B.Sc. Computer Science 3.2

ICS 2311 Computer Graphics

Submitted:  **February 23rd, 2025**

SCT211-0848/2018

jANY MUONG

ICS 2311: Computer Graphics - Assignment One

Assignment – write up:

mail to: Jkuatnotes8@gmail.com

subject: Ass1: Muong William Jany: SCT211-0848/2018

Background Context:

**Summary of image file formats, and line drawing algorithms using** OpenGL/GLUT/GLEW.

Part A: research and explain image formats, their abbreviations, history, whether they are raster or vector, and their typical applications.

Part B: implement six line-drawing algorithms using OpenGL/GLUT/GLEW and compute the points between the **given starting** and **ending points**.

part a: Image File Formats

Read on various image formats such as Ai, wmf, Cmx, cgm,svg ,odg, eps , dxf , bmp, jpeg ,Gif ,Tiff,PICT and png

* Explain what the abbreviation stand for and some history on the format.
* State whether each of the graphic format above is raster or vector.
* Briefly explain a typical application or area of usage of ν ν each of the format

I have listed out the given image formats and given brief descriptiosn of them in a table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Format | Abbreviation  Stands for | History | Raster/Vector | Typical Application |
| **Ai** | Adobe Illustrator Artwork | Developed by Adobe Systems for vector graphics. | Vector | Professional graphic design, logos, and illustrations. |
| **WMF** | Windows Metafile | Developed by Microsoft for Windows applications. | Vector | Clip art, logos, and vector graphics in Windows. |
| **CMx** | CorelDRAW Metafile Exchange | Developed by Corel for CorelDRAW application. | Vector | Vector graphics in CorelDRAW. |
| **CGM** | Computer Graphics Metafile | An ISO standard for 2D vector graphics. | Vector | Technical illustrations, engineering drawings. |
| **SVG** | Scalable Vector Graphics | Developed by W3C for web-based vector graphics in 1999. | Vector | Web graphics, icons, and animations. |
| **ODG** | OpenDocument Graphics | Part of the OpenDocument standard by OASIS. | Vector | Office applications like LibreOffice. |
| **EPS** | Encapsulated PostScript | Developed by Adobe for vector graphics. | Vector | Print media, professional publishing. |
| **DXF** | Drawing Exchange Format | Developed by Autodesk for CAD software. | Vector | CAD drawings and 3D modeling. |
| **BMP** | Bitmap Image File | Developed by Microsoft for Windows. Introduced with Windows 3.0 in 1990 | Raster | Simple image storage, icons. |
| **JPEG** | Joint Photographic Experts Group | Developed in 1992 for efficient image compression. Lossy. | Raster | Digital photography, web images. |
| **GIF** | Graphics Interchange Format | Developed by CompuServe for web graphics. | Raster | Web animations, simple graphics. |
| **TIFF** | Tagged Image File Format | Developed by **Aldus** for high-quality images. Developed in 1986. | Raster | Professional photography, printing. |
| **PICT** | Picture File | Developed by **Apple** for **Macintosh**. | Raster/Vector | Legacy Mac graphics. |
| **PNG** | Portable Network Graphics | Developed as a replacement for GIF. Lossless. | Raster | Web graphics, lossless compression. |

part b:

Calculate the points between the stated starting point and ending point and plot the line using the below algorithms ( show the calculations and attach the OPENGL code as zipped files ).

No Handwritten work.

1. Xiaolin Wu's line algorithm ((1,1) and(3, 5).
2. Gupta-Sproull algorithm (-2, 3) and (1, 4).
3. Midpoint Algorithm (- 3, 4) and (5, - 2).
4. Bresenham’s Line-Drawing Algorithm (1,5) and (2,8).
5. Midpoint Line-Drawing Algorithm (0,2) and (−1,4).
6. DDA line drawing Algorithm (5,2) and(10, 3).

pset 1: Xiaolin Wu's Line Algorithm

How the Algorithm Works:

- an anti-aliasing line drawing algorithm that produces smooth lines

- uses intensity values for pixels near the line to reduce the "jagged" appearance

Steps:

1. compute the slope of the line

2. handle steep and shallow lines differently

3. for each pixel:

- calculate the ideal line position

- set pixel intensities based on distance from ideal line

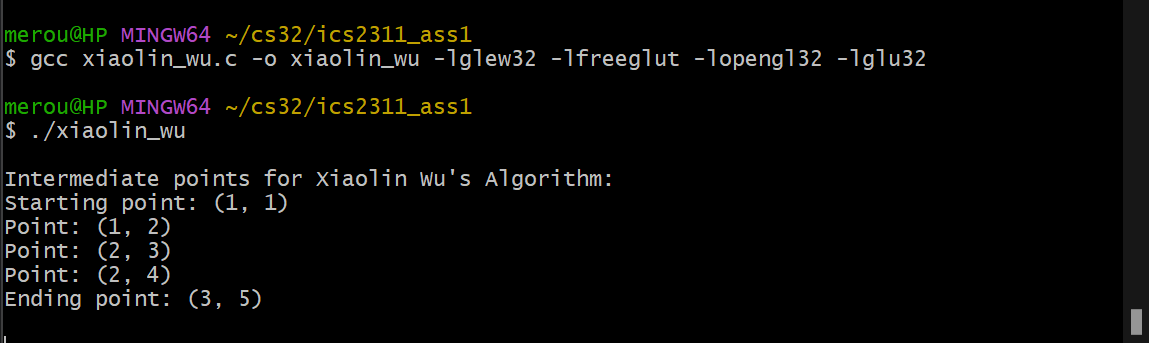
- use linear interpolation for anti-aliasing

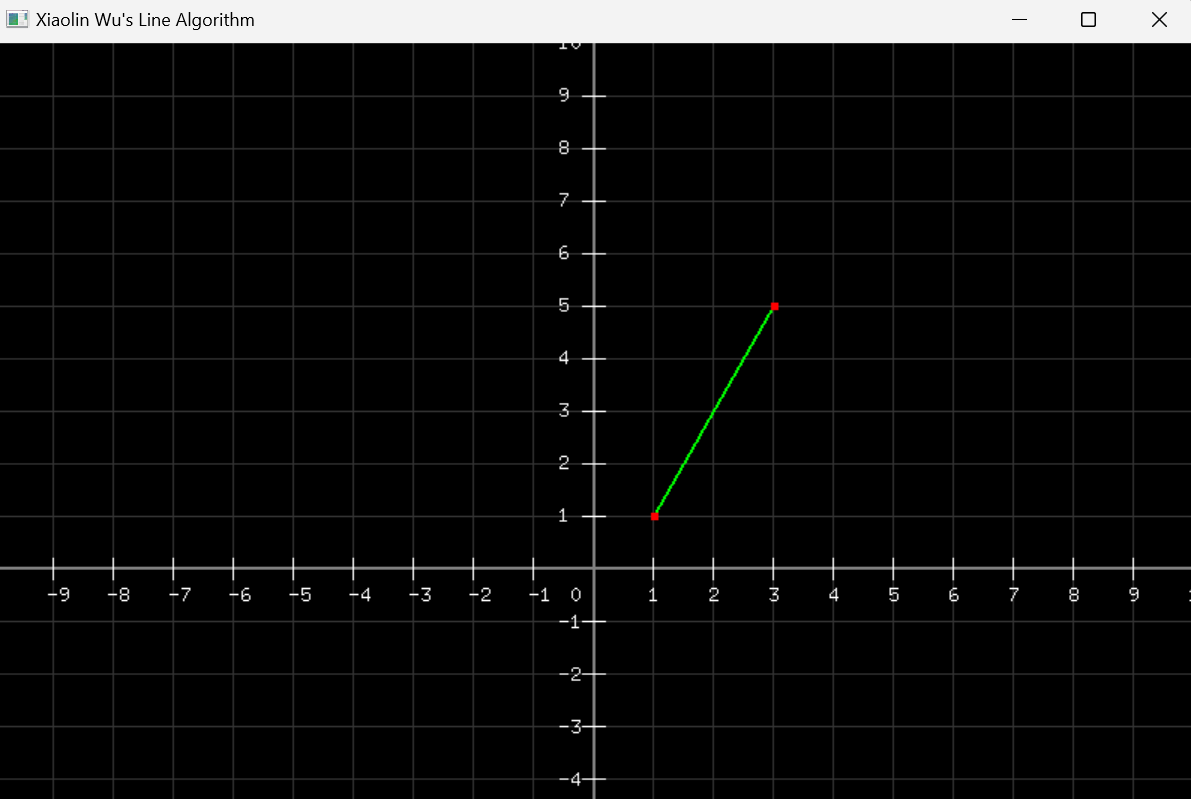
- plot pixels with appropriate intensity values

