Program 1

```
#include <GL/glut.h>
#include <stdio.h>
#include <math.h>
void init(void)
{
glClearColor(1.0,1.0,1.0,0.0);
glMatrixMode(GL_PROJECTION);
gluOrtho2D(0.0,400.0,0.0,400.0);
}
void setPixel(GLint x,GLint y)
{
glBegin(GL_POINTS);
glVertex2i(x,y);
glEnd();
}
void line()
{
int x0 = 50, y0=50, xn = 300, yn = 150, x, y;
int dx, dy, pk, k; //looping variable
glClear(GL_COLOR_BUFFER_BIT);
glColor3f( 1 ,0, 0);
setPixel(x0, y0); //plot first point
// difference between starting and ending points
dx = xn - x0;
dy = yn - y0;
pk = 2 * dy - dx;
x = x0; y = y0;
for ( k = 0; k < dx-1; ++k )
{
```

```
if ( pk < 0 )
{
pk = pk + 2 * dy; //calculate next pk
//next pixel: (x+1, y )
}
else
{
pk = pk + 2*dy - 2*dx; //calculate next pk
++y;
}
++x;
setPixel( x, y );
}
glFlush();
}
int main(int argc,char **argv){
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowPosition(0,0);
glutInitWindowSize(500,500);
glutCreateWindow("Bresenham Line");
init();
glutDisplayFunc( line );
glutMainLoop();
return 0;
}
```

Program 2

```
#include<windows.h>
#include<stdio.h>
#include<GL/glut.h>
float house[5][2]={{200,100},{400,100},{300,300}};
float h=200,k=100,theta;
void drawtriangle()
{
glBegin(GL_LINE_LOOP);
glVertex2fv(house[0]);
glVertex2fv(house[1]);
glVertex2fv(house[2]);
glEnd();
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
drawtriangle();
glTranslatef(h,k,0);
glRotatef(theta,0,0,1);
glTranslatef(-h,-k,0);
drawtriangle();
glFlush();
}
void init()
{
gluOrtho2D(-700,700,-700,700);
}
int main(int argc,char** argv)
{
```

```
printf("Enter the rotation angle.\n");
scanf("%f",&theta);
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE);
glutInitWindowPosition(600,100);
glutInitWindowSize(350,350);
glutCreateWindow("house:C Tathva");
glutDisplayFunc(display);
init();
glutMainLoop();
}
```

Program 3

```
#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}}; //VERTICES OF THE CUBE
GLfloat colors[][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}}; //COLOR ASSOCIATED WITH EACHVERTEX
GLubyte cubeIndices[]={0,3,2,1,2,3,7,6,0,4,7,3,1,2,6,5,4,5,6,7,0,1,5,4};
static GLfloat theta[]={0.0,0.0,0.0};
static GLint axis=2;
void display()
{
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);
glDrawElements(GL_QUADS,24,GL_UNSIGNED_BYTE,cubeIndices);
glutSwapBuffers();
glFlush();
}
void spincube()
{
 theta[axis]+=2.0; if(theta[axis]>360.0)
 {
 theta[axis]-=360.0;
 }
display();
}
```

```
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 init()
{
glMatrixMode(GL_PROJECTION);
glOrtho(-2.0,2.0,-2.0,2.0,-10.0,10.0);
glMatrixMode(GL_MODELVIEW);
}
int main(int argc,char **argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DOUBLE);
glutInitWindowSize(600,600);
glutCreateWindow("Spin a colorcube");
init();
glutDisplayFunc(display);
glutIdleFunc(spincube);
glutMouseFunc(mouse);
glEnable(GL_DEPTH_TEST);
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3,GL_FLOAT,0,vertices);
glEnableClientState(GL_COLOR_ARRAY);
glColorPointer(3,GL_FLOAT,0,colors);
glutMainLoop();
}
```

