

CS 467/567: Elements of Computational Geometry

Stefan D. Bruda

Winter 2020



POINTS AND SEGMENTS

- **Points** identified by their (x, y) coordinates
 - Some times useful to think about points as **vectors** $p = (x, y)$
- **Convex combination** of two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$: point $p_3 = (x_3, y_3)$ such that $p_3 = \alpha p_1 + (1 - \alpha)p_2$ for some $0 \leq \alpha \leq 1$ (meaning $x_3 = \alpha x_1 + (1 - \alpha)x_2$ and $y_3 = \alpha y_1 + (1 - \alpha)y_2$)
 - The set of all convex combinations of p_1 and p_2 is the **segment** $\overline{p_1 p_2}$
 - Some times the ordering of the end points matters = **directed segment** $\overrightarrow{p_1 p_2}$
- Interesting **basic algorithmic questions about segments**:
 - 1 Given $\overrightarrow{p_0 p_1}$ and $\overrightarrow{p_0 p_2}$, is $\overrightarrow{p_0 p_1}$ clockwise from $\overrightarrow{p_0 p_2}$ (with respect to the common point)?
 - 2 Given two segments $\overline{p_1 p_2}$ and $\overline{p_2 p_3}$, if we traverse $\overline{p_1 p_2}$ and then $\overline{p_2 p_3}$ do we make a left turn at p_2 ?
 - 3 Do segments $\overline{p_1 p_2}$ and $\overline{p_3 p_4}$ intersect?



POINTS AND SEGMENTS

- **Points** identified by their (x, y) coordinates
 - Some times useful to think about points as **vectors** $p = (x, y)$
- **Convex combination** of two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$: point $p_3 = (x_3, y_3)$ such that $p_3 = \alpha p_1 + (1 - \alpha)p_2$ for some $0 \leq \alpha \leq 1$ (meaning $x_3 = \alpha x_1 + (1 - \alpha)x_2$ and $y_3 = \alpha y_1 + (1 - \alpha)y_2$)
 - The set of all convex combinations of p_1 and p_2 is the **segment** $\overline{p_1 p_2}$
 - Some times the ordering of the end points matters = **directed segment** $\overrightarrow{p_1 p_2}$
- Interesting **basic algorithmic questions about segments**:
 - 1 Given $\overrightarrow{p_0 p_1}$ and $\overrightarrow{p_0 p_2}$, is $\overrightarrow{p_0 p_1}$ clockwise from $\overrightarrow{p_0 p_2}$ (with respect to the common point)?
 - 2 Given two segments $\overline{p_1 p_2}$ and $\overline{p_2 p_3}$, if we traverse $\overline{p_1 p_2}$ and then $\overline{p_2 p_3}$ do we make a left turn at p_2 ?
 - 3 Do segments $\overline{p_1 p_2}$ and $\overline{p_3 p_4}$ intersect?
 - Desired: $O(1)$ complexity
 - To be avoided: division (approximate) and trigonometric functions (expensive and also approximate)



- The cross product $p_1 \times p_2$ is the area of the parallelogram formed by $(0, 0)$, p_1 , p_2 , and $p_1 + p_2$:

$$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1 = -p_2 \times p_1$$



- The cross product $p_1 \times p_2$ is the area of the parallelogram formed by $(0, 0)$, p_1 , p_2 , and $p_1 + p_2$:

$$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1 = -p_2 \times p_1$$

- $p_1 \times p_2 > 0$ iff p_1 is **clockwise** from p_2



- The cross product $p_1 \times p_2$ is the area of the parallelogram formed by $(0, 0)$, p_1 , p_2 , and $p_1 + p_2$:

$$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1 = -p_2 \times p_1$$

- $p_1 \times p_2 > 0$ iff p_1 is **clockwise** from p_2
- Whether $\overrightarrow{p_0 p_1}$ clockwise from $\overrightarrow{p_0 p_2}$ can be solved by translating the segments so that p_0 is placed at $(0, 0)$ and then computing the cross product

$$(p_1 - p_0) \times (p_2 - p_0) = (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)$$

Then $(p_1 - p_0) \times (p_2 - p_0) > 0$ iff $\overrightarrow{p_0 p_1}$ is clockwise from $\overrightarrow{p_0 p_2}$



- The cross product $p_1 \times p_2$ is the area of the parallelogram formed by $(0, 0)$, p_1 , p_2 , and $p_1 + p_2$:

$$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1 = -p_2 \times p_1$$

- $p_1 \times p_2 > 0$ iff p_1 is **clockwise** from p_2
- Whether $\overrightarrow{p_0 p_1}$ clockwise from $\overrightarrow{p_0 p_2}$ can be solved by translating the segments so that p_0 is placed at $(0, 0)$ and then computing the cross product

$$(p_1 - p_0) \times (p_2 - p_0) = (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)$$

Then $(p_1 - p_0) \times (p_2 - p_0) > 0$ iff $\overrightarrow{p_0 p_1}$ is clockwise from $\overrightarrow{p_0 p_2}$

- When traversing $\overline{p_1 p_2}$ and $\overline{p_2 p_3}$ we turn left at p_2 iff



- The cross product $p_1 \times p_2$ is the area of the parallelogram formed by $(0, 0)$, p_1 , p_2 , and $p_1 + p_2$:

$$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1 = -p_2 \times p_1$$

- $p_1 \times p_2 > 0$ iff p_1 is **clockwise** from p_2
- Whether $\overrightarrow{p_0 p_1}$ clockwise from $\overrightarrow{p_0 p_2}$ can be solved by translating the segments so that p_0 is placed at $(0, 0)$ and then computing the cross product

$$(p_1 - p_0) \times (p_2 - p_0) = (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)$$

Then $(p_1 - p_0) \times (p_2 - p_0) > 0$ iff $\overrightarrow{p_0 p_1}$ is clockwise from $\overrightarrow{p_0 p_2}$

- When traversing $\overrightarrow{p_1 p_2}$ and $\overrightarrow{p_2 p_3}$ we turn left at p_2 iff $\overrightarrow{p_1 p_3}$ is counterclockwise from $\overrightarrow{p_1 p_2}$ that is, $(p_3 - p_1) \times (p_2 - p_1) \leq 0$



Two segments intersect iff either of the following conditions hold:

- 1 Each segment straddles the line containing the other
 - $\overline{p_1 p_2}$ straddles a line if p_1 is on one side of the line and p_2 on the other side
- 2 An end point of one segment lies on the other segment (boundary case)

Algorithm SEGMENTS-INTERSECT ($\overline{p_1 p_2}, \overline{p_3 p_4}$):

- 1 $d_1 \leftarrow \text{DIRECTION}(p_3, p_4, p_1)$
- 2 $d_2 \leftarrow \text{DIRECTION}(p_3, p_4, p_2)$
- 3 $d_3 \leftarrow \text{DIRECTION}(p_1, p_2, p_3)$
- 4 $d_4 \leftarrow \text{DIRECTION}(p_1, p_2, p_4)$
- 5 **if** $(d_1 > 0 \wedge d_2 < 0 \vee d_1 < 0 \wedge d_2 > 0) \wedge$
 $(d_3 > 0 \wedge d_4 < 0 \vee d_3 < 0 \wedge d_4 > 0)$ **then return TRUE**
- 6 **else if** $(d_1 == 0 \wedge \text{ON-SEGMENT}(p_3, p_4, p_1)) \vee$
 $(d_2 == 0 \wedge \text{ON-SEGMENT}(p_3, p_4, p_2)) \vee$
 $(d_3 == 0 \wedge \text{ON-SEGMENT}(p_1, p_2, p_3)) \vee$
 $(d_4 == 0 \wedge \text{ON-SEGMENT}(p_1, p_2, p_4))$ **then return TRUE**
- 7 **else return FALSE**



Algorithm DIRECTION (p_i, p_j, p_k) :

1 **return** $(p_k - p_i) \times (p_j - p_i)$

Algorithm ON-SEGMENT (p_i, p_j, p_k) :

1 **return** $\min(x_i, x_j) \leq x_k \leq \max(x_i, x_j) \wedge$
 $\min(y_i, y_j) \leq y_k \leq \max(y_i, y_j)$



- Problem: Given a set of segments, determine whether any two segments from the set intersect
 - Simplifying assumptions: no vertical segment, and no single-point intersection of three segments (or more)
- Solvable by **sweeping** – imaginary vertical **sweep line** passing through the objects left to right
- The sweep line at coordinate x defines a **preorder** \succsim_x over segments: $s_1 \succsim_x s_2$ iff the intersection of s_1 with the sweep line at x is higher than the intersection of s_2 with the same sweep line
 - Total order for all the segments that intersect the line at x
- Sweep algorithm based on the **sweep line status** – the relationship between the objects intersected by the sweep line
 - Can be stored using a binary search tree such as a red-black tree = $O(\log n)$ access time
 - $\text{INSERT}(T, s)$ = inserts segment s into T
 - $\text{DELETE}(T, s)$ = deletes s from T
 - $\text{ABOVE}(T, s)$ = returns the segment immediately above s in T
 - $\text{BELOW}(T, s)$ = returns the segment immediately below s in T



Algorithm ANY-SEGMENT-INTERSECT(S : set of segments):

- 1 $T \leftarrow \emptyset$
 - 2 Sort the endpoints of the segments in S from left to right; break ties by putting left endpoints before right endpoints and further putting points with lower y coordinates first
 - 3 **for each** point p in the sorted list **do**
 - 1 **if** p is the left endpoint of a segment s **then**
 - 1 INSERT(T, s)
 - 2 **if** ABOVE(T, s) exists and intersects s **or** BELOW(T, s) exists and intersects s **then return** TRUE
 - 2 **if** p is the right endpoint of a segment s **then**
 - 1 **if** both ABOVE(T, s) and BELOW(T, s) exist and intersect each other **then return** TRUE
 - 2 DELETE(T, s)
 - 4 **return** FALSE
- Complexity: $O(n \log n)$



The **convex hull** $CH(Q)$ of a set of points Q is the smallest convex polygon P for which each point in Q is either on the boundary of P or inside P

Algorithm GRAHAM-SCAN(Q):

- 1 let p_0 be the point in Q with the minimum coordinate, or the leftmost such point in case of a tie
 - 2 let $\langle p_1, p_2, \dots, p_m \rangle$ be the remaining points in Q sorted by polar angle in counterclockwise order around p_0
 - 1 remove all but the farthest from p_0 points that have the same angle
 - 3 let S be an empty stack
 - 4 PUSH(p_0, S); PUSH(p_1, S); PUSH(p_2, S)
 - 5 **for** $i \leftarrow 3$ **to** m **do**
 - 1 **while** the angle formed by NEXT-TO-TOP(S), TOP(S), and p_i makes a non-left turn **do** POP(S)
 - 2 PUSH(p_i, S)
 - 6 **return** S
- Complexity: $O(n \log n)$



- **Gift wrapping** or **Jarvis' march** has the complexity $O(nh)$ where h is the number of points in the convex hull
 - Asymptotically faster than the Graham scan whenever the convex hull is small ($o(\log n)$)

Algorithm JARVIS-MARCH(Q) returns H :

- 1 let p_0 be the point in Q with the minimum coordinate, or the leftmost such point in case of a tie
- 2 $H \rightarrow \emptyset$
- 3 Construct the **right chain**:
 - 1 $i \leftarrow 0$; add p_i to H
 - 2 **until** p_i is the highest vertex **do**
let p_{i+1} be the vertex with the smallest polar angle with respect to p_i
 $i \leftarrow i + 1$
- 4 Construct the **left chain**:
 - 1 **until** $p_i = p_0$ **do**
let p_{i+1} be the vertex with the smallest polar angle with respect to p_i **from the negative x axis**
add p_{i+1} to H ; $i \leftarrow i + 1$



Algorithm DNC-CONVEX-HULL(Q : set of points):

- 1 **if** $|Q| < 3$ **then** compute the hull directly (triangle or line) and return it
 - 2 **else**
 - 1 partition Q into equal sets Q_l with the lowest x coordinates and Q_h with the highest x coordinates.
 - 2 $H_l \leftarrow$ DNC-CONVEX-HULL(Q_l); $H_h \leftarrow$ DNC-CONVEX-HULL(Q_h)
 - 3 compute \overline{ab} and \overline{cd} , the lower and upper for H_l and H_h :
 - 1 let a be the rightmost point of H_l and b the leftmost point of H_h
 - 2 **while** \overline{ab} is not a lower tangent for H_l and H_h **do**
 - while** \overline{ab} is not a lower tangent for H_l **do** move a clockwise on H_l
 - while** \overline{ab} is not a lower tangent for H_h **do** move b counterclockwise on H_h
 - 3 compute the upper tangent similarly
 - 4 discard all the points between \overline{ab} and \overline{cd} and return the remaining points as the convex hull
- Complexity: $O(n \log n)$ (why?)



- The **Quickhull** algorithm:

Algorithm QUICKHULL(Q : set of points):

- 1 find the points a and b with minimum and maximum x coordinates (part of the convex hull)
- 2 **return** $\{a\} \cup \text{QUICKHUL-REC}(\overrightarrow{ab}) \cup \{b\} \cup \text{QUICKHUL-REC}(\overrightarrow{ba})$

Algorithm QUICKHUL-REC(\overrightarrow{ab}):

- 1 determine l , the point with the maximum distance from \overrightarrow{ab} and to the left of \overrightarrow{ab}
 - 2 **return** $\text{QUICKHUL-REC}(\overrightarrow{al}) \cup \{l\} \cup \text{QUICKHUL-REC}(\overrightarrow{lb})$
- Worst-case complexity $O(n^2)$, average-case complexity $O(n \log n)$ (just like Quicksort)
 - Unlike Quicksort there is no obvious randomized version with $O(n \log n)$ expected running time
 - Still, performs very well in practice



- Obvious lower bound for convex hull: $\Omega(n)$
- In practice some sorting is required (either implicit or explicit) so the lower bound becomes $\Omega(n \log n)$
- However, if it is possible to discard the points that do not belong to the hull before doing the sorting then the complexity becomes $\Omega(n \log h)$ (where h is the number of points in the convex hull – **output sensitive complexity/algorithm**)



- Obvious lower bound for convex hull: $\Omega(n)$
- In practice some sorting is required (either implicit or explicit) so the lower bound becomes $\Omega(n \log n)$
- However, if it is possible to discard the points that do not belong to the hull before doing the sorting then the complexity becomes $\Omega(n \log h)$ (where h is the number of points in the convex hull – **output sensitive complexity/algorithm**)
- Meeting (even exceeding!) the lower bound in practice (i.e., most of the time): Quickhull + the **Akl-Toussaint heuristic**:
 - Find $m_x, M_x, m_y,$ and $M_y,$ the extreme points on both axes
 - Compute the convex hull as $\{m_x\} \cup \text{QUICKHUL-REC}(\overrightarrow{m_x m_y}) \cup \{m_y\} \cup \text{QUICKHUL-REC}(\overrightarrow{m_y M_x}) \cup \{M_x\} \cup \text{QUICKHUL-REC}(\overrightarrow{M_x M_y}) \cup \{M_y\} \cup \text{QUICKHUL-REC}(\overrightarrow{M_y m_x})$
 - All the points in the quadrilateral $m_x m_y M_x M_y$ are effectively discarded from the outset
 - **Linear expected running time** for random point distribution with certain probability density functions common in practice

CONVEX HULL: MEETING THE LOWER BOUND (CONT'D)



- Properly meeting the lower bound (worst case analysis): Graham scan + gift wrapping = **Chan's algorithm** (1996)
- Idea (**Chan's partial convex hull algorithm**):
 - For some given m , split the points from Q in m groups of equal size ($O(n)$)
 - Compute the convex hull of each group using Graham's scan ($O(m \log m)$)
 - Run gift wrapping on the groups
 - $O(\log m)$ time to compute the tangent between a point and a convex hull
 - Still h gift wrapping steps, but only on n/m "points"
 - Overall complexity: $O(n + hn/m \log m)$ which is $O(n \log h)$ whenever $m = h$

CONVEX HULL: MEETING THE LOWER BOUND (CONT'D)



- Properly meeting the lower bound (worst case analysis): Graham scan + gift wrapping = **Chan's algorithm** (1996)
- Idea (**Chan's partial convex hull algorithm**):
 - For some given m , split the points from Q in m groups of equal size ($O(n)$)
 - Compute the convex hull of each group using Graham's scan ($O(m \log m)$)
 - Run gift wrapping on the groups
 - $O(\log m)$ time to compute the tangent between a point and a convex hull
 - Still h gift wrapping steps, but only on n/m "points"
 - Overall complexity: $O(n + hn/m \log m)$ which is $O(n \log h)$ whenever $m = h$
- **Chan's complete convex hull algorithm**:
 - Try increasingly larger values for m until we stumble upon $m \geq h$
 - Cannot do it iteratively (h multiplier) or using binary search ($\log n$ multiplier)
 - We are however OK with m reaching a polynomial in h rather than h itself: m will reach h^c and the overall complexity is still $O(n \log h)$
 - So we start with $m = 2$ and repeatedly **square** the previous value of m until we obtain $m \geq h$