Optimal Output-Sensitive Convex Hull Algorithms in Two and Three Dimensions

Timothy M. Chan*
Department of Computer Science
University of British Columbia
Vancouver, B.C. V6T 1Z4, Canada

Abstract

We present simple output-sensitive algorithms that construct the convex hull of a set of n points in two or three dimensions in worst-case optimal $O(n \log h)$ time and $O(n)$ space, where h denotes the number of vertices of the convex hull.

1 Introduction

Given a set P of n points in the Euclidean plane E^2 or Euclidean space E^3, we consider the problem of computing the *convex hull* of P, $\mathrm{conv}(P)$, which is defined as the smallest convex set containing P. The convex hull problem has received considerable attention in computational geometry [11, 21, 23, 25]. In E^2, an algorithm known as *Graham's scan* [15] achieves $O(n \log n)$ running time, and in E^3, an algorithm by Preparata and Hong [24] has the same complexity. These algorithms are optimal in the worst case, but if h, the number of hull vertices, is small, then it is possible to obtain better time bounds. For example, in E^2, a simple algorithm called *Jarvis's march* [19] can construct the convex hull in $O(nh)$ time. This bound was later improved to $O(n \log h)$ by an algorithm due to Kirkpatrick and Seidel [20], who also provided a matching lower bound; a simplification of their algorithm has been recently reported by Chan, Snoeyink, and Yap [2]. In E^3, one can obtain an $O(nh)$-time algorithm using the *gift-wrapping method*, an extension of Jarvis's march originated by Chand and Kapur [3]. A faster but more involved algorithm in E^3 was discovered by Edelsbrunner and Shi [13], having running time $O(n \log^2 h)$. Finally, by derandomizing an algorithm of Clarkson and Shor [8], Chazelle and Matoušek [7] succeeded in attaining optimal $O(n \log h)$ time in E^3. These algorithms, with complexity measured as a function of both n and the "output size" h, are said to be *output-sensitive*.

In this note, we point out a simple output-sensitive convex hull algorithm in E^2 and its extension in E^3, both running in optimal $O(n \log h)$ time. Previous optimal (deterministic) methods, including the algorithm by Kirkpatrick and Seidel and its improvement by Chan, Snoeyink, and Yap, all rely on the existence of a linear-time procedure for finding medians. In Chazelle and Matoušek's three-dimensional algorithm, even more complex tools for derandomization, such as ε-approximations, are used. Our algorithms avoid median-finding and derandomization altogether; Dobkin-Kirkpatrick

*Supported by a Killam Predoctoral Fellowship and an NSERC Postgraduate Scholarship.

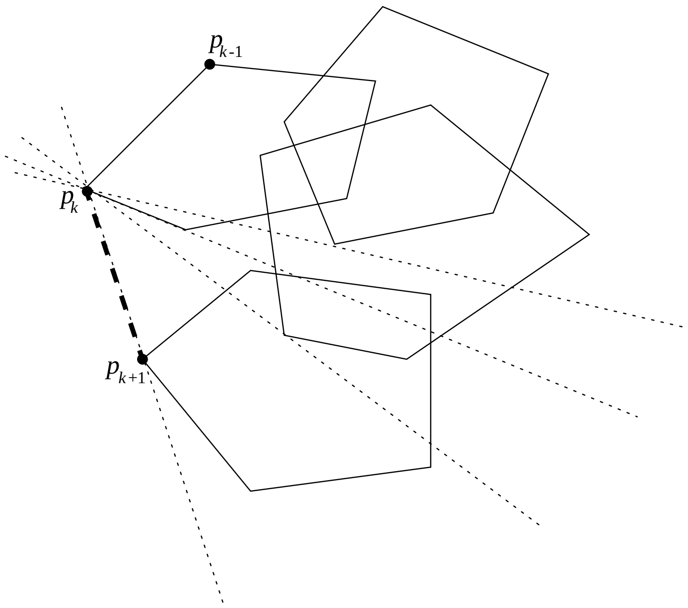

Figure 1: Wrapping a set of $\lceil n/m \rceil$ convex polygons of size m.

hierarchies [9, 10] are the only data structures used in the three-dimensional case. Our idea is to speed up Jarvis's march and the gift-wrapping method by using a very simple *grouping* trick.

