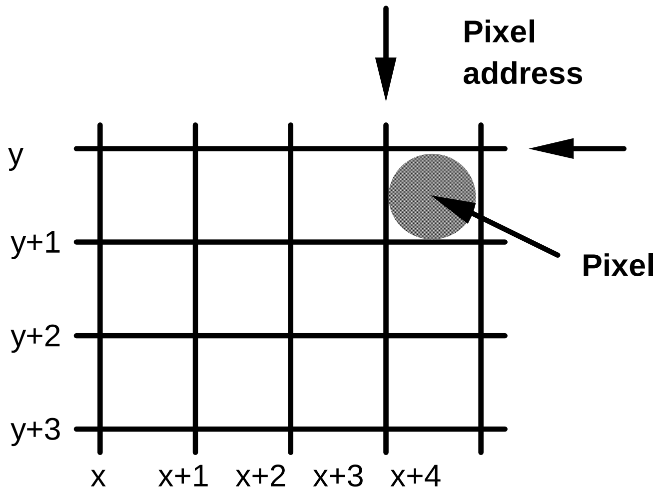
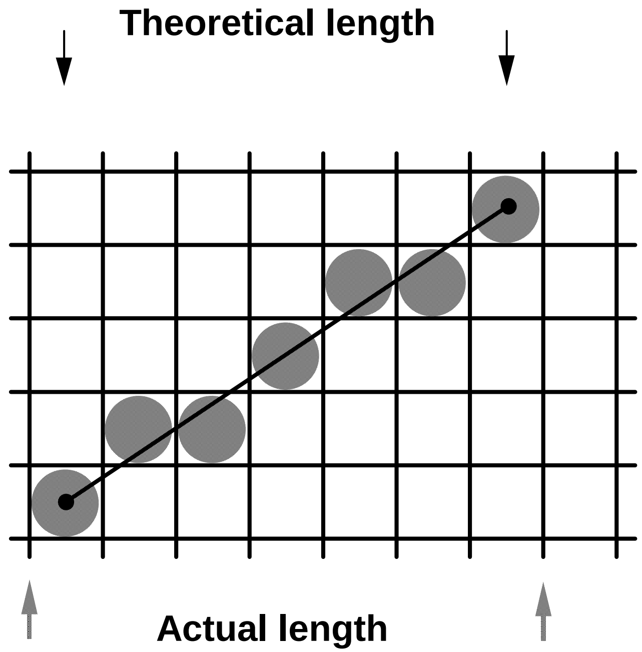
Raster Conversion Algorithms

DDA line algorithm

Bresenham line algorithm

Pixel Addressing in Raster Graphics

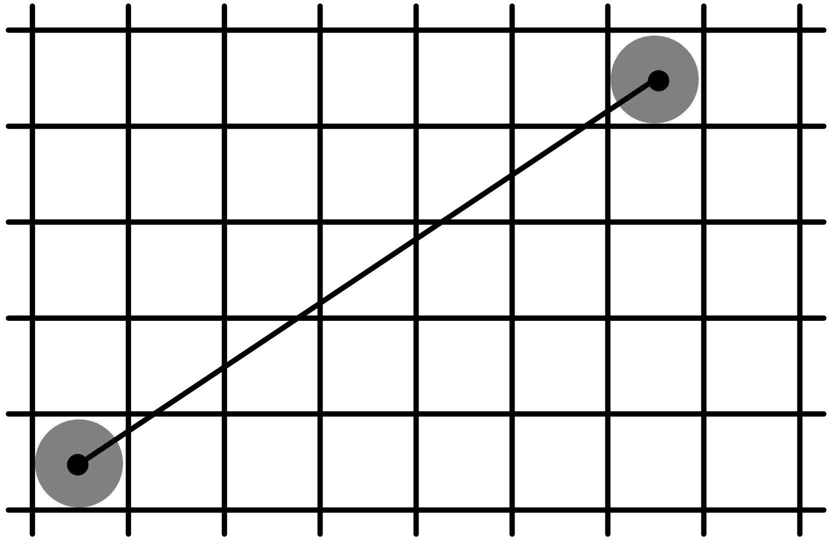




Raster conversion Algorithms: Requirements

- visual accuracy
- spatial accuracy
- speed

Line – Raster Representation



Basis for Line Drawing Algorithms

- The Cartesian slope-intercept for a straight line is
  - **y= m.x + b** ------------------→ 1
    - **m= Slope of the line**
    - **b= y intercept**
- The slope of the line is defined as

  $m=(y_2-y_1)/(x_2-x_1)$ ---------------→ 2
- The y intercept b is defined as

  b=y1- m . x1----------------------→3
- Algorithms for displaying straight lines are based on the above 3 eqns.
- For any given x interval Δx along a line, we compute the y-interval Δy.
  - **Δy=m . Δx** -----------------------------→4
  - Similarly we can obtain Δx .
    - **Δx= Δy/m** ----------------------------------→5
  - The eqns 4 and 5 form the basis for determining the deflection voltage in analog devices.

