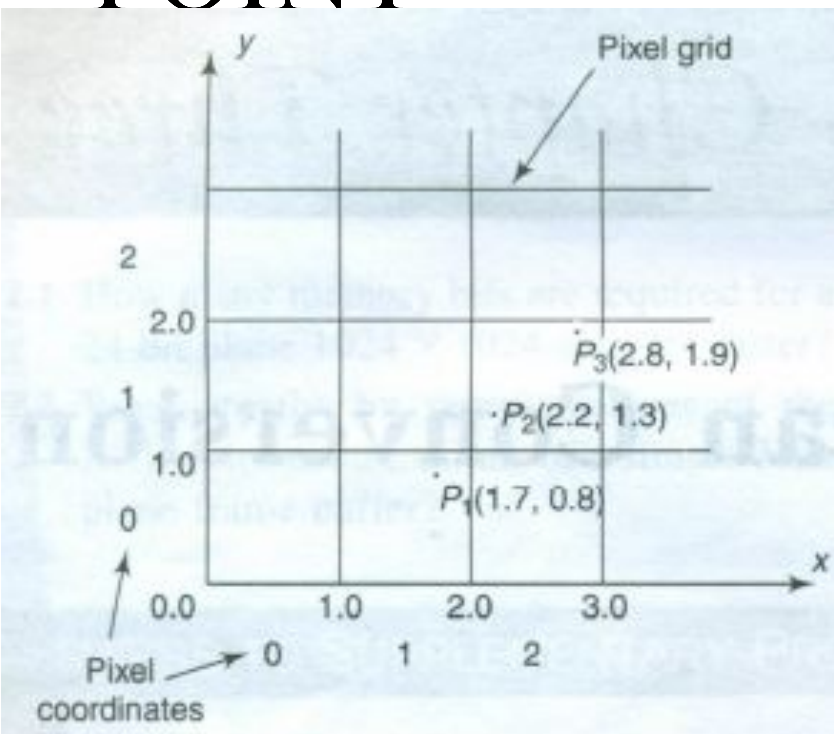
# COMPUTER GRAPHICS LECTURE-3 (SCAN CONVERSION-1)

# SCAN CONVERSION

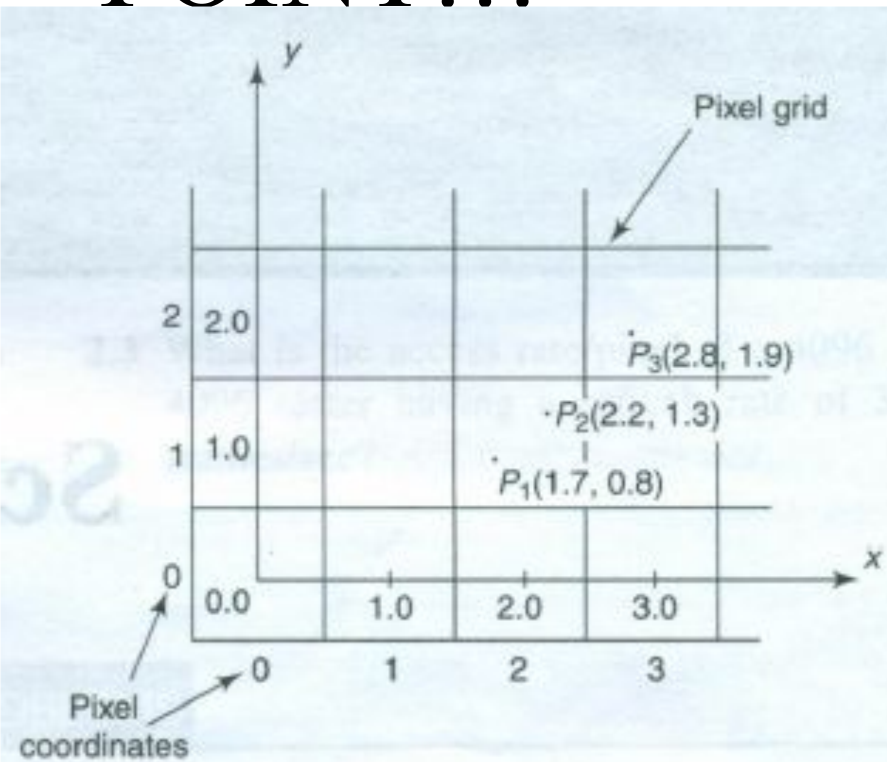
- **Rasterisation** (or **rasterization**) is the task of taking an image described in a vector graphics format (shapes) and converting it into a raster image (pixels or dots) for output on a video display or printer, or for storage in a bitmap file format.
- This is also known as **scan conversion**.

