Convex Hull of Imprecise Points in $o(n \log n)$ Time after Preprocessing

Esther Ezra* Wolfgang Mulzer†

Abstract

Motivated by the desire to cope with *data imprecision* [8], we study methods for preprocessing a set of planar regions such that whenever we are given a set of points, each of which lies on a distinct region, we can compute a specified structure on these points more efficiently than in "standard settings" (that is, without preprocessing).

In particular, we study the following problem. Given a set L of n lines in the plane, we wish to preprocess L such that later, upon receiving a set P of n points, each of which lies on a distinct line of L, we can construct the convex hull of P efficiently. We show that in quadratic time and space it is possible to construct a data structure on L that enables us to compute the convex hull of any such point set P in $O(n\alpha(n) \log^* n)$ expected time. The analysis applies almost verbatim when L is a set of line-segments, and yields the same asymptotic bounds.

1 Introduction

Most studies in computational geometry rely on an unspoken assumption: whenever we are given a set of input points, their precise locations are available to us. Nowadays, however, the input is often obtained via sensors from the real world, and hence it comes with an inherent imprecision. Accordingly, an increasing effort is being devoted to achieving a better understanding of data imprecision and to developing tools to cope with it (see, e.g., [8] and the references therein). The notion of imprecise data can be formalized in numerous ways [8]. We consider a particular setting that has recently attracted considerable attention (see [2] and the references therein). We are given a set of planar regions, each of which represents an estimate about an input point, and the exact coordinates of the points arrive some time later and need to be processed quickly. This situation could occur, e.g., during a two-phase measuring process: first the sensors quickly obtain a rough estimate of the data, and then they invest considerably more time to find the precise locations. This raises the necessity to preprocess the preliminary (imprecise) locations of the points, and store them in an appropriate data structure, so that when the exact measurements of the points arrive we can efficiently compute a prespecified structure on them. In settings of this kind, we assume that for each input point its corresponding region is known (note that by this assumption we also avoid a point-location overhead). In light of the applications, this is a reasonable assumption, and it can be implemented by, e.g., encoding this information in the ordering of P.

Data imprecision. Previous work has mainly focused on computing a triangulation for the input points. Held and Mitchell [5] were the first to consider this framework, and they obtained optimal bounds for preprocessing disjoint unit disks for point set triangulations, a result that was later generalized by van Kreveld *et al.* [7] to arbitrary disjoint polygonal regions. For *Delaunay* triangulations, Löffler and Snoeyink [10] obtained an optimal result for disjoint unit disks (see also [4, 9]), which was later simplified and generalized by Buchin *et al.* [2] to *fat*[1] and possibly intersecting regions. The preprocessing phase typically takes $O(n \log n)$ time and results in a linear size data structure; the time for finding the structure on the exact point set is usually linear or depends on the complexity (and the fatness) of the input regions.

Since the convex hull can be easily extracted from the Delaunay triangulation in linear time, the same bounds carry over. However, once the regions are not necessarily fat, the techniques in [2, 10] do not yield the aforementioned bounds anymore. In particular, if the regions consist of lines or line-segments, one cannot hope (under certain computational models) to construct the Delaunay triangulation of P in time $o(n \log n)$, regardless of preprocessing (see [10]). Nevertheless, if we are less ambitious and just wish to compute the *convex hull* of P, we can achieve better performance, which is the main problem studied in this paper.

Convex hull. Computing the convex hull of a planar n-point set is perhaps the most fundamental problem in computational geometry, and there are many algorithms available [1]. All these algorithms require $\Theta(n \log n)$ steps, which is optimal in the algebraic computation tree model. However, there are numerous ways to exploit additional information to improve

*Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA; esther@cims.nyu.edu.

†Institut für Informatik, Freie Universität Berlin, 14195 Berlin, Germany; mulzer@inf.fu-berlin.de.

