# Lecture 15

## Low-level graphics algorithms

OpenGL ES
WebGL
Vulkan

# Low-level algorithms

Curve generation
Polygon fill
Flood fill

# Curve generation

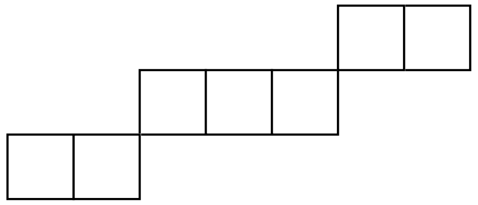
Problem: Generate a digital curve

Find a connected sequence of discrete pixels that follows the curve as closely as possible

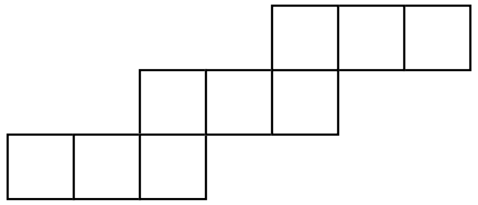
The curve should be either 4-connected or 8-connected, one pixel wide

# Connectivity

8-connected: horizontal, vertical and diagonal moves are allowed:

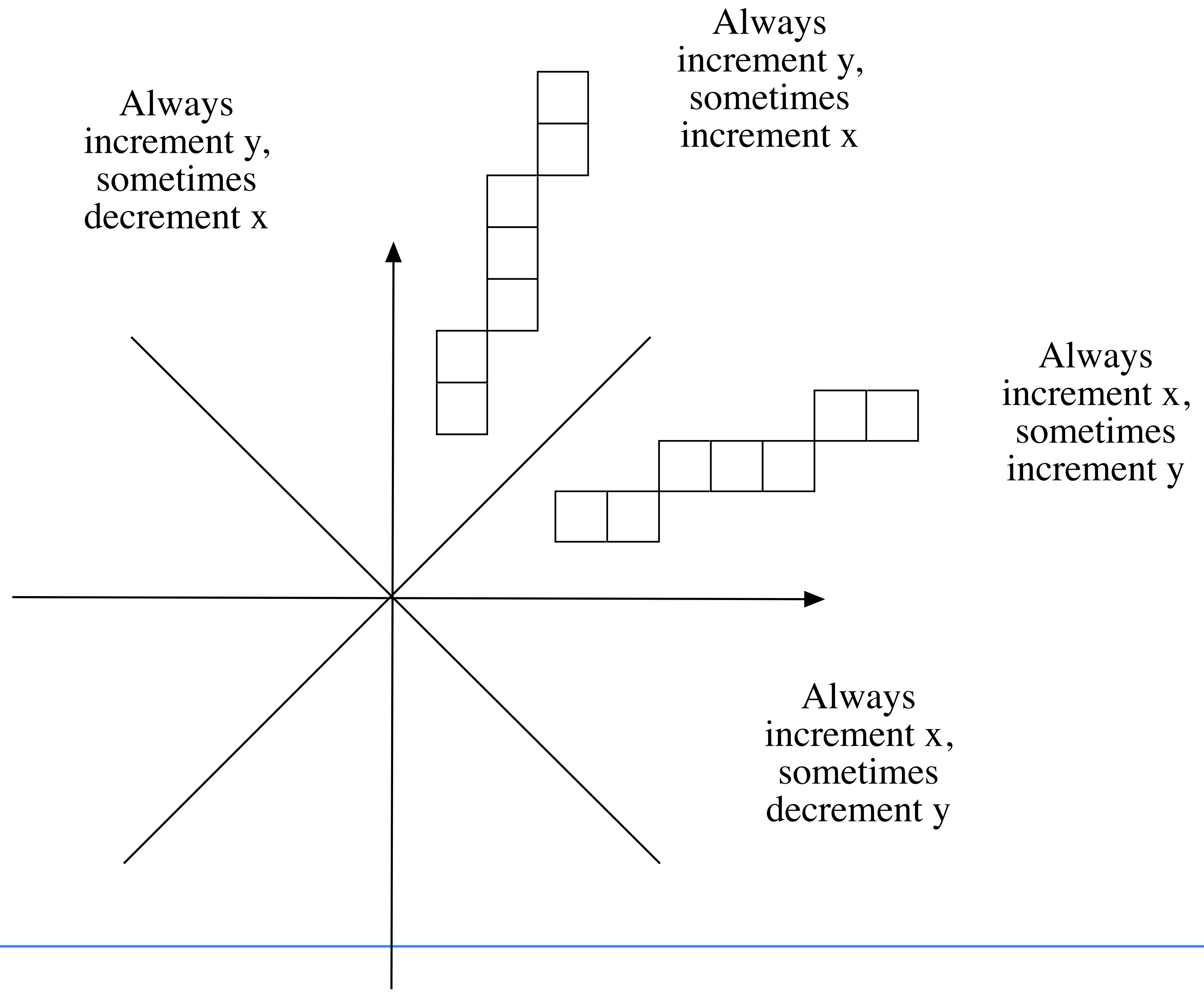


4-connected: diagonal moves are not allowed:



Choose one, don’t mix them! 8-connectedness most common for curve generation.

# We need to move differently in different directions


Always increment y, sometimes decrement x
Always increment y, sometimes increment x
Always increment x, sometimes increment y
Always increment x, sometimes decrement y

# Two line drawing algorithms:

**The DDA algorithm:**

**Simple but slow on low-end hardware**

**The Bresenham algorithm:**

**Extremely fast on any hardware**

# Line drawing, summary

**DDA algorithm**

**Floating-point**
**Simple and straightforward**

**Bresenham’s algorithm**

**Integer-based**
**Incremental; Additions and shifts only**
**Ideal for low-power hardware**

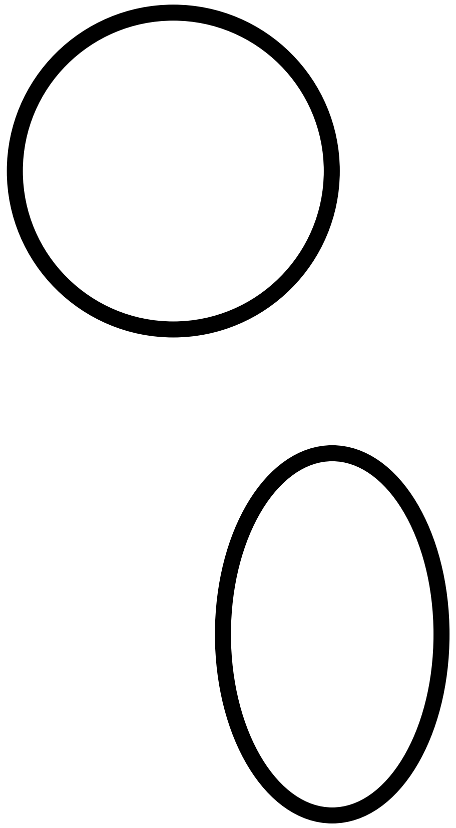
# Other curves

## Midpoint algorithm

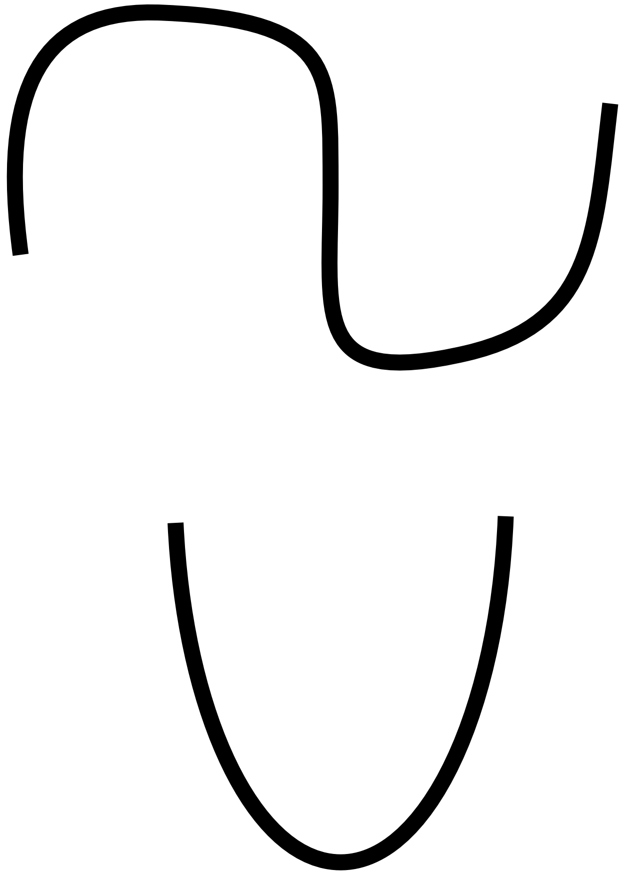
**Any curve that can be expressed by polynomial**

**”Midpoint” refers to measurements at the midpoint between candidates**

# The midpoint algorithm can draw (with excellent speed)



- Circles
- Ellipses
- Parabolas
- Most splines



# Curve attributes

**Width**
**Color and patterns**
**End caps of curves**
**Corner shapes**
**Dashed lines**

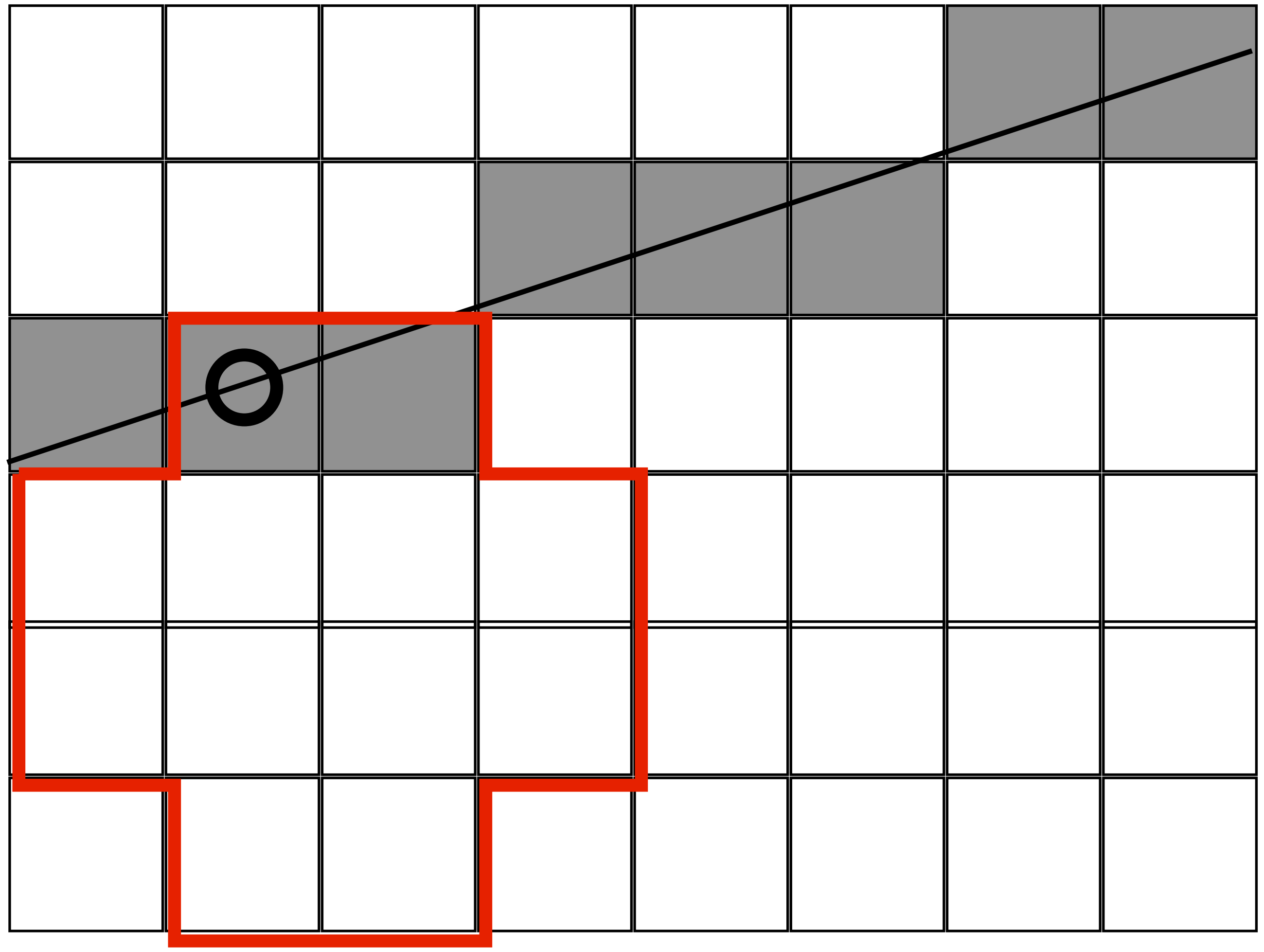
# Drawing curves with greater width than 1

