Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavre

Lab II: Implementation of Line Drawing Algorithms

COMP 342

Submitted by:

Manish Shivabhakti
Roll no: 63
CE, III year/ II Semester

Implementation of the Line Drawing Algorithms

Algorithms

Digital Differential Analyzer (DDA) Line Drawing Algorithm

1. Calculate the difference between the x-coordinates (dx) and y-coordinates (dy) of the two endpoints.
2. Determine the number of steps required for generating the line. This is the maximum of the absolute values of dx and dy.
3. Calculate the increment values for each step in the x and y directions (x_inc and y_inc).
4. Initialize the starting point coordinates (x and y).
5. For each step:
    a. Plot the current point.
    b. Increment the x and y coordinates by x_inc and y_inc respectively.

Bresenham's Line Drawing Algorithm (|m| < 1)

1. Calculate the differences dx and dy.
2. Initialize the decision parameter p as 2*dy - dx.
3. Initialize the starting point coordinates (x and y).
4. For each x-coordinate from the starting point to the end point:
    a. Plot the current point.
    b. If p is less than 0:
        i. Increment the x-coordinate.
        ii. Update p by adding 2*dy.
    c. Otherwise:
        i. Increment both the x and y coordinates.
        ii. Update p by adding 2*dy - 2*dx.

Bresenham's Line Drawing Algorithm (|m| >= 1)

1. Calculate the differences dx and dy.
2. Initialize the decision parameter p as 2*dx - dy.
3. Initialize the starting point coordinates (x and y).
4. For each y-coordinate from the starting point to the end point:
    a. Plot the current point.
    b. If p is less than 0:
        i. Increment the y-coordinate.
        ii. Update p by adding 2*dx.
    c. Otherwise:
        i. Increment both the x and y coordinates.
        ii. Update p by adding 2*dx - 2*dy.

Histogram Drawing

1. Calculate the differences between `dx` and `dy` for each line segment based on the frequency inputs.
2. Determine the appropriate line drawing algorithm (DDA or Bresenham) based on the slope.
3. Apply the chosen line drawing algorithm for each line segment.
4. Plot each line segment to form the histogram based on the frequency inputs.

Source Code

DDA

```
import pygame
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *


def dda(start_pos, end_pos, color):
    glColor3f(color[0] / 255.0, color[1] / 255.0, color[2] / 255.0)

    # Unpack start and end positions
    x1, y1 = start_pos
    x2, y2 = end_pos

    dx = x2 - x1
    dy = y2 - y1

    # Determine number of steps required for the algorithm
    steps = abs(dx) if abs(dx) > abs(dy) else abs(dy)

    # Calculate the increment in x and y for each step
    x_inc = dx / float(steps)
    y_inc = dy / float(steps)

    # Initialize starting point
    x = x1
    y = y1

    glBegin(GL_POINTS)
    for i in range(steps):
        glVertex2f(x, y)
        x += x_inc
        y += y_inc
    glEnd()


def init_gl(screen_width, screen_height):
    glViewport(0, 0, screen_width, screen_height)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluOrtho2D(0, screen_width, 0, screen_height)
```

```
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()


def main():
    # Initialize Pygame
    pygame.init()

    # Set screen size
    screen_width = 800
    screen_height = 600
    screen = pygame.display.set_mode(
        (screen_width, screen_height), DOUBLEBUF | OPENGL)
    pygame.display.set_caption("DDA Line Drawing Algorithm")
    # Set line color
    line_color = (255, 0, 0)  # Red

    # Set start and end positions for the line
    start_pos = (100, 100)
    end_pos = (500, 300)

    # Initialize OpenGL
    init_gl(screen_width, screen_height)

    # Main loop
    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

        # Draw the line using DDA
        dda(start_pos, end_pos, line_color)

        pygame.display.flip()

    # Quit Pygame
    pygame.quit()


if __name__ == "__main__":
    main()
```

Bresenham’s Line Drawing (for both m<1 and m>=1)

```
import pygame
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *


def draw_bresenham_line(x1, y1, x2, y2):
    dx = abs(x2 - x1)
    dy = abs(y2 - y1)
    sx = 1 if x1 < x2 else -1
    sy = 1 if y1 < y2 else -1

    if dx > dy:
        err = dx / 2.0
        while x1 != x2:
            glVertex2i(x1, y1)
            err -= dy
            if err < 0:
                y1 += sy
                err += dx
            x1 += sx
    else:
        err = dy / 2.0
        while y1 != y2:
            glVertex2i(x1, y1)
            err -= dx
            if err < 0:
                x1 += sx
                err += dy
            y1 += sy
    glVertex2i(x2, y2)  # Ensuring the last point is drawn


def display():
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

    glColor3f(1.0, 0.0, 0.0)  # Red color
    glBegin(GL_POINTS)
    draw_bresenham_line(-50, -50, 50, 50)
    glEnd()

    glColor3f(0.0, 1.0, 0.0)  # Green color
    glBegin(GL_POINTS)
```

```
    draw_bresenham_line(-50, 50, 50, -50)
    glEnd()

    glColor3f(0.0, 0.0, 1.0)  # Blue color
    glBegin(GL_POINTS)
    draw_bresenham_line(-50, 0, 50, 0)
    glEnd()

    glColor3f(0.0, 0.0, 0.0)  # black color
    glBegin(GL_POINTS)
    draw_bresenham_line(0, -50, 0, 50)
    glEnd()

    pygame.display.flip()


def main():
    pygame.init()
    pygame.display.set_mode((800, 600), DOUBLEBUF | OPENGL)
    pygame.display.set_caption('Bresenham Line Drawing Algorithm')
    gluOrtho2D(-100, 100, -100, 100)

    glClearColor(1.0, 1.0, 1.0, 1.0)
    glColor3f(1.0, 0.0, 0.0)

    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

        display()
        pygame.time.wait(10)

    pygame.quit()


if __name__ == "__main__":
    main()
```

