Computer Graphics and Visualization Laboratory (15CSL68)

1. Implement Brenham’s line drawing algorithm for all types of slope.

```
#include<GL/glut.h>
#include<stdio.h>
int x1,y2,x2,y3;

void drawText(float x, float y, float z,char* s){
        int i;
        glRasterPos3f(x,y,z);
        for(i=0;s[i] != '\0';i++)
                glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,s[i]);
}

void myInit()
{
    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 y3,int y2)
{
    int dx,dy,i,e;
    int incx,incy,inc1,inc2;
    int x,y;
    dx=x2-x1;
    dy=y2-y3;
    if(dx<0) dx=-dx;
    if(dy<0) dy=-dy;
    incx=1;
    if(x2<x1) incx=-1;
    incy=1;
    if(y2<y3) incy=-1;
    x=x1;
    y=y3;
    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()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0,0.0,0.0);
        drawText(50,200,0.5,"Line Drawing");
        drawText(50,210,0.7,"K K NITHIN B-2  1BI15CS066");
    draw_line(x1,x2,y3,y2);
    glFlush();
}

int main(int argc,char ** argv)
{
    printf("Enter end points of the line (x1,y1,x2,y2)\n");
    scanf("%d%d%d%d",&x1,&y3,&x2,&y2);
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Bresenhams Line Drawing");
    myInit();
    glutDisplayFunc(myDisplay);
    glutMainLoop();
    return 0;
}
```

2. Create and rotate a triangle about the origin and a fixed point.

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

GLfloat
triangle[3][3]={{100.0,250.0,175.0},{100.0,100.0,300.0},{1.0,1.0,1.0}};
GLfloat rotatemat[3][3]={{0},{0},{0}};
GLfloat result[3][3]={{0},{0},{0}};
GLfloat arbitrary_x=0;
GLfloat arbitrary_y=0;
float rotation_angle;

void draw_text(float x,float y,char* s)
{
     int i;
     glRasterPos2f(x,y);
     for(i=0;s[i]!='\0';i++)
     glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,s[i]);
}

void multiply()
{
     int i,j,k;
     for(i=0;i<3;i++)
          for(j=0;j<3;j++)
```

```
        {
            result[i][j]=0;
            for(k=0;k<3;k++)

    result[i][j]=result[i][j]+rotatemat[i][k]*triangle[k][j];
        }
}

void rotate()
{
    GLfloat m,n;
    rotation_angle=(3.14*rotation_angle)/180;
    m=-arbitrary_x*(cos(rotation_angle)-
1)+arbitrary_y*(sin(rotation_angle));
    n=-arbitrary_y*(cos(rotation_angle)-1)-
arbitrary_x*(sin(rotation_angle));
    rotatemat[0][0]=cos(rotation_angle);
    rotatemat[0][1]=-sin(rotation_angle);
    rotatemat[0][2]=m;
    rotatemat[1][0]=sin(rotation_angle);
    rotatemat[1][1]=cos(rotation_angle);
    rotatemat[1][2]=n;
    rotatemat[2][0]=0;
    rotatemat[2][1]=0;
    rotatemat[2][2]=1;
    multiply();
}

void drawtriangle()
{
    glColor3f(0.0,0.0,1.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(triangle[0][0],triangle[1][0]);
    glVertex2f(triangle[0][1],triangle[1][1]);
    glVertex2f(triangle[0][2],triangle[1][2]);
    glEnd();
}

void drawrotatedtriangle()
{
    glColor3f(1.0,0.0,0.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(result[0][0],result[1][0]);
    glVertex2f(result[0][1],result[1][1]);
    glVertex2f(result[0][2],result[1][2]);
    glEnd();
}
```

