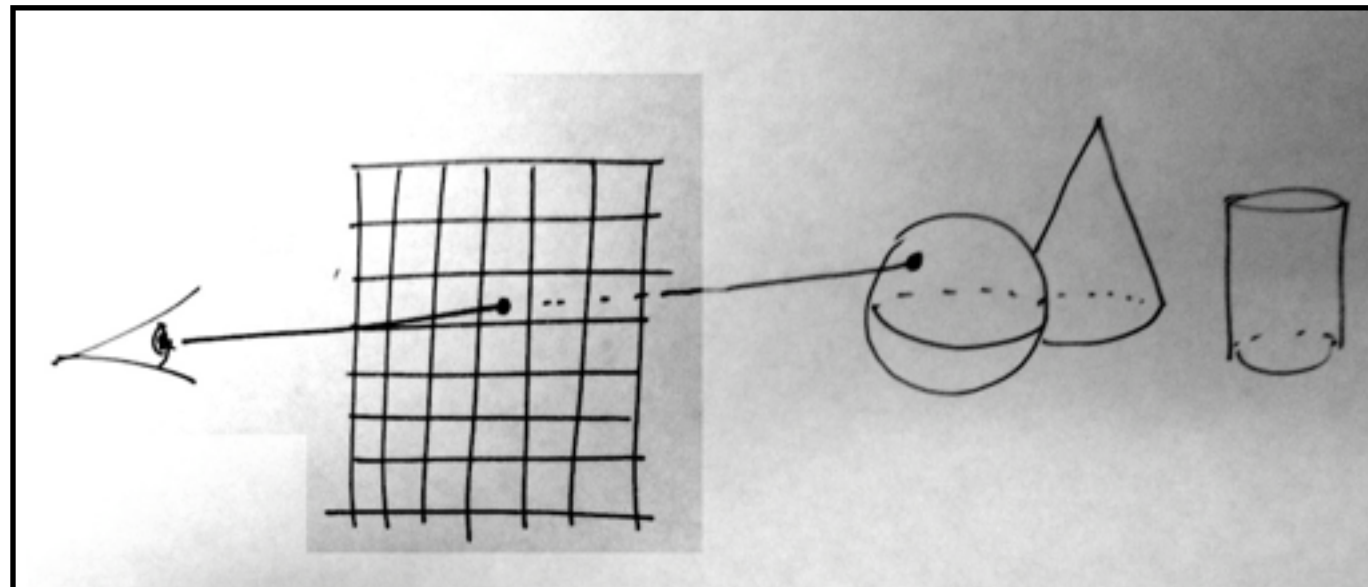
Rendering approaches

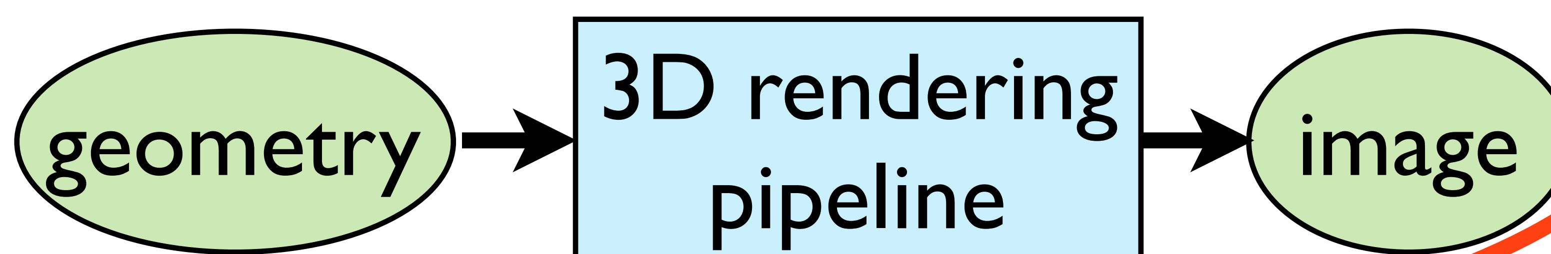
1. image-oriented

foreach pixel ...

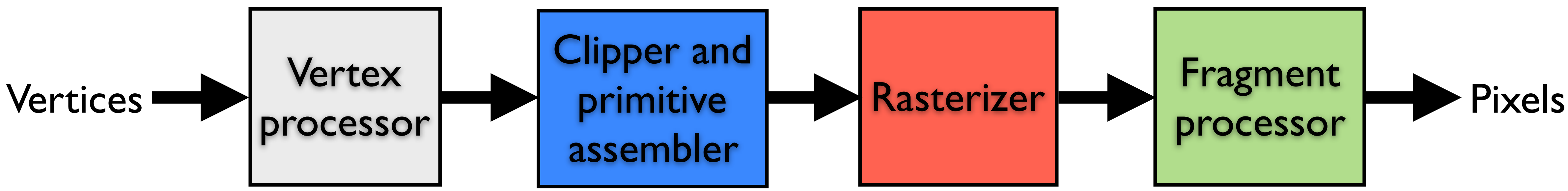


2. object-oriented

foreach object ...



3D graphics pipeline



Vertex processing: coordinate transformations and color

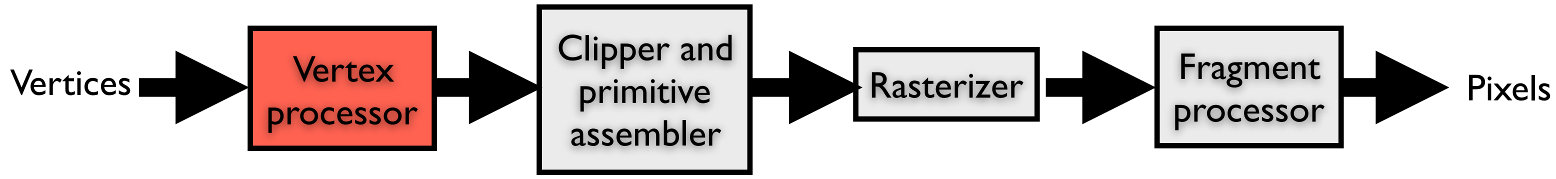
Clipping and primitive assembly: output is a set of primitives

Rasterization: output is a set of fragments for each primitive

Fragment processing: update pixels in the frame buffer

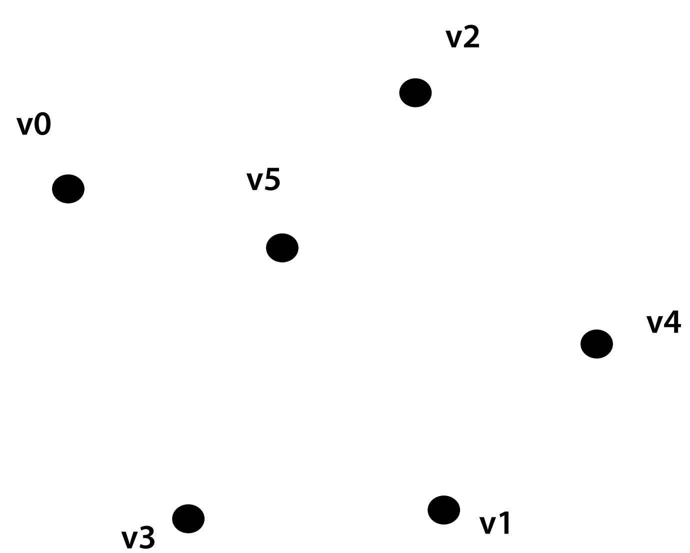
Graphics Pipeline

(slides courtesy K. Fatahalian)

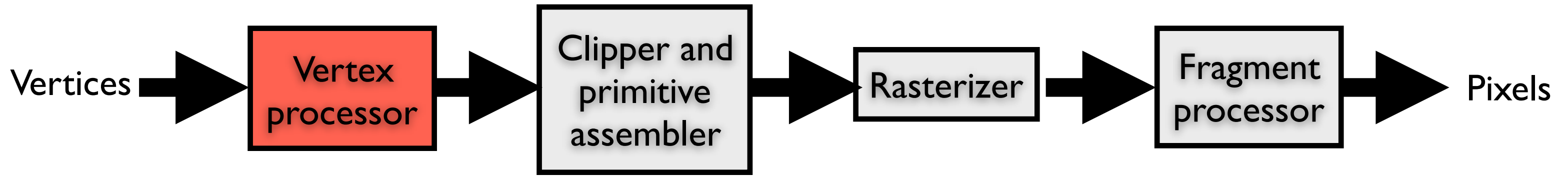


Vertex processing

Vertices are transformed into “screen space”

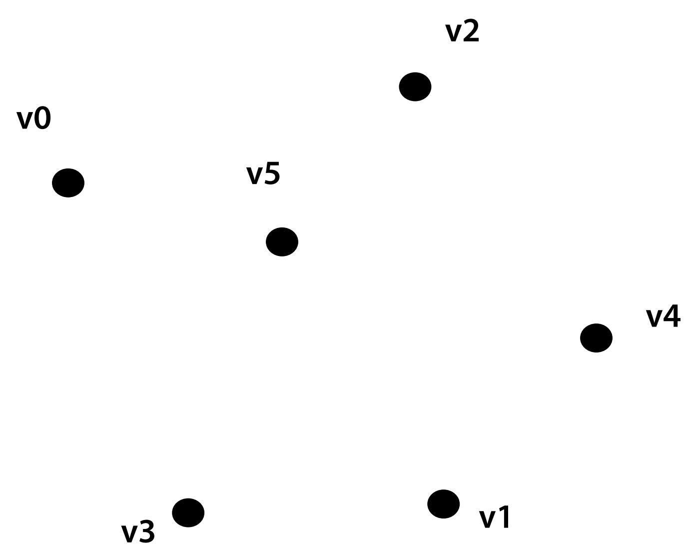


Vertices



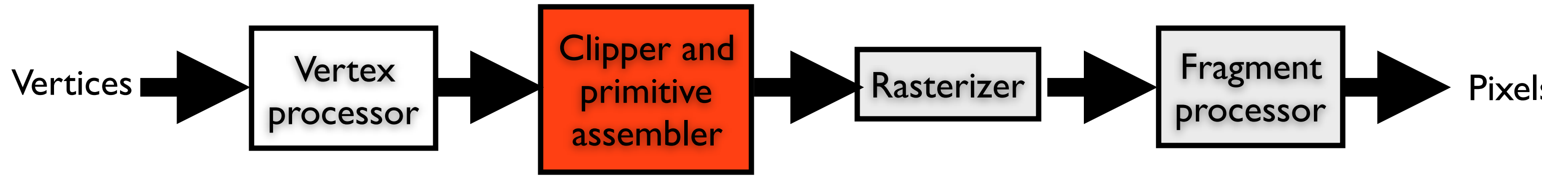
Vertex processing

Vertices are transformed into "screen space"



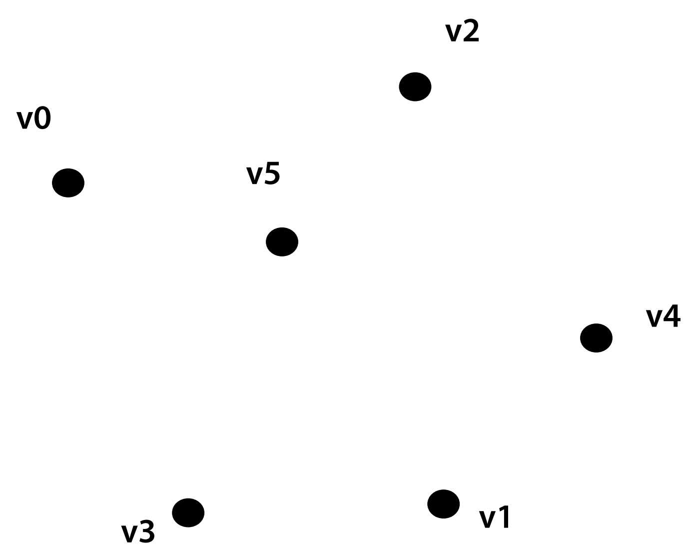
Vertices

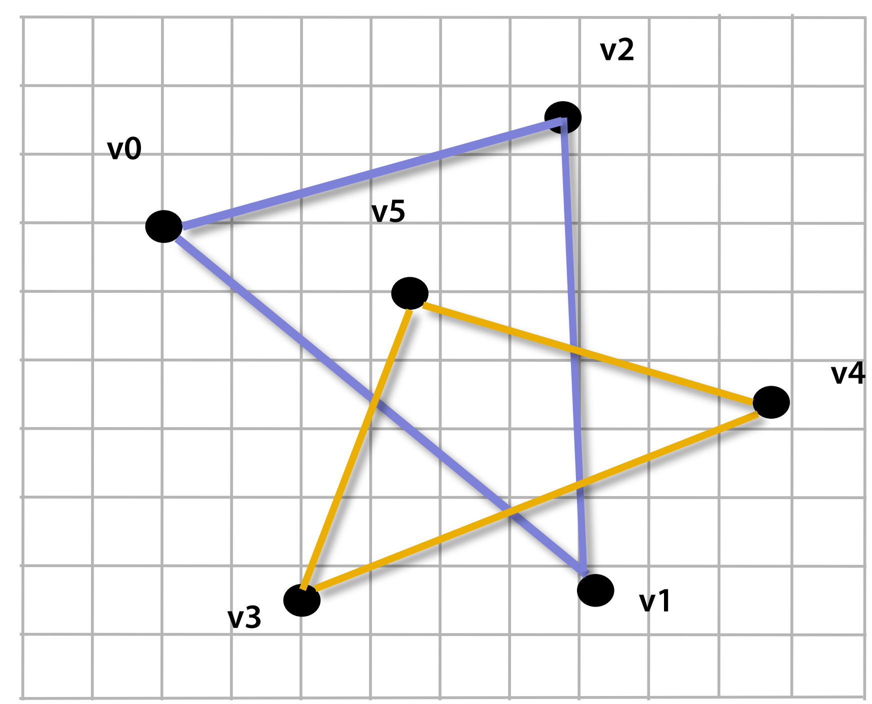
EACH VERTEX IS TRANSFORMED INDEPENDENTLY



Primitive processing

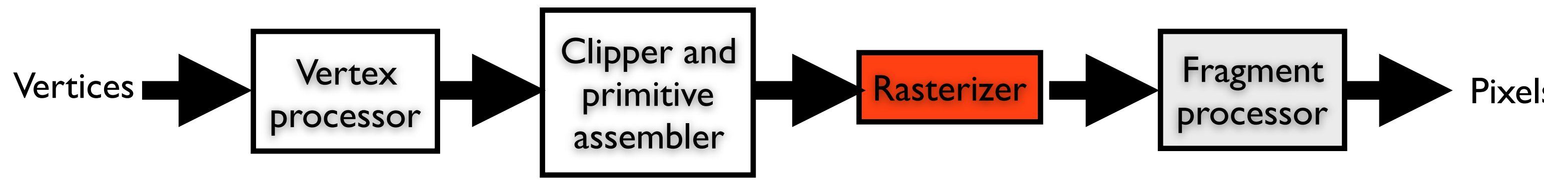
Then organized into primitives that are clipped and culled…





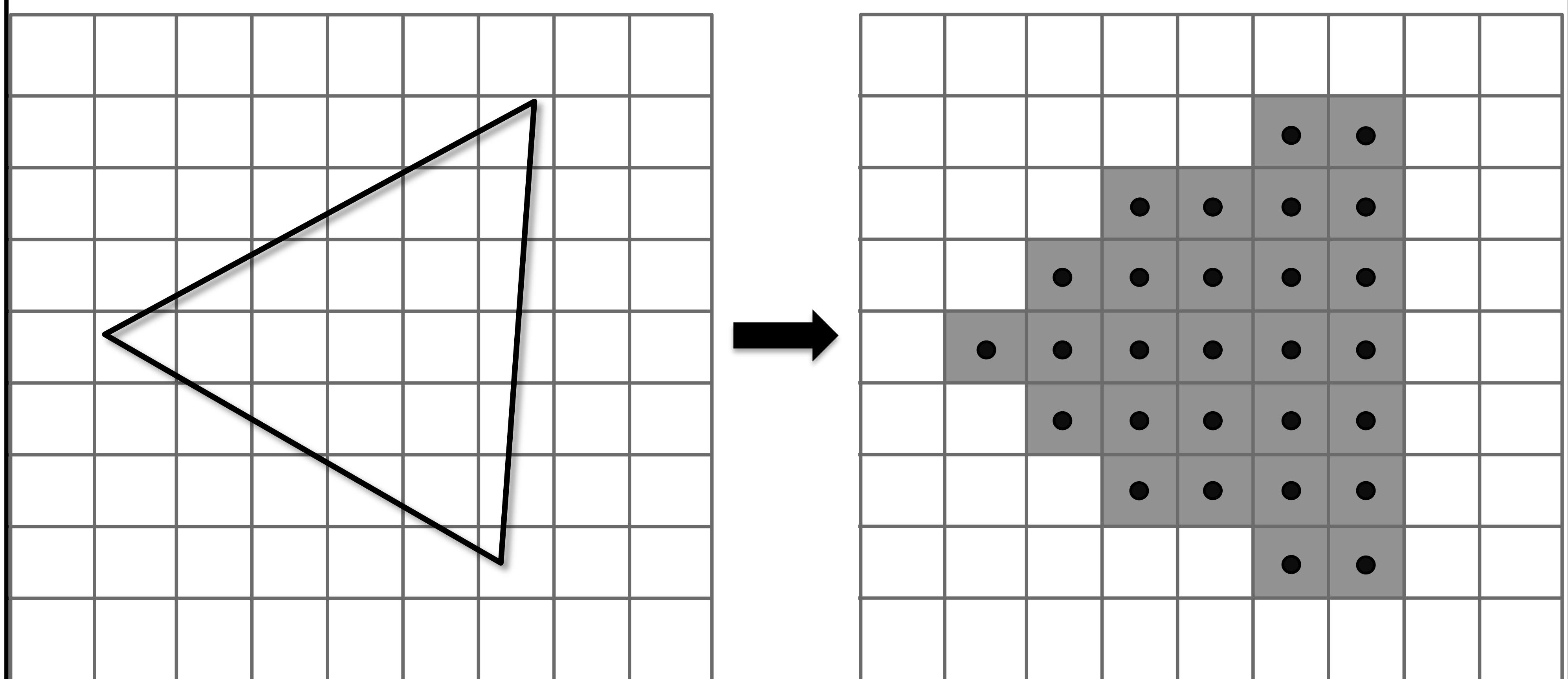
Vertices

Primitives (triangles)

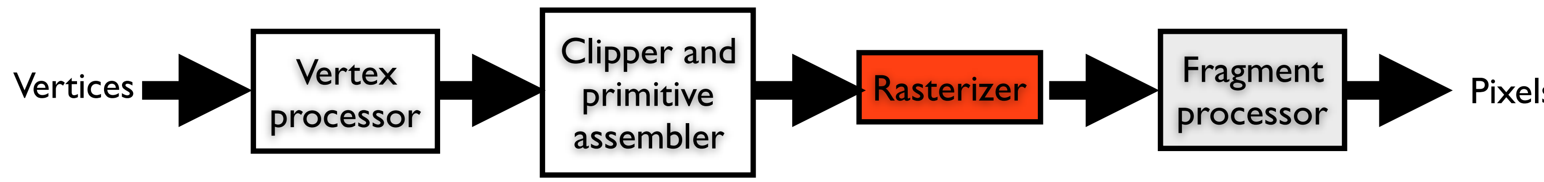


Rasterization

Primitives are rasterized into “pixel fragments”

