Presentasi computer grafik

//rotasimatriks

#include <iostream>

#include <cmath>

void rotate(double& x, double& y, double angle) {

    double radians = angle \* M\_PI / 180;

    double new\_x = x \* cos(radians) - y \* sin(radians);

    double new\_y = x \* sin(radians) + y \* cos(radians);

    x = new\_x;

    y = new\_y;

}

int main() {

    double x = 1.3;

    double y = 4.9;

    double rotation\_angle = 45.0;  // Sudut rotasi dalam derajat

    std::cout << "Koordinat awal: (" << x << ", " << y << ")\n";

    rotate(x, y, rotation\_angle);

    std::cout << "Koordinat setelah rotasi: (" << x << ", " << y << ")\n";

    return 0;

}

Fungsi rotate menerima tiga parameter: x dan y adalah koordinat titik yang akan dirotasi, dan angle adalah sudut rotasi dalam derajat. Fungsi mengubah sudut rotasi dari derajat menjadi radian, karena fungsi trigonometri di C++ umumnya menggunakan radian. Kemudian, koordinat titik (x, y) dirotasi dengan menggunakan matriks rotasi dalam dua dimensi. Hasil rotasi disimpan kembali ke x dan y. Program utama (main) menunjukkan contoh penggunaan rotate untuk merotasi titik (x, y) sebesar rotation\_angle derajat. Hasil rotasi ditampilkan.

//DDA

#include <iostream>

#include <cmath>

void drawLineDDA(int x1, int y1, int x2, int y2) {

    int dx = x2 - x1;

    int dy = y2 - y1;

    int steps = abs(dx) > abs(dy) ? abs(dx) : abs(dy);

    float xIncrement = static\_cast<float>(dx) / steps;

    float yIncrement = static\_cast<float>(dy) / steps;

    float x = x1;

    float y = y1;

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

        std::cout << "Titik (" << round(x) << ", " << round(y) << ")\n";

        x += xIncrement;

        y += yIncrement;

    }

}

int main() {

    int x1 = 1, y1 = 3;

    int x2 = 4, y2 = 9;

    std::cout << "Titik awal: (" << x1 << ", " << y1 << ")\n";

    std::cout << "Titik akhir: (" << x2 << ", " << y2 << ")\n";

    drawLineDDA(x1, y1, x2, y2);

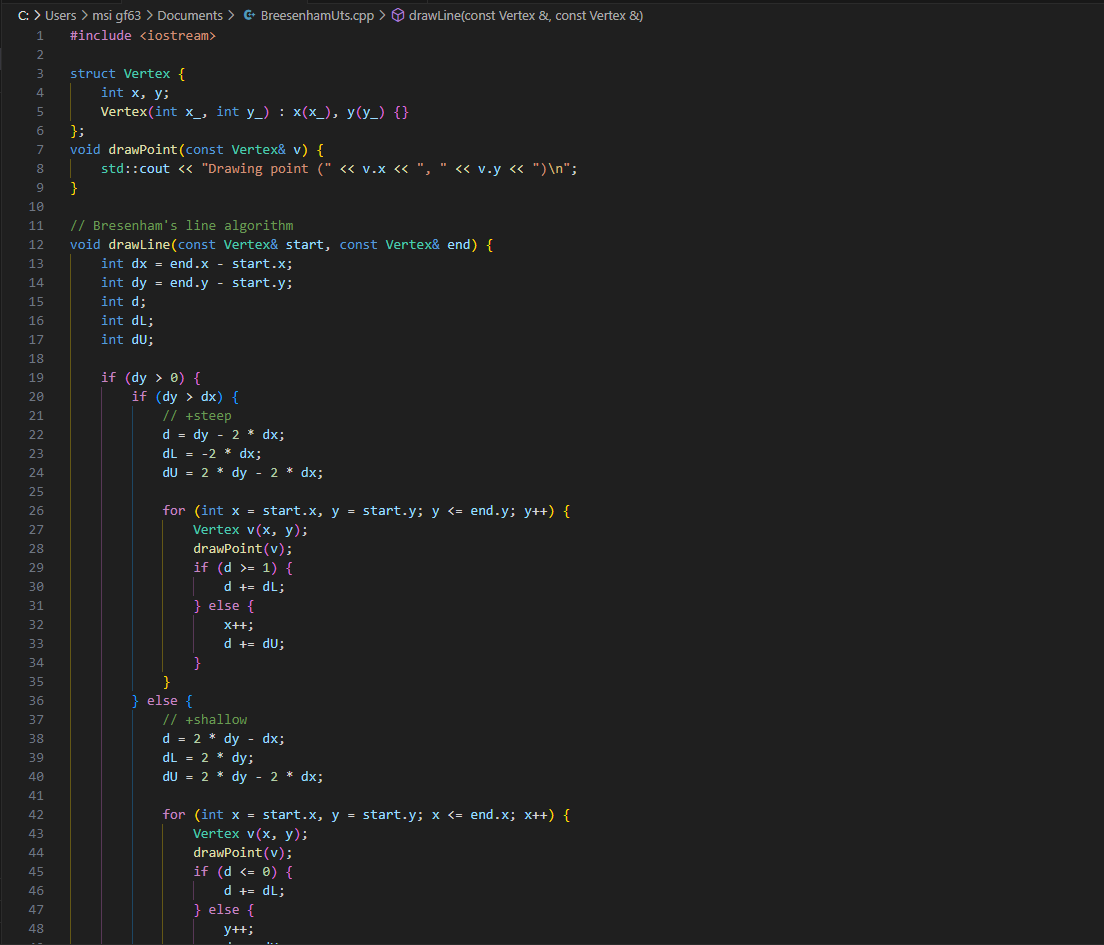
    return 0;

