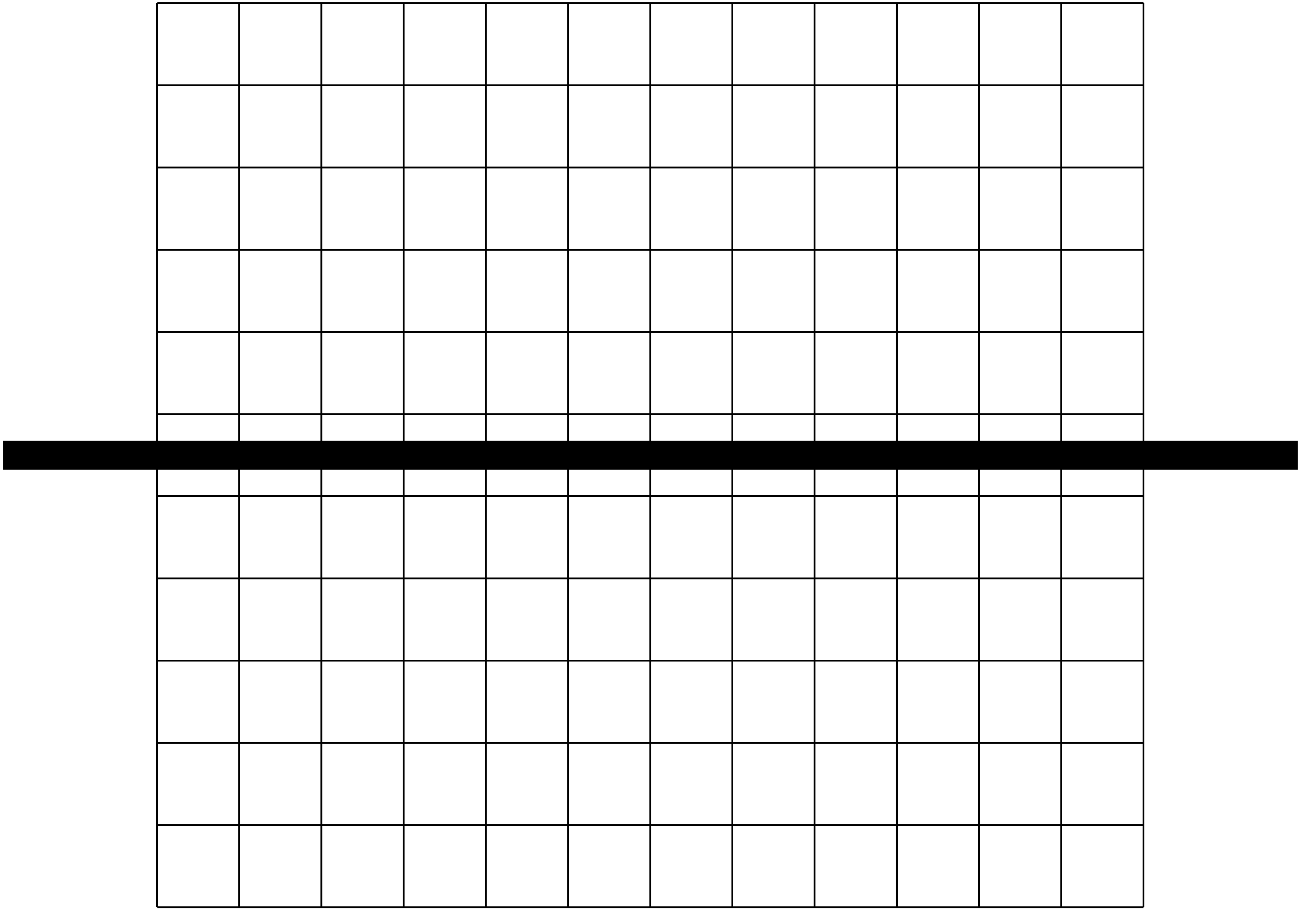
LINE DRAWING

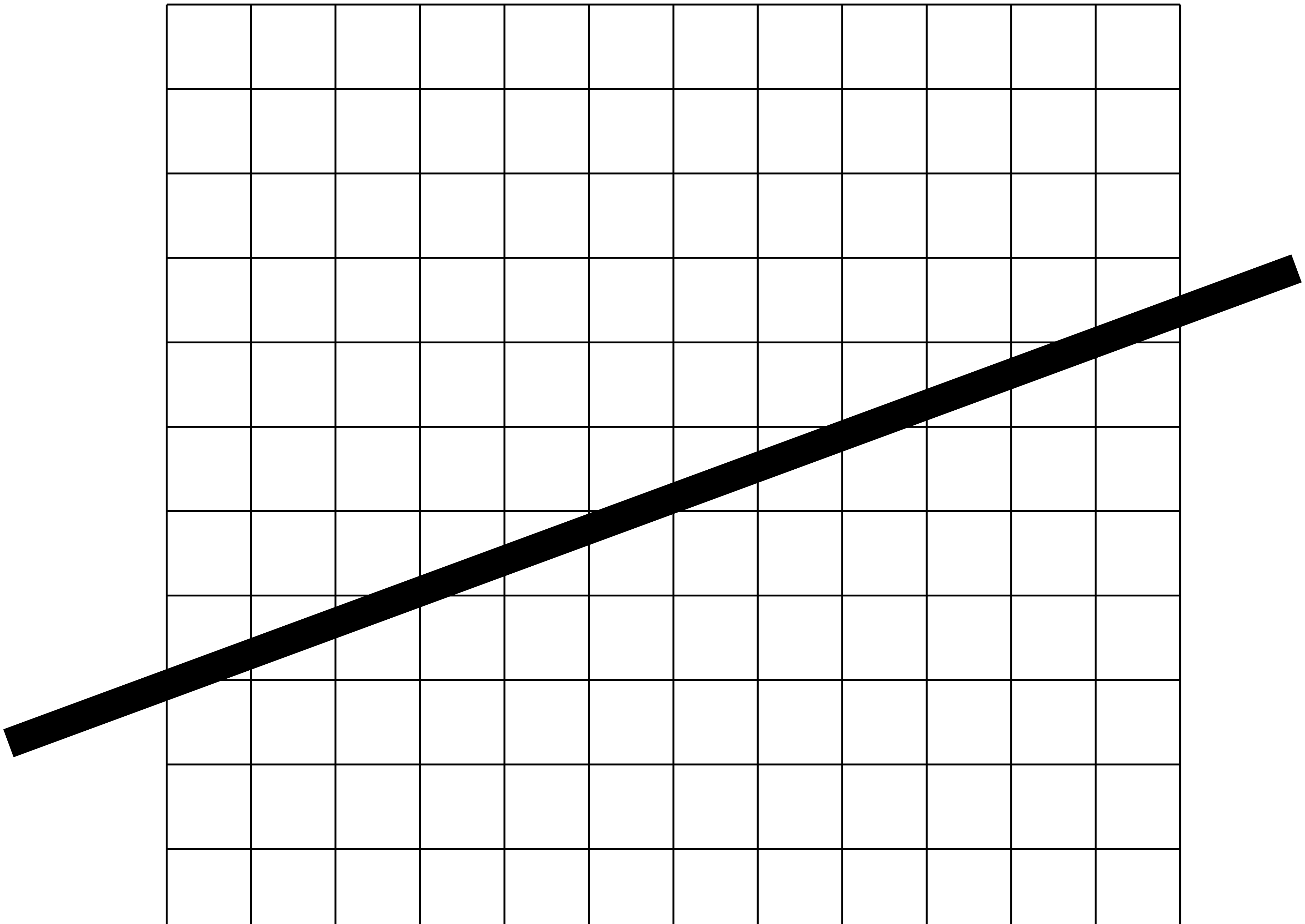
2011 Introduction to Graphics

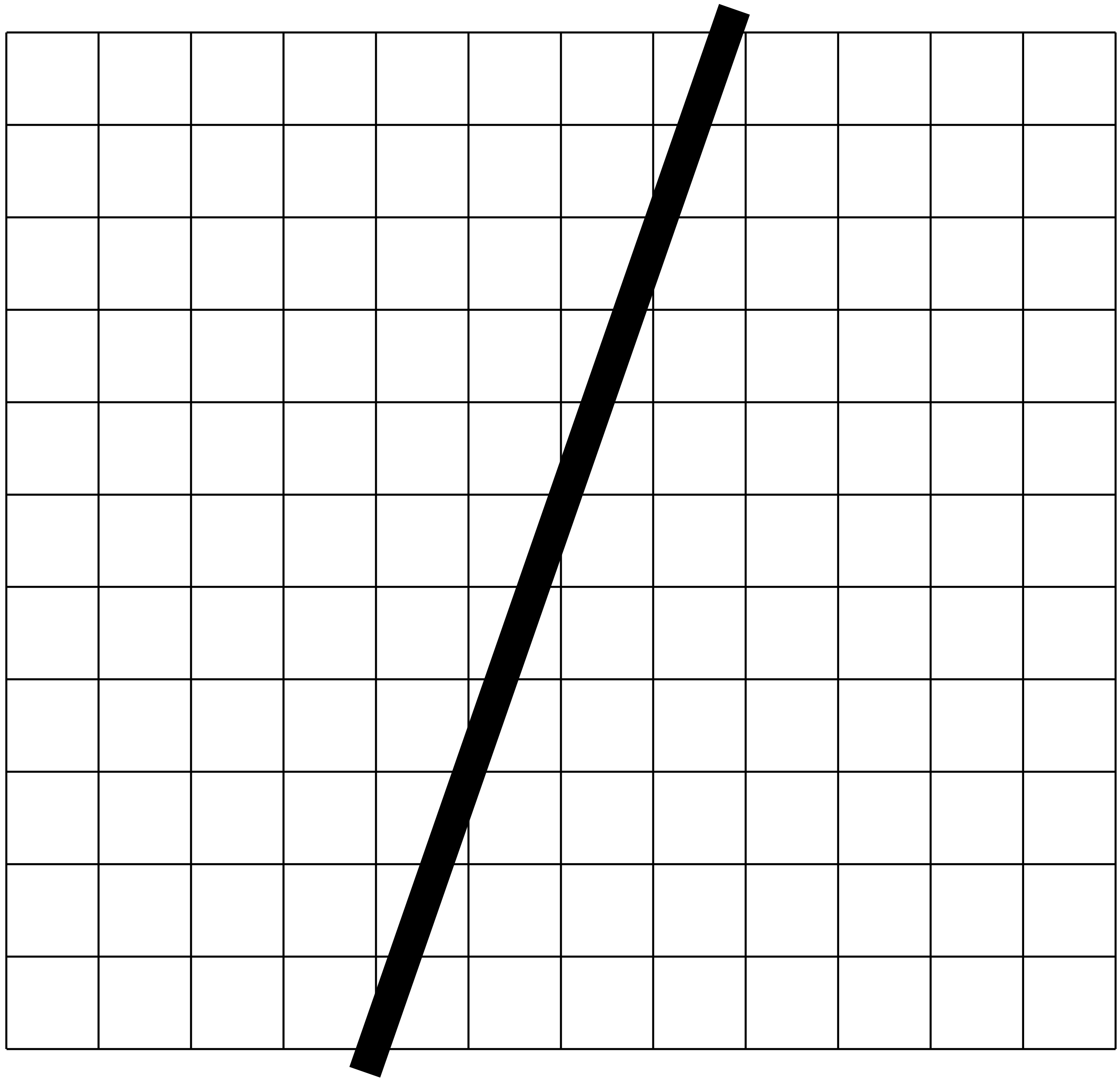
Lecture 8

Overview

- Line drawing is hard!
- Ideal lines and drawing them
- Bresenham's algorithm
  - Stages of optimisation
- Going Faster
- Going Faster Still





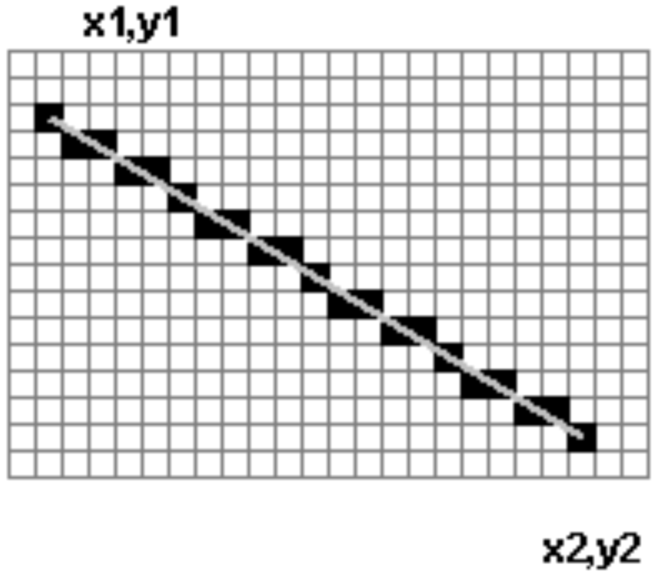


Ideal Lines

- From the equation of a line

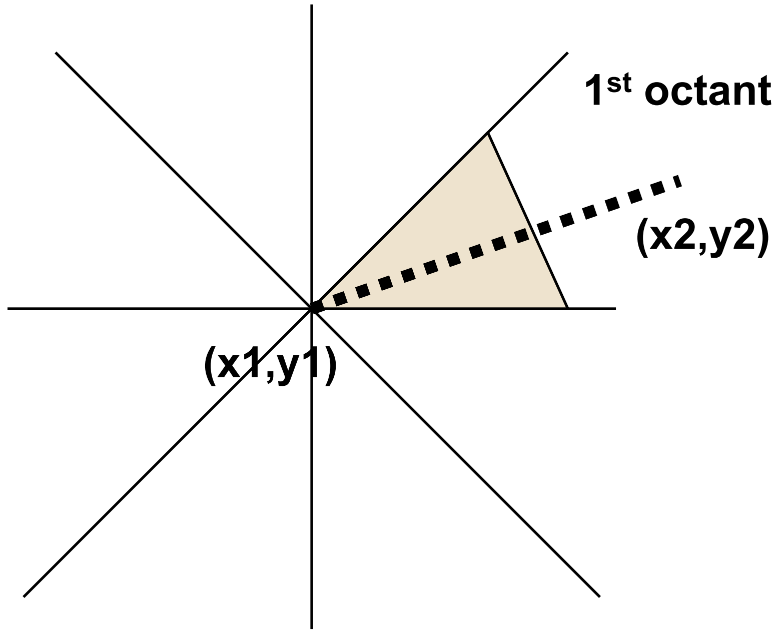
$$y = y_1 + \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x - x_1)$$

- Find a discretisation



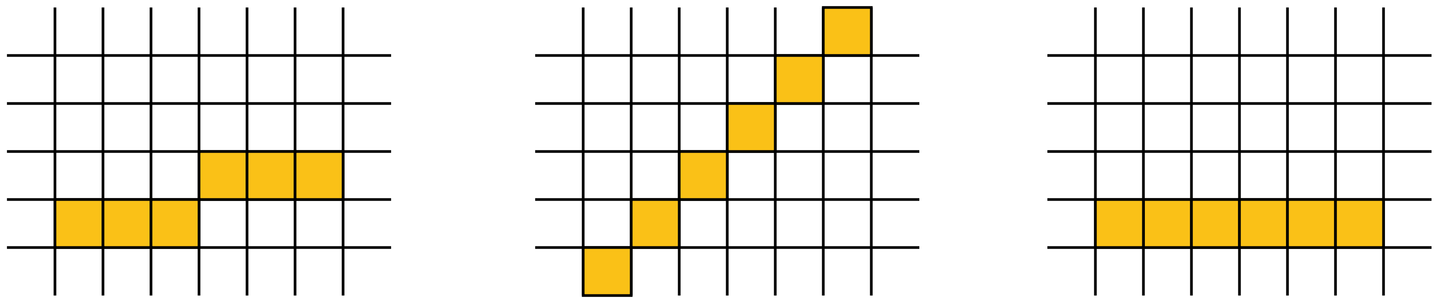
Octants

- We will choose one case (1^{st} octant)
  - Line gradient is > 0 and is < 1
- There are eight variations of the following algorithms
  - 4 iterate over X
  - 4 iterate over Y



In the 1st Octant

- We know that there is more than one pixel on every row (i.e. X increases faster than y)
- Also Y increase as X increases



- Note that horizontal, vertical and 45 degree lines often treated as special cases

Naïve Algorithm

```
dx = x2-x1
dy = y2 - y1
for (x=x1;x<=x2;x++) {
    y = round(y1+(dy/dx)*(x-x1))
    setPixel(x,y)
}
```

- Problems
  - one divide, one round, two adds, one multiply per pixel

First Speed Up - An Obvious Thing

- Obviously the gradient does not change each time through the loop
- Calculate $m = dy/dx = (y2-y1)/(x2-x1)$ **once**
- Note that
  - $y(x) = y1+m*(x-x1)$
  - $y(x+1) = y1+m*((x+1)-x1)$
  - $y(x+1)-y(x) = m$

Step 0

Step 1: **Convert to incremental algorithm**

Step 0:

```
dx = x2-x1
dy = y2 - y1
for (x=x1;x<=x2;x++) {
    y = round(y1+(dy/dx)*
                        (x-x1))
    setPixel(x,y)
}
```

Step 1:

```
dx = x2-x1
dy = y2 - y1
y = y1
for (x=x1;x<=x2;x++) {
    y += dy/dx
    setPixel(x,round(y))
}
```

Step 1

$dx = x_2 - x_1$

$dy = y_2 - y_1$

$y = y_1$

$for\ (x=x_1; x<=x_2; x++)\ \{$

$\quad y \mathrel{+}= dy/dx$

$\quad setPixel(x, round(y))$

$\}$

Step 2: **Replace round**

$dx = x_2 - x_1$

$dy = y_2 - y_1$

$y = y_1$

$for\ (x=x_1; x<=x_2; x++)\ \{$

$\quad y \mathrel{+}= dy/dx$

$\quad setPixel(x, (int)(y+0.5))$

$\}$

Step 2

$dx = x_2 - x_1$

$dy = y_2 - y_1$

$y = y_1$

for $(x=x_1; x<=x_2; x++)$ {

  y += dy/dx

  setPixel(x,(int)(y+0.5))

}

Step 3: **Split y into an integer and fraction part**

$dx = x_2 - x_1 \quad dy = y_2 - y_1$

$y_i = y_1 \quad y_f = 0.0$

for $(x=x_1; x<=x_2; x++)$ {

  y_f += dy/dx

  if $(y_f > 0.5)$ {

   y_i++

   y_f--

  }

  setPixel(x, y_i)

}

Note y_f ***is always in range –0.5 to 0.5***

Step 3

$dx = x_2 - x_1 \quad dy = y_2 - y_1$

$y_i = y_1 \quad y_f = 0.0$

for $(x=x_1;x<=x_2;x++)$ {

    y_f += dy/dx

    if $(y_f > 0.5)$ {

      y_i++

      y_f--

    }

    setPixel(x,y_i)

}

Step 4: **Shift y_f by 0.5**

$dx = x_2 - x_1 \quad dy = y_2 - y_1$

$y_i = y_1 \quad y_f = -0.5$

for $(x=x_1;x<=x_2;x++)$ {

    y_f+= dy/dx

    if $(y_f > 0.0)$ {

      y_i++

      y_f--

    }

    setPixel(x,y_i)

}

Step 4

```
dx = x2-x1   dy = y2 - y1
yi = y1   yf = -0.5
for (x=x1;x<=x2;x++) {
    yf += dy/dx
    if (yf > 0.0) {
      yi++
      yf--
    }
    setPixel(x,yi)
}
```

Step 5: **Multiply y_f through by 2dx**

```
dx = x2-x1   dy = y2 - y1
yi = y1   yf = -dx
for (x=x1;x<=x2;x++) {
    yf += 2dy
    if (yf > 0) {
      yi++
      yf += -2dx
    }
    setPixel(x,yi)
}
```