```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    drawtriangle();
    draw_text(-450,50,"USN:1BI15CS066, NAME:K K NITHIN, BATCH:B-2");
    draw_text(-450,10,"Rotation about Origin, Degree=90");
    drawrotatedtriangle();
    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(-499.0,499.0,-499.0,499.0);
}

int main(int argc,char **argv)
{
    int ch;
    printf("Enter your choice \n 1.Rotation about origin \n 2.Rotation 
about a Fixed point\n");
    scanf("%d",&ch);
    switch(ch)
    {
        case 1:
            printf("Enter the rotation angle in degree:");
            scanf("%f",&rotation_angle);
            rotate();
            break;
        case 2:
            printf("Enter the fixed points:");
            scanf("%f%f",&arbitrary_x,&arbitrary_y);
            printf("Enter rortation angle in degree:");
            scanf("%f",&rotation_angle);
            rotate();
            break;
    }
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Triangle Rotation");
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
    return 0;
}
```

3. Draw a colour 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 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 draw_text(float x,float y,char* s)
{
    int i;
    glRasterPos2f(x,y);
    for(i=0;s[i]!='\0';i++)
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,s[i]);
}

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();
    draw_text(-0.2,1.7,"USN:1BI15CS066, NAME:K K NITHIN, BATCH:B-2");
    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]+=1.0;
    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("Rotating a Color Cube");
     glutReshapeFunc(myReshape);
     glutDisplayFunc(display);
     glutIdleFunc(spinCube);
     glutMouseFunc(mouse);
     glEnable(GL_DEPTH_TEST);
     glutMainLoop();
}
```

4. Draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing.

```
#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]={{1.0,1.0,1.0},{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 draw_text(float x,float y,char* s)
{
    int i;
    glRasterPos2f(x,y);
    for(i=0;s[i]!='\0';i++)
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,s[i]);
}

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;
static GLdouble viewer[]={0.0,0.0,5.0};/* initial viewer location*/

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);
    draw_text(-0.2,1.7,"USN:1BI15CS066, NAME:K K NITHIN, BATCH:B-2");
    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)
```

```
{
     if(key=='x')viewer[0]-=1.0;
     if(key=='X')viewer[0]+=1.0;
     if(key=='y')viewer[0]-=1.0;
     if(key=='Y')viewer[0]+=1.0;
     if(key=='z')viewer[0]-=1.0;
     if(key=='Z')viewer[0]+=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);
}

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();
}
```

