CPSC 516 Computational Geometry **Fall 2022**

Lecture 5: September 22

Instructor: William Evans *Scribe: Aryan Tajmir Riahi*

Disclaimer: *Texts written in blue are not official material of this course. They just consist of some after-class thoughts and searchs.*

5.1 Efficient Convex Hull Algorithm

In the last lecture, convex hull problem as well as two convex hull algorithms were introduced e.g., *Graham's Scan* (with complexity $\mathcal{O}(n \log n)$) and *Jarvis March* (with complexity $\mathcal{O}(nh)$, where n denotes the total number of points and h denotes the number of points on convex hull's border. In this lecture the instructor will introduce *Chan*'s algorithm which combines both algorithms and solves the convex hull problem with time complexity $\mathcal{O}(n \log h)$, then takes the first step to prove it's the ultimate convex hull algorithm.

5.1.1 Outline of Algorithm

In this section the outline of *Chan*'s algorithm will be described. For this, they set a new input parameter m, and design an algorithm which returns the convex hull if $h \leq m$ and `FAIL` otherwise. In the following sections we will describe search methods for m. This outline is detailed in Algorithm 1.

Algorithm 1 *Chan*'s Algorithm (outline)

```
    Input a set of points $P = \{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^2$, maximum number of points on the hull $m \in \mathbb{N}$
1: Partition $P$ into $[\frac{n}{m}]$ subsets of at most $m$ points (arbitrarily), $P_1, P_2, \ldots, P_{[\frac{n}{m}]}$
2: Use Graham's Scan algorithm to compute the convex hull of $P_i$ (denoted by $H_i$), for each $1 \leq i \leq [\frac{n}{m}]$
3: Use a generalisation of Jarvis March to wrap $H_i$s and compute $H$ as the convex hull of all points
```

After running *Graham's Scan* on each subset we also will delete the point in $P_i - H_i$. Clearly a point that is not on the convex hull of its own subset is not on the convex hull of all points. Figure 5.1 show an exaple of the situation after running *Graham's Scan*. The key idea of this algorithm is to generalise *Jarvis March* to wrap H_is efficiently. This step will be described in the following section.

5.1.2 Wrapping Convex Polygons

To construct the convex hull assuming our current vertex p, we should find the minimum point with the Left Turn Check regarding p. To do it efficiently we use binary search on H_i (for each $1 \leq i \leq [\frac{n}{m}]$) to find the minimum point in P_i. The steps for an example point and polygon are illustrated in Figure 5.2.

5.1.3 Time Complexity Based on m

To analyze the time complexity of Algorithm 1 note that:

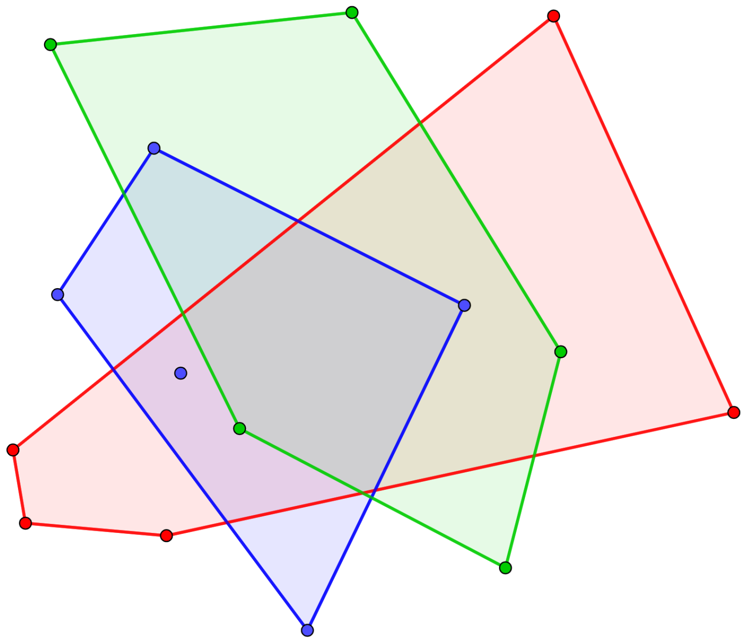

Figure 5.1: An example of the situation after running *Graham's Scan* algorithm .