[1]A planar region o is said to be fat if there exist two concentric disks, $D \subseteq o \subseteq D'$, such that the ratio between the radii of D' and D is bounded by some constant.

this bound. For example, if the points are sorted along any fixed direction, Graham's scan takes only linear time [1]. If we know that there are only h points on the hull, the running time reduces to $O(n \log h)$ [6]. Our work shows another setting in which additional information can be used to circumvent the theoretic lower bound.

Our results. We show that we can preprocess the input lines L such that given any set P of points, each of which lies on a distinct line of L, the convex hull $\mathrm{CH}(P)$ can be computed in expected time $O(n\alpha(n)\log^* n)$, where $\alpha(\cdot)$ is the (slowly growing) inverse Ackermann function [12, Chapter 2.1]. Our data structure has quadratic preprocessing time and storage, and the convex hull algorithm is based on a batched randomized incremental construction similar to Seidel's tracing technique [11]. As part of the construction, we repeatedly trace the *zone* of (the boundary of) an intermediate hull in the *arrangement* of the input lines.[2] The fact that the complexity of the zone is only $O(n\alpha(n))$ [12], and that it can be computed in the same asymptotic time bound (after having the arrangement at hand), is a key property of our solution. The analysis applies almost verbatim when L is a set of line-segments, and we obtain similar asymptotic bounds.

2 Convex Hulls

Preliminaries The input at the preprocessing stage is a set L of n lines in the plane. A *query* to the resulting data structure consists of any point set P such that each point lies on a distinct line in L, and for every point we are given its corresponding line. For simplicity, and without loss of generality, we assume that both L and P are in *general position* (see, e.g., [1, 12]). We denote by $\mathrm{CH}(P)$ the convex hull of P, and by $E(P)$ the edges of $\mathrm{CH}(P)$. We represent the vertices of $\mathrm{CH}(P)$ in clockwise order, and we direct each edge $e \in E(P)$ such that $\mathrm{CH}(P)$ lies to its right. Given a subset $Q \subset P$, a point $p \in P \setminus Q$, and an edge $e \in E(Q)$, we say that e is in *conflict* with p if p lies to the left of the line supported by e. The set of all points in $P \setminus Q$ in conflict with e is called the *conflict list* C_e of e, and the size of C_e is called the *conflict size* c_e of e.

2.1 The Construction

Preprocessing. We construct in $O(n^2)$ time (and storage) the *arrangement* $\mathcal{A}(L)$ of L, and produce its *vertical decomposition*, that is, we erect an upward and a downward vertical rays through each vertex v of $\mathcal{A}(L)$ until they meet some line of L (not defining v), or else extend to ∞. The *complexity* of a face f in $\mathcal{A}(L)$ is the number of edges incident to f. The *zone* of a curve γ consists of all faces that intersect γ, and the complexity of the zone is the sum of their complexities.

Queries. Given an exact point set $P = \{p_1, \ldots, p_n\}$ as described above, we obtain $\mathrm{CH}(P)$ through a *batched* randomized incremental construction as follows: Let $P_1 \subseteq P_2 \subseteq \cdots \subseteq P_{\log^* n} = P$ be a sequence of subsets, where P_{k-1} is a random sample of P_k of size $z_{k-1} := \min\{\lfloor n/\log^{(k-1)} n \rfloor, n\}$, for $k = 2, \ldots, \log^* n$. Here, $\log^{(i)} n$ is the ith iterated logarithm: $\log^{(0)} n = n$ and $\log^{(k)} n = \log(\log^{(k-1)} n)$. This sequence of subsets is called a *gradation*. The idea is to construct $\mathrm{CH}(P_1)$, $\mathrm{CH}(P_2)$, …, $\mathrm{CH}(P_{\log^* n})$ one by one. First, we have $|P_1| = O(n/\log n)$, so it takes $O(n)$ time to find $\mathrm{CH}(P_1)$, using, e.g., Graham's scan [1]. Then, for $k = 2, \ldots, \log^* n$, we incrementally construct $\mathrm{CH}(P_k)$ by updating $\mathrm{CH}(P_{k-1})$. This basic technique was introduced by Seidel [11] and it has later found many more applications.

