CS230 : Computer Graphics

Lecture 3: Rasterization

Tamar Shinar
Computer Science & Engineering
UC Riverside

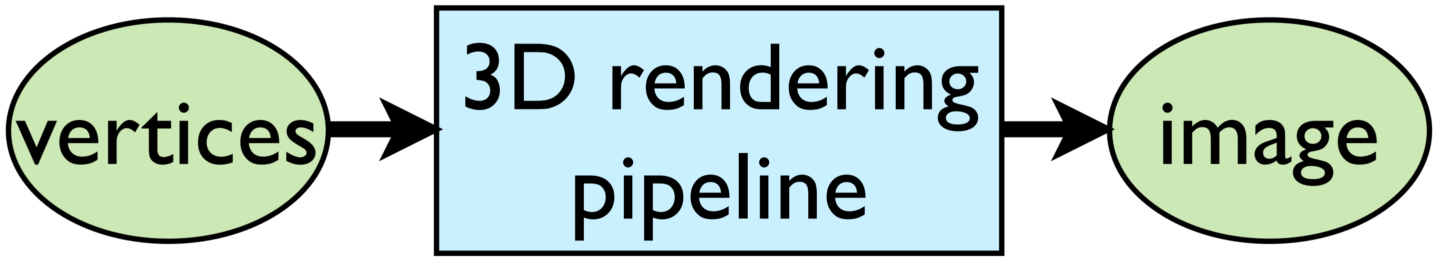
Rendering approaches

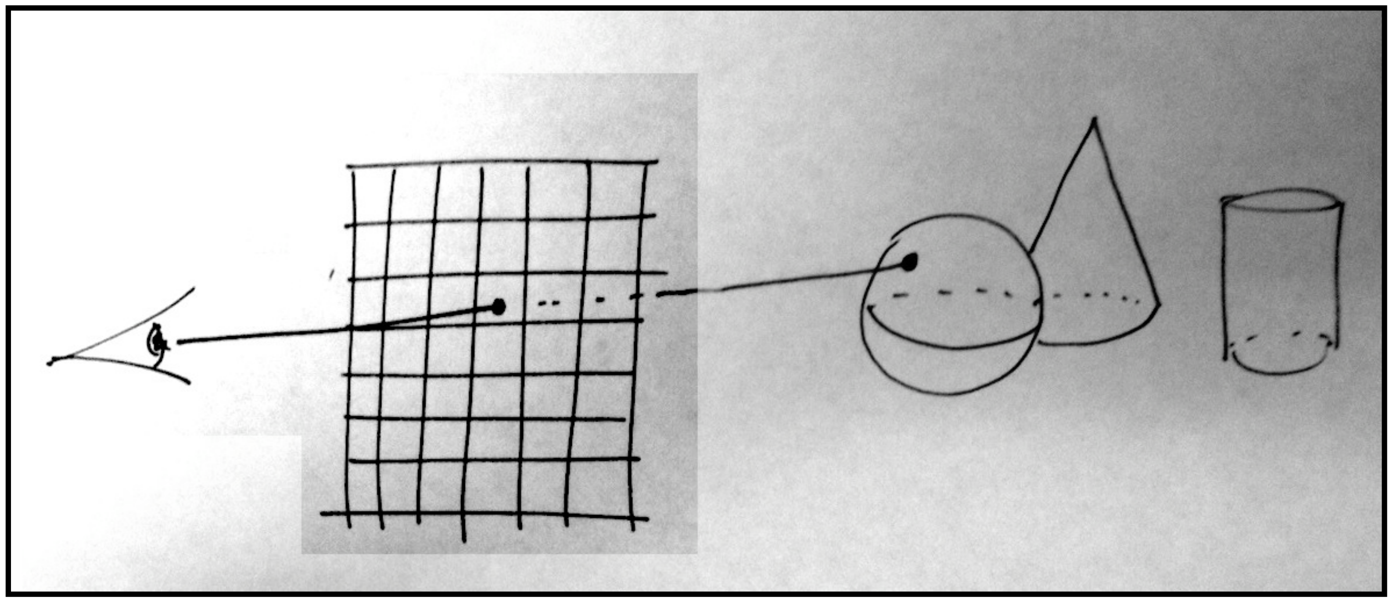
1. **object-oriented**

   foreach object ...

2. **image-oriented**

   foreach pixel ...





there's more than one way to do **object-oriented rendering** – e.g., OpenGL graphics pipeline vs. Renderman

Outline

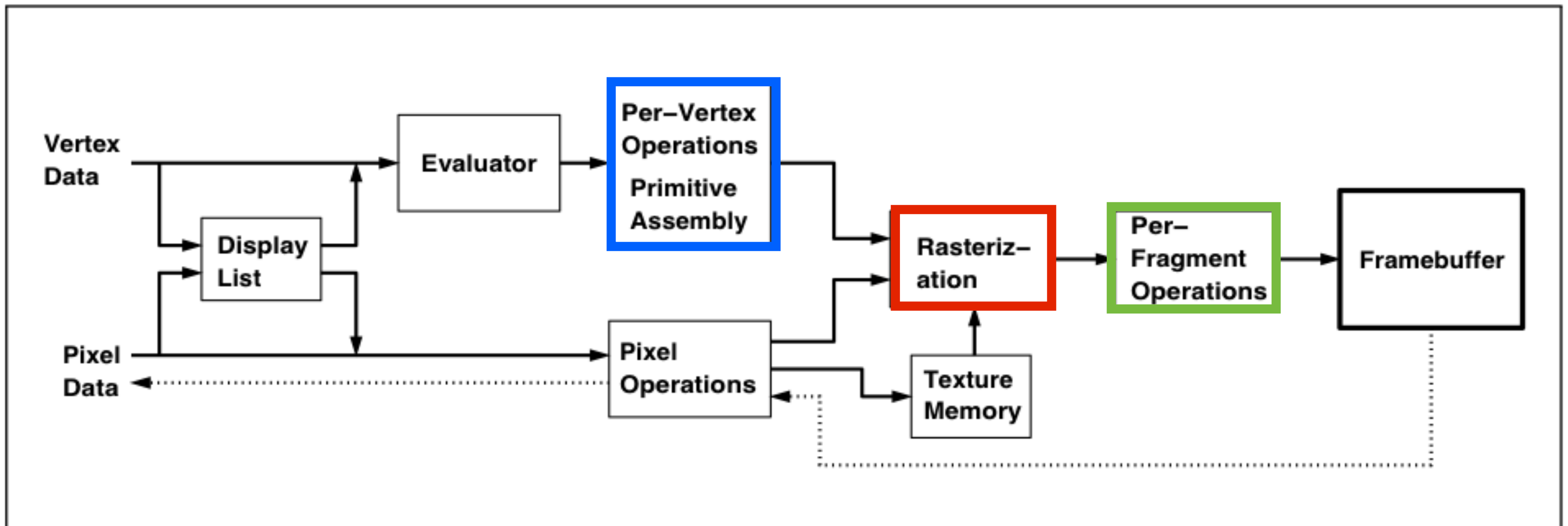


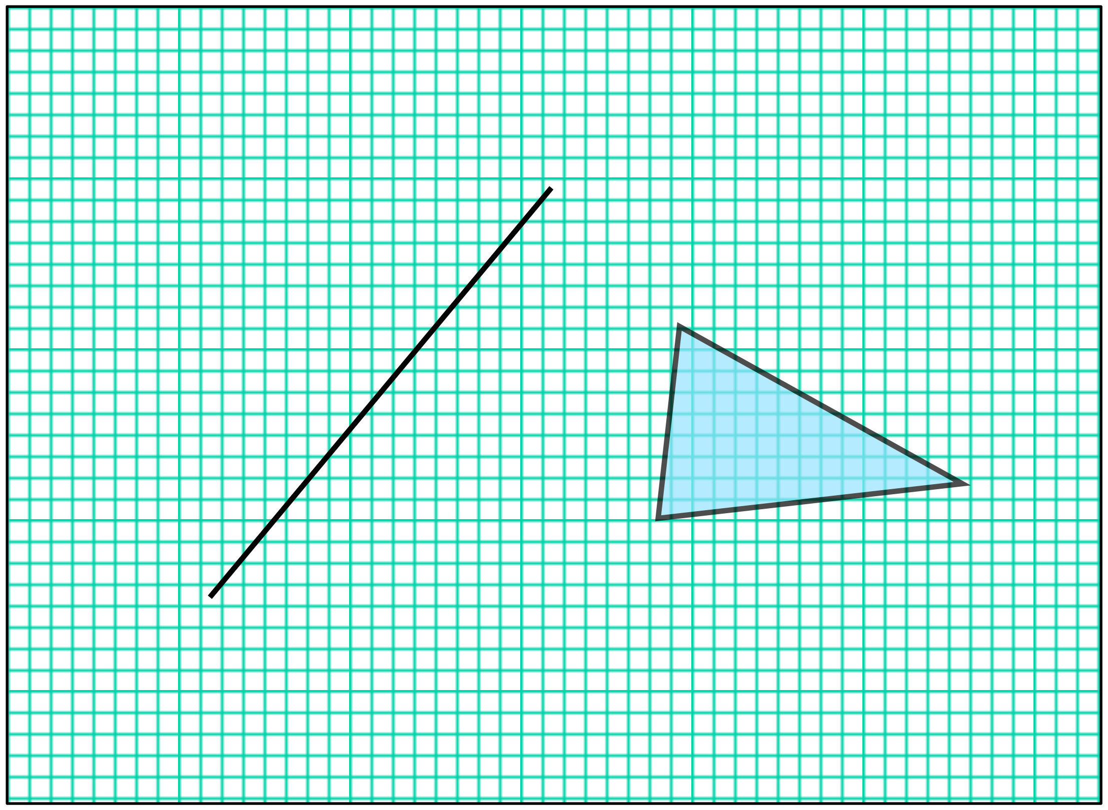
Figure 1. Block diagram of OpenGL.

rasterization - make fragments from clipped objects

clipping - clip objects to viewing volume

hidden surface removal - determine visible fragments

What is rasterization?



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

What is rasterization?

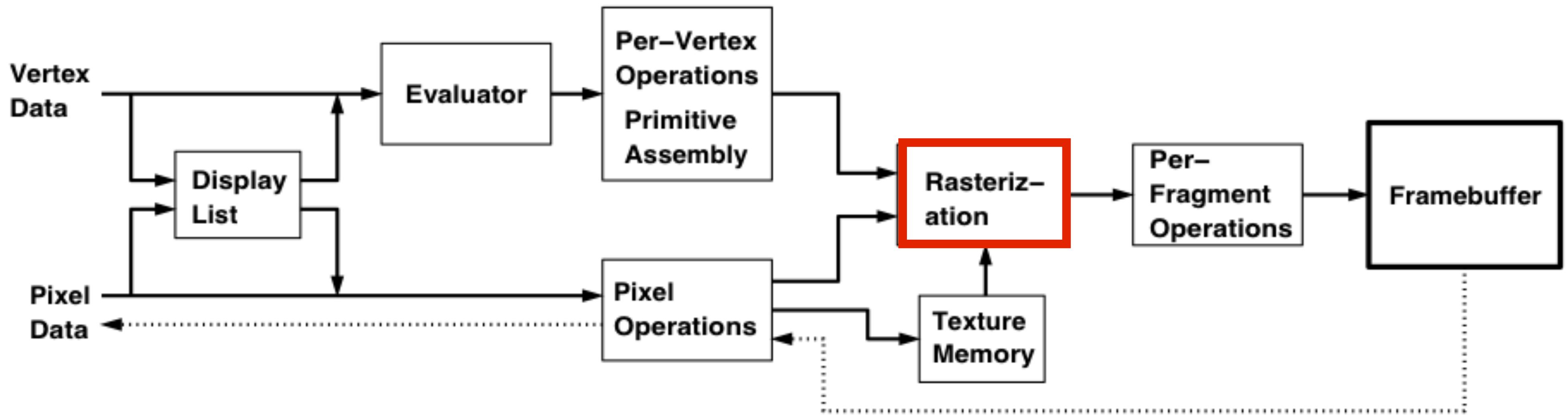


Figure 1. Block diagram of OpenGL.

