VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Belgaum, Karnataka-590 014

Laboratory Manual

Computer Graphics and Visualization

Laborator

[17CSL68/18CSL67]

Compiled by

1. Prof. Varalakshmi. B. D **2. Dhanya Jayan**

3. Swathi Mohan

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

(ACCREDITED BY NBA)

ACHARYA INSTITUTE OF TECHNOLOGY

Soldevanahalli, Bengaluru-560107

2020-2021

Table of contents

Vision of the Institute

Acharya Institute of Technology, committed to the cause of value-based education in all disciplines, envisions itself as a fountainhead of innovative human enterprise, with inspirational initiatives for Academic Excellence.

Mission of the institute

Acharya Institute of Technology strives to provide excellent academic ambiance to the students for achieving global standards of technical education, foster intellectual and personal development, meaningful research and ethical service to sustainable societal needs.

Vision, Mission of the Department

- **Vision of the Department**

Envisions to be recognized for quality education and research in the field of Computing, leading to creation of competent engineers, who are innovative and adaptable to the changing demands of industry and society

- **Mission of the Department:**
  - Act as a nurturing ground for young computing aspirants to attain the excellence by imparting quality education and professional ethics
  - Collaborate with industries and provide exposure to latest tools/ technologies.
  - Create an environment conducive for research and continuous learning.

Program Educational Objectives (PEOs)

Students shall

- Have a successful career in academia, R&D organizations, IT industry or pursue higher studies in specialized field of Computer Science and Engineering and allied disciplines.
- Be competent, creative and a valued professional in the chosen field
- Engage in life-long learning, professional development and adapt to the working environment quickly
- Become effective collaborators and exhibit high level of professionalism by leading or participating in addressing technical, business, environmental and societal challenges.

Program Specific Outcomes:

PSO	Statements
	Students shall
PSO-1:	Apply the knowledge of hardware, system software, algorithms, networking and data bases.
PSO-2:	Design, analyze and develop efficient, Secure algorithms using appropriate data structures, databases for processing of data.
PSO-3:	Be Capable of developing stand alone, embedded and web-based solutions having easy to operate interface using Software Engineering practices and contemporary computer programming languages.

Programme Outcomes

Engineering Graduates will be able to:

a. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
b. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
c. 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.
d. 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.
e. 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.

f. 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.
g. 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.
h. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
i. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
j. 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.
k. 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.
l. 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.

Course Outcomes:

The student will be able to

CO1: Implement computer graphics primitives and algorithms using OpenGL API's.

CO2: Write programs on 2D/3D objects using modelling, transformation and illumination concepts.

CO3: Design graphics applications using OpenGL API's.

CO4: Demonstrate technical information by means of oral presentations and written reports/records as a team.

Introduction to Computer Graphics
Introduction to Open GL

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 as a State Machine

OpenGL is a state machine. It is called a state machine because it can be put into various states until you change them. As you've already seen, the current color is a state variable. You can set the current color to white, red, or any other color, and thereafter every object is drawn with that color until you set the current color to something else.

The current color is only one of many state variables that OpenGL maintains. Others control such things as the current viewing and projection transformations; line and polygon stipple patterns, polygon drawing modes, pixel-packing conventions, positions and characteristics of lights, and material properties of the objects being drawn. Many state variables refer to modes that are enabled or disabled with the command **glEnable()** or **glDisable()**. Each state variable or mode has a default value, and at any point you can query the system for each variable's current value.

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 Hierarchy

Several levels of abstraction are provided

GL

- Lowest level: vertex, matrix manipulation
- glVertex3f(point.x, point.y, point.z)

GLU

- Helper functions for shapes, transformations
- gluPerspective(fovy, aspect, near, far)

- gluLookAt(0, 0, 10, 0, 0, 0, 0, 1, 0);

GLUT

- Highest level: Window and interface management
- glutSwapBuffers()
- glutInitWindowSize (500, 500);

OpenGL API

- As a programmer, you need to do the following things:

  Specify the location/parameters of camera.
- Specify the geometry (and appearance).
- Specify the lights (optional).

OpenGL: Camera

Two things to specify:

- Physical location of camera in the scene (MODELVIEW matrix in OpenGL).
- Projection properties of the camera (PROJECTION matrix in OpenGL):

void glFrustum(GLdouble left, GLdouble right, GLdouble bottom,GLdouble top, GLdouble near, GLdouble far);

void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);

OpenGL Conventions

- Many functions have multiple forms:
  - glVertex2f, glVertex3i, glVertex4dv, etc.
- Number indicates number of arguments
- Letters indicate type
  - f: float, d: double, ub: unsigned byte, etc.
- V (if present) indicates a single pointer argument

Event Loop

- OpenGL programs often run in an event loop:
  - Start the program
  - Run some initialization code
  - Run an infinite loop and wait for events such as
    - Key press
    - Mouse move, click
    - Reshape window
    - Expose event

•

OpenGL Command Syntax (1)

- OpenGL commands start with “gl”
- OpenGL constants start with “GL_”
- Some commands end in a number and one, two or three letters at the end (indicating number and type of arguments)
- A Number indicates number of arguments
- Characters indicate type of argument

OpenGL Command Syntax (2)

- glClearColor() – Specifies the background color
- glClear() – Erases the output with background color
- glMatrixMode() – Chooses projection/modelview matrix
- glBegin()/glEnd() – Model data pumped within this block
- glVertex() – Pumps vertex data into OpenGL
- glViewport() – Resizes the OpenGL viewport
- glOrtho() – Specifies orthogonal view volume
- glPolygonMode() – Specifies whether to draw filled polygons or wire-frame polygon.

OpenGL Program Organization

- **main:**
  - find GL visual and create window
  - initialize GL states (e.g. viewing, color, lighting)
  - initialize display lists
  - loop
    - check for events (and process them)
    - if window event (window moved, exposed, etc.)

- modify viewport, if needed
- redraw
- else if mouse or keyboard
- do something, e.g., change states and redraw

- **redraw:**
  - clear screen (to background color)
  - change state(s), if needed
  - render some graphics
  - change more states
  - render some more graphics

glMatrixMode

- glMatrixMode
  - - specify which matrix is the current matrix
- C Specification
  - void glMatrixMode(GLenum *mode*)
- Parameters
  - *mode* Specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted: GL_MODELVIEW, GL_PROJECTION, and GL_TEXTURE. The default value is GL_MODELVIEW.
- Description
  - glMatrixMode sets the current matrix mode. *mode* can assume one of three values: GL_MODELVIEW Applies subsequent matrix operations to the modelview matrix stack. GL_PROJECTION Applies subsequent matrix operations to the projection matrix stack.

OpenGL 3D Viewing Functions

Viewing-transformation function

- glMatrixMode (GL_MODELVIEW);
- gluLookAt(x0,y0,z0,xref,yref,zref,vx,vy,vz);
- Default: gluLookAt(0,0,0, 0,0,-1, 0,1,0);
- OpenGL orthogonal-projection function
- glMatrixMode(GL_PROJECTION);
- gluOrtho(xwmin,xwmax, ywmin,ywmax, dnear,dfar);
- Default: gluOrtho(-1,1, -1,1, -1,1);
- Note that
    - dnear and dfar must be assigned positive values
    - znear=-dnear and zfar=-dfar
    - The near clipping plane is the view plane

PROGRAMS

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

ALGORITHM:

Step 1 - Input the two end-points of line, storing the left end-point in (x0,y0). Step 2 - Plot the point (x0,y0).