To construct $\mathrm{CH}(P_k)$ from $\mathrm{CH}(P_{k-1})$, we use the data structure from the preprocessing to quickly construct the conflict lists of the edges in $E(P_{k-1})$ with respect to P_k. In the standard Clarkson-Shor randomized incremental construction [3] it takes $O(n \log n)$ time to maintain the conflict lists. However, once we have the arrangement $\mathcal{A}(L)$ at hand, this can be done significantly faster. In fact, we use a refinement of the conflict lists: we shoot an upward vertical ray from each point on the upper hull of P_{k-1}, and a downward vertical ray from each point on the lower hull. Furthermore, we erect vertical walls through the leftmost and the rightmost points of $\mathrm{CH}(P_{k-1})$. This partitions the complement of $\mathrm{CH}(P_{k-1})$ into vertical slabs $S(e)$, for each edge $e \in E(P_{k-1})$, and two boundary slabs $S(v_l)$, $S(v_r)$, associated with the respective leftmost and rightmost vertices v_l and v_r of $\mathrm{CH}(P_{k-1})$. The *refined conflict list* of e, C_e^*, is defined as $C_e^* := (P_k \setminus P_{k-1}) \cap S(e)$. We add to this collection the sets $C_{v_l}^* := (P_k \setminus P_{k-1}) \cap S(v_l)$ and $C_{v_r}^* := (P_k \setminus P_{k-1}) \cap S(v_r)$, which we call the refined conflict lists of v_l and v_r, respectively. Note that $C_e^* \subseteq C_e$, for every $e \in E(P_{k-1})$. Moreover, $C_{v_l}^*$ (resp., $C_{v_r}^*$) is contained in $C_{e_1} \cup C_{e_2}$, where $e_1, e_2 \in E(P_{k-1})$ are the two respective edges emanating from v_l (resp., v_r); see Figure 1(a). We now state a key property of the conflict lists C_e (this property is fairly standard and follows from related studies [3]):

Lemma 1 *Let Q be a planar m-point set, r a positive integer satisfying $1 \le r \le m$, and $R \subseteq Q$ a random subset of size r. Suppose that $f(\cdot)$ is a monotone non-decreasing function, so that $f(x)/x^c$ is decreasing, for*

[2]The arrangement of a set of planar lines is the decomposition of the plane into *vertices*, *edges* and *faces* (also called *cells*), each being a maximal connected set contained in the intersection of at most two lines and not meeting any other line.

some constant $c > 0$. *Then* $\mathbf{Exp}\left[\sum_{e \in E(R)} f(c_e)\right] = O\left(r \cdot f\left(\frac{m}{r}\right)\right)$, *where* c_e *is the number of points* $p \in Q \setminus R$ *in conflict with* $e \in E(R)$. $\square$

Constructing the refined conflict lists. We next present how to construct the refined conflict lists at the k-th round of the algorithm. We first construct, in a preprocessing step, the refined conflict lists $C^*_{v_l}$, $C^*_{v_r}$ in overall $O(z_k)$ time. For the sake of the analysis, we eliminate these points from P_k for the time being, and continue processing them only at the final step of the construction—see below.

Let $\mathrm{UH}(P_{k-1})$ be the upper hull of P_{k-1}, and let $\mathrm{LH}(P_{k-1})$ be its lower hull. Having these structures at hand, we construct the zones of $\mathrm{UH}(P_{k-1})$ and $\mathrm{LH}(P_{k-1})$ in $\mathcal{A}(L)$. This takes overall $O(n\alpha(n))$ time, using the vertical decomposition of $\mathcal{A}(L)$ and the fact that the zone complexity of a convex curve in a planar arrangement of n lines is $O(n\alpha(n))$; see [12, Theorem 5.11].