}

Penjelasan kode di atas:

drawLineDDA adalah fungsi yang mengimplementasikan algoritma DDA untuk menggambar garis antara dua titik (x1, y1) dan (x2, y2). Kita menghitung selisih antara koordinat x dan y titik awal dan titik akhir, yaitu dx dan dy. Selanjutnya, kita menentukan jumlah langkah yang dibutuhkan (steps) berdasarkan selisih yang lebih besar, abs(dx) atau abs(dy). Kita menghitung increment untuk x dan y, yaitu xIncrement dan yIncrement, yang dibutuhkan untuk menggeser titik awal ke titik akhir dalam jumlah langkah yang sesuai. Kita inisialisasi koordinat titik awal (x, y) dan kemudian menampilkan setiap titik dalam perulangan. Dalam perulangan, kita menampilkan setiap titik yang dihasilkan dengan menggunakan std::cout. Nilai x dan y diperbarui dengan increment setiap kali perulangan dilakukan. Program utama (main) menunjukkan contoh penggunaan drawLineDDA untuk menggambar garis antara dua titik (x1, y1) dan (x2, y2). Anda dapat menyesuaikan nilai x1, y1, x2, dan y2 sesuai dengan kebutuhan Anda.

//Breseenham



Penjelasan pada coding di atas yaitu :

Bresenham’s line algorithm adalah algoritma yang digunakan untuk menggambar garis pada layar komputer dengan cara yang efisien. Algoritma ini menghitung koordinat titik-titik yang diperlukan untuk menggambar garis antara dua titik A(x1, y1) dan B(x2, y2) pada layar komputer dengan piksel berkoordinat integer. Algoritma ini menghindari perkalian dan penjumlahan bilangan pecahan untuk menghitung mx + c, dan kemudian menghitung nilai bulat dari (mx + c) pada setiap langkah. Algoritma Bresenham bergerak melintasi sumbu x dalam interval satuan. Nilai x selalu ditingkatkan sebesar 1, dan kemudian dipilih nilai y berikutnya, apakah perlu naik ke y+1 atau tetap di y. Dengan kata lain, dari posisi manapun (Xk, Yk), kita perlu memilih antara (Xk+1, Yk) dan (Xk+1, Yk+1). Kita ingin memilih nilai y (di antara Yk+1 dan Yk) yang sesuai dengan titik yang lebih dekat dengan garis asli.

