

PROG-1

```
#include"stdafx.h"
#include<glut.h>
#include<stdio.h>
int x1, y1, x2,y2;
void draw_pixel(int x,int y)
{
glColor3f(1.0,0.0,0.0);
glBegin(GL_POINTS);
glVertex2i(x,y);
glEnd();
}
void brenhams_line_draw(int x1, int y1, int x2, int y2)
{
int dx=x2-x1,dy=y2-y1;
int p=2*dy*dx;
int twoDy=2*dy;
int twoDyMinusDx=2*(dy-dx);
int x=x1,y=y1;
if(dx<0)
{
x=x2;
y=y2;
x2=x1;
}
draw_pixel(x,y);
while(x<x2)
{
x++;
if(p<0)
p+=twoDy;
else
{
}
y++;
p+=twoDyMinusDx;
draw_pixel(x,y);
}
}
void myInit()
{
glClearColor(0.0,0.0,0.0,1.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,500.0,0.0,500.0);
glMatrixMode(GL_MODELVIEW);
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
brenhams_line_draw(x1,x2,y1,y2);
glFlush();
}
void main(int argc,char ** argv)
{
printf("enter(x1,y1,x2,y2)\n");
scanf_s("%d %d %d %d",&x1,&y1,&x2,&y2);
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
```

```

glutCreateWindow("'.....ABC.....Bresenham's line drawing");
myInit();
glutDisplayFunc(display);
glutMainLoop();
}

```

PROG-2

```

#include "stdafx.h"
#include <glut.h>
#include <math.h>
GLsizei winWidth = 600, winHeight=600;
GLfloat xwcMin = 0.0, xwcMax = 225.0;
GLfloat ywcMin = 0.0, ywcMax = 225.0;
class wcPt2D {
public:GLfloat x, y; };
typedef GLfloat Matrix3x3[3][3];
Matrix3x3 matComposite;
const GLdouble pi =3.14159;
void init (void)
{
glClearColor (1.0, 1.0, 1.0, 0.0);
}
void matrix3x3SetIdentity (Matrix3x3 matIdent3x3)
{
GLint row, col;

for (row = 0; row < 3; row++)
    for (col = 0; col < 3; col++)
        matIdent3x3 [row][col] = (row == col);
}
void matrix3x3PreMultiply (Matrix3x3 m1, Matrix3x3 m2)
{
GLint row, col;
Matrix3x3 matTemp;
for (row = 0; row < 3; row++)
    for (col = 0; col < 3 ; col++)
        matTemp [row][col] = m1 [row][0] * m2 [0][col] + m1 [row][1] * m2 [1][col] + m1
[row][2] * m2 [2][col];

for (row = 0; row < 3; row++)
    for (col = 0; col < 3; col++)
        m2 [row][col] = matTemp [row][col];
}
void translate2D (GLfloat tx, GLfloat ty)
{
Matrix3x3 matTransl;
matrix3x3SetIdentity (matTransl);
matTransl [0][2] = tx;
matTransl [1][2] = ty;
matrix3x3PreMultiply (matTransl, matComposite);
}
void rotate2D (wcPt2D pivotPt, GLfloat theta)
{
Matrix3x3 matRot;
matrix3x3SetIdentity (matRot);
matRot [0][0] = cos (theta);
matRot [0][1] = -sin (theta);
matRot [0][2] = pivotPt.x * (1 - cos (theta)) +

