Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

QuickhullDisk: A faster convex hull algorithm for disks

Nguyen Kieu Linh[a,1], Chanyoung Song[b,1], Joonghyun Ryu[c,d], Phan Thanh An[e,f], Nam-Dũng Hoang[g,*], Deok-Soo Kim[b,c,d,**]

[a] *Posts and Telecommunications Institute of Technology, Km 10, Nguyen Trai, Ha Dong District, Hanoi, Vietnam*
[b] *School of Mechanical Engineering, Hanyang University, 222 Wangsimni-ro, Seongdong-gu, Seoul, South Korea*
[c] *Voronoi Diagram Research Center, Hanyang University, 222 Wangsimni-ro, Seongdong-gu, Seoul, South Korea*
[d] *HYU-HPSTAR-CIS Global High Pressure Research Center, Hanyang University, 222 Wangsimni-ro, Seongdong-gu, Seoul, South Korea*
[e] *Institute of Mathematics, Vietnam Academy of Science and Technology, 18 Hoang Quoc Viet Road, Hanoi 10307, Vietnam*
[f] *Institute of Mathematical and Computer Sciences, University of São Paulo, Av. Trabalhador são-carlense, 400, São Carlos, SP, Brazil*
[g] *Faculty of Mathematics, Mechanics and Informatics, Vietnam National University, 334 Nguyen Trai, Thanh Xuan, Hanoi, Vietnam*

ARTICLE INFO

Article history:
Available online 8 August 2019

Keywords:
Weighted points
Incremental algorithm
Quicksort
Quickhull
Voronoi diagram
Divide and conquer

ABSTRACT

Convex hull is one of the most fundamental constructs in geometry and its construction has been extensively studied. There are many prior works on the convex hull of points. However, its counterpart for weighted points has not been sufficiently addressed despite important applications. Here, we present a simple and fast algorithm, `QuickhullDisk`, for the convex hull of a set of disks in $\mathbb{R}^2$ by generalizing the quickhull algorithm for points. `QuickhullDisk` takes $O(n\log n)$ time on average and $O(mn)$ time in the worst case where m represents the number of extreme disks which contribute to the boundary of the convex hull of n disks. These time complexities are identical to those of the quickhull algorithm for points in $\mathbb{R}^2$. Experimental result shows that the proposed `QuickhullDisk` algorithm runs significantly faster than the $O(n\log n)$ time incremental algorithm, proposed by Devillers and Golin in 1995, particularly for big data. `QuickhullDisk` is approximately 2.6 times faster than the incremental algorithm for random disks and is 1.2 times faster even for the disk sets where all disks are extreme. This speed-up is because the basic geometric operation of the `QuickhullDisk` algorithm is a predicate for the location of a point w.r.t. a line and is much faster than that of the incremental algorithm. The source code of `QuickhullDisk` is freely available from Mendeley Data and a GUI-version from Voronoi Diagram Research Center, Hanyang University (http://voronoi.hanyang.ac.kr/).

1. Introduction

Convex hull is one of the most fundamental geometric objects and its importance has long been well-addressed [1–3]. The convex hull of geometric objects is the smallest convex set containing the objects. Its properties and construction

* Corresponding author at: Faculty of Mathematics, Mechanics and Informatics, Vietnam National University, 334 Nguyen Trai, Thanh Xuan, Hanoi, Vietnam.
** Corresponding author at: School of Mechanical Engineering, Hanyang University, 222 Wangsimni-ro, Seongdong-gu, Seoul, South Korea.
E-mail addresses: linhnk@ptit.edu.vn (N.K. Linh), cysong.vdrc@gmail.com (C. Song), jhryu.vdrc@gmail.com (J. Ryu), thanhan@math.ist.utl.pt (P.T. An), hoangnamdung@hus.edu.vn (N.-D. Hoang), dskim@hanyang.ac.kr (D.-S. Kim).
[1] Both authors equally contributed to this work.

https://doi.org/10.1016/j.amc.2019.124626

algorithms have been extensively studied because of its theoretical and practical importance. Convex hull problems arise as one of the most important subproblems in many problems of computational geometry, optimization, etc. Algorithms have been reported for points in two-, three-, and even higher-dimensional Euclidean space.

History of convex hull algorithms. In 1972, the $O(n\log n)$ Graham's scan algorithm was introduced for constructing the convex hull of n points in the plane [4] and replaced the popular $O(n^3)$ algorithm which performed $O(n^2)$ half-plane containment tests for $n-2$ points. In 1973, the $O(mn)$ Jarvis march algorithm, a two-dimensional version of the output-sensitive Chand–Kapur gift-wrapping algorithm [5], was reported where m represents the number of vertices on the hull boundary [6]. In 1975, Preparata and Hong presented an $O(n\log n)$ divide-and-conquer algorithms for both 2D and 3D points [7,8]. In 1977 and 1978, Eddy [9] and Bykat [10] independently reported the quickhull algorithm for 2D points which were based on the idea of the well-known quicksort algorithm, respectively. The efficiency of the quickhull algorithm is $O(n\log n)$ time on average and $O(mn)$ in the worst case for m vertices of the convex hull of n 2D points [11–13]. In 1981, Seidel presented an incremental algorithm taking $O(n\log n + n^{\lfloor d+1\rfloor/2})$ time for the convex hull of d-dimensional points where d is an even number [14,15]. The convex hull construction problem has remained an attractive research problem to develop other algorithms such as the marriage-before-conquest algorithm by Kirkpatrick and Seidel in 1986 [16], Chan's algorithm in 1996 [17], a fast approximation algorithm for multidimensional points by Xu et al in 1998 [18], a new divide-and-conquer algorithm by Zhang et al. in 2010 [19] and An's recent improvements of the Graham's algorithm in 2010 [20] and the gift-wrapping algorithm in 2013 [21]. It is interesting and important to observe that the first theoretically solid algorithm for the convex hull construction was discussed in a general dimensional space by Chand and Kapur in 1970 which has been since then called the gift-wrapping algorithm [5]. They implemented the algorithm using FORTRAN to construct the convex hull of 1,000 six-dimensional points. Parallel algorithms were also reported for both 2D and 3D points [22]. In 1985, Atallah made a theoretical observation about the convex hull of moving points [23]. In 2012, Aurenhammer and Jüttler reported an $O(m\log n)$ algorithm for n circular arcs where m represents the size of the convex hull [24]. Recent efforts include the convex hull of probabilistic points whose locations are determined by a normal distribution [25,26]. It is interesting to observe Kalantari's recent work of Triangle Algorithm for convex hull membership problem for finite points which connects the convex hull problem to linear program [27,28] and to separating hyperplane theorem [29].

If points are generalized to disks or spheres with non-zero radii, the convex hull construction becomes more difficult and complicated. The radii of disks correspond to the weights associated with the center points of the disks. Let $Conv(D)$ be the convex hull of a set D of n disks in the plane which is the smallest convex region containing all of the disks. Its boundary $\partial Conv(D)$ is an ordered curve segments which consists of alternating sequence of arcs and tangent lines connecting consecutive arcs. We call each curve segment on $\partial Conv(D)$ a *hull edge* and, in particular, the line segment a *linear hull edge*. In 1992, Rappaport proposed an $\mathrm{O}(n\log n)$ divide-and-conquer algorithm for constructing the convex hull of disks and discussed its applications [30]. In 1995, Devillers and Golin [31] reported the $\mathrm{O}(n\log n)$ monotone chain algorithm which was a modification of the incremental algorithm. The algorithm used an AVL-tree to store disks touching the convex hull boundary in the leaf nodes and their clusters in internal nodes so that an incrementing disk could be efficiently determined if it was contained within the current convex hull or not. In 1996, Boissonnat et al. reported an algorithm taking $O(n\log n + n^{\lfloor d/2\rfloor})$ for d-dimensional spheres which was optimal in 3- and even dimensional space [32]. In 1998, Chen et al. introduced a parallel method using a divide-and-conquer approach and showed that the convex hull of n disks in the plane could be constructed in $O(\log^{1+\epsilon} n)$ time using $O(n/log^{\epsilon} n)$ processors or in $O(\log n\log\log n)$ time using $O(n\log^{1+\epsilon} n)$ processors for any positive constant ϵ [33]. In 1999, Yue et al. applied the idea of the Jarvis march to a set of straight and circular line segments [34]. In 2004, Habert and Pocchiola showed that the convex hull of a collection of n pairwise disjoint disks in the plane is computable in $O(n\log n)$ time using the chirotope of the collection of disks [35].

Contribution of this study. However, interestingly enough, we were not able to find any prior work on the quickhull-like algorithm for constructing the convex hull of 2D circular disks or 3D spherical balls. Considering the simplicity and efficiency of both the quicksort algorithm for numbers and the quickhull algorithm for points, we are certain that such a quicksort-like algorithm for disks and spheres are important for both theoretical and practical points of views. In this paper, we report an algorithm, `QuickhullDisk`, which constructs the convex hull of disks in the plane by generalizing the "more than 40-years-old" yet powerful quickhull algorithm for points in the plane. The *contributions of this paper* are as follows:

- A simple and fast `QuickhullDisk` algorithm for constructing the convex hull of disks in the plane.
- A robust implementation of the `QuickhullDisk` algorithm (Programs are freely available).

The well-known quickhull algorithm for points is a divide-and-conquer algorithm (For the correctness and efficiency, see [36]): Divide the input point set into subsets and merge the solutions of the subsets to get the solution of the entire set as quickly as possible; For each subset, apply the same rule recursively until a simple rule can immediately return the result. There are two critical observations from efficiency point of view: (i) Two subsets in each recursion are likely to be of similar sizes, and (ii) the merge takes time linear to the total number of elements in the two subsets in the worst case. Its idea is based on the quicksort algorithm and as the quicksort is frequently the fastest among sorting algorithms, the quickhull algorithm tends to be the fastest among the convex hull algorithms for points. The idea of the quickhull algorithm follows that of the quicksort algorithm reported by Hoare in 1961 [37–39] to improve the prior merge sort algorithm [40] (The initial idea of the quicksort algorithm was proposed by Shell in 1959 [41]).

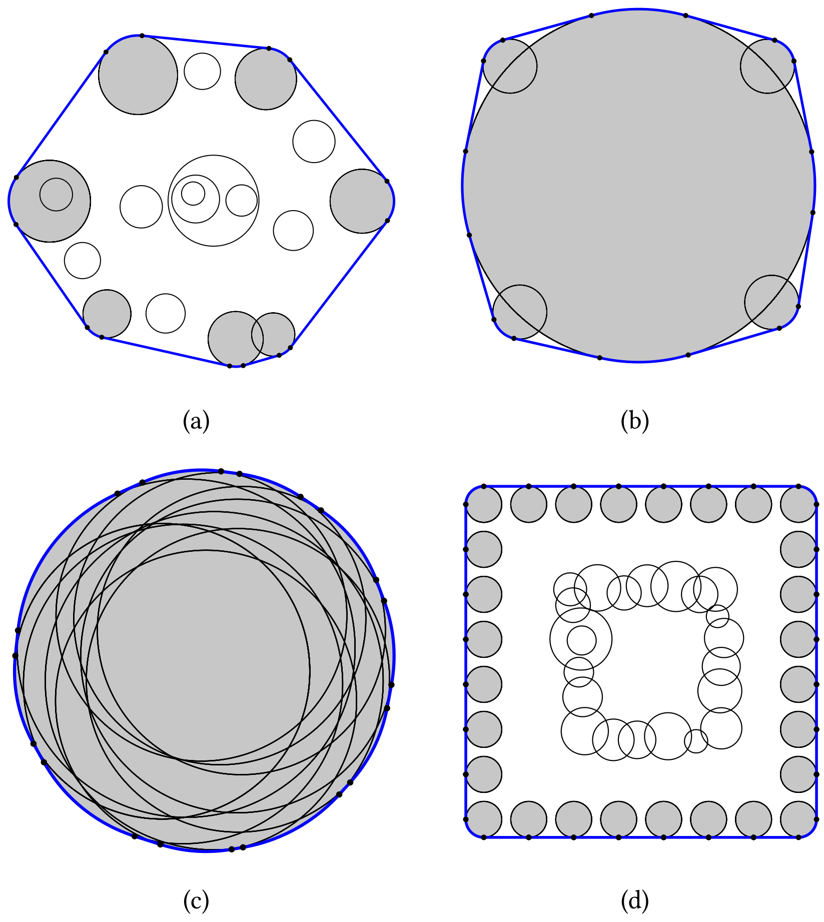


Fig. 1. Convex hulls of disks in $\mathbb{R}^2$ constructed by the `QuickhullDisk` algorithm. We allow intersection and containment among disks. Blue line segments and arcs are hull edges, i.e., the edges on the boundary of convex hull. (a) Random disks. (b) Each of the four small disks defines two linear hull edges to the big one in the middle independent of the other small ones: There are total 8 $(= (5-1)*2)$ linear hull edges on the convex hull boundary. (c) Ten disks with a common intersection: Every disk is extreme and each disk defines a linear hull edge with its counterclockwise neighbor disk. (d) Disks have common tangent lines. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

In this paper, we exploit the virtue of both the quicksort and quickhull algorithms for the construction of the convex hull of a finite set of disks in the plane, thus named the `QuickhullDisk` algorithm. `QuickhullDisk` takes an $O(n\log n)$ time on average and $O(mn)$ time in the worst case where m represents the number of extreme disks which contribute to the boundary of the convex hull of n input disks. These time complexities are identical to those of the quickhull algorithm for points in $\mathbb{R}^2$. Experimental result shows that the proposed `QuickhullDisk` algorithm runs significantly faster than the $O(n\log n)$ time incremental algorithm, proposed by Devillers and Golin in 1995, for most data sets we tested, particularly for big data. `QuickhullDisk` is approximately 2.6 times faster than the incremental algorithm for all types of random disks. `QuickhullDisk` is 1.2 times faster even for the disk sets where all disks are extreme, in which case the incremental algorithm is best-suited from efficiency point of view. Recall that the quicksort algorithm is much slower than other sorting algorithms when an input data set of numbers is sorted or near sorted. In this sense, this result is somewhat striking. This speed-up is because the basic geometric operation of the `QuickhullDisk` algorithm is a predicate for the location of a point w.r.t. a line and is much faster than that of the incremental algorithm. In addition, `QuickhullDisk` is easier than the incremental algorithm to handle degenerate cases: E.g. if an incrementing disk simultaneously touches two edges on a convex hull boundary, the incremental algorithm requires a special treatise whereas it is an ordinary case for `QuickhullDisk`.

Fig. 1 shows examples of the convex hulls of 2D disks constructed by the proposed `QuickhullDisk` algorithm: (a) random disks; (b) each of the four small disks defines two linear hull edges to the big one in the middle: the convex hull boundary contribution of the small disks are independent of each other and thus there are 8 $(= (5-1)*2)$ linear hull edges in total on the convex hull boundary; (c) ten disks with a common intersection and every disk is extreme: each disk defines a linear hull edge with its counterclockwise neighbor disk; (d) disks have common tangent lines. Note that disks may intersect and may include other disks.

Application of convex hull algorithms. In addition to theoretical significance, the convex hull of a set D of disks is useful as a computational building block for solving other important geometric problems as well summarized in [30]. The diameter of a set of points is the greatest distance between points and that of a convex set is the greatest distance between parallel support lines [42]. Thus, the diameter of D can be found by rotating a pair of parallel support lines about $Conv(D)$ to find the antipodal pair of points on $\partial Conv(D)$ which yields the maximum distance [30]. The intersection of n disks can be found using the convex hull algorithm [30]. The convex hull of disks can be used to construct the furthest Voronoi diagram for the disks in $O(n\log n)$ time [30]. The minimum spanning circle of a set is a smallest circle that intersects the set [1,43]. The minimum spanning circle of a set of disks can be computed using the furthest Voronoi diagram of the disks which again uses the convex hull of the disks [30]. A stabbing line (or also called a common transversal line) of a set of objects is a straight line that intersects all members of the set [44,45] and each edge on $\partial Conv(D)$ is a partial support line of D and immediately produces a stabbing region [30]. The convex hull of disks can be used as a filter to reduce solution space for finding the shortest paths avoiding circular obstacles [46]. In the study of stochastic convex hull, the convex hull of disks is becoming more useful, in connection with kinetic data structure [47], as the location uncertainties due to measurements are reflected in computational model [25,26,48]. Kallrath and Frey recently introduced a new computational geometry optimization problem to arrange a set of circles such that the length of the boundary of the convex hull of the non-overlapping circles is minimized [49]. They solved the problem as a mixed-integer nonlinear programming problem. For solving this type of optimization problem for larger number of circles, an efficient convex hull algorithm is necessary.

