SAPTHAGIRI COLLEGE OF ENGINEERING

Accredited by NAAC with "A" Grade, Accredited by NBA for ECE, CSE,ISE,ME,EEE

An ISO 9001 : 2015 & ISO 14001 : 2015 Certified Institution

Affiliated to Visvesvaraya Technological University, Belagavi& Approved by AICTE, New Delhi

#14/5, Chikkasandra, Hesaraghatta Main Road, Bangalore – 560057

Phone:080-28372800/1/2 www.sapthagiri.edu.in Fax: 080-28372797

LAB MANUAL

(2021-22) EVEN SEM

18CSL67
COMPUTER GRAPHICS LABORATORY WITH MINI PROJECT

VI Semester

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Name: ____

USN: ____

Batch: ____

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
Jnana Sangama, Belagavi, Karnataka – 590018

VI SEMESTER / B.E.
COMPUTER GRAPHICS LABORATORY WITH MINI PROJECT LAB MANUAL
(18CSL67)

Prepared By:

Prof. CHAITHRA
Associate Professor, Dept. of CSE
chaithra@sapthagiri.edu.in

VISION AND MISSION OF INSTITUTE

VISION

To be a best institution imparting **quality** engineering education to deal with **community** needs through **learning and performance**.

MISSION

- To **implement path** breaking student centric education methods.
- To augment talent, nurture **teamwork to transform to develop** individual as responsible citizen.
- To educate the students and faculties about entrepreneurship to meet vibrant **requirements of the society.**
- Strengthen **Industry-Institute Interaction** for knowledge sharing

VISION AND MISSION OF DEPARTMENT

Vision

- To be a potential quality source of expert computer engineers, addressing the requirements of industry and civic demands at large.

Mission

- To enable learning in emerging technologies adopting innovative methods.
- To enhance ability, cultivate collaboration to bring change in building students approach towards society.
- To make available platform for harnessing entrepreneurial and leadership qualities.
- To partner continuously with industry to inculcate practical knowledge

PROGRAM EDUCATIONAL OBJECTIVES (PEO)

The program educational objectives of Bachelor of Engineering in Computer Science & Engineering at Sapthagiri College of Engineering are broadly defined on following four counts.

PEO 1: Graduates will have the expertise in analyzing real time problems and providing appropriate solutions related to Computer Science & Engineering.

PEO 2: Graduates will have the knowledge of fundamental principles and innovative technologies to succeed in higher studies, and research.

PEO 3: Graduates will continue to learn and to adapt technology developments combined with deep awareness of ethical responsibilities in profession

PROGRAM SPECIFIC OUTCOMES(PSO)

The graduates of Computer Science and Engineering program at Sapthagiri College of engineering will be able to attain the following at the time of graduation.

PSO 1: Apply the knowledge gained from Mathematics, Basic Computing, Basic Sciences and Social Sciences in general and all computer science courses in particular to identify, formulate and solve real life engineering problems.

PSO2: Identify the various analysis & design methodologies for facilitating development of high quality system software products with focus on performance optimization.

PROGRAM OUTCOMES (POs)

Graduation of **Bachelor of Computer Science and Engineering** program at Sapthagiri College of Engineering will attain the following program outcomes **in the field of Computer Science & Engineering.**

PO1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2. Problem analysis: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Course objectives: This course will enable students to
• Demonstrate simple algorithms using OpenGL Graphics Primitives and attributes. • Implementation of line drawing and clipping algorithms using OpenGL functions • Design and implementation of algorithms Geometric transformations on both 2D and 3D objects.

Course Outcomes: The students should be able to:
• Analyze the concepts of computer graphics • Develop computer graphics applications using line drawing & geometrical transformations OpenGL • Develop real world problems using lighting & shading of OpenGL • Develop 2D & 3D animation using geometrical transformations , clipping & Color Model • Design & develop project using various concepts of Open GL

DO’S AND DON’TS

Do’s

- Do wear ID card and follow dress code. Do log off the computers when you finish.
- Be on time to LAB sessions.
- Do ask for assistance if you need help.
- Do keep your voice low when speaking to others in the LAB.
- Do make suggestions as to how we can improve the LAB.
- In case of any hardware related problem, ask LAB in charge for solution.
- If you are the last one leaving the LAB, make sure that the staff in charge of the LAB is informed to close the LAB
- Do keep the LAB as clean as possible.
- Do prepare for lab programs before entering the lab

Don’ts

- Do not use mobile phone inside the lab.
- Don’t do anything that can make the LAB dirty (like eating, throwing waste papers etc).
- Do not carry any external devices without permission.
- Don’t move the chairs of the LAB.
- Don’t interchange any part of one computer with another.
- Don’t leave the computers of the LAB turned on while leaving the LAB.
- Do not install or download any software or modify or delete any system files on any lab computers.
- Do not damage, remove, or disconnect any labels, parts, cables, or equipment.
- Don’t attempt to bypass the computer security system.
- Do not read or modify other user’s file.
- If you leave the lab, do not leave your personal belongings unattended.
- We are not responsible for any theft.

