Planar convex hulls (II)

Computational Geometry [csci 3250]
Laura Toma
Bowdoin College

Convex Hull

The problem: Given a set $P$ of points in 2D, describe an algorithm to compute their convex hull

output: list of points on the CH (in boundary order)

Convexity

A polygon $P$ is convex if for any $p, q \in P$, the segment $pq$ lies entirely in $P$.

Properties of CH

- All edges of CH are extreme and all extreme edges of $P$ are on the CH
- All points of CH are extreme and all extreme points of $P$ are on the CH
- All internal angles are $< 180$
- Walking counterclockwise $\rightarrow$ left turns
- Points on CH are sorted in radial order wrt a point inside

Outline

- Last time
  - Brute force
  - Gift wrapping
  - Quickhull
  - Graham scan
- Today
  - Andrew’s monotone chain algorithm
  - Exercises
  - Lower bound
  - More algorithms
    - Incremental CH
    - Divide-and-conquer CH
- Andrew’s Monotone Chain Algorithm (1979)
  - Alternative to Graham’s scan
  - Idea: Find upper hull and lower hulls separately
Andrew's Monotone Chain Algorithm (1979)
- Alternative to Graham's scan
- Idea: Find upper hull and lower hulls separately

Goal: find the CH of P
- Idea: Traverse points in (x,y) order (i.e. lexicographically)
Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)

13

Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)

14

Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)

15

Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)

16

Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)

17

Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)

18
Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)

19

Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)

20

Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)

21

Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)

22

Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)

23

Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)

24
Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)

And so on...

Alternative to Graham's scan
- Idea: Traverse points in (x,y) lexicographic order (instead of radial order)
- Runs in sort + scan
- Sorting lexicographically is faster than sorting radially
Convex hull: summary

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time Complexity</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive</td>
<td>$O(n^3)$</td>
<td></td>
</tr>
<tr>
<td>Gift wrapping</td>
<td>$O(nh)$</td>
<td>1970</td>
</tr>
<tr>
<td>Graham scan</td>
<td>$O(n \log n)$</td>
<td>1972</td>
</tr>
<tr>
<td>Quickhull</td>
<td>$O(n^2)$</td>
<td>1977</td>
</tr>
</tbody>
</table>

Can we do better?

What is a lower bound?

- Given an algorithm $A$, its worst-case running time is the largest running time on any input of size $n$:

$$\max_{|P|=n} T(n)$$

Where $T(n)$ is the running time of algorithm $A$ on input $P$.

- A lower bound for $CH$:
  - What is the worst-case running time of the best possible $CH$ algorithm?
  - Consider all possible $CH$ algorithms, and take the overall smallest worst-case running time

$$\min \max_{A} \max_{|P|=n} T(n)$$

Where $T(n)$ is the running time of algorithm $A$ on input $P$.

Lower bounds

Lower bounds depend on the machine model.

- Standard model: decision tree, or comparison model

Lower bounds by reduction

- Assume we can show that: we can sort by an instance of $CH$

**Sorting reduces to $CH$**

- Define function $\text{sort}(A)$
  - create a set $P$ of points from $A$
  - find $CH(P)$
  - form $CH$ infer sorted order of $A$

**Analysis**

- Assuming that the first and last step are $O(n)$, then

$$\text{sort} = \text{find convex hull} + O(n)$$

- If we can find such a reduction, this proves an $\Omega(n \log n)$ lower bound for $CH$

Why?
Assume we are given a set of numbers \(x_1, x_2, \ldots, x_n\) to sort.

Suppose we have an algorithm that constructs the CH of a set of \(n\) points in worst-case \(T(n)\) time.

Our goal is to argue that there exists some instance of a convex hull problem that sorts our numbers.

Input: set of points \(x_1, x_2, \ldots, x_n\).

Form a set of 2D points \((x_i, x^2_i)\).

Run the CH algorithm to construct their convex hull.
• They fall on a parabola, so every point is on the hull

• Find the lowest point on the hull, and walk from in ccw order.
  • This is sorted order!

• Input: set of points $x_1, x_2, \ldots, x_n$
  • Form a set of 2D points $(x_i, x_i^2)$
  • Run the CH algorithm to construct their convex hull.
  • Find the lowest point on the hull, and walk from in ccw order. This is sorted order!
  • If we could find the CH faster than $n \lg n$, then we could sort faster than $n \lg n$! Impossible!

