SAI VIDYA INSTITUTE OF TECHNOLOGY

(Affiliated to VTU, Belagavi, Approved by AICTE, New Delhi and Govt. of Karnataka)

Accredited by NBA, New Delhi (CSE, ISE, ECE, MECH, CIVIL), NAAC - "A" GRADE

Rajanukunte, Bengaluru-560064

Tel: 080-2846 8196, Fax: 2846 8193 / 98, Web: www.saividya.ac.in

MOTTO

"Learn to lead"

VISION

Contribute dedicated, skilled, intelligent engineers and business administrators to architect strong India and the world.

MISSION

To impart quality technical education and higher moral ethics associated with skilled training to suit the modern day technology with innovative concepts, so as to learn to lead the future with full confidence

COMPUTER GRAPHICS LABORATORY WITH MINI PROJECT

(18CSL67)

(As per Visvesvaraya Technological University Syllabus)

Compiled by:

Prof. Syed Matheen Pasha
Assistant Professor
Dept. of CSE

Prof. Vijayakumari G
Assistant Professor
Dept. of CSE

Name: ______________________________

USN :______________________________

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

2022-23

Trademark

Edition: 2022-23

Document Owner

The primary contacts for questions regarding this document are:

Author(s):	1. Prof. Syed Matheen Pasha 2. Prof. Vijayakumari G
Department:	Computer Science and Engineering

Contact email ids:

syedmatheen.pasha@saividya.ac.in
vijayakumari.g@saividya.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT VISION

Contribute dedicated, skilled, intelligent Computer Engineers to architect strong India and the world.

DEPARTMENT MISSION

Provide quality education in Computer Science by promoting excellence in Instruction, Research and Practice.

Promote Professional Interaction and Lifelong Learning

Encourage the youths to pursue career in Computer domain with modern innovation and ethics.

DEPARTMENT PROGRAM EDUCATIONAL OBJECTIVE

PEO 1:*Graduates will have the expertise in analyzing real time problems and providing appropriate solutions related to Computer Science & Engineering.*

PEO 2:*Graduates will have the knowledge of fundamental principles and innovative technologies to succeed in higher studies, and research.*

PEO 3:*Graduates will continue to learn and to adapt technology developments combined with deep awareness of ethical responsibilities in profession.*

.

Program Outcomes

1.Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2.Problem analysis: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3.Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4.Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5.Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6.The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice

7.Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8.Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9.Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10.Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11.Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12.Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes

PSO 1: Demonstrate the knowledge and understanding of working principles, design, implement, test and evaluate the hardware and software components of a computer system.

PSO2: Apply standard Software Engineering practices and strategies in software project development.

PSO3: Demonstrate the knowledge of Discrete Mathematics, Data management and Data engineering.

PROGRAM SPECIFIC OUTCOMES

Computer Science and Engineering Graduates will be able to:

PSO1: Demonstrate the knowledge and understanding of working principles, design, implement, test and evaluate the hardware and software components of a computer system.

PSO2: Apply standard Software Engineering practices and strategies in software project development.

PSO3: Demonstrate the knowledge of Discrete Mathematics, Data Management and Data Engineering.

COMPUTER GRAPHICS LABORATORY WITH MINI PROJECT

Subject Code: 18CSL67	**I.A. Marks : 40**
Hours/Week : 01I + 02 P	**Exam Hours : 03**
Total Hours : 40	**Exam Marks :60**

COURSE OUTCOMES:

After the completion of this course the students will be able to:

CO1 Apply the concepts of computer graphics

CO2 Implement computer graphics applications using OpenGL

CO3 Implement real world problems using OpenGL

PART A - Lab Experiments

Design, develop, and implement the following programs using OpenGL API

1. Implement Bresenham's line drawing algorithm for all types of slope.
2. Create and rotate a triangle about the origin and a fixed point.
3. Draw a colour cube and spin it using OpenGL transformation matrices.
4. Draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing.
5. Clip a lines using Cohen-Sutherland algorithm
6. To draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of the light source along with the properties of the surfaces of the solid object used in the scene.
7. Design, develop and implement recursively subdivide a tetrahedron to form 3D sierpinski gasket. The number of recursive steps is to be specified by the user.
8. Develop a menu driven program to animate a flag using Bezier Curve algorithm
9. Develop a menu driven program to fill the polygon using scan line algorithm

PART B - MINI-PROJECT

Student should develop mini project on the topics mentioned below or similar applications using Open GL API. Consider all types of attributes like color, thickness, styles, font, background, speed etc., while doing mini project.

(**During the practical exam: the students should demonstrate and answer Viva-Voce**)

Sample Topics: Simulation of concepts of OS, Data structures, algorithms etc.

Note:

1. All laboratory experiments from part A are to be included for practical examination.
2. Mini project has to be evaluated for 60 Marks.
3. Report should be prepared in a standard format prescribed for project work.
4. Students are allowed to pick one experiment from the lot.
5. Strictly follow the instructions as printed on the cover page of answer script.
6. Marks distribution:

Part A – Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks

Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks

Change of experiment is allowed only once and marks allotted to the procedure part to be made zero.

Introduction to OpenGL

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that you use to specify the objects and operations needed to produce interactive three-dimensional applications. OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. With OpenGL, you can build up your desired model from a small set of *geometric primitives* - points, lines, and polygons. A sophisticated library that provides these features could certainly be built on top of OpenGL. The OpenGL Utility Library (GLU) provides many of the modeling features. GLU is a standard part of every OpenGL implementation.

OpenGL-Related Libraries

OpenGL provides a powerful but primitive set of rendering commands, and all higher-level drawing must be done in terms of these commands. Also, OpenGL programs have to use the underlying mechanisms of the windowing system. A number of libraries exist to allow you to simplify your programming tasks, including the following:

- The OpenGL Utility Library (GLU) contains several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections, performing polygon tessellation, and rendering surfaces. This library is provided as part of every OpenGL implementation. GLU routines use the prefix **glu**.
- The OpenGL Utility Toolkit (GLUT) is a window system-independent toolkit. It contains rendering commands but is designed to be independent of any window system or operating system. Consequently, it contains no commands for opening windows or reading events from the keyboard or mouse. Since OpenGL drawing commands are limited to those that generate simple geometric primitives (points, lines, and polygons), GLUT includes several routines that create more complicated three-dimensional objects such as a sphere, a torus, and a teapot. GLUT may not be satisfactory for full-featured OpenGL applications, but you may find it a useful starting point for learning OpenGL.

Include Files

For all OpenGL applications, you want to include the gl.h header file in every file. Almost all OpenGL applications use GLU, the aforementioned OpenGL Utility Library, which requires inclusion of the glu.h header file. So almost every OpenGL source file begins with

#include <GL/gl.h>

#include <GL/glu.h>

If you are using GLUT for managing your window manager tasks, you should include

#include <GL/glut.h>

Note that glut.h includes gl.h, glu.h automatically, so including all three files is redundant.

OpenGL Primitives

Value	Meaning
GL_POINTS	individual points
GL_LINES	pairs of vertices interpreted as individual line segments
GL_POLYGON	boundary of a simple, convex polygon
GL_TRIANGLES	triples of vertices interpreted as triangles
GL_QUADS	quadruples of vertices interpreted as four-sided polygons
GL_LINE_STRIP	series of connected line segments
GL_LINE_LOOP	same as above, with a segment added between last and first vertices
GL_TRIANGLE_STRIP	linked strip of triangles
GL_TRIANGLE_FAN	linked fan of triangles
GL_QUAD_STRIP	linked strip of quadrilaterals

Run the following commands to install OpenGLon Ubuntu.

```
$ sudo apt-get update
$ sudo apt-get install libglu1-mesa-dev freeglut3-dev mesa-
common-dev
```

Sample programs

1. Program to create a basic Open GL window

```
#include<GL/glut.h>
void display (void)
{
glClearColor (0.0,0.0,0.0,1.0);
glClear (GL_COLOR_BUFFER_BIT);
glLoadIdentity ( );
gluLookAt (0.0,0.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);
glFlush ( );
}
int main (int argc, char **argv)
{
glutInit (&argc, argv);
glutInitDisplayMode (GLUT_SINGLE);
glutInitWindowSize (500,500);
glutInitWindowPosition (100,100);
glutCreateWindow ("A basic open GL window");
glutDisplayFunc (display);
glutMainLoop ( );
return 0;
}
```

2. Program to draw/display point in OpenGL

```
#include<GL/glut.h>
#include<stdlib.h>
voidmyInit(void)
{
glClearColor(2.0,2.0,2.0,4.0);
glColor3f(0.0f,0.0f,0.0f);
glPointSize(4.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,640.0,0.0,480.0);
}
void display(void)
{
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_POINTS);
glVertex2i(100,200);
glVertex2i(400,200);
```

```
glVertex2i(200,100);
glVertex2i(200,400);
glEnd();
glFlush();
}
void main(intargc, char** argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(100,150);
glutCreateWindow("My First Attempt");
glutDisplayFunc(display);
myInit();
glutMainLoop();
}
```

