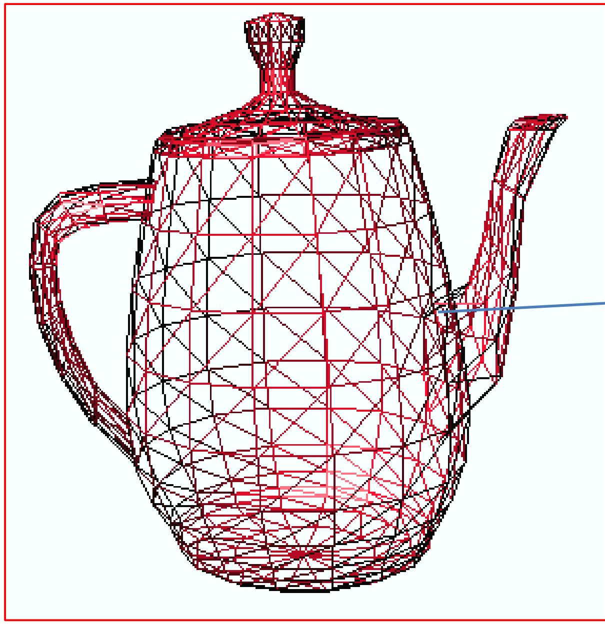
Subject : Computer Graphics
Subject_code : CS-2011
Course : B.Tech.(IV Sem.)

By
Poonam Saini
Department of Computer Science & Engineering
Sir Padampat Singhania University
Udaipur

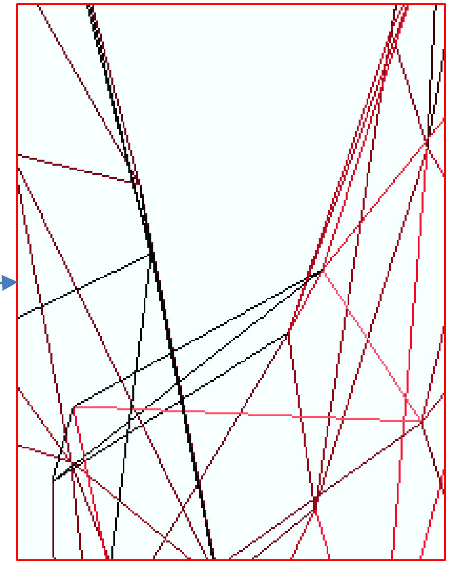
Output Primitives

Topics covered in this presentation:

- **Line Drawing**
- **Horizontal Line**
- **Vertical Line**
- **Scan converting a point and Line**
- **DDA algorithm for Line**
- **Bresenham’s Line drawing algorithm**
- **Bresenham’s Circle generation algorithm**
- **Mid Point Circle generation algorithm**
- **Mid Point Ellipse generation algorithm**



The lines of this object appear ***continuous***



However, they are ***made of pixels***

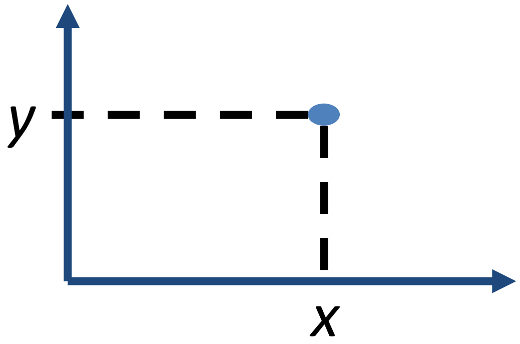
Points and Lines

•Point plotting is accomplished by converting a single coordinate position furnished by an application program into appropriate operations for the output device in use.

•With a CRT monitor, for example, the electron beam is turned on to illuminate the screen phosphor at the selected location.

Points

- **Single Coordinate Position**
  - Set the bit value(color code) corresponding to a specified screen position within the frame buffer



setPixel (*x*, *y*)

Lines

- Line drawing is accomplished by calculating intermediate positions along the line path between specified end points.
- An output device is then directed to fill in these positions between the endpoints.

- *Precise definition of line drawing*

Given two points P and Q in the plane, both with integer coordinates, determine which pixels on a raster screen should be *on* in order to make a picture of a unit-width line segment starting from P and ending at Q.

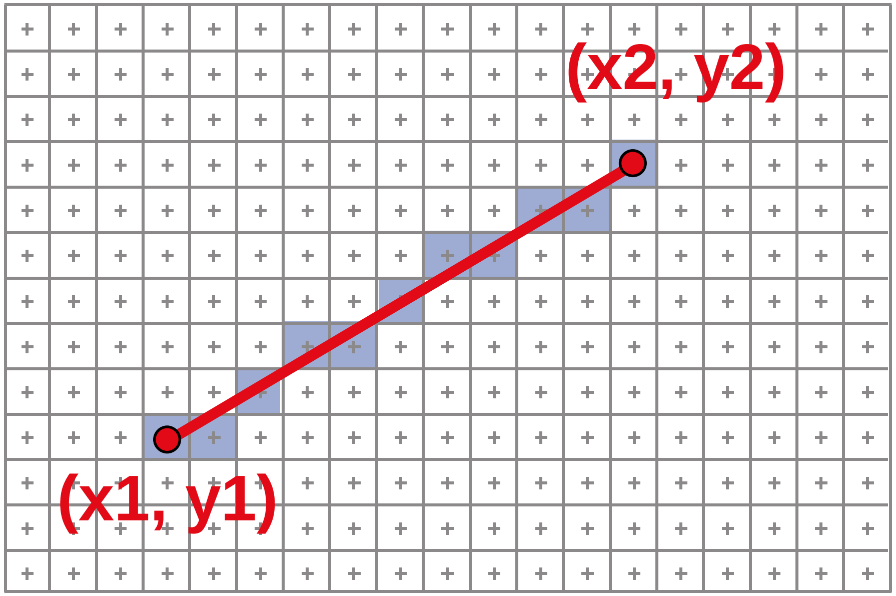
Scan Converting 2D Line Segments

Given:

- Segment endpoints (integers x1, y1; x2, y2)

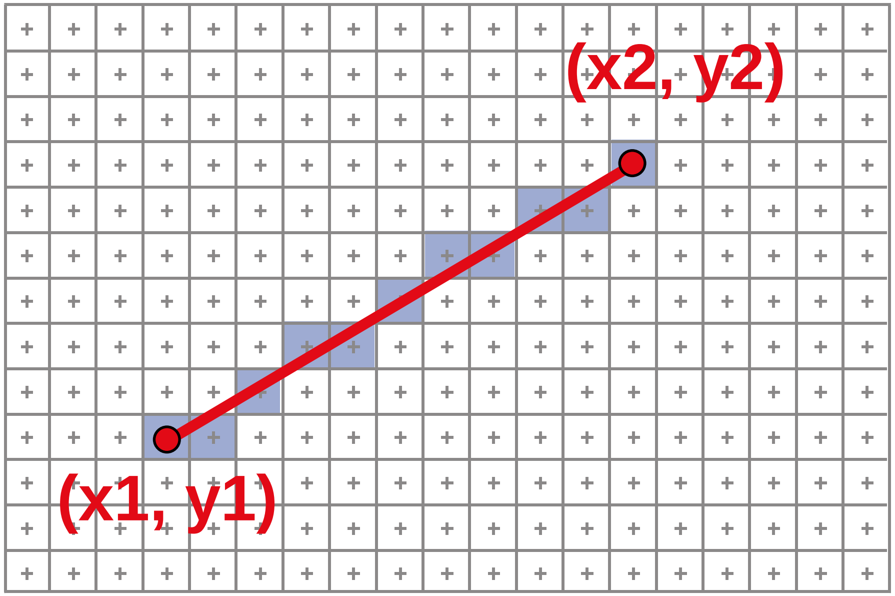
Identify:

- Set of pixels (x, y) to display for segment



Line Rasterization Requirements

- Transform continuous primitive into discrete samples
- Uniform thickness & brightness
- Continuous appearance
- No gaps
- Accuracy
- Speed



Line Drawing

<u>Horizontal Line</u>

- The horizontal line is obtained by keeping the value of y constant and repeatedly incrementing the x value by one unit.

- The following pseudo-code draw a horizontal line from

  (xstart,y) to (xend,y), xstart <= xend

  for (x=xstart; x<= xend ; x++) **do**

  putpixel(x,y,8);

If xstart>xend, in the **for** loop you must start from reverse order (high to low)

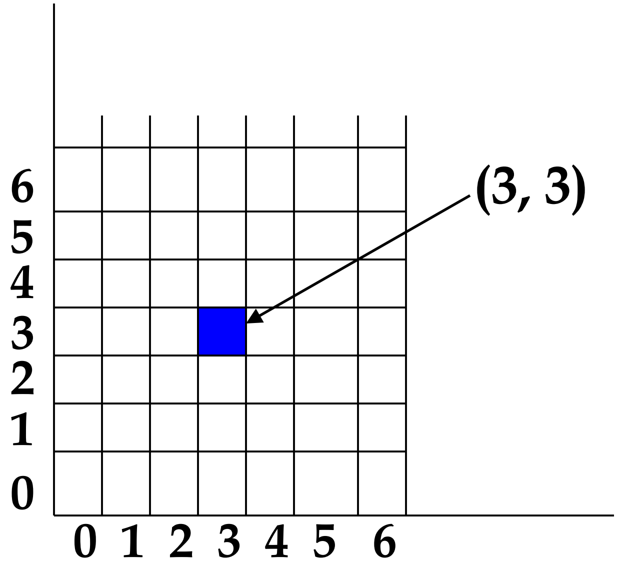
Line Drawing

The vertical line

- It is obtained by keeping the value of x constant and repeatedly incrementing the y value by one unit.
- The following code draw a vertical line from (x,ystart) to (x,yend), ystart <= yend.

```
for (y=ystart ; y<=yend ;y++) do
      putpixel(x,y,8);
```

If ystart>yend, the **for** loop must be replaced by in reserve counter (high to low).



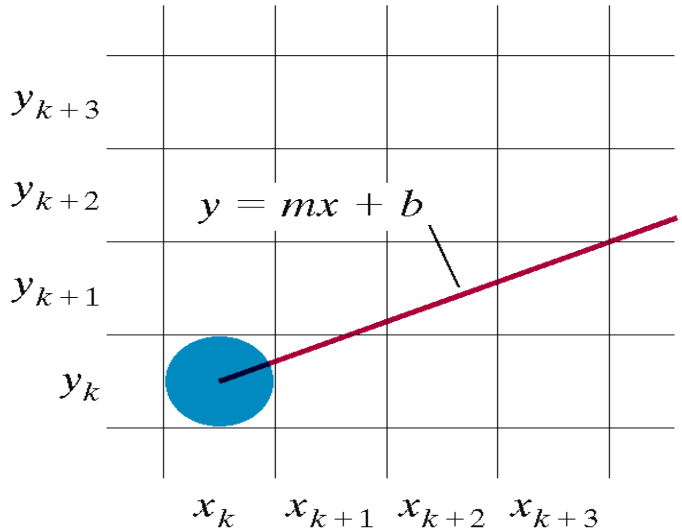


Figure 3-10

A section of the screen showing a pixel
in column x_k on scan line y_k that is to be plotted along the path of a line segment with slope $0 < m < 1$.

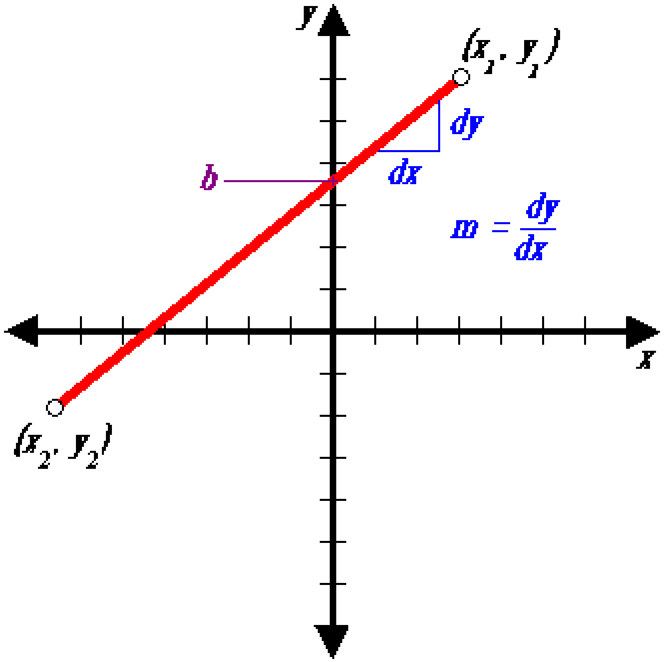
Scan Converting A Line

- The Cartesian slope- intercept equation for a straight line is:

$$\boxed{y = m \cdot x + b}$$

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

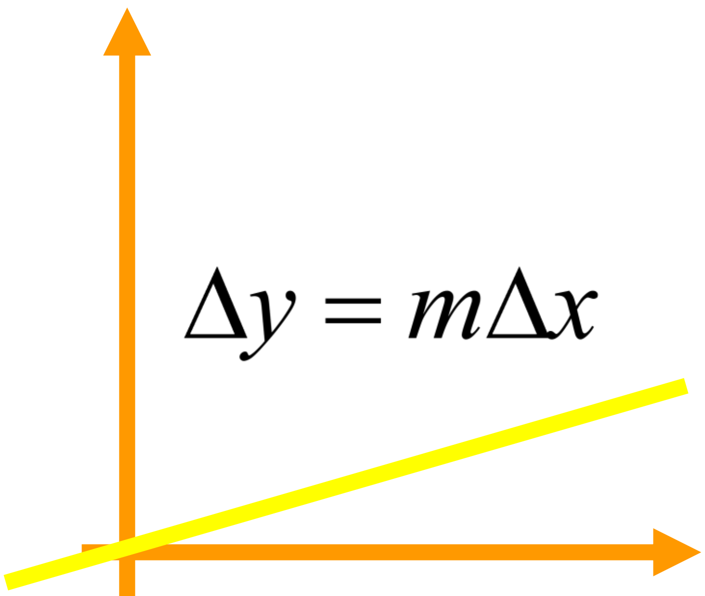
$$b = y_1 - m \cdot x_1$$



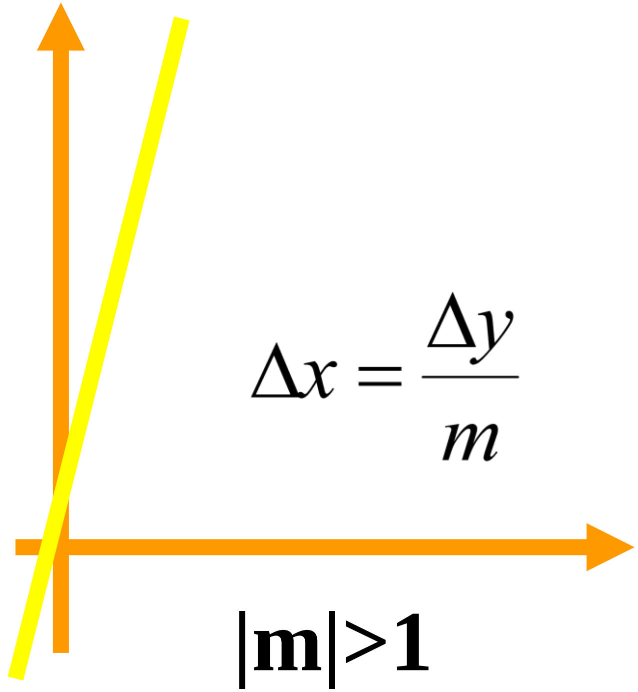
$$\Delta y = m \Delta x \qquad \Delta x = \frac{\Delta y}{m}$$

Scan Converting A Line

- These equation form the basic for determining deflection voltage in **analog devices**.



$|m|<1$



$|m|>1$

Line Drawing (cont)

- Also for any given x interval Δx along a line, we can compute the corresponding y interval Δy from

$$\Delta y = m . \Delta x$$

- Similarly we can obtain the x interval Δx corresponding to a specified Δy as

$$\Delta x = \Delta y \, / \, m$$

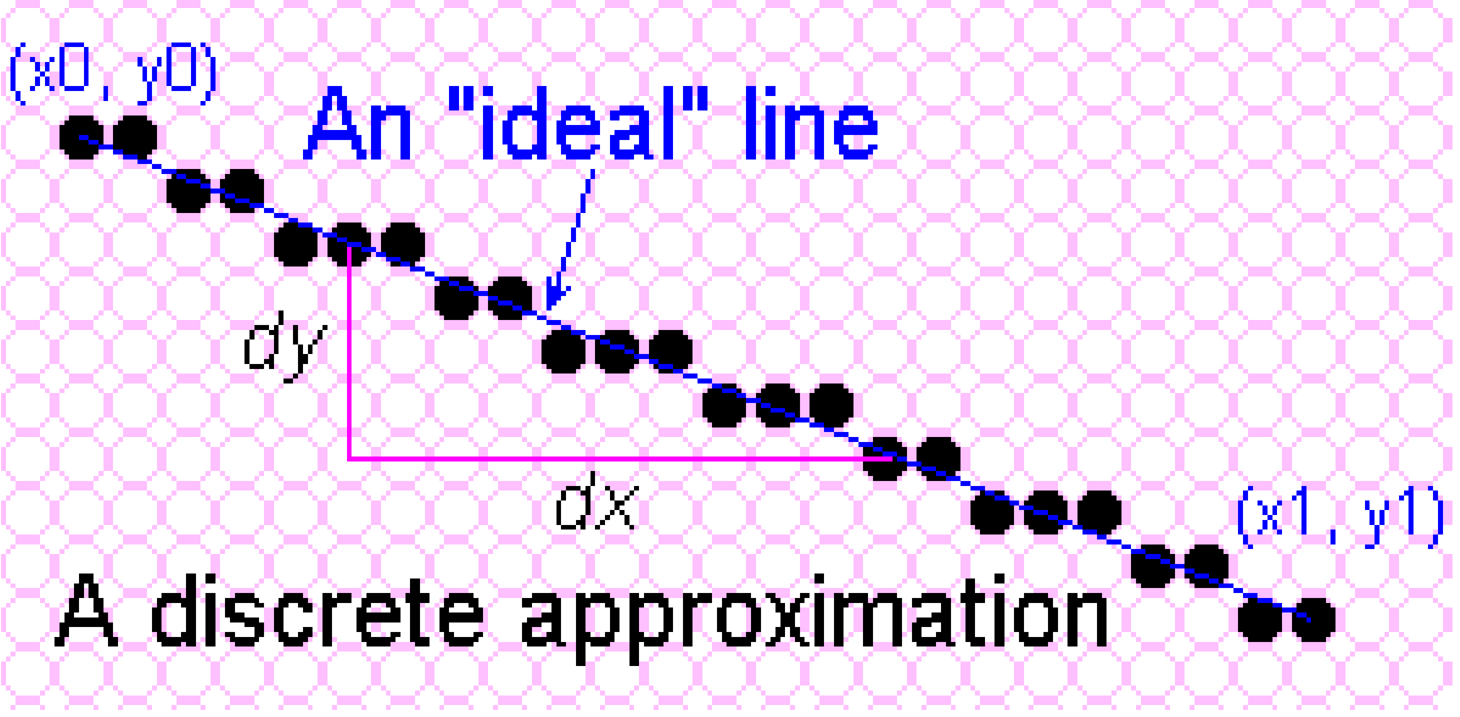
- These equations form the basis for determining deflection voltages in analog devices.

Line Drawing (cont)

- For lines with slope magnitudes $|m| < 1$, Δx can be set proportional to a small horizontal deflection voltage and the corresponding vertical deflection is then set proportional to Δy as calculated from Eq. $\Delta y = m . \Delta x$.
- For lines whose slopes have magnitudes $|m| > 1$, Δy can be set proportional to a small vertical deflection voltage with the corresponding horizontal deflection voltage set proportional to Δx, calculated from Eq. $\Delta x = \Delta y \,/\, m$.
- For lines with $m = 1$, $\Delta x = \Delta y$ and the horizontal and vertical deflections voltages are equal.

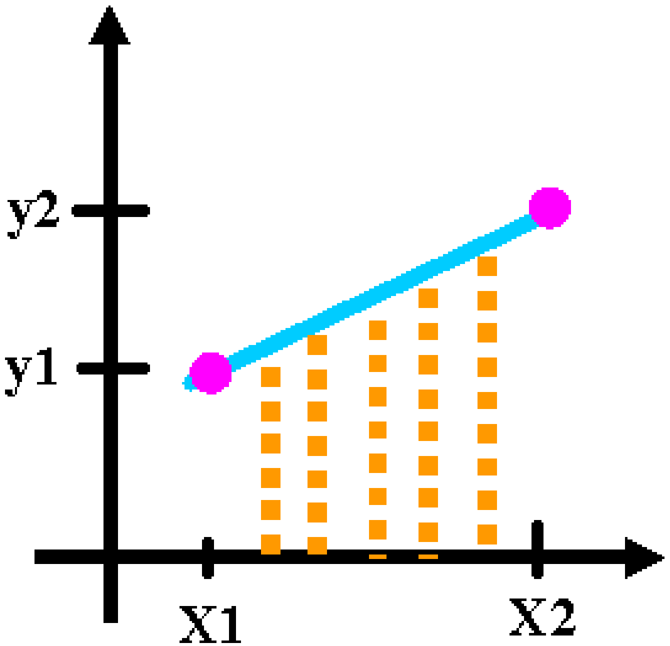
Scan Converting A Line

- On raster system, lines are plotted with pixels, and **step size** (horizontal & vertical direction) are constrained by **pixel separation**.



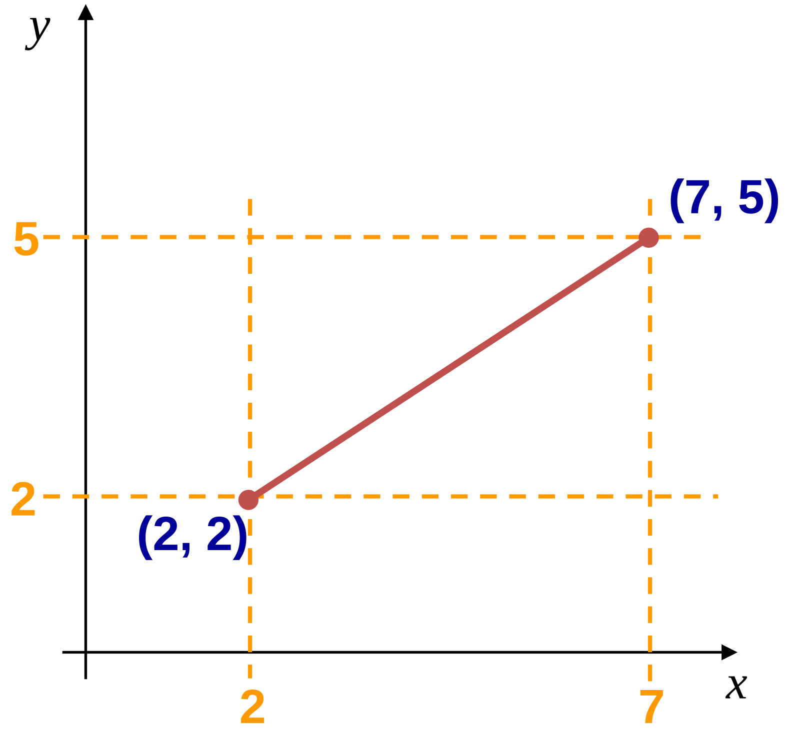
Scan Converting A Line

- We must ***sample*** a line at discrete positions and determine the nearest pixel to the line at each sampled position.

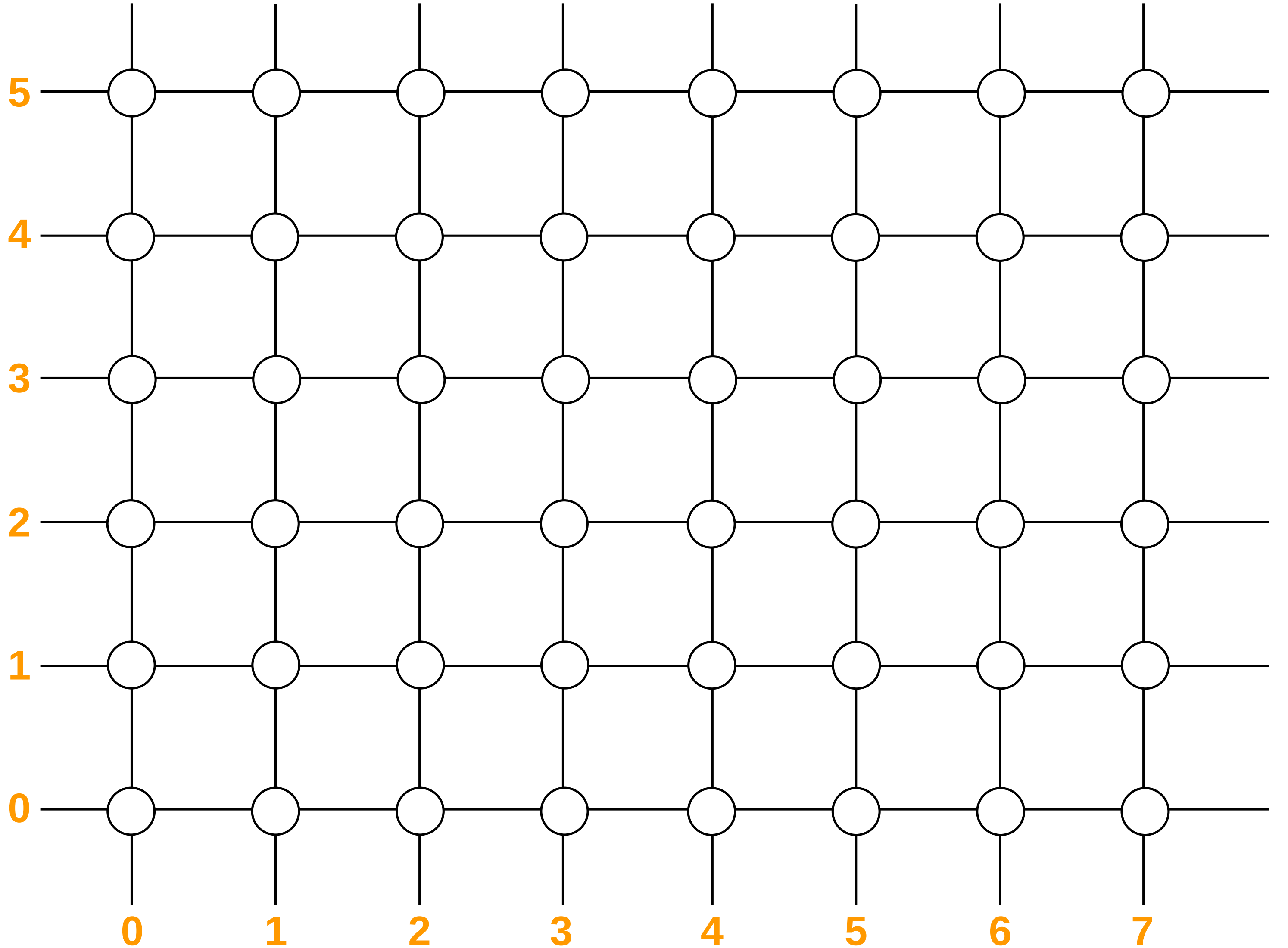


A Very Simple Solution

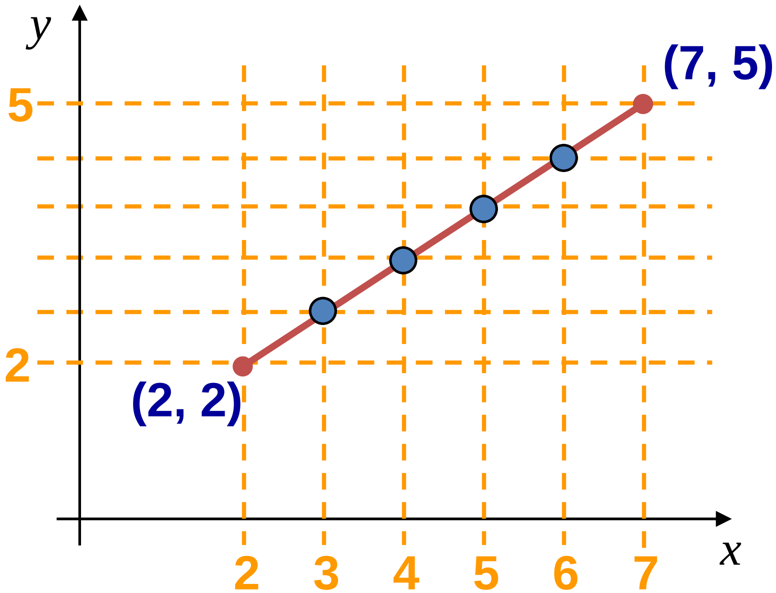
- We could simply work out the corresponding y coordinate for each unit x coordinate
- Let's consider the following example:



A Very Simple Solution (cont…)



A Very Simple Solution (cont…)



- First work out m and b:

$$m = \frac{5-2}{7-2} = \frac{3}{5}$$

$$b = 2 - \frac{3}{5} * 2 = \frac{4}{5}$$

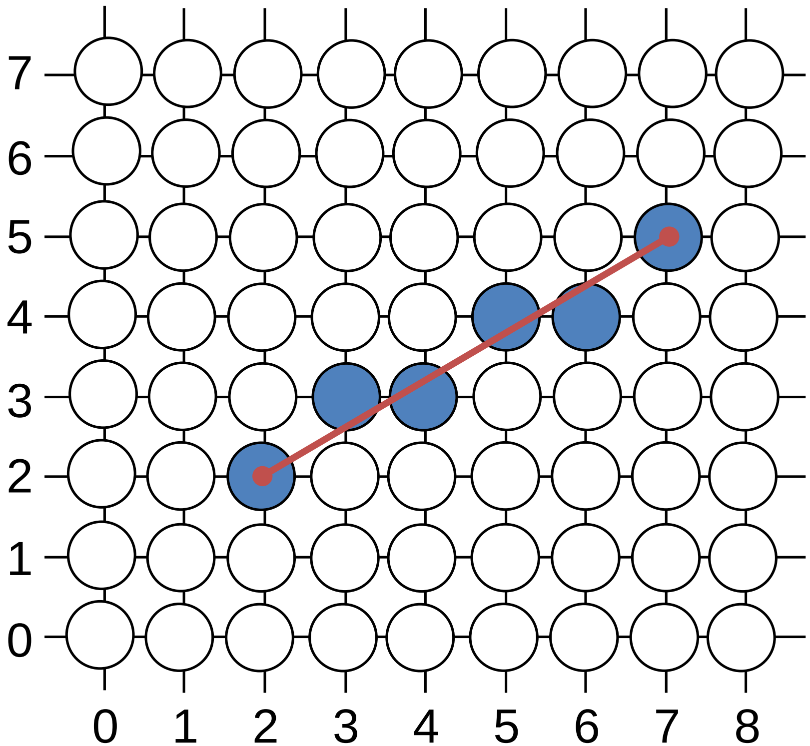
- Now for each x value work out the y value:

$$y(3) = \frac{3}{5} \cdot 3 + \frac{4}{5} = 2\frac{3}{5} \qquad y(4) = \frac{3}{5} \cdot 4 + \frac{4}{5} = 3\frac{1}{5}$$

$$y(5) = \frac{3}{5} \cdot 5 + \frac{4}{5} = 3\frac{4}{5} \qquad y(6) = \frac{3}{5} \cdot 6 + \frac{4}{5} = 4\frac{2}{5}$$

A Very Simple Solution (cont…)

- Now just round off the results and turn on these pixels to draw our line



$$y(3) = 2\frac{3}{5} \approx 3$$

$$y(4) = 3\frac{1}{5} \approx 3$$

$$y(5) = 3\frac{4}{5} \approx 4$$

$$y(6) = 4\frac{2}{5} \approx 4$$

A Very Simple Solution (cont…)

- However, this approach is just way too slow
- In particular look out for:
  - The equation $y = mx + b$ requires the multiplication of m by x
  - Rounding off the resulting y coordinates
- We need a faster solution

A Quick Note About Slopes

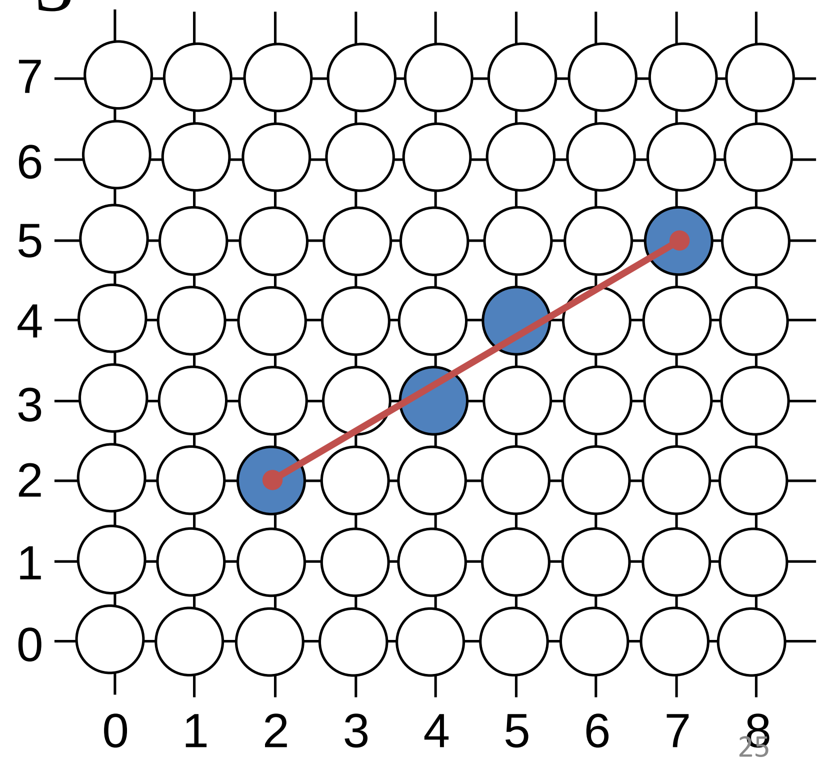
- In the previous example we chose to solve the parametric line equation to give us the *y* coordinate for each unit *x* coordinate
- What if we had done it the other way around?
- So this gives us: $$x = \frac{y - b}{m}$$
- where: $m = \dfrac{y_{end} - y_0}{x_{end} - x_0}$ and $b = y_0 - m \cdot x_0$

A Quick Note About Slopes (cont…)

- Leaving out the details this gives us:

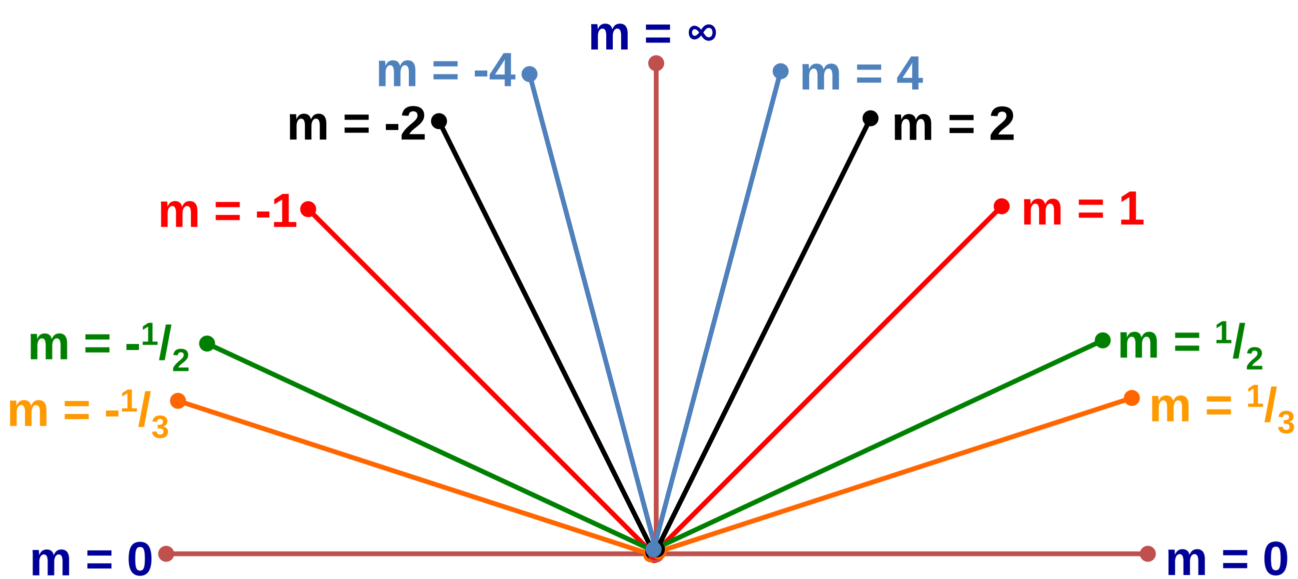
$$x(3) = 3\frac{2}{3} \approx 4 \qquad x(4) = 5\frac{1}{3} \approx 5$$

- We can see easily that this line doesn't look very good!
- We choose which way to work out the line pixels based on the slope of the line



A Quick Note About Slopes (cont…)

- If the slope of a line is between -1 and 1 then we work out the *y* coordinates for a line based on it's unit *x* coordinates
- Otherwise we do the opposite – *x* coordinates are computed based on unit *y* coordinates



Digital Differential Analyzer (DDA Algorithm)

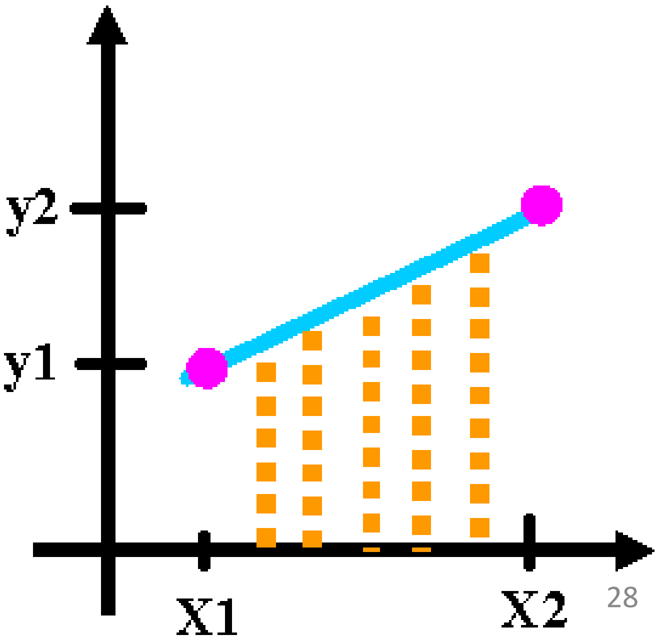
Digital Differential Analyzer Algorithm(DDA)

- Algorithm is an incremental scan conversion method.
- Based on calculating either Δy or Δx

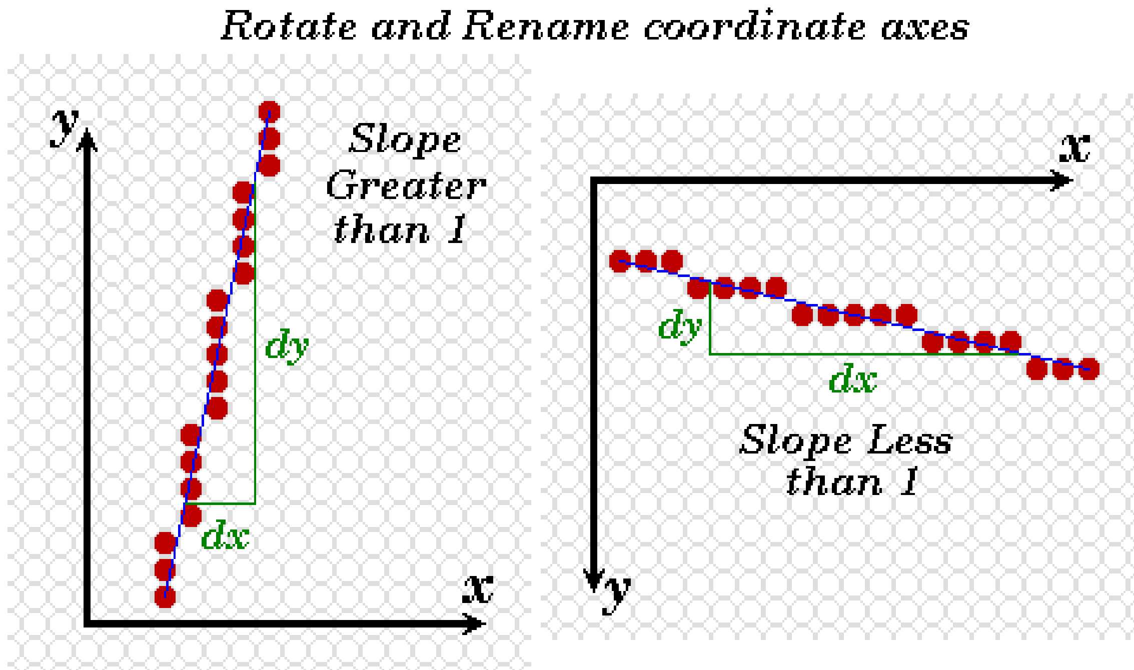
  If |m|<1,

$$\Delta y = m \Delta x$$

$$y_{K+1} = y_k + m$$



DDA Algorithm



If $|m|>1$, $(\Delta y = 1)$

$$\Delta x = \frac{\Delta y}{m}, x_{K+1} = x_k + \frac{1}{m}$$

For the above cases it is assumed that lines are to be processed from the left endpoint to the right endpoint.

If the process is reverse,

If $(\Delta x = -1)$

$$\Delta y = m\Delta x$$

$$y_{K+1} = y_k - m$$

If $(\Delta y = -1)$

$$\Delta y = m\Delta x$$

$$x_{K+1} = x_k - \frac{1}{m}$$

DDA Pseudo-code

```
Algorithm: DDA(float x1, float x2, float y1, float y2)
var dx, dy, steps, k: Integer
var xinc, yinc: real

Begin
Dx=x2-x1
Dy=y2-y1
If(abs(dx)>=abs(dy))
Then steps=abs(dx)
Else steps=abs(dy)
xinc = abs(dx/steps)
yinc = abs(dy/steps)
   x = x1;
   y = y1;
   setpixel(Round(x),Round(y),1);
   for k:=1 to steps
     do
      x :=x+ xinc;
      y :=y+ yinc;
      setpixel(Round(x),Round(y),1)
   end
end
```

Q: For each step, how many floating point operations are there?
A: 4

Q: For each step, how many integer operations are there?
A: 2

DDA Example

- 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?

steps = 12 – 2 = 10
xinc = 10/10 = 1.0
yinc = 5/10 = 0.5

k	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

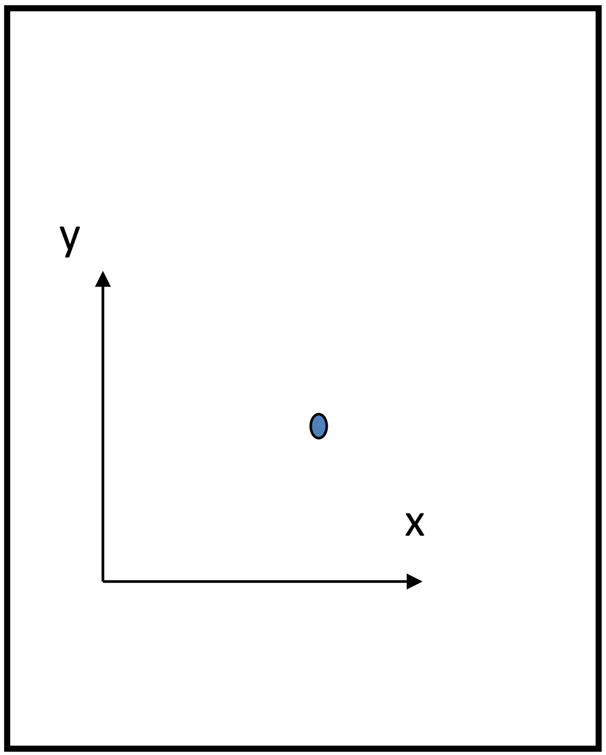
DDA ALGORITHM

- **Major advantages in the above approach :**
  - Faster method for calculating pixel positions than the direct use of Eq. $y=mx+c$
  - It eliminates the multiplication by making use of raster characteristics, so that appropriate increments are applied in the x or y direction to step to pixel positions along the line path.

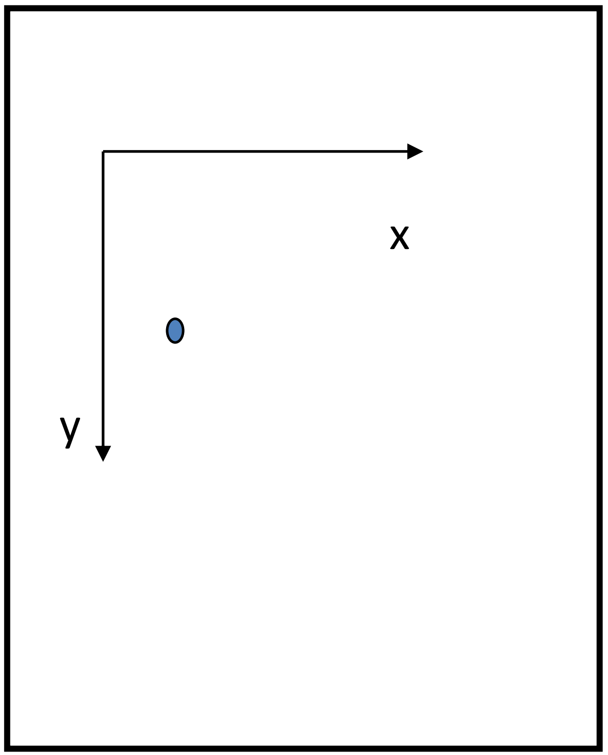
- **Major disadvantages in the this approach :**
  - The rounding operations and floating-point arithmetic in DDA algo. are still time-consuming.
  - We can improve the performance of the DDA algorithm by separating the increments m and l/m into integer and fractional parts so that all calculation are reduced to integer operations.

2D Cartesian Reference System

2D Cartesian Reference Frames



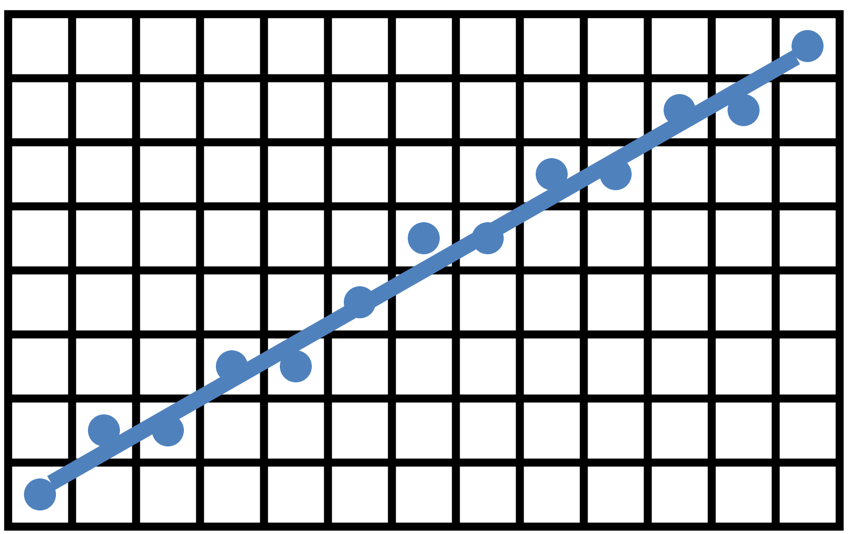
Coordinate origin at the lower-left screen corner



Coordinate origin in the upper-left screen corner

Lines

- **Intermediate Positions between Two Endpoints**
  - DDA, Bresenham's line algorithms



Jaggies = Aliasing

Bresenham’s Line Drawing Algorithm

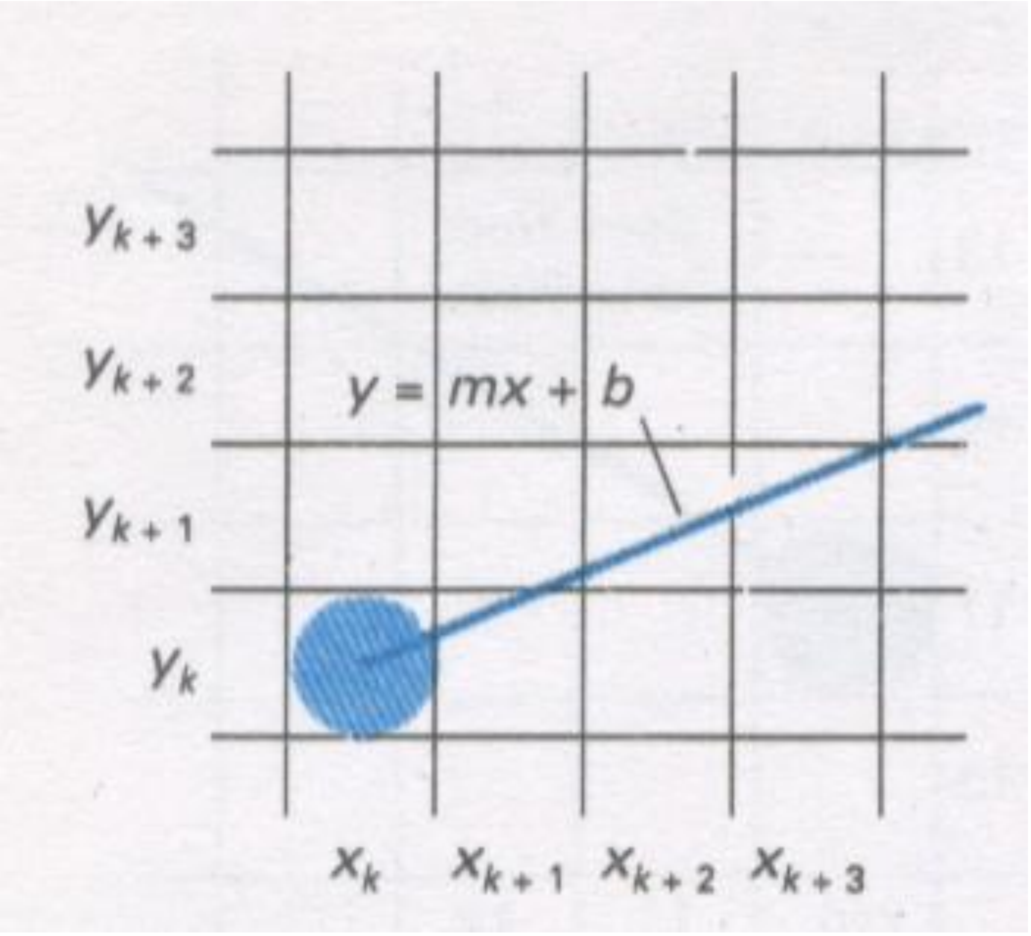
Bresenham's Line Algorithm

- An **accurate**, **efficient** raster line drawing algorithm developed by Bresenham, scan converts lines using only *incremental integer* calculations that can be adapted to display circles and other curves.

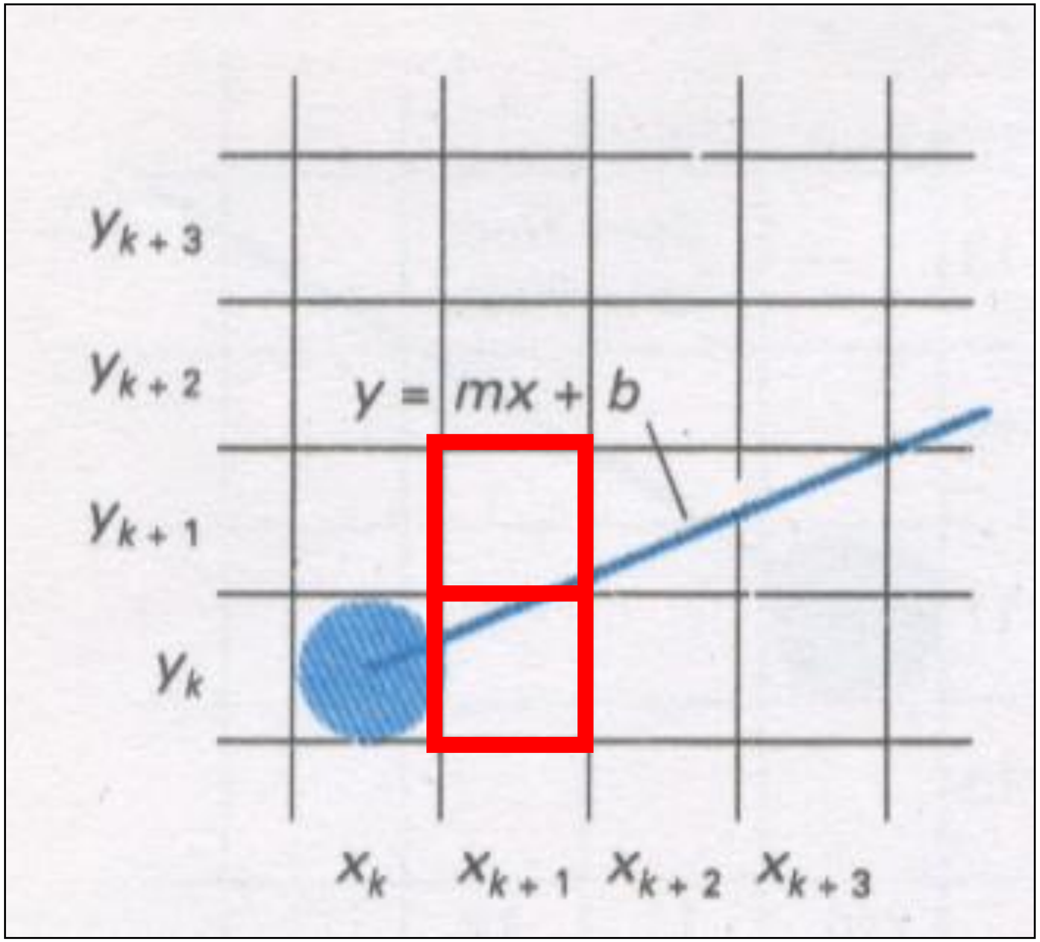
- Keeping in mind the symmetry property of lines, lets derive a more efficient way of drawing a line.

- Starting from the left end point (x_0,y_0) of a given line , we step to each successive column (x position) and plot the pixel whose scan-line y value closest to the line path
- Assuming we have determined that the pixel at (x_k,y_k) is to be displayed, we next need to decide which pixel to plot in column x_{k+1}.

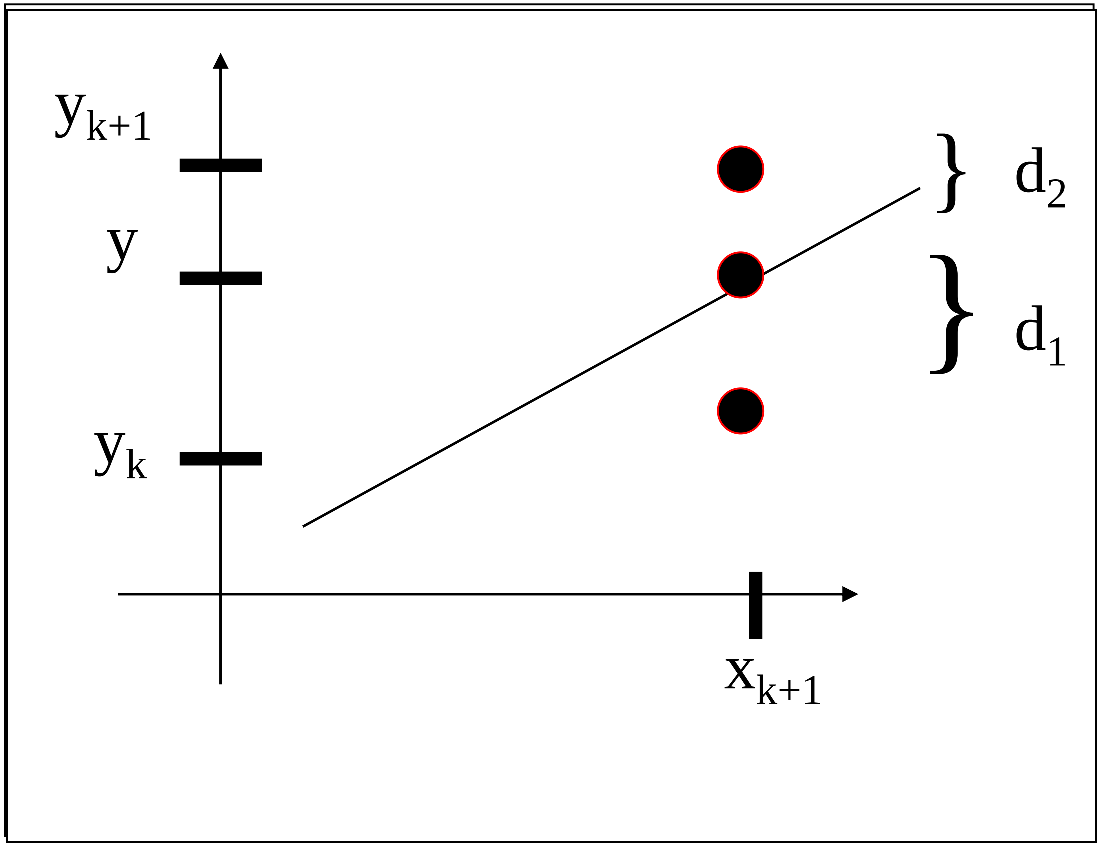
Bresenham's Line Algorithm



Bresenham's Line Algorithm



Bresenham's Line Algorithm



Bresenham's Line Algorithm

Choices are $(x_k + 1, y_k)$ and $(x_k + 1, y_K + 1)$

$$d_1 = y - y_k = m(x_k + 1) + b - y_k$$
$$d_2 = (y_k + 1) - y = y_k + 1 - m(x_k + 1) - b$$

- The difference between these 2 separations is

$$d1 - d2 = 2m(x_k + 1) - 2\,y_k + 2b - 1$$

- A decision parameter p_k for the k^{th} step in the line algorithm can be obtained by rearranging above equation so that it involves only *integer calculations*

Bresenham's Line Algorithm

- Define

$$p_k = \Delta x\ (d_1 - d_2) = 2\Delta y x_k - 2\ \Delta x y_k + c$$

- The sign of p_k is the same as the sign of d_1-d_2, since $\Delta x > 0$.

  Parameter c is a constant and has the value $2\Delta y + \Delta x(2b-1)$ (independent of pixel position)

- If *pixel at* y_k is closer to line-path than pixel at $y_k +1$ *(i.e, if* $d_1 < d_2$*)* then p_k is negative. We plot lower pixel in such a case. Otherwise , upper pixel will be plotted.

Bresenham's Line Algorithm

Coordinate changes along the line occur in unit steps in either the x or y directions. Therefore, we can obtain the values of successive decision parameters using incremental integer calculations.

- At step $k + 1$, the decision parameter can be evaluated as,

$$p_{k+1} = 2\Delta y.x_{k+1} - 2\Delta x.y_{k+1} + c$$

- Taking the difference of p_{k+1} and p_k we get the following.

$$p_{k+1} - p_k = 2\Delta y.(x_{k+1} - x_k) - 2\Delta x.(y_{k+1} - y_k)$$

- But, $x_{k+1} = x_k + 1$, so that

$$p_{k+1} = p_k + 2\Delta y - 2\,\Delta x(y_{k+1} - y_k)$$

- Where the term $y_{k+1} - y_k$ is either 0 or 1, depending on the sign of *parameter* p_k

Bresenham's Line Algorithm

- The first parameter p_0 is directly computed

  $p_0 = 2\ \Delta y x_0 - 2\ \Delta x y_0 + c = 2\ \Delta y x_0 - 2\ \Delta x y_0 + 2\ \Delta y + \Delta x\ (2b-1)$

- Since (x_0, y_0) satisfies the line equation , we also have

  $y_0 = \Delta y / \Delta x * x_0 + b$

- Combining the above 2 equations , we will have

  $p_0 = 2\Delta y - \Delta x$

  The constants $2\Delta y$ and $2\Delta y - 2\Delta x$ are calculated once for each time to be scan converted

Bresenham's Line Algorithm

- So, the arithmetic involves only integer addition and subtraction of 2 constants

1. Input the two end points and store the left end point in $(x_{0,}y_0)$

2. Load (x_0,y_0) into the frame buffer ***(plot the first point)***

3. Calculate the constants Δx, Δy, $2\Delta y$ and $2\Delta y$-$2\Delta x$ and obtain the starting value for the decision parameter as

$$p_0 = 2\Delta y - \Delta x$$

Bresenham's Line Algorithm

4. At each x_k along the line, starting at k=0, perform the following test:

If $p_k < 0$, the next point is (x_k+1, y_k) and

$$p_{k+1} = p_k + 2\Delta y$$

Otherwise
Point to plot is (x_k+1, y_k+1)

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

5. Repeat step 4 (above step) Δx times

Example 3-1 Bresenham Line Drawing

To illustrate the algorithm, we digitize the line with endpoints (20, 10) and (30, 18). This line has a slope of 0.8, with

$$\Delta x = 10, \qquad \Delta y = 8$$

The initial decision parameter has the value

$$\begin{aligned} p_0 &= 2\Delta y - \Delta x \\ &= 6 \end{aligned}$$

and the increments for calculating successive decision parameters are

$$2\Delta y = 16, \qquad 2\Delta y - 2\Delta x = -4$$

We plot the initial point $(x_0, y_0) = (20, 10)$, and determine successive pixel positions along the line path from the decision parameter as

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)

Bresenham’s Line Algorithm

- Suppose we want to draw a line starting at pixel (2,3) and ending at pixel (12,8).
- What are the values of p0, dx and dy?
- What are the values of the variable p at each timestep?
- What are the pixels colored, according to Bresenham’s algorithm?

dx = 12 – 2 = 10
dy = 8 – 3 = 5
p0 = 2dy – dx = 15

2dy = 10
2dy – 2dx = -10

t	p	P(x)	P(y)
0	0	2	3
1	-10	3	4
2	0	4	4
3	-10	5	5
4	0	6	5
5	-10	7	6
6	0	8	6
7	-10	9	7
8	0	10	7
9	-10	11	8
10	0	12	8

How do we draw a circle?

Properties of a circle:

- **A circle is defined as a set of points that are all the given distance (x_c, y_c). This distance relationship is expressed by the pythagorean theorem in Cartesian coordinates as**

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

- **We could use this equation to calculate the points on the circle circumference by stepping along x-axis in unit steps from $x_c - r$ to $x_c + r$ and calculate the corresponding y values at each position as**

$$y = y_c \;+(-)\; (r^2 - (xc - x)^2)^{1/2}$$

- **This is not the best method:**
  - ***Considerable amount of computation***
  - ***Spacing between plotted pixels is not uniform***

Polar co-ordinates for a circle

- **We could use polar coordinates r and θ,**

  $$x = x_c + r\cos\theta \qquad y = y_c + r\sin\theta$$

- **A fixed angular step size can be used to plot equally spaced points along the circumference**
- **A step size of 1/r can be used to set pixel positions to approximately 1 unit apart for a continuous boundary**
- **But, note that circle sections in adjacent octants within one quadrant are symmetric with respect to the 45 deg line dividing the two octants**
- **Thus we can generate all pixel positions around a circle by calculating just the points within the sector from x=0 to x=y**
- **This method is still computationally expensive**

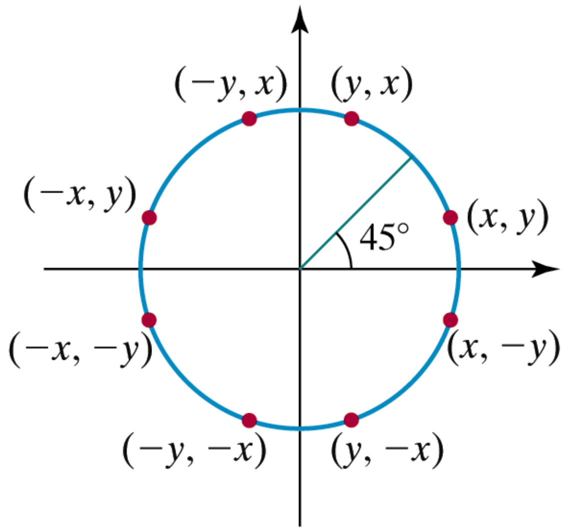


Figure 3-18

Symmetry of a circle. Calculation of a circle point (x, y)in one octant yields the circle points shown for the other seven octants.

We need to compute only one 45-degree segment to determine the circle completely. For a circle centered at the origin (0,0), the eight symmetrical points can be displayed with procedure circlepoints().

```
Void circlepoints (int x, int y)
{
setpixel ( x, y);
setpixel ( y, x);
setpixel ( y, -x);
setpixel ( x, -y);
setpixel ( -x, -y);
setpixel ( -y, -x);
setpixel ( -y, x);
setpixel ( -x, y);
}
```

Suppose the point (xcenter, ycenter) is the center of the circle. Then the above function can be modified as:

```
Void circlepoints(xcenter, ycenter, x, y)
intxcenter, ycenter, x, y;
{
setpixel ( xcenter + x, ycenter + y);
setpixel ( xcenter + y, ycenter + x);
setpixel ( xcenter + y, ycenter - x);
setpixel ( xcenter + x, ycenter - y);
setpixel ( xcenter - x, ycenter - y);
setpixel ( xcenter - y, ycenter - x);
setpixel ( xcenter -y, ycenter + x);
setpixel ( xcenter - x, ycenter + y);
}
```

Bresenham's ALGORITHM for circle

1. Set the initial values of the variable:

(h,k) coordinates of the center of the circle, x=0, y=r and d=3-2r

2. Test to determine whether the entire circle has been scan converted or not. If x>y stop.

3. Plot the eight points by symmetry w.r.t. the centre (h,k) at the current (x,y) coordinates.

Plot(x+h,y+k), Plot(y+h,x+k), Plot(-y+h,x+k), Plot(-x+h,y+k) , Plot(-x+h,-y+k), Plot(-y+h,-x+k), Plot(y+h,-x+k), Plot(x+h,-y+k)

4. Compute the location of the next pixel.

If d<0 then d=d+4x+6 and x=x+1

Else

d=d+4(x-y)+10 and x=x+1, y=y-1

5. GOTO step 2

Bresenham to Midpoint

- Bresenham's line algorithm for raster displays is adapted to circle generation by setting up decision parameters for finding the closest pixel to the circumference at each sampling step.
- Bresenham's circle algorithm avoids square-root calculations by comparing the squares of the pixel separation distances.

Midpoint Circle Algorithm

- We will first calculate pixel positions for a circle centered around the origin (0,0). Then, each calculated position (x,y) is moved to its proper screen position by adding x_c to x and y_c to y
- Note that along the circle section from x=0 to x=y in the first octant, the slope of the curve varies from 0 to -1
- Therefore, we can take unit steps in the positive ***x*** direction over this octant and use a decision parameter to determine which of the two possible y positions is closer to the circle path at each step. Positions in the other seven octants are then obtained by symmetry.
- Circle function around the origin is given by

$$f_{circle}(x,y) = x^2 + y^2 - r^2$$

- Any point (x,y) on the boundary of the circle satisfies the equation $f_{circle}(x,y) = 0$

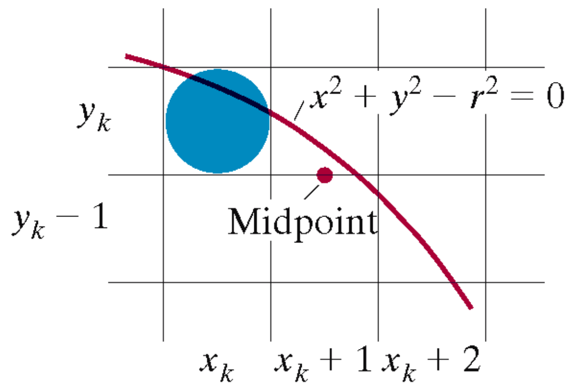


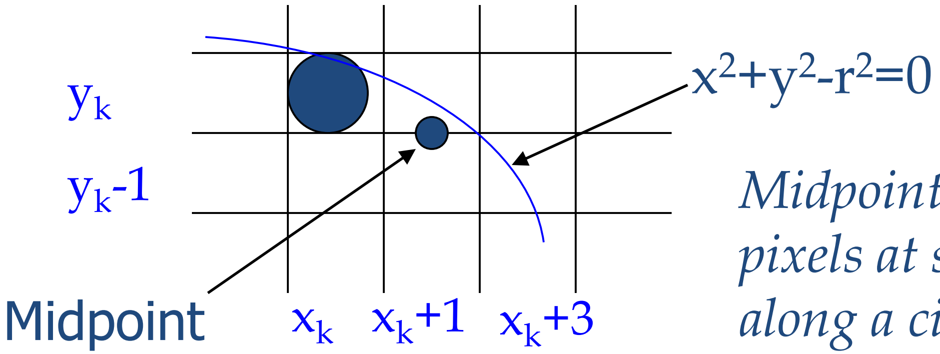
Figure 3-19

Midpoint between candidate pixels at
sampling position $x_k + 1$ along a circular path.

Computer Graphics with Open GL, Third Edition, by Donald Hearn and M.Pauline Baker.
ISBN 0-13-0-15390-7

Midpoint Circle Algorithm

- For a point in the interior of the circle, the circle function is negative and for a point outside the circle, the function is positive
- Thus,
  - $f_{circle}(x,y) < 0$ if (x,y) is inside the circle boundary
  - $f_{circle}(x,y) = 0$ if (x,y) is on the circle boundary
  - $f_{circle}(x,y) > 0$ if (x,y) is outside the circle boundary



Midpoint between candidate pixels at sampling position x_k+1 along a circular path

Midpoint Circle Algorithm

- Assuming we have just plotted the pixel at (x_k, y_k) , we next need to determine whether the pixel at position $(x_k + 1, y_k - 1)$ is closer to the circle
- Our decision parameter is the circle function evaluated at the midpoint between these two pixels

$$p_k = f_{circle}(x_k + 1, y_k - 1/2) = (x_k + 1)^2 + (y_k - 1/2)^2 - r^2$$

If $p_k < 0$, this midpoint is inside the circle and the pixel on the scan line y_k is closer to the circle boundary. Otherwise, the mid position is outside or on the circle boundary, and we select the pixel on the scan line $y_k - 1$

Midpoint Circle Algorithm

- **Successive decision parameters are obtained using incremental calculations**

$$p_{k+1} = f_{circle}(x_{k+1}+1, y_{k+1}-1/2)$$
$$= [(x_{k+1})+1]^2 + (y_{k+1} - 1/2)^2 - r^2$$

OR

$$p_{k+1} = p_k + 2(x_K+1) + (y_{K+1}{}^2 - y_k{}^2) - (y_k+1 - y_k)+1$$

Where y_{k+1} **is either** y_k **or** y_{k-1}**, depending on the sign of** p_k

- **Increments for obtaining** p_{k+1}**:**

$2x_{k+1}+1$ *if* p_k *is negative*

$2x_{k+1}+1-2y_{k+1}$ *otherwise*

Case1:

$If\ p_k < 0$, then, $Y_{k+1} = Y_k$

$$p_{k+1} = p_k + 2(x_{k+1}) + 1 = p_k + 2x_k + 3$$

Case2:

$If\ p_k > 0$, then $\quad y_{k+1} = y_k - 1$

$$p_{k+1} = p_k + 2(x_k + 1) + 1((y_{k-1})^2 - y_k^2) - (y_{k-1} - y_k))$$

$$then, p_{k+1} = p_k + 2x_k + 3 + y_k^2 + 1 - 2y_k - y_k^2 + 1$$

Then, $p_{k+1} = p_k + 2x_k - 2y_k + 5$

$$p_{k+1} = p_k + 2(x_k - y_k) + 5$$

$$\boldsymbol{therefore, p_{k+1} = p_k + 2(x_k - y_k) + 5, \mathbf{for}\ \ p_k > 0}$$

and, $\boldsymbol{p_k + 2x_k + 3}$ **for** $\boldsymbol{p_k < 0}$

Midpoint circle algorithm

- Initial decision parameter is obtained by evaluating the circle function at the start position $(x_0,y_0) = (0,r)$

  $$p_0 = f_{circle}(1, r-1/2) = 1+ (r-1/2)^2-r^2$$

OR

$$P_0 = 5/4 -r$$

- If radius r is specified as an integer, we can round p_0 to

  $$p_0 = 1-r$$

The Mid point Circle algorithm

1: Input radius r and circle center (x_c, y_c) and obtain the first point on the circumference of the circle centered on the origin as: $(x_0, y_0) = (0, r)$

2: **Calculate the initial value of the decision parameter as**

$P_0 = 5/4 - r$ **but for integer radius P0=1-r;**

3: At each x_k position starting at $k = 0$, perform the following test:

If $p_k < 0$, the next point along the circle centered on (0,0) is (x_{k+1}, y_k) **and** $p_{k+1} = p_k + 2x_k + 3$

The algorithm

Otherwise the next point along the circle is (x_{k+1}, y_{k-1}) and

$$p_{k+1} = p_{k+1} = p_k + 2(x_k - y_k) + 5$$

And $x_{k+1} = x_k + 1, \quad y_{k+1} = y_k + 1$

Determine the other 7 octant points,

Move each calculated pixel position (x,y) onto the circular path centered on (x_c ,y_c) and plot the coordinate values

5: $x = x + x_c$, $y = y + y_c$

6: Repeat steps 3 through 5 until x >= y

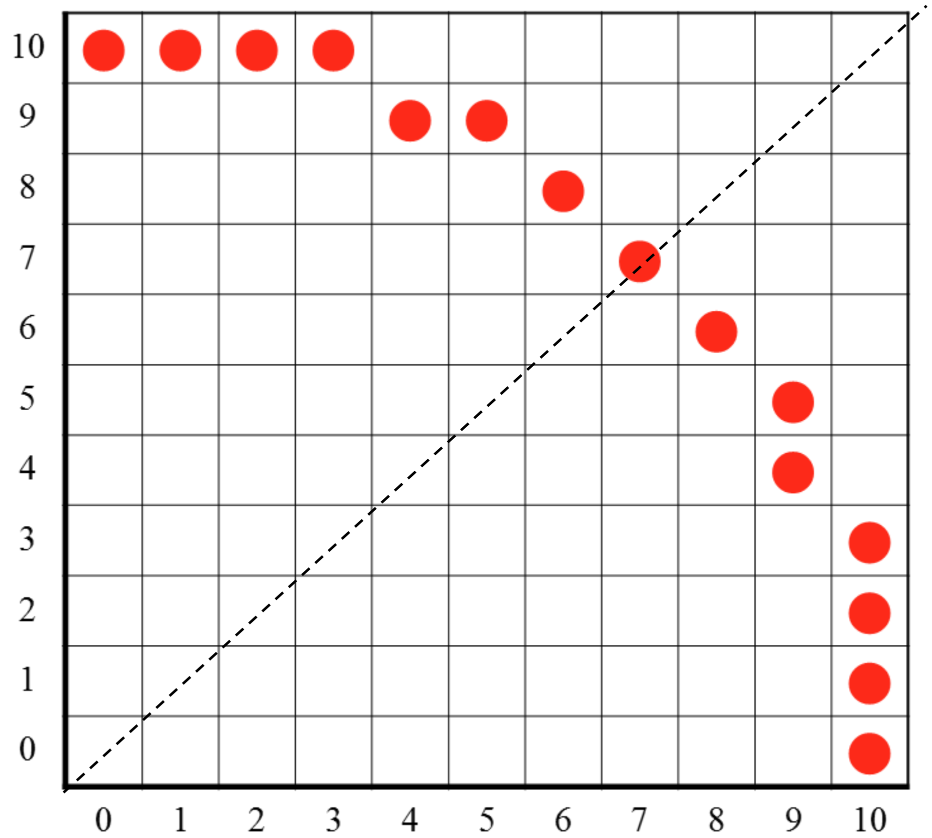
Example

$r = 10$

$p_0 = 1 - r = -9$ (if r is integer round $p_0 = 5/4 - r$ to integer)

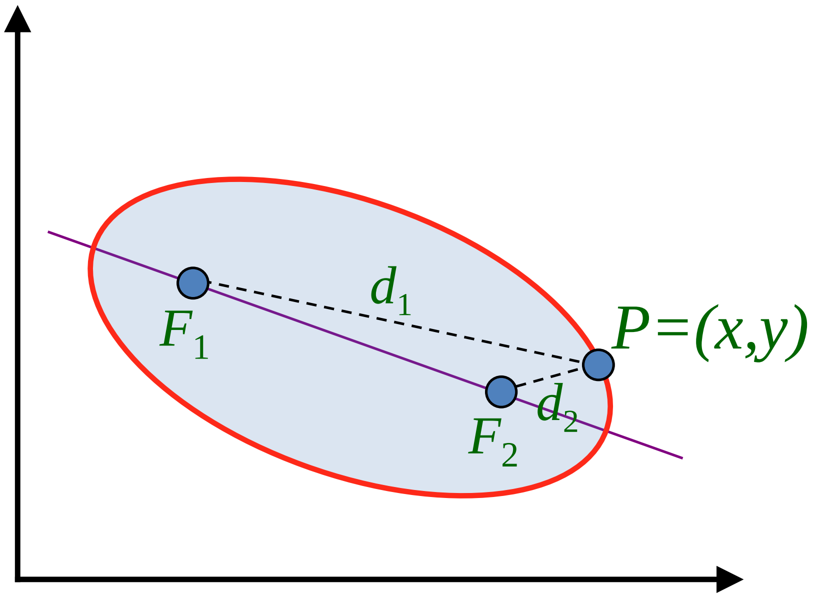
Initial point $(x_0, y_0) = (0, 10)$

i	p_i	x_{i+1}, y_{i+1}	$2x_{i+1}$	$2y_{i+1}$
0	-9	(1, 10)	2	20
1	-6	(2, 10)	4	20
2	-1	(3, 10)	6	20
3	6	(4, 9)	8	18
4	-3	(5, 9)	10	18
5	8	(6, 8)	12	16
6	5	(7, 7)		



Ellipse-Generating Algorithms

- **Ellipse** – A modified circle whose radius varies from a maximum value in one direction (major axis) to a minimum value in the perpendicular direction (minor axis).



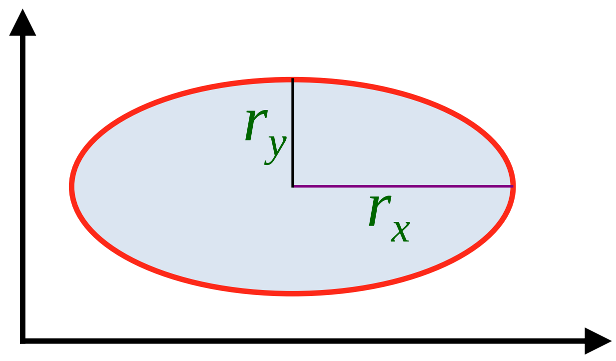
The sum of the two distances d_1 and d_2, between the fixed positions F_1 and F_2 (called the *foci* of the ellipse) to any point P on the ellipse, is the same value, i.e.

$$\boldsymbol{d_1 + d_2 = \text{constant}}$$

Ellipse Properties

- Expressing distances d_1 and d_2 in terms of the focal coordinates $F_1 = (x_1, x_2)$ and $F_2 = (x_2, y_2)$, we have:

$$\sqrt{(x-x_1)^2+(y-y_1)^2}+\sqrt{(x-x_2)^2+(y-y_2)^2}=\text{constant}$$



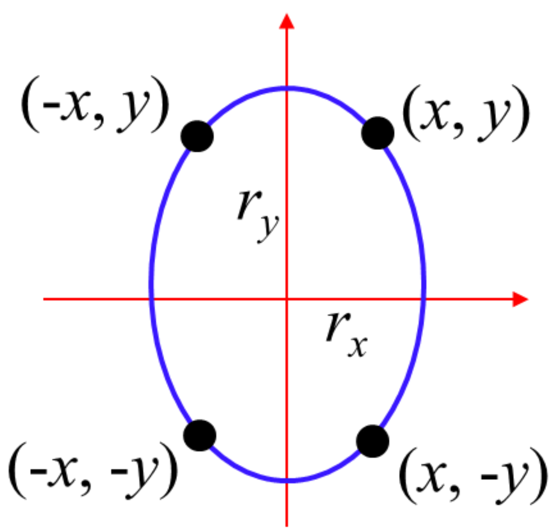
- Cartesian coordinates: $\left(\frac{x-x_c}{r_x}\right)^2+\left(\frac{y-y_c}{r_y}\right)^2=1$

- Polar coordinates:

$$x = x_c + r_x\cos\theta$$
$$y = y_c + r_y\sin\theta$$

Ellipse Algorithms

- Symmetry between quadrants
- Not symmetric between the two octants of a quadrant
- Thus, we must calculate pixel positions along the elliptical arc through one quadrant and then we obtain positions in the remaining 3 quadrants by symmetry

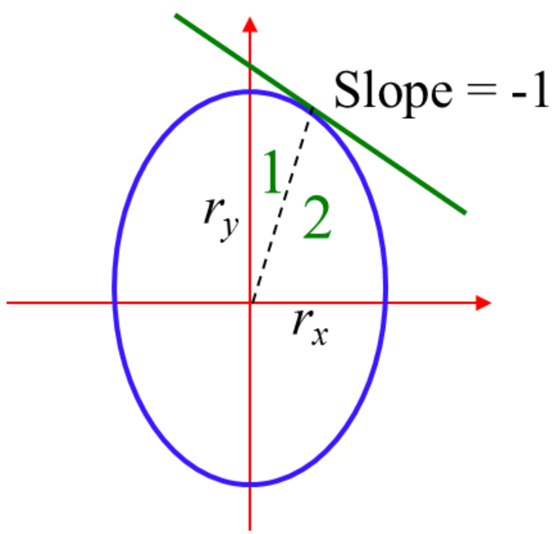


Ellipse Algorithms

$$f_{ellipse}(x, y) = r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2$$

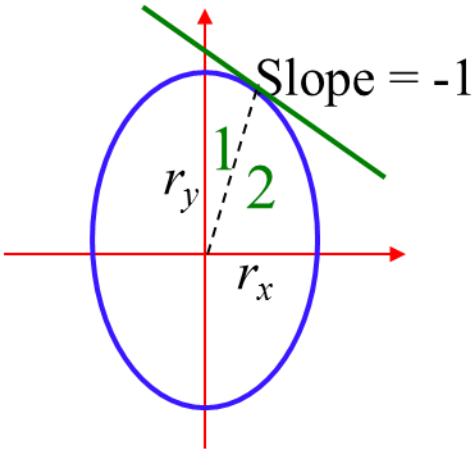
- Decision parameter:

$$f_{ellipse}(x, y) = \begin{cases} < 0 & \text{if } (x, y) \text{ is inside the ellipse} \\ = 0 & \text{if } (x, y) \text{ is on the ellipse} \\ > 0 & \text{if } (x, y) \text{ is outside the ellipse} \end{cases}$$



$$Slope = \frac{dy}{dx} = -\frac{2r_y^2 x}{2r_x^2 y}$$

Ellipse Algorithms



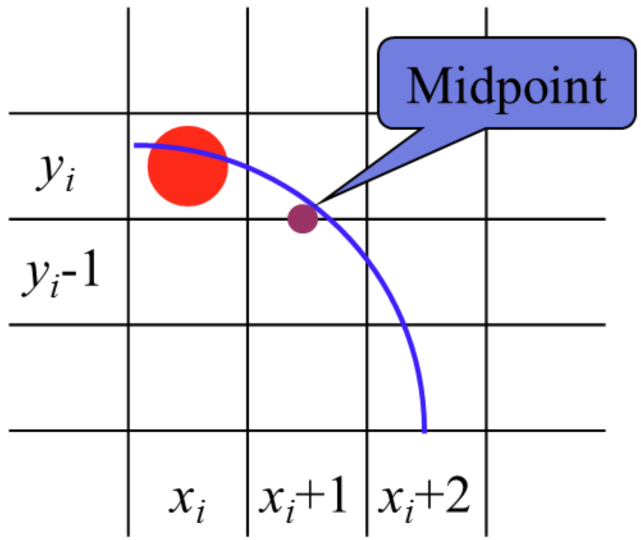
- Starting at $(0, r_y)$ we take unit steps in the x direction until we reach the boundary between region 1 and region 2. Then we take unit steps in the y direction over the remainder of the curve in the first quadrant.
- At the boundary

$$\frac{dy}{dx} = -1 \quad \Rightarrow \quad 2r_y^2 x = 2r_x^2 y$$

- therefore, we move out of region 1 whenever

$$2r_y^2 x \geq 2r_x^2 y$$

Midpoint Ellipse Algorithm



Assuming that we have just plotted the pixels at **(x_i , y_i)**.
The next position is determined by:

$$p1_i = f_{ellipse}(x_i + 1, y_i - \tfrac{1}{2})$$
$$= r_y^2 (x_i + 1)^2 + r_x^2 (y_i - \tfrac{1}{2})^2 - r_x^2 r_y^2$$

If $p1_i < 0$ the midpoint is inside the ellipse $\Rightarrow$ y_i is closer
If $p1i \geq 0$ the midpoint is outside the ellipse $\Rightarrow$ $y_i - 1$ is closer

Decision Parameter (Region 1)

At the next position $[x_{i+1} + 1 = x_i + 2]$

$$p1_{i+1} = f_{ellipse}(x_{i+1} + 1, y_{i+1} - \tfrac{1}{2})$$
$$= r_y^2 (x_i + 2)^2 + r_x^2 (y_{i+1} - \tfrac{1}{2})^2 - r_x^2 r_y^2$$

OR

$$p1_{i+1} = p1_i + 2r_y^2 (x_i + 1)^2 + r_y^2 + r_x^2 \left[(y_{i+1} - \tfrac{1}{2})^2 - (y_i - \tfrac{1}{2})^2 \right]$$

where $y_{i+1} = y_i$

or $\quad y_{i+1} = y_i - 1$

Decision Parameter (Region 1)

Decision parameters are incremented by:

$$increment = \begin{cases} 2r_y^2 x_{i+1} + r_y^2 & \text{if } p1_i < 0 \\ 2r_y^2 x_{i+1} + r_y^2 - 2r_x^2 y_{i+1} & \text{if } p1_i \geq 0 \end{cases}$$

Use only addition and subtraction by obtaining

$$2r_y^2 x \quad \text{and} \quad 2r_x^2 y$$

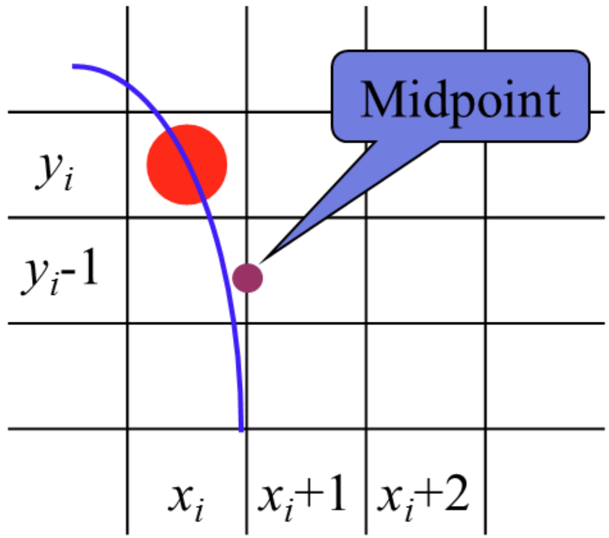
At initial position **(0, r_y)**

$$2r_y^2 x = 0$$
$$2r_x^2 y = 2r_x^2 r_y$$

$$p1_0 = f_{ellipse}(1, r_y - \tfrac{1}{2}) = r_y^2 + r_x^2 (r_y - \tfrac{1}{2})^2 - r_x^2 r_y^2$$
$$= r_y^2 - r_x^2 r_y + \tfrac{1}{4} r_x^2$$

Region 2

Over region 2, step in the negative y direction and midpoint is taken between horizontal pixels at each step.



Decision parameter:

$$p2_i = f_{ellipse}(x_i + \tfrac{1}{2}, y_i - 1)$$
$$= r_y^2 (x_i + \tfrac{1}{2})^2 + r_x^2 (y_i - 1)^2 - r_x^2 r_y^2$$

If $p2_i > 0$ the midpoint is outside the ellipse $\Rightarrow$ $\boldsymbol{x_i}$ is closer

If $p2i \leq 0$ the midpoint is inside the ellipse $\Rightarrow$ $\boldsymbol{x_i} + 1$ is closer

Decision Parameter (Region 2)

At the next position $[y_{i+1} - 1 = y_i - 2]$

$$p2_{i+1} = f_{ellipse}(x_{i+1} + \tfrac{1}{2}, y_{i+1} - 1)$$
$$= r_y^2(x_{i+1} + \tfrac{1}{2})^2 + r_x^2(y_i - 2)^2 - r_x^2 r_y^2$$

OR

$$p2_{i+1} = p2_i - 2r_x^2(y_i - 1) + r_x^2 + r_y^2\left[(x_{i+1} + \tfrac{1}{2})^2 - (x_i + \tfrac{1}{2})^2\right]$$

where $x_{i+1} = x_i$

or $\quad x_{i+1} = x_i + 1$

Decision Parameter (Region 2)

Decision parameters are incremented by:

$$increment = \begin{cases} -2r_x^2 y_{i+1} + r_x^2 & \text{if } p2_i > 0 \\ 2r_y^2 x_{i+1} - 2r_x^2 y_{i+1} + r_x^2 & \text{if } p2_i \le 0 \end{cases}$$

At initial position **(x_0, y_0)** is taken at the last position selected in region 1

$$p2_0 = f_{ellipse}(x_0 + \tfrac{1}{2}, y_0 - 1)$$
$$= r_y^2 (x_0 + \tfrac{1}{2})^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$

Midpoint Ellipse Algorithm

1. Input r_x, r_y, and ellipse center (x_c, y_c), and obtain the first point on an ellipse centered on the origin as

   $$(x_0, y_0) = (0, r_y)$$

2. Calculate the initial parameter in region 1 as

   $$p1_0 = r_y^2 - r_x^2 r_y + \tfrac{1}{4} r_x^2$$

3. At each x_i position, starting at $i = 0$, if $p1_i < 0$, the next point along the ellipse centered on (0, 0) is $(x_i + 1, y_i)$ and

   $$p1_{i+1} = p1_i + 2r_y^2 x_{i+1} + r_y^2$$

   otherwise, the next point is $(x_i + 1, y_i - 1)$ and

   $$p1_{i+1} = p1_i + 2r_y^2 x_{i+1} - 2r_x^2 y_{i+1} + r_y^2$$

   and continue until $2r_y^2 x \geq 2r_x^2 y$

Midpoint Ellipse Algorithm

4. (x_0, y_0) is the last position calculated in region 1. Calculate the initial parameter in region 2 as

$$p2_0 = r_y^2(x_0 + \tfrac{1}{2})^2 + r_x^2(y_0 - 1)^2 - r_x^2 r_y^2$$

5. At each y_i position, starting at $i = 0$, if $p2_i > 0$, the next point along the ellipse centered on (0, 0) is $(x_i, y_i - 1)$ and

$$p2_{i+1} = p2_i - 2r_x^2 y_{i+1} + r_x^2$$

otherwise, the next point is $(x_i + 1, y_i - 1)$ and

$$p2_{i+1} = p2_i + 2r_y^2 x_{i+1} - 2r_x^2 y_{i+1} + r_x^2$$

Use the same incremental calculations as in region 1. Continue until $y = 0$.

6. For both regions determine symmetry points in the other three quadrants.
7. Move each calculated pixel position (x, y) onto the elliptical path centered on (x_c, y_c) and plot the coordinate values

$$\boldsymbol{x = x + x_c}\,, \qquad \boldsymbol{y = y + y_c}$$

Example

$r_x = 8$, $r_y = 6$

$2r_y^2x = 0$ (with increment $2r_y^2 = 72$)

$2r_x^2y = 2r_x^2r_y$ (with increment $-2r_x^2 = -128$)

Region 1

$(x_0, y_0) = (0, 6)$

$$p1_0 = r_y^2 - r_x^2 r_y + \tfrac{1}{4} r_x^2 = -332$$

i	p_i	x_{i+1}, y_{i+1}	$2r_y^2x_{i+1}$	$2r_x^2y_{i+1}$
0	-332	(1, 6)	72	768
1	-224	(2, 6)	144	768
2	-44	(3, 6)	216	768
3	208	(4, 5)	288	640
4	-108	(5, 5)	360	640
5	288	(6, 4)	432	512
6	244	(7, 3)	504	384

Move out of region 1 since
$2r_y^2x > 2r_x^2y$

Example

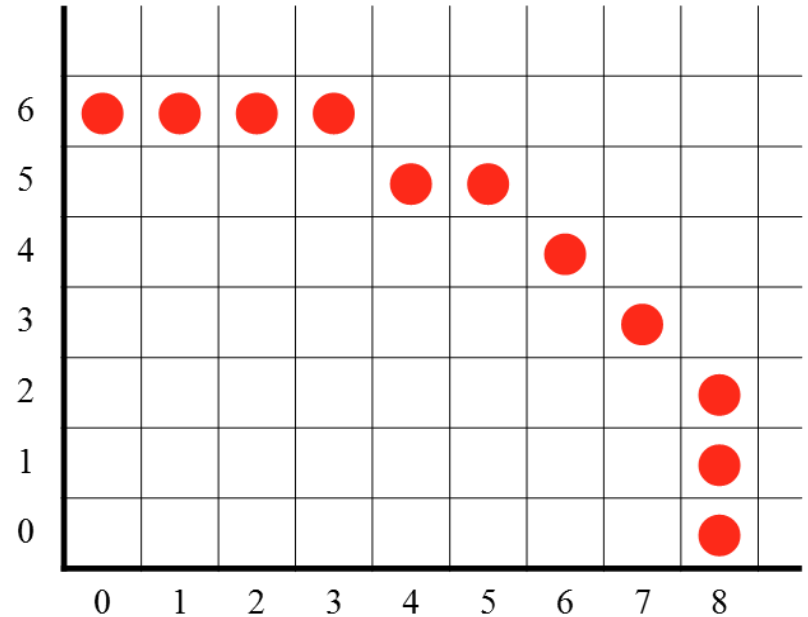
Region 2

$(x_0, y_0) = (7, 3)$ (Last position in region 1)

$$p2_0 = f_{ellipse}(7+\tfrac{1}{2}, 2) = -151$$

i	p_i	x_{i+1}, y_{i+1}	$2r_y^2 x_{i+1}$	$2r_x^2 y_{i+1}$
0	-151	(8, 2)	576	256
1	233	(8, 1)	576	128
2	745	(8, 0)	-	-

Stop at $y = 0$



References:

1. Computer Graphics C version by Donald Hearn and M.P. Baker
2. http://www.geeksforgeeks.org/dda-line-generation-algorithm-computer-graphics/
3. https://users.soe.ucsc.edu/~pang/160/f12/slides/dda2.pdf