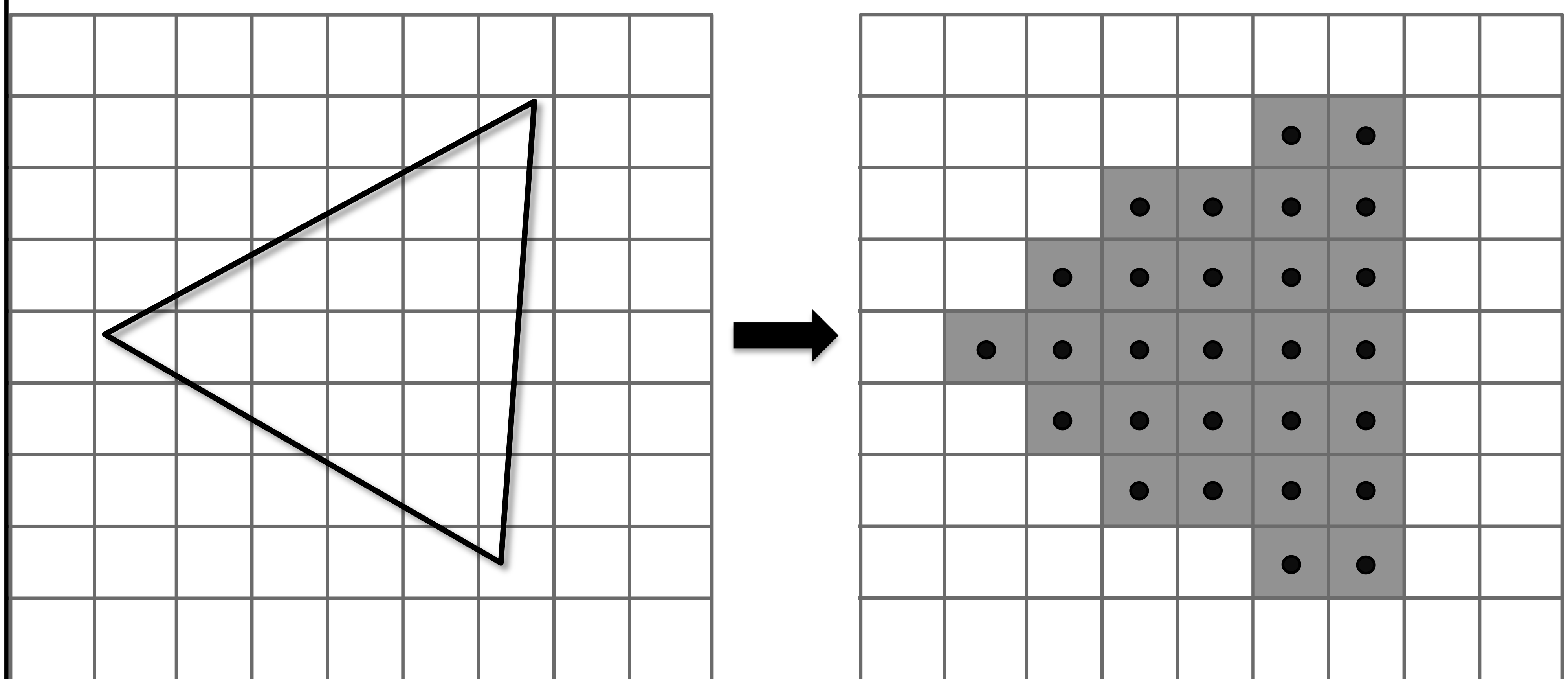


Fragments

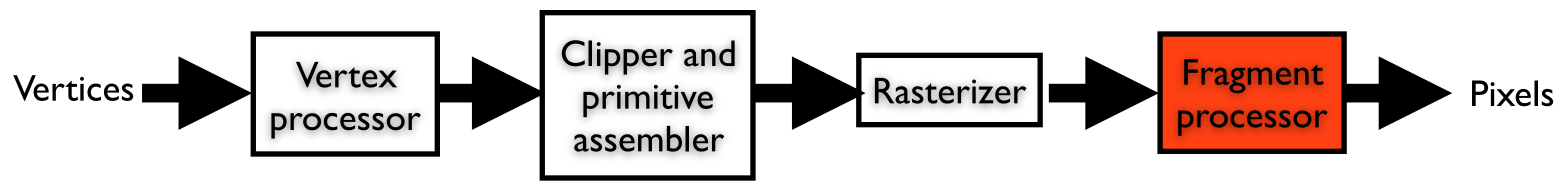


Rasterization

Primitives are rasterized into “pixel fragments”

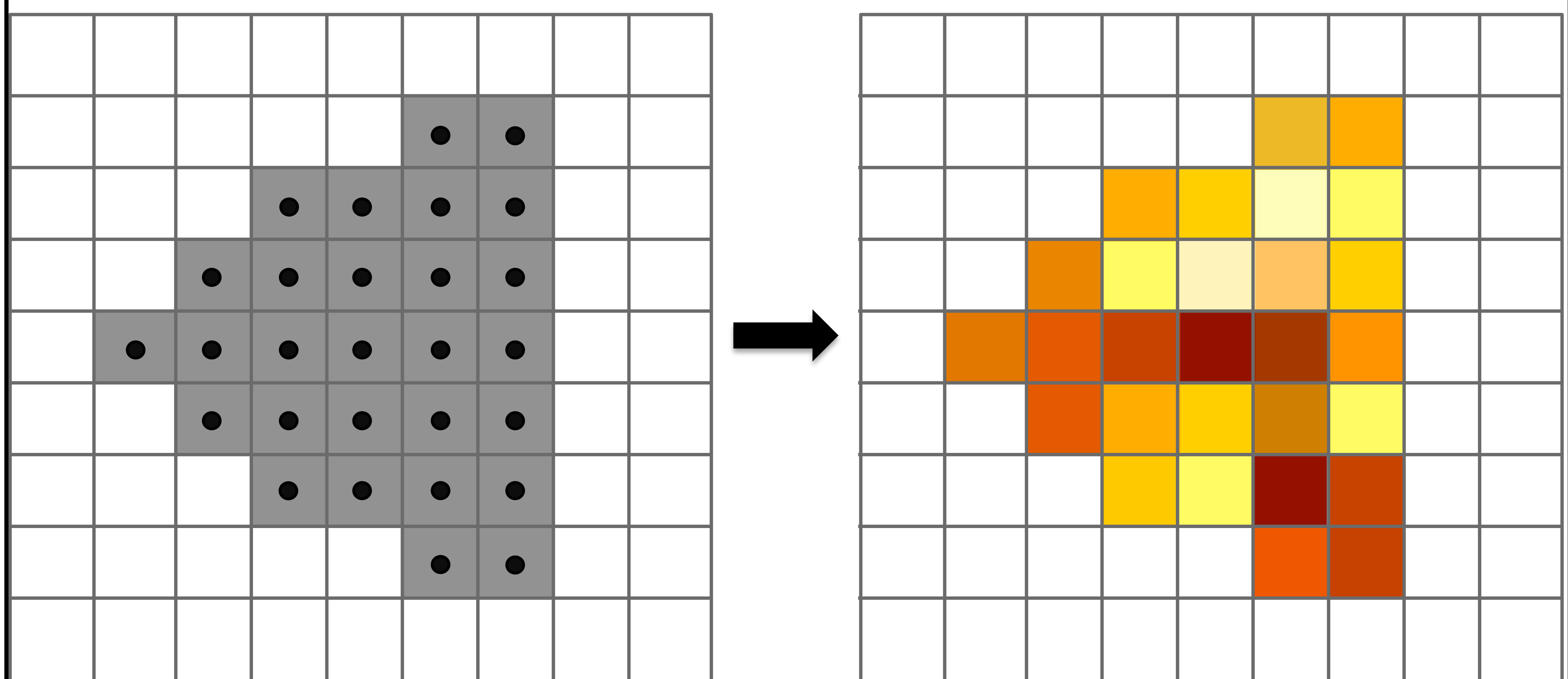


EACH PRIMITIVE IS RASTERIZED INDEPENDENTLY

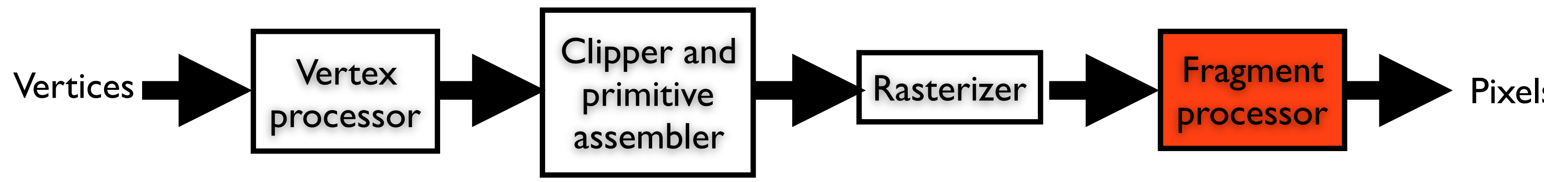


Fragment processing

Fragments are shaded to compute a color at each pixel

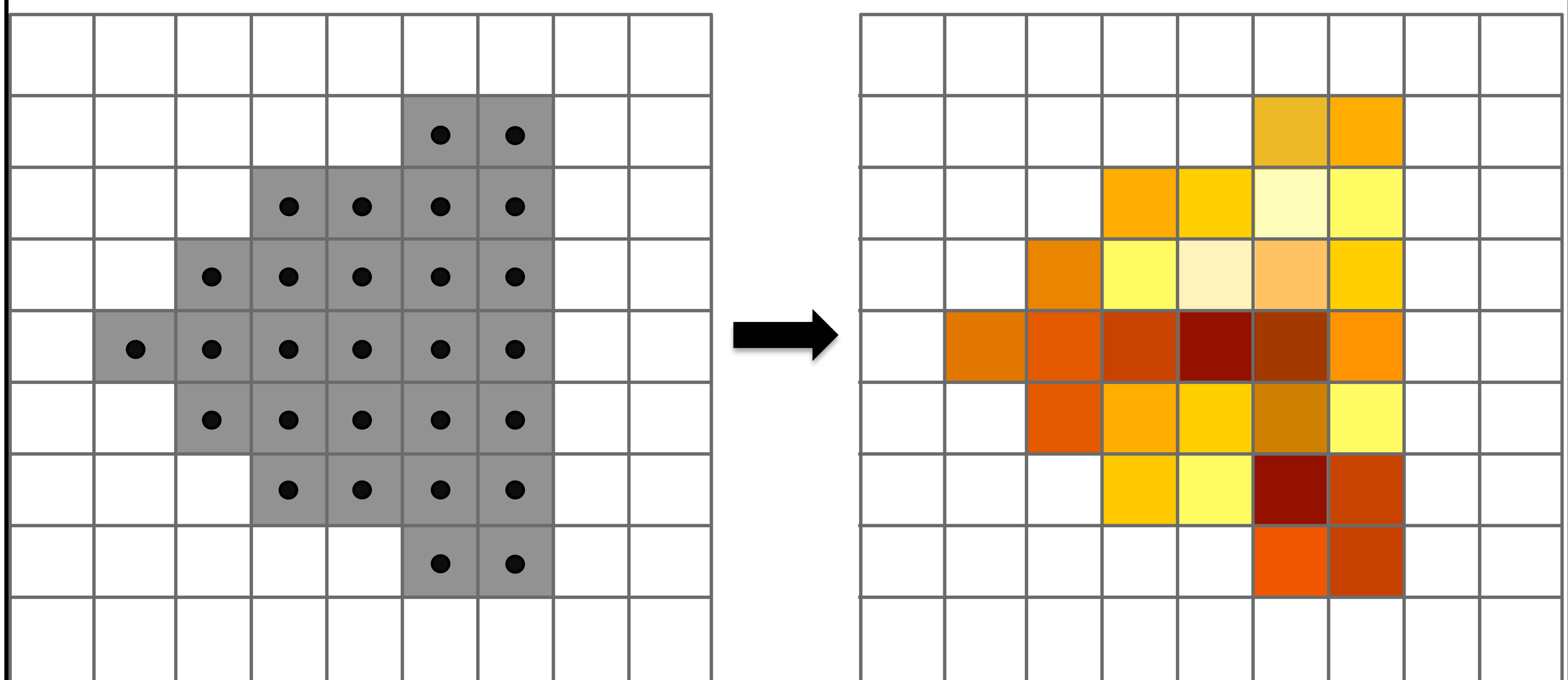


Shaded fragments

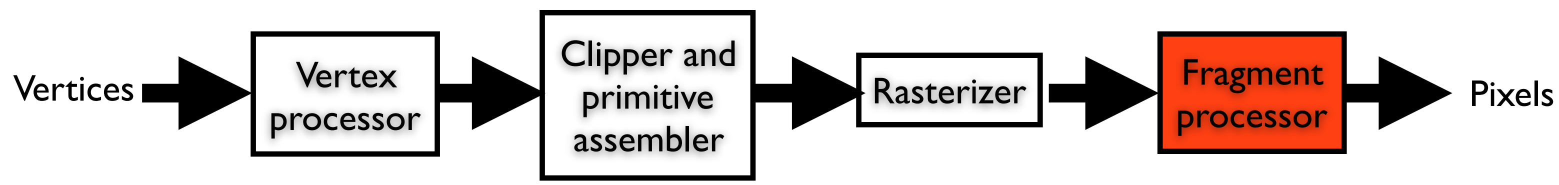


Fragment processing

Fragments are shaded to compute a color at each pixel

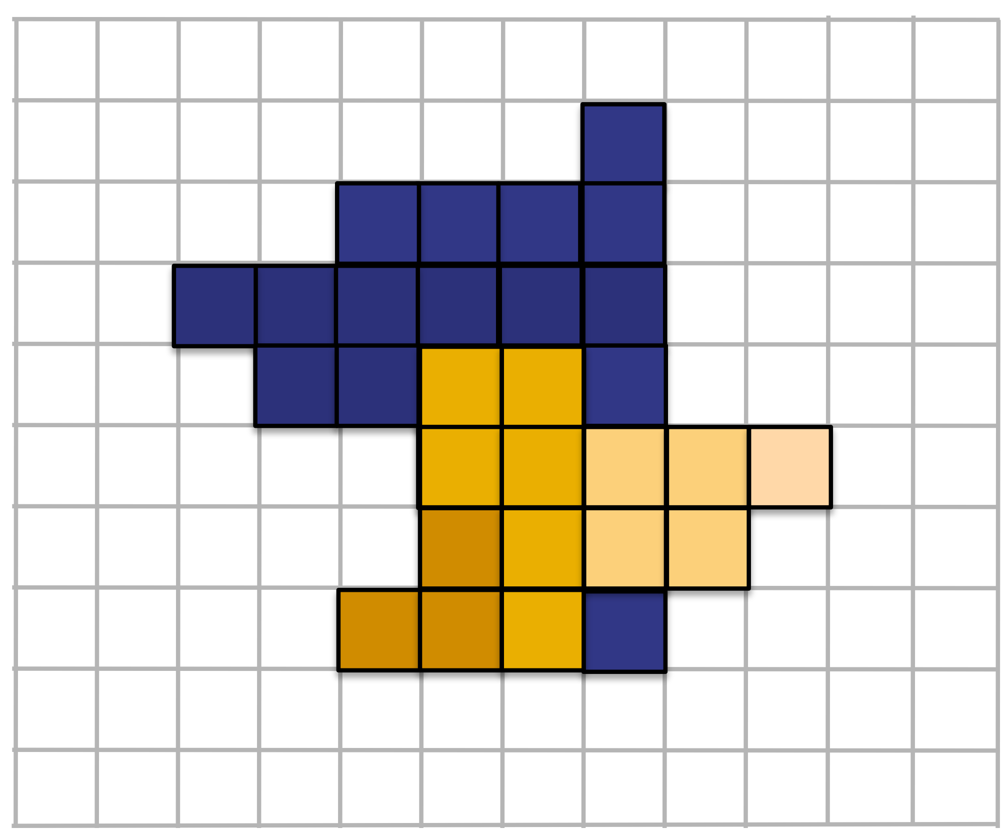


EACH FRAGMENT IS PROCESSED INDEPENDENTLY



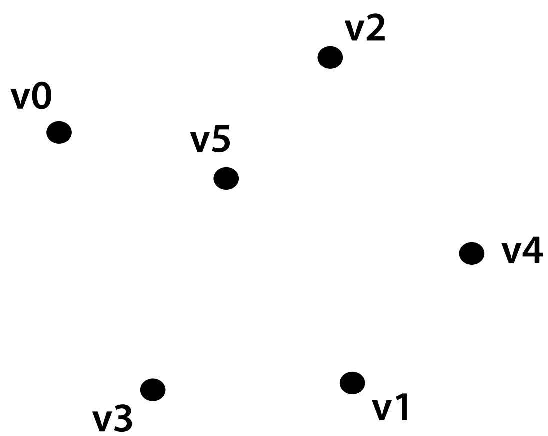
Pixel operations

Fragments are blended into the frame buffer at their pixel locations (z-buffer determines visibility)

