/* Bresenhams Line Drawing Algorithm */

```
#include <GL/glut.h>
#include <math.h>
#include <stdio.h>
int x00, y00, xEnd, yEnd;

void display()
{
    int i, j;
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    void line(x00, y00, xEnd, yEnd);
    glFlush();
}

void drawPixel(int x, int y)
{
    glPointSize(3.0);
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
}

void line(int x00, int y00, int xEnd, int yEnd)
{
    int dx = fabs(xEnd - x00), dy = fabs(yEnd - y00);
    int p = 2 * dy - dx;
    int twoDy = 2 * dy - dx, twoDyMinusDx = 2 * (dy - dx);
    int x, y;

    /*Determine which endpoint to use as starting position*/

    if (x00 > xEnd) {
        x = xEnd;
        y = yEnd;
        xEnd = x00;
    } else {
        x = x00;
        y = y00;
    }

    drawPixel(x, y);

    while (x < xEnd) {
        x++;
        if (p < 0)
            p += twoDy;
        else {
            y++;
            p += twoDyMinusDx;
        }

        drawPixel(x, y);
    }
}

void myinit()
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glPointSize(2.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 950.0, 0.0, 950.0);
}

void main(int argc, char** argv)
{
    glutInit(&argc, argv);
```

```
68      printf("Enter two end points of the line");
69      scanf("%d %d %d %d", &x00, &y00, &xEnd, &yEnd);
70
71      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
72      glutInitWindowSize(500, 500);
73      glutInitWindowPosition(10, 10);
74      glutCreateWindow("BRESENHAM'S LINE DRAWING ALGORITHM");
75      glutDisplayFunc(display);
76      myinit();
77      glutMainLoop();
78  }
```

/* Triangle rotation */

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

GLfloat tri[3][3] = { { 100, 100, 0 }, { 200, 100, 0 }, { 150, 200, 0 }
    };
GLfloat arb_x = 100;
GLfloat arb_y = 100;
GLfloat rot_angle;

void drawtri()
{

    glBegin(GL_LINE_LOOP);
    glVertex3fv(tri[0]);
    glVertex3fv(tri[1]);
    glVertex3fv(tri[2]);
    glEnd();
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1, 0, 0);
    drawtri();
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(arb_x, arb_y, 0.0);
    glRotatef(rot_angle, 0.0, 0.0, 1.0);
    glTranslatef(-arb_x, -arb_y, 0.0);
    glColor3f(0, 1, 0);
    drawtri();
    glTranslatef(0.0, 0.0, 0.0);
    glRotatef(rot_angle, 0.0, 0.0, 1.0);
    glTranslatef(-0.0, -0.0, 0.0);
    glColor3f(0, 0, 1);
    drawtri();

    glFlush();
}
void myinit()
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glColor3f(1.0, 0.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-250.0, 499.0, -250.0, 499.0);
}

void main(int argc, char* argv[])
{
    printf("\nENTER THE ROTATION ANGLE :-\n");
    scanf("%f", &rot_angle);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("House Rotation");
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
}
```

/* Spin colored cube using transformation matrices */

```
#include <GL/glut.h>
#include <stdlib.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();
}
```

```
68
69 void mouse(int btn, int state, int x, int y)
70 {
71     if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
72         axis = 0;
73     if (btn == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN)
74         axis = 1;
75     if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
76         axis = 2;
77 }
78
79 void myReshape(int w, int h)
80 {
81     glViewport(0, 0, w, h);
82     glMatrixMode(GL_PROJECTION);
83     glLoadIdentity();
84     if (w <= h)
85         glOrtho(-2.0, 2.0, -2.0 * (GLfloat)h / (GLfloat)w, 2.0 * (
               GLfloat)h / (GLfloat)w, -10.0, 10.0);
86     else
87         glOrtho(-2.0 * (GLfloat)w / (GLfloat)h, 2.0 * (GLfloat)w / (
               GLfloat)h, -2.0, 2.0, -10.0, 10.0);
88     glMatrixMode(GL_MODELVIEW);
89 }
90
91 int main(int argc, char** argv)
92 {
93     glutInit(&argc, argv);
94     glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
95     glutInitWindowSize(500, 500);
96     glutCreateWindow("Rotating a Color Cube");
97     glutReshapeFunc(myReshape);
98     glutDisplayFunc(display);
99     glutIdleFunc(spinCube);
100    glutMouseFunc(mouse);
101    glEnable(GL_DEPTH_TEST);
102    glutMainLoop();
103    return 0;
104 }
```

/* Perspective viewing of a colored cube */

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