Scheme of valuation:

Mini Project	10 marks
Record	10 marks
Write-up	08 marks
Execution	08 marks
Viva	04 marks
IA Total	40 marks

INDEX

COMPUTER GRAPHICS LABORATORY WITH MINI PROJECT
Subject Code: 17CSL68

PART A

Design, develop, and implement the following programs using OpenGL API

PART –B (MINI-PROJECT) :

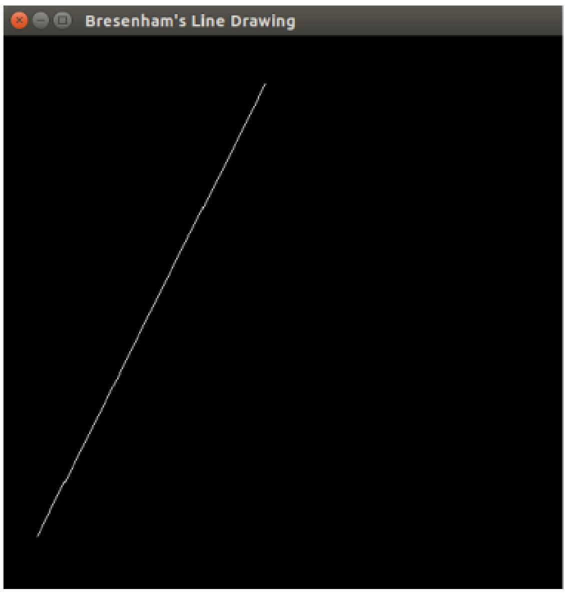
Student should develop mini project on the topics mentioned below or similar applications using Open GL API. Consider all types of attributes like color, thickness, styles, font, background, speed etc., while doing mini project.
(**During the practical exam: the students should demonstrate and answer Viva-Voce**)

PROGRAM 1

AIM:Implement Brenham's line drawing algorithm for all types of slope

```
#include <GL/glut.h>
#include <stdio.h>
int x1, y1, x2, y2;
void myInit() {
glClear(GL_COLOR_BUFFER_BIT);
glClearColor(0.0, 0.0, 0.0, 1.0);
glMatrixMode(GL_PROJECTION);
gluOrtho2D(0, 500, 0, 500);
}
void draw_pixel(int x, int y) {
glBegin(GL_POINTS);
glVertex2i(x, y);
glEnd();
}
void draw_line(int x1, int x2, int y1, int y2) {
int dx, dy, i, e;
int incx, incy, inc1, inc2;
int x,y;
dx = x2-x1;
dy = y2-y1;
if (dx < 0) dx = -dx;
if (dy < 0) dy = -dy;
incx = 1;
if (x2 < x1) incx = -1;

incy = 1;
if (y2 < y1) incy = -1;
x = x1; y = y1;
if (dx > dy) {
draw_pixel(x, y);
e = 2 * dy-dx;
inc1 = 2*(dy-dx);
inc2 = 2*dy;
for (i=0; i<dx; i++) {
if (e >= 0) {
y += incy;
e += inc1;
}
else
e += inc2;
x += incx;
draw_pixel(x, y);
}
} else {
draw_pixel(x, y);
e = 2*dx-dy;
inc1 = 2*(dx-dy);
inc2 = 2*dx;
for (i=0; i<dy; i++) {
if (e >= 0) {
x += incx;
e += inc1;
}
else
e += inc2;
y += incy;
draw_pixel(x, y);
}
}
}
void myDisplay() {
draw_line(x1, x2, y1, y2);
glFlush();
}
int main(int argc, char **argv) {
printf( "Enter (x1, y1, x2, y2)\n");
scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);

glutInitWindowSize(500, 500);
glutInitWindowPosition(0, 0);
glutCreateWindow("Bresenham's Line Drawing");
myInit();
glutDisplayFunc(myDisplay);
glutMainLoop();
return 0;
}
```

Output:



PROGRAM 2

AIM:Create and rotate a triangle about the origin and a fixed point

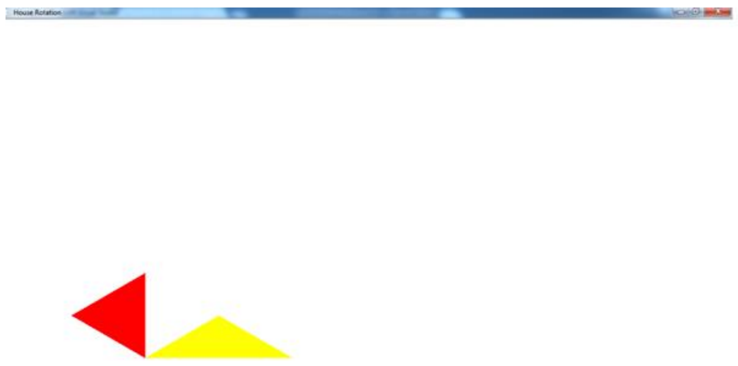
```
#include<stdio.h>
#include<math.h>
#include<GL/glut.h>
GLfloat house[3][3]={{100.0,150.0,200},{100.0,150.0,100.0},{1.0,1.0,1.0}};
GLfloat rot_mat[3][3]={{0},{0},{0}};
GLfloat result[3][3]={{0},{0},{0}};
GLfloat h;
GLfloat k;
GLfloat theta,rad;
int ch;

void multiply()
{
        int i,j,l;
        for(i=0;i<3;i++)
        for(j=0;j<3;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(GLfloat mat[3][3])
{

        glBegin(GL_TRIANGLES);
        glVertex2f(mat[0][0],mat[1][0]);
        glVertex2f(mat[0][1],mat[1][1]);
        glVertex2f(mat[0][2],mat[1][2]);
```

```
	glEnd();
}
void display()
{
	glClear(GL_COLOR_BUFFER_BIT);
	theta=rad;
	glColor3f(1.0,1.0,0.0);
	drawhouse(house);
	if(ch==1)
		{
			h=100;
			k=100;
			rotate();
			glColor3f(1.0,0.0,0.0);
		}
	if(ch==2)
	{
		h=(house[0][0]+house[0][1]+house[0][2])/3;
		k=(house[1][0]+house[1][1]+house[1][2])/3;
		rotate();
		glColor3f(1.0,0.0,1.0);
	}

	drawhouse(result);
	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);
}

int main(int argc,char** argv)
{
	printf("\nEnter the rotation angle : ");
	scanf("%f",&theta);
printf("\n enter the choice 1. fixed point 2. origin");
	scanf("%d",&ch);
	rad=theta*(3.14/180.0);
	glutInit(&argc,argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
	glutInitWindowSize(500,500);
	glutInitWindowPosition(0,0);
	glutCreateWindow("  House Rotation");
	glutDisplayFunc(display);
```