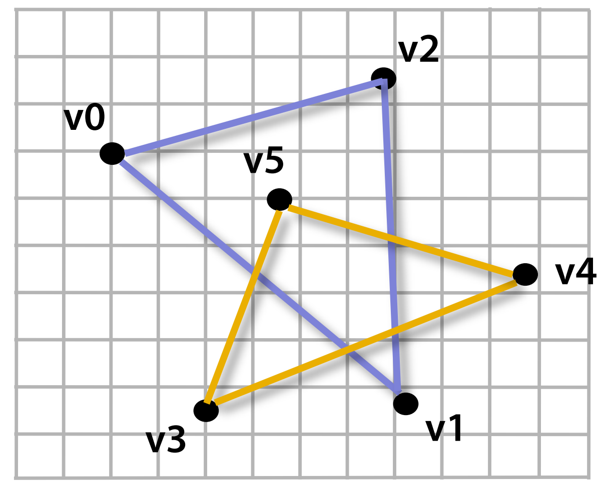


Pixels

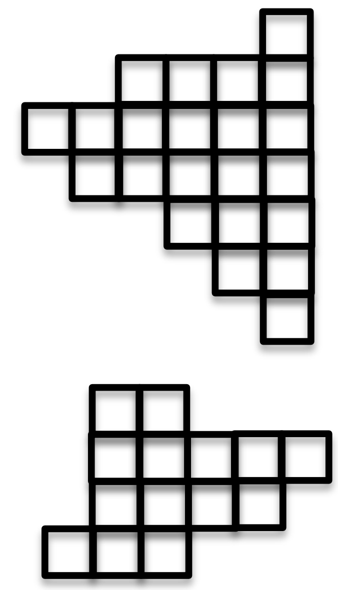
Pipeline entities



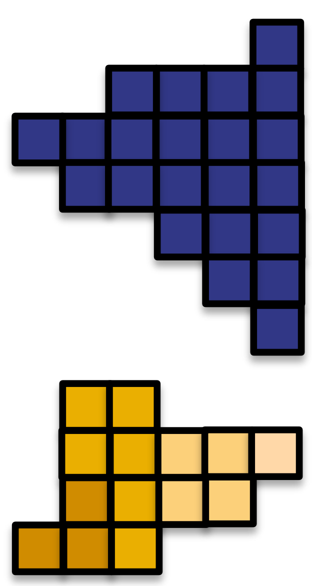
Vertices



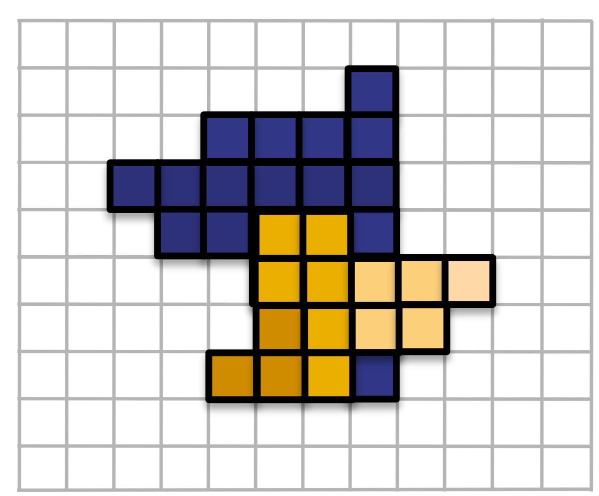
Primitives



Fragments



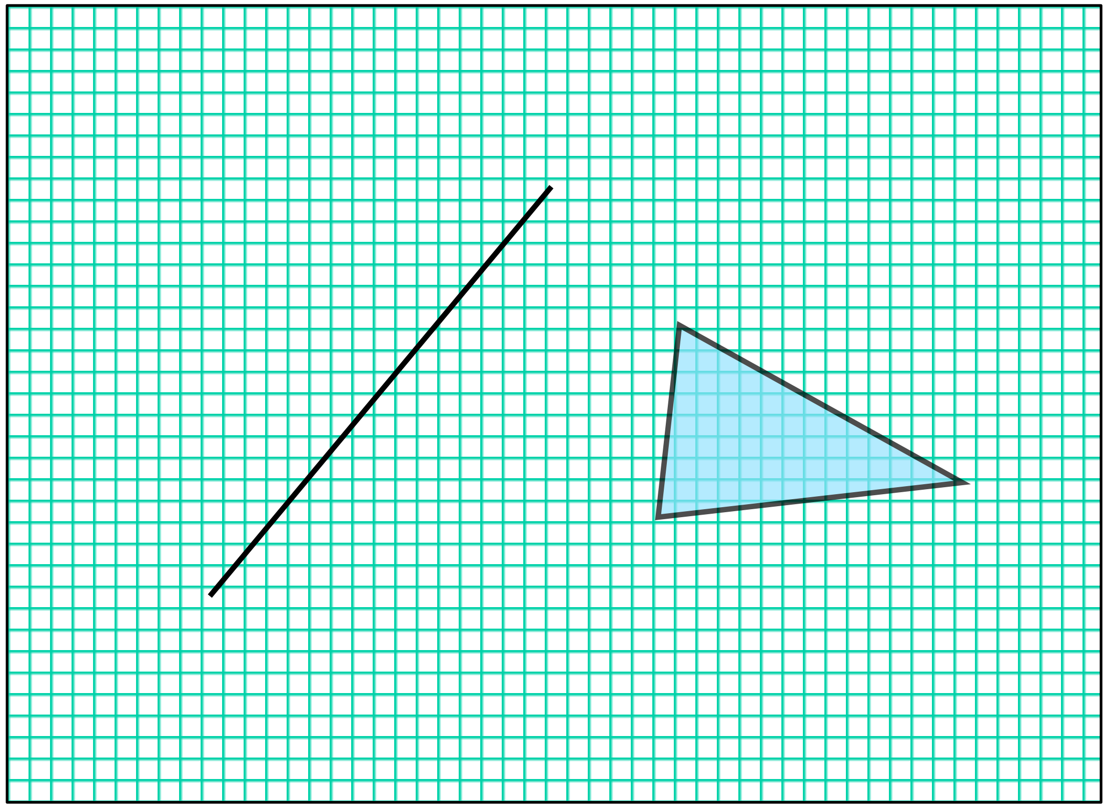
Fragments (shaded)



Pixels

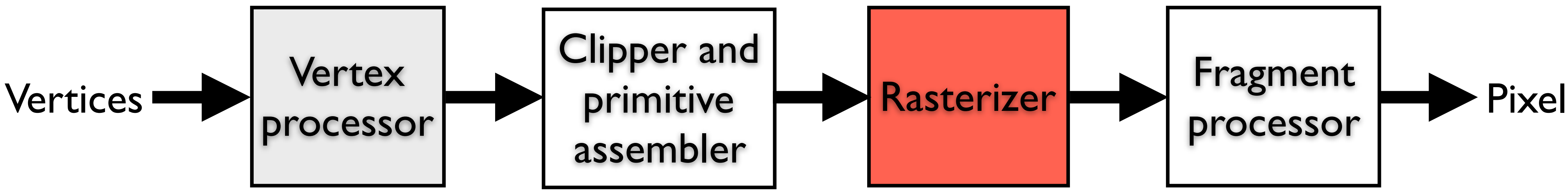
Rasterization

What is rasterization?



Rasterization is the process of determining which pixels are "covered" by the primitive

What is rasterization?



input: primitives **output**: fragments

enumerate the pixels covered by a primitive

interpolate attributes across the primitive

Rasterization

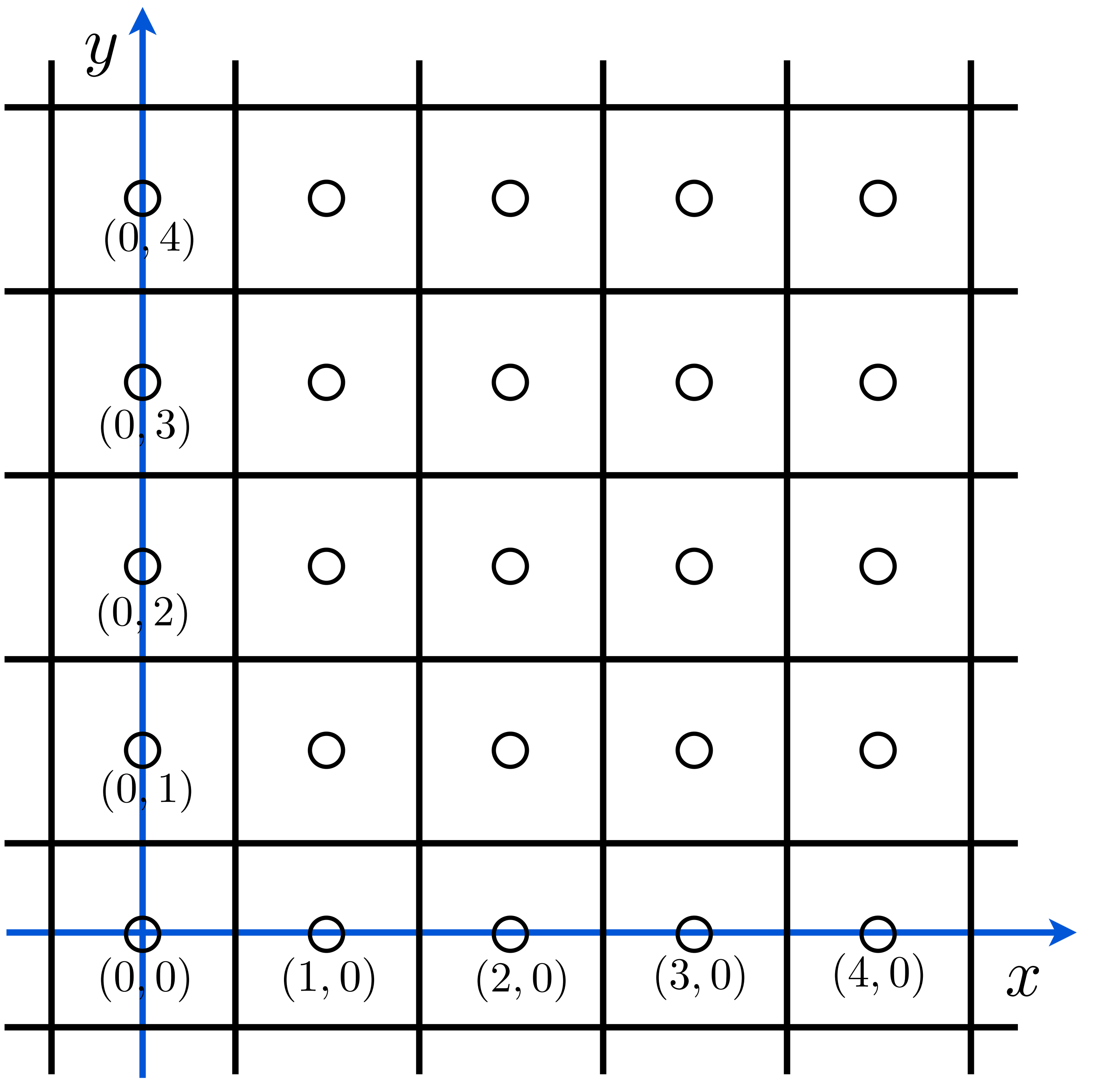
Compute integer coordinates for pixels covered by the 2D primitives

Algorithms are invoked many, many times and so must be efficient

Output should be visually pleasing, for example, lines should have constant density

Obviously, should be able to draw all possible 2D primitives

Screen coordinates

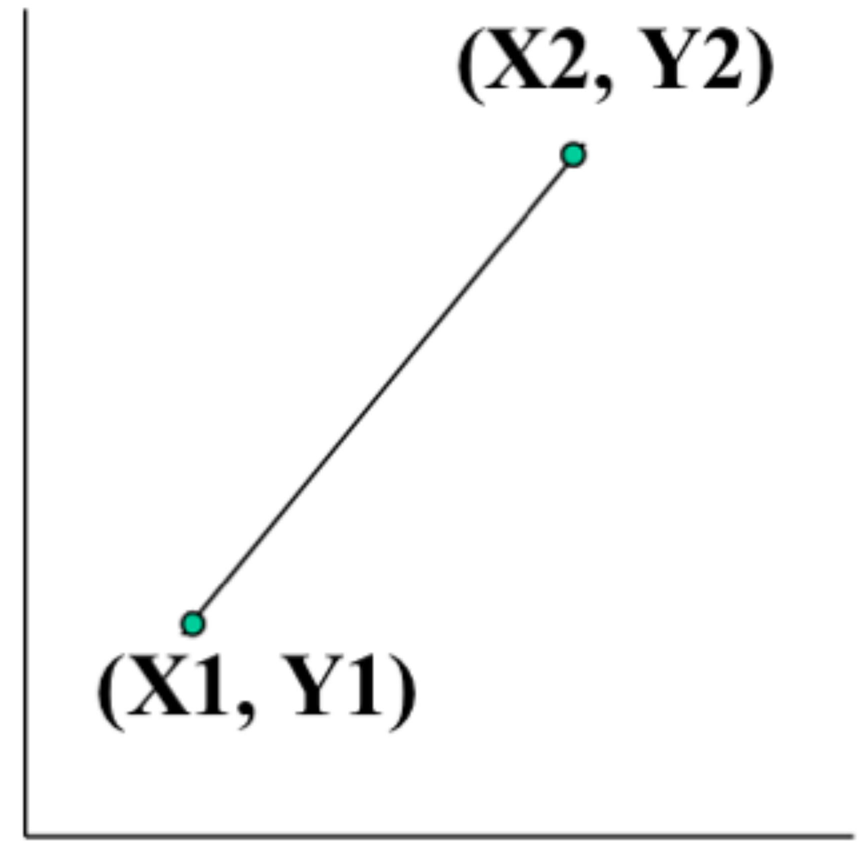


Line Representation

Math Review

- 2D math for lines

How do we determine the equation of the line?



Math Review

- 2D math for lines

Slope-Intercept formula for a line

Slope = (Y2 – Y1)/(X2 – X1)

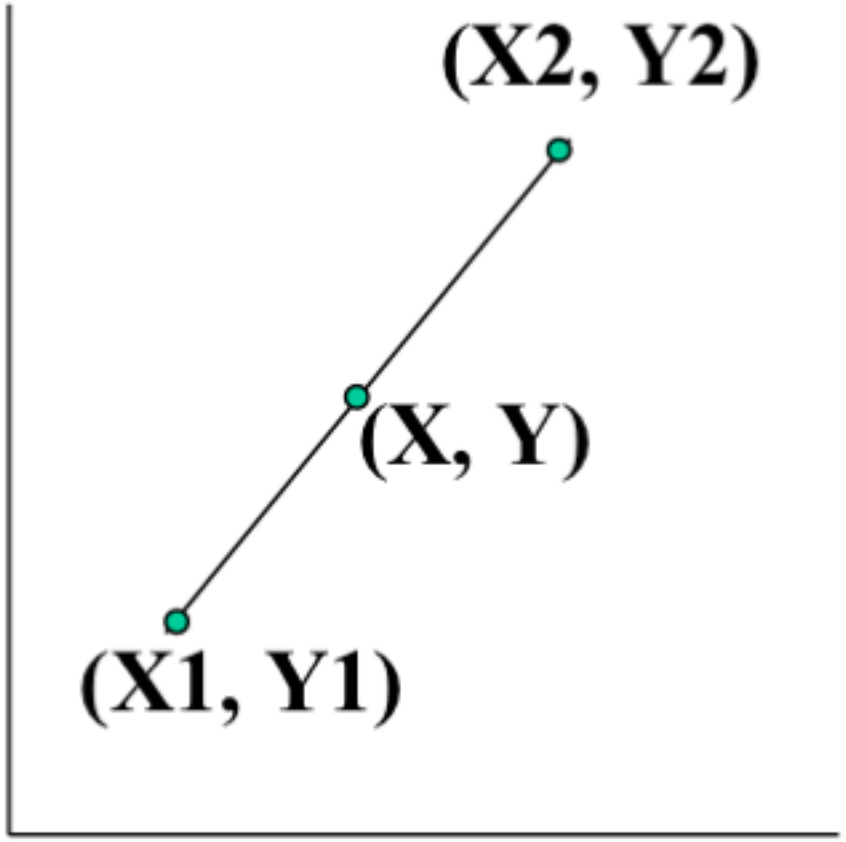
(Y - Y1)/(X - X1)

Solving For Y

Y = [(Y2 – Y1)/(X2 – X1)]X

+ [-(Y2 – Y1)/(X2 - X1)]X1 + Y1 or

Y = m X + b



Math Review

- Explicit (functional) representation

  $y = f(x)$

  y is the dependent, x independent variable

  Find value of y from value of x

  Example, for a line: $y = mx + b$

  for a circle: $x^2 + y^2 = r^2$

Math Review

- Parametric Representation

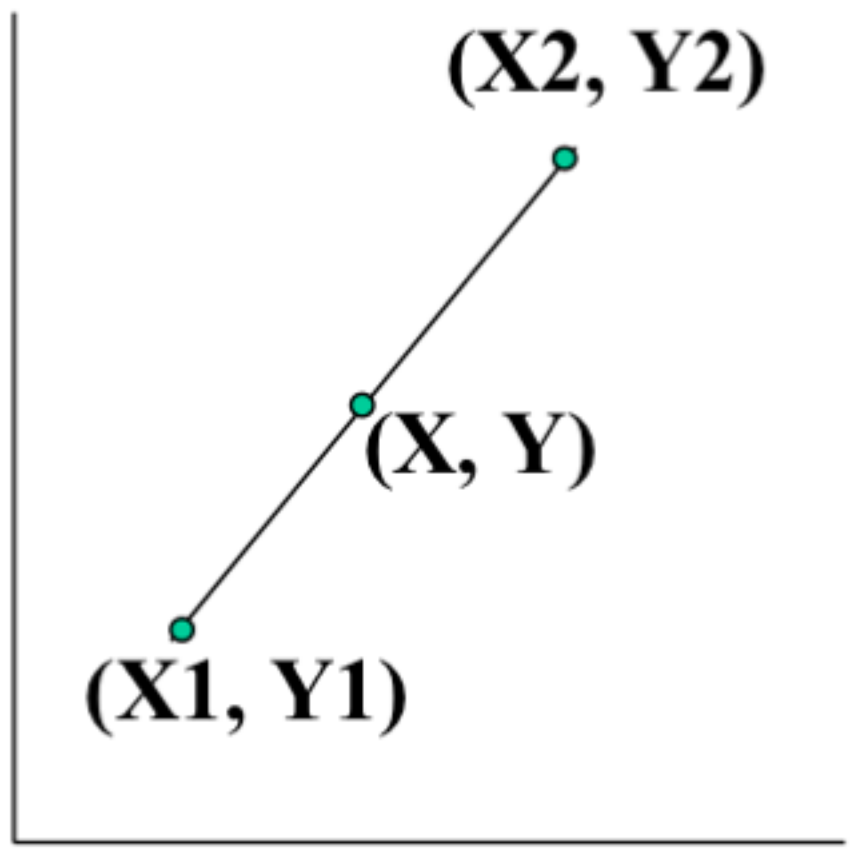
$$x = x(u),\ y = y(u)$$

where new parameter u (or often t) determines the value of x and y (and possibly z) for each point

x,y treated the same, axis invariant

Math Review

Parametric formula for a line



$X = X1 + t(X2 - X1)$

$Y = Y1 + t(Y2 - Y1)$

for parameter t from 0 to 1

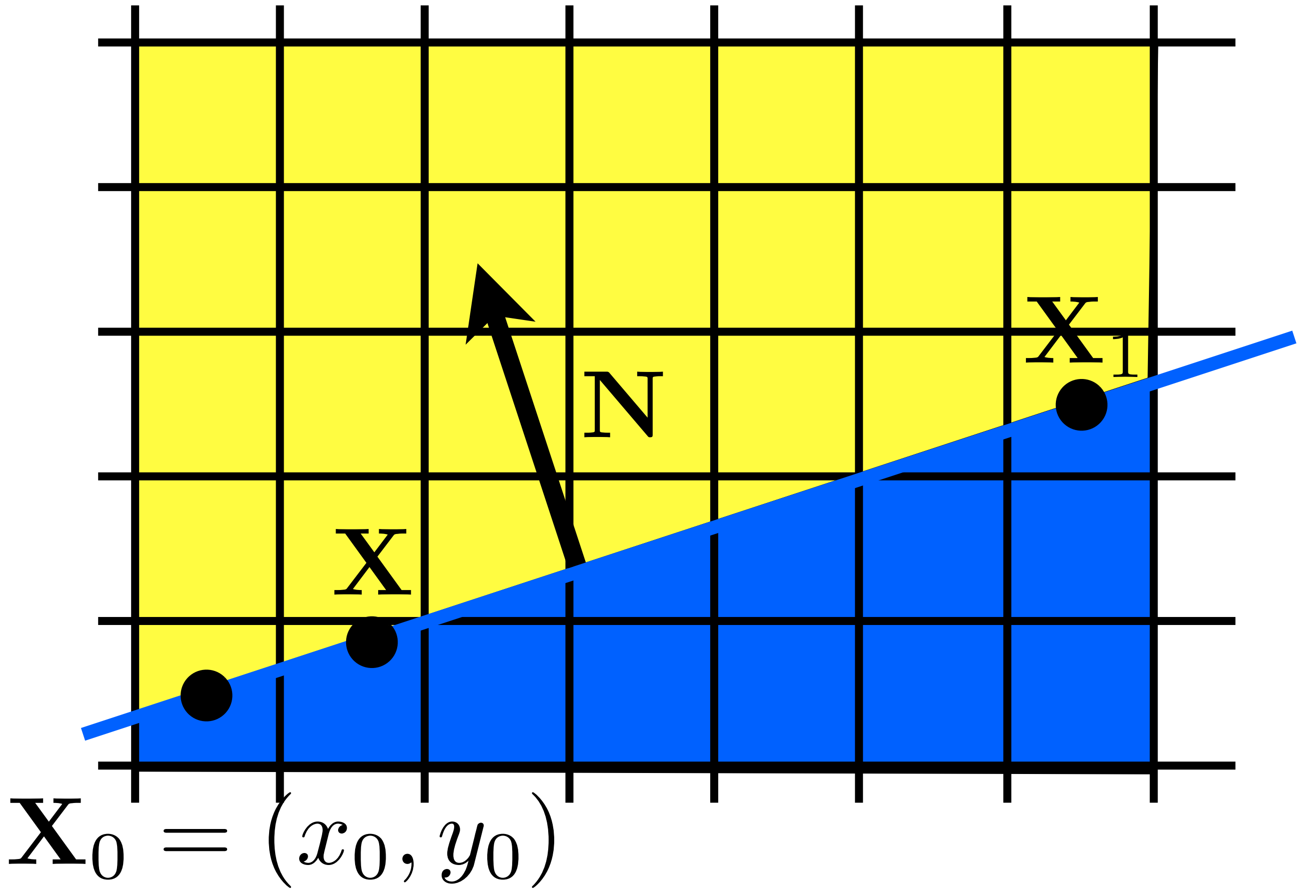
Therefore, when

$t = 0$ we get (X1,Y1)

$t = 1$ we get (X2,Y2)

Varying t gives the points along the line segment

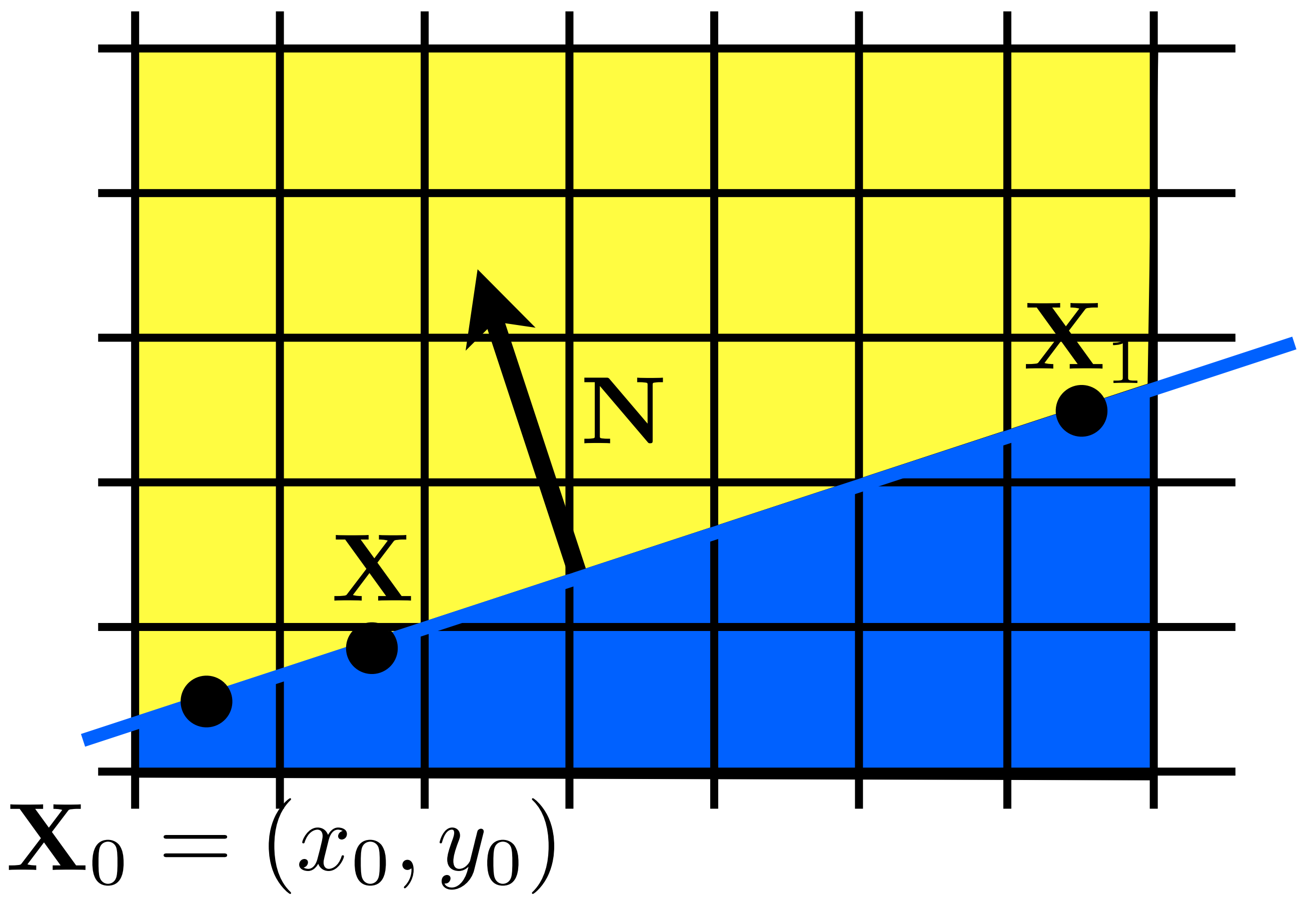
Implicit Line Equation



$$f(\mathbf{X}) = \mathbf{N} \cdot (\mathbf{X} - \mathbf{X}_0) = 0$$

<whiteboard>

Implicit Line Equation



decision variable, d

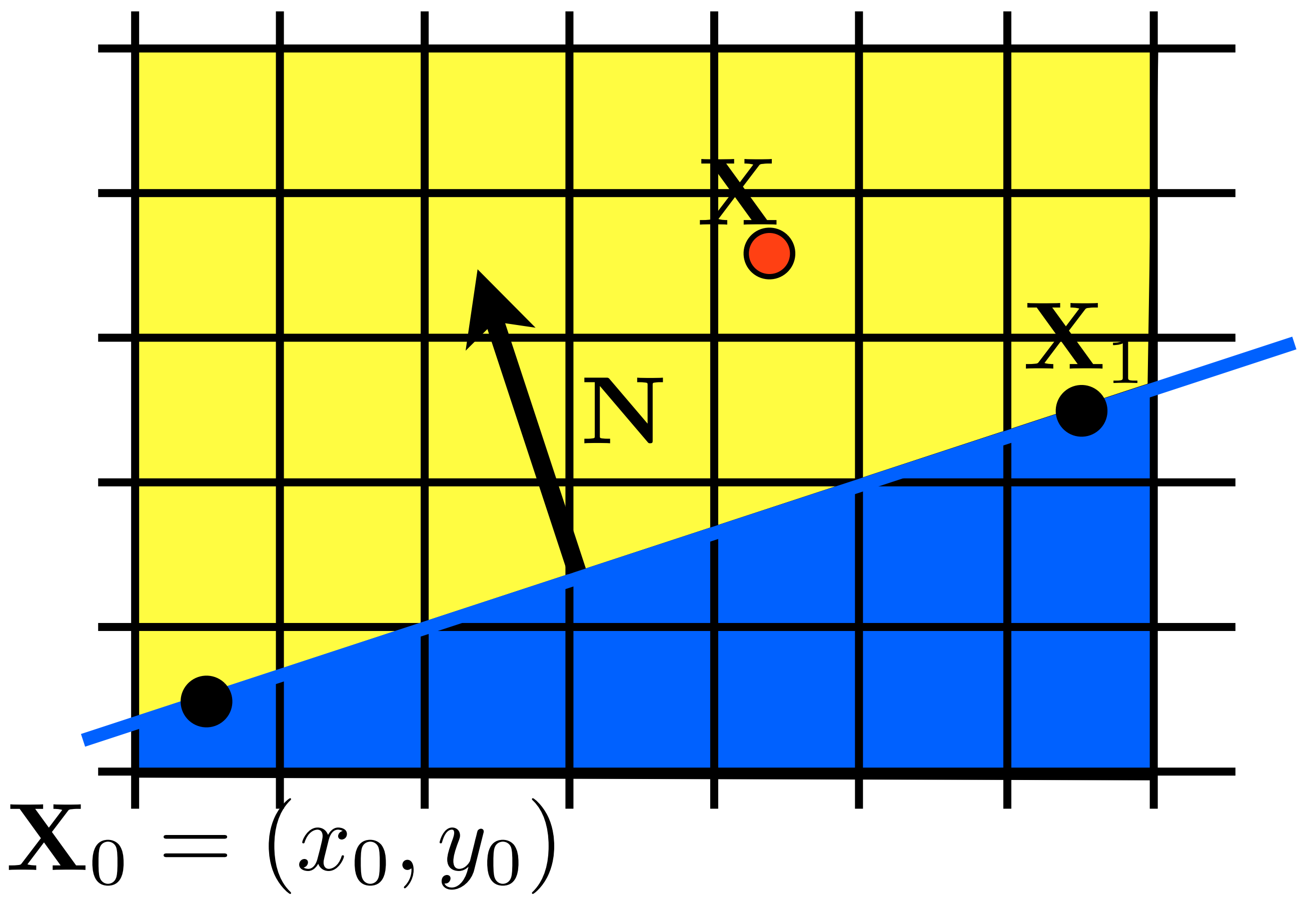
$$f(\mathbf{X}) = \mathbf{N} \cdot (\mathbf{X} - \mathbf{X}_0) = d$$

$$d > 0$$

$$d < 0$$

$$d = 0$$

Implicit Line Equation



decision variable, d

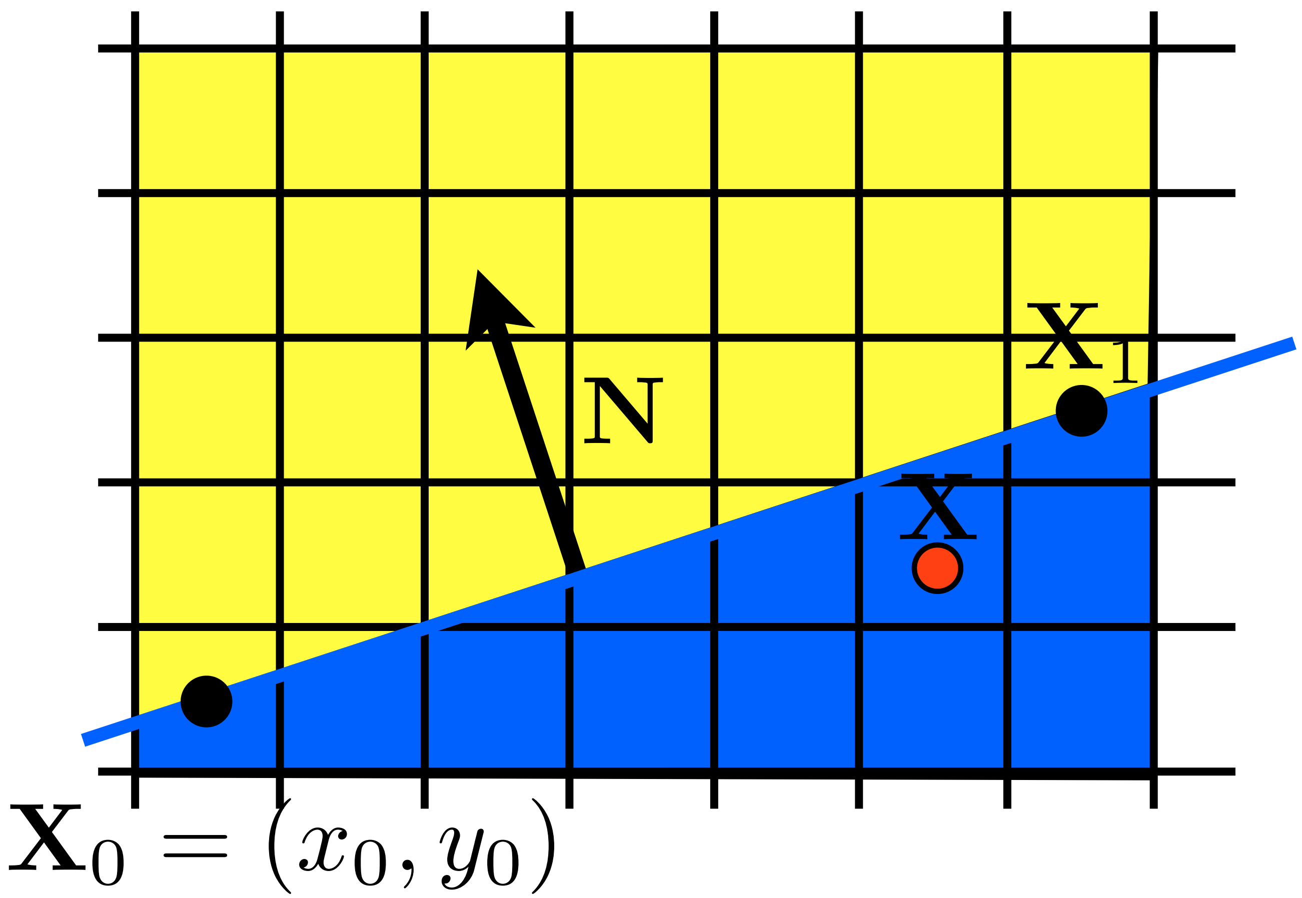
$$f(\mathbf{X}) = \mathbf{N} \cdot (\mathbf{X} - \mathbf{X}_0) = d$$

$$d > 0$$

$$d < 0$$

$$d = 0$$

Implicit Line Equation



decision variable, d

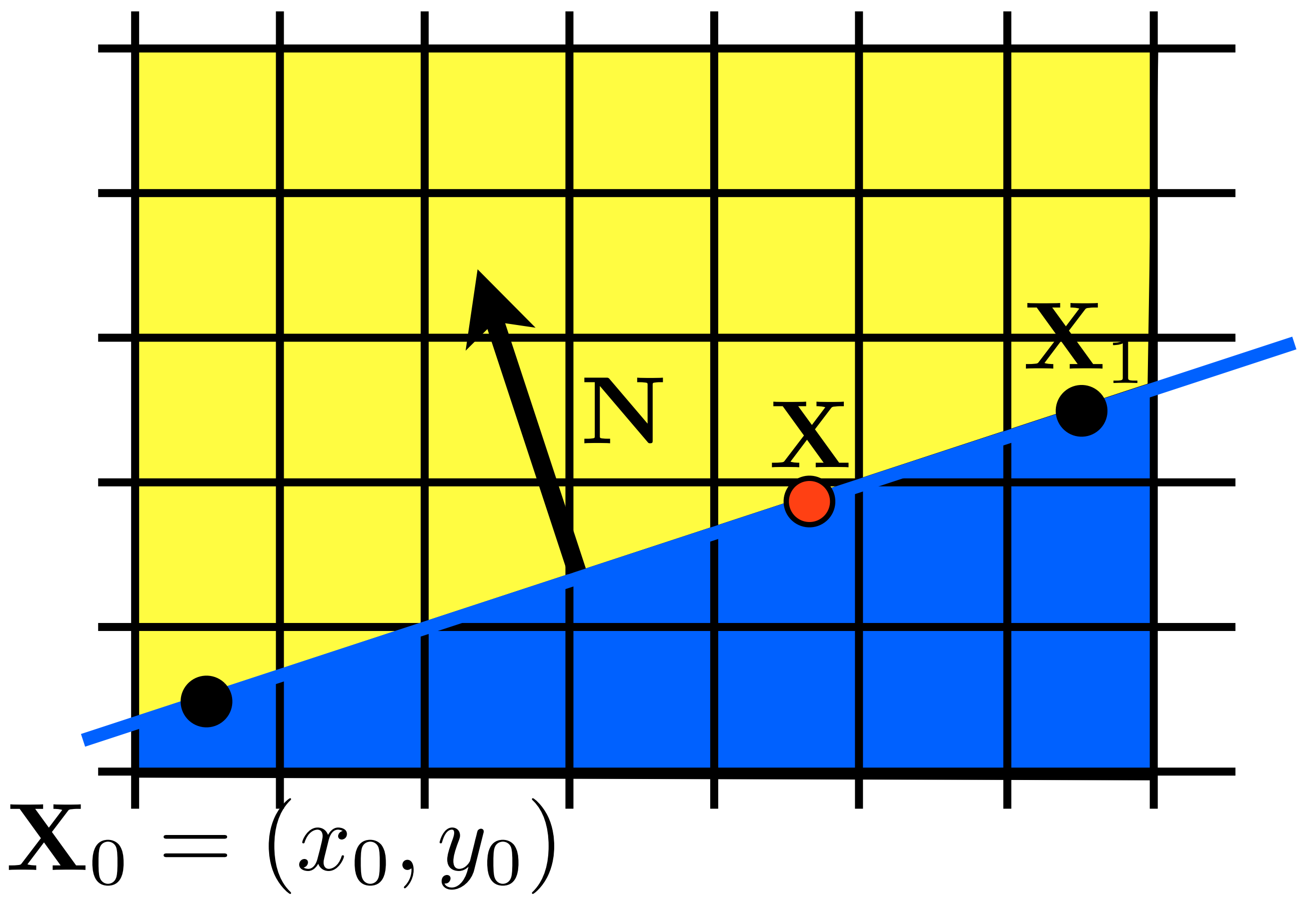
$$f(\mathbf{X}) = \mathbf{N} \cdot (\mathbf{X} - \mathbf{X}_0) = d$$

