

# Contents

<b>1</b>	<b>Algorithms: Overview</b>	<b>2</b>
<b>2</b>	<b>Intro: Computational Geometry - A User's Guide (Notes)</b>	<b>3</b>
2.1	Hierarchical Searching Method . . . . .	3
2.2	Divide and Conquer Method . . . . .	3
2.3	Duality Method . . . . .	4
<b>3</b>	<b>Convex Hull</b>	<b>5</b>
3.1	Geometry of Convex Hull . . . . .	5
3.2	<b>Algorithm</b> Naive Convex Hull - $O(n^4)$ . . . . .	6
3.3	<b>Algorithm</b> Ways to find Left/Right Turn . . . . .	6
3.4	<b>Algorithm</b> SLOWCONVEXHULL( $P$ ) - Naive $O(n^3)$ . . . . .	6
3.5	<b>Algorithm</b> ConvexHull( $P$ ) - Incremental Algorithm $O(n \log n)$ . . . . .	7
3.6	<b>Algorithm</b> Jarvis March (Gift Wrapping) - $O(n^2)$ . . . . .	9
3.7	<b>Algorithm</b> Graham Scan - $O(n \log n)$ . . . . .	10
3.8	Bridges . . . . .	10
3.9	Divide & Conquer Convex Hull Methods . . . . .	10
3.10	<b>Algorithm</b> Quick Hull - $O(n^2)$ . . . . .	11
3.11	<b>Algorithm</b> Ultimate Planar Convex Hull (Marriage Before Conquest ) - $O(n \log h)$ . . . . .	11
3.12	<b>Algorithm</b> Dynamic $\mathcal{CH}$ / Order Decomposable Problem - $O(\log^2 n)$ . . . . .	13
3.13	3D Convex Hull . . . . .	14

# 1 Algorithms: Overview

TOPIC/type	Name	Best	Average	Worst	Amortized
<b>Convex Hull</b>		$\Omega(n)$		$O(n^4)$	
incremental	Naive Approach (Triples)	$\Omega(n^4)$	$\Theta(n^4)$	$O(n^4)$	
incremental	Slow $\mathcal{CH}$ (Pairs)	$\Omega(n^3)$	$\Theta(n^3)$	$O(n^3)$	
incremental	Jarvis March (Gift Wrapping)	$\Omega(nh)$	$\Theta(n \log n)$	$O(n^2)$	
incremental	Graham Scan	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n \log n)$	
D&C	Quick Hull	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n^2)$	
prune&search	Marriage before Conquest (Ultimate)	$\Omega(n \log h)$	$\Theta(n \log h)$	$O(n \log h)$	
order decomp	Dynamic Convex Hull	$\Omega(\log^2 n)$	$\Theta(\log^2 n)$	$O(\log^2 n)$	
	Name	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n \log n)$	

$h$  is the number of points on  $\mathcal{CH}$ .

## 2 Intro: Computational Geometry - A User's Guide (Notes)

Introduction to algorithms for computations that are geometric in nature. Souvaine & Dobkin describe some methods with geometric applications.

Three families of Geometric Algorithms:

- Decomposition of problem into subproblems – [Hierarchical Searching Method](#)
- Decomposition of problem into subproblems – [Divide and Conquer Method](#)
- Transform a problem into a new (maybe more tractable) format – [Duality Method](#)

Geometric Principles

- Planar Point Location
- Convex Hull Construction and Updating
- Computation of Polygon
- Computation of Disk
- Computation of Half-Space Intersections

### 2.1 Hierarchical Searching Method

- Geometric problem is preprocessed to a coarse representation, such that it can be broken down, and search queries can be called on localized region where the problem is solved.
- Algorithmic efficiency is then balanced against preprocessing time and storage space requirements.
- Binary Search of Sorted Array

#### 2.1.1 [Binary Search](#) on Geometric Problems

**Input:** A collection of  $N$  disjointed polygons in the plane.

**Output:** For a given point  $P$ , find all polygons to which it belongs

**Naive Solution:** check for  $P$  in each points of the polygons.

**Rectangle Search I & II & III**

**General & Dynamic Polygon Search**

### 2.2 Divide and Conquer Method

- Problem broken into smaller subproblems, and solved recursively. A method is defined for combining solutions to subproblems to come up with solution for entire problem.