The well-known connection between the convex hull and the Voronoi diagram is of interest: connecting the generators of adjacent unbounded Voronoi cells in the Voronoi diagram produces the convex hull of the generators [1,2,50]. Hence, if the Voronoi diagram is available by a preprocessing (either by the optimal $O(n\log n)$ divide-and-conquer algorithm, by the robust $O(n^2)$ topology-oriented incremental algorithm [51], or by the edge-flipping algorithm [52,53]), the convex hull can be constructed in $O(n)$ time. However, this approach is computationally more expensive than directly constructing convex hulls using the above-mentioned algorithms [24]. This is because convex hull construction is only one of many applications of Voronoi diagram which is a neutral data structure.

Let us see the opposite side. We view an important function of Voronoi diagram materialized by the convex hull algorithm. Voronoi diagrams require to represent infinity, using a rectangular or circular container containing all input generators [54,55], or using a few fictitious generators representing infinity [50,56,57]. Given a Voronoi diagram with an appropriately represented infinity, we may need to evaluate the Voronoi diagram structure to obtain the convex hull boundary when it is requested. We note that the decisions in this procedure are based on numerical computation, i.e., floating-point arithmetic, and thus may cause inconsistency with the symbolic information stored in the topology of the Voronoi diagram. This is particularly critical for disk generators in that their Voronoi edges are non-linear. Hence, we instead use `QuickhullDisk` independently to help to construct the convex hull boundary only when an application needs.

Notes. In this paper, all discussions are in $\mathbb{R}^2$ and all time complexities are for the worst cases unless otherwise stated. If more than two disks are tangent to a common line, we consider all such disks contribute to the convex hull boundary. A point in the plane is represented as $a = a(a_x, a_y)$. The source code of `QuickhullDisk` is freely available from Mendeley Data [58] and a GUI-version from Voronoi Diagram Research Center, Hanyang University (http://voronoi.hanyang.ac.kr/).

Section 2 defines terminologies necessary for describing the theory and algorithm. Section 3 describes the main part of the `QuickhullDisk` algorithm for determining the convex hull of a finite set of disks. Section 4 describes the return condition of recursions in `QuickhullDisk`. Section 5 describes the strategy to handle degenerate or near degenerate cases. Section 6 summarizes the algorithm and Section 7 presents the proofs of important properties of `QuickhullDisk` including its termination and computational complexity. Section 8 presents experimental results using diverse data sets.

2. Preliminaries

Let $D = \{d_1, d_2, \ldots, d_n\}$, $n \geq 2$, be a set of disks $d_i(c_i, r_i)$ with center $c_i(x_i, y_i) \in \mathbb{R}^2$ and the radius $r_i \geq 0$. Each disk is unique and is represented by three float point numbers. Two disks may intersect each other and one may contain another. The convex hull of X, denoted by $Conv(X)$, is the smallest convex set in $\mathbb{R}^2$ containing all elements of a set X. We are interested in the construction of $Conv(D)$ when the disks are polysized: i.e., the disk radii are arbitrary. If D contains only congruent disks, the construction of the convex hull of the disk centers is sufficient. This is because the disks whose centers are the extreme points in the convex hull of the center points are also the extreme disks in the convex hull of the disks.

Let $\Delta(a, b)$ be an oriented line passing through two points a and b with the orientation from a to b in $\mathbb{R}^2$. Consider three points in the plane: $a = (a_x, a_y)$, $b = (b_x, b_y)$, and $c = (c_x, c_y)$. Let $\hat{a} = [1, a_x, a_y]$, $\hat{b} = [1, b_x, b_y]$, and $\hat{c} = [1, c_x, c_y]$. Then, Orient(a, b, c) $= det([\hat{a}, \hat{b}, \hat{c}]^T)$ is a determinant that gives twice the signed area of the triangle formed by a, b and c.

It is well-known that an ordered triplet of points (a, b, c) are a counterclockwise turn if Orient$(a, b, c) > 0$, a clockwise turn if Orient$(a, b, c) < 0$, and collinear if Orient$(a, b, c) = 0$.

It immediately follows that if Orient$(a, b, c) > 0$, c is placed in the left half-plane divided by $\Delta(a, b)$. If Orient$(a, b, c) < 0$, c is placed in the right half-plane divided by $\Delta(a, b)$. If Orient$(a, b, c) = 0$, c is on $\Delta(a, b)$.

Definition 1 see [12]**.** Let (a, b, c) be an ordered triple of points in $\mathbb{R}^2$.

(i) (a, b, c) is said to have a positive orientation if Orient$(a, b, c) > 0$, a negative orientation if Orient$(a, b, c) < 0$, and a zero orientation if Orient$(a, b, c) = 0$.

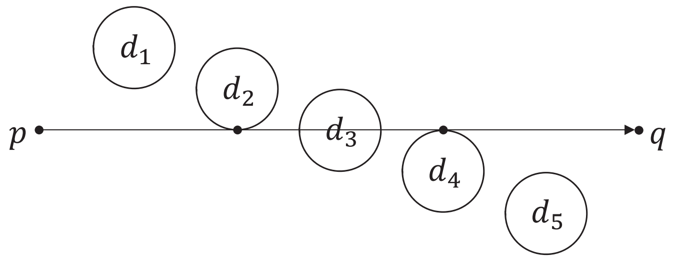


Fig. 2. Disk positions w.r.t. an oriented line $\Delta(p, q)$: d_1: positive; d_2: on-positive; d_3: crossing; d_4: on-negative; d_5: negative.

(ii) Given two half-planes divided by $\Delta = \Delta(a, b)$, the left half-plane is said positive and the right half-plane is said negative.

(iii) If Orient(a, b, c) > 0, c is said positive w.r.t. Δ and if Orient(a, b, c) < 0, c is said negative w.r.t. Δ.

(iv) If Orient(a, b, c) ≥ 0, c is said non-negative w.r.t. Δ and if Orient(a, b, c) ≤ 0, c is said non-positive w.r.t. Δ.

Definition 2. A disk d is said positive (or negative) w.r.t. an oriented line Δ if all points of d are positive (or negative) w.r.t. Δ. A disk d is said *on-positive* (or on-negative) w.r.t. Δ if d is tangent to Δ from the positive (or negative) half-plane. A disk d is said *crossing* if d has both positive and negative points (i.e., d intersects Δ).

In Fig. 2, d_1, d_2, d_3, d_4, and d_5 are positive, on-positive, crossing, on-negative, and negative, respectively. If a disk d is positive (or negative) w.r.t. Δ, d is placed in the left (or right) half-plane of Δ and $d \cap \Delta = \emptyset$. In the presentation of `QuickhullDisk` algorithm, negative, on-negative, or crossing disks are of primary interest.

Definition 3. A set D of disks is said positive, on-positive, crossing, on-negative, or negative w.r.t. an oriented line Δ if every disk $d \in D$ is positive, on-positive, crossing, on-negative, or negative, respectively. D is said *non-negative* if every disks $d \in D$ is either positive or on-positive. D is said *non-positive* if every disks $d \in D$ is either negative or on-negative. D is said *expanded non-negative* if every disks $d \in D$ is either positive, on-positive, or crossing. D is said *expanded non-positive* if every disks $d \in D$ is either negative, on-negative, or crossing.

Some disk sets from Fig. 2: $\{d_1\}$ positive; $\{d_2\}$ on-positive; $\{d_3\}$ crossing; $\{d_4\}$ on-negative; $\{d_5\}$ negative; $\{d_1, d_2\}$ non-negative; $\{d_5, d_4\}$ non-positive; $\{d_1, d_2, d_3\}$ expanded non-negative; $\{d_5, d_4, d_3\}$ expanded non-positive. In the presentation of `QuickhullDisk` algorithm, expanded non-positive disk sets are of primary interest.

Definition 4. $D' \subseteq D$ is maximal if D' contains all possible disks in D satisfying a desired property.

For example, if D' contains all positive disks of D w.r.t. Δ, D' is the maximal positive subset of D w.r.t. Δ; If D' contains all possible negative or on-negative disks of D w.r.t. Δ, D' is the maximal non-positive subset of D w.r.t. Δ.

Consider an ordered triple of points a, b, and c. The signed distance of c from $\Delta = \Delta(a, b)$ is given by

$$\text{dist}(c, \Delta) = \frac{\text{Orient}(a, b, c)}{\sqrt{(b_x - a_x)^2 + (b_y - a_y)^2}}. \tag{1}$$

The point c is in the positive side of $\Delta(a, b)$ if and only if dist(c, Δ) > 0, in the negative side if and only if dist(c, Δ) < 0, and on Δ if and only if dist(c, Δ) = 0. Therefore, we can easily verify that a disk $d(c, r)$ is positive if and only if

$$\text{dist}(c, \Delta) > r. \tag{2}$$

Observe that the denominator is independent of c. If $\text{dist}(c, \Delta) < -r$, d is entirely placed in the negative side of Δ. If dist(c, Δ) ≥ r, d is non-negative and if $\text{dist}(c, \Delta) \le -r$, d is non-positive.

Definition 5. An oriented line Δ is a *non-negative support* of a point set S if all points of S are non-negative w.r.t. Δ and $\Delta \cap S \neq \emptyset$. Δ is a *non-negative support* of a set D of disks if D is non-negative w.r.t. Δ and there exists $d \in D$ such that $d \cap \Delta \neq \emptyset$. A non-negative support of D is a *convex hull support*.

Definition 6. A point $p \in S$ is an *extreme point* of $Conv(S)$ if and only if $p \notin Conv(S \backslash \{p\})$.

In Fig. 3(a), $\Delta(p_1, p_2)$ is a non-negative support and a convex hull support of the point set. $p_1, p_2, \ldots, p_8$ are extreme points. A point on the convex hull boundary is called a *hull point*. Hence, an extreme point is also a hull point. In the figure, p_9 in addition to p_1 through p_8 is a hull point. The other unfilled circles do not affect to the convex hull even if they are removed from S.

Definition 7. A disk $d \in D$ is an *extreme disk* (or a *hull disk*) if, and only if, ∂d passes through an extreme point (or a hull point) of $Conv(D)$ and d is not contained by another disk.

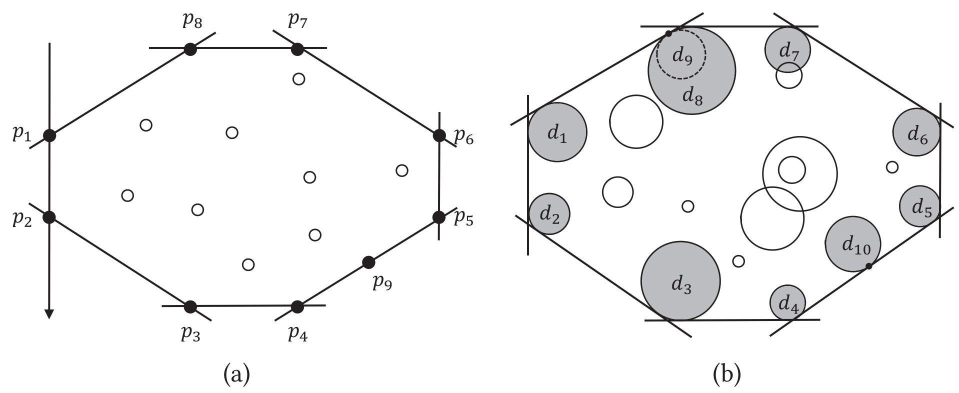


Fig. 3. Extreme points, hull points, extreme disks, and hull disks. (a) Point set. $\Delta(p_1, p_2)$ is a non-negative support and the convex hull support of the point set. $p_1, p_2, \ldots, p_8$ are extreme points. $p_1, p_2, \ldots, p_9$ are hull points. (b) Disk set. $d_1, d_2, \ldots, d_8$ are extreme disks. d_9 is not an extreme disk ($d_9 \subset d_8$). d_{10} is a hull disk.

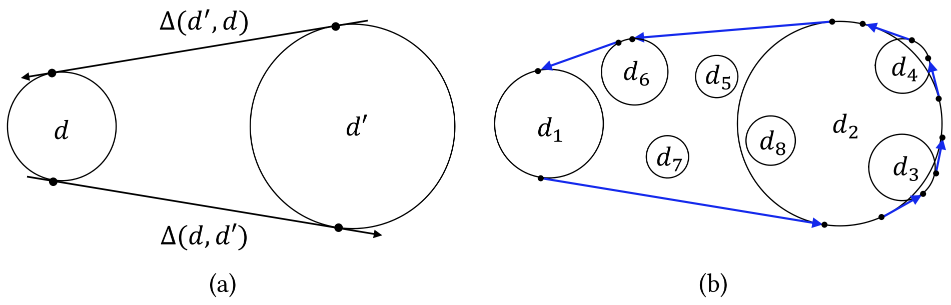


Fig. 4. Convex hull supports and the representation of hull disks. (a) The non-negative convex hull supports of two disks. (b) Representation of the convex hull as a counterclockwise-ordered hull disks in HullDisks$(D) = (d_1, d_2, d_3, d_2, d_4, d_2, d_6, d_1)$. Seven blue arrows represent the oriented linear hull edges. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

In Fig. 3(b), $d_1, d_2, \ldots, d_8$ are extreme disks; d_{10} is a hull disk. Let d and d' be two consecutive extreme disks in D in counterclockwise orientation. Let $\Delta = \Delta(d, d')$ be an oriented common tangent line of d and d' where (i) both d and d' are on-positive w.r.t. Δ and (ii) its orientation is given from the tangential point at d to the tangential point at d'. Then, $\Delta(d, d')$ is said the non-negative support of the disk pair (i.e., d and d') and D.

Caution. In Fig. 3(b), d_9 is not an extreme disk because it is contained by d_8. In such a very special case, it is possible that more than one disk can contact at an extreme point and we choose the disk having the largest radius. Note. We remove the contained smaller disk on-the-fly during convex hull computation. We do not preprocess to detect and remove such a tangential contacts because this preprocessing may take a significant amount of time. Three possible approaches for this preprocessing if wanted: $O(n^2)$ time brute-force pairwise test; $O(n\log n)$ time plane-sweeping method [59]; Expected $O(n)$ time bucket-based hashing method.

Lemma 1. *Suppose that d and d′ are consecutive extreme disks of Conv(D) in counterclockwise orientation. Then, $\Delta(d, d')$ is the convex hull support of D.*

Let $\Lambda = \Lambda(d, d') \subset \Delta(d, d')$ be the oriented subsegment with the start point on d and the end point on d'. Then, $\Lambda(d, d')$ is a *convex hull edge* (which is a linear hull edge) on $\partial Conv(D)$. Fig. 4(a) shows two disks d and d' together with two oriented tangent lines $\Delta(d, d')$ and $\Delta(d', d)$. Both lines are non-negative support of $\{d, d'\}$: Both d and d' are placed in the left half-plane divided by the oriented tangent lines. A convex hull can be represented by either (i) an ordered set of edges connecting extreme points, or (ii) an ordered set of edges connecting hull points. If we adopt the second approach of using hull points, we have more information about the arrangement of input disks than using extreme points only but with using an increased computational resources. Thus, if the marginal increase of computational resource is not significant, the latter approach is usually preferred: We take the second approach in this study.

