



fsmk.org/labmanual

VI
SEMESTER
FOR CSE & ISE
COMPUTER GRAPHICS AND
VISUALIZATION LABORATORY
[10CSL67]

LAB MANUAL

VTU SYLLABUS 2010

FREE SOFTWARE MOVEMENT
KARNATAKA





Content

- 1. Introduction**
- 2. Contributors**
- 3. Foreword**
- 4. Introduction to OpenGL**
- 5. Syllabus**
- 6. Problems and Solutions**



About Free Software Movement Karnataka (FSMK)



Free Software Movement Karnataka (FSMK) is a nonprofit organization formed in March 2009 to spread the ideals of free software. We try to increase the understanding of the philosophy behind free software and encourage its use in educational institutions, our very own community center and other organisations.

The phrase "free and open source software" can be used to collectively describe a set of operating systems and standalone applications that are free from the clutches of large corporate organisations which produce software only for commercial purposes. Free software is often freely available and is created by a thriving community of programmers, designers and writers. The source code (i.e. the computer program) that underlies an application or an operating system is also open to the perusal of the common individual, which allows every curious tinkerer, student or otherwise, to play around with the source code. Most free software organisations welcome the general public to be part of their community and to contribute in any of the three spheres mentioned above. In the event of you not being a programmer, designer or writer, you can also become a member of the community by actively using free software applications, thereby reporting any issues that you notice, and also by spreading awareness about free software among your friends and family. At FSMK, we believe that it is unfortunate that schools, colleges and other small organisations that can use Linux and related free software for no cost, instead invest in using proprietary and commercial software, thereby spending large amounts annually on licensing and other fees, which can instead be used for the betterment of education or related services, and thereby, the betterment of society.

I want to be involved Website: <http://fsmk.org/>

Mailing list: <http://www.fsmk.org/?q=mailinglistssubscribe>

About Spoken Tutorial Project:



The Spoken Tutorial project is an initiative of National Mission on Education through ICT, Government of India, to promote IT literacy through Free and Open Source Software. The project is being developed and coordinated by IIT-Bombay and led by Dr. Kannan M. Moudgalya. The project aims at building a repository of self learning courses through video tutorials of various open source softwares. These courses are then used to organize 2 hour workshops in government organizations, NGOs, SMEs and School and Colleges in India completely free of cost for the participants. These tutorials are not only available in English but also in various regional languages for the learner to be able to learn in the language he/she is comfortable in. Currently, Spoken Tutorials are available for free software tools like Blender, GIMP, Latex, Scilab, LibreOffice Suite, Ubuntu Linux, Mozilla Firefox, Thunderbird, MySQL and also programming languages and scripts like C, C++, Python, Ruby, Perl, PHP, Java. All the spoken tutorials which are released under Creative Commons License are available for download free of cost at their website <http://spoken-tutorial.org>.

About Jnana Vikas Institute of Technology, Bidadi



Jnana Vikas Institute of Technology was established in the year 2001 by JNANA VIKAS VIDYA SANGHA with a mission to not just provide a solid educational foundation to students but to build their careers, to make them eminent personalities in the society and to make the industry doors open to them. It is approved by AICTE, New Delhi and affiliated to VTU, Belgaum. It has a residential campus with nearly 73 faculties, 28 technical and non-technical supporting staff, 27 administrative and supporting staff and 590 students and is a self-contained campus located in a beautiful green land of about 25 acres. The institute has four academic departments in various disciplines of engineering and three departments in general science with nearly 19 laboratories all together, organized in a unique pattern. There is a separate department for management discipline. The campus is located at Bidadi, in southern part of city of Bengaluru. More information about the college is provided at their website, <http://www.jvitedu.in/>



Contributors

Following is the list of all the volunteers who contributed to the making of the Lab Manual.

- Ananda Kumar H N, Faculty, A.T.M.E College of Engineering, Mysore
- Arun Chavan L, Faculty, The Oxford College of Engineering, Bangalore
- Bhavya D, Faculty, P.E.S. College of Engineering, Mandya
- Bindu Madavi K P, Faculty, The Oxford College of Engineering, Bangalore
- Byregowda B K, Faculty, Sir M. Visvesvaraya Institute of Technology, Bangalore
- Deepika, Faculty, P.E.S. College of Engineering, Mandya
- Kiran B, Faculty, A.T.M.E College of Engineering, Mysore
- Manjunatha H C, Faculty, Sir M. Visvesvaraya Institute of Technology, Bangalore
- Neeta Ann Jacob, Faculty, The Oxford College of Engineering, Bangalore
- Rajesh N, Faculty, Sir M. Visvesvaraya Institute of Technology, Bangalore
- Shashidhar S, Faculty, A.T.M.E College of Engineering, Mysore
- Shwetha M K, Faculty, P.E.S. College of Engineering, Mandya
- Abdul Nooran, Student, Vivekananda College of Engineering and Technology, Puttur
- Abhiram R., Student, P.E.S Institute of Technology, Bangalore South Campus
- Ajith K.S., Student, The Oxford College of Engineering, Bangalore
- Akshay Gudi, Student, P.E.S. College of Engineering, Mandya
- Arjun M.Y., Student, P.E.S. College of Engineering, Mandya
- Chaitra Kulkarani, Student, Government Engineering College, Hassan
- John Paul S., Student, Jnana Vikas Institute of Technology, Bidadi
- Karan Jain, Student, P.E.S Institute of Technology, Bangalore South Campus
- Kuna Sharathchandra, Student, P.E.S. College of Engineering, Mandya
- Manas J.K., Student, Dr. Ambedkar Institute of Technology, Bangalore
- Manu H., Student, P.E.S. College of Engineering, Mandya
- Meghana S., Student, Government Engineering College, Hassan
- Nagaraj, Student, Vivekananda College of Engineering and Technology, Puttur
- Nandan Hegde, Student, Vivekananda College of Engineering and Technology, Puttur
- Narmada B., Student, P.E.S Institute of Technology, Bangalore South Campus
- Nawaf Abdul, Student, P.A. College of Engineering, Mangalore
- Nitesh A. Jain, Student, B.M.S. Institute of Technology, Bangalore
- Nitin R., Student, The Oxford College of Engineering, Bangalore
- Padmavathi K., Student, B.M.S. Institute of Technology, Bangalore
- Poojitha Koneti, Student, B.M.S. Institute of Technology, Bangalore
- Rahul Kondi, Student, S.J.B. Institute of Technology, Bangalore
- Rohit G.S., Student, Dr. Ambedkar Institute of Technology, Bangalore
- Samruda, Student, Student, Government Engineering College, Hassan
- Samyama H.M., Student, Government Engineering College, Hassan
- Santosh Kumar, Student, Dr. Ambedkar Institute of Technology, Bangalore
- Shashank M C., Student, S.J.B. Institute of Technology, Bangalore
- Soheb Mohammed, Student, P.E.S Institute of Technology, Bangalore South Campus
- Indra Priyadarshini, Student, P.E.S Institute of Technology, Bangalore South Campus
- Suhas, Student, P.A. College of Engineering, Mangalore
- Vamsikrishna G., Student, P.E.S Institute of Technology, Bangalore South Campus
- Vikram S., Student, P.E.S Institute of Technology, Bangalore South Campus
- Vivek Basavraj, Student, Jnana Vikas Institute of Technology, Bidadi
- Aruna S, Core member of FSMK
- HariPrasad, Core member of FSMK
- Isham, Core member of FSMK
- Jayakumar , Core member of FSMK
- Jeeva J, Core member of FSMK
- Jickson, Core member of FSMK