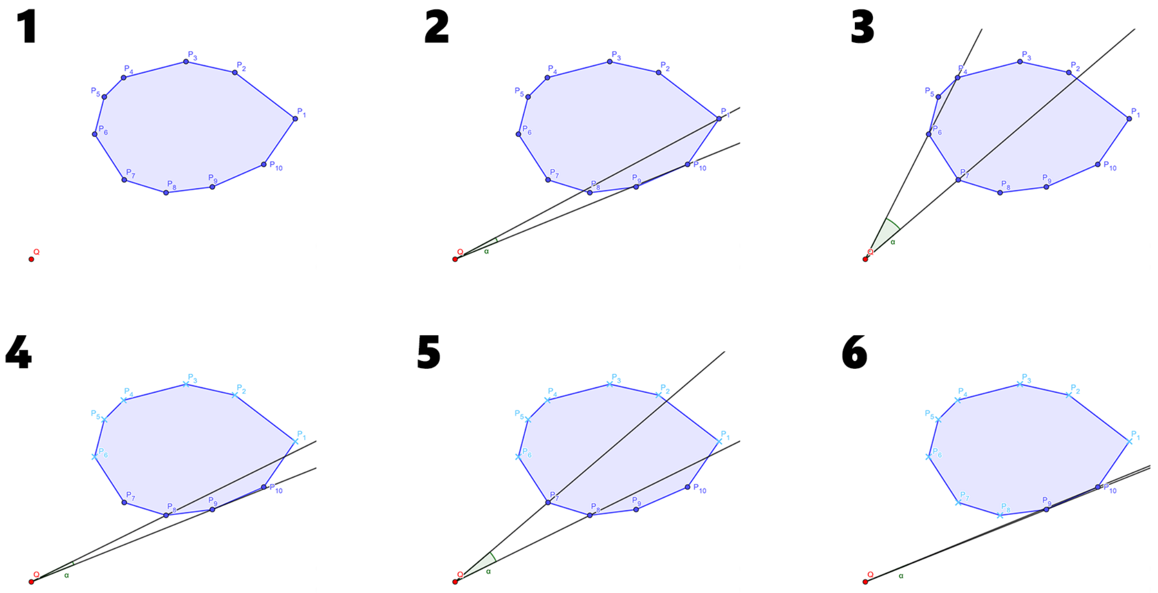

Figure 5.2: An example of the steps of binary search performed to wrap convex hulls (Blue vertices show the current nodes algorithm is taking minimum between, while the cyan points indicates ignored area).

- Step 1: This step clearly takes $\mathcal{O}(1)$.
- Step 2: This step consists of $[\frac{n}{m}]$ instances of *Graham's Scan* algortihm each has time complexity $\mathcal{O}(n \log m)$ which is of $\mathcal{O}([\frac{n}{m}]m \log m) = \mathcal{O}(n \log m)$ in total.
- Step 3: In this step each binary search takes $\mathcal{O}(\log m)$ and there are $[\frac{n}{m}]$ convex polygons so each step of *Jarvis March* algorithm has time complexity $\mathcal{O}([\frac{n}{m}] \log m)$. Note that *Jarvis March* stops after either it reaches a repetitive node (`SUCCESS` case) or after m steps (`FAIL` case). So the total time complexity of this step is $\mathcal{O}(\min(h, m)[\frac{n}{m}] \log m)$.

5.1.4 Search for m

To complete *Chan*'s Algorithm we should introduce a method to search for m bigger than h but close to it (while we don't know h). Note that we can't afford to use a binary search on the whole interval of $1 \leq m \leq n$ as it would give us a factor of $\mathcal{O}(\log n)$ which is not desired in the case of tiny h (we were looking for something of $\mathcal{O}(n \log h)$). So our search method would consist of an increasing guessing sequence of $m_1 = 1 < m_2 < m_3 \ldots$.

5.1.4.1 Try Doubling Search

We try $m_i = 2 \times m_{i-1}$, i.e. $m_i = 2^i$. With this system we will have $m_{\log h} > h$ and the time complexity is:

$$\sum_{i=1}^{\log h} \mathcal{O}(n \log 2^i + m_i[\frac{n}{m_i}] \log 2^i) = \sum_{i=1}^{\log h} \mathcal{O}(n \log 2^i) = \mathcal{O}(n \sum_{i=1}^{\log h} i) = \mathcal{O}(n \log^2 h)$$

Unfortunately this method gives us slightly worse time complexity than what we desired.

5.1.4.2 Grow Faster

To guess more efficiently we try $m_i = m_{i-1}^2$, i.e. $m_i = 2^{2^i}$. In this method $m_{\log \log h} > h$ so the time complexity is:

$$\sum_{i=1}^{\log \log h} \mathcal{O}(n \log 2^{2^i} + m_i[\frac{n}{m_i}] \log 2^{2^i}) = \sum_{i=1}^{\log \log h} \mathcal{O}(n \log 2^{2^i}) = \mathcal{O}(n \sum_{i=1}^{\log \log h} 2^i) = \mathcal{O}(n 2^{\log \log h}) = \mathcal{O}(n \log h)$$

So this method gives us exactly what we want. In fact any guessing sequence where m_i is a polynomial with degree > 1 of m_{i-1} gives us the same time complexity.

5.1.4.3 What If ...?

But what if we grow way faster, like trying $m_i = 2^{m_{i-1}}$? To think of it, for the last guess of m in the worst case we have $m = 2^{h-1}$, so even just considering Step 2 of Algorithm 1 it leaves us $\Omega(n \log m) = \Omega(n \log 2^{h-1}) = \Omega(nh)$. So in the worst case this method fails drastically.

5.2 The Lower Bound for a Convex Hull Algorithm

5.2.1 Algebraic Decision Tree

This computation is used to give a lower bound on time complexity of 2D convex hull algorithms. This model is defined on the Euclidean space, i.e. real values shown by $v \in \mathbb{R}^n$. This computation model consists of a tree just like the comparison based model used for sorting, but in each vertex instead of simple comparison, we compare two algebraic functions of degree d of v and based on the result of the comparison we will move to one the children vertices.

Theorem 5.1. *[Ben-Or 1983] Let W subset$\mathbb{R}^n$ be any subset and let T be any d-th order algebraic decision tree that decides membership in W. If W has m disjoint connected components then T has height atleast $\Omega((\log m) - n)$.*

5.2.1.1 Connected Components

Definition 5.2. *For a $W \subset \mathbb{R}^n$, points $a, b \in W$ are* connected *if and only if there is a continuous function $f : [0, 1] \to W$ with $f(0) = a, f(1) = b$. It is easy to see this notion of* connectivity *defines an equivalence relation on W. The equivalence classes of this relation are called* connected components *of W.*

5.2.2 Multiset Size Verification Problem

In the miultiset size verification problem, a multiset (a set where repetition is allowed) $Z = \{z_1, \dots, z_n\} \in \mathbb{R}^n$ is given and the algorithm should return true if and only if $|Z| = k$.

Claim 5.3. *Define $Z_k = \{Z \in \mathbb{R}^n | \ |Z| = k\}$. Define $X = \{X = (x_1, \dots x_n) \in \mathbb{R}^n | x_i \in \{1, \dots, k\}(1 \leq i \leq n)$ and $x_1, \dots, x_k$ form a permutation of $\{1, 2, \dots, k\}\}$. Then elements of X are in Z_k and pairwise disconnected within Z_k.*

Corollary 5.4. *X has $k! \times k^{n-k}$ elements. Using Theorem 5.1. on it the height of an algebraic decision tree that decides Z_k is at least $\Omega(\log k! \times k^{n-k} - n) = \Omega(k \log k + (n-k) \log k - n) = \Omega(n \log k)$. So an algorithm that solves this problem has time complexity of $\Omega(n \log k)$.*

Proof of Claim 5.3. We say $u = (u_1, \dots, u_n), v = (v_1, \dots, v_n) \in \mathbb{R}^n$ have the same pattern *(an equivalence relation) if and only if we have*

$$u_i = u_j \iff v_i = v_j$$
$$u_i < u_j \iff v_i < v_j$$
$$u_i > u_j \iff v_i > v_j$$

for all $1 \leq i, j \leq n$. Clearly elements of X have pairwise different patterns. Assume there is $a, b \in X \subset Z_k$ where they are connected within Z_k, i.e. there is a continuous $f : [0, 1] \to Z_k$ with $f(0) = a, f(1) = b$. Define $P = \{0 \leq p \leq 1 | f(p)$has not the same pattern as $b\}$. By the assumption we have $a \in P, b \notin P$. So we can define $p^* = \sup P$. Now assuming $f(p^*) = (f(p^*)_1, \dots, f(p^*)_n)$ we can set

$$\delta = \min_{1 \leq i,j \leq n, f(p^*)_i \neq f(p^*)_j} |f(p^*)_i - f(p^*)_j|/2.$$

As f is continuous there is an $\epsilon > 0$ such that

$$\forall p : |p - p^*| < \epsilon \Rightarrow |f(p) - f(p^*)| < \delta \Rightarrow \left\{ \begin{array}{l} f(p^*)_i < f(p^*)_j \Rightarrow f(p)_i < f(p)_j \\ f(p^*)_i > f(p^*)_j \Rightarrow f(p)_i > f(p)_j \end{array} \right. .$$

The last deduction is based on the definition of δ. Using the fact that $f(p)$ and $f(p^*)$ should have exactly k distinct elements we conclude that $f(p^*)_i = f(p^*)_j \Rightarrow f(p)_i = f(p)_j$. Taking it all together $f(p)$ has the same pattern as $f(p^*)$ for all $|p - p^*| < \epsilon$. Now there are two cases:

1. *$f(p^*)$ has the same pattern as b. Then for any $p^* - \epsilon < p < p^*$, based on the aforementioned equation we conclude $f(p)$ has the same pattern as $f(p^*)$ which is the same as b. So this case is in contradiction with the definition of p^* ($p^* - \epsilon$ is an smaller lower bound for P).*

2. *$f(p^*)$ does not have the same pattern as b. Then let $p = p^* + \epsilon/2$ we conclude that $p \in P$ so this case is in contradiction with definition of p^*.*

So Claim 5.3. is proved.