$$d > 0$$

$$d < 0$$

$$d = 0$$

Implicit Line Equation



decision variable, d

$$f(\mathbf{X}) = \mathbf{N} \cdot (\mathbf{X} - \mathbf{X}_0) = d$$

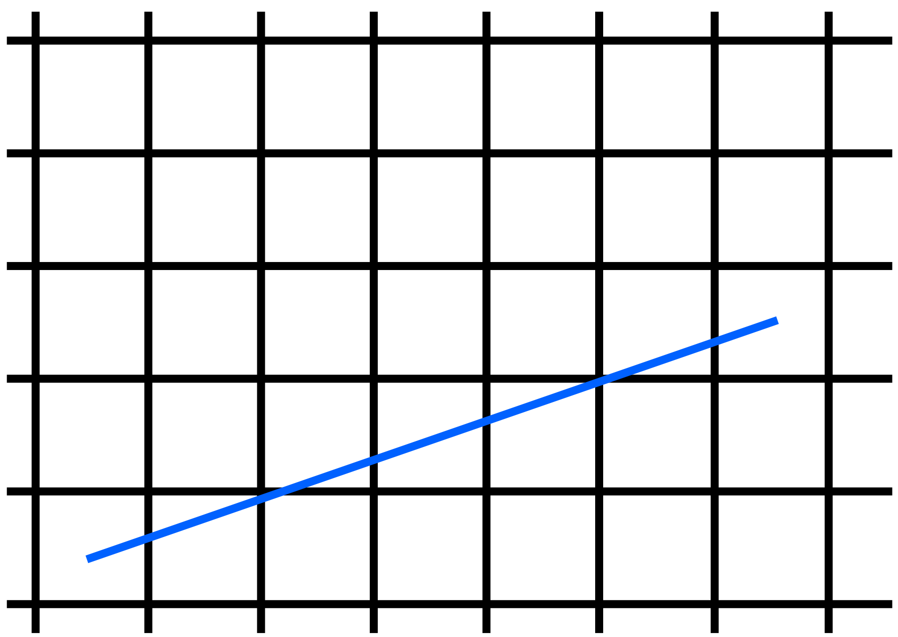
$$d > 0$$

$$d < 0$$

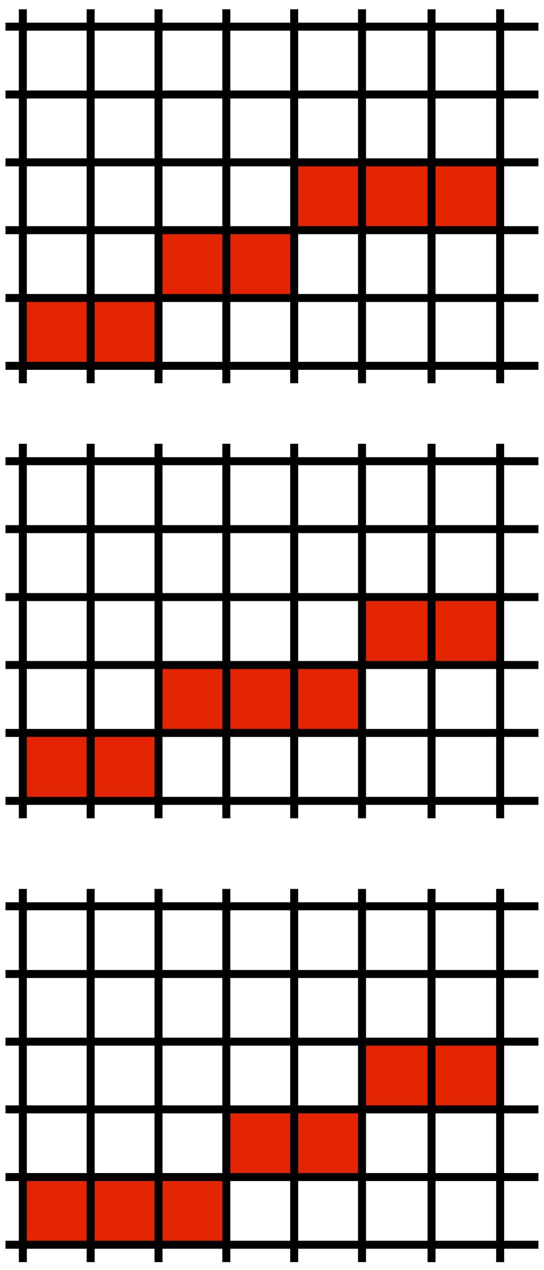
$$d = 0$$

Line Drawing

Which pixels should be used to approximate a line?



Draw the thinnest possible line that has no gaps



DDA algorithm for lines

Parametric Lines: the DDA algorithm
(digital differential analyzer)

$Y_{i+1} = m\, x_{i+1} + B$

$= m(x_i + \Delta x) + B \qquad \Delta x = (x_{i+1} - x_i)$

$= y_i + m(\Delta x)$ **<- must round to find int**

If we increment by 1 pixel in X, we turn on
[xi, Round(yi)] or same for Y if m > 1

Scan conversion for lines

DDA includes Round( ); and this is fairly slow

For Fast Lines, we want to do only integer math +,-

We do this using the **Midpoint Algorithm**

To do this, lets look at lines with y-intercept B
and with slope between 0 and 1:

$$y = (dy/dx)x + B \quad \Longrightarrow$$

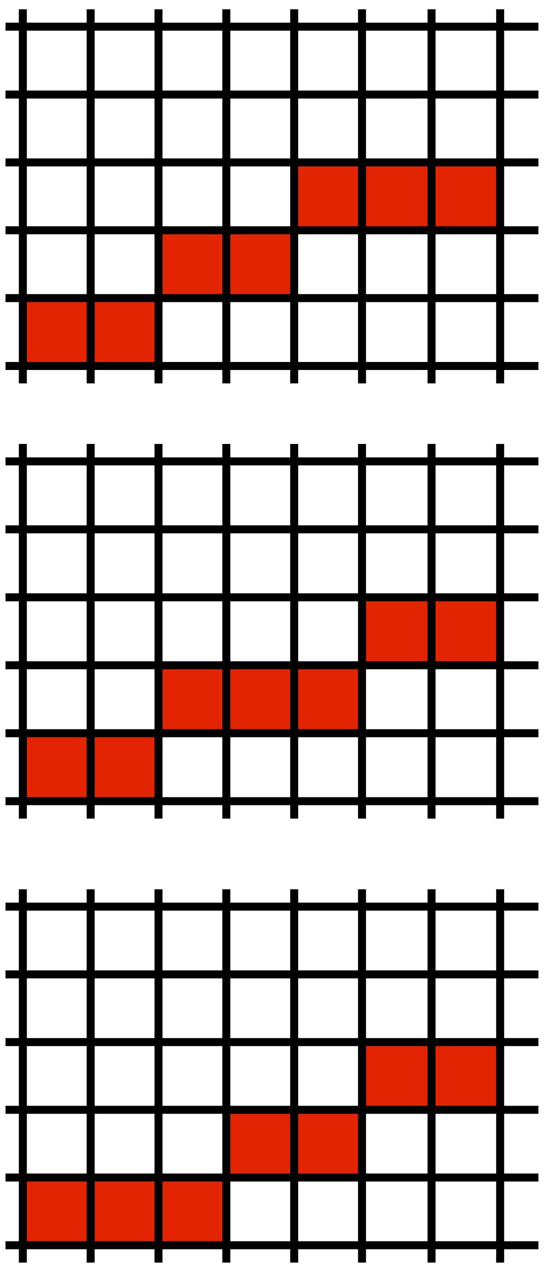
$$f(x,y) = (dy)x - (dx)y + B(dx) = 0$$

Removes the division => slope treated as 2 integers

Line drawing algorithm

(case: 0 < m <= 1)

```
y = y0
for x = x0 to x1 do
    draw(x,y)
    if (<condition>) then
        y = y+1
```

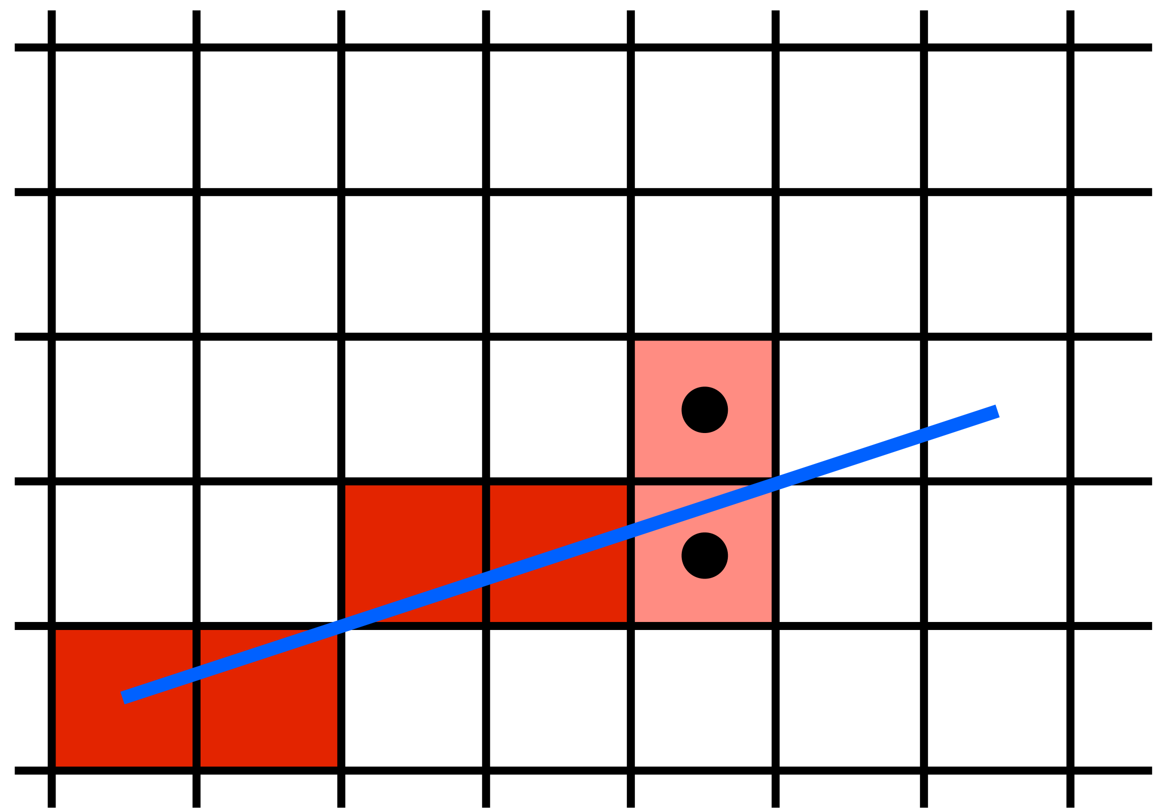


- move from left to right
- choose between (x+1,y) and (x+1,y+1)

Line drawing algorithm

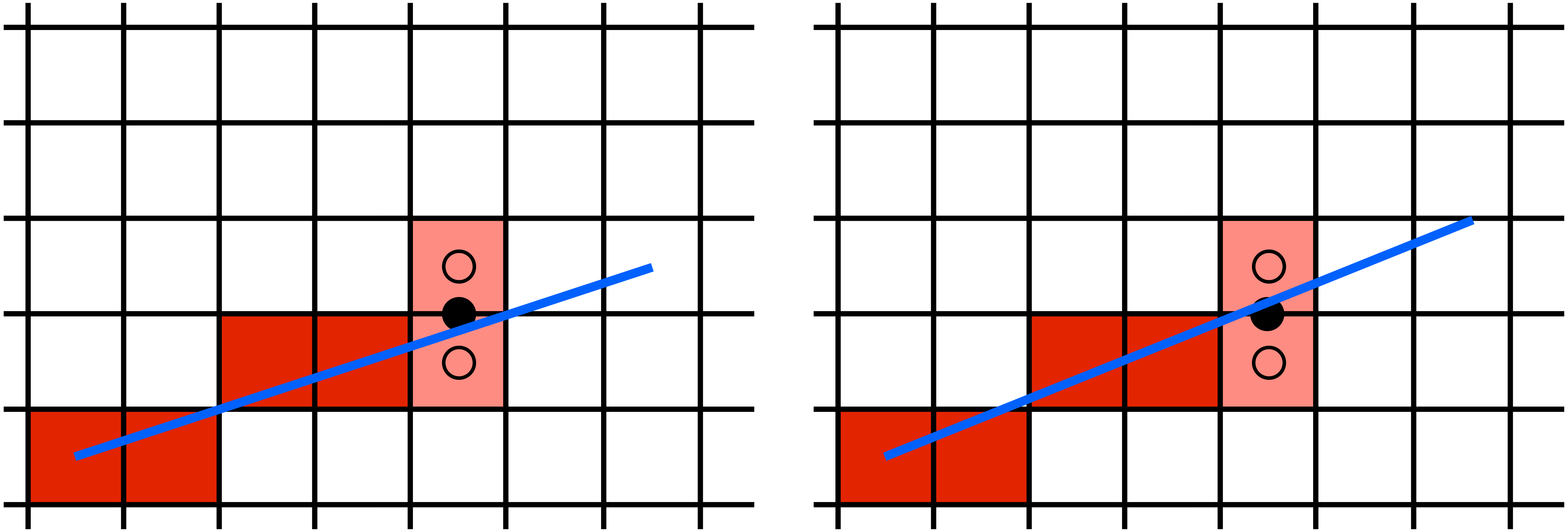
(case: 0 < m <= 1)

```
y = y0
for x = x0 to x1 do
    draw(x,y)
    if (<condition>) then
        y = y+1
```

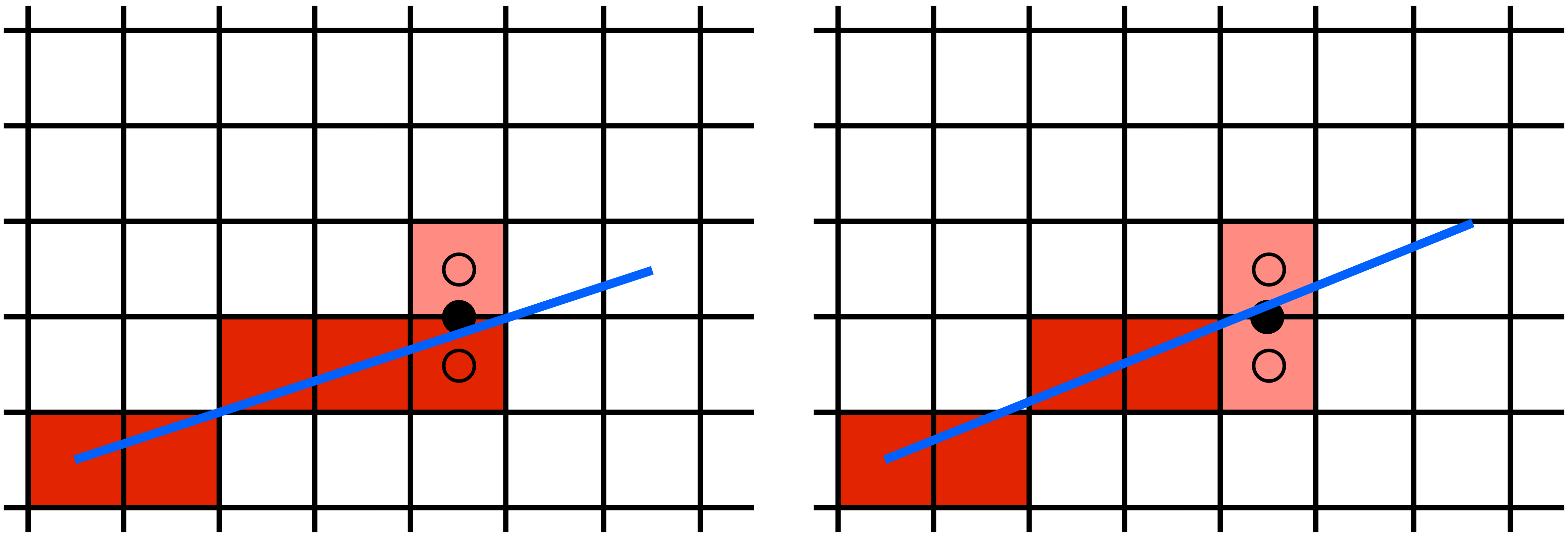


- move from left to right
- choose between (x+1,y) and (x+1,y+1)

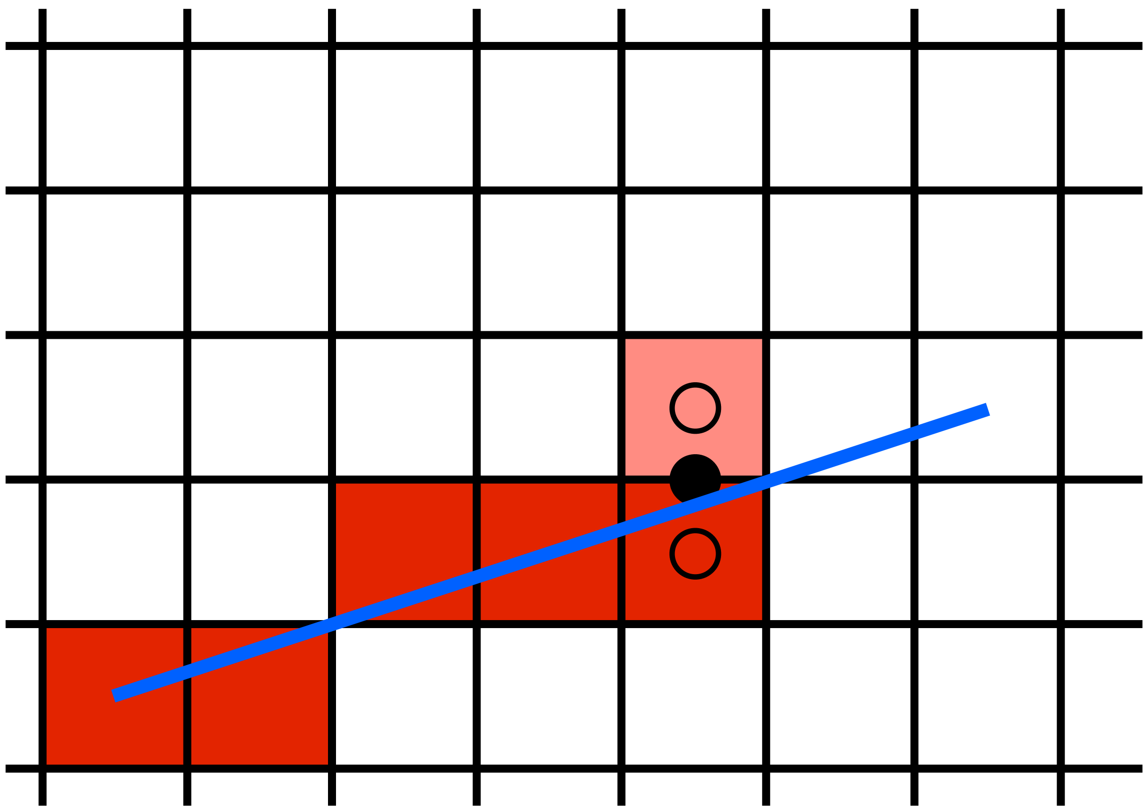
Use the midpoint between the two pixels to choose

