**LAB MANUAL**

**COMPUTER GRAPHICS**

**//1.Program to recursively subdivide a tetrahedron to from 3D Sierpinski gasket. The number of recursive steps is to be specified by the user.**

//Program1-Sierpinski gasket

#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);

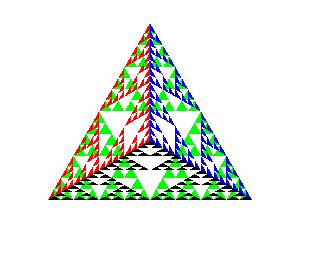
glutMainLoop();

}

**Output:**

**Enter the number the number of recursive steps you want**

**4**

****

**2. Program to implement Liang-Barsky line clipping algorithm.**

//Program2-Liang Barskey Line Clipping Algorithm

#include<GL/glut.h>

#define true 1

#define false 0

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

GLdouble xvmin=250,yvmin=250,xvmax=300,yvmax=300;

int cliptest(GLdouble p,GLdouble q,GLdouble \*te,GLdouble\*tl)

{

GLdouble t=q/p;

if(p<0.0)

{

if(t>\*te) \*te=t;

if(t>\*tl)

return(false);

}

if(p>0.0)

{

if(t<\*tl) \*tl=t;

if(t<\*te) return(false);

}

if(p==0.0)

{

if(q<0.0)return(false);

}

return(true);

}

void LBLineClipDraw(GLdouble x0,GLdouble y0,GLdouble

x1,GLdouble y1)

{

GLdouble dx=x1-x0,dy=y1-y0,te=0.0,tl=1.0;

if(cliptest(-dx,x0-xmin,&te,&tl)!=false)

{

if(cliptest(dx,xmax-x0,&te,&tl)!=false)

{

if(cliptest(-dy,y0-ymin,&te,&tl)!=false)

{

if(cliptest(dy,ymax-y0,&te,&tl)!=false)

{

if(tl<1.0)

{

x1=x0+tl\*dx;

y1=y0+tl\*dy;

}

if(te>0.0)

{

x0=x0+te\*dx;

y0=y0+te\*dy;

}

}

}

}

}

glColor3f(1.0,0.0,0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(xvmin,yvmin);

glVertex2f(xvmax,yvmin);

glVertex2f(xvmax,yvmax);

glVertex2f(xvmin,yvmax);

glEnd();

GLdouble vx0=x0+200;

GLdouble vy0=y0+200;

GLdouble vx1=x1+200;

GLdouble vy1=y1+200;

glColor3f(0.0,0.0,1.0);

glBegin(GL\_LINES);

glVertex2d(vx0,vy0);

glVertex2d(vx1,vy1);

glEnd();

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT); //TO CLEAR THE LAST OUTPUT AND AVOID TRANSPARENCY

glClearColor(1.0,1.0,1.0,1.0); //TO SET BACKGROUND COLOR

GLdouble x0=60,y0=20,x1=80,y1=120;

glColor3f(0.0,0.0,1.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(xmin,ymin);

glVertex2f(xmax,ymin);

glVertex2f(xmax,ymax);

glVertex2f(xmin,ymax);

glEnd();

glColor3f(1.0,0.0,0.0);

glBegin(GL\_LINES);

glVertex2d(x0,y0);

glVertex2d(x1,y1);

glEnd();

LBLineClipDraw(x0,y0,x1,y1);

glFlush();

}

void init()

{

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0.0,500.0,0.0,500.0);

}

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

{

glutInit(&argc,argv);

glutInitWindowSize(500,500);

glutInitWindowPosition(0,0);

glutCreateWindow("Liang Barsky Line Clipping Alg");

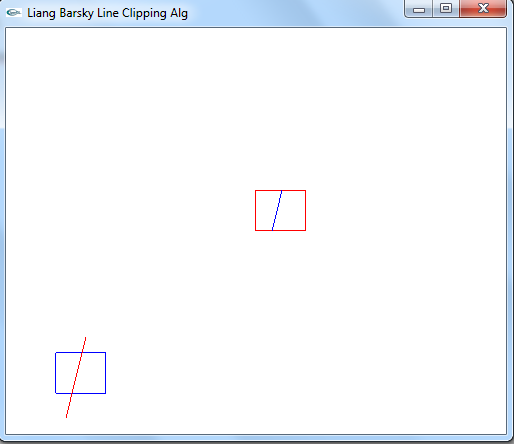
init();

glutDisplayFunc(display);

glutMainLoop();