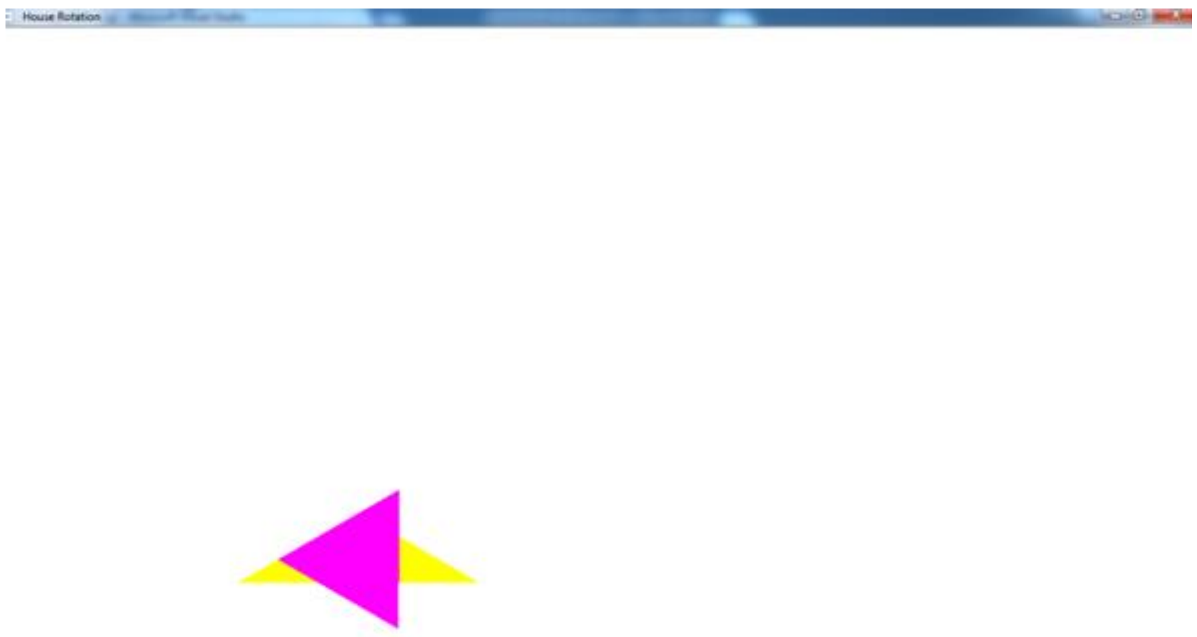
```
        myinit();
        glutMainLoop();
}
```

Output

```
Enter the rotation angle

Enter the choice 1: Fixed point 2:Origin
1
```



```
Enter the rotation angle

Enter the choice 1: Fixed point 2:Origin
2
```

