Plane Geometry Algorithms

Geometry Algorithms

Show up every now and then

- 0-1 questions per contest

Almost always 2D (plane) geometry

Geometry Algorithms

Show up every now and then

- 0-1 questions per contest

Almost always 2D (plane) geometry

- 3D algorithms “too hard”/specialized
- test cases easier to create
- diagrams easier to draw…

Geometry Algorithms

Key challenges:

1. Remembering (or deriving) formulas

Blast Zone

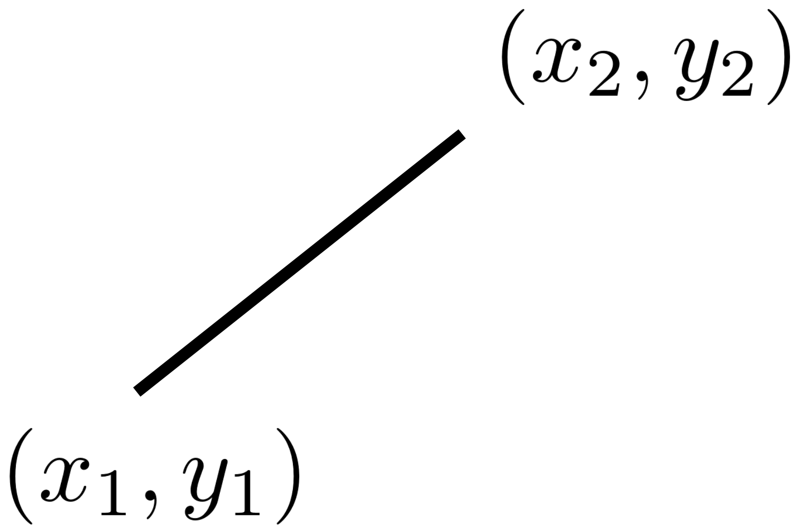
Given:

- list of points ($\mathbf{x}_i$, $\mathbf{y}_i$)
- epicenter ($\mathbf{x}_c$, $\mathbf{y}_c$)
- radius $\mathbf{r}$

Count number of points distance <= $\mathbf{r}$ from epicenter

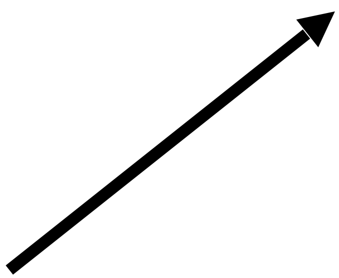
All inputs are ints between -2^{30} and 2^{30}

Vectors and Distance



Vectors and Distance

$$(x_2, y_2)$$



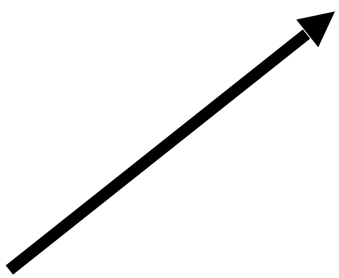
$$(x_1, y_1)$$

vector **from** (x_1, y_1) **to** (x_2, y_2) :

$$(x_2 - x_1, y_2 - y_1)$$

Vectors and Distance

(x_2, y_2)



(x_1, y_1)

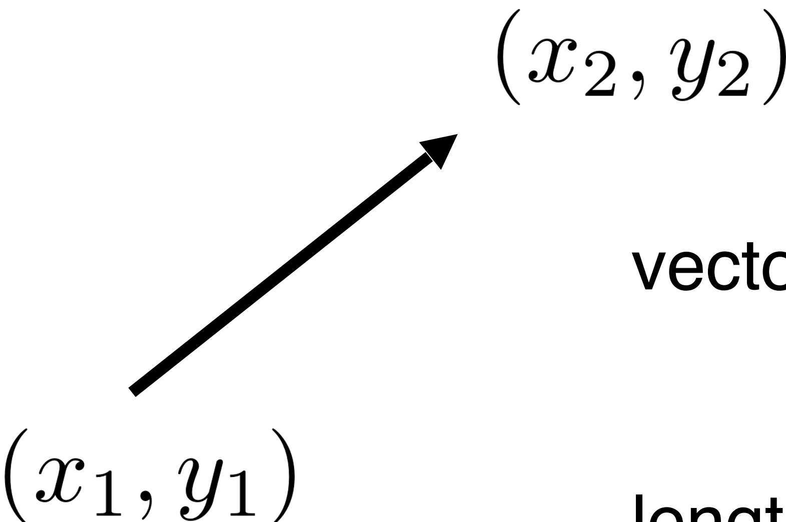
vector **from** (x_1, y_1) **to** (x_2, y_2) :

$$(x_2 - x_1, y_2 - y_1)$$

length of vectors:

$$\|(a, b)\| = \sqrt{a^2 + b^2}$$

Vectors and Distance



vector **from** (x_1, y_1) **to** (x_2, y_2) :

$$(x_2 - x_1, y_2 - y_1)$$

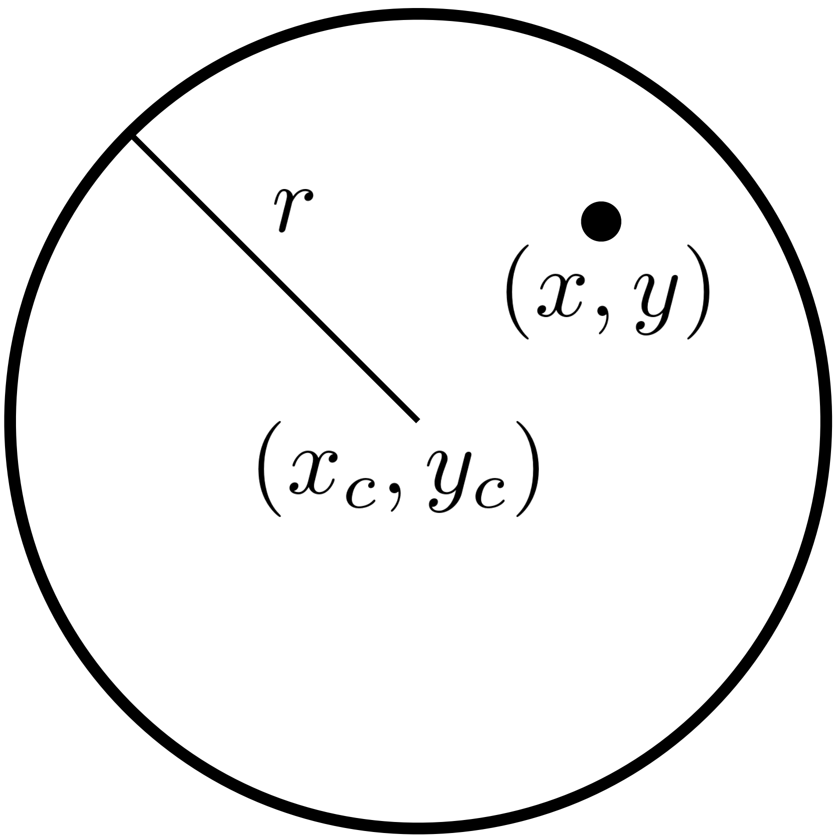
length of vectors:

$$\|(a, b)\| = \sqrt{a^2 + b^2}$$

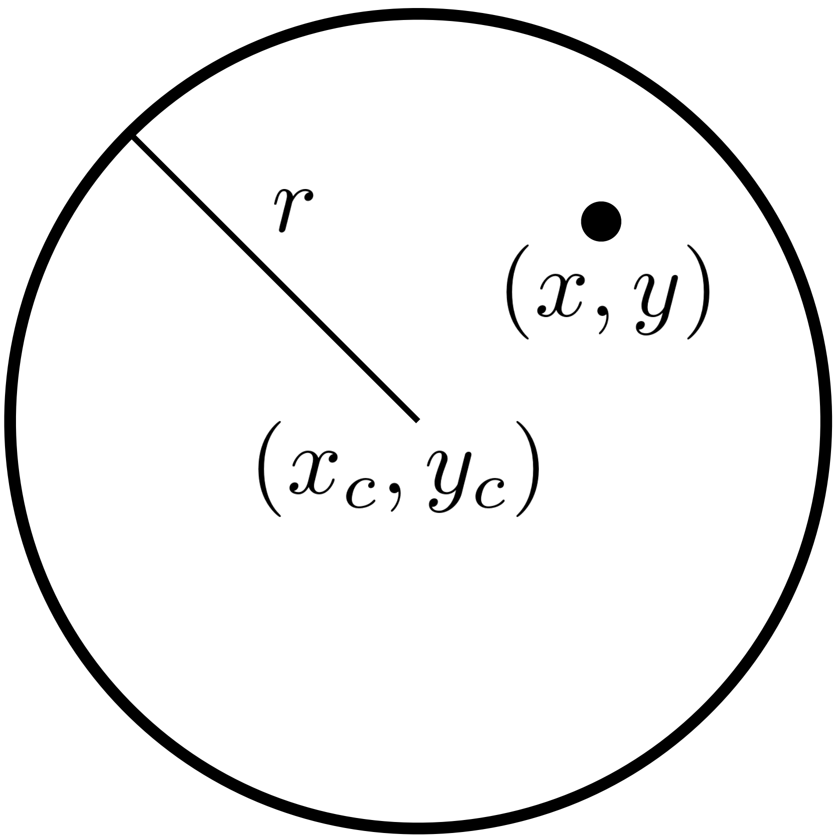
distance:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Circle-Point Queries



Circle-Point Queries



point inside circle if

$$\|(x, y) - (x_c, y_c)\| \leq r$$

$$\sqrt{(x - x_c)^2 + (y - y_c)^2} \leq r$$

Blast Zone?

Given:

- list of points ($\mathbf{x}_i$, $\mathbf{y}_i$)
- epicenter ($\mathbf{x}_c$, $\mathbf{y}_c$)
- radius $\mathbf{r}$

All inputs are ints between -2^{30} and 2^{30}

O(points) solution: loop over all points, and check if

$$\sqrt{(x_i - x_c)^2 + (y_i - y_c)^2} \leq r$$

Geometry Algorithms

Key challenges:

1. Remembering (or deriving) formulas

2. Dealing with precision issues

Two Kinds of Geometry Problems

Exact Arithmetic	Floating Point
	• Inputs are doubles • Problem statement says "answer is accepted if it is within [tolerance]" • Problem statement says "answer will not change if inputs are slightly perturbed"

Two Kinds of Geometry Problems

Exact Arithmetic

- Pretty much **any** other situation
- Problem statements with “round the answer to **n** digits” are **not** safe for floating point!

Floating Point

- Inputs are doubles
- Problem statement says “answer is accepted if it is within [tolerance]”
- Problem statement says “answer will not change if inputs are slightly perturbed”

Two Kinds of Geometry Problems

Exact Arithmetic

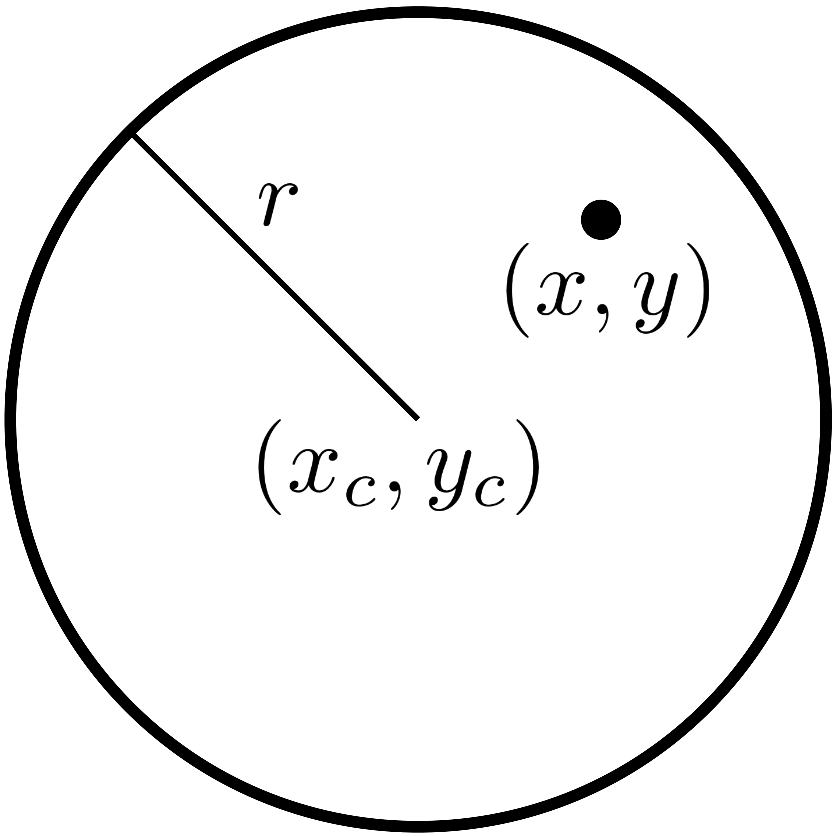
- Pretty much **any** other situation
- Problem statements with "round the answer to **n** digits" are **not** safe for floating point!