We represent the constructed $Conv(D)$ by adopting Rappaport's method to store hull disks of D in a counterclockwise-ordered sequence in HullDisks$(D) = (d_1, d_2, ..., d_h, d_{h+1})$, $d_1 = d_{h+1}$ [30]. An edge of $\partial Conv(D)$ is an appropriate subsegment Λ of the non-negative support tangent line $\Delta(d_t, d_{t+1})$ for $t = 1, ..., h$. A disk may appear more than once in HullDisks(D) because it may contribute more than one arc on $\partial Conv(D)$. In Fig. 4(b), D has eight disks, d_1 through d_8, and HullDisks$(D) = (d_1, d_2, d_3, d_2, d_4, d_2, d_6, d_1)$: d_2 appears three times in HullDisks(D).

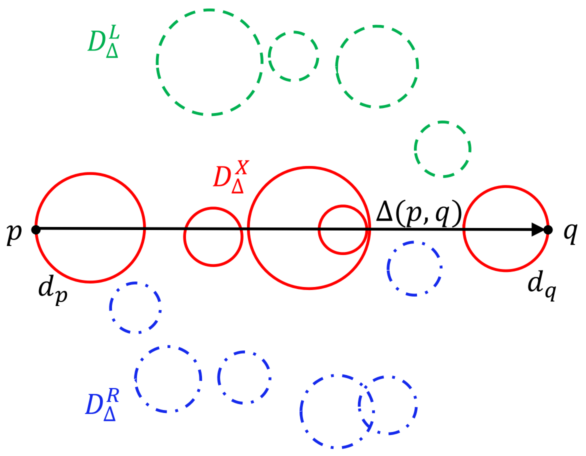


Fig. 5. Oriented line $\Delta(p, q)$, D_Δ^X, D_Δ^L, and D_Δ^R. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

Rappaport showed a tight bound $|\text{HullDisks}(D)| \le 2n - 1$ which means $2(n-1)$ tangent line segments on $\partial Conv(D)$ of n disks in the worst case [30]. We observe that this worst case can be realized only when the disks are arranged in a configuration satisfying the following rule.

Rule 1. (Disk placement rule for maximal linear hull edges) Each disk creates two linear hull edges without removing any linear hull edge created by other disks.

This situation can happen in two types of arrangements. The first is a *round arrangement* as shown in [30]: $n-1$ small disks are adjacent to one large disk in the middle. Fig. 1(b) shows such an example where $n = 5$. In this case, each disk satisfies the disk placement rule except the large one. The second is a *linear arrangement*: Disks are arranged in a linear fashion (known as a sausage configuration [60]) such as Fig. 16(d) where n congruent disks are placed on horizontal grids: Each disk satisfies the disk arrangement rule except the very first one in the left. In this case, the leftmost and the rightmost disks are extreme disks and the others are all hull disks. It is also possible in a linear arrangement that the disks are polysized in that each disk in the right is smaller than the one in the left. In both round and linear arrangements, the distributions of disk sizes and locations are rather constrained.

3. The `QuickhullDisk` algorithm

The proposed `QuickhullDisk` algorithm is a divide-and-conquer algorithm. It constructs the convex hull of a disk set D by dividing it into two subsets and quickly conquering the results of the subsets to get the solution of the entire set D. The algorithm recurs until one or two disks are left in the set so that a stopping-condition for a further recursion is encountered.

3.1. Finding two extreme points and their corresponding extreme disks

The algorithm starts with two disks in the disk set D which are guaranteed to be extreme disks. Let p and q be the highest leftmost point and the lowest rightmost point of D, respectively. Then, p and q are extreme points: i.e., $p, q \in \partial Conv(D)$. Let $p \in \partial d_p$ and $q \in \partial d_q$. Then, d_p and d_q are extreme disks contributing to the convex hull boundary. Both d_p and d_q can be found in $O(n)$ time by scanning D.

Given a pair of extreme points p and q, the algorithm recurs with subsets of D to find an additional extreme disk(s) if it exists. Given an oriented line, called a *base line*, $\Delta(p, q)$, it suffices to find the maximal expanded non-positive disk set (where every disk is in the right half-plane of $\Delta(p, q)$ or intersects $\Delta(p, q)$).

3.2. Determining the initial subsets D_R^{Init} and D_L^{Init}

Let $D_\Delta^R \subseteq D$ be maximal non-positive (i.e., the set of all negative and on-negative disks in D) w.r.t. an oriented line Δ. In other words, $d \in D_\Delta^R$ is placed in the right half-plane of or tangent to Δ. Let $D_\Delta^L \subseteq D$ be maximal non-negative w.r.t. the oriented line Δ. Let $D_\Delta^X \subseteq D$ be maximal crossing. Let Δ^{-1} be the line with the reversed orientation of Δ. Hence, D_Δ^R and D_Δ^L are maximal non-positive w.r.t. Δ and Δ^{-1}, respectively. Let $D_\Delta^{R+} = D_\Delta^R \cup D_\Delta^X$ and $D_\Delta^{L+} = D_\Delta^L \cup D_\Delta^X$. Hence, D_Δ^{R+} and D_Δ^{L+} are *maximal expanded non-positive disk sets* w.r.t. Δ and Δ^{-1}, respectively. See Fig. 5: D and $\Delta(p, q)$ of the extreme points p and q. The blue (dash-dotted) disks define D_Δ^R, the green (dashed) ones D_Δ^L, and the red (solid) ones D_Δ^X. The blue and red disks together define D_Δ^{R+} and the green and red disks together D_Δ^{L+} w.r.t. $\Delta(p, q)$.

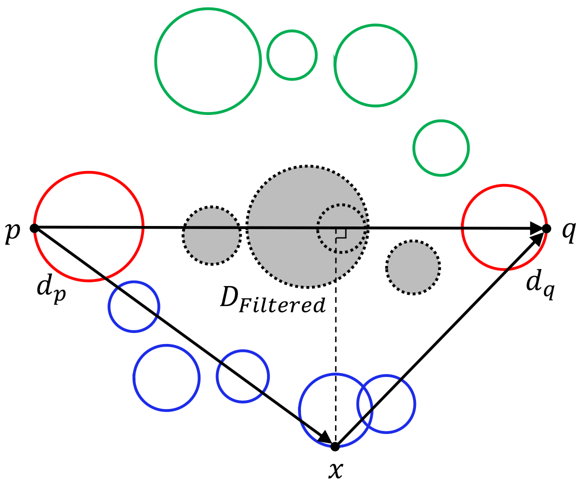


Fig. 6. The triangle apex x w.r.t. $\Delta(p, q)$, a triangle filter $Tri(p, x, q)$ defined by three ordered vertices p, x, and q, and the filtered disk set $D_{Filtered}$ by $Tri(p, x, q)$ (shown by the dotted circles).

Note that $D^L_\Delta = D^R_{\Delta^{-1}}$ and $D^{L+}_\Delta = D^{R+}_{\Delta^{-1}}$.

In the initial phase, `QuickhullDisk` splits D into two sets of disks, say $D^{Init}_R \subseteq D$ and $D^{Init}_L \subseteq D$, and conquers the returning results from the two recursions using each of D^{Init}_R and D^{Init}_L to obtain the result of D. Let $D^{Init}_R = D^{R+}_\Delta$ and $D^{Init}_L = D^{L+}_\Delta$ w.r.t. $\Delta(p, q)$ of two extreme points p and q. The `QuickhullDisk` algorithm begins the recursion with each of D^{Init}_R and D^{Init}_L.

Another interpretation. In the initial phase, the two extreme points p and q need to be used twice with opposite contexts: $\Delta(p, q)$ and $\Delta(q, p)$. With each oriented line, we need to process D to find its expanded non-positive disk set D^{R+}_Δ. As this requires scanning D twice, we instead collect D^{Init}_R and D^{Init}_L in one scan and interprete D^{Init}_L as the expanded non-positive disk set D^{R+}_Δ for $\Delta(q, p)$. The following lemma holds.

Lemma 2. *D^{Init}_R and D^{Init}_L can be constructed in $O(n)$ time for a set D of n disks.*

3.3. Pivoting disks using the apex disk

The most critical idea of `QuickhullDisk` is pivoting the disks in D. *Pivoting* is a process to divide the disks in D into three different types of subsets w.r.t. a disk called a *pivot disk*. Pivoting is critical in that the choice of a pivot disk has a direct influence on both correctness and efficiency of the algorithm. Hereafter D may represent the set of disks in a recursion process and in such a case, it is a subset of the input disk set. If it is necessary, we use D^* to explicitly denote the entire input disk set.

Definition 8. (Fig. 6) Assume D and $\Delta(p, q)$ for extreme points p and q. The point $x \in d \in D^{R+}_\Delta$ with the maximal distance from Δ is called the *(highest) triangle apex*. The disk which gives the highest triangle apex is called the *(triangle) apex disk*.

Lemma 3. *A triangle apex is an extreme point if it is unique. Otherwise, a triangle apex is a hull point. The corresponding apex disk is an extreme or hull disk.*

We use a triangle apex disk as a pivot disk. The apex disk can be found by scanning the disks in D, taking $O(|D|)$ time. (*Multiple apexes case:* If there are multiple triangle apexes with the same height in D, we choose one at random. Be aware that this choice of pivot disk is critical to keep the expected $O(n\log n)$ time complexity for highly degenerate data sets.)

Definition 9. Let x be the triangle apex w.r.t. $\Delta(p, q)$ for extreme points p and q. Let $Tri(p, x, q)$ be the triangle, called a *triangle filter*, defined by three ordered vertices p, x, and q. The oriented edges $Edge_F(p, x)$, $Edge_B(x, q)$, and $Edge_{BASE}(p, q)$ are called the *front edge*, *back edge* and *base edge* of $Tri(p, x, q)$, respectively.

Be aware that $Edge_F(p, x)$ and $Edge_B(x, q)$ are in the counterclockwise order around $Conv(D)$. Let $p \in \partial d_p$ and $q \in \partial d_q$. We call d_p and d_q the *pre-apex disk* and the *post-apex disk*, respectively.

Pivoting proceeds as follows. Given a disk set D, we find $\Delta(p, q)$ and $Tri(p, x, q)$. Let $D^{R+}_{\Delta,F} = D^{R+}_\Delta$ w.r.t. $\Delta = \Delta(p, x)$ which corresponds to $Edge_F(p, x)$. We collect $D^{R+}_{\Delta,F} \subset D$ and recurs with it. Then, we collect $D^{R+}_{\Delta,B} = D^{R+}_\Delta$ w.r.t. $\Delta = \Delta(x, q)$ from $Edge_B(x, q)$ and recurs with it. The recursion stops when a stopping condition is encountered and returns an appropriate output. The following lemma holds.

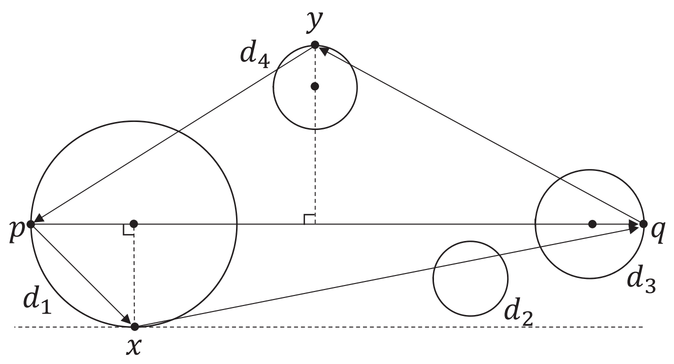


Fig. 7. The triangle apex x occurring in the same disk that p is defined.

Lemma 4. *The pivoting of D can be done in $O(|D|)$ time in each recursion.*

Data structure: The initial disk set D^* is stored in a linked list $List_{D^*}$ and each node of $List_{D^*}$ has the definition of each disk. The subset $D \subset D^*$ which goes into the recursion process is represented as a linked list of pointers to the appropriate nodes of $List_{D^*}$.

Pivot effect: Let $D(Tri(p, x, q))$ be the set of disks which intersect $Tri(p, x, q)$. It is guaranteed that the disks in $D_{Filtered} = D(Tri(p, x, q)) \backslash \{D^{R+}_{\Delta,F} \cup D^{R+}_{\Delta,B}\}$ do not contribute to $\partial Conv(D)$. Thus, `QuickhullDisk` ignores $D_{Filtered}$ from the recursion and in this regard $Tri()$ is called a filter. The dotted disks in Fig. 6 belong to a $D_{Filtered}$. The size of $D_{Filtered}$ is closely related to the actual computation time. Recall the expected $O(n\log n)$ time quicksort algorithm does not ignore any data element in any level of recursion. On the other hand, `QuickhullDisk` can ignore $D_{Filtered}$ during recursion and its running time can have a relatively small coefficient for the expected $O(n\log n)$ time complexity.

4. Recursion stop conditions

One of the most critical tasks of the recursive function is to determine when to stop the recursion and what to return. There are two types of stop-condition which depends on the size of D: $|D| = 1$ and $|D| = 2$. We model the recursion process as a binary tree called a *recursion tree* where a leaf node represents a recursion stop situation.

4.1. D *has one disk :* $|D| = 1$.

In this case, the apex disk, the pre-apex disk, and the post-apex disk are identical. The recursion stops and D is entirely returned. This corresponds to one of the leaf nodes of the recursion tree.

See Fig. 7 which shows $D = \{d_1, d_2, d_3, d_4\}$ and $\Delta(p, q)$ for the two extreme points p and q. Hence, $D^{R+}_{\Delta} = \{d_1, d_2, d_3\}$, and $D^{L+}_{\Delta} = \{d_1, d_3, d_4\}$ w.r.t. $\Delta(p, q)$. Note $p \in d_p \equiv d_1$ and $q \in d_q \equiv d_3$. For D^{R+}_{Δ}, the triangle apex x is also defined on ∂d_p so that $d_p \equiv d_x$: i.e., the triangle apex occurs at the pre-apex disk. Hence, the two recursions in the next iteration are based on $D^{R+}_{\Delta,F} = \{d_1\}$ and $D^{R+}_{\Delta,B} = \{d_1, d_2, d_3\}$ where $\Delta = \Delta(p, q)$. Therefore, the next recursion with $D^{R+}_{\Delta,F}$ stops because $|D^{R+}_{\Delta,F}| = 1$ but the one with $D^{R+}_{\Delta,B}$ further recurs because $|D^{R+}_{\Delta,B}| = 3 > 2$.

4.2. D *has two disks :* $|D| = 2$.

There are four cases of the stop-condition when $|D| = 2$: See Fig. 8(a) through (d).

Case I. Fig. 8(a) shows an ordinary case of $|D| = 2$ where $d_p \not\equiv d_q$ for the extreme points p and q. In other words, the two extreme points occur at different disks. In this case, the recursion stops and returns D to the parent in the recursion tree. This corresponds to one of the leaf nodes of the recursion tree. The convex hull support of the two disks in D, which is straightforward to compute, is the convex hull support of the entire set of the input disks. Example. Consider $D = D^{R+}_{\Delta} = \{d_1, d_3, d_4\}$ w.r.t. $\Delta(q, p)$ in Fig. 7. The triangle apex is shown as $y \in \partial d_4$. Note $Tri(q, y, p)$ with the front edge $Edge_F(q, y)$ and the back edge $Edge_B(y, p)$. We recurse in the next iteration with $D^{R+}_{\Delta,F} = \{d_3, d_4\}$ w.r.t. $\Delta(q, y)$ and with $D^{R+}_{\Delta,B} = \{d_4, d_1\}$ w.r.t. $\Delta(y, p)$. Then, each recursion becomes the case of Fig. 8(a).

Case II. Fig. 8(b) shows the case that $d_x \equiv d_p \equiv d_q$ and $d' \subset d_p$. Then, this case is identical to the case $|D| = 1$. Hence, we stop the recursion and return d_p to the parent in the recursion tree. This corresponds to one of the leaf nodes of recursion tree.