4. points: (1,1) to (3,5)

OpenGL:

gcc xiaolin\_wu.c -o xiaolin\_wu -lglew32 -lfreeglut -lopengl32 -lglu32





C - Code

// SCT211-0848/2018 - Jany Muong

#include <GL/glew.h>

#include <GL/freeglut.h>

#include <GL/glut.h>

#include <stdio.h>

#include <math.h>

#include <string.h>

// function to draw the coordinate system (axes, grid, and labels)

void drawCoordinateSystem() {

    // draw main axes

    glColor3f(0.5, 0.5, 0.5);  // gray color for axes

    glLineWidth(2.0);

    glBegin(GL\_LINES);

        // x-axis

        glVertex2f(-10.0, 0.0);

        glVertex2f(10.0, 0.0);

        // y-axis

        glVertex2f(0.0, -10.0);

        glVertex2f(0.0, 10.0);

    glEnd();

    // draw grid lines

    glLineWidth(1.0);

    glColor3f(0.2, 0.2, 0.2);  // grid color

    glBegin(GL\_LINES);

    for (int i = -10; i <= 10; i++) {

        if (i == 0) continue; // skip the axes

        // vertical lines

        glVertex2f(i, -10.0);

        glVertex2f(i, 10.0);

        // horizontal lines

        glVertex2f(-10.0, i);

        glVertex2f(10.0, i);

    }

    glEnd();

    // draw tick marks and numbers

    glColor3f(1.0, 1.0, 1.0);  // white color for numbers

    for (int i = -10; i <= 10; i++) {

        if (i == 0) continue; // Skip origin

        // draw tick marks

        glBegin(GL\_LINES);

            // x-axis ticks

            glVertex2f(i, -0.2);

            glVertex2f(i, 0.2);

            // y-axis ticks

            glVertex2f(-0.2, i);

            glVertex2f(0.2, i);

        glEnd();

        // draw numbers

        char str[10];

        // x-axis numbers

        glRasterPos2f(i - 0.1, -0.6);

        sprintf(str, "%d", i);

        for (char\* c = str; \*c != '\0'; c++) {

            glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13, \*c);

        }

        // y-axis numbers

        if (i != 0) {  // Skip 0 for y-axis

            glRasterPos2f(-0.6, i - 0.1);

            sprintf(str, "%d", i);

            for (char\* c = str; \*c != '\0'; c++) {

                glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13, \*c);

            }

        }

    }

    // draw origin "0"

    glRasterPos2f(-0.4, -0.6);

    glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13, '0');

}

// function to compute intermediate points and print them

void computeAndPrintPoints(int x0, int y0, int x1, int y1) {

    int dx = x1 - x0, dy = y1 - y0;

    int steep = abs(dy) > abs(dx);

    if (steep) {

        int temp = x0; x0 = y0; y0 = temp;

        temp = x1; x1 = y1; y1 = temp;

    }

    if (x0 > x1) {

        int temp = x0; x0 = x1; x1 = temp;

        temp = y0; y0 = y1; y1 = temp;

    }

    dx = x1 - x0;

    dy = y1 - y0;

    float gradient = dx == 0 ? 1 : (float)dy / dx;

    float y = y0 + gradient;

    printf("\nIntermediate points for Xiaolin Wu's Algorithm:\n");

    printf("Starting point: (%d, %d)\n", x0, y0);

    for (int x = x0 + 1; x < x1; x++) {

        printf("Point: (%d, %d)\n", steep ? (int)y : x, steep ? x : (int)y);

        y += gradient;

    }

    // Correct the end point printing

    printf("Ending point: (%d, %d)\n\n", steep ? y1 : x1, steep ? x1 : y1);

}

void display() {

    glClear(GL\_COLOR\_BUFFER\_BIT);

    // draw coordinate system

    drawCoordinateSystem();

    // draw the line in green

    glColor3f(0.0, 1.0, 0.0);  // green color for the line

    glLineWidth(2.0);

    glBegin(GL\_LINES);

        glVertex2i(1, 1);  // start point

        glVertex2i(3, 5);  // end point

    glEnd();

    // draw endpoints as points

    glColor3f(1.0, 0.0, 0.0);  // red color for points

    glPointSize(5.0);

    glBegin(GL\_POINTS);

        glVertex2i(1, 1);  // start point

        glVertex2i(3, 5);  // end point

    glEnd();

    glFlush();

}

// initialize OpenGL

void init() {

    glClearColor(0.0, 0.0, 0.0, 1.0);  // background

    glMatrixMode(GL\_PROJECTION);

    glLoadIdentity();

    gluOrtho2D(-10, 10, -10, 10);  // cartesian plane calibration

}

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

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowSize(800, 800);

    glutInitWindowPosition(100, 100);

    glutCreateWindow("Xiaolin Wu's Line Algorithm");

    glewInit();

    init();

    // compute and print intermediate points

    computeAndPrintPoints(1, 1, 3, 5);

    glutDisplayFunc(display);

    glutMainLoop();

    return 0;

}

PSET 2. Gupta-Sproull Algorithm

About:

- anti-aliased line drawing algorithm with intensity calculation

- uses perpendicular distance to determine pixel intensity

- creates relatively smooth lines compared to basic algorithms

**Steps**:

1. calculate line parameters (dx, dy)

2. determine major and minor axes

3. for each pixel along major axis:

- calculate perpendicular distance to ideal line

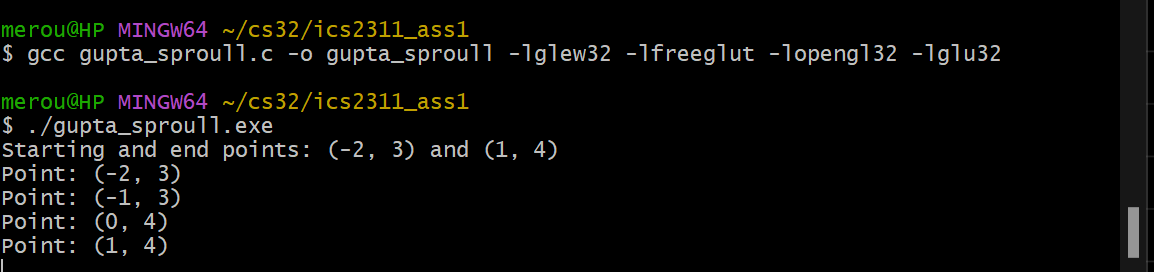
- compute intensity based on distance

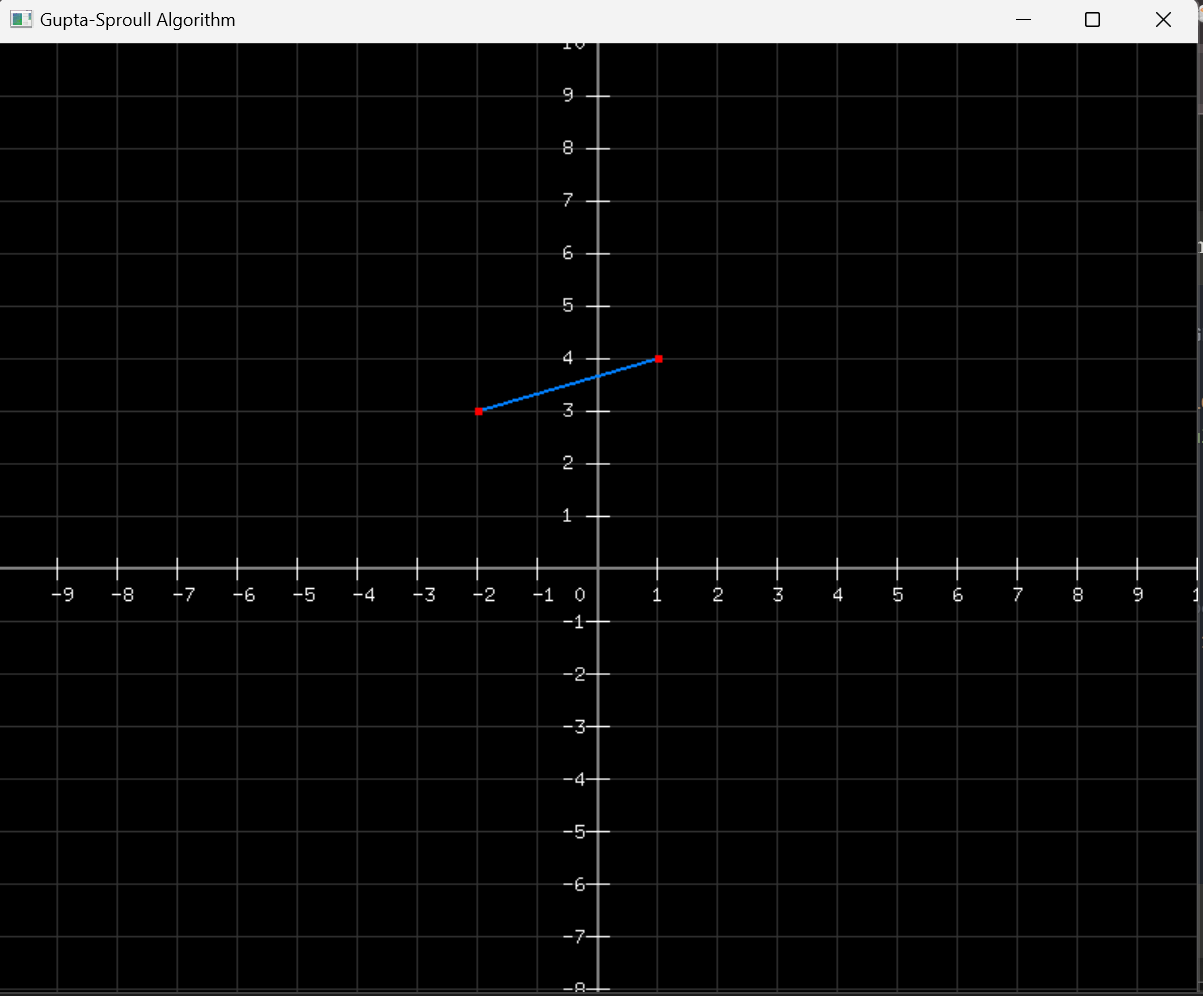
- plot pixel with calculated intensity

- update error terms and continue

OpenGL

gcc gupta\_sproull.c -o gupta\_sproull -lglew32 -lfreeglut -lopengl32 -lglu32

****



C Code:

// SCT211-0848/2018 - Jany Muong

// gupta\_sproull.c - draw points from (-2,3) to (1,4)

#include <GL/glew.h>

#include <GL/freeglut.h>

#include <GL/glut.h>

#include <stdio.h>

#include <math.h>

#include <string.h>

void drawCoordinateSystem() {

    // draw main axes

    glColor3f(0.5, 0.5, 0.5);  // Gray color for axes

    glLineWidth(2.0);

    glBegin(GL\_LINES);

        // X-axis

        glVertex2f(-10.0, 0.0);

        glVertex2f(10.0, 0.0);

        // Y-axis

        glVertex2f(0.0, -10.0);

        glVertex2f(0.0, 10.0);

    glEnd();

    // grid lines

    glLineWidth(1.0);

    glColor3f(0.2, 0.2, 0.2);  // Darker gray for grid

    glBegin(GL\_LINES);

    for(int i = -10; i <= 10; i++) {

        if(i == 0) continue; // skip the axes

        // Vertical lines

        glVertex2f(i, -10.0);

        glVertex2f(i, 10.0);

        // Horizontal lines

        glVertex2f(-10.0, i);

        glVertex2f(10.0, i);

    }

    glEnd();

    // draw tick marks and numbers

    glColor3f(1.0, 1.0, 1.0);  // White color for numbers

    for(int i = -10; i <= 10; i++) {

        if(i == 0) continue; // Skip origin

        // draw tick marks

        glBegin(GL\_LINES);

            // X-axis ticks

            glVertex2f(i, -0.2);

            glVertex2f(i, 0.2);

            // Y-axis ticks

            glVertex2f(-0.2, i);

            glVertex2f(0.2, i);

        glEnd();

        // draw numbers

        char str[10];

        // x-axis numbers

        glRasterPos2f(i - 0.1, -0.6);

        sprintf(str, "%d", i);

        for(char\* c = str; \*c != '\0'; c++) {

            glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13, \*c);

        }

        // y-axis numbers

        if(i != 0) {  // Skip 0 for y-axis

            glRasterPos2f(-0.6, i - 0.1);

            sprintf(str, "%d", i);

            for(char\* c = str; \*c != '\0'; c++) {

                glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13, \*c);

            }

        }

    }

    // draw origin "0"

    glRasterPos2f(-0.4, -0.6);

    glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13, '0');

}

void computeAndPrintPoints(int x0, int y0, int x1, int y1) {

    int dx = abs(x1 - x0), dy = abs(y1 - y0);

    int sx = x0 < x1 ? 1 : -1, sy = y0 < y1 ? 1 : -1;

    int err = dx - dy;

    // printf("\nIntermediate points for Gupta-Sproull Algorithm:\n");

    printf("Starting and end points: (%d, %d) and (%d, %d)\n", x0, y0, x1, y1);

    while (1) {

        printf("Point: (%d, %d)\n", x0, y0);

        if (x0 == x1 && y0 == y1) break;

        int e2 = 2 \* err;

        if (e2 > -dy) { err -= dy; x0 += sx; }

        if (e2 < dx) { err += dx; y0 += sy; }

    }

}

void display() {

    glClear(GL\_COLOR\_BUFFER\_BIT);

    // draw coordinate system

    drawCoordinateSystem();

    // draw the line in blue

    glColor3f(0.0, 0.5, 1.0);

    glLineWidth(2.0);  // line width

    glBegin(GL\_LINES);

        glVertex2i(-2, 3); // start point

        glVertex2i(1, 4);  // end point

    glEnd();

    // draw endpoints as points

    glColor3f(1.0, 0.0, 0.0);  // Red color for points

    glPointSize(5.0);

    glBegin(GL\_POINTS);

        glVertex2i(-2, 3); // start point

        glVertex2i(1, 4);  // end point

    glEnd();

    glFlush();