5. Clip a lines using Cohen-Sutherland algorithm.

```
#include<stdio.h>
#include<GL/glut.h>
#include<stdbool.h>
double xmin=50,ymin=50,xmax=100,ymax=100;
double xvmin=200,yvmin=200,xvmax=300,yvmax=300;
double xmat[6][2],ymat[6][2];
const int TOP=8,BOTTOM=4,RIGHT=2,LEFT=1;
```

```
int computeoutcode(double x,double y)
{
    int 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 draw_text(float x,float y,char *s)
{
    int i=0;
    glRasterPos2f(x,y);
    for(i=0;s[i]!='\0';i++)
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,s[i]);
}
void cohensutherland(double x0,double y0,double x1,double y1)
{
    int outcode0,outcode1,outcodeout;
    bool accept=false,done=false;
    outcode0=computeoutcode(x0,y0);
    outcode1=computeoutcode(x1,y1);
    do
    {
        if((outcode0|outcode1)==0)
        {
            accept=true;
            done=true;
        }
        else if(outcode0&outcode1)
            done=true;
        else
        {
            double x,y;
            outcodeout=outcode0?outcode0:outcode1;
            float slope=(y1-y0)/(x1-x0);
            if(outcodeout&TOP)
            {
                x=x0+(1/slope)*(ymax-y0);
                y=ymax;
            }
            else if(outcodeout&BOTTOM)
            {
                x=x0+(1/slope)*(ymin-y0);
                y=ymin;
            }
            else if(outcodeout&RIGHT)
            {
                y=y0+slope*(xmax-x0);
                x=xmax;
            }
            else
            {
                y=y0+slope*(xmin-x0);
                x=xmin;
            }
            if(outcodeout==outcode0)
            {
                x0=x;
                y0=y;
                outcode0=computeoutcode(x0,y0);
            }
            else
            {
                x1=x;
                y1=y;
                outcode1=computeoutcode(x1,y1);
            }
        }
    }while(!done);
    if(accept)
    {
        double sx=(xvmax-xvmin)/(xmax-xmin);
        double sy=(yvmax-yvmin)/(ymax-ymin);
        double vx0=xvmin+(x0-xmin)*sx;
        double vy0=yvmin+(y0-ymin)*sy;
        double vx1=xvmin+(x1-xmin)*sx;
        double vy1=yvmin+(y1-ymin)*sy;
        glColor3f(0.0,0.0,1.0);
        glBegin(GL_LINES);
        glVertex2f(vx0,vy0);
        glVertex2f(vx1,vy1);
        glEnd();
    }
}
void display()
{
    int i;
    glClear(GL_COLOR_BUFFER_BIT);
    draw_text(50,450,"Cohen-Sutherland Line Clipping");
    draw_text(50,430,"K K NITHIN");
    draw_text(50,410,"1BI15CS066");
    draw_text(50,390,"Batch:B2");
    for(i=0;i<6;i++)
    {
        glColor3f(1.0,0.0,0.0);
        glBegin(GL_LINES);
        glVertex2f(xmat[i][0],ymat[i][0]);
        glVertex2f(xmat[i][1],ymat[i][1]);
        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();
    glColor3f(1.0,0.0,0.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(xvmin,yvmin);
    glVertex2f(xvmax,yvmin);
    glVertex2f(xvmax,yvmax);
    glVertex2f(xvmin,yvmax);
    glEnd();
    for(i=0;i<6;i++)
        cohensutherland(xmat[i][0],ymat[i][0],xmat[i][1],ymat[i][1]);
    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(0,500,0,500);
}
void main(int argc,char **argv)
{
    int i;
    printf("Enter the end points of the lines\n");
    for(i=0;i<6;i++)
    {
        printf("For line %d:",i+1);
     scanf("%lf%lf%lf%lf",&xmat[i][0],&ymat[i][0],&xmat[i][1],&ymat[i][1]);
    }
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutCreateWindow("Cohen Sutherland Line Clipping");
    glutInitWindowSize(1000,1000);
    glutInitWindowPosition(0,0);
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
}
```

6. 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.

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

GLfloat mat_ambient[]={1.0,0.3,0.3,1.0};
GLfloat mat_diffuse[]={0.5,0.5,0.5,1.0};
GLfloat mat_specular[]={1.0,1.0,1.0,1.0};
const GLfloat mat_shininess[]={50.0};
GLfloat Light_intensity[]={0.7,0.7,0.7,1.0};
GLfloat Light_position[]={2.0,6.0,3.0,0.0};

void init()
{
    glMaterialfv(GL_FRONT,GL_AMBIENT,mat_ambient);
    glMaterialfv(GL_FRONT,GL_DIFFUSE,mat_diffuse);
    glMaterialfv(GL_FRONT,GL_SHININESS,mat_shininess);
    glMaterialfv(GL_FRONT,GL_SPECULAR,mat_specular);
    glLightfv(GL_LIGHT0,GL_POSITION,Light_position);
    glLightfv(GL_LIGHT0,GL_DIFFUSE,Light_intensity);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-2.0,2.0,-2.0,2.0,-10.0,10.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(2.0,1.0,2.0,0.0,0.2,0.2,0.0,1.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
}

void draw_text(float x,float y,float z,char *s)
{

    int i;
    glRasterPos3f(x,y,z);
    for(i=0;s[i]!='\0';i++)
```

```
        {
            glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12,s[i]);
        }

