CG LAB

1. Program to implement Mid Point Line Algorithm. The line coordinates should be specified by the user.

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

float x00, y00, x01, y01;

void swap(float *a, float *b) {
    float temp = *a;
    *a = *b;
    *b = temp;
}

void init() {
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-500, 500, -500, 500);
}

void plot(float x, float y) {
    glColor3f(1, 0, 0);
    glBegin(GL_POINTS);
        glVertex2f(x, y);
    glEnd();
    glFlush();
}

void display() {
    float dx = abs(x01 - x00), dy = abs(y01 - y00);
    int slopegt1 = 0;

    if(dy > dx) {
        swap(&x00, &y00);
        swap(&x01, &y01);
```

```
        swap(&dx, &dy);
        slopegt1 = 1;
    }
    if(x00 > x01) {
        swap(&x00, &x01);
        swap(&y00, &y01);
    }

    float incrY = 1;
    if(y00 > y01)
        incrY = -1;

    float d = 2 * dy - dx;
    float incrE = 2 * dy;
    float incrNE = 2 * (dy - dx);

    while(x00 < x01) {
        if(d <= 0)
            d += incrE;
        else {
            d += incrNE;
            y00 += incrY;
        }
        if(slopegt1)
            plot(y00, x00);
        else
            plot(x00, y00);
        ++x00;
    }
    glFlush();
}

int main(int argc, char *argv[]) {
    printf("Enter the line co-ordinates (x00 y00 x01 y01): ");
    scanf("%f%f%f%f", &x00, &y00, &x01, &y01);

    glutInit(&argc, argv);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Midpoint-Line");
```

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

2. Program to implement Mid Point Circle Algorithm. The radius and center of the circle should be specified by the user.

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

float h, k, r;

void init() {
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-500, 500, -500, 500);
}

void plot(float x, float y) {
    glColor3f(1, 0, 0);
    glBegin(GL_POINTS);
        glVertex2f(x, y);
    glEnd();
    glFlush();
}

void display() {
    float x = 0, y = r, d = 5.0 / 4.0 - r;
    while(y > x) {
        if(d < 0)
            d += (2 * x + 3);
        else {
            d += (2 * (x - y) + 5);
            --y;
        }
        ++x;
```

```
        plot(x + h, y + k);
        plot(-x + h, y + k);
        plot(x + h, -y + k);
        plot(-x + h, -y + k);
        plot(y + h, x + k);
        plot(-y + h, x + k);
        plot(y + h, -x + k);
        plot(-y + h, -x + k);
    }
    glFlush();
}

int main(int argc, char *argv[]) {
    printf("Enter the centre of the circle (x, y): ");
    scanf("%f%f", &h, &k);
    printf("Enter the radius: ");
    scanf("%f", &r);

    glutInit(&argc, argv);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Midpoint-Circle");
    glutDisplayFunc(display);
    init();
    glutMainLoop();
}
```

3. Program to implement the Cohen-Sutherland line-clipping algorithm. Make provision for the user to specify the input line and window coordinates for clipping.

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

void display();

float xmin, ymin, xmax, ymax, lx0, ly0, lx1, ly1;
int TOP = 8, BOTTOM = 4, RIGHT = 2, LEFT = 1;

void init() {
```

```
    glClearColor(0, 0, 0, 0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-500, 500, -500, 500);
}

int getOutcode(float x, float y) {
    int c = 0;
    if(y > ymax)
        c = TOP;
    if(y < ymin)
        c = BOTTOM;
    if(x > xmax)
        c |= RIGHT;
    if(x < xmin)
        c |= LEFT;
    return c;
}

void clipLine(float x1, float y1, float x2, float y2) {
    int outcode1 = getOutcode(x1, y1);
    int outcode2 = getOutcode(x2, y2);
    float m = (y2 - y1) / (x2 - x1);

    while((outcode1 | outcode2) != 0) {
        if((outcode1 & outcode2) != 0) {
            lx0 = ly0 = lx1 = ly1 = -500;
            break;
        }

        float x, y;
        float xi = x1, yi = y1;
        int c = outcode1;
        if(c == 0) {
            xi = x2;
            yi = y2;
            c = outcode2;
        }

