**DDA and Bresenham**

#include <GL/glut.h>

#include <cmath>

#include <vector>

using namespace std;

struct Point {

int x, y;

};

Point p1, p2;

bool firstPointSet = false;

int lineType = 1; // 1: Simple, 2: Dotted, 3: Dashed, 4: Solid

bool useDDA = true;

void drawPixel(int x, int y) {

glBegin(GL\_POINTS);

glVertex2i(x, y);

glEnd();

}

void drawLineDDA(Point p1, Point p2) {

int dx = p2.x - p1.x;

int dy = p2.y - p1.y;

int steps = max(abs(dx), abs(dy));

float xInc = dx / (float)steps;

float yInc = dy / (float)steps;

float x = p1.x, y = p1.y;

for (int i = 0; i <= steps; i++) {

if (lineType == 2 && i % 4 == 0)

drawPixel(round(x), round(y));

else if (lineType == 3 && (i / 5) % 2 == 0)

drawPixel(round(x), round(y));

else if (lineType == 1 || lineType == 4)

drawPixel(round(x), round(y));

x += xInc;

y += yInc;

}

}

void drawLineBresenham(Point p1, Point p2) {

int x1 = p1.x, y1 = p1.y;

int x2 = p2.x, y2 = p2.y;

int dx = abs(x2 - x1), dy = abs(y2 - y1);

int sx = x1 < x2 ? 1 : -1;

int sy = y1 < y2 ? 1 : -1;

int err = dx - dy;

int count = 0;

while (true) {

if (lineType == 2 && count % 4 == 0)

drawPixel(x1, y1);

else if (lineType == 3 && (count / 5) % 2 == 0)

drawPixel(x1, y1);

else if (lineType == 1 || lineType == 4)

drawPixel(x1, y1);

if (x1 == x2 && y1 == y2) break;

int e2 = 2 \* err;

if (e2 > -dy) { err -= dy; x1 += sx; }

if (e2 < dx) { err += dx; y1 += sy; }

count++;

}

}

void drawAxes() {

glColor3f(0.5, 0.5, 0.5);

for (int i = -250; i <= 250; i++) {

drawPixel(i, 0);

drawPixel(0, i);

}

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

drawAxes();

glFlush();

}

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

if (key == 'c') {

glClear(GL\_COLOR\_BUFFER\_BIT);

drawAxes();

firstPointSet = false;

glFlush();

}

}

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

if (state != GLUT\_DOWN) return;

int mx = x - 250;

int my = 250 - y;

if (button == GLUT\_LEFT\_BUTTON) {

p1 = { mx, my };

firstPointSet = true;

} else if (button == GLUT\_RIGHT\_BUTTON) {

if (firstPointSet) {

p2 = { mx, my };

glColor3f(1.0, 0.0, 0.0);

if (useDDA)

drawLineDDA(p1, p2);

else

drawLineBresenham(p1, p2);

glFlush();

firstPointSet = false;

}

} else if (button == GLUT\_MIDDLE\_BUTTON) {

glutAttachMenu(GLUT\_RIGHT\_BUTTON); // Required to re-attach

}

}

void menu(int option) {

switch (option) {

case 1: lineType = 1; break;

case 2: lineType = 2; break;

case 3: lineType = 3; break;

case 4: lineType = 4; break;

case 5: useDDA = true; break;

case 6: useDDA = false; break;

case 7: glClear(GL\_COLOR\_BUFFER\_BIT); drawAxes(); glFlush(); break;

}

}

void createMenu() {

int submenu1 = glutCreateMenu(menu);

glutAddMenuEntry("Simple Line", 1);

glutAddMenuEntry("Dotted Line", 2);

glutAddMenuEntry("Dashed Line", 3);

glutAddMenuEntry("Solid Line", 4);

int submenu2 = glutCreateMenu(menu);

glutAddMenuEntry("Use DDA", 5);

glutAddMenuEntry("Use Bresenham", 6);

glutCreateMenu(menu);

glutAddSubMenu("Line Type", submenu1);

glutAddSubMenu("Algorithm", submenu2);

glutAddMenuEntry("Clear Screen", 7);

glutAttachMenu(GLUT\_RIGHT\_BUTTON); // menu will show on middle button via our handling

}

void init() {

glClearColor(1, 1, 1, 1);

gluOrtho2D(-250, 250, -250, 250);

glPointSize(2.0);

}

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

glutInit(&argc, argv);

glutInitWindowSize(500, 500);

glutCreateWindow("Line Drawing with Menu and Reset");

init();

glutDisplayFunc(display);

glutMouseFunc(mouse);

glutKeyboardFunc(keyboard);

createMenu();

glutMainLoop();

return 0;

}

**Bezier Curve**

#include <GL/glut.h>

#include <cmath>

#include <vector>

using namespace std;

struct Point {

float x, y;

};

vector<Point> controlPoints;

const int maxPoints = 4;

int recursionDepth = 5;

void drawBezier(const vector<Point>& ctrlPoints, int level) {

if (level == 0) {

glBegin(GL\_LINE\_STRIP);

for (float t = 0; t <= 1.0f; t += 0.01f) {

float u = 1 - t;

float x = pow(u, 3) \* ctrlPoints[0].x +

3 \* t \* pow(u, 2) \* ctrlPoints[1].x +

3 \* pow(t, 2) \* u \* ctrlPoints[2].x +

pow(t, 3) \* ctrlPoints[3].x;

float y = pow(u, 3) \* ctrlPoints[0].y +

3 \* t \* pow(u, 2) \* ctrlPoints[1].y +

3 \* pow(t, 2) \* u \* ctrlPoints[2].y +

pow(t, 3) \* ctrlPoints[3].y;

glVertex2f(x, y);

