Computer Graphics:
Vectors and Line drawing Introduction

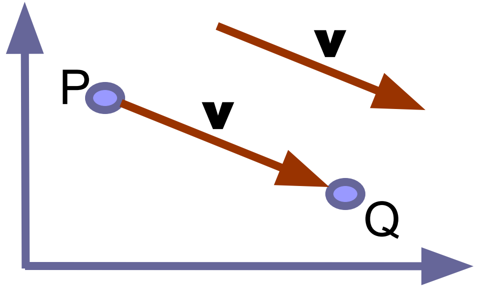
Basic Definitions

- Points specify location in space (or in the plane).
- Vectors have magnitude and direction (like velocity).

Points $\neq$ Vectors

P

Basics of Vectors



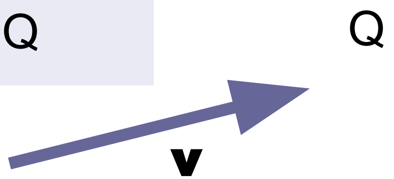
Vector as displacement:

v is a vector from point P to point Q.

The **difference** between two points is a vector: **v** = Q - P

Another way:

The **sum** of a point and a vector is a point : P + **v** = Q



Operations on Vectors

Two operations

Addition

$\mathbf{a} + \mathbf{b}$

$\mathbf{a} = (3,5,8)$, $\mathbf{b} = (-1,2,-4)$

$\mathbf{a} + \mathbf{b} = (2,7,4)$

Multiplication be scalars

$s\mathbf{a}$

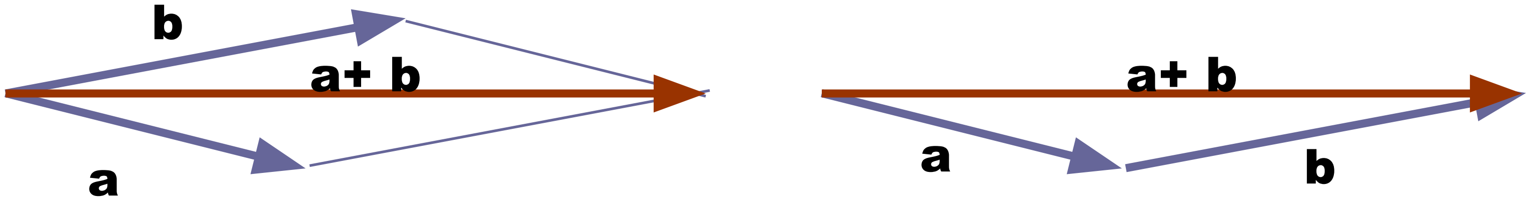
$\mathbf{a} = (3,-5,8)$, $s = 5$

$5\mathbf{a} = (15,-25,40)$

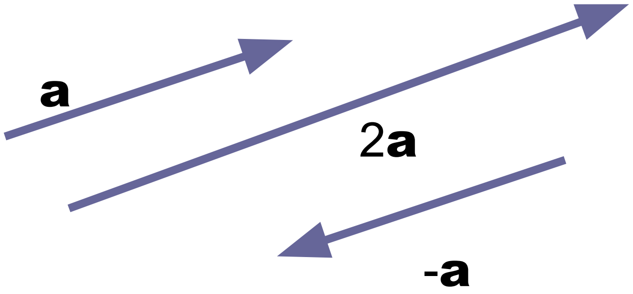
operations are done ***componentwise***

Operations on vectors

Addition



Multiplication by scalar



Properties of vectors

Length or size

$$\mathbf{w} = (w_1, w_2, \ldots, w_n)$$

$$|\,\mathbf{w}\,| = \sqrt{w_1^2 + w_2^2 + \ldots + w_n^2}$$

Unit vector

$$\hat{\mathbf{a}} = \frac{\mathbf{a}}{|\mathbf{a}|}$$

- The process is called **normalizing**
- Used to refer **direction**

The **standard unit vectors**: **i** = (1,0,0), **j** = (0,1,0) and **k** = (0,0,1)

Dot Product

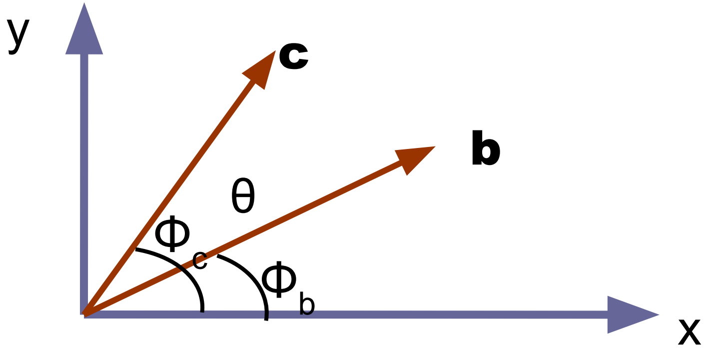
The dot product ***d*** of two vectors $\mathbf{v} = (v_1, v_2, \ldots, v_n)$ and $\mathbf{w} = (w_1, w_2, \ldots w_n)$:

Properties

1. Symmetry: $\mathbf{a} \cdot \mathbf{b} = \mathbf{b} \cdot \mathbf{a}$
2. Linearity: $(\mathbf{a}+\mathbf{c}) \cdot \mathbf{b} = \mathbf{a} \cdot \mathbf{b} + \mathbf{c} \cdot \mathbf{b}$
3. Homogeneity: $(s\mathbf{a}) \cdot \mathbf{b} = s(\mathbf{a} \cdot \mathbf{b})$
4. $|\mathbf{b}|^2 = \mathbf{b} \cdot \mathbf{b}$