- Sort-merge
- Can work with hierarchical search methods, e.g. divide-and-conquer to sort a set, and then use binary search to find target.

## **2.3 Duality Method**

- Duality is used as a transformation. Given two sets  $A$  and  $B$  and a problem about their interrelationship, apply transform  $T$  and solve the (ideally simpler) problem about  $T(A)$  and  $T(B)$ .

### 3 Convex Hull

Covered in [1] CH. 1.1

HW 1

#### 3.1 Geometry of Convex Hull



Figure 1: Example of Convexity

- The computation of **planar convex hull** was one of the first computational geometry problems.
- *Convexity 1:* A subset  $S$  of the plane is **convex** if and only if for any pair of points  $p, q \in S$ , the line segment  $\overline{pq}$  is completely contained in  $S$ . See Fig. 1a. The *convex hull*,  $\mathcal{CH}(S)$ , of a set  $S$  is the smallest convex set that contains  $S$ ; it is the intersection of all convex sets that contain  $S$ .
- *Convexity 2:* How to compute the convex hull of a finite set  $P$  of  $n$  points in the plane? The area enclosed in the shaded region is the convex hull of  $P$ . See Fig. 1b. It is the unique convex polygon whose vertices are points from  $P$ .



Figure 2: computing convex hull

- Fig. 2a: To compute a convex hull of a set of points,  $P = \{p_1, p_2, \dots, p_9\}$ , we compute a list of those vertices from  $P$  that are the vertices of  $\mathcal{CH}(P)$ , i.e.  $\{p_4, p_5, p_8, p_2, p_9\}$ , and list them in clockwise order. Defining  $\mathcal{CH}(P)$  as a convex polygon is more useful than discussing the intersection of all convex sets.
- Fig. 2b: For points  $p$  and  $q$  that are endpoints of an edge and that are in  $P$ , we direct a line through  $p$  and  $q$ , and if  $\mathcal{CH}(P)$  lies to one side, then all points in  $P$  must lie to

that side of the  $\overline{pq}$  line. And if all points of  $P \setminus \{p, q\}$  lie to one side of  $\overline{pq}$ , then  $\overline{pq}$  is an edge of the  $\mathcal{CH}(P)$ .

From Class, Algorithmic **Paradigms** covered:

- Sweepline/Incremental
- Divide & Conquer
- Prune & Search

### 3.2 Algorithm Naive Convex Hull - $O(n^4)$

From class:

For set of points in polygon  $P$ , for every triple  $i, j, k$ , for every point  $l$  not equal to  $i, j, k$ , if  $l \in \Delta ijk$ , discard  $l$ , as point inside the  $\Delta$  can't be on the  $\mathcal{CH}$ . This runs  $O(n^4)$  time ( $n \times n \times n - 1 \times n - 2$ ).

### 3.3 Algorithm Ways to find Left/Right Turn

From class:

$\Theta(1)$  time: You can find the counter clockwise (left) or clockwise (right) turn of an angle given three points but calculating the determinant based on the order of the points that they were given in.

The determinant is twice the area of a triangle. Given points  $A, B, C$ , they form a ccw turn if the determinant is positive; o.w. if the area is negative, it is a right turn. This is verified using basic geometry or linear algebra.

$$\begin{aligned} D &= \begin{vmatrix} x_a & y_a & 1 \\ x_b & y_b & 1 \\ x_c & y_c & 1 \end{vmatrix} = x_a \times \begin{vmatrix} y_b & 1 \\ y_c & 1 \end{vmatrix} - y_a \times \begin{vmatrix} x_b & 1 \\ x_c & 1 \end{vmatrix} + \begin{vmatrix} x_b & y_b \\ x_c & y_c \end{vmatrix} \\ &= x_a(y_b - y_c) - y_a(x_b - x_c) + (x_b y_c - x_c y_b) \\ &= (x_a y_b - x_a y_c) + (x_c y_a - x_b y_a) + (x_b y_c - x_c y_b) \end{aligned}$$

A set of points is a  $\mathcal{CH}$  if all the points turn in the same direction. If one different direction is detected, then the polygon is either complex or concave.