As soon as we have the zones as above, we can determine for each line $\ell \in L$ the edges $e \in E(P_{k-1})$ that ℓ intersects (if any). Let L_1 be the lines that intersect $\mathrm{CH}(P_{k-1})$, and put $L_2 := L \setminus L_1$. (At this stage of the analysis, we ignore all lines corresponding to the points in P_k that were eliminated at the time we processed $C^*_{v_l}$, $C^*_{v_r}$.)

Next, we wish to find, for each point $p \in P_k \setminus P_{k-1}$ the edges in $E(P_{k-1})$ in conflict with p. If p lies inside $\mathrm{CH}(P_{k-1})$, there are no conflicts. Otherwise, we efficiently find an edge $e_p \in E(P_{k-1})$ visible from p, whence we search for the slab $S(e^*_p)$ containing p—see below.

Let us first consider the points on the lines in L_1. Fix a line $\ell \in L_1$, let $p \in P$ be the point on ℓ, and let q_1, q_2 be the intersections between ℓ and the boundary of $\mathrm{CH}(P_{k-1})$. The points q_1, q_2 subdivide ℓ into two rays ρ_1, ρ_2, and the line segment $\overline{q_1q_2}$. By convexity, $\overline{q_1q_2} \subseteq \mathrm{CH}(P_{k-1})$ and the rays ρ_1, ρ_2 lie outside $\mathrm{CH}(P_{k-1})$. Hence, if p lies on $\overline{q_1q_2}$, it must be contained in $\mathrm{CH}(P_k)$. Otherwise, p sees an edge of $E(P_{k-1})$ that meets one of the rays ρ_1, ρ_2, and we thus set e_p to be this edge (which can be determined in constant time); see Figure 1(b).

We next process the lines in L_2. Note that all points on the lines in L_2 conflict with at least one edge in $E(P_{k-1})$, since no line in L_2 meets $\mathrm{CH}(P_{k-1})$. To find these edges we determine for each $\ell \in L_2$ a vertex p_ℓ on the boundary of $\mathrm{CH}(P_{k-1})$ that is extreme for ℓ.[3] This can be done in total time $O(n)$ by ordering $E(P_{k-1})$ and L_2 according to their slopes (the latter being performed during preprocessing), and then merging these two lists in linear time. Next, fix such a line $\ell \in L_2$, and let $p \in \ell$ be a query point, then p must see one of the two edges in $E(P_{k-1})$ incident to p_ℓ (which can be determined in constant time given p_ℓ), and we thus set e_p to be the corresponding edge; see Figure 1(c).

[3]By this we mean that p_ℓ is extremal in the direction of the outer normal of the halfplane that is bounded by ℓ and contains $\mathrm{CH}(P_{k-1})$.

We are now ready to determine, for each point $p \in P_k$ outside $\mathrm{CH}(P_{k-1})$, the slab $S(e^*_p)$ that contains it (note that e^*_p must be vertically visible from p). If e_p is vertically visible from p, we set $e^*_p := e_p$. Otherwise, we walk along (the boundary of) $\mathrm{CH}(P_{k-1})$, starting from e_p and progressing in an appropriate direction (uniquely determined by p and e_p), until the appropriate slab is found. Using cross pointers between the edges and the points, we can thus easily compute C^*_e for each $e \in E(P_{k-1})$. By construction, all traversed edges are in conflict with p, and thus the overall time for this procedure is proportional to the total size of the conflict lists C_e. Recalling that $c_e = |C_e|$, we obtain

$$\mathbf{Exp}\Big[\sum_{e \in E(P_{k-1})} c_e\Big] = O(z_k) = O(n),$$

by Lemma 1 with $f : m \mapsto m$. This concludes the construction of the refined conflict lists.