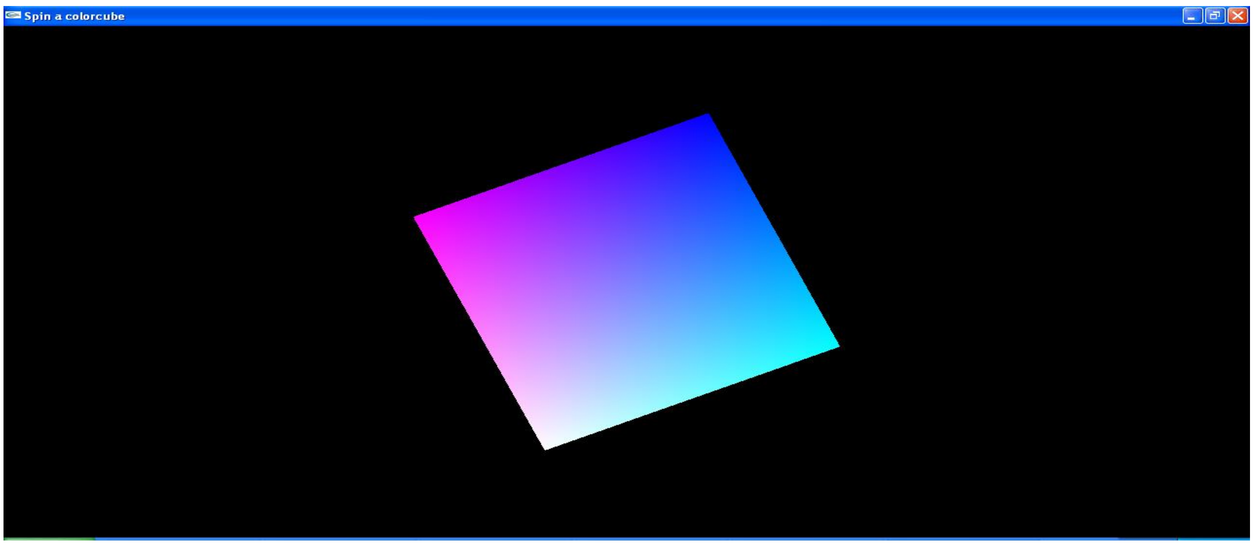


PROGRAM 3

AIM: To draw a color cube and spin it using OpenGL transformation matrices.

```
#include<stdlib.h>
#include<GL/glut.h>
GLfloat vertices[][3]={{-1.0,-1.0,1.0},{-1.0,1.0,1.0},{1.0,1.0,1.0},{1.0,-1.0,1.0},
                        {-1.0,-1.0,-1.0},{-1.0,1.0,-1.0},{1.0,1.0,-1.0},{1.0,-1.0,-1.0}};
GLfloat colors[][3]={{0.0,0.0,0.0},{1.0,0.0,0.0},{1.0,1.0,0.0},{0.0,1.0,0.0},
                        {0.0,0.0,1.0},{1.0,0.0,1.0},{1.0,1.0,1.0},{0.0,1.0,1.0}};
static GLfloat theta[]={0.0,0.0,0.0};
static GLint axis=2;

void polygon(int a,int b,int c,int d)
{
        glBegin(GL_POLYGON);
        glColor3fv(colors[a]);
        glVertex3fv(vertices[a]);
        glColor3fv(colors[b]);
        glVertex3fv(vertices[b]);
        glColor3fv(colors[c]);
        glVertex3fv(vertices[c]);
        glColor3fv(colors[d]);
        glVertex3fv(vertices[d]);
        glEnd();
}
void colorcube()
{
        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);
}
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]-=360.0;
        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 color cube");
   glutReshapeFunc(myReshape);
   glutDisplayFunc(display);
   glutIdleFunc(spincube);
   glutMouseFunc(mouse);
   glEnable(GL_DEPTH_TEST);
   glutMainLoop();
}
```

Output:



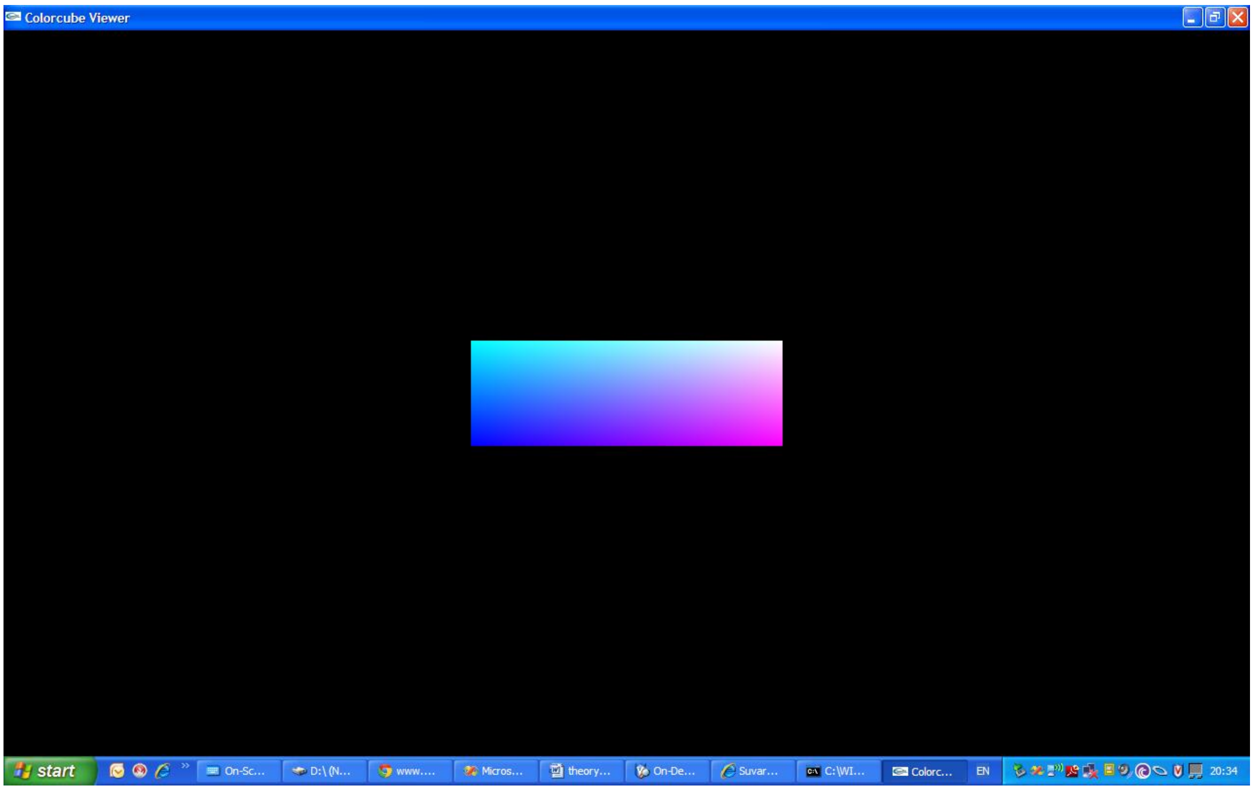
PROGRAM 4

AIM: To draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing.

```
/* Rotating cube with viewer movement */
/* We use the Lookat function in the display callback to point
the viewer, whose position can be altered by the x,X,y,Y,z, and Z keys.
The perspective view is set in the reshape callback */

#include <stdlib.h>
#include <GL/glut.h>
GLfloat vertices[][3] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},
        {1.0,1.0,-1.0}, {-1.0,1.0,-1.0}, {-1.0,-1.0,1.0},
        {1.0,-1.0,1.0}, {1.0,1.0,1.0}, {-1.0,1.0,1.0}};

GLfloat colors[][3] = {{0.0,0.0,0.0},{1.0,0.0,0.0},
        {1.0,1.0,0.0}, {0.0,1.0,0.0}, {0.0,0.0,1.0},
        {1.0,0.0,1.0}, {1.0,1.0,1.0}, {0.0,1.0,1.0}};
static GLfloat theta[] = {0.0,0.0,0.0};
static GLint axis = 2;
static GLdouble viewer[]= {0.0, 0.0, 5.0}; /* initial viewer location */

void polygon(int a, int b, int c , int d)
{
        glBegin(GL_POLYGON);
                glColor3fv(colors[a]);
                glVertex3fv(vertices[a]);
                glColor3fv(colors[b]);
                glVertex3fv(vertices[b]);
                glColor3fv(colors[c]);
                glVertex3fv(vertices[c]);
                glColor3fv(colors[d]);
                glVertex3fv(vertices[d]);
        glEnd();
}
void colorcube()
{
        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);
}
void display(void)
{
         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        /* Update viewer position in modelview matrix */
        glLoadIdentity();
        gluLookAt(viewer[0],viewer[1],viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

        /* rotate cube */
        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;
        display();
}
void keys(unsigned char key, int x, int y)
{
   /* Use x, X, y, Y, z, and Z keys to move viewer */

  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);

        /* Use a perspective view */

         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, 0.0);
        else
                glFrustum(-2.0, 2.0, -2.0 * (GLfloat) w/
                (GLfloat) h, 2.0* (GLfloat) w / (GLfloat) h, 2.0, 20.0);

        /* Or we can use gluPerspective */

         /* gluPerspective(45.0, w/h, -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("Colorcube Viewer");
        glutReshapeFunc(myReshape);
        glutDisplayFunc(display);
       glutMouseFunc(mouse);
       glutKeyboardFunc(keys);
       glEnable(GL_DEPTH_TEST);
        glutMainLoop();
}
```