}

void init() {

    glClearColor(0.0, 0.0, 0.0, 1.0);

    glMatrixMode(GL\_PROJECTION);

    glLoadIdentity();

    gluOrtho2D(-10, 10, -10, 10);

}

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

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowSize(800, 800);

    glutInitWindowPosition(100, 100);

    glutCreateWindow("Gupta-Sproull Algorithm");

    glewInit();

    init();

    // compute and print points before entering the main loop

    computeAndPrintPoints(-2, 3, 1, 4);

    glutDisplayFunc(display);

    glutMainLoop();

    return 0;

}

Pset 3. Midpoint Algorithm

Description:

- uses midpoint decision criterion to determine pixel placement

- efficient integer-based calculations

- good for basic line drawing needs

Steps:

1. calculate initial decision parameter (d)

2. for each x step:

- if d < 0, select pixel at (x+1, y)

- if d ≥ 0, select pixel at (x+1, y+1)

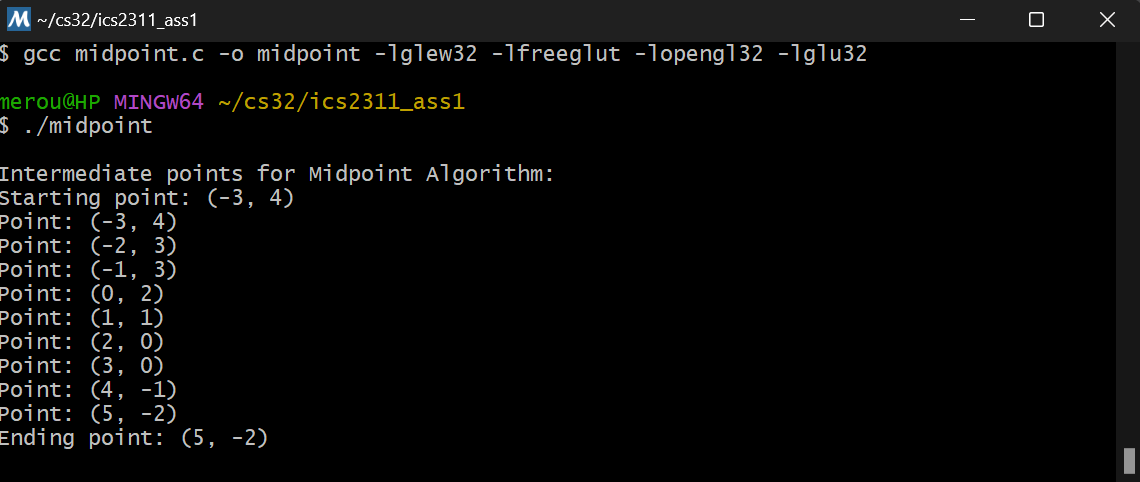
- update decision parameter

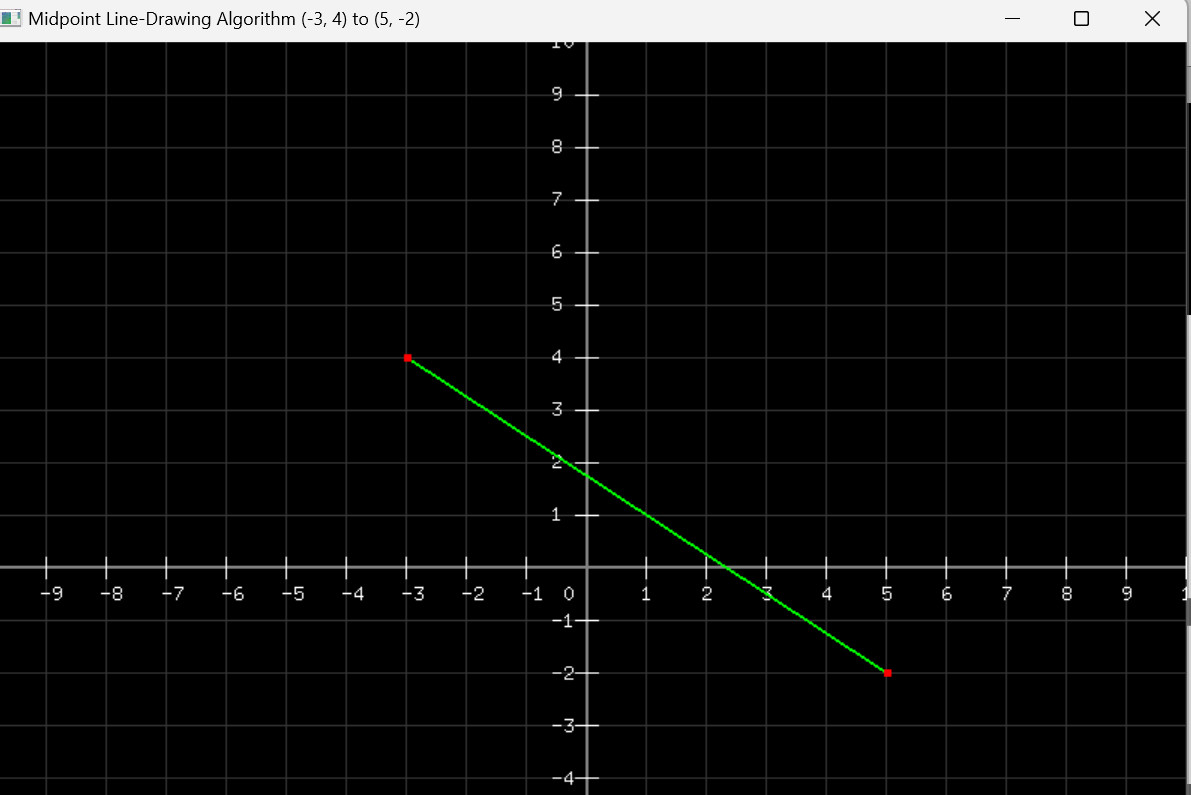
- plot selected pixel

3. points (- 3, 4) and (5, - 2).

OpenGL:

gcc midpoint.c -o midpoint -lglew32 -lfreeglut -lopengl32 -lglu32





C Code:

// SCT211-0848/2018 - Jany Muong

// midpoint.c

#include <GL/glew.h>

#include <GL/freeglut.h>

#include <GL/glut.h>

#include <stdio.h>

#include <math.h>

#include <string.h>

// function to draw the coordinate system (axes, grid, and labels)

void drawCoordinateSystem() {

    // main axes

    glColor3f(0.5, 0.5, 0.5);  // gray color for axes

    glLineWidth(2.0);

    glBegin(GL\_LINES);

        // x-axis

        glVertex2f(-10.0, 0.0);

        glVertex2f(10.0, 0.0);

        // y-axis

        glVertex2f(0.0, -10.0);

        glVertex2f(0.0, 10.0);

    glEnd();

    // draw grid lines

    glLineWidth(1.0);

    glColor3f(0.2, 0.2, 0.2);  // grid color

    glBegin(GL\_LINES);

    for (int i = -10; i <= 10; i++) {

        if (i == 0) continue; // skip the axes

        // vertical lines

        glVertex2f(i, -10.0);

        glVertex2f(i, 10.0);

        // Horizontal lines

        glVertex2f(-10.0, i);

        glVertex2f(10.0, i);

    }

    glEnd();

    // draw tick marks and numbers

    glColor3f(1.0, 1.0, 1.0);  // White color for numbers

    for (int i = -10; i <= 10; i++) {

        if (i == 0) continue; // Skip origin

        // draw tick marks

        glBegin(GL\_LINES);

            // x-axis ticks

            glVertex2f(i, -0.2);

            glVertex2f(i, 0.2);

            // y-axis ticks

            glVertex2f(-0.2, i);

            glVertex2f(0.2, i);

        glEnd();