- Karthik Bhat, Core member of FSMK
- Prabodh C P, Core member of FSMK
- RaghuRam N, Core member of FSMK
- Rameez Thonnakkal, Core member of FSMK
- Sarath M S, Core member of FSMK
- Shijil TV, Core member of FSMK
- Vignesh Prabhu, Core member of FSMK
- Vijay Kulkarni, Core member of FSMK
- Yajnesh, Core member of FSMK



Foreword

"Free Software is the future, future is ours" is the motto with which Free Software Movement Karnataka has been spreading Free Software to all parts of society, mainly amongst engineering college faculty and students. However our efforts were limited due to number of volunteers who could visit different colleges physically and explain what is Free Software and why colleges and students should use Free Software. Hence we were looking for ways to reach out to colleges and students in a much larger way. In the year 2013, we conducted two major Free Software camps which were attended by close to 160 students from 25 different colleges. But with more than 150 engineering colleges in Karnataka, we still wanted to find more avenues to reach out to students and take the idea of free software in a much larger way to them.

Lab Manual running on Free Software idea was initially suggested by Dr. Ganesh Aithal, Head of Department, CSE, P.A. Engineering College and Dr. Swarnajyothi L., Principal, Jnana Vikas Institute of Technology during our various interactions with them individually. FSMK took their suggestions and decided to create a lab manual which will help colleges to migrate their labs to Free Software, help faculty members to get access to good documentation on how to conduct various labs in Free Software and also help students by providing good and clear explanations of various lab programs specified by the university. We were very clear on the idea that this lab manual should be produced also from the students and faculty members of the colleges as they knew the right way to explain the problems to a large audience with varying level knowledge in the subject. FSMK promotes freedom of knowledge in all respects and hence we were also very clear that the development and release of this lab manual should under Creative Commons License so that colleges can adopt the manual and share, print, distribute it to their students and thereby helping us in spreading free software.

Based on this ideology, we decided to conduct a documentation workshop for college faculty members where they all could come together and help us produce this lab manual. As this was a first attempt for even FSMK, we decided to conduct a mock documentation workshop for one day at Indian Institute of Science, Bangalore on 12 Jan, 2014. Close to 40 participants attended it, mainly our students from various colleges and we tried documenting various labs specified by VTU. Based on this experience, we conducted a 3 day residential documentation workshop jointly organized with Jnana Vikas Institute of Technology, Bidadi at their campus from 23 January, 2014. It was attended by 16 faculty members of different colleges and 40 volunteers from FSMK. The documentation workshop was sponsored by Spoken Tutorial Project, an initiative by Government of India to promote IT literacy through Open Source software. Spoken Tutorials are very good learning material to learn about various Free Software tools and hence the videos are excellent companion to this Lab Manual. The videos themselves are released under Creative Commons license, so students can easily download them and share it with others. We would highly recommend our students to go through the Spoken Tutorials while using this Lab Manual and web links to the respective spoken tutorials are shared within the lab manual also.

Finally, we are glad that efforts and support by close to 60 people for around 3 months has led to creation of this Lab Manual. However like any Free Software project, the lab manual will go through constant improvement and we would like the faculty members and students to send us regular feedback on how we can improve the quality of the lab manual. We are also interested to extend the lab manual project to cover MCA departments and ECE departments and are looking for volunteers who can put the effort in this direction. Please contact us if you are interested to support us.



OpenGL User Guide

1. How to install OpenGL?

◦ Using terminal:

To install OpenGL, run the following command from a terminal prompt:

\$sudo apt-get install freeglut3 freeglut3-dev

◦ Using Ubuntu software Center:

Using Ubuntu Software Center, install following packages:

- freeglut3
- freeglut3-dev

2. Compilation and Execution

◦ For a C program

Compilation:

`$gcc -lglut -lGL -lGLU filename.c -o output.x`

Execution:

`$/output.x`

◦ For a C++ program

Compilation:

`$g++ -lglut -lGL -lGLU filename.cpp -o output.x`

Execution:

`$/output.x`



Lab Programs list for Computer Graphics and Visualization Lab as specified by VTU for 6th Semester students:

1. Program to recursively subdivide a tetrahedron to form 3D Sierpinski gasket. The number of recursive steps is to be specified by the user.
2. Program to implement Liang-Barsky line clipping algorithm.
3. Program to draw a color cube and spin it using OpenGL transformation matrices.
4. Program to create a house like figure and rotate it about a given fixed point using OpenGL functions.
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.
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.
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 surfaces of the solid object used in the scene.
8. Program to draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing. Use OpenGL functions.
9. Program to fill any given polygon using scan-line area filling algorithm. (Use appropriate data structures.)
0. Program to display a set of values $\{f_{ij}\}$ as a rectangular mesh.

Project: 11. Develop a suitable Graphics package to implement the skills learnt in the theory and the exercises indicated in Part A. Use the OpenGL.



Aim:

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

Theory

A geometric method of creating the gasket is to start with a triangle and cut out the middle piece as shown in the generator below. This results in three smaller triangles to which the process is continued. The nine resulting smaller triangles are cut in the same way, and so on, indefinitely. The gasket is perfectly self similar, an attribute of many fractal images. Any triangular portion is an exact replica of the whole gasket. The construction of the 3 dimensional version of the gasket follows similar rules for the 2D case except that the building blocks are square based pyramids instead of triangles.

Algorithm:

1. Start with a single triangle.
2. Inside this triangle, draw a smaller upside down triangle. Its corners should be exactly in the centers of the sides of the large triangle
3. Now, draw 3 smaller triangles in each of the 3 triangles that are pointing upwards, again with the corners in the centers of the sides of the triangles that point upwards
4. Now there are 9 triangles pointing upwards. In each of these 9, draw again smaller upside down triangles.
5. In the 27 triangles pointing upwards, again draw 27 triangles pointing downwards.
6. repeat

Code: sierpanski.c

```
#include<stdio.h>
#include<GL/glut.h>
typedef float point[3];
point v[] = {
    {0.0,0.0,1.0},{0.0,0.942809,-0.333333},
    {-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;

//repeat 'm' no of times as specified by user
if(m>0)
{
    for(j=0;j<3;j++)
        // get midpoint of first edge
        v1[j]=(a[j]+b[j])/2;
    for(j=0;j<3;j++)
        // get midpoint of second edge
        v2[j]=(a[j]+c[j])/2;
    for(j=0;j<3;j++)
        // get midpoint of third edge
        v3[j]=(b[j]+c[j])/2;

    // consider midpoints as vertex and divide bigger triangle
    // to 3 parts recursively
    divide_triangle(a,v1,v2,m-1);
    divide_triangle(c,v2,v3,m-1);
    divide_triangle(b,v3,v1,m-1);
}
//draw the sub divided traingles
else
{
    (triangle(a,b,c));
}
}

// tetrahedron has 4 faces. each face is traingle. we send each
// face of tetrahedron and divide each faces/triangles into 3
// triangles recursively 'm' times
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);
}

//this is called everytime the display is refreshed. Here it draw a tetrahedron.
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    tetrahedron(n);
    glFlush();
}

// This function is executed when the wiindow size is changed.
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);    //to
get exact aspect ratio

```

```

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("no. of divisions \n");
    scanf("%d", &n);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("3DGasket");    //window with a title
    glutReshapeFunc(myReshape);
    glutDisplayFunc(display);
    glEnable(GL_DEPTH_TEST);
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glutMainLoop();
}

```

Output:

Commands for execution:-

- Open a terminal and Change directory to the file location in both the terminals.
- compile as `gcc -lGLU -lGL -lglut sierpanski.c -o sierpanski`
- If no errors, run as `./sierpanski`




Aim:

Program to implement Liang-Barsky line clipping algorithm.

