**Koch Curve**

#include <GL/glut.h>

#include <cmath>

int depth = 4; // Depth of recursion

struct Point {

float x, y;

};

void drawKoch(Point a, Point b, int level) {

if (level == 0) {

glBegin(GL\_LINES);

glVertex2f(a.x, a.y);

glVertex2f(b.x, b.y);

glEnd();

return;

}

Point c, d, e;

// Divide the segment into 3 parts

c.x = (2 \* a.x + b.x) / 3;

c.y = (2 \* a.y + b.y) / 3;

d.x = (a.x + 2 \* b.x) / 3;

d.y = (a.y + 2 \* b.y) / 3;

// Calculate the peak point of the triangle (equilateral)

float angle = M\_PI / 3; // 60 degrees

e.x = c.x + (d.x - c.x) \* cos(angle) - (d.y - c.y) \* sin(angle);

e.y = c.y + (d.x - c.x) \* sin(angle) + (d.y - c.y) \* cos(angle);

// Recursively draw the 4 parts

drawKoch(a, c, level - 1);

drawKoch(c, e, level - 1);

drawKoch(e, d, level - 1);

drawKoch(d, b, level - 1);

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0, 0, 1); // Blue lines

Point a = { -0.8f, 0.0f };

Point b = { 0.8f, 0.0f };

drawKoch(a, b, depth);

glFlush();

}

void keyboard(unsigned char key, int x, int y) {

if (key == 'c' || key == 'C') {

depth = 0;

glutPostRedisplay();

}

if (key >= '0' && key <= '6') {

depth = key - '0'; // change depth using 0–6 keys

glutPostRedisplay();

}

}

void init() {

glClearColor(1, 1, 1, 1); // white background

gluOrtho2D(-1, 1, -1, 1); // set coordinate system

}

int main(int argc, char\*\* argv) {

glutInit(&argc, argv);

glutInitWindowSize(800, 600);

glutCreateWindow("Koch Curve Fractal");

init();

glutDisplayFunc(display);

glutKeyboardFunc(keyboard);

glutMainLoop();

return 0;

}

**1. Header and Globals**

cpp

CopyEdit

#include <GL/glut.h>

#include <cmath>

int depth = 4; // Depth of recursion

* GL/glut.h: for OpenGL graphics and windowing.
* cmath: for trigonometric functions like cos() and sin().
* depth: how many recursive levels the Koch curve will go.

**2. Point Structure**

cpp

CopyEdit

struct Point {

float x, y;

};

A simple structure to store coordinates.

**3. Recursive Koch Curve Function**

cpp

CopyEdit

void drawKoch(Point a, Point b, int level)

Draws a Koch curve between points a and b.

**🔸 Base Case:**

cpp

CopyEdit

if (level == 0) {

glBegin(GL\_LINES);

glVertex2f(a.x, a.y);

glVertex2f(b.x, b.y);

glEnd();

return;

}

If recursion level is 0, just draw a straight line from a to b.

**4. Divide the Segment**

cpp

CopyEdit

Point c, d, e;

We divide the line segment ab into three equal parts:

cpp

CopyEdit

c.x = (2 \* a.x + b.x) / 3;

c.y = (2 \* a.y + b.y) / 3;

d.x = (a.x + 2 \* b.x) / 3;

d.y = (a.y + 2 \* b.y) / 3;

**📐 Formula:**

If you have points a and b, then:

* First third:

c=(2a+b3)c = \left( \frac{2a + b}{3} \right)c=(32a+b​)

* Second third:

d=(a+2b3)d = \left( \frac{a + 2b}{3} \right)d=(3a+2b​)

These create 3 segments: a → c, c → d, and d → b.

**5. Create the Triangle Peak (Point e)**

cpp

CopyEdit

float angle = M\_PI / 3; // 60 degrees

e.x = c.x + (d.x - c.x) \* cos(angle) - (d.y - c.y) \* sin(angle);

e.y = c.y + (d.x - c.x) \* sin(angle) + (d.y - c.y) \* cos(angle);

This rotates the vector cd by **+60° counter-clockwise** around point c to create point e.

**🔺 Formula:**

This uses the 2D rotation formula:

x′=x⋅cos⁡θ−y⋅sin⁡θy′=x⋅sin⁡θ+y⋅cos⁡θx' = x \cdot \cos\theta - y \cdot \sin\theta \\ y' = x \cdot \sin\theta + y \cdot \cos\thetax′=x⋅cosθ−y⋅sinθy′=x⋅sinθ+y⋅cosθ

Here:

* Vector cd is rotated 60°.
* The result is added to point c to position e.