Case III. Fig. 8(c) shows the case that $d_p \equiv d_q$ and $d_x \not\equiv d_p$. In this case, we recurse once more with $D^{R+}_{\Delta,F} = \{d, d'\}$ for $Edge_F(p, x)$ and $D^{R+}_{\Delta,B} = \{d, d'\}$ for $Edge_B(x, q)$. Note that each of the recursions with $D^{R+}_{\Delta,F}$ and $D^{R+}_{\Delta,B}$ becomes the case in Fig. 8(a) in the next recursion. Therefore, this does not yet correspond to a leaf node of recursion tree.

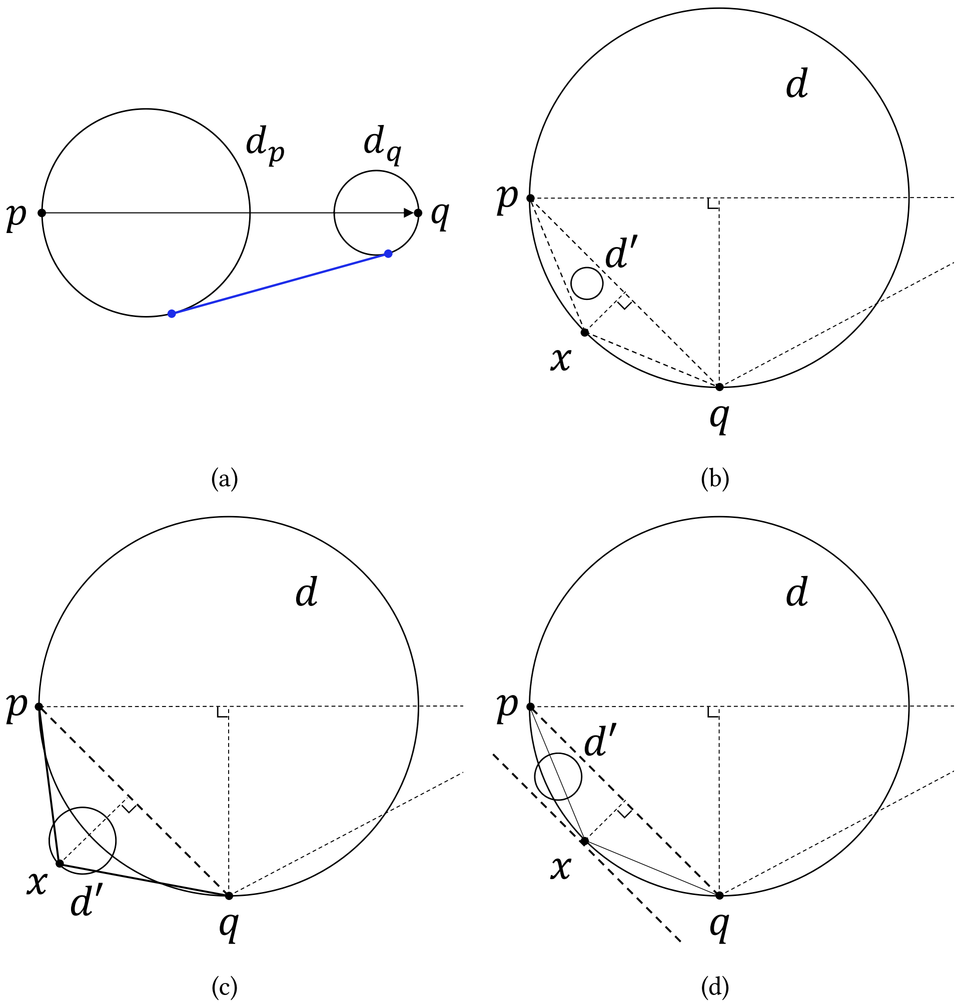


Fig. 8. Four cases of two disks in D, two extreme points p and q, and $\Delta(p, q)$. Recursion may stop or may continue depending on condition. (a) $p \in d_p$ and $q \in d_q$ where $d_p \not\equiv d_q$. (b) $d_x \equiv d_p \equiv d_q$ and $d' \subset d_p$. (c) $d_p \equiv d_q$ and $d_x \not\equiv d_p$. (d) $d_x \equiv d_p \equiv d_q$ and $d' \not\subset d_p$.

Case IV. Fig. 8(d) shows the case that $d_x \equiv d_p \equiv d_q$ and $d' \not\subset d_p$. In this case, we recurse once more with $D^{R+}_{\Delta,F} = \{d, d'\}$ for $Edge_F(p, x)$ and $D^{R+}_{\Delta,B} = \{d\}$ for $Edge_B(x, q)$. Then, the recursion with $D^{R+}_{\Delta,F}$ is the case of $|D| = 2$ and it may recurse "a few more times" depending on the size and the location of d' before it eventually become the case in Fig. 8(c).[2] The recursion with $D^{R+}_{\Delta,B}$ is the case with $|D| = 1$ in the next recursion which immediately stops the recursion. Therefore, Case IV does not yet correspond to a leaf node of the recursion tree but we need not pay any extra attention on this case.

5. Sliver triangle filter: an intriguing situation

Fig. 9(a) shows eight congruent disks placed at grids so that each triplet of disks on the convex hull boundary is tangent to a common line. To be specific, d_1, d_2, and d_3 are hull disks which are tangent to a common tangent line. With $\Delta(p, q)$ with the two extreme points $p \in \partial d_1$ and $q \in \partial d_5$, the triangle apex is found at $x \in \partial d_3$. Fig. 9(b) shows the details of this situation. Given the triangle apex $x \in \partial d_3$, the next apex using $\Delta(p, x)$ is determined at $x' \in \partial d_3$ with the disk set $\{d_1, d_2, d_3\}$. Another iteration also yields $x'' \in \partial d_3$ with the same disk set. Even if d_1, d_2, and d_3 are all tangent to a common tangent

[2] From both theoretical and practical perspectives, "a few more times" indeed means a small constant number of times in that the new apex x of each recursion bisects the angle $\angle pcq$ where $d(c, r)$ has center c and radius r. It can be easily shown that, for a unit disk d (i.e., $r = 1.0$), 25 recursions guarantee that the length of the line segment pq is less than 10^{-7} (which is approximately the machine epsilon of single precision floating point number and similar observation with 55 recursions guarantees that the length of pq is less than 10^{-16} which is approximately the machine epsilon of double precision floating point number. For simplicity of presentation, hereafter we only discuss with single precision case). If one wishes, as a variation, it is obviously possible to find d' directly to avoid a few recursions but at the cost of having an algorithm fragment to handle the special situation.

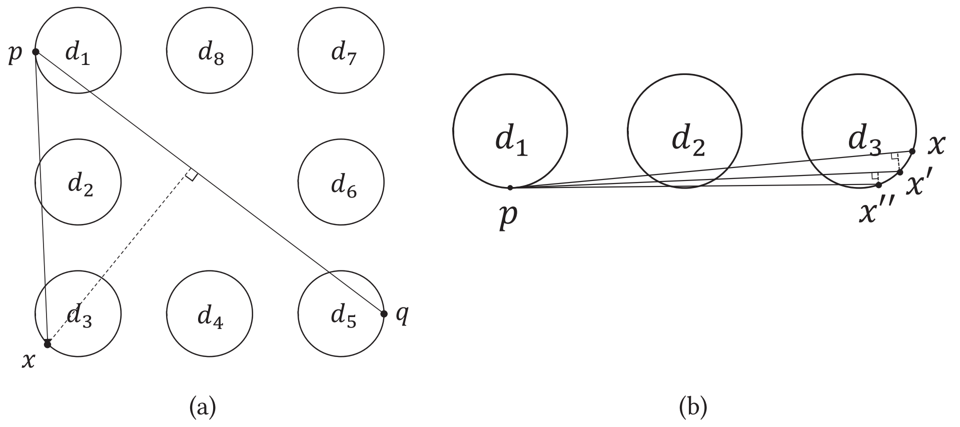


Fig. 9. Captured apex event on d_3. All apexes x, x', and x'' occur on ∂d_3. (a) Eight congruent disks placed at grids so that each triplet of disks on the convex hull boundary is tangent to a common line. (b) Recursion process with $\Delta(p, q)$, $\Delta(p, x)$, $\Delta(p, x')$, $\Delta(p, x'')$, ...

line, this cotangency information cannot be directly obtained (Note that it is possible to obtain the similar information for a multiple triangle apexes case).

In Fig. 9, we observe $Tri(p, x', x)$ and $Tri(p, x'', x')$ are *sliver triangles* like needles and the disk triplets which define the triangle filters do not change after pivot operations. In such a case, we also observe all three apexes x, x', and x'' occur on the boundary of an identical disk, i.e., ∂d_3 in this example: We call this phenomenon a *captured apex property*. Regardless how many times we repeat pivoting operation in this case, we end up with the same disk set $\{d_1, d_2, d_3\}$ for $Edge_F$, only with a slightly improved apex on the same disk d_3. When a triangle filter is a sliver triangle, a persistent disk set tends to repeat and recursions may possibly not terminate. If we do not have a criteria to detect such a situation and enforce to stop recursion, we may run into an infinite loop. Note that such a case can occur in engineering designs.

Definition 10. Consider the disk sets $D_F \subset D$ and $D_B \subset D$ respectively corresponding to $Edge_F$ and $Edge_B$ after a pivoting, $|D| \geq 3$. D has a captured apex property if (i) $|D_F| = |D|$ and $|D_B| = 1$, or (ii) $|D_F| = 1$ and $|D_B| = |D|$. If D has a captured apex property, D is called *persistent*.

Definition 10 implies that if a disk set D is persistent, the arrangement of the disks in D is skewed. We can detect the skewed arrangement simply by counting the number of disks in the expanded non-positive disk sets corresponding to $Edge_F$ and $Edge_B$. Hence, it can be done in an $O(|D|)$ time.

Regularizing a sliver triangle filter: When a skewed disk arrangement is detected due to a captured apex by a sliver triangle filter, we want to quickly walk around a possible degeneracy due to many cotangent disks. If such a case as shown in Fig. 9 is detected, we improve the shape of the sliver triangle filter so that it becomes closer to a regular triangle as much as possible. Given the extreme points p and q and their corresponding disks d_p and d_q, respectively, we first compute an oriented non-negative tangent line $\Delta = \Delta(d_p, d_q)$ between d_p and d_q which places the two disks in its positive space. Let $\tilde{p} = \partial d_p \cap \Delta$, i.e., the point where d_p and Δ have a tangential contact. Let $\tilde{q} = \partial d_q \cap \Delta$. Then, we test every disk $d \in D$ if d is positive or not w.r.t. Δ. Let $\tilde{\Delta} = \tilde{\Delta}(\tilde{p}, \tilde{q})$ be an oriented line. Then, $\Delta = \tilde{\Delta}$. Let $D^{Non\text{-}Pos} \subset D$ be the maximal expanded non-positive disks of D w.r.t. Δ. Let $D^{On\text{-}Pos} \subset D$ be the maximal on-positive disks of D w.r.t. Δ.

- Case (A) $D^{Non\text{-}Pos} = \emptyset$ and $|D^{On\text{-}Pos}| = 2$. In this case, Δ is a convex hull support of d_p and d_q and at the same time it is the convex hull support of the entire input disk set: i.e., all disks are in the positive half-plane (except the on-positive d_p and d_q). (Fig. 10(a)). We choose either $\tilde{p}$ or $\tilde{q}$ as an improved triangle apex $\tilde{x}$.
- Case (B) $D^{Non\text{-}Pos} = \emptyset$ and $|D^{On\text{-}Pos}| \geq 3$ i.e., there are (is) on-positive disks to Δ. (Fig. 10(b)). In this case, we choose a disk $d \in D^{On\text{-}Pos}$ at random or by a bisection method and improve the triangle apex with $\tilde{x} = \partial d \cap \Delta$.
- Case (C) $D^{Non\text{-}Pos} \neq \emptyset$ and there exists at least one disk $d \in D^{Non\text{-}Pos}$ such that d is crossing, on-negative, or negative w.r.t. Δ. For each $d \in D^{Non\text{-}Pos}$, let $\tilde{x}_d = \partial d \cap L^-$ where (i) L^- is the line parallel to Δ, and (ii) the height of $\tilde{x}_d$ from Δ is maximal.
  - Case (C-I) Suppose that there exists a unique disk d which defines a unique highest apex point $\tilde{x}_d \subset \partial d$ (Fig. 10(c)). Then, let $\tilde{x} = \tilde{x}_d$.
  - Case (C-II) Suppose that there are two or more disks which correspond to triangle apexes with an identical height. In this case, we choose a disk d among these disks and define $\tilde{x} = \tilde{x}_d \subset \partial d$ (Fig. 10(d)). The choice can be made either at random or in a bisection method.

Lemma 5. *$\tilde{x}$ is a hull point.*

Based on Lemma 5, we regularize the triangle filter to improve from $Tri(p, x, q)$ to $Tri(p, \tilde{x}, q)$ and repeat the current recursion once more.

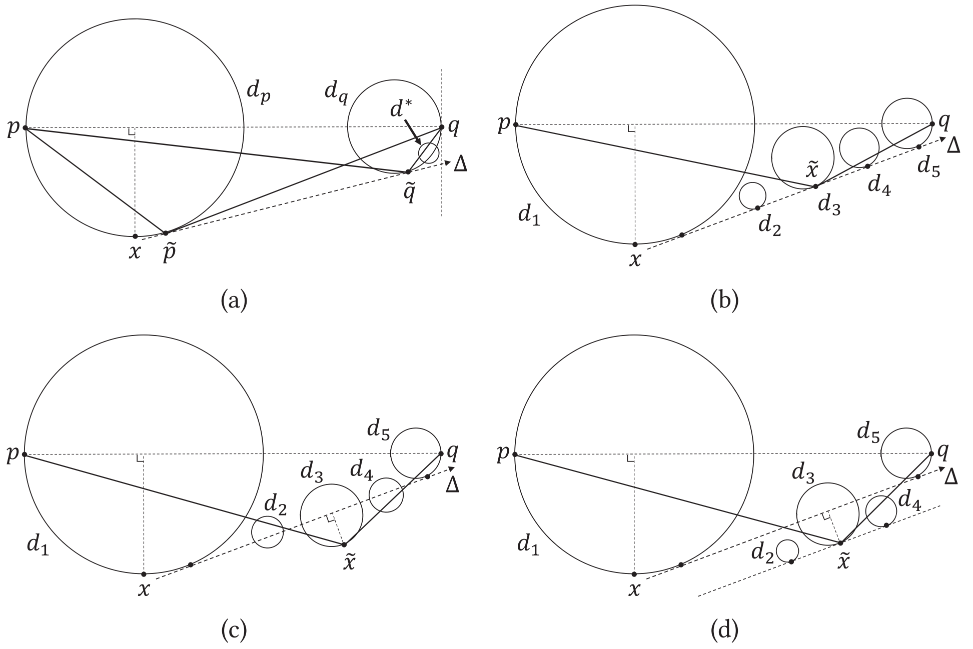


Fig. 10. Sliver triangle filters and their regularizatoin. (a) Case (A). All disks in D are in the positive space (except the on-positive d_p and d_q) w.r.t. Δ. The convex hull support of d_p and d_q are the convex hull support of entire input disks D^*. Improving $Tri(p, x, q)$ to either $Tri(p, \tilde{p}, q)$ or $Tri(p, \tilde{q}, q)$ helps to walk around the case caused by d^*. (b) Case (B). Multiple on-positive disks w.r.t. Δ. A pivot disk is chosen either at random or via a bisection. $Tri(p, \tilde{x}, q)$ is an improved triangle filter where $\tilde{x} = \Delta \cap \partial d$. (c) Case (C-I). There exists a unique disk d_3 which defines a unique highest apex point $\tilde{x} \subset \partial d_3$ w.r.t. Δ. We improve $Tri(p, x, q)$ to $Tri(p, \tilde{x}, q)$. (d) Case (C-II). There are two or more disks which correspond to triangle apexes with an identical height w.r.t. Δ. We choose a pivot disk d_3 at random or via a bisection to make a unique highest apex point $\tilde{x} \subset \partial d_3$ w.r.t. Δ. We improve $Tri(p, x, q)$ to $Tri(p, \tilde{x}, q)$.