Theory

In computer graphics, 'line clipping' is the process of removing lines or portions of lines outside of an area of interest. Typically, any line or part thereof which is outside of the viewing area is removed. The Liang-Barsky algorithm uses the parametric equation of a line and inequalities describing the range of the clipping window to determine the intersections between the line and the clipping window. With these intersections it knows which portion of the line should be drawn.

Algorithm

1. Set $t_{\min} = 0$ and $t_{\max} = 1$
2. Calculate the values of t_L , t_R , t_T , and t_B ([tvalues](#)).
 - if $t < t_{\min}$ or $t > t_{\max}$ ignore it and go to the next edge
 - otherwise classify the t value as [entering or exiting](#) value (using inner product)
 - if t is entering value set $t_{\min} = t$; if t is exiting value set $t_{\max} = t$
3. If  then **draw a line** from $(x_1 + dx \cdot t_{\min}, y_1 + dy \cdot t_{\min})$ to $(x_1 + dx \cdot t_{\max}, y_1 + dy \cdot t_{\max})$
4. If the line crosses over the window, you will see $(x_1 + dx \cdot t_{\min}, y_1 + dy \cdot t_{\min})$ and $(x_1 + dx \cdot t_{\max}, y_1 + dy \cdot t_{\max})$ [intersection](#) between line and edge.

Code: liangBasky.c

```
#include<stdio.h>
#include<GL/glut.h>
double xmin=50,xmax=100,ymax=100,ymin=50;    //window boundary
double xmin=200,ymin=200,xvmax=300,yvmax=300;    //viewport boundary
#define true 1
#define false 0
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;    //line portion is outside
    }
    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;    //line portion is outside
return true;
}

void liang(double x0,double y0,double x1,double y1)
{
    double dx=x1-x0,dy=y1-y0,tc=0.0,t1=1.0;
    if(cliptest(-dx,x0-xmin,&tc,&t1))    //inside test wrt left edge
        if(cliptest(dx,xmax-x0,&tc,&t1))    //inside test wrt Right edge
            if(cliptest(-dy,y0-ymin,&tc,&t1))    //inside test wrt Bottom edge
                if(cliptest(dy,ymax-y0,&tc,&t1))    //inside test wrt Top edge
                {
                    if(t1<1.0)
                    {
                        x1=x0+t1*dx;
                        y1=y0+t1*dy;
                    }
                    if(tc>0.0)
                    {
                        x0=x0+tc*dx;
                        y0=y0+tc*dy;
                    }
                    //sx and sy is used to scale the line. it zooms the the clipping window
                    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;
                    //draw red coloured viewport
                    glColor3f(1.0,0.0,0.0);
                    glBegin(GL_LINE_LOOP);    //draw a box to show clipped line.
                        glVertex2f(xvmin,yvmin);
                        glVertex2f(xvmax,yvmin);
                        glVertex2f(xvmax,yvmax);
                        glVertex2f(xvmin,yvmax);
                    glEnd();
                    glColor3f(0.0,0.0,1.0);
                    glBegin(GL_LINES);    //draw line after clipping
                        glVertex2d(vx0,vy0);
                        glVertex2d(vx1,vy1);
                    glEnd();
                }
}

void display()
{
    double x0=60,y0=20,x1=80,y1=120;    //line coordinates
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0,0.0,0.0);
    glBegin(GL_LINES);    //draw a line that needs to be clipped
        glVertex2d(x0,y0);
        glVertex2d(x1,y1);
    glEnd();
}

```

```

    glColor3f(0.0,0.0,1.0);
    glBegin(GL_LINE_LOOP);          //draw a box
        glVertex2f(xmin,ymin);
        glVertex2f(xmax,ymin);
        glVertex2f(xmax,ymax);
        glVertex2f(xmin,ymax);
    glEnd();
    liang(x0,y0,x1,y1);             //function liang() is called by passing line endpoints
    glFlush();
}
//initialises point sizes and colors
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);
}

int main(int argc,char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Liang Barksy Line Clipping algorithm");
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
    return 0;
}

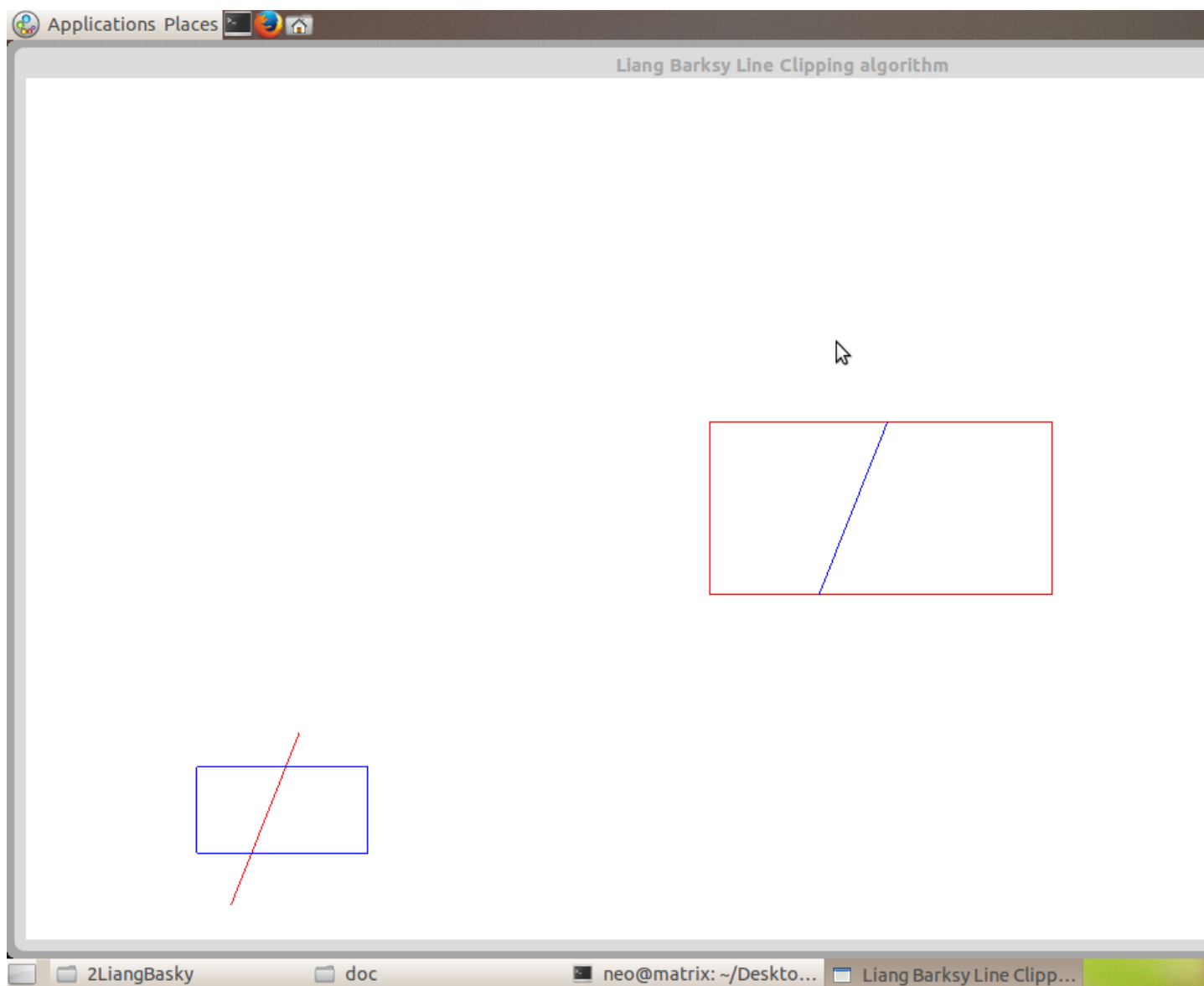
```