    }

    void teapot()
    {
        glPushMatrix();
        glTranslated(0.4,0.0,0.4);
        glRotated(30,0,1,0);
        glutSolidTeapot(0.2);
        glPopMatrix();
    }

    void tabletop()
    {
        glPushMatrix();
        glTranslated(0.0,-0.3,0.0);
        glScaled(7.0,0.5,7.0);
        glutSolidCube(0.2);
        glPopMatrix();
    }

    void frontleg()
    {
        glPushMatrix();
        glTranslated(0.5,-0.7,0.5);
        glScaled(0.5,7.0,0.5);
        glutSolidCube(0.1);
        glPopMatrix();
    }

    void leftleg()
    {
        glPushMatrix();
        glTranslated(-0.5,-0.7,0.5);
        glScaled(0.5,7.0,0.5);
        glutSolidCube(0.1);
        glPopMatrix();
    }

    void rightleg()
    {
        glPushMatrix();
        glTranslated(0.5,-0.7,-0.5);
        glScaled(0.5,7.0,0.5);
        glutSolidCube(0.1);
        glPopMatrix();
    }

    void backleg()
    {
        glPushMatrix();
        glTranslated(-0.5,-0.7,-0.5);
        glScaled(0.5,7.0,0.5);
        glutSolidCube(0.1);
        glPopMatrix();
    }

    void leftwall()
    {
        glPushMatrix();
        glTranslated(-1.0,0.0,0.0);
        glScaled(0.1,10.0,10.0);
        glutSolidCube(0.2);
        glPopMatrix();
    }

    void bottomfloor()
    {
        glPushMatrix();
        glTranslated(0.0,-1.0,0.0);
        glScaled(10.0,0.1,10.0);
        glutSolidCube(0.2);
        glPopMatrix();
    }

    void rightwall()
    {
        glPushMatrix();
        glTranslated(0.0,0.0,-1.0);
        glScaled(10.0,10.0,0.1);
        glutSolidCube(0.2);
        glPopMatrix();
    }

    void display()
    {
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        glClearColor(1.0,1.0,1.0,1.0);
        draw_text(-0.2,1.55,-0.5,"Shaded Scene");
        draw_text(-0.2,1.65,-0.5," USN:1BI15CS066 , NAME:K K NITHIN");
        glColor3f(0.0,0.7490,1.0);
        teapot();
        glColor3f(0.9333,0.3607,0.2588);
        tabletop();
        glColor3f(0.0,0.0,1.0);
        frontleg();
        glColor3f(0.0,1.0,0.0);
        leftleg();
        glColor3f(1.0,0.0,0.0);
        rightleg();
        glColor3f(1.0,0.2039,0.7019);
        backleg();
        glColor3f(0.3,0.3,0.3);
        bottomfloor();
        glColor3f(0.8039,0.7764,0.4509);
        rightwall();
        glColor3f(0.8039,0.7764,0.4509);
        leftwall();
        glFlush();
    }

    void main(int argc,char **argv)
    {
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(50,50);
        glutCreateWindow("Shaded Scene");
        init();
        glutDisplayFunc(display);
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glShadeModel(GL_SMOOTH);
        glEnable(GL_COLOR_MATERIAL);
        glEnable(GL_DEPTH_TEST);
        glEnable(GL_NORMALIZE);
        glClearColor(1.0,1.0,1.0,0.0);
        glViewport(0,0,640,480);
        glutMainLoop();

    }
```

7. Design, develop and implement recursively subdivide a tetrahedron to form 3D sierpinski gasket. The number of recursive steps is to be specified by the user.

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

```
typedef GLfloat point[3];
point v[]={{-1.0,-0.5,0.0},{1.0,-0.5,0.0},{0.0,1.0,0.0},{0.0,0.0,1.0}};
GLfloat
colors[4][3]={{1.0,1.0,0.0},{0.0,1.0,0.0},{0.0,0.0,1.0},{1.0,0.0,0.0}};
int n;

void drawText(float x, float y,char* s){
        int i;
        glRasterPos2f(x,y);
        for(i=0;s[i] != '\0';i++)
                glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,s[i]);
}


void triangle(point a,point b,point c)
{
        glBegin(GL_POLYGON);
        glVertex3fv(a);
        glVertex3fv(b);
        glVertex3fv(c);
        glEnd();
}

void tetra(point a,point b,point c,point d)
{
        glColor3fv(colors[0]);
        triangle(a,b,c);
        glColor3fv(colors[1]);
        triangle(a,c,d);
        glColor3fv(colors[2]);
        triangle(a,d,b);
        glColor3fv(colors[3]);
        triangle(b,d,c);
}