2 An Output-Sensitive Algorithm in Two Dimensions

Let $P \subset E^2$ be a set of $n \geq 3$ points. For simplicity, we assume that the points of P are in general position, i.e., no three points are collinear; see Section 4 for how to deal with degenerate point sets.

Recall that *Jarvis's march* [19, 23, 25] computes the h vertices of the convex hull one at a time, in counterclockwise (ccw) order, by a sequence of h *wrapping steps*: if p_{k-1} and p_k are the previous two vertices computed, then the next vertex p_{k+1} is set to be the point $p \in P$ that maximizes the angle $\angle p_{k-1}p_kp$ with $p \neq p_k$. One wrapping step can obviously be done in $O(n)$ time by scanning all n points; with an appropriate initialization the method constructs the entire convex hull in $O(nh)$ time.

We observe that a wrapping step can be done faster if we preprocess the points. Choose a parameter m between 1 and n and partition P into $\lceil n/m \rceil$ groups each of size at most m. Compute the convex hull of each group in $O(m \log m)$ time by, say, Graham's scan [15]. This gives us $\lceil n/m \rceil$ possibly overlapping convex polygons each with at most m vertices, after a preprocessing time of $O(\frac{n}{m}(m \log m)) = O(n \log m)$. Now, a wrapping step can be done by scanning all $\lceil n/m \rceil$ polygons and computing tangents or supporting lines of the polygons through the current vertex p_k, as shown in Figure 1. Since tangent finding takes logarithmic time for a convex polygon by binary or Fibonacci search [5, 25] (the dual problem is to intersect a convex polygon with a ray), the time required for a wrapping step is then $O(\frac{n}{m} \log m)$. As h wrapping steps are needed to compute the hull, the total time of the algorithm becomes $O(n \log m + h(\frac{n}{m} \log m)) = O(n(1 + h/m) \log m)$.

The following is a pseudocode of the algorithm just described. The procedure always runs within $O(n(1 + H/m) \log m)$ time and successfully returns the list of vertices of $\mathrm{conv}(P)$ in ccw order when

$H \geq h$.

Algorithm Hull2D(P, m, H), where $P \subset E^2$, $3 \leq m \leq n$, and $H \geq 1$

partition P into subsets $P_1, \ldots, P_{\lceil n/m \rceil}$ each of size at most m
for $i = 1, \ldots, \lceil n/m \rceil$ do
  compute $\mathrm{conv}(P_i)$ by Graham's scan and store its vertices in an array in ccw order
$p_0 \leftarrow (0, -\infty)$
$p_1 \leftarrow$ the rightmost point of P
for $k = 1, \ldots, H$ do
  for $i = 1, \ldots, \lceil n/m \rceil$ do
    compute the point $q_i \in P_i$ that maximizes $\angle p_{k-1} p_k q_i$ $(q_i \neq p_k)$ by performing a binary search on the vertices of $\mathrm{conv}(P_i)$
  $p_{k+1} \leftarrow$ the point q from $\{q_1, \ldots, q_{\lceil n/m \rceil}\}$ that maximizes $\angle p_{k-1} p_k q$
  if $p_{k+1} = p_1$ then return the list $\langle p_1, \ldots, p_k \rangle$
return *incomplete*

By choosing $m = H$, the complexity of the algorithm is then $O(n(1 + H/m)\log m) = O(n \log H)$. Since the value of h is not known in advance, we use a sequence of H's to "guess" its value as shown below (the same strategy is used in Chazelle and Matoušek's algorithm):

Algorithm Hull2D(P), where $P \subset E^2$

for $t = 1, 2, \ldots$ do
  $L \leftarrow$ Hull2D(P, m, H), where $m = H = \min\{2^{2^t}, n\}$
  if $L \neq$ *incomplete* then return L