• What we actually proved is that
  • Any CH algorithm that produces the boundary in order must take $\Omega(n \lg n)$ in the worst case.
  • If we did not want the boundary in order, can the CH be constructed faster?
    • It was an open problem for a while.
    • Finally, it was established that a convex hull algorithm, even if it does not produce the boundary in order, still needs $\Omega(n \lg n)$

• Yes, Graham scan is the ultimate CH algorithm but...
  • not output sensitive
  • does not extend to 3D
  • The (wo)search continues
Incremental algorithms

- Goal: solve problem P
- Idea: traverse points one at a time and solve the problem for points seen so far
- Incremental Algorithm
  - Initialize solution $S = \text{initial solution}$
  - for $i = 1$ to $n$
    - $S$ represents solution of $p_1, \ldots, p_{i-1}$
    - add $p_i$ to $S$ and update $S$ to represent solution of $p_1, \ldots, p_i$

Incremental algo for CH

1. $CH = \emptyset$
2. for $i = 1$ to $n$
   - $CH$ represents the CH of $p_1, p_i$
   - add $p_i$ to $CH$ and update $CH$ to represent the CH of $p_1, p_i$
Incremental algo for CH

\[ CH = \{ \} \]

\[ \text{for } i = 1 \text{ to } n \]

\[ \quad \text{//CH represents the CH of } p_1, p_i \]

\[ \quad \text{add } p_i \text{ to CH and update CH to represent the } \text{CH of } p_1, p_i \]

55

56

57

Incremental algo for CH

\[ CH = \{ \} \]

\[ \text{for } i = 1 \text{ to } n \]

\[ \quad \text{//CH represents the CH of } p_1, p_i \]

\[ \quad \text{add } p_i \text{ to CH and update CH to represent the } \text{CH of } p_1, p_i \]

58

59

60
Incremental algo for CH

CH = {}  
for i=1 to n  
    //CH represents the CH of p_1..p_{i-1}  
    add p_i to CH and update CH to represent the CH of p_1..p_i
Incremental algo for CH

- \( \text{CH} = \{ \} \)
- for \( i = 1 \) to \( n \)
  - //CH represents the CH of \( p_1 \ldots p_{i-1} \)
  - add \( p_i \) to CH and update CH to represent the CH of \( p_1 \ldots p_i \)

The basic operation is adding a point to a convex polygon

- CASE 1: \( p \) is in polygon
- CASE 2: \( p \) outside polygon

Issues to solve

- What's a good representation for a polygon?
- We need a point-in-polygon test?
- How to handle CASE 2?

Representing a polygon

A polygon is represented as a list of vertices in boundary order.
(the convention is counter-clockwise order)

```
typedef struct _polygon{
  int k; //number of vertices
  Point* vertices; //the vertices, ccw in boundary order
} Polygon;
```

or

```
Vector<Point> //note: the vertices, ccw in boundary order
```
Point in convex polygon

//return TRUE iff p on the boundary or inside H; H is convex a polygon
bool point_in_polygon(point p, polygon H)

What has to be true in order for p to be inside?

Case 2:

We want to find pi and pj

Hint: Check the orientation of p wrt the edges of the polygon.

Analysis:
- O(k) where k is the size of the polygon

IDEAS?

Case 2:

We want to find pi and pj

Hint: Check the orientation of p wrt the edges of the polygon.

What do you notice? How can we use this to find the tangent points? Sketch an algorithm. How long does it take?
Finding tangent points

Input: point \( p \) outside \( H \)

polygon \( H = [p_0, p_1, \ldots, p_k] \) convex

\[
\text{for } i = 0 \text{ to } k-1 \text{ do} \\
\quad \text{prev} = ((i == 0)? k-1: i-1); \\
\quad \text{next} = (i==k-1)? 0: k+1); \\
\quad \text{if XOR (p is left-or-on (p\text{prev}, p_i), p is left-or-on(p_i, p_{\text{next}}))} \\
\quad \text{then } p_i \text{ is a tangent point}
\]

After finding \( p_i \) and \( p_j \), how would you update \( H \)?

Back to an incremental algorithm for CH

Incremental algo for CH

\[
\text{H} = [p_1, p_2, p_3] \\
\text{for } i = 4 \text{ to } n \text{ do} \\
\quad \text{if point_in_polygon(p, H)} \\
\quad \text{\quad do nothing} \\
\quad \text{else} \\
\quad \quad \text{find } p_i \text{ the tangent point where orientation changes from L to R} \\
\quad \quad \text{find } p_j \text{ the tangent point where orientation changes from R to L} \\
\quad \quad \text{(note: } p_i \text{ not necessarily before } p_j \text{ in the vertex array of } H) \\
\quad \quad \text{cut out the part from } p_i \text{ to } p_j \text{ in } H \text{ (note: view } H \text{ as wrapping around) } \\
\quad \quad \text{and replace it with } \text{vertex } p
\]

Simulate the algorithm on a couple of examples.
Think how \( p_i \) could come before \( p_j \) in \( H \) or the other way around.

