Computational Geometry


Problems in Application Domains  \[\rightarrow\]  Specific Geometrical Problems  \[\rightarrow\]  Algorithms to solve Geometric Problem

Modifying methods for original problems (implementations)

1970s: Recognized need for CG, progress on ① & ②
Today: "Mastered" ① & ②, not so successful on ③
(this problem is recognized & being addressed now)

Application Domains

Computer Graphics & VR
2D & 3D: intersections, hidden surface elimination, ray tracing
VR: collision detection (intersection)

Robotics
Motion Planning, assembly orderings (part 1 before part 2, etc.)
collision detection, shortest path finding

GIS
Large data sets \[\Rightarrow\] data structures
overlays \[\Rightarrow\] find pts in multiple layers
interpolation \[\Rightarrow\] know values at certain pts, want others (altitude)

CAD/CAM
design 3D objects & manipulate them
\[\Rightarrow\] merge (union), separate, move (subsets)
"design for assembly": test in CAD/CAM model
for ease of assembly, maintenance, etc.
An Example: Convex Hulls in 2d (CH 1)

- A subset S of the plane is convex iff for every p,q ∈ S \( \overline{pq} \) is completely contained in S.
- The convex hull of a set S CH(S) is the smallest convex set containing S. (intersection of all convex sets containing S.)

\[ \Rightarrow \text{points \& nails, rubber band encloses CH (area)} \]

Problem: Compute convex hull of n pts in plane

Alternative def: CH(P) is the unique convex polygon whose vertices are points of P that contains all pts P.

How to compute Convex hull?

- What does this mean?
  - one answer: listing of vertices of P in clockwise order
  e.g. 2, 6, 8, 3, 4

- How to identify edges?
  - points p,q ∈ P and edge \( \overline{pq} \) of CH(P)
  \( \Rightarrow \) all points of P lie on one side of line through \( p+q \).

1 + 2? No...
1 + 4? Yes...
Algorithm 1: Convex Hull \( (P) \)

**input:** A set \( P \) of points in the plane

**output:** A list \( L \) of vertices of \( CH(P) \) in clockwise order

1. \( E \leftarrow \emptyset \)
2. for all ordered pairs \( (p,q) \in P \times P \) \( (p \neq q) \)
3. \( \text{valid} \leftarrow \text{true} \) (an edge)
4. for all \( r \neq p \) and \( r \neq q \) and \( r \in P \)
5. \( \text{if} \) \( (r \) lies to the left of \( \overrightarrow{pq}) \) \( \text{then} \) \( \text{valid} \leftarrow \text{false} \)
6. \( \text{end for} \)
7. \( \text{if} \) \( \text{(valid)} \) \( \text{then} \) add \( \overrightarrow{pq} \) to \( E \)
8. \( \text{end for} \)
9. Construct list \( L \) from edges in \( E \)

**Note:**
- Test in line 5 (point to left of line)
  - **primitive operation**
    - libraries
      - \( O(n) \) time (but not necessarily easy!)

- Constructing \( L \) from \( E \) (line 10)
  - directed edges - match destinations + origins of edges
    - scan + pick out one by one \( O(n^2) \) time.
    - can do in \( O(n \log n) \) but two for loops require \( O(n^3) \) anyway...
    - \( O(n^2) \) pairs of points - for each check \( O(n) \)

- Degeneracies
- Exact arithmetic
- Robustness
Degeneracies (Degenerate Input)

What does our algorithm do with collinear points \(x, y, z\)?

\[ \Rightarrow \text{question is "to the left" test.} \]

- such an input would be called degenerate.

- to be truly correct, algorithm must be able to handle all such "degenerate" cases

\[ \Rightarrow \text{hard! often many cases} \]

- sometimes hard to even be sure have them all

General Position Assumption (Algorithm Specific!)