Computing $\mathrm{CH}(P_k)$**.** We next describe how to construct the upper hull of P_k, the analysis for the lower hull is analogous. Let $\langle e_1, \ldots, e_s \rangle$ be the edges along the upper hull of P_{k-1}, ordered from left to right. For each e_i, we sort the points in $C^*_{e_i}$ according to their x-order, using, e.g., merge sort, as well as the points in $C^*_{v_l}$, $C^*_{v_r}$. We then concatenate the sorted lists $C^*_{v_l}, C^*_{e_1}, C^*_{e_2}, \ldots, C^*_{e_s}, C^*_{v_r}$, and merge the result with the vertices of the upper hull of P_{k-1}. Call the resulting list Q, and use Graham's scan to find the upper hull of Q in time $O(|Q|)$. This is also the upper hull of P_k. Applying once again Lemma 1 with $f : m \mapsto m \log m$, and putting $c^*_e := |C^*_e|$, $c^*_{v_l} := |C^*_{v_l}|$, $c^*_{v_r} := |C^*_{v_r}|$, and $A > 0$ an absolute constant, the overall expected running time of this step is bounded by

$$\begin{aligned}
&\mathbf{Exp}\Big[A \cdot (c^*_{v_l} \log c^*_{v_l} + c^*_{v_r} \log c^*_{v_r} + \sum_{e \in E(P_{k-1})} c^*_e \log c^*_e)\Big] \\
&\le \mathbf{Exp}\Big[3A \cdot \sum_{e \in E(P_{k-1})} c_e \log c_e\Big] \\
&= O\left(z_k \log\left(z_k / z_{k-1}\right)\right) = O(n),
\end{aligned}$$

because by definition $C^*_e \subseteq C_e$, so $c^*_e \le c_e$, and $c^*_{v_l} \le c_{e_1} + c_{e_2}$ for two edges e_1, e_2 (and similarly for $c^*_{v_r}$). In total, we obtain that the expected time to construct $\mathrm{CH}(P_k)$ given $\mathrm{CH}(P_{k-1})$ is $O(n\alpha(n))$, and since there are $\log^* n$ iterations, the total running time is $O(n\alpha(n)\log^* n)$. We have thus shown:

Theorem 2 *Using* $O(n^2)$ *space and time, we can preprocess a set* L *of* n *lines in the plane, such that given*

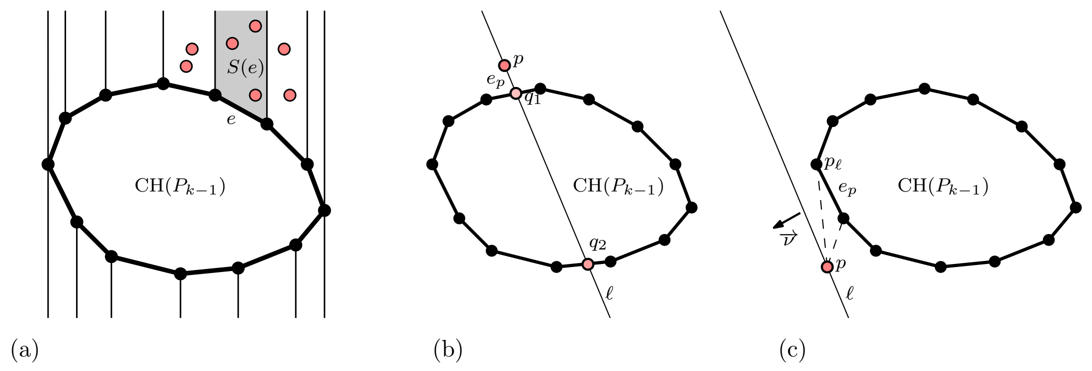

Figure 1: (a) The conflict list C_e of the edge $e \in E(P_{k-1})$ contains all the lightly-shaded points, whereas the refined conflict list $C^*(e)$ has only those points in the vertical slab $S(e)$; (b–c) The edge e_p of $E(P_{k-1})$ is visible to p when (b) ℓ intersects $\mathrm{CH}(P_{k-1})$, or (c) ℓ does not meet $\mathrm{CH}(P_{k-1})$. In this case p_ℓ is an extreme vertex in the direction $\overrightarrow{\nu}$, and the two dashed lines depict the visibility lines between p and the two respective endpoints of e_p.

any point set P with each point lying on a distinct line in L, we can construct $\mathrm{CH}(P)$ *in expected time* $O(n\alpha(n)\log^* n)$.