Analysis:

\[\text{SUM } O(i) = O(n^2)\]
The "straightforward" incremental algorithm is $O(n^2)$.

Improvement:
- pre-sort the points by their x-coordinates and add them in this order.
- What does this give us?
- It was shown that $O(n \log n)$ incremental algorithm is possible.
- avoid re-computing all directions every time.
- replace the search for tangent points with some sort of binary search.

Incremental algo for CH

A divide-and-conquer algorithm for CH

Divide-and-conquer

```
DC(input P)
if P is small, solve and return
else
    // divide
    divide input P into two halves, P1 and P2
    / merge
    result1 = DC(P1)
    result2 = DC(P2)
    do_something_to_figure_out_result_for_P
    return result
```

Analysis: $T(n) = 2T(n/2) + O(merge\ phase)$
- if merge phase is $O(n)$, $T(n) = 2T(n/2) + O(n) \Rightarrow O(n \log n)$

CH via divide-and-conquer

- find vertical line that splits $P$ in half

CH via divide-and-conquer

- find vertical line that splits $P$ in half
- let $P_1, P_2$ be set of points to the left/right of line
CH via divide-and-conquer
- Find vertical line that splits P in half
- Let P1, P2 be set of points to the left/right of line
- Recursively find CH(P)

Merging two hulls... in linear time
- Need to find the two tangents

Here it looks like the upper tangent is between the top points in P1 and P2
- Is that always true?
Not necessarily...

The top-most point overall is on the CH, but not necessarily on the upper tangent.

Merging two hulls in linear time

- Naive algorithm: try all segments \((a,b)\) with \(a\) in \(H_1\) and \(b\) in \(H_2\).
  - Too slow: \(O(n^2)\) merge, \(O(n^2 \log n)\) CH algorithm.

Finding the lower tangent

- Claim: All points in \(H_1\) and \(H_2\) are to the left of \(ab\).

Finding the lower tangent

- Claim: Points \(a,b\) are on the lower hulls of \(H_1\) and \(H_2\), respectively.

Finding the lower tangent

- Idea:
  - Start with \(a = \) rightmost point in \(H_1\), \(b = \) leftmost point in \(H_2\).
  - Lower \(a\) until all \(H_1\) is left of \(ab\).
  - Lower \(b\) until all \(H_2\) is left of \(ab\).
  - Repeat.
Finding the lower tangent

- Idea:
  - start with \( a \) = rightmost point in \( H_1 \), \( b \) = leftmost point in \( H_2 \)
  - lower \( a \) until all \( H_1 \) is left of \( ab \)
  - lower \( b \) until all \( H_2 \) is left of \( ab \)
  - repeat

Finding the lower tangent

- Idea:
  - start with \( a \) = rightmost point in \( H_1 \), \( b \) = leftmost point in \( H_2 \)
  - lower \( a \) until all \( H_1 \) is left of \( ab \)
  - lower \( b \) until all \( H_2 \) is left of \( ab \)
  - repeat

Finding the lower tangent

- Idea:
  - start with \( a \) = rightmost point in \( H_1 \), \( b \) = leftmost point in \( H_2 \)
  - lower \( a \) until all \( H_1 \) is left of \( ab \)
  - lower \( b \) until all \( H_2 \) is left of \( ab \)
  - repeat

Finding the lower tangent

- Idea:
  - start with \( a \) = rightmost point in \( H_1 \), \( b \) = leftmost point in \( H_2 \)
  - lower \( a \) until all \( H_1 \) is left of \( ab \)
  - lower \( b \) until all \( H_2 \) is left of \( ab \)
  - repeat

(why) does this work?

Claim: At any point during the algorithm, segment \( ab \) cannot intersect the interior of the polygons

- \( a \) cannot move into the upper hull of \( P_1 \), \( b \) cannot move into the upper hull of \( P_2 \)
- The algorithm terminates
- O(n)
CH via divide-and-conquer

- Yet another illustration of divide-and-conquer paradigm
- Runs in \(O(n \log n)\)
- Extends to \(O(n \log h)\)
- Extends nicely to 3D