The procedure stops with the list of hull vertices as soon as the value of H in the for-loop reaches or exceeds h. The number of iterations in the loop is $\lceil \log\log h \rceil$ (using base-2 logarithms), and the t-th iteration takes $O(n \log H) = O(n 2^t)$ time. Therefore, the total running time of the algorithm is $O(\sum_{t=1}^{\lceil \log\log h \rceil} n 2^t) = O(n 2^{\lceil \log\log h \rceil + 1}) = O(n \log h)$. The storage requirement is clearly linear.

3 An Output-Sensitive Algorithm in Three Dimensions

Let $P \subset E^3$ be a set of $n \geq 4$ points. Again we assume general position, i.e., no four points are coplanar (see Section 4). It suffices to construct the $2h - 4$ facets (triangular faces) of the convex hull; with the aid of a dictionary, we can easily produce the set of h vertices and $3h - 6$ edges together with their adjacency and order information in additional $O(h \log h)$ time.

The higher-dimensional analogue of Jarvis's march is Chand and Kapur's *gift-wrapping method* [3, 25, 26], which computes the hull facets one at a time as follows: from a given facet f, we generate its three adjacent facets f_j by performing a wrapping step about each of the three edges e_j of f $(j = 1, 2, 3)$. Here, a *wrapping step* about e_j is to compute a point $p_j \in P$ that maximizes the angle between f and $\mathrm{conv}(e_j \cup \{p_j\})$ with $p_j \notin e_j$. Since such a step can be done in $O(n)$ time, we can find the facets adjacent to f in $O(n)$ time. Assuming an initial facet f_0 is given (which can be found in two wrapping steps), a breadth-first or depth-first search can then generate all facets of the convex hull. Using a dictionary to detect duplication, we can ensure that each facet is processed once. This implies that the algorithm performs $3(2h-4)$ wrapping steps and thus runs in $O(nh)$ time.

We can use the same grouping idea from the previous section to improve the time complexity to optimal $O(n \log h)$ while maintaining linear space. The calls to Graham's scan (line 3 of Hull2D(P, m, H)) are now replaced by calls to Preparata and Hong's three-dimensional convex hull algorithm [24], which has the same complexity. To make line 8 work in E^3, we need to calculate tangents or supporting planes of convex polyhedra through a given line (or, in dual space, intersect convex polyhedra with a ray). If we use the hierarchical representation of Dobkin and Kirkpatrick [9, 10] to store these polyhedra (which requires only linear-time preprocessing), then the tangents can be computed in logarithmic time each, as before. The analysis is thus identical to that of the two-dimensional algorithm. The pseudocode is as follows:

Algorithm Hull3D(P, m, H), where $P \subset E^3$, $4 \le m \le n$, and $H \ge 1$
1. partition P into subsets $P_1, \ldots, P_{\lceil n/m \rceil}$ each of size at most m
2. for $i = 1, \ldots, \lceil n/m \rceil$ do
3.   compute conv(P_i) by Preparata and Hong's algorithm and store it in a Dobkin-Kirkpatrick hierarchy
4. $F, Q \leftarrow \{f_0\}$, where f_0 is some initial facet of conv(P)
5. for $k = 1, \ldots, 2H - 4$ do
6.   if $Q = \emptyset$ then return F
7.   pick some $f \in Q$ and set $Q \leftarrow Q - \{f\}$
8.   let e_j be the edges of f $(j = 1, 2, 3)$
9.   for $j = 1, 2, 3$ do
10.   for $i = 1, \ldots, \lceil n/m \rceil$ do
11.   compute the point $q_i \in P_i$ that maximizes the angle between f and conv$(e_j \cup \{q_i\})$ by searching the hierarchy of conv(P_i)
12. $p_j \leftarrow$ the point q from $\{q_1, \ldots, q_{\lceil n/m \rceil}\}$ that maximizes the angle between f and conv$(e_j \cup \{q\})$ $(q \notin e_j)$
13. $f_j \leftarrow$ conv$(e_j \cup \{p_j\})$
14.   if $f_j \notin F$ then
15. $F \leftarrow F \cup \{f_j\}$, $Q \leftarrow Q \cup \{f_j\}$
16. return *incomplete*