Output:

Commands for execution:-

- Open a terminal and Change directory to the file location in both the terminals.
- compile as `gcc -lGLU -lGL -lglut liangBasky.c -o liang`
- If no errors, run as `./liang`

Screenshots:-





Aim:

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

Algorithm

1. Choose eight 3 dimensional coordinate points such that will make a cube
2. Select the axis to rotate.
3. Rotate the cube by a small angle every small interval of time.
4. Repeat 3rd step as long as axis is not changed.

Code: spinCube.c

```
#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,-1.0,-1.0}, {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]+=2.0;
    if(theta[axis]>360.0)
        theta[axis]-=360;
    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;
    spincube();
}

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("Color Cube and Spin it! ");
    glutReshapeFunc(myReshape);
    glutDisplayFunc(display);
    glutIdleFunc(spincube);
    glutMouseFunc(mouse);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
}

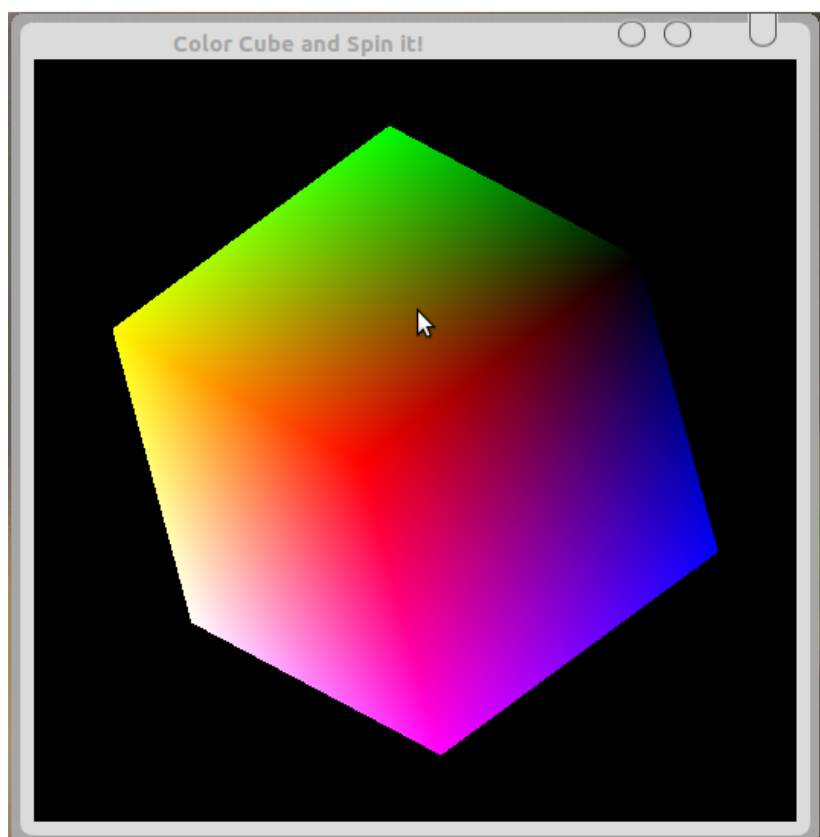
```

Output:

Commands for execution:-

- Open a terminal and Change directory to the file location in both the terminals.
- compile as `gcc -lGLU -lGL -lglut spinCube.c -o spincube`
- If no errors, run as `./spincube`

Screenshots:-





Aim

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

Algorithm

1. Draw a house by choosing appropriate coordinate points.
2. Calculate the rotation matrix, which is calculate w.r.t rotation angle.
3. Multiply rotation matrix with coordinate points of house.
4. This gives us coordiante points of rotated house

Code: rotateHouse.c

```
#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();
}
```

```
//Draw Initial house 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(1.0,1.0,0.0);
    glPointSize(10.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0,499.0,0.0,499.0);
}
```

```
int main(int argc,char **argv)
{
    printf("Enter the rotation angle\n");
    scanf("%f",&theta);
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("house rotation");
    glutDisplayFunc(display);
    myinit();
}
```

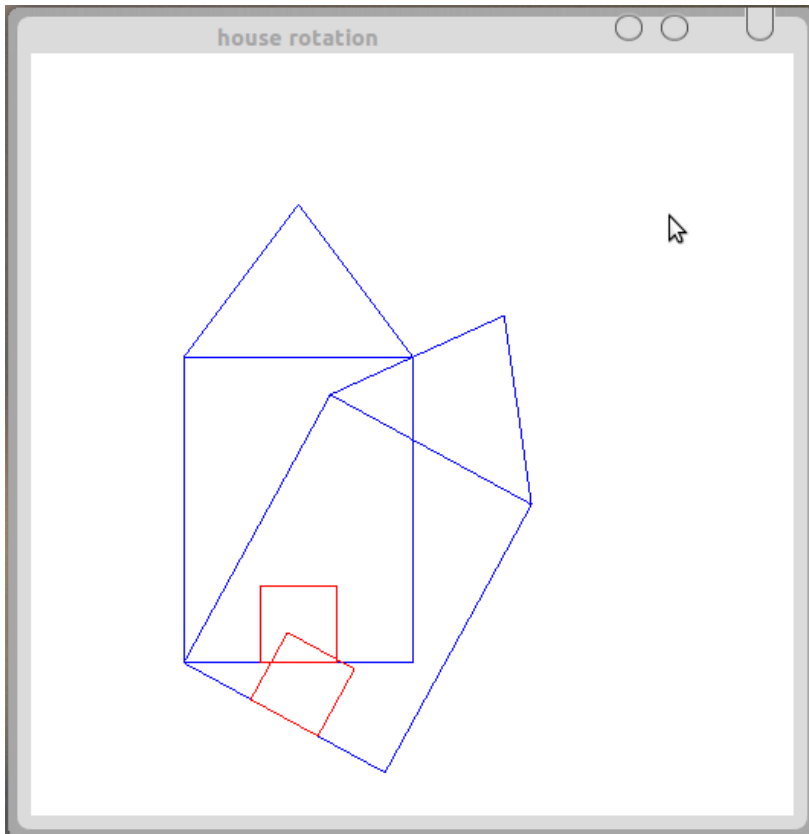
```
glutMainLoop();  
}
```

Output:

Commands for execution:-

- Open a terminal and Change directory to the file location in both the terminals.
- compile as `gcc -lGLU -lGL -lglut rotateHouse.c -o rthouse`
- If no errors, run as `./rthouse`

Screenshots:-





Aim: 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.