Step 3 - Calculate the constants dx, dy, 2dy, and (2dy – 2dx) and get the first value for the decision parameter as -

p0=2dy-dx

Step 4 - At each Xk along the line, starting at k = 0, perform the following test -

If pk< 0, the next point to plot is (xk+1,yk) and pk+1=pk+2dy

Otherwise, (xk,yk+1)

pk+1=pk+2dy-2dx

Step 5 - Repeat step 4 (dx – 1) times.

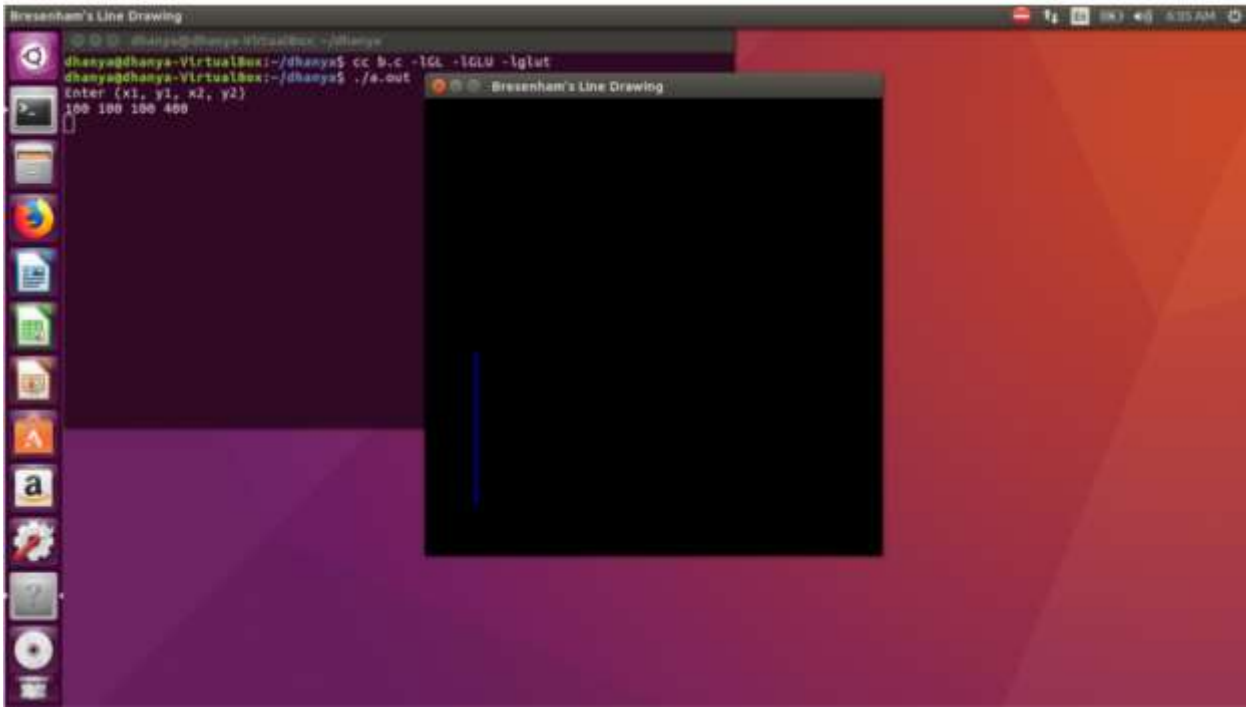
For m > 1, find out whether you need to increment x while incrementing y each time.

After solving, the equation for decision parameter Pk
will be very similar, just the x and y in the equation gets interchanged.

PROGRAM:

```
#include<GL/glut.h>
#include<stdio.h>
#include<stdlib.h>
int x1, y11, x2, y2;
void myInit()
{
      glClearColor(0.0, 0.0, 0.0, 1.0);
         glClear(GL_COLOR_BUFFER_BIT);
      glMatrixMode(GL_PROJECTION);
      gluOrtho2D(0, 900, 0, 900);


}
void draw_pixel(int x, int y)
{
      glEnable(GL_POINT_SMOOTH);
      glPointSize(2.0f);
      glBegin(GL_POINTS);
      glVertex2i(x, y);
      glColor3f(0.0,0.0,1.0);
      glEnd();
}

void draw_line(int x1, int x2, int y11, int y2)
{

      int dx, dy, i, e;
      int incx, incy, inc1, inc2;
      int x,y;
      dx = x2-x1;
      dy = y2-y11;
      if (dx < 0)
            dx = -dx;
      if (dy < 0)
            dy = -dy;
      incx = 1;
      if (x2 < x1)
            incx = -1;
      incy = 1;
      if (y2 < y11)
            incy = -1;
      x = x1; y = y11;
      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, y11, y2);
      glFlush();
}
int main(int argc, char **argv)
{
      printf( "Enter (x1, y1, x2, y2)\n");
      scanf("%d %d %d %d", &x1, &y11, &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();
         return 0;
}
```

