Unions of Onions

Maarten Löffler[*] Wolfgang Mulzer[†]

Abstract

Let $\mathcal{D}$ be a set of n pairwise disjoint unit disks in the plane. We describe how to build a data structure for $\mathcal{D}$ so that for any point set P containing exactly one point from each disk, we can quickly find the onion decomposition (convex layers) of P.

Our data structure can be built in $O(n \log n)$ expected time and has linear size. Given P, we can find its onion decomposition in $O(n \log k)$ time, where k is the number of layers. We also provide a lower bound showing that the running time must depend on k.

Our solution is based on a recursive space decomposition, combined with a fast algorithm to compute the union of two disjoint onion decompositions.

1 Introduction

Let P be a planar n-point set. Take the convex hull of P and remove it; repeat until P becomes empty. This process is called *onion peeling*, and the resulting decomposition of P into convex polygons is the *onion decomposition*, or *onion* for short, of P. It can be computed in $O(n \log n)$ time [4]. Onions provide a natural, more robust, generalization of the convex hull, and they have applications in pattern recognition, statistics, and planar halfspace range searching [5, 9].

Recently, a new paradigm has emerged for modeling data imprecision. Suppose we need to compute some interesting property of a planar point set. Suppose further that we have some advance knowledge about the possible locations of the points, e.g., from an imprecise sensor measurement. We would like to preprocess this information, so that once the precise inputs are available, we can obtain our structure faster. Many problems have been considered in this model, e.g., point set triangulation, Voronoi diagrams, and convex hulls [2, 6–8, 11, 12]. We will study the complexity of computing onions in this framework.

1.1 Results

We begin by showing that the union of two disjoint onions can be computed in $O(n+k^2 \log n)$ time, where k is the number of layers in the resulting onion.

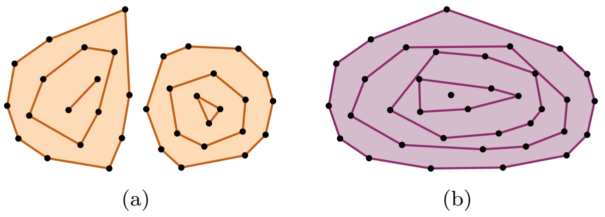

Figure 1: (a) Two disjoint onions. (b) Their union.

We apply this algorithm to obtain an efficient solution to the onion preprocessing problem mentioned in the introduction. Given n pairwise disjoint unit disks that model an imprecise point set, we build a data structure of size $O(n)$ such that the onion decomposition of an instance can be retrieved in $O(n \log k)$ time, where k is the number of layers in the resulting onion. The expected preprocessing time is $O(n \log n)$.

We also show that without paramerising by k, it is not possible to speed up the computation of the onion decomposition: in the worst case, any algorithm can be forced to take $\Omega(n \log n)$ time on some instances.

2 Preliminaries and Definitions

Let P be a set of n points in $\mathbb{R}^2$. The *onion decomposition*, or *onion*, of P, is the sequence 🧅(P) of pairwise disjoint convex polygons with vertices from P, constructed recursively as follows: if $P \neq \emptyset$, we set 🧅$(P) := \{\mathrm{ch}(P)\} \cup$ 🧅$(P \setminus \mathrm{ch}(P))$, where $\mathrm{ch}(P)$ is the convex hull of P; if $P = \emptyset$, then 🧅$(P) := \emptyset$ [4]. An element of 🧅(P) is called a *layer* of P.

Let $\mathcal{D}$ be a set of disjoint unit disks in $\mathbb{R}^2$. We say a point set P is a *sample* from $\mathcal{D}$ if every disk in $\mathcal{D}$ contains exactly one point from P. We write $\log$ for the logarithm with base 2.

3 The Algorithm

In the following sections, we will describe the individual pieces required for our result.

3.1 Unions of Onions

Suppose we have two point sets P and Q, together with their onions. We show how to find 🧅$(P \cup Q)$

[*]Universiteit Utrecht, `m.loffler@uu.nl`. Funded by Netherlands Organisation for Scientific Research (NWO) grant 639.021.123.

[†]Freie Universität Berlin, `mulzer@inf.fu-berlin.de`. Supported in part by DFG project MU/3501/1.