```

```

pivotPt.y * sin (theta);
matRot [1][0] = sin (theta);
matRot [1][1] = cos (theta);

matRot [1][2] = pivotPt.y * (1 - cos (theta)) - pivotPt.x * sin(theta);
matrix3x3PreMultiply (matRot, matComposite);
}
void scale2D (GLfloat sx, GLfloat sy, wcPt2D fixedPt)
{
Matrix3x3 matScale;
matrix3x3SetIdentity(matScale);
matScale [0][0] = sx;
matScale [0][2] = (1 - sx) * fixedPt.x;
matScale [1][1] = sy;
matScale [1][2] = (1 - sy) * fixedPt.y;
matrix3x3PreMultiply(matScale, matComposite);
}
void transformVerts2D(GLint nVerts, wcPt2D *verts)
{
GLint k;
GLfloat temp;
for (k = 0; k < nVerts; k++)
{
temp = matComposite [0][0] * verts [k].x + matComposite [0][1] * verts [k].y + matComposite
[0][2];
verts [k].y = matComposite [1][0] * verts [k].x + matComposite [1][1] * verts [k].y +
matComposite [1][2];
verts [k].x = temp;
}
}
void triangle(wcPt2D *verts)
{
GLint k;
glBegin (GL_TRIANGLES);
for (k = 0; k < 3; k++)
glVertex2f (verts [k].x, verts [k].y);
glEnd ( );
}
void displayFcn (void)
{
GLint nVerts = 3;
wcPt2D verts [3] = { {50.0, 25.0}, {150.0, 25.0}, {100.0, 100.0} };
wcPt2D centroidPt;
GLint k, xSum = 0, ySum = 0;

for (k = 0; k < nVerts; k++)
{
xSum += verts [k].x;
ySum += verts [k].y;
}
centroidPt.x = GLfloat (xSum) / GLfloat (nVerts);
centroidPt.y = GLfloat (ySum) / GLfloat (nVerts);
wcPt2D pivPt, fixedPt;
pivPt = centroidPt;
fixedPt = centroidPt;
GLfloat tx = 0.0, ty = 100.0;
GLfloat sx = 0.5, sy = 0.5;
GLdouble theta = pi/2.0;
glClear (GL_COLOR_BUFFER_BIT);
glColor3f (0.0, 0.0, 1.0);
triangle(verts);
matrix3x3SetIdentity(matComposite);
scale2D (sx, sy, fixedPt);
rotate2D (pivPt, theta);
translate2D (tx, ty);
transformVerts2D(nVerts, verts);
glColor3f (1.0, 0.0, 0.0);
triangle(verts);

```

```

glFlush ( );
}
void winReshapeFcn (GLint newWidth, GLint 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_SINGLE |
GLUT_RGB); glutInitWindowPosition (50, 50);
glutInitWindowSize (winWidth, winHeight);
glutCreateWindow ("Geometric Transformation Sequence");
init ( );
glutDisplayFunc (displayFcn);
glutReshapeFunc (winReshapeFcn);
glutMainLoop ( );
}

```

PROG-3

```

#include "stdafx.h"
#include<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(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] += 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();
}

```

PROG-4

```

#include "stdafx.h"
#include <stdlib.h>
#include <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;
display();
}
void keys(unsigned char key, int x, int y)
{
if(key == 'x')
    viewer[0]-= 1.0;
if(key == 'X')
    viewer[0]+=1.0;
if(key == 'y')
    viewer[1]-=1.0;
if(key == 'Y')
    viewer[1]+= 1.0;
if(key == 'z')
    viewer[2]-= 1.0;
if(key == 'Z')
    viewer[2]+= 1.0;
display();
}
void myReshape(int w, int h)
{
glViewport(0, 0, w, h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
if(w<=h)
glFrustum(-2.0, 2.0, -2.0 * (GLfloat) h/ (GLfloat) w, 2.0* (GLfloat) h / (GLfloat) w, 2.0, 20.0);

```

```

else
glFrustum(-2.0, 2.0, -2.0 * (GLfloat) w/ (GLfloat) h, 2.0* (GLfloat) w / (GLfloat) h, 2.0, 20.0);
glMatrixMode(GL_MODELVIEW);
}
void main(int argc, char **argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowSize(500, 500);
glutCreateWindow("Colorcube Perspective viewing");
glutReshapeFunc(myReshape);
glutDisplayFunc(display);
glutMouseFunc(mouse);
glutKeyboardFunc(keys);
glEnable(GL_DEPTH_TEST);
glutMainLoop();
}

```

PROG-5

```

#include"stdafx.h"
#include<stdio.h>
#include<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;
            }
        }
    } while (!done);
    if(accept)
        printf("Accepted\n");
    else
        printf("Rejected\n");
}

```

```

    }
    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_s("%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();
}