void divide_tetra(point a,point b,point c,point d,int m)
{
        point mid[6];
        int j;
        if(m>0)
        {
                for(j=0;j<3;j++)
                {
                        mid[0][j]=(a[j]+b[j])/2.0;
                        mid[1][j]=(a[j]+c[j])/2.0;
                        mid[2][j]=(a[j]+d[j])/2.0;
                        mid[3][j]=(b[j]+c[j])/2.0;
                        mid[4][j]=(c[j]+d[j])/2.0;
                        mid[5][j]=(b[j]+d[j])/2.0;
                }
                divide_tetra(a,mid[0],mid[1],mid[2],m-1);
                divide_tetra(mid[0],b,mid[3],mid[5],m-1);
                divide_tetra(mid[1],mid[3],c,mid[4],m-1);
                divide_tetra(mid[2],mid[5],mid[4],d,m-1);
        }
        else
        tetra(a,b,c,d);
}

void display()
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glClearColor(0.0,0.0,0.0,0.0);
        divide_tetra(v[0],v[1],v[2],v[3],n);
    glColor3f(1.0,0.0,0.0);
    drawText(0.30,0.80,"3D Sierpinski Gasket");
        drawText(0.30,0.70,"Number of Divisions = 3");
        drawText(0.30,0.60,"NAME : PRAJWAL H P    USN : 1BI15CS116
BATCH : C-2");
        glFlush();
}

void myReshape(int w,int h)
{
        glViewport(0,0,w,h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        if(w<=h)
                glOrtho(-1.0,1.0,-
1.0*((GLfloat)h/(GLfloat)w),1.0*((GLfloat)h/(GLfloat)w),-1.0,1.0);
        else
                glOrtho(-
1.0*((GLfloat)w/(GLfloat)h),1.0*((GLfloat)w/(GLfloat)h),-1.0,1.0,-1.0,1.0);
                glMatrixMode(GL_MODELVIEW);
                glutPostRedisplay();
}