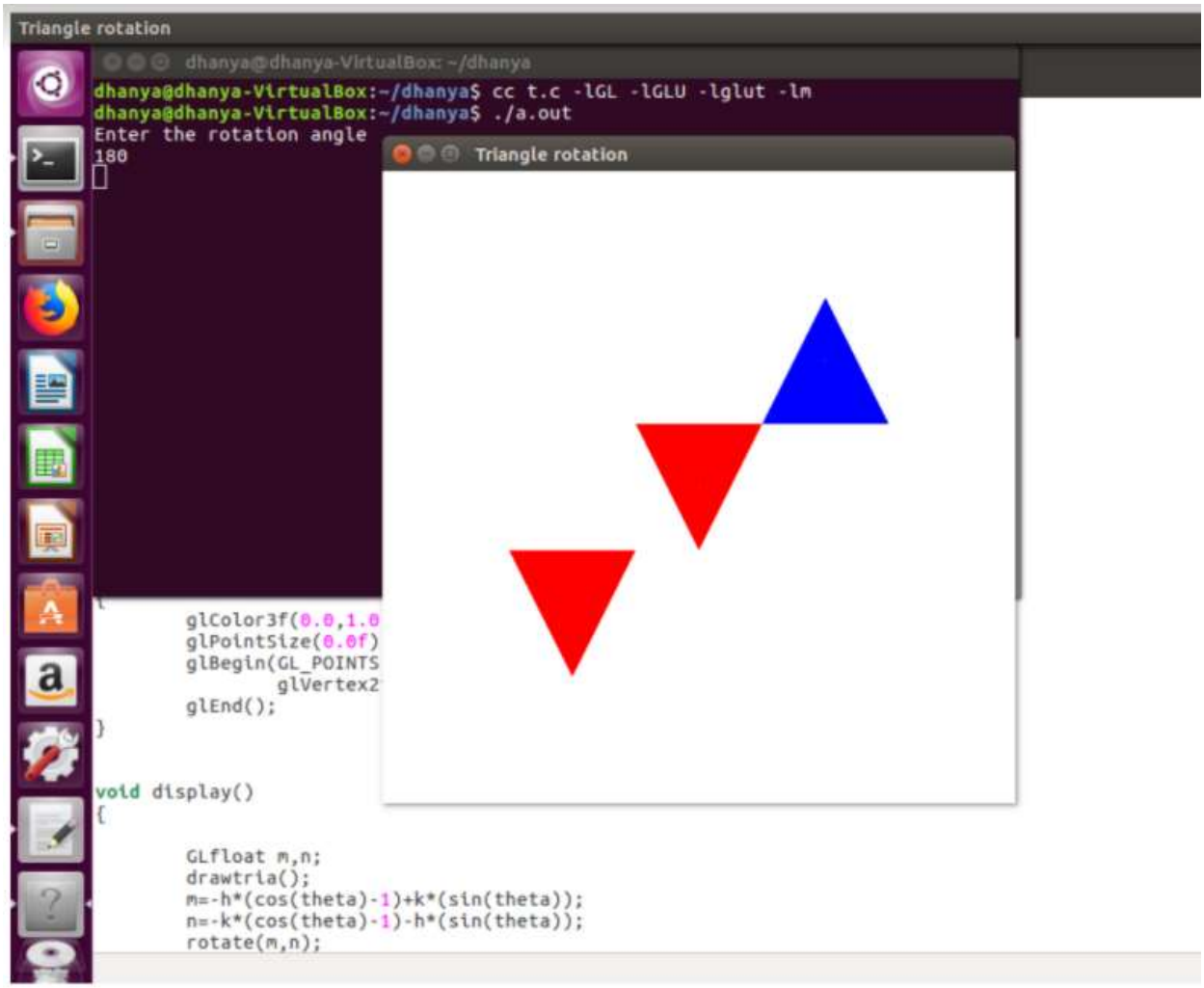

Bresenham's Line Drawing
dhanya@dhanya-VirtualBox:~/dhanya$ cc b.c -lGL -lGLU -lglut
dhanya@dhanya-VirtualBox:~/dhanya$ ./a.out
Enter (x1, y1, x2, y2)
100 100 100 400
Bresenham's Line Drawing

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

```
#include<stdio.h>
#include<math.h>
#include<GL/glut.h>
#define PI 3.1425
GLfloat tria[3][3]={{100.0,300.0,200.0},{100.0,100.0,300.0},{1.0,1.0,1.0}};
GLfloat rot_mat[3][3]={{0},{0},{0}};
GLfloat result[3][3]={{0},{0},{0}};
GLfloat h=100.0;
GLfloat k=100.0;
GLfloat theta;
void multiply()
{
int i,j,l;
for(i=0;i<3;i++)
for(j=0;j<3;j++)
{
result[i][j]=0;
for(l=0;l<3;l++)
result[i][j]=result[i][j]+rot_mat[i][l]*tria[l][j];
}
}
void rotate(GLfloat m,GLfloat n)
{
rot_mat[0][0]=cos(theta);
rot_mat[0][1]=-sin(theta);
rot_mat[0][2]=m;
rot_mat[1][0]=sin(theta);
rot_mat[1][1]=cos(theta);
rot_mat[1][2]=n;
rot_mat[2][0]=0;
rot_mat[2][1]=0;
rot_mat[2][2]=1;
multiply();
}
```

```
void drawtria()
{
glColor3f(0.0,0.0,1.0);
glBegin(GL_TRIANGLES);
glVertex2f(tria[0][0],tria[1][0]);
glVertex2f(tria[0][1],tria[1][1]);
glVertex2f(tria[0][2],tria[1][2]);
glEnd();
}
void drawrotatedtria()
{
glColor3f(1.0,0.0,1.0);
glBegin(GL_TRIANGLES);
glVertex2f(result[0][0],result[1][0]);
glVertex2f(result[0][1],result[1][1]);
glVertex2f(result[0][2],result[1][2]);
glEnd();
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
GLfloat m,n;
drawtria();
m=-h*(cos(theta)-1)+k*(sin(theta));
n=-k*(cos(theta)-1)-h*(sin(theta));
rotate(m,n);
glColor3f(0.0,1.0,0.0);
drawrotatedtria();
rotate(0,0);
glColor3f(0.0,1.0,1.0);
drawrotatedtria();
glFlush();
}
void myinit()
{
glClearColor(1.0,1.0,1.0,1.0);
glColor3f(1.0,1.0,0.0);
glPointSize(1.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,499.0,0.0,499.0);
}
void main(int argc,char **argv)
{
printf("Enter the rotation angle\n");
scanf("%f",&theta);
theta=theta*PI/180;
glutInit(&argc,argv);
```

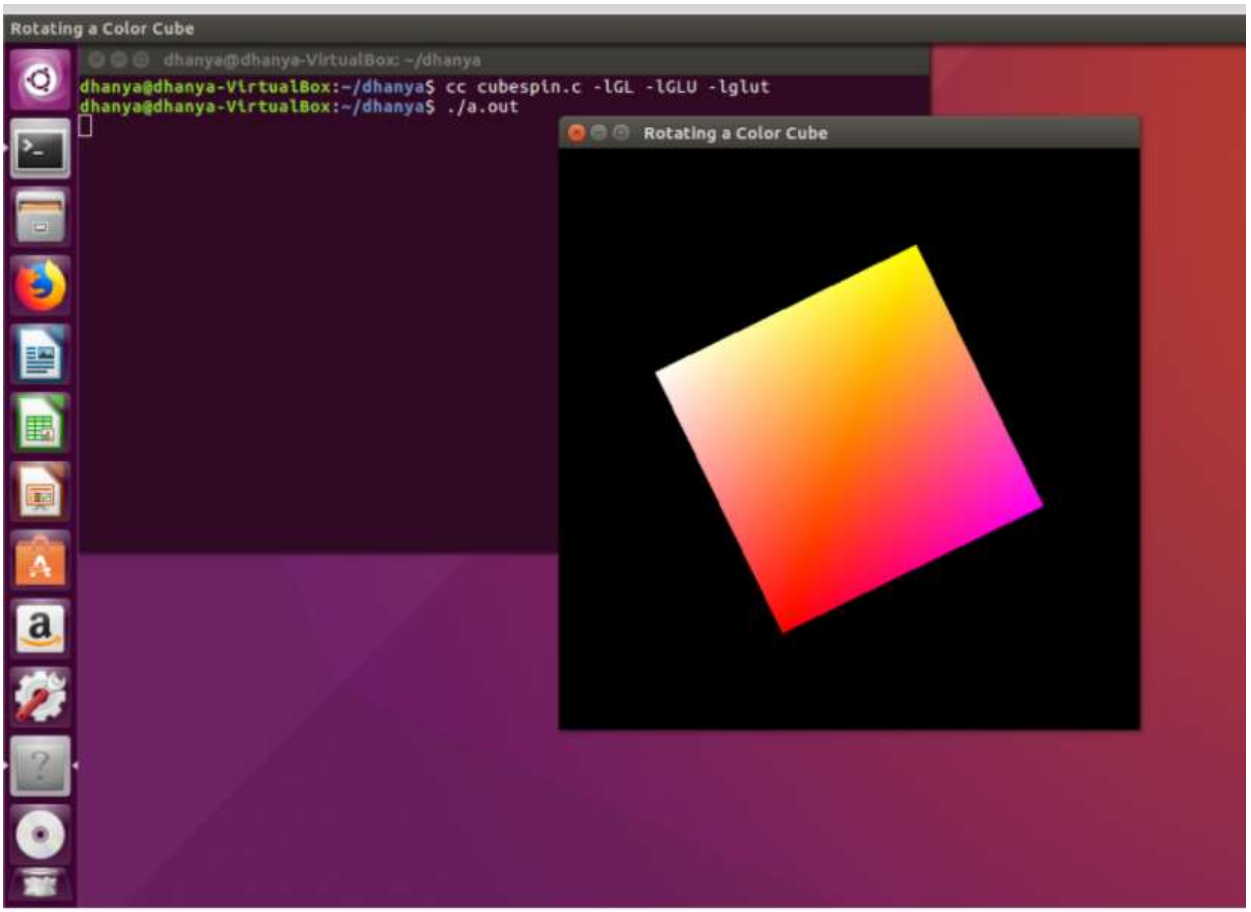
```
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("Triangle rotation");
glutDisplayFunc(display);
myinit();
glutMainLoop();
}
```