Output:



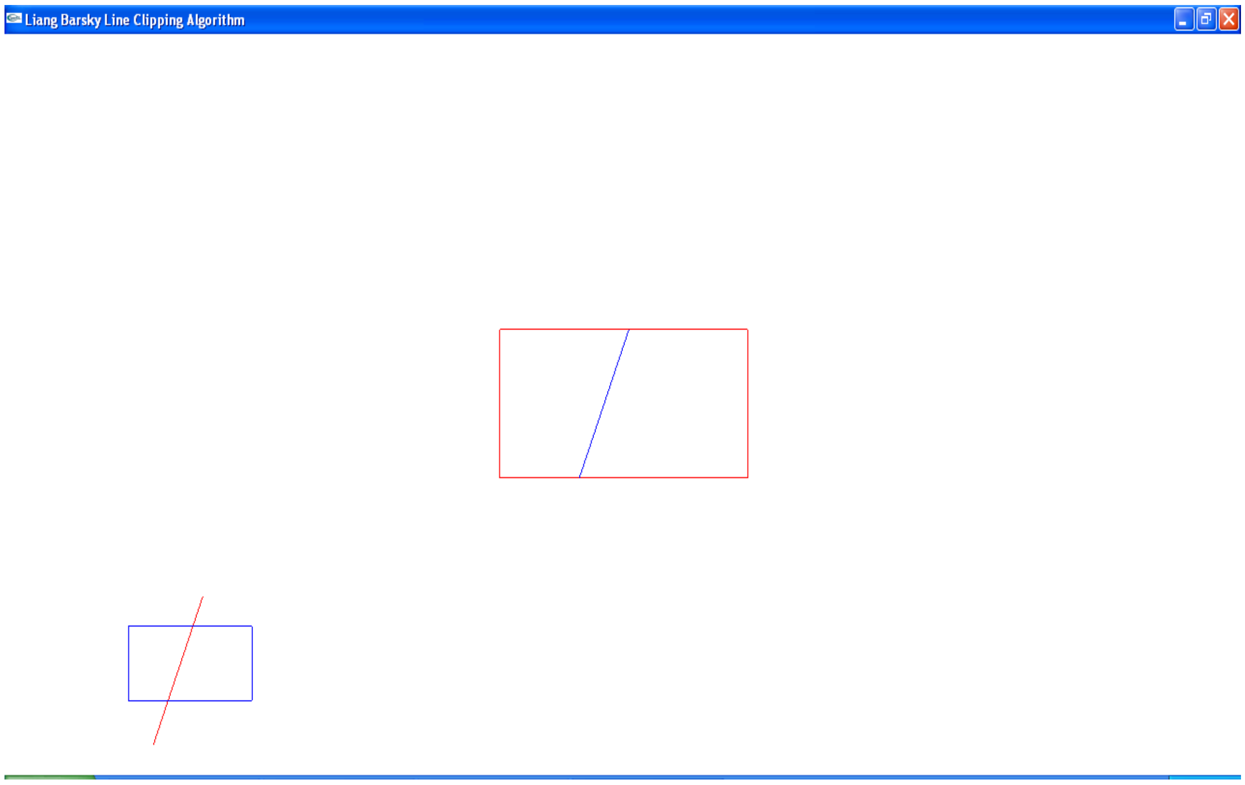
PROGRAM 5

AIM: To clip lines using Cohen Sutherland algorithm

```
#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;
double x0,y0,x1,y1;
const int RIGHT=8;
const int LEFT=2;
const int TOP=4;
const int BOTTOM=1;
outcode ComputeOutCode(double x, double y);

void CohenSutherland(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&BOTTOM)
                        {
                                x=x0+(x1-x0)*(ymin-y0)/(y1-y0);
                                y=ymin;
                        }
                        else if(outcodeOut&RIGHT)
                        {
                                y=y0+(y1-y0)*(xmax-x0)/(x1-x0);
                                x=xmax;
                        }
                        else
                        {
                                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=BOTTOM;
        if(x>xmax)
                code=RIGHT;
```

```
	else if(x<xmin)
		code=LEFT;
	return code;
}

void display()
{
	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();
	CohenSutherland(x0,y0,x1,y1);
	glFlush();
}

void myinit()
{
	glClearColor(1.0,1.0,1.0,1.0);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(0.0,499.0,0.0,499.0);
}


void main(int argc, char** argv)
{
	printf("Enter the end points of the line: ");
	scanf("%lf%lf%lf%lf", &x0,&y0,&x1,&y1);
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutInitWindowSize(500,500);
	glutInitWindowPosition(0,0);
	glutCreateWindow("Cohen-Sutherland Line Clipping");
	glutDisplayFunc(display);
	myinit();
	glutMainLoop();
}
```