We can use a queue or a stack to implement Q and a dictionary to implement F. As there are only $O(h)$ dictionary operations, they can be carried out in $O(h \log h)$ time. In fact, more clever implementations of the gift-wrapping method via a shelling order replace the need for dictionaries with just a priority queue.

As before, we choose the group size $m = H$ and guess the value of h with a sequence of H's:

Algorithm Hull3D(P), where $P \subset E^3$
1. for $t = 1, 2, \ldots$ do
2. $L \leftarrow$ Hull3D(P, m, H), where $m = H = \min\{2^{2^t}, n\}$
3.   if $L \neq$ *incomplete* then return L

4 Refinements

In this section, we suggest ideas on possible improvements that may speed up our algorithms in practice; we also discuss how degenerate cases can be handled.

Idea 1. First, points found to be in the interior of $\mathrm{conv}(P_i)$ in line 3 of Hull2D(P, m, H) or Hull3D(P, m, H) can be eliminated from further consideration. This may potentially save work during future iterations of the algorithm, although it does not affect the worst-case complexity.

Idea 2. In Hull2D(P) and Hull3D(P), we choose the group size $m = H$ so as to balance the $O(n \log m)$ preprocessing cost and the $O(H(\frac{n}{m} \log m))$ cost for the $O(H)$ wrapping steps. Alternatively, we can choose $m = \min\{H \log H,\ n\}$ (or set $H = m/\log m$). This choice of m does not affect the former cost except in the lower-order terms, but it reduces the latter cost from $O(n \log H)$ to $O(n)$ and thus results in a smaller constant factor overall.

Idea 3. With Idea 2, the dominant cost of algorithm Hull2D(P, m, H) lies in the preprocessing, i.e., the computation of the convex hulls of the groups in line 3. To reduce this cost, we may consider reusing hulls computed from the previous iteration and merging them as the group size is increased. Suppose m' is the previous group size. Since the convex hull of two convex polygons can be computed in linear time (the dual problem is to intersect two convex polygons), we can compute the convex hull of $\lceil m/m' \rceil$ convex m'-gons in $O(m \log(m/m'))$ time by the standard "mergehull" divide-and-conquer algorithm [25]. Thus, the $\lceil n/m \rceil$ hulls in line 3 can be constructed in $O(n \log(m/m'))$ rather than $O(n \log m)$ time. The same can be said for the three-dimensional case, but merging two convex polyhedra, though possible in linear time [4], is more complicated.

Idea 4. In Hull2D(P), we use the sequence of group sizes $m = 2^{2^t}$, $t = 1, 2, \ldots$, to guess h. The improvements from Ideas 2 and 3 in fact permit us to choose slower growing sequences and still retain optimal $O(n \log h)$ complexity. For example, one possible sequence is simply $m = 2^t$, $t = 2, 3, \ldots$, which corresponds to doubling the group size after each iteration. Note that a coarser sequence approximates h less well while a denser sequence requires more iterations. We may try to optimize the worst-case constant factor and lower-order terms using sequences with different growth rates. We suggest the sequence $m = 2^{t^2}$, $t = 2, 3, \ldots$

Idea 5. E. Welzl has observed that the binary search in line 8 of algorithm Hull2D(P, m, H) can be replaced by a simpler linear search without changing the time complexity of the algorithm. The following monotonicity property provides the justification: during the course of the algorithm, the variable q_i in line 8 can only advance in the ccw direction along $\mathrm{conv}(P_i)$ for each fixed i. As a result, the h-vertex convex hull of p convex polygons with a total of n vertices can be computed in $O(n+hp)$ time by gift-wrapping; the two-polygon ($p = 2$) version of the algorithm is in fact the dual of an intersection algorithm by O'Rourke et al. [22] (see also [23, 25]). The total cost of Hull2D(P, m, H) can then be reduced to $O(n \log m + H(n/m))$ time, which is a $\log m$ factor saving in the second term. Although the overall constant factor is unaffected by the saving if Idea 2 is employed (as the first term is the dominant one), the linear search is easier to implement. There does not seem to be an analogous simplification in three dimensions.