3) Program to draw a color cube and spin it using openGL transformation matrices.

```
#include<GL/glut.h>
GLfloat vertices[][3]={{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},{1.0,1.0,-1.0},
{-1.0,1.0,-1.0},{-1.0,-1.0,1.0},{1.0,-1.0,1.0},
{1.0,1.0,1.0},{-1.0,1.0,1.0}};

GLfloat colors[][3]={{0.0,0.0,0.0},{0.0,0.0,1.0},{0.0,1.0,0.0},
{0.0,1.0,1.0},{1.0,0.0,0.0},{1.0,0.0,1.0},
{1.0,1.0,0.0},{1.0,1.0,1.0}};
void polygon(int a,int b,int c,int d)
{
glBegin(GL_POLYGON);
        glColor3fv(colors[a]);
        glVertex3fv(vertices[a]);
        glColor3fv(colors[b]);
        glVertex3fv(vertices[b]);
        glColor3fv(colors[c]);
        glVertex3fv(vertices[c]);
        glColor3fv(colors[d]);
        glVertex3fv(vertices[d]);

glEnd();
}
void colorcube(void)
{
        polygon(0,3,2,1);
        polygon(2,3,7,6);
        polygon(0,4,7,3);
        polygon(1,2,6,5);
        polygon(4,5,6,7);
        polygon(0,1,5,4);
}

static GLfloat theta[]={0.0,0.0,0.0};
static GLint axis=2;



void display(void)
/* Display callback, clear frame buffer and Z buffer, rotate cube and draw, swap buffers
*/
{       glClear(GL_COLOR_BUFFER_BIT| GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        glRotatef(theta[0],1.0,0.0,0.0);
        glRotatef(theta[1],0.0,1.0,0.0);
```

```
        glRotatef(theta[2],0.0,0.0,1.0);

colorcube();
glFlush();
glutSwapBuffers();
}

void spinCube()
/* Idle callback, spin cube 1 degrees about selected axis */
{
        theta[axis]+=1.0;
        if(theta[axis]>360.0)theta[axis]-=360.0;
/*Display */
glutPostRedisplay();
}
void mouse(int btn,int state,int x,int y)
/* mouse callback, selects an axis about which to rotate */
{

        if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN) axis=0;
        if(btn==GLUT_MIDDLE_BUTTON && state==GLUT_DOWN) axis=1;
        if(btn==GLUT_RIGHT_BUTTON && state==GLUT_DOWN) axis=2;
}

void myReshape(int w,int h)
{
glViewport(0,0,w,h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
if(w<=h)
glOrtho(-2.0,2.0,-2.0 *(GLfloat) h/(GLfloat) w, 2.0*(GLfloat) h/ (GLfloat) w, -10.0,10.0);
else
glOrtho(-2.0 *(GLfloat) w/(GLfloat) h, 2.0*(GLfloat) w/ (GLfloat) h, -2.0,2.0,-10.0,10.0);
glMatrixMode(GL_MODELVIEW);
glutPostRedisplay();
}
void main(int argc, char **argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB |GLUT_DEPTH);
glutInitWindowSize(500,500);
glutCreateWindow("Rotating a Color Cube");
glutReshapeFunc(myReshape);
glutDisplayFunc(display);
glutIdleFunc(spinCube);
glutMouseFunc(mouse);
glEnable(GL_DEPTH_TEST); /* Enable hidden – surface—removal */
glutMainLoop();}
```


Rotating a Color Cube
dhanya@dhanya-VirtualBox: ~/dhanya
dhanya@dhanya-VirtualBox:~/dhanya$ cc cubespin.c -lGL -lGLU -lglut
dhanya@dhanya-VirtualBox:~/dhanya$ ./a.out
Rotating a Color Cube