Output:



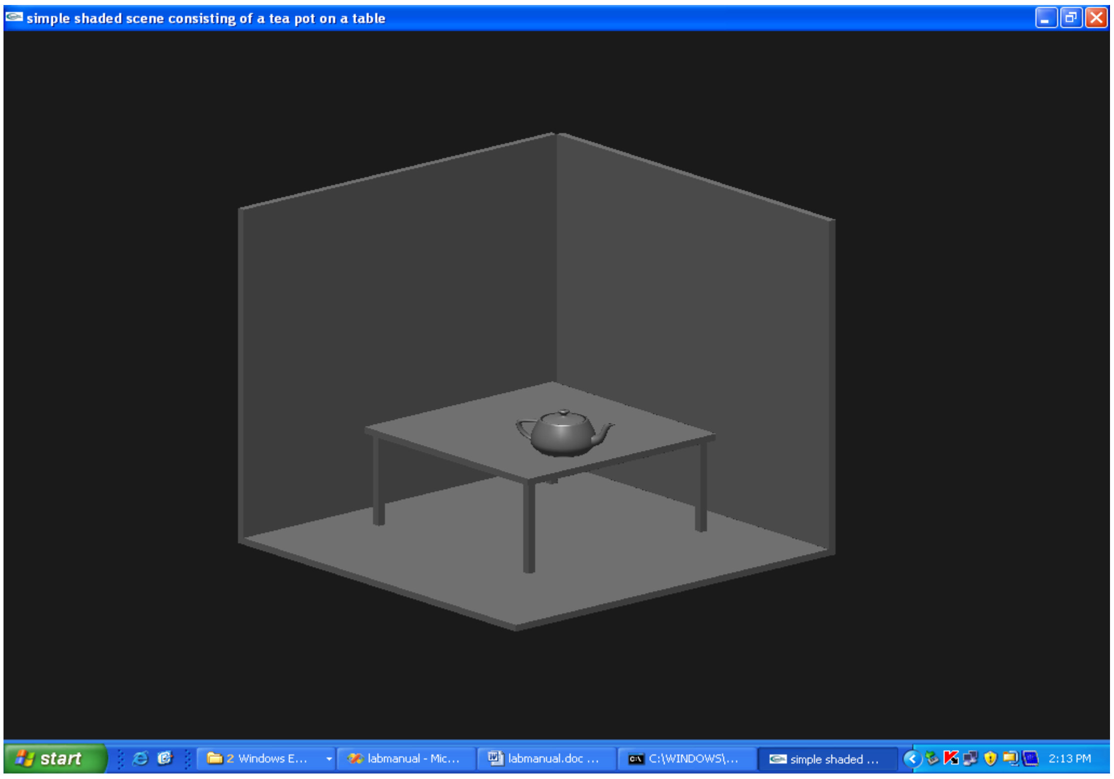
PROGRAM 6

Aim:To draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of light source along with the properties of the surfaces of the solid object used in the scene.

```
#include<gl/glut.h>
void obj(double tx,double ty,double tz,double sx,double sy,double sz)
{
glRotated(50,0,1,0);
glRotated(10,-1,0,0);
glRotated(11.7,0,0,-1);
glTranslated(tx,ty,tz);
glScaled(sx,sy,sz);
glutSolidCube(1);
glLoadIdentity();
}
void display()
{
glViewport(0,0,700,700);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
obj(0,0,0.5,1,1,0.04); // three walls
obj(0,-0.5,0,1,0.04,1);
obj(-0.5,0,0,0.04,1,1);
obj(0,-0.3,0,0.02,0.2,0.02); // four table legs
obj(0,-0.3,-0.4,0.02,0.2,0.02);
obj(0.4,-0.3,0,0.02,0.2,0.02);
obj(0.4,-0.3,-0.4,0.02,0.2,0.02);
obj(0.2,-0.18,-0.2,0.6,0.02,0.6); // table top
glRotated(50,0,1,0);
glRotated(10,-1,0,0);
glRotated(11.7,0,0,-1);
glTranslated(0.3,-0.1,-0.3);
glutSolidTeapot(0.09);
glFlush();
glLoadIdentity();
}
void main()
{
float ambient[]={1,1,1,1};
float light_pos[]={27,80,2,3};
glutInitWindowSize(700,700);
glutCreateWindow("scene");
glutDisplayFunc(display);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glMaterialfv(GL_FRONT,GL_AMBIENT,ambient);
glLightfv(GL_LIGHT0,GL_POSITION,light_pos);
glEnable(GL_DEPTH_TEST);
glutMainLoop();
```

}

Output:

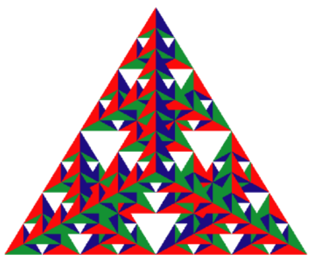

simple shaded scene consisting of a tea pot on a table
start
2 Windows E...
labmanual - Mic...
labmanual.doc ...
C:\WINDOWS\...
simple shaded ...
2:13 PM

PROGRAM 7

AIM: To recursively subdivided a Tetrahedron to form 3D Sierpinski Gasket. The number of recursive steps is to be specified by the user.

```
#include<stdio.h>
#include<GL/glut.h>
typedef float point[3];
point v[]={{0.,0.0,1.0},{0.0,0.942809,-0.3333},
        {-0.816497,-0.471405,-0.3333},
        {0.816497,-0.471405,0.3333}};
int n;
void triangle(point a,point b,point c)
{
        glBegin(GL_POLYGON);
          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]=(c[j]+b[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 Division?: ");
        scanf("%d",&n);
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH;
        glutCreateWindow("3D gasket");
        glutReshapeFunc(myReshape);
        glutDisplayFunc(display);
        glEnable(GL_DEPTH_TEST);
        glClearColor(1.0,1.0,1.0,0.0);
        glutMainLoop();
}
```