# SCAN CONVERSION OF A POINT




- A point $(x, y)$ within an image area, scan converted to a pixel at location $(x', y')$.
- $x' = \mathrm{Floor}(x)$ and $y' = \mathrm{Floor}(y)$.
- All points satisfying $x' \le x < x' + 1$ and $y' \le y < y' + 1$ are mapped to pixel $(x', y')$.
- Point P1(1.7, 0.8) is represented by pixel (1, 0) and points $P_2(2.2,1.3)$ and $P_3(2.8,1.9)$ are both represented by pixel (2, 1).

# SCAN CONVERSION OF A POINT…




- Another approach is to align the integer values in the co-ordinate system for $(x, y)$ with the pixel co-ordinates.
- Here $x' = \text{Floor}(x + 0.5)$ and $y' = \text{Floor}(y + 0.5)$
- Points $P_1$ and $P_2$ both are now represented by pixel (2, 1) and $P_3$ by pixel (3, 2).

# LINE DRAWING ALGORITHM

- Need algorithm to figure out which intermediate pixels are on line path
- Pixel ($x$, $y$) values constrained to integer values
- Actual computed intermediate line values may be floats
- Rounding may be required. Computed point

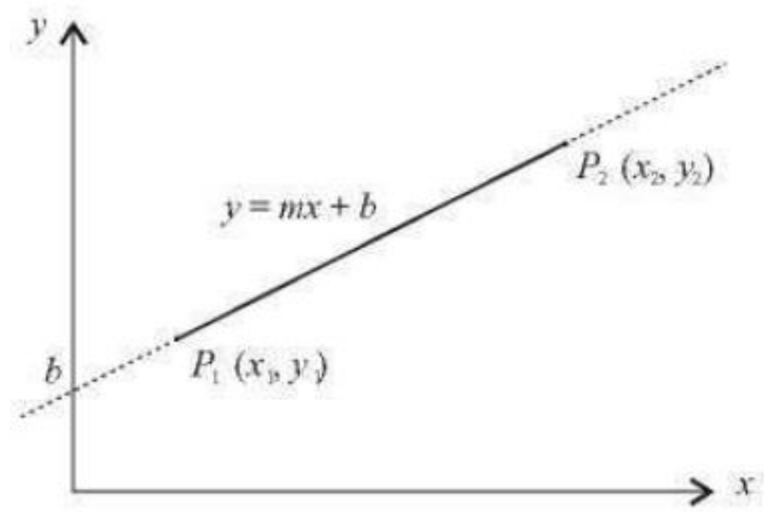
  (10.48, 20.51) rounded to (10, 21)
- Rounded pixel value is off actual line path (jaggy!!)
- Sloped lines end up having jaggies
- Vertical, horizontal lines, no jaggies

# LINE DRAWING ALGORITHM

- Line is defined by its two endpoints and the line equation y = mx + b
  - where m is called the slope
  - In Fig. 3-2 the two endpoints are described by Pi(xl ,yl) and P2(x2,y2).
  - The line equation describes the coordinates of all the points that lie between the two endpoints.
- Given two end points (x0,y0), (x1, y1), how to compute m and b?

$$m = \frac{dy}{dx} = \frac{y1 - y0}{x1 - x0}$$

$$b = y0 - m * x0$$



# DIRECT USE OF THE LINE EQUATION

A simple approach to scan-converting a line is to first scan-convert $P_1$ and $P_2$ to pixel coordinates $(x'_1, y'_1)$ and $(x'_2, y'_2)$, respectively; then set $m = (y'_2 - y'_1)/(x'_2 - x'_1)$ and $b = y'_1 - mx'_1$. If $|m| \leq 1$, then for every integer value of $x$ between and excluding $x'_1$ and $x'_2$, calculate the corresponding value of $y$ using the equation and scan-convert $(x, y)$. If $|m| > 1$, then for every integer value of $y$ between and excluding $y'_1$ and $y'_2$, calculate the corresponding value of $x$ using the equation and scan-convert $(x, y)$.

# DISADVANTAGE OF DIRECT USE OF THE LINE EQUATION

- It involves floating-point computation (multiplication and addition) in every step that uses the line equation since m and b are generally real numbers.
- The challenge is to find a way to achieve the same goal as quickly as possible.

# DIGITAL DIFFERENTIAL ANALYZER (DDA): LINE DRAWING ALGORITHM

- The digital differential analyzer (DDA) algorithm is an incremental scan-conversion method.
- Such an approach is characterized by performing calculations at each step using results from the preceding step.

