1) Program to display a set of values {fij} as a rectangular mesh.

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

#define maxx 20

#define maxy 20

#define dx 15

#define dy 20

GLfloat x[maxx]={0.0},y[maxy]={0.0};

GLfloat x0=50,y01=50;

GLint i,j;

void init()

{

glClearColor(1.0,1.0,1.0,1.0);

glPointSize(10.0);

glMatrixMode(GL_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0,499.0,0.0,499.0);

glutPostRedisplay();

}

void display(void)

{

glClear(GL_COLOR_BUFFER_BIT);
```

```
for(i=0;i<maxx;i++)
x[i]=x0+i*dx;
for(j=0;j<maxy;j++)
y[j]=y01+j*dy;
glColor3f(0.1,0.7,0.1);
for(i=0;i<maxx-1;i++)
{
for(j=0;j<maxy-1;j++)
{
glBegin(GL_LINE_LOOP);
glVertex2f(x[i],y[j]);
glVertex2f(x[i],y[j+1]);
glVertex2f(x[i+1],y[j+1]);
glVertex2f(x[i+1],y[j]);
glEnd();
glFlush();
}
}
glFlush();
}
int main(int argc,char **argv)
```

```
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("rectangular mesh");
glutDisplayFunc(display);
init();
glutMainLoop();
return 0;
}
```

2) Program to recursively sub-divide a tetrahedron to from 3D Sierpinski gasket. The number of recursive steps is to be specified by the user

```
#include<stdio.h>

#include<stdlib.h>

#include<GL/glut.h>

typedef float point[3];

point v[]={{0.0,0.0,0.0},{0.0,0.942,-0.333},{-0.816,-0.471,-0.333},{0.816,-0.471,-0.333}};

int n;

void triangle(point a,point b,point c)

{

glBegin(GL_POLYGON);

glVertex3fv(a);

glVertex3fv(b);

glVertex3fv(c);

glEnd();

}

void div_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;
div_triangle(a,v1,v2,m-1);
div_triangle(c,v2,v3,m-1);
div_triangle(b,v3,v1,m-1);
}
else (triangle(a,b,c));
}
void tetrahedron(int m)
{
glColor3f(0.9,0.8,0.4);
div_triangle(v[0],v[1],v[2],m);
glColor3f(0.8,0.4,0.3);
div_triangle(v[3],v[2],v[1],m);
glColor3f(0.4,0.8,0.7);
div_triangle(v[0],v[3],v[1],m);
```

```
glColor3f(0.3,0.3,0.3);
div_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,-100.0,100.0);
else
glOrtho(-2.0*(GLfloat)h/(GLfloat)w,2.0*(GLfloat)h/(GLfloat)w,-2.0,2.0,-10.0,10.0);
glMatrixMode(GL_MODELVIEW);
glutPostRedisplay();
```

```
}
int main(int argc,char **argv)
{
printf("entr the no of div\n");
scanf("%d",&n);
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(500.0,500.0);
glutInitWindowPosition(0.0,0.0);
glutCreateWindow("3D gasket");
glutReshapeFunc(myReshape);
glutDisplayFunc(display);
glEnable(GL_DEPTH_TEST);
glClearColor(1.0,1.0,1.0,1.0);
glutMainLoop();
return 0;
}
```

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

```
#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(void)
```

```
{
polygon(0,1,2,3);
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]+=0.8;
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,-100.0,100.0);
else
glOrtho(-2.0*(GLfloat)h/(GLfloat)w,2.0*(GLfloat)h/(GLfloat)w,-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("color cube");
glutReshapeFunc(myReshape);
glutDisplayFunc(display);
glutIdleFunc(spincube);
glutMouseFunc(mouse);
glEnable(GL_DEPTH_TEST);
glutMainLoop();
return 0;
}
```

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

```
#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(void)
{
polygon(0,1,2,3);
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 spincube()
{
theta[axis]+=0.8;
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 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[2]+=1.0;
display();
}
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,-100.0,100.0);
else
glOrtho(-2.0*(GLfloat)h/(GLfloat)w,2.0*(GLfloat)h/(GLfloat)w,-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(600,600);
glutCreateWindow("color cube-perspective");
glutReshapeFunc(myReshape);
glutDisplayFunc(display);
glutIdleFunc(spincube);
glutMouseFunc(mouse);
glutKeyboardFunc(keys);
glEnable(GL_DEPTH_TEST);
glutMainLoop();
return 0;
}
```

5) Program to create a cylinder and a parallelepiped by extruding a circle and quadrilateral respectively. Allow the user to specify the circle and the quadrilateral.