```
67      if (btn == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN)
68          axis = 1;
69      if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
70          axis = 2;
71      theta[axis] += 2.0;
72      if (theta[axis] > 360.0)
73          theta[axis] -= 360.0;
74      display();
75  }
76
77  void keys(unsigned char key, int x, int y)
78  {
79      if (key == 'x')
80          viewer[0] -= 1.0;
81      if (key == 'X')
82          viewer[0] += 1.0;
83      if (key == 'y')
84          viewer[1] -= 1.0;
85      if (key == 'Y')
86          viewer[1] += 1.0;
87      if (key == 'z')
88          viewer[2] -= 1.0;
89      if (key == 'Z')
90          viewer[2] += 1.0;
91      display();
92  }
93
94  void myReshape(int w, int h)
95  {
96      glViewport(0, 0, w, h);
97      glMatrixMode(GL_PROJECTION);
98      glLoadIdentity();
99      if (w <= h)
100         glFrustum(-2.0, 2.0, -2.0 * (GLfloat)h / (GLfloat)w, 2.0 * (
              GLfloat)h / (GLfloat)w, 2.0, 20.0);
101     else
102         glFrustum(-2.0, 2.0, -2.0 * (GLfloat)w / (GLfloat)h, 2.0 * (
              GLfloat)w / (GLfloat)h, 2.0, 20.0);
103     glMatrixMode(GL_MODELVIEW);
104 }
105
106 int main(int argc, char** argv)
107 {
108     glutInit(&argc, argv);
109     glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
110     glutInitWindowSize(500, 500);
111     glutCreateWindow("Colorcube Viewer");
112     glutReshapeFunc(myReshape);
113     glutDisplayFunc(display);
114     glutMouseFunc(mouse);
115     glutKeyboardFunc(keys);
116     glEnable(GL_DEPTH_TEST);
117     glutMainLoop();
118     return 0;
119 }
```

/* Program: Clip a lines using Cohen-Sutherland algorithm */

```
1
2  #include <GL/glut.h>
3  #include <stdio.h>
4  #define outcode int
5
6  double xmin = 50, ymin = 50, xmax = 100, ymax = 100;
7  double xvmin = 200, yvmin = 200, xvmax = 300, yvmax = 300;
8
9  float X0, Y0, X1, Y1;
10 const int RIGHT = 2;
11 const int LEFT = 1;
12 const int TOP = 8;
13 const int BOTTOM = 4;
14
15 outcode ComputeOutCode(double x, double y);
16
17 void CohenSutherlandLineClipAndDraw(double x0, double y0, double x1,
       double y1)
18 {
19     outcode outcode0, outcode1, outcodeOut;
20     bool accept = false, done = false;
21     outcode0 = ComputeOutCode(x0, y0);
22     outcode1 = ComputeOutCode(x1, y1);
23
24     do {
25         if (!(outcode0 | outcode1)) {
26             accept = true;
27             done = true;
28         } else if (outcode0 & outcode1)
29             done = true;
30         else {
31             double x, y, m;
32             m = (y1 - y0) / (x1 - x0);
33             outcodeOut = outcode0 ? outcode0 : outcode1;
34             if (outcodeOut & TOP) {
35                 x = x0 + (ymax - y0) / m;
36                 y = ymax;
37             } else if (outcodeOut & BOTTOM) {
38                 x = x0 + (ymin - y0) / m;
39                 y = ymin;
40             } else if (outcodeOut & RIGHT) {
41                 y = y0 + (xmax - x0) * m;
42                 x = xmax;
43             } else {
44                 y = y0 + (xmin - x0) * m;
45                 x = xmin;
46             }
47             if (outcodeOut == outcode0) {
48                 x0 = x;
49                 y0 = y;
50                 outcode0 = ComputeOutCode(x0, y0);
51             } else {
52                 x1 = x;
53                 y1 = y;
54                 outcode1 = ComputeOutCode(x1, y1);
55             }
56         }
57     } while (!done);
58     glColor3f(1.0, 0.0, 0.0);
59     glBegin(GL_LINE_LOOP);
60     glVertex2f(xvmin, yvmin);
61     glVertex2f(xvmax, yvmin);
62     glVertex2f(xvmax, yvmax);
63     glVertex2f(xvmin, yvmax);
64     glEnd();
65     printf("\n%f   %f :  %f   %f", x0, y0, x1, y1);
66
```

