CS 164 & CS 266: Computational Geometry

Lecture 1

Coordinates, Primitives, and Convex Hulls

David Eppstein
University of California, Irvine

Fall Quarter, 2025

About the course

General Course Information

Instructor: David Eppstein

Teaching assistant: Cole Groen

Resources: Course info: `https://www.ics.uci.edu/~eppstein/164`

Ed Discussion (online forum for course-related questions)

Gradescope (for exam scores)

Coursework

Weekly problem sets (not graded!), two midterms, and final exam

We will provide problem set solutions (a week later)

Exams are not cumulative
(not intended to test you on earlier material but may assume it as background)

They will be short-answer, in-person, closed book and closed notes.

For CS266 students: Weekly reading: a research paper related to each week's topics

You should understand what the paper is claiming to do (and may be tested on this)

Undergrads are welcome to read these too! (But you will not be tested on them.)

Why are the problem sets not graded?

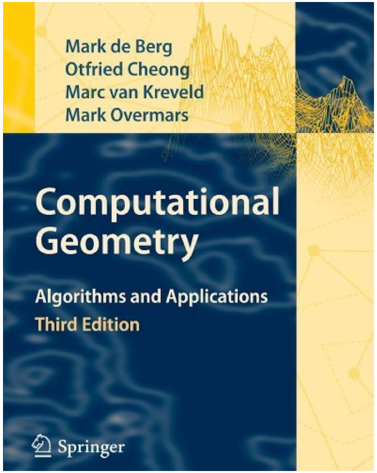
Main reason: I don't want to give an unfair advantage to students who look up the answers (or ask an LLM), compared to the students who do their own work

Secondary reasons:

- Similar questions will be on the exams (or in some cases the same), so having them in problem sets (rather than keeping homework and exam questions separate) should provide more opportunity for practice
- I can assign problems that match each week's lectures, rather than delaying them until those lectures are complete

Takeaway message: you should set aside time to work on the problems because it will show up in your exam scores

Textbook



Computational Geometry: An Introduction (3rd ed.)

Mark de Berg, Otfried ("Mark") Cheong, Marc van Kreveld, Mark Overmars

Strongly recommended and free online from UCI internet addresses via a campus subscription to Springer books:
`https://link.springer.com/book/10.1007/978-3-540-77974-2`

Syllabus provides readings corresponding to lecture content

Area

A toy example

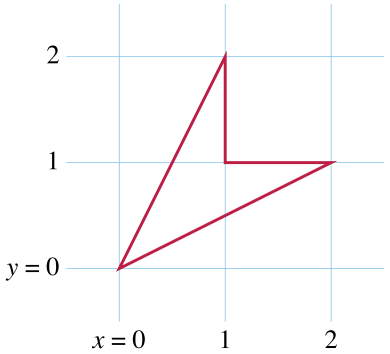
What is the area of this polygon?

Clever solution:

Cut along the grid lines

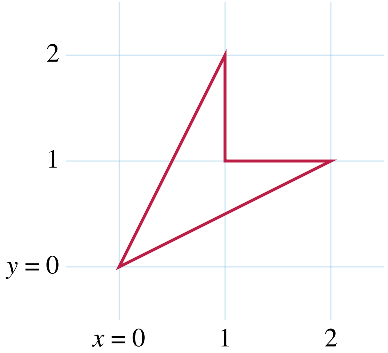
Flip the cut-off triangles into the lower left kite

The result exactly covers a square, area 1



Area of polygons

We want a solution without cleverness that a computer can follow



Vertex coordinates:

$[(0,0),\ (1,2),\ (1,1),\ (2,1)]$

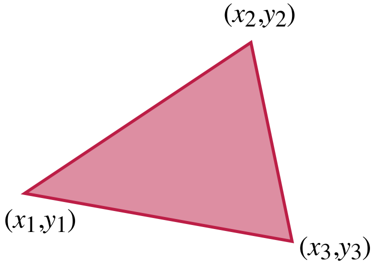
Area: 1

Input: Clockwise sequence of vertices
Each given by integer Cartesian coordinates

Output: A number, the area

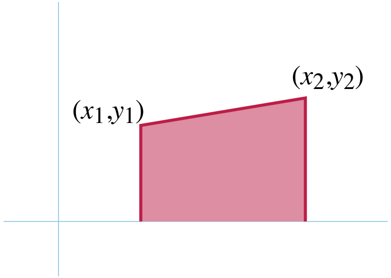
Idea: decompose into simpler shapes

Triangles



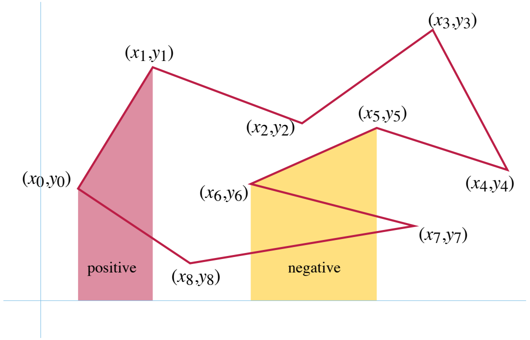
$$A = \frac{1}{2}\det\begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{pmatrix}$$
$$= \frac{1}{2}(x_1y_2 - x_1y_3 + x_2y_3 - x_2y_1 + x_3y_1 - x_3y_2)$$

Trapezoids



$$A = \frac{1}{2}(x_2 - x_1)(y_1 + y_2)$$

Using trapezoids to compute area

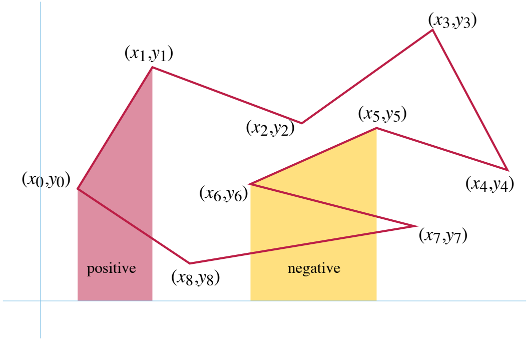


Add the areas of trapezoids below each upper edge of the polygon

Subtract the areas of trapezoids below each lower edge

Outside polygon, positive and negative areas cancel
Inside points all covered by one more positive than negative

Trapezoid formula for area



$$A = \sum_{i=0}^{n-1} \frac{1}{2}(x_{i+1} - x_i)(y_i + y_{i+1})$$

All indexes computed modulo n so $(x_n, y_n) = (x_0, y_0)$

Produces a positive number for positive trapezoids, negative for negative trapezoids

Trapezoid formula as an algorithm

Summation $\Rightarrow$ for-loop

```
def area(P):
    area = 0
    for i in 0, 1, 2, ... n-1:
        j = (i + 1) mod n
        xi,yi = P[i]
        xj,yj = P[j]
        area += (xj-xi)*(yj+yi)/2
    return area
```

Easy, time $= O(n)$

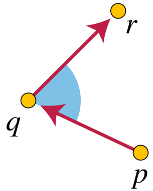
Geometric primitives

What is a primitive?

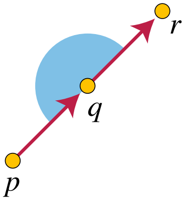
A constant-time formula/algorithm/subroutine for higher-level information than the input coordinates

Example: Triangle area $= \frac{1}{2}(x_1y_2 - x_1y_3 + x_2y_3 - x_2y_1 + x_3y_1 - x_3y_2)$

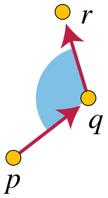
Another example (same subroutine, different interpretation):
If you travel from p to q, then turn and travel from q to r,
which way did you turn?



right turn:
$\text{area}(p, q, r) > 0$



straight:
$\text{area}(p, q, r) = 0$

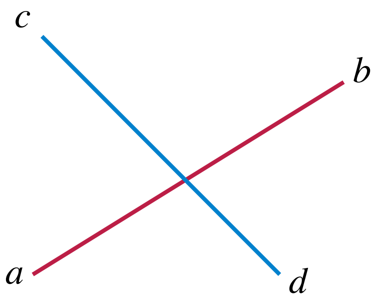


left turn:
$\text{area}(p, q, r) < 0$

Crossing test primitive

Idea: build more complicated primitives from simpler ones

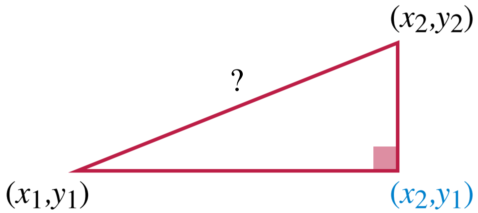
Does line segment *ab* cross line segment *cd*?



Yes, if: *abc* turns the opposite way from *abd*, and
cda turns the opposite way from *cdb*

Distance/length primitive

Distance from (x_1, y_1) to (x_2, y_2) (length of segment)?



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

(apply Pythagoras to right triangle with sides $|x_1 - x_2|$, $|y_1 - y_2|$)

Coordinate systems

Coordinate systems for points in the plane

Cartesian coordinates (x, y)

Simple, familiar
Generalize to higher dims
Widely used, familiar

Polar coordinates (r, θ)

Angle and distance from origin
Widely known, not as useful

Complex numbers

Built into some programming languages (Python)

Make certain transformations easy:

Translate by t:	$q \mapsto q + t$
Scale by s:	$q \mapsto qs$
Rotate by θ:	$q \mapsto q(\cos\theta + i\sin\theta)$

Not as easy to generalize to higher dimensions

Formulas for coordinate conversion

Cartesian to polar

$$r = \sqrt{x^2 + y^2} = \mathrm{hypot}(x, y)$$
$$\theta = \mathrm{atan2}(y, x)$$

Polar to Cartesian

$$x = r \cdot \cos\theta$$
$$y = r \cdot \sin\theta$$

Cartesian to complex

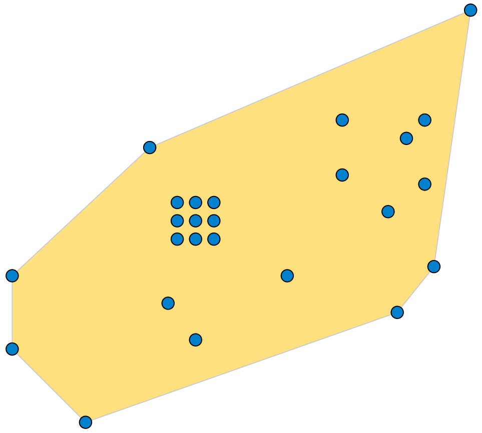
$$q = x + i\,y = x + 1\mathrm{j} * y$$

Complex to Cartesian

$$x = \Re(q) = q.\mathrm{real}$$
$$y = \Im(q) = q.\mathrm{imag}$$

Convex hulls

Motivating question: What is the shape of a set of points?



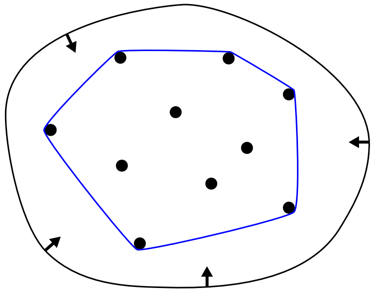
Approximate the points by a convex polygon, called the “convex hull”

Only boundary points affect its shape

Starting point for computing many other properties of the data

- Area
- Diameter (maximum distance between two points)
- Radius of smallest enclosing circle
- Classification in machine learning

What is the convex hull?



https://commons.wikimedia.org/wiki/File:ConvexHull.svg

Input: n points

Output (intuitive): polygon formed by stretching a rubber band around the points

Simplifying assumptions:

- No three in line
- No two with same x

More formal definitions

- Min perimeter polygon surrounding all points
- Min area convex polygon surrounding all points
- Intersection of all halfplanes that contain all points
- Union of all points, line segments through two points, and triangles formed by three points
- Set of all convex combinations (weighted averages)

$$c_0 p_0 + c_1 p_1 + c_2 p_2 + \cdots$$

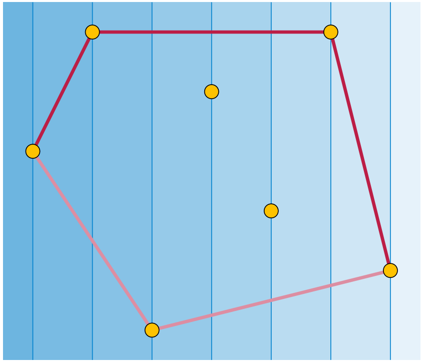
  for points p_i, positive coefficients c_i summing to one

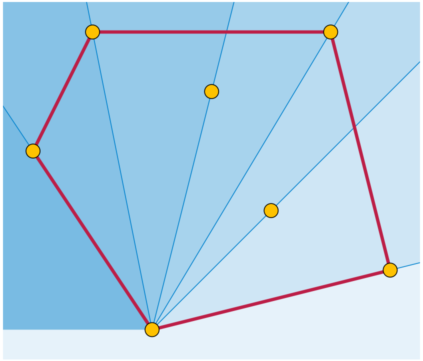
All equivalent to each other!

Graham scan

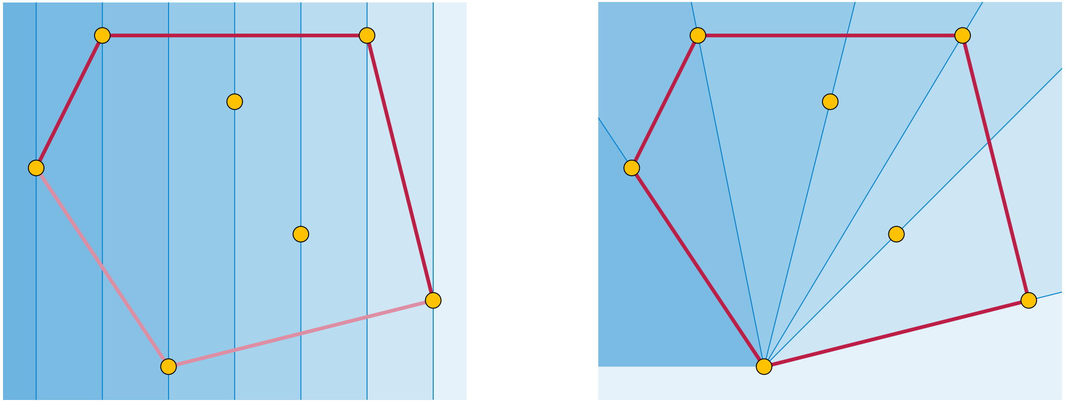
A simple, fast algorithm using only sorting and a stack. Two different versions:

- Our book: sort by x-coordinates (easier)
  Find "upper hull" and "lower hull" separately
- Some sources: sort radially around bottom point; find whole hull in a single pass



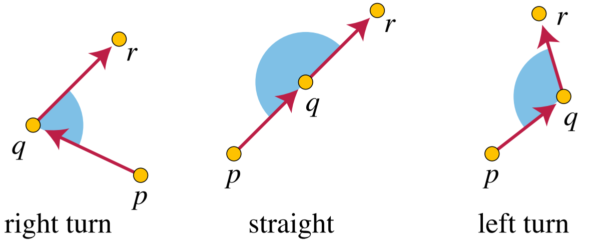


Upper and lower hulls



Split into two paths at leftmost and rightmost vertex

Left and right turns



If you travel in a consistent direction along the hull, you always turn the same way

- Left-to-right on upper hull: always turn right
- Left-to-right on lower hull: always turn left

Idea for algorithm: check if this is true and fix wrong-direction turns when we find them

To find upper hull of point set P

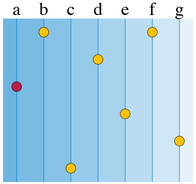
- Sort P by x-coordinates
- Create an empty stack S
  (Will contain upper hull of points seen so far)
- For each point p_i in sorted order:

```
        While $S$ contains $\geq 2$ points
        and second-last – last – $p_i$ is a left turn:  ⇐ Primitive: sign of triangle area!
                pop $S$
        push $p_i$ onto $S$
```

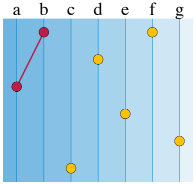
- Return S

Lower hull: Change “left” to “right”, or reverse the sorted order

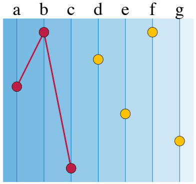
Example of Graham scan



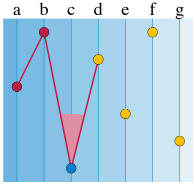
push a; stack = [a]



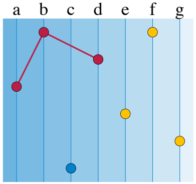
push b; stack = [a,b]



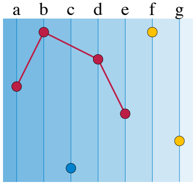
push c; stack = [a,b,c]



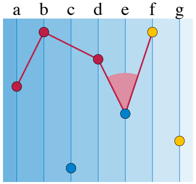
bcd=left; pop; stack = [a,b]



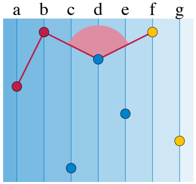
push d; stack = [a,b,d]