Output:



Enter the Number of Divisions: 5

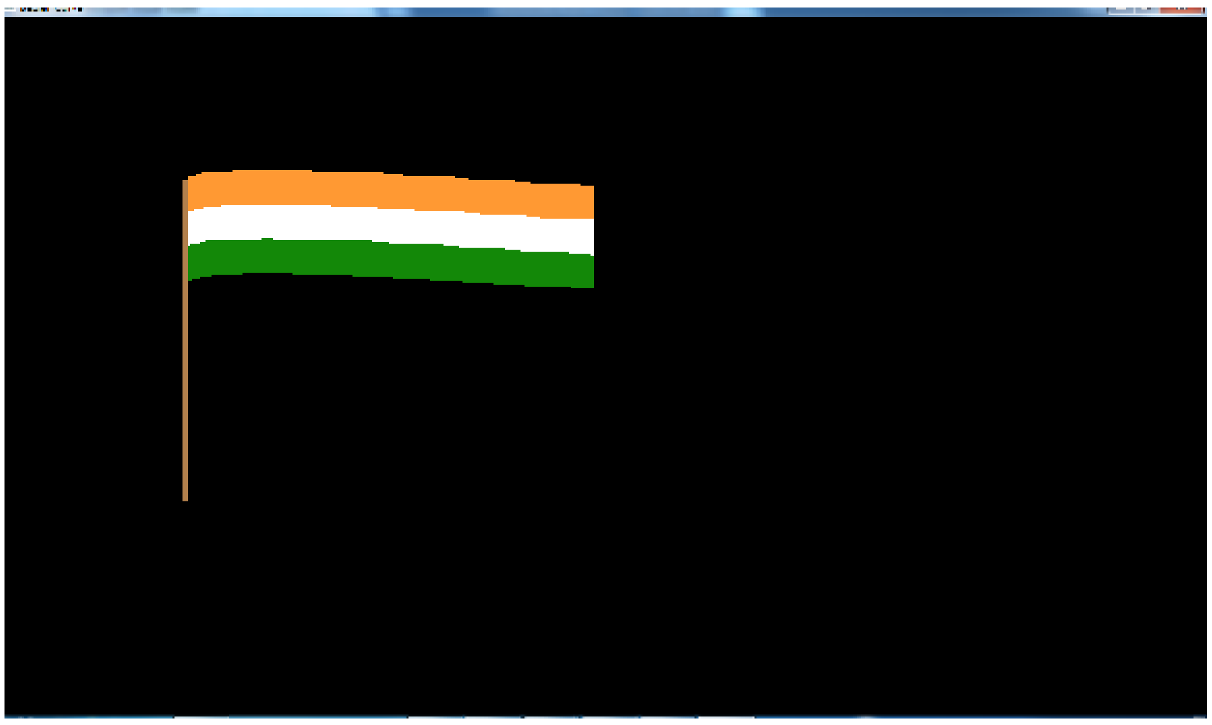
PROGRAM 8

Aim: Develop a menu driven program to animate a flag using Bezier curve algorithm
Output :

```
#include<GL/glut.h>
#include<stdio.h>
#include<math.h>
#define PI 3.1416

float theta = 0;

struct point
{
   GLfloat x, y, z;
};

int factorial(int n)
{
   if (n<=1)
      return(1);
   else
      n=n*factorial(n-1);
   return n;
}

void computeNcR(int n, int *hold_ncr_values)
{
   int r;
   for(r=0; r<=n; r++) //start from nC0, then nC1, nC2, nC3 till nCn
   {
      hold_ncr_values[r] = factorial(n) / (factorial(n-r) * factorial(r));
   }
}

void computeBezierPoints(float t, point *actual_bezier_point, int number_of_control_points,
point *control_points_array, int *hold_ncr_values)
{
   int i, n = number_of_control_points-1;

   float bernstein_polynomial;

   actual_bezier_point -> x = 0;
   actual_bezier_point -> y = 0;
   actual_bezier_point -> z = 0;

   for(i=0; i < number_of_control_points; i++)
   {
      bernstein_polynomial = hold_ncr_values[i] * pow(t, i) * pow( 1-t, n-i);

      actual_bezier_point -> x += bernstein_polynomial * control_points_array[i].x;
```

```
        actual_bezier_point -> y += bernstein_polynomial * control_points_array[i].y;
        actual_bezier_point -> z += bernstein_polynomial * control_points_array[i].z;
    }
}

void bezier(point *control_points_array, int number_of_control_points, int
number_of_bezier_points)
{
    point actual_bezier_point;
    float t;
    int *hold_ncr_values, i;

    hold_ncr_values = new int[number_of_control_points]; // to hold the nCr values

    computeNcR(number_of_control_points-1, hold_ncr_values); // call nCr function to
calculate nCr values

    glBegin(GL_LINE_STRIP);
        for(i=0; i<=number_of_bezier_points; i++)
        {
            t=float(i)/float(number_of_bezier_points);
            computeBezierPoints(t, &actual_bezier_point, number_of_control_points,
control_points_array, hold_ncr_values);
            glVertex2f(actual_bezier_point.x, actual_bezier_point.y);
        }
    glEnd();

    delete[] hold_ncr_values;
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    int number_of_control_points = 4, number_of_bezier_points = 20;

    point control_points_array[4] = {{100, 400, 0}, {150, 450, 0}, {250, 350, 0},{300, 400,
0}};

    control_points_array[1].x += 50*sin(theta * PI/180.0);
    control_points_array[1].y += 25*sin(theta * PI/180.0);

    control_points_array[2].x -= 50*sin((theta+30) * PI/180.0);
    control_points_array[2].y -= 50*sin((theta+30) * PI/180.0);

    control_points_array[3].x -= 25*sin((theta-30) * PI/180.0);
    control_points_array[3].y += sin((theta-30) * PI/180.0);

    theta += 2;

    glPushMatrix();
    glPointSize(5);

    glColor3f(1, 0.4, 0.2); //Indian flag: Saffron color code
    for(int i=0; i<50; i++)
    {
        glTranslatef(0, -0.8, 0 );
        bezier(control_points_array, number_of_control_points, number_of_bezier_points);
    }

    glColor3f(1, 1, 1); //Indian flag: white color code
    for(int i=0; i<50; i++)
    {
        glTranslatef(0, -0.8, 0);
        bezier(control_points_array, number_of_control_points, number_of_bezier_points);
    }

    glColor3f(0, 1, 0); //Indian flag: green color code
    for(int i=0; i<50; i++)
    {
        glTranslatef(0, -0.8, 0);
        bezier(control_points_array, number_of_control_points, number_of_bezier_points);
    }

    glPopMatrix();

    glLineWidth(5);

    glColor3f(0.7, 0.5,0.3); //pole colour
    glBegin(GL_LINES);
        glVertex2f(100,400);
        glVertex2f(100,40);
    glEnd();

    glutPostRedisplay();
    glutSwapBuffers();
}

void init()
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0,500,0,500);
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowPosition(0, 0);
```

```
   glutInitWindowSize(500,500);
   glutCreateWindow("Bezier Curve - updated");

   init();

   glutDisplayFunc(display);
   glutMainLoop();
}
```