Caution. Let $Arc(d;\ \alpha,\ \beta)$ be the counterclockwise arc of ∂d from $\alpha \in \partial d$ to $\beta \in \partial d$. There is no guarantee that $Arc = Arc(d_p; p, \tilde{p})$ is entirely on $\partial Conv(D)$. This is because there can be one or more tiny disk which contributes to $\partial Conv(D)$ in the middle of Arc. The disk d^* in Fig. 10(a) shows such a case on $Arc(d_q; \tilde{q}, q)$. The regularization of a triangle filter guarantees solution correctness.

Note. The improved filter $Tri(p, \tilde{q}, q)$ is can be actually sliver than the original $Tri(p, x, q)$ (E.g. $Tri(p, \tilde{q}, q)$ in Fig. 10(a)). The "regularization" might be a misnomer in this regard. However, the regularization concept is proper in other cases and we stick to this notion.

Notes on cotangent disks: We make a few observations on cotangent disks. First, suppose that we have identified $k > 0$ cotangent disks and assume that $k-2$ disks are placed between d_1 and d_k. Depending on how we define the visibility of collinear points, we may either detect the $k-2$ internal disks as hull disks or may treat them as non-hull (but internal) ones. In this paper, we treat them as extreme ones. One of the obvious reasons is to obtain as much information about the input data as possible unless the marginal computational cost is too expensive. Note that, as a straightforward minor variation, one can ignore the $k-2$ disks between d_1 and d_k to accelerate. Second, why do we avoid a sequential visit to the k cotangent disks (in the Euclidean sense)? This is simply because we want to have the expected $O(n\log n)$ time complexity for n input disks. If we pivot so that two subsets D_F and D_B have 2 and $k-1$ disks, respectively, the pivoting process eventually exhibits an $O(k^2)$ worst case time complexity. See Fig. 22. Third, the regularization of sliver triangle is a conceptual aide in that the improved triangle filter is still sliver for degenerate or near degenerate disk arrangements. Fourth, an oriented line $\Delta(p,\ q)$ of extreme points p and q can be an oriented tangent line $\Delta(d_p,\ d_q)$ of the disks d_p and d_q which correspond to

Algorithm 1: `QuickhullDisk`$(D^*, \text{HullDisks}(D^*))$.

```
    Input: $D^*$: Input disks
    Output: HullDisks($D^*$): ordered sequence of hull disks on convex hull boundary
1   If $|D^*| = 1$, then return HullDisks$(D^*) = D^*$.
2   Find $p$, $q$, $d_p$, and $d_q$ (where $p \in \partial d_p$, $q \in \partial d_q$).
3   Divide $D^*$ into two sets of disks, $D_R^{Init} (\equiv D_\Delta^{R+}) \subseteq D^*$ and $D_L^{Init} (\equiv D_\Delta^{L+}) \subseteq D^*$ w.r.t. $\Delta(p, q)$.
4   FindHull$(D_R^{Init}, d_p, d_q, \text{HullDisks}(D_R^{Init}))$; FindHull$(D_L^{Init}, d_q, d_p, \text{HullDisks}(D_L^{Init}))$.
5   Return HullDisks$(D^*) \equiv$ HullDisks$(D_R^{Init}) \cup$ HullDisks$(D_L^{Init})$.
```

Algorithm 2: `FindHull`$(D, d_p, d_q, \text{HullDisks}(D))$.

```
1 If ($(|D| = 1)$ or $(|D| = 2$ and $d_p \not\equiv d_q)$, then return HullDisks($D$).
2 Find apex disk $d_x$.
3 Find front edge $E_F$ and back edge $E_B$ of apex triangle.
4 Make expanded non-positive disks $D_F \subseteq D$ and $D_B \subseteq D$ w.r.t. $E_F$ and $E_B$, respectively.
5 If triangle filter is sliver, then
6     RegularizeSliverTriangleNPivotDisks($D, D_F, D_B, d_p, d_q, d_x$).
7 End if
8 FindHull($D_F, d_p, d_x$, HullDisks($D_F$)); FindHull($D_B, d_x, d_q$, HullDisks($D_B$))).
9 Return HullDisks($D$) $\equiv$ HullDisks($D_F$) $\cup$ HullDisks($D_B$).
```

Algorithm 3: `RegularizeSliverTriangleNPivotDisks`$(D, D_F, D_B, d_p, d_q, d_x)$.

```
1 Make oriented tangent line $\Delta = \Delta(\tilde{p}, \tilde{q})$ where $\tilde{p} = \partial d_p \cap \Delta$ and $\tilde{q} = \partial d_q \cap \Delta$.
2 Make expanded non-positive disks $D^{Non\text{-}Pos} \subseteq D$ and on-positive disks $D^{On\text{-}Pos} \subseteq D$ w.r.t. $\Delta$.
3 Switch (Condition of $D^{Non\text{-}Pos}$ and $D^{On\text{-}Pos}$) do
4     Case A: ($D^{Non\text{-}Pos} = \emptyset$ and $|D^{On\text{-}Pos}| = 2$ where $D^{On\text{-}Pos} \equiv \{d_p, d_q\}$)
5         Choose $d_x \in \{d_p, d_q\}$ at random and $\tilde{p}$ or $\tilde{q}$ as (triangle) apex $\tilde{x}$.
6     Case B: ($D^{Non\text{-}Pos} = \emptyset$ and $|D^{On\text{-}Pos}| \geq 3$)
7         Choose $d_x \in D^{On\text{-}Pos} \backslash \{d_p, d_q\}$ at random and apex $\tilde{x} = \partial d_x \cap \Delta$.
8     Case C-I: ($|D^{Non\text{-}Pos}| = 1$)
9         Choose $d_x \in D^{Non\text{-}Pos}$ defining unique apex $\tilde{x} \subset \partial d_x$.
10    Case C-II: ($|D^{Non\text{-}Pos}| \geq 2$)
11        Choose $d_x \in D^{Non\text{-}Pos}$ at random corresponding to apexes ($\tilde{x} \subset \partial d_x$) with identical heights.
12 End switch
13 Find front edge $E_F$ and back edge $E_B$ of apex triangle.
14 Make expanded non-positive disks $D_F \subseteq D$ and $D_B \subseteq D$ w.r.t. $E_F$ and $E_B$, respectively.
15 Return $D_F$, $D_B$, and $d_x$.
```

p and q, respectively (refer to Fig. 11(b)). In this case, we should collect *on-positive* disks as well as *expanded non-positive* disks because *on-positive* disks constitute the hull disks. Otherwise, we collect only *expanded non-positive* disks as shown in Fig. 11(a). Fifth, we want to emphasize that the incremental algorithm is prone to crash with multiple cotangent disks, particularly when an incrementing disk simultaneously touches two edges on a convex hull boundary.

6. Summary of algorithms

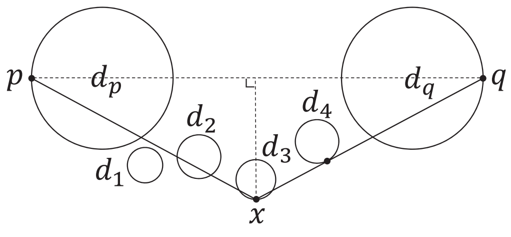
The discussions above is summarized as three pseudocode algorithms `QuickhullDisk`, `FindHull`, and `RegularizeSliverTriangleNPivotDisks` below. For simplicity of presentation, we intentionally abuse HullDisks(D) like a set. We should collect on-positive disks $D_{\Delta}^{On\text{-}Pos}$ as well as expanded non-positive disks only if an oriented line $\Delta(p, q)$ is an oriented tangent line of d_p and d_q. However, we do not include $D_{\Delta}^{On\text{-}Pos}$ in the pseudocodes for simplicity of notation.

An example of `QuickhullDisk` execution. Suppose that the entire input disk set D^* has 15 disks arranged as shown in Fig. 12(a). Fig. 12(b) shows the sequence of `QuickhullDisk`'s recursive executions with the corresponding triangle filters where each produces the related disk set D_{Δ}^{R+}. Observe that the pair p and q has two uses: $\Delta(p, q)$ and $\Delta(q, p)$. Fig. 12(c) shows the hierarchy of recursions in a binary tree. The root node A corresponds to the initial run using D^* (Not shown in Fig. 12(b)) and the rectangular leaf nodes correspond to the convex hull boundary. Hence, the number of leaf nodes is identical to the number of linear hull edges. Note the binary tree in this example is height-balanced. Table 1 shows the content of each node in the recursion tree. Observe that the execution sequence of the `QuickhullDisk` recursions is represented by the preorder traversal of the recursion tree. The leaf nodes of the tree visited by the preorder traversal provide the counterclockwise sequence of the extreme disks of D, i.e., the definition of the convex hull boundary. Note that a leaf node corresponds to a hull edge and different traversal methods produce different orderings of hull edges on the convex hull boundary.

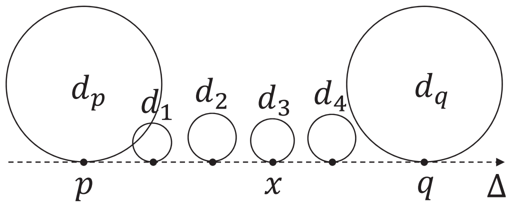
Theorem 1. *`QuickhullDisk` constructs the convex hull of n disks in $O(mn)$ time in the worst case and in $O(n\log n)$ time on average where m represents the number of extreme disks.*

7. Proofs for the performance of `QuickhullDisk`

The `QuickhullDisk` algorithm terminates because the disk set D monotonically shrinks unless a skewed arrangement is encountered (as shown in Fig. 10(a)–(d)). In other words, each pivot operation produces two subsets $D_1 \subset D$ and $D_2 \subset D$

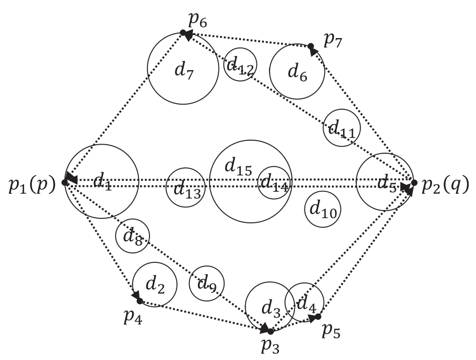


(a)

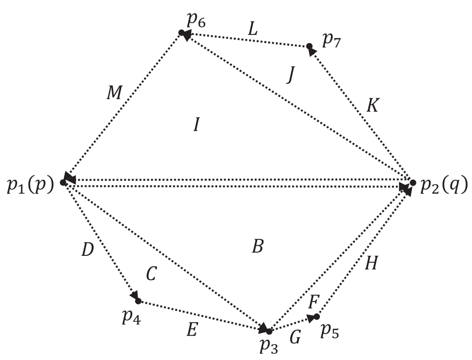


(b)

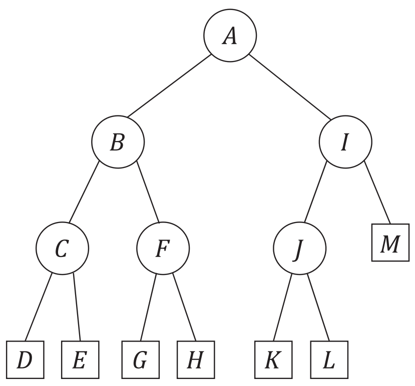
Fig. 11. Examples for an oriented line $\Delta(p,\,q)$ of a disk set D whose extreme points are p and q. (a) Non-cotangent case: disks of D are not cotangent to $\Delta(p,\,q)$. We collect $D^{R+}_{\Delta,F} = \{d_p, d_1, d_2, d_3\} \subset D$ w.r.t. $\Delta = \Delta(p, x)$ and $D^{R+}_{\Delta,B} = \{d_3, d_q\} \subset D$ w.r.t. $\Delta = \Delta(x, q)$ where $D = \{d_p, d_1, d_2, d_3, d_4, d_q\}$. Then we could recur with $D^{R+}_{\Delta,F}$ and $D^{R+}_{\Delta,B}$, respectively. (b) Cotangent case: disks of D are cotangent to $\Delta(p,\,q)$. $\Delta(p,\,q)$ becomes an oriented tangent line of d_p and d_q. Suppose that d_3 is chosen as the apex disk of a triangle apex x. $D^{R+}_{\Delta,F} = \emptyset$ w.r.t. $\Delta = \Delta(p, x)$ and $D^{R+}_{\Delta,B} = \emptyset$ w.r.t. $\Delta = \Delta(x, q)$. In this case, we need to collect on-positive disks $D^{On\text{-}Pos}_{\Delta,F} = \{d_p, d_1, d_2, d_3\}$ and $D^{On\text{-}Pos}_{\Delta,B} = \{d_3, d_4, d_q\}$ w.r.t. both $\Delta(p, x)$ and $\Delta(x, q)$, respectively, because $D^{On\text{-}Pos}_{\Delta,F}$ and $D^{On\text{-}Pos}_{\Delta,B}$ constitute the hull disks.



(a)



(b)



(c)

Fig. 12. Example of `QuickhullDisk` execution. (a) Input disk set D^*. (b) The execution sequence (identified by the corresponding highest triangles or hull edges). (c) Recursion tree (where the preorder traversal yields the execution sequence). Leaf nodes correspond to the convex hull boundary.

Table 1
The contents of nodes in the recursion tree of `QuickhullDisk` (Disk set in Fig. 12).

Input			Output				Elements of input disk set
Node	Line1 (1st line CCW)	Line2 (2nd line CCW)	$D^{R+}_{\Delta,F}$		$D^{R+}_{\Delta,B}$		
			Left child	Triangle apex	Right child	Triangle apex	
A	$\Delta(p_1, p_2)$	$\Delta(p_2, p_1)$	B	p_3	I	p_6	$d_i, i = 1, 2, \ldots, 15$
B	$\Delta(p_1, p_3)$	$\Delta(p_3, p_2)$	C	p_4	F	p_5	$d_1, d_2, d_3, d_4, d_5, d_8, d_9, d_{10}, d_{13}, d_{14}, d_{15}$
C	$\Delta(p_1, p_4)$	$\Delta(p_4, p_3)$	D		E		d_1, d_2, d_3, d_8, d_9
D			$\varnothing$		$\varnothing$		d_1, d_2
E			$\varnothing$		$\varnothing$		d_2, d_3
F	$\Delta(p_3, p_5)$	$\Delta(p_5, p_2)$	G		H		d_3, d_4, d_5
G			$\varnothing$		$\varnothing$		d_3, d_4
H			$\varnothing$		$\varnothing$		d_4, d_5
I	$\Delta(p_2, p_6)$	$\Delta(p_6, p_1)$	J	p_7	M		$d_1, d_5, d_6, d_7, d_{11}, d_{12}, d_{13}, d_{14}, d_{15}$
J	$\Delta(p_2, p_7)$	$\Delta(p_7, p_6)$	K		L		$d_5, d_6, d_7, d_{11}, d_{12}$
K			$\varnothing$		$\varnothing$		d_5, d_6
L			$\varnothing$		$\varnothing$		d_6, d_7
M			$\varnothing$		$\varnothing$		d_7, d_1

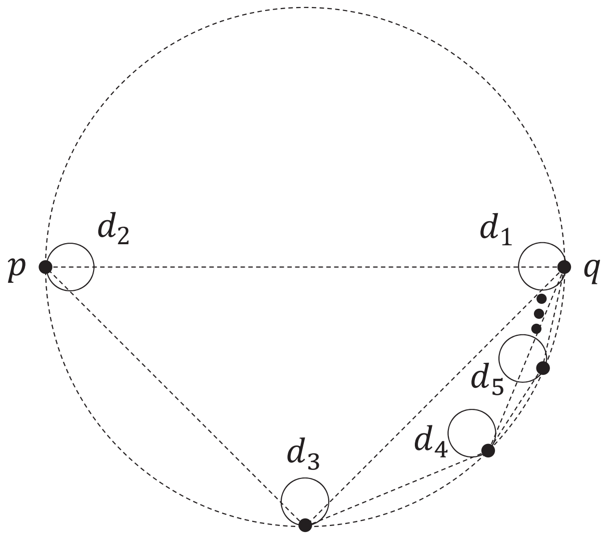


Fig. 13. Type I worst case of `QuickhullDisk`. The contacts between each disk and the container occur at the angles 0 (with d_1), π (with d_2), $3\pi/2$ (with d_3), $7\pi/4$ (with d_4), ... Note that d_3 is the apex disk w.r.t. d_1 and d_2; d_4 is the apex disk w.r.t. d_1 and d_3; This process continues for d_5, d_6, and so on. The minimum distance between two contact points quickly approaches to the single precision machine epsilon 10^{-7}. When the container has the unit radius, the distance between the extreme points on ∂d_{27} and ∂d_1 is less than 10^{-7} and on those ∂d_{24} and ∂d_1 is less than 10^{-6}.

if D is not skewed. If a skewed arrangement is indeed encountered, it was shown that, after a few iterations of special recursions, the disk set enters to a normal recursion so that the disk set renews another shrinking process. An input disk set is eventually decomposed into a set of one or two disks which immediately or soon defines a hull edge. Hence, D shrinks monotonically to terminate. The correctness of `QuickhullDisk` is proven by the discussion through out the paper.