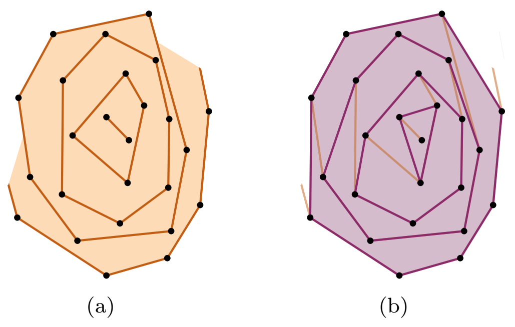

Figure 2: (a) A half-eaten onion; (b) the restored onion.

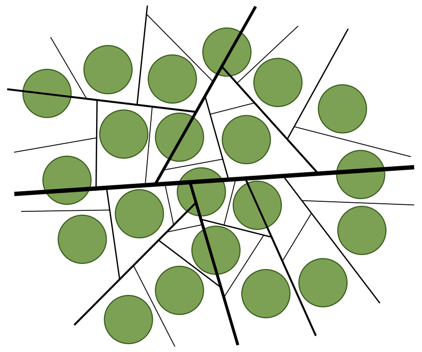

Figure 3: A space decomposition tree for 21 unit disks.

quickly, given that 🧅(P) and 🧅(Q) are disjoint. Deleting points can only decrease the number of layers, so:

Observation 1 *Let $P, Q \subseteq \mathbb{R}^2$. Then* 🧅$(P)$ *and* 🧅(Q) *cannot have more layers than* 🧅$(P \cup Q)$. $\square$

The following lemma constitutes the main ingredient of our onion-union algorithm. By a *convex chain*, we mean any connected subset of a convex closed curve.

Lemma 1 *Let A and B be two non-crossing convex chains. We can find $\mathrm{ch}(A \cup B)$ in $O(\log n)$ time.*

Proof. Since A and B do not cross, the pieces of A and B that appear on $\mathrm{ch}(A \cup B)$ are both connected: otherwise, $\mathrm{ch}(A \cup B)$ would contain four points belonging to A, B, A, and B, in that order. However, the points on A must be connected inside $\mathrm{ch}(A \cup B)$; as do the points on B. Thus, the chains A and B cross, which is impossible. Since A and B are convex chains, we can compute $\mathrm{ch}(A), \mathrm{ch}(B)$ in $O(\log n)$ time. Furthermore, since A and B are disjoint, we can also, in $O(\log n)$ time, make sure that $\mathrm{ch}(A) \cap \mathrm{ch}(B) = \emptyset$, by removing parts from A or B, if necessary. Now we can find the bitangents of $\mathrm{ch}(A)$ and $\mathrm{ch}(B)$ in logarithmic time [10]. $\square$

Lemma 2 *Suppose* 🧅(P) *has k layers. Let A be the outer layer of* 🧅(P), *and p, q be two vertices of A. Let A_1 be the points on A between p and q, going counterclockwise. We can find* 🧅$(P \setminus A_1)$ *in $O(k \log n)$ time.*

Proof. The points p and q partition A into two pieces, A_1 and A_2. Let B be the second layer of 🧅(P). The outer layer of 🧅$(P \setminus A_1)$ is the convex hull of $P \setminus A_1$, i.e., the convex hull of A_2 and B. By Lemma 1, we can find it in $O(\log n)$ time. Let $p', q' \in P$ be the points on B where the outer layer of 🧅$(P \setminus A_1)$ connects. We remove the part between p' and $q'$from B, and use recursion to compute the remaining layers of 🧅$(P \setminus A_1)$ in $O((k-1)\log n)$ time; see Figure 2. $\square$

We conclude with the main theorem of this section:

Theorem 3 *Let P and Q be two planar point sets of total size n. Suppose that* 🧅(P) *and* 🧅(Q) *are disjoint. We can find the onion* 🧅$(P \cup Q)$ *in $O(k^2 \log n)$ time, where k is the resulting number of layers.*

Proof. By Observation 1, 🧅(P) and 🧅(Q) each have at most k layers. We use Lemma 1 to find $\mathrm{ch}(P \cup Q)$ in $O(\log n)$ time. By Lemma 2, the remainders of 🧅(P) and 🧅(Q) can be restored to proper onions in $O(k \log n)$ time. The result follows by induction. $\square$

3.2 Space Decomposition Trees

We now describe how to preprocess the disks in $\mathcal{D}$ for fast divide-and-conquer. A *space decomposition tree* T is a rooted binary tree where each node v is associated with a planar region R_v. The root corresponds to all of $\mathbb{R}^2$; for each leaf v of T, the region R_v intersects at most one disk in $\mathcal{D}$. Furthermore, each inner node v in T is associated with a directed line ℓ_v, so that if u is the left child and w the right child of v, then $R_u := R_v \cap \ell_v^+$ and $R_w := R_v \cap \ell_v^-$. Here, ℓ_v^+ is the halfplane to the left of ℓ_v and ℓ_v^- the halfplane to the right of ℓ_v.