Degeneracies. In both algorithms Hull2D(P, m, H) and Hull3D(P, m, H), we have assumed that the points of P are in general position. One way to cope with degenerate point sets is to apply general perturbation methods such as [12, 14]; however, these methods may cause the output size h to increase, as a point that is not a hull vertex but lies on the hull boundary may become a vertex after perturbation. Thus, it is better to handle the degenerate cases directly. For algorithm Hull2D(P, m, H), this is not difficult to do: when there is more than one point q that maximizes the angle $\angle p_{k-1}p_kq$ in line 9, pick the point q that is farthest from p_k; use the same rule to break ties in line 8.

For algorithm Hull3D(P, m, H), we can do the following: In line 8, let $e_j = \overline{a_jb_j}$ with a_j and b_j oriented in ccw order around f. When there is more than one point q that maximizes the angle between f and $\mathrm{conv}(e_j \cup \{q\})$ in line 12, pick the point q that maximizes the angle $\angle b_ja_jq$; and if there is still more than one q that achieves the maximum, pick the one farthest from a_j. Use the same rule to break ties in line 11. For degenerate point set, it is easier to keep track of edges rather than facets, since facets can be convex polygons rather than triangles. So, make F and Q sets of edges instead, and in line 15, add the oriented edges $\overrightarrow{b_ja_j}$ and $\overrightarrow{a_jq}$ to F and Q. Although we may not have a complete description of the facet incident to these two edges, we know the equation of the plane containing the facet; this equation is sufficient to perform wrapping about these edges.

5 Extensions

We have presented new optimal output-sensitive convex hull algorithms in E^2 and E^3. The algorithms are simpler than previous $O(n \log h)$ algorithms, particularly in the three-dimensional case, and the constant factors behind the big-Oh are likely to be smaller than those of the previous algorithms (in the worst case).

Besides its simplicity, our approach has the advantage that it is applicable to a variety of other problems. As an illustration, consider the problem of computing the *lower envelope* $\mathcal{L}(S)$ of a set S of n line segments in the plane, which we define as the boundary of $\bigcup_{s\in S} \hat{s}$ where $\hat{s}$ denotes the unbounded trapezoid $\mathrm{conv}(s \cup \{(0, +\infty)\})$ for a given segment s. (Convex hulls correspond to lower envelopes of lines in the dual.) Let h be the output size, i.e., the number of edges in the envelope; it is known that h is at most $O(n\alpha(n))$ [16]. Hershberger [17] has given a worst-case optimal algorithm that computes lower envelopes in $O(n \log n)$ time. We now describe how his algorithm can be made output-sensitive with our technique.

First, observe that we can trace the h edges in $\mathcal{L}(S)$ from left to right by performing h *ray shooting operations*, where a ray shooting operation is: given a ray ρ emanating from a point on or beneath $\mathcal{L}(S)$, find the first trapezoid $\hat{s}$ $(s \in S)$ that ρ crosses. As such an operation can be done in $O(n)$ time, this gives us a naïve $O(nh)$ method, like Jarvis's march. To improve the running time, partition S into $\lceil n/m \rceil$ groups each of at most m segments and compute the lower envelope of each group by Hershberger's algorithm; this takes $O(n \log m)$ time in total. Using known data structures such as [6, 18], we can perform ray shooting under each of these $\lceil n/m \rceil$ envelopes in $O(\log m)$ time after $O(m\alpha(m))$ preprocessing (the ray shooting methods can be simplified in our case since envelopes are monotone). This implies that the h ray shooting operations on $\mathcal{L}(S)$ can be done in $O(h(\frac{n}{m} \log m))$ time. Choosing an appropriate group size m and guessing the output size h give us an optimal output-sensitive $O(n \log h)$ algorithm for computing the lower envelope.