```

PROG-6

```

#include "stdafx.h"
#include <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[] = {1.0f, 0.0f, 0.0f, 1.0f};
GLfloat mat_diffuse[] = {.5f, .5f, .5f, 1.0f};
GLfloat mat_specular[] = {1.0f, 1.0f, 1.0f, 1.0f};
GLfloat mat_shininess[] = {75.0f};
glMaterialfv (GL_FRONT, GL_AMBIENT, mat_ambient);
glMaterialfv (GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv (GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv (GL_FRONT, GL_SHININESS, mat_shininess);
GLfloat lightIntensity[] = {0.7f, 0.7f, 0.7f, 1.0f}; GLfloat
light_position[] = {2.0f, 6.0f, 3.0f, 0.0f}; glLightfv
(GL_LIGHT0, GL_POSITION, light_position); glLightfv
(GL_LIGHT0, GL_DIFFUSE, lightIntensity);
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 (0.6, 0.38, 0.5);
glRotated (30, 0, 1, 0);
glutSolidTeapot (0.08);
glPopMatrix ();
glPushMatrix();
glTranslated (0.4, 0, 0.4);
table (0.6, 0.02, 0.02, 0.3);
glPopMatrix();
wall (0.02);
glPushMatrix();
glRotated (90.0, 0.0, 0.0, 1.0);
wall (0.02);
glPopMatrix();
glPushMatrix();
glRotated (-90.0, 1.0, 0.0, 0.0);
wall (0.02);
glPopMatrix();
glFlush();
}
void main (int argc, char ** argv)
{
glutInit (&argc, argv);
glutInitDisplayMode (GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize (640, 480);
glutInitWindowPosition (100, 100);
glutCreateWindow ("simple shaded scene consisting of a tea pot on a table");
glutDisplayFunc (displaySolid);
glEnable (GL_LIGHTING);
glEnable (GL_LIGHT0);
glShadeModel (GL_SMOOTH);
glEnable (GL_DEPTH_TEST);
glEnable (GL_NORMALIZE);
glClearColor (0.1, 0.1, 0.1, 0.0);
glViewport (0, 0, 640, 480);
glutMainLoop();
}

```

PROG-7

```
#include "stdafx.h"
#include <stdlib.h>
#include <stdio.h>
#include<glut.h>
typedef float point[3];
point v[]={0.0, 0.0, 1.0},{0.0, 0.9, -0.3},{-0.8, -0.5, -0.3},{0.8, -0.5, -0.3}};
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)
{
    int j;
    point v1,v2,v3;
    if(m>0)
    {
        for(j=0; j<3; j++)
            v1[j]=(a[j]+b[j])/2;
        for(j=0; j<3; j++)
            v2[j]=(a[j]+c[j])/2;
        for(j=0; j<3; j++)
            v3[j]=(b[j]+c[j])/2;
        divide_triangle(a, v1, v2, m-1);
        divide_triangle(c, v2, v3, m-1);
        divide_triangle(b, v3, v1, m-1);
    }
    else{
        triangle(a,b,c);
    }
}
void tetrahedron( int m)
{
    glColor3f(1.0,0.0,0.0);
    divide_triangle(v[0], v[1], v[2], m);
    glColor3f(0.0,0.0,0.0);
    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(0.0,1.0,0.0);
    divide_triangle(v[0], v[2], v[3], m);
}
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    tetrahedron(n);
    glFlush();
}

void myReshape(int w, int h)
{
    glClearColor (1.0, 1.0, 1.0, 1.0);
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
```

```

        glLoadIdentity();
        if (w <= h)
        {
            glOrtho(- 2.0, 2.0, -2.0 * (GLfloat) h / (GLfloat) w, 2.0 * (GLfloat) h / (GLfloat) w,
-10.0, 10.0);
        }
        else
        {
            glOrtho(-2.0 * (GLfloat) w / (GLfloat) h, 2.0 * (GLfloat) w / (GLfloat) h, -2.0, 2.0, -
10.0, 10.0);
        }
        glMatrixMode(GL_MODELVIEW);
        glutPostRedisplay();
    }
    int main(int argc, char **argv)
    {
        printf(" No. of Divisions ? ");
        scanf_s("%d",&n);
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(500, 500);
        glutCreateWindow("3D Gasket by joydev");
        glutReshapeFunc(myReshape);
        glutDisplayFunc(display);
        glEnable(GL_DEPTH_TEST);
        glutMainLoop();
        return 1;
    }
}

```

PROG-8

```

#include "stdafx.h"
#include<glut.h>
#include<stdio.h>
#include<math.h>
#define PI 3.1416
typedef struct point
{
    GLfloat x, y, z;
};
void bino(int n, int *C)
{
    int 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(float u, point *pt1, int cPt, point *pt2, int *C)
{
    int k, n=cPt-1;
    float bFcn;
}

