**NAME:- SURWADE TRISHARAN RAJESH**

**ROLL NO:- 47**

**CLASS -MSC(CS)-IInd Year**

**5. Windowing and clipping algorithm (Point, line and polygon clipping)**

#include <graphics.h>

#include <iostream.h>

#include <conio.h>

#include <math.h>

#include <dos.h>

class Rectangle {

public:

int xleft, xright, ytop, ybottom;

Rectangle(int xl, int xr, int yt, int yb)

: xleft(xl), xright(xr), ytop(yt), ybottom(yb) {}

void draw() const {

rectangle(xleft, ytop, xright, ybottom);

}

int getCode(int x, int y) const {

int code = 0;

if (x < xleft) code |= 1; // Left

if (x > xright) code |= 2; // Right

if (y < ytop) code |= 4; // Top

if (y > ybottom) code |= 8; // Bottom

return code;

}

};

class Line {

private:

float x1, y1, x2, y2;

public:

Line(float xStart, float yStart, float xEnd, float yEnd)

: x1(xStart), y1(yStart), x2(xEnd), y2(yEnd) {}

void draw(int color = RED) const {

setcolor(color);

line(x1, y1, x2, y2);

}

void clipAndDraw(const Rectangle& clipRect, int color = GREEN) {

int code1 = clipRect.getCode(x1, y1);

int code2 = clipRect.getCode(x2, y2);

int accept = 0; // Replacing `bool accept = false` with `int accept = 0`

while (1) {

if ((code1 == 0) && (code2 == 0)) {

accept = 1; // Setting `accept` to 1 instead of `true`

break;

} else if (code1 & code2) {

break;

} else {

int codeOut;

float x, y;

if (code1 != 0)

codeOut = code1;

else

codeOut = code2;

if (codeOut & 8) { // Point is above the window

x = x1 + (x2 - x1) \* (clipRect.ybottom - y1) / (y2 - y1);

y = clipRect.ybottom;

} else if (codeOut & 4) { // Point is below the window

x = x1 + (x2 - x1) \* (clipRect.ytop - y1) / (y2 - y1);

y = clipRect.ytop;

} else if (codeOut & 2) { // Point is to the right of window

y = y1 + (y2 - y1) \* (clipRect.xright - x1) / (x2 - x1);

x = clipRect.xright;

} else if (codeOut & 1) { // Point is to the left of window

y = y1 + (y2 - y1) \* (clipRect.xleft - x1) / (x2 - x1);

x = clipRect.xleft;

}

if (codeOut == code1) {

x1 = x;

y1 = y;

code1 = clipRect.getCode(x1, y1);

} else {

x2 = x;

y2 = y;

code2 = clipRect.getCode(x2, y2);

}

}

}

if (accept) { // Checking if `accept` is 1 instead of `true`

setcolor(color);

line(x1, y1, x2, y2);

}

}

};

int main() {

int gd = DETECT, gm;

initgraph(&gd, &gm, "C:\\Turboc3\\BGI");

Rectangle clipRect(100, 540, 100, 400);

clipRect.draw();

int numLines;

cout << "Enter number of lines: ";

cin >> numLines;

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

float x1, y1, x2, y2;

cout << "Enter coordinates for line " << i + 1 << " (x1 y1 x2 y2): ";

cin >> x1 >> y1 >> x2 >> y2;

Line line(x1, y1, x2, y2);

// Draw initial line in red

line.draw(RED);

// Apply line clipping and draw clipped line in green

delay(1000); // Delay for visual effect

cleardevice();

clipRect.draw();

line.clipAndDraw(clipRect, GREEN);

}

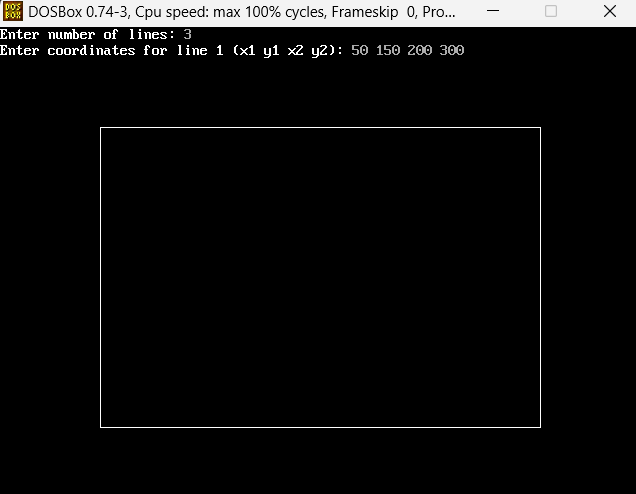
getch();

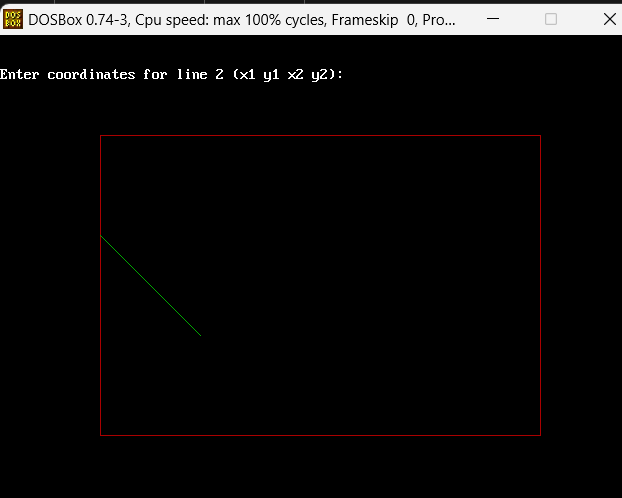
closegraph();

