Computer Graphics 2D

3rd class

Lecture 2: Line Drawing Algorithms

Department of Computer Science
College of Computer and Information Technology
University of Anbar

Assist Prof. Shumoos T. Alfahdawi

2.1 Basic Line Drawing

Line drawing algorithms are fundamental in computer graphics for rendering straight lines on raster displays. Several algorithms have been developed to handle the challenges of converting mathematical line equations into pixel representations. As shown in figure 1 the lines of this object appear continuous, However, they are made of pixels. Some of key line drawing algorithms:

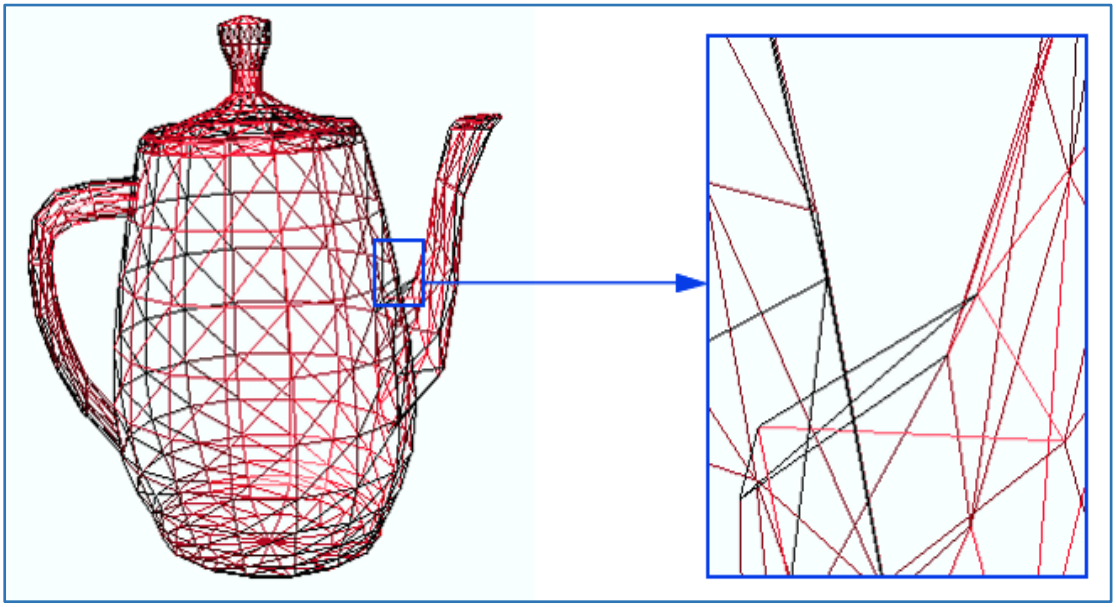

Figure (1): The lines of this object made of pixels.

2.2 DDA (Digital Differential Analyzer) Algorithm:

The **Digital Differential Analyzer (DDA) Algorithm** is a simple and widely used method for drawing straight lines in computer graphics. It works by calculating intermediate points between two given endpoints of the line and determining which pixels should be plotted to represent the line on a raster display.

How the DDA Algorithm Works

The DDA algorithm incrementally generates points between the start and end points of the line using the slope of the line. Here's a step-by-step explanation:

1. **Calculate the Difference Between Endpoints:**

- Let the start point be (X_0, Y_0) and the end point be (X_1, Y_1)
- Calculate the differences:

$$\Delta x = X_1 - X_0$$
$$\Delta y = Y_1 - Y_0$$

2. **Determine the Number of Steps:**

- The number of steps required to draw the line is determined by the greater of Δx or Δy. This ensures that the line is drawn with uniform steps either in the **x**-direction or the **y**-direction.

$$\textbf{Steps} = \max(|\Delta x|, |\Delta y|)$$

3. **Calculate the Increment for Each Step:**

- Depending on the number of steps, calculate the increment in the **x** and **y** coordinates for each step:

$$x_{increment} = \Delta x \,/\, \textbf{Steps}$$
$$y_{increment} = \Delta y \,/\, \textbf{Steps}$$

4. Initialize the Starting Point:

- Start with the initial point $(x, y) = (X_0, Y_0)$.

5. **Iterate to Plot Points:**

- For each step from **1** to the total number of steps:
  - Plot the current point (x, y) by rounding the coordinates to the nearest integer.
  - Update the coordinates for the next point:

$$x = x + x_{incrementx}$$
$$y = y + y_{incrementy}$$

6. **End the Algorithm:**

- The algorithm stops after all the points between the start and end points are plotted.

✓ **Example 1:** Consider a line with endpoints (**2,3**) and (**10,8**) using DDA Algorithm.