```

```

pt1 ->x =pt1 ->y = pt1->z=0.0;
for(k=0; k< cPt; k++)
{
    bFcn = C[k] * pow(u, k) * pow( 1-u, n-k);
    pt1 ->x += pt2[k].x * bFcn;
    pt1 ->y += pt2[k].y * bFcn;
    pt1 ->z += pt2[k].z * bFcn;
}
}

void bezier(point *pt1, int cPt, int bPt)
{
    point bcPt;
    float u;
    int *C, k;
    C= new int[cPt];
    bino(cPt-1, C);
    glBegin(GL_LINE_STRIP);
    for(k=0; k<=bPt; k++)
    {
        u=float(k)/float(bPt);
        computeBezPt(u, &bcPt, cPt, pt1, C);
        glVertex2f(bcPt.x, bcPt.y);
    }
    glEnd();
    delete[]C;
}

float theta = 0;
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    int nCtrlPts = 4, nBCPts =20;
    point ctrlPts[4] = {{100, 400, 0}, {150, 450, 0}, {250, 350, 0},
    {300, 400, 0}};
    ctrlPts[1].x +=50*sin(theta * PI/180.0);
    ctrlPts[1].y +=25*sin(theta * PI/180.0);
    ctrlPts[2].x -= 50*sin((theta+30) * PI/180.0);
    ctrlPts[2].y -= 50*sin((theta+30) * PI/180.0);
    ctrlPts[3].x -= 25*sin((theta) * PI/180.0);
    ctrlPts[3].y += sin((theta-30) * PI/180.0);
    theta+=0.2;
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glPointSize(5);
    glPushMatrix();
    glLineWidth(5);
    glColor3f(1, 0.4, 0.2); //Indian flag: Orange color code
    for(int i=0;i<50;i++)
    {
        glTranslatef(0, -0.8, 0);
        bezier(ctrlPts, nCtrlPts, nBCPts);
    }
    glColor3f(1, 1, 1); //Indian flag: white color code
    for(int i=0;i<50;i++)
    {
        glTranslatef(0, -0.8, 0);
        bezier(ctrlPts, nCtrlPts, nBCPts);
    }
    glColor3f(0, 1, 0); //Indian flag: green color code
    for(int i=0;i<50;i++)

```

```

{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBCPts);
}
glPopMatrix();
glColor3f(0.7, 0.5, 0.3);
glLineWidth(5);
glBegin(GL_LINES);
glVertex2f(100, 400);
glVertex2f(100, 40);
glEnd();
glutPostRedisplay();
glutSwapBuffers();
}
void init()
{
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0, 500, 0, 500);
}

void main(int argc, char **argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
glutInitWindowPosition(0, 0);
glutInitWindowSize(500, 500);
glutCreateWindow("Bezier Curve");
init();
glutDisplayFunc(display);
glutMainLoop();
}

```

PROG-9

```

#include "stdafx.h"
#include <stdlib.h>
#include <stdio.h>
#include <glut.h>
float x1, x2, x3, x4, y1, y2, y3, y4;
int fillFlag=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)
{
glColor3f(1.0,1.0,0.0);
glBegin(GL_POINTS);
glVertex2i(x,y);
glEnd();
}
void scanfill(float x1,float y1,float x2,float y2,float x3,float y3,float x4,float y4)
{
int le[500],re[500];
int i,y;
for(i=0;i<500;i++)
{
le[i]=500;
re[i]=0;
}
edgedetect(x1,y1,x2,y2,le,re);
edgedetect(x2,y2,x3,y3,le,re);
edgedetect(x3,y3,x4,y4,le,re);
edgedetect(x4,y4,x1,y1,le,re);
for(y=0;y<500;y++)
{
for(i=(int)le[y];i<(int)re[y];i++)
draw_pixel(i,y);
}
}

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();
if(fillFlag==1)
scanfill(x1,y1,x2,y2,x3,y3,x4,y4);
glFlush();
}
void init()
{
glClearColor(0.0,0.0,0.0,1.0);
glColor3f(1.0,0.0,0.0);
glPointSize(1.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,499.0,0.0,499.0);
}
void fillMenu(int option)
{
if(option==1)
fillFlag=1;
if(option==2)

```

```
fillFlag=2;
display();
}
void main(int argc, char* argv[])
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("Filling a Polygon using Scan-line Algorithm");
init();
glutDisplayFunc(display);
glutCreateMenu(fillMenu);
glutAddMenuEntry("Fill Polygon",1);
glutAddMenuEntry("Empty Polygon",2);
glutAttachMenu(GLUT_RIGHT_BUTTON);
glutMainLoop();
}
```