3. Program to implement horizontal and vertical lines

```
#include<GL/glut.h>
#include<stdlib.h>
void myInit(void)
{
glClearColor(2.0,2.0,2.0,4.0);
glColor3f(0.0f,0.0f,0.0f);
glLineWidth(4.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,640.0,0.0,480.0);
}
Void drawLineInt(GLint x1,GLint y1,GLint x2,GLint y2)
{
glBegin(GL_LINES);
glVertex2i(x1,y1);
glVertex2i(x2,y2);
glEnd();
}
void display(void)
{
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_LINES);
glVertex2i(100,200);
glVertex2i(400,200);
glVertex2i(200,100);
glVertex2i(200,400);
glEnd();
```

```
glFlush();
}
void main(int argc, char** argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(100,150);
glutCreateWindow("My First Attempt");
glutDisplayFunc(display);
myInit();
drawLineInt(100,200,40,60);
glutMainLoop();
}
```

4. Program to create keyboard interface & window sizing

```
#include<GL/glut.h>
void display()
{
glClearColor(1.0,1.0,1.0,0.0);
glClear(GL_COLOR_BUFFER_BIT);
glFlush();
}
void keyboard(unsigned char key, int x, int y)
{ /* called when a key is pressed */
if(key==27) exit(0); /* 27 is the escape */
}
int main(int argc, char **argv)
{
glutInit(&argc, argv); /* Initialize OpenGL*/
glutInitWindowSize(500,500); /*Set Window size*/
glutInitWindowPosition(10,10); /*Set Window Position*/
glutCreateWindow("Hai"); /* Create the window*/
glutDisplayFunc(display);
glutKeyboardFunc(keyboard);
glutMainLoop();
}
```

5. Program to rotate a cube

```
#include<GL/glut.h>
GLfloat angle=0.0;
void spin(void)
{
Angle+=1.0;
glutPostRedisplay()
}
void display(void)
{
glClear(GL_COLOR_BUFFER_BIT);
glLoadIdentity();
gluLookAt(0.0,0.0,0.5,0.0,0.0,0.0,0.0,1.0,0.0);
glRotatef(angle,1,0,0);
glRotatef(angle,0,1,0);
glRotatef(angle,0,0,1);
glutWireCube(2.0);
}
void reshape(int width, int height)
{
glViewport(0,0,(GLsizei)width, (GLsizei)height);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(60, (GLfloat)width / (GLfloat)height,1.0,100);
glMatrixMode(GL_MODELVIEW);
}
int main(int argc,char **argv)
{
glutInit(&argc,argv);
glutInitWindowSize(500,500);
glutInitWindowPosition(100,100);
glutCreateWindow("Rotating Cube");
glutDisplayFunc(display);
glutReshape(reshape);
gltuIdleFunc(spin);
glutMainLoop();
}
```

PART A

1. Implement Brenham's line drawing algorithm for all types of slope

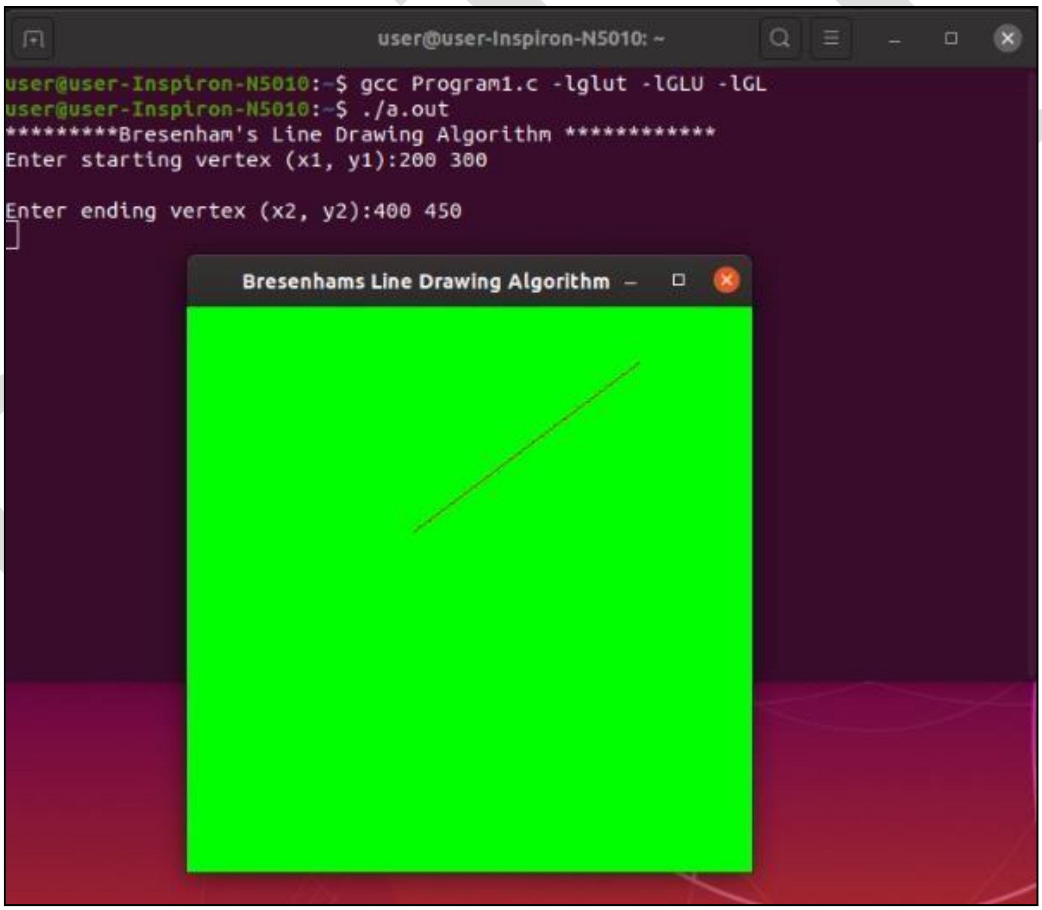
```
#include <GL/glut.h>
#include<math.h>
#include<stdio.h>

GLint xOne, yOne, xTwo, yTwo;

void init();
void setPixel(GLint, GLint);
void lineBres_L1(GLint, GLint, GLint, GLint, GLfloat) ;
void lineBres_GE1(GLint, GLint, GLint, GLint, GLfloat);
void display();

void main(int argc, char**argv)
{
printf("*********Bresenham's Line Drawing Algorithm********");
printf("\nEnter starting vertex (x1, y1):");
scanf("%d%d",&xOne, &yOne);
printf("\nEnter ending vertex (x2, y2):");
scanf("%d%d",&xTwo, &yTwo);
glutInit(&argc,argv);//initialize GLUT
//initialize display mode
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(400,400); //set display-window width & height
//set display-window upper-left position
glutInitWindowPosition(200,200);
glutCreateWindow("Bresenhams Line Drawing Algorithm");
//create display-window with a title
init();
//call graphics to be displayed on the window
glutDisplayFunc(display);
glutMainLoop(); //display everything and wait
}


void init()
{
glClearColor(0.0, 1.0, 0.0, 0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,500,0.0,500);
}
```

```
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
GLfloat m;
m=(float)(yTwo-yOne)/(xTwo-xOne);
//compute slope
//call required function based on value of slope
if(fabs(m)>=1)
lineBres_GE1(xOne,yOne,xTwo,yTwo,m);
else
lineBres_L1(xOne, yOne, xTwo,yTwo, m);
}


//Bresenham line-drawing procedure for |m| < 1.0
void lineBres_L1(GLint x0, GLint y0, GLint xEnd, GLint yEnd,
GLfloat m)
{
GLint dx = abs(xEnd - x0);
GLint dy = abs(yEnd - y0);
GLint p = 2 * dy - dx;
GLint twoDy = 2 * dy;
GLint twoDyMinusDx = 2 * (dy-dx);
GLint x=x0,y=y0;
// determine which point to use as start position
if (x0 > xEnd)
{
     x = xEnd;
     y = yEnd;
     xEnd = x0;
}
else
{
     x = x0;
     y = y0;
}
setPixel(x,y);
while(x<xEnd)
{
     x++;
     if(p<0)
          p += twoDy;
          else
          {
               if(m<0)
                    y--;
```

```
               else
                    y++;
                    p += twoDyMinusDx;
          }
     setPixel(x,y);
}
}
//Bresenham line-drawing procedure for |m| >= 1.0
void lineBres_GE1(GLint x0, GLint y0, GLint xEnd, GLint yEnd,
GLfloat m)
{
GLint dx = abs(xEnd - x0);
GLint dy = abs(yEnd - y0);
GLint p=2*dx-dy;
GLint twoDx = 2*dx;
GLint twoDxMinusDy=2*(dx-dy);
GLint x=x0,y=y0;
// determine which point to use as start position
if (y0 > yEnd)
{
     x = xEnd;
     y = yEnd;
     yEnd = y0;
}
else
{
     x = x0;
     y = y0;
}
setPixel(x,y);
while(y<yEnd)
{
     y++;
     if(p<0)
     p+=twoDx;
     else
     {
          if(m<0)
               x--;
          else
               x++;
               p+=twoDxMinusDy;
     }
     setPixel(x,y);
}
}
```

```
void setPixel(GLint xCoordinate, GLint yCoordinate)
{
glColor3f(1.0, 0.0, 0.0);
glBegin(GL_POINTS);
     glVertex2i(xCoordinate,yCoordinate);
glEnd();
//executes all OpenGL functions as quickly as possible
glFlush();
}
```