        if((c & TOP) != 0) {
```

```
            y = ymax;
            x = xi + 1.0 / m * (ymax - yi);
        } else if((c & BOTTOM) != 0) {
            y = ymin;
            x = xi + 1.0 / m * (ymin - yi);
        } else if((c & RIGHT) != 0) {
            x = xmax;
            y = yi + m * (xmax - xi);
        } else if((c & LEFT) != 0) {
            x = xmin;
            y = yi + m * (xmin - xi);
        }
        if(c == outcode1) {
            lx0 = x;
            ly0 = y;
            outcode1 = getOutcode(lx0, ly0);
        }
        if(c == outcode2) {
            lx1 = x;
            ly1 = y;
            outcode2 = getOutcode(lx1, ly1);
        }
    }
    display();
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1, 0, 0);
    glBegin(GL_LINE_LOOP);
        glVertex2f(xmin, ymin);
        glVertex2f(xmax, ymin);
        glVertex2f(xmax, ymax);
        glVertex2f(xmin, ymax);
    glEnd();

    glColor3f(0, 1, 0);
    glBegin(GL_LINES);
```

```
        glVertex2f(lx0, ly0);
        glVertex2f(lx1, ly1);
    glEnd();
    glFlush();
}

void keypress(unsigned char key, int x, int y) {
    if(key == 'c') {
        clipLine(lx0, ly0, lx1, ly1);
        printf("Line clipped!\n");
        glFlush();
    }
}

int main(int argc, char *argv[]) {
    printf("(Clipping window parameters format: xMin, yMin, xMax,
yMax)\n");
    printf("Enter the clipping window parameters: ");
    scanf("%f%f%f%f", &xmin, &ymin, &xmax, &ymax);

    printf("\n(Line co-ordinates format: x0, y0, x1, y1)\n");
    printf("Enter line co-ordinates: ");
    scanf("%f%f%f%f", &lx0, &ly0, &lx1, &ly1);

    glutInit(&argc, argv);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Cohen-Sutherland");
    glutDisplayFunc(display);
    glutKeyboardFunc(keypress);
    init();
    glutMainLoop();
}
```

4. Program to implement the Liang-Barsky line clipping algorithm. Make provision for the user to specify the input line and window coordinates for clipping.

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

```
float t1 = 0, t2 = 1;
float x1, y1, x2, y2, x3, y3, x4, y4;
float p[4], q[4];
float xmin, ymin, xmax, ymax;

void init() {
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-500, 500, -500, 500);
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0, 0, 1);
    glBegin(GL_LINE_LOOP);
        glVertex2f(xmin, ymin);
        glVertex2f(xmax, ymin);
        glVertex2f(xmax, ymax);
        glVertex2f(xmin, ymax);
    glEnd();
    glColor3f(1, 0, 0);
    glBegin(GL_LINES);
        glVertex2f(x1, y1);
        glVertex2f(x2, y2);
    glEnd();
    glFlush();
}

void lineClip(float x1, float y1, float x2, float y2) {
    float dx = x2 - x1, dy = y2 - y1, t;
    p[0] = -dx;
    p[1] = dx;
    p[2] = -dy;
    p[3] = dy;
    q[0] = x1 - xmin;
    q[1] = xmax - x1;
    q[2] = y1 - ymin;
```

```
    q[3] = ymax - y1;

    for(int i = 0; i < 4; ++i) {
        t = q[i] / p[i];
        if((p[i] == 0) && (q[i] < 0))
            return;
        if((p[i] < 0) && ((t > t1) && (t < t2))) {
            t1 = t;
        } else if((p[i] > 0) && ((t > t1) && (t < t2))) {
            t2 = t;
        }
    }
    if(t1 < t2) {
        x3 = x1 + t1 * (x2 - x1);
        x4 = x1 + t2 * (x2 - x1);
        y3 = y1 + t1 * (y2 - y1);
        y4 = y1 + t2 * (y2 - y1);
        if(
            ((x3 >= xmin) && (x3 <= xmax) && (y3 >= ymin) && (y3 <=
ymax)) &&
            ((x4 >= xmin) && (x4 <= xmax) && (y4 >= ymin) && (y4 <=
ymax))
        ) {
            glColor3f(1, 0, 0);
            glBegin(GL_LINES);
                glVertex2f(x3, y3);
                glVertex2f(x4, y4);
            glEnd();
            glFlush();
        }
    }
}

void keypress(unsigned char key, int x, int y) {
    if(key == 'c') {
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0, 0, 1);
        glBegin(GL_LINE_LOOP);
            glVertex2f(xmin, ymin);
```