- input: primitives, output: fragments
- enumerate the pixels covered by a primitive
- interpolate attributes across the primitive

– **output** 1 fragment per pixel covered by the primitive

Rasterization

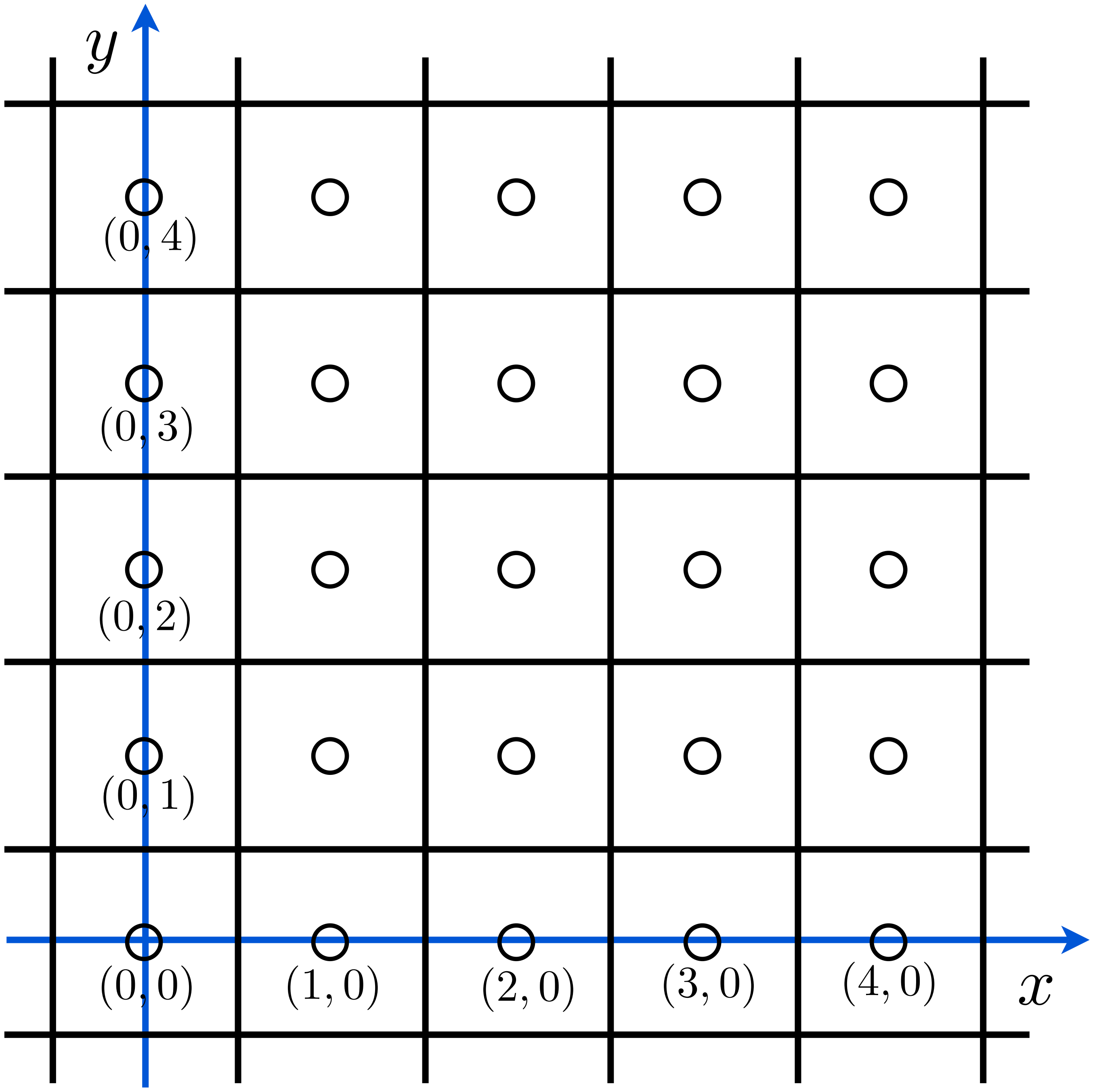
Compute integer coordinates for pixels near the 2D primitives

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

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

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

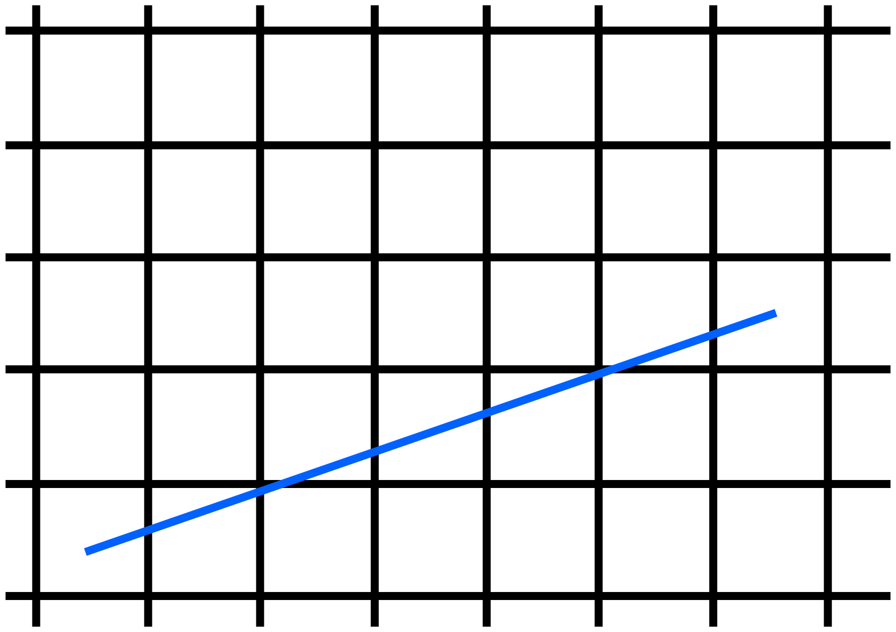
Screen coordinates



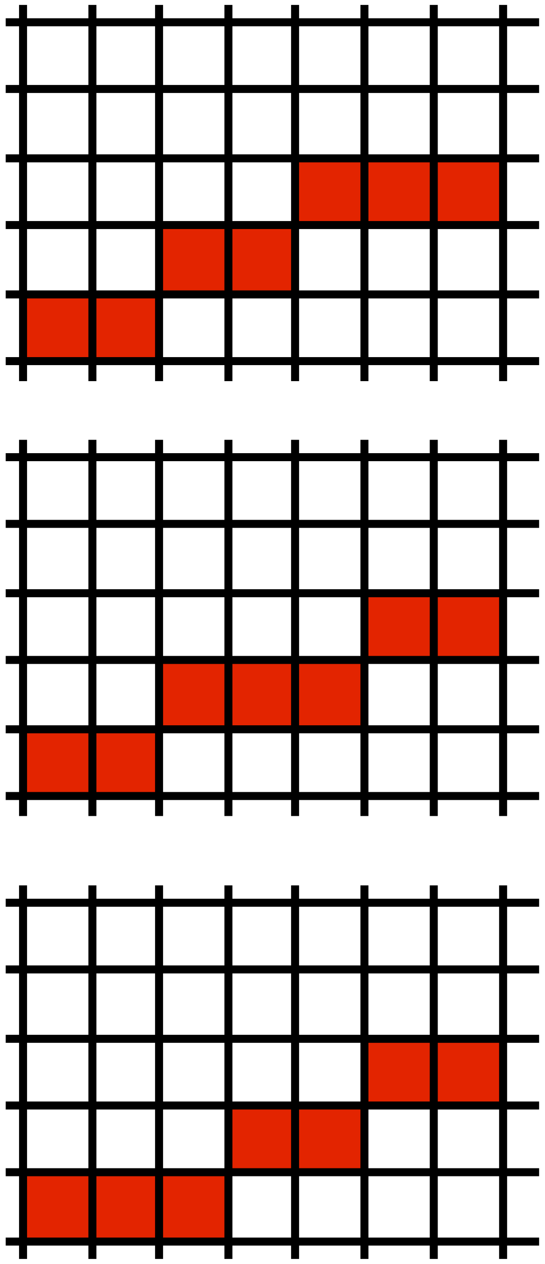
we’ll assume stuff has been converted to **normalized device coordinates**

Line drawing

Which pixels should be used to approximate a line?



Draw the thinnest possible line that has no gaps

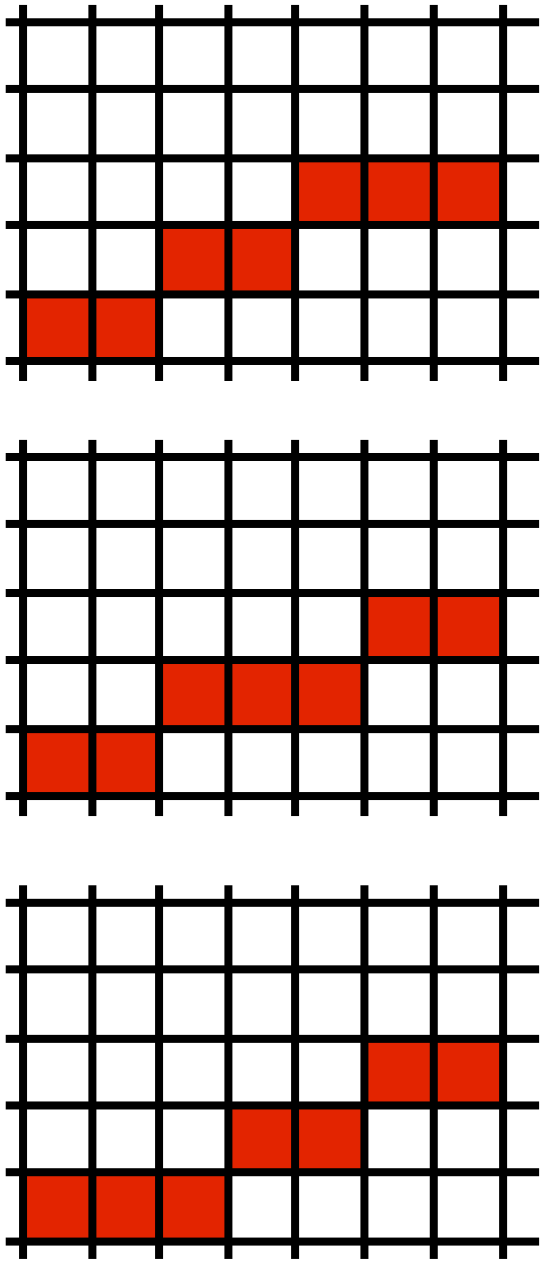


Line drawing algorithm

(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)



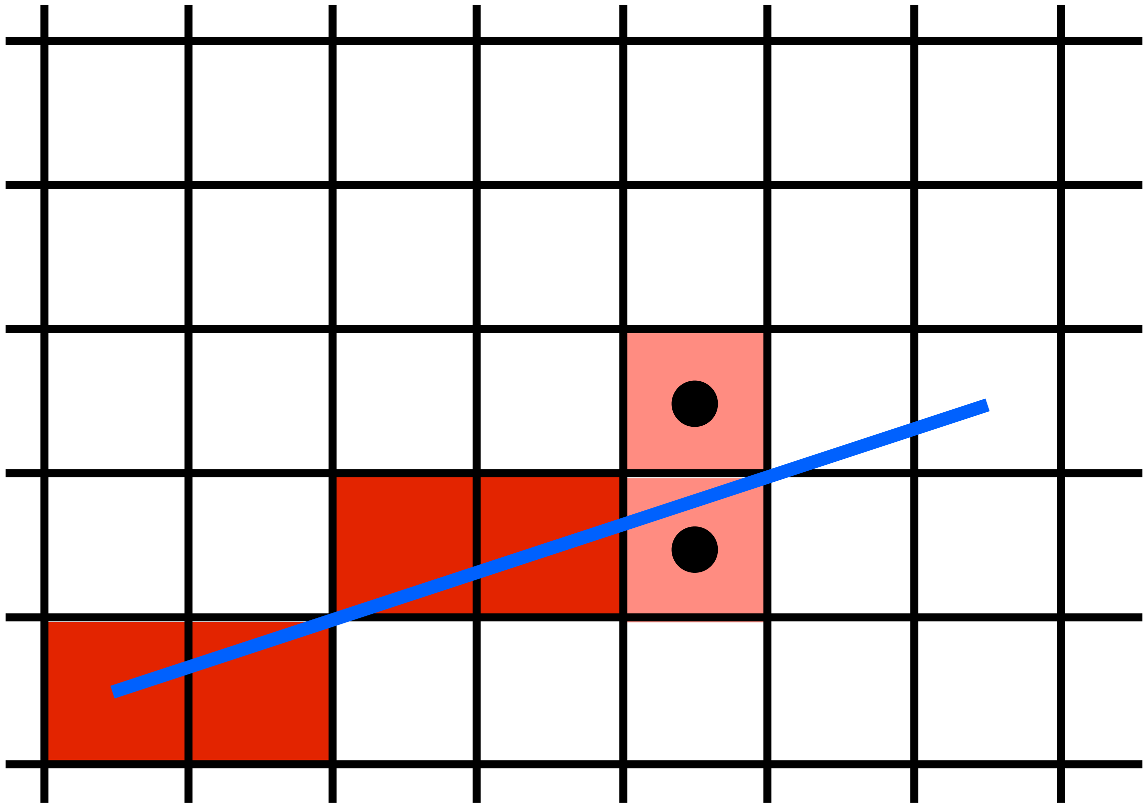
draw pixels from left to right, occasionally move up