        // draw numbers

        char str[10];

        // x-axis numbers

        glRasterPos2f(i - 0.1, -0.6);

        sprintf(str, "%d", i);

        for (char\* c = str; \*c != '\0'; c++) {

            glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13, \*c);

        }

        // y-axis numbers

        if (i != 0) {  // Skip 0 for y-axis

            glRasterPos2f(-0.6, i - 0.1);

            sprintf(str, "%d", i);

            for (char\* c = str; \*c != '\0'; c++) {

                glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13, \*c);

            }

        }

    }

    // draw origin "0"

    glRasterPos2f(-0.4, -0.6);

    glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13, '0');

}

// function to compute intermediate points using midpoint algorithm

void computeAndPrintPoints(int x0, int y0, int x1, int y1) {

    int dx = abs(x1 - x0);

    int dy = abs(y1 - y0);

    int sx = (x0 < x1) ? 1 : -1;

    int sy = (y0 < y1) ? 1 : -1;

    int err = dx - dy;

    printf("\nIntermediate points for Midpoint Algorithm:\n");

    printf("Starting point: (%d, %d)\n", x0, y0);

    while (1) {

        printf("Point: (%d, %d)\n", x0, y0);

        if (x0 == x1 && y0 == y1) break;

        int e2 = 2 \* err;

        if (e2 > -dy) {

            err -= dy;

            x0 += sx;

        }

        if (e2 < dx) {

            err += dx;

            y0 += sy;

        }

    }

    printf("Ending point: (%d, %d)\n\n", x1, y1);

}

void display() {

    glClear(GL\_COLOR\_BUFFER\_BIT);

    // draw coordinate system

    drawCoordinateSystem();

    // draw the line in green

    glColor3f(0.0, 1.0, 0.0);  // green color for the line

    glLineWidth(2.0);

    glBegin(GL\_LINES);

        glVertex2i(-3, 4);  // start point

        glVertex2i(5, -2);  // end point

    glEnd();

    // draw endpoints as points

    glColor3f(1.0, 0.0, 0.0);  // red color for points

    glPointSize(5.0);

    glBegin(GL\_POINTS);

        glVertex2i(-3, 4);  // start point

        glVertex2i(5, -2);  // end point

    glEnd();

    glFlush();

}

// initialize OpenGL

void init() {

    glClearColor(0.0, 0.0, 0.0, 1.0);  // background

    glMatrixMode(GL\_PROJECTION);

    glLoadIdentity();

    gluOrtho2D(-10, 10, -10, 10);  // cartesian plane calibration

}

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

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowSize(800, 800);

    glutInitWindowPosition(100, 100);

    glutCreateWindow("Midpoint Line-Drawing Algorithm (-3, 4) to (5, -2)");

    glewInit();

    init();

    // compute and print intermediate points

    computeAndPrintPoints(-3, 4, 5, -2);

    glutDisplayFunc(display);

    glutMainLoop();

    return 0;

}

Pset 4. Bresenham's Line Algorithm

How It Works:

- uses only integer arithmetic

- highly efficient for hardware implementation

- no floating-point calculations required

Steps:

1. calculate **dx** and **dy**

2. initialize **error term**

3. for each step:

- plot current pixel

- update **error term**

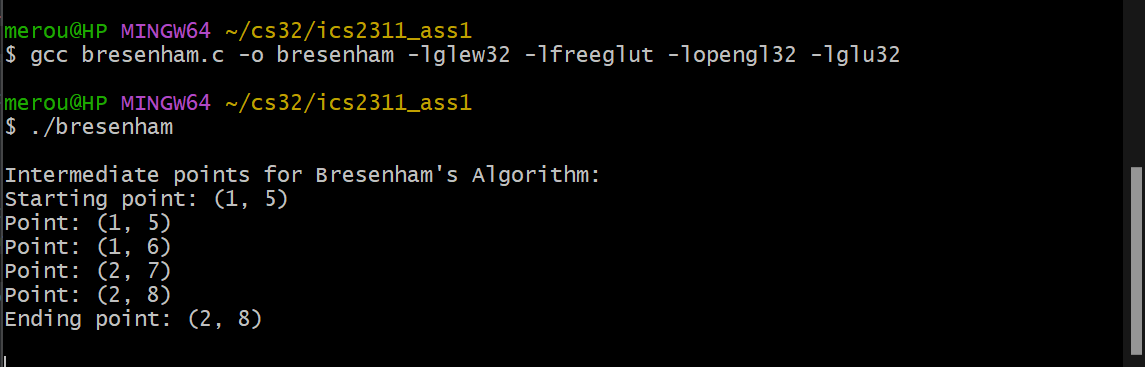
- choose next pixel based on error

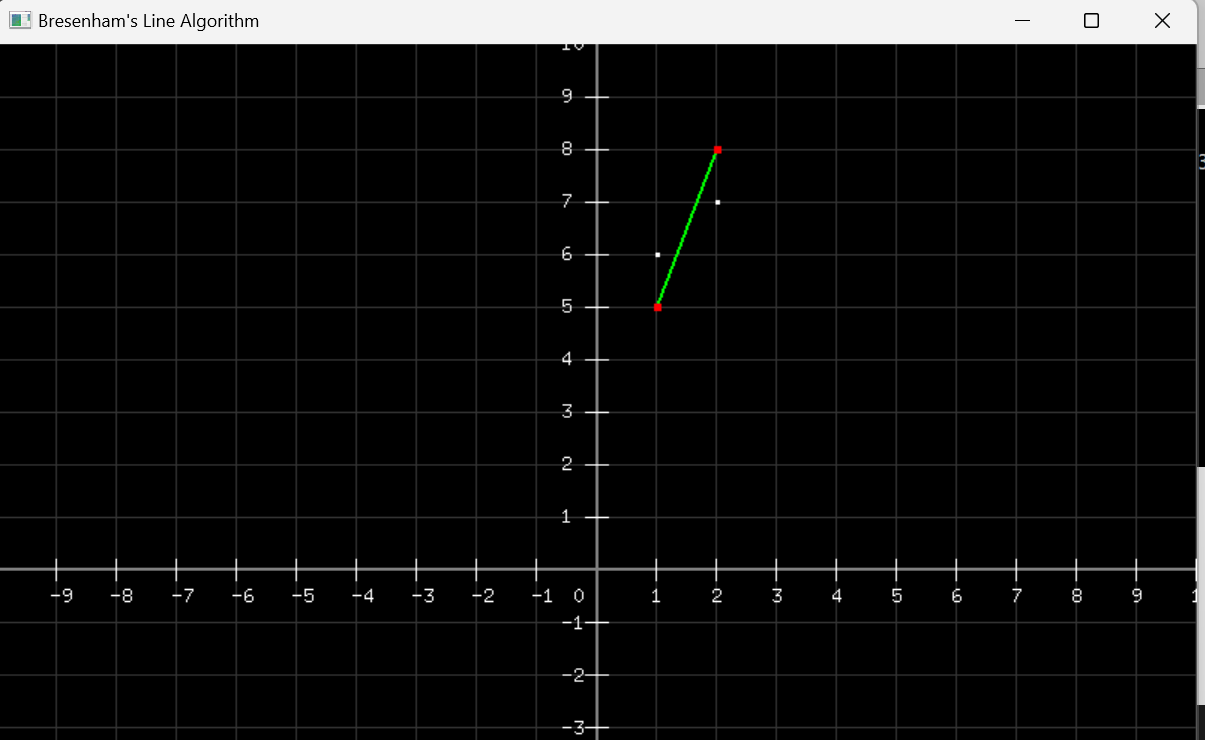
- update coordinates accordingly

4. points: (1,5) to (2,8)