OUTPUT

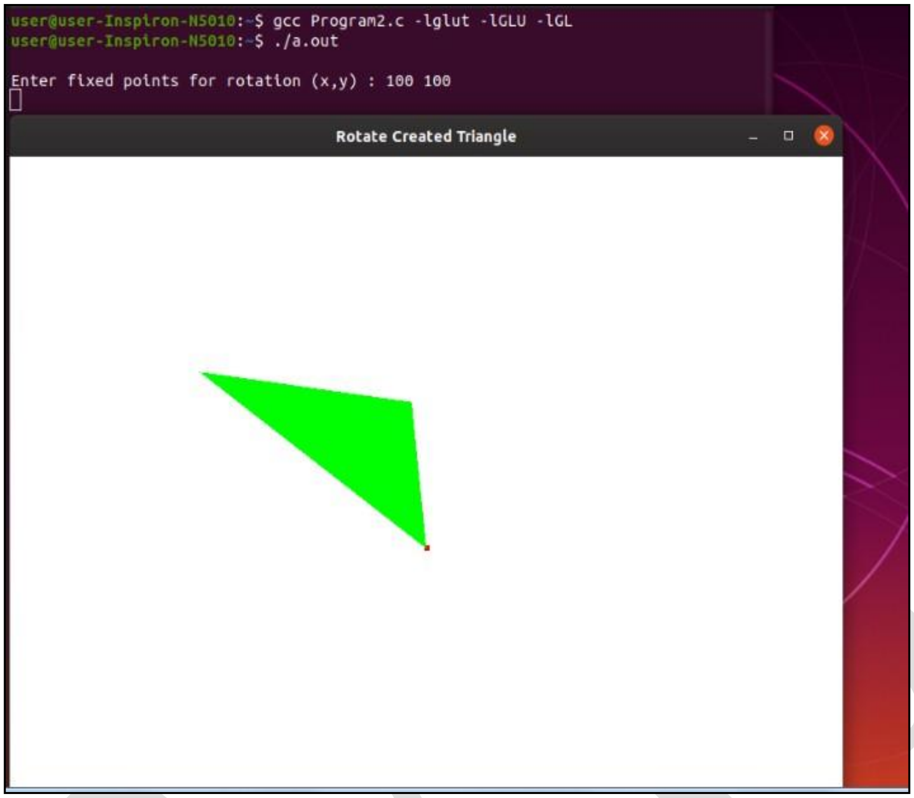


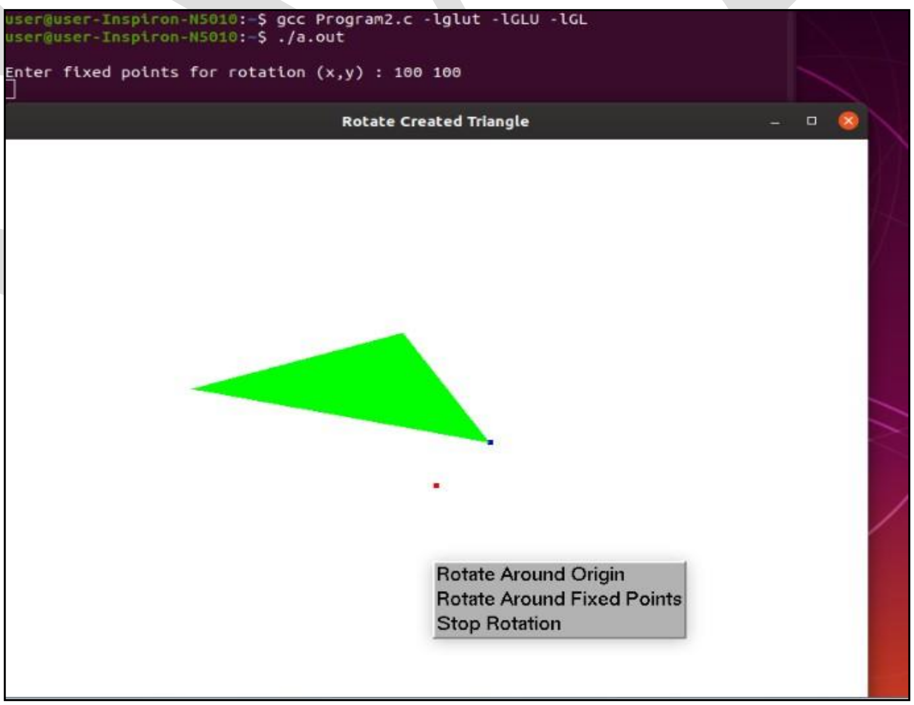
2. Create and rotate a triangle about the origin and a fixed point

```
#include<stdio.h>
#include<GL/glut.h>
int x,y;
int where_to_rotate=0;
float translate_x=0.0,translate_y=0.0,rotate_angle=0.0;

void draw_pixel(float x1,float y1)
{
     glPointSize(5.0);
     glBegin(GL_POINTS);
          glVertex2f(x1,y1);
     glEnd();
}
void triangle(int x,int y)
{
     // set interior color of triangle to green
     glColor3f(0.0,1.0,0.0);
     glBegin(GL_POLYGON);
          glVertex2f(x,y);
          glVertex2f(x+400,y+400);
          glVertex2f(x+300,y+0);
     glEnd();
     glFlush();
}
void display()
{
     glClear(GL_COLOR_BUFFER_BIT);
     glLoadIdentity();
     glColor3f(1.0,0.0,0.0); //color of point
     draw_pixel(0.0,0.0);
     if(where_to_rotate==1)
     {
          translate_x=0.0;
          translate_y=0.0;
          rotate_angle+=0.9;
     }
     if(where_to_rotate==2)
     {
          translate_x=x;
          translate_y=y;
          rotate_angle+=0.9;
          glColor3f(0.0,0.0,1.0);
          draw_pixel(x,y);
     }
```

```
	glTranslatef(translate_x,translate_y,0.0);
	glRotatef(rotate_angle,0.0,0.0,1.0);
	glTranslatef(-translate_x,-translate_y,0.0);
	triangle(translate_x,translate_y);
	glutPostRedisplay();
	glutSwapBuffers();
}
void init()
{
	glClearColor(1.0,1.0,1.0,1.0); //background color to white
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(-800.0,800.0,-800.0,800.0);
	glMatrixMode(GL_MODELVIEW);
}
void rotate_menu(int option)
{
	if(option==1)
		where_to_rotate=1;
	if(option==2)
		where_to_rotate=2;
	if(option==3)
		where_to_rotate=3;
	display();
}
int main(int argc,char **argv)
{
	printf("\nEnter fixed points for rotation (x,y) : ");
	scanf("%d%d",&x,&y);
	glutInit(&argc,argv);
	glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
	glutInitWindowSize(800,800);
	glutInitWindowPosition(0,0);
	glutCreateWindow("Rotate Created Triangle");
	init();
	glutDisplayFunc(display);
	glutCreateMenu(rotate_menu);
		glutAddMenuEntry("Rotate Around Origin",1);
		glutAddMenuEntry("Rotate Around Fixed Points",2);
		glutAddMenuEntry("Stop Rotation",3);
	glutAttachMenu(GLUT_RIGHT_BUTTON);
	glutMainLoop();
}
```

OUTPUT


user@user-Inspiron-N5010:~$ gcc Program2.c -lglut -lGLU -lGL
user@user-Inspiron-N5010:~$ ./a.out
Enter fixed points for rotation (x,y) : 100 100
Rotate Created Triangle


user@user-Inspiron-N5010:~$ gcc Program2.c -lglut -lGLU -lGL
user@user-Inspiron-N5010:~$ ./a.out
Enter fixed points for rotation (x,y) : 100 100
Rotate Created Triangle
Rotate Around Origin
Rotate Around Fixed Points
Stop Rotation

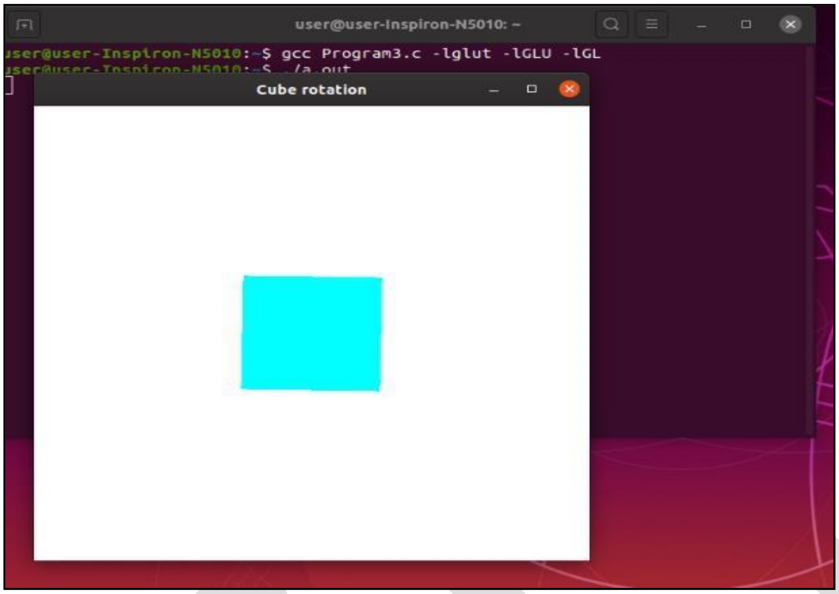
3. Draw a color cube and spin it using OpenGL transformation matrices