**6. Recursive Calls**

cpp

CopyEdit

drawKoch(a, c, level - 1);

drawKoch(c, e, level - 1);

drawKoch(e, d, level - 1);

drawKoch(d, b, level - 1);

You now have 4 new segments:

* a → c
* c → e (triangle left side)
* e → d (triangle right side)
* d → b

Each segment becomes a Koch curve recursively.

**7. OpenGL Display Function**

cpp

CopyEdit

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0, 0, 1); // Blue

Point a = { -0.8f, 0.0f };

Point b = { 0.8f, 0.0f };

drawKoch(a, b, depth);

glFlush();

}

* Clears the screen.
* Defines a horizontal line from left to right.
* Calls drawKoch() with the current recursion depth.

**8. Keyboard Interaction**

cpp

CopyEdit

void keyboard(unsigned char key, int x, int y) {

if (key == 'c' || key == 'C') {

depth = 0;

glutPostRedisplay();

}

if (key >= '0' && key <= '6') {

depth = key - '0'; // keys 0–6 control recursion depth

glutPostRedisplay();

}

}

* 'c': clears the curve.
* '0'–'6': sets Koch curve recursion depth interactively.

**9. Initialization**

cpp

CopyEdit

void init() {

glClearColor(1, 1, 1, 1); // white background

gluOrtho2D(-1, 1, -1, 1); // 2D orthographic view

}

**10. Main Function**

cpp

CopyEdit

int main(int argc, char\*\* argv)

* Sets up the OpenGL window.
* Assigns display and keyboard callback functions.
* Enters the main loop.

**Polygon Filling**

#include <GL/glut.h>

#include <vector>

#include <iostream>

using namespace std;

vector<pair<int, int>> polygon;

bool isPolygonClosed = false;

char fillMode = 'n'; // 'f' = flood fill, 'b' = boundary fill

int winWidth = 800, winHeight = 600;

void setPixel(int x, int y, float r, float g, float b) {

glColor3f(r, g, b);

glBegin(GL\_POINTS);

glVertex2i(x, y);

glEnd();

glFlush();

}

void getPixelColor(int x, int y, float\* color) {

glReadPixels(x, winHeight - y, 1, 1, GL\_RGB, GL\_FLOAT, color);

}

bool colorsMatch(float\* c1, float\* c2) {

return (int(c1[0]\*255) == int(c2[0]\*255) &&

int(c1[1]\*255) == int(c2[1]\*255) &&

int(c1[2]\*255) == int(c2[2]\*255));

}

void floodFill(int x, int y, float\* oldColor, float\* newColor) {

float current[3];

getPixelColor(x, y, current);

if (!colorsMatch(current, oldColor) || colorsMatch(current, newColor))

return;

setPixel(x, y, newColor[0], newColor[1], newColor[2]);

floodFill(x + 1, y, oldColor, newColor);

floodFill(x - 1, y, oldColor, newColor);

floodFill(x, y + 1, oldColor, newColor);

floodFill(x, y - 1, oldColor, newColor);

}

void boundaryFill(int x, int y, float\* boundaryColor, float\* fillColor) {

float current[3];

getPixelColor(x, y, current);

if (colorsMatch(current, boundaryColor) || colorsMatch(current, fillColor))

return;

setPixel(x, y, fillColor[0], fillColor[1], fillColor[2]);

boundaryFill(x + 1, y, boundaryColor, fillColor);

boundaryFill(x - 1, y, boundaryColor, fillColor);

boundaryFill(x, y + 1, boundaryColor, fillColor);

boundaryFill(x, y - 1, boundaryColor, fillColor);

}

void drawPolygon() {

glColor3f(0.0, 0.0, 0.0); // black border

glBegin(GL\_LINE\_LOOP);

for (auto& p : polygon) {

glVertex2i(p.first, p.second);

}

glEnd();

glFlush();

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

if (isPolygonClosed)

drawPolygon();

glFlush();

}

void mouse(int button, int state, int x, int y) {

y = winHeight - y; // Invert y coordinate

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN && !isPolygonClosed) {

polygon.push\_back({x, y});

int sz = polygon.size();

if (sz > 1) {

glColor3f(0, 0, 0); // black lines

glBegin(GL\_LINES);

glVertex2i(polygon[sz - 2].first, polygon[sz - 2].second);

glVertex2i(polygon[sz - 1].first, polygon[sz - 1].second);

glEnd();

glFlush();

} else {

// plot first point

setPixel(x, y, 0, 0, 0);

}

}

else if (button == GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN && !isPolygonClosed && polygon.size() > 2) {

// Close polygon

isPolygonClosed = true;

drawPolygon();