```
            glVertex2f(xmax, ymin);
            glVertex2f(xmax, ymax);
            glVertex2f(xmin, ymax);
        glEnd();
        lineClip(x1, y1, x2, y2);
        printf("Line clipped!\n");
    }
}

int main(int argc, char *argv[]) {
    printf("(Clipping window parameters format: xMin, yMin, xMax,
yMax)\n");
    printf("Enter the clipping window parameters: ");
    scanf("%f%f%f%f", &xmin, &ymin, &xmax, &ymax);

    printf("\n(Line co-ordinates format: x1, y1, x2, y2)\n");
    printf("Enter line co-ordinates: ");
    scanf("%f%f%f%f", &x1, &y1, &x2, &y2);

    glutInit(&argc, argv);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Liang-Barsky");
    glutDisplayFunc(display);
    glutKeyboardFunc(keypress);
    init();
    glutMainLoop();
}
```

5. Program to recursively subdivide a triangle to form 2D Sierpinski gasket. The number of recursive steps is to be specified by the user.

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

int n;
float a[2] = {1, 1}, b[2] = {6, 1}, c[2] = {3.5, 5};

void init() {
    glClearColor(0, 0, 0, 0);
```

```
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0, 10, 0, 10);
}

void drawTrinagle(float a[], float b[], float c[]) {
    glVertex2fv(a);
    glVertex2fv(b);
    glVertex2fv(c);
}

void divideTriangle(float a[], float b[], float c[], int k) {
    float ab[2], ca[2], bc[2];
    if(k > 0) {
        for(int i = 0; i < 2; ++i) {
            ab[i] = (a[i] + b[i]) / 2;
            bc[i] = (b[i] + c[i]) / 2;
            ca[i] = (c[i] + a[i]) / 2;
        }
        divideTriangle(a, ab, ca, k - 1);
        divideTriangle(b, bc, ab, k - 1);
        divideTriangle(c, ca, bc, k - 1);
    } else {
        drawTrinagle(a, b, c);
    }
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
        glColor3f(1, 0, 0);
        divideTriangle(a, b, c, n);
    glEnd();
    glFlush();
}

int main(int argc, char *argv[]) {
    printf("Enter the number of divisions: ");
    scanf("%d", &n);
    glutInit(&argc, argv);
```

```
    glutInitWindowSize(500, 500);
    glutCreateWindow("2D Gasket");
    glutDisplayFunc(display);
    init();
    glutMainLoop();
}
```

6. Program to draw a color cube and spin it using OpenGL transformation matrices along x, y and z axes.

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

GLfloat vertices[][3] = {
    {-1, -1, -1}, {1, -1, -1}, {1, 1, -1}, {-1, 1, -1},
    {-1, -1, 1}, {1, -1, 1}, {1, 1, 1}, {-1, 1, 1}
};
GLfloat normals[][3] = {
    {-1, -1, -1}, {1, -1, 1}, {1, 1, -1}, {-1, 1, -1},
    {-1, -1, 1}, {1, -1, 1}, {1, 1, 1}, {-1, 1, 1}
};
GLfloat colors[][3] = {
    {0, 0, 0}, {0, 0, 1}, {0, 1, 0}, {0, 1, 1},
    {1, 0, 0}, {1, 0, 1}, {1, 1, 0}, {1, 1, 1}
};

static GLfloat theta[] = {0, 0, 0};
static GLint axis = 2;

void drawSide(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 drawCube() {
    drawSide(0, 3, 2, 1);
    drawSide(2, 3, 7, 6);
    drawSide(0, 3, 7, 4);
    drawSide(0, 4, 5, 1);
    drawSide(4, 5, 6, 7);
    drawSide(1, 2, 6, 5);
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glRotatef(theta[0], 1, 0, 0);
    glRotatef(theta[1], 0, 1, 0);
    glRotatef(theta[2], 0, 0, 1);
    drawCube();
    glFlush();
    glutSwapBuffers();
}