PROGRAM 9

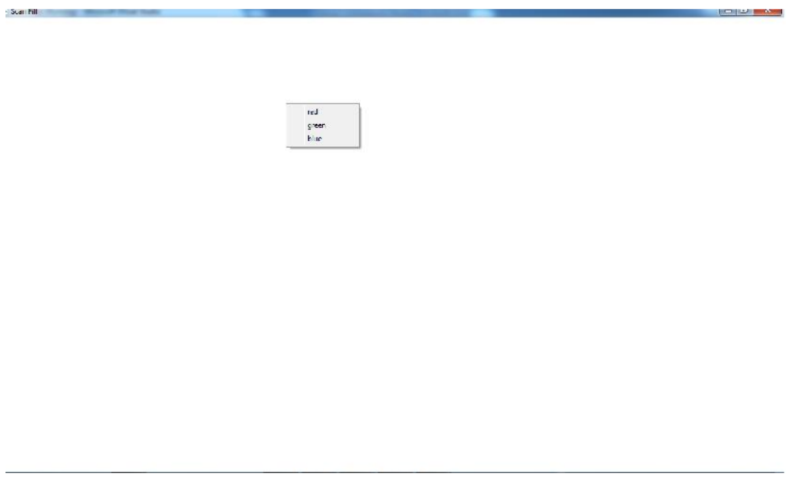
Aim: Develop a menu driven program To fill the polygon using scan-line algorithm

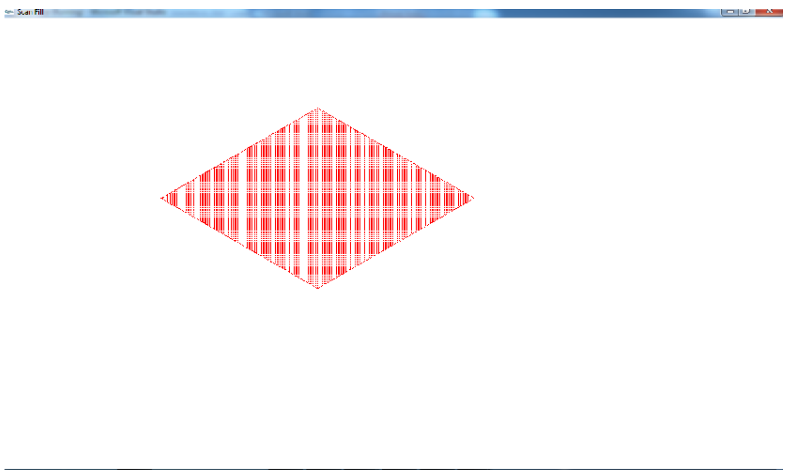
```
#include<GL/glut.h>
#define BLACK 0
float x1,x2,x3,x4,y1,y2,y3,y4;
void edgedetect(float x1,float y1,float x2,float y2,int *le,int *re)
{
        float mx,x,temp;
        int i;
        if((y2-y1)<0)
        {
                temp=y1; y1=y2; y2=temp;
                temp=x1; x1=x2; x2=temp;
        }
        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)
{
        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);
                    glFlush();

                  }
        }

}

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);

        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);

        glPointSize(1.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0,499.0,0.0,499.0);
}

void menu(int id)
{
        switch(id)
        {
        case 0: glColor3f(1.0,0.0,0.0);
                break;
        case 1: glColor3f(0.0,1.0,0.0);
                break;
        case 2: glColor3f(0.0,0.0,1.0);
                break;
        }
```

```
            glutPostRedisplay();
}

int main(int argc,char **argv)
{
            glutInit(&argc,argv);
            glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
            glutInitWindowSize(500,500);
            glutInitWindowPosition(0,0);
            glutCreateWindow("Scan Fill");
            glutDisplayFunc(display);
            glutCreateMenu(menu);
     glutAddMenuEntry("red",0);
     glutAddMenuEntry("green", 1);
     glutAddMenuEntry("blue", 2);
     glutAttachMenu(GLUT_LEFT_BUTTON);
            myinit();
            glutMainLoop();
            return 0;
}
```

Output:




Scan Fill