OpenGL:

gcc bresenham.c -o bresenham -lglew32 -lfreeglut -lopengl32 -lglu32





C Code:

// bresenham.c - points from (1,5) to (2,8)

#include <GL/glew.h>

#include <GL/freeglut.h>

#include <GL/glut.h>

#include <stdio.h>

#include <math.h>

#include <string.h>

// store points for drawing

struct Point {

    int x;

    int y;

};

struct Point points[1000];

int num\_points = 0;

void drawCoordinateSystem() {

    // draw main axes

    glColor3f(0.5, 0.5, 0.5);

    glLineWidth(2.0);

    glBegin(GL\_LINES);

        // x-axis

        glVertex2f(-10.0, 0.0);

        glVertex2f(10.0, 0.0);

        // y-axis

        glVertex2f(0.0, -10.0);

        glVertex2f(0.0, 10.0);

    glEnd();

    // draw grid lines

    glLineWidth(1.0);

    glColor3f(0.2, 0.2, 0.2);

    glBegin(GL\_LINES);

    for(int i = -10; i <= 10; i++) {

        if(i == 0) continue;

        // vertical lines

        glVertex2f(i, -10.0);

        glVertex2f(i, 10.0);

        // horizontal lines

        glVertex2f(-10.0, i);

        glVertex2f(10.0, i);

    }

    glEnd();

    // draw tick marks and numbers

    glColor3f(1.0, 1.0, 1.0);

    for(int i = -10; i <= 10; i++) {

        if(i == 0) continue;

        // draw tick marks

        glBegin(GL\_LINES);

            // x-axis ticks

            glVertex2f(i, -0.2);

            glVertex2f(i, 0.2);

            // y-axis ticks

            glVertex2f(-0.2, i);

            glVertex2f(0.2, i);

        glEnd();

        // draw numbers

        char str[10];

        // x-axis numbers

        glRasterPos2f(i - 0.1, -0.6);

        sprintf(str, "%d", i);

        for(char\* c = str; \*c != '\0'; c++) {

            glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13, \*c);

        }

        // y-axis numbers

        if(i != 0) {

            glRasterPos2f(-0.6, i - 0.1);

            sprintf(str, "%d", i);

            for(char\* c = str; \*c != '\0'; c++) {

                glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13, \*c);

            }

        }

    }

    // draw origin "0"

    glRasterPos2f(-0.4, -0.6);

    glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13, '0');

}

void storePoint(int x, int y) {

    points[num\_points].x = x;

    points[num\_points].y = y;

    num\_points++;

    printf("Point: (%d, %d)\n", x, y);

}

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

    num\_points = 0;  // reset points array

    printf("\nIntermediate points for Bresenham's Algorithm:\n");

    printf("Starting point: (%d, %d)\n", x1, y1);

    int dx = abs(x2 - x1);

    int dy = abs(y2 - y1);

    int x = x1;

    int y = y1;

    int sx = x1 < x2 ? 1 : -1;

    int sy = y1 < y2 ? 1 : -1;

    int err = dx - dy;

    while (1) {

        storePoint(x, y);

        if (x == x2 && y == y2) break;

        int e2 = 2 \* err; // error terms

        if (e2 > -dy) {

            err -= dy;

            x += sx;

        }

        if (e2 < dx) {

            err += dx;

            y += sy;

        }

    }

    printf("Ending point: (%d, %d)\n\n", x2, y2);

}

void display() {

    glClear(GL\_COLOR\_BUFFER\_BIT);

    // draw coordinate system

    drawCoordinateSystem();

    // draw the actual line first

    glColor3f(0.0, 1.0, 0.0);  // green

    glLineWidth(2.0);

    glBegin(GL\_LINES);

        glVertex2i(1, 5);   // start point

        glVertex2i(2, 8);   // End point

    glEnd();

    // draw the computed points

    glColor3f(1.0, 1.0, 1.0);  // white color for computed points

    glPointSize(3.0);

    glBegin(GL\_POINTS);

    for(int i = 0; i < num\_points; i++) {

        glVertex2i(points[i].x, points[i].y);

    }

    glEnd();

    // draw endpoints

    glColor3f(1.0, 0.0, 0.0);  // red color for endpoints

    glPointSize(5.0);

    glBegin(GL\_POINTS);

        glVertex2i(1, 5);  // start point

        glVertex2i(2, 8);  // end point

    glEnd();

    glFlush();

}

void init() {

    glClearColor(0.0, 0.0, 0.0, 1.0);

    glMatrixMode(GL\_PROJECTION);

    glLoadIdentity();

    gluOrtho2D(-10, 10, -10, 10);

}

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

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowSize(800, 800);

    glutInitWindowPosition(100, 100);

    glutCreateWindow("Bresenham's Line Algorithm");

    glewInit();

    init();

    // compute points

    computeAndPrintPoints(1, 5, 2, 8);

    glutDisplayFunc(display);

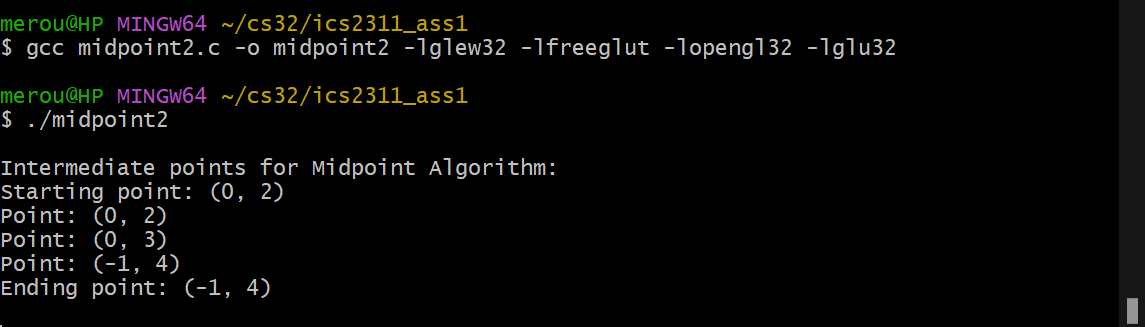
    glutMainLoop();

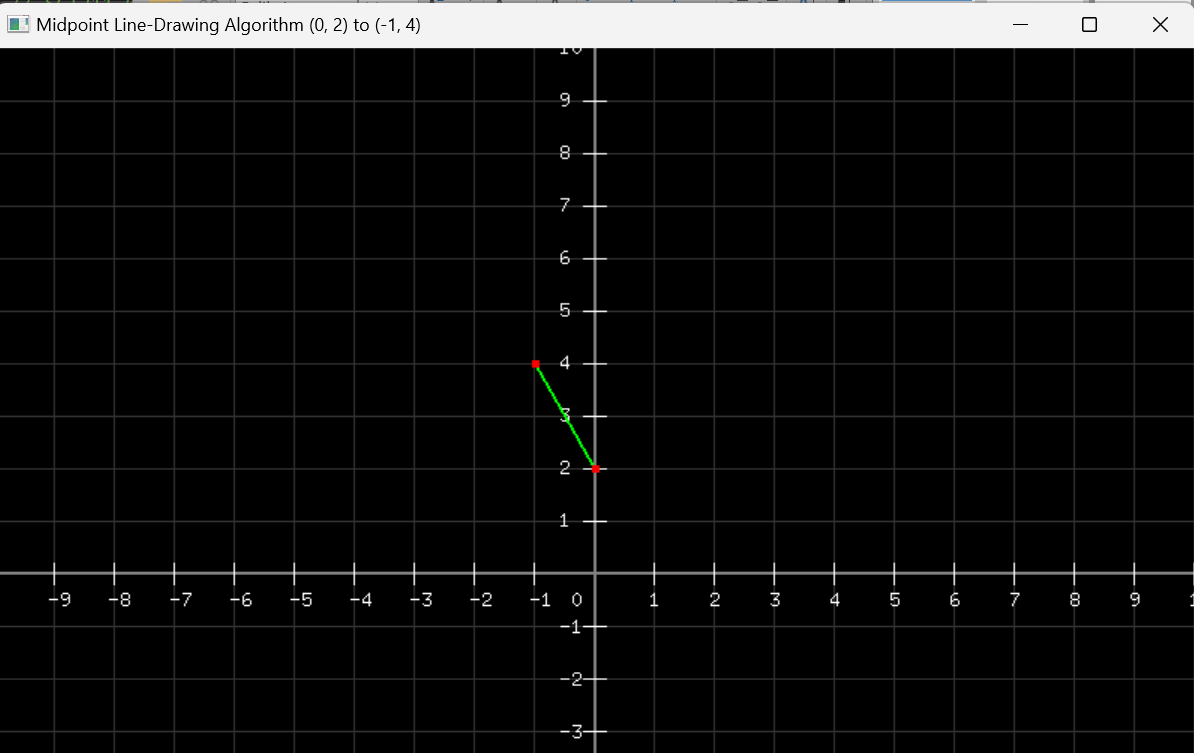
    return 0;