void main(int argc,char **argv)
{
        printf("Number of division:");
        scanf("%d",&n);
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(500,500);
        glutCreateWindow("3D gasket");
        glutDisplayFunc(display);
        glutReshapeFunc(myReshape);
        glEnable(GL_DEPTH_TEST);
        glutMainLoop();
}
```

8. Develop a menu driven program to animate a flag using Bezier Curve algorithm.

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<GL/glut.h>
#define PI 3.1416
GLsizei winWidth=600,winHeight=600;
GLfloat xwcMin=0.0,xwcMax=130.0;
GLfloat ywcMin=0.0,ywcMax=130.0;
int size,submenu;

GLint nCtrlPts=4,nBezCurvePts=20;
static float theta=0;

void draw_text(float x,float y,char* s)
{
     int i;
     glRasterPos2f(x,y);
     for(i=0;s[i]!='\0';i++)
     glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,s[i]);
}


struct wcPt3D
{
     GLfloat x;
     GLfloat y;
     GLfloat z;
};

wcPt3D ctrlPts[4]={{20,100,0},{30,110,0},{50,90,0},{60,100,0}};

typedef struct wcPt3D cp;

void bino(GLint n,GLint *C)
{
     GLint k,j;
     for(k=0;k<=n;k++)
     {
          C[k]=1;
          for(j=n;j>=k+1;j--)
               C[k]*=j;
          for(j=n-k;j>=2;j--)
               C[k]/=j;
     }
}

void computeBezPt(GLfloat u,cp *bezPt,GLint nCtrlPts,cp *ctrlPts,GLint *C)
{
     GLint k,n=nCtrlPts-1;
     GLfloat bezBlendFcn;
     bezPt ->x =bezPt ->y= bezPt->z=0.0;
     for(k=0;k<nCtrlPts;k++)
     {
          bezBlendFcn=C[k]*pow(u,k)*pow(1-u,n-k);
          bezPt ->x+= ctrlPts[k].x* bezBlendFcn;
          bezPt ->y+= ctrlPts[k].y* bezBlendFcn;
          bezPt ->z+= ctrlPts[k].z* bezBlendFcn;
     }
}

void bezier(cp *ctrlPts,GLint nCtrlPts,GLint nBezCurvePts)
{
     cp bezCurvePt;
     GLfloat u;
     GLint *C,k;
     C=new GLint[nCtrlPts];
     bino(nCtrlPts-1,C);
     glBegin(GL_LINE_STRIP);
          for(k=0;k<=nBezCurvePts;k++)
          {
               u=GLfloat(k)/GLfloat(nBezCurvePts);
               computeBezPt(u,&bezCurvePt,nCtrlPts,ctrlPts,C);
               glVertex2f(bezCurvePt.x,bezCurvePt.y);
          }
     glEnd();
     delete[] C;
}

void displayFcn()
{
     int i;
     glClear(GL_COLOR_BUFFER_BIT);
     glColor3f(1.0,1.0,1.0);
     glPointSize(5);
     glPushMatrix();
     glLineWidth(5);
     glColor3f(255/255.0,153/255.0,51/255.0);
```

```
	for(i=0;i<8;i++)
	{
		glTranslatef(0,-0.8,0);
		bezier(ctrlPts,nCtrlPts,nBezCurvePts);
	}
	glColor3f(1.0,1.0,1.0);
	for(i=0;i<8;i++)
	{
		glTranslatef(0,-0.8,0);
		bezier(ctrlPts,nCtrlPts,nBezCurvePts);
	}
	glColor3f(19/255.0,136/255.0,8/255.0);
	for(i=0;i<8;i++)
	{
		glTranslatef(0,-0.8,0);
		bezier(ctrlPts,nCtrlPts,nBezCurvePts);
	}
	glPopMatrix();
	glColor3f(0.7,0.5,0.3);
	glLineWidth(5);
	glBegin(GL_LINES);
		glVertex2f(20.0,100.0);
		glVertex2f(20.0,40.0);
	glEnd();
	draw_text(50,450,"USN:1BI15CS116, NAME:PRAJWAL H P, BATCH:C-2");
	draw_text(50,470,"Rotation about Origin, Degree=90");
	glFlush();
	glutPostRedisplay();
	glutSwapBuffers();
}

void menufunc(int n)
{
	switch(n)
	{
	case 1 :
		ctrlPts[1].x+=10*sin(theta*PI/180.0);
		ctrlPts[1].y+=5*sin(theta*PI/180.0);
		ctrlPts[2].x-=10*sin((theta+30)*PI/180.0);
		ctrlPts[2].y-=10*sin((theta+30)*PI/180.0);
		ctrlPts[3].x-=4*sin((theta)*PI/180.0);
		ctrlPts[3].y-=sin((theta-30)*PI/180.0);
		theta+=0.1;
		break;

	case 2 :
		ctrlPts[1].x-=10*sin(theta*PI/180.0);
		ctrlPts[1].y-=5*sin(theta*PI/180.0);
```

```
            ctrlPts[2].x+=10*sin((theta+30)*PI/180.0);
            ctrlPts[2].y+=10*sin((theta+30)*PI/180.0);
            ctrlPts[3].x+=4*sin((theta)*PI/180.0);
            ctrlPts[3].y-=sin((theta-30)*PI/180.0);
            theta+=0.1;
            break;

      case 3 : exit(0);
      }
}

void winReshapeFun(GLint newWidth,GLint newHeight)
{
      glViewport(0,0,newWidth,newHeight);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(xwcMin,xwcMax,ywcMin,ywcMax);
      glClear(GL_COLOR_BUFFER_BIT);
}

int main(int argc,char **argv)
{
      glutInit(&argc,argv);
      glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
      glutInitWindowPosition(50,50);
      glutInitWindowSize(winWidth,winHeight);
      glutCreateWindow("Bezier Curve");
      submenu=glutCreateMenu(menufunc);
      glutCreateMenu(menufunc);
      glutAddMenuEntry("Dwn-Mov",1);
      glutAddMenuEntry("Up-Mov",2);
      glutAddMenuEntry("Exit",3);
      glutAttachMenu(GLUT_RIGHT_BUTTON);
      glutDisplayFunc(displayFcn);
            glutReshapeFunc(winReshapeFun);
      glutMainLoop();
      return 0;
}
```