Line drawing algorithm

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

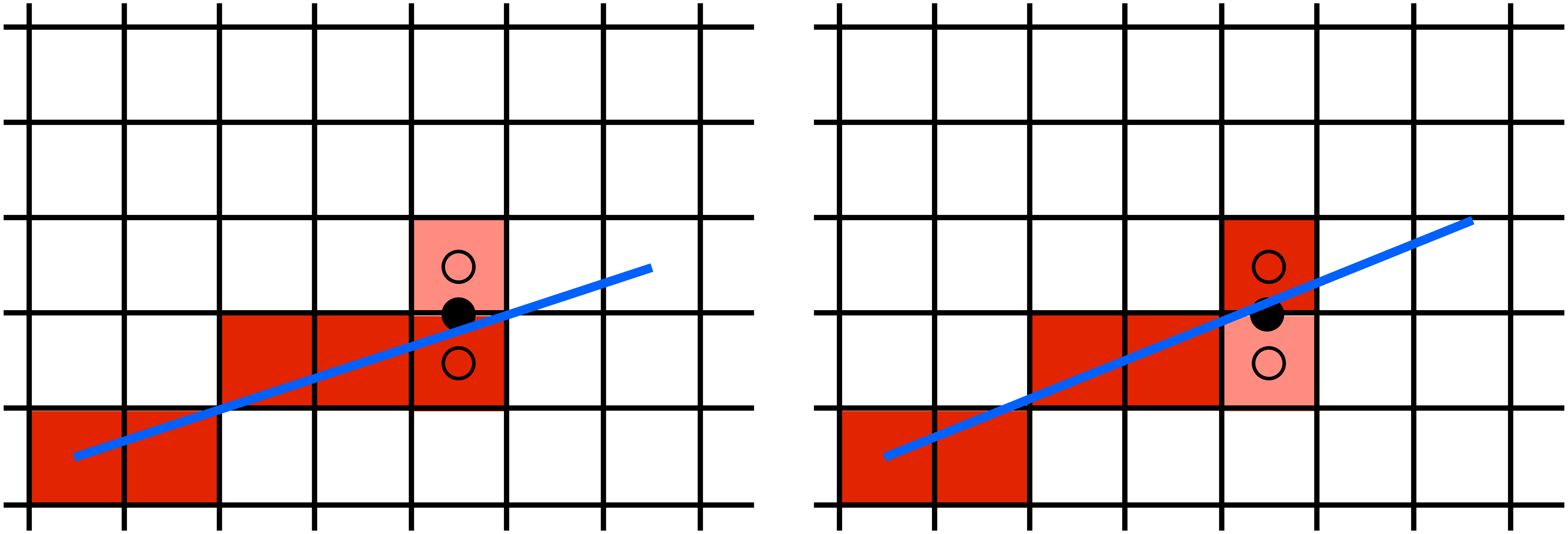
```
y = y0
for x = x0 to x1 do
    draw(x,y)
    if (<condition>) then
        y = y+1
```

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



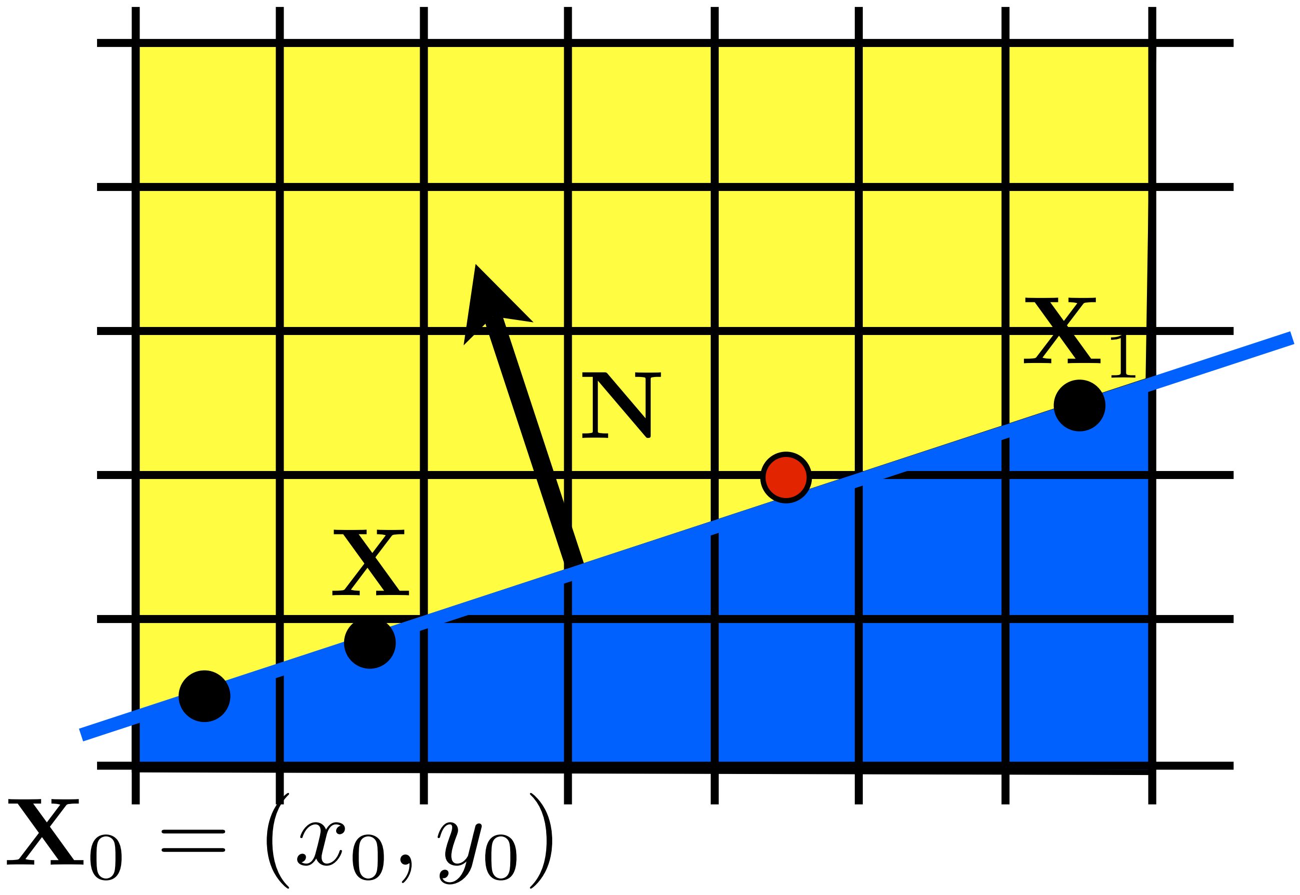
draw pixels from left to right, occasionally move up

Use the midpoint between the two pixels to choose



If the line falls **below** the midpoint, use the bottom pixel
if the line falls **above** the midpoint, use the top pixel

Use the midpoint between the two pixels to choose



implicit line equation:

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

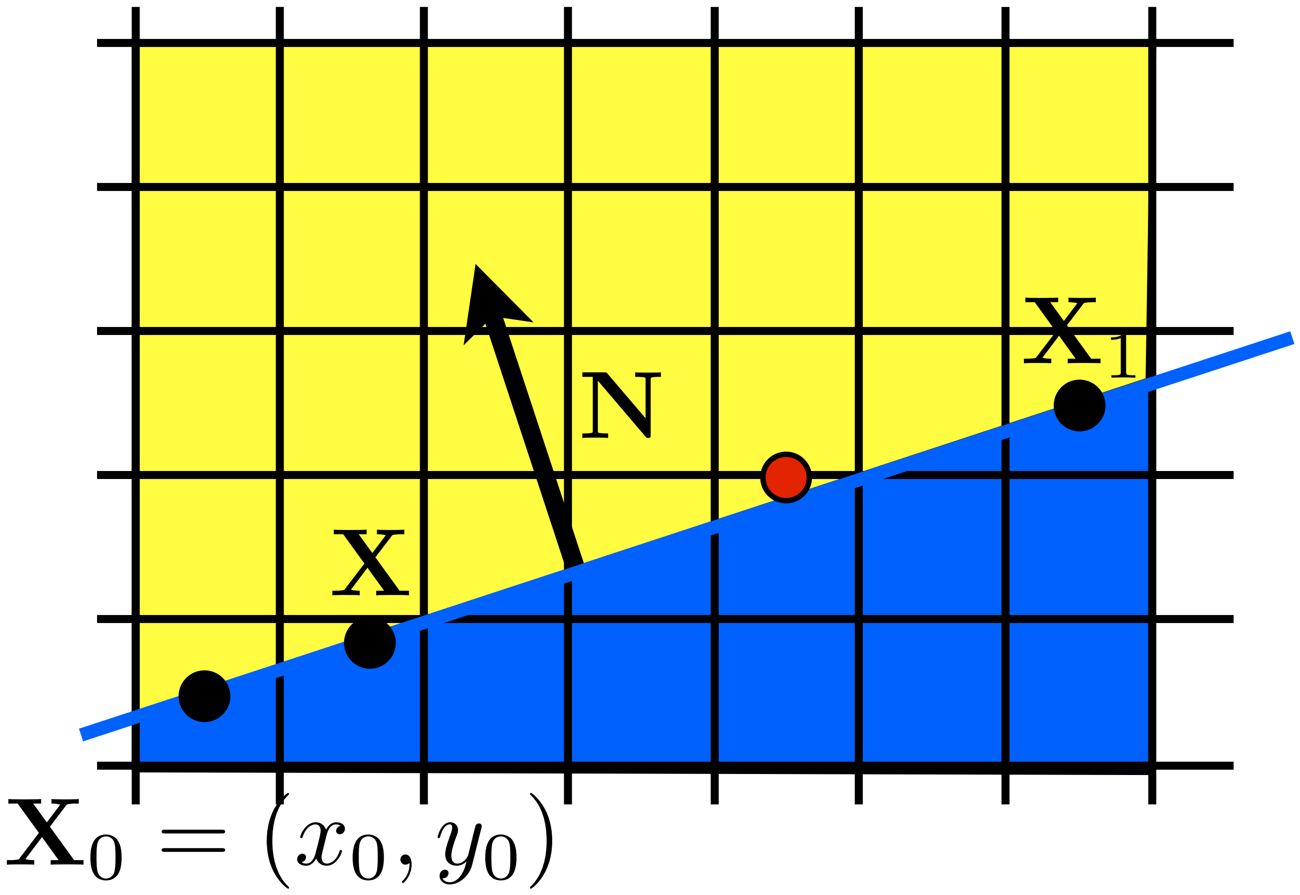
<whiteboard>

evaluate f at midpoint:

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

<whiteboard>: work out the implicit line equation in terms of X0 and X1
Question: will f(x,y+1/2) be > 0 or < 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:

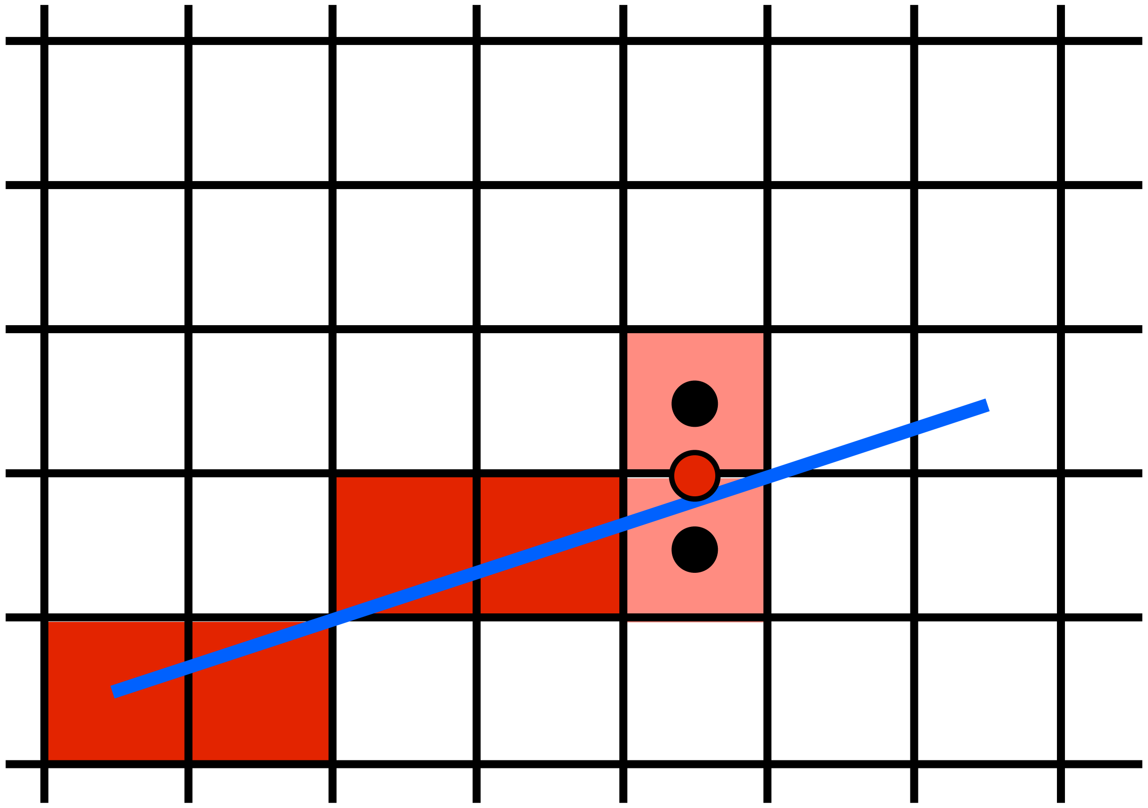
$$f(x, y + \frac{1}{2}) \ > \ 0$$

this means midpoint is above the line -> line is closer to bottom pixel

Line drawing algorithm

(case: 0 < m <= 1)

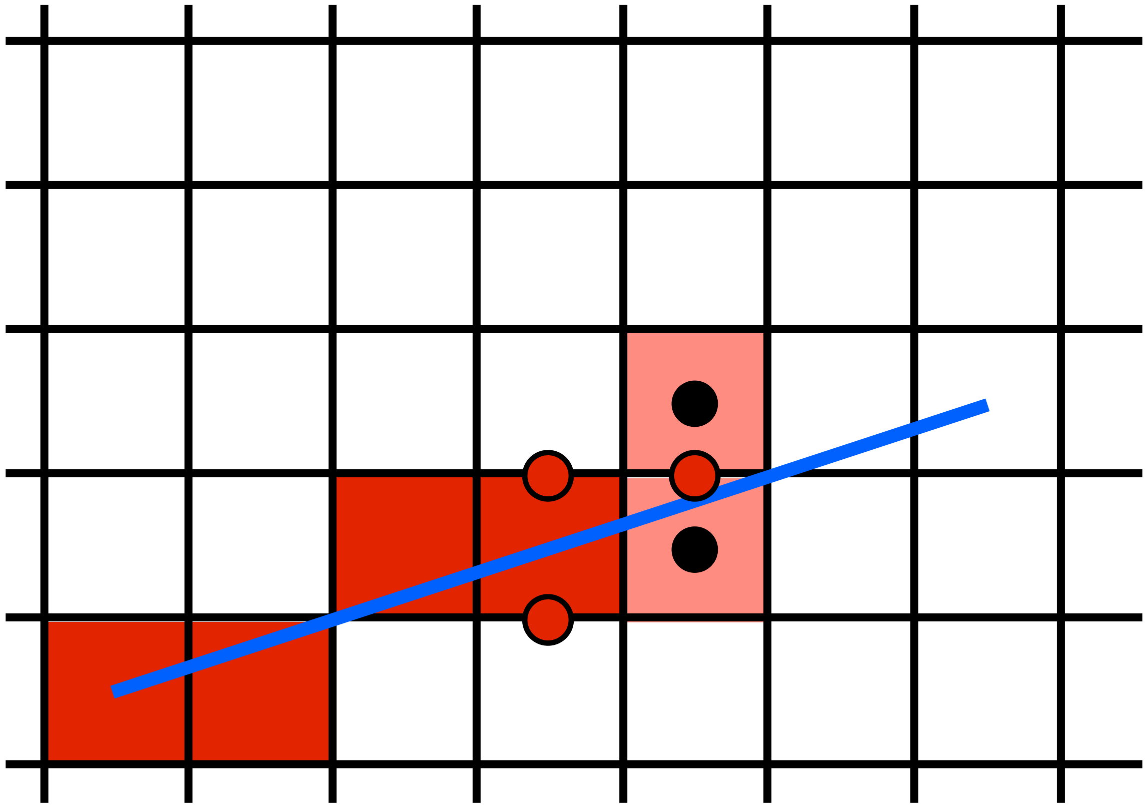
```
y = y0
for x = x0 to x1 do
    draw(x,y)
    if ($f(x+1, y+\frac{1}{2}) < 0$) then
        y = y+1
```



can now fill in the **condition**

We can make the Midpoint Algorithm more efficient

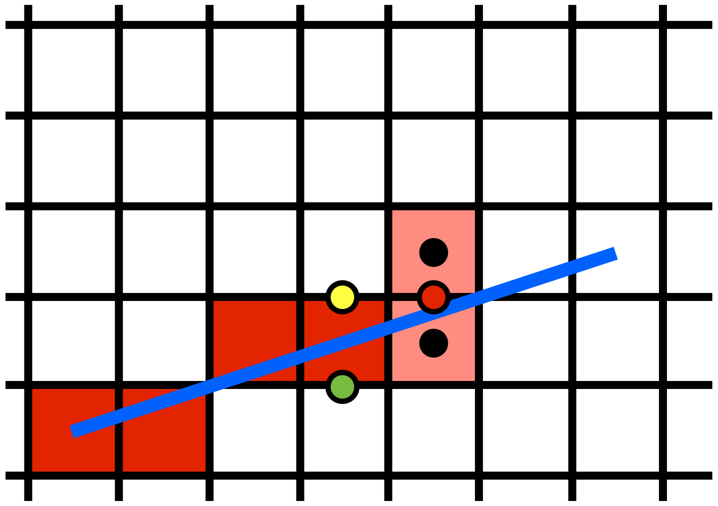
```
y = y0
for x = x0 to x1 do
    draw(x,y)
    if ($f(x+1, y+\frac{1}{2}) < 0$) then
        y = y+1
```



by making it **incremental**
in the last step, we computed **f(x,y+1/2)** or **f(x,y−1/2)**

We can make the Midpoint Algorithm more efficient

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



$$f(x,y) = (y_0 - y_1)x + (x_1 - x_0)y + x_0y_1 - x_1y_0 = 0$$

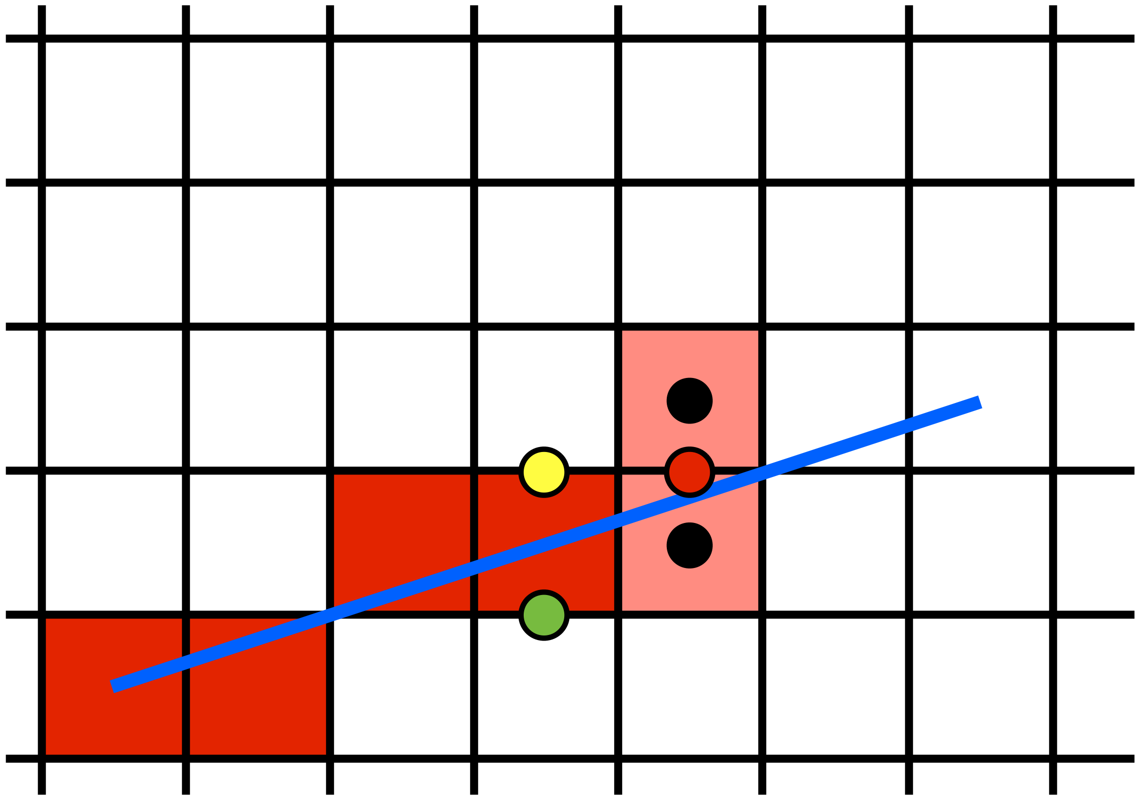
🟡 $f(x+1, y) = f(x,y) + (y_0 - y_1)$

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

we need to use one of these two update rules
which one?