Other applications of our technique can be found in [1], including the output-sensitive construction of higher-dimensional convex hulls and k-levels. In many cases, our grouping idea, combined with appropriate data structures, can be used to obtain optimal $O(n \log h)$ algorithms if the output size h is sufficiently small, i.e., if $h = o(n^{\alpha})$ for a suitable constant α.

Acknowledgements

I wish to thank Jack Snoeyink for his great support and encouragement as well as for his many valuable suggestions.

References

[1] T. M. Chan. Output-sensitive results on convex hulls, extreme points, and related problems. *Discrete Comput. Geom.*, this issue.

[2] T. M. Chan, J. Snoeyink, and C.-K. Yap. Output-sensitive construction of polytopes in four dimensions and clipped Voronoi diagrams in three. In *Proc. 6th ACM-SIAM Sympos. on Discrete Algorithms*, 282–291, 1995.

[3] D. R. Chand and S. S. Kapur. An algorithm for convex polytopes. *J. ACM*, 17:78–86, 1970.

[4] B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM J. Comput.*, 21:671–696, 1992.

[5] B. Chazelle and D. P. Dobkin. Intersection of convex objects in two and three dimensions. *J. ACM*, 34:1–27, 1987.

[6] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. In *Proc. 18th Int. Colloq. Automata, Languages, and Programming*, Lect. Notes in Comput. Sci., vol. 510, 661–673, 1991.

[7] B. Chazelle and J. Matoušek. Derandomizing an output-sensitive convex hull algorithm in three dimensions. *Comput. Geom. Theory Appl.*, 5:27–32, 1995.

[8] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[9] D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersection. *Theoretical Comput. Sci.*, 27:241–253, 1983.

[10] D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of preprocessed polyhedra: a unified approach. In *Proc. 17th Int. Colloq. Automata, Languages, and Programming*, Lect. Notes in Comput. Sci., vol. 443, 440–413, 1990.

[11] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Berlin, 1987.

[12] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graphics*, 9:66–104, 1990.

[13] H. Edelsbrunner and W. Shi. An $O(n \log^2 h)$ time algorithm for the three-dimensional convex hull problem. *SIAM J. Comput.*, 20:259–277, 1991.

[14] I. Emiris and J. Canny. An efficient approach to removing geometric degeneracies. In *Proc. 8th ACM Sympos. Comput. Geom.*, 74–82, 1992.

[15] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inform. Process. Lett.*, 1:132–133, 1972.

[16] S. Hart and M. Sharir. Nonlinearity of Davenport-Schinzel sequences and of generalized path compression schemes. *Combinatorica*, 6:151–177, 1986.

[17] J. Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Inform. Process. Lett.*, 33:169–174, 1989.

[18] J. Hershberger and S. Suri. A pedestrian approach to ray shooting: shoot a ray, take a walk. In *Proc. 4th ACM-SIAM Sympos. on Discrete Algorithms*, 54–63, 1993.

[19] R. A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Inform. Process. Lett.*, 2:18–21, 1973.

[20] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM J. Comput.*, 15:287–299, 1986.

[21] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice Hall, New York, 1993.

[22] J. O'Rourke, C.-B. Chien, T. Olson, and D. Naddor. A new linear algorithm for intersecting convex polygons. *Comput. Graph. Image Process.*, 19:384–391, 1982.

[23] J. O'Rourke. *Computational Geometry in C.* Cambridge University Press, 1994.

[24] F. P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM*, 20:87–93, 1977.

[25] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, 1985.

[26] G. F. Swart. Finding the convex hull facet by facet. *J. Algorithms*, 6:17–48, 1985.