Floating Point

- Inputs are doubles
- Problem statement says "answer is accepted if it is within [tolerance]"
- Problem statement says "answer will not change if inputs are slightly perturbed"

Incorrectly classifying the type of problem is a **fatal noob trap!!**

Circle-Point Queries



point inside circle if

$$\|(x, y) - (x_c, y_c)\| \leq r$$

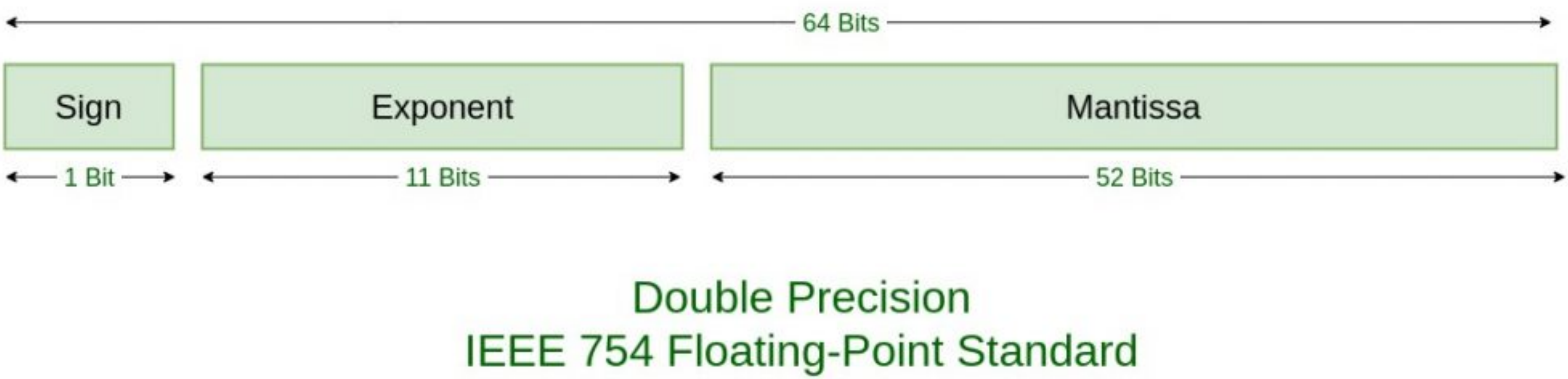
$$\sqrt{(x - x_c)^2 + (y - y_c)^2} \leq r$$

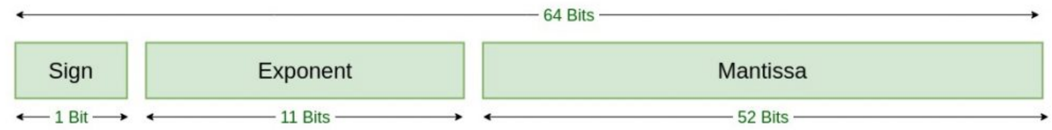
$$(x - x_c)^2 + (y - y_c)^2 \leq r^2$$

Doubles or Ints?

IEEE floating point is complicated, but there are some basics you should know

Structure of a **double**:

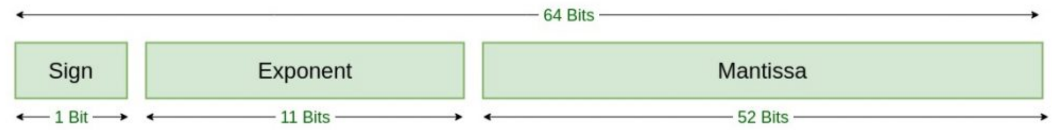




IEEE Floating Point

Double Precision
IEEE 754 Floating-Point Standard

Integers up to 2^{52} (about 15 decimal digits) can be **exactly** represented

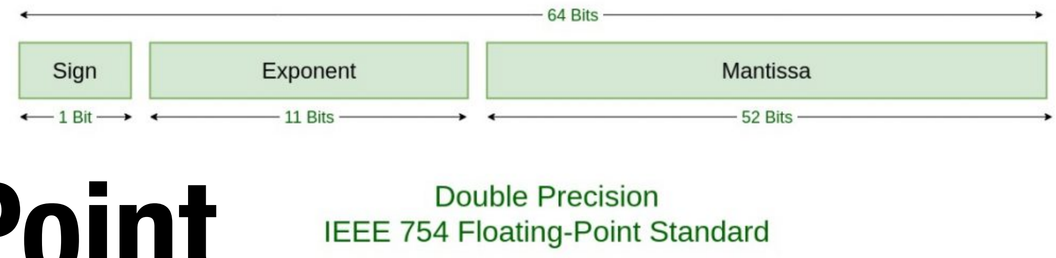


Double Precision
IEEE 754 Floating-Point Standard

IEEE Floating Point

Integers up to 2^{52} (about 15 decimal digits) can be **exactly** represented

Other numbers have up to 52 **bits of precision**. This goes down as errors accumulate in intermediate calculations



IEEE Floating Point

Integers up to 2^{52} (about 15 decimal digits) can be **exactly** represented

Other numbers have up to 52 **bits of precision**. This goes down as errors accumulate in intermediate calculations

When in doubt, use exact arithmetic. (And **never** use **float**s…)

Blast Zone? Take II

Given:

- list of points ($\mathbf{x}_i$, $\mathbf{y}_i$)
- epicenter ($\mathbf{x}_c$, $\mathbf{y}_c$)
- radius $\mathbf{r}$

All inputs are ints between -2^{30} and 2^{30}

O(points) solution: loop over all points, and check if

$$(x_i - x_c)^2 + (y_i - y_c)^2 \leq r^2$$

ints or doubles?

Geometry Algorithms

Key challenges:

1. Remembering (or deriving) formulas

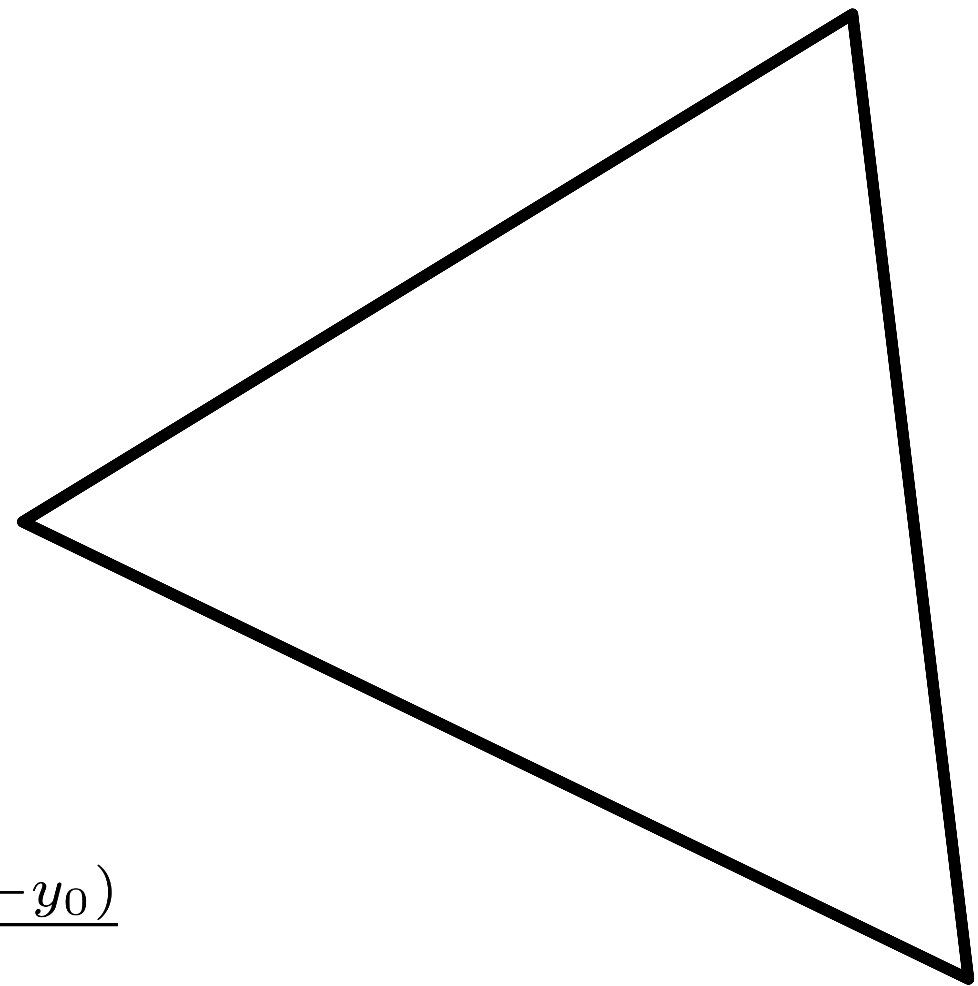
2. Dealing with precision issues

3. Dealing with potential overflow

Triangle Area

Signed area:

$$\frac{1}{2}\det\begin{bmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{bmatrix}$$



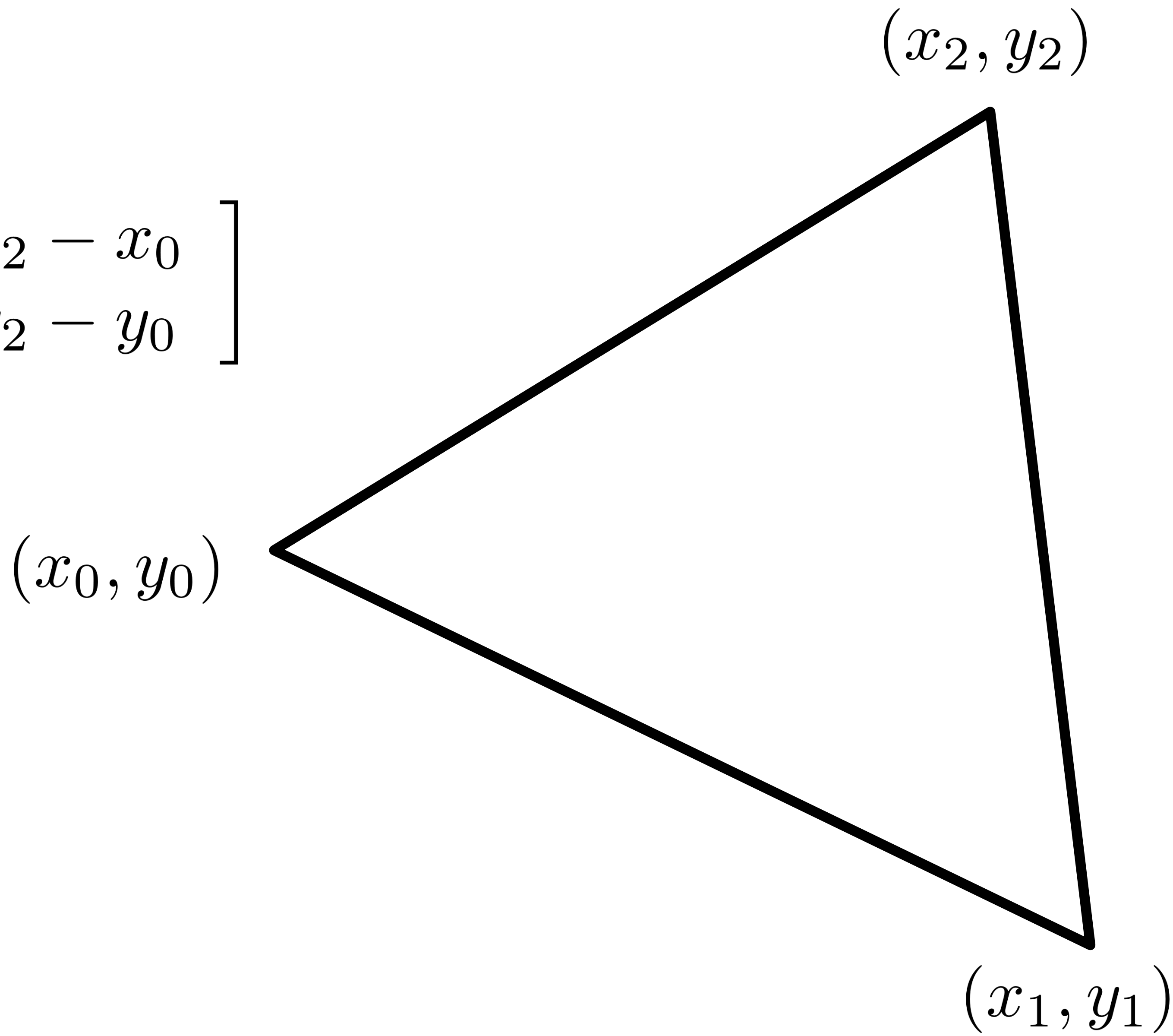
$$= \frac{(x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)}{2}$$

Triangle Area

Signed area:

$$\frac{1}{2}\det\begin{bmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{bmatrix}$$

Why **signed**?