Time Complexity. Suppose that an input disk set D consists of n disks and HullDisks(D) has m elements. Recall that an extreme disk may appear more than once in HullDisks(D). Section 2 states that $m \leq 2n - 1$ [30]: The equality holds when Rule 1 for the disk placement is maintained in an either round or linear arrangement. Each call of `FindHull` takes $O(|D|)$ time for the search of apex disk and the pivoting operation in the disk set D.

A theoretical worst case of the `QuickhullDisk` algorithm may occur with a highly unbalanced disk set (Type I worst case). Consider Fig. 13 where the set D of n disks touches the circular container (centered at the origin) from inside. The contacts between each disk and the container occur at the angles 0 (with d_1), π (with d_2), $3\pi/2$ (with d_3), $7\pi/4$ (with d_4),

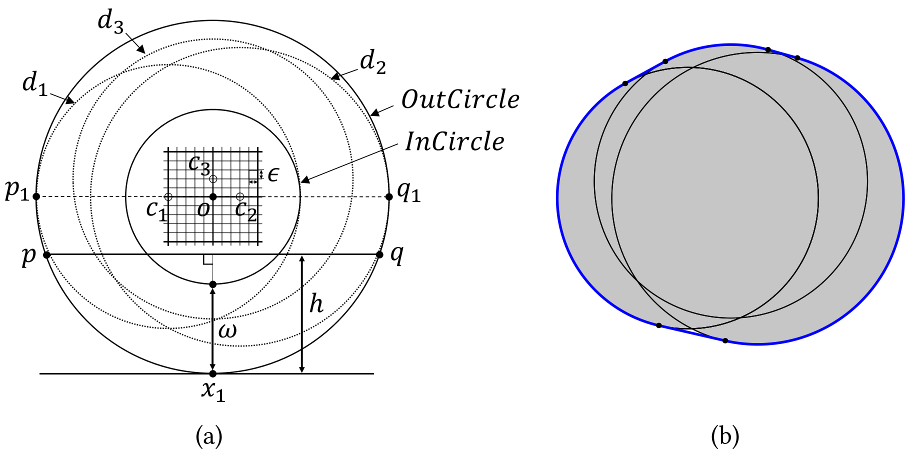


Fig. 14. Type II worst case of QuickhullDisk. The boundaries of the disks in D which are placed within an annulus, centered at the coordinate origin O, with the outer circle *OutCircle* and inner circle *InCircle* which define an width ω (In the figure, D has three disks). Assume *OutCircle* has a unit radius. Disk d_i is centered at c_i. Any choice of p and q produces both $D_F = D$ and $D_B = D$ if $h \geq \omega$. If $\omega = 10^{-6}$, there can be at most 274 distinct single precision disks within the annulus. (a) Schematic diagram. (b) Convex hull constructed by QuickhullDisk. $OutCircle(c(x=0, y=0), r=250)$, $InCircle((0, 0), 125)$, $d_1((-62.9, 0.5), 189.5)$, $d_2((38.5, 0.0)), 211.7)$, and $d_3((-0.4, 24.8), 198.3)$.

.... Note that d_3 is the apex disk w.r.t. d_1 and d_2; d_4 is the apex disk w.r.t. d_1 and d_3; This process continues for d_5, d_6, and so on. Observe that this is an angle-bisection process. Starting with $\Delta(p, q)$, d_1, and d_2, the apex disk is d_3. Recall that there are two recursive calls of FindHull after finding a triangle apex: One for the front edge and the other for the back edge of each triangle filter. Pivoting D with the pivot disk d_3 produces $D_F = \{d_2, d_3\}$ and $D_B = \{d_3, d_4, d_5, \dots, d_1\}$ where $|D_F| = 2$ and $|D_B| = n-1$ which requires accesses to n disks. The recursive call with D_F immediately stops the recursion but the one with D_B proceeds further recursions. In FindHull with D_B, we find the next apex disk d_4 and pivot D_B into two subsets, say $D_{B,F}$ and $D_{B,B}$ where $|D_{B,F}| = 2$ and $|D_{B,B}| = n-2$, which requires access to $n-1$ disks. This process may continue to d_5, d_6 and so on and takes $n-2$, $n-3$ accesses to disks, respectively. Hence, QuickhullDisk can show $O(n^2)$ time complexity behavior in the worst case.

A careful observation immediately reveals that, in the worst case, FindHull is called only $2(m-1)$ times and each FindHull takes $O(n)$ times for m extreme disks on the convex hull boundary. Hence, the time complexity of QuickhullDisk can be more carefully stated as $O(mn)$ in the worst case.

We observe that the minimum distance between two contact points quickly approaches to the single precision machine epsilon, which is approximately $1.2 * 10^{-7}$. When the container has a unit radius, the distance between the extreme points on ∂d_{27} and ∂d_1 is less than the machine epsilon and that between those on ∂d_{24} and ∂d_1 is less than 10^{-6} which is frequently used as a tight error bound in numerical computation. The sizes of disks do not matter as far as the disks are contained by the container. The distance between two contact points of d_n and d_2 is $\sqrt{2(1-\cos(\theta_n))}$ where $\theta_n = \pi/2^{n-2}$.

The worst case scenario is, we believe, highly unlikely to encounter in reality. If this type of worst case indeed occurs, the size of D cannot be large because this strategy of disk placement quickly reaches to machine epsilon. Hence, in this case, the quadratic time complexity does not have a practical significance. Recall that the theory of time complexity is asymptotic.

Type II worst case is that the disk set never shrinks during recursion. An example disk arrangement is shown in Fig. 14 in that the boundaries of the disks are placed within an annulus, centered at the coordinate origin O, with the outer circle *OutCircle* and inner circle *InCircle* which define an width ω. In this case, all disks have a common intersection, every disk is extreme, and each disk defines a linear hull edge with its counterclockwise neighbor. Assume *OutCircle* has a unit radius. Suppose that the disk d_i is centered at c_i, $i = 1, 2, \dots, n$. There can be at most 274 distinct disks (represented in single precision) within the annulus with $\omega = 10^{-6}$.[3] Their centers are located at grids within the annulus and radii are constrained by the boundary circles of the annulus. Fig. 14 shows only three of them centered at c_1, c_2, and c_3. Fig. 1(c) shows a case with ten disks.

QuickhullDisk begins the computation with both $p_1 \in OutCircle$ and $q_1 \in OutCircle$ on the horizontal axis in the worst case: i.e., $dist(p_1, q_1) = 2$. Let x_1 be the triangle apex w.r.t. $\Delta(p_1, q_1)$. Then, $x_1 \in OutCircle$ is on the vertical axis in the worst case. Note that $D_F = D_B = D$ for the triangle filter $Tri(p_1, x_1, q_1)$. Then, each recursion with both D_F and D_B behaves similarly in that each node in the recursion tree is associated with a disk set identical to D. In other words, the disk set never shrinks. However, the height of the recursion tree is strictly limited. Let h_n be the height from $\Delta(p_n, q_n)$ to $x_n \in OutCircle$.

[3] Suppose that the annulus is centered at the origin with the inner circle radius r and the annulus width ω $(= 10^{-6})$. Suppose the machine epsilon is 10^{-7}. Let $d((x, y), r+\delta)$ be a disk where $\delta \geq 0$. Then, ∂d is placed within the annulus if $\sqrt{x^2+y^2} \leq \delta \leq \omega - \sqrt{x^2+y^2}$ where $-\omega \leq x \leq \omega$, $-\omega \leq y \leq \omega$, and $0 \leq \delta < \omega$. There are at most 274 d's satisfying the constraints because x, y, and δ can have values in the unit of machine epsilon 0.1ω.

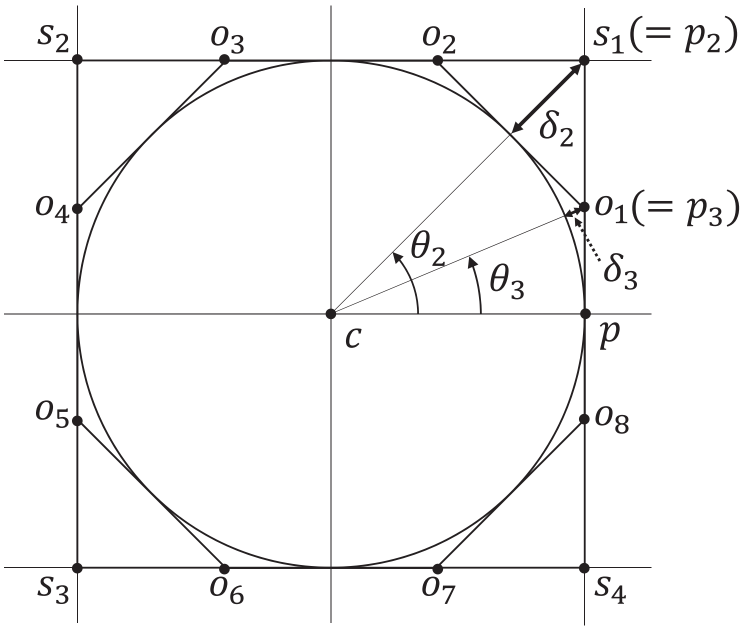


Fig. 15. Unit circle with the unit radius with its bounding polygons: a square S and an octagon O. Let Ξ_n, $n \geq 2$, be a regular 2^n-gon bounding the unit circle which is centered at the coordinate center c. Let p_i be the vertex of Ξ_n with the smallest angle θ_n $(= \angle pcp_n = \pi/2^n)$ in the first quadrant. Let δ_n be the distance from p_n to the unit circle boundary, Then, $\delta_n = \sec(\theta_n) - 1$. Note that $\delta_{13} < 10^{-7}$. Hence, a regular 2^{13}-gon with 8192 vertices is a perfect unit circle with an error of the machine epsilon. Fig. 19 shows that the convex hull of this disk set (the 8,192 disks with zero radii and the unit disk in the center) can be constructed in less than 0.1 sec by both `QuickhullDisk` and the incremental algorithm.

Then, $h_n = 1 - \cos(\theta_n)$ where the angle $\theta_n = \angle p_n c x_n = \pi/2^n$: This is again an angle-bisection process. Note that $h_{12} < 10^{-6}$ (and $h_{13} < 10^{-7}$) leading to only 2^{12} leaf nodes at most. Hence, it takes $O((2^{13} - 1)n)$ time where $n \leq 274$ because each node can be associated with as many as 274 disks.

Average time complexity. Suppose that each disk is equally likely to be an extreme disk. This implies that each disk is equally likely to be selected as the pivot disk for $D \backslash \{d_p, d_q\}$ in each call of `FindHull` which again implies that the recursion tree is highly likely to be height-balanced. Recall that the leaf nodes of the recursion tree one-to-one correspond to linear hull edges. As there are at most $2(n-1)$ linear hull edges, the recursion tree has at most the same number of leaf nodes. Being a binary tree, the recursion tree thus has at most $4n-5$ nodes: $2n-3$ internal and $2n-2$ leaf nodes. Therefore, the recursion tree has an expected $O(\log n)$ height. In Section 2, we elaborated the two possible disk arrangements with the maximal linear hull edges: A round and a linear arrangement. Suppose D has a round arrangement of disks as shown in Fig. 15. All small disks placed around a single large one. As the small disks are equally likely to be extreme disks, $\Delta(p, q)$ divides D into two subsets D_1 and D_2 with probably equal sizes. Therefore, the recursion tree is height-balanced to prove an expected $O(\log n)$ height. Note that a single large disk belongs to both D_1 and D_2, and in fact it belongs to the disk sets in all recursions: None behaves similarly. The merge at each depth of the recursion tree has at most an expected $O(n)$ time complexity because a simple concatenation of two lists suffices the conquer process of the outputs of two recursion calls with divided subsets. The merge at the "depth of leaf nodes" takes $O(n)$ time whereas that at depth 1 takes $O(1)$ time (We use a bit abused expression "depth of leaf nodes" to avoid an unnecessarily detailed explanation). Therefore, `QuickhullDisk` has an expected $O(n\log n)$ time complexity. The proof for a linear arrangement is obviously similar. Section 8.4 illustrates this observation well. We emphasize that the running behavior of `QuickhullDisk` for random disk sets is slightly better than $O(n\log n)$ because a subset of input disk set is removed from the consideration in the next recursion. This observation implies that the effort required to conquer the outputs of two recursions is slightly lower than linear for random disks.

8. Experiments and discussions

We implemented the `QuickhullDisk` algorithm and compared it with the incremental algorithm in [31] using diverse data sets. In our implementation of the incremental algorithm, we stored extreme disks in an AVL-tree for the $O(n\log n)$ time worst case complexity. Both were implemented in Visual C++ on Microsoft Visual Studio Community 2017. Boost 1.68.0 (released in Aug. 2018) was used for the AVL-tree. Computational environment: Intel Core i7-7700 3.60 GHz; 16 GB RAM; Windows 10 Professional (64 bit). Only one core was used in the experiment. For efficiency consideration, recursions of `QuickhullDisk` in this paper were implemented as iterations.

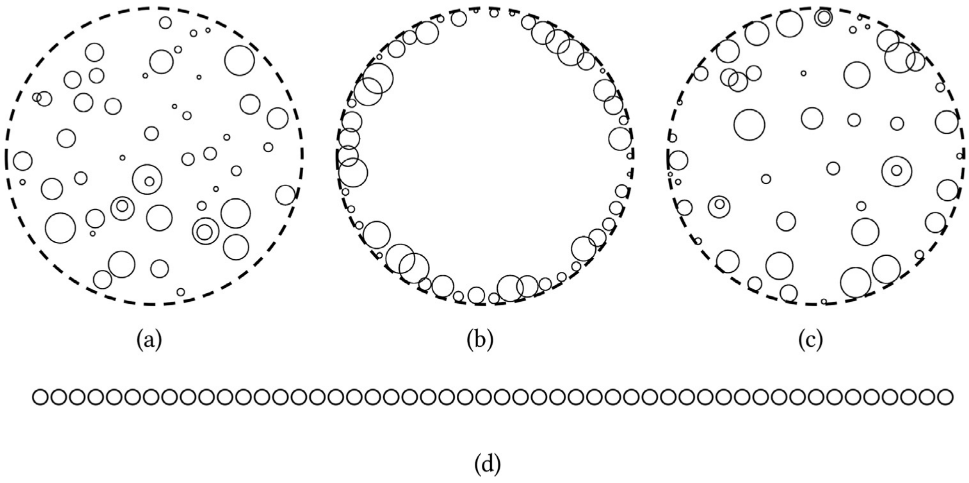


Fig. 16. Test set examples (50 disks for each). (a) RANDOM. (b) ON-BNDRY. (c) MIXED. (d) ON-A-LINE.