### 3.4 Algorithm SlowConvexHull( $P$ ) - Naive $O(n^3)$

*Input:* A set  $P$  of points in a plane.

*Output:* A list  $\mathcal{L}$  containing vertices of  $\mathcal{CH}(P)$  in clockwise order.

1.  $E \leftarrow \emptyset$
2.  $\forall$  ordered pairs  $(p, q) \in P \times P$ , where  $p \neq q$

3.     **do**  $valid \leftarrow \text{true}$
4.          $\forall r \in P, r \neq p, r \neq q$
5.             **do if**  $r$  lies to the left of directed line from  $p$  to  $q$
6.                 **then**  $valid \leftarrow \text{false}$
7.     **if**  $valid$  **then** Add add directed edge  $\vec{pq}$  to  $E$ .
8. From the set  $E$  of edges, construct a list  $\mathcal{L}$  of vertices of  $\mathcal{CH}(P)$ , sorted in clockwise order.
  - This is “piecewise linear” in finding the edges of a polygon.
  - “Supporting Line” is such that all points of the polygon is on the line or to one side of the line (on the closed half plane).
  - *\*Assume for now that methods to test if a point is to the right or left of a line is available. Assume this primitive operation is  $O(1)$ .*
  - *\*initially ignores the degenerate case, where a point  $r$  may lie on  $\vec{pq}$ .* To consider the degeneracy, must specify that  $\vec{pq}$  is an edge of  $\mathcal{CH}(P)$  if and only if all other  $r \in P$  lie strictly on the right or left of  $\vec{pq}$ , or they lie on the open line segment  $\overline{pq}$ .
  - *Problems with rounding error could arise should coordinates are represented in floating point numbers, leading to unexpected results.*
  - Constructing  $\mathcal{L}$  takes about  $O(n^2)$  time. For an edge  $e_1 \in E$ , take the source and destination points, add them to  $\mathcal{L}$ . Using the destination point of  $e_1$ , find the  $e_2$  that has that as it’s origin, and add  $e_2$ ’s destination point to  $\mathcal{L}$ . Repeat until only one edge is left in  $E$ .
  - Complexity Analysis:
    - Check each of the  $n^2 - n$  pairs of points. For each pair, look at  $n - 2$  other points to see if they lie to one side. Total:  $O(n^3)$ .
    - Constructing  $\mathcal{L}$  is  $O(n^2)$ .
    - Total overall:  $O(n^3)$ .

### 3.5 Algorithm ConvexHull( $P$ ) - Incremental Algorithm $O(n \log n)$

Needs only a sorting method and a method to test if three points can make a right turn.

Briefly: Given the set  $P$  of points on a plane, sort the points  $p_1, \dots, p_n$  ordering them by their x-coordinate. Compute the convex hull vertices on the *upper hull* first, from left to right, from point  $p_1$  to  $p_n$ . Then compute the convex hull vertices of the *lower hull* from right to left, from  $p_n$  to  $p_1$ .

Updating of the upper hull after adding point  $p_i$  is important. Suppose there is a list  $\mathcal{L}_{up}$  containing the left to right upper hull vertices seen thus far,  $\{p_1, \dots, p_{i-1}\}$ . Append  $p_i$  to  $\mathcal{L}_{up}$ .

It is correct if  $p_i$  is the rightmost point so far, and if the last three points in  $\mathcal{L}_{up}$  make a *right* turn. Move on to  $p_{i+1}$  if  $p_i$  can be in the upper hull thus far. If a left turn is made, delete the middle point from the upper hull, and keep rechecking the last three points until a right turn is verified.