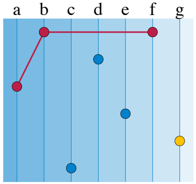
push e; stack = [a,b,d,e]



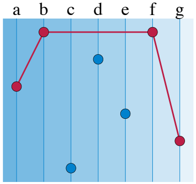
def=left; pop; stack = [a,b,d]



bdf=left; pop; stack = [a,b]



push f; stack = [a,b,f]



push g; stack = [a,b,f,g]

Analysis of Graham scan

Outer loop runs once per point, just does simple stack operations

Each time through the inner loop, we pop a point

Each point is only pushed once, so it can only be popped once

$\Rightarrow O(n)$ total times through inner loop

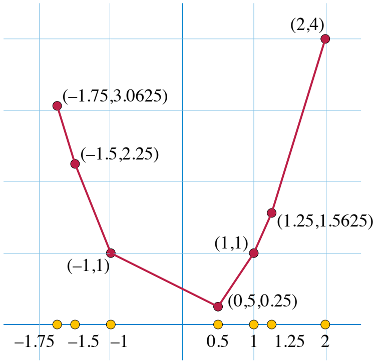
Total $=$ sorting $+ O(n)$

A lower bound on convex hulls

Given input list L of n numbers, $x_0, \ldots x_{n-1}$
Transform into n points (x_i, x_i^2)

Lower hull consists of all points in sorted order

So fast convex hull algorithm $\Rightarrow$ fast sorting



But this does not prove that convex hulls require $\Omega(n \log n)$ time

The $\Omega(n \log n)$ sorting lower bound is for a limited model of computing where we can only compare input numbers, but this model is unable to compute hulls!

It does prove: If we only use primitives with binary answers (like the left turn / right turn primitive), $\Omega(n \log n)$ calls to the primitives are required.

Numerical precision

Primitives used for decisions must be exact!

If the proof of an algorithm's correctness depends on it making correct choices, then all the choices it makes must be correct

Primitives used for if-then-else tests, loop termination, etc., must always produce the intended results

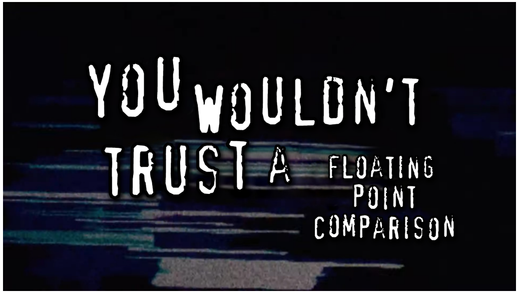
Otherwise, programs using them are likely to produce wrong results or, worse, crash!

https://commons.wikimedia.org/wiki/File:Kobaysahi_crash_Canadian_GP_2010_(cropped).jpg

What kind of numbers should we use?

Coordinates = numbers

The obvious choice is floating point, but that is inexact $\Rightarrow$ crashes

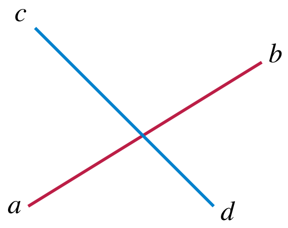


Instead, geometric algorithms generally need to represent coordinates as integers, and compute primitives to enough precision to guarantee exact results

Area / turn direction / crossing primitives use products of two coordinates and should be computed to twice as many bits of precision as the input coordinates

Is the crossing test really that simple?

Does line segment *ab* cross line segment *cd*?



Yes, if: *abc* turns the opposite way from *abd*, and
cda turns the opposite way from *cdb*

But what if some of these turns are straight?

Simplifying assumption: no three input points are in a line

General position

"General position" = no unexpected numerical coincidences

E.g. when using turn direction primitive, never comes out zero (no three points are on a line)

This assumption simplifies algorithm design (and lecturing!)
but is a poor match for real-world inputs

Instead, we can:

- Do more analysis to carefully handle special cases
- Perturb inputs by small distances to make them general position
- Use numerical libraries that automatically simulate the results of infinitesimally small perturbations

The problem with distances

Recall the distance formula:

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

Even when all coordinates are integers, square root is usually not!

Perimeter of a polygon = sum of square roots

How many bits of precision do we need to compare two perimeters and tell which one is shorter? Unknown!

Partial solution: when comparing distances, but not adding them, we can compare squared distance before taking its square root