Algorithm:

1. End-points pairs are check for trivial acceptance or trivial rejected using the outcode.
2. If not trivial-acceptance or trivial-rejected, divided into two segments at a clip edge.
3. Iteratively clipped by testing trivial-acceptance or trivial-rejected, and divided into two segments until completely inside or trivial-rejected.
4. To perform trivial accept and reject tests, we extend the edges of the clip rectangle to divide the plane of the clip rectangle into nine regions. Each region is assigned a 4-bit code determined by where the region lies with respect to the outside halfplanes of the clip-rectangle edges. Each bit in the outcode is set to either 1 (true) or 0 (false); the 4 bits in the code correspond to the following conditions:
 - Bit 1 : outside halfplane of top edge, above top edge $Y > Y_{max}$
 - Bit 2 : outside halfplane of bottom edge, below bottom edge $Y < Y_{min}$
 - Bit 3 : outside halfplane of right edge, to the right of right edge $X > X_{max}$
 - Bit 4 : outside halfplane of left edge, to the left of left edge $X < X_{min}$

Theory

In computer graphics, 'line clipping' is the process of removing lines or portions of lines outside of an area of interest. Typically, any line or part thereof which is outside of the viewing area is removed. The Cohen-Sutherland algorithm is a computer graphics algorithm used for line clipping. The algorithm divides a two-dimensional space into 9 regions (or a three-dimensional space into 27 regions), and then efficiently determines the lines and portions of lines that are visible in the center region of interest (the viewport). The algorithm quickly detects and dispenses with two common and trivial cases. To clip a line, we need to consider only its endpoints. If both endpoints of a line lie inside the window, the entire line lies inside the window. It is trivially accepted and needs no clipping. On the other hand, if both endpoints of a line lie entirely to one side of the window, the line must lie entirely outside of the window. It is trivially rejected and needs to be neither clipped nor displayed.

Code: cohenSutherland.c

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

// outcode is same as int
#define outcode int

// window boundary
double xmin=50,ymin=50,xmax=100,ymax=100;

// viewport boundary
double xvmin=200,yvmin=200,xvmax=300,yvmax=300;

// bitcode for the right,left,top & bottom
const int RIGHT=8;    // 8 is 1000
const int LEFT=2;     // 2 is 0010
const int TOP=4;      // 4 is 0100
const int BOTTOM=1;   // 1 is 0001
int x1,x2,y1,y2;
```

```

// used to compute bitcode of a point
outcode ComputeOutcode(double x,double y);

// Cohen-Sutherland clipping algo clips a line from
// P0=(x0,y) to P1=(x1,y1) against a rectangle with
// diagonal from (xmin,ymin) to (xmax,ymax)
void CohenSutherlandLineClipAndDraw(double x0,double y0,double x1,double y1)
{
    outcode outcode0,outcode1,outcodeOut;
    bool accept=false,done=false;

    // compute outcodes
    outcode0=ComputeOutcode(x0,y0);
    outcode1=ComputeOutcode(x1,y1);
    do
    {
        // logical or is 0 Trivially accept & exit
        if(!(outcode0|outcode1))
        {
            accept=true;
            done=true;
        }

        // logical and is not 0.Trivially reject & exit
        else if(outcode0 & outcode1)
        {
            done=true;
        }
        else
        {
            // failed both tests,so calculate the line segment to clip
            // from an outside point to an intersection with clip edge
            double x,y;

            //Atleast one endpoint is outside the clip rectangle,pick it.
            outcodeOut=outcode0?outcode0:outcode1;

            //Now find the intersection point
            //use formula  $y=y_0+slope*(x-x_0)$ ,  $x=x_0+(1/slope)*(y-y_0)$ 
            //point is above the rectangular clip
            if(outcodeOut & TOP)
            {
                 $x=x_0+(x_1-x_0)*(y_{max}-y_0)/(y_1-y_0);$ 
                y=ymax;
            }

            // point is below the clip rectangle
            else if(outcodeOut & BOTTOM)
            {
                 $x=x_0+(x_1-x_0)*(y_{min}-y_0)/(y_1-y_0);$ 
                y=ymin;
            }

            // point lies right of clipping rectangle
            else if(outcodeOut & RIGHT)
            {
                 $y=y_0+(y_1-y_0)*(x_{max}-x_0)/(x_1-x_0);$ 
                x=xmax;
            }

            // point lies leftside of clipping rectangle
            else

```



```

        {
            y=y0+(y1-y0)*(xmin-x0)/(x1-x0);
            x=xmin;
        }

        // now we move outside point to intersection point to
        // clip and get ready for next pass
        if(outcodeOut==outcode0)
        {
            x0=x;
            y0=y;
            outcode0=ComputeOutcode(x0,y0);
        }
        else
        {
            x1=x;
            y1=y;
            outcode1=ComputeOutcode(x1,y1);
        }
    }
}
while(!done);
if(accept)
{
    // Window to viewport mappings

    // scaling parameters
    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;

    // draw a red coloured 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();

    // draws blue colour viewport
    glColor3f(0.0,0.0,1.0);
    glBegin(GL_LINES);
    glVertex2d(vx0,vy0);
    glVertex2d(vx1,vy1);
    glEnd();
}
}

// compute the bitrate for a point(x,y) using the clip rectangle
// bounded diagonally by (xmin,ymin) and (xmax,ymax)
outcode ComputeOutcode(double x,double y)
{
    outcode code=0;
    if(y>yvmax)
    {
        code|=TOP;
    }
}

```

```

    else if(y<ymin)
    {
        code|=BOTTOM;
    }
    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);

    // draw the line with red colour
    glColor3f(1.0,0.0,0.0);

    glBegin(GL_LINES);

    <!--ToDo Why are these two lines commented?-->
    //glVertex2d(x0,y0);
    //glVertex2d(x1,y1);
    glVertex2d(x1,y1);
    glVertex2d(x2,y2);
    glEnd();