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)
```

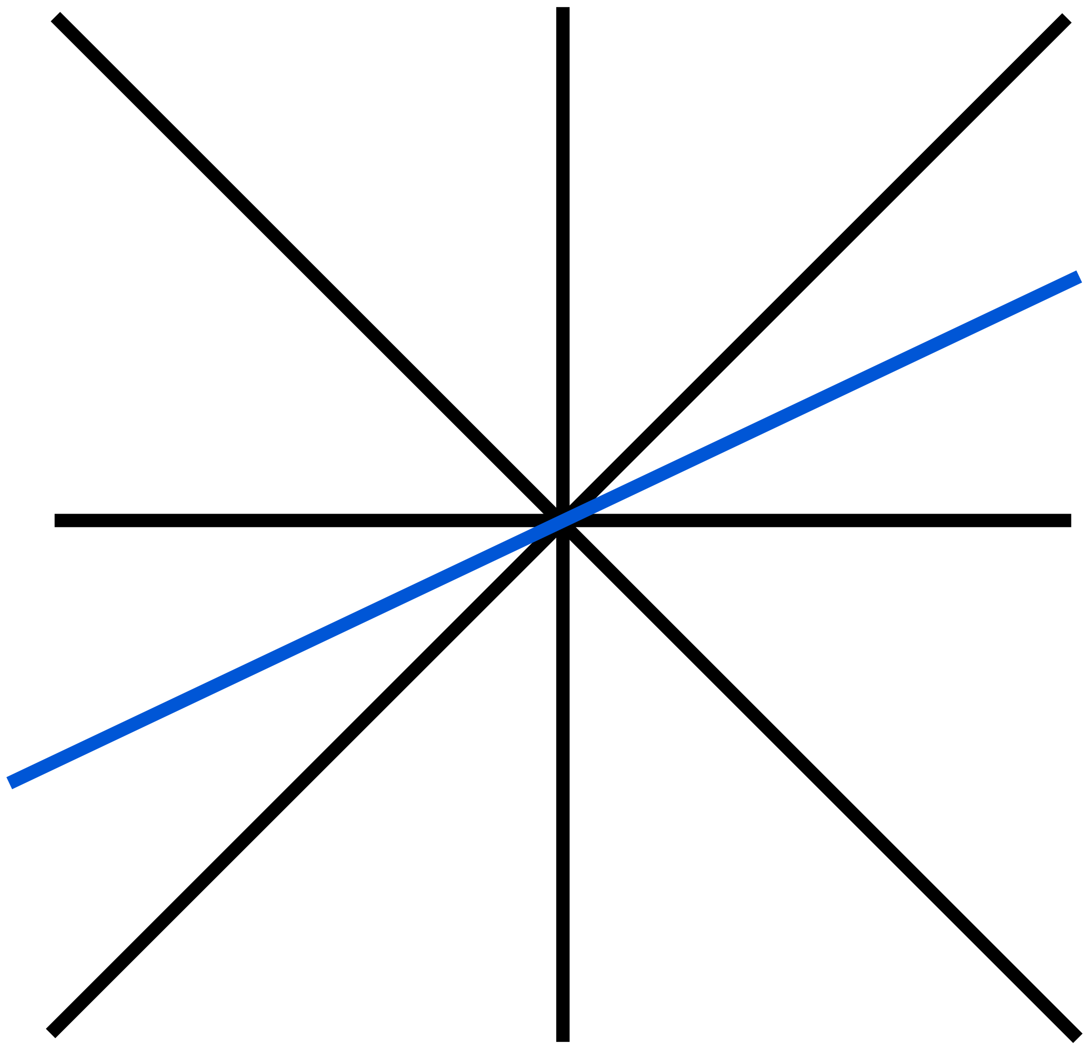


● $f(x+1,y) = f(x,y) + (y_0 - y_1)$

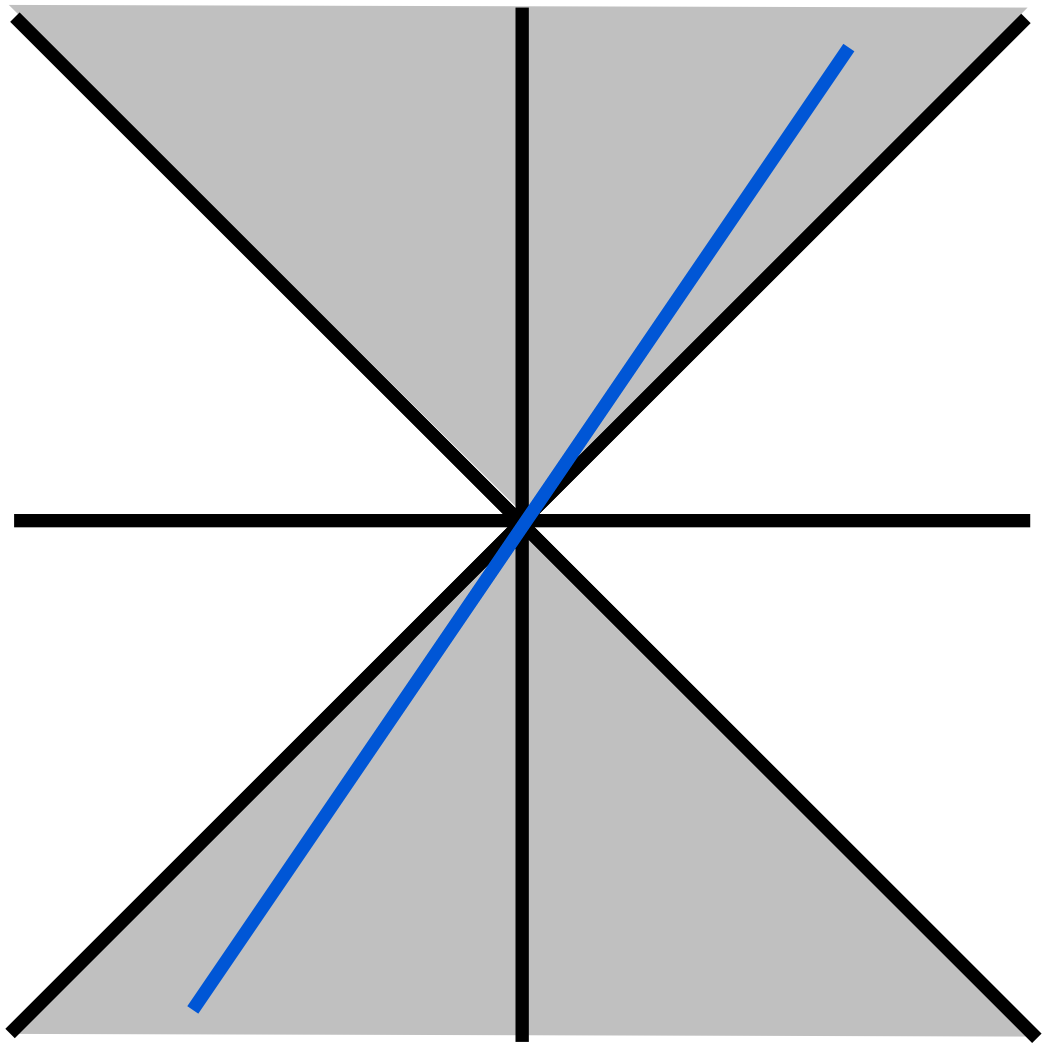
● $f(x+1,y+1) = f(x,y) + (y_0 - y_1) + (x_1 - x_0)$

algorithm is **incremental** and uses only **integer arithmetic**

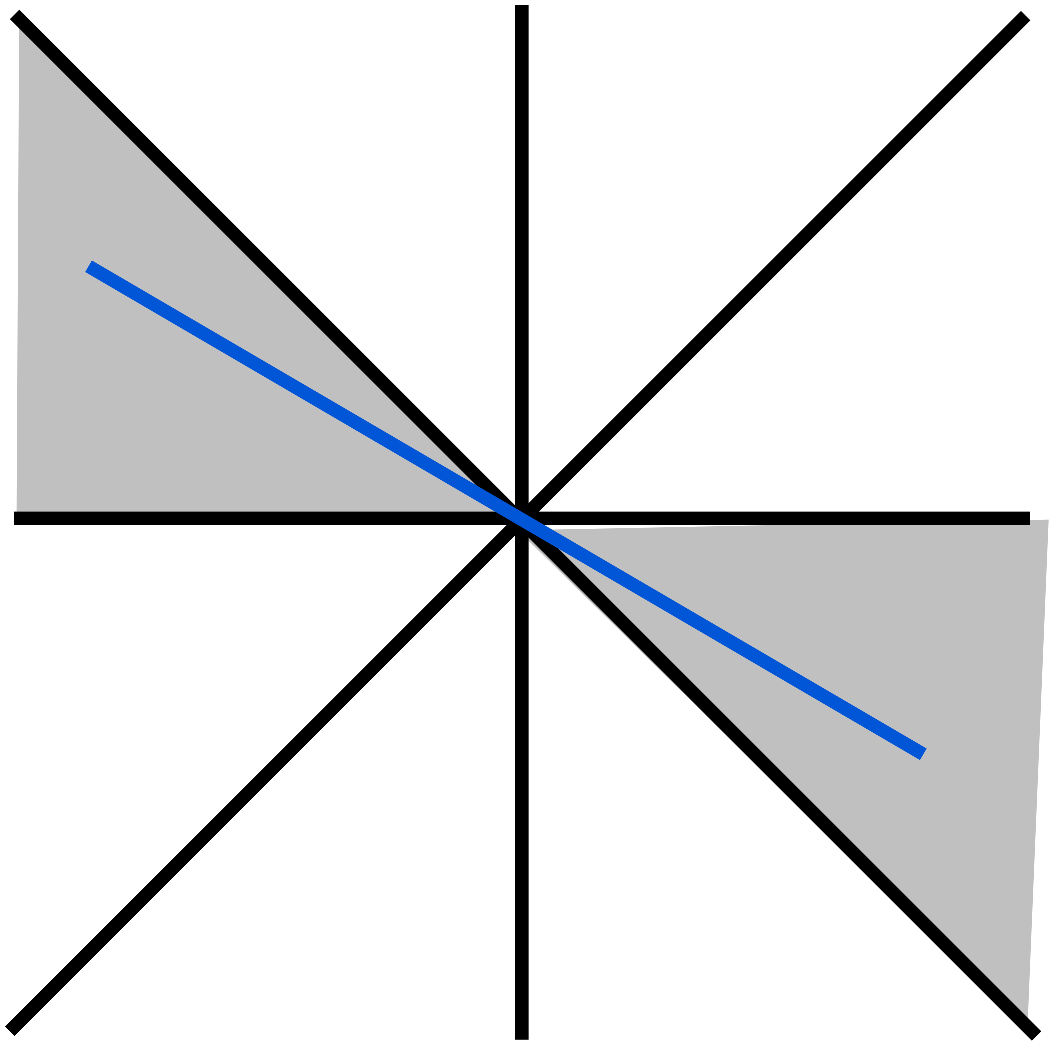
Adapt Midpoint Algorithm for other cases



Adapt Midpoint Algorithm for other cases



Adapt Midpoint Algorithm for other cases

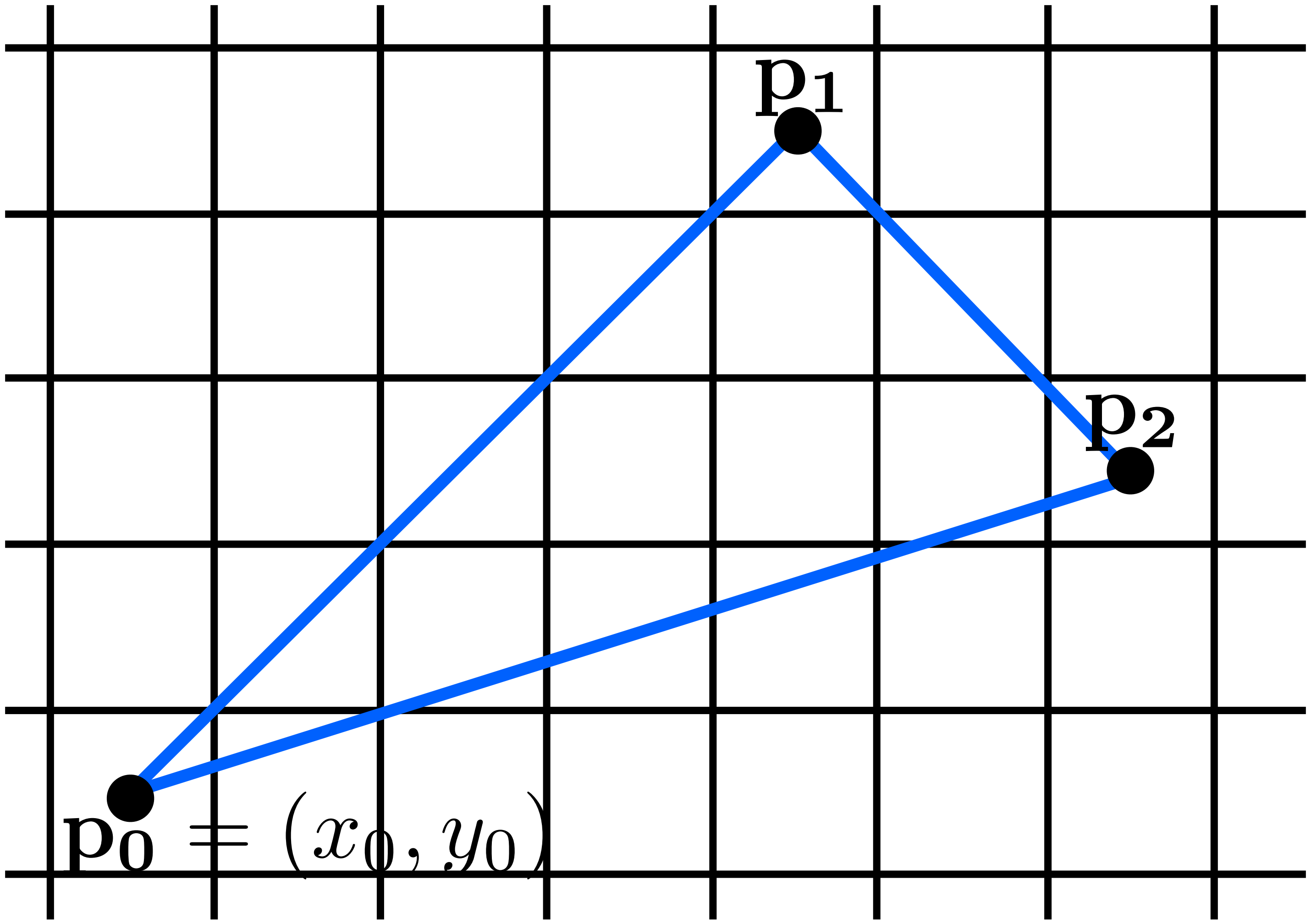


Line drawing references

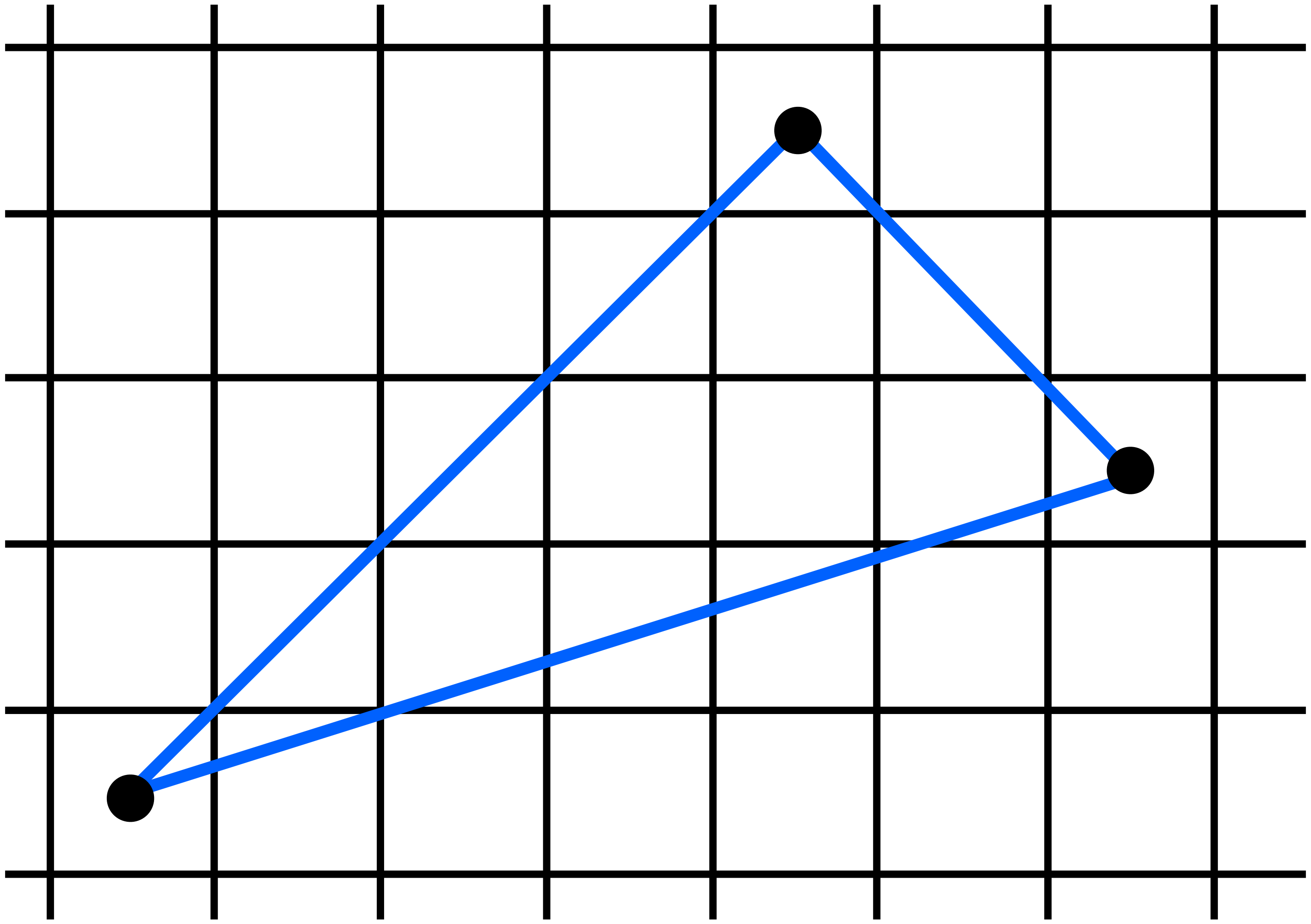
- the algorithm we just described is the *Midpoint Algorithm* (Pitteway, 1967), (van Aken and Novak, 1985)
- draws the same lines as the *Bresenham Line Algorithm* (Bresenham, 1965)

Triangle rasterization

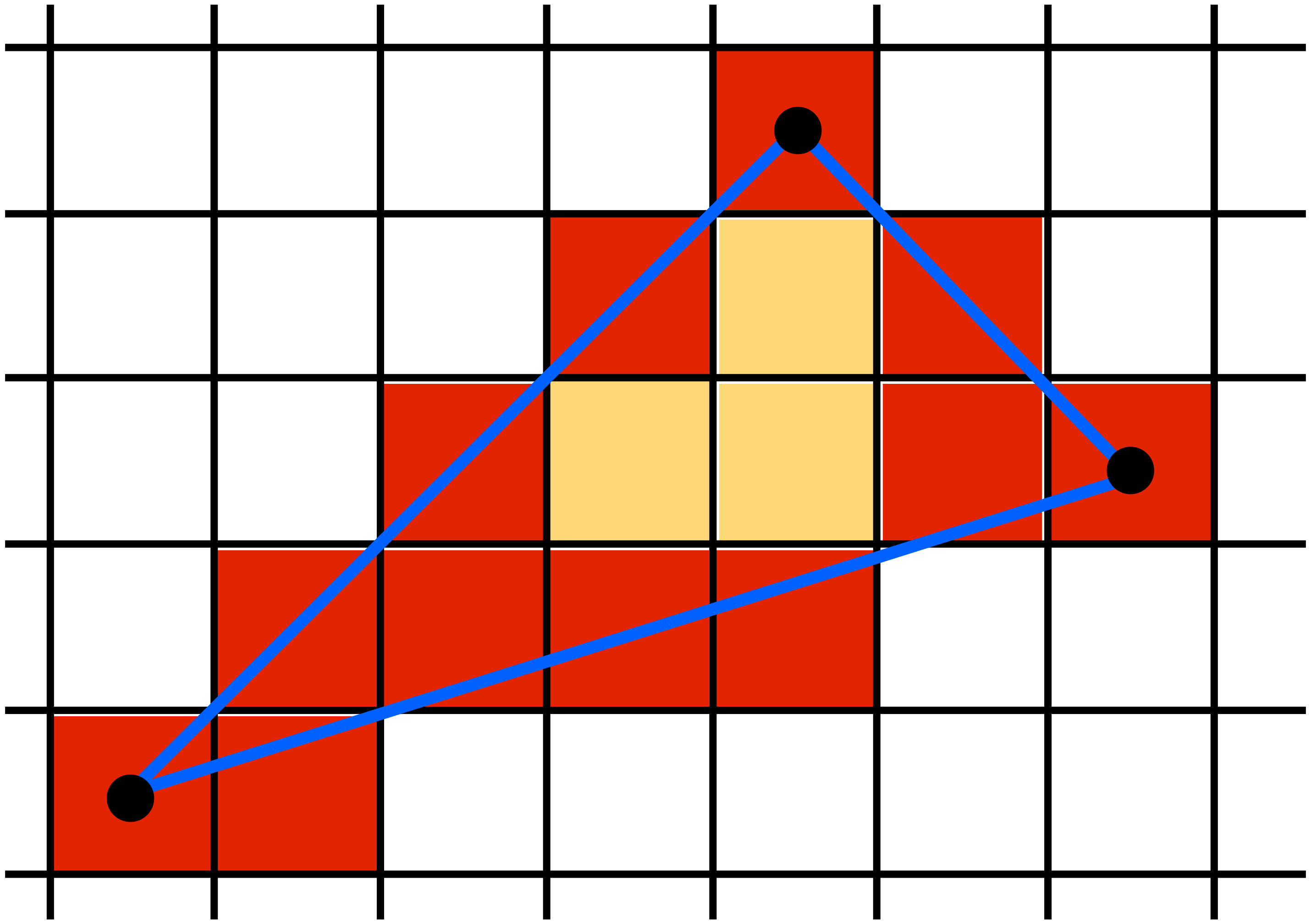
Which pixels should be used to approximate a triangle?



Triangle rasterization issues

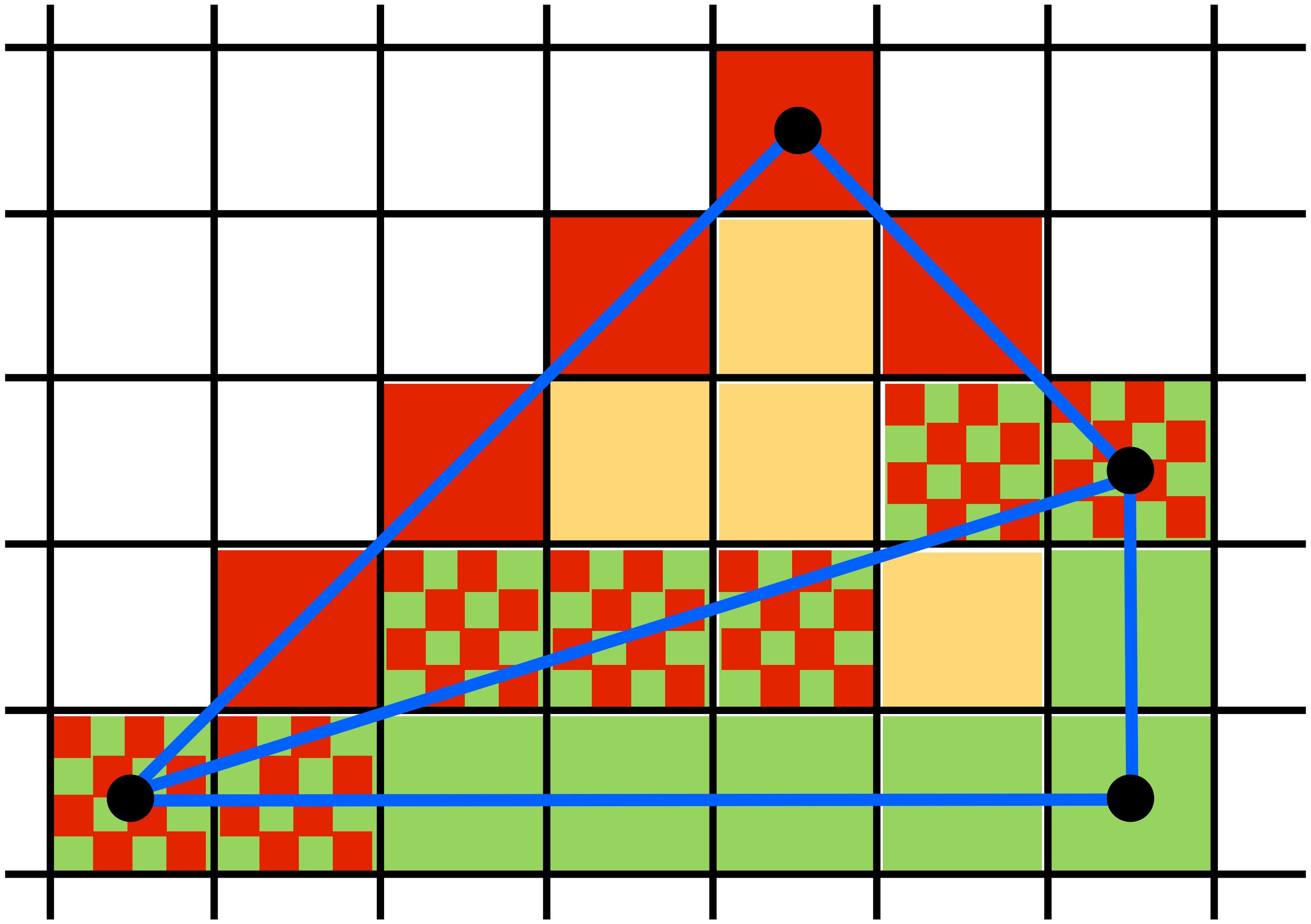


How should we rasterize a triangle?



Use Midpoint Algorithm for edges and fill in

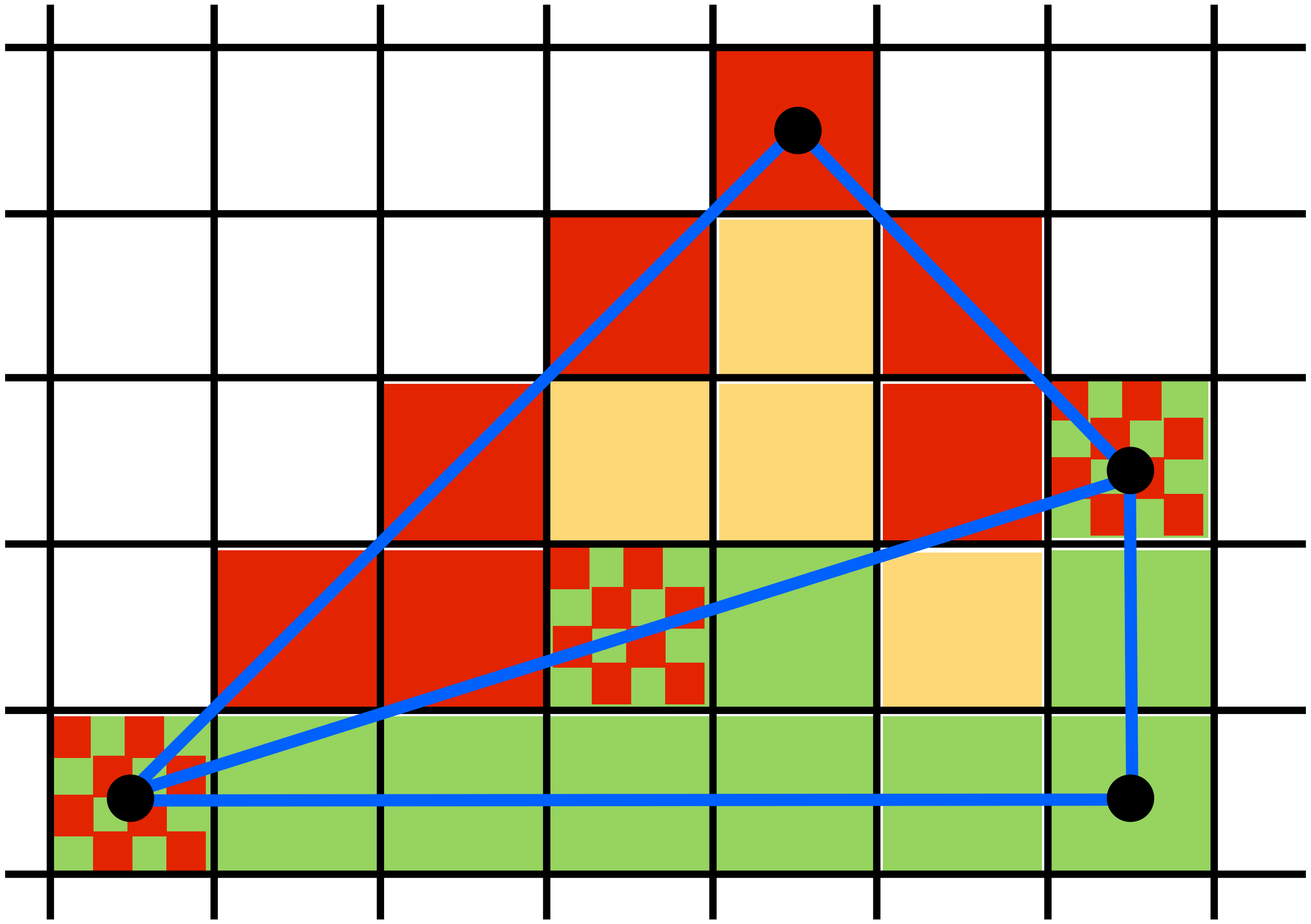
How should we rasterize a triangle?



Who should fill in shared edge?

but who should fill in pixels for a shared edge?

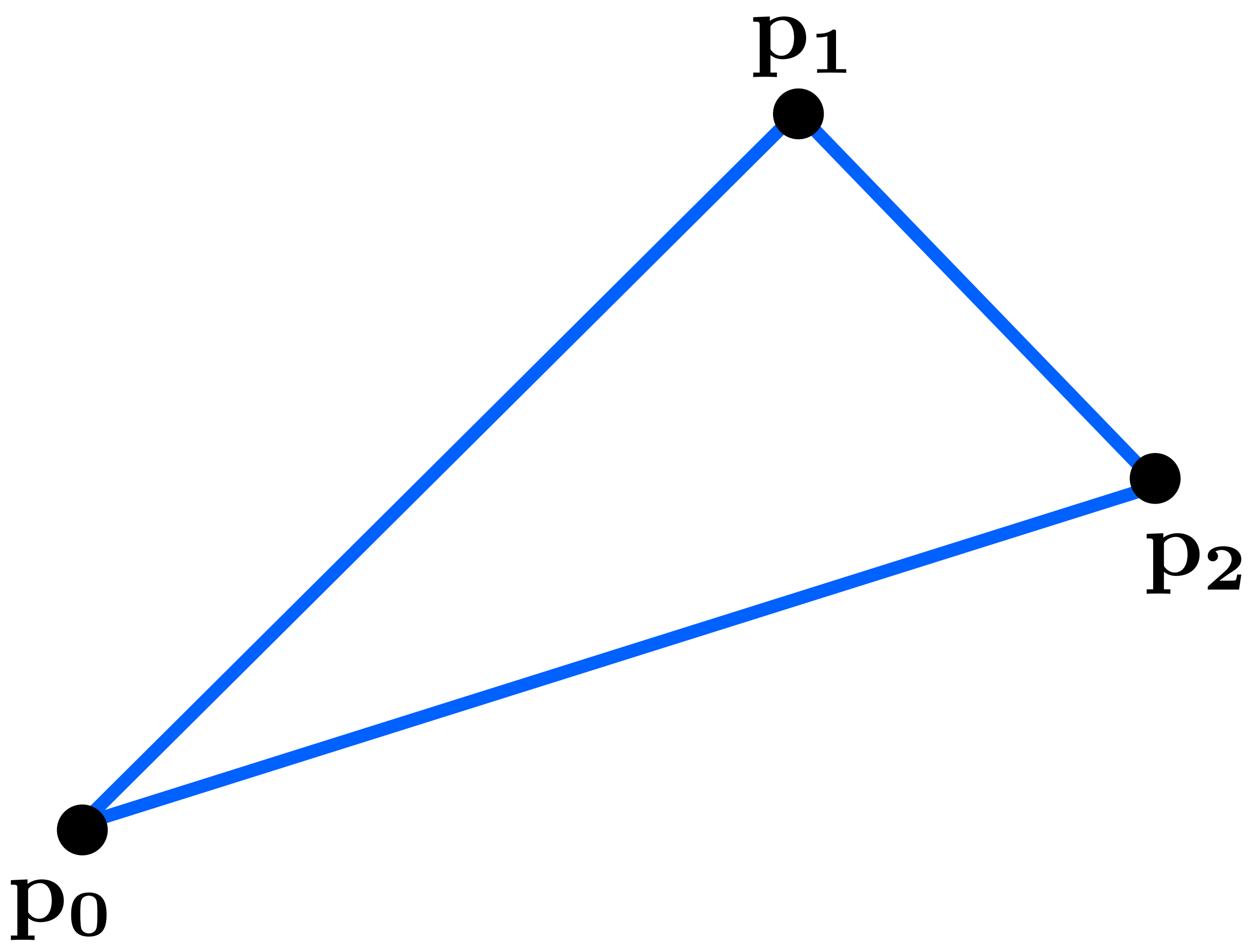
How should we rasterize a triangle?



Who should fill in shared edge?

give to triangle that contains pixel center
- but we have some **ties**
why can’t neither/both triangles draw the pixel?
we went a **unique** assignment

barycentric coordinates



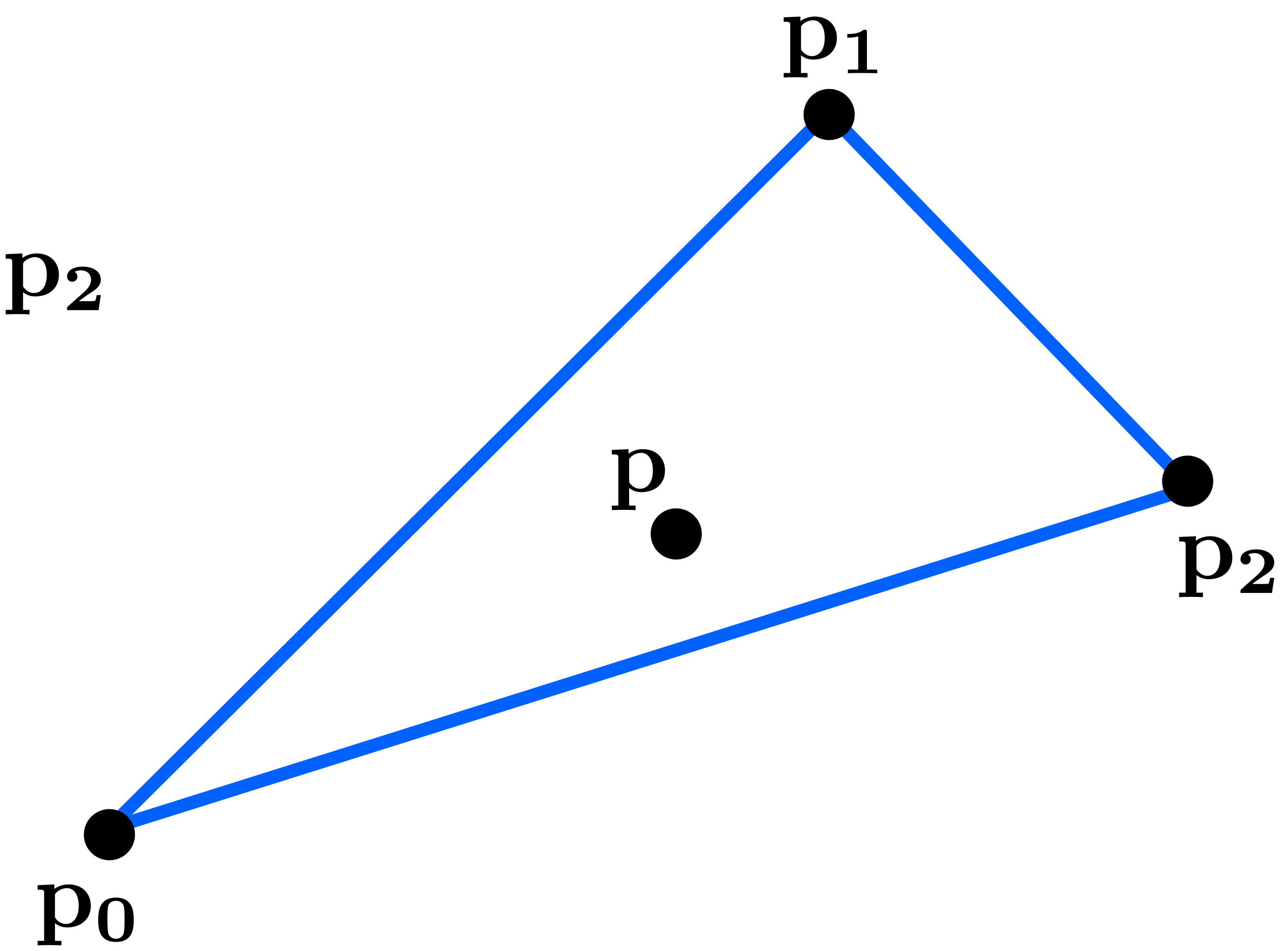
barycentric coordinates

$$\mathbf{p} = f(\mathbf{p_0}, \mathbf{p_1}, \mathbf{p_2})$$

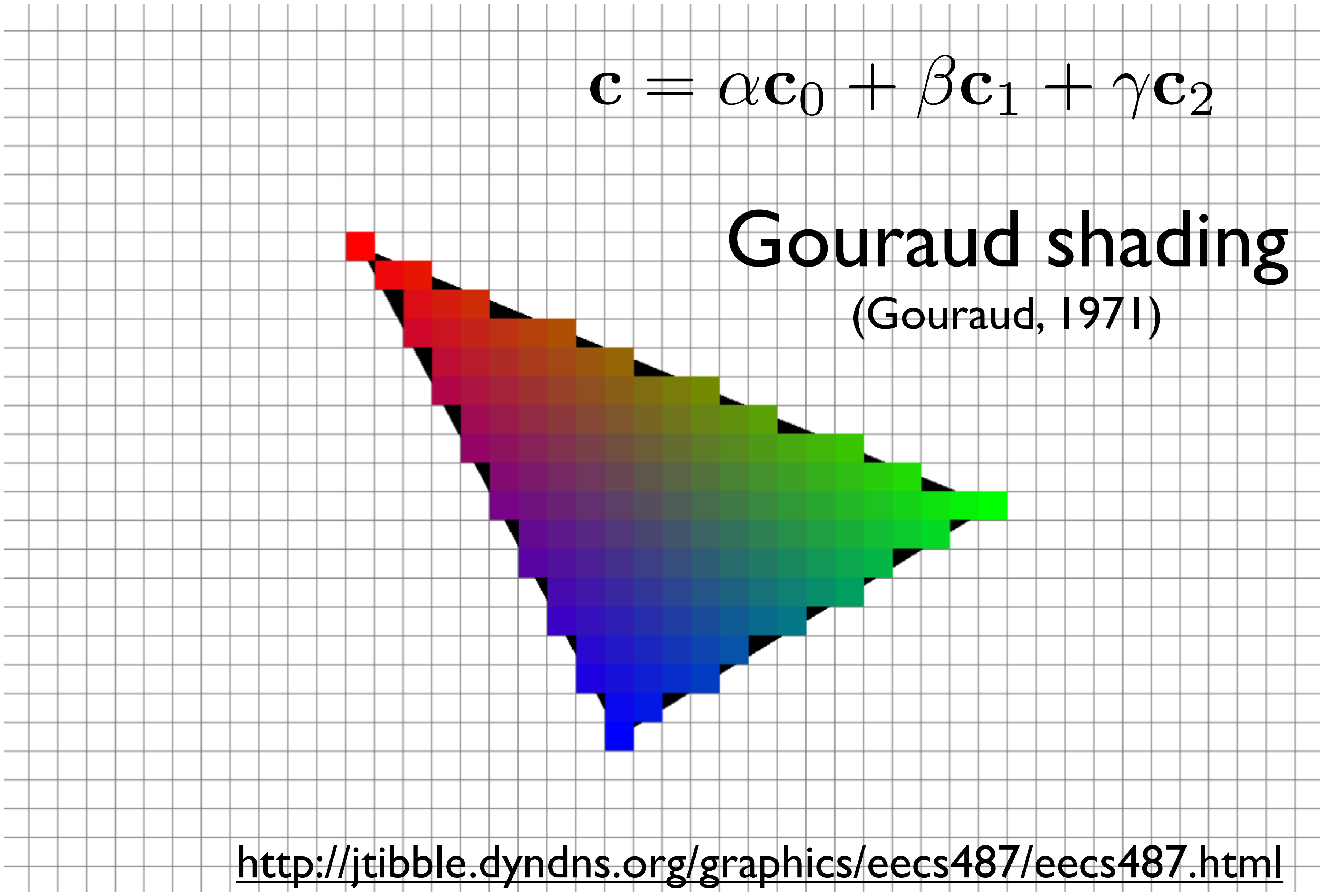
$$\mathbf{p} = \alpha\mathbf{p_0} + \beta\mathbf{p_1} + \gamma\mathbf{p_2}$$

What are (α, β, γ) ?

<whiteboard>



We can interpolate attributes using barycentric coordinates



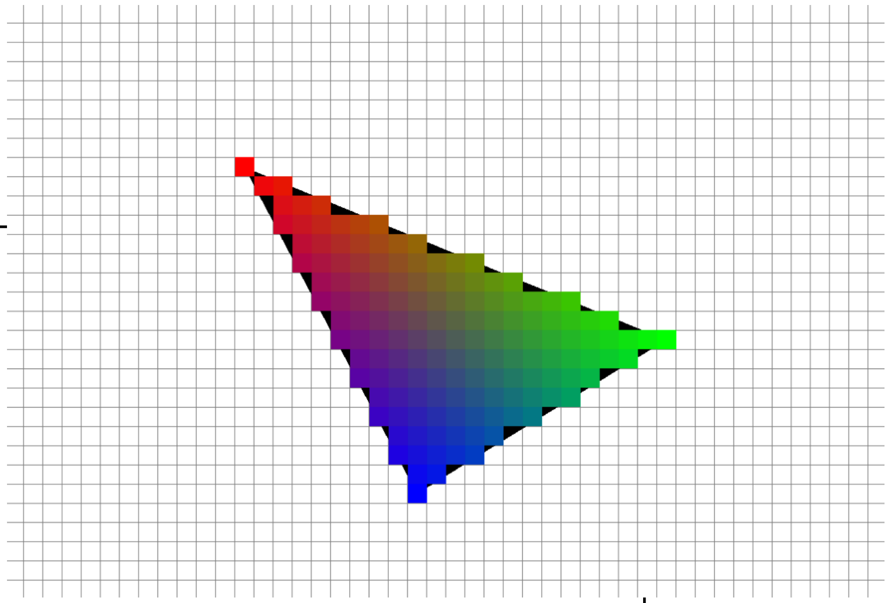
$$\mathbf{c} = \alpha\mathbf{c}_0 + \beta\mathbf{c}_1 + \gamma\mathbf{c}_2$$

Gouraud shading

(Gouraud, 1971)

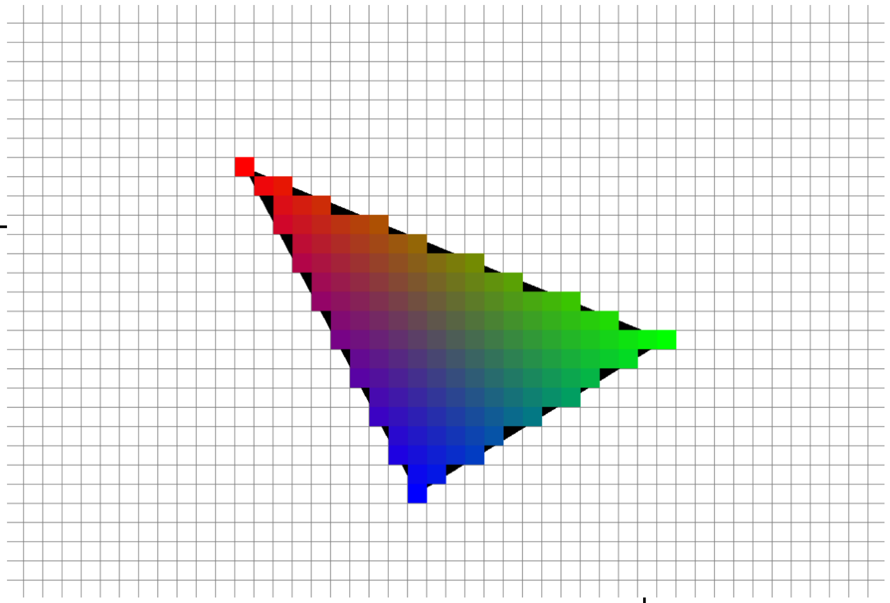
http://jtibble.dyndns.org/graphics/eecs487/eecs487.html

Triangle rasterization algorithm