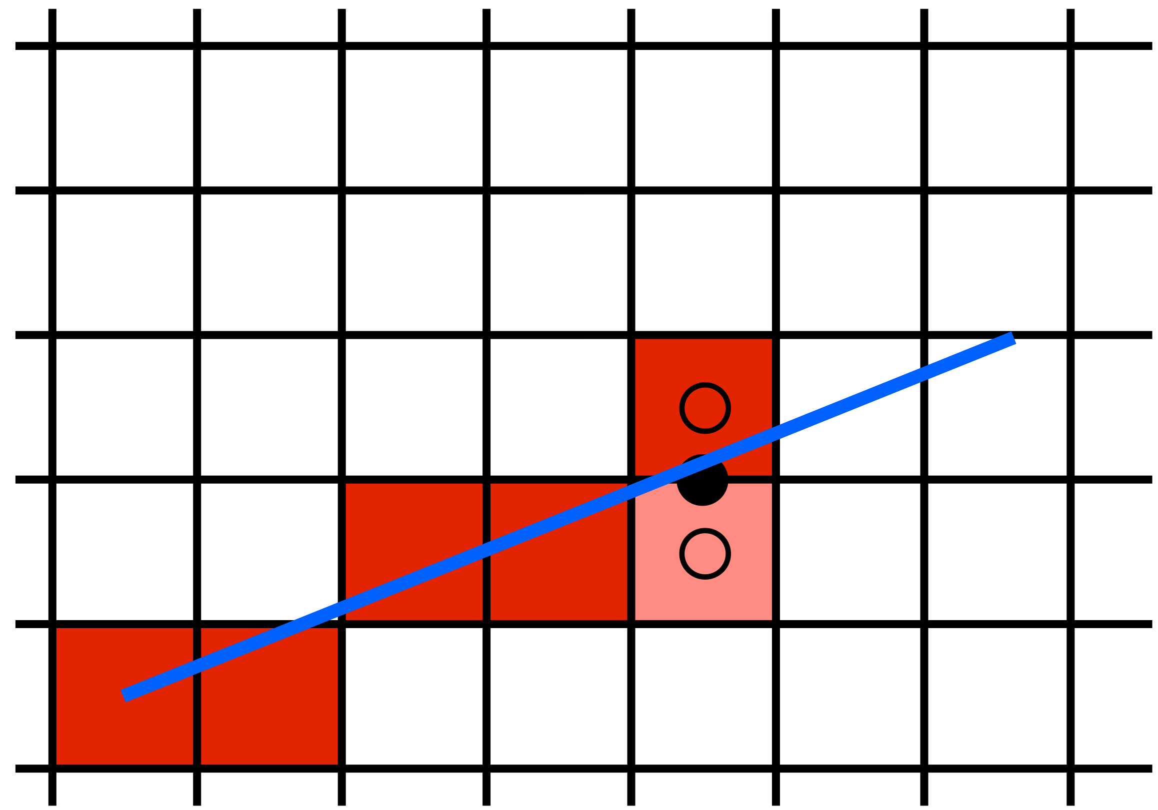


Use the midpoint between the two pixels to choose

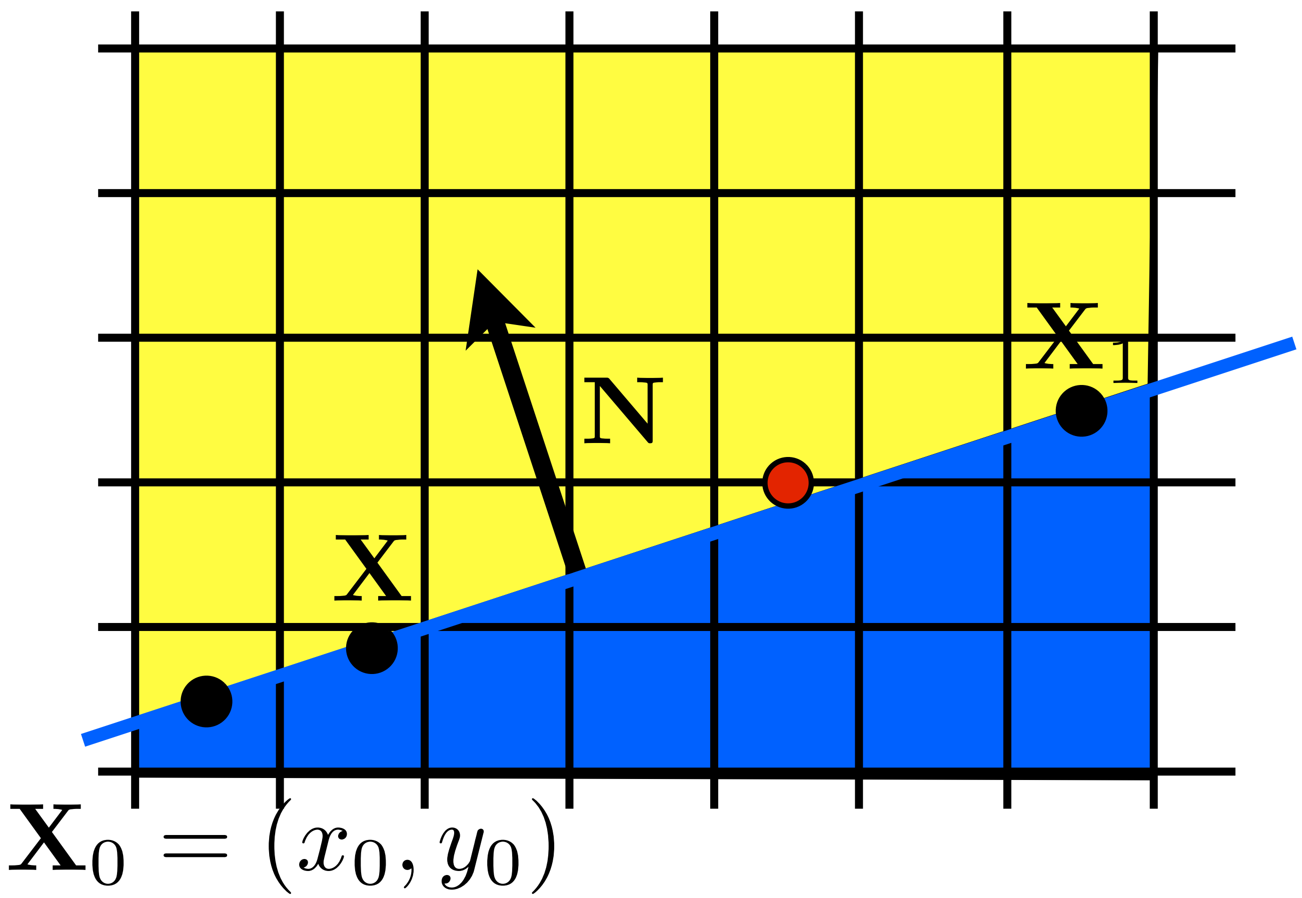


Use the midpoint between the two pixels to choose





Use the midpoint between the two pixels to choose



implicit line equation:

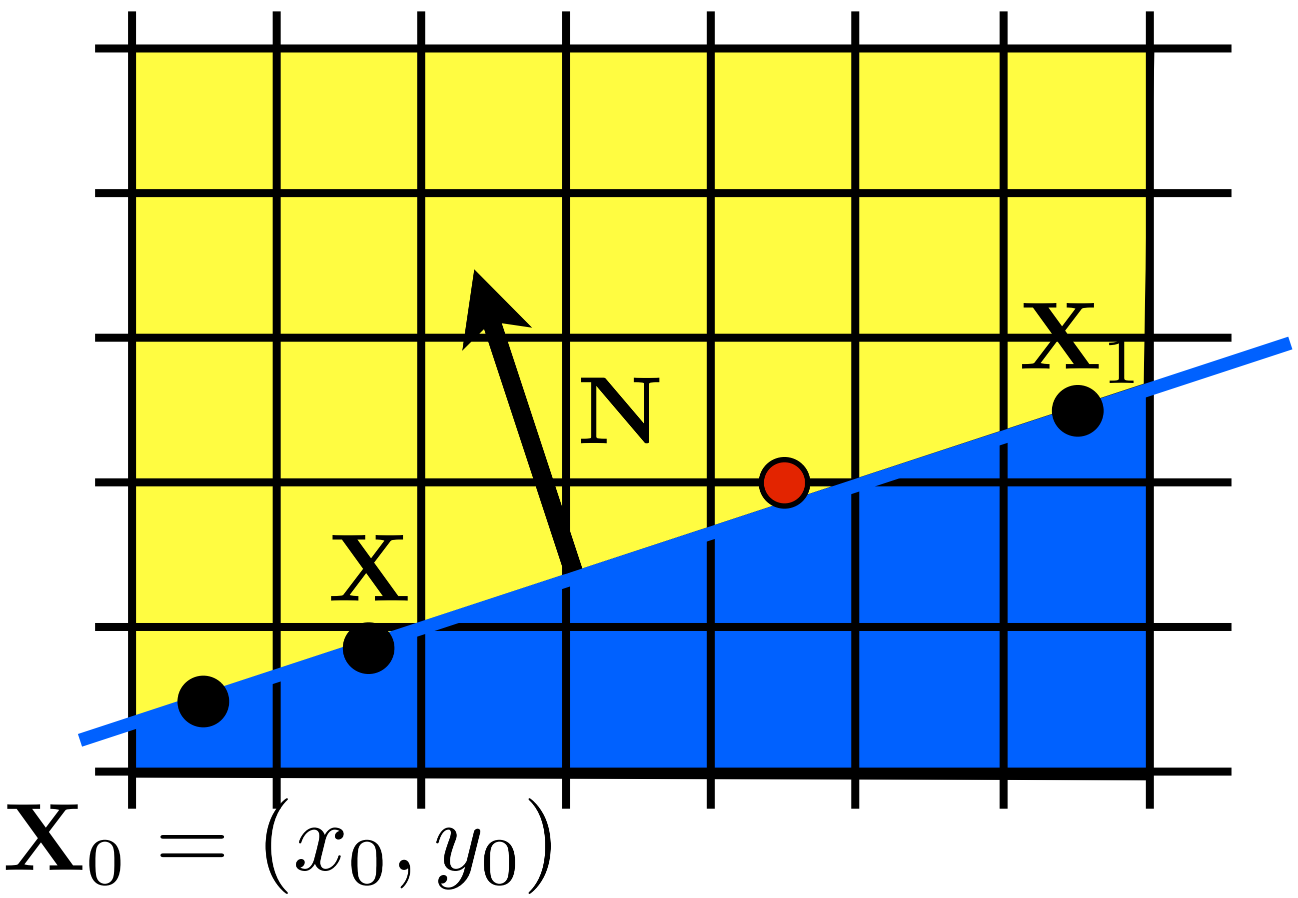
$$f(\mathbf{X}) = \mathbf{N} \cdot (\mathbf{X} - \mathbf{X}_0) = 0$$

<whiteboard>

evaluate f at midpoint:

$$f(x, y + \frac{1}{2}) \ ? \ 0$$

Use the midpoint between the two pixels to choose



implicit line equation:

$$f(\mathbf{X}) = \mathbf{N} \cdot (\mathbf{X} - \mathbf{X}_0) = 0$$

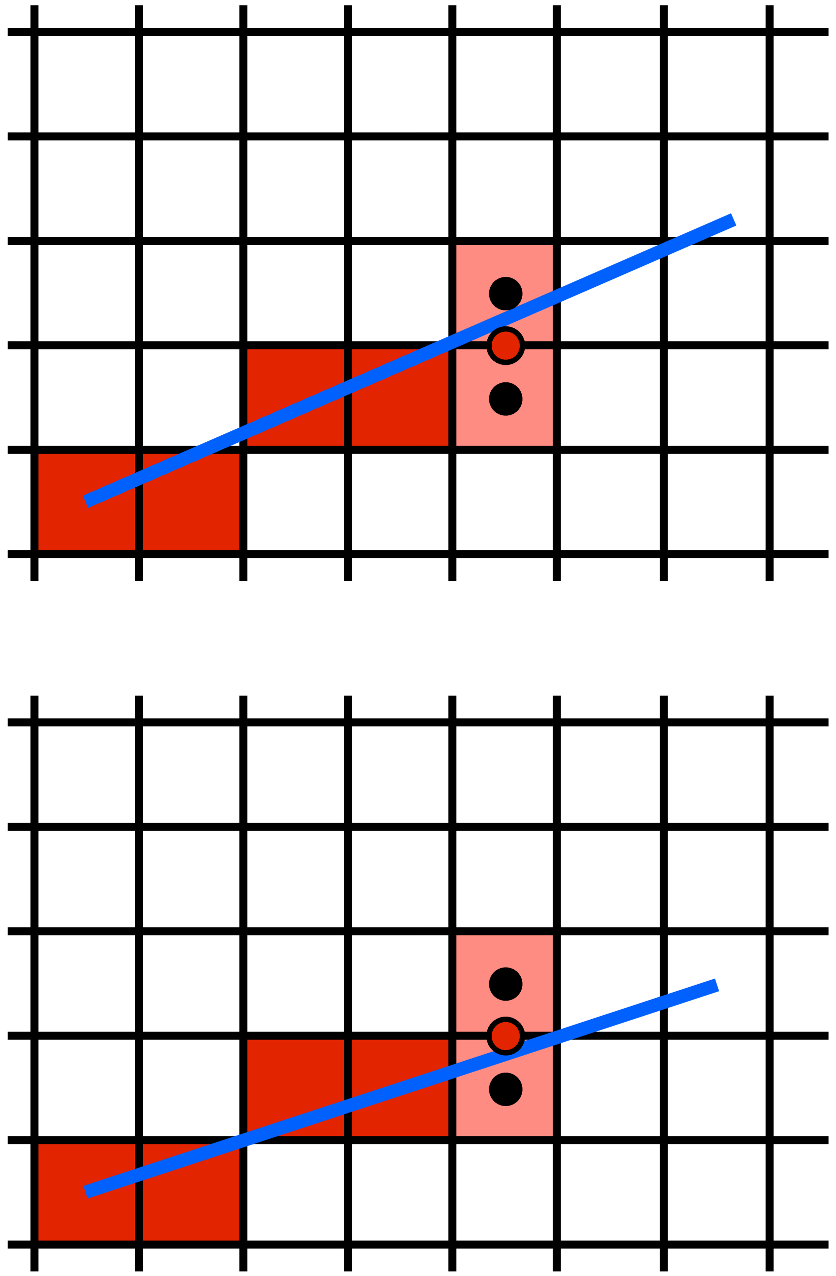
evaluate f at midpoint:

$$f(x, y + \frac{1}{2}) \;>\; 0$$

Line drawing algorithm

(case: $0 < m <= 1$)

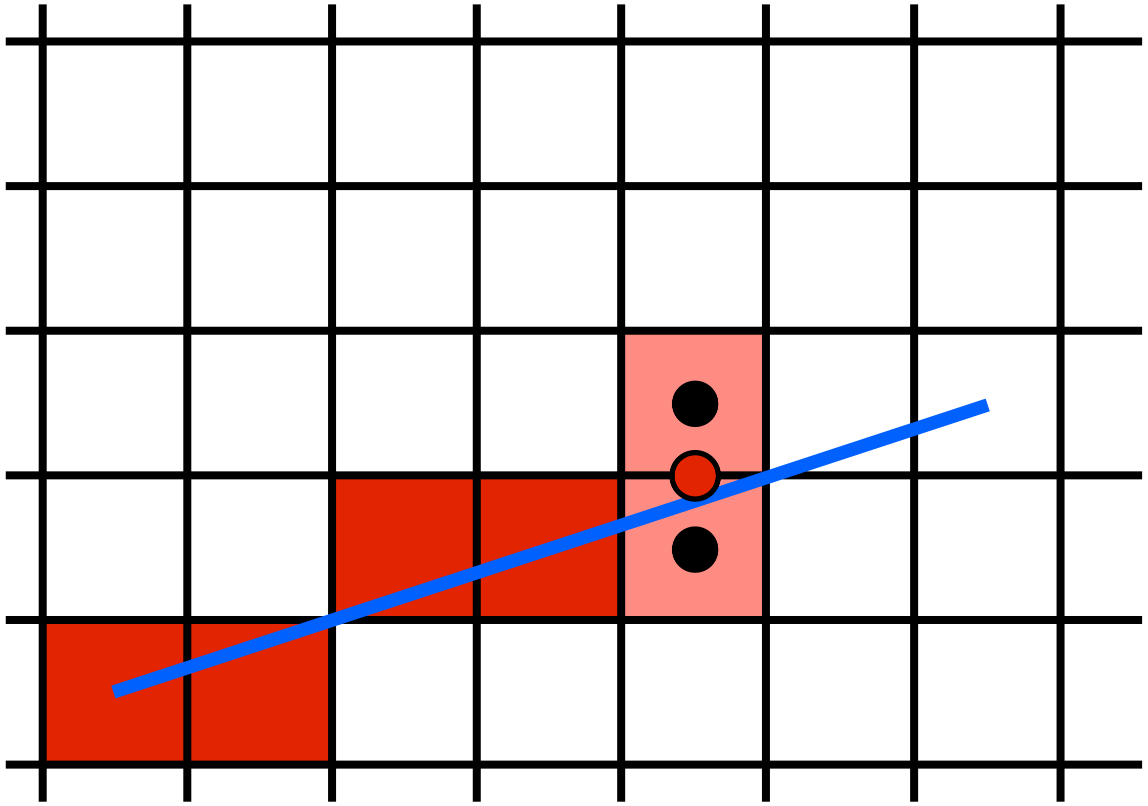
```
y = y0
for x = x0 to x1 do
    draw(x,y)
    if (f(x+1, y+1/2) < 0) then
        y = y+1
```



We can make the Midpoint Algorithm more efficient

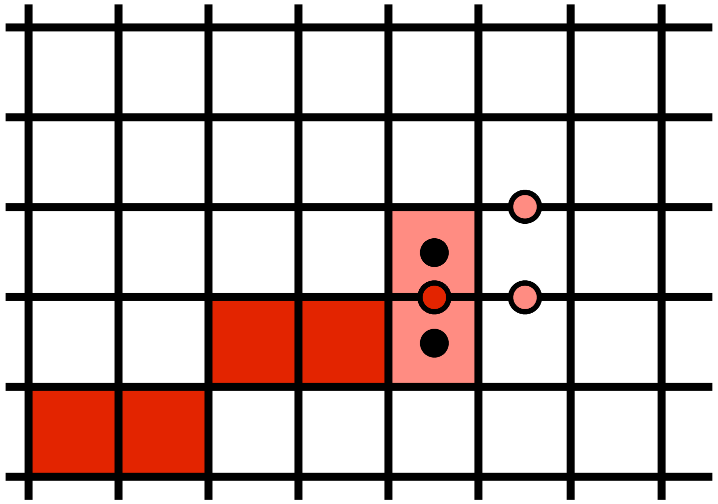
```
y = y0
for x = x0 to x1 do
    draw(x,y)
    if (f(x+1, y+1/2) < 0) then
        y = y+1
```

$$\text{if } \left(f(x+1, y+\tfrac{1}{2}) < 0 \right) \text{ then}$$



We can make the Midpoint Algorithm more efficient

by making it incremental!



