**2D transformations**

#include <GL/glut.h>

#include <iostream>

#include <vector>

using namespace std;

struct Point {

float x, y;

};

vector<Point> polygon;

float angle = 0; // Rotation angle

float scaleFactor = 1.0f; // Scaling factor

bool isPolygonClosed = false;

int winWidth = 800, winHeight = 600;

Point rotationCenter = {400, 300}; // Rotation center (arbitrary point)

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 drawPolygon() {

glColor3f(0.0, 0.0, 0.0); // black color for polygon

glBegin(GL\_LINE\_LOOP);

for (auto& p : polygon) {

glVertex2i(p.x, p.y);

}

glEnd();

glFlush();

}

// Scaling transformation

void scalePolygon(float sx, float sy) {

for (auto& p : polygon) {

p.x \*= sx;

p.y \*= sy;

}

}

// Rotation transformation about an arbitrary point

void rotatePolygon(float angle) {

for (auto& p : polygon) {

float xNew = cos(angle) \* (p.x - rotationCenter.x) - sin(angle) \* (p.y - rotationCenter.y) + rotationCenter.x;

float yNew = sin(angle) \* (p.x - rotationCenter.x) + cos(angle) \* (p.y - rotationCenter.y) + rotationCenter.y;

p.x = xNew;

p.y = yNew;

}

}

// Reflection transformation about the x-axis

void reflectPolygonX() {

for (auto& p : polygon) {

p.y = -p.y;

}

}

// Reflection transformation about the y-axis

void reflectPolygonY() {

for (auto& p : polygon) {

p.x = -p.x;

}

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

if (isPolygonClosed) {

drawPolygon();

}

glFlush();

}

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({(float)x, (float)y});

int sz = polygon.size();

if (sz > 1) {

glColor3f(0, 0, 0);

glBegin(GL\_LINES);

glVertex2i(polygon[sz - 2].x, polygon[sz - 2].y);

glVertex2i(polygon[sz - 1].x, polygon[sz - 1].y);

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) {

isPolygonClosed = true;

drawPolygon();

}

}

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

switch (key) {

case 's': // Scaling

scalePolygon(scaleFactor, scaleFactor); // Uniform scaling

cout << "Scaling by factor: " << scaleFactor << endl;

break;

case 'r': // Rotation

rotatePolygon(angle \* (3.14159265359f / 180.0f)); // Convert to radians

cout << "Rotating by angle: " << angle << " degrees" << endl;

break;

case 'f': // Reflect about x-axis

reflectPolygonX();

cout << "Reflection about X-axis" << endl;

break;

case 'g': // Reflect about y-axis

reflectPolygonY();

cout << "Reflection about Y-axis" << endl;

break;

case 'c':

polygon.clear();

isPolygonClosed = false;

glClear(GL\_COLOR\_BUFFER\_BIT);

break;

case '+':

scaleFactor += 0.1f;

cout << "Scale factor: " << scaleFactor << endl;

break;

case '-':

scaleFactor -= 0.1f;

cout << "Scale factor: " << scaleFactor << endl;

break;

case 'a': // Increase rotation angle

angle += 5.0f;

cout << "Rotation angle: " << angle << endl;

break;

case 'z': // Decrease rotation angle

angle -= 5.0f;

cout << "Rotation angle: " << angle << endl;

break;

default:

break;

}

glClear(GL\_COLOR\_BUFFER\_BIT);

drawPolygon();

glFlush();

}

void init() {

glClearColor(1, 1, 1, 1); // Set background color to white

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0, winWidth, 0, winHeight); // Set 2D orthogonal projection

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(winWidth, winHeight);

glutCreateWindow("2D Transformations");

init();

glutDisplayFunc(display);

glutMouseFunc(mouse);

glutKeyboardFunc(keyboard);

glutMainLoop();

return 0;

}

**Circle Drawing**

#include <GL/glut.h>

#include <iostream>

#include <cmath>

using namespace std;

int centerX = 400, centerY = 300; // Center of the circle (origin)

int radius = 100; // Radius of the circle

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

glColor3f(r, g, b); // Set pixel color (red, green, blue)

glBegin(GL\_POINTS); // Plotting point

glVertex2i(x, y); // Set pixel at (x, y)

glEnd();

glFlush();

}

void bresenhamCircle(int cx, int cy, int r) {

int x = 0, y = r;

int p = 3 - 2 \* r;

// Plot points in all octants (quadrants) of the circle

while (x <= y) {

// Plot the 8 symmetric points

setPixel(cx + x, cy + y, 1.0, 0.0, 0.0); // Quadrant I

setPixel(cx - x, cy + y, 0.0, 1.0, 0.0); // Quadrant II

setPixel(cx + x, cy - y, 0.0, 0.0, 1.0); // Quadrant IV

setPixel(cx - x, cy - y, 1.0, 1.0, 0.0); // Quadrant III

setPixel(cx + y, cy + x, 1.0, 0.0, 1.0); // Quadrant I

setPixel(cx - y, cy + x, 0.0, 1.0, 1.0); // Quadrant II

setPixel(cx + y, cy - x, 1.0, 1.0, 1.0); // Quadrant IV

setPixel(cx - y, cy - x, 0.5, 0.5, 0.5); // Quadrant III

if (p < 0) {

p = p + 4 \* x + 6;

} else {

p = p + 4 \* (x - y) + 10;

y--;

}

x++;

}

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

bresenhamCircle(centerX, centerY, radius);

glFlush();

}

void init() {

glClearColor(1, 1, 1, 1); // Set background color to white

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0, 800, 0, 600); // Set up a 2D orthogonal projection

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(800, 600); // Set window size

glutCreateWindow("Bresenham Circle Algorithm");

init();

glutDisplayFunc(display);

glutMainLoop();

return 0;

}