Cases to handle

- Case 1:**Slope |m| < 1**
  - Δx=small horizontal deflection voltage
  - Δy is calculated proportional to slope.(Δy =m. Δx)
- Case 2:**Slope |m| >1**
  - Δy=small vertical deflection voltage
  - Δx is calculated proportional to slope.(Δx =Δy/m)
- Case 3:**Slope m = 1**
  - Δx=Δy (Both the voltages are same)

Line drawing – DDA algorithm

- (DDA) is a scan-conversion line algorithm based on calculating either Δy or Δx.
  - $\Delta y = m * \Delta x$
  - $\Delta x = \Delta y / m$

- we have many cases based on sign of the slope, value of the slope, and the direction of drawing.
  - Slope sign: positive or negative.
  - Slope value: <= 1 or >1.
  - Direction: (left – right) or (right – left)

DDA – case 1

- **Positive slope and left to right:**

- If slope <= 1 then:

  $\mathbf{x_{k+1} = x_k + 1}$

  $\mathbf{y_{k+1} = y_k + m}$

- If slope > 1 then:

  $\mathbf{x_{k+1} = x_k + 1/m}$

  $\mathbf{y_{k+1} = y_k + 1}$

DDA – case 2

- **Positive slope and right to left:**

- If slope <= 1 then:

  $$\mathbf{x_{k+1} = x_k - 1}$$

  $$\mathbf{y_{k+1} = y_k - m}$$

- If slope > 1 then:

  $$\mathbf{x_{k+1} = x_k - 1/m}$$

  $$\mathbf{y_{k+1} = y_k - 1}$$

DDA – case 3

- **Negative slope and left to right:**

- If $|m| <= 1$ then:

  $\mathbf{x_{k+1} = x_k + 1}$

  $\mathbf{y_{k+1} = y_k - m}$

- If $|m| > 1$ then:

  $\mathbf{x_{k+1} = x_k + 1/m}$

  $\mathbf{y_{k+1} = y_k - 1}$

DDA – case 4

- **Negative slope and right to left:**

- If $|m| <= 1$ then:

  $$\mathbf{x_{k+1} = x_k - 1}$$

  $$\mathbf{y_{k+1} = y_k + m}$$

- If $|m| > 1$ then:

  $$\mathbf{x_{k+1} = x_k - 1/m}$$

  $$\mathbf{y_{k+1} = y_k + 1}$$

Digital Differential Algorithm

- Input line endpoints, (x_1,y_1) and (x_2, y_2)
- set pixel at position (x_1,y_1)
- calculate slope $m=(y_2-y_1)/(x_2-x_1)$
- **For +ve slope (left to right)**
  - **Case $|m| \le 1$**: Sample at unit x intervals and compute each successive y.
  - Repeat the following steps until (x_2, y_2) is reached:

    $\mathbf{y_{k+1} = y_k + m}$ where$(m=\Delta y/ \Delta x)$

    $\mathbf{x_{k+1} = x_k + 1}$

    set pixel at position $\mathbf{(x_{k+1},Round(y_{k+1}))}$
  - **Case $|m|>1$**: Sample at unit y intervals and compute each successive x.
  - Repeat the following steps until (x_2, y_2) is reached:

    $\mathbf{x_{k+1} = x_k + 1/m}$

    $\mathbf{y_{k+1} = y_k + 1}$

    set pixel at position $\mathbf{(Round(x_{k+1}), y_{k+1})}$

Digital Differential Algorithm

- **For +ve slope (right endpoint to left endpoint)**
  - **Case |m|≤1**: Sample at unit x intervals and compute each successive y.
  - Repeat the following steps until (x_2, y_2) is reached:

    $\mathbf{y_{k+1} = y_k - m}$ where$(m=\Delta y/ \Delta x)$

    $\mathbf{x_{k+1} = x_k - 1}$

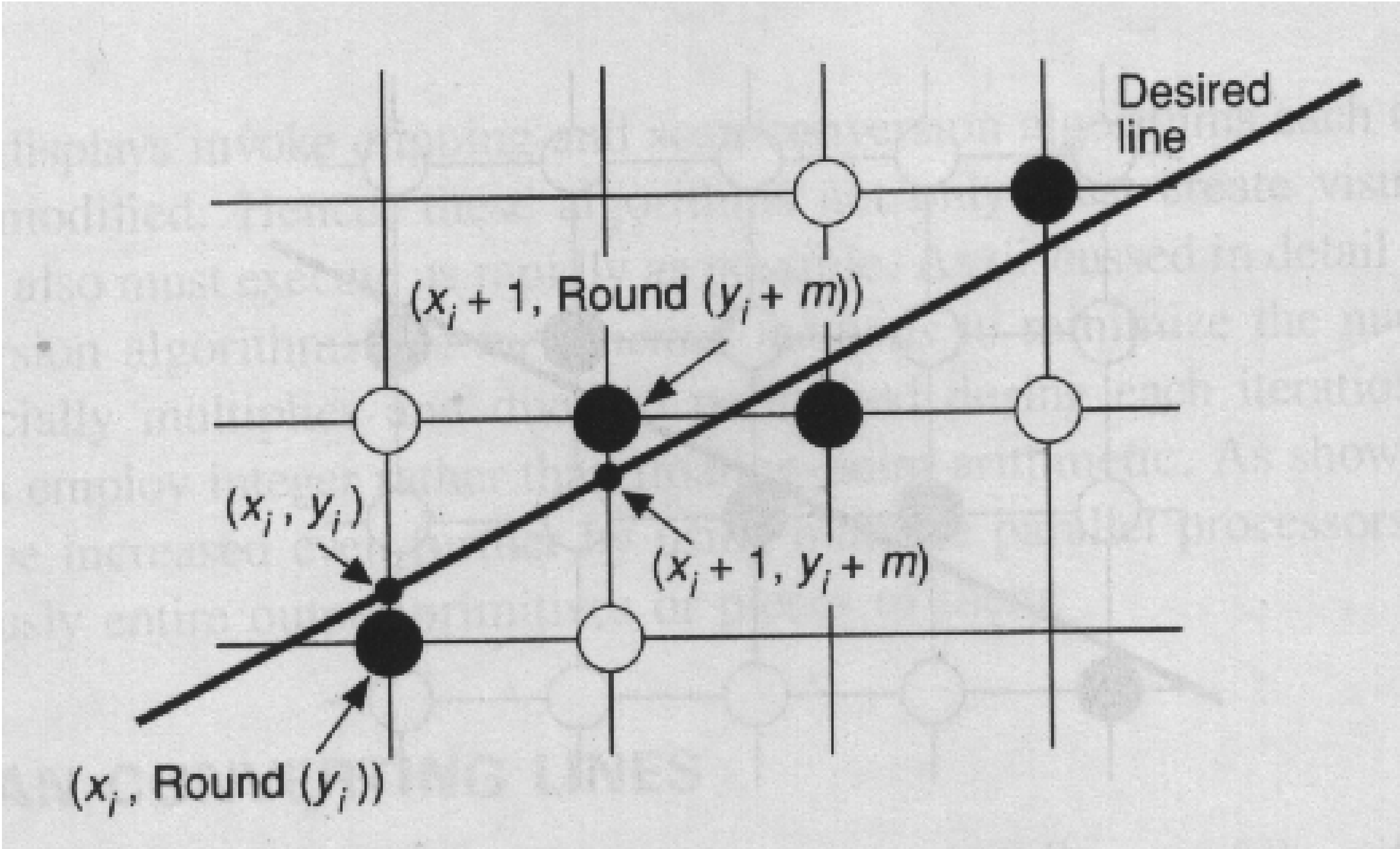
    set pixel at position $(x_{k-1},\text{Round}(y_{k-1}))$
  - **Case |m|>1**: Sample at unit y intervals and compute each successive x.
  - Repeat the following steps until (x_2, y_2) is reached:

    $\mathbf{x_{k+1} = x_k - 1/m}$

    $\mathbf{y_{k+1} = y_k - 1}$

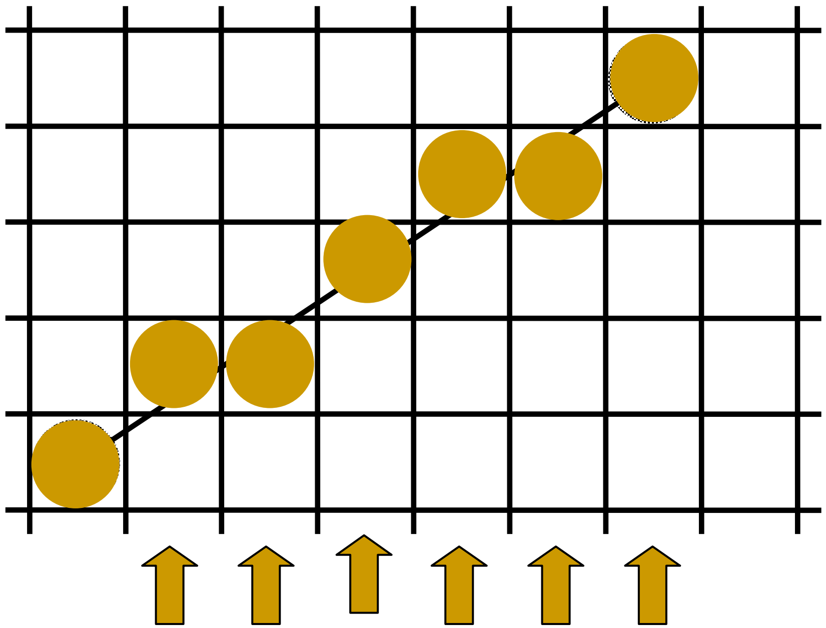
    set pixel at position $(\text{Round}(x_{k-1}), y_{k-1})$

Scan Conversion Process



DDA (Digital Differential Algorithm)

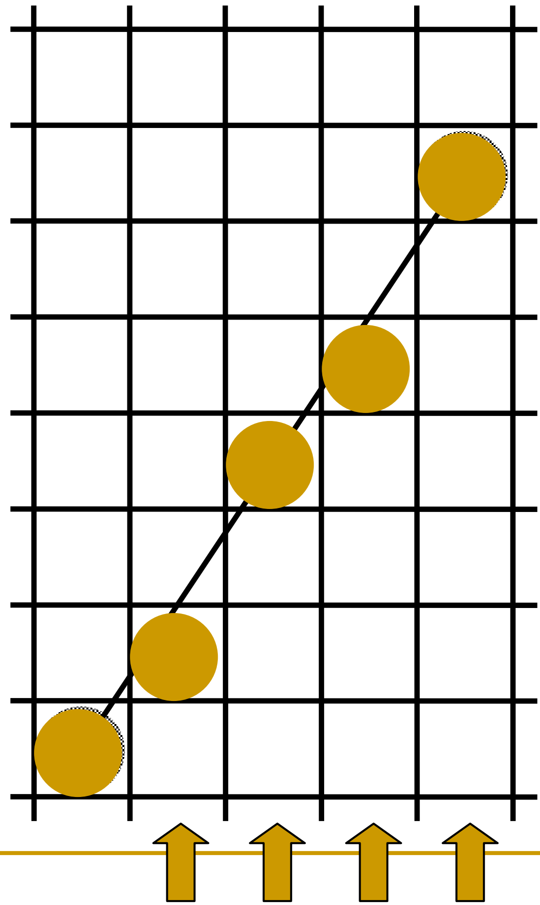
m < 1



- Δx=small horizontal deflection voltage (Δx=1)
- Δy proportional to slope m (Δy= Δx . m)

DDA (Digital Differential Algorithm)

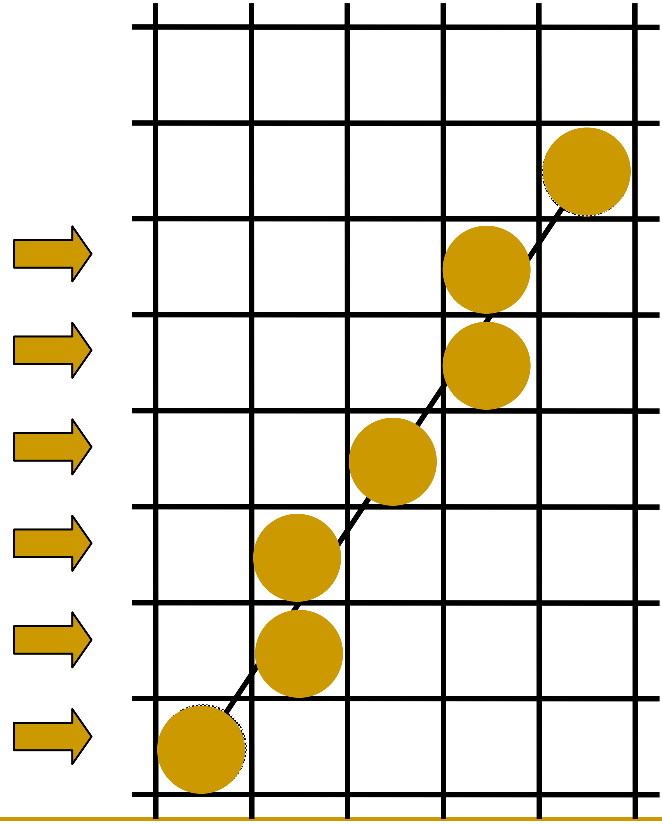
m > 1



- Δy=small horizontal deflection voltage (Δy=1)
- Δx proportional to slope m (Δx= Δy/m)

DDA (Digital Differential Algorithm)

m > 1



DDA Line algorithm - Procedure

- Procedure lineDDA(xa,xb,ya,yb:integer);
- Var
  - dx,dy,steps,k : integer;
  - xIncrement, yIncrement, x , y: real;
  - Begin
    - dx:=xb-xa;
    - dy:=yb-ya;
    - if abs(dx) >abs(dy) then steps:=abs(dx)
    - else steps:=abs(dy)
    - xIncrement :=dx/steps;
    - yIncrement :=dy/steps;
    - x:=xa;
    - y:=ya;

DDA Line algorithm – Procedure contd.

- setPixel(round(x),round(y),1);
- for k:=1 to steps do
- Begin
  - x:=x+xIncrement;
  - y:=y+yIncrement;
  - setPixel(round(x), round(y),1);

  End

End {lineDDA}

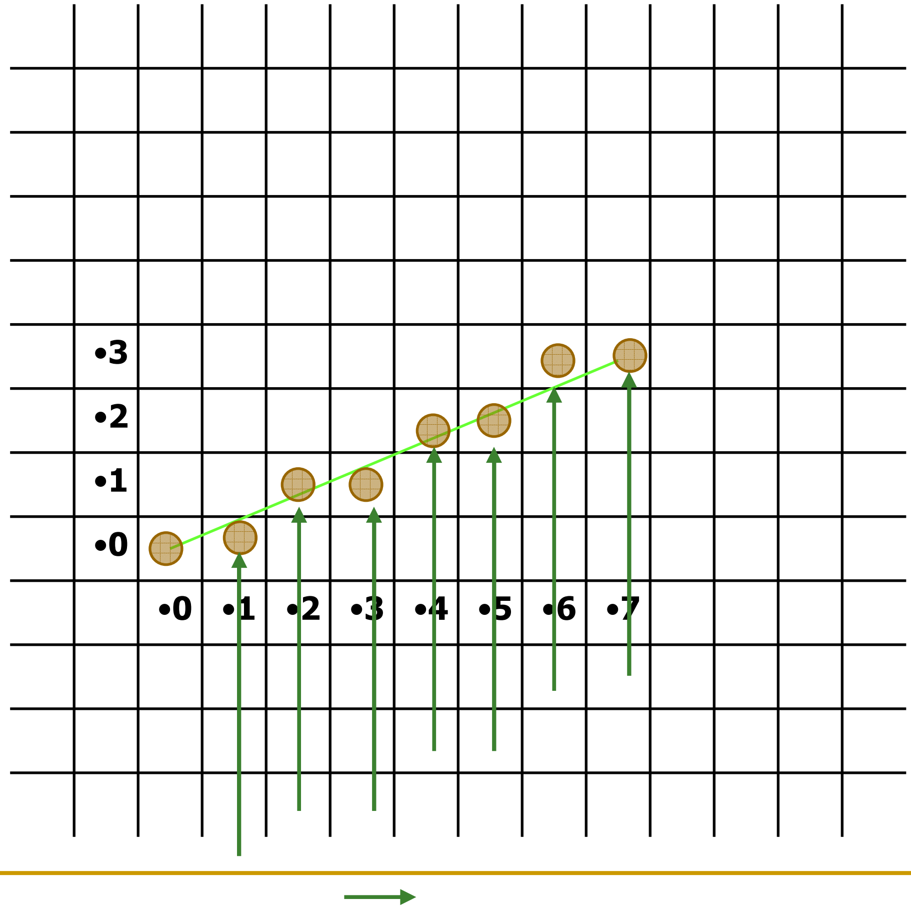
Example

- **Consider endpoints:** P1 (0,0)   P2 (7,3)
- $m = (3 – 0)/(7 - 0)$

  $= 0.429$ ($m<1$)(+ve slope)
- $dx = 1$
- $x_0 = 0$, $y_0 = 0$
- $x_1 = x_0 + 1 = 1$
- $y_1 = y_0 + 0.429$

  $= 0.429 \approx 0$
- $x_1 = 1$, $y_1 = 0.429$
- $x_2 = x_1 + 1 = 2$
- $y_2 = y_1 + 0.429$

  $= 0.859 \approx 1$



Example contd.

- $\mathbf{x_2 = 2, y_2 = 0.858}$
- $x_3 = x_2 + 1 = 3$
- $y_3 = y_2 + 0.429$

  $= 1.287 \approx 1$
- $\mathbf{x_3 = 3, y_3 = 1.287}$
- $x_4 = x_3 + 1 = 4$
- $y_4 = y_3 + 0.429$

  $= 1.716 \approx 2$
- $\mathbf{x_4 = 4, y_4 = 2}$
- $x_5 = x_4 + 1 = 5$
- $y_5 = y_4 + 0.429$

  $= 2.145 \approx 2$

Example contd.

- $\mathbf{x_5 = 5, y_5 = 2}$
- $x_6 = x_5 + 1 = 6$
- $y_6 = y_5 + 0.429$

  $= 2.574 \approx 3$
- $\mathbf{x_6 = 6, y_6 = 3}$
- $x_7 = x_6 + 1 = 7$
- $y_7 = y_6 + 0.429$

  $= 3.003 \approx 3$

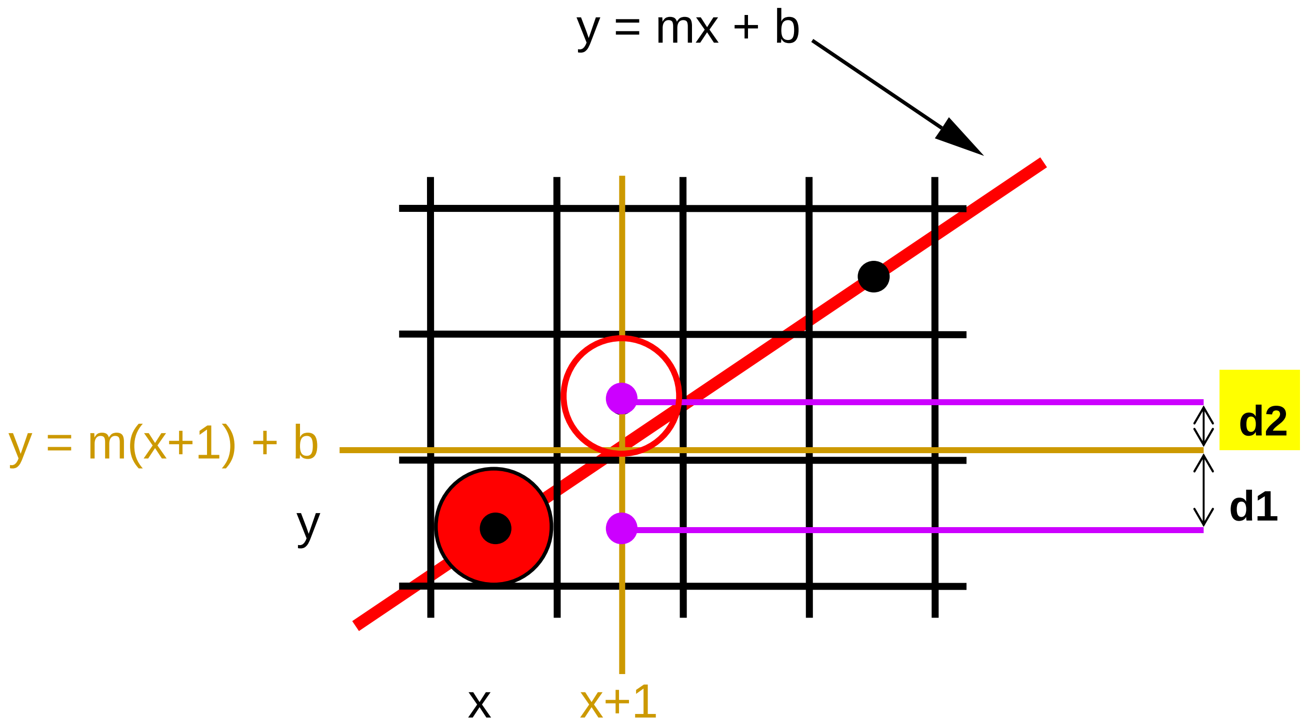
  $\mathbf{x_7=7,y_7=3}$

Disadvantages of DDA algorithm

- DDA works with floating point arithmetic
- Rounding to integers necessary

Bresenham's Line Algorithm

- An accurate and efficient raster line generating-algorithms developed by Bresenham.
- Scan converts lines using only incremental integer calculations.



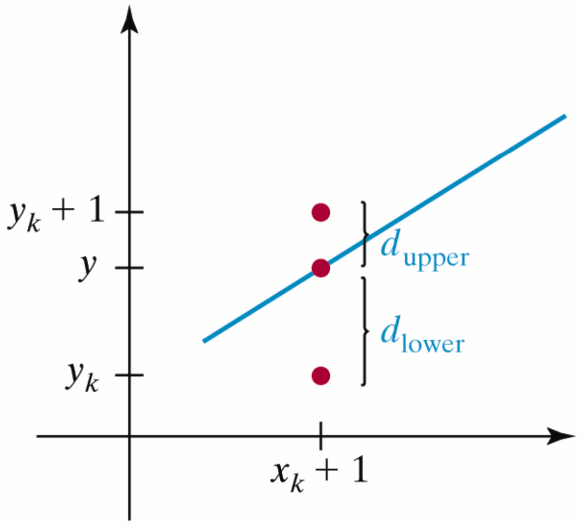


Figure 3-11

Vertical distances between pixel positions and the line y coordinate at sampling position $x_k + 1$.

Bresenham's line algorithm (slope ≤ 1)

- Input line endpoints, (x_0,y_0) and (x_n, y_n)
- calculate $\boldsymbol{\Delta x = x_n - x_0}$ and $\boldsymbol{\Delta y = y_n - y_0}$
- Assuming the pixel (x_k , y_k) is to displayed
  - Determine the positions whether at $\boldsymbol{(x_k+1 , y_k)}$ **and** $\boldsymbol{(x_k+1 , y_k+1)}$
- From x_k+1 we label vertical pixel separations from line as d1 and d2
- The y coordinate at pixel column position x_k+1 is calculated as
  - $\boldsymbol{y=m(x_k+1)+b.}$
  - **Then**
  - $\mathbf{d1}=y-y_k=m(x_k+1)+b-y_k$

and

- $\mathbf{d2}=y_k+1-y=y_k+1- m(x_k+1)-b.$

Bresenham's line algorithm (slope ≤ 1)

- d1-d2=2 $\mathbf{m(x_k+1)-2y_k+2b-1}$
- Decision parameter p_k for the kth step in line algorithm obtained by rearranging the above equations.
- Substitute $m=\Delta y/\Delta x$, where Δy, Δx are horizontal and vertical seperations.
  - $P_{k=}\ \Delta x(d1-d2)== \mathbf{2\ \Delta y.\ x_k - 2\ \Delta x\ .\ y_k + c}$**------------------→1**
  - C is constant has the value $2\ \Delta y+\ \Delta x(2b-1)$
  - When P_k is negative plot the lower pixel else plot the upper pixel.
- Coordinate changes along the line occur in unit steps in x and y directions.
- The values of successive decision parameter can be evaluated from
  - $P_{k+1=}\mathbf{2\ \Delta y.\ x_{k+1} - 2\ \Delta x\ .\ y_{k+1} + c}$**----------------------------→2**
- Subtracting 1 & 2 we get
  - $P_{k+1}=P_k+\ \mathbf{2\ \Delta y\text{-}\ 2\ \Delta x\ (y_{k+1}-y_k)}$ **(since $\mathbf{x_{k+1}=x_k+1}$)**
  - The term $(y_{k+1}-y_k)$ is either 1 or 0 depending on the sign of parameter p_k
  - The recursive calculation of decision parameters is performed at each integer x position.
- The parameter p_0 is evaluated from eq1 $p_0=2\ \mathbf{\Delta y\text{-}\ \Delta x}$

Bresenham's Line Algorithm

- Input line endpoints, (x_0,y_0) and (x_n, y_n)
- Load (x_0,y_0) into the frame buffer that is first point
- Calculate the constants Δx, Δy, $2\Delta y$ and $2\Delta y - 2\Delta x$
- calculate parameter $p_0 = 2\,\Delta y - \Delta x$
- Set pixel at position (x_0,y_0)
- repeat the following steps until (x_n, y_n) is reached:
- if $p_k < 0$
- set the next pixel at position $(x_k +1, y_k)$
- calculate new $p_{k+1} = p_k + 2\,\Delta y$
- if $p_k \geq 0$
- set the next pixel at position $(x_k +1, y_k + 1)$
- calculate new $pk_{+1} = p_k + 2(\Delta y - \Delta x)$
- Repeat last step Δx times.

Advantages of Bresenham's Line Algorithm

- Bresenham’s algorithm uses integer arithmetic
- Constants need to be computed only once
- Bresenham’s algorithm generally faster than DDA

Bresenham's line drawing Procedure

- procedure lineBres (xa, ya, xb, yb : integer)
- var
  - dx, dy, x, y, xEnd, p: integer;
- begin
  - dx :=abs(xa-xb);
  - dy :=abs(ya-yb);
  - p:=2 * dy – dx;
  - if xa > xb then
    - begin
      - x:= xb;
      - y:= yb;
      - xEnd := xa;
  - else
  - begin
    - x:=xa;
    - y:=ya;

Contd…

- xEnd := xb;
- end;
- setPixel (x,y,1);
- while x < xEnd do
- begin
  - x:= x+1;
  - if p < 0 then p:= p+2 * dy
  - else

    begin

        y:=y+1;

    p:=p+2 * (dy-dx)

    end;

    setPixel (x,y,1);

    end

    end; {lineBres}

Bresenham's Line Algorithm (Example)

- Note: Bresenham's algorithm is used when slope is <= 1.

- using Bresenham's Line-Drawing Algorithm, Digitize the line with endpoints (20,10) and (30,18).

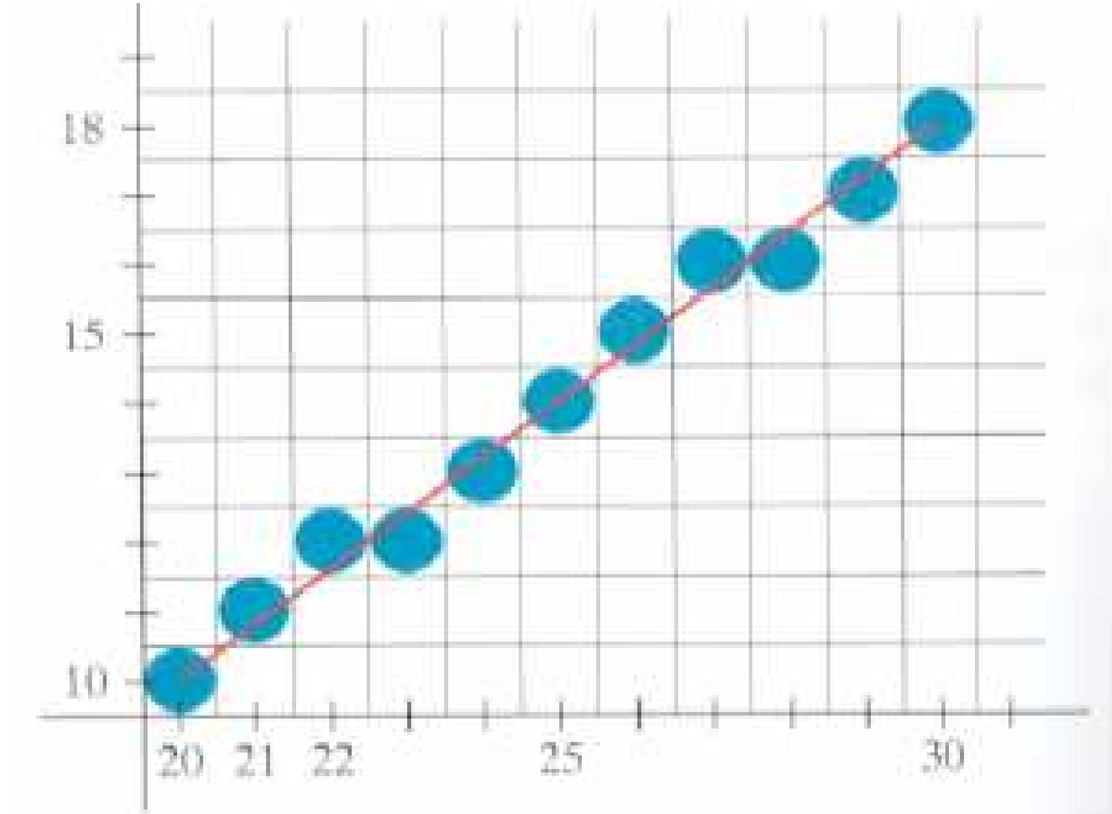
- $\Delta y = 18 - 10 = 8$
- $\Delta x = 30 - 20 = 10$
- $m = \Delta y / \Delta x = 0.8$

- plot the first point (x0, y0) = (20, 10)
- $p0 = 2 * \Delta y - \Delta x = 2 * 8 - 10 = 6$, so the next point is (21, 11)

Example (cont.)

K	P_k	(x_{k+1}, y_{k+1})	K	P_k	(x_{k+1}, y_{k+1})
0	6	(21,11)	5	6	(26,15)
1	2	(22,12)	6	2	(27,16)
2	-2	(23,12)	7	-2	(28,16)
3	14	(24,13)	8	14	(29,17)
4	10	(25,14)	9	10	(30,18)

Example (cont.)



- Thank you