1. **Calculate Differences:**

   $\Delta x = 10-2 = 8$

   $\Delta y = 8-3 = 5$

2. **Determine the Number of Steps:**

   Steps = max (8,5) = 8

3. **Calculate Increments:**

   $x_{increment} = 8/8 = 1.0$

   $y_{increment} = 5/8 = 0.625$

4. **Initialize Starting Point:**

   (**x, y**) = (2,3)

5. **Plot Points:**

- Step 1: (2,3)
- Step 2: (3,3.625) → round to (3,4)
- Step 3: (4,4.25) → round to (4,4)
- Step 4: (5,4.875) → round to (5,5)
- Step 5: (6,5.5) → round to (6,6)
- Step 6: (7,6.125) → round to (7,6)
- Step 7: (8,6.75) → round to (8,7)
- Step 8: (9,7.375) → round to (9,7)
- Step 9: (10,8)

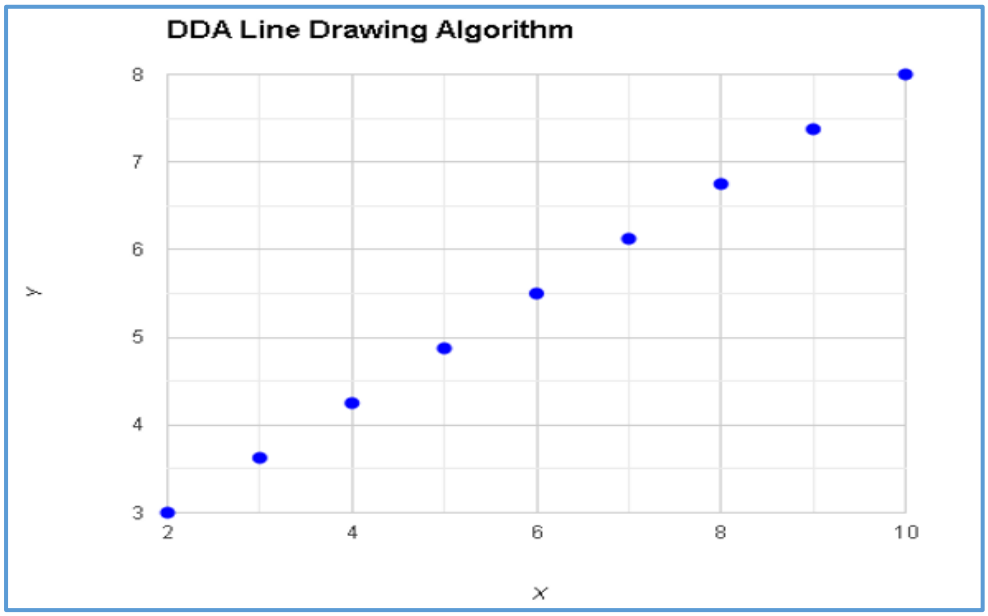

Figure (2): DDA Line drawing algorithm

✓ **Example 2:** Drawing a Line from (**2, 3**) to (**8, 7**) using DDA Algorithm

Given:

- Start point ($\mathbf{X_0,Y_0}$) = (2, 3)
- End point ($\mathbf{X_1,Y_1}$) = (8,7)

Steps to Implement the DDA Algorithm:

1. **Calculate Differences:**
   - $\mathbf{\Delta x} = \mathbf{X_1 - X_0} = 8-2 = \mathbf{6}$
   - $\mathbf{\Delta y} = \mathbf{Y_1 - Y_0} = 7-3 = \mathbf{4}$
2. **Determine the Number of Steps:**
   - The number of steps is determined by the maximum of **Δx** or **Δy**:

     Steps = max(| **Δx** |,| **Δy** |) = max(6, 4) = **6**
3. **Calculate the Increments:**
   - Calculate the increments for x and y per step:

     $\mathbf{x}_{increment} = \Delta x$ / Steps = 6/6 = 1.0

     $\mathbf{y}_{increment} = \Delta y$ / Steps = 4/6 ≈ 0.67
4. **Initialize the Starting Point:**
   - Start at (**x, y**) = (**2, 3**)
5. **Plot the Points:**
   - For each step, increment **x** and **y** by the calculated increments and round them to the nearest integer to determine the pixel position.

Iteration Details:

Step	x	y	Rounded Point	Plotted Pixel
1	2	3	(2, 3)	(2, 3)
2	3.0	3.67	(3, 4)	(3, 4)
3	4.0	4.34	(4, 4)	(4, 4)
4	5.0	5.01	(5, 5)	(5, 5)
5	6.0	5.68	(6, 6)	(6, 6)
6	7.0	6.35	(7, 6)	(7, 6)
7	8.0	7.02	(8, 7)	(8, 7)

Advantages of the DDA Algorithm

1. **Simplicity:**
   The DDA algorithm is easy to understand and implement. It follows a straightforward approach to line drawing, making it suitable for educational purposes and simple applications.
2. **Uniformity:**
   The algorithm ensures that the line is drawn with a uniform step size, leading to consistent and smooth lines.
3. **Versatility:**
   DDA can be used for lines with arbitrary slopes, making it flexible for different kinds of line drawing tasks.
4. **Incremental Calculation:**
   The algorithm uses incremental calculations, reducing the amount of computation required for each subsequent point on the line.

Disadvantages of the DDA Algorithm

1. **Floating-Point Operations:**
   DDA involves floating-point arithmetic for calculating the increments in x and y coordinates. On systems without hardware support for floating-point operations, this can be computationally expensive and slower than integer arithmetic.
2. **Rounding Errors:**
   The algorithm requires rounding the calculated coordinates to the nearest integer, which can introduce small inaccuracies. Over long lines, these rounding errors can accumulate and cause noticeable deviations from the ideal line.
3. **Efficiency:**
   While DDA is simple, it is not the most efficient algorithm for line drawing. The floating-point operations and rounding steps make it less suitable for real-time applications compared to algorithms like Bresenham's, which uses only integer arithmetic.
4. **Not Optimized for Performance:**

   In scenarios where performance is critical, such as in high-resolution displays or real-time graphics, DDA may not be the best choice due to its reliance on floating-point calculations.