Program 4

```
#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}}; //VERTICES OF THE CUBE
GLfloat colors[][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}}; //COLOR ASSOCIATED WITH EACH VERTEX

GLubyte cubeIndices[]={0,3,2,1,2,3,7,6,0,4,7,3,1,2,6,5,4,5,6,7,0,1,5,4
};
static GLfloat theta[]={0.0,0.0,0.0};
static GLint axis=2;
static GLint viewer[]={0.0,0.0,5.0};
void display()
{ 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);
glDrawElements(GL_QUADS,24,GL_UNSIGNED_BYTE,cubeIndices);
glutSwapBuffers();
glFlush();
}
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]=theta[axis]+2.0;
 if(theta[axis]>360.0)
 {
 theta[axis]=theta[axis]-360.0;
 display();
 }
}
void keys(unsigned char key,int x,int y)
{
if(key=='x')viewer[0]=viewer[0]-1.0;
if(key=='X')viewer[0]=viewer[0]+1.0;
if(key=='y')viewer[1]=viewer[1]-1.0;
if(key=='Y')viewer[1]=viewer[1]+1.0;
if(key=='z')viewer[2]=viewer[2]-1.0;
if(key=='Z')viewer[2]=viewer[2]+1.0;
display();
}
void init()
{
glMatrixMode(GL_PROJECTION);
glFrustum(-1,1,-1,1,2,10);
glMatrixMode(GL_MODELVIEW);
}
int main(int argc,char **argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DOUBLE);
glutInitWindowSize(600,600);
glutCreateWindow("Colorcube viewer");
init();
```

```
glutDisplayFunc(display);
glutMouseFunc(mouse);
glutKeyboardFunc(keys);
glEnable(GL_DEPTH_TEST);
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3,GL_FLOAT,0,vertices);
glEnableClientState(GL_COLOR_ARRAY);
glColorPointer(3,GL_FLOAT,0,colors);
glutMainLoop();
}
```

Program 5

```
#include<stdio.h>
#include<stdlib.h>
#include<stdio.h>
#include<GL/glut.h>
#define outcode int
double x0,y0,x1,y1;
double xmax,ymax,xmin,ymin;
double xvmin,yvmin,xvmax,yvmax;
const int TOP=1;
const int BOTTAM=2;
const int RIGHT=4;
const int LEFT=8;
outcode computecode(double x,double y)
{
outcode code=0;
if(y>ymax)
code |=TOP;
if(y<ymin)
code |=BOTTAM;
if(x>xmax)
code |=LEFT;
return code;
}
void cohenclipanddraw(double x0,double y0,double x1,double y1)
{
outcode outcode0,outcode1,outcodeout;
double sx,sy,vx0,vy0,vx1,vy1;
int accept=0,done=0;
outcode0=computecode(x0,y0);
```

```
outcode1=computecode(x1,y1);
do
{
if(!(outcode0|outcode1))
{
accept=1;
done=1;
}
else if(outcode0&outcode1)
done=1;
else
{
double x,y;
outcodeout=outcode0?outcode0:outcode1;
if(outcodeout&TOP)
{
x=x0+(x1-x0)*(ymax-y0)/(y1-y0);
y=ymax;
}
else if(outcodeout&BOTTAM)
{
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=computecode(x0,y0);
}
else
{
x1=x;y1=y;
outcode1=computecode(x1,y1);
}
}
}
while(!done);
glColor3f(0.0,1.0,0.0);
glBegin(GL_LINE_LOOP);
glVertex2f(xvmin,yvmin);
glVertex2f(xvmax,yvmin);
glVertex2f(xvmax,yvmax);
glVertex2f(xvmin,yvmax);
glEnd();
if(accept)
{
sx=(xvmax-xvmin)/(xmax-xmin);
sy=(yvmax-yvmin)/(ymax-ymin);
vx0=xvmin+(x0-xmin)*sx;
vy0=yvmin+(y0-ymin)*sy;
```

```
vx1=xvmin+(x1-xmin)*sx;
vy1=yvmin+(y1-ymin)*sy;
glColor3f(1.0,0.0,0.0);
glBegin(GL_LINES);
glVertex2f(vx0,vy0);
glVertex2f(vx1,vy1);
glEnd();
}
}
void myinit()
{
glClearColor(1.0,1.0,1.0,1.0);
glColor3f(1.0,0.0,0.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,500.0,0.0,500.0);
glMatrixMode(GL_MODELVIEW);
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0,0.0,0.0);
glBegin(GL_LINES);
glVertex2f(x0,y0);
glVertex2f(x1,y1);
glEnd();
glColor3f(0.0,0.0,1.0);
glBegin(GL_LINE_LOOP);
glVertex2f(xmin,ymin);
glVertex2f(xmax,ymin);
```

```
glVertex2f(xmax,ymax);
glVertex2f(xmin,ymax);
glEnd();
cohenclipanddraw(x0,y0,x1,y1);
glFlush();
}
int main(int argc,char **argv)
{
printf("enter the starting point\n");
scanf("%lf%lf",&x0,&y0);
printf("enter the end point\n");
scanf("%lf%lf",&x1,&y1);
printf("enter the lower left of window\n");
scanf("%lf%lf",&xmin,&ymin);
printf("enter the upper right of window\n");
scanf("%lf%lf",&xmax,&ymax);
printf("enter the lower left of Viewport\n");
scanf("%lf%lf",&xvmin,&yvmin);
printf("enter the lower left of viewport\n");
scanf("%lf%lf",&xvmax,&yvmax);
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("SutherLand");
glutDisplayFunc(display);
myinit();
glutMainLoop();
}
```