    // draw blue coloured 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();
    CohenSutherlandLineClipAndDraw(x1,y1,x2,y2);
    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(0.0,500.0,0.0,500.0);
}

void main(int argc,char **argv)
{
    printf("Enter End Points:(x0,x1,y0,y1)");
    scanf("%d%d%d%d",&x1,&x2,&y1,&y2);
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500,500);

```

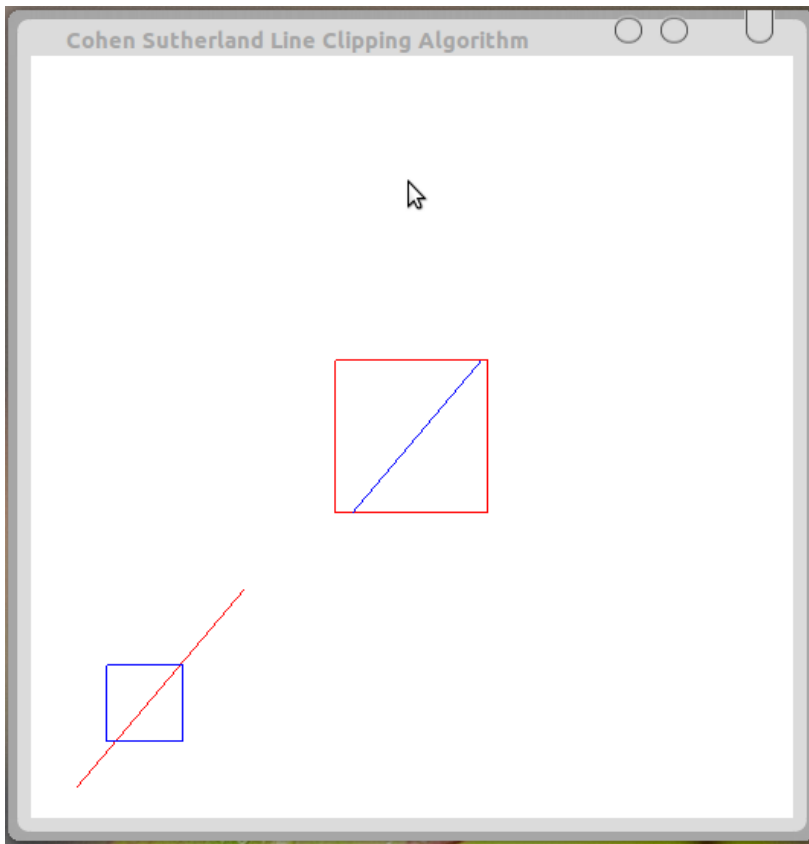
```
glutInitWindowPosition(0,0);  
glutCreateWindow("Cohen Sutherland Line Clipping Algorithm");  
glutDisplayFunc(display);  
myinit();  
glutMainLoop();  
}
```

Output:

Commands for execution:-

- Open a terminal and Change directory to the file location in both the terminals.
- compile as `gcc -lGLU -lGL -lglut cohenSutherland.c -o cohen`
- If no errors, run as `./cohen`

Screenshots:-





Aim:

Program to create a cylinder and a parallelepiped by extruding a circle and quadrilateral respectively.

Algorithm:

1. Under `Cylinder_Draw` function, we call the `Circle_draw` function inside a for loop.
2. The `Circle_Draw` function is a midpoint circle drawing algorithm which draws a circle by calling `plotpixels` method. The `plotpixels` method draws the cylinder by plotting multiple pixels.
3. The `parellopped_draw` module calls the `parellopped` module inside a for loop which draws the parallelepiped onto the screen.

Code:

```
//Cylinder and Parallelepiped by extruding 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
}
int 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 Quadrilaterals"); // Create Display Window
    init();
    glutDisplayFunc(display); // Send the graphics to Display Window
    glutMainLoop();
    return 0;
}

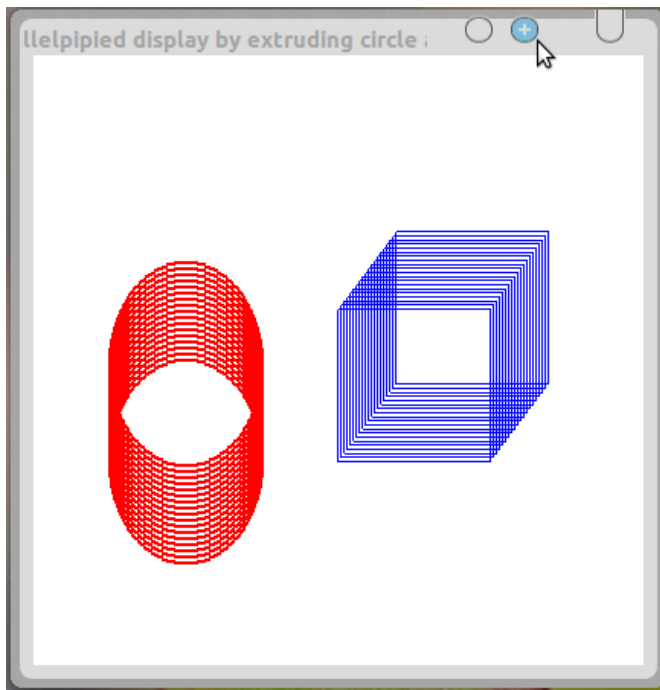
```

Output:

Commands for execution:-

- Open a terminal and Change directory to the file location in both the terminals.
- compile as `gcc -lGLU -lGL -lglut parallelepiped.c -o parallelepiped`
- If no errors, run as `./parallelepiped`

Screenshots:-





Aim:

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 surfaces of the solid object used in the scene.

Algorithm:

1. Use function `glutSolidCube()` to draw wall and table
2. Use the same function to draw 4 cubes and then scale it in downward direction to make it look like tablelegs
3. `glutSolidTeapot()` is used to draw teapot

Code:

```
//teapot.c
#include<stdio.h>
#include<GL/glut.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 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/2.0-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)
```

```

{
    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};

    //The glMaterialfv function specifies material parameters for the lighting model.
    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 lightIntensity[]={0.7f,0.7f,0.7f,0.7f};
    GLfloat light_position[]={2.0f,6.0f,3.0f,0.0f};

    //The glLightfv function returns light source parameter values.
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, lightIntensity);

    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(0.4, 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);
    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);    //draw second wall after rotating x axis by 90degree
    wall(0.02);
    glPopMatrix();
    glPushMatrix();
    glRotated(-90.0, 1.0, 0.0, 0.0);    //draw floor
    wall(0.02);
    glPopMatrix();
    glFlush();
}

int 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 teapot");
glutDisplayFunc(displaySolid);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glShadeModel(GL_SMOOTH);//Specifies a symbolic value representing a shading technique. Accepted values are GL_FLAT and GL_SMOOTH.
glEnable(GL_DEPTH_TEST);
glEnable(GL_NORMALIZE);
glClearColor(0.1,0.1,0.1,0.0);
glViewport(0,0,640,480);
glutMainLoop();
}

