Program1: Program to recursively subdivide a tetrahedron to form 3D sierpinski gasket. The number of recursive steps is to be specified by the user.

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

typedef float point[3];
point v[]={{0.0,0.0,1.0},{0.0,0.942809,-0.33333},{-0.816497,-0.471405,-
0.333333},{0.816497,-0.471405,-0.333333}};
static GLfloat theta[]={0.0,0.0,0.0};
int n;
void triangle(point a,point b,point c)
{
        glBegin(GL_POLYGON);
        glNormal3fv(a);
        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]=(b[j]+c[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 divisions\n");
        scanf("%d",&n);
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
        glutInitWindowSize(500,500);
        glutCreateWindow("3dgasket");
        glutReshapeFunc(myReshape);
        glutDisplayFunc(display);
        glEnable(GL_DEPTH_TEST);
        glClearColor(1.0,1.0,1.0,1.0);
        glutMainLoop();
}
```

Program 2: Program to implement Liang-Barsky Line Clipping Algorithm.

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

double xmin=50,ymin=50, xmax=100,ymax=100; // Window boundaries
double xvmin=200,yvmin=200,xvmax=300,yvmax=300; // Viewport boundaries

int cliptest(double p, double q, double *t1, double *t2)
{
double t=q/p;
if(p < 0.0)    // potentially enry point, update te
  {
          if( t > *t1) *t1=t;
          if( t > *t2) return(false); // line portion is outside
  }
  else
  if(p > 0.0)    //  Potentially leaving point, update tl
  {
          if( t < *t2) *t2=t;
          if( t < *t1) return(false); // line portion is outside
  }
  else
          if(p == 0.0)
          {
                  if( q < 0.0) return(false); // line parallel to edge but outside
          }
 return(true);
}

void LiangBarskyLineClipAndDraw (double x0, double y0,double x1, double y1)
{
        double dx=x1-x0, dy=y1-y0, te=0.0, tl=1.0;
        if(cliptest(-dx,x0-xmin,&te,&tl))  // inside test wrt left edge
        if(cliptest(dx,xmax-x0,&te,&tl)) // inside test wrt right edge
        if(cliptest(-dy,y0-ymin,&te,&tl)) // inside test wrt bottom edge
        if(cliptest(dy,ymax-y0,&te,&tl)) // inside test wrt top edge
        {
                if( tl < 1.0 )
                {
                        x1 = x0 + tl*dx;
                        y1 = y0 + tl*dy;
                }
                if( te > 0.0 )
                {   x0 = x0 + te*dx;
                        y0 = y0 + te*dy;
```

```
                }

                        // 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();
        }
        }// end of line clipping


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();
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(1.0);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(0.0,499.0,0.0,499.0);
}
void main(int argc, char** argv)
{
	//int x1, x2, y1, y2;
	//printf("Enter End points:");
	//scanf("%d%d%d%d", &x1,&x2,&y1,&y2);

	glutInit(&argc,argv);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutInitWindowSize(500,500);
	glutInitWindowPosition(0,0);
	glutCreateWindow("Liang Barsky Line Clipping Algorithm");
	glutDisplayFunc(display);
	myinit();
	glutMainLoop();
}
```

Program 3: Program to draw a color cube and spin it using open GL transformation matrices.

```
#include<stdlib.h>
#include<GL/glut.h>
GLfloat vertices[]={-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[]={-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[]={0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0,1.0,0.0,1.0,0.0,1.0,1.0,1.0,
1.0,1.0,0.0,1.0,1.0};

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(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);
   glDrawElements(GL_QUADS,24,GL_UNSIGNED_BYTE,CubeIndices);
        glBegin(GL_LINES);
        glVertex3f(0.0,0.0,0.0);
        glVertex3f(1.0,1.0,1.0);
   glEnd();
        glFlush();
        glutSwapBuffers();
}

void SpinCube()
{
        theta[axis]+=5.0;
        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);
}


void main(int argc, char **argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
        glutInitWindowSize(500,500);
        glutCreateWindow("spin a colorcube");
        glutReshapeFunc(myReshape);
        glutDisplayFunc(display);
        glutIdleFunc(SpinCube);
        glutMouseFunc(mouse);
        glEnable(GL_DEPTH_TEST);
        glEnableClientState(GL_COLOR_ARRAY);
        glEnableClientState(GL_NORMAL_ARRAY);
        glEnableClientState(GL_VERTEX_ARRAY);
        glVertexPointer(3,GL_FLOAT,0,vertices);
        glColorPointer(3,GL_FLOAT,0,colors);
        glNormalPointer(GL_FLOAT,0,normals);
        glColor3f(1.0,1.0,1.0);
        glutMainLoop();

}
```

Program 4: Program to create a House like figure and rotate it about a given fixed point using open GL functions.

```
#include<stdio.h>
#include<math.h>
#include<GL/glut.h>
GLfloat house[3][9]={{100.0,100.0,175.0,250.0,250.0,150.0,150.0,200.0,200.0},{100.0,300.0,
400.0, 300.0,100.0,100.0,150.0,150.0,100.0},{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=100.0;
GLfloat k=100.0;
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)-h*(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]=1;
        multiply();
}
void drawhouse()
{
        glColor3f(0.0,0.0,1.0);
        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(1.0,0.0,0.0);
        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(0.0,0.0,1.0);
        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(0.0,0.0,1.0);
        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(1.0,0.0,0.0);
        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(0.0,0.0,1.0);
        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 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)
{
	printf("Enter the rotation angle\n");
		scanf("%f",&theta);
	theta=theta*3.142/180.0;
	glutInit(&argc,argv);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutInitWindowSize(500,500);
	glutInitWindowPosition(0,0);
	glutCreateWindow("HOUSE ROTATION");
	glutDisplayFunc(display);
	myinit();
	glutMainLoop();
}
```

Program 5: Program to implement the Cohen-Sutherland line-clipping algorithm. Make provision to specify the input line, window for clipping and viewport for displaying the clipped image.

```
#include<stdio.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 BOTTAM=1;
outcode ComputeOutCode(double x,double y);
void CohenSutherlandLineClipAndDraw(double x0,double y0,double x1,double y1)
{
	outcode 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;
		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=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(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();
        }
}
      outcode ComputeOutCode(double x,double y)
{
        outcode code=0;
   if(y>ymax)
```

```
                code |=TOP;
        else if(y<ymin)
                code |=BOTTAM;
        if(x>xmax)
                code |=RIGHT;
        else if(x<xmin)
                code |=LEFT;
        return code;
}

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();
  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("cohensuderland line clipping algorithm");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```

Program 6: program to create a cylinder and a parallel piped by extruding a circle and quadrilateral respectively. Allow the user to specify the circle and quadrilateral.

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

void draw_pixel(GLint cx, GLint cy)
{
        glColor3f(1.0,0.0,0.0);
        glBegin(GL_POINTS);
        glVertex2i(cx,cy);
        glEnd();
}

void plotpixels(GLint h, GLint k, GLint x, GLint y)
{
        draw_pixel(x+h,y+k);
        draw_pixel(-x+h,y+k);
        draw_pixel(x+h,-y+k);
        draw_pixel(-x+h,-y+k);
        draw_pixel(y+h,x+k);
        draw_pixel(-y+h,x+k);
        draw_pixel(y+h,-x+k);
        draw_pixel(-y+h,-x+k);
}
void Circle_draw(GLint h, GLint k, GLint r)  // Midpoint Circle Drawing Algorithm
{
        GLint d =  1-r, x=0, y=r;
        while(y > x)
        {
                plotpixels(h,k,x,y);
                if(d < 0) d+=2*x+3;
                else
                {d+=2*(x-y)+5;
                 --y;
                }
                ++x;
        }
        plotpixels(h,k,x,y);
}

void Cylinder_draw()
{
        GLint xc=100, yc=100, r=50;
        GLint i,n=50;
```

```
        for(i=0;i<n;i+=3)
        {
        Circle_draw(xc,yc+i,r);
        }
}
void parallelepiped(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 parallelepiped_draw()
{
        int x1=200,x2=300,y1=100,y2=175,y3=100,y4=175;
        GLint i,n=40;
         for(i=0;i<n;i+=2)
         {
         parallelepiped(x1+i,x2+i,y1+i,y2+i,y3+i,y4+i);
         }

}

void init(void)
{
        glClearColor(1.0,1.0,1.0,0.0);  // Set display window color to white
        glMatrixMode(GL_PROJECTION);  // Set Projection parameters
        gluOrtho2D(0.0,400.0,0.0,300.0);
}
void display(void)
{
        glClear(GL_COLOR_BUFFER_BIT);  // Clear Display Window
        glColor3f(1.0,0.0,0.0); // Set circle color to red  (R G B)
        glPointSize(2.0);
        Cylinder_draw(); // Call cylinder
        parallelepiped_draw();// call parallelepiped
        glFlush(); // Process all OpenGL routines as quickly as possible
}

void main(int argc, char **argv)
```

```
{
glutInit(&argc,argv); // Initialize GLUT
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); // Set Display mode
        glutInitWindowPosition(50,50);  // Set top left window position
        glutInitWindowSize(400,300); // Set Display window width and height
        glutCreateWindow("Cylinder and parallelePiped Display by Extruding Circle and
Quadrilaterl "); // Create Display Window
        init();
        glutDisplayFunc(display); // Send the graphics to Display Window
        glutMainLoop();
}
```

Program 7: program, using opengl functions, to draw a simple shaded scene consisting of tea pot on a table. Define suitablly the position on the properties of the light source along with the properties of the surfaces of the solid object in the scene.

```
#include <GL/glut.h>
void wall (double thickness)
{
        //draw thin wall with top = xz-plane, corner at origin
        glPushMatrix();
        glTranslated (0.5, 0.5 * thickness, 0.5);
        glScaled (1.0, thickness, 1.0);
        glutSolidCube (1.0);
        glPopMatrix();
}

//draw one table leg
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)
{
        //draw the table - a top and four legs
        //draw the top first
        glPushMatrix();
        glTranslated (0, legLen, 0);
        glScaled(topWid, topThick, topWid);
        glutSolidCube (1.0);
        glPopMatrix();
        double dist = 0.95 * topWid/2.0 - legThick/2.0;
        glPushMatrix();
        glTranslated (dist, 0, dist);
        tableLeg (legThick, legLen);
        glTranslated (0.0, 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 displaySolid (void)
{
	//set properties of the surface material
	GLfloat mat_ambient[] = {0.7f, 0.7f, 0.7f, 1.0f}; // gray
	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);

	//set the light source properties
	GLfloat lightIntensity[] = {0.7f, 0.7f, 0.7f, 1.0f};
	GLfloat light_position[] = {2.0f, 6.0f, 3.0f, 0.0f};
	glLightfv (GL_LIGHT0, GL_POSITION, light_position);
	glLightfv (GL_LIGHT0, GL_DIFFUSE, lightIntensity);

	//set the camera
	glMatrixMode (GL_PROJECTION);
	glLoadIdentity();
	double winHt = 1.0; //half-height of window
	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);

	//start drawing
	glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glPushMatrix();
	glTranslated (0.4, 0.4, 0.6);
	glRotated (45, 0, 0, 1);
	glScaled (0.08, 0.08, 0.08);

	glPopMatrix();

	glPushMatrix();
	glTranslated (0.6, 0.38, 0.5);
	glRotated (30, 0, 1, 0);
	glutSolidTeapot (0.08);
	glPopMatrix ();
	glPushMatrix();
	glTranslated (0.25, 0.42, 0.35);
	//glutSolidSphere (0.1, 15, 15);
	glPopMatrix();
```

```
	glPushMatrix();
	glTranslated (0.4, 0, 0.4);
	table (0.6, 0.02, 0.02, 0.3);
	glPopMatrix();
	wall (0.02);
	glPushMatrix();
	glRotated (90.0, 0.0, 0.0, 1.0);
	wall (0.02);
	glPopMatrix();
	glPushMatrix();
	glRotated (-90.0, 1.0, 0.0, 0.0);
	wall (0.02);
	glPopMatrix();

	glFlush();
}

void main (int argc, char ** argv)
{
	glutInit (&argc, argv);
	glutInitDisplayMode (GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
	glutInitWindowSize (640, 480);
	//glutInitWindowPosition (100, 100);
	glutCreateWindow ("simple shaded scene consisting of a tea pot on a table");
	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 8: program to draw a color cube and allow the 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 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}};

//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};

void polygon(int a,int b,int c,int d)
{
        glColor3f(1.0,0.0,0.0);
        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()
 {
         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);
        //glDrawElements(GL_QUADS,24,GL_UNSIGNED_BYTE,CubeIndices);
        //glBegin(GL_LINES);
        //glVertex3f(0.0,0.0,0.0);
        //glVertex3f(1.0,1.0,1.0);
        colorcube();
        //glEnd();
        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[2]+=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);
}

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();
}
```

Program 9: program to fill any given polygon using scan-line area filling algorithm.

```
#define BLACK 0
#include <stdlib.h>
#include <stdio.h>
#include <GL/glut.h>
float x1,x2,x3,x4,y1,y2,y3,y4;
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,int value)
{
        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++)
        {
                if(le[y]<=re[y])
                        for(i=(int)le[y];i<(int)re[y];i++)
                                draw_pixel(i,y,BLACK);

        }
}
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();
  scanfill(x1,y1,x2,y2,x3,y3,x4,y4);
  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("Filling a Polygon using Scan-line Algorithm");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```

Program 10: To diplay a set of values as a rectangular mesh.

```
#include<stdlib.h>
#include<GL/glut.h>
#define maxx 20
#define maxy 25
#define dx 15
#define dy 10
GLfloat x[maxx]={0.0},y[maxy]={0.0};
GLfloat x0=50,y0=50;
GLint i,j;

void init()
{
	glClearColor(1.0,1.0,1.0,1.0);
	//glColor3f(1.0,0.0,0.0);
	glPointSize(5.0);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(0.0,500.0,0.0,500.0);
	glutPostRedisplay();
}

void display(void)
{
	glClear(GL_COLOR_BUFFER_BIT);
	//glColor3f(0.0,0.0,1.0);
	for(i=0;i<maxx;i++)
		x[i]=x0+i*dx;
	for(j=0;j<maxy;j++)
		y[j]=y0+j*dy;
	glColor3f(0.0,0.0,1.0);
	for(i=0;i<maxx-1;i++)
		for(j=0;j<maxy-1;j++)
		{
			glColor3f(0.0,0.0,1.0);
			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();
```

```
}

void 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();
}
```