}

glEnd();

return;

}

Point p0 = ctrlPoints[0];

Point p1 = ctrlPoints[1];

Point p2 = ctrlPoints[2];

Point p3 = ctrlPoints[3];

Point q0 = { (p0.x + p1.x) / 2, (p0.y + p1.y) / 2 };

Point q1 = { (p1.x + p2.x) / 2, (p1.y + p2.y) / 2 };

Point q2 = { (p2.x + p3.x) / 2, (p2.y + p3.y) / 2 };

Point r0 = { (q0.x + q1.x) / 2, (q0.y + q1.y) / 2 };

Point r1 = { (q1.x + q2.x) / 2, (q1.y + q2.y) / 2 };

Point s = { (r0.x + r1.x) / 2, (r0.y + r1.y) / 2 };

vector<Point> first = { p0, q0, r0, s };

vector<Point> second = { s, r1, q2, p3 };

drawBezier(first, level - 1);

drawBezier(second, level - 1);

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

if (controlPoints.size() == maxPoints) {

glColor3f(0.2f, 0.2f, 0.9f); // Blue curve

drawBezier(controlPoints, recursionDepth);

}

// Draw control points

glPointSize(5);

glColor3f(1, 0, 0); // Red points

glBegin(GL\_POINTS);

for (const auto& pt : controlPoints) {

glVertex2f(pt.x, pt.y);

}

glEnd();

glFlush();

}

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

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN && controlPoints.size() < maxPoints) {

float fx = (float)(x - 300) / 300; // Convert to [-1, 1]

float fy = (float)(300 - y) / 300; // Convert to [-1, 1]

controlPoints.push\_back({ fx, fy });

glutPostRedisplay();

}

}

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

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

controlPoints.clear();

glutPostRedisplay();

}

}

void init() {

glClearColor(1, 1, 1, 1);

gluOrtho2D(-1, 1, -1, 1);

}

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

glutInit(&argc, argv);

glutInitWindowSize(600, 600);

glutCreateWindow("Interactive Bezier Fractal with Mouse");

init();

glutDisplayFunc(display);

glutMouseFunc(mouse);

glutKeyboardFunc(keyboard);

glutMainLoop();

return 0;

}

**1. Includes and Data Structures**

cpp

CopyEdit

#include <GL/glut.h>

#include <cmath>

#include <vector>

using namespace std;

struct Point {

float x, y;

};

* GL/glut.h: for OpenGL and GLUT functions.
* cmath: used for math operations like pow.
* vector: stores control points dynamically.
* Point struct: stores x and y coordinates of a point.

**2. Global Variables**

cpp

CopyEdit

vector<Point> controlPoints;

const int maxPoints = 4;

int recursionDepth = 5;

* controlPoints: stores user-selected Bezier control points.
* maxPoints = 4: we need exactly 4 control points for a **cubic** Bezier curve.
* recursionDepth = 5: depth of the fractal. Each recursion adds more detail.

**3. drawBezier() - Recursive Function**

cpp

CopyEdit

void drawBezier(const vector<Point>& ctrlPoints, int level)

This function draws a cubic Bezier curve **recursively**.

**🔹 Base Case:**

cpp

CopyEdit

if (level == 0) {

glBegin(GL\_LINE\_STRIP);

for (float t = 0; t <= 1.0f; t += 0.01f) {

...

glVertex2f(x, y);

}

glEnd();

return;

}

* Uses the **Bezier curve formula** to plot a smooth curve between the 4 control points.
* B(t)=(1−t)3P0​+3(1−t)2tP1​+3(1−t)t2P2​+t3P3
* Plots points from t = 0 to t = 1 in small steps (0.01).
* Draws it as a line strip using glBegin(GL\_LINE\_STRIP).

**🔹 Recursive Case:**

cpp

CopyEdit

// Split into two sub-curves using De Casteljau's algorithm

The algorithm splits a cubic Bezier into two smaller cubic Bezier curves:

* Left half: first
* Right half: second

Then it calls itself recursively:

cpp

CopyEdit

drawBezier(first, level - 1);

drawBezier(second, level - 1);

This creates the **fractal structure** by drawing more and more curves as the depth increases.

**4. display() - Main Drawing Function**

cpp

CopyEdit

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

* Clears the screen before redrawing.
* If 4 control points are available, it draws the fractal using drawBezier().

cpp

CopyEdit

// Draw control points as red dots

glPointSize(5);

glColor3f(1, 0, 0);

glBegin(GL\_POINTS);

...

* Control points are shown as red dots for reference.

**5. mouse() - Handling Mouse Clicks**

cpp

CopyEdit

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

Handles **left-clicks**:

cpp

CopyEdit

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN && controlPoints.size() < maxPoints) {

...

controlPoints.push\_back({ fx, fy });

glutPostRedisplay();

}

* Gets the mouse (x, y) and converts it to OpenGL coordinates [-1, 1].
* Adds the point to controlPoints.
* Triggers a redraw (glutPostRedisplay()).

Once 4 points are clicked, the curve is drawn automatically in display().

**6. keyboard() - Press 'C' to Clear**

cpp

CopyEdit

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

cpp

CopyEdit

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

controlPoints.clear();

glutPostRedisplay();

}

* Pressing 'c' clears all points and redraws the screen.
* Lets you restart the control point selection.

**7. init() - OpenGL Setup**

cpp

CopyEdit

void init() {

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

gluOrtho2D(-1, 1, -1, 1); // set 2D coordinate system

}

**8. main() - Entry Point**

cpp

CopyEdit

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

Sets up the OpenGL/GLUT window:

* Initializes window size: 600×600
* Sets the drawing function, mouse handler, and keyboard handler
* Starts the GLUT main loop