## Two approaches:

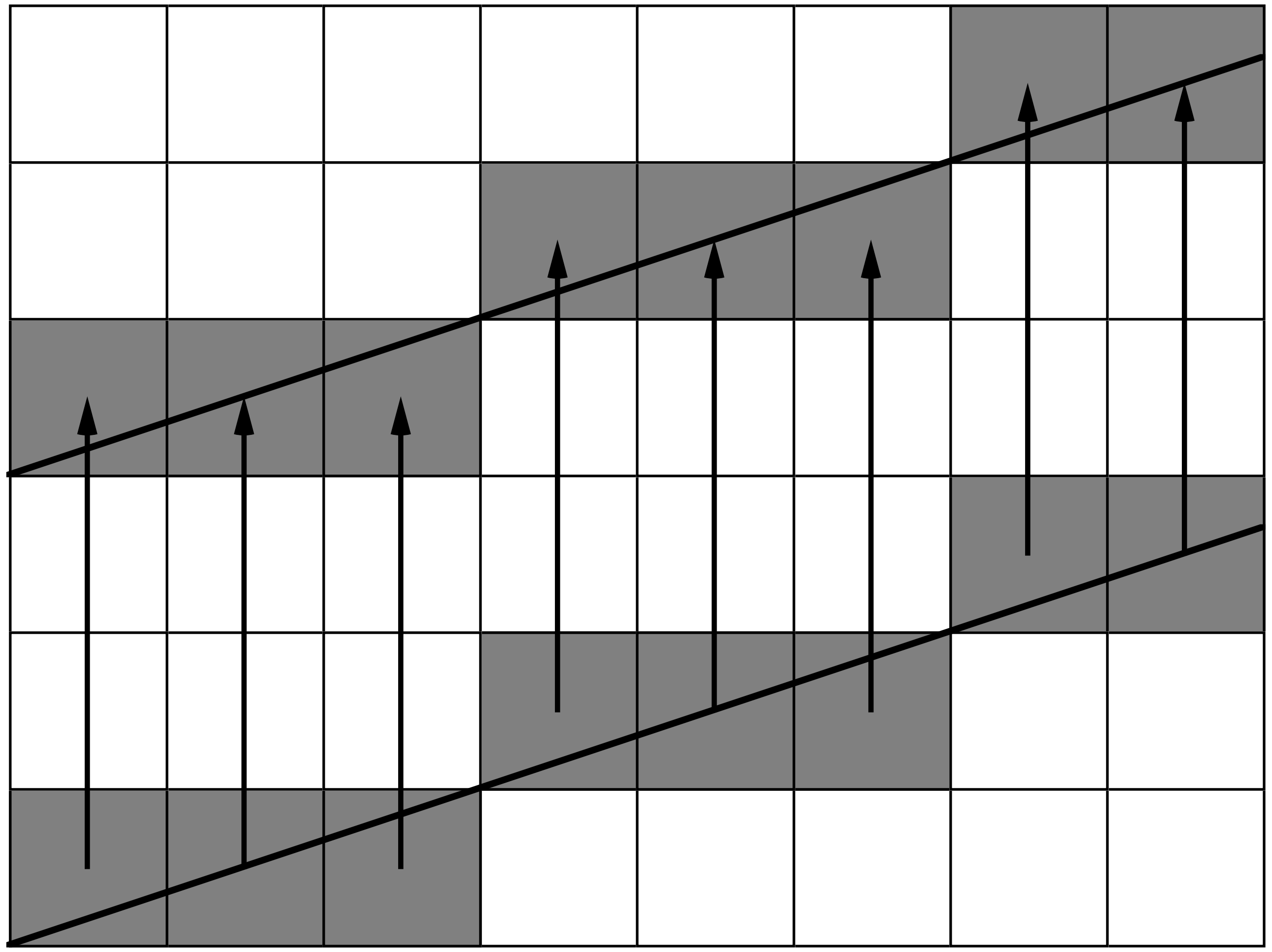
(1) Using a pen shape

(2) Using two parallel curves

# Using a pen shape

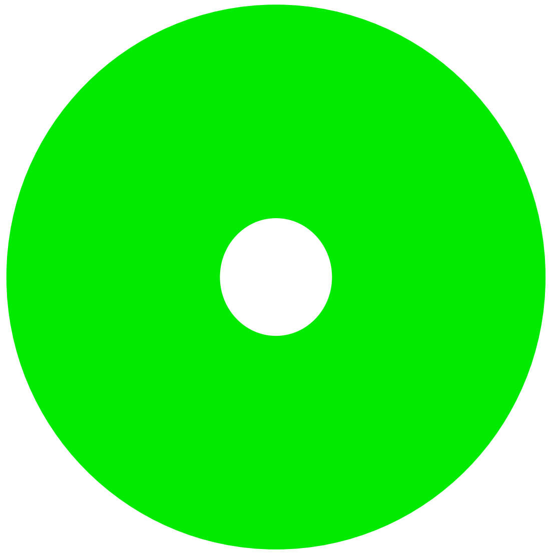


# Using two parallel curves

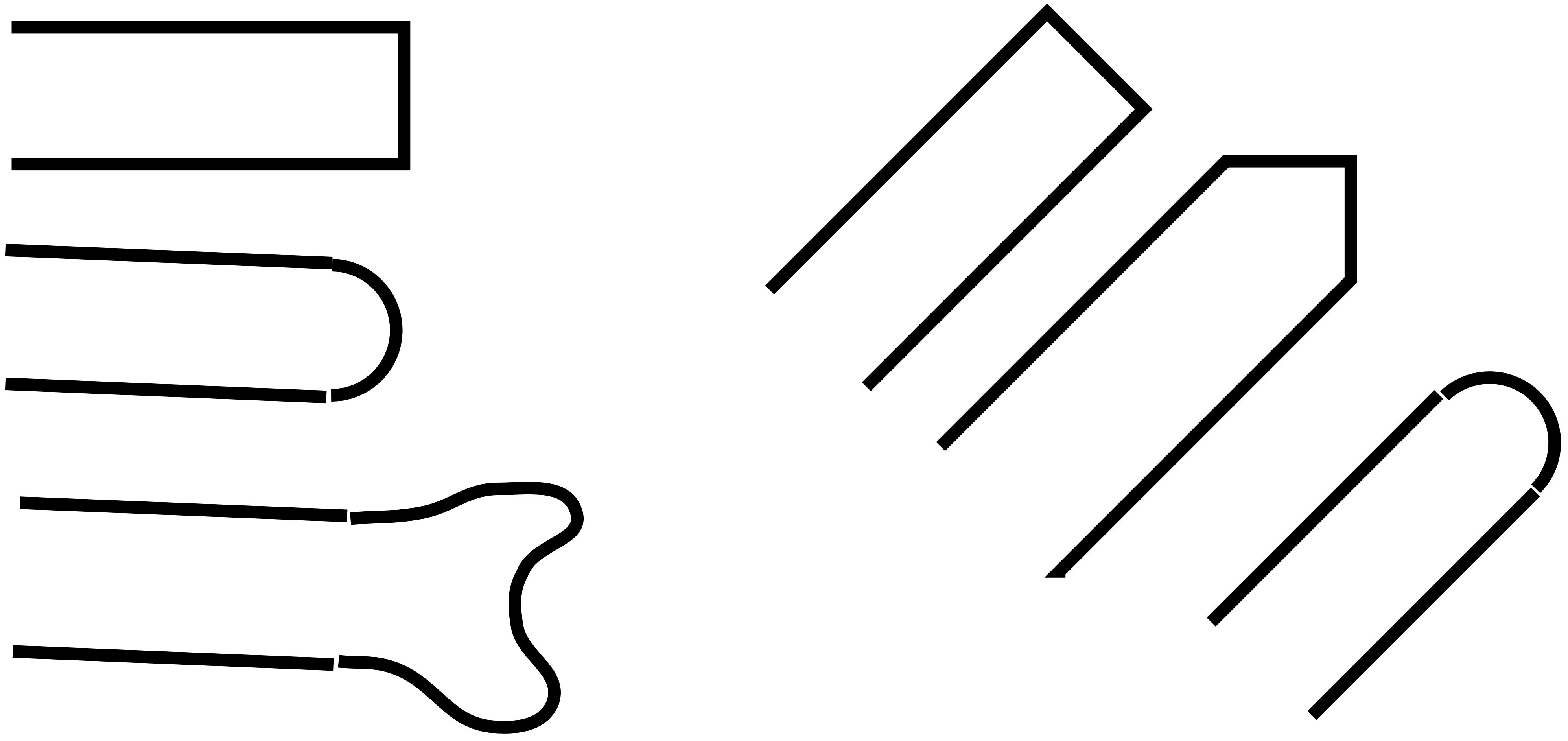


# Drawing a wide circle

## How?



# End caps



# OpenGL vs line and point drawing

For OpenGL, everything are polygons!

Even lines and points are drawn with polygons.

-> Simplifies the optimized OpenGL kernel

# When do I need a line drawing algorithm?

**Drawing lines: Rarely. You probably have a well optiized algorithm in any library.**

**BUT it can be used for other purposes. For example ray marching! (Ray-casting in grid!)**