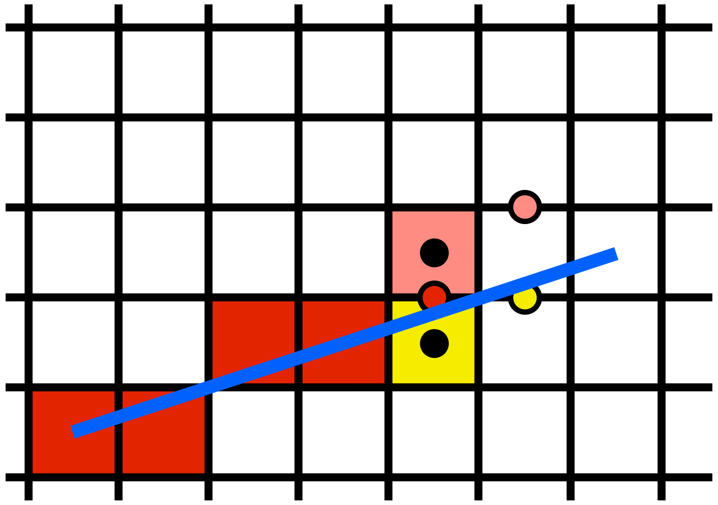
$$f(x,y) = (y_0 - y_1)x + (x_1 - x_0)y + x_0 y_1 - x_1 y_0 = 0$$

$$f(x+1, y) = f(x,y) + (y_0 - y_1)$$

$$f(x+1, y+1) = f(x,y) + (y_0 - y_1) + (x_1 - x_0)$$

We can make the Midpoint Algorithm more efficient

$$f(x+1, y+\frac{1}{2}) > 0$$



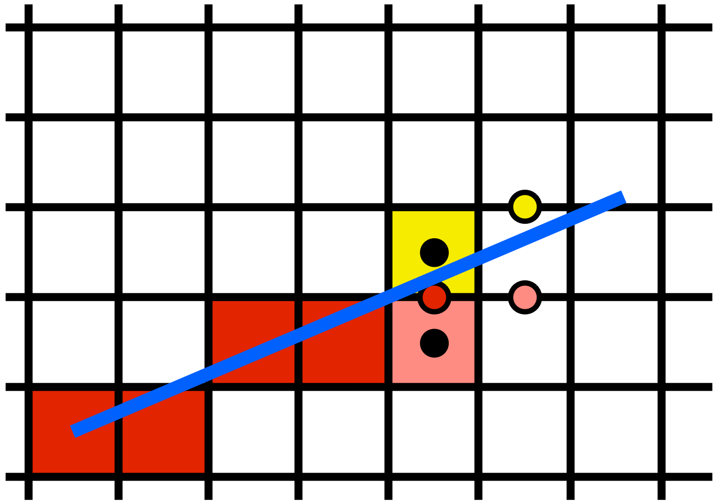
$$f(x,y) = (y_0 - y_1)x + (x_1 - x_0)y + x_0 y_1 - x_1 y_0 = 0$$

$$f(x+1, y) = f(x,y) + (y_0 - y_1)$$

$$f(x+1, y+1) = f(x,y) + (y_0 - y_1) + (x_1 - x_0)$$

We can make the Midpoint Algorithm more efficient

$$f(x+1, y+\frac{1}{2}) < 0$$



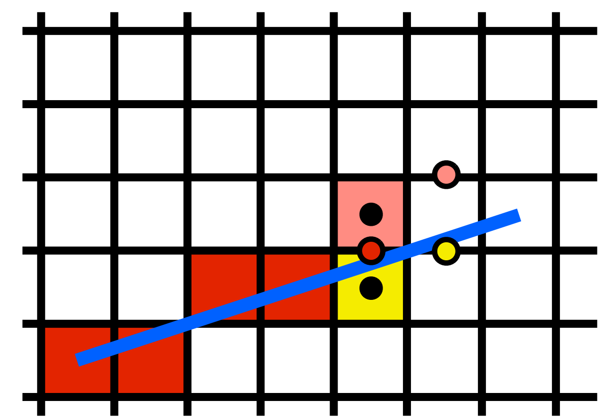
$$f(x,y) = (y_0 - y_1)x + (x_1 - x_0)y + x_0 y_1 - x_1 y_0 = 0$$

$$f(x+1, y) = f(x,y) + (y_0 - y_1)$$

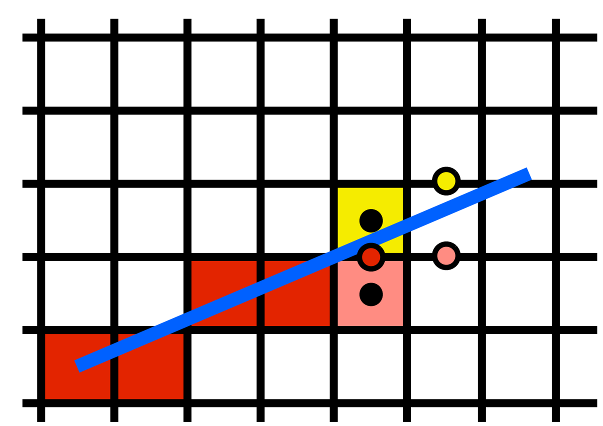
$$f(x+1, y+1) = f(x,y) + (y_0 - y_1) + (x_1 - x_0)$$

We can make the Midpoint Algorithm more efficient

```
y = y0
d = f(x0+1,y0+1/2)
for x = x0 to x1 do
    draw(x,y)
    if (d<0) then
        y = y+1
        d = d+(y0-y1)+(x1-x0)
    else
        d = d+(y0-y1)
```

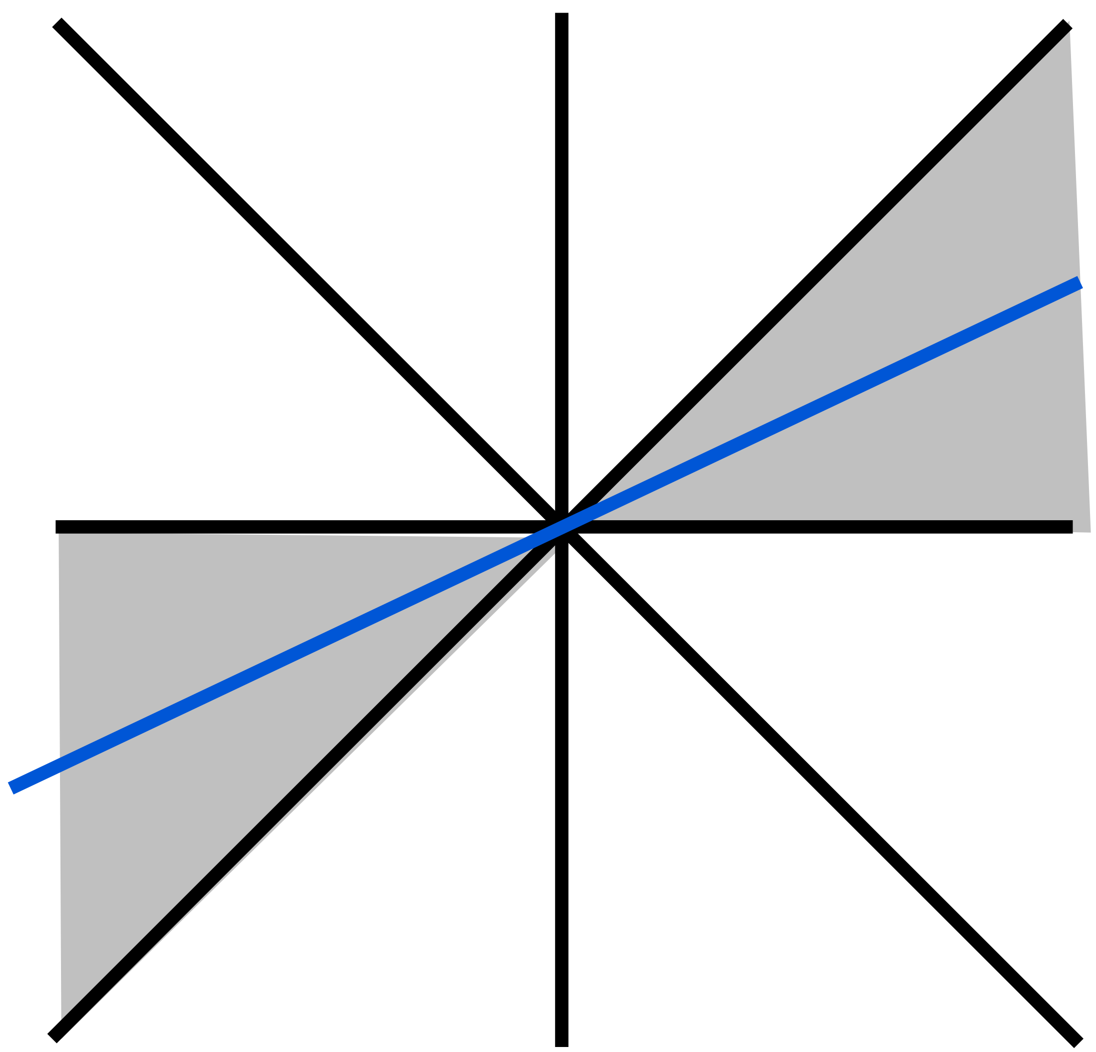


$$f(x+1,y) = f(x,y) + (y_0 - y_1)$$

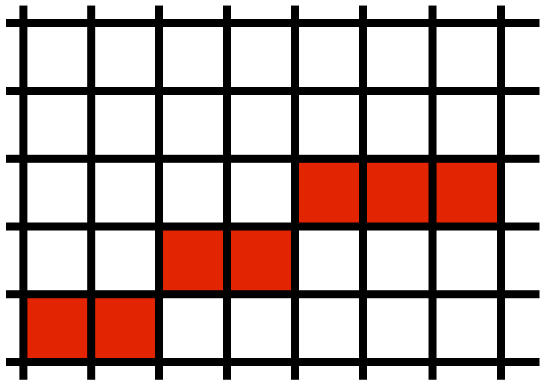


$$f(x+1,y+1) = f(x,y) + (y_0 - y_1) + (x_1 - x_0)$$

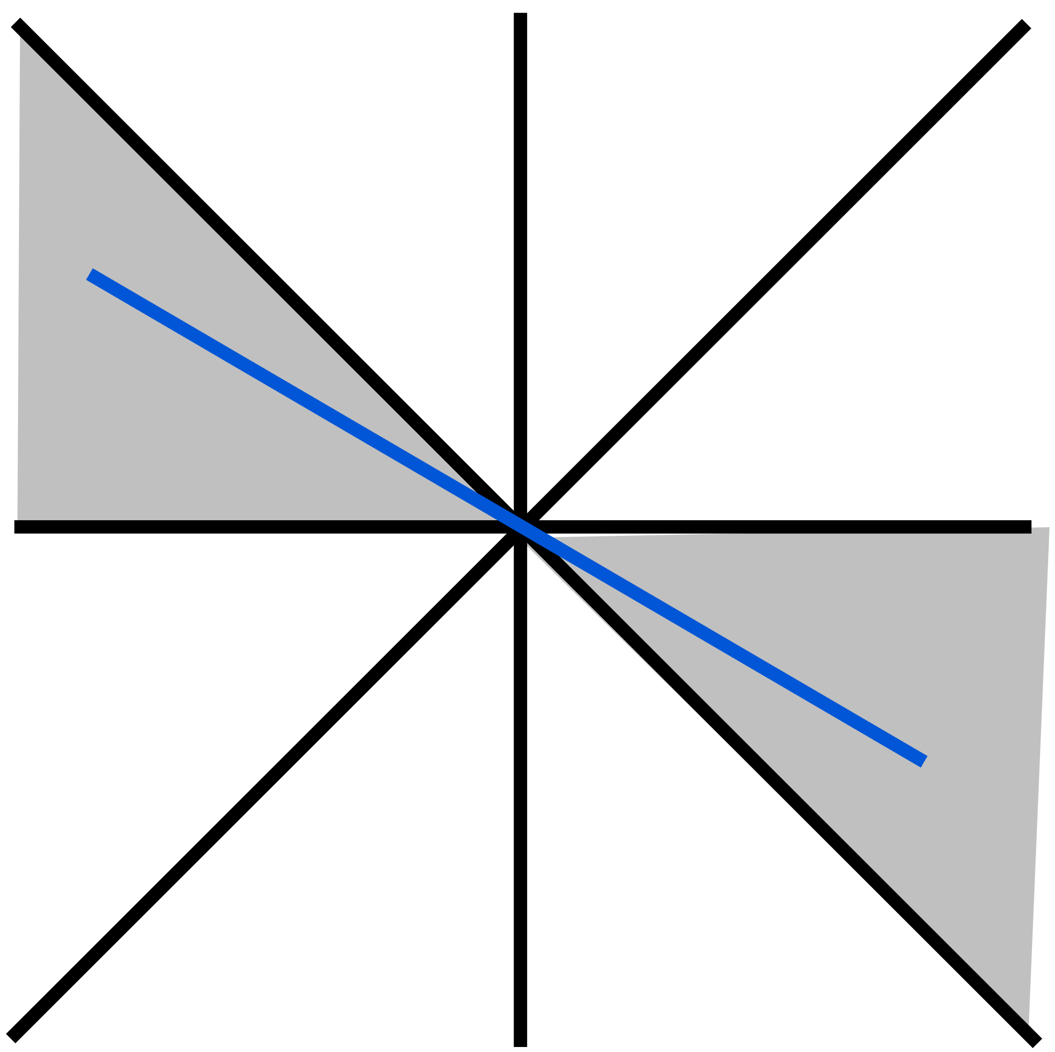
Adapt Midpoint Algorithm for other cases



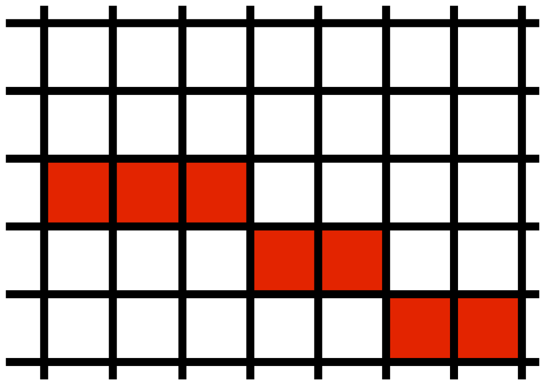
case: $0 < m <= 1$



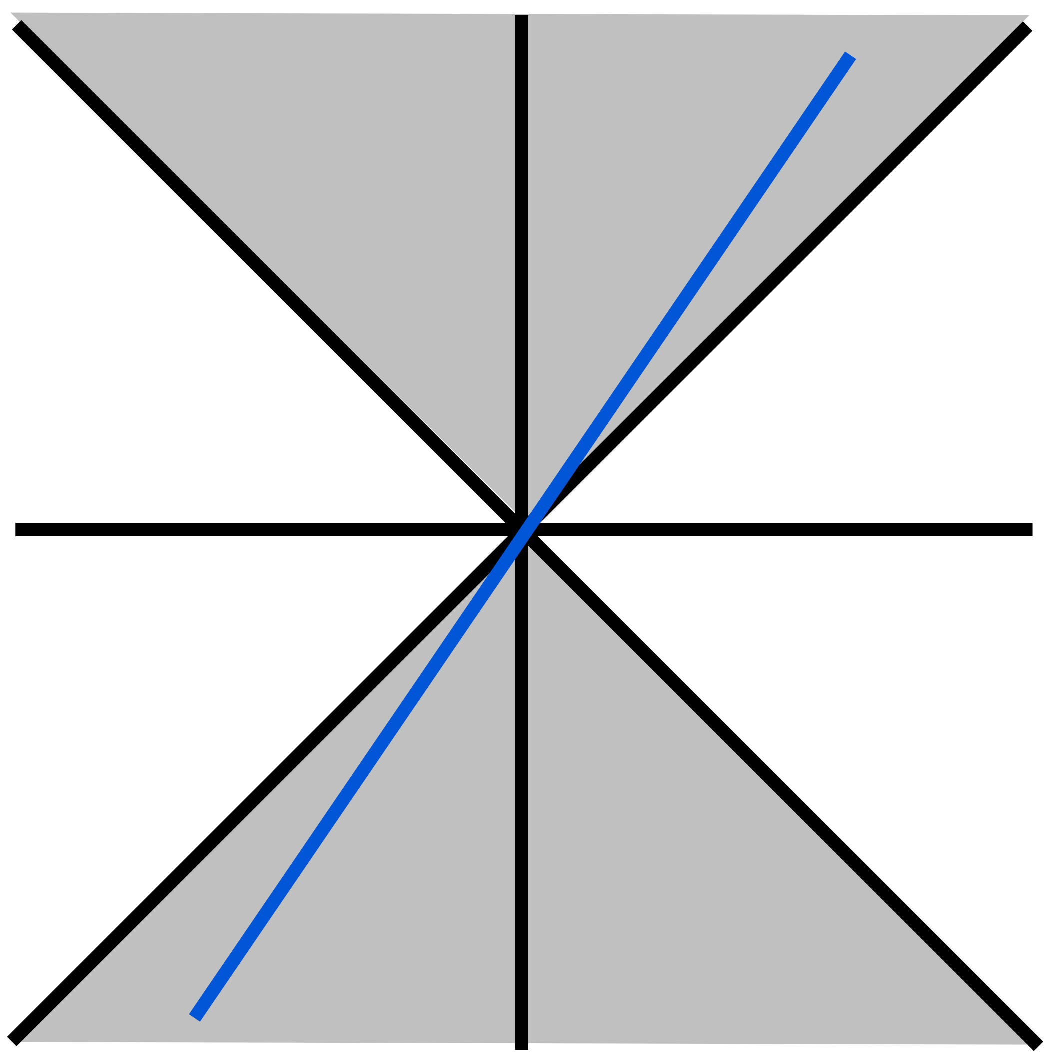
Adapt Midpoint Algorithm for other cases