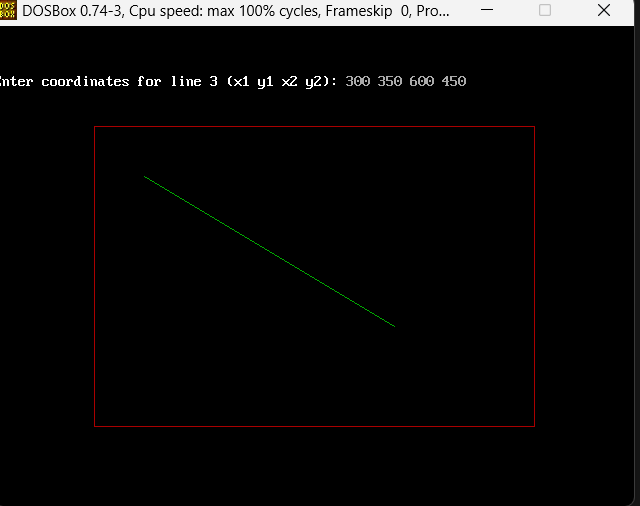
return 0;

}

**OUTPUT:**





****

**Polygaon Clippling Algorithm**

#include <iostream.h>

#include <graphics.h>

#include <math.h>

#include <dos.h>

#include <conio.h>

class PolygonClipper {

private:

int xleft, xright, ytop, ybottom;

int vertices[10][2]; // Array to hold polygon vertices

int n; // Number of vertices

// Function to compute the code for a point

int computeCode(int x, int y) {

int code = 0;

if (x < xleft) code |= 1; // Left

else if (x > xright) code |= 2; // Right

if (y < ytop) code |= 4; // Top

else if (y > ybottom) code |= 8; // Bottom

return code;

}

// Function to clip a line segment (x1, y1) to (x2, y2)

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

int code1 = computeCode(x1, y1);

int code2 = computeCode(x2, y2);

int accept = 0;

while (1) {

if ((code1 == 0) && (code2 == 0)) { // Both points inside

accept = 1;

break;

} else if (code1 & code2) { // Both points outside

break;

} else {

int codeOut;

int x, y;

// At least one endpoint is outside

if (code1 != 0) codeOut = code1;

else codeOut = code2;

if (codeOut & 1) { // Left

x = xleft;

y = y1 + (y2 - y1) \* (xleft - x1) / (x2 - x1);

} else if (codeOut & 2) { // Right

x = xright;

y = y1 + (y2 - y1) \* (xright - x1) / (x2 - x1);

} else if (codeOut & 4) { // Top

y = ytop;

x = x1 + (x2 - x1) \* (ytop - y1) / (y2 - y1);

} else if (codeOut & 8) { // Bottom

y = ybottom;

x = x1 + (x2 - x1) \* (ybottom - y1) / (y2 - y1);

}

// Replace the outside point with the intersection point

if (codeOut == code1) {

x1 = x;

y1 = y;

code1 = computeCode(x1, y1);

} else {

x2 = x;

y2 = y;

code2 = computeCode(x2, y2);

}

}

}

if (accept) {

setcolor(GREEN);

line(x1, y1, x2, y2);

delay(500); // Adding a delay to observe the drawing process

}

}

public:

// Constructor to initialize the clipping window and polygon

PolygonClipper(int left, int right, int top, int bottom)

: xleft(left), xright(right), ytop(top), ybottom(bottom) {

n = 0;

}

// Method to input the polygon vertices

void inputPolygon() {

cout << "Enter the number of vertices: ";

cin >> n;

cout << "Enter the coordinates of the vertices:\n";

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

cout << "Vertex " << (i + 1) << " (x y): ";

cin >> vertices[i][0] >> vertices[i][1];

}

// Connect the last vertex to the first to close the polygon

vertices[n][0] = vertices[0][0];

vertices[n][1] = vertices[0][1];

}

// Method to draw the polygon before clipping

void drawPolygon() {

setcolor(RED);

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

line(vertices[i][0], vertices[i][1], vertices[i + 1][0], vertices[i + 1][1]);

delay(500); // Delay for visibility

}

}

// Method to clip and draw the polygon within the clipping window

void clipPolygon() {

setcolor(BLUE);

rectangle(xleft, ytop, xright, ybottom);

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

clipLine(vertices[i][0], vertices[i][1], vertices[i + 1][0], vertices[i + 1][1]);

}

}

};

int main() {

int gd = DETECT, gm;

initgraph(&gd, &gm, "C:\\Turboc3\\BGI");

PolygonClipper clipper(100, 540, 100, 400);

clipper.inputPolygon();

setbkcolor(WHITE);

outtextxy(250, 60, "BEFORE POLYGON CLIPPING");

clipper.drawPolygon();

getch();

cleardevice();

outtextxy(250, 60, "AFTER POLYGON CLIPPING");

clipper.clipPolygon();

getch();

closegraph();

return 0;

}

**OUTPUT:**