Application of Dot Product

Angle between two unit vectors **b** and **c**



$$\cos(\theta) = \hat{\mathbf{b}} \cdot \hat{\mathbf{c}}$$

Two vectors **b** and **c** are <u>**perpendicular**</u> (**orthogonal**/**normal**) if
b·**c** = 0

Cross Product

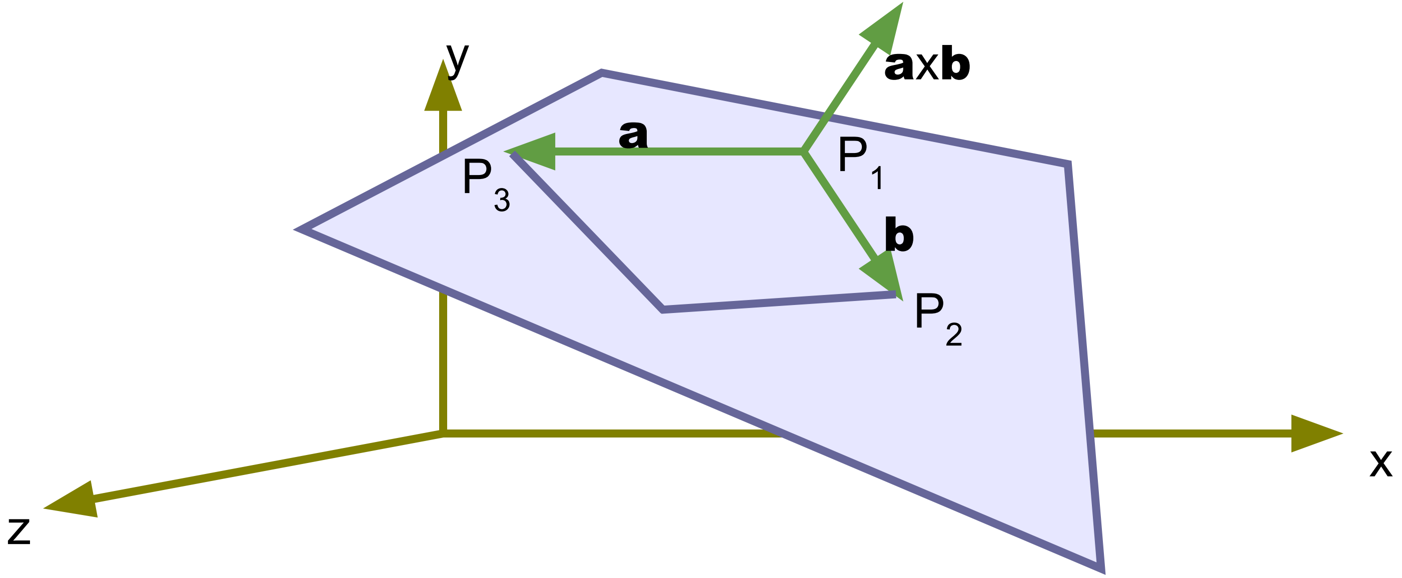
- Also called **vector product**.
- Defined for **3D** vectors only.

$$\mathbf{a} \times \mathbf{b} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix}$$

Properties

1. Antisymmetry: **a** X **b** = - **b** X **a**
2. Linearity: (**a** **+c**) X **b** = **a** X **b** + **c** X **b**
3. Homogeneity: (s**a**) X **b** = s(**a** X **b**)

Geometric Interpretation of Cross Product

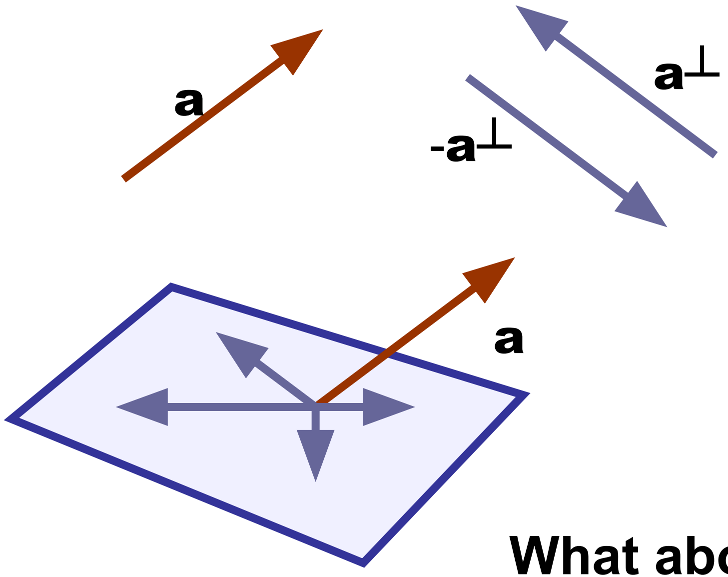


1. **a**X**b** is perpendicular to both **a** and **b**
2. | **a**X**b** | = area of the parallelogram defined by **a** and **b**

2D perp Vector

Which vector is perpendicular to the 2D vector $\mathbf{a} = (a_x, a_y)$?

