11. Computational Geometry and Convex Hulls

CPSC 535 ~ Spring 2019

Kevin A. Wortman

CALIFORNIA STATE UNIVERSITY FULLERTON

November 18, 2019

Big Idea: Output Sensitive Algorithm

- **input sensitive**: time efficiency is a function of the input e.g. size n, # edges m
- **output sensitive**: efficiency is also a function of the *output* size e.g. # items returned
- most relevant when the size of the output could be the bottleneck

Computational Geometry

computational X: interdisciplinary study of computer science with X

(computational finance, epidemiology, physics, finance, etc.)

computational geometry (CG): algorithms, data structures, asymptotic analysis, of geometric objects: points, lines, circles, triangle meshes, etc.

Computational Geometry Applications

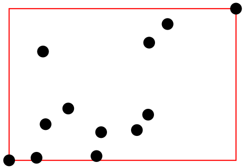
Applications of CG:

- 3D computer graphics
- graphical user interfaces (GUIs)
- geographic information systems (GIS), geographic databases
- scene reconstruction, self-driving cars (e.g. LIDAR)
- business operations research (e.g. linear programming, aircraft control)
- manufacturing (e.g. feasibility of assembly, castings)

Putting the Geo in CG

Some general algorithms can actually solve geometric problems efficiently, without any awareness of geometry.

bounding box problem
input: set of 2D points $P = \{p_1, p_2, \ldots, p_n\}$
output: points $tl = (x_l, y_t)$ and $rb = (x_r, y_b)$ such that the rectangle with top-left corner tl and bottom-right corner rb contains P



Naïve, optimal algorithm:
$x_l = \min x, y_t = \max y, x_r = \max x, y_b = \min y$; $\Theta(n)$ time

Computational geometers are more interested when geometric properties matter.

Line Segment Predicates

We can use arithmetic to answer any of the following predicates (questions) about points p_0, p_1, p_2, p_3 in $\Theta(1)$ time **(sketch)**:

1. Is line segment $\overline{p_0 p_1}$ clockwise from $\overline{p_0 p_2}$ around the common endpoint p_0?
2. If we follow $\overline{p_0 p_1}$ and then $\overline{p_1 p_2}$, do we turn right or left?
3. Do line segments $\overline{p_0 p_1}$ and $\overline{p_2 p_3}$ intersect?

$\Longrightarrow$ We may use any of these in pseudocode.

Degeneracy and Non-Degeneracy Assumptions

degenerate object: has the proper shape/type, but the values are a special case that betrays the spirit of the definition

Example: triangle ≡ three points (p_1, p_2, p_3)
degenerate triangle: $p_1 = p_2 = p_3$, or two points colinear **(sketch)**

non-degeneracy assumption:

- constraint that input to a CG algorithm is not degenerate in specific ways
- simplifies algorithm design
- assume that in practice, some combination of
  - degeneracies do not occur
  - input can be preprocessed to remove degeneracies
  - implementer can modify algorithm to handle degeneracies

Sweep Algorithms

A pattern in CG algorithms:

- *line sweep:* envision a line “sweeping” through the input
- e.g. a vertical line sweeping left-to-right **(sketch)**
- helps us visualize a 2D situation as a 1D situation that changes over time
- like duality, doesn’t actually change the problem, but might help us problem-solve
- generalizes to higher dimensions e.g. plane sweep in 3D, hyperplane sweep in any dimension

Convex Hulls

convex hull problem
input: set of $n \geq 3$ points Q
output: $CH(Q)$, the subset of Q that is the set of vertices on the convex hull of Q

convex hull $\equiv$ boundary of convex polygon enclosing all of Q
(sketch)

Applications

- object intersection in raytracing, video games, GUIs
- drawing implicit regions in GIS
- finding farthest points (they're always CH vertices)
- component of other algorithms

Approaches to Convex Hulls

Like the sorting problem, many algorithm patterns work for convex hulls, and there is a rich literature of competitive algorithms.

- Greedy pattern: line-sweep, update hull as we go
- Divide-and-conquer: divide Q in half, compute convex hulls for each half, merge two convex hulls into one
- Iterative improvement: start with a superset of $CH(Q)$; refine by repeatedly eliminating a constant fraction of the points until only $CH(Q)$ remains

Baseline Algorithm

Observe

- any two input points define a line ℓ **(sketch)**
- when those points are both in $CH(Q)$, remaining $n-2$ points are all on the same side of ℓ **(a geometric property)**
- $\Longrightarrow$ for each pair of input points p, q, see whether all other points are on the same side of ℓ
- if so include p, q in $CH(Q)$ **(sketch)**

Baseline Pseudocode

function NAIVE-CONVEX-HULL(Q)
    $H = \varnothing$
    for distinct points $p, q \in Q$ **do**
        form line ℓ intersecting p and q
        $k = \#$ points above ℓ
        if $k = (n-2)$ or $k = 0$ **then**
            $H = H \cup \{p, q\}$
        end if
    end for
    return H
end function

Analysis: $\Theta(n^2)$ iterations, counting $\#$points is $\Theta(n)$
$\implies \Theta(n^3)$ time

Graham Scan Idea

- greedy pattern, reduction-to-sorting
- **Geometric property**: when touring a CH in counter-clockwise order, we **only make left turns (sketch)**
- right turn = exiting a concavity, middle point not in hull **(sketch)**
- $\therefore$ sweep counter-clockwise, keep points that participate in left turns, drop points in the middle of right turns **(sketch)**
- alternative kind of line sweep: rotating the line (not left-to-right)

Graham Scan Greedy Heuristic

- $p_1, \ldots, p_m = Q$ sorted into counter-clockwise order, eliminating ties
- stack S of points; contains hull of points visited *already*
- base case: push first 3 points onto S
  - for any three points p, q, r forming a non-degenerate triangle, $CH(\{p, q, r\}) = \{p, q, r\}$
- inductive case:
  - examine next input point p_i, top of stack t, next-lowest stack point r
  - if $\angle rtp_i$ is not a left turn $\implies$ t not on hull
- Note: need stack data structure w/ accessor to top **two** elements

Graham Scan Pseudocode

```
function GRAHAM-SCAN(Q)                              ▷ guaranteed |Q| ≥ 3
    p_0 = lowest point in Q (break ties by choosing leftmost point)
    p_1 ... p_m = sort Q − {p_0} into counter-clockwise order, by polar
angle with p_0; break ties by keeping only the point farthest from p_0
    S = new stack
    S.PUSH(p0)
    S.PUSH(p1)
    S.PUSH(p2)
    for i from 3 through m do
        while ∠ p_i, S.TOP, S.BELOWTOP is non-left turn do
            S.POP()
        end while
        S.PUSH(p_i)
    end for
    return set of point still in S
end function
```

Graham Scan Analysis

- find p_0: $\Theta(n)$
- sort: $\Theta(n \log n)$
- eliminate tied points: $\Theta(n)$
- each stack operation is $\Theta(1)$
- **for** loop repeats $m < n$ times
- turn angle test, stack operations are $\Theta(1)$
- $\Rightarrow \Theta(n \log n)$ time
- dominating term is sort *(reduction to sorting)*
- organizing data structure is arrayed stack
- $\Rightarrow$ good constant factors

Jarvis March

Alternative greedy heuristic: moving around the hull counter-clockwise, each step from one vertex to the next is *the input point whose angle is shallowest.* **(sketch)**

$\Rightarrow$ we can start from a CH point, then incrementally find one more CH point until we're done.

Called "gift wrapping" b/c this resembles carefully wrapping up an irregular object in paper or foil. **(sketch)**

(Jarvis march is sometimes called the *gift-wrapping algorithm.*)

Jarvis March Pseudocode

Jarvis march (Q)

1. $H = \varnothing$
2. Let ℓ = lowest point in Q (min. y-coord.)
3. Let h = highest point in Q
4. (right chain) Starting from ℓ and until we reach h:
    4.1 Linear search Q for the next point p_i, minimizing the angle between p_i and the previous point
    4.2 Include p_i in H and continue the loop at p_i.
5. (left chain) Repeat the previous process but starting from h and ending at ℓ.
6. Return H

Jarvis March Analysis

Preprocessing to find $h, \ell : \Theta(n)$

Each iteration of the left/right-chain loops identifies one hull point $\implies$ in total they iterate h times, where $h \equiv$ number of points on the hull.

linear search inside the loops takes $\Theta(n)$ time.

$\therefore \Theta(nh)$ total time.

Comparison of Convex Hull Algorithms

Algorithm	**Time**	**Main Idea**
Graham Scan	$\Theta(n \log n)$	sort, skip right turns
Jarvis March	$\Theta(nh)$	gift-wrapping

What is the relationship between n and h?

n vs h

Recall

- $n \equiv$ # input points = $|Q|$
- $h \equiv$ # output points = # vertices of convex hull = $|CH(Q)|$

For fixed n,

- minimum $h = 3$ when all input points are enclosed in a triangle **(sketch)**
- maximum $h = n$ when all input points happen to be convex hull vertices **(sketch)**

$$3 \leq h \leq n$$

Summary of Convex Hull Algorithms

FYI

- **Chan's algorithm** is an optimal output-sensitive algorithm
- (not covered in book or class)
- combines both algorithms, divides input points using Graham's heuristic, merges hulls using Jarvis' heuristic
- $\Theta(n \log h)$ time

Algorithm	**Time**	$h \in O(1)$	$h \in \Theta(n)$
Graham Scan	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$
Jarvis March	$\Theta(nh)$	$\Theta(n)$	$\Theta(n^2)$
Chan's algorithm	$\Theta(n \log h)$	$\Theta(n)$	$\Theta(n \log n)$