```
    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);
        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();
    CohenSutherlandLineClipAndDraw(X0, Y0, X1, Y1);
    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 main(int argc, char** argv)
{
    printf("Enter end points : ");
    scanf("%f%f%f%f", &X0, &Y0, &X1, &Y1);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Cohen Sutherland Line Clipping
            Algorithm");
    glutDisplayFunc(display);
```

```
136     myinit();
137     glutMainLoop();
138 }
```

/* Simple shaded scene with a teapot */

```
#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()
{
    GLfloat mat_ambient[] = { 0.7f, 0.7f, 0.7f, 1.0f };
    GLfloat mat_diffuse[] = { 0.5f, 0.5f, 0.5f, 1.0f };
    GLfloat mat_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
    GLfloat mat_shininess[] = { 50.0f };
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

    GLfloat 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;
```

```
67      glOrtho(-winHt * 64 / 48.0, winHt * 64 / 48.0, -winHt, winHt, 0.1,
            100.0);
68      glMatrixMode(GL_MODELVIEW);
69      glLoadIdentity();
70      gluLookAt(2.3, 1.3, 2.0, 0.0, 0.25, 0.0, 0.0, 1.0, 0.0);
71
72      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
73
74      glPushMatrix();
75      glTranslated(0.6, 0.38, 0.5);
76      glRotated(30, 0, 1, 0);
77      glutSolidTeapot(0.08);
78      glPopMatrix();
79
80      glPushMatrix();
81      glTranslated(0.4, 0, 0.4);
82      table(0.6, 0.02, 0.02, 0.3);
83      glPopMatrix();
84
85      wall(0.02);
86      glPushMatrix();
87      glRotated(90.0, 0.0, 0.0, 1.0);
88      wall(0.02);
89      glRotated(90.0, 1.0, 0.0, 0.0);
90      wall(0.02);
91      glPopMatrix();
92      glFlush();
93  }
94  void main(int argc, char** argv)
95  {
96      glutInit(&argc, argv);
97      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
98      glutInitWindowSize(640, 480);
99      glutInitWindowPosition(100, 100);
100     glutCreateWindow("Simple shaded scene of a teapot on a table");
101     glutDisplayFunc(displaySolid);
102     glEnable(GL_LIGHTING);
103     glEnable(GL_LIGHT0);
104     glShadeModel(GL_SMOOTH);
105     glEnable(GL_DEPTH_TEST);
106     glEnable(GL_NORMALIZE);
107     glClearColor(0.1, 0.1, 0.1, 0.0);
108     glViewport(0, 0, 640, 480);
109     glutMainLoop();
110 }
```

/* 3-D Sierpinski Gasket */

```
#include <GL/glut.h>
#include <stdio.h>
#include <stdlib.h>
typedef float point[3];
point v[] = { { 0.0, 0.0, 1.0 }, { 0.0, 1.0, 0.0 },
    { -1.0, -0.5, 0.0 }, { 1.0, -0.5, 0.0 } };
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;
            v2[j] = (a[j] + c[j]) / 2;
            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 Recursive steps/Division: ");
    scanf("%d", &n);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutCreateWindow(" 3D Sierpinski gasket");
    glutReshapeFunc(myReshape);
    glutDisplayFunc(display);
    glEnable(GL_DEPTH_TEST);
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glutMainLoop();
}
```

/* Bezier's Curve Flag */

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

#define PI 3.1416
GLsizei winWidth = 600, winHeight = 600;
GLfloat xwcMin = 0.0, xwcMax = 130.0;
GLfloat ywcMin = 0.0, ywcMax = 130.0;
typedef struct wcPt3D {
    GLfloat x, y, z;
};

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, wcPt3D* bezPt, GLint nCtrlPts, wcPt3D*
   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(wcPt3D* ctrlPts, GLint nCtrlPts, GLint nBezCurvePts)
{
    wcPt3D 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()
{
    GLint nCtrlPts = 4, nBezCurvePts = 20;

    static float theta = 0;

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