Histogram

```
import pygame
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *


def bresenham_line(x1, y1, x2, y2):
    points = []
    dx = abs(x2 - x1)
    dy = abs(y2 - y1)
    sx = 1 if x1 < x2 else -1
    sy = 1 if y1 < y2 else -1
    err = dx - dy

    while True:
        points.append((x1, y1))
        if x1 == x2 and y1 == y2:
            break
        e2 = err * 2
        if e2 > -dy:
            err -= dy
            x1 += sx
        if e2 < dx:
            err += dx
            y1 += sy
    return points


def draw_histogram(frequencies):
    # Setup the initial OpenGL environment
    pygame.init()
    display = (800, 600)
    pygame.display.set_mode(display, DOUBLEBUF | OPENGL)
    gluOrtho2D(0, 800, 0, 600)
    glClearColor(1.0, 1.0, 1.0, 1.0)

    glColor3f(0.0, 0.0, 0.0)

    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
```

```
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

        # Draw axes
        glBegin(GL_LINES)
        glVertex2f(50, 50)
        glVertex2f(50, 550)
        glVertex2f(50, 50)
        glVertex2f(750, 50)
        glEnd()

        # Draw histogram lines
        bar_width = (700 - 50) // len(frequencies)
        x1, y1 = 50, 50
        for i, frequency in enumerate(frequencies):
            x2 = 50 + (i + 1) * bar_width
            y2 = 50 + frequency * 5  # Scale frequency for better
visualization
            points = bresenham_line(x1, y1, x2, y2)
            glBegin(GL_POINTS)
            for point in points:
                glVertex2f(point[0], point[1])
            glEnd()
            x1, y1 = x2, y2

        pygame.display.flip()
        pygame.time.wait(10)

    pygame.quit()


if __name__ == "__main__":
    # Sample frequency inputs for the histogram
    frequencies = [10, 20, 15, 25, 30, 20, 10, 5, 30, 20]
    draw_histogram(frequencies)
```

Output

Figure 1. DDA Line Drawing

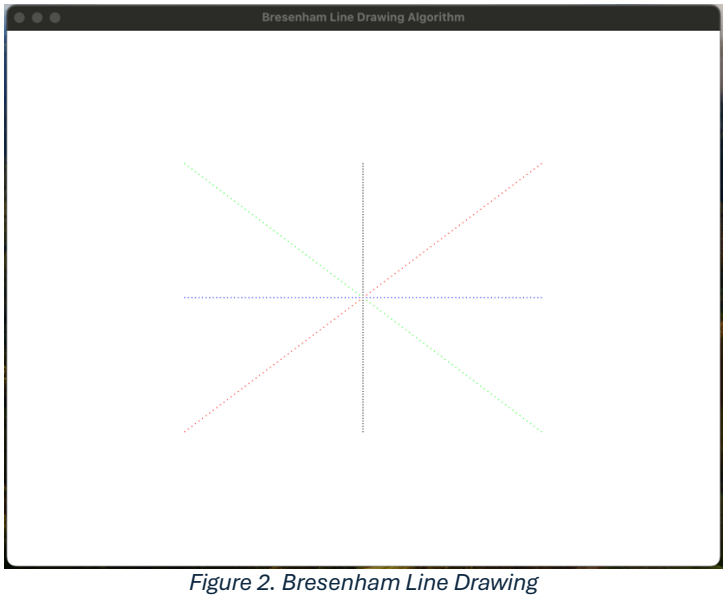

Figure 2. Bresenham Line Drawing

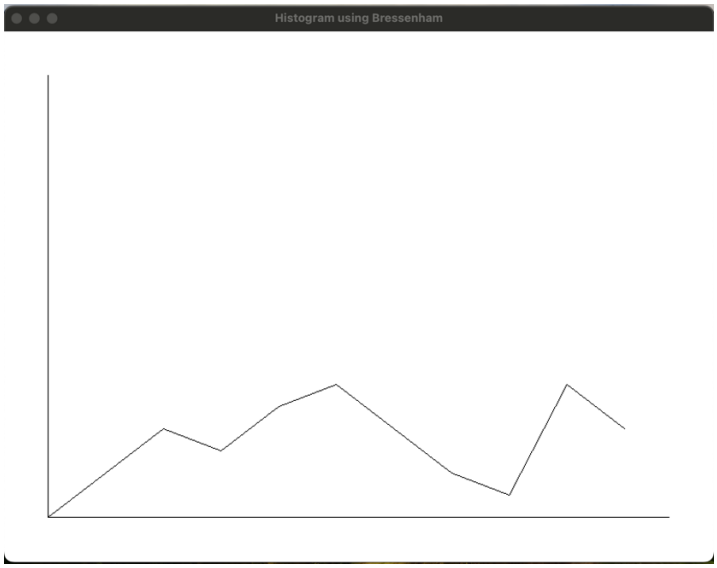

Figure 3. Histogram using Bresenham

Conclusion

The implementation of the Digital Differential Analyzer (DDA) and Bresenham Line Drawing algorithms demonstrates the effectiveness and efficiency of each method in drawing lines on a graphical display. The DDA algorithm provides a straightforward approach with floating-point calculations, while Bresenham's algorithm offers an integer-based method that improves performance, especially in raster graphics. Both algorithms are essential for various graphical applications, including drawing line histograms for visualizing frequency inputs.