}

**Output:**

****

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

//Program 3-spin cube

#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;

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);

}

void 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);

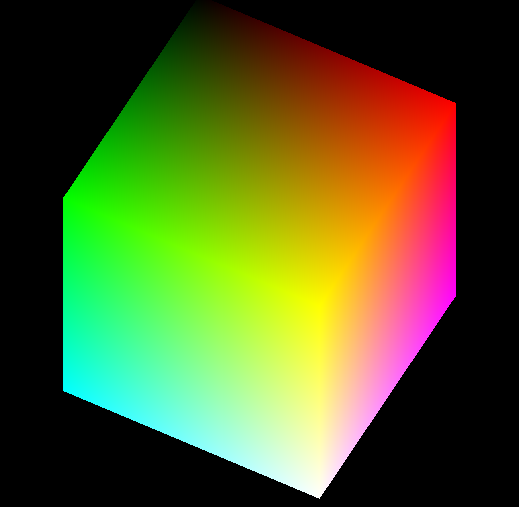
glEnableClientState(GL\_COLOR\_ARRAY);

glColorPointer(3,GL\_FLOAT,0,colors);

glutMainLoop();

}

**Output:**



**4. Program to create a house like figure and rotate it about a given fixed point using OpenGL functions.**

//Program4-Rotating House

#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];

GLfloat result[3][9];

GLfloat h=100.0;

GLfloat k=100.0;

GLfloat theta;

void multiply()

{

int i,j,k;

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

{

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

{

result[i][j]=0;

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

{

result[i][j]=result[i][j]+rot\_mat[i][k]\*house[k][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);//to avoid previous output

glClearColor(1.0,1.0,1.0,1.0);//to set background color

drawhouse();

rotate();

drawrotatedhouse();

glFlush();

}

void init()

{

//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("\nEnter the rotation angle: ");

scanf("%f",&theta);

theta=theta\*(3.14/180.0);

glutInit(&argc,argv);

glutInitWindowSize(600,600);

glutInitWindowPosition(0,0);

glutCreateWindow("House rotation");

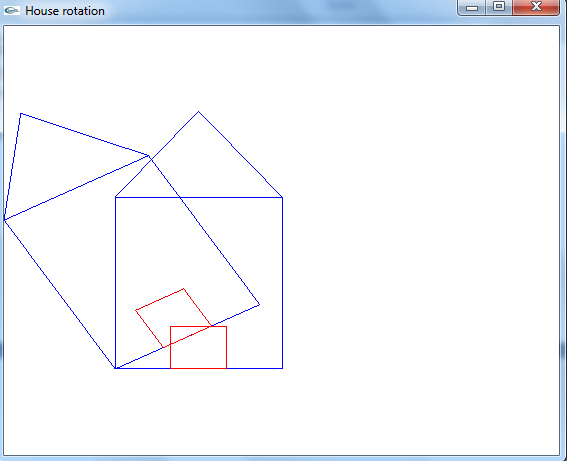
init();

glutDisplayFunc(display);

glutMainLoop();

}

**Output:**

****

**5. 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**

//Program 5-Cohen sutheland line clipping

#include<GL/glut.h>

//#define outcode int

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

GLdouble xvmin=250,yvmin=250,xvmax=300,yvmax=300;

const int RIGHT=8;

const int TOP=4;

const int LEFT=2;

const int BOTTOM=1;

int computeoutcode(GLdouble x, GLdouble y)

{

int code=0;

if(y>ymax)

{

code=code|TOP;

}if(y<ymin)

{

code=code|BOTTOM;

}if(x>xmax)

{

code=code|RIGHT;

}if(x<xmin)

{

code=code|LEFT;

}return code;