}

else if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN && isPolygonClosed) {

float oldColor[3], fillColor[3] = {1.0, 0.0, 0.0}; // red fill

getPixelColor(x, y, oldColor);

if (fillMode == 'f') {

floodFill(x, y, oldColor, fillColor);

} else if (fillMode == 'b') {

float boundaryColor[3] = {0.0, 0.0, 0.0}; // black boundary

boundaryFill(x, y, boundaryColor, fillColor);

}

}

}

void keyboard(unsigned char key, int x, int y) {

switch (key) {

case 'c':

case 'C':

polygon.clear();

isPolygonClosed = false;

fillMode = 'n';

glClear(GL\_COLOR\_BUFFER\_BIT);

glFlush();

break;

case 'f':

case 'F':

fillMode = 'f';

cout << "Flood Fill Mode. Click inside polygon.\n";

break;

case 'b':

case 'B':

fillMode = 'b';

cout << "Boundary Fill Mode. Click inside polygon.\n";

break;

}

}

void init() {

glClearColor(1, 1, 1, 1); // white background

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0, winWidth, 0, winHeight); // set origin to bottom-left

}

int main(int argc, char\*\* argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(winWidth, winHeight);

glutCreateWindow("Polygon Fill - Flood & Boundary");

init();

glutDisplayFunc(display);

glutMouseFunc(mouse);

glutKeyboardFunc(keyboard);

glutMainLoop();

return 0;

}

**1. Include Libraries**

cpp

CopyEdit

#include <GL/glut.h>

#include <vector>

#include <iostream>

* GL/glut.h: This is the OpenGL Utility Toolkit (GLUT) that helps in creating windows, handling input, and drawing basic shapes.
* vector: Used to store the vertices of the polygon.
* iostream: Used for printing messages in the console (like instructions to the user).

**2. Global Variables**

cpp

CopyEdit

vector<pair<int, int>> polygon;

bool isPolygonClosed = false;

char fillMode = 'n'; // 'f' = flood fill, 'b' = boundary fill

int winWidth = 800, winHeight = 600;

* polygon: A vector of pairs, each holding the x and y coordinates of each vertex of the polygon.
* isPolygonClosed: A boolean flag to check whether the polygon is closed.
* fillMode: Keeps track of which fill mode is currently active ('f' for Flood Fill, 'b' for Boundary Fill, 'n' for none).
* winWidth and winHeight: Define the size of the window.

**3. setPixel() - Drawing Points**

cpp

CopyEdit

void setPixel(int x, int y, float r, float g, float b) {

glColor3f(r, g, b);

glBegin(GL\_POINTS);

glVertex2i(x, y);

glEnd();

glFlush();

}

* This function sets a pixel at a specific (x, y) coordinate with the color (r, g, b).
* It uses OpenGL's glBegin(GL\_POINTS) to draw individual points and glEnd() to finish the drawing.

**4. getPixelColor() - Reading Pixel Color**

cpp

CopyEdit

void getPixelColor(int x, int y, float\* color) {

glReadPixels(x, winHeight - y, 1, 1, GL\_RGB, GL\_FLOAT, color);

}

* Reads the color of a pixel at (x, y) and stores the RGB values in the color array.
* The y-coordinate is inverted because OpenGL's default origin is at the bottom-left, but the screen's origin is usually at the top-left.

**5. colorsMatch() - Comparing Colors**

cpp

CopyEdit

bool colorsMatch(float\* c1, float\* c2) {

return (int(c1[0]\*255) == int(c2[0]\*255) &&

int(c1[1]\*255) == int(c2[1]\*255) &&

int(c1[2]\*255) == int(c2[2]\*255));

}

* Compares two colors by checking if their RGB components are the same. It scales the floating-point values (ranging from 0.0 to 1.0) to integer values (0 to 255) for comparison.

**6. Flood Fill Algorithm**

cpp

CopyEdit

void floodFill(int x, int y, float\* oldColor, float\* newColor) {

float current[3];

getPixelColor(x, y, current);

if (!colorsMatch(current, oldColor) || colorsMatch(current, newColor))

return;

setPixel(x, y, newColor[0], newColor[1], newColor[2]);

floodFill(x + 1, y, oldColor, newColor);

floodFill(x - 1, y, oldColor, newColor);

floodFill(x, y + 1, oldColor, newColor);

floodFill(x, y - 1, oldColor, newColor);

}

* **Flood Fill** starts from a point inside the polygon and recursively "fills" adjacent pixels.
* It checks if the current pixel color matches the old color (the one to be replaced), and if not, it stops.
* Recursively applies the fill to the four neighboring pixels (left, right, up, and down).