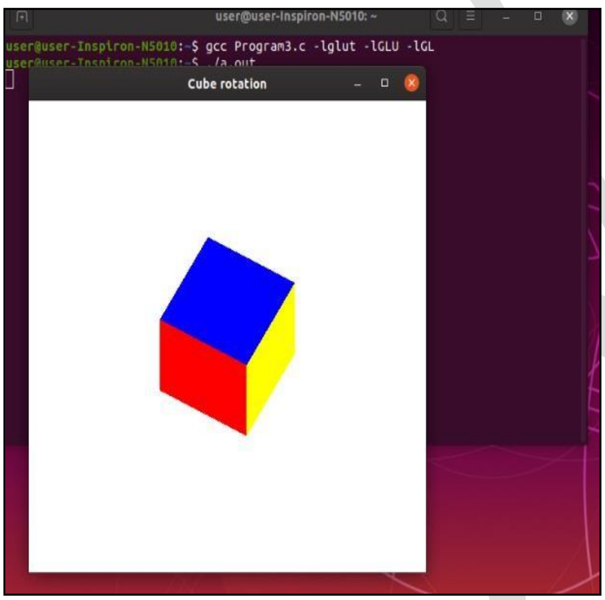
```
#include<stdio.h>
#include<math.h>
#include<GL/glut.h>

// 8 vertices of the cube with origin as its centroid
float v[][3] = { { -1,-1,-1 } ,{ -1,1,-1 } ,{ 1,1,-1 } ,{ 1,-1,-
1 } ,{ -1,-1,1 } ,{ -1,1,1 } ,{ 1,1,1 } ,{ 1,-1,1 } };

int t[] = { 0,0,0 };  // degree of rotation along {x,y,z}
int ax = 2; // axis of rotation

void init()
{
     glMatrixMode(GL_PROJECTION);
     glOrtho(-4, 4, -4, 4, -10, 10);
     // location where your object gets modelled
     glMatrixMode(GL_MODELVIEW);  }

// function used to draw one face of a cube at a time
void polygon(int a, int b, int c, int d)
{
     glBegin(GL_POLYGON);// draw the square using polygon
     // 4 coordinates of the square face
     // each v[i] contains 3 values (x,y,z) which denotes a
     point in 3D plane

     glVertex3fv(v[a]);
     glVertex3fv(v[b]);
     glVertex3fv(v[c]);
     glVertex3fv(v[d]);
     glEnd();
}

//function used to color each face of the cube seperately
void colorcube()
{
     glColor3f(0, 0, 1);       //color of front square
     polygon(0, 1, 2, 3);      // drawing the front square
     glColor3f(0, 1, 1);       // color of the left square
     polygon(4, 5, 6, 7);      // drawing the left square
     glColor3f(0, 1, 0);       // color of the right square
     polygon(0, 1, 5, 4);      // drawing the right square
     glColor3f(1, 0, 0);       // color of the top square
```

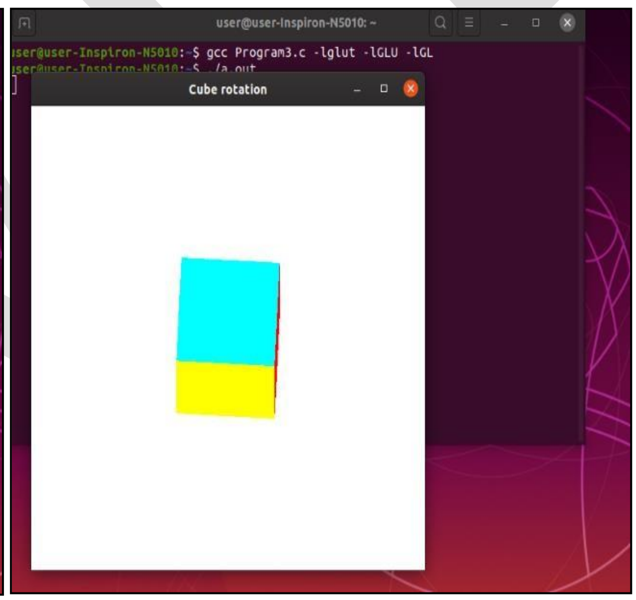
```
    polygon(2, 6, 7, 3);     // drawing the top square
    glColor3f(1, 1, 0);      // color of the bottom square
    polygon(0, 4, 7, 3);     // drawing the bottom square
    glColor3f(1, 0, 1);      // color of the back square
    polygon(1, 5, 6, 2);     // drawing the back square
}

void spincube()
{
    // rotating the cube by 1 degree at a time on the given
    axis "ax" ( ax = 0 is x axis , ax =1 is y axis , ax =2 is z
    axis)

    t[ax] += 1;
    if (t[ax] == 360)
    // when the rotation along any axis reaches 360 reset the
    axis to 0
    t[ax] -= 360;
    glutPostRedisplay();     // calling the display again..
}


// function is used to capture the events of the mouse and
rotate cube accordingly

void mouse(int btn, int state, int x, int y)
{

    // on left click, state of the left button is set to DOWN
    and ax =0
    if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    ax = 0;
    //ie rotate along x axis
    if (btn == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN)
    // on middle click, state of the middle button is set to
    DOWN .. and ax=1
    ax = 1;
    //ie rotate along y axis
    if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
    // on right click, state of the right button is set to
    DOWN.. and ax=2
    ax = 2;
    //ie rotate along z axis
}

void display() // display function
```

```
{
     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
     //clears the color buffer and depth buffer
     glClearColor(1, 1, 1, 1);
     //sets the backround screen color
     glLoadIdentity();
     //loads identity matrix into modelview
     //glrotatef(angle of rotation,x,y,z)
     glRotatef(t[0], 1, 0, 0);
     //rotate cube at an angle of t[0] degrees wrt vector(1,0,0)
     glRotatef(t[1], 0, 1, 0);
     //rotate cube at an angle of t[1] degrees wrt vector(0,1,0)
     glRotatef(t[2], 0, 0, 1);
     //rotate cube at an angle of t[2] degrees wrt vector(0,0,1)
     colorcube();
     // call the function to color each square of cube with
     different colors
     glutSwapBuffers();
     glFlush();
}

int main(int argc, char **argv)
{
     glutInit(&argc, argv);
     glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
     glutInitWindowPosition(100, 100); // set window position
     glutInitWindowSize(500, 500); //set window size
     glutCreateWindow("Cube rotation");
     init();
     glutIdleFunc(spincube);
     glutMouseFunc(mouse);    // calls the mouse function...
     glutmousefunc captures your mouse activity
     glEnable(GL_DEPTH_TEST); //enabling the depth buffer
     glutDisplayFunc(display);
     glutMainLoop();
     return 0;
}
```

OUTPUT


user@user-Inspiron-N5010: ~
user@user-Inspiron-N5010:~$ gcc Program3.c -lglut -lGLU -lGL
Cube rotation


user@user-Inspiron-N5010: ~
user@user-Inspiron-N5010:~$ gcc Program3.c -lglut -lGLU -lGL
Cube rotation


user@user-Inspiron-N5010: ~
user@user-Inspiron-N5010:~$ gcc Program3.c -lglut -lGLU -lGL
Cube rotation

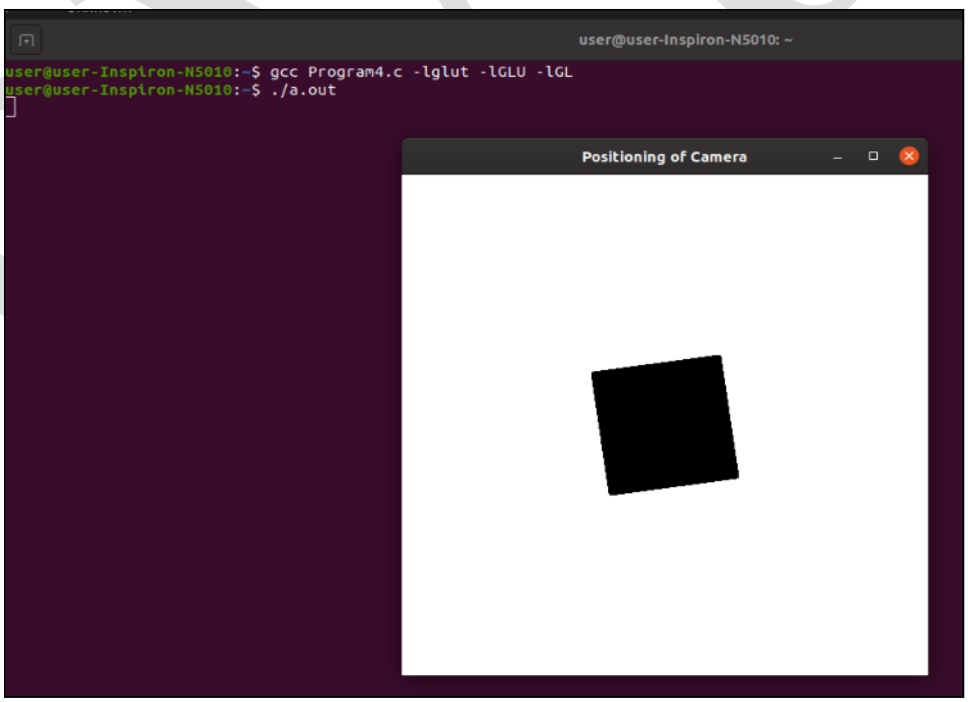
4. Draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing.

