Convex Hull Algorithms

• 2D
  • Basic facts
  • Algorithms: Naïve, Gift wrapping, Graham scan, Quick hull, Divide-and-conquer
  • Lower bound
• 3D
  • Basic facts
  • Algorithms: Gift wrapping, Divide and conquer, incremental
  • Convex hulls in higher dimensions

Convex hull: basic facts

**Problem**: give a set of $n$ points $P$ in the plane, compute its convex hull $CH(P)$.

**Basic facts**:
• $CH(P)$ is a convex polygon with complexity $O(n)$.
• Vertices of $CH(P)$ are a subset of the input points $P$. 
**Naive algorithm**

**Algorithm**
- For each pair of points construct its connecting segment and **supporting line**.
- Find all the segments whose supporting lines divide the plane into two halves, such that one half plane contains **all** the other points.
- Construct the convex hull out of these segments.

**Time complexity**
- All pairs:
  - Check all points for each pair: $O(n)$ each, $O(n^3)$ total.

---

**Triangle Area**

The determinant is twice the area of the triangle whose vertices are the rows of the matrix.
Orientation and point classification

- The area sign indicates the orientation of the points.
- Positive area $\equiv$ counterclockwise orientation $\equiv$ left turn.
- Negative area $\equiv$ clockwise orientation $\equiv$ right turn.
- This test can be used to determine whether a given point is “above” or “below” a given line.

Triangle Area

The determinant is twice the area of the triangle whose vertices are the rows of the matrix.
Orientation and point classification

- The area sign indicates the orientation of the points.
- Positive area $\equiv$ counterclockwise orientation $\equiv$ left turn.
- Negative area $\equiv$ clockwise orientation $\equiv$ right turn.
- This test can be used to determine whether a given point is “above” or “below” a given line.

Possible problems

- Degenerate cases, e.g., 3 collinear points, may harm the correctness of the algorithm. Segments $AB$, $BC$ and $AC$ will all be included in the convex hull.

- Numerical problems – We might conclude that none of the three segments (or a wrong pair of them) belongs to the convex hull.

- **Question:** How is colinearity detected?
General position assumption

• When designing a geometric algorithm, we first make some simplifying assumptions, e.g.:
  • No three collinear points;
  • No two points with the same $x$ or $y$ coordinate;
  • Other configurations: no three points on a circle, …

• Later, we consider the general case:
  • Behavior of algorithm to degenerate cases?
  • Will the correctness be preserved?
  • Will the running time remain the same?
  • Modify/extend algorithm to handle degeneracies

Gift wrapping algorithm

Algorithm:
1. Find the lowest point $p_1$ and its hull edge $e$
2. For each remaining point $p_i$ ($i > 2$) do
   • Compute the CCW angle $\alpha_i$ from the previous hull edge
   • Let $p_j$ be the point with the smallest $\alpha_i$
   • Make $(p_1, p_i)$ the new hull edge

   Rotate counterclockwise a line through $p_1$
   until it touches one of the other points

Time complexity: $O(n^2)$
In fact, the complexity is $O(nh)$,
where $n$ is the input size and $h$ is the hull size.
**Line equation and angle**

- Let \((x_1, y_1)\) and \((x_2, y_2)\) be two points.
- The explicit line equation is:

  \[ y = mx + c \]

  \[ m = \tan \theta \]

- Singularity at \(x_1 = x_2\) (vertical line)

---

**Graham’s scan algorithm**

**Algorithm:**
- Find a point \(p_0\) in the interior of the hull.
- Compute the CCW angle \(\alpha_i\) from \(p_0\) to all other points.
- Sort the points by angle \(\alpha_i\).
- Construct the boundary by scanning the points in the sorted order and performing only “right turns” (trim off “left turns”).

Use a stack to process sorted points

**Time Complexity:** \(O(n \log n)\)

**Question:** How do we check for a right or left turn?
Graham’s scan: complexity analysis

- Sorting – $O(n \log n)$
- $D_i = \text{number of points popped on processing } p_i$,

- Each point is pushed on the stack only once.
- Once a point is popped – it cannot be popped again.
- Hence

Quick hull algorithm

Algorithm:
- Find four extreme points of $P$: highest $a$, lowest $b$, leftmost $c$, rightmost $d$.
- Discard all points in the quadrilateral interior
- Find the hulls of the four triangular regions exterior to the quadrilateral.
- To process triangular regions, find the extreme point in linear time. Two new exterior regions $A, B$ will be formed, each processed separately.
- Recurse until no points are left in the exterior: the convex hull is the union of the exteriors.

Time Complexity:
$T(n) = O(n) + T(\alpha) + T(\beta)$ where $\alpha + \beta = n - 1$
$\Rightarrow T(n-1) + O(n) = O(n^2)$ worst case!
**Divide-and-Conquer**

**Algorithm:**
- Find a point with a median $x$ coordinate in $O(n)$ time
- Partition point set in two halves
- Compute the convex hull of each half (recursive execution)
- Combine the two convex hulls by finding their upper and lower tangents in $O(n)$.

**Time Complexity:** $O(n \log n)$

---

**Finding tangents (1)**
- Two disjoint convex polygons have four tangents that classify them as either being entirely to the left (+) or to the right (−) of the line:
Finding tangents (2)

Lower tangent:
Connect rightmost point of left hull to leftmost point in right hull and “walk” around both boundaries until the lower tangent is reached.

Algorithm: LOWER_TANGENT
- $a$ = rightmost point of $A$.
- $b$ = leftmost point of $B$.
- while $T$ is not lower tangent to both $A$ and $B$ do
  - while $T$ not lower tangent to $A$ do
    - $a = a - 1$
  - while $T$ not lower tangent to $B$ do
    - $b = b + 1$

Claim: Convex hull computation takes $\Theta(n \log n)$

Proof: reduction from Sorting to Convex Hull:
- Given $n$ real values $x_i$, generate $n$ points on the graph of a convex function, e.g. $(x_i, x_i^2)$.
- Compute the (ordered) convex hull of the points.
- The order of the convex hull points is the order of the $x_i$. Complexity(CH)=$\Omega(n \log n)$
  - Since there is a $O(n \log n)$-time algorithm,
  - Complexity(CH)=$\Theta(n \log n)$
Convex hulls in 3D

Input:
Points in 3D

Output: Convex hull

Representation:
Planar subdivision

Convex hull in 3D: properties

**Theorem:** A convex polyhedron with $|V| = n$ vertices has at most $|E| = 3n–6$ edges and $|F| = 2n–4$ faces.

**Proof:** from Euler’s formula for planar graphs:

$$|V| – |E| + |F| = 2$$

Every face has at least 3 edges (triangle)

Every edge is incident to at least 2 faces

$3|F| \leq 2|E| \Rightarrow |E| \leq 3n–6$ and $|F| \leq 2n–4$

- The complexity of the convex hull is $O(n)$.
- The equality holds when all faces are triangles.
- When no three points are on a line and no four on a plane, the faces of the convex hull are all triangles.
Gift wrapping in 3D (1)

**Idea**: generalize the 2D procedure. The wrapping element is a plane instead of a line.

Pivot a plane around the edges of the hull; find the smallest angle of the planes $\Pi_i$ containing segment $ab$ and points $p_i$.

Gift wrapping in 3D (2)

- Form a triangular face containing $a,b,c$ and repeat the operation for edges $bc$ and $ac$.

- Start from lowest edge in the convex hull and “work around and upwards” until the wrap is over.
Gift wrapping in 3D (3)

Algorithm sketch
Maintain a queue of facets and examine their edges in turn, computing for each the smallest angle with it.

Complexity
• $O(n)$ operations are required at each edge to find the minimum angle.
• Each edge is visited at most once.
• Since there are $O(n)$ edges, the complexity is $O(n^2)$.
• In fact, it is $O(n|F|)$, where $|F|$ is the number of facets in the final hull (as in 2D case).