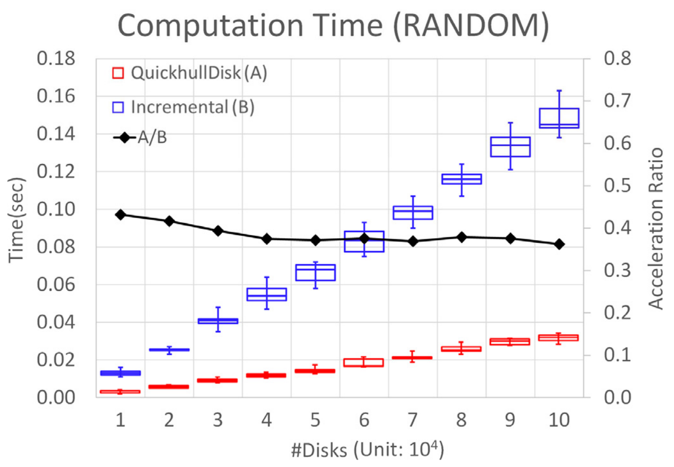


Fig. 17. Average computation time of `QuickhullDisk` and Incremental algorithms (Data set: RANDOM). #disks: 10 K, 20 K, ..., 100 K where each has 10 different model files ($K = 10^3$). Random radii in [1.0, 10.0]. $\rho \approx 0.1$. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

8.1. Test data set

We generated a benchmark data set which consists of four different categories of disk sets and publicized in Data in Brief [61]. The first is RANDOM $= \{D_{ij} | i, j = 1, 2, \ldots, 10\}$ where D_{ij} contains 10, 000*i random disks. Thus there are ten different files with the same number of disks. We placed the disks within a circular container centered at the origin O which has a sufficient radius so that the packing ratio ρ is maintained approximately 0.1 unless otherwise stated. Note that ρ is the ratio of the union of the area of individual disks to the area of container. In RANDOM (and the other two test sets ON-BNDRY and MIXED), each disk $d(c, r)$ has a random radius $r \in [1.0, 10.0]$, two disks may intersect each other, and one may include another. In RANDOM, d is centered at a random location c inside the container. The second is ON-BNDRY $= \{D_i | i = 1, 2, \ldots, 10\}$ where D_i contains 10, 000*i disks touching a circular container from inside. The third is a hybrid of RANDOM and ON-BNDRY: MIXED $= \{D_{ij} | i, j = 1, 2, \ldots, 10\}$ where D_{ij} contains 1000*i*j disks (out of 10, 000*j disks) touching a circular container from inside. The remaining $1000 * (10 - i) * j$ disks not touching the container are randomly positioned inside the container. Hence, a (boundary) *touching ratio* τ is well defined as the ratio of the number of extreme disks to the total number of the disks. The fourth is a set of congruent disks centered on a line: ON-A-LINE $= \{D_i | i = 1, 2, \ldots, 20\}$ where D_i contains 1000*i disks on the linear grid from left to right. Each disk $d \in D_i$ has a radius 1.0 and the distance between the boundaries of two neighbor disks is kept 0.5. Fig. 16 shows examples of these four types of disk sets with reduced sizes.

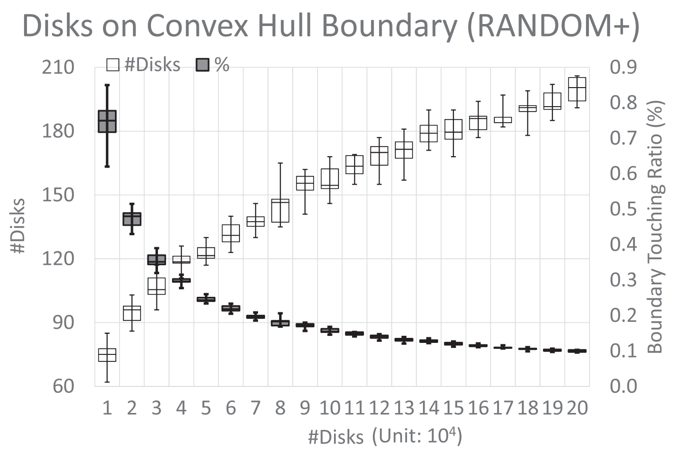


Fig. 18. Number of disks touching convex hull boundary in random disks (Data set: RANDOM+) and the touching ratio τ. #disks: 10K, 20K, ..., 200 K where each has 10 different model files ($K = 10^3$). Random radii in [1.0, 10.0]. $\rho \approx 0.1$.

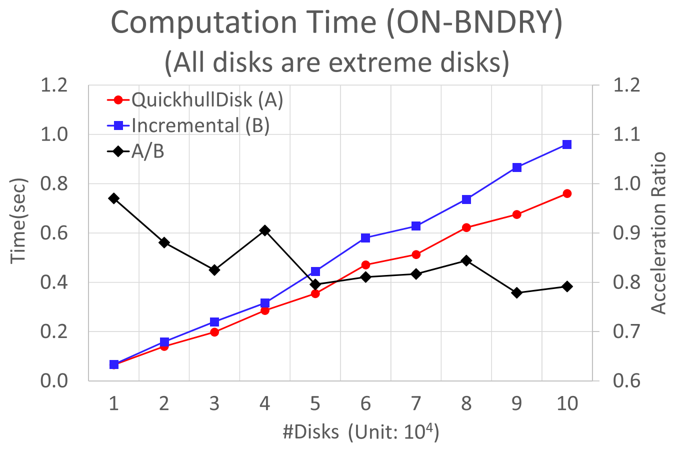


Fig. 19. Computation time of `QuickhullDisk` and Incremental algorithms (Data set: ON-BNDRY). #disks: 10 K, 20 K, ..., 100 K ($K = 10^3$). Random radii in [1.0, 10.0]. $\rho \approx 0.1$. Ratio of #disks touching convex hull boundary over #disks: 1.0.

8.2. *Verification of solution quality*

We verified solution quality as follows. The convex hulls of the data sets in the ON-BNDRY and MIXED data sets are known: The disks touching the container are the extreme disks contributing to the convex hull boundary and their orders are determined. The solutions of the data sets in ON-A-LINE are also known. Hence, we checked the convex hulls of these data sets constructed by both `QuickhullDisk` and the incremental algorithm with the known solutions: all were good. For RANDOM data sets, we checked if both `QuickhullDisk` and the incremental algorithm produced the same results: all produced same results.

8.3. *Computational efficiency*

Fig. 17 shows the box plot of the computation time of the two algorithms using the RANDOM data set: the horizontal and the left vertical axes denote test data size and time, respectively. Each of the ten horizontal coordinate corresponds to ten experimental results. Note the strong linearity of both algorithms. The right vertical axis denotes the *acceleration ratio* λ of the red curve (`QuickhullDisk`) to the blue curve (the incremental algorithm). `QuickhullDisk` takes about 39% on average (with the standard deviation(s.d.): 6%) of the computation time taken by the incremental algorithm. In other words, `QuickhullDisk` is approximately 2.6 times faster than the incremental algorithm for random disks. Note that `QuickhullDisk` has more speed-up as the size of disk set increases.

Fig. 18 shows the number of extreme disks (i.e., the disks touching the convex hull boundary) in RANDOM+, an extension of RANDOM, where RANDOM+ $= \{D_{ij} | i = 1, 2, \dots, 20; j = 1, 2, \dots, 10\}$ where D_{ij} contains 10, 000*i random disks following the disk generation rule of RANDOM. The horizontal axis denotes data sizes and the left vertical axis denotes the number of extreme disks found on the boundaries of the convex hulls (Shown by the white boxes). We observe that the number

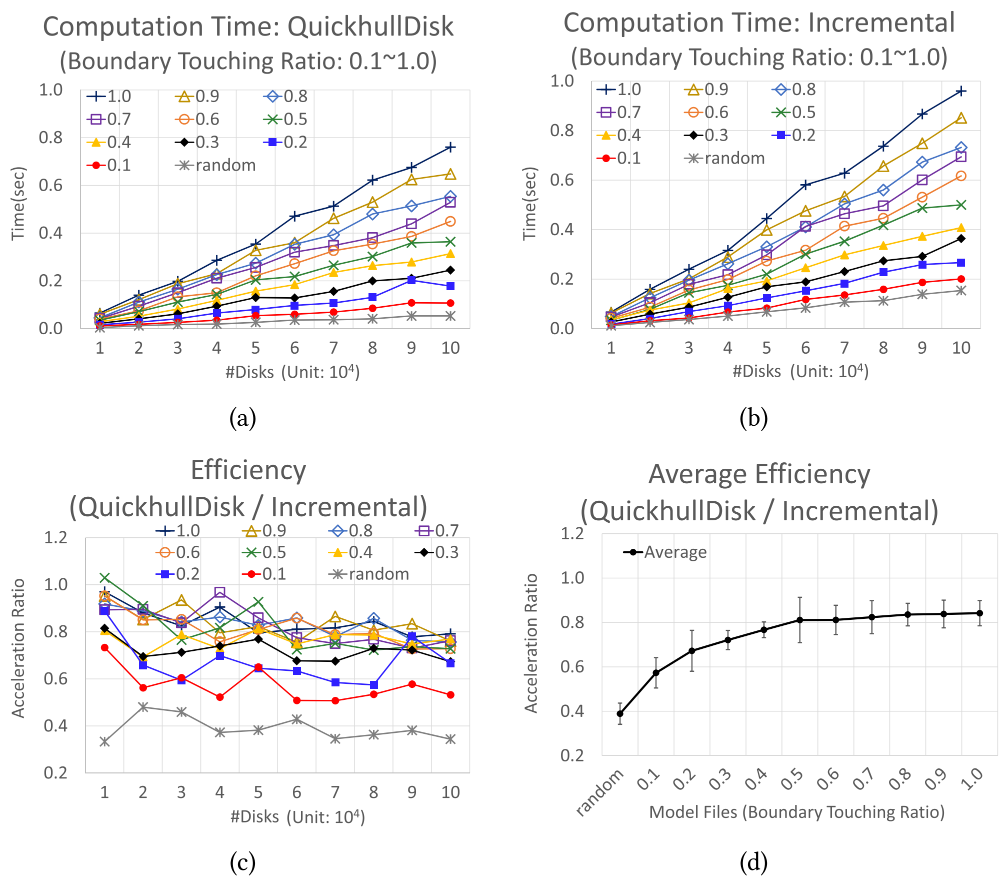


Fig. 20. Comparison of the computation time between `QuickhullDisk` and the incremental algorithm (Data set: MIXED). Touching ratio τ (#extreme disks/#disks): 0.1, 0.2, ..., 1.0. (a) `QuickhullDisk`. (b) The incremental algorithm. (c) and (d) The acceleration ratio (i.e., The ratio of `QuickhullDisk` to the incremental algorithm). ((c) w.r.t. the number of disks. (d) w.r.t. τ.). (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

of extreme disks increases with the number of input disks but increases much slower than linear. The right vertical axis denotes the touching ratio τ (Shown by the shaded boxes). Observe the curve shape follows an exponential decay function with a shrinking deviation. The convergences of the τ curve and its deviation are of interest.

Fig. 19 shows the comparison of computation time using ON-BNDRY. The strong linearity is maintained in both the `QuickhullDisk` and incremental algorithms. `QuickhullDisk` takes about 84% (s.d.: 6%) of the computation time of the incremental algorithm. Note that incremental algorithm is particularly well-tuned for the cases with a high ratio of extreme disks. We emphasize that `QuickhullDisk` is still significantly (approximately 1.2 times) faster than the incremental algorithm. Note that `QuickhullDisk` has a higher speed-up as the size of disk set increases.

We wanted to observe the computational requirement behavior of the two algorithms while we strictly maintain the touching ratio τ of extreme disks. Fig. 20(a) shows the computation time of `QuickhullDisk` using the MIXED data set. The horizontal and the vertical axes denote data size and computation time, respectively. There are eleven curves with different colors where each corresponds to a boundary touching ratio τ. For example, the rightmost red circle in the red curve corresponds to the case of 10,000 extreme disks out of 100,000 disks.

We make the following two observations from Fig. 20(a): (i) A strong linear increase for all cases, and (ii) the increased computation time with the increase of the touching ratio τ. We consider these two observations are critical evidences asserting the $O(mn)$ worst case time complexity of `QuickhullDisk`. Fig. 20(b) is for the incremental algorithm: We observe similar phenomena as `QuickhullDisk`.

Fig. 20 (c) and (d) show the acceleration ratio (i.e., the ratio of `QuickhullDisk` to the incremental algorithm): (c) w.r.t. the number of disks, and (d) w.r.t. the touching ratio τ. Fig. 20(c) shows the acceleration ratio of `QuickhullDisk` to the incremental algorithm w.r.t. the size of input disk sets: `QuickhullDisk` has higher speed-ups as the size of disk set increases. Fig. 20(d) shows the (average) acceleration ratio w.r.t. the touching ratio τ: The lower τ is, the better the speed-up is. When all disks are extreme ones (i.e., $\tau = 1.0$), `QuickhullDisk` takes 82% of the time that the incremental algorithm takes. For random data (which has $\tau \approx 0.0031(s.d. 0.0018)$, `QuickhullDisk` takes only 39%.

We summarize the experimental result as follows.

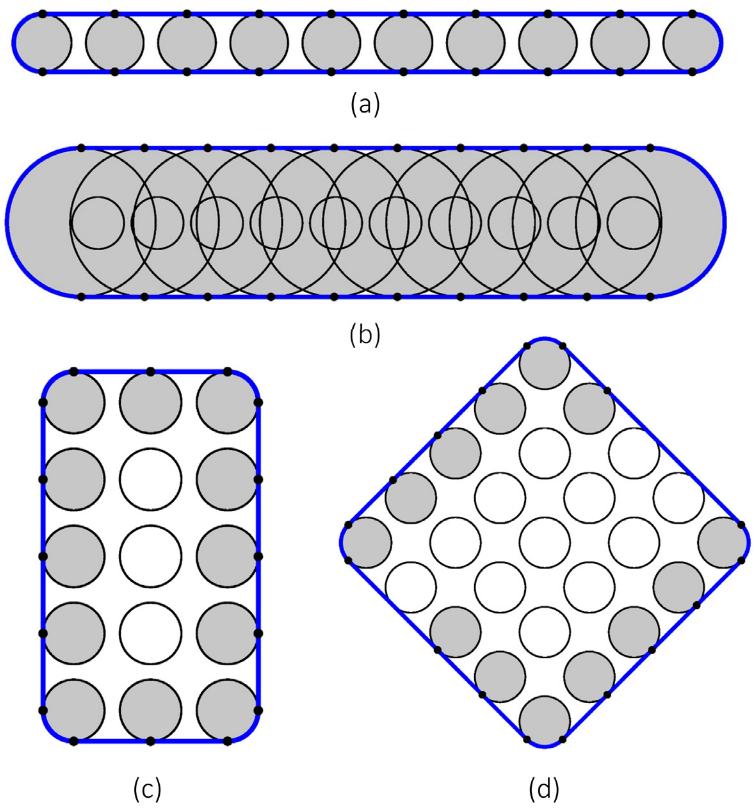


Fig. 21. Examples of congruent disks placed on grids and their convex hulls constructed using QuickhullDisk. The black dots are the hull vertices on the convex hulls. (a) 10 = 1*10 disks. (b) 20 = 1*10*2 disks. (c) 15 = 5*3 disks. (d) 25 = 5*5 disks. (a), (b), and (c): Disks placed on axis-parallel grids. (d): Disks are 45 degrees rotated from the axis-parallel grids. Some hull disks are not correctly recognized.