Suppose at step $i$ we have calculated $(x_i, y_i)$ to be a point on the line. Since the next point $(x_{i+1}, y_{i+1})$ should satisfy $\Delta y/\Delta x = m$ where $\Delta y = y_{i+1} - y_i$ and $\Delta x = x_{i+1} - x_i$, we have

$$y_{i+1} = y_i + m\Delta x$$

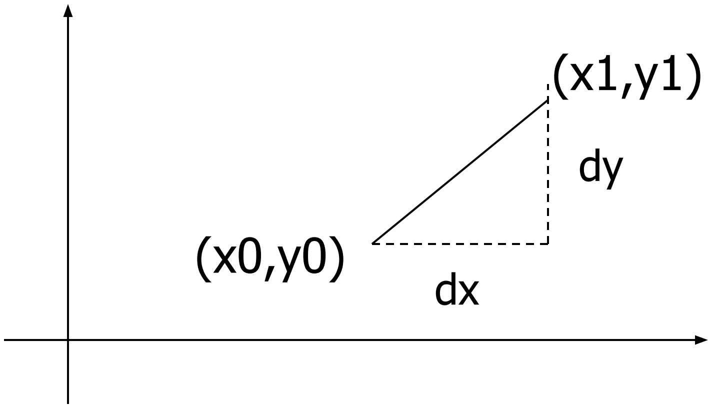
or

$$x_{i+1} = x_i + \Delta y/m$$

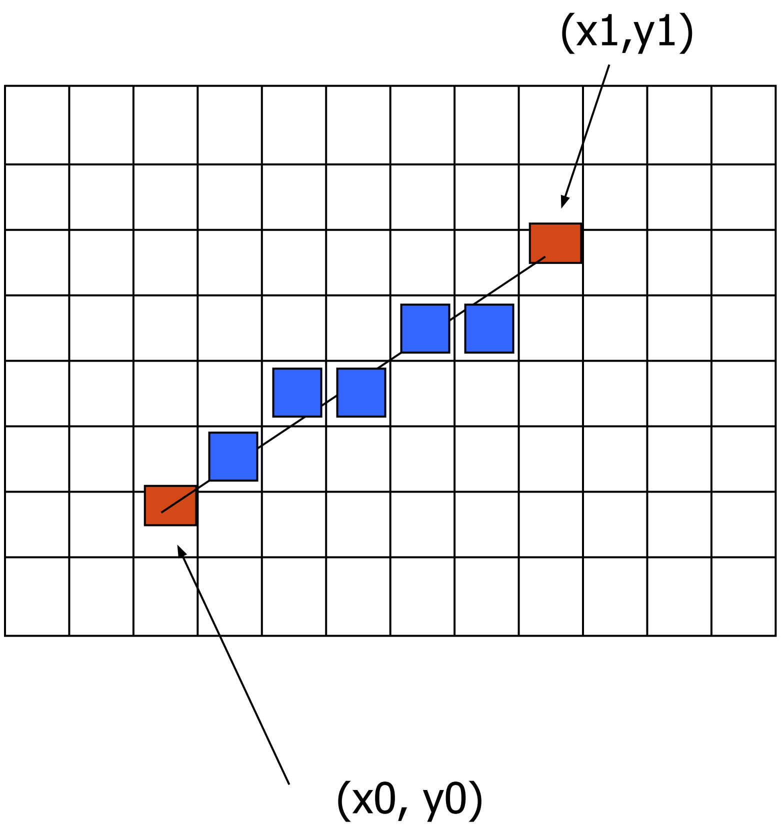
These formulas are used in the DDA algorithm as follows. When $|m| \leq 1$, we start with $x = x'_1$ (assuming that $x'_1 < x'_2$) and $y = y'_1$, and set $\Delta x = 1$ (i.e., unit increment in the $x$ direction). The $y$ coordinate of each successive point on the line is calculated using $y_{i+1} = y_i + m$. When $|m| > 1$, we start with $x = x'_1$ and $y = y'_1$ (assuming that $y'_1 < y'_2$), and set $\Delta y = 1$ (i.e., unit increment in the $y$ direction). The $x$ coordinate of each successive point on the line is calculated using $x_{i+1} = x_i + 1/m$. This process continues until $x$ reaches $x'_2$ (for the $|m| \leq 1$ case) or $y$ reaches $y'_2$ (for the $|m| > 1$ case) and all points found are scan-converted to pixel coordinates.