Program 6

```
#include <GL/glut.h>
#include <stdio.h>
#include <stdlib.h>
void wall(double thickness)
{
glPushMatrix();
glTranslated(0.5,0.5*thickness,0.5);
glScaled(1.0,thickness,1.0);
glutSolidCube(1.0);
glPopMatrix();
}
void tableleg(double thick,double len)
{
glPushMatrix();
glTranslated(0,len/2,0);
glScaled(thick,len,thick);
glutSolidCube(1.0);
glPopMatrix();
}
void table(double topw,double topt,double legt,double legl)
{
glPushMatrix();
glTranslated(0,legl,0);
glScaled(topw,topt,topw);
glutSolidCube(1.0);
glPopMatrix();
double dist=0.95*topw/2.0-legt/2.0;
glPushMatrix();
glTranslated(dist,0,dist);
```

```
tableleg(legt,legl);
glTranslated(0,0,-2*dist);
tableleg(legt,legl);
glTranslated(-2*dist,0,2*dist);
tableleg(legt,legl);
glTranslated(0,0,-2*dist);
tableleg(legt,legl);
glPopMatrix();
}
void displaysolid(void)
{
GLfloat mat_ambient[]={0.7f,0.7f,0.7f,1.0f};
GLfloat mat_diffuse[]={0.5f,0.5f,0.5f,1.0f};
GLfloat mat_specular[]={1.0f,1.0f,1.0f,1.0f};
GLfloat mat_shininess[]={50.0f};
glMaterialfv(GL_FRONT,GL_AMBIENT,mat_ambient);
glMaterialfv(GL_FRONT,GL_DIFFUSE,mat_diffuse);
glMaterialfv(GL_FRONT,GL_SPECULAR,mat_specular);
glMaterialfv(GL_FRONT,GL_SHININESS,mat_shininess);
GLfloat lightint[]={0.7f,0.7f,0.7f,1.0f};
GLfloat lightpos[]={2.0f,6.0f,3.0f,0.0f};
glLightfv(GL_LIGHT0,GL_POSITION,lightpos);
glLightfv(GL_LIGHT0,GL_DIFFUSE,lightint);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
double winht=1.0;
glOrtho(-winht*64/48.0,winht*64/48.0,-winht,winht,0.1,100.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(2.3,1.3,2.0,0.0,0.25,0.0,0.0,1.0,0.0);
```

```
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glRotated(90.0,0.0,0.0,1.0);
wall(0.02); glPopMatrix();
wall(0.02); glPushMatrix();
glRotated(-90.0,1.0,0.0,0.0);
wall(0.02);
glPopMatrix();
glPushMatrix();
glTranslated(0.4,0,0.4);
table(0.6,0.02,0.02,0.3);
glPopMatrix();
glPushMatrix();
glTranslated(0.6,0.38,0.5);
glRotated(30,0,1,0);
glutSolidTeapot(0.08);
glPopMatrix(); glFlush();
}
int main(int argc,char**argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("teapot");
glutDisplayFunc(displaysolid);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glShadeModel(GL_SMOOTH);
glEnable(GL_DEPTH_TEST);
```

```
glEnable(GL_NORMALIZE);
glClearColor(0.1,0.1,0.1,0.0);
glViewport(0,0,640,480);
glutMainLoop();
}
```