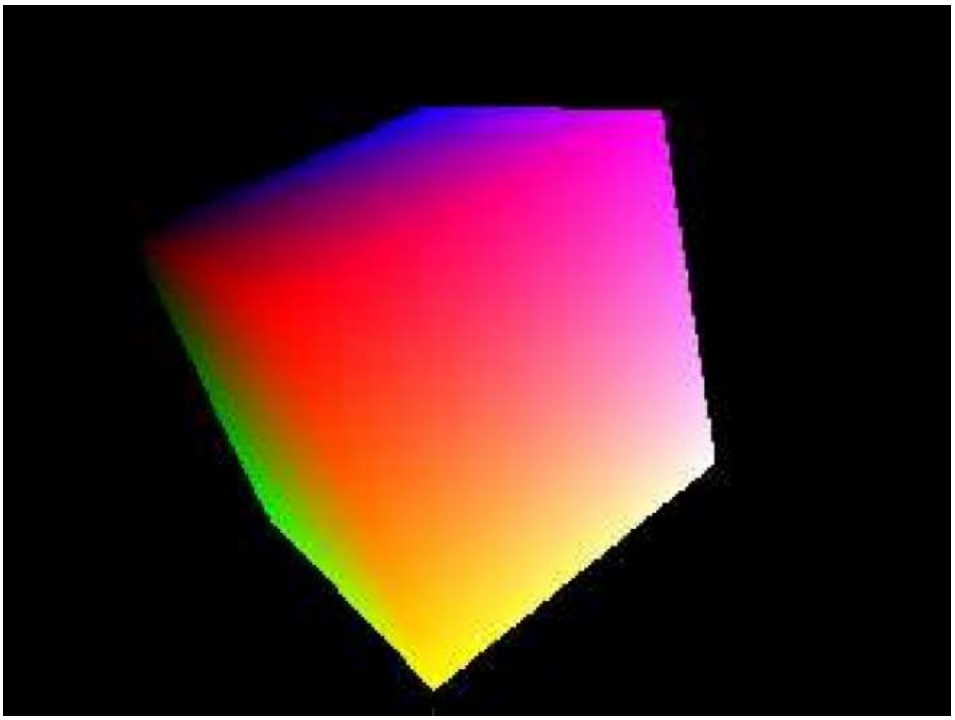
4) Program to draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing using OpenGL functions

```
/* We use the Lookat function in the display callback to point
the viewer, whose position can be altered by the x,X,y,Y,z, and Z keys.
The perspective view is set in the reshape callback */

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


        GLfloat vertices[][3] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},
        {1.0,1.0,-1.0}, {-1.0,1.0,-1.0}, {-1.0,-1.0,1.0},
        {1.0,-1.0,1.0}, {1.0,1.0,1.0}, {-1.0,1.0,1.0}};

        GLfloat colors[][3] = {{0.0,0.0,0.0},{1.0,0.0,0.0},
        {1.0,1.0,0.0}, {0.0,1.0,0.0}, {0.0,0.0,1.0},
        {1.0,0.0,1.0}, {1.0,1.0,1.0}, {0.0,1.0,1.0}};

void polygon(int a, int b, int c , int d)
{
                glBegin(GL_POLYGON);
                glColor3fv(colors[a]);
                glVertex3fv(vertices[a]);
                glColor3fv(colors[b]);
                glVertex3fv(vertices[b]);
                glColor3fv(colors[c]);
                glVertex3fv(vertices[c]);
                glColor3fv(colors[d]);
                glVertex3fv(vertices[d]);
        glEnd();

                }

void colorcube()
{
        polygon(0,3,2,1);
        polygon(2,3,7,6);
        polygon(0,4,7,3);
        polygon(1,2,6,5);
        polygon(4,5,6,7);
        polygon(0,1,5,4);
}

static GLfloat theta[] = {0.0,0.0,0.0};
static GLint axis = 2;
static GLdouble viewer[]= {0.0, 0.0, 5.0};
```

```
/* initial viewer location */

void display(void)
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
/* Update viewer position in modelview matrix */
        glLoadIdentity();
        gluLookAt(viewer[0],viewer[1],viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
/* rotate cube */
        glRotatef(theta[0], 1.0, 0.0, 0.0);
        glRotatef(theta[1], 0.0, 1.0, 0.0);
        glRotatef(theta[2], 0.0, 0.0, 1.0);

colorcube();

glFlush();
        glutSwapBuffers();
}

void mouse(int btn, int state, int x, int y)
{
if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN) axis = 0;
if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) axis = 1;
if(btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN) axis = 2;
        theta[axis] += 2.0;
if( theta[axis] > 360.0 ) theta[axis] -= 360.0;
        display();
}

void keys(unsigned char key, int x, int y)
{
/* Use x, X, y, Y, z, and Z keys to move viewer */
if(key == 'x') viewer[0]-= 1.0;
if(key == 'X') viewer[0]+= 1.0;
if(key == 'y') viewer[1]-= 1.0;
if(key == 'Y') viewer[1]+= 1.0;
if(key == 'z') viewer[2]-= 1.0;
if(key == 'Z') viewer[2]+= 1.0;
display();
}

void myReshape(int w, int h)
{
glViewport(0, 0, w, h);
/* Use a perspective view */
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
```

```
        if(w<=h) glFrustum(-2.0, 2.0, -2.0 * (GLfloat) h/ (GLfloat) w,2.0* (GLfloat) h /
(GLfloat) w, 2.0, 20.0);
        else glFrustum(-2.0, 2.0, -2.0 * (GLfloat) w/ (GLfloat) h,2.0* (GLfloat) w /
(GLfloat) h, 2.0, 20.0);
/* Or we can use gluPerspective */
/* gluPerspective(45.0, w/h, -10.0, 10.0); */
glMatrixMode(GL_MODELVIEW);
}

void main(int argc, char **argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowSize(500, 500);
glutCreateWindow("Colorcube Viewer");
glutReshapeFunc(myReshape);
glutDisplayFunc(display);
glutMouseFunc(mouse);
glutKeyboardFunc(keys);
glEnable(GL_DEPTH_TEST);
glutMainLoop();

}
```

Outtput



5) Clip the lines using Cohen-Sutherland algorithm

Algorithm:

To perform the trivial acceptance and rejection tests, we extend the edges of the window to divide the plane of the window into the nine regions. Each end point of the line segment is then assigned the code of the region in which it lies.

1. Given a line segment with endpoint $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$
2. Compute the 4-bit codes for each endpoint.

   If both codes are **0000**,(bitwise OR of the codes yields 0000) line lies completely **inside** the window: pass the endpoints to the draw routine.

   If both codes have a 1 in the same bit position (bitwise AND of the codes is **not** 0000), the line lies **outside** the window. It can be trivially rejected.

3. If a line cannot be trivially accepted or rejected, at least one of the two endpoints must lie outside the window and the line segment crosses a window edge. This line must be **clipped** at the window edge before being passed to the drawing routine.
4. Examine one of the endpoints, say $P_1 = (x_1, y_1)$. Read P_1 's 4-bit code in order: **Left**-to-**Right**, **Bottom**-to-**Top**.
5. When a set bit (1) is found, compute the **intersectionI** of the corresponding window edge with the line from P_1 to P_2. Replace P_1 with **I** and repeat the algorithm.

Cohen-Sutherland clipping algorithm

- To clip a line, find out which regions its two endpoints lie in.
- If they are both in region 0000, then it's completely in.
- If the two region numbers both have a 1 in the same bit position, the line is completely out.
- Otherwise, we have to do some more calculations.