void spinCube() {
    theta[axis] += 1;
    if(theta[axis] > 360)
        theta[axis] -= 360;
    glutPostRedisplay();
}

void mouseEvent(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 reshape(int w, int h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if(w <= h)
        glOrtho(-2, 2, -2.0 * (GLfloat) h / (GLfloat) w, 2.0 * (GLfloat) h /
(GLfloat) w, -10, 10);
    else
        glOrtho(-2.0 * (GLfloat) w / (GLfloat) h, 2.0 * (GLfloat) w /
(GLfloat) h, -2, 2, -10, 10);
    glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char *argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Color Cube");
    glutDisplayFunc(display);
    glutMouseFunc(mouseEvent);
    glutIdleFunc(spinCube);
    glutReshapeFunc(reshape);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
}
```

7. Program to fill any given polygon using scan-line area filling algorithm. Make provision for the user to enter the vertices of the polygon.

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

float x1, y1, x2, y2, x3, y3, x4, y4;
```

```
void swap(float *a, float *b) {
    int t = *a;
    *a = *b;
    *b = t;
}

void edgeDetect(float x1, float y1, float x2, float y2, int *le, int *re) {
    float x, mx;
    if((y2 - y1) < 0) {
        swap(&x1, &x2);
        swap(&y1, &y2);
    }
    if((y2 - y1) != 0)
        mx = (x2 - x1) / (y2 - y1);
    else
        mx = (x2 - x1);
    x = x1;
    for(int 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 plot(int x, int y) {
    glColor3f(0, 1, 0);
    glBegin(GL_POINTS);
        glVertex2f(x, y);
    glEnd();
}

void scanLine(float x1, float y1, float x2, float y2, float x3, float y3, float
x4, float y4) {
    int le[500], re[500];
    for(int 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(int y = 0; y < 500; ++y) {
        if(le[y] <= re[y])
            for(int i = (int)le[y]; i <= (int) re[y]; ++i)
                plot(i, y);
    }
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINE_LOOP);
        glVertex2f(x1, y1);
        glVertex2f(x2, y2);
        glVertex2f(x3, y3);
        glVertex2f(x4, y4);
    glEnd();
    scanLine(x1, y1, x2, y2, x3, y3, x4, y4);
    glFlush();
}

void init() {
    glClearColor(1, 1, 1, 1);
    glPointSize(1);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 500, 0, 500);
}

int main(int argc, char *argv[]) {
    printf("Enter the polygon vertices: ");
    printf("\nEnter the first vertex (x, y): ");
    scanf("%f%f", &x1, &y1);
    printf("\nEnter the second vertex (x, y): ");
```

```
    scanf("%f%f", &x2, &y2);
    printf("\nEnter the third vertex (x, y): ");
    scanf("%f%f", &x3, &y3);
    printf("\nEnter the fourth vertex (x, y): ");
    scanf("%f%f", &x4, &y4);

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Scan Fill");
    glutDisplayFunc(display);
    init();
    glutMainLoop();
}
```

8. Program to create a random figure and rotate it about a given fixed point using transformation matrices. Make provision for user to give angle of rotation and pivot point.

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

float theta;
GLfloat rotMat[3][3], res[9][3], m, n, tx, ty;

GLfloat house[9][3] = {
    {200, 200, 1}, {300, 200, 1}, {300, 300, 1},
    {200, 300, 1}, {250, 350, 1}, {225, 200, 1},
    {275, 200, 1}, {275, 250, 1}, {225, 250, 1}
};

void init() {
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0, 500, 0, 500);
}
```