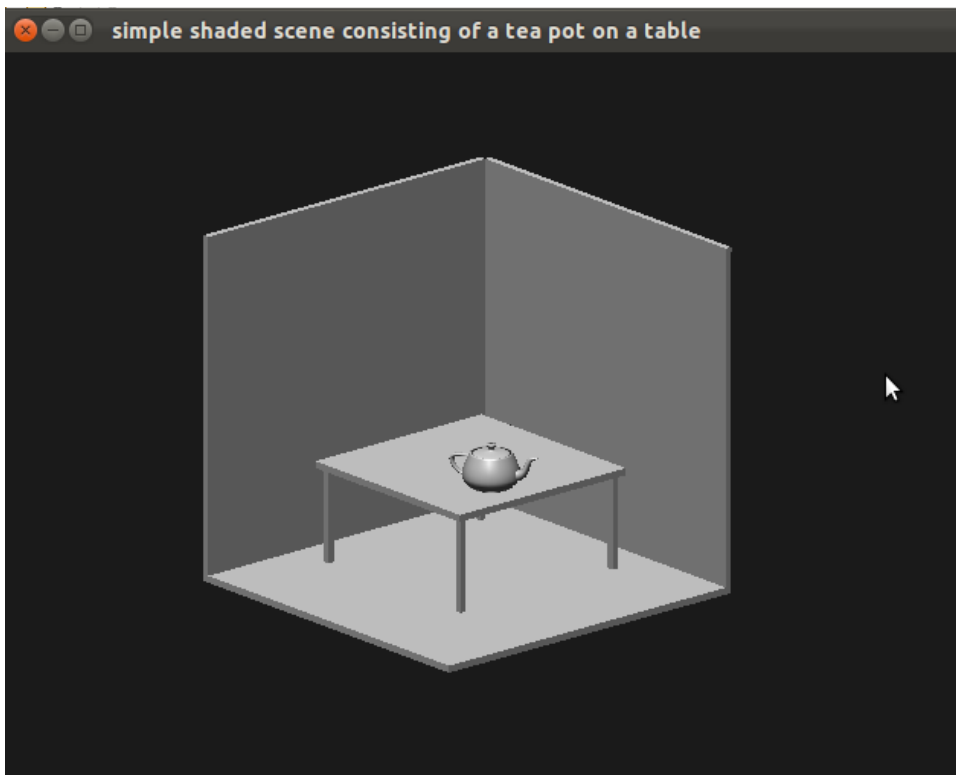
```

Output:

Commands for execution:-

- Open a terminal and Change directory to the file location in both the terminals.
- compile as `gcc -lGLU -lGL -lglut teapot.c -o teapot`
- If no errors, run as `./teapot`

Screenshots:-





Aim:

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

Algorithm

1. Choose appropriate coordinates to make a cube
2. Rotate or move the cube by a small variation on user input
3. Repeat

Code moveCube.c

```
#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()
{
    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;
    glutPostRedisplay();
}

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),-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("colorcube viewer");
    glutReshapeFunc(myReshape);
    glutDisplayFunc(display);
    glutMouseFunc(mouse);
}

```

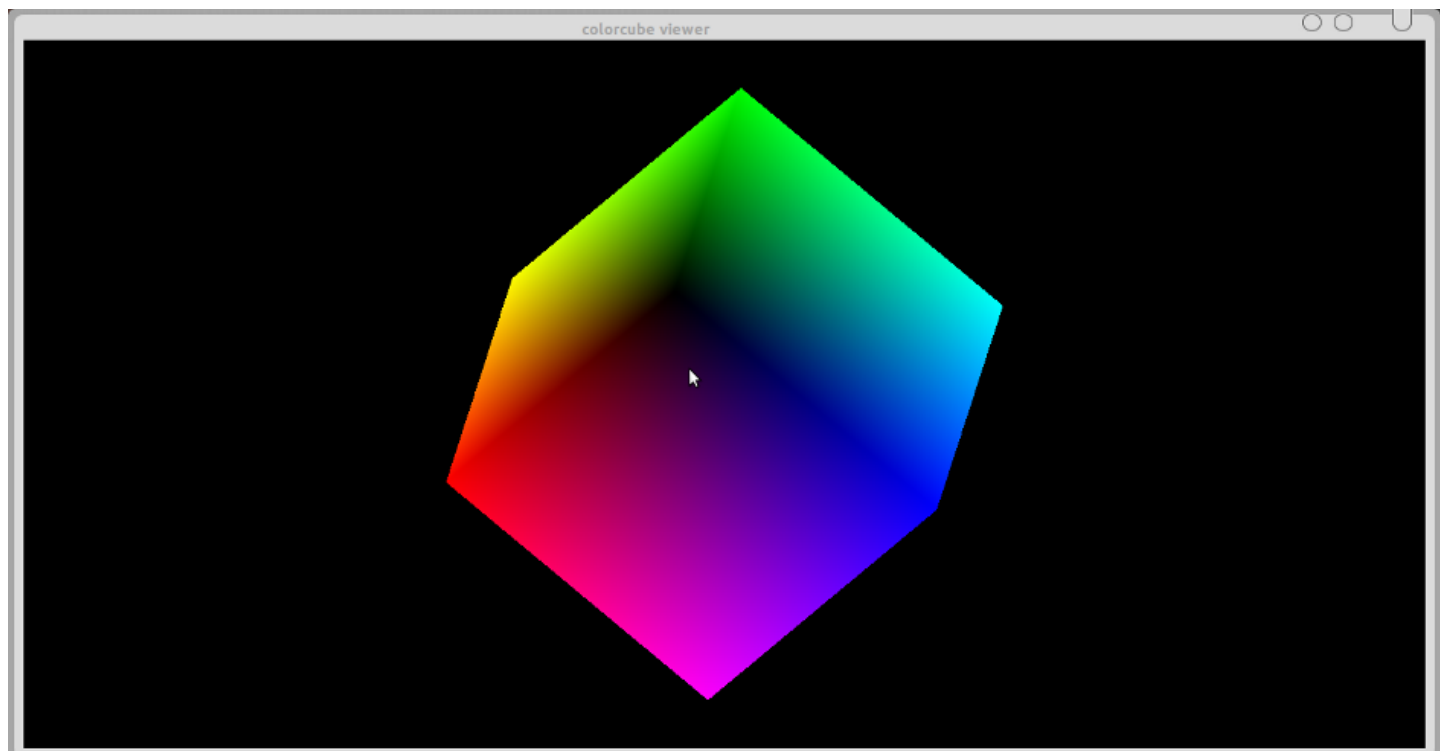
```
glutKeyboardFunc(keys);  
glEnable(GL_DEPTH_TEST);  
glutMainLoop();  
}
```

Output:

Commands for execution:-

- Open a terminal and Change directory to the file location in both the terminals.
- compile as `gcc -lGLU -lGL -lglut moveCube.c -o movecube`
- If no errors, run as `./movecube`

Screenshots:-





Aim:

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

Algorithm:

For each scan line

1. Find the intersections of the scan line with all edges of the polygon.
2. Sort the intersections by increasing x-coordinate.
3. Fill in all pixels between pairs of intersections.

Code :

```
//scanFill.c
#define BLACK 0
#include<stdlib.h>
#include<stdio.h>
#include<GL/glut.h>
float x1,x2,x3,x4,y1,y2,y3,y4;
int k=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,int value)
{
    glColor3f(1.0,0.0,0.0);
    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,j;
for(i=0;i<500;i++)
{
    le[i]=500;
    //le[i]=20;
    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(j=0;j<500;j=j+1)
{
    if(le[j]<=re[j])
        for(i=(int)le[j];i<(int)re[j];i=i+1)
            draw_pixel(i,j,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 mykey(unsigned char key, int x, int y)
{
    if(key=='k')
        k++;
}