}

void CSlineclipdraw(GLdouble x0,GLdouble y0,GLdouble x1,GLdouble y1)

{

int outcode0,outcode1,outcodeout;

bool accept=false;

bool done=false;

outcode0=computeoutcode(x0,y0);

outcode1=computeoutcode(x1,y1);

do

{

if((outcode0|outcode1)==0000)

{

accept=true;

done=true;

}else if((outcode0 & outcode1)!=0000)

{

accept=false;

done=true;

}

else

{

GLdouble x,y;

if(outcode0!=0000)

{

outcodeout=outcode0;

}else

{

outcodeout=outcode1;

}if(outcodeout & TOP)

{

x=x0+(x1-x0)\*(ymax-y0)/(y1-y0);

y=ymax;

}if(outcodeout & BOTTOM)

{

x=x0+(x1-x0)\*(ymin-y0)/(y1-y0);

y=ymin;

}if(outcodeout & RIGHT)

{

x=xmax;

y=y0+(y1-y0)\*(xmax-x0)/(x1-x0);

}if(outcodeout & LEFT)

{

x=xmin;

y=y0+(y1-y0)\*(xmin-x0)/(x1-x0);

}if(outcodeout==outcode0)

{

x0=x;

y0=y;

outcode0=computeoutcode(x0,y0);

}if(outcodeout==outcode1)

{

x1=x;

y1=y;

outcode1=computeoutcode(x1,y1);

}

}

}while(!done);

if(accept=true)

{

glColor3f(1.0,0.0,0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(xvmin,yvmin);

glVertex2f(xvmax,yvmin);

glVertex2f(xvmax,yvmax);

glVertex2f(xvmin,yvmax);

glEnd();

GLdouble vx0=x0+(xvmin-xmin);

GLdouble vy0=y0+(yvmin-ymin);

GLdouble vx1=x1+(xvmin-xmin);

GLdouble vy1=y1+(yvmin-ymin);

glColor3f(0.0,0.0,1.0);

glBegin(GL\_LINES);

glVertex2d(vx0,vy0);

glVertex2d(vx1,vy1);

glEnd();

}

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT); //TO CLEAR THE LAST OUTPUT AND

AVOID TRANSPARENCY

glClearColor(1.0,1.0,1.0,1.0); //TO SET BACKGROUND COLOR

GLdouble x0=60,y0=20,x1=80,y1=120;

glColor3f(0.0,0.0,1.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(xmin,ymin);

glVertex2f(xmax,ymin);

glVertex2f(xmax,ymax);

glVertex2f(xmin,ymax);

glEnd();

glColor3f(1.0,0.0,0.0);

glBegin(GL\_LINES);

glVertex2d(x0,y0);

glVertex2d(x1,y1);

glEnd();

CSlineclipdraw(x0,y0,x1,y1);

glFlush();

}

void init()

{

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0.0,500.0,0.0,500.0);

}

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

{

glutInit(&argc,argv);

glutInitWindowSize(500,500);

glutInitWindowPosition(0,0);

glutCreateWindow("CohenSutherland Line Clipping Algorithm");

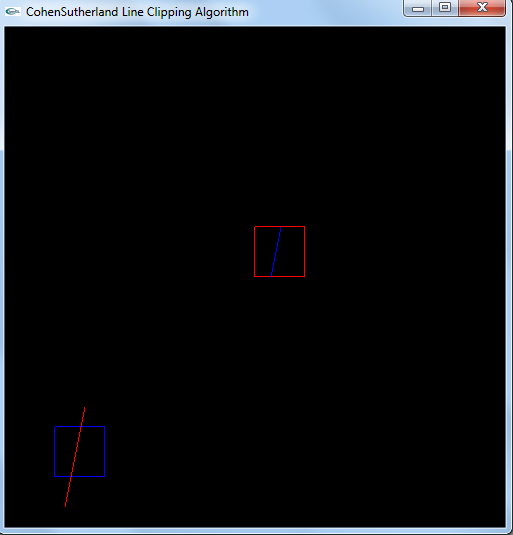
init();

glutDisplayFunc(display);

glutMainLoop();

}

**Output:**



**6.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.**

//Program 6-cylinde and parallelpiped

#include<GL/glut.h>

void draw\_pixel(GLint m,GLint n)

{

glColor3f(1.0,0.0,0.0);

glBegin(GL\_POINTS);

glVertex2i(m,n);

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)

{

GLint d=1-r,x=0,y=r;

plotpixels(h,k,x,y);

while(y>x)

{

if(d<0)

{

d+=2\*x+3;

}

else

{

d+=2\*(x-y)+5;

--y;

}

++x;

plotpixels(h,k,x,y);

}

}

void cylinder\_draw()

{

GLint h=100,k=100,r=50;

GLint i,n=50;

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

{

circle\_draw(h,k+i,r);

}

}

void rectangle\_draw(GLint x1,GLint x2,GLint y1,GLint y2)

{

glColor3f(0.0,0.0,1.0);

glBegin(GL\_LINE\_LOOP);

glVertex2i(x1,y1);

glVertex2i(x2,y1);

glVertex2i(x2,y2);

glVertex2i(x1,y2);

glEnd();

}

void parallelopiped\_draw()

{

GLint x1=200,x2=300,y1=100,y2=175;

GLint i,n=40;

for(i=0;i<n;i+=2)

{

rectangle\_draw(x1+i,x2+i,y1+i,y2+i);

}

}

void init()

{

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0.0,400.0,0.0,300.0);

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT); //TO CLEAR THE LAST OUTPUT AND AVOID TRANSPARENCY

glClearColor(1.0,1.0,1.0,1.0); //TO SET BACKGROUND COLOR

cylinder\_draw();

parallelopiped\_draw();

glFlush();

}

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

{

glutInit(&argc,argv);

glutInitWindowPosition(50,50);

glutInitWindowSize(500,400);

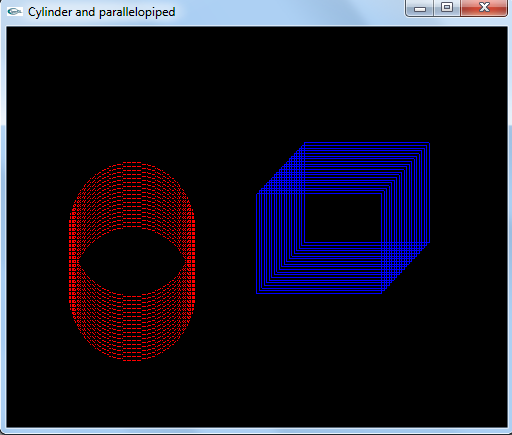
glutCreateWindow("Cylinder and parallelopiped");

init();

glutDisplayFunc(display);

glutMainLoop();

}

**Output:**

**7. 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.**

//Program7-tea pot

#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);

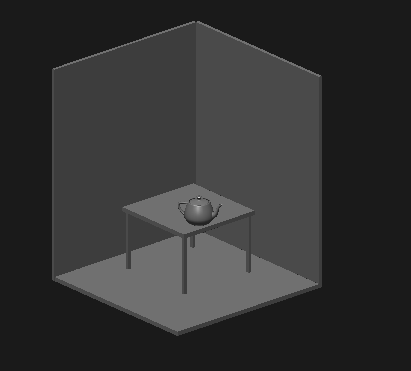
glEnable(GL\_NORMALIZE);

glClearColor(0.1,0.1,0.1,0.0);

glViewport(0,0,640,480);

glutMainLoop();

}

**Output:**

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

//Program8-color cube

#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);

glOrtho(-2.0,2.0,-2.0,2.0,-10.0,10.0);

glMatrixMode(GL\_MODELVIEW);

}

void 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);

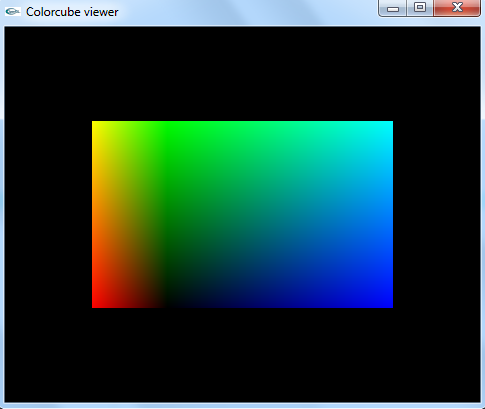
glEnableClientState(GL\_COLOR\_ARRAY);

glColorPointer(3,GL\_FLOAT,0,colors);

glutMainLoop();

}

**Output:**

****

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

//Program 9-scanfill

#include<GL/glut.h>

GLint x1=200,x2=100,x3=200,x4=300,y1=200,y2=300,y3=400,y4=300;

void edgedetect(GLint x1,GLint y1,GLint x2,GLint y2,GLint \*le,GLint \*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<le[i])

{

le[i]=x;

}

if(x>re[i])

{

re[i]=x;

}x

+=mx;

}

}

void draw\_pixel(int m,int n)

{

glColor3f(1.0,0.0,0.0);

glBegin(GL\_POINTS);

glVertex2i(m,n);

glEnd();

}

void scanfill()

{

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=le[y];i<=re[y];i++)

{

draw\_pixel(i,y);

}

}

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT); //TO AVOID TRANSPARENCY AND REMOVE LAST OUTPUT

glClearColor(1.0,1.0,1.0,0.0); //TO SET BACKGROUND COLOR

glBegin(GL\_LINE\_LOOP);

glVertex2f(x1,y1);

glVertex2f(x2,y2);

glVertex2f(x3,y3);

glVertex2f(x4,y4);

glEnd();

scanfill();

glFlush();

}

void init()

{

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0.0,400.0,0.0,500.0); //these 4 parameters will give us a RHOMBUS, whereas(0.0,400.0,0.0,300.0) will give a triangle

}

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

{

glutInit(&argc,argv);

glutInitWindowPosition(50,50);

glutInitWindowSize(800,600);

glutCreateWindow("scan line algorithm");

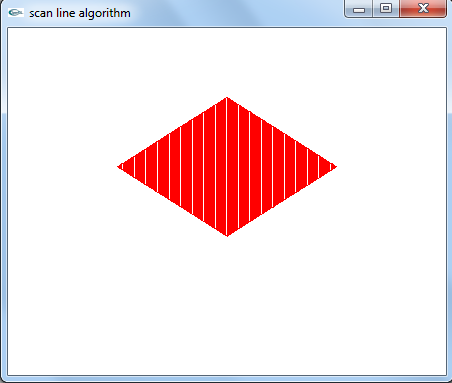
glutDisplayFunc(display);

init();

glutMainLoop();

}

**Output:**

****

**10. Program to display a set of values {fij} as a rectangular mesh.**

//Program10-Mesh

#include<GL/glut.h>

#define maxx 20

#define maxy 25

#define dx 15

#define dy 10

GLint x[maxx],y[maxy];

GLint x0=50,y0=50;

GLint i,j;

void init()

{

glClearColor(1.0,1.0,1.0,1.0); //TO SET BACKGROUND COLOR

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0.0,500.0,0.0,400.0);

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT); //TO AVOID TRANSPARENCY

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

{

x[i]=x0+i\*dx;

}for(j=0;j<maxy;j++)

{

y[j]=y0+j\*dy;

}

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);

glVertex2i(x[i],y[j]);

glVertex2i(x[i],y[j+1]);

glVertex2i(x[i+1],y[j+1]);

glVertex2i(x[i+1],y[j]);

glEnd();

glFlush();

}

}

}

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

{

glutInit(&argc,argv);

glutInitWindowPosition(0,0);

glutInitWindowSize(500,400);

glutCreateWindow("rectangular Mesh");

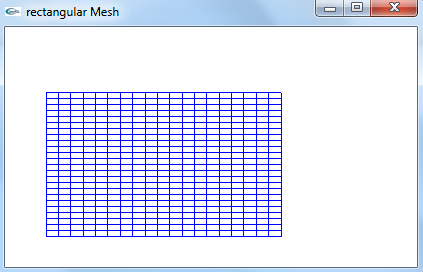
init();

glutDisplayFunc(display);

glutMainLoop();

}

**Output:**

****