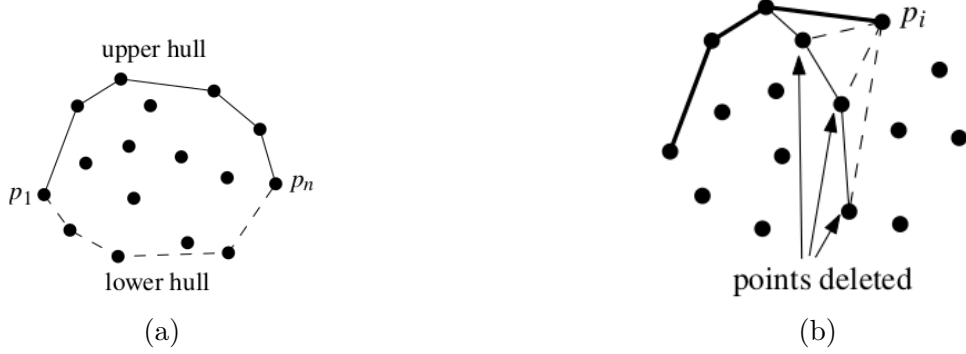


Figure 3: Convex Hull Algorithm

*Input:* A set  $P$  of points in a plane.

*Output:* A list  $\mathcal{L}$  containing vertices of  $\mathcal{CH}(P)$  in clockwise order.

1. Sort the points by x-coordinate, resulting in sequence  $p_1, \dots, p_n$ .
  2. Put  $p_1$  and  $p_2$  in  $\mathcal{L}_{up}$ , with  $p_1$  as the first point.
  3. **for**  $i \leftarrow 3$  **to**  $n$
  4.     **do** Append  $p_i$  to  $\mathcal{L}_{up}$
  5.     **while**  $|\mathcal{L}_{up}| > 2$  **and** the last three points in  $\mathcal{L}_{up}$  don't make a right turn,
  6.         **do** delete the middle of the last three points in  $\mathcal{L}_{up}$
  7. Put points  $p_n$  and  $p_{n-1}$  in  $\mathcal{L}_{low}$ , with  $p_n$  as the first point.
  8. **for**  $i \leftarrow n - 2$  **down to** 1
  9.     **do** Append  $p_i$  to  $\mathcal{L}_{low}$
  10.     **while**  $|\mathcal{L}_{low}| > 2$  **and** the last three points in  $\mathcal{L}_{low}$  don't make a right turn,
  11.         **do** delete the middle of the last three points in  $\mathcal{L}_{low}$
  12. Remove the first and last points from  $\mathcal{L}_{low}$  (avoid duplicate points of where upper and lower hull meet).
  13. Append  $\mathcal{L}_{low}$  to  $\mathcal{L}_{up}$ , call the result  $\mathcal{L}$
  14. **return**  $\mathcal{L}$
- We assumed no two points have the same x-coordinate. To consider that, sort same-x-coord points by their y-coord.



- We will say for three collinear points (make a straight line), they make a left turn.
- Points very close together could create sharp left turns. For these, consider them the same point by rounding.
- Algorithm will compute a closed polygonal chain.
- **Theorem** *The convex hull of a set of  $n$  points in the plane can be computed in  $O(n \log n)$  time.*
- **Proof:** See [1] page 8.
  - Correctness of computation of upper hull (and lower) is proof by induction. Briefly, the set  $\mathcal{L}_{up}$  of  $\{p_1, p_2\}$  is trivially the upper hull.  $\mathcal{L}_{up}$  containing the chain  $\{p_1, \dots, p_{i-1}\}$  is known, by induction to only make right turns, and that all points fall below the chain. When considering point  $p_i$ , we know that  $p_1$  is the smallest point and  $p_i$  will be the biggest point thus far. There can be no points above the old chain, because if there were, then it would have to lie between  $p_{i-1}$  and  $p_i$  in sorted order.
  - Sorting the points can be done in  $O(n \log n)$  time. Computing upper hull is done in  $O(n)$  time, because the for loop is executed a linear number of times, as any extra executions (from the while loop) is bound by  $n$  since extra points can only be deleted once during the hull construction. Similarly, lower hull construction is  $O(n)$  time. Therefore total time for computing convex hull is  $O(n \log n)$ .

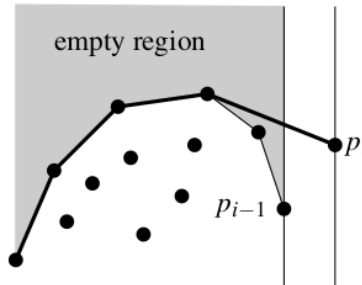


Figure 4: Convex Hull Algorithm - Correctness

### 3.6 Algorithm Jarvis March (Gift Wrapping) - $O(n^2)$

<https://iq.opengenus.org/gift-wrap-jarvis-march-algorithm-convex-hull/>

- Sweepline/Incremental algorithm
- Starting with an extreme point (leftmost/rightmost/etc.), and keep wrapping the points in ccw direction.
- Finding the *next* point involves calculating if the the next candidate makes a ccw orientation with the previous two points. Decide on the directionality, but essentially:
  - Calculate all the slopes from current point with all the other points.