}

Pset 5. Second Midpoint Algorithm

OpenGL - for points: (0,2) to (-1,4)

****



C/OpenGL Code:

// SCT211-0848/2018 - Jany Muong - midpoint2.c

#include <GL/glew.h>

#include <GL/freeglut.h>

#include <GL/glut.h>

#include <stdio.h>

#include <math.h>

#include <string.h>

// function to draw the coordinate system (axes, grid, and labels)

void drawCoordinateSystem() {

    // draw main axes

    glColor3f(0.5, 0.5, 0.5);  // gray color for axes

    glLineWidth(2.0);

    glBegin(GL\_LINES);

        // x-axis

        glVertex2f(-10.0, 0.0);

        glVertex2f(10.0, 0.0);

        // y-axis

        glVertex2f(0.0, -10.0);

        glVertex2f(0.0, 10.0);

    glEnd();

    // draw grid lines

    glLineWidth(1.0);

    glColor3f(0.2, 0.2, 0.2);  // grid color

    glBegin(GL\_LINES);

    for (int i = -10; i <= 10; i++) {

        if (i == 0) continue; // skip the axes

        // vertical lines

        glVertex2f(i, -10.0);

        glVertex2f(i, 10.0);

        // horizontal lines

        glVertex2f(-10.0, i);

        glVertex2f(10.0, i);

    }

    glEnd();

    // draw tick marks and numbers

    glColor3f(1.0, 1.0, 1.0);  // white color for numbers

    for (int i = -10; i <= 10; i++) {

        if (i == 0) continue; // skip origin

        // draw tick marks

        glBegin(GL\_LINES);

            // x-axis ticks

            glVertex2f(i, -0.2);

            glVertex2f(i, 0.2);

            // y-axis ticks

            glVertex2f(-0.2, i);

            glVertex2f(0.2, i);

        glEnd();

        // draw numbers

        char str[10];

        // x-axis numbers

        glRasterPos2f(i - 0.1, -0.6);

        sprintf(str, "%d", i);

        for (char\* c = str; \*c != '\0'; c++) {

            glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13, \*c);

        }

        // y-axis numbers

        if (i != 0) {  // skip 0 for y-axis

            glRasterPos2f(-0.6, i - 0.1);

            sprintf(str, "%d", i);

            for (char\* c = str; \*c != '\0'; c++) {

                glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13, \*c);

            }

        }

    }

    // draw origin "0"

    glRasterPos2f(-0.4, -0.6);

    glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13, '0');

}

// function to compute intermediate points using Midpoint Algorithm

void computeAndPrintPoints(int x0, int y0, int x1, int y1) {

    int dx = abs(x1 - x0);

    int dy = abs(y1 - y0);

    int sx = (x0 < x1) ? 1 : -1;

    int sy = (y0 < y1) ? 1 : -1;

    int err = dx - dy;

    printf("\nIntermediate points for Midpoint Algorithm:\n");

    printf("Starting point: (%d, %d)\n", x0, y0);

    while (1) {

        printf("Point: (%d, %d)\n", x0, y0);

        if (x0 == x1 && y0 == y1) break;

        int e2 = 2 \* err;

        if (e2 > -dy) {

            err -= dy;

            x0 += sx;

        }

        if (e2 < dx) {

            err += dx;

            y0 += sy;

        }

    }

    printf("Ending point: (%d, %d)\n\n", x1, y1);

}

void display() {

    glClear(GL\_COLOR\_BUFFER\_BIT);

    // draw coordinate system

    drawCoordinateSystem();

    // draw the line in green

    glColor3f(0.0, 1.0, 0.0);  // green color for the line

    glLineWidth(2.0);

    glBegin(GL\_LINES);

        glVertex2i(0, 2);  // start point

        glVertex2i(-1, 4); // end point

    glEnd();

    // draw endpoints as lines

    glColor3f(1.0, 0.0, 0.0);  // red color for points

    glPointSize(5.0);

    glBegin(GL\_POINTS);

        glVertex2i(0, 2);  // start point

        glVertex2i(-1, 4); // end point

    glEnd();

    glFlush();

}

// initialize OpenGL

void init() {

    glClearColor(0.0, 0.0, 0.0, 1.0);  // background

    glMatrixMode(GL\_PROJECTION);

    glLoadIdentity();

    gluOrtho2D(-10, 10, -10, 10);  // cartesian plane calibration

}

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

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowSize(800, 800);

    glutInitWindowPosition(100, 100);

    glutCreateWindow("Midpoint Line-Drawing Algorithm (0, 2) to (-1, 4)");

    glewInit();

    init();

    // compute and print intermediate points

    computeAndPrintPoints(0, 2, -1, 4);

    glutDisplayFunc(display);

    glutMainLoop();

    return 0;

}

Pset 6. DDA (Digital Differential Analyzer) - Points: (5,2) to (10,3)

How It Works:

- Uses floating-point arithmetic

- Based on calculus concept of differential equations

**Steps**:

1. compute dx and dy

2. determine number of steps

3. compute x and y increments

4. for each step:

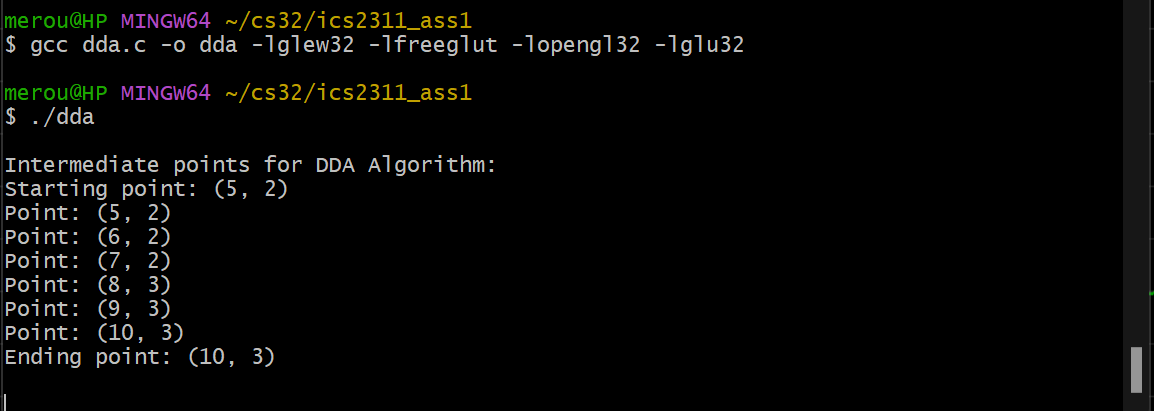
- add increment values

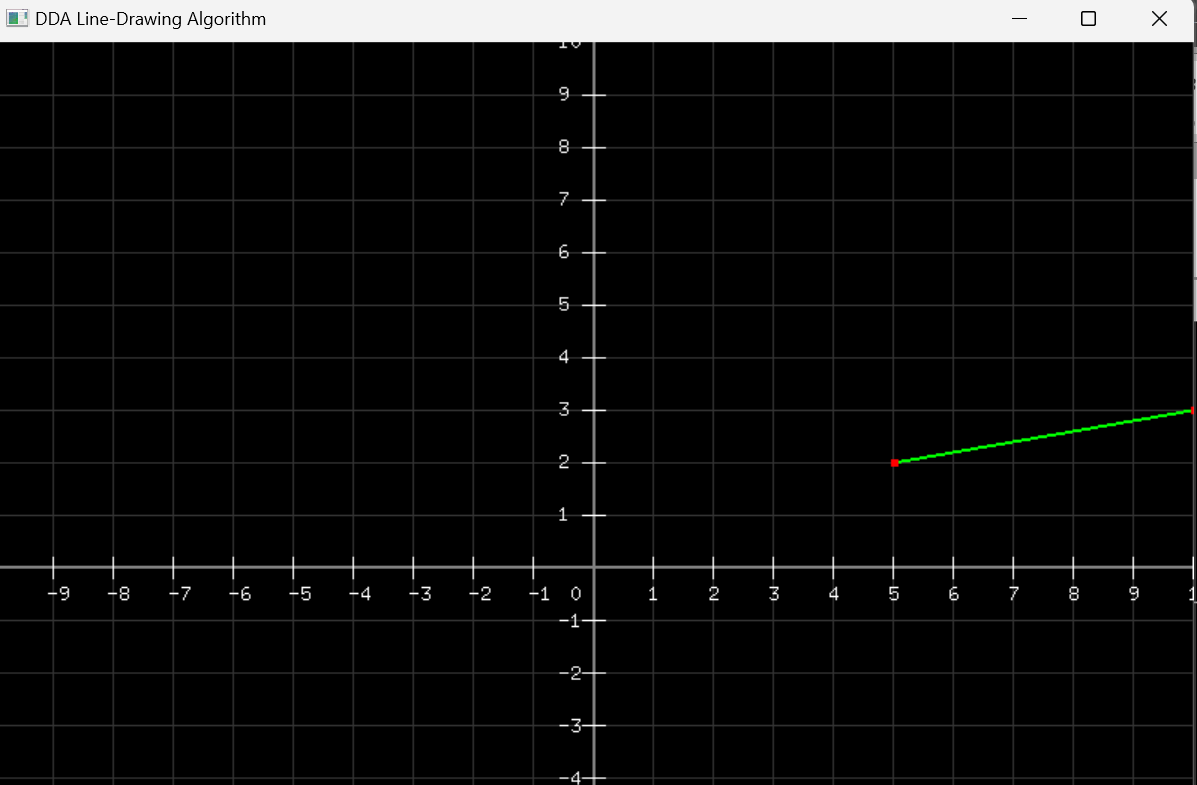
- round to nearest pixel

- plot pixel at rounded coordinates

OpenGL:

gcc dda.c -o dda -lglew32 -lfreeglut -lopengl32 -lglu32





C Code:

// SCT211-0848/2018 - Jany Muong

// dda.c - DIGITAL DIFFERENTIAL ANALYZER

#include <GL/glew.h>

#include <GL/freeglut.h>

#include <GL/glut.h>

#include <stdio.h>

#include <math.h>

#include <string.h>

void drawCoordinateSystem() {

    // draw main axes

    glColor3f(0.5, 0.5, 0.5);  // gray color for axes

    glLineWidth(2.0);

    glBegin(GL\_LINES);

        // x-axis

        glVertex2f(-10.0, 0.0);

        glVertex2f(10.0, 0.0);

        // y-axis

        glVertex2f(0.0, -10.0);

        glVertex2f(0.0, 10.0);

    glEnd();

    // draw grid lines

    glLineWidth(1.0);

    glColor3f(0.2, 0.2, 0.2);  // grid

    glBegin(GL\_LINES);

    for(int i = -10; i <= 10; i++) {

        if(i == 0) continue; // skip the axes

        // vertical lines

        glVertex2f(i, -10.0);

        glVertex2f(i, 10.0);

        // horizontal lines

        glVertex2f(-10.0, i);

        glVertex2f(10.0, i);

    }

    glEnd();

    // draw tick marks and numbers

    glColor3f(1.0, 1.0, 1.0);  // White color for numbers

    for(int i = -10; i <= 10; i++) {

        if(i == 0) continue; // Skip origin

        // draw tick marks

        glBegin(GL\_LINES);

            // x-axis ticks

            glVertex2f(i, -0.2);

            glVertex2f(i, 0.2);

            // y-axis ticks

            glVertex2f(-0.2, i);

            glVertex2f(0.2, i);

        glEnd();

        // draw numbers

        char str[10];

        // x-axis numbers

        glRasterPos2f(i - 0.1, -0.6);

        sprintf(str, "%d", i);

        for(char\* c = str; \*c != '\0'; c++) {

            glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13, \*c);

        }

        // y-axis numbers

        if(i != 0) {  // Skip 0 for y-axis

            glRasterPos2f(-0.6, i - 0.1);

            sprintf(str, "%d", i);

            for(char\* c = str; \*c != '\0'; c++) {

                glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13, \*c);

            }

        }

    }

    // draw origin "0"

    glRasterPos2f(-0.4, -0.6);

    glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13, '0');

}

void computeAndPrintPoints(int x0, int y0, int x1, int y1) {

    int dx = x1 - x0, dy = y1 - y0, steps;

    float xInc, yInc, x = x0, y = y0;

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

    xInc = dx / (float)steps;

    yInc = dy / (float)steps;

    printf("\nIntermediate points for DDA Algorithm:\n");

    printf("Starting point: (%d, %d)\n", x0, y0);

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

        printf("Point: (%d, %d)\n", (int)round(x), (int)round(y));

        x += xInc;

        y += yInc;

    }

    printf("Ending point: (%d, %d)\n\n", x1, y1);

}

void display() {

    glClear(GL\_COLOR\_BUFFER\_BIT);

    // draw coordinate system

    drawCoordinateSystem();

    // draw the line in green

    glColor3f(0.0, 1.0, 0.0);

    glLineWidth(2.0);

    glBegin(GL\_LINES);

        glVertex2i(5, 2);  // start point

        glVertex2i(10, 3); // end point

    glEnd();

    // draw endpoints as points

    glColor3f(1.0, 0.0, 0.0);  // red color for points

    glPointSize(5.0);

    glBegin(GL\_POINTS);

        glVertex2i(5, 2);  // start point

        glVertex2i(10, 3); // end point

    glEnd();

    glFlush();

}

void init() {

    glClearColor(0.0, 0.0, 0.0, 1.0);

    glMatrixMode(GL\_PROJECTION);

    glLoadIdentity();

    gluOrtho2D(-10, 10, -10, 10);

}

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

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowSize(800, 800);

    glutInitWindowPosition(100, 100);

    glutCreateWindow("DDA Line-Drawing Algorithm");

    glewInit();

    init();

    // compute and print points

    computeAndPrintPoints(5, 2, 10, 3);

    glutDisplayFunc(display);

    glutMainLoop();

    return 0;

}

End file: