Computatinal Geometry: Theory and Experimentation (2018) Project 2: Convex Hull Computation

Peyman Afshani

September 12, 2018

You have two options when it comes to this project:

1. Do parts (A), (B), (C), and (D). The rest of the project is optional (these are marked with a +).
2. Or you can do parts (A), (E), (F), and (H) and treat the rest of the project as optional. If you choose this, you don't need to run extensive tests; simply confirm that the algorithm runs in $O(n \log n)$ time by running a few experiments.

In this project, you will implement a few convex hull algorithms and you will compare their performance. After implementing each algorithm and making sure that it runs correctly, you must evaluate its performance on some "test cases". You should try to follow sound algorithm engineering principles in doing so (select a good design point, i.e., factors and levels). **For this project, and to reduce your workload, you can run all your experiments on the same machine**.

However, when running these tests, you should pay attention to create a diverse set of "input classes". The first class should be input points that are generated uniformly randomly inside a square, the second point sets generated uniformly randomly inside a circle, and the third point sets whose points lie on the curve $Y = X^2$. You are encouraged to pick additional test classes to improve the quality of your report.

To put your report more inline with sound practices of algorithm engineering, you should think well about "performance metrics" that you wish to measure. The prominent one would be the running time, however, depending on your experiment or your hypothesis, you can measure the running time differently. For example, you can measure the following times separately:

1. The time it takes to read the input
2. The time needed for computation
3. The time for smaller computational tasts (e.g., separate the sorting time in Graham's scan).

In addition, also discuss the following performance metric in your report:

- The number of points on the convex hull

To further increase the quality of your report, you can do the following:

- Come up with the model that describes the number of points on the convex hull for each input class
- Verify your model experimentally

Finally, include any other interesting finding that comes up during your experiments for a fuller and more satisfying report.

Part A. Implement and test the incremental convex hull algorithm (Graham's Scan) discussed in the class (the one discussed in pages 6 and 7 of BKOS). Let's call this algorithm INC_CH.

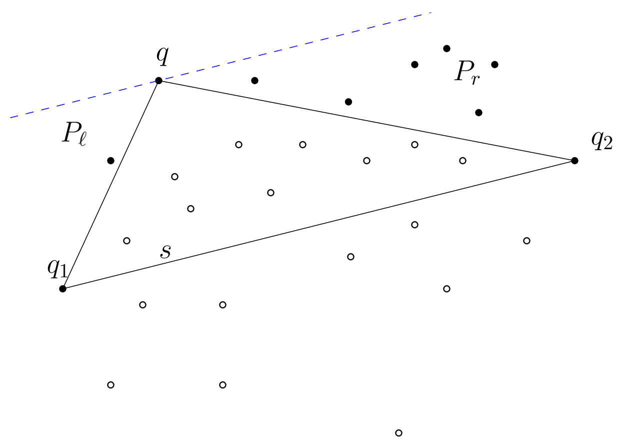

Figure 1: QuickHull finds the upper hull between q_1 and q_2. It is assumed that all the points below the segment $s = \overline{q_1 q_2}$ have been pruned. We find the point furthest away from the line segment s, then we prune all the points inside the triangle $q_1 q_2 s$ and then we recurse on the point sets P_ℓ and P_r that are to the left and right of q respectively.

Part B. Implement the quickhull algorithm, described by the following pseudocode. Let's call this algorithm QH_CH. This algorithm builds the upper hull.

```
· To initialize, find the point $q_1$ with the smallest $x$-coordinate and the point $q_2$ with the largest $x$-coordinate, and form the line segment $s$ by connecting them. Then prune all the points below $s$.
· QuickHull($\overline{q_1 q_2}, P$): // Finds the upper hull of point set $P$ above segment $s = \overline{q_1 q_2}$
1. Find the point $q$ above $s$ that has the largest distance to $s$.
2. Add $q$ to the upper hull
3. Prune all the points inside the triangle formed by $q, q_1$ and $q_2$, and partition the remaining points into two subsets $P_\ell$ and $P_r$ consisting of points that lie to the left of $q$ and points that lie to the right of $q$.
4. Connect $q_1$ to $q$ to form segment $s_1$ and then connect $q_2$ to $q$ to form segment $s_2$.
5. Recurse on QuickHull($s_1, P_\ell$) and QuickHull($s_2, P_r$)
```

Part C. Implement and test the gift-wrapping convex hull algorithm discussed in the class. Let's call this algorithm GIFT_CH. This algorithm finds the full convex hull.

```
· GiftWrapping($P$): // Finds the convex hull of point set $P$
· To initialize, find the point $q_1$ with the smallest $x$-coordinate. Initialize an upwards ray $\overrightarrow{r}$ from $q_1$ (i.e., a vertical line segment with one endpoint being $q_1$ and the other endpoint at $Y = +\infty$). Set the pivot $p$ to be $q_1$.
1. Do the following:
    (a) Iterate over all the points in $P$ and find the point $v$ that minimizes the angle between $\overrightarrow{r}$ and $\overrightarrow{pv}$.
    (b) Add $v$ to the convex hull
    (c) Set $\overrightarrow{r} \leftarrow \overrightarrow{pv}$
    (d) Set $p \leftarrow v$
2. Until $p$ equals $q_1$.
```

Part D. Implement the marriage-before-conquest convex hull algorithm. Follow the pseudocode given below for the upper hull construction (use a similar code for the lower hull). Let's call this algorithm MbC_CH.

1. Find the point with median x coordinate $p_m = (x_m, y)$ and partition the input into two sets P_ℓ and P_r where P_ℓ contains all the points with x-coordinate smaller than x_m and P_r contains the rest of the points.
2. Find the "bridge" over the vertical line $X = x_m$ (i.e., the upper hull edge that intersects line $X = x_m$). You need to implement linear programming for this step. Let (x_i, y_i) and (x_j, y_j) be the left and right end points of the bridge.
3. Prune the points that lie under the line segment $(x_i, y_i), (x_j, y_j)$ (these will be the points whose x-coordinates lie between x_i and x_j.
4. Recursively compute the upper hull of P_ℓ and P_r.

Next, add one more pruning step to the above algorithm and call it MbC2_CH. This extra pruning step is the step 2 in the algorithm below.

1. Find the point with median x coordinate $p_m = (x_m, y)$ and partition the input into two sets P_ℓ and P_r where P_ℓ contains all the points with x-coordinate smaller than x_m and P_r contains the rest of the points.
2. Find the point p_ℓ with the smallest x-coordinate (if there are more than one, take the one with the largest y-coordinate) and the point p_r with the largest x-coordinate (if there are more than one, take the one with the smallest y-coordinate). Note that these can be done at the same time as step 1. Prune all the points that lie under the line segment $p_\ell p_r$.
3. Find the "bridge" over the vertical line $X = x_m$ (i.e., the upper hull edge that intersects line $X = x_m$). Let (x_i, y_i) and (x_j, y_j) be the left and right end points of the bridge.
4. Prune the points that lie under the line segment $(x_i, y_i), (x_j, y_j)$ (these will be the points whose x-coordinate lie between x_i and x_j.
5. Recursively compute the upper hull of P_ℓ and P_r.

Remark. Instead of finding the median of a set S exactly, you can use an approximate median: sample 3 (or 5) random elements from S and then return the median of the sampled elements.

Part E+. Prove that MbC_CH algorithm runs in $O(n \log h)$ time where h is the number of points on the convex hull (hint: look at the recursion tree. Observe that each time you recurse, the number of points halves).

Part F+. Let $x_1 < \cdots < x_h$ be the x-coordinates of the points on the upper hull and let n_i be the number of input points $p = (x, y)$ such that $x_i \le x < x_{i+1}$, $1 \le i < h$. Show that the upper hull computation in MbC_CH runs in time

$$O\left(\sum_{i=1}^{h-1} n_i \log\left(\frac{n}{n_i}\right)\right).$$

Part G$^+$. Implement and measure the performance of Chan, Snoeyick and Yap's convex hull algorithm. Let's call this CSY_CH. For reference, the rough pseudocode of the upper hull construction in this algorithm is the following.

UpperHull(P, p_ℓ, p_r): Here, P is the input point set, p_ℓ is the point with the smallest x-coordinate in P, P_r is the point with the largest x-coordinate in P and the procedure computes the upper hull of P.

1. Prune points below the line segment $p_\ell p_r$.
2. Pair points arbitrarily into $\lfloor n/2 \rfloor$ pairs, (s_i, t_i), $1 \le i \le \lfloor n/2 \rfloor$, in which s_i has smaller x-coordinate than t_i.
3. Find the pair (s_m, t_m) with the median slope among the pairs (you can use the median finding heuristic mentioned in the previous hint).
4. Find the maximal point $p_m = (x_m, y_m)$ in the direction of (s_m, t_m).
5. Partition P into two sets: P_ℓ and P_r in which P_ℓ contains all the points of P with x-coordinate larger than x_m and P_r contains the rest.
6. Prune P_r: if P_r contains a pair (s_i, t_i) with slope larger than the median slope, then prune s_i.
7. Prune P_ℓ: if P_r contains a pair (s_i, t_i) with slope smaller than the median slope, then prune t_i.
8. Recursively compute the upper hull of P_r and P_ℓ.

Part H$^+$. Consider a partition, Δ, of the point set P into k disjoint subsets $P_1, \cdots, P_k$ and let $n_i = |P_i|$, $1 \le i \le k$. We say that Δ is "good" if for every i, $1 \le i \le k$, either $n_i = 1$ or P_i can be placed inside a triangle t_i in such a way that t_i completely lies below the upper hull of P. Define the "entropy" of Δ as

$$H_\Delta = \sum_{i=1}^{k} \frac{n_i \log(\frac{n}{n_i})}{n}.$$

Prove that for any good partition Δ, the upper hull computation in CSY_CH algorithm or MbC2_CH algorithm runs in $O(nH_\Delta)$ time. Prove that this is not the case for MbC_CH algorithm, that is, there exists a point set P and a good partition Δ such that nH_Δ is asymptotically smaller than the time it takes for MbC_CH algorithm to compute the upper hull of P.