Contents

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

Introduction to algorithims 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

1.1 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

1.1.1 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

1.2 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 heierarchical search methods, e.g. divide-and-conquer to sort a set, and then use binary search to find target.

1.3 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).

2 Convex Hull

Covered in [1] CH. 1.1

2.1 Geometry of Convex Hull

Figure 1: Example of Convexity

- The computation of **planar convex hull** was one of the first computational geomtry 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, $P = \{p_1, p_2, ...p_9\}$, 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 $\mathcal{CH}(P)$.

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

Input: A set P of points in a plane.

Output: A list $\mathcal{L}$ cotnaining 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* $\leftarrow$ **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* $\leftarrow$ **false**
7. **if** *valid* **then** Add add directed edge $\overrightarrow{pq}$ to E.
8. From the set E of edges, construct a list $\mathcal{L}$ of vertices of $\mathcal{CH}(P)$, sorted in clockwise order.

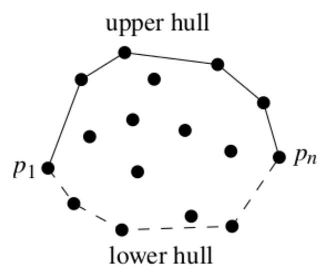
- **Assume for now that methods to test if a point is to the right or left of a line is available. Assume this primative operation is O(1).*
- **initially ignores the degenerate case, where a point r may lie on $\overrightarrow{pq}$.* To consider the degeneracy, must specify that $\overrightarrow{pq}$ is an edge of $\mathcal{CH}(P)$ if and only if all other $r \in P$ lie strictly on the right or left of $\overrightarrow{pq}$, 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: $O(n^3)$.

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

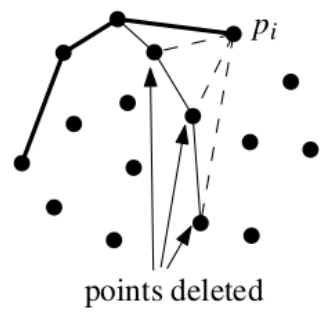
Needs only a sorting method and a method to test if three points can make a right turn.

Briefly: Given the set P of points on a plane, sort the points $p_1, ..., p_n$ 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 copute 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, $\{p_1, ..., p_{i-1}\}$. 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.



(a)



(b)

Figure 3: Convex Hull Algorithm

Input: A set P of points in a plane.

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

1. Sort the points by x-coordinate, resulting in sequence $p_1, ..., p_n$.
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 $\{p_1, ..., p_{i-1}\}$ is known, by induction to only make right turns, and that all points fall below the chain. When considering point p_i, be 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 onces during the hull construction. Similarly, lower hull construction is $O(n)$ time. Therefore total time for computing convex hull is $O(n \log n)$.

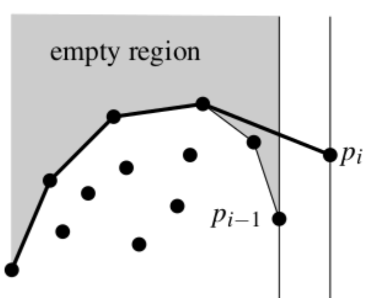


Figure 4: Convex Hull Algorithm - Correctness

2.4 3D Convex Hull

Introduced (background) in [1] CH. 11

3 Hello World

Link to Figures section

```
(; This is a comment. ;)

(; This is a
  multi-lined comment. ;)
```

3.1 Fonts and Styles:

- hello world
- `world world`
- hello world
- HELLO WORLD
- $\Theta(n)$

3.2 Enum List Style:

(a) One

(b) Two

(c) Three

3.3 A basic Table:

hello	world

3.4 Figure

Basic single figure (commented out):

Code environment:

```
int main() {
   cout << "Hello World" << endl;
}
```

One way to put code side-by-side:

```
<animal.h>= // C++
class Animal {
...
};
```

```
<animal.cpp>= // C++
...
```

One way to align multiple (2) figures in a row:

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