To make algorithm design simpler, at first often "assume" input has no degeneracies

- after have correct alg, then add capability to handle degenerate cases

- sometimes resulting alg is cumbersome

\[ \Rightarrow \text{would have been better to consider all from the start...} \]

<table>
<thead>
<tr>
<th>general position</th>
<th>Sorting</th>
<th>Convex Hull (our alg.1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique values</td>
<td>1, 9, 7, 2</td>
<td>no 3 collinear pts</td>
</tr>
<tr>
<td>degenerate</td>
<td>allow repeats 1, 9, 1</td>
<td>allow collinear pts</td>
</tr>
</tbody>
</table>

- Symbolic perturbation techniques: perturb input to ensure there are no degeneracies
Robustness (Exact Precision)
(arises during implementation)

What if points x, y, z are so "close" to collinear our algorithm can't distinguish using floating point arithmetic?

\[ \text{e.g.} \]
\[ \begin{align*}
  \text{case 1} & : \text{ } & \text{case 2} & : \\
  P & \quad Q & \quad P & \quad Q \\
  & \quad & \quad & \\
\end{align*} \]

Serious Problem!
\[ \Rightarrow \] can get inconsistent output if don't always get consistent answers.

\[ \text{e.g.} \overrightarrow{PQ}, \overrightarrow{PR} \] may both be put in \( E \) in our alg.
(structural integrity of hull harmed)

Many Different Approaches to this Robustness Problem

- exact arithmetic (slow)
  - many packages available
- algorithm specific approach (faster?)
  - detect & deal w/ inconsistencies explicitly (may not guarantee correct output - whatever that means)
Algorithm 2: Convex Hull (P)

input: a set P of points in the plane
output: a list L of vertices of CH(P) in clockwise order

1. Sort points by x-coord to get \( P_1, P_2, \ldots, P_n \)
2. \( \text{Upper} := \{P_1, P_2\} \)
3. for \( i = 3 \) to \( n \)  
   /* compute upper hull */
4. append \( P_i \) to end of \( \text{Upper} \)
5. while (\( |\text{Upper}| > 2 \) and last 3 pts in \( \text{Upper} \) make right turn)
6. delete middle of last 3 pts in \( \text{Upper} \)
7. end for
8. \( \text{Lower} := \{P_n, P_{n-1}\} \)
9. for \( i = n-2 \) down to 1  
   /* compute lower hull */
10. append \( P_i \) to end of \( \text{Lower} \)
11. while (\( |\text{Lower}| > 2 \) and last 3 pts don't make right turn)
12. delete middle of last 3 pts in \( \text{Lower} \)
13. end for
14. remove 1st and last point from \( \text{Lower} \)
15. \( L := \text{Lower} \) concatenated w/ \( \text{Upper} \)

Notes:

- Incremental algorithm: add input pts one at a time
- Degeneracy: what about if points have duplicate x-coords?  
  ⇒ easy to fix... lexicographic sort - x then y
- 3 colinear pts?  
  ⇒ say false, i.e. not a right turn.
- Robustness?  
  ⇒ rounding errors can cause us to keep/discard pts incorrectly... yes.
  But structural integrity of hull is preserved + often enough.
Running Time

1. sorting - \(O(n \log n)\)

2. upper hull (lower hull will be the same)
   - for loop executed \(n-2\) times (for each pt \(\neq p_i + p_j\))
   - what about while loop?
     - at least once, at most \(n-2\) ...
     - each "extra" time deletes a point
     - each pt can only be deleted once
   - so in total, over all iterations of for loop, the
     while loop is executed \(O(n)\) times
   \(\Rightarrow\) upper hull in \(O(n)\) time

total time = \(O(n \log n)\) \[\# dominated by sorting \]

Correctness

Correctness of upper hull (lower hull analogous)
by induction on \# pts treated
basis \(L_{u2} = \{p_1, p_2\}\) is trivially \(CH\) of \(\{p_1, p_2\}\) \(\checkmark\)
inductive hypothesis: \(L_{u1}\) is \(CH\) of \(\{p_1, p_2, \ldots, p_{i-1}\}\)
inductive step

- add \(p_i\)
- argue that after while loop \(L_{ui}\) is \(CH\) of \(\{p_1, p_2, \ldots, p_i\}\)

Suppose not. We know \(p_1, \ldots, p_{i-1}\) are on or under \(L_{u1}\)
+ \(L_{ui}\) under \(L_{ui}\). \(\Rightarrow\) only place in for pt in slab \(x\).
\(\Rightarrow\) but this is not possible since lexicographic order says none. \(\Box\)