```
void multiply() {
    for(int p = 0; p < 9; ++p)
        for(int q = 0; q < 3; ++q) {
            res[p][q] = 0;
            for(int r = 0; r < 3; ++r)
                res[p][q] += (house[p][r] * rotMat[r][q]);
        }
}

void rotate() {
    m = (-tx * cos(theta)) + (ty * sin(theta)) + tx;
    n = (-tx * sin(theta)) - (ty * cos(theta)) + ty;

    rotMat[0][0] = cos(theta);
    rotMat[0][1] = sin(theta);
    rotMat[0][2] = 0;
    rotMat[1][0] = -sin(theta);
    rotMat[1][1] = cos(theta);
    rotMat[1][2] = 0;
    rotMat[2][0] = m;
    rotMat[2][1] = n;
    rotMat[2][2] = 1;

    multiply();
}

void drawHouse(GLfloat mat[9][3]) {
    glBegin(GL_LINE_LOOP);
        glVertex2f(mat[0][0], mat[0][1]);
        glVertex2f(mat[1][0], mat[1][1]);
        glVertex2f(mat[2][0], mat[2][1]);
        glVertex2f(mat[3][0], mat[3][1]);
    glEnd();
    glBegin(GL_LINE_LOOP);
        glVertex2f(mat[2][0], mat[2][1]);
        glVertex2f(mat[3][0], mat[3][1]);
        glVertex2f(mat[4][0], mat[4][1]);
    glEnd();
    glBegin(GL_LINE_LOOP);
```

```
        glVertex2f(mat[5][0], mat[5][1]);
        glVertex2f(mat[6][0], mat[6][1]);
        glVertex2f(mat[7][0], mat[7][1]);
        glVertex2f(mat[8][0], mat[8][1]);
    glEnd();
}

void display() {
    glColor3f(1, 0, 0);
    drawHouse(house);
    rotate();
    glColor3f(0, 0, 1);
    drawHouse(res);
    glFlush();
}

int main(int argc, char *argv[]) {
    printf("Enter the pivot point (x, y): ");
    scanf("%f%f", &tx, &ty);
    printf("Enter the angle of rotation: ");
    scanf("%f", &theta);
    theta = (3.14 * theta) / 180;

    glutInit(&argc, argv);
    glutInitWindowSize(500, 500);
    glutCreateWindow("House Rotation");
    glutDisplayFunc(display);
    init();
    glutMainLoop();
}
```

9. Program to display a set of values {f(i, j)} as a rectangular mesh.

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

#define dx 25
#define dy 20
```

```
int maxx, maxy, x0 = 100, y0 = 100, x[100] = {0}, y[100] = {0};

void init() {
   glClearColor(0, 0, 0, 0);
   glClear(GL_COLOR_BUFFER_BIT);
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D(0, 500, 0, 500);
}

void display() {
   glColor3f(1, 0, 0);

   for(int i = 0; i < maxx; ++i)
      x[i] = x0 + i * dx;
   for(int j = 0; j < maxy; ++j)
      y[j] = y0 + j * dy;

   for(int i = 0; i < maxx - 1; ++i)
      for(int j = 0; j < maxy - 1; ++j) {
         glBegin(GL_LINE_LOOP);
            glVertex2f(x[i], y[j]);
            glVertex2f(x[i + 1], y[j]);
            glVertex2f(x[i + 1], y[j + 1]);
            glVertex2f(x[i], y[j + 1]);
         glEnd();
      }
   glFlush();
}

int main(int argc, char *argv[]) {
   printf("Enter the size of the mesh (maxX maxY): ");
   scanf("%d%d", &maxx, &maxy);

   glutInit(&argc, argv);
   glutInitWindowSize(500, 500);
   glutCreateWindow("Rectangular Mesh");
   glutDisplayFunc(display);
   init();
   glutMainLoop();
```

```
}
```

10. Program to create a random object and to implement the suggested mouse and keyboard interactions through OpenGL function.

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

int WIDTH = 1533, HEIGHT = 845;

void init() {
    glClearColor(1, 1, 1, 1);
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0, WIDTH, HEIGHT, 0);
}

void drawPoint(int x, int y) {
    glPointSize(5);
    glColor3f(0, 0, 0);
    glBegin(GL_POINTS);
        glVertex2f(x, y);
    glEnd();
    glFlush();
}

void drawLine(int x, int y) {
    glLineWidth(5);
    glColor3f(0, 0, 0);
    glBegin(GL_LINES);
        glVertex2f(x - 50, y - 50);
        glVertex2f(x + 50, y + 50);
    glEnd();
    glFlush();
}

void drawTriangle(int x, int y) {
    glColor3f(1, 0, 0);
    glBegin(GL_TRIANGLES);
```