# DIGITAL DIFFERENTIAL ANALYZER (DDA): LINE DRAWING ALGORITHM

- Walk through the line, starting at (x0,y0)
- Constrain x, y increments to values in [0,1] range
- Case a: x is incrementing faster ($m < 1$)
    - Step in x=1 increments, compute and round y
- Case b: y is incrementing faster ($m > 1$)
    - Step in y=1 increments, compute and round x



# DDA LINE DRAWING ALGORITHM (CASE A: M < 1)




x = x0 y = y0

Illuminate pixel (x, round(y))

x = x0 + 1 y = y0 + 1 * m

Illuminate pixel (x, round(y))

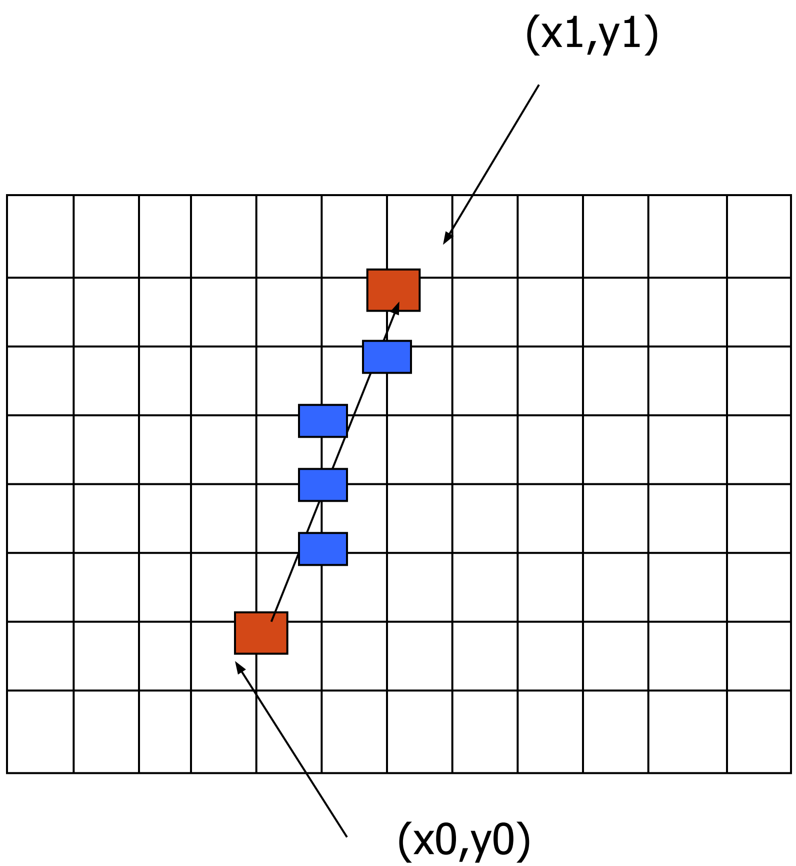
x = x + 1 y = y + 1 * m

Illuminate pixel (x, round(y))

…

Until x == x1

# DDA LINE DRAWING ALGORITHM (CASE B: M > 1)




x = x0      y = y0

Illuminate pixel (round(x), y)

y = y0 + 1      x = x0 + 1 * 1/m

Illuminate pixel (round(x), y)

y = y + 1      x = x + 1 /m

Illuminate pixel (round(x), y)

…

Until y == y1

# DDA LINE DRAWING ALGORITHM PSEUDOCODE

```
compute m;
if m < 1:
{
   float y = y0;        // initial value
   for(int x = x0;x <= x1; x++, y += m)
              setPixel(x, round(y));
}
else // m > 1
{
   float x = x0;        // initial value
   for(int y = y0;y <= y1; y++, x += 1/m)
              setPixel(round(x), y);
}
```

- Note: **`setPixel(x, y)`** writes current color into pixel in column x and row y in frame buffer

# DDA EXAMPLE (CASE A: M < 1)

- Suppose we want to draw a line starting at pixel (2,3) and ending at pixel (12,8).
- What are the values of the variables x and y at each timestep?
- What are the pixels colored, according to the DDA algorithm?

| t | x | y | R(x) | R(y) |
|---|---|---|---|---|
| 0 | 2 | 3 | 2 | 3 |
| 1 | 3 | 3.5 | 3 | 4 |
| 2 | 4 | 4 | 4 | 4 |
| 3 | 5 | 4.5 | 5 | 5 |
| 4 | 6 | 5 | 6 | 5 |
| 5 | 7 | 5.5 | 7 | 6 |
| 6 | 8 | 6 | 8 | 6 |
| 7 | 9 | 6.5 | 9 | 7 |
| 8 | 10 | 7 | 10 | 7 |
| 9 | 11 | 7.5 | 11 | 8 |
| 10 | 12 | 8 | 12 | 8 |

# DDA ALGORITHM DRAWBACKS

- DDA is the simplest line drawing algorithm
  - Not very efficient
  - Floating point operations and rounding operations are expensive.