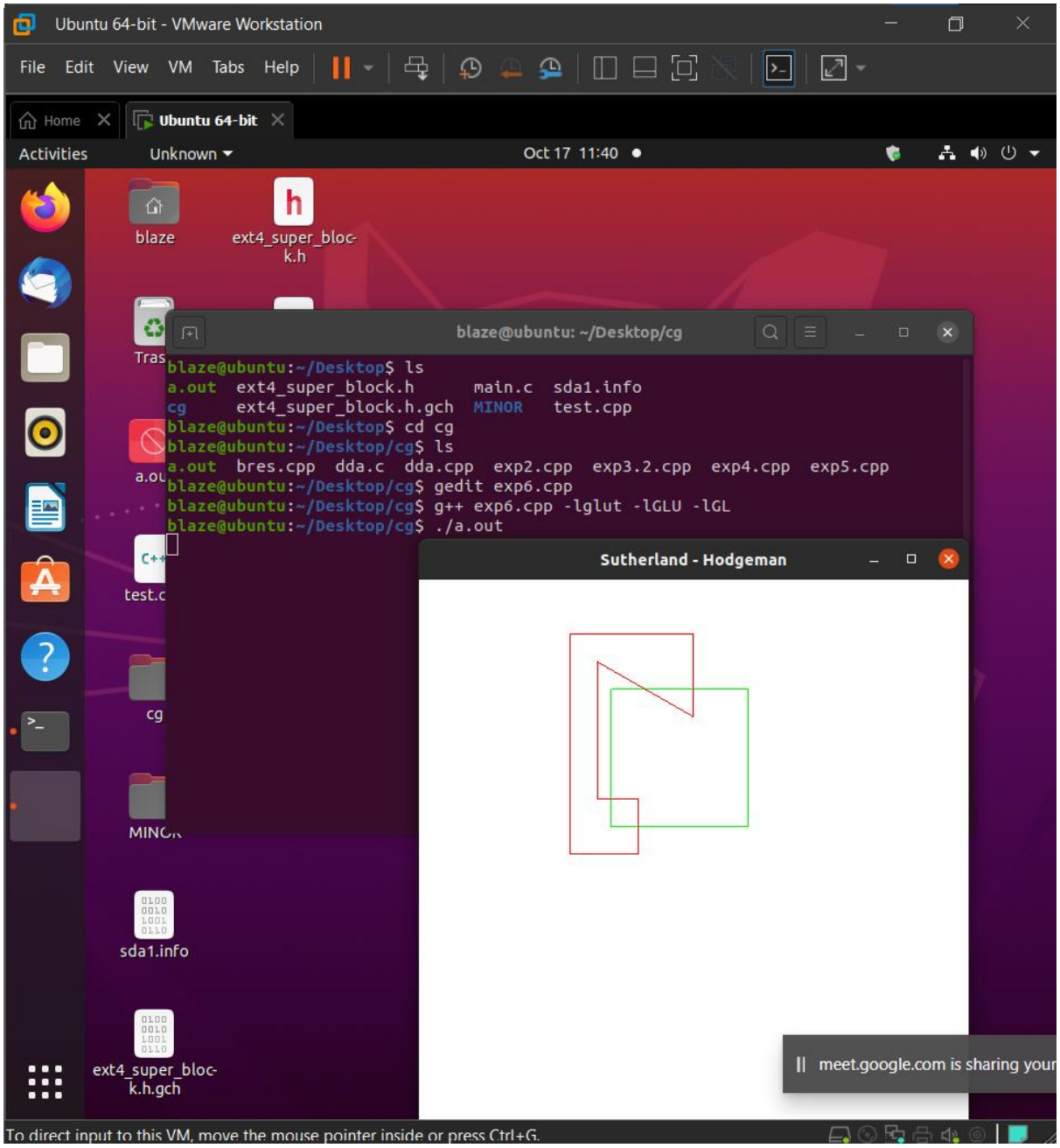
```
clipper_size)
{
//i and k are two consecutive indexes
for (int i=0; i<clipper_size; i++)
{
int k = (i+1) % clipper_size;
// We pass the current array of vertices, it's size
// and the end points of the selected clipper line
clip(poly_points, poly_size, clipper_points[i][0],
clipper_points[i][1], 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:


Ubuntu 64-bit - VMware Workstation
File Edit View VM Tabs Help
Home
Ubuntu 64-bit
Activities
Unknown
Oct 17 11:42
blaze
ext4_super_block.h
blaze@ubuntu: ~/Desktop/cg
blaze@ubuntu:~/Desktop$ ls
a.out ext4_super_block.h main.c sda1.info
cg ext4_super_block.h.gch MINOR test.cpp
blaze@ubuntu:~/Desktop
a.out bres.cpp dda.c
blaze@ubuntu:~/Desktop
blaze@ubuntu:~/Desktop
blaze@ubuntu:~/Desktop
Polygon clipped
Sutherland - Hodgeman
cg
sda1.info
ext4_super_block.h.gch
meet.google.com is sharing your
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.



EXP – 7 : 2 D Transformation

Ques : Write an interactive program for following basic transformation.

- **Translation**
- **Rotation**
- **Scaling**
- **Reflection**
- **Shearing**

Code : -