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_fill algorithm");
    glutDisplayFunc(display);
    glutKeyboardFunc(mykey);
    myinit();
}

```

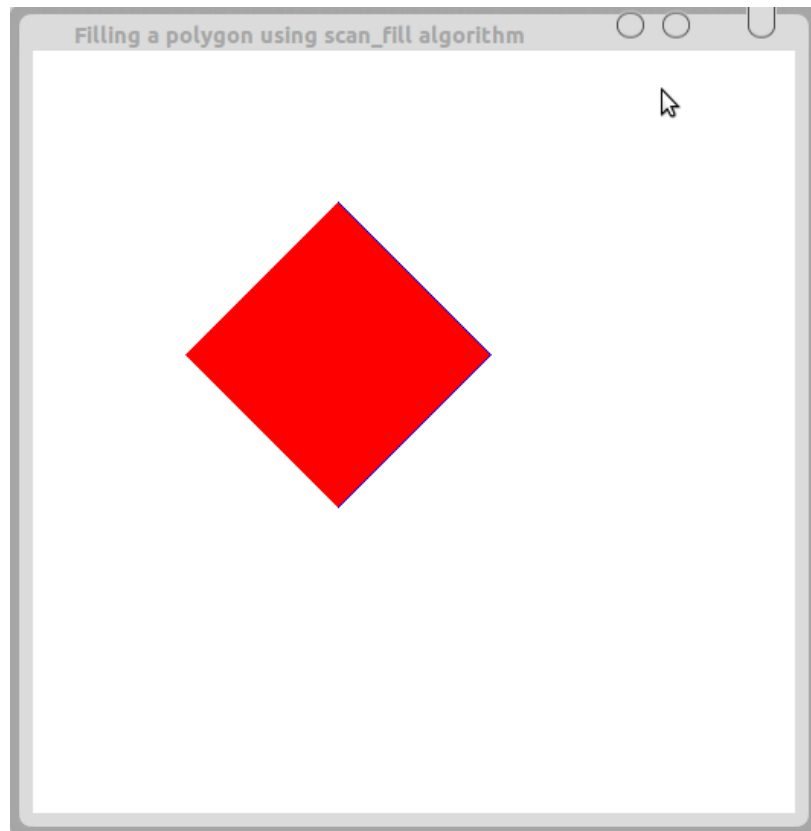
```
    glutMainLoop();  
}
```

Output:

Commands for execution:-

- Open a terminal and Change directory to the file location in both the terminals.
- compile as `gcc -lGLU -lGL -lglut scanfill.c -o scanfill`
- If no errors, run as `./scanfill`.

Screenshots:-





Aim:

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

Algorithm :

1. Define two variables maxx and maxy for the sides of the rectangular mesh.
2. Define dx,dy and also define two arrays x and y.
3. Under init() , we call glLoadIdentity() to start over from the origin.
4. In the display function , you run for loops that update the arrays with (x[i],y[j]) positions.
5. Display the rectangular mesh by passing GL_LINE_LOOP to glBegin as a parameter.

Code: rectMesh.c

```
#include<stdio.h>
#include<stdlib.h>
#include<GL/glut.h>
#define maxx 20
#define maxy 25
#define dx 10
#define dy 15
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(0.0,499.0,0.0,499.0);
    glutPostRedisplay();
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    //glColor3f(1.0,0.0,0.0);
    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();
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 400);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Rectangular Mesh");
    glutDisplayFunc(display);
    init();
    glutMainLoop();
}

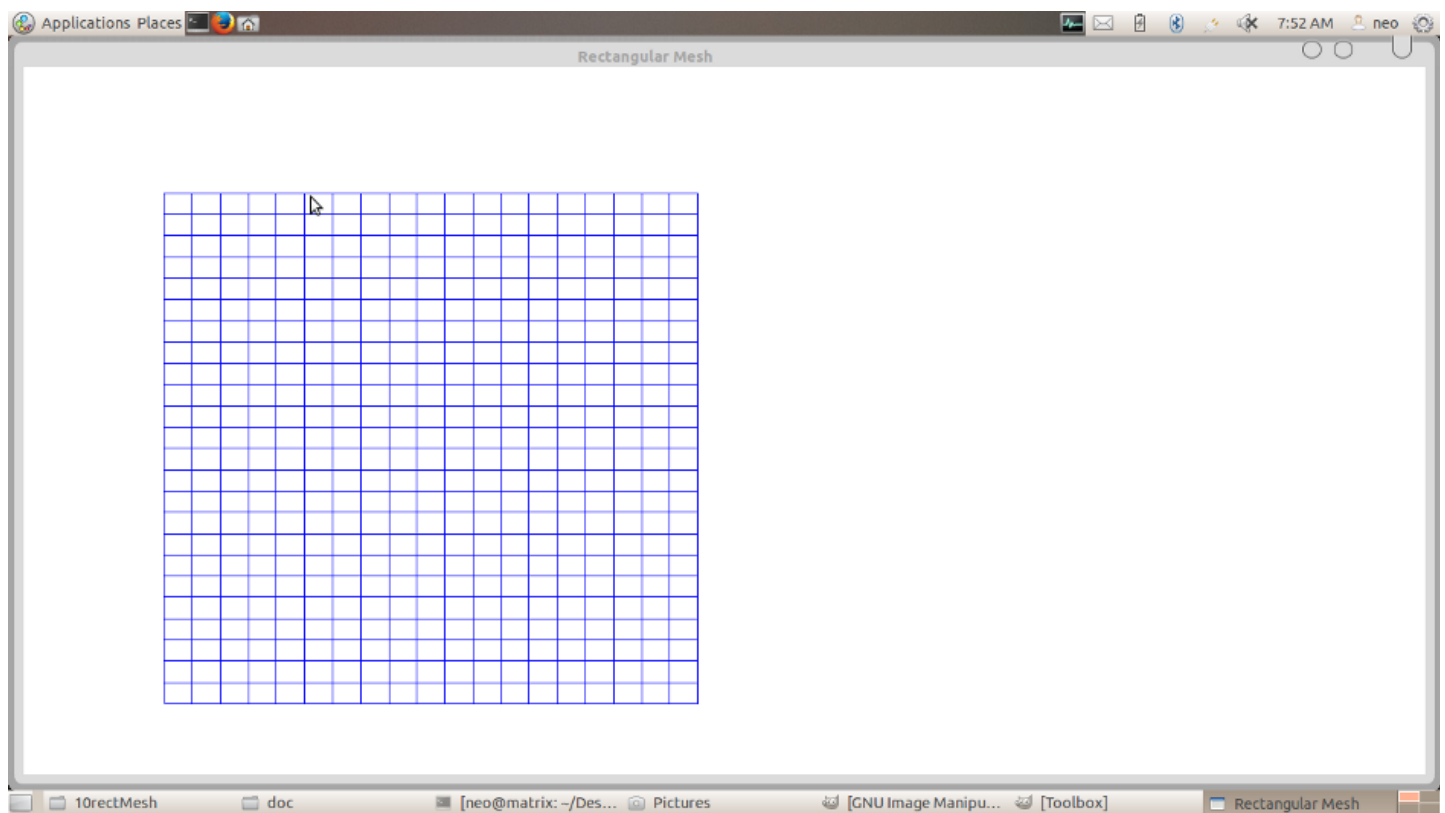
```

Output:

Commands for execution:-

- Open a terminal and Change directory to the file location in both the terminals.
- compile as `gcc -lGLU -lGL -lglut rectMesh.c -o mesh`
- If no errors, run as `./mesh`

Screenshots:-



Tools Required



ubuntu



This book is done using free softwares only



Released under CC By
SA License