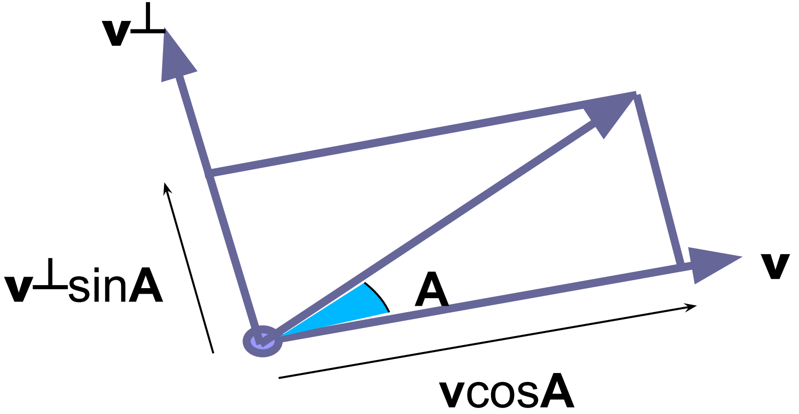
Let $\mathbf{a} = (a_x, a_y)$.
Then $\mathbf{a}^{\perp} = (-a_y, a_x)$
is the **counterclockwise perpendicular** to **a**.



What about 3D case?

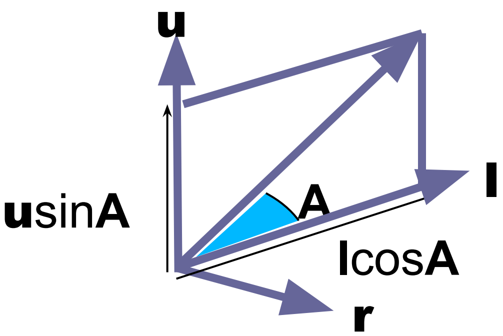
Rotation in 2d

- We want to rotate a 2d vector **v** counterclockwise by an angle **A**
- First we determine perp(**v**), **v**⊥
- Then we scale **v** by cos**A** and scale v⊥ by sin**A** and take their sum



Rotation in 3d

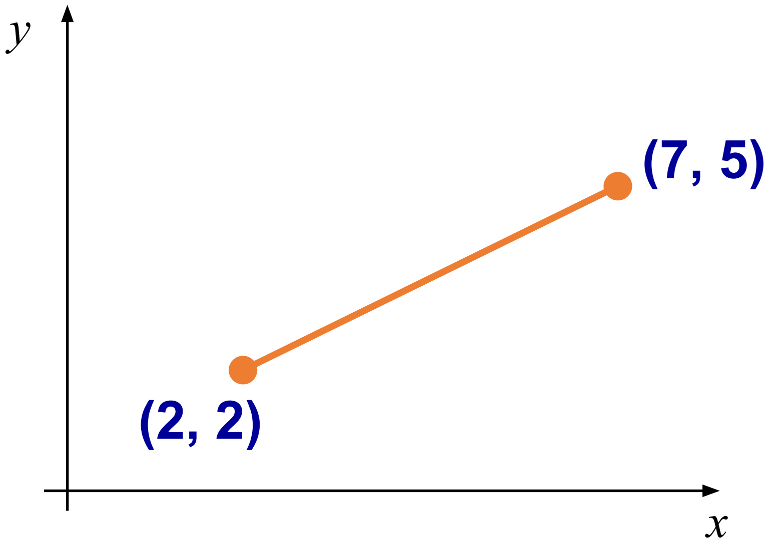
- We want to rotate a 3d vector **l** counterclockwise with respect to a 3d unit vector **r** by an angle **A,** where **l** and **r** are perpendicular to each other
- First we determine the vector **u**, that is perpendicular to both **l** and **r** and have a length equal to that of **l**
- So, **u** = **r** X **l**
- Then we scale **l** by cos**A** and scale **u** by sin**A** and take their sum



* note that, this method is applicable only in cases where the axis of rotation and the vector which is to be rotated are perpendicular to each other

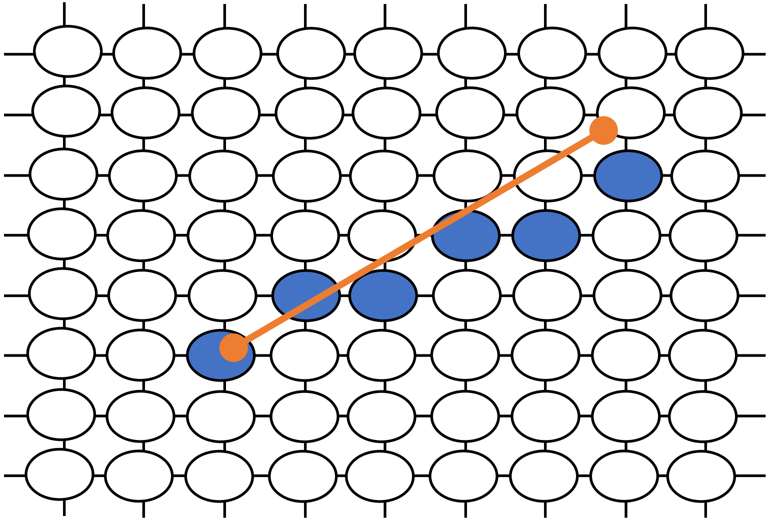
Scan Conversion

A line segment in a scene is defined by the coordinate positions of the line end-points



The Problem Of Scan Conversion

But what happens when we try to draw this on a pixel based display?



How do we choose which pixels to turn on?

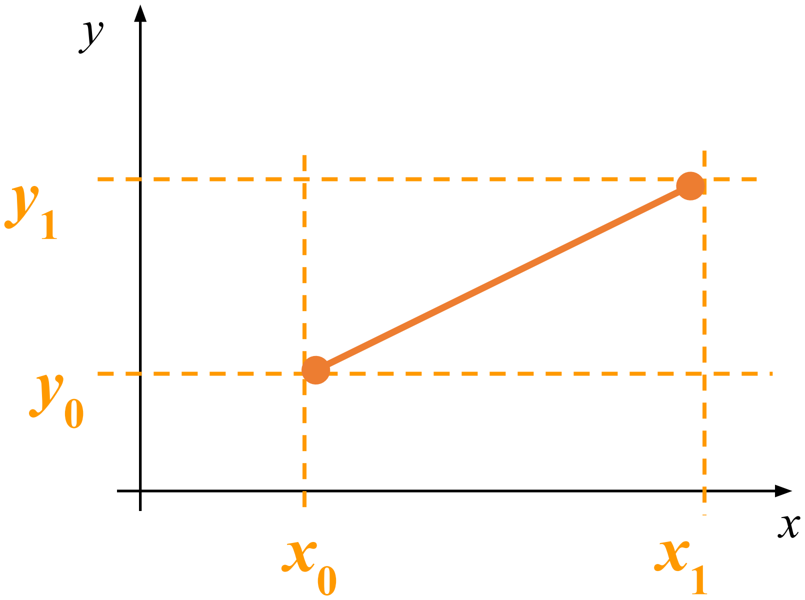
Considerations

Considerations to keep in mind:

- The line has to look good
  - Avoid *jaggies*
- It has to be lightening fast!
  - How many lines need to be drawn in a typical scene?
  - This is going to come back to bite us again and again

Line Equations

Let's quickly review the equations involved in drawing lines



Slope-intercept line equation:

$$y = m \cdot x + b$$

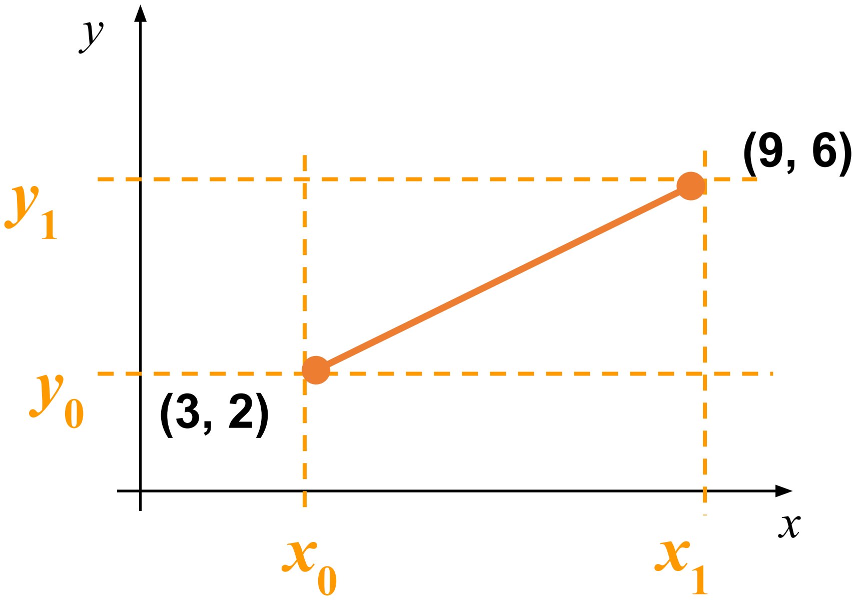
where:

$$m = \frac{y_1 - y_0}{x_1 - x_0}$$

$$b = y_0 - m \cdot x_0$$

Line Equations

Let's quickly review the equations involved in drawing lines



Slope-intercept line equation:

$$y = m \cdot x + b$$

where:

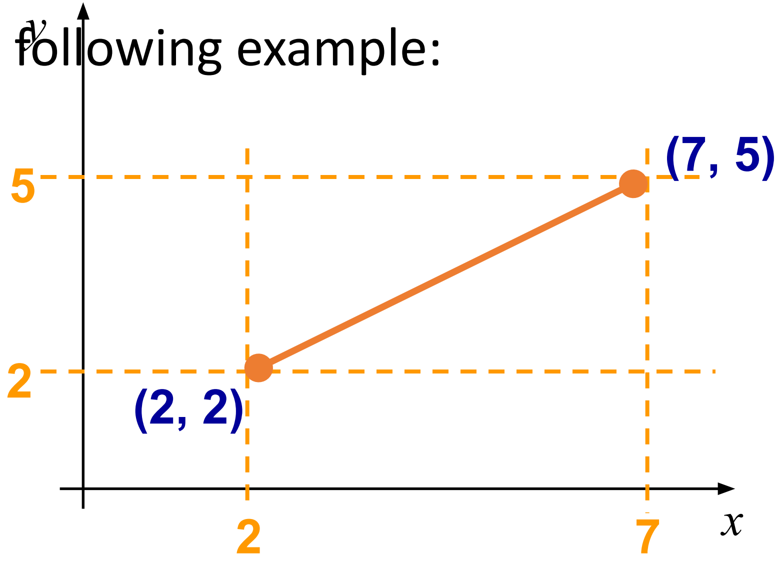
$$m = \frac{6-2}{9-3} = \frac{4}{6} = \frac{2}{3}$$