```
for all x do
    for all y do
        compute $(\alpha, \beta, \gamma)$ for (x,y)
        if $(\alpha \in [0,1] \text{ and } \beta \in [0,1] \text{ and } \gamma \in [0,1])$ then
            $\mathbf{c} = \alpha\mathbf{c}_0 + \beta\mathbf{c}_1 + \gamma\mathbf{c}_2$
            drawpixel(x,y) with color c
```

Triangle rasterization algorithm

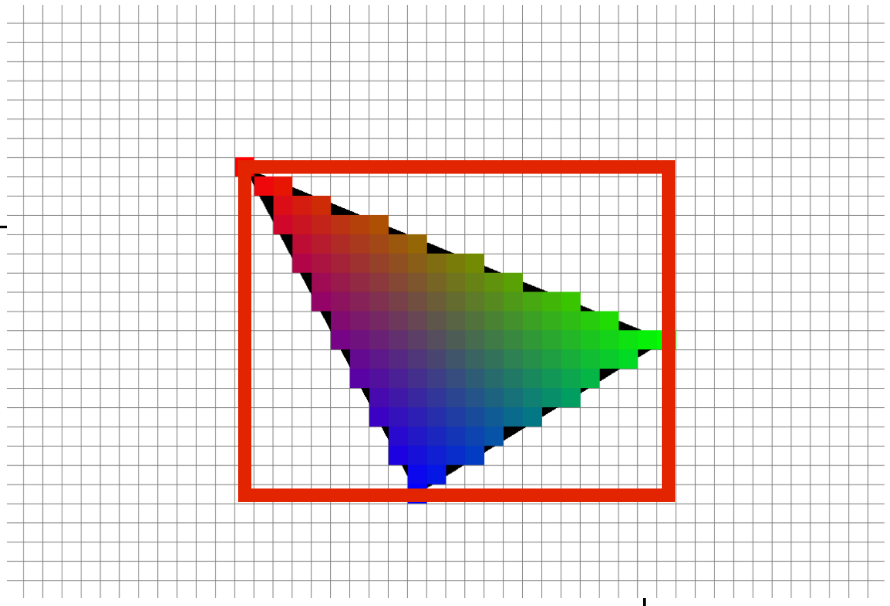