```
#include<stdio.h>

#include<math.h>

#include<GL/glut.h>

void draw_pixels(GLint cx,GLint cy)

{

glColor3f(1.0,0.0,0.0);

glBegin(GL_POINTS);

glVertex2i(cx,cy);

glEnd();

}

void plot_pixels(GLint h,GLint k,GLint x,GLint y)

{

draw_pixels(x+h,y+k);

draw_pixels(-x+h,y+k);

draw_pixels(x+h,-y+k);

draw_pixels(-x+h,-y+k);

draw_pixels(y+h,x+k);

draw_pixels(-y+h,x+k);

draw_pixels(y+h,-x+k);
```

```
draw_pixels(-y+h,-x+k);
}
void circledraw(GLint h,GLint k,GLint r)
{
GLint d=1-r,x=0,y=r;
while(y>x)
{
plot_pixels(h,k,x,y);
if(d<0)
d+=2*x+3;
else
{
d+=2*(x-y)+5;
--y;
}
++x;
}
plot_pixels(h,k,x,y);
}
void cyldraw()
{
```

```
GLint xc=100,yc=100,r=50,i,n=50;
for(i=0;i<n;i+=3)
{
circledraw(xc,yc+i,r);
}
}
void parpiped(int x1,int x2,int y1,int y2,int y3,int y4)
{
glColor3f(0.0,0.0,1.0);
glPointSize(2.0);
glBegin(GL_LINE_LOOP);
glVertex2i(x1,y1);
glVertex2i(x2,y3);
glVertex2i(x2,y4);
glVertex2i(x1,y2);
glEnd();
}
void parpipeddraw()
{
int x1=200,x2=300,y1=100,y2=175,y3=100,y4=175;
GLint i,n=40;
```

```
for(i=0;i<=n;i+=2)
{
parpiped(x1+i,x2+i,y1+i,y2+i,y3+i,y4+i);
}
}
void init(void)
{
glClearColor(1.0,1.0,1.0,1.0);
glMatrixMode(GL_PROJECTION);
gluOrtho2D(0.0,499.0,0.0,499.0);
}
void display(void)
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0,0.0,0.0);
glPointSize(2.0);
cyldraw();
parpipeddraw();
glFlush();
}
int main(int argc,char **argv)
```

```
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("cyl nd parpipe");
init();
glutDisplayFunc(display);
glutMainLoop();
return 0;
}
```

6) Program to implement the Cohen-Sutherland line-clipping algorithm. Make provision to specify the input line, window for clipping and view port for displaying the clipped image.

```
#include<stdlib.h>

#include<GL/glut.h>

#define outcode int

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

double xvmin=200,yvmin=200,xvmax=300,yvmax=300;

const int RIGHT=8;

const int LEFT=2;

const int TOP=4;

const int BOTTOM=1;

outcode computeoutcode(double x,double y);

void cohensu(double x0,double y0,double x1,double y1)

{

outcode outcode0,outcode1,outcodeout;

int accept=0,done=0;

outcode0=computeoutcode(x0,y0);

outcode1=computeoutcode(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 & 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 if(outcodeout & LEFT)
{
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)
{
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;
glColor3f(0.0,1.0,0.0);
glBegin(GL_LINE_LOOP);
glVertex2f(xvmin,yvmin);
glVertex2f(xvmax,yvmin);
glVertex2f(xvmax,yvmax);
glVertex2f(xvmin,yvmax);
glEnd();
glColor3f(0.1,0.0,1.0);
glBegin(GL_LINES);
glVertex2f(vx0,vy0);
```

```
glVertex2f(vx1,vy1);
glEnd();
}
}
outcode computeoutcode(double x,double y)
{
outcode code=0;
if(y>ymax)
code |=TOP;
if(y<ymin)
code |=BOTTOM;
if(x>xmax)
code |=RIGHT;
if(x<xmin)
code |=LEFT;
return code;
}
void display()
{
double x0=60,y0=20,x1=80,y1=120;
glClear(GL_COLOR_BUFFER_BIT);
```

```
glColor3f(0.1,0.2,1.0);
glBegin(GL_LINES);
glVertex2f(x0,y0);
glVertex2f(x1,y1);
glEnd();
glBegin(GL_LINE_LOOP);
glVertex2d(xmin,ymin);
glVertex2d(xmax,ymin);
glVertex2d(xmax,ymax);
glVertex2d(xmin,ymax);
glEnd();
cohensu(x0,y0,x1,y1);
glFlush();
}
void myinit()
{
glClearColor(1.0,1.0,1.0,1.0);
glColor3f(1.0,0.0,0.0);
glPointSize(10.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
```

```
gluOrtho2D(0.0,499.0,0.0,499.0);

glutPostRedisplay();

}

int main(int argc,char ** argv)

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);

glutInitWindowSize(500,500);

glutInitWindowPosition(0,0);

glutCreateWindow("COHEN SUTHERLAND");

glutDisplayFunc(display);

myinit();

glutMainLoop();

return 0;

}
```

7) Program to create a house like figure and rotate it about a givenfixed point using OpenGL functions.