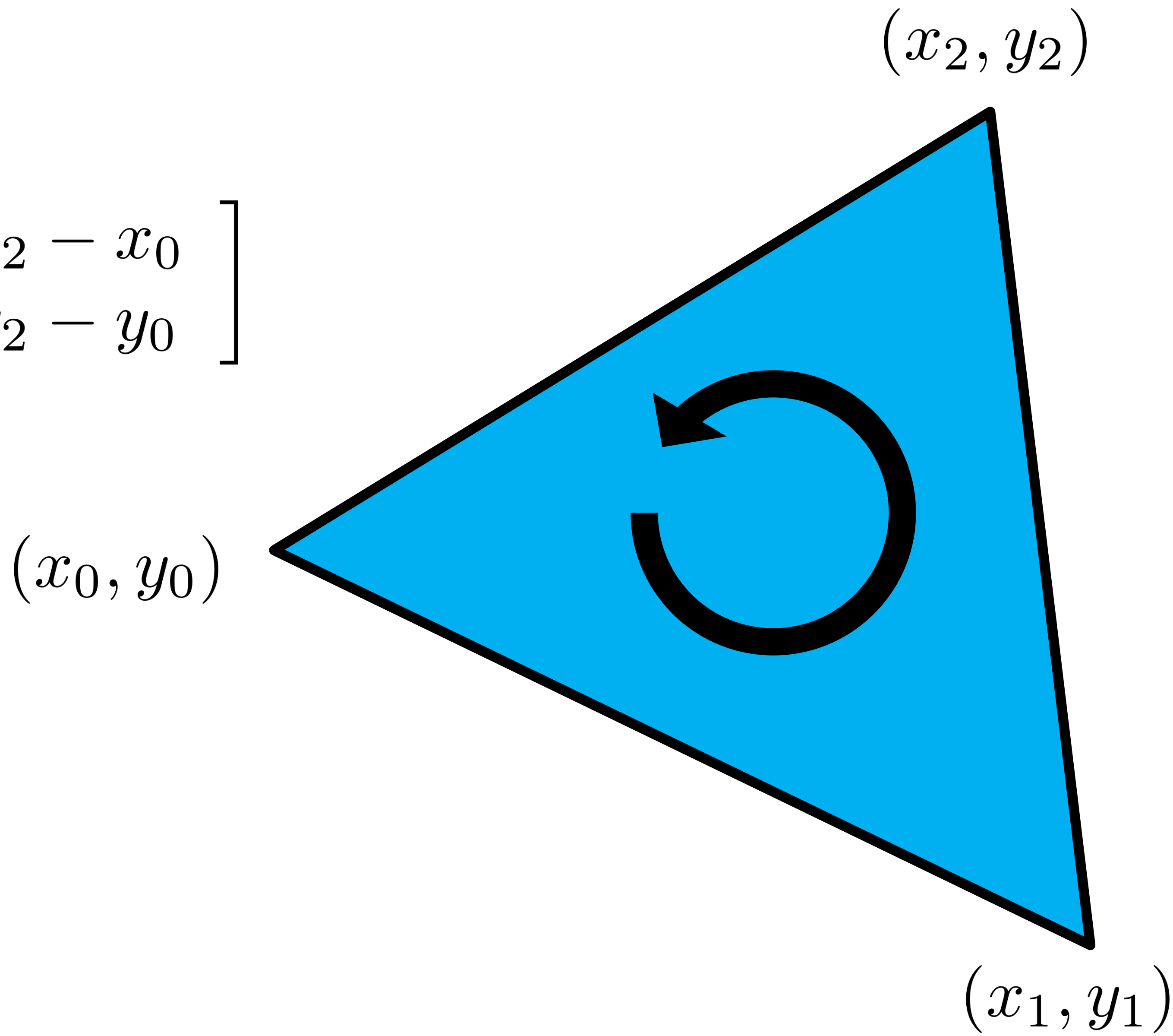


Triangle Area

Signed area:

$$\frac{1}{2}\det\begin{bmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{bmatrix}$$

Why **signed**?

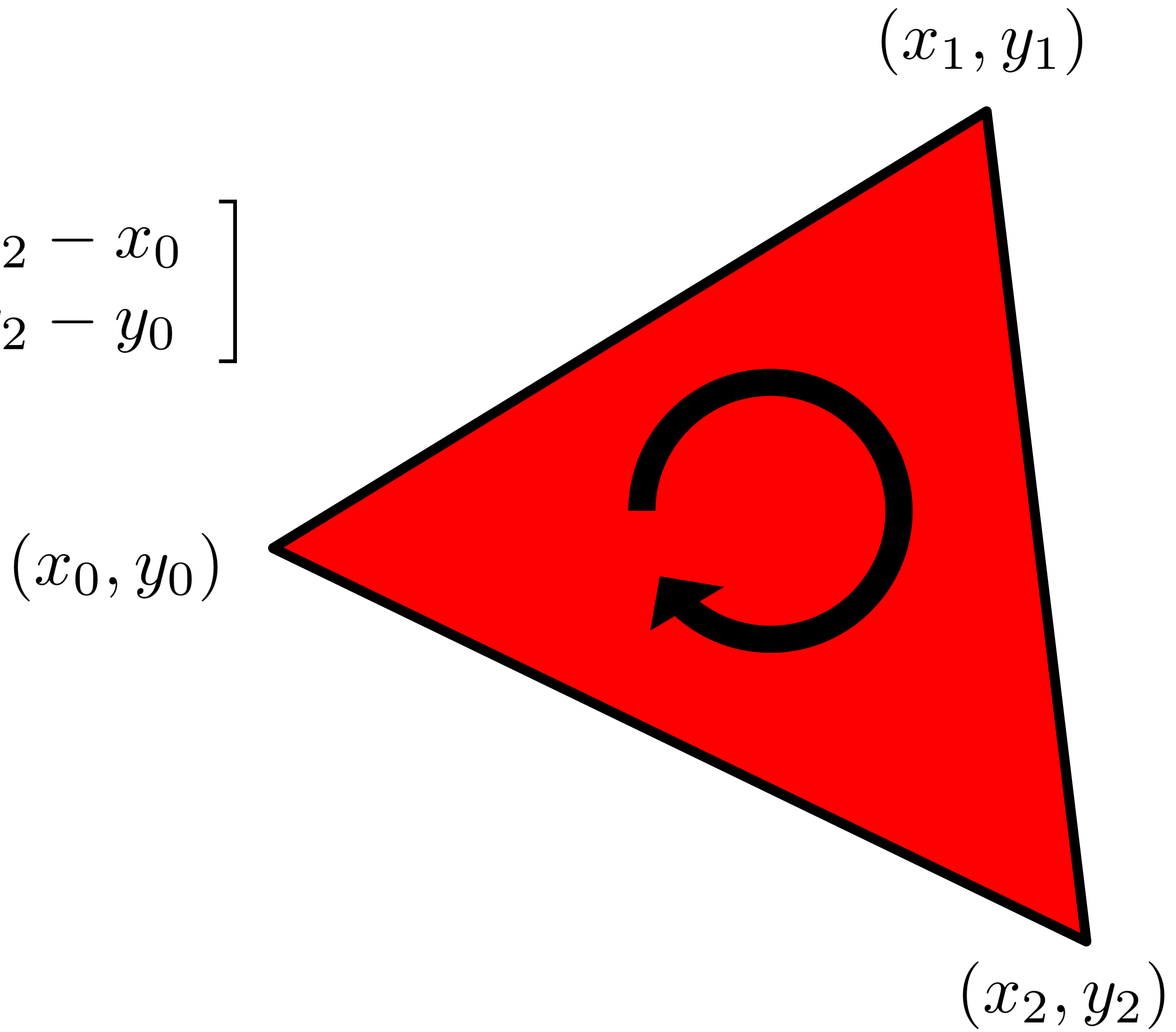


Triangle Area

Signed area:

$$\frac{1}{2}\det\begin{bmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{bmatrix}$$

Why **signed**?

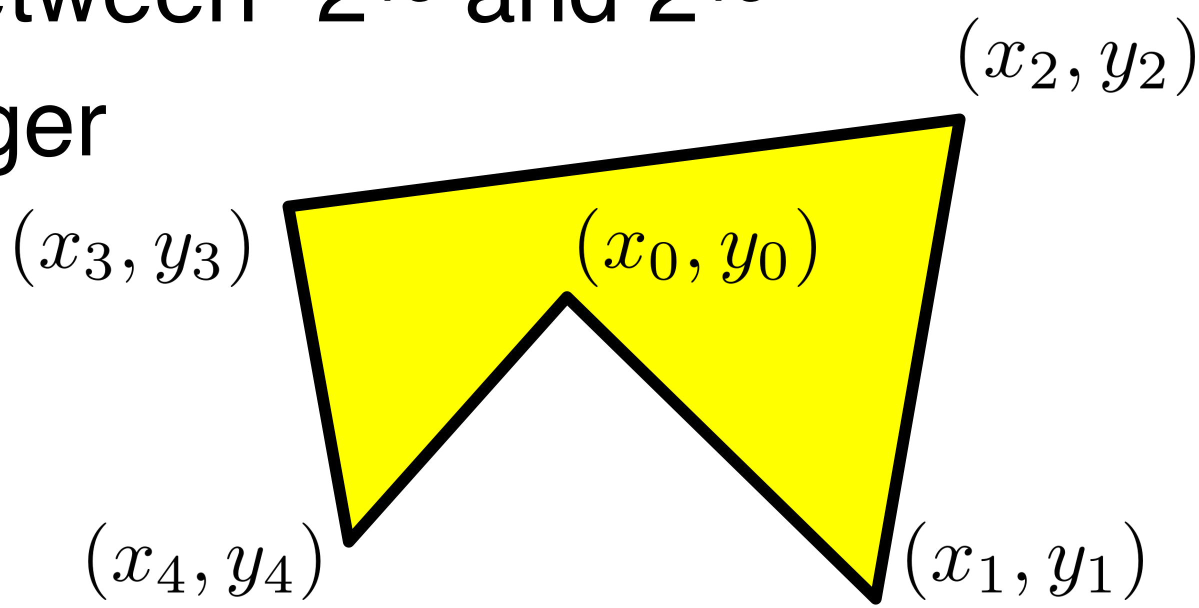


Polygon Area

Given a list of integer points in the plane, calculate area of polygon

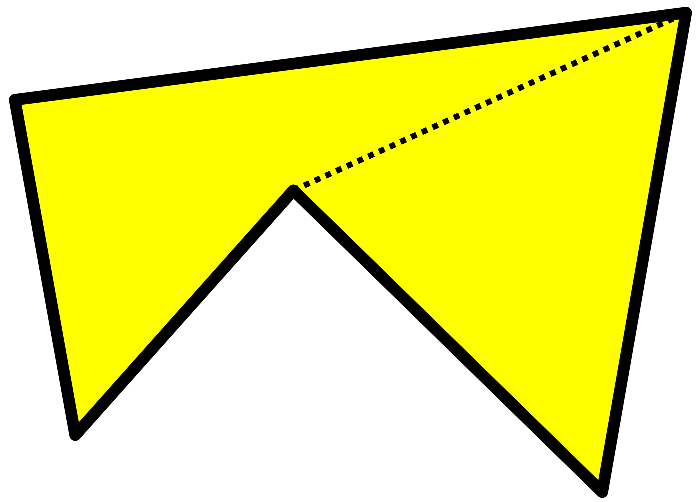
All input ints between -2^{15} and 2^{15}

Area is an integer



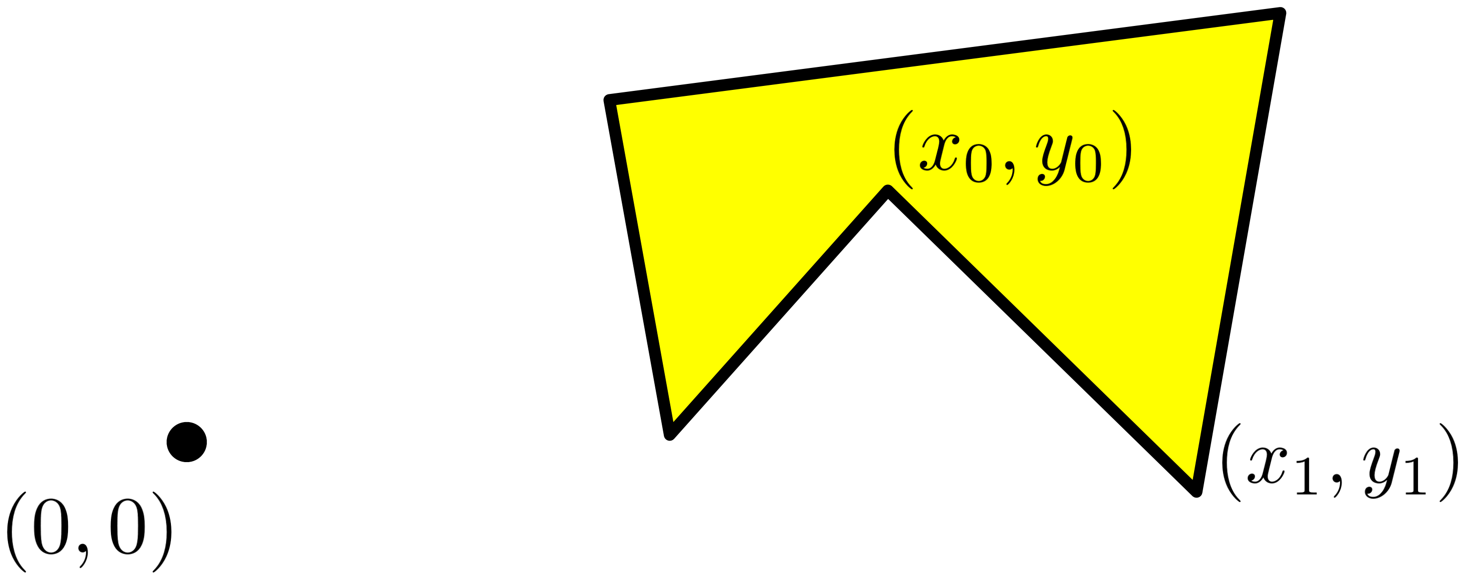
Polygon Area

The hard way: divide and conquer “ear cutting”



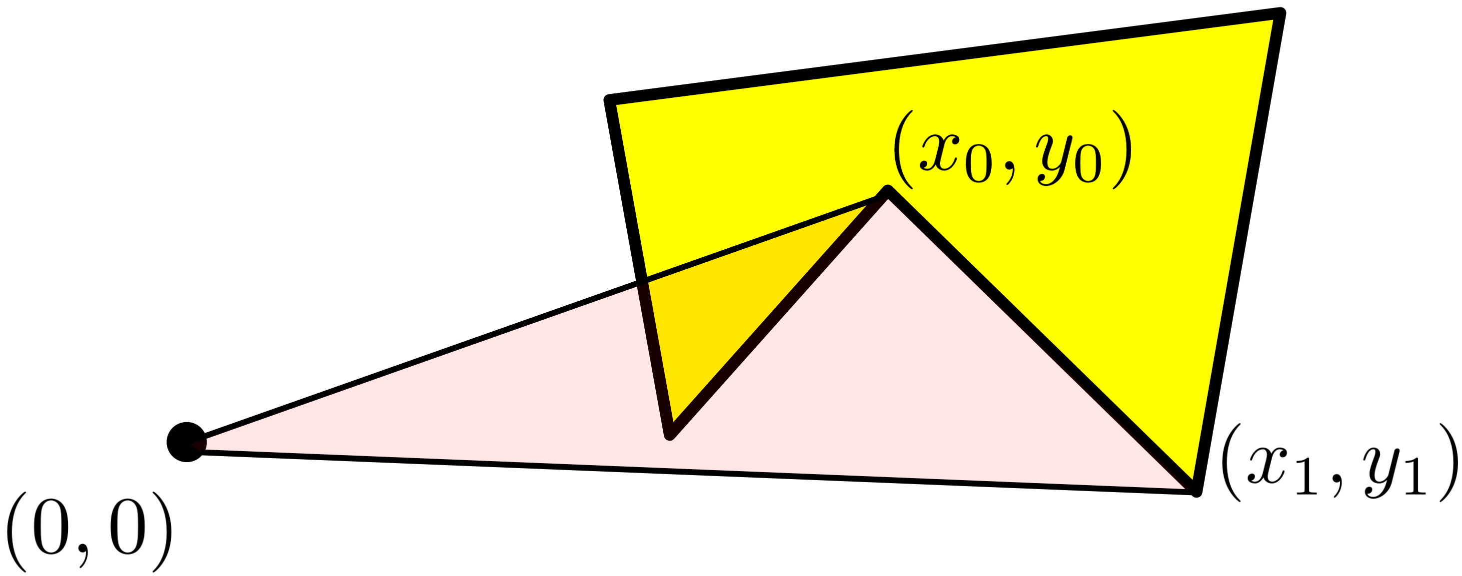
Polygon Area

The easy way:



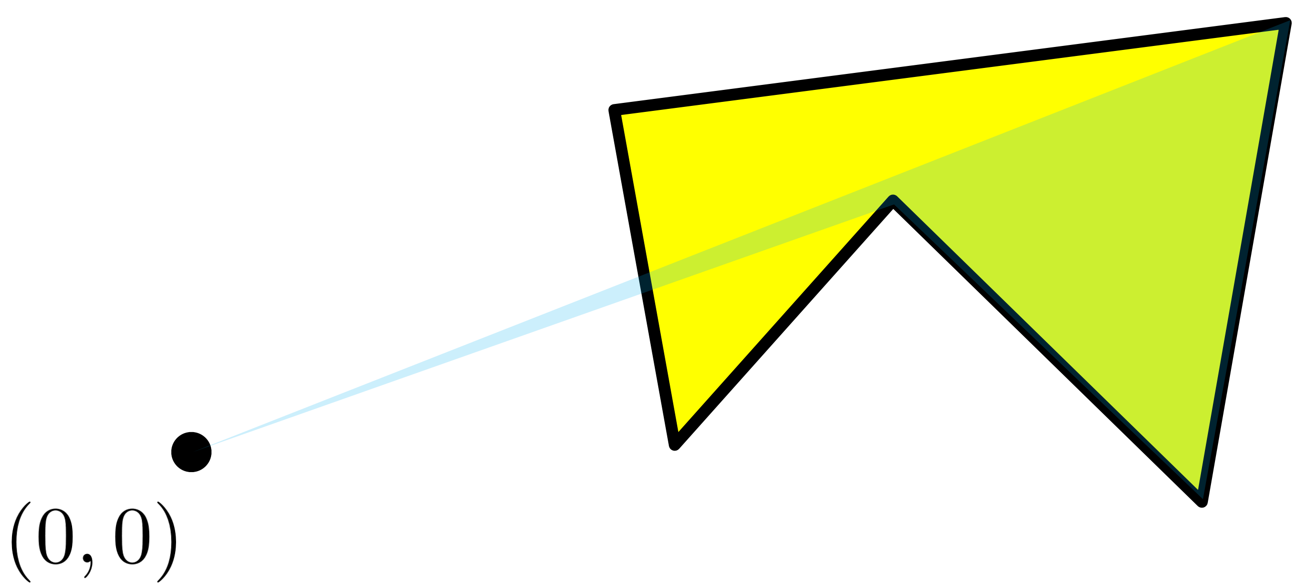
Polygon Area

The easy way:



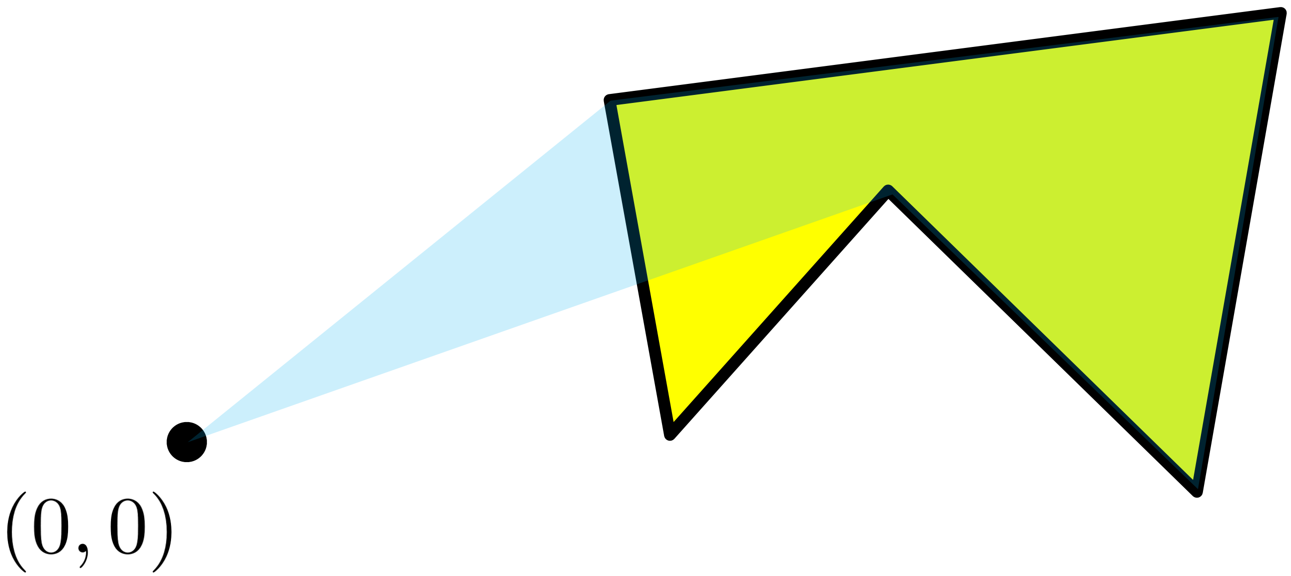
Polygon Area

The easy way:



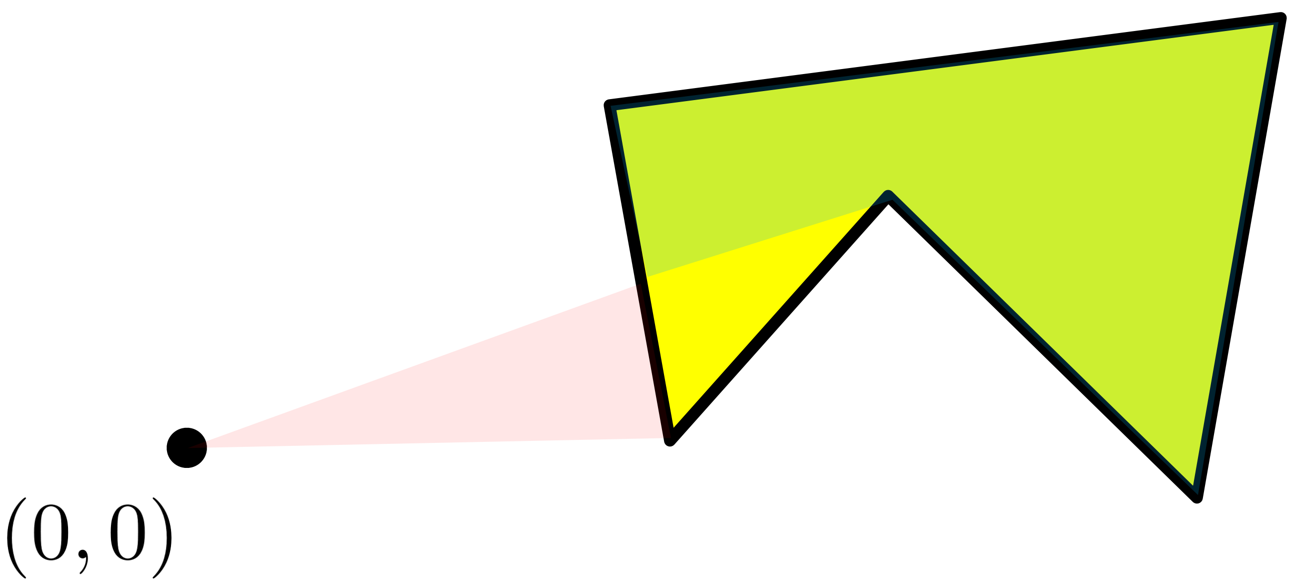
Polygon Area

The easy way:



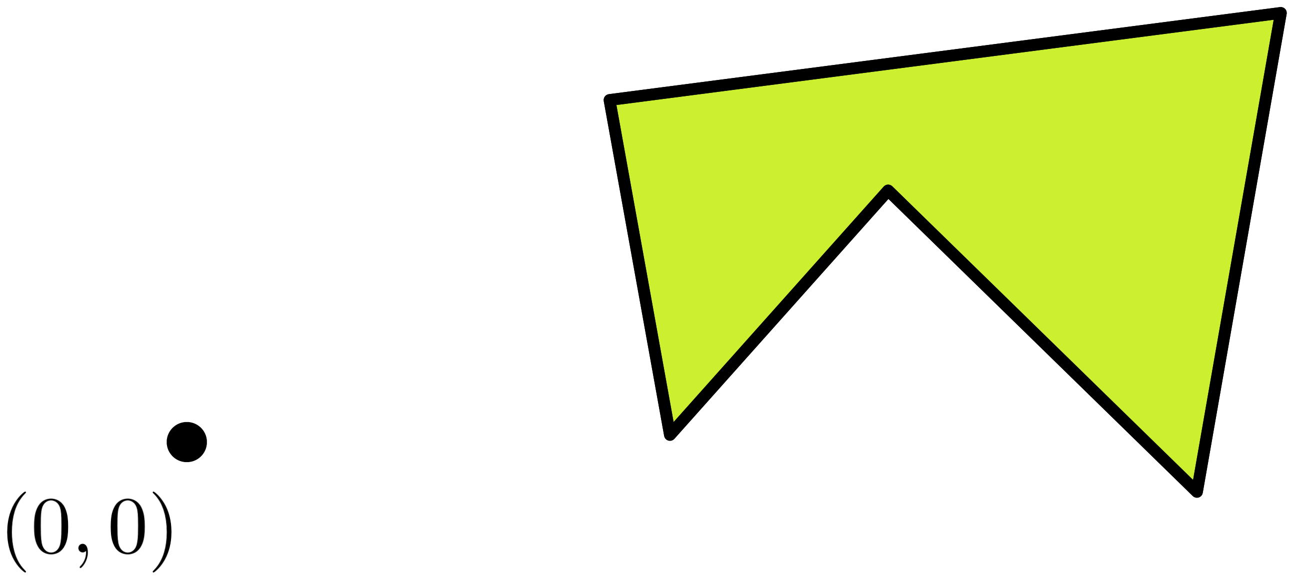
Polygon Area

The easy way:



Polygon Area

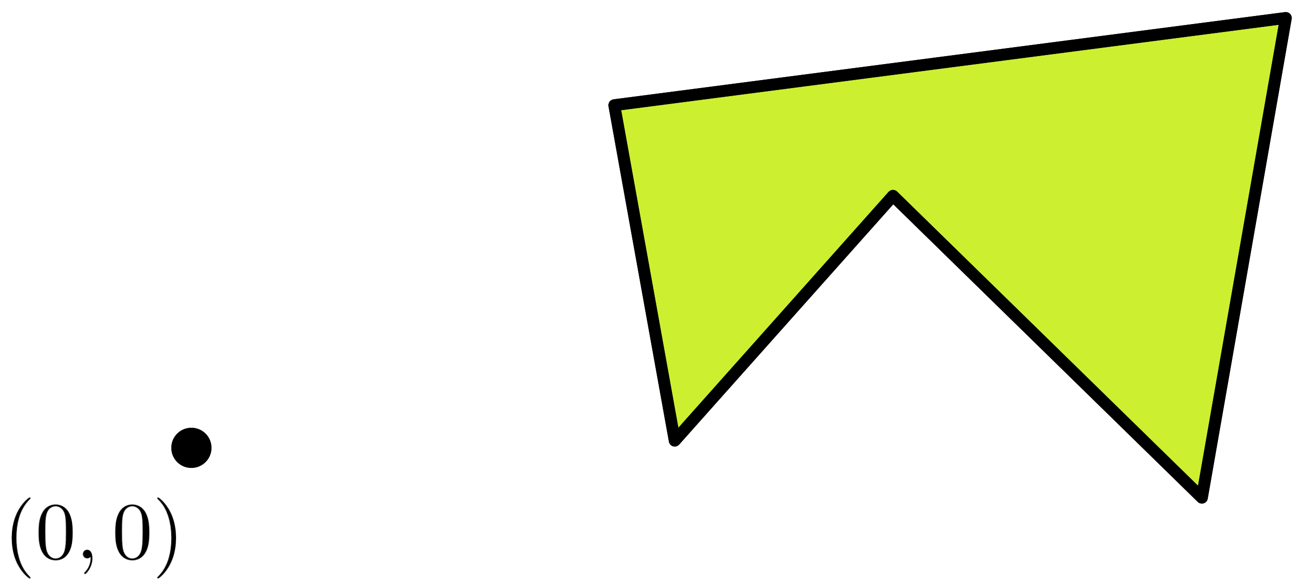
The easy way:



Polygon Area

The easy way: "shoelace formula"

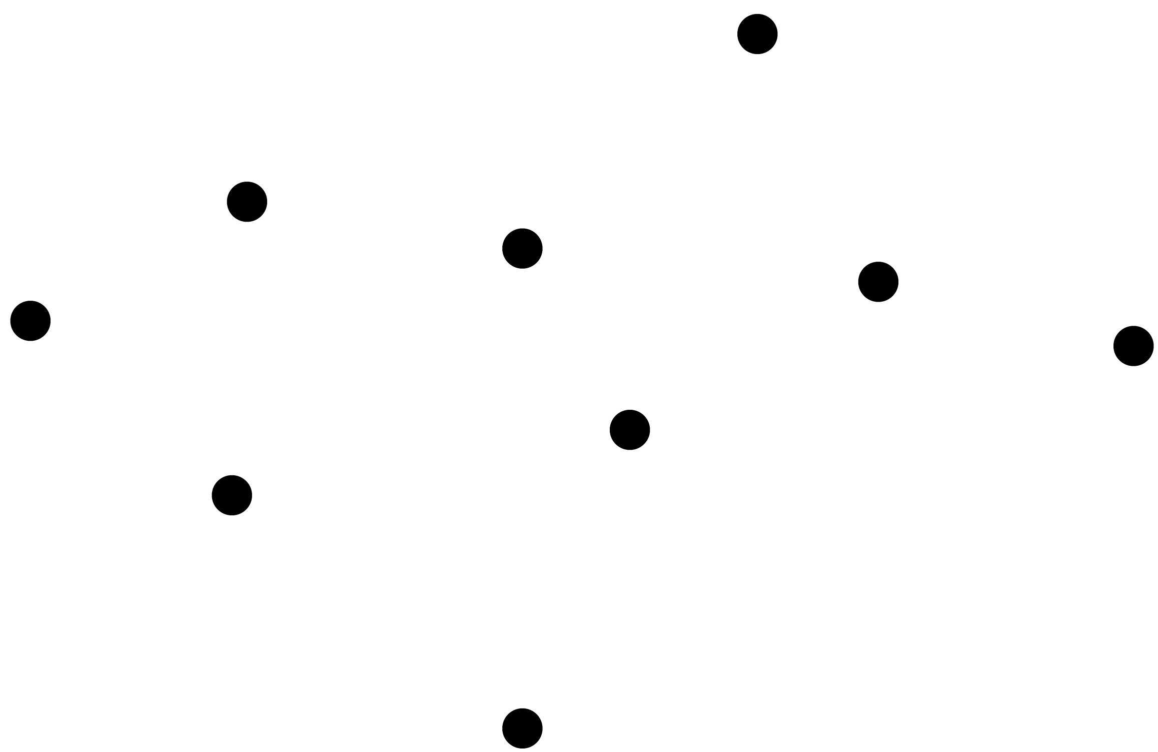
$$\frac{1}{2}\sum\left[(x_i - 0)(y_{i+1} - 0) - (x_{i+1} - 0)(y_i - 0)\right]$$



Convex Hull

“Envelope” of set of points

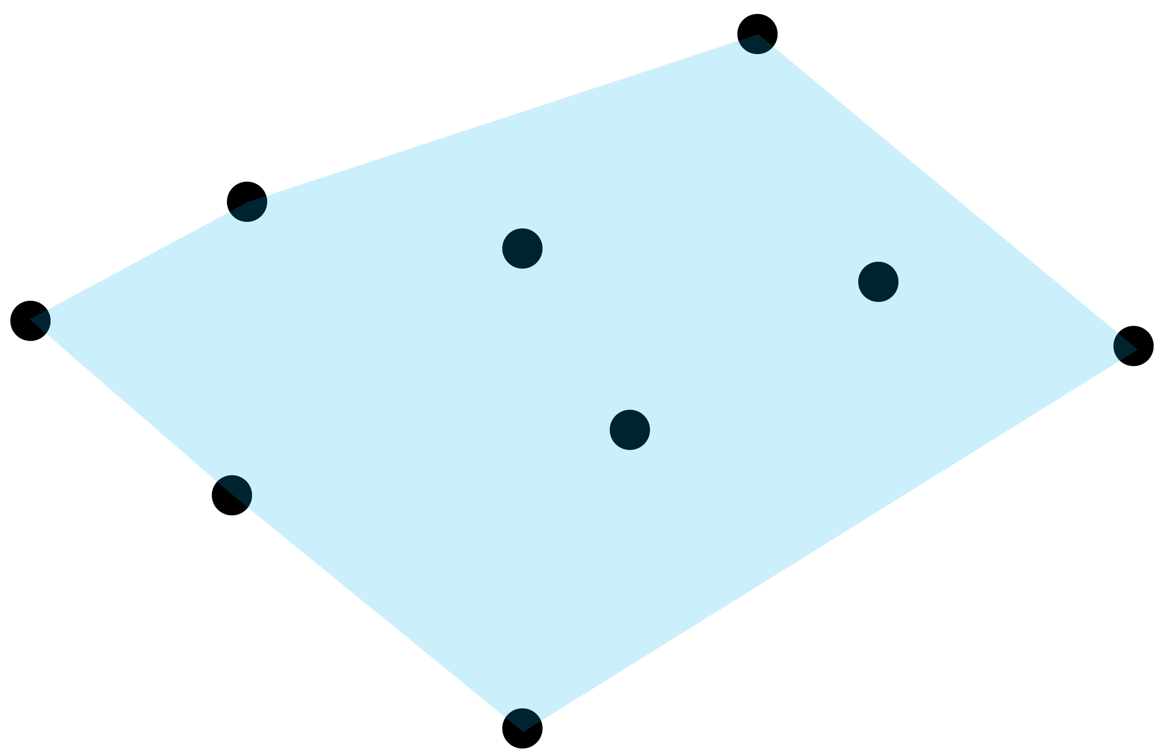
- rubber band analogy



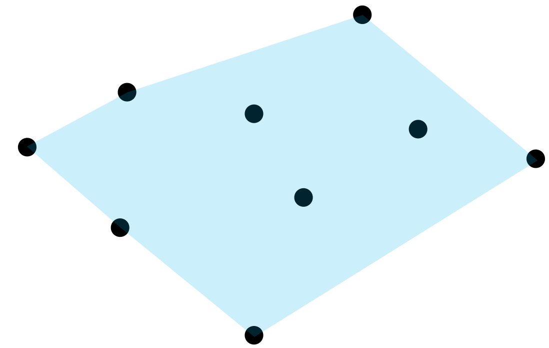
Convex Hull

"Envelope" of set of points

- rubber band analogy



Computing Convex Hull

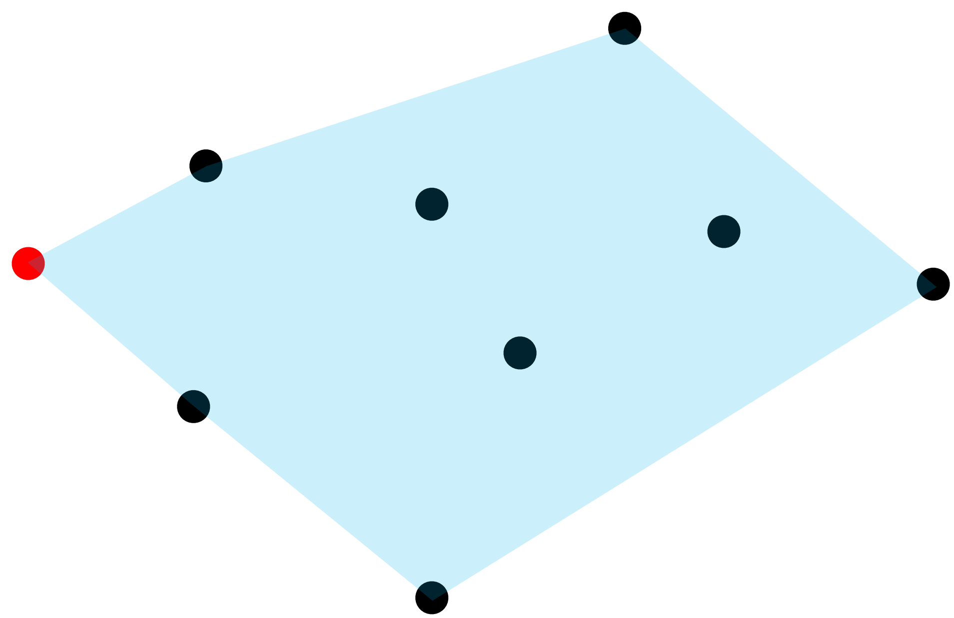


1. Find one point on boundary

2. Until we have complete polygon, walk counterclockwise around boundary

Computing Convex Hull

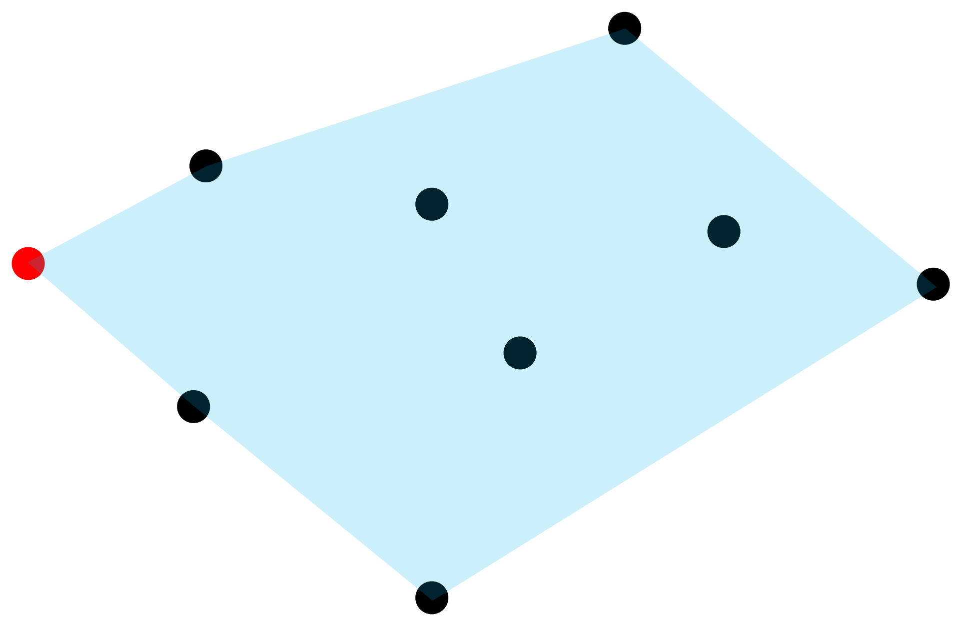
1. Find one point on boundary



Computing Convex Hull

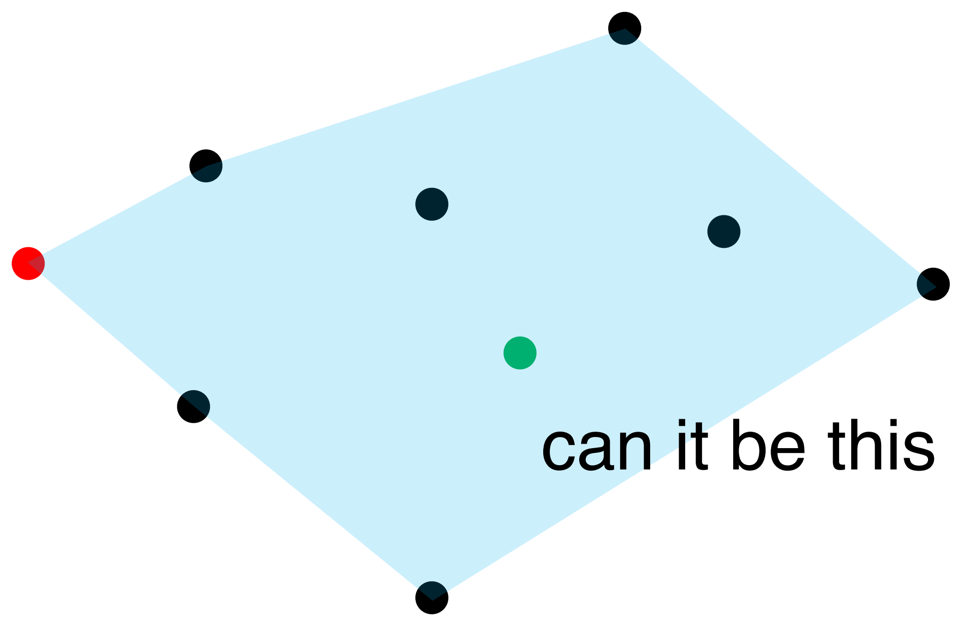
1. Find one point on boundary

- e.g. point with min x coord



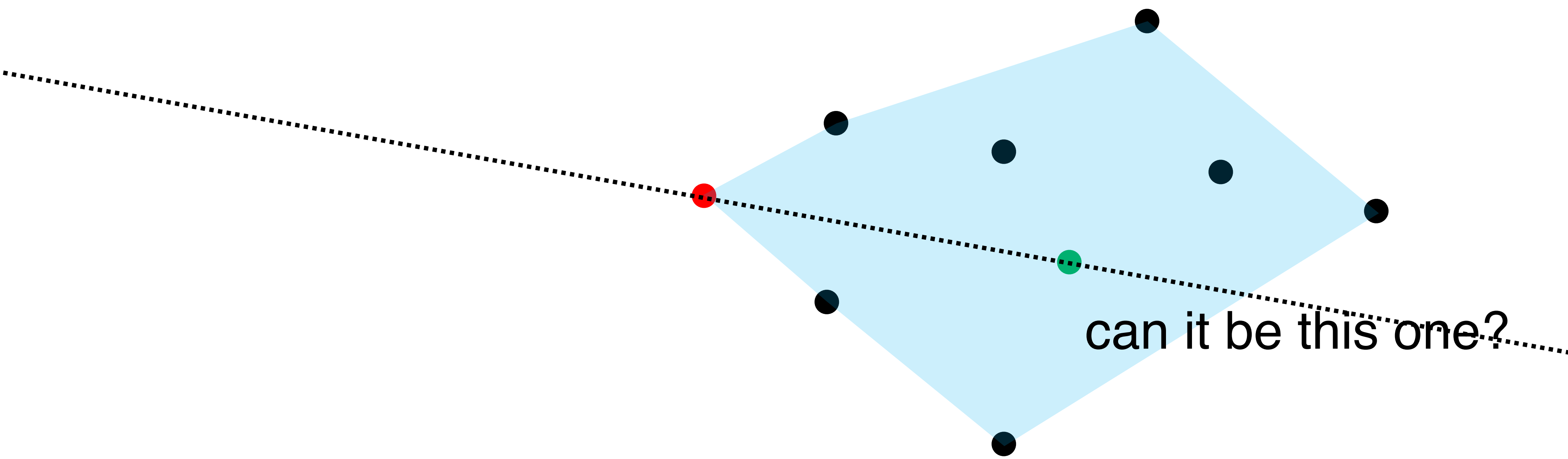
Computing Convex Hull

How to find next boundary point?



Computing Convex Hull

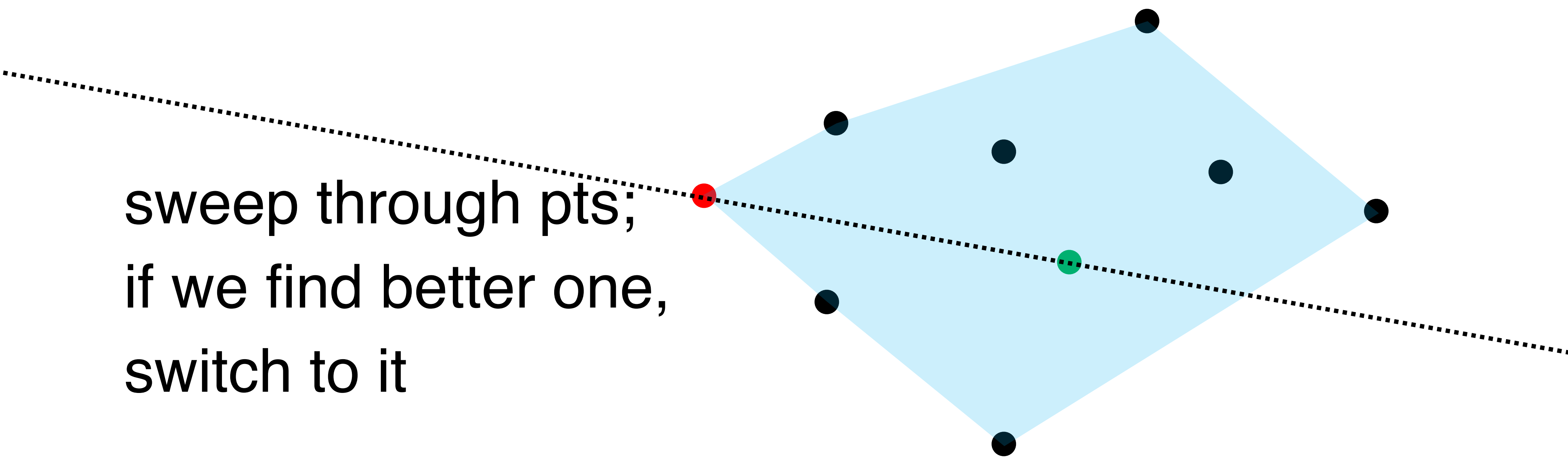
How to find next boundary point?



Computing Convex Hull

How to find next boundary point?

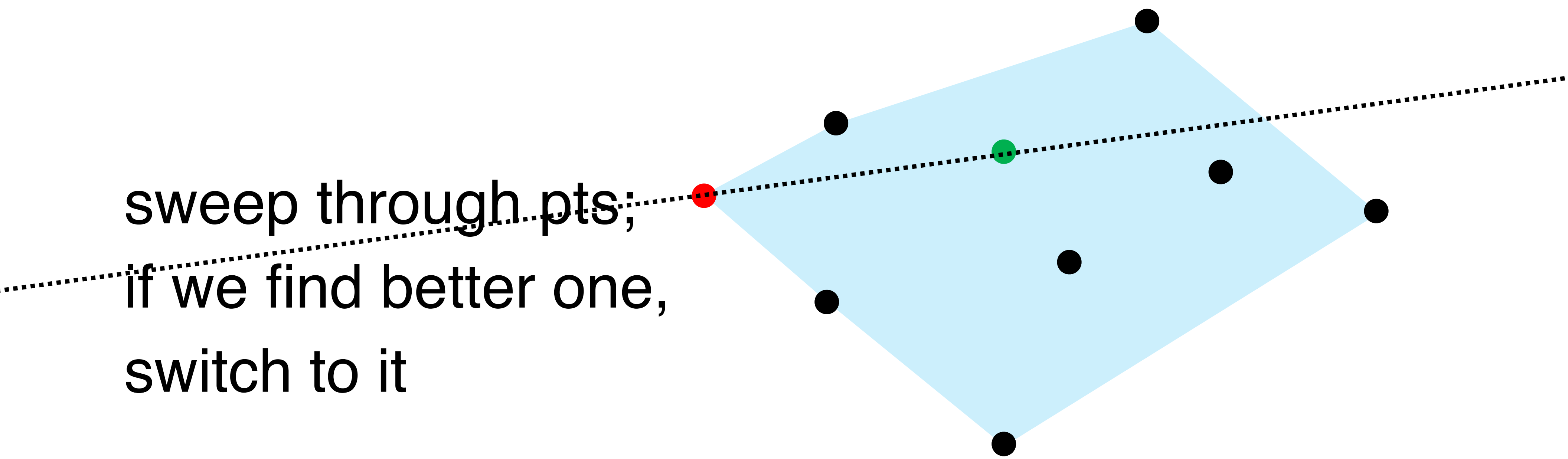
sweep through pts;
if we find better one,
switch to it



Computing Convex Hull

How to find next boundary point?

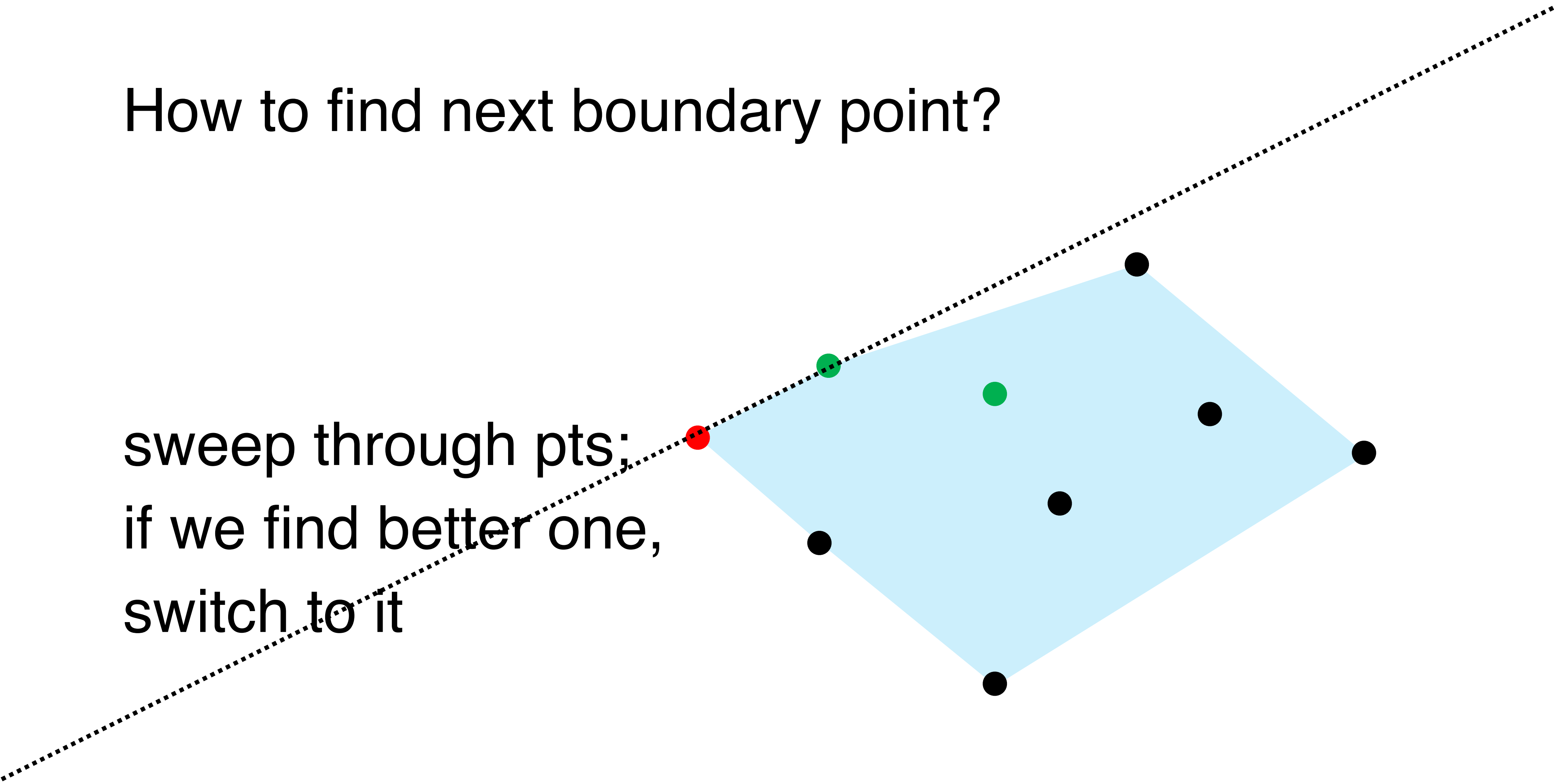
sweep through pts;
if we find better one,
switch to it



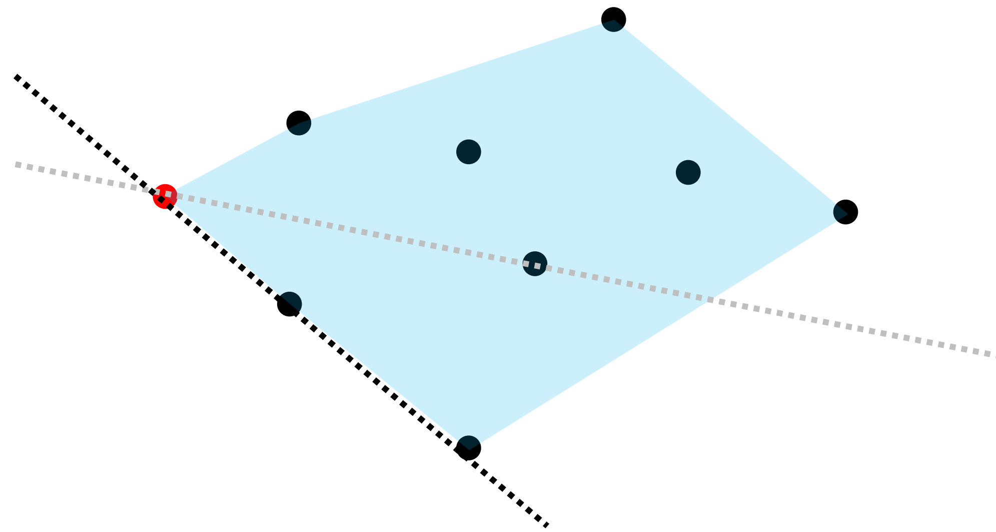
Computing Convex Hull

How to find next boundary point?

sweep through pts;
if we find better one,
switch to it



Jarvis Marching



Pros:

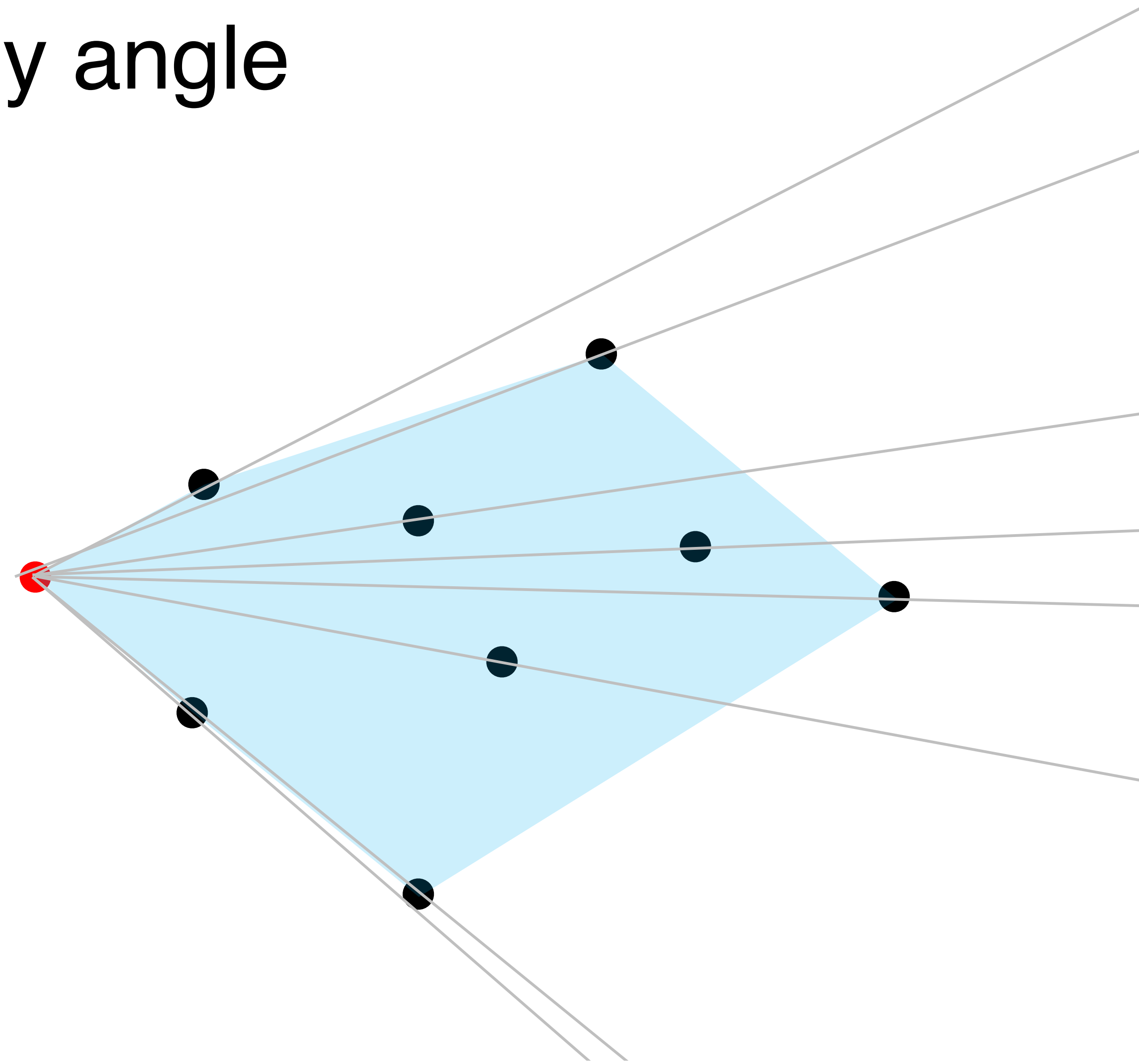
- Intuitive
- Easy to code

Cons:

- Doesn't work in 3D+
- O(n^2)

Graham Scan

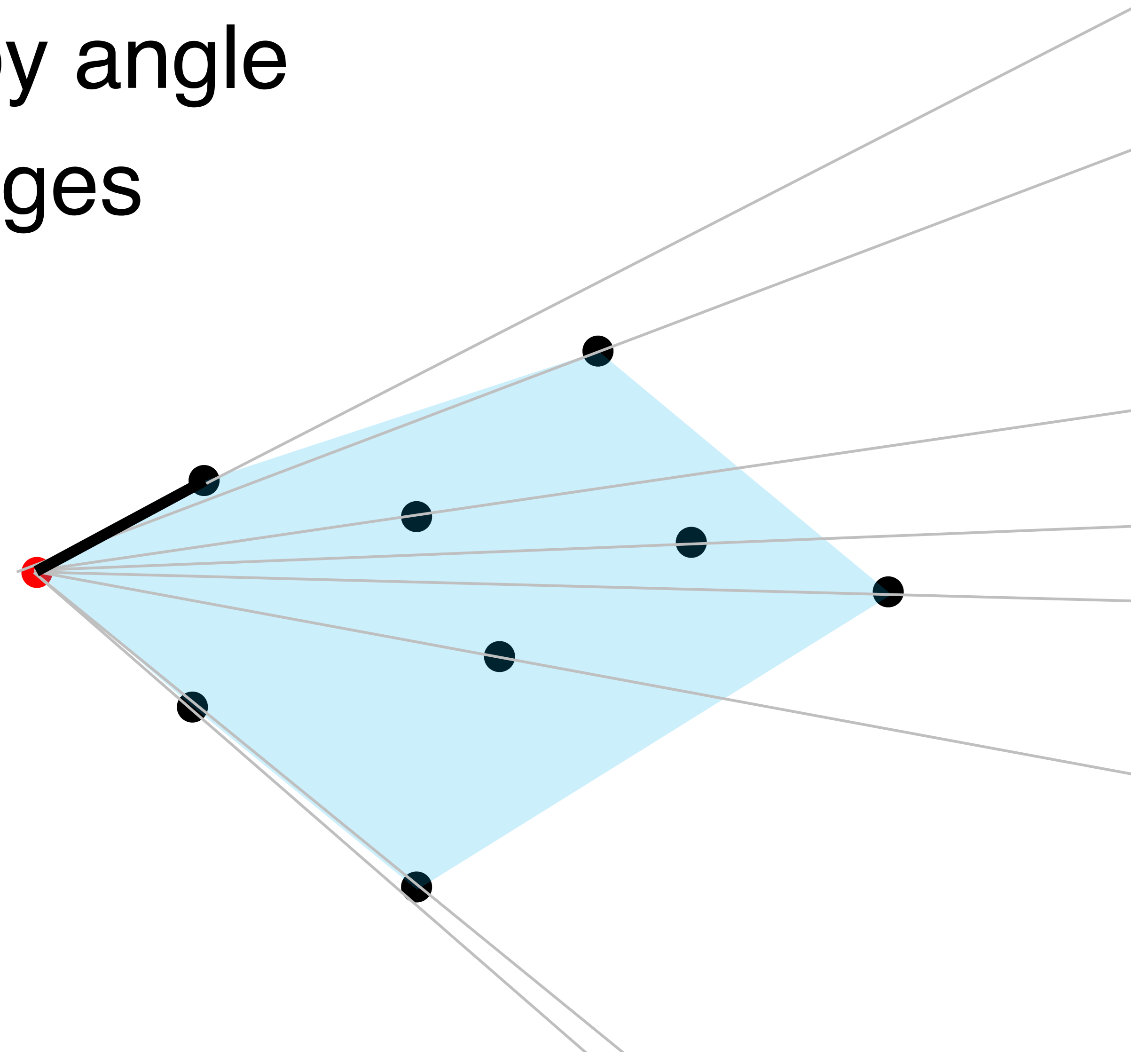
Idea: sort points by angle



Graham Scan

Idea: sort points by angle

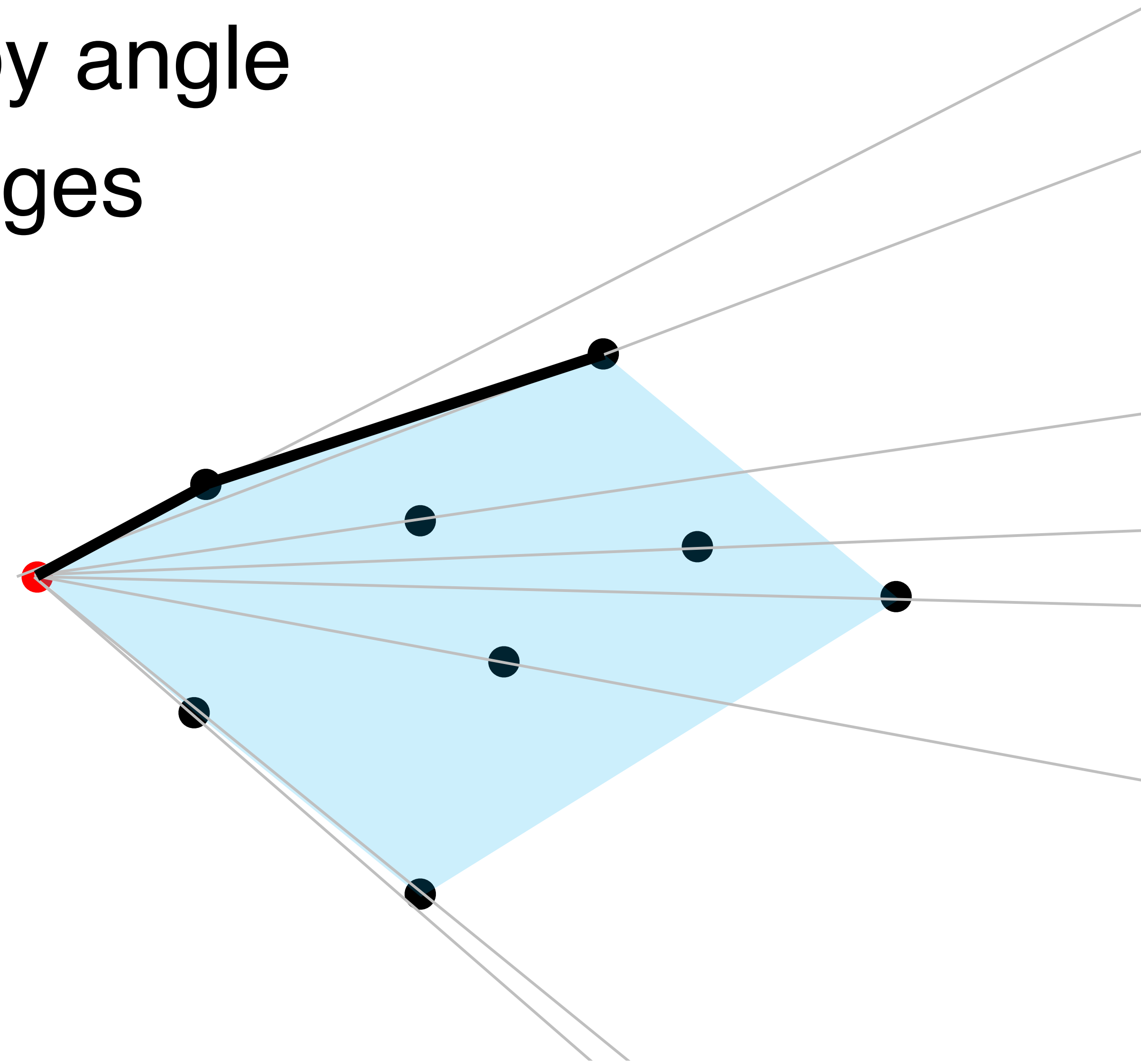
- Start adding edges



Graham Scan

Idea: sort points by angle

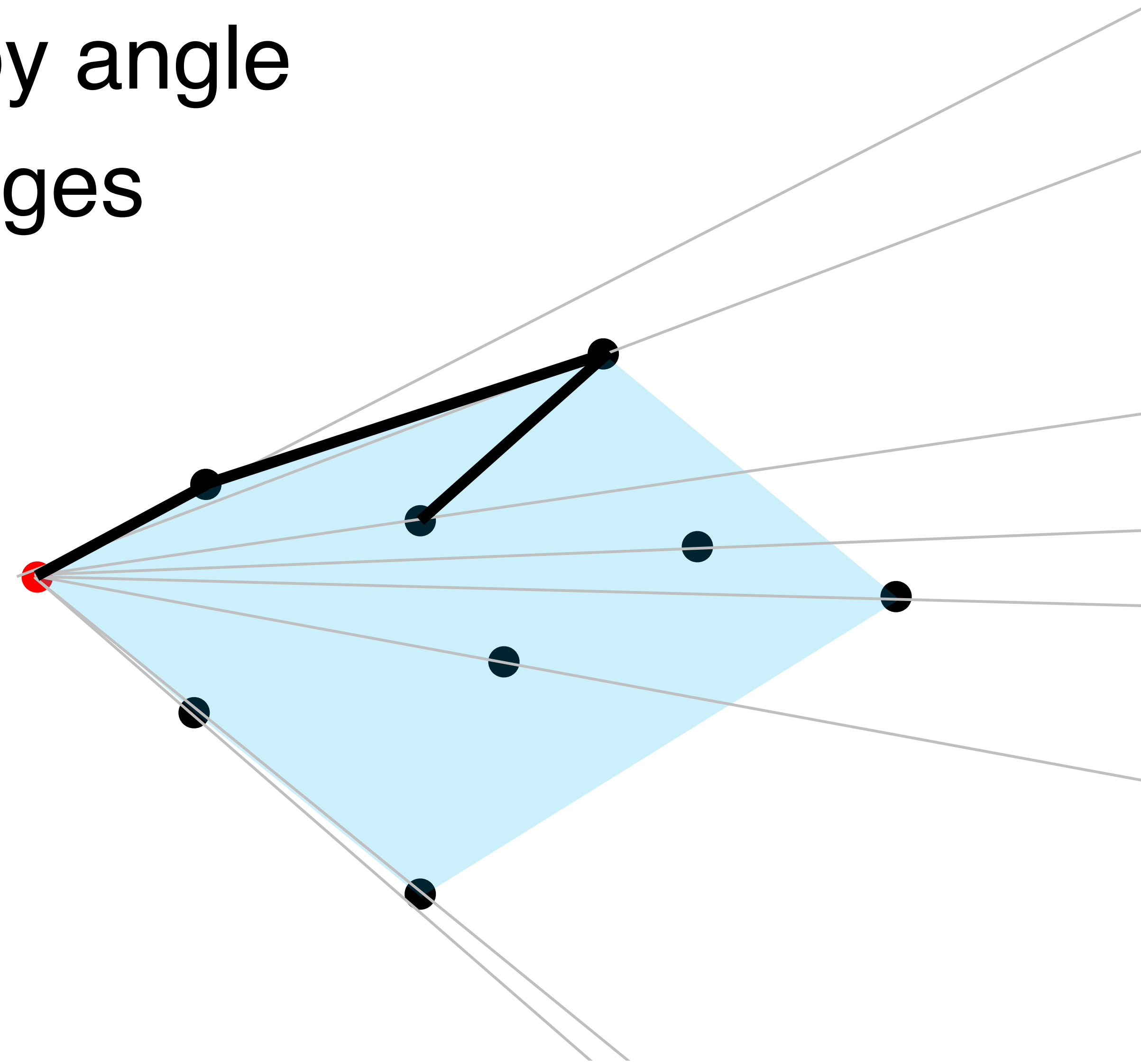
- Start adding edges



Graham Scan

Idea: sort points by angle

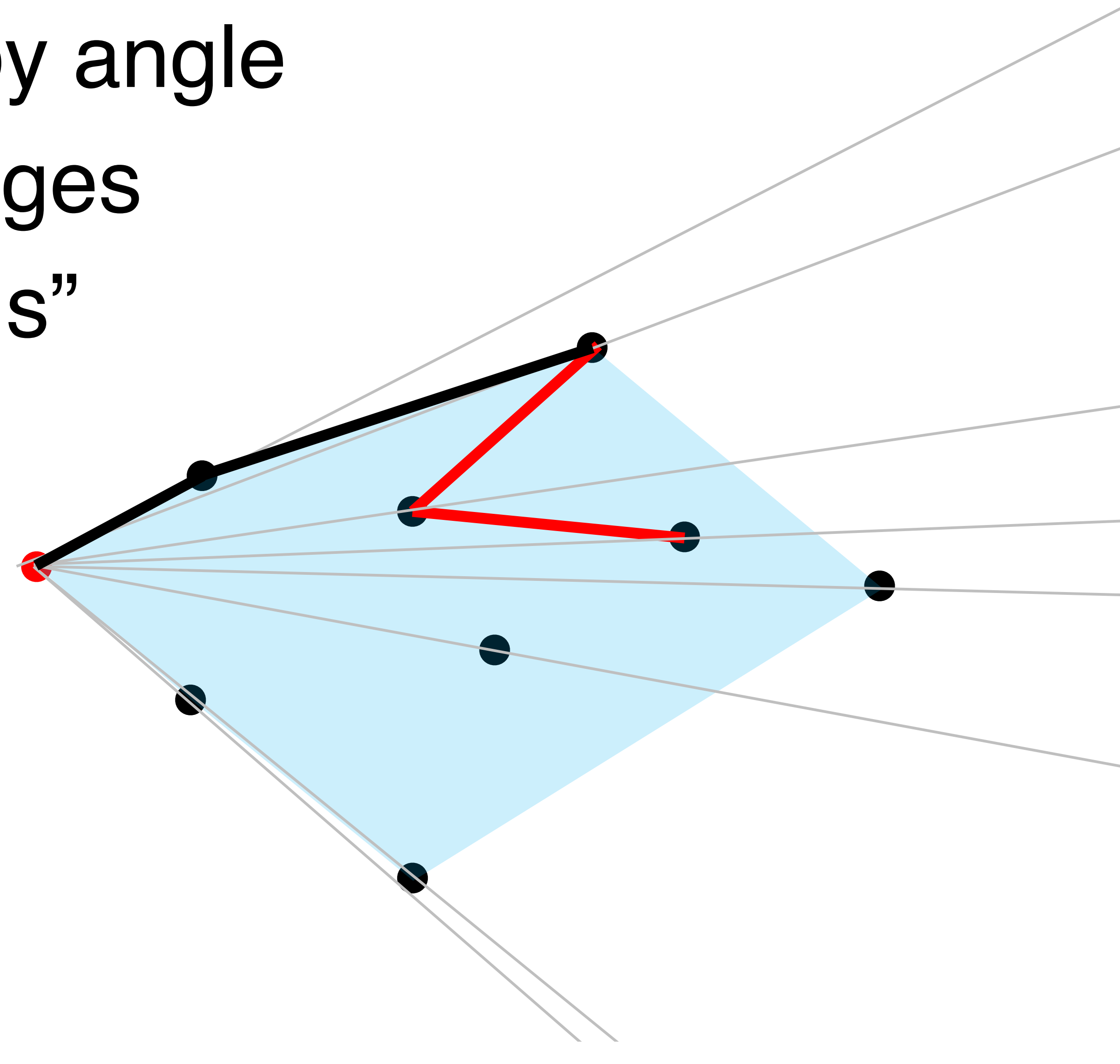
- Start adding edges



Graham Scan

Idea: sort points by angle

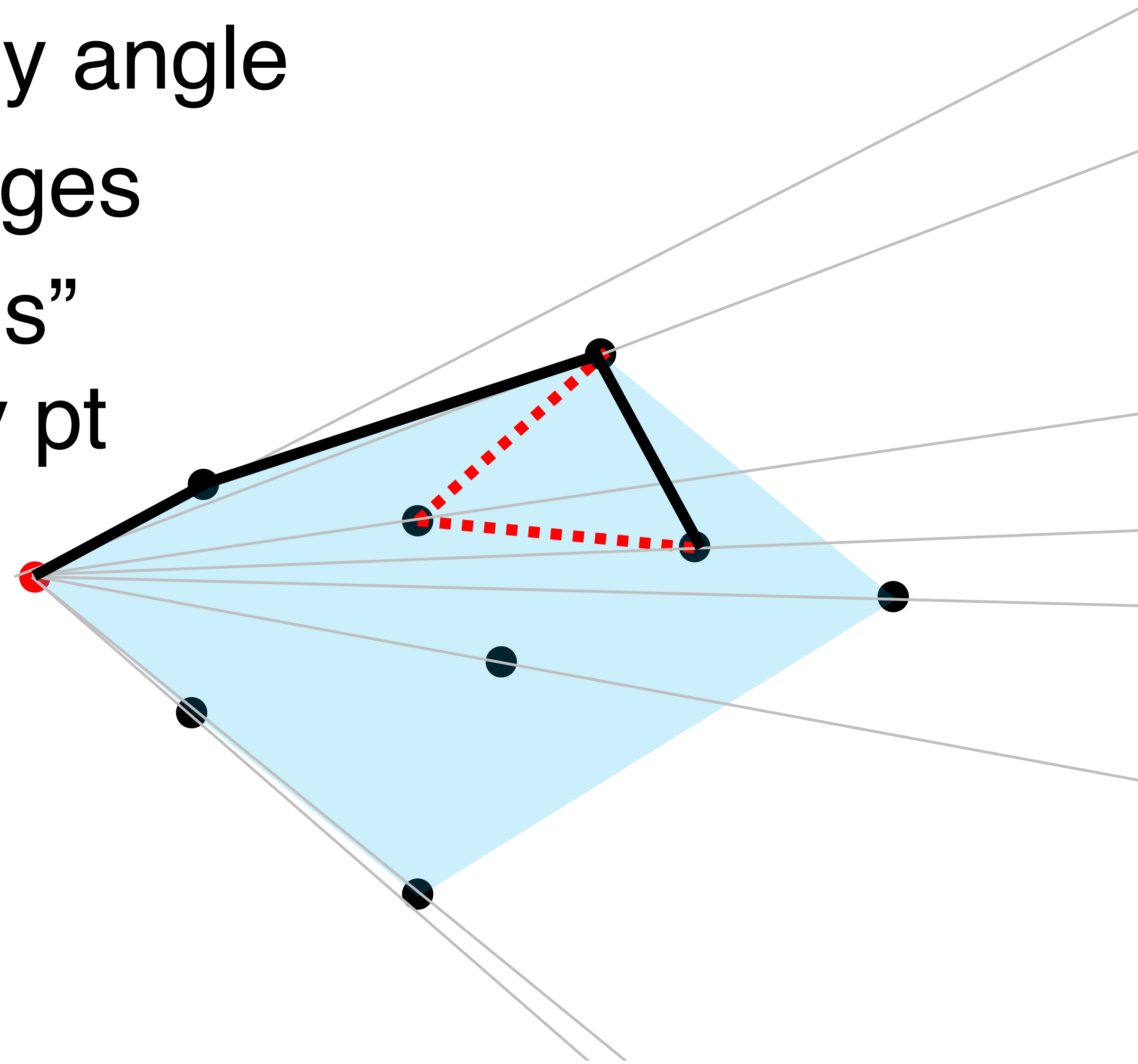
- Start adding edges
- Detect “left turns”



Graham Scan

Idea: sort points by angle

- Start adding edges
- Detect “left turns” and delete prev pt
- Check again for a left turn

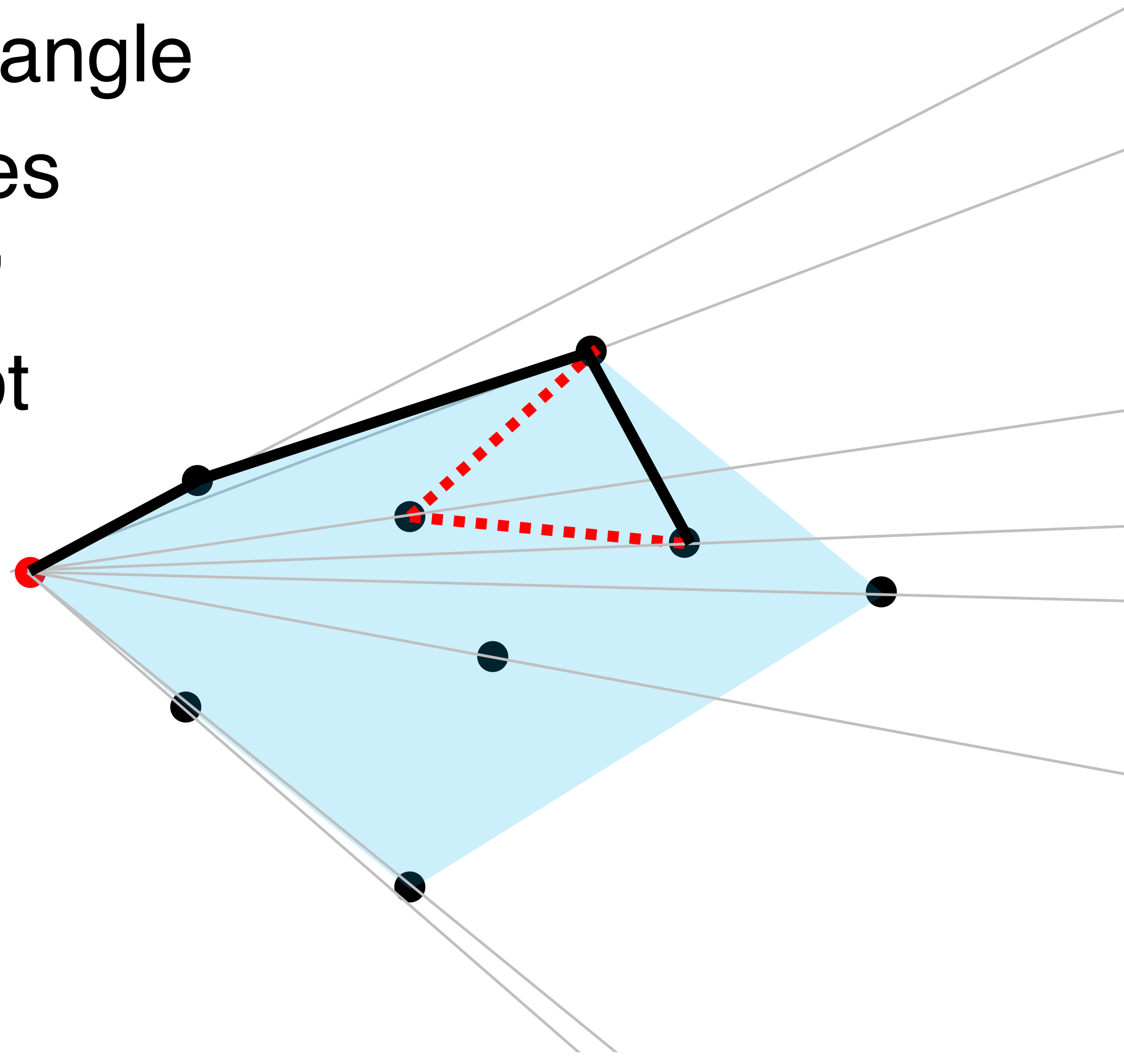


Graham Scan

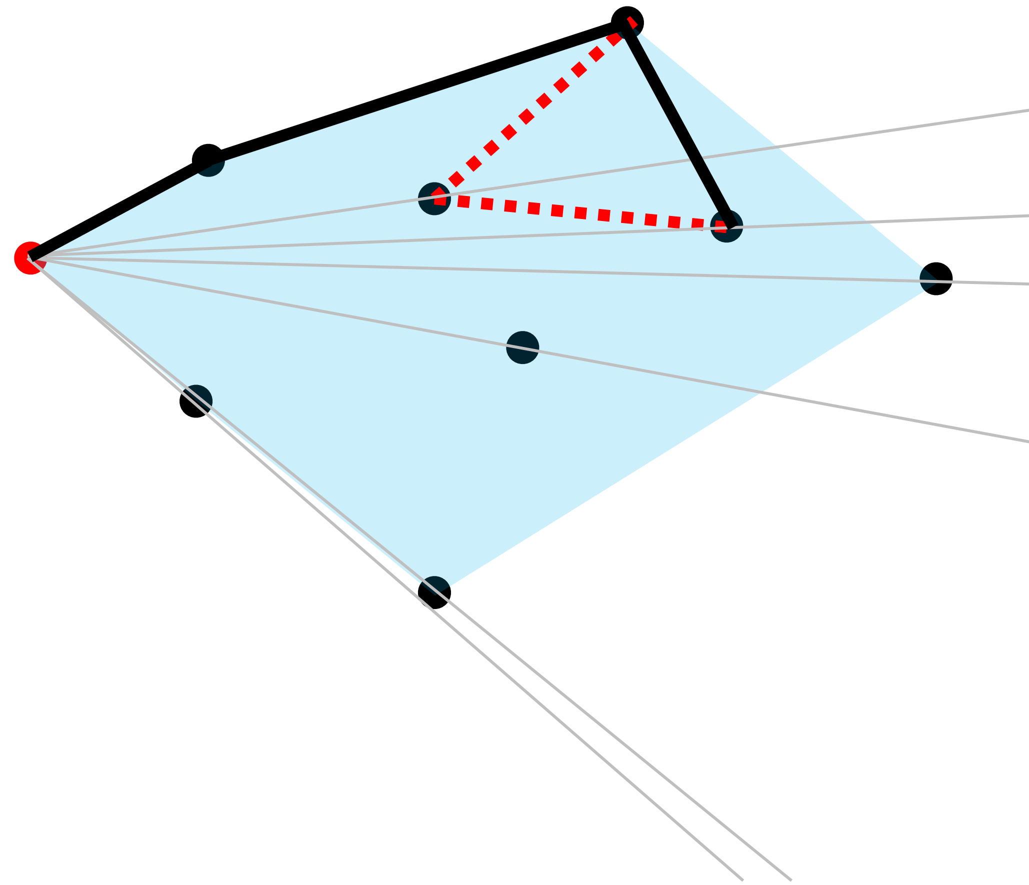
Idea: sort points by angle

- Start adding edges
- Detect “left turns” and delete prev pt
- Check again for a left turn

Now O(n log n)



Graham Scan



Pros:

- Fast (O(n log n))

Cons:

- Still doesn’t work in 3D+
- Numerical robustness issues if points are close to colinear