**7. Boundary Fill Algorithm**

cpp

CopyEdit

void boundaryFill(int x, int y, float\* boundaryColor, float\* fillColor) {

float current[3];

getPixelColor(x, y, current);

if (colorsMatch(current, boundaryColor) || colorsMatch(current, fillColor))

return;

setPixel(x, y, fillColor[0], fillColor[1], fillColor[2]);

boundaryFill(x + 1, y, boundaryColor, fillColor);

boundaryFill(x - 1, y, boundaryColor, fillColor);

boundaryFill(x, y + 1, boundaryColor, fillColor);

boundaryFill(x, y - 1, boundaryColor, fillColor);

}

* **Boundary Fill** works similarly to Flood Fill but stops when it encounters a boundary (typically the polygon's edge).
* It fills until it reaches pixels that match the boundary color or the fill color, preventing it from filling outside the polygon.

**8. Drawing the Polygon**

cpp

CopyEdit

void drawPolygon() {

glColor3f(0.0, 0.0, 0.0); // black border

glBegin(GL\_LINE\_LOOP);

for (auto& p : polygon) {

glVertex2i(p.first, p.second);

}

glEnd();

glFlush();

}

* Once the polygon is closed, this function draws the polygon using GL\_LINE\_LOOP, which automatically connects the first and last vertices to close the shape.
* It sets the color to black for the polygon's border.

**9. Mouse Input (Left and Right Clicks)**

cpp

CopyEdit

void mouse(int button, int state, int x, int y) {

y = winHeight - y; // Convert to OpenGL coordinate system

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN && !isPolygonClosed) {

polygon.push\_back({x, y});

int sz = polygon.size();

if (sz > 1) {

glColor3f(0, 0, 0); // black lines

glBegin(GL\_LINES);

glVertex2i(polygon[sz - 2].first, polygon[sz - 2].second);

glVertex2i(polygon[sz - 1].first, polygon[sz - 1].second);

glEnd();

glFlush();

} else {

setPixel(x, y, 0, 0, 0); // plot first point

}

}

else if (button == GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN && !isPolygonClosed && polygon.size() > 2) {

// Close polygon

isPolygonClosed = true;

drawPolygon();

}

else if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN && isPolygonClosed) {

// Fill Polygon

float oldColor[3];

float fillColor[3] = {1.0, 0.0, 0.0}; // red fill

getPixelColor(x, y, oldColor);

if (fillMode == 'f') {

floodFill(x, y, oldColor, fillColor);

} else if (fillMode == 'b') {

float boundaryColor[3] = {0.0, 0.0, 0.0}; // black boundary

boundaryFill(x, y, boundaryColor, fillColor);

}

}

}

* **Left-click** adds a vertex to the polygon and draws lines between consecutive vertices.
* **Right-click** closes the polygon by connecting the last vertex to the first.
* Once the polygon is closed, **Left-click** inside the polygon to fill it with the selected fill mode (Flood or Boundary).

**10. Keyboard Input**

cpp

CopyEdit

void keyboard(unsigned char key, int x, int y) {

switch (key) {

case 'c':

case 'C':

polygon.clear();

isPolygonClosed = false;

fillMode = 'n';

glClear(GL\_COLOR\_BUFFER\_BIT);

glFlush();

break;

case 'f':

case 'F':

fillMode = 'f';

cout << "Flood Fill Mode. Click inside polygon.\n";

break;

case 'b':

case 'B':

fillMode = 'b';

cout << "Boundary Fill Mode. Click inside polygon.\n";

break;

}

}

* 'c': Clears the screen and resets everything.
* 'f': Activates **Flood Fill** mode.
* 'b': Activates **Boundary Fill** mode.

**11. OpenGL Initialization**

cpp

CopyEdit

void init() {

glClearColor(1, 1, 1, 1); // white background

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0, winWidth, 0, winHeight); // set origin to bottom-left

}

* Sets up the OpenGL context and view. The projection matrix is set to use 2D coordinates with the origin at the bottom-left corner.

**12. Main Function**

cpp

CopyEdit

int main(int argc, char\*\* argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(winWidth, winHeight);

glutCreateWindow("Polygon Fill - Flood & Boundary");

init();

glutDisplayFunc(display);

glutMouseFunc(mouse);

glutKeyboardFunc(keyboard);

glutMainLoop();

return 0;

}

* Initializes the GLUT window and sets up the display, mouse, and keyboard callbacks.
* Enters the main event loop with glutMainLoop().

**✨ How to Use:**

1. **Left-click** to add vertices of the polygon.
2. **Right-click** to close the polygon.
3. **Press 'f'** to activate **Flood Fill** and click inside the polygon to fill it.
4. **Press 'b'** to activate **Boundary Fill** and click inside the polygon to fill it.
5. **Press 'c'** to clear the screen and reset everything.