```
#include<stdio.h>

#include<math.h>

#include<GL/glut.h>

GLfloat house[3][9]={{250.0,250.0,325.0,400.0,400.0,300.0,300.0,350.0,350.0},

{250.0,400.0,475.0,400.0,250.0,250.0,300.0,300.0,250.0},

{1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0}};

GLfloat rot_mat[3][3]={{0},{0},{0}};

GLfloat result[3][9]={{0},{0},{0}};

GLfloat h=250.0;

GLfloat k=250.0;

GLfloat theta;

void multiply()

{

int i,j,l;

for(i=0;i<3;i++)

for(j=0;j<9;j++)

{

result[i][j]=0;

for(l=0;l<3;l++)
```

```
result[i][j]=result[i][j]+rot_mat[i][l]*house[l][j];
}
}
void rotate()
{
GLfloat m,n;
m=-h*(cos(theta)-1)+k*(sin(theta));
n=-k*(cos(theta)-1)-k*(sin(theta));
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]=0;
multiply();
}
void drawhouse()
{
```

```
glColor3f(1.0,0.0,0.0);
glBegin(GL_LINE_LOOP);
glVertex2f(house[0][0],house[1][0]);
glVertex2f(house[0][1],house[1][1]);
glVertex2f(house[0][3],house[1][3]);
glVertex2f(house[0][4],house[1][4]);
glEnd();
glColor3f(0.0,0.45,0.0);
glBegin(GL_LINE_LOOP);
glVertex2f(house[0][5],house[1][5]);
glVertex2f(house[0][6],house[1][6]);
glVertex2f(house[0][7],house[1][7]);
glVertex2f(house[0][8],house[1][8]);
glEnd();
glColor3f(0.0,0.0,0.25);
glBegin(GL_LINE_LOOP);
glVertex2f(house[0][1],house[1][1]);
glVertex2f(house[0][2],house[1][2]);
glVertex2f(house[0][3],house[1][3]);
glEnd();
}
```

```
void drawrotatedhouse()
{
glColor3f(1.0,0.0,0.0);
glBegin(GL_LINE_LOOP);
glVertex2f(result[0][0],result[1][0]);
glVertex2f(result[0][1],result[1][1]);
glVertex2f(result[0][3],result[1][3]);
glVertex2f(result[0][4],result[1][4]);
glEnd();
glColor3f(1.0,0.45,0.0);
glBegin(GL_LINE_LOOP);
glVertex2f(result[0][5],result[1][5]);
glVertex2f(result[0][6],result[1][6]);
glVertex2f(result[0][7],result[1][7]);
glVertex2f(result[0][8],result[1][8]);
glEnd();
glColor3f(1.0,0.0,0.25);
glBegin(GL_LINE_LOOP);
glVertex2f(result[0][1],result[1][1]);
glVertex2f(result[0][2],result[1][2]);
glVertex2f(result[0][3],result[1][3]);
```

```
glEnd();
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
drawhouse();
rotate();
drawrotatedhouse();
glFlush();
}
void init()
{
glClearColor(1.0,1.0,1.0,1.0);
glColor3f(0.5,0.1,0.0);
glPointSize(1.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,499.0,0.0,499.0);
}
int main(int argc,char *argv[])
{
```

```
printf("enter the angle in floating point\n");
scanf("%f",&theta);
theta=(3.14/180)*theta;
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("house rotated");
glutDisplayFunc(display);
init();
glutMainLoop();
return 0;
}
```

8) Program, using OpenGL functions, 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 properties of the surfaces of the solid object used in the scene.

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

void wall(double thick)

{

glPushMatrix();

glTranslated(0.5,0.5*thick,0.5);

glScaled(1.0,thick,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 topwid,double topthick,double legthick,double leglen)

{
```

```
glPushMatrix();
glTranslated(0,leglen,0);
glScaled(topwid,topthick,topwid);
glutSolidCube(1.0);
glPopMatrix();
double dist=0.95*topwid/2-legthick/2;
glPushMatrix();
glTranslated(dist,0,dist);
tableLeg(legthick,leglen);
glTranslated(0,0,-2*dist);
tableLeg(legthick,leglen);
glTranslated(-2*dist,0,2*dist);
tableLeg(legthick,leglen);
glTranslated(0,0,-2*dist);
tableLeg(legthick,leglen);
glPopMatrix();
}
void display(void)
{
GLfloat mat_ambient[]={0.7f,0.7f,0.7f,1.0f};
GLfloat mat_diffuse[]={.5f,.5f,.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();
glTranslated(0.6,0.38,0.5);
glRotated(30,0,1,0);
```

```
glutSolidTeapot(0.08);
glPopMatrix();
glPushMatrix();
glTranslated(0.4,0.0,0.4);
table(0.6,0.02,0.02,0.3);
glPopMatrix();
wall(0.02);
glPushMatrix();
glRotated(90.0,0,0,1.0);
wall(0.02);
glPopMatrix();
glPushMatrix();
glRotated(-90.0,1.0,0.0,0.0);
wall(0.02);
glPopMatrix();
glFlush();
}
int main(int argc,char **argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_DEPTH|GLUT_RGB);
```

```
glutInitWindowSize(640,480);
glutInitWindowPosition(100,100);
glutCreateWindow("teapot");
glutDisplayFunc(display);
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(0.0,0.0,640,480);
glutMainLoop();
return 0;
}
```