```
#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);
	}
```

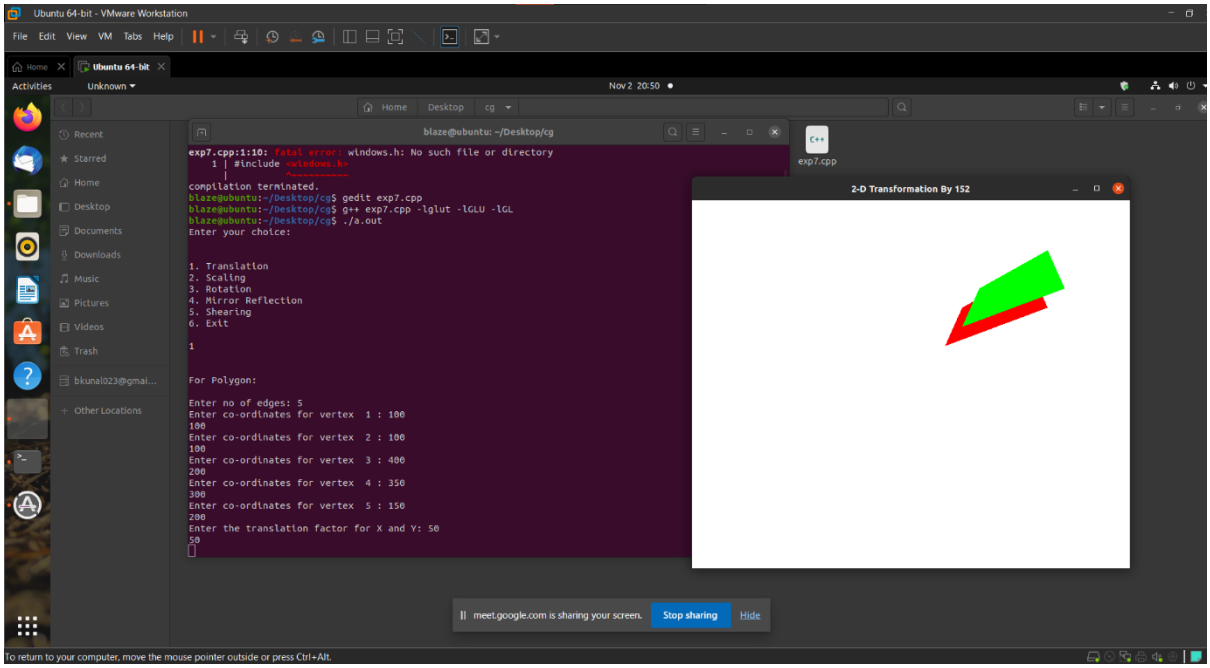
```
	glEnd();
}

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);
```

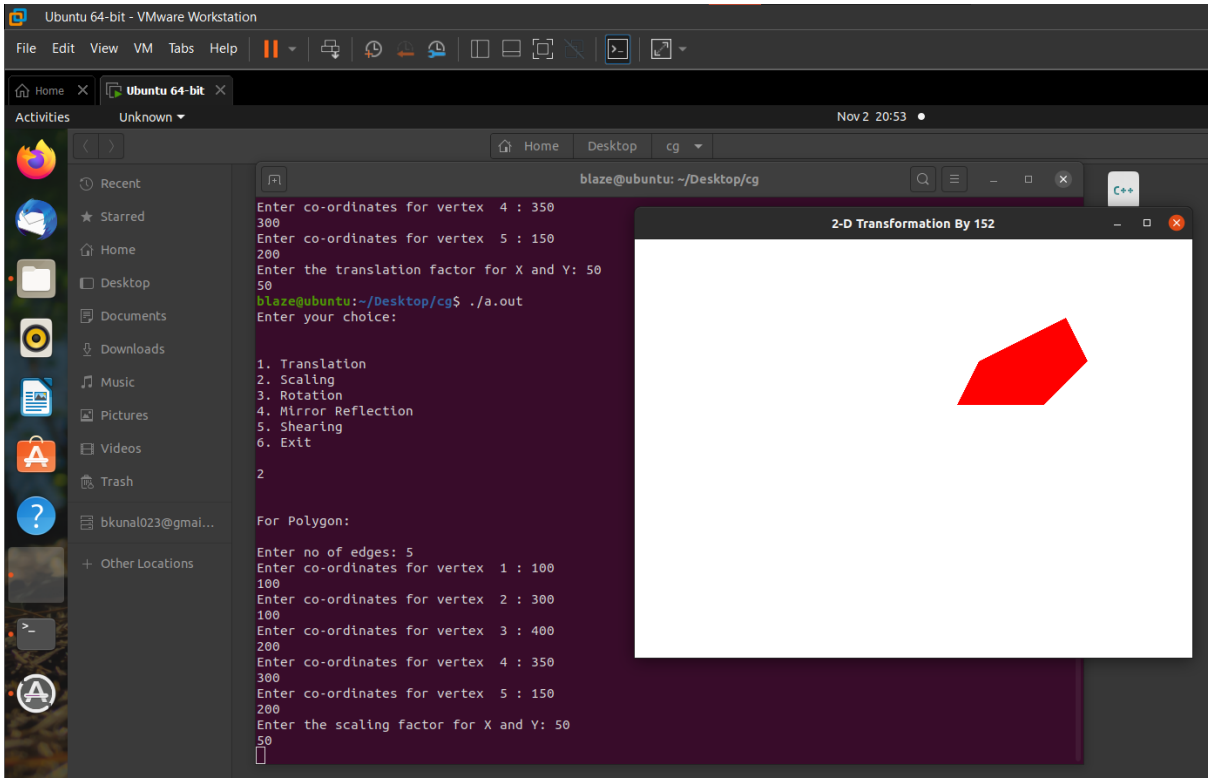
```
    glutInitWindowPosition(100, 150);
    glutCreateWindow("2-D Transformation By 152");
    glutDisplayFunc(myDisplay);
    myInit();
    glutMainLoop();

}
```

- **Output Are As Follows : -**

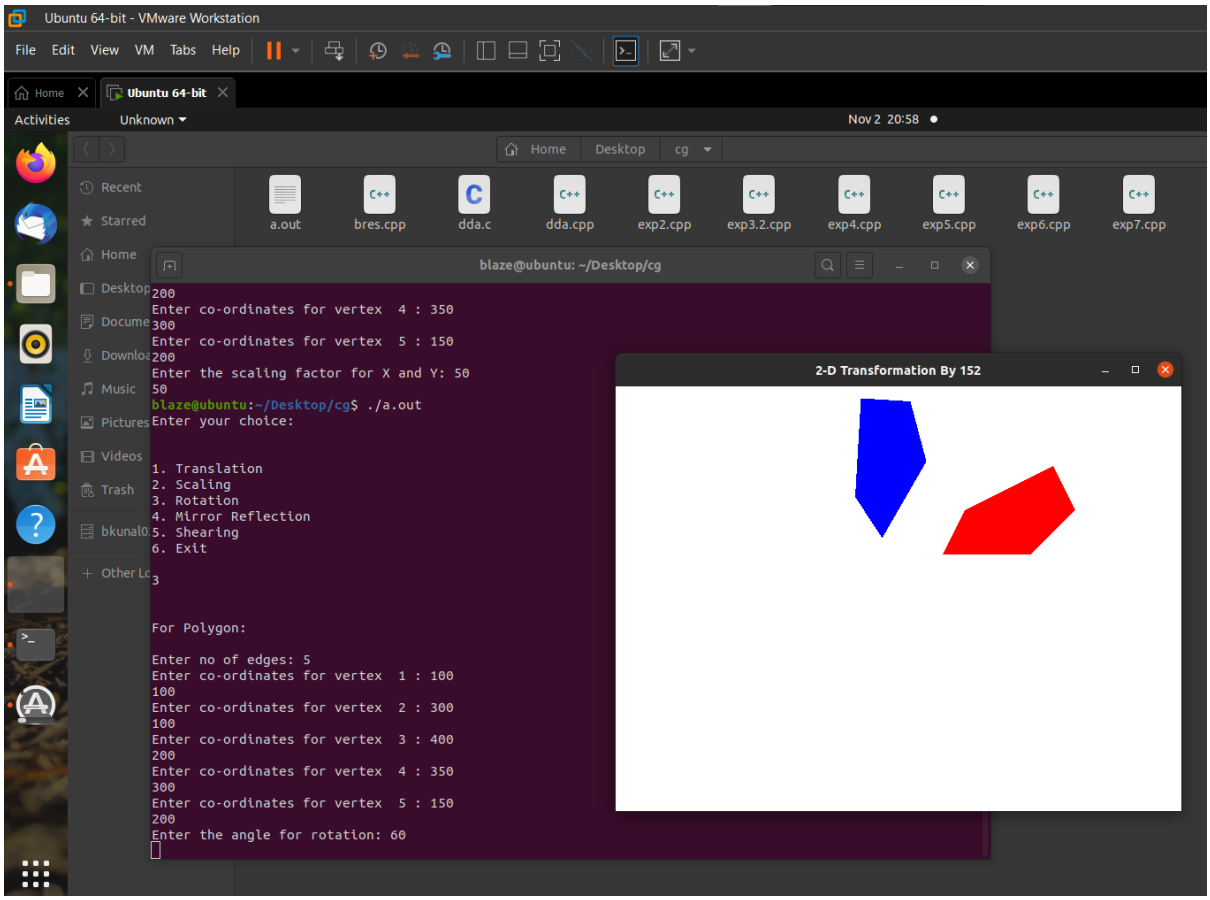
1.)Translation :



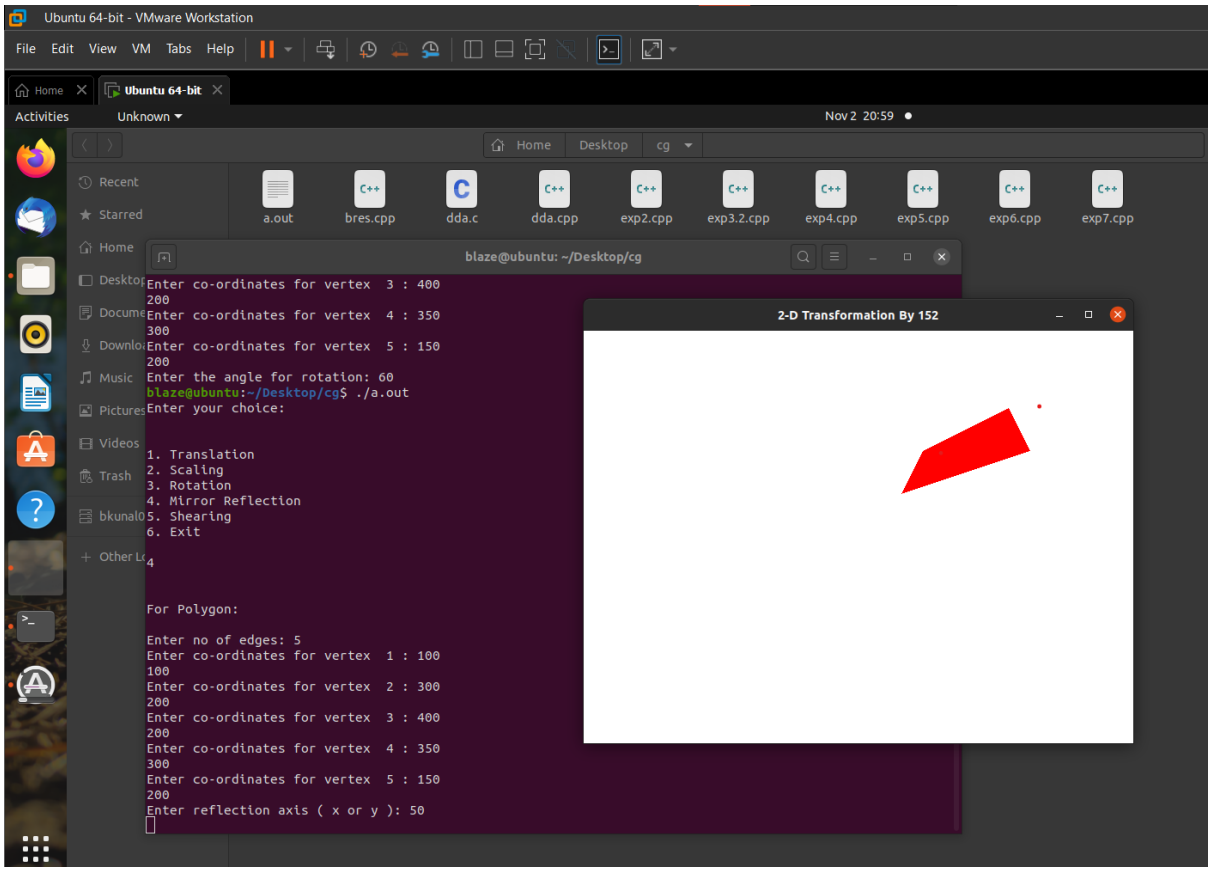
2.)Scaling


Ubuntu 64-bit - VMware Workstation
File Edit View VM Tabs Help
Activities
Unknown
Nov 2 20:53
Home
Desktop
cg
blaze@ubuntu: ~/Desktop/cg
2-D Transformation By 152
Enter co-ordinates for vertex 4 : 350
300
Enter co-ordinates for vertex 5 : 150
200
Enter the translation factor for X and Y: 50
50
blaze@ubuntu:~/Desktop/cg$ ./a.out
Enter your choice:
1. Translation
2. Scaling
3. Rotation
4. Mirror Reflection
5. Shearing
6. Exit
2
For Polygon:
Enter no of edges: 5
Enter co-ordinates for vertex 1 : 100
100
Enter co-ordinates for vertex 2 : 300
100
Enter co-ordinates for vertex 3 : 400
200
Enter co-ordinates for vertex 4 : 350
300
Enter co-ordinates for vertex 5 : 150
200
Enter the scaling factor for X and Y: 50
50

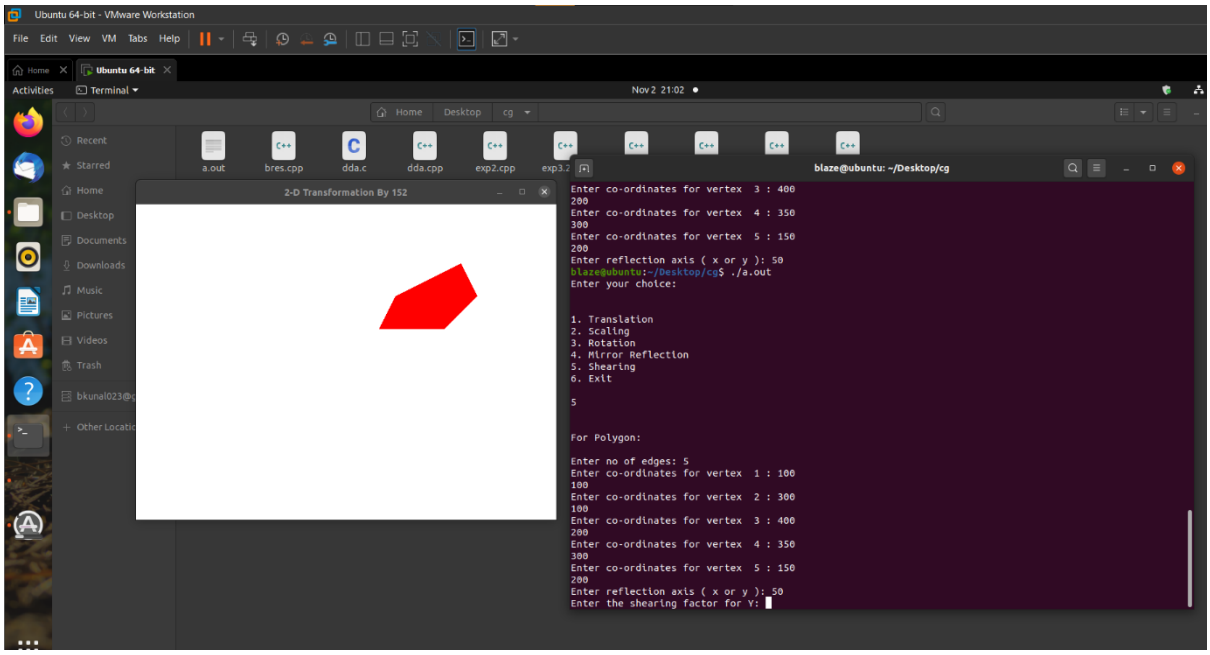
3.)Rotation


Ubuntu 64-bit - VMware Workstation
File Edit View VM Tabs Help
Activities
Unknown
Nov 2 20:58
Home
Desktop
cg
a.out
bres.cpp
dda.c
dda.cpp
exp2.cpp
exp3.2.cpp
exp4.cpp
exp5.cpp
exp6.cpp
exp7.cpp
blaze@ubuntu: ~/Desktop/cg
2-D Transformation By 152
200
Enter co-ordinates for vertex 4 : 350
300
Enter co-ordinates for vertex 5 : 150
200
Enter the scaling factor for X and Y: 50
50
blaze@ubuntu:~/Desktop/cg$ ./a.out
Enter your choice:
1. Translation
2. Scaling
3. Rotation
4. Mirror Reflection
5. Shearing
6. Exit
3
For Polygon:
Enter no of edges: 5
Enter co-ordinates for vertex 1 : 100
100
Enter co-ordinates for vertex 2 : 300
100
Enter co-ordinates for vertex 3 : 400
200
Enter co-ordinates for vertex 4 : 350
300
Enter co-ordinates for vertex 5 : 150
200
Enter the angle for rotation: 60

4.)Reflection



5.)Shearing



Experiment - 8 : 3D Transformation

Ques : Write an interactive program for following basic transformation.

- **Translation**

- **Rotation**
- **Scaling**
- **Reflection**

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]);
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();
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]);
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=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();
```

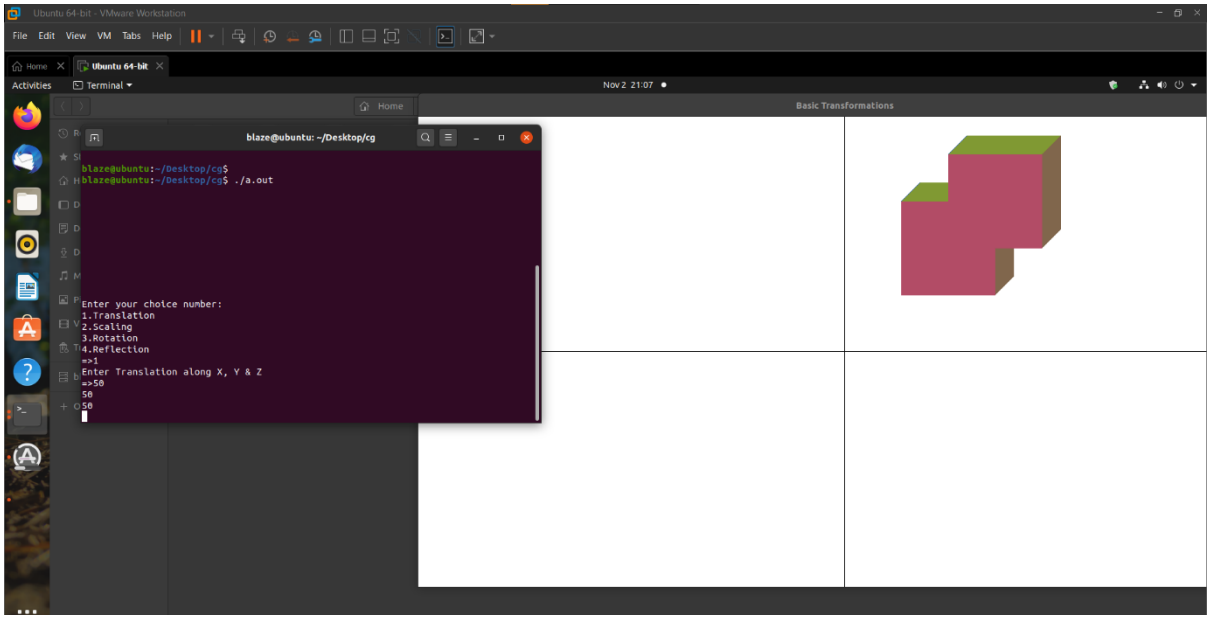
```
}
void init(void)
 {
 glClearColor (1.0, 1.0, 1.0, 1.0);
    // Set the Background color to WHITE
 glOrtho(-454.0, 454.0, -250.0, 250.0, -250.0, 250.0);
    // Set the no. of Co-ordinates along X & Y axes and their gappings
 glEnable(GL_DEPTH_TEST);
    // To Render the surfaces Properly according 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=>");
 scanf("%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.X-Z\n=>");
scanf("%d",&choiceRef);
break;
default:    printf("Please enter a valid choice!!!\n");
return 0;
}
glutDisplayFunc(display);
glutMainLoop();
```

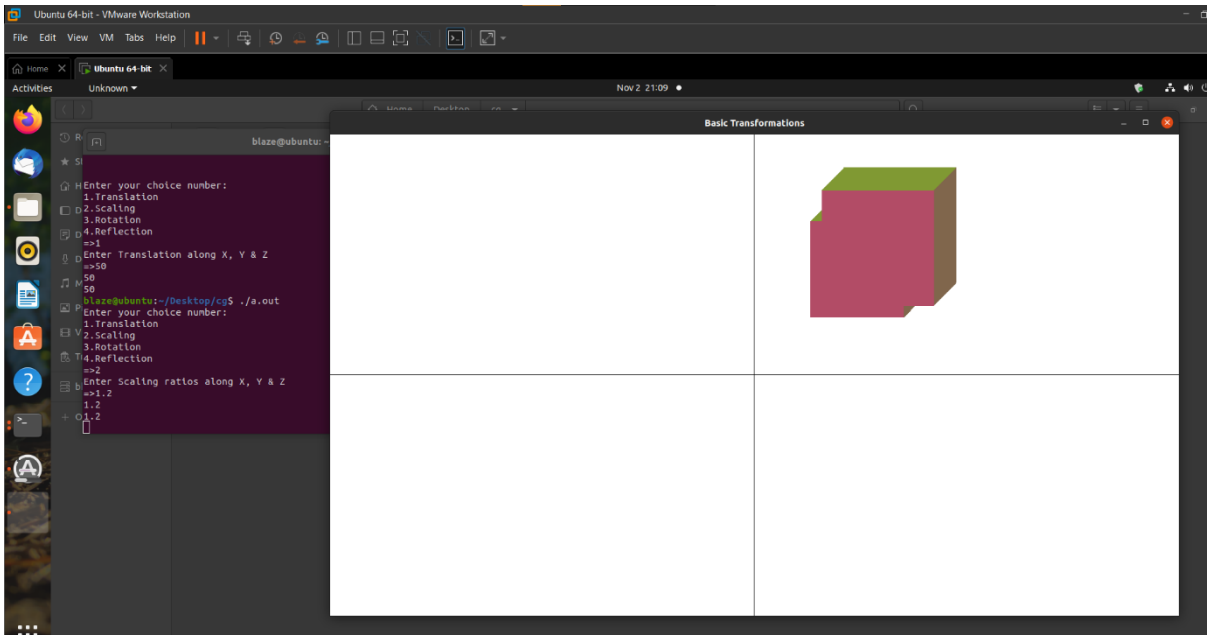
```
 return 0;
}
```

Output are as Follows ; -

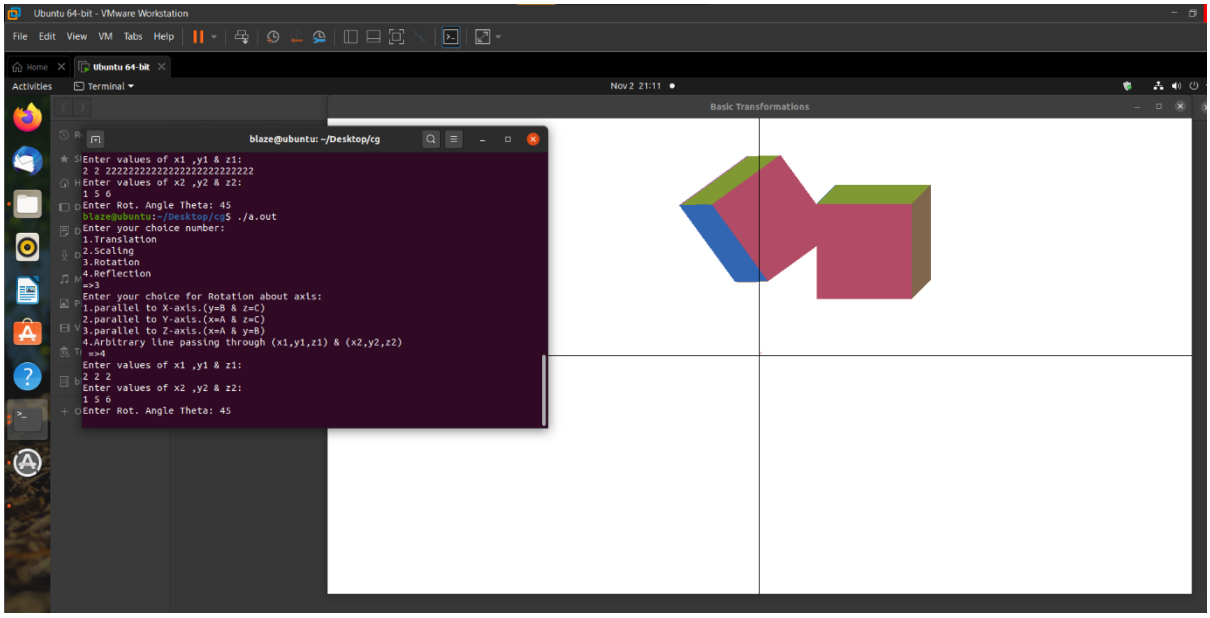
1.) Translation



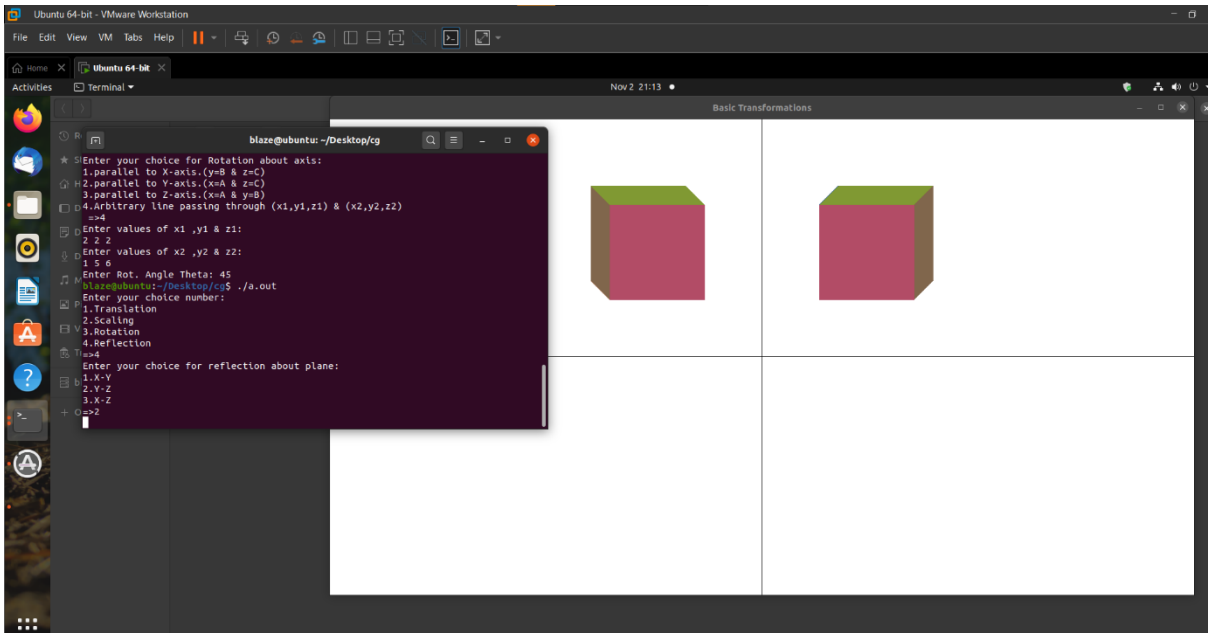
2.) Scaling



3.) Rotation



4.) Reflection



EXP-9

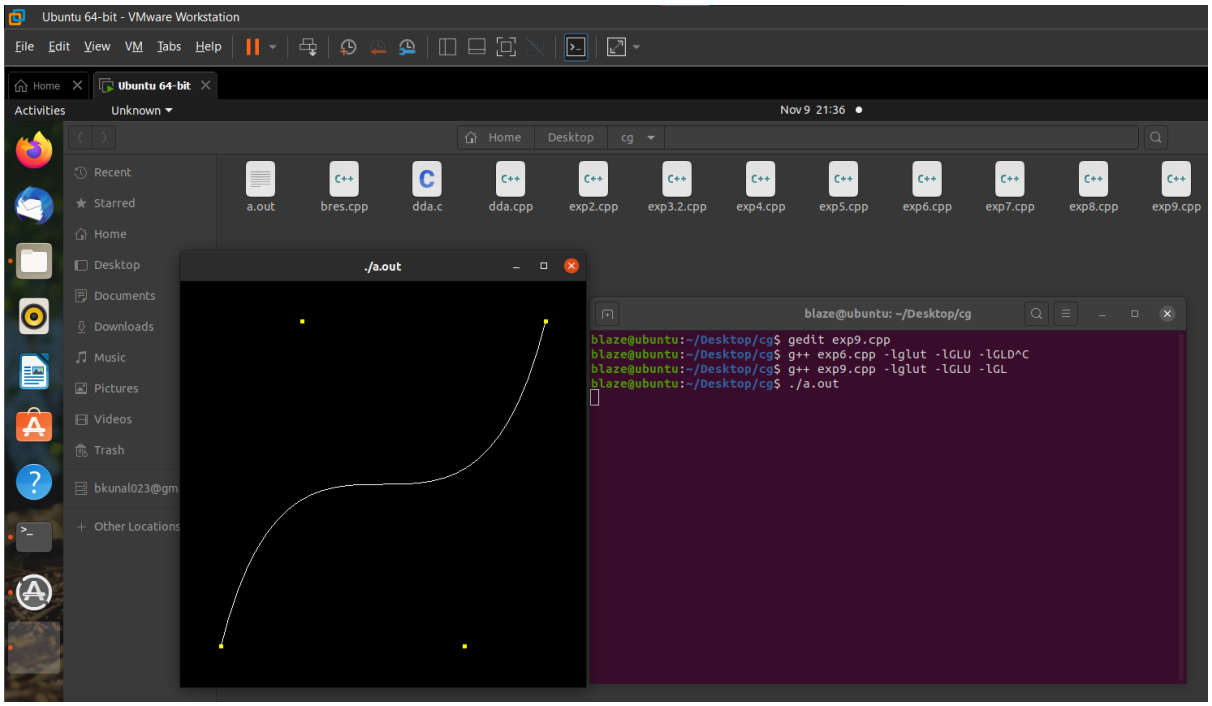
AIM: Construct a Bezier Curve

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:



EXP 10

Aim : Construct the following 3d Shapes: Cube and Sphere

a) CUBE CODE:

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

GLfloat xRotated, yRotated, zRotated; void init(void)

{ glClearColor(0,0,0,0);

}

void DrawCube(void) { glMatrixMode(GL_MODELVIEW);

// clear the drawing buffer. glClear(GL_COLOR_BUFFER_BIT); glLoadIdentity();

glTranslatef(0.0,0.0,-10.5);

glRotatef(xRotated,1.0,0.0,0.0); // rotation about Y axis glRotatef(yRotated,0.0,1.0,0.0);
// rotation about Z axis glRotatef(zRotated,0.0,0.0,1.0);

glBegin(GL_QUADS); // Draw The Cube Using quads glColor3f(0.0f,1.0f,0.0f); // Color 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);

glColor3f(1.0f,0.5f,0.0f); // Color Orange glVertex3f( 1.0f,-1.0f, 1.0f); // Top Right Of The Quad (Bottom)

// Top Right Of The Quad (Top)

// Top Left Of The Quad (Top)

// Bottom Left Of The Quad (Top)
```

```
// Bottom Right Of The Quad (Top)

glVertex3f(-1.0f,-1.0f, 1.0f); glVertex3f(-1.0f,-1.0f,-1.0f); glVertex3f( 1.0f,-1.0f,-1.0f);

glColor3f(1.0f,0.0f,0.0f); // Color 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); glColor3f(1.0f,1.0f,0.0f); // Color 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); glColor3f(0.0f,0.0f,1.0f); // Color 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); glColor3f(1.0f,0.0f,1.0f); // Color Violet

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);

// Top Right Of The Quad (Right)

// Top Left Of The Quad (Right)

// Bottom Left Of The Quad (Right)

// Top Left Of The Quad (Bottom)

// Bottom Left Of The Quad (Bottom) // Bottom Right Of The Quad (Bottom)

// Top Right Of The Quad (Front)

// Top Left Of The Quad (Front)

// Bottom Left Of The Quad (Front) // Bottom Right Of The Quad (Front)

// Top Right Of The Quad (Back)

// Top Left Of The Quad (Back)

// Bottom Left Of The Quad (Back)

// Bottom Right Of The Quad (Back)

// Top Right Of The Quad (Left)

// Top Left Of The Quad (Left)

// Bottom Left Of The Quad (Left) // Bottom Right Of The Quad (Left) // Bottom Right Of

The Quad (Right) glEnd(); // End Drawing The Cube glFlush(); } void animation(void) {

yRotated += 0.01; xRotated += 0.02; DrawCube();

}

void reshape(int x, int y) {

if (y == 0 || x == 0) return; //Nothing is visible then, so return //Set a new projection matrix
glMatrixMode(GL_PROJECTION);

glLoadIdentity();

//Angle of view:40 degrees
```

//Near clipping plane distance: 0.5 //Far clipping plane distance: 20.0

gluPerspective(40.0,(GLdouble)x/(GLdouble)y,0.5,20.0); glMatrixMode(GL_MODELVIEW);

glViewport(0,0,x,y); //Use the whole window for rendering

}

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

//we initizlilze the glut. functions glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB); glutInitWindowPosition(100, 100); glutCreateWindow(argv[0]);

init();

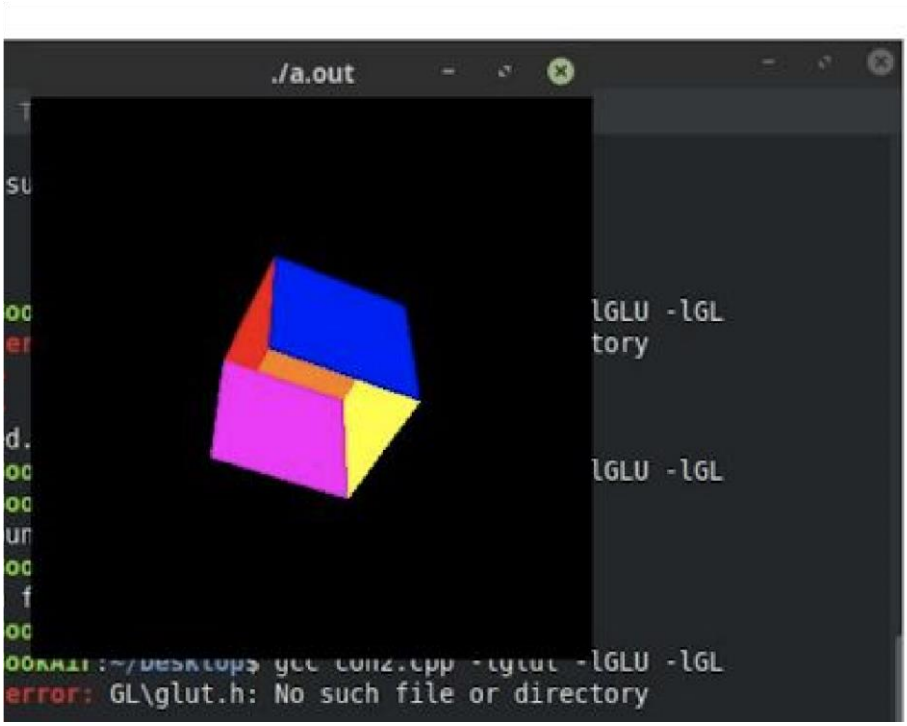
glutDisplayFunc(DrawCube); glutReshapeFunc(reshape);

//Set the function for the animation. glutIdleFunc(animation); glutMainLoop();

return 0;

}

Output :



b) SPHERE CODE:

#include <GL/glut.h>

GLfloat xRotated, yRotated, zRotated; GLdouble radius=1; void

redisplayFunc(void) { glMatrixMode(GL_MODELVIEW);

// clear the drawing buffer. glClear(GL_COLOR_BUFFER_BIT); // clear the identity matrix. glLoadIdentity();

```
// traslate the draw by z = -4.0

// Note this when you decrease z like -8.0 the drawing will looks far , or smaller. glTranslatef(0.0,0.0,-

4.5); // Red color used to draw. glColor3f(0.8, 0.2, 0.1);

// changing in transformation matrix.

// rotation about X axis glRotatef(xRotated,1.0,0.0,0.0);

// rotation about Y axis glRotatef(yRotated,0.0,1.0,0.0);

// rotation about Z axis glRotatef(zRotated,0.0,0.0,1.0);

// scaling transfomation 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();

// sawp buffers called because we are using double buffering // glutSwapBuffers();

}

void reshapeFunc(int x, int y) {

if (y == 0 || x == 0) return; //Nothing is visible then, so return //Set a new projection matrix
glMatrixMode(GL_PROJECTION);

glLoadIdentity();

//Angle of view:40 degrees

//Near clipping plane distance: 0.5 //Far clipping plane distance: 20.0

gluPerspective(40.0,(GLdouble)x/(GLdouble)y,0.5,20.0); glMatrixMode(GL_MODELVIEW);


glViewport(0,0,x,y); //Use the whole window for rendering } void

idleFunc(void) { yRotated += 0.01; redisplayFunc(); } int main (int argc, char

**argv) {

//Initialize GLUT glutInit(&argc, argv);

//double buffering used to avoid flickering problem in animation glutInitDisplayMode(GLUT_SINGLE |
GLUT_RGB);

// window size

glutInitWindowSize(400,350); // create the

window

glutCreateWindow("Sphere Rotating Animation"); glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);

xRotated = yRotated = zRotated = 30.0; xRotated=33;
```

```
yRotated=40; glClearColor(0.0,0.0.0.0,0.0); //Assign the function used in events
glutDisplayFunc(redisplayFunc); glutReshapeFunc(reshapeFunc); glutIdleFunc(idleFunc);

//Let start glut loop glutMainLoop(); return 0;

}
```

OUTPUT: