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

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

Computer Graphics Labortory
program 1

```
#include<GL/glut.h>
#include<stdio.h>
int x1,y1,x2,y2;
void draw_pixel(int x,int y)
{
glColor3f(1.0,0.0,0.0);
glBegin(GL_POINTS);
glVertex2i(x,y);
glEnd();
}
void breseuhams_line_draw(int x1,int y1,int x2,int y2)
{
int dx=x2-x1;
int dy=y2-y1;
int m=dy/dx;
if(m<1)
{
int decision_parameter=2*dy-dx;
int x=x1;
int y=y1;
if(dx<0)
{
x=x2;
y=y2;
x2=x1;
}
draw_pixel(x,y);
while(x<x2)
{
if(decision_parameter>=0)
{
x=x+1;
y=y+1;
decision_parameter=decision_parameter+2*dy-2*dx*(y+1-y);
}
else
{
x=x+1;
y=y;
decision_parameter=decision_parameter + 2 * dy - 2 * dx *(y-y);
}
draw_pixel(x,y);
}
}
else if (m>1)
{
int decision_parameter=2*dx-dy;
int x=x1;
```

```
int y=y1;
if(dy<0)
{
x=x2;
y=y2;
y2=y1;
}
draw_pixel(x,y);
while(y<y2)
{
if(decision_parameter>=0)
{
x=x+1;
y=y+1;
decision_parameter=decision_parameter+2*dx-2*dy*(x+1-x);
}
else
{
y=y+1;
x=x;
decision_parameter=decision_parameter+2*dx-2*dy*(x-x);
}
draw_pixel(x,y);
}
}
else if(m==1)
{
int x=x1;
int y=y1;
draw_pixel(x,y);
while(x<x2)
{
x=x+1;
y=y+1;
draw_pixel(x,y);
}
}
}
void init()
{
glClearColor(1,1,1,1);
gluOrtho2D(0.0,500.0,0.0,500.0);
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
breseuhams_line_draw(x1,y1,x2,y2);
glFlush();
}
int main(int argc,char **argv)
{
printf("enter start points (x1,y1)\n");
```

```
scanf("%d %d",&x1,&y1);
printf("enter end points (x2,y2)\n");
scanf("%d %d", &x2,&y2);
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(220,200);
glutCreateWindow("Bresenham's Line Drawing");
init();
glutDisplayFunc(display);
glutMainLoop();
}
```

compile: g++ prg1.c -lglut -lGL -lGLU

2.

```
#include<GL/glut.h>
#include<stdio.h>
int x,y;
int where_to_rotate=0;
float rotate_angle=0;
float translate_x=0,translate_y=0;
void draw_pixel(float x1, float y1)
{
   glPointSize(5);
   glBegin(GL_POINTS);
      glVertex2f(x1,y1);
   glEnd();
}
void triangle(int x, int y)
{
   glColor3f(1,0,0);
   glBegin(GL_POLYGON);
      glVertex2f(x,y);
      glVertex2f(x+400,y+300);
      glVertex2f(x+300,y+0);
   glEnd();
}
void display()
{
   glClear(GL_COLOR_BUFFER_BIT);
   glLoadIdentity();
   glColor3f(1,1,1);
   draw_pixel(0,0);
   if (where_to_rotate == 1)
   {
      translate_x = 0;
      translate_y = 0;
      rotate_angle += 1;
   }
   if (where_to_rotate == 2)
   {
      translate_x = x;
      translate_y = y;
      rotate_angle += 1;
      glColor3f(0,0,1);
      draw_pixel(x,y);
   }
   glTranslatef(translate_x, translate_y, 0);
   glRotatef(rotate_angle, 0, 0, 1);
   glTranslatef(-translate_x, -translate_y, 0);
   triangle(translate_x,translate_y);
   glutPostRedisplay();
   glutSwapBuffers();
}
void init()
{
```

```
    glClearColor(0,0,0,1);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-800, 800, -800, 800);
    glMatrixMode(GL_MODELVIEW);
}
void rotateMenu (int option)
{
    if(option==1)
        where_to_rotate=1;
    if(option==2)
        where_to_rotate=2;
    if(option==3)
        where_to_rotate=3;
}
int main(int argc, char **argv)
{
    printf( "Enter Fixed Points (x,y) for Rotation: \n");
    scanf("%d %d", &x, &y);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
    glutInitWindowSize(800, 800);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Create and Rotate Triangle");
    init();
    glutDisplayFunc(display);
    glutCreateMenu(rotateMenu);
    glutAddMenuEntry("Rotate around ORIGIN",1);
    glutAddMenuEntry("Rotate around FIXED POINT",2);
    glutAddMenuEntry("Stop Rotation",3);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    glutMainLoop();
}
```

Program 3

```
#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 normals[][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]);
glNormal3fv(normals[a]);
glVertex3fv(vertices[a]);
glColor3fv(colors[b]);
glNormal3fv(normals[b]);
glVertex3fv(vertices[b]);
glColor3fv(colors[c]);
glNormal3fv(normals[c]);
glVertex3fv(vertices[c]);
glColor3fv(colors[d]);
glNormal3fv(normals[d]);
glVertex3fv(vertices[d]);
glEnd();
}
void colorcube(void)
{
polygon(0,3,2,1);
polygon(2,3,7,6);
polygon(0,4,7,3);
```

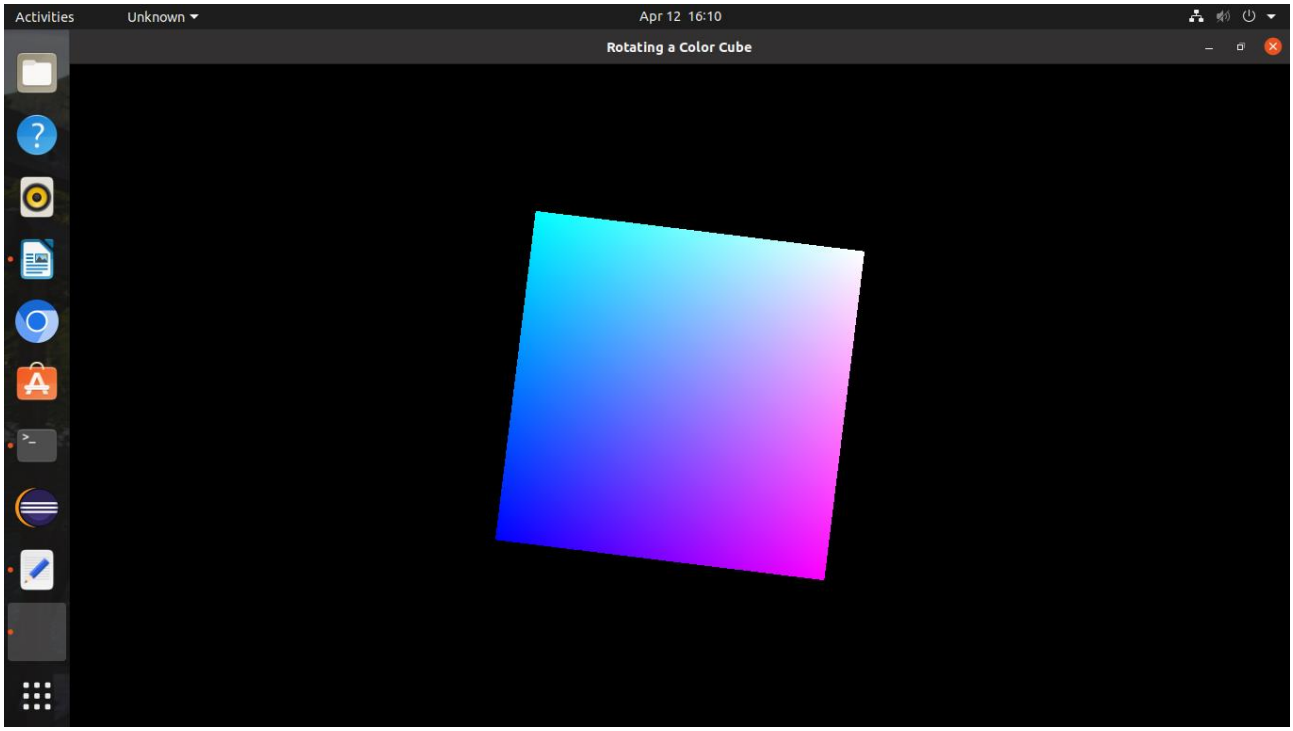
```
polygon(1,2,6,5);
polygon(4,5,6,7);
polygon(0,1,5,4);
}
static GLfloat theta[] = {0.0,0.0,0.0};
static GLint axis = 2;
void display(void)
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
glRotatef(theta[0], 1.0, 0.0, 0.0);
glRotatef(theta[1], 0.0, 1.0, 0.0);
glRotatef(theta[2], 0.0, 0.0, 1.0);
colorcube();
glFlush();
glutSwapBuffers();
}
void spinCube()
{
theta[axis] += 1.0;
if( theta[axis] > 360.0 )
theta[axis] -= 360.0;
glutPostRedisplay();
}
void mouse(int btn, int state, int x, int y)
{
if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN)
axis = 0;
if(btn==GLUT_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);
}
int main(int argc, char **argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowSize(500, 500);
glutCreateWindow("Rotating a Color Cube");
```

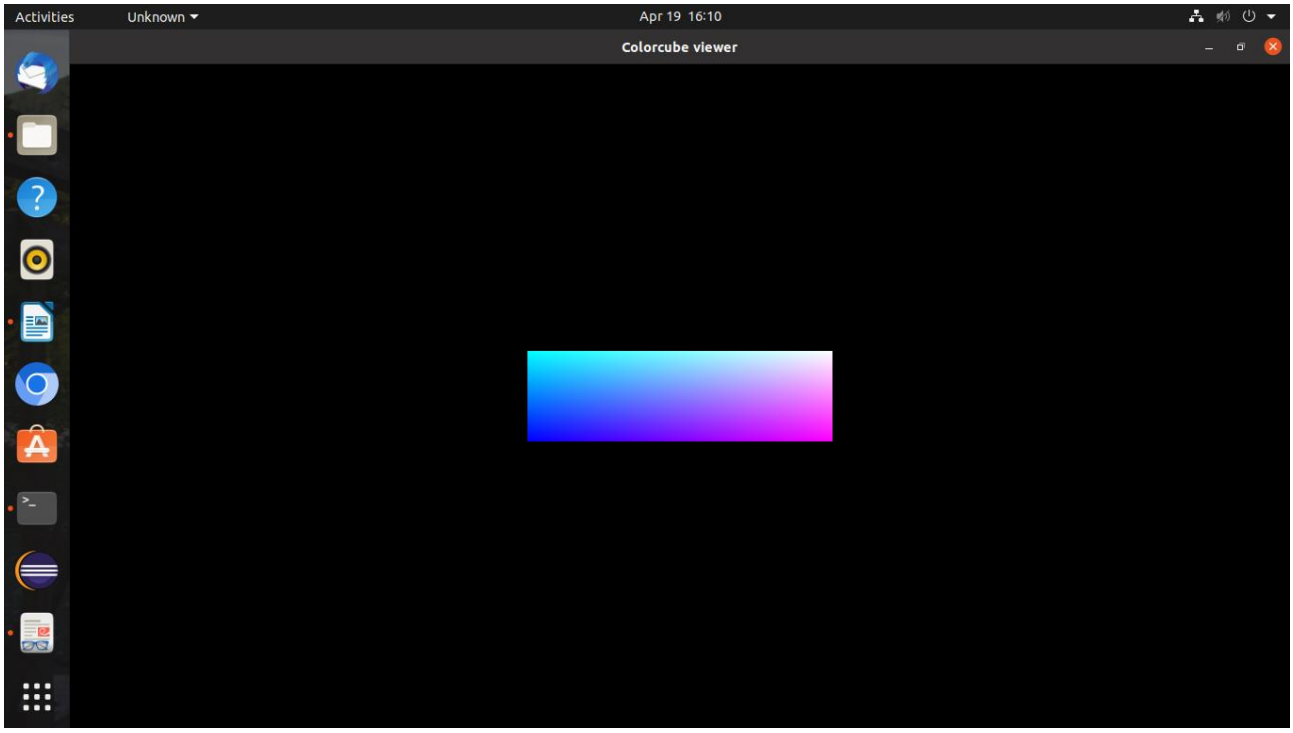
```
glutReshapeFunc(myReshape);
glutDisplayFunc(display);
glutIdleFunc(spinCube);
glutMouseFunc(mouse);
glEnable(GL_DEPTH_TEST);
glutMainLoop();
}
```

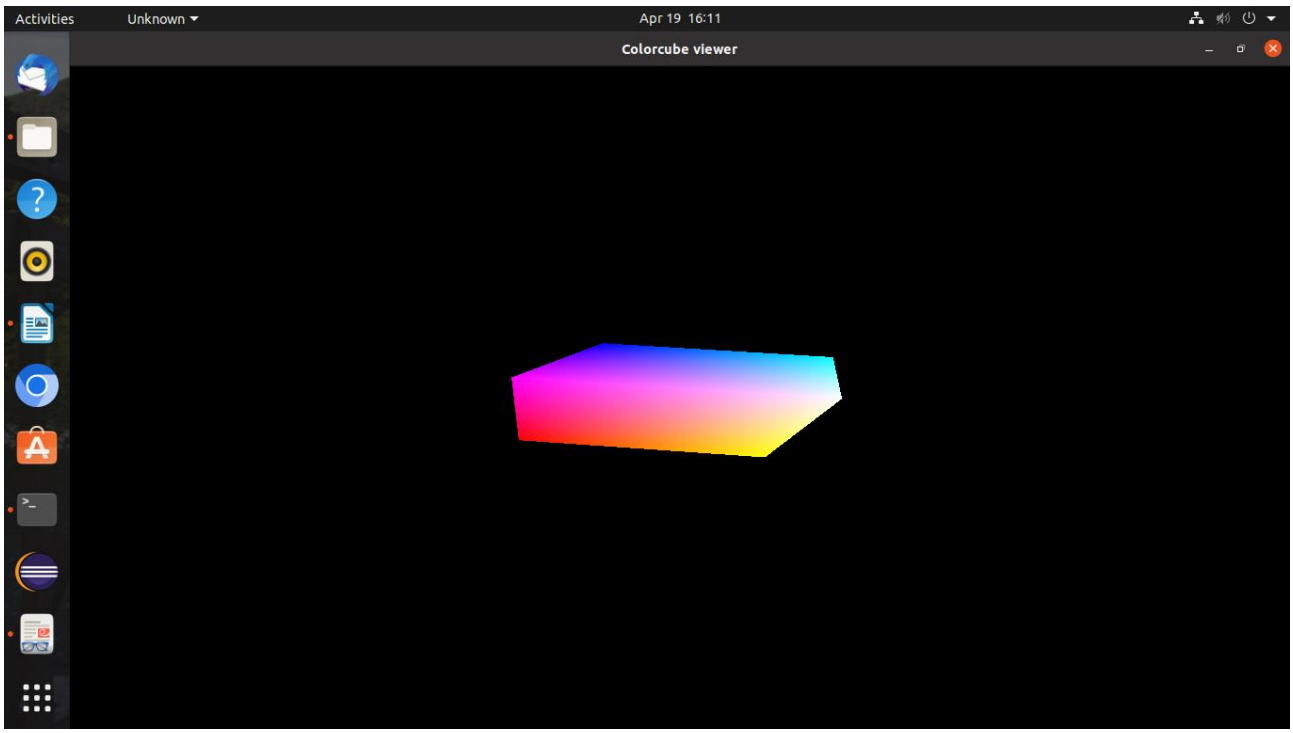
Program 4 computer Graphics

```
#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};
void display(void)
{
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
gluLookAt(viewer[0],viewer[1],viewer[2],0.0,0.0,0.0,0.0,1.0,0.0);
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)
{
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[0]+=1.0;
display();
}
void myReshape(int w,int h)
{
glViewport(0,0,w,h);
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);
glMatrixMode(GL_MODELVIEW);
}
int 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();
}
```


Activities
Unknown
Apr 12 16:10
Rotating a Color Cube


Activities
Unknown
Apr 19 16:10
Colorcube viewer


Activities
Unknown
Apr 19 16:11
Colorcube viewer

5. Program to clip a line using Cohen-Sutherland line-clipping algorithm.

```
#include <stdbool.h>
#include <stdio.h>
#include <GL/glut.h>
double xmin = 50, ymin = 50, xmax = 100, ymax = 100;
double xvmin = 200, yvmin = 200, xvmax = 300, yvmax = 300;
const int LEFT = 1;
const int RIGHT = 2;
const int BOTTOM = 4;
const int TOP = 8;
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 CohenSutherland(double x0, double y0, double x1, double y1)
{
 int outcode0, outcode1, outcodeOut;
 bool accept = false, done = false;
 outcode0 = ComputeOutCode (x0, y0);
 outcode1 = ComputeOutCode (x1, y1);
 do
 {
 if ( ! (outcode0 | outcode1))
 {
 accept = true;
 done = true;
 }
 else if (outcode0 & outcode1)
 done = true; //both are outside
 else
 {
 double x, y;
 double m = (y1 - y0) / (x1 - x0);
 outcodeOut = outcode0 ? outcode0: outcode1;
if (outcodeOut & TOP)
 {
 x = x0 + (1/m) * (ymax - y0);
 y = ymax;
 }
 else if (outcodeOut & BOTTOM)
 {
 x = x0 + (1/m) * (ymin - y0);
 y = ymin;
```

```
 }
 else if (outcodeOut & RIGHT)
 {
 y = y0 + m * (xmax - x0);
 x = xmax;
 }
 else
 {
 y = y0 + m * (xmin - x0);
 x = xmin;
 }


 if (outcodeOut == outcode0)
 {
 x0 = x;
 y0 = y;
 outcode0 = ComputeOutCode (x0, y0);
 }
 else
 {
 x1 = x;
 y1 = y;
 outcode1 = ComputeOutCode (x1, y1);
 }
 }
 }
 while (!done);

 if (accept)
 {
 double sx = (xvmax - xvmin) / (xmax - xmin);
 double sy = (yvmax - yvmin) / (ymax - ymin);
 double vx0 = xvmin + (x0 - xmin) * sx;
 double vy0 = yvmin + (y0 - ymin) * sy;
 double vx1 = xvmin + (x1 - xmin) * sx;
 double vy1 = yvmin + (y1 - ymin) * sy;
glBegin(GL_LINE_LOOP);
 glVertex2f (xvmin, yvmin);
 glVertex2f (xvmax, yvmin);
 glVertex2f (xvmax, yvmax);
 glVertex2f (xvmin, yvmax);
 glEnd();
 glBegin(GL_LINES);
 glVertex2d (vx0, vy0);
 glVertex2d (vx1, vy1);
 glEnd();
 }
}
void display()
{
 double x0 = 60, y0 = 20, x1 = 80, y1 = 120;
```

```
 glClear (GL_COLOR_BUFFER_BIT);
 glColor3f(1, 1, 1);
 glBegin (GL_LINES);
 glVertex2d (x0, y0);
 glVertex2d (x1, y1);
 glEnd ();
 glBegin (GL_LINE_LOOP);
 glVertex2f (xmin, ymin);
 glVertex2f (xmax, ymin);
 glVertex2f (xmax, ymax);
 glVertex2f (xmin, ymax);
 glEnd ();
 CohenSutherland (x0, y0, x1, y1);
 glFlush ();
}
void init()
{
 glClearColor (0, 0, 0, 1);
gluOrtho2D (0, 500, 0, 500);
}
int main(int argc, char **argv)
{
 glutInit (&argc,argv);
 glutInitDisplayMode (GLUT_SINGLE|GLUT_RGB);
 glutInitWindowSize (500, 500);
 glutInitWindowPosition (0, 0);
 glutCreateWindow ("Cohen Sutherland Line Clipping Algorithm");
 init();
 glutDisplayFunc(display);
 glutMainLoop();
}
```

6.

```
#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();
}
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.

```
#include<stdlib.h>
#include<stdio.h>
#include<GL/glut.h>
typedef float point[3];
point v[]={{0.0,0.0,1.0},{0.0,1.0,0.0},
{-1.0,-0.5,0.0}, {1.0,-0.5,0.0}};
int n;
void triangle(point a,point b,point c)
{
glBegin(GL_POLYGON);
glVertex3fv(a);
glVertex3fv(b);
glVertex3fv(c);
glEnd();
}
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]=(c[j]+b[j])/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(0.0,0.0,0.0);
divide_triangle(v[0],v[2],v[3],m);
}
void display(void)
{
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
tetrahedron(n);
glFlush();
```

```
}
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)
{
printf("No of Division?: ");
scanf("%d",&n);
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
glutCreateWindow("3D gasket");
glutReshapeFunc(myReshape);
glutDisplayFunc(display);
glEnable(GL_DEPTH_TEST);
glClearColor(1.0,1.0,1.0,0.0);
glutMainLoop();
}
```

8.

```
#include<GL/glut.h>
#include<stdio.h>
#include<math.h>
#define PI 3.1416
static int win,val=0,CMenu;
void CreateMenu(void);
void Menu(int value);
struct wcPt3D
{
GLfloat x, y, z;
};
GLsizei winWidth = 600, winHeight = 600;
GLfloat xwcMin = 0.0, xwcMax = 130.0;
GLfloat ywcMin = 0.0, ywcMax = 130.0;
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,struct wcPt3D *bezPt, GLint nCtrlPts,struct wcPt3D *ctrlPts,
GLint *C)
{
GLint k, n=nCtrlPts-1;
GLfloat bezBlendFcn;
bezPt ->x =bezPt ->y = bezPt->z=0.0;
for(k=0; k< nCtrlPts; k++)
{
bezBlendFcn = C[k] * pow(u, k) * pow( 1-u, n-k);
bezPt ->x += ctrlPts[k].x * bezBlendFcn;
bezPt ->y += ctrlPts[k].y * bezBlendFcn;
bezPt ->z += ctrlPts[k].z * bezBlendFcn;
}
}
void bezier(struct wcPt3D *ctrlPts, GLint nCtrlPts, GLint nBezCurvePts)
{
struct wcPt3D bezCurvePt;
GLfloat u;
GLint *C, k;
C= new GLint[nCtrlPts];
bino(nCtrlPts-1, C);
glBegin(GL_LINE_STRIP);
for(k=0; k<=nBezCurvePts; k++)
{
u=GLfloat(k)/GLfloat(nBezCurvePts);
```

```
computeBezPt(u, &bezCurvePt, nCtrlPts, ctrlPts, C);
glVertex2f(bezCurvePt.x, bezCurvePt.y);
}
glEnd();
delete[]C;
}
void displayFcn()
{
GLint nCtrlPts = 4, nBezCurvePts =20;
static float theta = 0;
struct 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);

if(val==1){
glPushMatrix();
glLineWidth(5);
glColor3f(1.0,0.5,0);
for(int i=0;i<8;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
glColor3f(1,1,1);
for(int i=0;i<8;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
glColor3f(0,1.0,0);
for(int i=0;i<8;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
glPopMatrix();
glColor3f(0.7, 0.5,0.3);
glLineWidth(5);
glBegin(GL_LINES);
glVertex2f(20,100);
glVertex2f(20,40);
glEnd();
glFlush();
}
```

```
if(val==2){
glPushMatrix();
glLineWidth(5);
glColor3f(1.0, 1.0, 0.0);
for(int i=0;i<12;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
glColor3f(1, 0.0, 0.0);
for(int i=0;i<12;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
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 CreateMenu(void)
{
CMenu= glutCreateMenu(Menu);
glutAddMenuEntry("Indian Flag",1);
glutAddMenuEntry("Karnataka Flag",2);
glutAddMenuEntry("Exit",0);
glutAttachMenu(GLUT_RIGHT_BUTTON);
}
void Menu(int value)
{
if(value==0)
{
glutDestroyWindow(win);
exit(0);
}
else {
```

```
val=value;
}
}
int main(int argc, char **argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
glutInitWindowPosition(50, 50);
glutInitWindowSize(winWidth, winHeight);
glutCreateWindow("Prg. 8 Bezier Curve");
CreateMenu();
glutDisplayFunc(displayFcn);
glutReshapeFunc(winReshapeFun);
glutMainLoop();
}
```

9.

```
#include <stdlib.h>
#include <stdio.h>
#include <GL/glut.h>
float x1,x2,x3,x4,y11,y2,y3,y4;
int fillFlag=0;
void edgedetect(float x1,float y11,float x2,float y2,int *le,int *re)
{
float mx,x,temp;
int i;
if((y2-y11)<0)
{
temp=y11;y11=y2;y2=temp;
temp=x1;x1=x2;x2=temp; }
if((y2-y11)!=0)
mx=(x2-x1)/(y2-y11);
else mx=x2-x1;
x=x1;
for(i=y11;i<=y2;i++)
{
if(x<(float)le[i])
le[i]=(int)x;
if(x>(float)re[i])
re[i]=(int)x;
x+=mx;
}
}
void draw_pixel(int x,int y)
{ glColor3f(1.0,1.0,0.0);
glBegin(GL_POINTS);
glVertex2i(x,y);
glEnd();
}
void scanfill(float x1,float y11,float x2,float y2,float x3,float y3,float x4,float y4)
{
int le[500],re[500];
int i,y;
for(i=0;i<500;i++)
{
le[i]=500;
re[i]=0;
}
edgedetect(x1,y11,x2,y2,le,re);
edgedetect(x2,y2,x3,y3,le,re);
edgedetect(x3,y3,x4,y4,le,re);
edgedetect(x4,y4,x1,y11,le,re);
for(y=0;y<500;y++)
{
for(i=(int)le[y];i<(int)re[y];i++)
draw_pixel(i,y);
}
}
```

```
void display()
{
x1=200.0;y11=200.0;x2=100.0;y2=300.0;x3=200.0;y3=400.0;x4=300.0;y4=300.0;
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0, 0.0, 1.0);
glBegin(GL_LINE_LOOP);
glVertex2f(x1,y11);
glVertex2f(x2,y2);
glVertex2f(x3,y3);
glVertex2f(x4,y4);
glEnd();
if(fillFlag==1)
scanfill(x1,y11,x2,y2,x3,y3,x4,y4);
glFlush();
}
void init()
{
glClearColor(0.0,0.0,0.0,1.0);
glColor3f(1.0,0.0,0.0);
glPointSize(1.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,499.0,0.0,499.0);
}
void fillMenu(int option)
{
if(option==1)
fillFlag=1;
if(option==2)
fillFlag=2;
display();
}
void main(int argc, char* argv[])
{ glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("Filling a Polygon using Scan-line Algorithm");
init();
glutDisplayFunc(display);
glutCreateMenu(fillMenu);
glutAddMenuEntry("Fill Polygon",1);
glutAddMenuEntry("Empty Polygon",2);
glutAttachMenu(GLUT_RIGHT_BUTTON);
glutMainLoop();
}
```