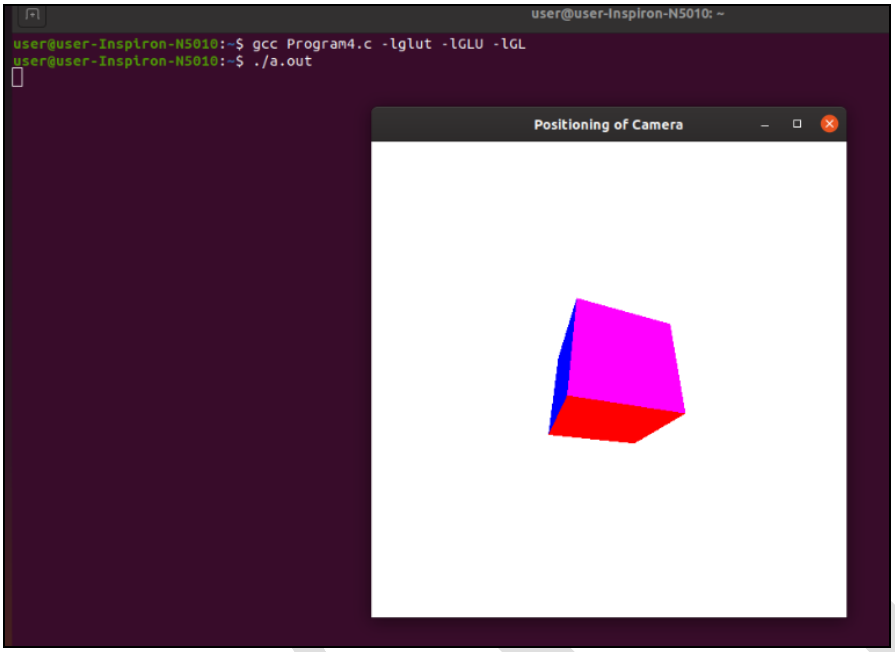
```
#include<stdio.h>
#include<math.h>
#include<GL/glut.h>

float pts[8][3] = {
                {-1,-1,-1},
                {-1,1,-1},
                {1,1,-1},
                {1,-1,-1},
                {-1,-1,1},
                {-1,1,1},
                {1,1,1},
                {1,-1,1}
               };
float theta[] ={0,0,0};
int axis = 2;
float viewer[]={5,0,0};

void myInit()
{
     glMatrixMode(GL_PROJECTION);
     glFrustum(-2,2,-2,2,2,10);
     glMatrixMode(GL_MODELVIEW);
}

void draw_polygon(int a, int b, int c, int d)
{
     glBegin(GL_QUADS);
          glVertex3fv(pts[a]);
          glVertex3fv(pts[b]);
          glVertex3fv(pts[c]);
          glVertex3fv(pts[d]);
     glEnd();
}

void draw_cube(float pts[8][3])
{
     glColor3f(0,0,1);
     draw_polygon(0,1,2,3); //front face
     glColor3f(0,1,0);
     draw_polygon(4,5,6,7);   //behind face
     glColor3f(1,0,0);
     draw_polygon(0,1,5,4);   //left face
```

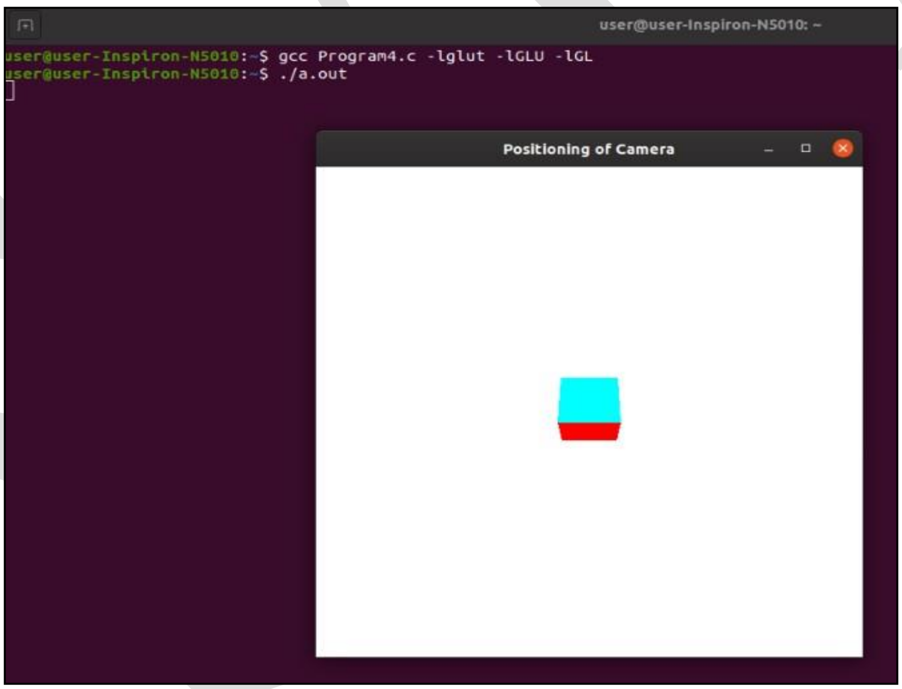
```
     glColor3f(0,0,0);
     draw_polygon(3,2,6,7);   //right face
     glColor3f(0,1,1);
     draw_polygon(0,4,7,3);   //bottom face
     glColor3f(1,0,1);
     draw_polygon(1,5,6,2);   //top face
}

void myDisplay()
{
     glClearColor(1,1,1,1);
     glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
     glLoadIdentity();
     gluLookAt(viewer[0],viewer[1],viewer[2],0,0,0,0,1,0);
     glRotatef(theta[2],0,0,1);
     glRotatef(theta[1],0,1,0);
     glRotatef(theta[0],1,0,0);
     draw_cube(pts);
     glFlush();
     glutSwapBuffers();
}

void spincube()
{
     theta[axis] = theta[axis]+4;
     if(theta[axis]>360)
          theta[axis]=0;
     glutPostRedisplay();
}

void mouse(int btn , int state , int x , int y)
{
     if((btn==GLUT_LEFT_BUTTON)&&(state==GLUT_DOWN))
          axis=0;
     if((btn==GLUT_RIGHT_BUTTON)&&(state==GLUT_DOWN))
          axis=2;
     if((btn==GLUT_MIDDLE_BUTTON)&&(state==GLUT_DOWN))
          axis=1;
     spincube();
}

void keyboard(unsigned char key, int x, int y)
{
     if(key=='X') viewer[0]+=1;
     if(key=='x') viewer[0]-=1;
     if(key=='Y') viewer[1]+=1;
```

```
	if(key=='y') viewer[1]-=1;
	if(key=='Z') viewer[2]+=1;
	if(key=='z') viewer[2]-=1;
	glutPostRedisplay();
}

int main (int argc, char ** argv)
{
	glutInit(&argc,argv);
	glutInitDisplayMode( GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
	glutInitWindowPosition(50,50);
	glutInitWindowSize(500,500);
	glutCreateWindow("Positioning of Camera");
	myInit();
	glEnable(GL_DEPTH_TEST);
	glutDisplayFunc(myDisplay);
	glutKeyboardFunc(keyboard);
	glutMouseFunc(mouse);
	glutMainLoop();
}
```

OUTPUT




user@user-Inspiron-N5010: ~
user@user-Inspiron-N5010:~$ gcc Program4.c -lglut -lGLU -lGL
user@user-Inspiron-N5010:~$ ./a.out
Positioning of Camera


user@user-Inspiron-N5010: ~
user@user-Inspiron-N5010:~$ gcc Program4.c -lglut -lGLU -lGL
user@user-Inspiron-N5010:~$ ./a.out
Positioning of Camera