$$\boldsymbol{b = 2 - (2/3).3 = 2 - 2 = 0}$$

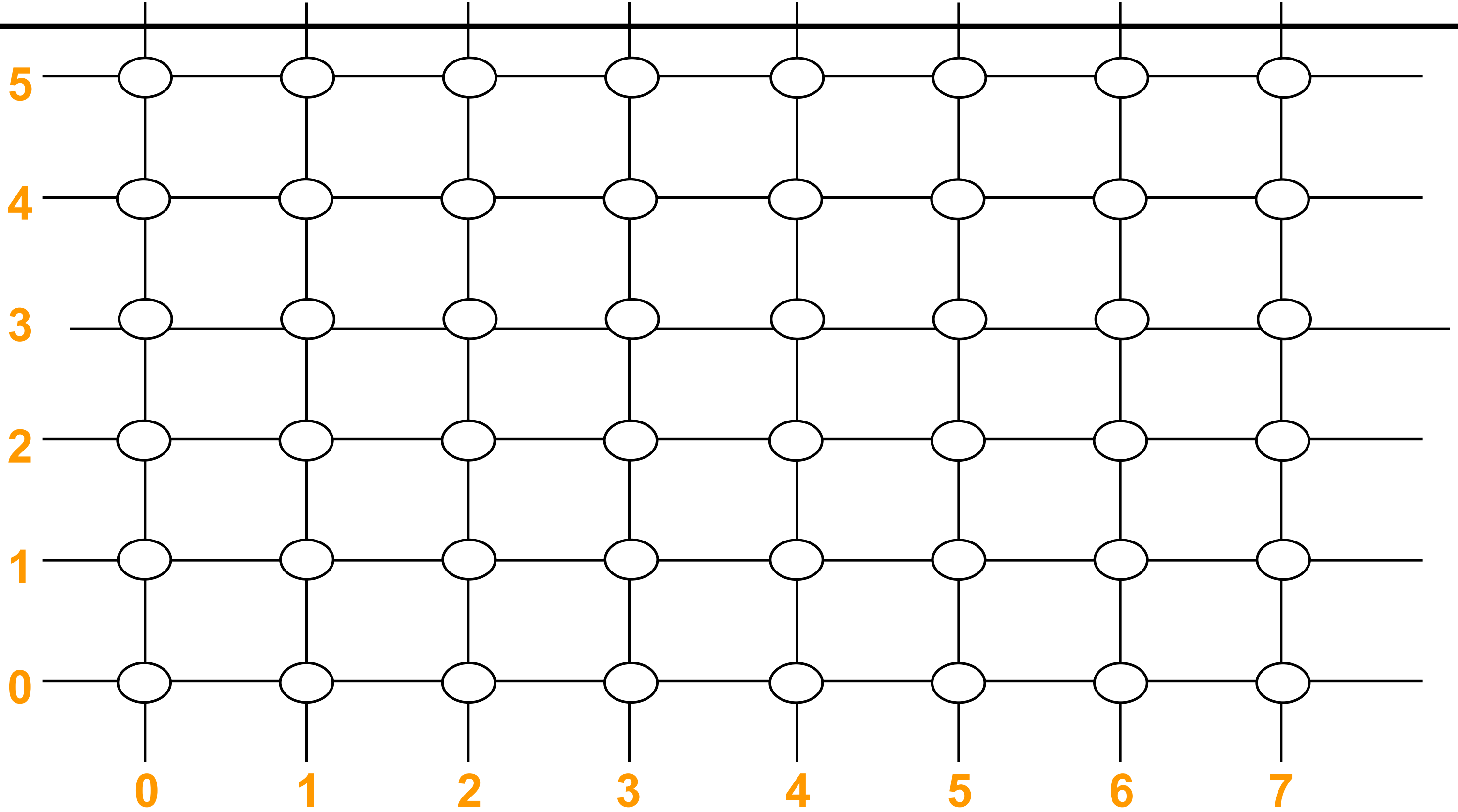
A Very Simple Solution

We could simply work out the corresponding y coordinate for each unit x coordinate

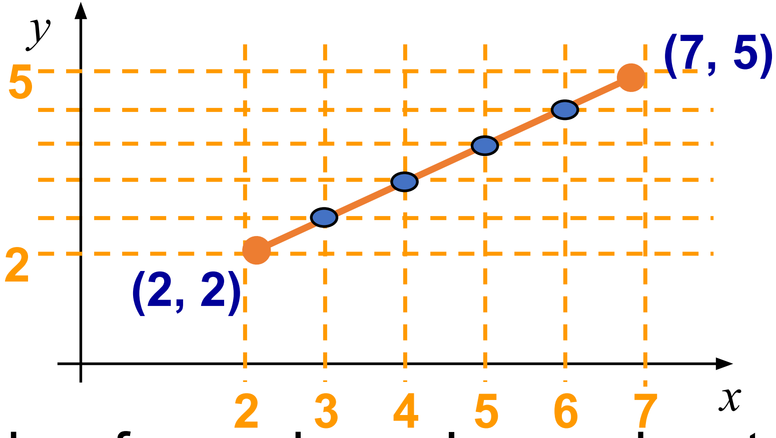
Let's consider the following example:



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:

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

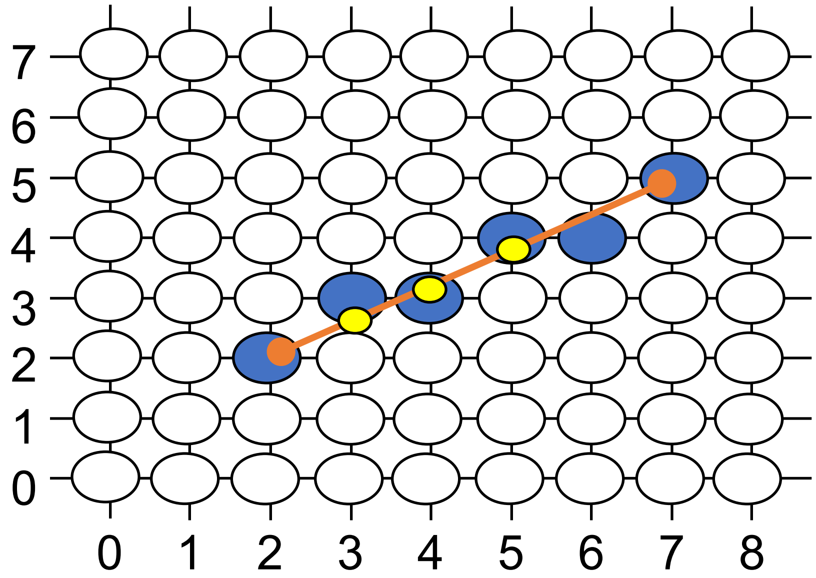
$$y(4) = \frac{3}{5} \cdot 4 + \frac{4}{5} = 3\frac{1}{5}$$

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

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

A Very Simple Solution (cont…)

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



$$y(3) = 2\frac{3}{5} = 2.6 \approx 3$$

$$y(4) = 3\frac{1}{5} = 3.2 \approx 3$$

$$y(5) = 3\frac{4}{5} = 3.8 \approx 4$$

$$y(6) = 4\frac{2}{5} = 4.4 \approx 4$$

Pixel
(3, 3)
(4,3)
(5,4)
(6,4)

A Very Simple Solution (cont…)

However, this approach is just way too slow

In particular look out for:

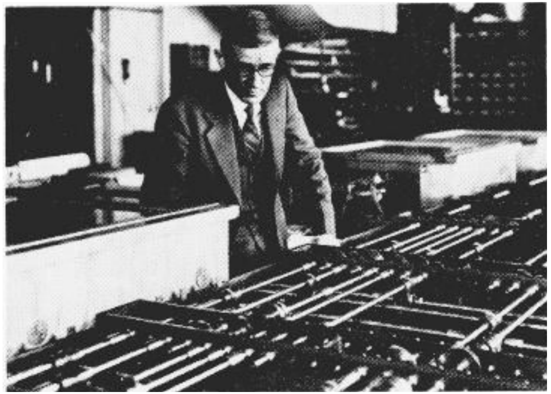
- The equation $y = mx + b$ requires the multiplication of m by x
- Rounding off the resulting y coordinates

We need a faster solution

The DDA Algorithm

The *digital differential analyzer* (DDA) algorithm takes an incremental approach in order to speed up scan conversion

Simply calculate y_{k+1} based on y_k



The original differential analyzer was a physical machine developed by Vannevar Bush at MIT in the 1930's in order to solve ordinary differential equations.

The DDA Algorithm (cont…)

Consider the list of points that we determined for the line in our previous example:

$$(2, 2), (3, 2^3/_5), (4, 3^1/_5), (5, 3^4/_5), (6, 4^2/_5), (7, 5)$$

Notice that as the x coordinates go up by one, the y coordinates simply go up by the slope of the line

This is the key insight in the DDA algorithm

The DDA Algorithm (cont…)

When the slope of the line is between -1 and 1 begin at the first point in the line and, by incrementing the x coordinate by 1, calculate the corresponding y coordinates as follows:

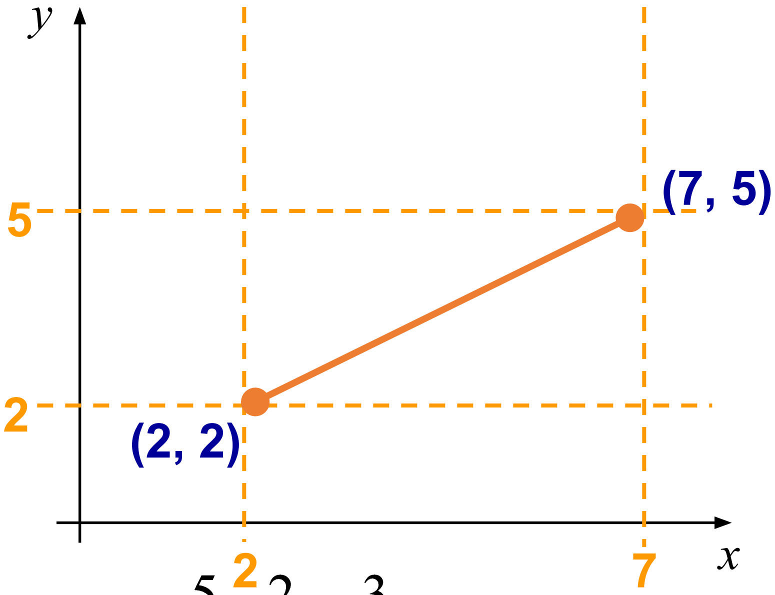
$$y_{k+1} = y_k + m$$

When the slope is outside these limits, increment the y coordinate by 1 and calculate the corresponding x coordinates as follows:

$$x_{k+1} = x_k + \frac{1}{m}$$

DDA Algorithm Example

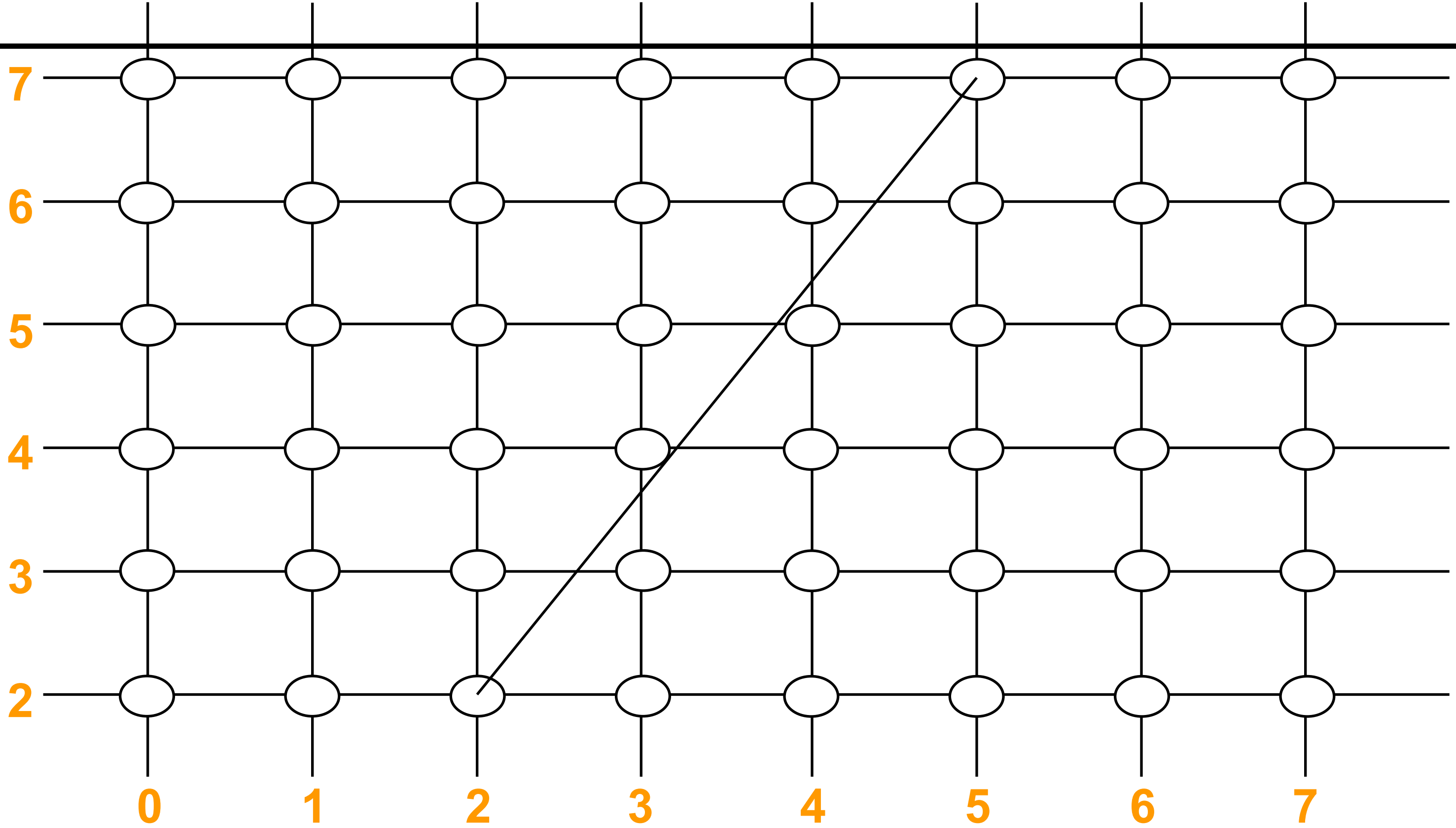
Let's try out the following examples:



$$m = \frac{5-2}{7-2} = \frac{3}{5} = 0.6$$

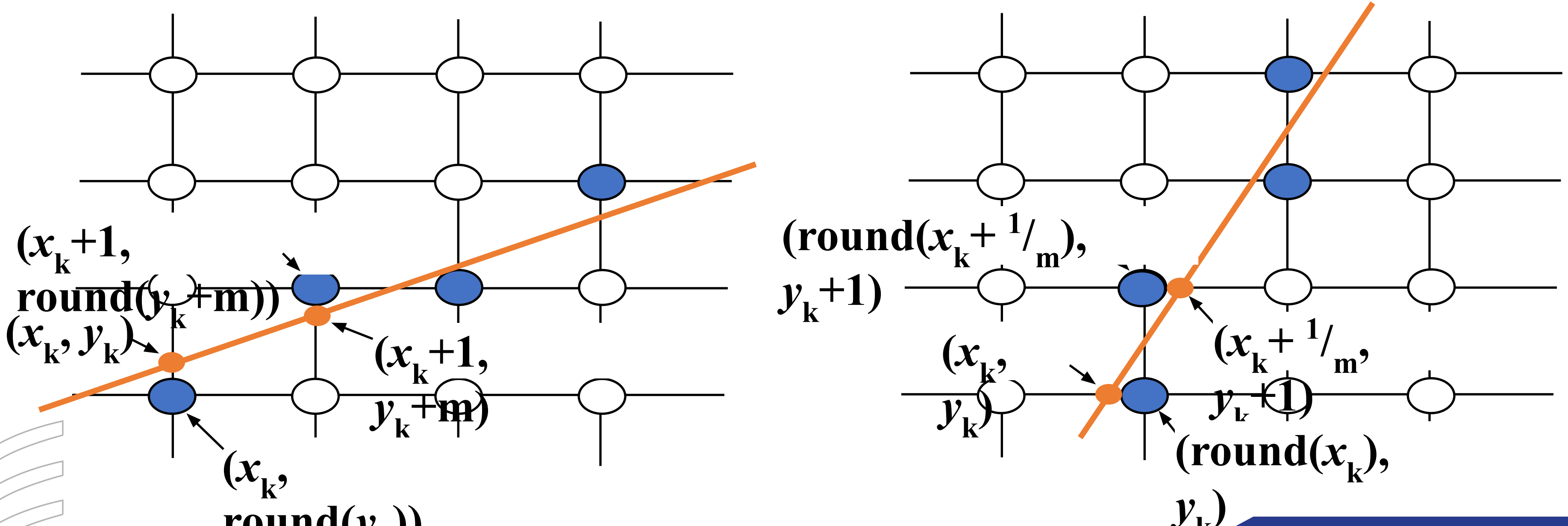
x(+1)	y(+m)	y(roundoff)	pixel
2	2		
3	2.6	3	(3, 3)
4	3.2	3	(4, 3)
5	3.8	4	(5, 4)
6	4.4	4	(6, 4)

DDA Algorithm Example (cont…)



The DDA Algorithm (cont…)

Again the values calculated by the equations used by the DDA algorithm must be rounded to match pixel values



The DDA Algorithm (cont…)

If -1<m<1 then

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

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

Then roundoff y.
Otherwise,

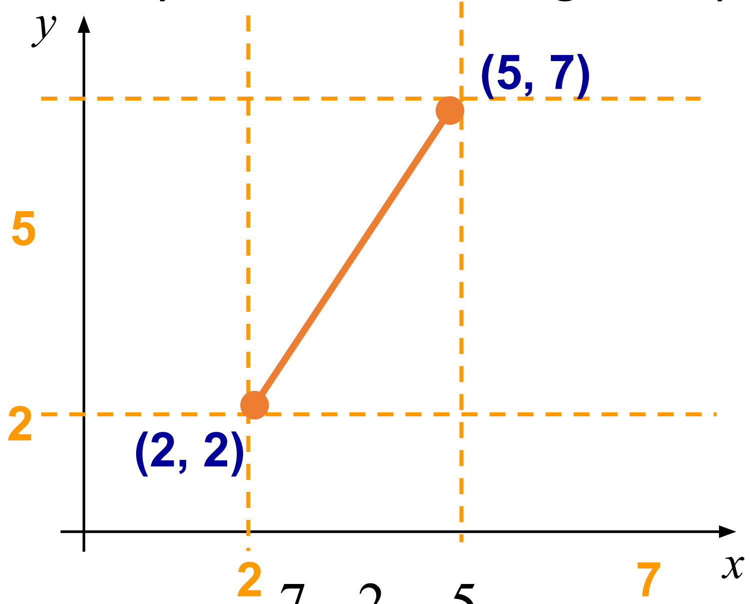
$$y_{k+1} = y_k + 1$$

$$x_{k+1} = x_k + \frac{1}{m}$$

Then roundoff x.

DDA Algorithm Example

Let's try out the following examples:



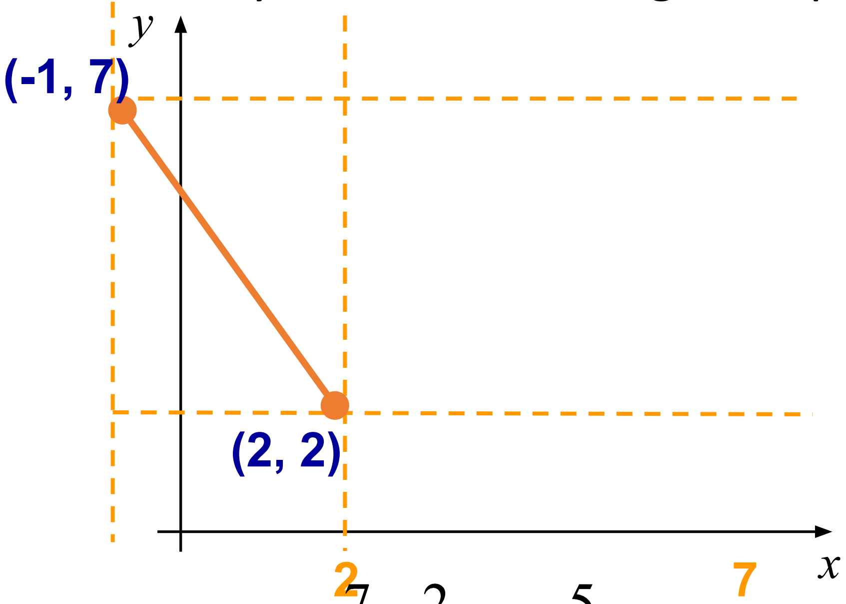
$$m = \frac{7-2}{5-2} = \frac{5}{3}$$

$$\frac{1}{m} = \frac{3}{5} = 0.6$$

y(+1)	x(+1/m)	x(roundoff)	pixel
2	2		
3	2.6	3	(3, 3)
4	3.2	3	(3, 4)
5	3.8	4	(4, 5)
6	4.4	4	(4, 6)

DDA Algorithm Example

Let's try out the following examples:



$$m = \frac{7-2}{-1-2} = -\frac{5}{3}$$

$$\frac{1}{m} = -\frac{3}{5} = -0.6$$

y(+1)	x(+1/m)	x(roundoff)	pixel
2	2		
3	1.4	1	(1, 3)
4	0.8	1	(1, 4)
5	0.2	0	(0, 5)
6	-0.4	0	(0, 6)

The DDA Algorithm Summary

The DDA algorithm is much faster than our previous attempt

- In particular, there are no longer any multiplications involved

However, there are still two big issues:

- Accumulation of round-off errors can make the pixelated line drift away from what was intended
- The rounding operations and floating point arithmetic involved are time consuming