```
for all x do
    for all y do
        compute $(\alpha, \beta, \gamma)$ for (x,y)
        if $(\alpha \in [0,1] \text{ and } \beta \in [0,1] \text{ and } \gamma \in [0,1])$ then
            $\mathbf{c} = \alpha\mathbf{c}_0 + \beta\mathbf{c}_1 + \gamma\mathbf{c}_2$
            drawpixel(x,y) with color c
```

the rest of the algorithm is to make the steps in **red** more **efficient**

Triangle rasterization algorithm

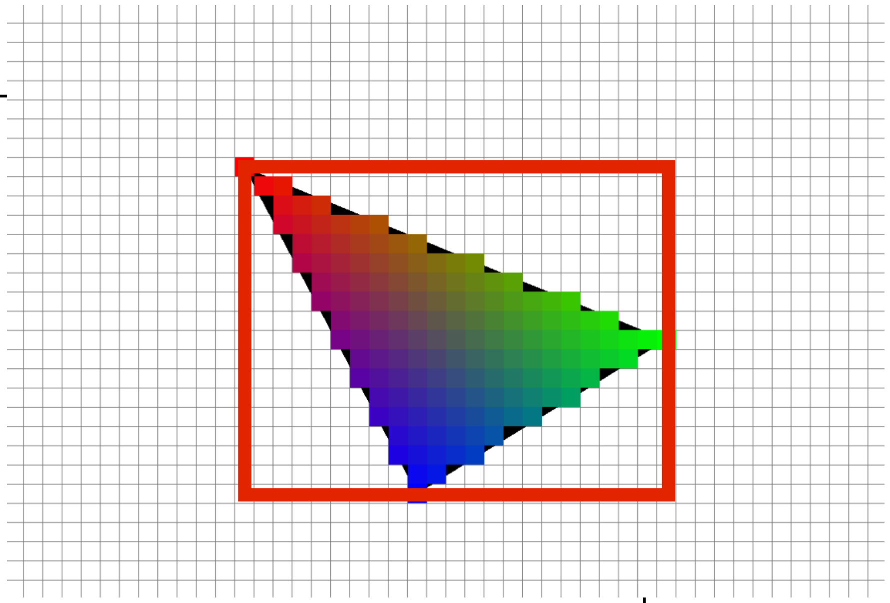
use a bounding rectangle