We would like to construct a space decomposition tree for $\mathcal{D}$ whose height is as low as possible. For this, we use the following lemma, which is a constructive version of a result by Alon *et al.* [1, Theorem 1.2].

Lemma 4 *There exists a constant $c \geq 0$, so that for any set $\mathcal{D}$ of m congruent nonoverlapping disks in the plane, there is a line ℓ with at least $m/2 - c\sqrt{m \log m}$ disks completely to each side of it. We can find ℓ in $O(m)$ expected time.*

Proof. Our proof closely follows Alon *et al.* [1, Section 2]. Set $r := \lfloor \sqrt{m / \log m} \rfloor$. Pick a random integer z between 1 and $r/2$. Find a line ℓ whose angle with the x-axis is $(z/r)\pi$ and that has $\lfloor m/2 \rfloor$ centers of disks in $\mathcal{D}$ on each side. Given z, we can find ℓ in $O(m)$ time by a median computation. The proof by Alon *et al.* shows that with probability at least $1/2$ over the choice of z, the line ℓ intersects at most

$c\sqrt{m \log m}$ disks in $\mathcal{D}$, for some constant $c \geq 0$. Thus, we need two tries in expectation to find a good line ℓ. The expected total running time is $O(m)$. □

To obtain our space decomposition tree T, we apply Lemma 4 recursively until the region for each node intersects at most one disk in $\mathcal{D}$. The next lemma helps us analyze the complexity of T.

Lemma 5 *Let T be a space decomposition tree obtained by recursive application of Lemma 4. For $k = 0, \dots, \log n$, let d_k be the maximum number of disks that intersect the region for a node in T of depth k. Then $\log d_k \leq \log n - k + O(1)$. Furthermore, the tree T has $O(n)$ nodes and height $\log n + O(1)$.*

Proof. We have $d_0 = n$. Furthermore, by Lemma 4,

$$d_k/2 \leq d_{k+1} \leq d_k/2 + c\sqrt{d_k \log d_k}.$$

Define

$$\beta_k := \sum_{i=0}^{k-1} \left(\frac{2^i}{n}\right)^{1/3}.$$

Note that $\beta_k \leq 4$, for $k = 0, \dots, \log n$. We assume that $n \geq 2^{12}(3c+2)^9$, and we prove by induction that $\log d_k \leq \log n - k + \beta_k$, for $k = 0, \dots, \log n - 12 - 9\log(3c+2)$. For $k = 0$, this is clear, since $\beta_0 = 0$. Now note that for every $k \geq 0$, we have $d_k \leq n$ and $d_k \geq n/2^k$. Thus,

$$\begin{aligned}
&\log d_{k+1} \leq \log(d_k/2 + c\sqrt{d_k \log d_k}) \\
&= \log(d_k/2) + \log\left(1 + 2c\sqrt{\frac{\log d_k}{d_k}}\right) \\
&\leq \log d_k - 1 + \log\left(1 + 2c\sqrt{\frac{2^k \log d_k}{n}}\right) \\
&\leq \log n - (k+1) + \beta_k + \frac{2c}{\ln 2}\sqrt{\frac{2^k(\log n - k + 4)}{n}} \\
&= \log n - (k+1) + \beta_{k+1},
\end{aligned}$$

since for $k \leq \log n - 12 - 9\log(3c+2)$, we have

$$\frac{2c}{\ln 2}\sqrt{\frac{2^k(\log n - k + 4)}{n}} \leq \left(\frac{2^k}{n}\right)^{1/3}.$$

It follows that after $k^* := \log n - 12 - 9\log(3c+2)$ iterations, there are at most $2^{12+9\log(3c+2)+\beta_{k^*}} = O(1)$ disks per region left. Hence, T has height $\log n + O(1)$ and $O(n)$ nodes. □