5. Clip a lines using Cohen-Sutherland algorithm.

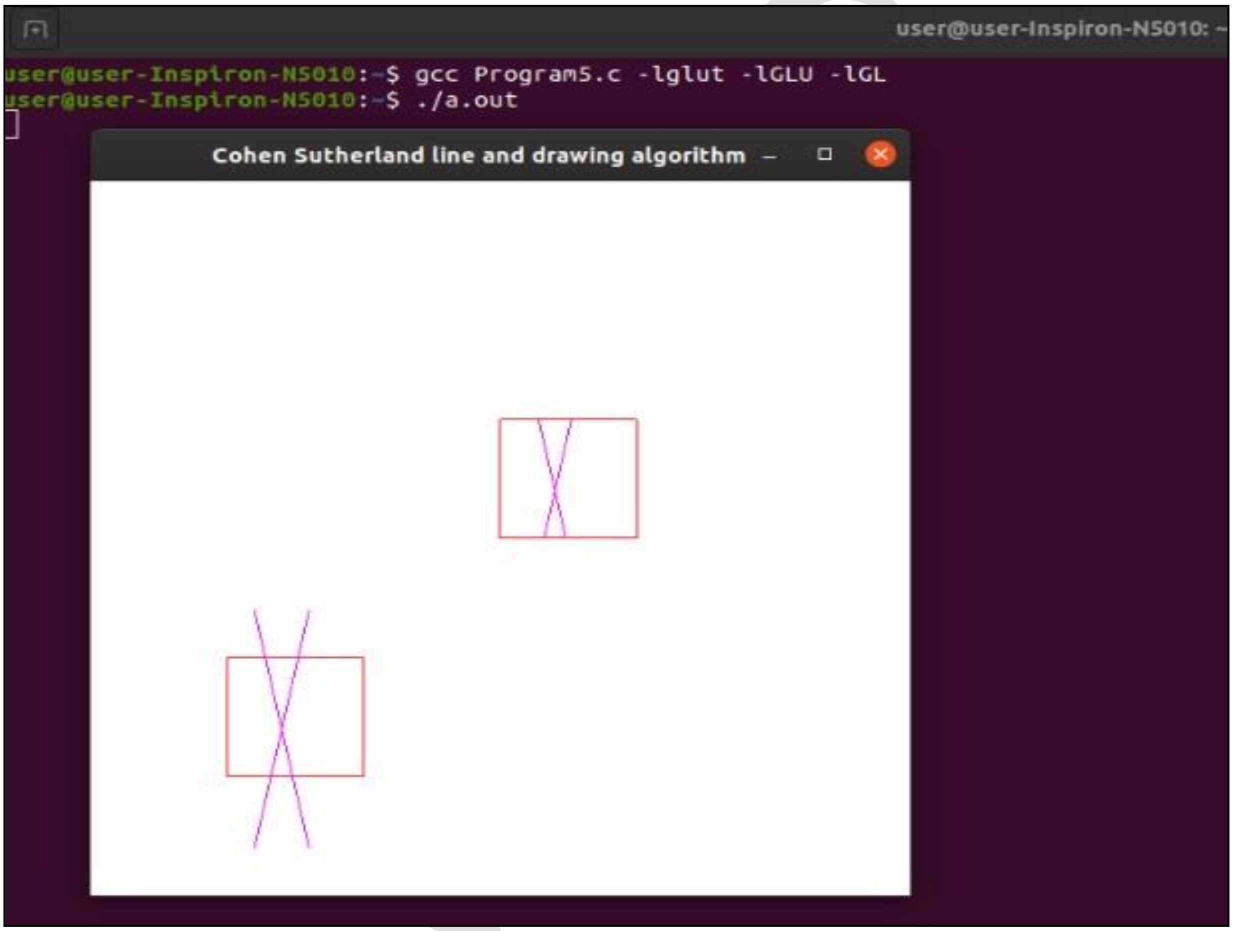
```
#include<stdio.h>
#include<GL/glut.h>

#define true 1;
#define false 0;
#define bool int;
double x,y;

int xmin=50,xmax=100,ymin=50,ymax=100;

const int RIGHT=8,LEFT=2,TOP=4,BOTTOM=1;

int outcode0,outcode1,outcodeout,done,accept;

int computeoutcode(double x,double y)
{
  int code=0;
  if(y>ymax)
    code|=TOP;
  else if(y<ymin)
    code|=BOTTOM;
  if(x>xmax)
    code|=RIGHT;
  else if(x<xmin)
    code|=LEFT;
  return code;
}


void LineClip(double x0,double y0,double x1,double y1)
{
  int accept=false;
  int done=false;
  outcode0=computeoutcode(x0,y0);
  outcode1=computeoutcode(x1,y1);
  do{
    if(!(outcode0|outcode1))
    {
      accept=true;
      done=true;
    }
    else if(outcode0&outcode1)
    {
```

```
      done=true;
    }
    else
    {
      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)
  {
    glPushMatrix();
    glTranslatef(100,100,0);
    glColor3f(1.0,0.0,0.0);
    glBegin(GL_LINE_LOOP);
    glVertex2i(50,50);
    glVertex2i(100,50);
    glVertex2i(100,100);
    glVertex2i(50,100);
    glEnd();
```

```
    glColor3f(1.0,0.0,1.0);
    glBegin(GL_LINES);
    glVertex2i(x0,y0);
    glVertex2i(x1,y1);
    glEnd();
    glPopMatrix();
    glFlush();
   }
}

void display()
{
  glClearColor(1,1,1,1);
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0,0.0,0.0);
  glBegin(GL_LINE_LOOP);
     glVertex2i(50,50);
      glVertex2i(100,50);
     glVertex2i(100,100);
     glVertex2i(50,100);
  glEnd();
  glColor3f(1.0,0.0,1.0);
  glBegin(GL_LINES);
     glVertex2i(60,20);
     glVertex2i(80,120);
     glVertex2i(80,20);
     glVertex2i(60,120);
  glEnd();
  LineClip(60,20,80,120);
  LineClip(80,20,60,120);
  glFlush();
}

void init()
{
  glMatrixMode(GL_PROJECTION);
  gluOrtho2D(0,300,0,300);
  glMatrixMode(GL_MODELVIEW);
}

int main(int argc,char** argv)
{
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
  glutInitWindowPosition(0,0);
  glutInitWindowSize(500,500);
```

```
glutCreateWindow("Cohen Sutherland line and drawing algorithm");
init();
glutDisplayFunc(display);
glutMainLoop();
}
```

OUTPUT

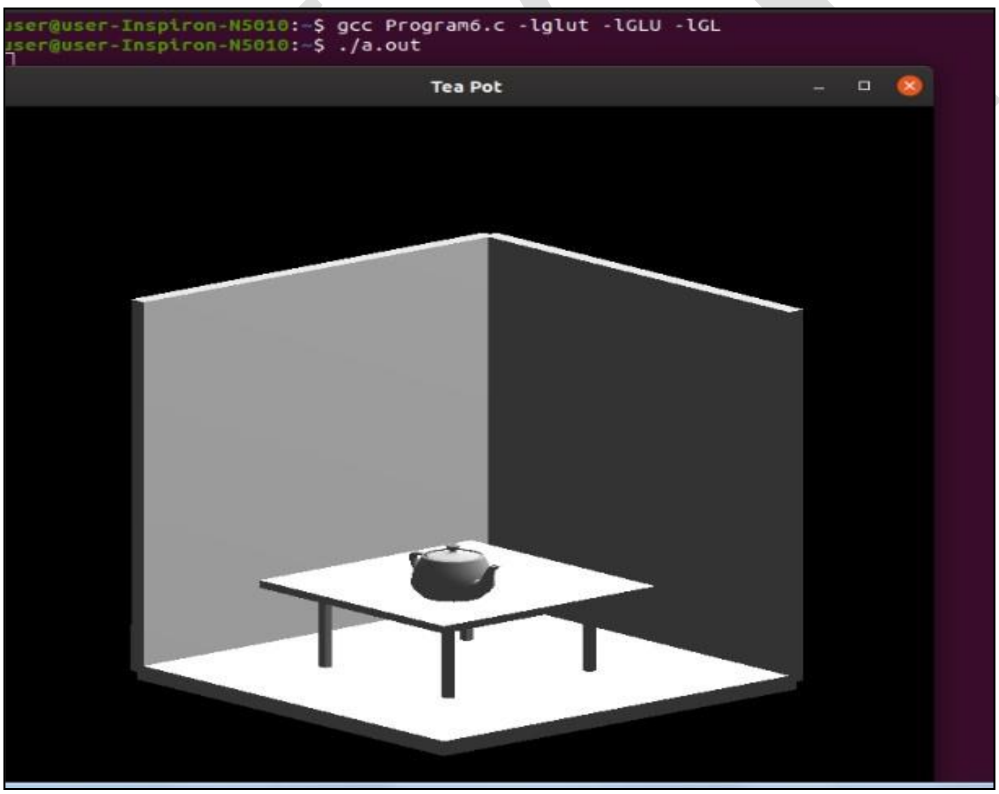


6. To draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of the light source along with the properties of the surfaces of the solid object used in the scene.