9) Program to implement Liang-Barsky line clipping algorithm.

```
#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;
int false=0,true=1;
int cliptest(double p,double q,double *t1,double *t2)
{
double t=q/p;
if(p<0.0)
{
if(t>*t1) *t1=t;
if(t>*t2) return(false);
}
else if(p>0.0)
{
if(t<*t2) *t2=t;
if(t<*t1) return(false);
}
else if(p==0.0)
{
```

```
if(q<0.0)
return(false);
}
return(true);
}
void LiangBarskyLineClipAndDraw(double x0,double y0,double x1,double y1)
{
double dx=x1-x0,dy=y1-y0,te=0.0,t1=1.0;
if(cliptest(-dx,x0-xmin,&te,&t1))
if(cliptest(dx,xmax-x0,&te,&t1))
if(cliptest(-dy,y0-ymin,&te,&t1))
if(cliptest(dy,ymax-y0,&te,&t1))
{
if(t1<1.0)
{
x1=x0+t1*dx;
y1=y0+t1*dy;
}
if(te>0.0)
{
x0=x0+te*dx;
```

```
y0=y0+te*dy;
}
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;
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);
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(1.0,0.0,0.0);
glBegin(GL_LINES);
glVertex2d(x0,y0);
glVertex2d(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();
LiangBarskyLineClipAndDraw(x0,y0,x1,y1);
glFlush();
}
void myinit()
```

```
{
glClearColor(1.0,1.0,1.0,1.0);
glColor3f(1.0,0.0,0.0);
glPointSize(50.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,499.0,0.0,499.0);
}
int main(int argc,char **argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("liong barsky line clipping alg..");
glutDisplayFunc(display);
myinit();
glutMainLoop();
return 0;
}
```

10) Program to fill any given polygon using scan-line area filling algorithm. (Use appropriate data structures.)

```
#include<stdlib.h>

#include<stdio.h>

#include<GL/glut.h>

float x1,x2,x3,x4,y11,y2,y3,y4;

void ed(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 dp(int x,int y,int val)
{
        glColor3f(1,1,0);
        glBegin(GL_POINTS);
        glVertex2i(x,y);
        glEnd();
}
void scan(float x1,float y11,float x2,float y2,float x3,float y3,float x4,float y4)
{
        int le[500],re[500];
        int i,j;
        for(i=0;i<500;i++)
        {
```

```
            le[i]=500;
            re[i]=0;
        }
        ed(x1,y11,x2,y2,le,re);
        ed(x2,y2,x3,y3,le,re);
        ed(x3,y3,x4,y4,le,re);
        ed(x4,y4,x1,y11,le,re);
        for(j=0;j<500;j++)
        {
            if(le[j]<=re[j])
                for(i=(int)le[j];i<(int)re[j];i++)
                    dp(i,j,0);
        }
}
void disp()
{
        x1=200;y11=200;
        x2=100;y2=300;
        x3=200;y3=400;
        x4=300;y4=300;
        glClear(GL_COLOR_BUFFER_BIT);
```

```
        glColor3f(0,0,0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(x1,y11);
        glVertex2f(x2,y2);
        glVertex2f(x3,y3);
        glVertex2f(x4,y4);
        glEnd();
        scan(x1,y11,x2,y2,x3,y3,x4,y4);
        glFlush();
}
void init()
{
        glClearColor(1,1,1,1);
        glColor3f(1,0,0);
        glPointSize(1);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0,499,0,499);
}
int main(int argc,char **argv)
{
```

```
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("scan");
    glutDisplayFunc(disp);
    init();
    glutMainLoop();
    return 0;
}
```