By Lemma 5, T induces a planar subdivision G_T with $O(n)$ faces. We add a large enough bounding box to G_T and triangulate the resulting graph. Since G_T is planar, the triangulation has complexity $O(n)$ and can be computed in the same time (no need for heavy machinery—all faces of G_T are convex). With each disk in $\mathcal{D}$, we store the list of triangles that intersect it (recall that each triangle intersects at most one disk). This again takes $O(n)$ time and space. We conclude with the main theorem of this section:

Theorem 6 *Let $\mathcal{D}$ be a set of n disjoint unit disks in $\mathbb{R}^2$. In $O(n \log n)$ expected time, we can construct a space partition tree T for $\mathcal{D}$ with $O(n)$ nodes and height $\log n + O(1)$. Furthermore, for each disk $D \in \mathcal{D}$, we have a list of triangles T_D that cover the leaf regions of T that intersect D.* □

3.3 Processing a Precise Input

Let P be a sample from $\mathcal{D}$. Suppose first that we know k, the number of layers in ◎(P). For each input point p_i, let $D_i \in \mathcal{D}$ be the corresponding disk. We check all triangles in T_{D_i}, until we find the one that contains p_i. Since there are $O(n)$ triangles, this takes $O(n)$ time. Afterwards, we know for each point in P the leaf of T that contains it.

The *upper tree* T_u of T consists of all nodes with depth at most $\log n - 2\log k$. Each leaf of T_u is the root of a subtree of T of height at most $2\log k + O(1)$. Hence it corresponds to a subset of P with $O(k^2)$ points. For each such subset, we use Chazelle's algorithm [4] to find its onion decomposition in $O(k^2 \log k)$ time. This takes $O(n \log k)$ total time. Now, in order to obtain ◎(P), we perform a postorder traversal of T_u, using Theorem 3 in each node to unite the onions of its children.

For a node of depth i, this takes time $O(k^2 \log d_i) = O(k^2(\log n - i + 1))$, by Lemma 5. Thus, the total work is proportional to

$$\begin{aligned}
&\sum_{i=0}^{\log n - 2\log k} 2^i k^2 (\log n - i + 1) \\
&= k^2 \frac{n}{k^2} \sum_{i=0}^{\log \frac{n}{k^2}} \frac{2\log k + i + 1}{2^i} \\
&= O(n \log k).
\end{aligned}$$

So far, we have assumed that k is given. Using standard exponential search, this requirement can be removed. More precisely, let $k_i = 2^{2^i}$, for $i = 1, \dots, \log\log n$. Run the algorithm for $k_0, k_1, \dots$. If the algorithm succeeds, report the result. If not, abort as soon as it turns out that an intermediate onion has more than k_i layers and try k_{i+1}. The total time is

$$\sum_{i=0}^{\log\log k} O(n2^i) = O(n \log k),$$

as desired. This finally proves our main result.

Theorem 7 *Let $\mathcal{D}$ be a set of n disjoint unit disks in $\mathbb{R}^2$. We can build a data structure that stores $\mathcal{D}$, of size $O(n)$, in $O(n \log n)$ expected time, such that given a sample P of $\mathcal{D}$, we can compute ◎(P) in $O(n \log k)$ time, where k is the number of layers in ◎(P).* □

Remark. Using the same approach, without the exponential search, we can also compute the outermost

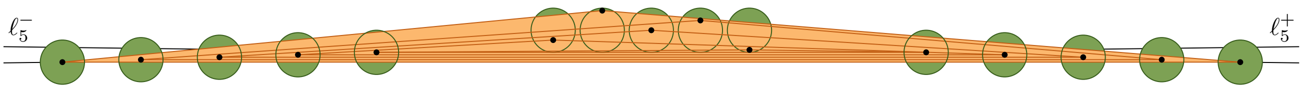

Figure 4: The lower bound construction consists of $n/3$ unit disks centered on a horizontal line (5 in the figure), and two groups of $n/3$ points sufficiently far to the left and to the right of the disks. Distances not to scale.

k layers of an onion with arbitrarily many layers in $O(n \log k)$ time, for any k. In order to achieve this, we simply abort the union algorithm whenever k layers have been found, and note that the points in P not on the outermost k layers of $◎(P)$ will never be part of the outermost k layers of $◎(Q)$ for any $Q \supset P$.

4 Lower Bounds

We now show that the query time must depend on k (i.e. for onions with many layers, we cannot hope to speed up the computation).