We note that the analysis proceeds almost verbatim when L is just a set of *line-segments* in the plane. In this case, we preprocess the lines containing the input segments, and proceed as in the original problem. Omitting the straightforward details, we obtain:

Corollary 3 *If L is a set of n line-segments, we can preprocess L in $O(n^2)$ space and time, such that given a point set P with each point lying on a distinct segment of L, we can construct* $\mathrm{CH}(P)$ *in expected time* $O(n\alpha(n)\log^* n)$.

Further results. In the full version of the paper, we show that under the "obliviousness assumption" we can improve the expected running time of our algorithm to $O(n\alpha(n))$, while leaving the preprocessing and storage time unchanged. The obliviousness assumption states that the random choices of the preprocessing phase are unknown to the adversary and that the inputs are chosen in a manner oblivious of the current data structure. We also show how to make the algorithm output sensitive, and how to trade off the space requirement for an increased running time.

We can extend our result to computing k-sets, and we achieve an improved running time as long as $k = o(\log n)$. Finally, we provide lower bounds in the algebraic computation tree model for related problems, such as preprocessing lines in the plane for sorting a point set according to its x order, where each point in the set lies on a distinct line; and preprocessing a set of planes in $\mathbb{R}^3$ for computing the convex hull of a point set with each point incident to a distinct plane. In all these cases preprocessing does not help to beat the $\Theta(n \log n)$ running time.

Acknowledgments The authors wish to thank Maarten Löffler for suggesting the problem, and Boris Aronov and Timothy Chan for helpful discussions. E. Ezra was supported by a PSC-CUNY Research Award; W. Mulzer was supported in part by NSF grant CCF-0634958, NSF CCF 083279, and a Wallace Memorial Fellowship in Engineering.

References

[1] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational geometry: algorithms and applications.* Springer-Verlag, Berlin, third edition, 2008.

[2] K. Buchin, M. Löffler, P. Morin, and W. Mulzer. Preprocessing imprecise points for Delaunay triangulation: Simplified and extended. *Algorithmica.* To appear.

[3] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry. II. *DCG*, 4(5):387–421, 1989.

[4] O. Devillers. Delaunay triangulation of imprecise points, preprocess and actually get a fast query time. Technical Report 7299, INRIA, 2010.

[5] M. Held and J. S. B. Mitchell. Triangulating input-constrained planar point sets. *IPL*, 109(1):54–56, 2008.

[6] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SICOMP*, 15(1):287–299, 1986.

[7] M. J. van Kreveld, M. Löffler, and J. S. B. Mitchell. Preprocessing imprecise points and splitting triangulations. In *Proc. 19th ISAAC*, pages 544–555, 2008.

[8] M. Löffler. *Data Imprecision in Computational Geometry.* PhD thesis, Utrecht University, 2009.

[9] M. Löffler and W. Mulzer. Triangulating the square: quadtrees and Delaunay triangulations are equivalent. Proc. 22nd SODA. To appear., 2011.

[10] M. Löffler and J. Snoeyink. Delaunay triangulation of imprecise points in linear time after preprocessing. *CGTA*, 43(3):234–242, 2010.

[11] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *CGTA*, 1(1):51–64, 1991.

[12] M. Sharir and P. K. Agarwal. *Davenport-Schinzel sequences and their geometric applications.* Cambridge University Press, New York, NY, USA, 1995.