1001	1000	1010
0001	0000	0010
0101	0100	0110

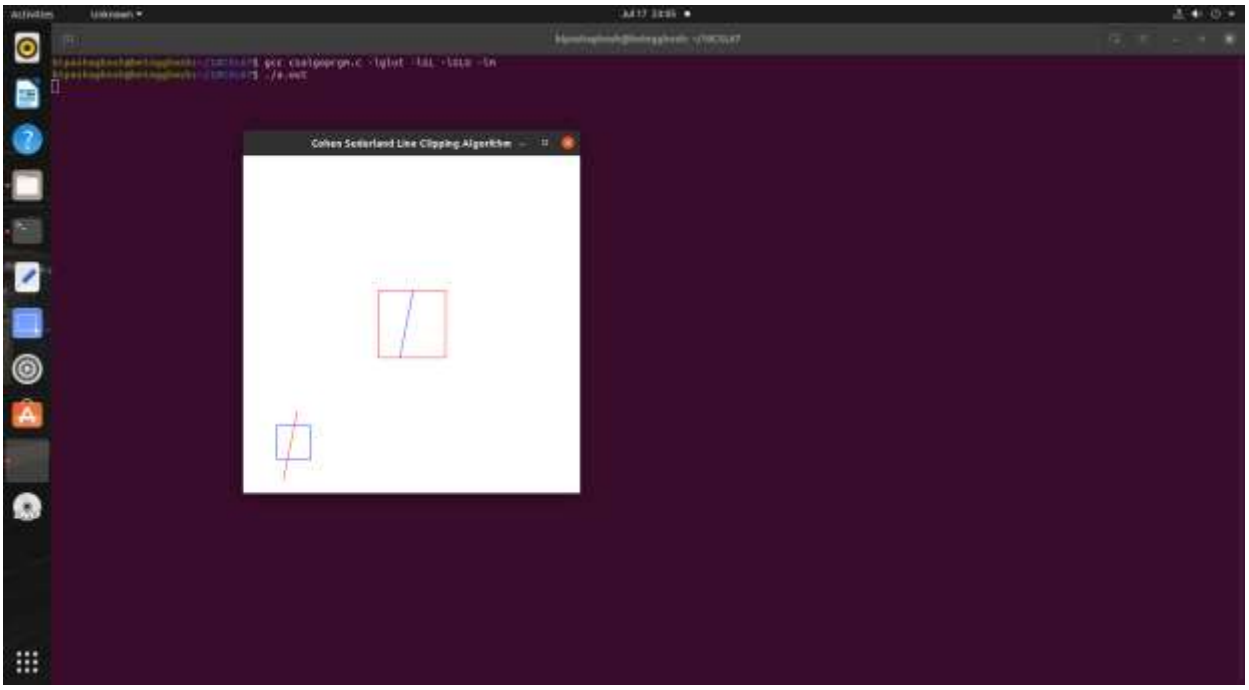
PROGRAM

```
#include <stdio.h>
#include <GL/glut.h>
#define bool int

#define true 1
#define false 0

#define outcode int
double xmin=50,ymin=50, xmax=100,ymax=100; // Window boundaries
double xvmin=200,yvmin=200,xvmax=300,yvmax=300; // Viewport boundaries
//bit codes for the right, left, top, & bottom
const int RIGHT = 2;
const int LEFT = 1;
const int TOP = 8;
const int BOTTOM = 4;

//used to compute bit codes of a point
outcode ComputeOutCode (double x, double y);

//Cohen-Sutherland clipping algorithm clips a line from
//P0 = (x0, y0) to P1 = (x1, y1) against a rectangle with
//diagonal from (xmin, ymin) to (xmax, ymax).
void CohenSutherlandLineClipAndDraw (double x0, double y0,double x1, double y1)
{
//Outcodes for P0, P1, and whatever point lies outside the clip rectangle
        outcode outcode0, outcode1, outcodeOut;
        bool accept = false, done = false;

//compute outcodes
        outcode0 = ComputeOutCode (x0, y0);
        outcode1 = ComputeOutCode (x1, y1);

        do{
                if (!(outcode0 | outcode1))     //logical or is 0 Trivially accept & exit
                {
                        accept = true;
                        done = true;
                }
                else if (outcode0  & outcode1)    //logical and is not 0. Trivially reject  and
exit
                        done = true;
                else
                {
//failed both tests, so calculate the line segment to clip
//from an outside point to an intersection with clip edge
                        double x, y;
```

```
//At least one endpoint is outside the clip rectangle; pick it.
                        outcodeOut = outcode0? outcode0: outcode1;

//Now find the intersection point;
//use formulas y=y0+slope*(x-x0),x =x0+(1/slope)*(y-y0)
                        if (outcodeOut & TOP)
//point is above the clip rectangle
                        {
                        x = x0 + (x1 - x0) * (ymax - y0)/(y1 - y0);
                        y = ymax;
                        }
                        else if (outcodeOut & BOTTOM)
//point is below the clip rectangle
                        {
                        x = x0 + (x1 - x0) * (ymin - y0)/(y1 - y0);
                                y = ymin;
                        }
                        else if (outcodeOut & RIGHT)
//point is to the right of clip rectangle
                        {
                        y = y0 + (y1 - y0) * (xmax - x0)/(x1 - x0);
                        x = xmax;
                        }
                        else
//point is to the left of clip rectangle
                        {
                        y = y0 + (y1 - y0) * (xmin - x0)/(x1 - x0);
                        x = xmin;
                        }

//Now we move outside point to intersection point to clip
//and get ready for next pass.
                        if (outcodeOut == outcode0)
                        {
                                x0 = x;
                                y0 = y;
                                outcode0 = ComputeOutCode (x0, y0);
                        }
                        else
                        {
                                x1 = x;
                                y1 = y;
                                outcode1 = ComputeOutCode (x1, y1);
                        }
                }

        }while (!done);
```

```
        if (accept)
        {           // Window to viewport mappings
                double sx=(xvmax-xvmin)/(xmax-xmin);
                // Scale parameters
                double sy=(yvmax-yvmin)/(ymax-ymin);
                double vx0=xvmin+(x0-xmin)*sx;
                double vy0=yvmin+(y0-ymin)*sy;
                double  vx1=xvmin+(x1-xmin)*sx;
                double vy1=yvmin+(y1-ymin)*sy;
                //draw a red colored viewport
                glColor3f(1.0, 0.0, 0.0);
                glBegin(GL_LINE_LOOP);
                        glVertex2f(xvmin,  yvmin);
                        glVertex2f(xvmax,  yvmin);
                        glVertex2f(xvmax, yvmax);
                        glVertex2f(xvmin, yvmax);
                glEnd();
                glColor3f(0.0,0.0,1.0);
                // draw blue colored clipped line
                glBegin(GL_LINES);
                        glVertex2d (vx0, vy0);
                        glVertex2d (vx1, vy1);
                glEnd();
        }
}

//Compute the bit code for a point (x, y) using the clip rectangle
//bounded diagonally by (xmin, ymin), and (xmax, ymax)
outcode ComputeOutCode (double x, double y)
{
        outcode code = 0;
        if (y > ymax)           //above the clip window
                code |= TOP;
        else if (y < ymin)      //below the clip window
                code |= BOTTOM;
        if (x > xmax)           //to the right of clip window
                code |= RIGHT;
        else if (x < xmin)      //to the left of clip window
                code |= LEFT;
        return code;
}

void display()
{
double x0=60,y0=20,x1=80,y1=120;
glClear(GL_COLOR_BUFFER_BIT);
//draw the line with red color
glColor3f(1.0,0.0,0.0);
```