9. Develop a menu driven program to fill the polygon using scan line algorithm.

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

float et[4][4]={{100,250,200,-
1},{100,250,200,1},{200,150,300,1},{200,350,300,-1}};
```

```
int np=4;
float ae[4][3];
float js;
int iaet=0;
int ymax=0;
static int window;
static int menu_id;
static int submenu_id;
static int value=0;


void addaet()
{
     int i;
     for(i=0;i<np;i++)
     {
           printf("Scan line=%f & iate=%d\n",js,iaet);
           if(js==et[i][0])
           {
            ae[iaet][0]=et[i][1];
            ae[iaet][1]=et[i][2];
            ae[iaet][2]=et[i][3];
            if(ae[iaet][1]>ymax)
                ymax=ae[iaet][1];
            iaet++;
           }
     }
}

void upaet()
{
     int i;
     for(i=0;i<np;i++)
           ae[i][0]=ae[i][0]+ae[i][2];
}

void draw_pixel(float x1,float x2)
{
     int i;
     float n;

     for(n=x1;n<=x2;n++)
     {
           glBegin(GL_POINTS);
           glVertex2f(n,js);
           glEnd();
     }
           glFlush();
     printf("x1=%f x2=%f\n",x1,x2);
}

void drawText(float x, float y,float z,char* s){
        int i;
        glRasterPos2f(x,y);
        for(i=0;s[i] != '\0';i++)
                glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,s[i]);
}

void fill_poly()
{
     float x[3]={0,0,0};
     int i=0,j;
     do
     {
          i=0;
          addaet();
          printf("1=%f 2=%f %f\n",ae[0][1],ae[1][1],js);
          for(j=0;j<np;j++)
               if(ae[j][1]>js)
               {
                    x[i]=ae[j][0];
                    i++;
               }
          draw_pixel(x[0],x[1]);
          upaet();
          js++;
     }while(js <= ymax);
}

void empty_ae()
{
     js=et[0][0];
     iaet=0;
}

void display()
{
     int i,j;
     glClear(GL_COLOR_BUFFER_BIT);
     glColor3f(0.0,1.0,0.0);
     drawText(200,50,0.5,"Scanline Filling Algorithm");
        drawText(200,80,0.6,"NAME : K K NITHIN    USN : 1BI15CS066
BATCH : B-2");


     switch(value)
```

```
    {
        case 1 : return;
            break;
        case 2 : glColor3f(1.0,0.0,0.0);
            fill_poly();
            empty_ae();
            break;
        case 3 : glColor3f(0.0,1.0,0.0);
            fill_poly();
            empty_ae();
            break;
        case 4 : glColor3f(0.0,0.0,1.0);
            fill_poly();
            empty_ae();
            break;
    }
}

void myinit()
{
    glClearColor(1.0,1.0,1.0,1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0,1000.0,0.0,1000.0);
}

void menu(int num)
{
    if(num==0)
    {
        glutDestroyWindow(window);
        exit(0);
    }
    else  value=num;
}

void createMenu(void)
{
    submenu_id=glutCreateMenu(menu);
    glutAddMenuEntry("Red",2);
    glutAddMenuEntry("Green",3);
    glutAddMenuEntry("Blue",4);
    menu_id=glutCreateMenu(menu);
    glutAddMenuEntry("Clear",1);
    glutAddSubMenu("Color",submenu_id);
    glutAddMenuEntry("Quit",0);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
}
```

```
int main(int argc,char ** argv)
{
     js=et[0][0];
     glutInit(&argc,argv);
     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
     glutInitWindowSize(500,500);
     glutInitWindowPosition(0,0);
     glutCreateWindow("Polygon Filling Algorithm");
     createMenu();
     glutDisplayFunc(display);
     myinit();
     glutMainLoop();
     return 0;
}
```