Divide-and-Conquer in 3D

Idea: generalize the 2D procedure. Recursively split the point set into two disjoint sets, compute their hulls and merge them in linear time. Recursion ends when 4 points are left (tetraededron).

Key step: merging two disjoint convex polyhedra in $O(n)$. 
Merging two disjoint convex polyhedra

Idea:
1. Identify the merge boundaries of $A$ and $B$.
2. Create new triangular faces with two vertices from the merge boundary of $A$ and one vertex from the merge boundary of $B$ (and vice-versa).
3. Delete hidden faces of $A$ and $B$.

Remarks:
• The “chain” of hidden faces starts and ends at the merge boundary edges.
• The “band” of new faces has the topology of a cylinder with no caps (it wraps around).

Algorithm sketch
1. Find the lowest new edge of the hull formed by one vertex $a$ of $A$ and one vertex $b$ of $B$.
2. “Pivot” a plane $\Pi$ around edge $ab$ to find the first $p$.
3. Form a triangular face $(abp)$ and repeat with the new pivot edge formed by $p$ and its opposite (either $a$ or $b$)
4. Repeat step 3 until the wrapping is done.
5. Delete the hidden faces by following the faces around $A$ and $B$ whose edges are on the merge boundary.
Find the hull starting edge

How do we find the next point $p$?
How do we know that the faces will not self-intersect?

Plane rotation

**Lemma:** when the plane $\Pi$ is rotated around segment $ab$, the first point encountered $c$, must be adjacent to either $a$ or $b$.

**Proof:** by convexity arguments (omitted here)

Only the neighbors of edge $ab$ need to be tested at each time. Since there are at most $n-1$ neighbors, the overall complexity is $O(n^2)$. 
Merge boundary triangulation

1. Start from lowest hull edge
2. Alternate between left and right merge boundary points, creating triangles.
   
   No search or testing necessary.

   Complexity: $O(n)$.

Overall complexity:
$O(n^2)$

Can we do better?
$O(n \log n)$

Look again at rotation.

Plane rotation – improvement (1)

**Improvement:** the testing of all neighboring points is wasteful and repeats work. Only $O(n)$ vertices should be examined overall (amortized cost).

→ Keep track of *A-winners* and *B-winners*:
  - *A-winner*: the vertex $\alpha$ adjacent to $a$ with smallest angle.
  - *B-winner*: the vertex $\beta$ adjacent to $a$ with smallest angle.

**Lemma:** If $\alpha_i$ is a winner at iteration $i$, the *B-winner* at the next iteration $\beta_{i+1}$ is counterclockwise of $\beta_i$ around $b$ (same for $\alpha$ and $\beta$ reversed –proof omitted).
Divide and Conquer in 3D (end)

- The number of candidate vertices in each loop iteration decreases monotonically. Each edge is examined at most twice \( \rightarrow \) operation takes \( O(n) \).
- Divide-and-conquer recurrence equation:

\[
T(n) = \begin{cases} 
O(1) & \text{if } n = 1 \\
T(n/2) + O(n) & \text{otherwise}
\end{cases}
\]

- **Theorem:** the convex hull of a set of points in space can be computed in optimal \( O(n \log n) \) time and \( O(n) \) space.

\( \rightarrow \) Same complexity as 2D problem!

Incremental convex hull in 3D

**Idea:** incrementally add a point to a convex polyhedra \( P \)

Two cases:
1. The new point is inside \( P \) \( \rightarrow \) do nothing
2. The new point is outside of \( P \) \( \rightarrow \) fix \( P \! \)

Membership test is done in \( O(n) \) time.

What needs to be done to add a point?
Incremental convex hull in 3D

How to update the convex polyhedron:

1. Add faces from edges of the *horizon* to the new point.
2. Remove hidden faces starting from edges in the horizon.

Once we know what the horizon is, both operations can be performed in $O(n)$ time.
Face visibility
The visibility of a face $F$ from a point $p$ can be determined by the signed volume of the tetrahedron formed by three points on the face $abc$ and $p$.

\[ V = \text{Signed Volume} \]

- Visible $V < 0$
- Not visible $V > 0$

Incremental 3D convex hull algorithm
- For all faces, compute the signed volume.
- Keep hidden faces and discard visible faces.
- Edges sharing visible and hidden faces form the horizon.

**Algorithm sketch**

Initialize $CH$ to the tetrahedron $(p_1,p_2,p_3,p_4)$

**For each** remaining point, do

- **For each face** $F$ of $CH$, compute the signed volume $V$ of the tetrahedron formed by $p$ and $F$.
  - Mark $F$ visible if $V<0$
  - If no faces are visible, discard $p$ ($p$ is inside $CH$)
  - Else for each border face $F$ construct cone face.
    - for each visible face $F$, delete $F$. 

59
Complexity and degeneracies

- Overall complexity is $O(n^2)$. Can we do better?
- **Degeneracy**: if points can be coplanar, coplanar triangles will be created. This can be fixed by deleting the shared diagonal, thereby creating a larger face.

![Diagram showing a face $f$ and a point $p_r$.]

Improvement (1)

- Look-ahead computation to make it cheaper to find visible facets.
- Maintain for each facet $f$ of the current convex hull $CH(P_i)$ a conflict set:
  $$P_{\text{conflict}}(f) = \{p_{i+1}, \ldots, p_n\}$$
  containing the points that $f$ can see.
- For each point $p_j, j>i$, maintain the set of facets of $CH(P_i)$ visible from $p_j$, called $F_{\text{conflict}}(p_j)$.
- Point $p_j$ is in conflict with a face $f$ because once $p_j$ is added, the face must be deleted from the convex hull.
Improvement (2)

- We obtain a bipartite graph, called the conflict graph $G$.

- Update the conflict graph when adding $p_i$:
  - Discard neighbors of $p_i$.
  - Add nodes to $G$ for the newly created facets and discard $p_i$.
  - Find conflict list of new facets. All others remain unaffected!

- **Algorithm sketch**: add points $p_1, \ldots, p_n$ sequentially, using the conflict graph $G$ to determine visible faces.

**Complexity**: $O(n \log n)$ expected randomized

---

**Algorithm** `CONVEXHULL(P)`

- **Input**: A set $P$ of $n$ points in three-space.
- **Output**: The convex hull $CH(P)$ of $P$.

1. Find four points $p_1, p_2, p_3, p_4$ in $P$ that form a tetrahedron.
2. $C \leftarrow CH(\{p_1, p_2, p_3, p_4\})$
3. Compute a random permutation $p_5, p_6, \ldots, p_n$ of the remaining points.
4. Initialize the conflict graph $\tilde{G}$ with all visible pairs $(p_i, f)$, where $f$ is a facet of $\tilde{C}$ and $r > 4$.
5. For $r = 5$ to $n$
6.   
7.     
8.     
9.     
10.     
11.     
12.     
13.     
14.     
15.     
16.     
17.     
18.     
19.     
20.     
21. **return** $C$

- **Complexity**: $O(n \log n)$ expected randomized
Convex hulls in higher dimensions

**Problem**: given \( n \) points in \( \mathbb{R}^d \), find their convex hull (also called a convex *polytope*).

- Faces become hyperfaces of dimension 2,3,...,\( d-1 \).
- Hyperfaces form a graph structure where adjacencies between features of dimension \( i \) and \( i-1 \) are stored.
- Some of the previous algorithms “scale up” (applicable in principle) with proper extensions.

**Theorem 1**: the convex hull of \( n \) points in \( d \)-dimensional space has at most \( \text{hyperfaces} \).

**Theorem 2**: the convex hull can be computed with the gift-wrapping algorithm in