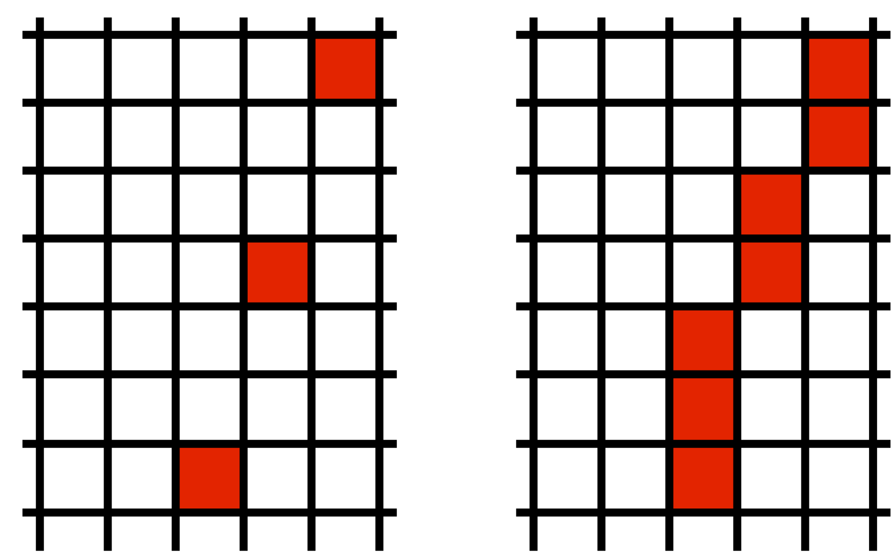
case: -1 <= m < 0



Adapt Midpoint Algorithm for other cases



case: 1 <= m
or m <= -1

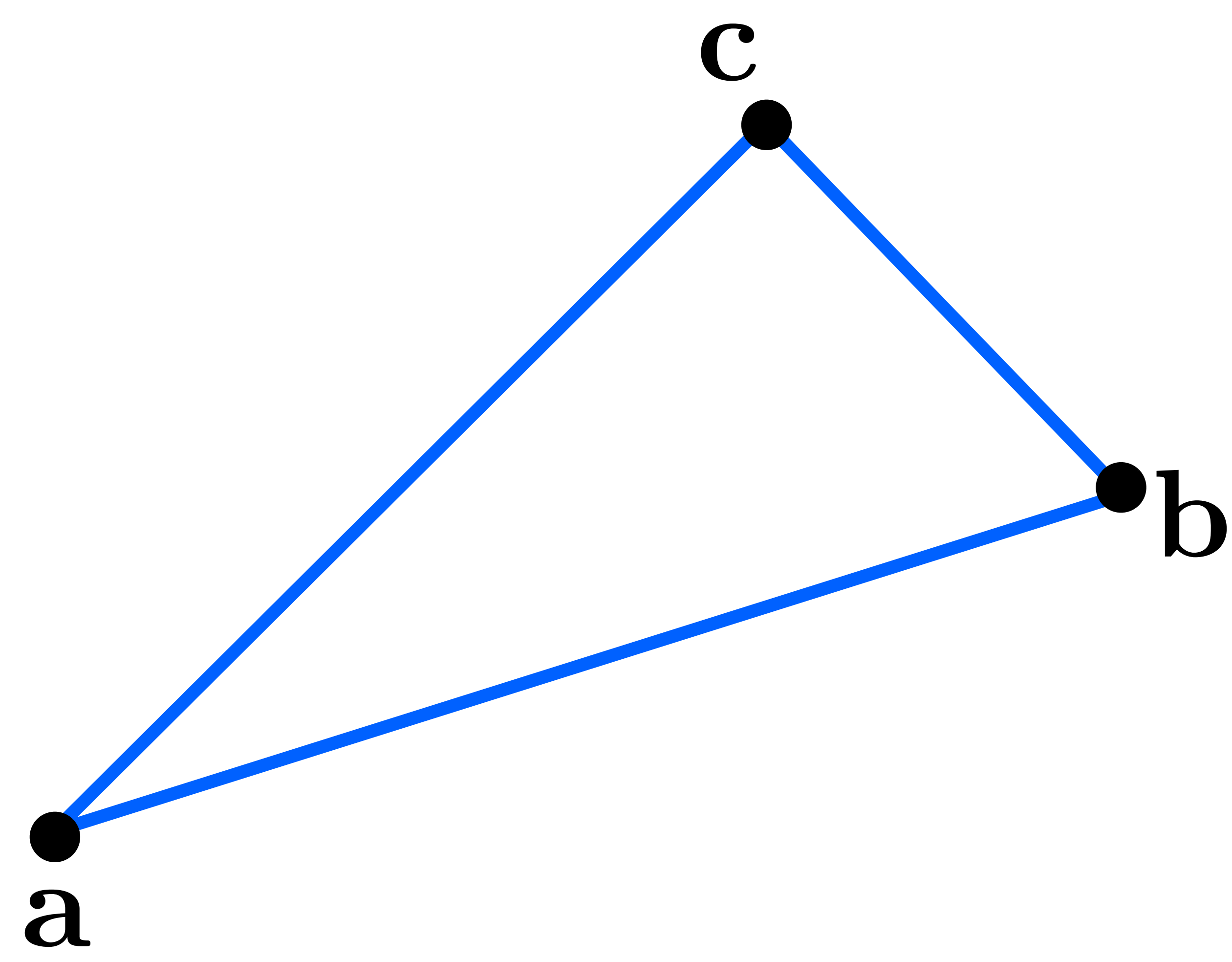


Line drawing references

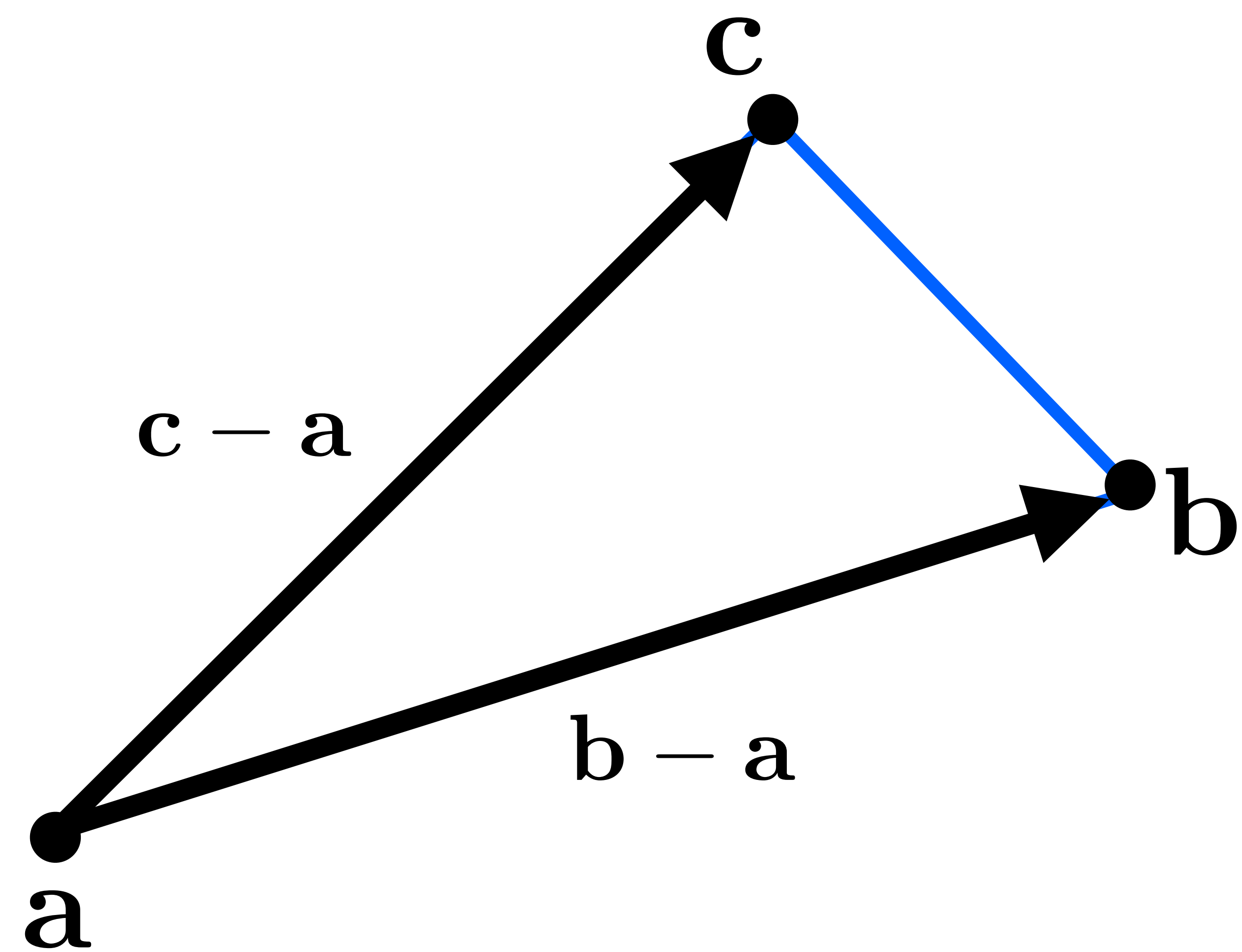
- the algorithm we just described is the *Midpoint Algorithm* (Pitteway, 1967), (van Aken and Novak, 1985)
- draws the same lines as the *Bresenham Line Algorithm* (Bresenham, 1965)

Triangles

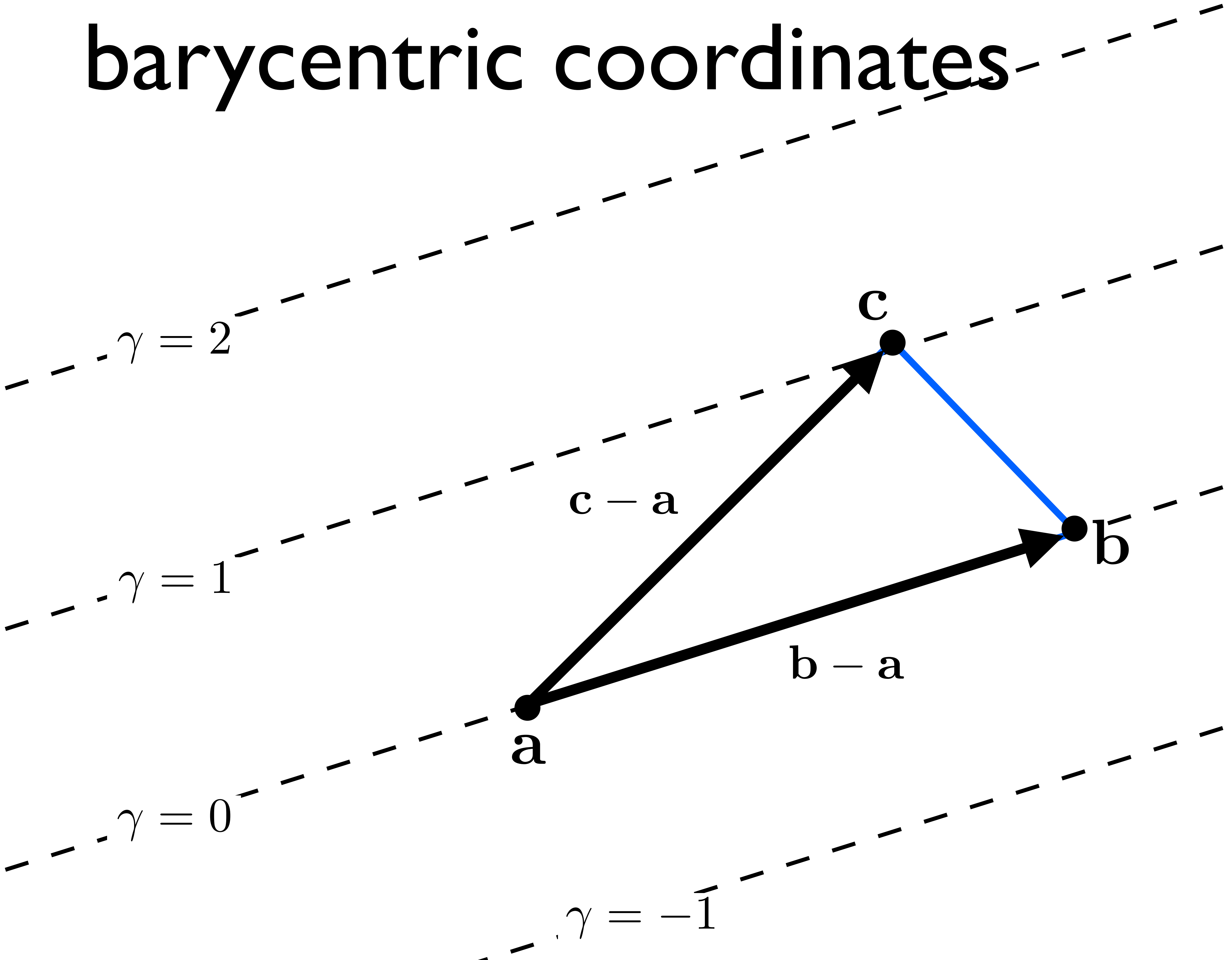
barycentric coordinates



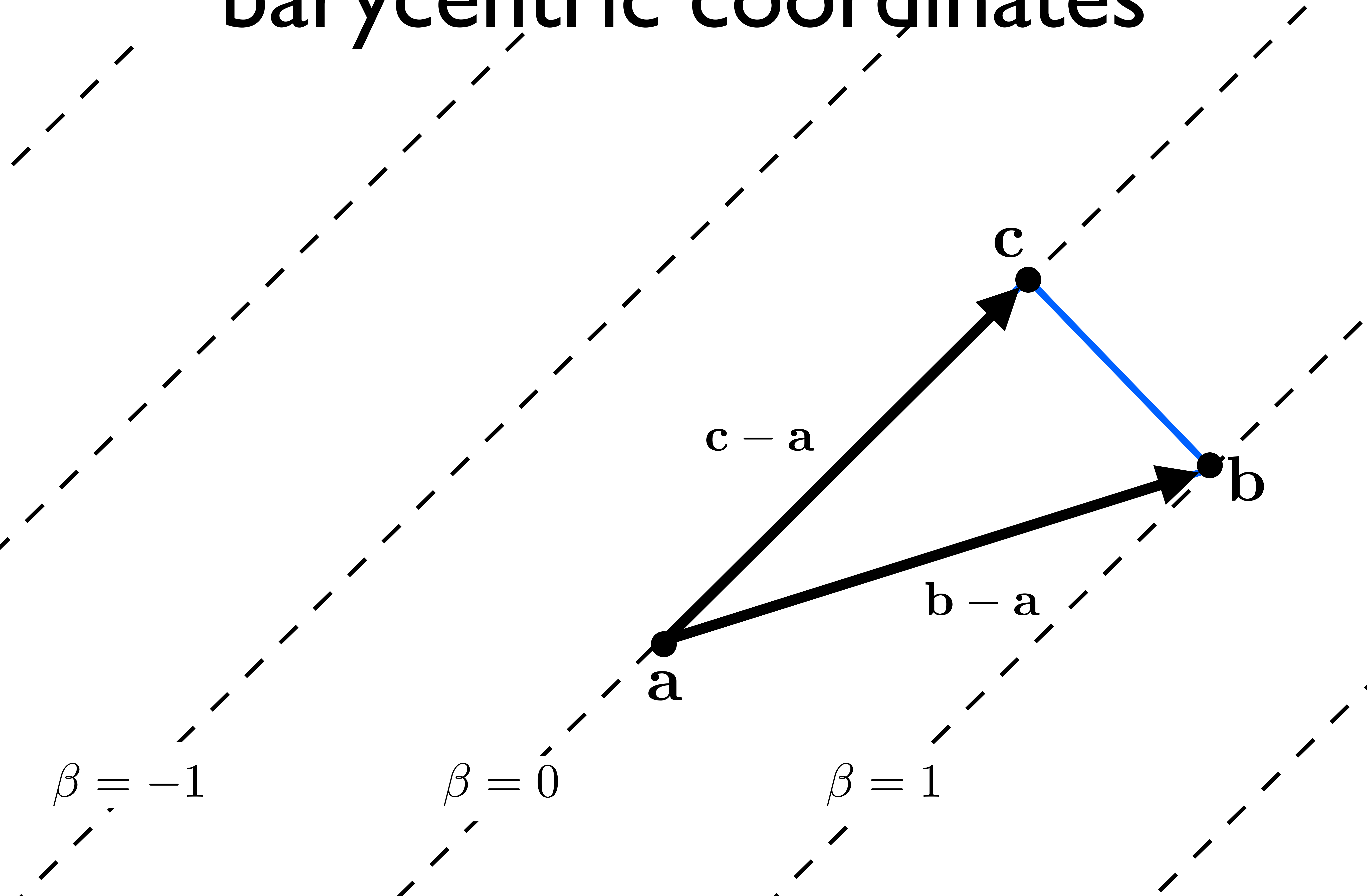
barycentric coordinates



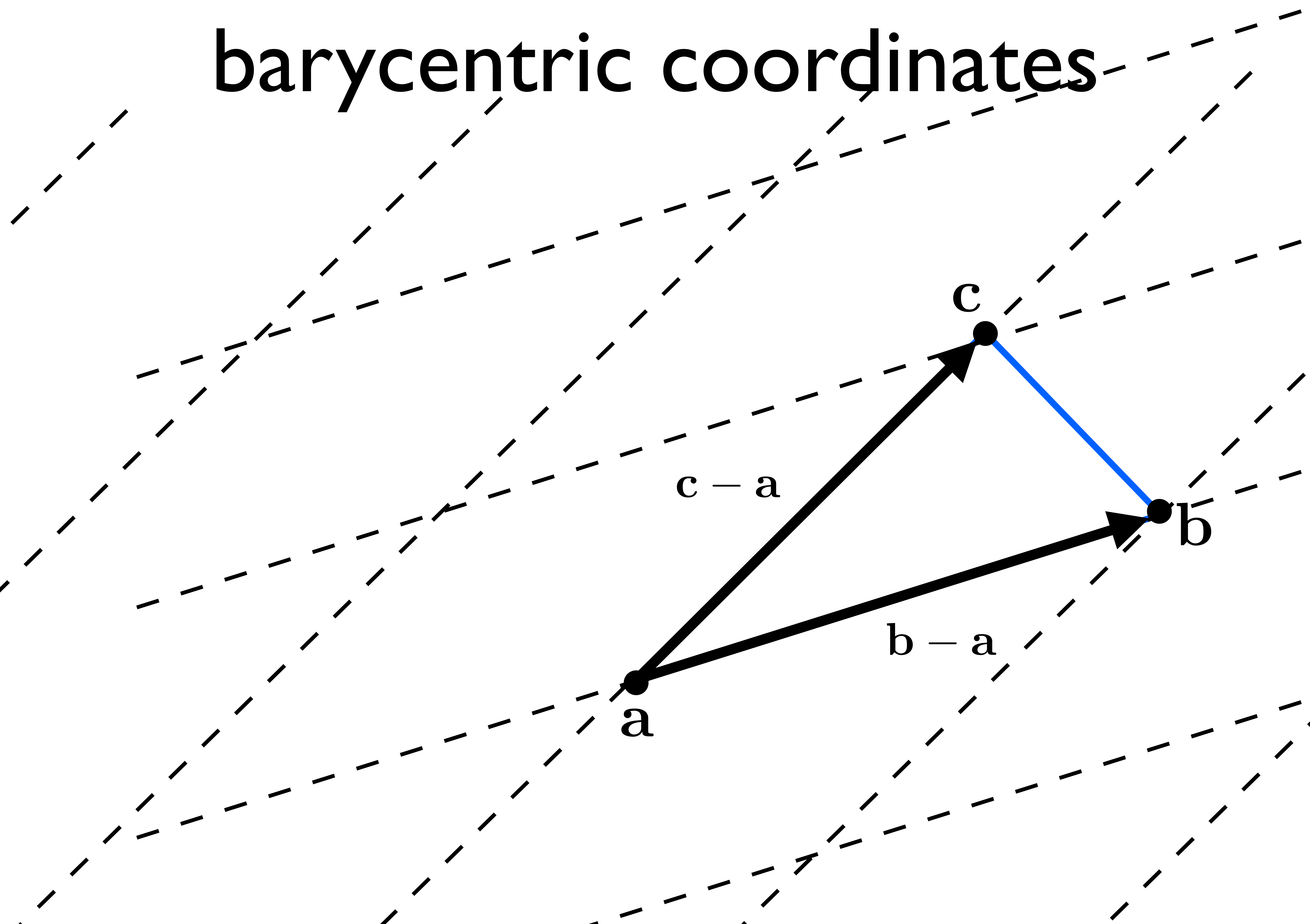
barycentric coordinates



barycentric coordinates



barycentric coordinates



barycentric coordinates

$$\mathbf{p} = \alpha \mathbf{a} + \beta \mathbf{b} + \gamma \mathbf{c}$$

What are (α, β, γ) ?

<whiteboard>