```
67
68      ctrlPts[1].x += 10 * sin(theta * PI / 180.0);
69      ctrlPts[1].y += 5 * sin(theta * PI / 180.0);
70      ctrlPts[2].x -= 10 * sin((theta + 30) * PI / 180.0);
71      ctrlPts[2].y -= 10 * sin((theta + 30) * PI / 180.0);
72      ctrlPts[3].x -= 4 * sin((theta)*PI / 180.0);
73      ctrlPts[3].y += sin((theta - 30) * PI / 180.0);
74      theta += 0.1;
75      glClear(GL_COLOR_BUFFER_BIT);
76      glColor3f(1.0, 1.0, 1.0);
77      glPointSize(5);
78      glPushMatrix();
79      glLineWidth(5);
80      glColor3f(255 / 255, 153 / 255.0, 51 / 255.0);
81      for (int i = 0; i < 8; i++) {
82          glTranslatef(0, -0.8, 0);
83          bezier(ctrlPts, nCtrlPts, nBezCurvePts);
84      }
85      glColor3f(1, 1, 1);
86      for (int i = 0; i < 8; i++) {
87          glTranslatef(0, -0.8, 0);
88          bezier(ctrlPts, nCtrlPts, nBezCurvePts);
89      }
90      glColor3f(19 / 255.0, 136 / 255.0, 8 / 255.0);
91      for (int i = 0; i < 8; i++) {
92          glTranslatef(0, -0.8, 0);
93          bezier(ctrlPts, nCtrlPts, nBezCurvePts);
94      }
95      glPopMatrix();
96      glColor3f(0.7, 0.5, 0.3);
97      glLineWidth(5);
98      glBegin(GL_LINES);
99      glVertex2f(20, 100);
100     glVertex2f(20, 40);
101     glEnd();
102     glFlush();
103     glutPostRedisplay();
104     glutSwapBuffers();
105 }
106
107 void winReshapeFun(GLint newWidth, GLint newHeight)
108 {
109     glViewport(0, 0, newWidth, newHeight);
110     glMatrixMode(GL_PROJECTION);
111     glLoadIdentity();
112     gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);
113     glClear(GL_COLOR_BUFFER_BIT);
114 }
115
116 int main(int argc, char** argv)
117 {
118     glutInit(&argc, argv);
119     glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
120     glutInitWindowPosition(50, 50);
121     glutInitWindowSize(winWidth, winHeight);
122     glutCreateWindow("Bezier Curve");
123     glutDisplayFunc(displayFcn);
124     glutReshapeFunc(winReshapeFun);
125     glutMainLoop();
126     return 0;
127 }
```

/* Scan-Line algorithm for filling a polygon */

```
#define BLACK 0
#include <GL/glut.h>
#include <stdio.h>
#include <stdlib.h>
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)
{
    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++) {
        if (le[y] <= re[y])

            for (i = (int)le[y]; i < (int)re[y]; i++)

                draw_pixel(i, y, BLACK);
    }
}

void display()
{
    x1 = 200.0;
    y1 = 200.0;
    x2 = 100.0;
```

```
66      y2 = 300.0;
67      x3 = 200.0;
68      y3 = 400.0;
69      x4 = 300.0;
70      y4 = 300.0;
71      glClear(GL_COLOR_BUFFER_BIT);
72      glColor3f(0.0, 0.0, 1.0);
73      glBegin(GL_LINE_LOOP);
74      glVertex2f(x1, y1);
75      glVertex2f(x2, y2);
76      glVertex2f(x3, y3);
77      glVertex2f(x4, y4);
78      glEnd();
79      scanfill(x1, y1, x2, y2, x3, y3, x4, y4);
80      glFlush();
81  }
82
83  void myinit()
84  {
85      glClearColor(1.0, 1.0, 1.0, 1.0);
86      glColor3f(1.0, 0.0, 0.0);
87      glPointSize(1.0);
88      glMatrixMode(GL_PROJECTION);
89      glLoadIdentity();
90      gluOrtho2D(0.0, 499.0, 0.0, 499.0);
91  }
92
93  void main(int argc, char** argv)
94  {
95      glutInit(&argc, argv);
96      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
97      glutInitWindowSize(500, 500);
98      glutInitWindowPosition(0, 0);
99      glutCreateWindow("Filling a Polygon usingScan-line Algorithm");
100     glutDisplayFunc(display);
101     myinit();
102     glutMainLoop();
103 }
```