Let n be a multiple of 3, and consider the lines

$$\ell_n^- : y = -1/2 - 6/n - x/n^2;$$
$$\ell_n^+ : y = -1/2 - 6/n + x/n^2.$$

Let $\mathcal{D}_n$ consist of $n/3$ disks centered on the x-axis at x-coordinates between $-n/6$ and $n/6$; a group of $n/3$ disks centered on ℓ_n^- at x-coordinates between n^2 and $n^2+n/3$; and a symmetric group of $n/3$ disks centered on ℓ_n^+ at x-coordinates between $-n^2 - n/3$ and $-n^2$. Figure 4 shows $\mathcal{D}_{15}$.

Lemma 8 *Let π be a permutation on $n/3$ elements. There is a sample P of $\mathcal{D}_n$ such that p_i (the point for the ith disk from the left in the main group) lies on layer $\pi(i)$ of $◎(P)$.*

Proof. Take P as the $n/3$ centers of the disks in $\mathcal{D}$ on ℓ_n^-, the $n/3$ centers of the disks in $\mathcal{D}$ on ℓ_n^+, and for each disk $D_i \in \mathcal{D}$ on the x-axis the point $p_i = (i - n/6, \pi(i) \cdot 3/n - 1/2)$. By construction, the outermost layer of $◎(P)$ contains at least the leftmost point on ℓ_n^+, the rightmost point on ℓ_n^-, and the highest point (with y-coordinate $1/2$). However, it does not contain any more points: the line segments connecting these three points have slope at most $2/n^2$. The second highest point lies $3/n$ lower, and at most $n/3$ further to the left or the right. The lemma follows by induction. $\square$

There are $(n/3)! = 2^{\Theta(n \log n)}$ permutations π; so any corresponding decision tree has height $\Omega(n \log n)$.

Theorem 9 *For any n, there is a set $\mathcal{D}$ of n disjoint unit disks in $\mathbb{R}^2$, such that any decision-based algorithm to compute $◎(P)$ for P a sample of $\mathcal{D}$, based only on prior knowledge of $\mathcal{D}$, takes $\Omega(n \log n)$ time in the worst case.*

We expect that our lower bound can be strengthened to $\Omega(n \log k)$ and that it also applies to randomized algorithms. Details will follow in the full version.

5 Conclusion and Further Work

It would be interesting how much the parameter k can vary for a set of imprecise bounds and how to estimate k efficiently. Further work includes considering more general regions, such as overlapping disks, disks of different sizes, or fat regions. Furthermore, three-dimensional onions are not well understood. The best general algorithm needs $O(n \log^6 n)$ expected time [3], giving more room for improvement in our setting.

References

[1] N. Alon, M. Katchalski, and W. Pulleyblank. Cutting disjoint disks by straight lines. *DCG*, 4:239–243, 1989.

[2] K. Buchin, M. Löffler, P. Morin, and W. Mulzer. Preprocessing imprecise points for Delaunay triangulation: simplified and extended. *Algorithmica*, 61(3):675–693, 2011.

[3] T. M. Chan. A dynamic data structure for 3-D convex hulls and 2-D nearest neighbor queries. *J. ACM*, 57(3):Art. 16, 15 pp., 2010.

[4] B. Chazelle. On the convex layers of a planar set. *IEEE Trans. Inform. Theory*, 31(4):509–517, 1985.

[5] B. Chazelle, L. J. Guibas, and D. T. Lee. The power of geometric duality. *BIT*, 25(1):76–90, 1985.

[6] O. Devillers. Delaunay triangulation of imprecise points: preprocess and actually get a fast query time. *J. Comput. Geom.*, 2(1):30–45, 2011.

[7] E. Ezra and W. Mulzer. Convex hull of points lying on lines in $o(n \log n)$ time after preprocessing. *Comput. Geom.*, 46(4):417–434, 2013.

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

[9] P. J. Huber. Robust statistics: A review. *Ann. Math. Statist.*, 43:1041–1067, 1972.

[10] D. Kirkpatrick and J. Snoeyink. Computing common tangents without a separating line. In *Proc. 4th WADS*, pages 183–193, 1995.

[11] M. van Kreveld, M. Löffler, and J. S. B. Mitchell. Preprocessing imprecise points and splitting triangulations. *SIAM J. Comput.*, 39(7):2990–3000, 2010.

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