Observation 1. QuickhullDisk is faster than the incremental algorithm for most cases we tested.

- QuickhullDisk is faster when the size of input disk set gets larger.
- QuickhullDisk is significantly faster when the boundary touching ratio τ gets lower.

8.4. Degeneracy

Engineering designs frequently contain patterns consisting of disks in degenerate arrangements. Fig. 21 shows examples of the arrangements of congruent disks placed on grids and their convex hulls constructed by QuickhullDisk. The black dots on the convex hull boundaries denote hull vertices.

Fig. 21 (a) shows $n = 10$ disks with two common tangent lines: the upper one and the lower one. Suppose that the disks are indexed as d_1, d_2, ..., d_n from left to right. Then, p and q are defined on d_1 and d_n, respectively. Note that each of the n disks has a triangle apex of the same height from $\Delta(p, q)$. Suppose that we choose d_2, the one closest to d_1, as the pivot disk to divide D into D_1 and D_2 with $|D_1| = 2$ and $|D_2| = n - 1$. Then, QuickhullDisk recurs with D_2, i.e., the set with only one less disk than before, and encounters a similar situation. Hence, if the pivot disk is sequentially selected in this way, the height of the recursion tree is $O(n)$ where the computational cost at each level in the tree requires $O(n)$. Therefore, QuickhullDisk shows an $O(n^2)$ worst case time complxity.

The data set ON-A-LINE is prepared for testing QuickhullDisk with such a degeneracy. In Fig. 22, the black curve shows an $O(n^2)$ time behavior due to the sequential choice of pivot disks. The red curve corresponds to the randomization in the choice of the pivot disks: The curve shows an $O(n\log n)$ time behavior as Theorem 1 states. If we deterministically divide the set D of cotangent disks into D_1 and D_2 with the size difference at most one, we could instead get the worst case time complexity of $O(n\log n)$.

A remark on the incremental algorithm with the ON-A-LINE data set. It is necessary for the incremental algorithm to take care of the special case that an incrementing disk touches two edges on the boundary of the current convex hull, which is an ordinary case for QuickhullDisk.

Fig. 21 (b) and (c) show different disk arrangements on grids. The QuickhullDisk program correctly recognizes the hull vertices of all hull disks as the disks are placed at axis-parallel grids. Fig. 21(d) shows a 25 disks arranged on 5 by 5

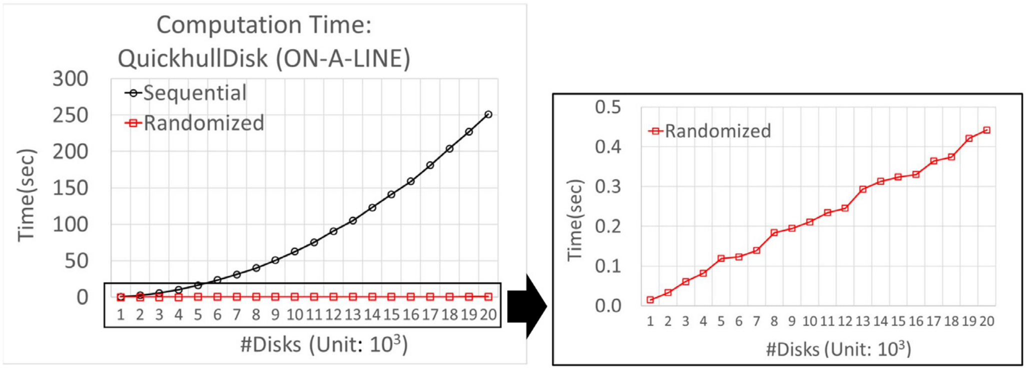


Fig. 22. Worst case behavior of `QuickhullDisk` and the effect of randomization (Data set: ON-A-LINE up to 20,000 disks). Disk radii: 1.0. Distance between the boundaries of two neighbor disks: 0.5. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

grids followed by a 45° rotation. Observe that `QuickhullDisk` misses some hull disks because of the numerical error due to the rotation operation.

9. Conclusion

In this paper, we propose the simple and fast `QuickhullDisk` algorithm for constructing the convex hull of a finite set of disks in the plane. `QuickhullDisk` takes $O(n\log n)$ time on average and $O(mn)$ time in the worst case where m represents the number of extreme disks which contribute to the boundary of the convex hull of n disks. These time complexities are identical to those of the quickhull algorithm for points in $\mathbb{R}^2$. The proposed `QuickhullDisk` algorithm was implemented, tested, and compared to the well-known $O(n\log n)$ time incremental algorithm. Experimental results show that the proposed `QuickhullDisk` algorithm runs significantly faster than the well-known $O(n\log n)$ time incremental algorithm, proposed by Devillers and Golin in 1995, for most data sets we tested, particularly for big data. `QuickhullDisk` is approximately 2.6 times faster than the incremental algorithm for all types of random disks. `QuickhullDisk` is approximately 1.2 times faster even for the disk sets where all disks are extreme, for which cases the incremental algorithm is particularly well-tuned. Applying a similar idea for the construction of the convex hull of three-dimensional spheres might be a challenging research topic in near future from both theoretical and practical point of view.

Acknowledgment

Authors thank Professor Hoang Xuan Phu for his insightful comments on the paper. Song, Ryu, and Kim were supported by the National Research Foundation of Korea (NRF/MSIT, 2017R1A3B1023591, 2016K1A4A3914691). An and Hoang are funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 101.01-2017.321.

References

[1] F.P. Preparata, M.I. Shamos, Computational Geometry: An Introduction, Springer-Verlag, 1985.
[2] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, Computational Geometry: Algorithms and Applications, 2nd ed., Springer, Berlin, 2000.
[3] C.D. Tóth, J. O'Rourke, J.E. Goodman, Handbook of Discrete and Computational Geometry, Discrete mathematics and its applications, third ed., Chapman & Hall/CRC, 2017.
[4] R.L. Graham, An efficient algorithm for determining the convex hull of a finite planar set, Inf. Process. Lett. 1 (1972) 132–133.
[5] D. Chand, S.S. Kapur, An algorithm for convex polytopes, J. ACM 17 (1) (1970) 78–86.
[6] R.A. Jarvis, On the identification of the convex hull of a finite set of points in the plane, Inf. Process. Lett. 2 (1) (1973) 18–21.
[7] F.P. Preparata, S.J. Hong, Convex Hulls of Finite Planar and Spatial Sets of Points, Technical Report 1-21, Coordinated Science Laboratory, University of Illinois, Urbana-Champaign Urbana, Illinois, 1975. 61801.
[8] F.P. Preparata, S.J. Hong, Convex hulls of finite sets of points in two and three dimensions, Commun. ACM 20 (2) (1977) 87–93.
[9] W.F. Eddy, A new convex hull algorithm for planar sets, ACM Trans. Math. Softw. (TOMS) 3 (4) (1977) 398–403.
[10] A. Bykat, Convex hull of a finite set of points in two dimensions, Inf. Process. Lett. 7 (8) (1978) 96–98.
[11] D.M. Mount, Computational geometry, 2002, (Lecture notes).
[12] J. O'Rourke, Computational Geometry in C, 2nd ed., Cambridge University Press, 1998.
[13] C. Barber, D.P. Dobkin, H. Huhdanpaa, The quickhull algorithm for convex hulls, ACM Trans. Math. Softw. 22 (4) (1996) 469–483.
[14] R. Seidel, A convex hull algorithm optimal for point sets in even dimensions, University of British Columbia, 1981 Master thesis.
[15] M. Kallay, The complexity of incremental convex hull algorithms in $\mathbb{R}^d$, Inf. Process. Lett. 19 (4) (1984) 197.
[16] D.G. Kirkpatrick, R. Seidel, The ultimate planar convex hull algorithm, SIAM J. Comput. 15 (1) (1986) 287–299.
[17] T.M. Chan, Optimal output-sensitive convex hull algorithms in two and three dimensions, Discrete Comput. Geom. 16 (6) (1996) 361–368.
[18] Z.-B. Xu, J.-S. Zhang, Y.-W. Leung, An approximate algorithm for computing multidimensional convex hulls, Appl. Math. Comput. 94 (2-3) (1998) 193–226.
[19] X. Zhang, Z. Tang, J. Yu, M. Guo, L. Jiang, Convex hull properties and algorithms, Appl. Math. Comput. 216 (11) (2010) 3209–3218.

[20] P.T. An, Method of orienting curves for determining the convex hull of a finite set of points in the plane, Optimization 59 (2) (2010) 175–179.
[21] P.T. An, L.H. Trang, An efficient convex hull algorithm for finite point sets in 3d based on the method of orienting curves, Optimization 62 (7) (2013) 975–988.
[22] M. Gao, T.-T. Cao, A. Nanjappa, T.-S. Tan, Z. Huang, ghull: a gpu algorithm for 3d convex hull, ACM Trans. Math. Softw.(TOMS) 40 (1) (2013).
[23] M.J. Atallah, Some dynamic computational geometry problems, Comput. Math. Appl. 11 (12) (1985) 1171–1181.
[24] F. Aurenhammer, B. Jüttler, On computing the convex hull of *piecewise* curved objects, Math. Comput. Sci. 6 (3) (2012) 261–266.
[25] M. Löffler, M.V. Kreveld, Largest and smallest convex hulls for imprecise points, Algorithmica 56 (2010) 235–269.
[26] F.B. Atalay, S.A. Friedler, D. Xu, Convex hull for probabilistic points, in: Proceedings of the Conference on Graphics, Patterns and Images [SIBGRAPI 16], 2016, pp. 1–14.
[27] B. Kalantari, A characterization theorem and an algorithm for a convex hull problem, Ann. Oper. Res. 226 (1) (2015) 301–349.
[28] P. Awasthi, B. Kalantari, Y. Zhang, Robust vertex enumeration for convex hulls in high dimensions, in: Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS), 84, 2018.
[29] B. Kalantari, An algorithmic separating hyperplane theorem and its applications, Discrete Appl. Math. 256 (2019) 59–82.
[30] D. Rappaport, A convex hull algorithm for disks and application, Comput. Geom. 1 (3) (1992) 171–187.
[31] O. Devillers, M.J. Golin, Incremental algorithms for finding the convex hulls of circles and the lower envelopes of parabolas, Inf. Process. Lett. 56 (3) (1995) 157–164.
[32] J.-D. Boissonnat, A. Cérézo, O. Devillers, J. Duquesne, M. Yvinec, An algorithm for constructing the convex hull of a set of spheres in dimension *d*, Comput. Geom. Theory Appl. 6 (1996) 123–130.
[33] W. Chen, K. Wada, K. Kawaguchi, D.Z. Chen, Finding the convex hull of discs in parallel, Int. J. Comput. Geom. Appl. 8 (3) (1998) 305–319.
[34] Y. Yue, J.L. Murray, J.R. Corney, D.E.R. Clark, Convex hull of a planar set of straight and circular line segments, Eng. Comput. 16 (8) (1999) 858–875.
[35] L. Habert, M. Pocchiola, Computing the convex hull of disks using only their chirotope, in: Proceedings of the European Workshop on Computational Geometry [20TH EWCG], 2004.
[36] J.S. Greenfield, A Proof for a QuickHull Algorithm, Technical Report, College of Engineering and Computer Science, 1990.
[37] C.A.R. Hoare, Algorithm 63 and 64, Commun. ACM 4 (7) (1961) 321.
[38] C.A.R. Hoare, Quicksort, Comput. J. 5 (1) (1962) 10–16.
[39] R.S. Scowen, Algorithm 271: Quickersort, Commun. ACM 8 (11) (1965) 669–670.
[40] E.H. Friend, Sorting electronic computer systems, J. ACM 3 (3) (1956) 134–168.
[41] D.L. Shell, A high-speed sorting procedure, Commun. ACM 2 (7) (1959) 30–32.
[42] G. Toussaint, Solving geometric problems with the rotating calipers, in: Proceedings of the IEEE Melecon, 1983, pp. 1–8.
[43] N. Megiddo, On the ball spanned by balls, Discrete Comput. Geom. 4 (1989) 605–610.
[44] B. Grünbaum, On common transversals, Archiv der Mathematik 9 (5) (1958) 465–469.
[45] M. Atallah, C. Bajaj, Efficient algorithms for common transversals, Inf. Process. Lett. 25 (2) (1987) 87–91.
[46] D.-S. Kim, K. Yu, Y. Cho, D. Kim, C. Yap, Shortest paths for disc obstacles, in: A.L., et al. (Eds.), Proceedings of the International Conference on Computational Science and Its Applications ICCSA, Lecture notes in computer science, 3045, 2004, pp. 62–70.
[47] J. Basch, L.J. Guibas, J. Hershberger, Data structures for mobile data, J. Algorithms 31 (1) (1999) 1–28.
[48] J. Xue, Y. Li, R. Janardan, On the expected diameter, width, and complexity of a stochastic convex-hull, in: [WADS 2017] Workshop on Algorithms and Data Structures, in: Lecture Notes in Computer Science, 10389, 2017, pp. 581–592.
[49] J. Kallrath, M.M. Frey, Packing circles into perimeter-minimizing convex hulls, J. Glob. Optim. (2018) 1–37, doi:10.1007/s10898-018-0724-0.
[50] A. Okabe, B. Boots, K. Sugihara, S.N. Chiu, Spatial Tessellations: Concepts and Applications of Voronoi Diagrams, 2nd ed., John Wiley & Sons, Chichester, 1999.
[51] M. Lee, K. Sugihara, D.-S. Kim, Topology-oriented incremental algorithm for the robust construction of the Voronoi diagrams of disks, ACM Trans. Math. Softw. 43 (2) (2016) 14:1–14:23.
[52] D.-S. Kim, D. Kim, K. Sugihara, Voronoi diagram of a circle set from Voronoi diagram of a point set: I. topology, Comput. Aided Geom. Des. 18 (2001) 541–562.
[53] D.-S. Kim, D. Kim, K. Sugihara, Voronoi diagram of a circle set from Voronoi diagram of a point set: II. geometry, Comput. Aided Geom. Des. 18 (2001) 563–585.
[54] D. Kim, D.-S. Kim, K. Sugihara, Euclidean Voronoi diagram for circles in a circle, Int. J. Comput. Geom. Appl. 15 (2) (2005) 209–228.
[55] F. Aurenhammer, R. Klein, D.-T. Lee, Voronoi Diagrams and Delaunay Triangulations, World Scientific, 2013.
[56] D.-S. Kim, Y. Cho, D. Kim, Euclidean Voronoi diagram of 3D balls and its computation via tracing edges, Comput. Aided Des. 37 (13) (2005) 1412–1424.
[57] D. Kim, D.-S. Kim, Region-expansion for the Voronoi diagram of 3D spheres, Comput. Aided Des. 38 (5) (2006) 417–430.
[58] C. Song, J. Ryu, D.-S. Kim, The source code and benchmark dataset for `QuickhullDisk`, a quickhull-like algorithm for constructing the convex hull of 2D disks, Mendeley Data, v1, (2019) doi:10.17632/h7c6rhb4vr.1.
[59] D.-S. Kim, B. Lee, C.-H. Cho, K. Sugihara, Plane-sweep algorithm of O(nlogn) for the inclusion hierarchy among circles, Lecture Notes in Computer Science 3045 (2004) 53–61.
[60] L.F. Tóth, Research problem 13, Periodica Mathematica Hungarica 6 (1975) 197–199.
[61] C. Song, J. Ryu, D.-S. Kim, Benchmark dataset for the convex hull of 2D disks, Data Brief (2019) (Under review).