- Pick the point that gave use the minimum slope.
- The min slope point is now the current point. Repeat this process until we return to the first point.
- *You may need to pick a directionality when you get to the other extreme and start to “turn around”.*
- This algorithm is output sensitive. So runtime is  $O(nh)$ , and worst is when  $h = n$ .

### 3.7 Algorithm Graham Scan - $O(n \log n)$

<https://iq.opengenus.org/graham-scan-convex-hull/>

- Uses a stack to remove concavities. Incremental.
- Efficient and can run  $O(n \log n)$
- The runtime rather comes from finding a central position (or even any), calculating all their polar angles (or slopes), and *sorting* them by the angles first. This algorithm comes with a **Pre-processing** step.
- Add the smallest y-coord point to the stack (finding it takes  $O(n)$  time), add the next two points in the order to the stack (pre-processing took  $O(n \log n)$ ). Calculate their orientation. The the next point is a candidate; calculate the orientation of the candidate and top two points in the stack. If you get the same orientation, add canddicate to stack move on to next point; else, pop the stack, and check again. Repeat the popping until you get the orientation desired. Every element is pushed and popped from the stack at most once, and assuming each stack operation takes  $O(1)$ , so forming the hull is  $O(n)$ .

### 3.8 Bridges

- A *bridge* is the support line that forms the edge of  $\mathcal{CH}$  or one that forms a new  $\mathcal{CH}$  edge between two convex polygons when we want to form a new convex hull from their merge.
- Bridge finding is the goal of a divide and conquer method for finding convex hulls.
- Bridge finding takes logarithmic time. And there are 8 cases to examine.

### 3.9 Divide & Conquer Convex Hull Methods

- Typically  $T(n) = 2 T(\frac{n}{2}) + \Theta(n)$ . Outline for an unsorted set of points  $S$ :
  - *assume pre-sort; or sort ( $O(n \log n)$ )*
  - Find  $\mathcal{CH}(S, 1, \frac{n}{2})$
  - Find  $\mathcal{CH}(S, \frac{n}{2} + 1, n)$

- Merge the  $\mathcal{CH}$ 's
- Something like binary search ( $O(\log n)$ ) can be used to find the bridge, for  $T(n) = 2 T(\frac{n}{2}) + \Theta(\log n)$ .
- Some useful data structures, such as **concatenable queue**, can make inserting a point, deleting a point, or finding a point, splitting, joining (implant) take time  $O(\log n)$ .
- 

### 3.10 Algorithm Quick Hull - $O(n^2)$

<https://iq.opengenus.org/quick-hull-convex-hull/>

<https://www.geeksforgeeks.org/quickhull-algorithm-convex-hull/>

- Like quicksort. Divide and Conquer method.
- Find the leftmost and rightmost points.
- Find the highest and lowest points relative to the left-right's slope.
- Form a triangle with the left, right, and lowest points. Discard all points inside triangle.
- Repeat this process with one of the left or right, and the extreme point you picked (i.e. one of the edge of the triangle).
- Best case is  $O(n \log n)$ . Worse case is  $O(n^2)$ .

### 3.11 Algorithm Ultimate Planar Convex Hull (Marriage Before Conquest ) - $O(n \log h)$

<https://iq.opengenus.org/kirkpatrick-seidel-algorithm-convex-hull/>

- Prune & Search. Output sensitive.
- Instead of recursing on each half after splitting the set like in a D&C, this computes how the sets should be merged, then recurse on subsets. This finds upper hull, and then when you find the lower hull, the two sets can be concatenated.
- Briefly: *Given a set, find the median vertical. Next goal is to find the bridge that crosses the median vertical. By definition of  $\mathcal{CH}$ , all points beneath this line is not in the  $\mathcal{CH}$ , so you discard/prune these points. The remaining points are split into the  $L$  and  $R$  subsets, and you recurse on them.*
- The Kirkpatrick-Seidel algorithm:
  1. *don't sort!*
  2. Find the median vertical of the set of points ( $T(n) = T(\frac{n}{5}) + T(\frac{2n}{4}) + \Theta(n) = \Theta(n) = \Theta(n)$ ).

3. PAIRING: Randomly pair up the points, and calculate the misc. slopes. We have  $\frac{n}{2}$  pairs.
4. Find the median slope ( $O(n)$ ),  $m_{\text{median}}$ .
5. Calculate all the y-intercept of all the points if they have slope  $m_{\text{median}}$ . “Sweep slope of  $m_{\text{median}}$ ”, and find the most extreme max y-intercept,  $b_{\text{max}}$ .
6. BridgeFind:  $T_{\text{bridge}}(n) \leq T_{\text{bridge}}(n - \lfloor \frac{n}{4} \rfloor) + O(n) = O(n)$

This is when we start removing points that don't meet our criteria:

- (a) rotate the line  $y_b = m_{\text{median}}x_b + b_{\text{max}}$ . Find the bridge with the first point you hit.
- (b) For the example where  $b_{\text{max}}$  is on the right side and has a negative slope for  $m_{\text{median}}$ , if it has a point that is paired with a point that forms a negative slope, discard the lower point, as you can only ever hit the upper point to be on the hull. Keep all other points.
- (c) More succinctly, for this pruning step, using the extreme point,  $p_{\text{max}}$ , you found:
  - If  $p_{\text{max}}$  is on the *right* of line  $x_{\text{median}}$  (where  $x_{p_{\text{max}}} > x_{\text{median}}$ ), for every line with a slope less than  $m_{\text{median}}$ , discard its lower/right point.
  - If  $p_{\text{max}}$  is on the *left* of line  $x_{\text{median}}$  (where  $x_{p_{\text{max}}} < x_{\text{median}}$ ), for every line with a slope greater than  $m_{\text{median}}$ , discard its upper/left point.
- (d) This process discards  $\frac{1}{4}$  of the points in the set, the pruning.
- (e) Repeat from the PAIRING step until 2 points are left. These are the points of the Bridge.

$$T(n, h) \leq \begin{cases} 0 & \text{if } h = 1 \\ O(n) & \text{if } h = 2 \\ T(\frac{n}{2}, h_l) + T(\frac{n}{2}, h_r) + T_{\text{bridge}}(n) & \text{if } h \geq 3, \\ & \text{where } h_l + h_r = h, h_r, h_l \geq 1 \end{cases}$$

7. Repeat from step 2 for remainder subsets  $L$  and  $R$  on either side of the bridge.

### Proof

*Claim:*  $T_{\text{bridge}}(n) \leq T_{\text{bridge}}(n - \lfloor \frac{n}{4} \rfloor) + O(n)$

Show by induction on size of  $h$  that  $T(n, h) \leq cn \log h$ .

*Basis:* If  $h = 2$  (minimum for top hull of the  $\mathcal{CH}$ ), the top edge has at least 2 points. ( $\log 2 = 1$ ).

$$T(n, 2) = c_1 n \leq cn \log 2 \quad (\text{pick } c \geq c_1)$$

*Inductive Hypothesis:*  $T(n, h) \leq ch \log h$ , for  $h \leq k$ .

*Inductive Step:*

$$\begin{aligned}
T(n, h) &\leq c_1 n + T\left(\frac{n}{2}, h_l\right) + T\left(\frac{n}{2}, h_r\right) \\
&\leq c_1 n + \frac{cn}{2} \log h_l + \frac{cn}{2} \log h_r \\
&\leq c_1 n + \frac{cn}{2} (\log h_l + \log h_r) \\
&\leq c_1 n + \frac{cn}{2} \log(h_l h_r) \\
&\leq c_1 n + \frac{cn}{2} \log(h_l (h - h_l)) \\
&\leq c_1 n + \frac{cn}{2} \log\left(\frac{h}{2}\right)^2 \\
&\leq c_1 n + cn \log\left(\frac{h}{2}\right) \\
&\leq c_1 n + cn \log h - cn \log 2 \\
&\leq cn \log h \quad \text{if } c = c_1
\end{aligned}$$

### 3.12 Algorithm Dynamic $\mathcal{CH}$ / Order Decomposable Problem - $O(\log^2 n)$

<https://www.geeksforgeeks.org/dynamic-convex-hull-adding-points-existing-convex-hull/>

See Dynamic Convex Hull Notes.

See user guide section on this.

- Order Decomposable Problem, Point Inclusion, Detecting Intersection
- This algorithm addresses when a new point is added to the current set we already computed the convex hull for, or delete a point from the set. This avoids recalculating the entire set for changes regarding single points.
- Using a concatenable queue makes bridge finding between two hulls is  $O(\log n)$ ; then we can concatenate the left portion of the left hull with the right portion of the right hull. This relies on sorting the points though, so sorting dominates the runtime at  $O(n \log n)$ , while the merge gives a recurrence of  $T(n) = 2T(\frac{n}{2}) + O(\log n) = O(n)$ .
- Best to use a dynamic (augmented) tree as the data structure. The root contains the points of the  $\mathcal{CH}$ , while the subtrees/node has info about the partial hulls, and can point to another tree. We can retain the info of points that aren't in the  $\mathcal{CH}$ . The leaves are the actual data/points.
- There are 8 cases to examine to find the bridge between hulls  $A$  and  $B$ . See Class Nodes for Dynamic Convex Hull. But succinctly, when examining if line  $\overline{ab}$  could be the bridge, we consider the angles given by  $a$  and its neighbors and by  $b$  and its neighbors ( $\angle a_l a a_r$  and  $\angle b_l b b_r$ ), the angles are relative to the line  $\overline{ab}$ :

1.  $\angle a_l a a_r, \angle b_l b b_r \leq 180^\circ$  (both edges of each angle are above  $\overline{ab}$ )  $\longrightarrow \overline{ab}$  is the bridge.
2.  $\angle b a a_l \geq 180^\circ$  ( $\overline{a a_l}$  falls “below”  $\overline{ab}$ )  $\longrightarrow$  points  $a$  and all to the right of  $a$  (“above”  $\overline{ab}$ ) can’t be the bridge point, discard. The bridge will be found in  $B$  and the left subchain of  $A$  from  $a$ .
3.  $\angle a b b_r \geq 180^\circ$  ( $\overline{b b_r}$  falls “below”  $\overline{ab}$ )  $\longrightarrow$  points  $b$  and all to the left of  $b$  (“above”  $\overline{ab}$ ) can’t be the bridge point, discard. The bridge will be found in  $A$  and the right subchain of  $B$  from  $b$ .
4. Both  $\angle a_l a a_r, \angle a_r a a_l > 90^\circ$  (both  $\overline{a a_r}$  and  $\overline{b b_l}$  fall “below”  $\overline{ab}$ , and their vectors extended intersect at point  $v = (x, y)$ ). Let  $M_A$  be the max x-coord of  $A$ , and let  $m_B$  be the min x-coord of  $B$ . There is a region between  $M_A$  and  $m_B$ :
  - (a)  $x < M_A$  and  $x < m_B$  ( $x$  is to the left of the region), then no points on the left subchain of  $A$  including  $a$  can be a bridge point. Remove those points.
  - (b)  $x > m_B$  and  $x > M_A$  ( $x$  is to the right of the region), then no points on the right subchain of  $B$  including  $b$  can be a bridge point. Remove those points.
  - (c)  $M_A < x < m_B$  ( $x$  is in the region), then no points on the left subchain of  $A$  including  $a$  and no points on the right subchain of  $B$  including  $b$  can be a bridge point. Remove those points.
5. Only  $\overline{b b_l}$  falls below  $\overline{ab}$   $\longrightarrow$  discard the right subchain in  $A$  and  $B$ .
6. Only  $\overline{a a_r}$  falls below  $\overline{ab}$   $\longrightarrow$  discard the left subchain in  $A$  and  $B$ .

### 3.13 3D Convex Hull

Introduced (background) in [1] CH. 11

## References

- [1] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. 2008. Computational Geometry: Algorithms and Applications (3rd ed. ed.). Springer-Verlag TELOS, Santa Clara, CA, USA.
- [2] Hi