Program 7

```
#include <GL/glut.h>
#include <stdlib.h>
#include<stdio.h>
typedef float point[3];
point v[]={ {0.0,0.0,1.0},
 {0.0,0.943,-0.33},
 {-0.816,-0.471,-0.33},
 {0.816,-0.471,0.33}};
int n;
void triangle(point a,point b,point c)
{
glBegin(GL_POLYGON);
glNormal3fv(a);
glVertex3fv(a);
glVertex3fv(b);
glVertex3fv(c);
glEnd();
}
void divide_tri(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_tri(a,v1,v2,m-1);
 divide_tri(c,v2,v3,m-1);
 divide_tri(b,v3,v1,m-1);
}
else
 triangle(a,b,c);
}
void tetrahedron(int m)
{
glColor3f(1.0,0.0,0.0);
divide_tri(v[0],v[1],v[2],m);
glColor3f(0.0,1.0,0.0);
divide_tri(v[3],v[2],v[1],m);
glColor3f(0.0,0.0,1.0);
divide_tri(v[0],v[3],v[1],m);
glColor3f(0.0,0.0,0.0);
divide_tri(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();
}
int main(int argc,char **argv)
{
printf("Enter the number of recursive steps you want\n");
 scanf("%d", &n);
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(500,500);
glutCreateWindow("3d gasket");
glutReshapeFunc(myReshape);
glutDisplayFunc(display);
glEnable(GL_DEPTH_TEST);
glClearColor(1.0,1.0,1.0,1.0);
glutMainLoop();
}
```

Program 8

```
#include<GL/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;
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 computeBezPt(GLfloat u, wcPt3D *bezPt, GLint nCtrlPts, 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(wcPt3D *ctrlPts, GLint nCtrlPts, GLint nBezCurvePts)
{
 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;
 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, nBezCurvePts);
}
glColor3f(1, 1, 1); //Indian flag: white color code
for(int i=0;i<8;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
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, 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);
}
int main(int argc, char **argv)
{
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
 glutInitWindowPosition(50, 50);
 glutInitWindowSize(winWidth, winHeight);
 glutCreateWindow("Bezier Curve");
 glutDisplayFunc(displayFcn);
 glutReshapeFunc(winReshapeFun);
 glutMainLoop();
}
```

Program 9

```
#include <stdlib.h>
#include <stdio.h>
#include <GL/glut.h>
#include <windows.h>
float x1,x2,x3,x4,y1,y2,y3,y4;
int fillFlag=0;
void edgedetect(float x1,float y1,float x2,float y2,int *le,int *re)
{
float mx,x,temp;
int i;
if((y2-y1)<0)
{
temp=y1;y1=y2;y2=temp;
temp=x1;x1=x2;x2=temp;
}
if((y2-y1)!=0)
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 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 y1,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,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);
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;y1=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,y1);
glVertex2f(x2,y2);
glVertex2f(x3,y3);
glVertex2f(x4,y4);
glEnd();
if(fillFlag==1)
scanfill(x1,y1,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();
}
int 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();
}
```