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

void obj(double tx,double ty,double tz,double sx,double
sy,double sz)
{
    glRotated(50,0,1,0);
     glRotated(10,-1,0,0);
     glRotated(11.7,0,0,-1);
     glTranslated(tx,ty,tz);
     glScaled(sx,sy,sz);
     glutSolidCube(1);
     glLoadIdentity();
}

void display()
{
     glViewport(0,0,700,700);
     glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
     obj(0,0,0.5,1,1,0.04);
     obj(0,-0.5,0,1,0.04,1);
     obj(-0.5,0,0,0.04,1,1);
     obj(0,-0.3,0,0.02,0.2,0.02);
     obj(0,-0.3,-0.4,0.02,0.2,0.02);
     obj(0.4,-0.3,0,0.02,0.2,0.02);
     obj(0.4,-0.3,-0.4,0.02,0.2,0.02);
     obj(0.2,-0.18,-0.2,0.6,0.02,0.6);
     glRotated(50,0,1,0);
     glRotated(10,-1,0,0);
     glRotated(11.7,0,0,-1);
     glTranslated(0.3,-0.1,-0.3);
     glutSolidTeapot(0.09);
     glFlush();
     glLoadIdentity();
}

int main(int argc,char **argv)
{
     glutInit(&argc,argv);
     float ambient[]={1,1,1,1};
```

```
    float light_pos[]={27,80,2,3};
    glutInitWindowSize(700,700);
    glutCreateWindow("Tea Pot");
    glutDisplayFunc(display);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glMaterialfv(GL_FRONT,GL_AMBIENT,ambient);
    glLightfv(GL_LIGHT0,GL_POSITION,light_pos);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
}
```

OUTPUT

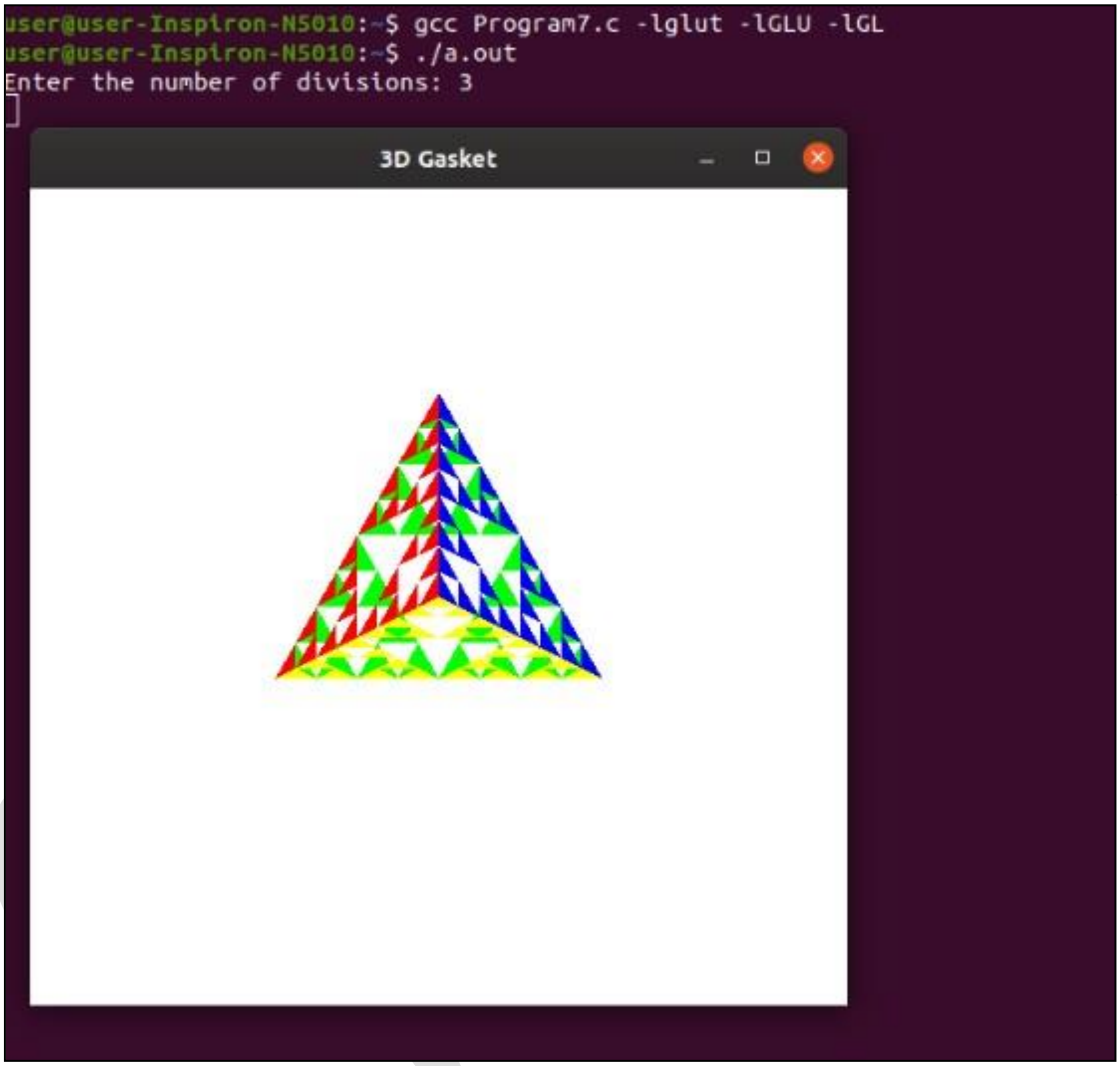


7. Design, develop and implement recursively subdivide a tetrahedron to form 3D sierpinski gasket. The number of recursive steps is to be specified by the user.

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

float v[4][3] = {
                    { 0.0,0.0,1.0 },
                    { 0,1,-1 },
                    { -0.8,-0.4,-1 },
                    { 0.8,-0.4,-1 }
               };
int n;

void triangle(float a[], float b[], float c[])
{
     glBegin(GL_POLYGON);
     glVertex3fv(a);
     glVertex3fv(b);
     glVertex3fv(c);
     glEnd();
}

void divide_triangle(float a[], float b[], float c[], int m)
{
     float v1[3], v2[3], v3[3];
     int i;
     if (m>0)
     {
          for (i = 0; i<3; i++) v1[i] = (a[i] + b[i]) / 2;
          for (i = 0; i<3; i++) v2[i] = (a[i] + c[i]) / 2;
          for (i = 0; i<3; i++) v3[i] = (b[i] + c[i]) / 2;
          divide_triangle(a, v1, v2, m - 1);
          divide_triangle(c, v2, v3, m - 1);
          divide_triangle(b, v3, v1, m - 1);
     }
     else (triangle(a, b, c));
}
```

```
void tetrahedron(int m)
{
     glColor3f(1.0, 0.0, 0.0);
     divide_triangle(v[0], v[1], v[2], m);
     glColor3f(0.0, 1.0, 0.0);
     divide_triangle(v[3], v[2], v[1], m);
     glColor3f(0.0, 0.0, 1.0);
     divide_triangle(v[0], v[3], v[1], m);
     glColor3f(1.0, 1.0, 0.0);
     divide_triangle(v[0], v[2], v[3], m);
}

void display()
{
     glMatrixMode(GL_PROJECTION);
     glLoadIdentity();
     glOrtho(-2.0, 2.0, -2.0, 2.0, -10.0, 10.0);
     glMatrixMode(GL_MODELVIEW);
     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
     tetrahedron(n);
     glFlush();
     glutPostRedisplay();
}

int main(int argc, char* argv[])
{
     printf("Enter the number of divisions: ");
     scanf("%d", &n);
     glutInit(&argc, argv);
     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
     glutInitWindowSize(500, 500);
     glutInitWindowPosition(0, 0);
     glutCreateWindow("3D Gasket");
     glutDisplayFunc(display);
     glEnable(GL_DEPTH_TEST);
     glClearColor(1.0, 1.0, 1.0, 1.0);
     glutMainLoop();
     return 0;
}
```

OUTPUT


user@user-Inspiron-N5010:~$ gcc Program7.c -lglut -lGLU -lGL
user@user-Inspiron-N5010:~$ ./a.out
Enter the number of divisions: 3
3D Gasket