```
        glVertex2f(x - 50, y - 25);
        glVertex2f(x + 50, y - 25);
        glVertex2f(x, y + 50);
    glEnd();
    glFlush();
}

void drawSquare(int x, int y) {
    glColor3f(0, 1, 0);
    glBegin(GL_POLYGON);
        glVertex2f(x - 50, y - 50);
        glVertex2f(x + 50, y - 50);
        glVertex2f(x + 50, y + 50);
        glVertex2f(x - 50, y + 50);
    glEnd();
    glFlush();
}

void drawPentagon(int x, int y) {
    glColor3f(1, 1, 0);
    glBegin(GL_POLYGON);
        glVertex2f(x - 50, y - 40);
        glVertex2f(x + 50, y - 40);
        glVertex2f(x + 50, y + 40);
        glVertex2f(x, y + 80);
        glVertex2f(x - 50, y + 40);
    glEnd();
    glFlush();
}

void drawStar(int x, int y) {
    glColor3f(0, 0, 1);
    glBegin(GL_TRIANGLES);
        glVertex2f(x - 50, y - 25);
        glVertex2f(x + 50, y - 25);
        glVertex2f(x, y + 50);
        glVertex2f(x - 50, y + 25);
        glVertex2f(x + 50, y + 25);
        glVertex2f(x, y - 50);
```

```
    glEnd();
    glFlush();
}

void mouseclick(int btn, int state, int x, int y) {
    if(btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
        drawPoint(x, y);
    if(btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
        drawLine(x, y);
}

void keypress(unsigned char key, int x, int y) {
    if(key == 't' || key == 'T')
        drawTriangle(x, y);
    else if(key == 's' || key == 'S')
        drawSquare(x, y);
    else if(key == 'p' || key == 'P')
        drawPentagon(x, y);
    else if(key == 'n' || key == 'N')
        drawStar(x, y);

    if(key == 'c' || key == 'C') {
        glClear(GL_COLOR_BUFFER_BIT);
        glFlush();
    }
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}

int main(int argc, char *argv[]) {
    glutInit(&argc, argv);
    glutInitWindowSize(WIDTH, HEIGHT);
    glutCreateWindow("Events");
    glutDisplayFunc(display);
    glutKeyboardFunc(keypress);
    glutMouseFunc(mouseclick);
```

```
    init();
    glutMainLoop();
}
```

EXTRA: 3D Sierpinski gasket

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

typedef float point[3];

point v[] = {
    {0, 0, 1}, {0, 0.9, -0.3},
    {-0.8, -0.4, -0.3}, {0.8, -0.4, -0.9}
};

int n;

void drawTriangle(point a, point b, point c) {
    glBegin(GL_POLYGON);
        glNormal3fv(a);
        glVertex3fv(a);
        glVertex3fv(b);
        glVertex3fv(c);
    glEnd();
}

void divideTriangle(point a, point b, point c, int m) {
    point v1, v2, v3;
    if(m > 0) {
        for(int j = 0; j < 3; ++j) {
            v1[j] = (a[j] + b[j]) / 2;
            v2[j] = (a[j] + c[j]) / 2;
            v3[j] = (b[j] + c[j]) / 2;
        }
        divideTriangle(a, v1, v2, m - 1);
        divideTriangle(c, v2, v3, m - 1);
        divideTriangle(b, v3, v1, m - 1);
    } else {
```

```
        drawTriangle(a, b, c);
    }
}

void drawTetrahedron(int m) {
    glColor3f(1, 0.2, 0);
    divideTriangle(v[0], v[1], v[2], m);

    glColor3f(0, 0, 1);
    divideTriangle(v[3], v[2], v[1], m);

    glColor3f(0.5, 1.6, 1.7);
    divideTriangle(v[0], v[3], v[1], m);

    glColor3f(0, 1, 0);
    divideTriangle(v[0], v[2], v[3], m);
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    drawTetrahedron(n);
    glFlush();
}

void reshape(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();
}

int main(int argc, char *argv[]) {
    printf("Enter the number of divisions: ");
    scanf("%d", &n);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("3D Gasket");
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glEnable(GL_DEPTH_TEST);
    glClearColor(1, 1, 1, 1);
    glutMainLoop();
}
```