We now have all integer arithmetic

Step 5

$dx = x_2 - x_1 \quad dy = y_2 - y_1$

$y_i = y_1 \quad y_f = -dx$

$for\ (x=x_1; x<=x_2; x++)\ \{$

$\quad y_f \mathrel{+}= 2dy$

$\quad if\ (y_f > 0)\ \{$

$\quad\quad y_i{+}{+}$

$\quad\quad y_f \mathrel{+}= -2dx$

$\quad \}$

$\quad setPixel(x, y_i)$

$\}$

Step 6: **Re-arrange if**

$dx = x_2 - x_1 \quad dy = y_2 - y_1$

$y_i = y_1 \quad y_f = -dx$

$for\ (x=x_1; x<=x_2; x++)\ \{$

$\quad if\ (y_f > 0)\ \{$

$\quad\quad y_i{+}{+}$

$\quad\quad y_f \mathrel{+}= 2dy - 2dx$

$\quad \}\ else\ \{$

$\quad\quad y_f \mathrel{+}= 2dy$

$\quad \}$

$\quad setPixel(x, y_i)$

$\}$

This has one less add statement

Step 6

```
dx = x2-x1   dy = y2 - y1
yi = y1   yf = -dx
for (x=x1;x<=x2;x++) {
    if (yf > 0) {
      yi++
      yf += 2dy -2dx
    } else {
      yf += 2dy
    }
    setPixel(x,yi)
}
```

Reduced to one if, 1 or 2 adds

Step 7: **Make all constants**

```
dx = x2-x1   dy = y2 - y1
yi = y1   yf = -dx
e = 2dy – 2dx
f = 2dy
for (x=x1;x<=x2;x++) {
    if (yf > 0) {
      yi++
      yf +=e
    } else {
      yf += f
    }
    setPixel(x,yi)
}
```

What have we got

- Only integer arithmetic
- One if, 1/2 adds per pixel
  - One hidden if in the loop

- Can we get faster than that?

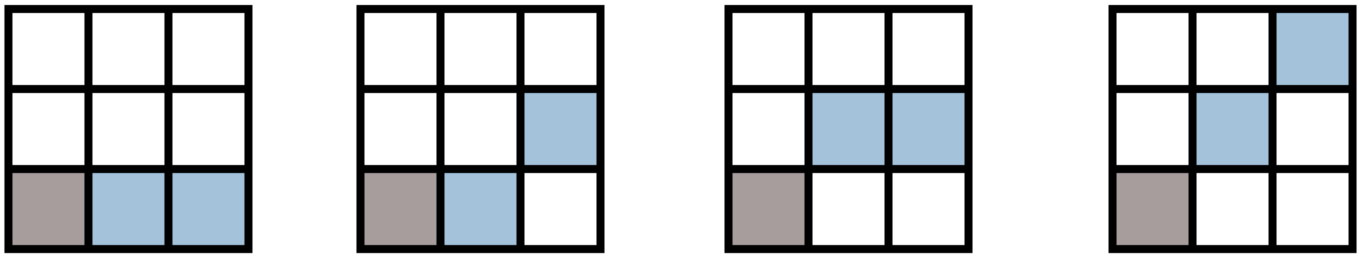
Faster - Mid-Point Drawing

- Note that lines are symmetric
- (a,b) to (c,d) should generate the same pixels. Thus the lines are symmetric about the mid-point
- Implies…
  - draw outwards from mid-point in both directions
- Is almost twice as fast (one extra add gives an extra pixel)

Faster - Two-Step

- Note we considered whether to choose between

- What if we choose between the four cases



- Almost twice the speed again

Questions

- How would you draw thick lines?
- Will Bresenham really result in a big speed-up (think setPixel())?
- How would you do anti-aliasing?

Summary

- Bresenham's algorithm uses only integer arithmetic and 2/3 ops per pixel
  - Ideal for hardware implementation
- Can be improved almost four-fold in speed, with added complexity
  - Good for software implementations
  - Pixel scanning is usually not the bottleneck these days so wouldn't be cast into hardware