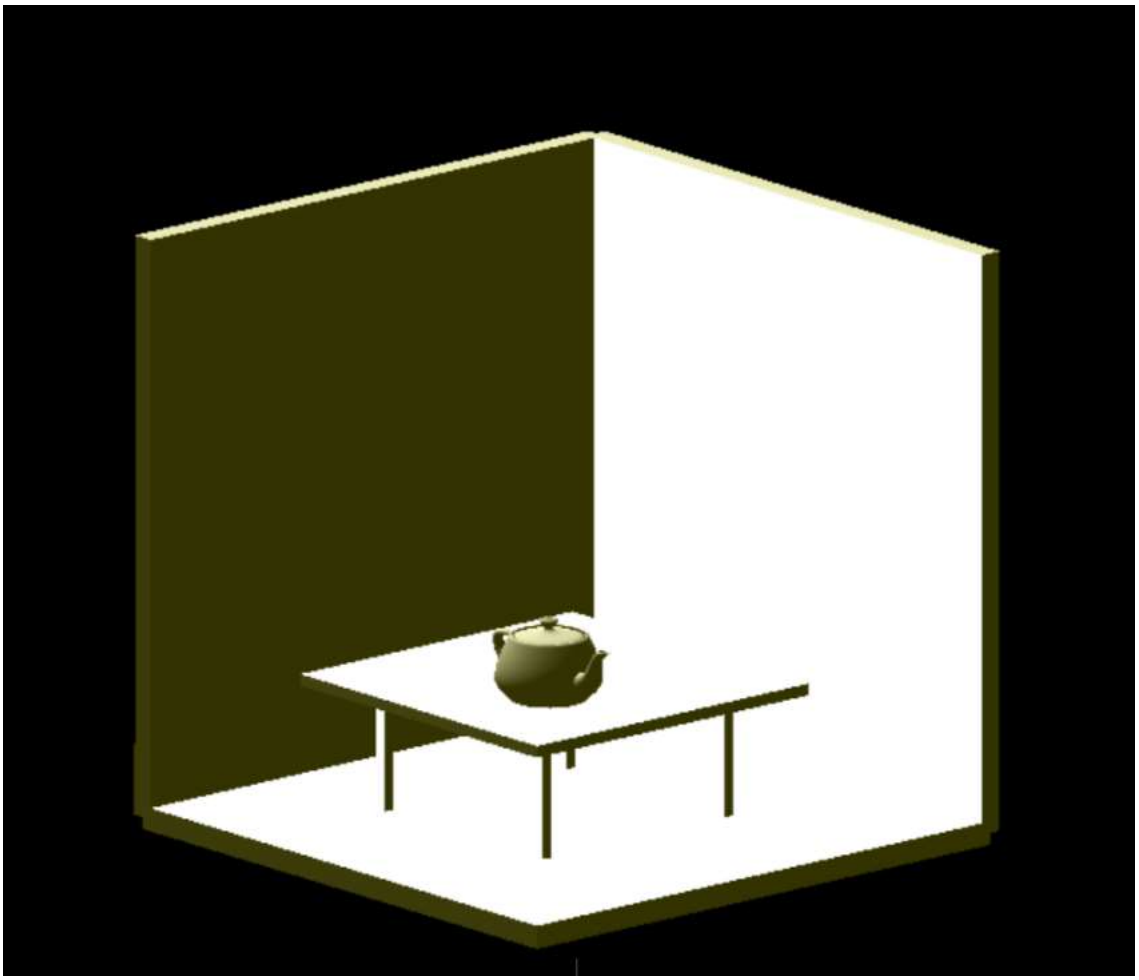
```
//bres(120,20,340,250);
glBegin(GL_LINES);
                        glVertex2d (x0, y0);
                        glVertex2d (x1, y1);
                glEnd();

//draw a blue colored window
glColor3f(0.0, 0.0, 1.0);

glBegin(GL_LINE_LOOP);
glVertex2f(xmin, ymin);
glVertex2f(xmax, ymin);
glVertex2f(xmax, ymax);
glVertex2f(xmin, ymax);
glEnd();
CohenSutherlandLineClipAndDraw(x0,y0,x1,y1);
glFlush();
}


void myinit()
{
        glClearColor(1.0,1.0,1.0,1.0);
        glColor3f(1.0,0.0,0.0);
        glPointSize(1.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0,499.0,0.0,499.0);
}


void main(int argc, char** argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(0,0);
        glutCreateWindow("Cohen Suderland Line Clipping Algorithm");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```


Cohen Sutherland Line Clipping 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.

```
#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); // three walls
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); // four table legs
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); // table top
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();
}
void 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("scene");
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();
}
```



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

typedef float point[3];
int n;

point v[] = {{ 0.0,  0.0, 1.0},
             {-0.8, -0.4, 0.0},
             { 0.0,  0.8, 0.0},
             { 0.8, -0.4, 0.0}};

void divide_triangle(point a, point b, point c, int m){

    point v1, v2, v3;
    int j;

    if(m > 0) {

        for(j=0; j<3; j++) v1[j] = (a[j] + b[j]) / 2;
        for(j=0; j<3; j++) v2[j] = (a[j] + c[j]) / 2;
        for(j=0; j<3; j++) v3[j] = (b[j] + c[j]) / 2;

        divide_triangle(a, v1, v2, m-1);
        divide_triangle(b, v1, v3, m-1);
        divide_triangle(c, v2, v3, m-1);

      } else {
       glBegin(GL_POLYGON);
         glNormal3fv(a);
         glVertex3fv(a);
         glVertex3fv(b);
         glVertex3fv(c);
        glEnd();
    }
}


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[0], v[2], v[3], m);
```

```
glColor3f(0.0, 0.0, 1.0);
    divide_triangle(v[0], v[1], v[3], m);

    glColor3f(1.0, 1.0, 0.0);
    divide_triangle(v[1], v[2], v[3], m);

}
void display(){

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glMatrixMode(GL_PROJECTION);

    glLoadIdentity();

    tetrahedron(n);

    glFlush();

}

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

    printf("How many divisions?: ");
    scanf("%d",&n);

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);

    glutInitWindowPosition(500, 100);
    glutInitWindowSize(500, 500);

    glutCreateWindow("3D Sierpinski Gasket");

    glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);

    glutDisplayFunc(display);

    glEnable(GL_DEPTH_TEST);

    glClearColor(1, 1, 1, 1);

    glutMainLoop();









}
```

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

```
#include<glut.h>
#include<stdio.h>
#include<math.h>
#define PI 3.1416
GLsizei winWidth = 600, winHeight = 600;
GLfloat xwcMin = 0.0, xwcMax = 130.0;
GLfloat ywcMin = 0.0, ywcMax = 130.0;
static int window;
static int menu_id=2;
static int submenu_id=1;
static int value = 0;
typedef struct wcPt3D
{
        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 computebzPt(GLfloat u, wcPt3D *bzPt, Glint nCtrlPts, wcPt3D *ctrlPts, GLint *C)
{
        GLint k, n = nCtrlPts - 1;
        GLfloat  bzBdFcn;
```

```
        bzPt->x = bzPt->y = bzPt->z = 0.0;
        for (k = 0; k< nCtrlPts; k++)
        {
                bzBdFcn = C[k] * pow(u, k) * pow(1 - u, n - k);
                bzPt->x += ctrlPts[k].x * bzBdFcn;
                bzPt->y += ctrlPts[k].y * bzBdFcn;
                bzPt->z += ctrlPts[k].z * bzBdFcn;
        }
}
void bezier(wcPt3D *ctrlPts, GLint nCtrlPts, GLint nbzCPts)
{
        wcPt3D bzCPt;
        GLfloat u;
        GLint *C, k;
        C = new GLint[nCtrlPts];
        bino(nCtrlPts - 1, C);
        glBegin(GL_LINE_STRIP);
        for (k = 0; k <= nbzCPts; k++)
        {
                u = GLfloat(k) / GLfloat(nbzCPts);
                computebzPt(u, &bzCPt, nCtrlPts, ctrlPts, C);
                glVertex2f(bzCPt.x, bzCPt.y);
        }
        glEnd();
        delete[]C;
}
void displayFunc()
{
        GLint nCtrlPts = 4, nbzCPts = 20;
        static float theta = 0;
        wcPt3D ctrlPts[4] = {
                { 20, 100, 0 },
```

```
                { 30, 110, 0 },
                { 50, 90, 0 },
                { 60, 100, 0 }
};
        ctrlPts[1].x += 10 * sin(theta * PI / 180.0);
        ctrlPts[1].y += 5 * sin(theta * PI / 180.0);
        ctrlPts[2].x -= 10 * sin((theta + 30) * PI / 180.0);
        ctrlPts[2].y -= 10 * sin((theta + 30) * PI / 180.0);
        ctrlPts[3].x -= 4 * sin((theta)* PI / 180.0);
        ctrlPts[3].y += sin((theta - 30) * PI / 180.0);
        theta += 0.1;
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(1.0, 1.0, 1.0);
        glPointSize(5);
        glPushMatrix();
        glLineWidth(5);
        glColor3f(255 / 255, 153 / 255.0, 51 / 255.0); //Indian flag: Orange color code

        for (int i = 0; i<8; i++)
        {
                glTranslatef(0, -0.8, 0);
                bezier(ctrlPts, nCtrlPts, nbzCPts);
        }
        glColor3f(1, 1, 1); //Indian flag: white color code
        for (int i = 0; i<8; i++)
        {
                glTranslatef(0, -0.8, 0);
                bezier(ctrlPts, nCtrlPts, nbzCPts);
        }
        glColor3f(19 / 255.0, 136 / 255.0, 8 / 255.0); //Indian flag: green color code
        for (int i = 0; i<8; i++)
        {
                glTranslatef(0, -0.8, 0);
```

```
                bezier(ctrlPts, nCtrlPts, nbzCPts);
        }
        glPopMatrix();
        glColor3f(0.7, 0.5, 0.3);
        glLineWidth(5);
        glBegin(GL_LINES);
        glVertex2f(20, 100);
        glVertex2f(20, 40);
        glEnd();
        glFlush();
        glutPostRedisplay();
        glutSwapBuffers();
}
void winReshapeFun(GLint newWidth, GLint newHeight)
{
        glViewport(0, 0, newWidth, newHeight);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);
        glClear(GL_COLOR_BUFFER_BIT);
}

 void display(void)
{
        glClear(GL_COLOR_BUFFER_BIT);
        if (value == 1)
       {
                return; //glutPostRedisplay();
        }
        else if (value == 2)
      {
        glPushMatrix();
        glColor3d(1.0, 0.0, 0.0);
```

```
        glutDisplayFunc(displayFunc);
        glutWireSphere(0.5, 50, 50);
        glPopMatrix();
        }
        glFlush();
}
void menu(int num)
{
        if (num == 0)
       {
        glutDestroyWindow(window);
        exit(0);
       }
       else
      {
       value = num;
       }
    glutPostRedisplay();
}
void createMenu(void)
{
        submenu_id = glutCreateMenu(menu);
        glutAddMenuEntry("draw a flag", 2);
        menu_id = glutCreateMenu(menu);
 glutAddMenuEntry("Clear", 1);
        glutAddSubMenu("Draw", submenu_id);
        glutAddMenuEntry("Quit", 0);
        glutAttachMenu(GLUT_RIGHT_BUTTON);
}
void myinit()
{
        glViewport(0, 0, 500, 500);
        glClearColor(1.0, 1.0, 1.0, 1.0);
        glColor3f(1.0, 0.0, 0.0);
```

```
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0, 499.0, 0.0, 499.0);
}

int main(int argc, char **argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
        glutInitWindowSize(500, 500);
        glutInitWindowPosition(100, 100);
        window = glutCreateWindow("Menus and Submenus - Programming Techniques");
        createMenu();
       glClearColor(0.0, 0.0, 0.0, 0.0);
       glutDisplayFunc(display);
        glutReshapeFunc(winReshapeFun);
        myinit();
        glutMainLoop();
}
```