8. Develop a menu driven program to animate a flag using Bezier Curve algorithm.

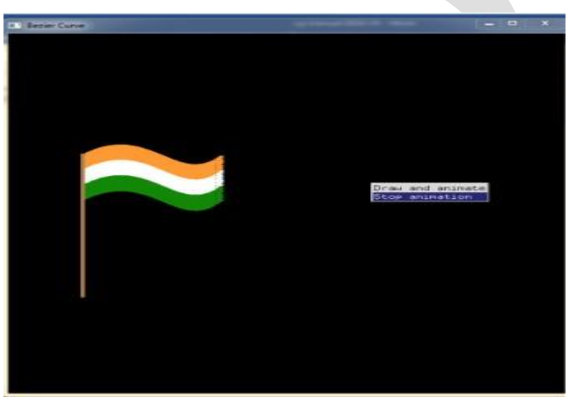
```
#include<GL/glut.h>
#include<stdio.h>
#include<math.h>
#define pi 3.1416
static float th = 0;
GLint nCP = 4, nBCP = 20;

typedef struct wc
{
GLfloat x, y, z;
};

void bino(GLint n, GLint *c)
{
GLint k, j;
for (k = 0; k <= n; k++)
{
c[k] = 1;
for (j = n; j >= k + 1; j--)
c[k] *= j;
for (j = n - k; j >= 2; j--)
c[k] /= j;
}
}

void computeBezPt(GLfloat u, wc *bP, GLint nCP, wc *cP,GLint *c)
{
GLint k, n = nCP - 1;
GLfloat BEZ;
bP->x = bP->y = bP->z = 0;
for (k = 0; k<nCP; k++)
{
BEZ = c[k] * pow(u, k)*pow(1 - u, n - k);
bP->x += cP[k].x*BEZ;
bP->y += cP[k].y*BEZ;
bP->z += cP[k].z*BEZ;
}
}
```

```
void bezier(wc *cP, GLint nCP, GLint nBCP)
{
wc bCP;
GLfloat u;
GLint *c, k;
c = new GLint[nCP];
bino(nCP - 1, c);
glBegin(GL_LINE_STRIP);
for (k = 0; k <= nBCP; k++)
{
u = GLfloat(k) / GLfloat(nBCP);
computeBezPt(u, &bCP, nCP, cP, c);
glVertex2f(bCP.x, bCP.y);
}
glEnd();
delete[]c;
}


void display()
{
     glClearColor(0, 0, 0, 1);
}

void draw_and_animate()
{
wc cP[4] = {
               { 20,100,0 },
               { 30,110,0 },
               { 50,90,0 },
               { 60,100,0 }
          };
cP[1].x += 10 * sin(th*pi / 180);
cP[1].y += 5 * sin(th*pi / 180);
cP[2].x -= 10 * sin((th + 30)*pi / 180);
cP[2].y -= 10 * sin((th + 30)*pi / 180);
cP[3].x -= 4 * sin(th*pi / 180);
cP[3].x += sin((th - 30)*pi / 180);
th += 0.1;
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1, 1, 1);
glPushMatrix();
glLineWidth(5);
```

```
glColor3f(255 / 255, 153 / 255.0, 51 / 255.0); //saffron
for (int i = 0; i<8; i++)
{
glTranslatef(0, -.8, 0);
bezier(cP, nCP, nBCP);
}

glColor3f(1, 1, 1); //white
for (int i = 0; i<8; i++)
{
glTranslatef(0, -.8, 0);
bezier(cP, nCP, nBCP);
}


glColor3f(19 / 255.0, 136 / 255.0, 8 / 255.0); //green
for (int i = 0; i<8; i++)
{
glTranslatef(0, -.8, 0);
bezier(cP, nCP, nBCP);
}

glPopMatrix();

glColor3f(.7, .5, .3); //flag pole
glLineWidth(5);
glBegin(GL_LINES);
glVertex2f(20, 100);
glVertex2f(20, 40);
glEnd();

glFlush();
glutPostRedisplay();
glutSwapBuffers();
}

void reshape(GLint w, GLint h)
{
glViewport(0, 0, w, h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0, 150, 0, 150);
glClear(GL_COLOR_BUFFER_BIT);
}
```

```
void menu(int id)
{
switch (id)
	{
	case 1:glutIdleFunc(draw_and_animate);
	break;
	case 2:glutIdleFunc(NULL);
	break;
	}
	glutPostRedisplay();
}

int main(int argc, char **argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
glutInitWindowPosition(50, 50);
glutInitWindowSize(640, 840);
glutCreateWindow("Bezier Curve");
glutReshapeFunc(reshape);
glutDisplayFunc(display);
glClearColor(0, 0, 0, 1);
glFlush();
glutCreateMenu(menu);
glutAddMenuEntry("Draw and animate", 1);
glutAddMenuEntry("Stop animation", 2);
glutAttachMenu(GLUT_LEFT_BUTTON);
glutMainLoop();
return 0;
}
```

OUTPUT



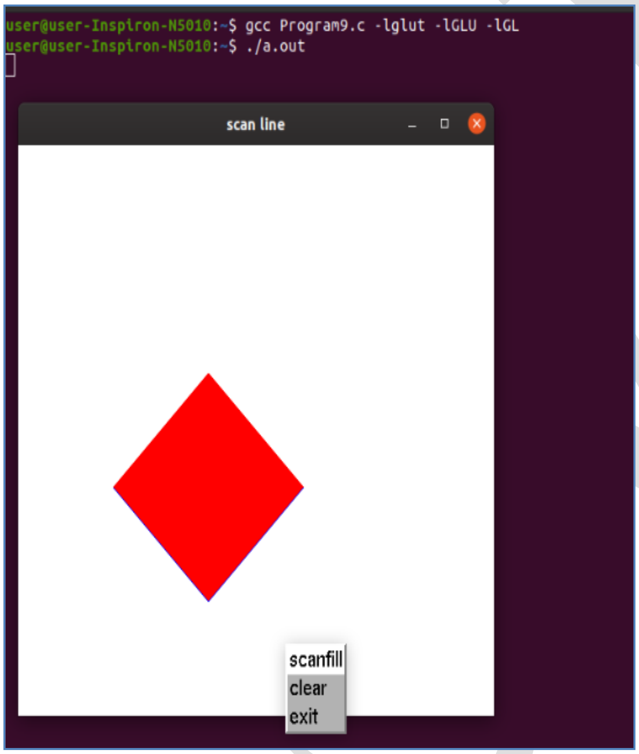
9. Develop a menu driven program to fill the polygon using scan line algorithm.

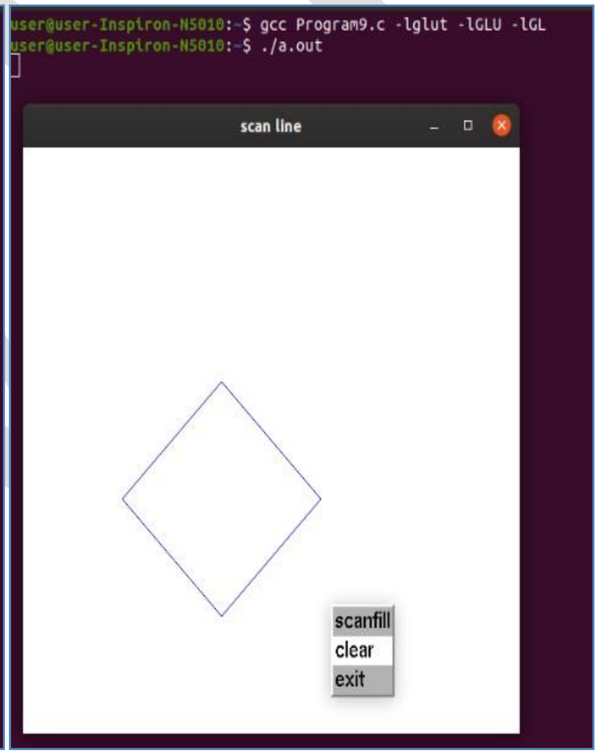
```
#include<stdio.h>
#include<math.h>

#include<GL/glut.h>
int le[500], re[500], flag=0 ,m;

void init()
{
     gluOrtho2D(0, 500, 0, 500);
}

void edge(int x0, int y0, int x1, int y1)
{
     if (y1<y0)
     {
          int tmp;
          tmp = y1;
          y1 = y0;
          y0 = tmp;
          tmp = x1;
          x1 = x0;
          x0 = tmp;
     }
     int x = x0;
     m = (y1 - y0) / (x1 - x0);
     for (int i = y0; i<y1; i++)
     {
          if (x<le[i])
               le[i] = x;
          if (x>re[i])
               re[i] = x;
          x += (1 / m);
     }
}

void display()
{
     glClearColor(1, 1, 1, 1);
     glClear(GL_COLOR_BUFFER_BIT);
     glColor3f(0, 0, 1);
     glBegin(GL_LINE_LOOP);
     glVertex2f(200, 100);
     glVertex2f(100, 200);
```

```
	glVertex2f(200, 300);
	glVertex2f(300, 200);
	glEnd();
	for (int i = 0; i<500; i++)
	{
		le[i] = 500;
		re[i] = 0;
	}
	edge(200, 100, 100, 200);
	edge(100, 200, 200, 300);
	edge(200, 300, 300, 200);
	edge(300, 200, 200, 100);
	if (flag == 1)
	{
		for (int i = 0; i < 500; i++)
		{
			if (le[i] < re[i])
			{
				for (int j = le[i]; j < re[i]; j++)
				{
					glColor3f(1, 0, 0);
					glBegin(GL_POINTS);
					glVertex2f(j, i);
					glEnd();
				}
			}
		}
	}
	glFlush();
}

void ScanMenu(int id)
{
	if (id == 1) {
		flag = 1;
	}
	else if (id == 2) {
		flag = 0;
	}
	else { exit(0); }
	glutPostRedisplay();
}

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

```
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(500, 500);
    glutCreateWindow("scan line");
    init();
    glutDisplayFunc(display);
    glutCreateMenu(ScanMenu);
    glutAddMenuEntry("scanfill", 1);
    glutAddMenuEntry("clear", 2);
    glutAddMenuEntry("exit", 3);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    glutMainLoop();
    return 0;
}
```

OUTPUT





VIVA QUESTIONS

1. Define Computer graphics.
2. Define Computer Animation?
3. Define Pixel?
4. Define Raster graphics?
5. Define Image?
6. Define Rendering?
7. Define Projection?
8. Define 3D Projection?
9. Define Grayscale?
10. Define Image Resolution?
11. Define WCS?
12. Write the two techniques for producing color displays with a CRT?
13. What is resolution?
14. What is pixel map?
15. Write the types of clipping?
16. What is persistence?
17. What is frame buffer?
18. Define Circle?
19. What are the various attributes of a line?
20. What is anti-aliasing?
21. What is Transformation?
22. What is translation?
23. What is rotation?
24. What is scaling?
25. What is shearing?
26. What is reflection?
27. What are the two classifications of shear transformation?
28. Distinguish between window port & view port?
29. Define clipping?
30. What is the need of homogeneous coordinates?
31. What is rasterization?
32. Differentiate plasma panel display and thin film electro luminescent display?
33. Write The Important Applications of Computer Graphic?
34. What Is Scan Conversion?
35. What Is Pix Map?
36. Where The Video Controller Is Used?
37. What is OpenGL?
38. What are the advantages of OpenGL over other API's?
39. How Display lists are implemented in OpenGL?