```
for x in [x_min, x_max]
    for y in [y_min, y_max]
        compute $(\alpha, \beta, \gamma)$ for (x,y)
        if $(\alpha \in [0,1] \text{ and } \beta \in [0,1] \text{ and } \gamma \in [0,1])$ then
            $\mathbf{c} = \alpha\mathbf{c}_0 + \beta\mathbf{c}_1 + \gamma\mathbf{c}_2$
            drawpixel(x,y) with color c
```

Triangle rasterization algorithm

```
for x in [x_min, x_max]
    for y in [y_min, y_max]
        $\alpha = f_{12}(x,y)/f_{12}(x_0,y_0)$
        $\beta = f_{20}(x,y)/f_{20}(x_1,y_1)$
        $\gamma = f_{01}(x,y)/f_{01}(x_2,y_2)$
        if $(\alpha \in [0,1] \text{ and } \beta \in [0,1] \text{ and } \gamma \in [0,1])$ then
            $\mathbf{c} = \alpha\mathbf{c}_0 + \beta\mathbf{c}_1 + \gamma\mathbf{c}_2$
            drawpixel(x,y) with color c
```



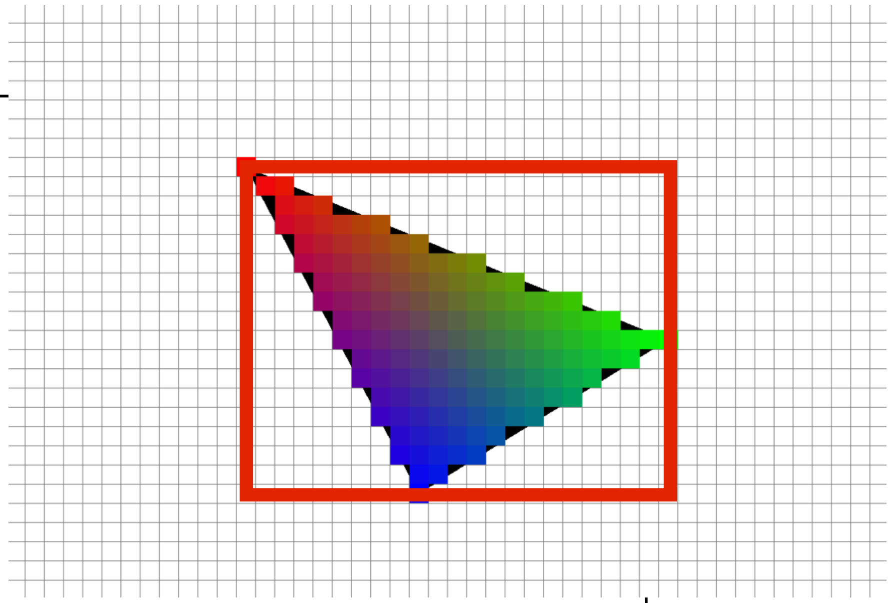
<whiteboard>

<whiteboard> : computing alpha, beta, and gamma

Triangle rasterization algorithm

Optimizations?

```
for x in [x_min, x_max]
    for y in [y_min, y_max]
        $\alpha = f_{12}(x,y)/f_{12}(x_0,y_0)$
        $\beta = f_{20}(x,y)/f_{20}(x_1,y_1)$
        $\gamma = f_{01}(x,y)/f_{01}(x_2,y_2)$
        if $(\alpha \in [0,1] \text{ and } \beta \in [0,1] \text{ and } \gamma \in [0,1])$ then
            $\mathbf{c} = \alpha\mathbf{c}_0 + \beta\mathbf{c}_1 + \gamma\mathbf{c}_2$
            drawpixel(x,y) with color c
```



1. can make computation of bary. coords. **incremental**

– $f(x,y) = Ax+By+C$

– $f(x+1,y) = f(x,y) + A$

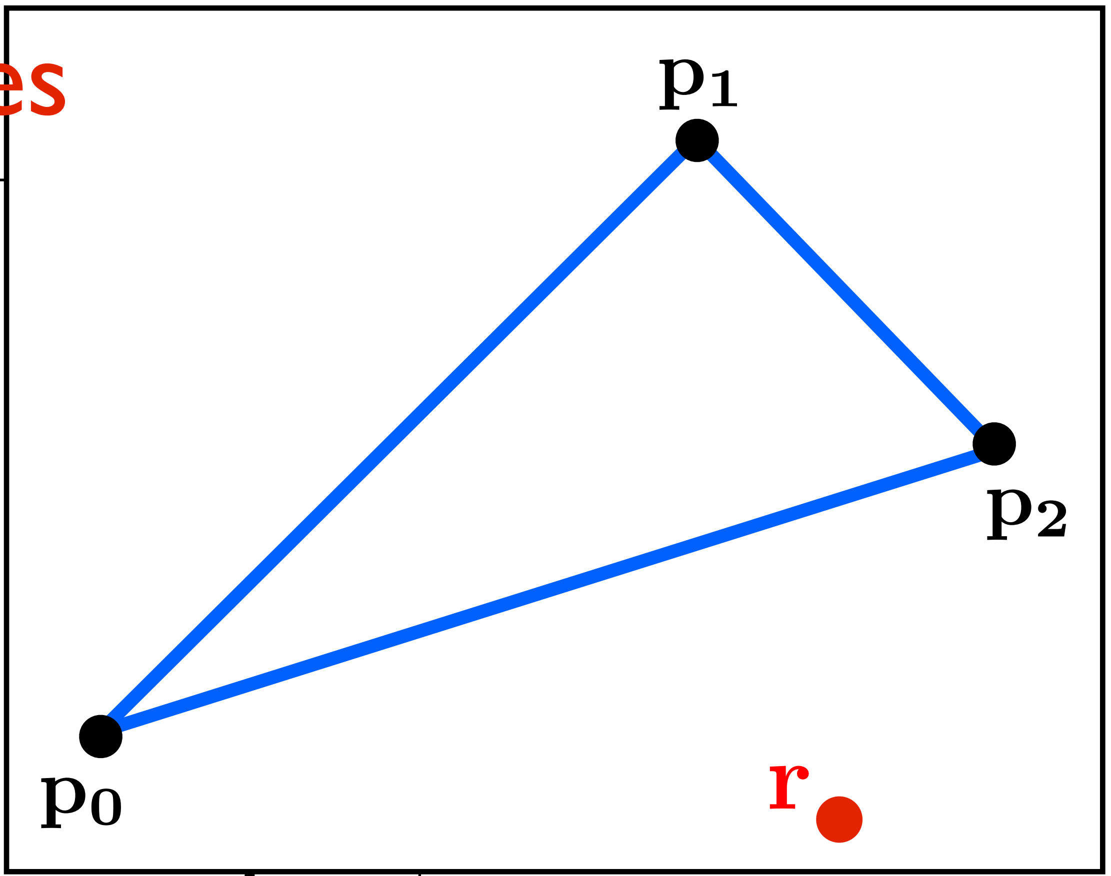
2. **color** computation can also be made **incremental**

3. **alpha > 0 and beta > 0 and gamma > 0** (if true => they are also less than one)

Triangle rasterization algorithm

dealing with shared triangle edges

```
for x in [x_min, x_max]
    for y in [y_min, y_max]
        $\alpha = f_{12}(x,y)/f_{12}(x_0,y_0)$
        $\beta = f_{20}(x,y)/f_{20}(x_1,y_1)$
        $\gamma = f_{01}(x,y)/f_{01}(x_2,y_2)$
        if $(\alpha \geq 0 \text{ and } \beta \geq 0 \text{ and } \gamma \geq 0)$ then
            if $(\alpha > 0 \text{ or } f_{12}(\mathbf{p}_0)f_{12}(\mathbf{r}) > 0) \text{ and }$ then
               $(\beta > 0 \text{ or } f_{20}(\mathbf{p}_1)f_{20}(\mathbf{r}) > 0) \text{ and }$
               $(\gamma > 0 \text{ or } f_{01}(\mathbf{p}_2)f_{01}(\mathbf{r}) > 0)$
                $\mathbf{c} = \alpha\mathbf{c}_0 + \beta\mathbf{c}_1 + \gamma\mathbf{c}_2$
                drawpixel(x,y) with color c
```



– compute f_12(r), f_20(r) and f_01(r) and make sure r doesn't hit a line