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

ALGORITHM:

Step 1 − Find out the Yminand Ymax from the given polygon.

Step 2 − ScanLine intersects with each edge of the polygon from Ymin to Ymax. Name each intersection point of the polygon. As per the figure shown above, they are named as p0, p1, p2, p3.

Step 3 − Sort the intersectionpoint in the increasing order of X coordinatei.e.(p0, p1), (p1,p2), and (p2, p3).

Step 4 − Fill all those pair of coordinates that are inside polygons and ignore the alternatepairs.

```
#include <GL/glut.h>
#include<stdio.h>
static int window;
static int menu_id;
static int submenu_id;
static int value = 1;
float x1 = 200.0, y1_ = 200.0, x2 = 100.0, y2 = 300.0, x3 = 200.0, y3
= 400.0, x4 = 300.0, y4 = 300.0;
void draw_pixel(int x, int y)
{if(value==2){
glColor3f(0.0, 0.0, 1.0);}
else{glColor3f(1.0,1.0,1.0);}
glBegin(GL_POINTS);
glVertex2i(x, y);
glEnd();
glFlush();
}

void edgedetect(float x1, float y1_, float x2, float y2, int *le, int
*re)
{
float mx, x, temp;
int i;
if ((y2 - y1_)<0) // y1_>y2 ie. y1_=400 ,y2=300
{
temp = y1_; y1_ = y2; y2 = temp;
temp = x1; x1 = x2; x2 = temp;
}

if ((y2 - y1_) != 0) //not a horizontal line
mx = (x2 - x1) / (y2 - y1_);
else mx = x2 - x1;
x = x1;
```

```
for (i = y1_; i <= y2; i++)
{
if (x<(float)le[i])
le[i] = (int)x;
if (x>(float)re[i])
re[i] = (int)x;
x += mx;
}
}

void scanfill(float x1, float y1_, float x2, float y2, float x3, float
y3, float x4, float y4)
{
int le[500], re[500], i, y;
for (i = 0; i<500; i++)
{
le[i] = 500;
re[i] = 0;
}

edgedetect(x1, y1_, x2, y2, le, re);
edgedetect(x2, y2, x3, y3, le, re);
edgedetect(x3, y3, x4, y4, le, re);
edgedetect(x4, y4, x1, y1_, le, re);
if(value==2){
for (y = 0; y<500; y++)
{
if (le[y] <= re[y])
for (i = (int)le[y]+1; i<(int)re[y]; i++)
draw_pixel(i, y);
}}else{
for (y = 499; y>=0; y--)
{
if (le[y] <= re[y])
for (i = (int)re[y] - 1; i>(int)le[y]; i--)
draw_pixel(i, y);
}}
}

void menu(int num){
if (num == 0)
{
glutDestroyWindow(window);
exit(0);
}
else if(num==1){
glClear(GL_COLOR_BUFFER_BIT);
}
else{
value = num;
}
glutPostRedisplay();
}
```

```
void createMenu(void)
{
submenu_id = glutCreateMenu(menu);
glutAddMenuEntry("scanfill polygon", 2);
menu_id = glutCreateMenu(menu);
glutAddMenuEntry("Clear", 1);
glutAddSubMenu("Draw", submenu_id);
glutAddMenuEntry("Quit", 0);
glutAddMenuEntry("Reverse", 3);
glutAttachMenu(GLUT_RIGHT_BUTTON);
}
void display(void)
{
if (value == 1)
{
glClear(GL_COLOR_BUFFER_BIT);
return; //glutPostRedisplay();
}
else if (value == 2)
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0, 0.0, 0.0);
glBegin(GL_LINE_LOOP);
glVertex2f(x1, y1_);
glVertex2f(x2, y2);
glVertex2f(x3, y3);
glVertex2f(x4, y4);
glEnd();
scanfill(x1, y1_, x2, y2, x3, y3, x4, y4);
value=0;
glFlush();
}
else if (value == 3){
scanfill(x1, y1_, x2, y2, x3, y3, x4, y4);
value=0;
glFlush();
}
}
void myinit()
{
glClearColor(1.0, 1.0, 1.0, 1.0);
glColor3f(1.0, 0.0, 0.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, 499.0, 0.0, 499.0);
}
int main(int argc, char **argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
glutInitWindowSize(500, 500);
glutInitWindowPosition(100, 100);
window = glutCreateWindow("Menu driven Programming for Scan filling
");
createMenu();
glClearColor(0.0, 0.0, 0.0, 0.0);
```

```
glutDisplayFunc(display);
myinit();
glutMainLoop();
return EXIT_SUCCESS;
}
```