# Decremental Greedy Polygons and Polyhedra Without Sharp Angles

David Eppstein*


## Abstract

We show that the max-min-angle polygon in a planar point set can be found in time $O(n \log n)$ and a max-min-solid-angle convex polyhedron in a three-dimensional point set can be found in time $O(n^2)$. We also study the maxmin-angle polygonal curve in 3d, which we show to be NP-hard to find if repetitions are forbidden but can be found in near-cubic time if repeated vertices or line segments are allowed, by reducing the problem to finding a bottleneck cycle in a graph. We formalize a class of problems on which a decremental greedy algorithm can be guaranteed to find an optimal solution, generalizing our max-min-angle and bottleneck cycle algorithms, together with a known algorithm for graph degeneracy.


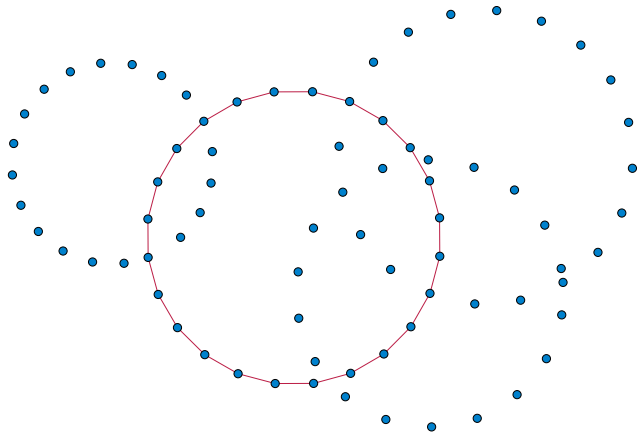

Figure 1: The maxmin-angle simple polygon in a given set of points

## 1 Introduction

In this work, we study the natural problem of finding a polygon with vertices drawn from $n$ given points, maximizing its minimum (sharpest) angle (Fig. 1). As we show, there exists an optimal polygon that is convex. To find it, we define the quality of a point, in a given subset, to be $2\pi$ if it is interior to the convex hull of the subset, or its interior angle if it is a vertex of the convex hull. This quality is monotonic: as we delete points, the quality of any remaining point can only decrease, as it becomes a hull vertex or as it loses hull neighbors. Therefore, it is safe to delete the point of minimum quality: any better polygon than the convex hull of the current subset cannot include the deleted point. A greedy algorithm that repeatedly deletes the sharpest hull vertex, and then returns the best polygon found throughout this deletion process, finds the maxmin-angle polygon in time $O(n \log n)$. After detailing this method we extend it to analogous problems of finding the max-min-solid angle convex polyhedron in 3d We reduce max-min-angle polygons in 3d to finding bottleneck cycles in graphs, to which we apply related decremental greedy algorithms.

The decremental greedy nature of our algorithms both for geometry problems (max-min angle polygons and polyhedra) and graph problems (bottleneck cycles) suggests that they share a common generalization. We formalize a class of bottleneck optimization problems that includes these problems and can be solved optimally by decremental greedy algorithms. Our formalization also encompasses the known problem of computing graph degeneracy, the minimum degree of a vertex in a subgraph chosen to maximize this degree. A classical linear-time algorithm for degeneracy [19] repeatedly removes a minimum-degree vertex until a given graph becomes empty; the degeneracy equals the maximum of the degrees of the vertices at the times of their removal. Generalizing convex hull angles and vertex degrees to other measures of element quality, we define the *bottleneck subset problem*, in which we seek a (nonempty) subset of a given set of elements whose worst element is as good as possible, according to a quality measure that can only worsen as other elements are removed. As we show, these problems can be solved by a decremental greedy algorithm that repeatedly removes the lowest-quality element.

Although greedy algorithms are commonly associated with matroids, our formalization does not apply to decremental greedy algorithms for a max-min matroid base such as a maximum spanning tree, nor to greedy algorithms such as Dijkstra's algorithm. Intuitively, this difference comes from the direction of monotonicity. In decremental greedy matroid algorithms, elements become more valuable as other elements are removed and they become needed to complete a base, and in Dijkstra's algorithm, vertices in the priority queue become more valuable as better paths are found to reach them. In contrast, in the algorithms we study, elements become

*Department of Computer Science, University of California, Irvine. Research supported in part by NSF grant CCF-2212129.

less valuable as other elements are removed. Our formalization is related to a different type of greedoid, an *antimatroid* [17]; see discussion following Theorem 1.

### 1.1 New results

Section 2 formalizes the bottleneck subset problem for monotonic quality measures, and describes the general decremental greedy algorithm for solving this problem. We provide in Section 3 the following applications to geometric optimization:

- We prove that a maxmin-angle simple polygon in a planar point set (Fig. 1) can be chosen to be a convex polygon, and we show how to find it in time $O(n \log n)$.
- We prove that a maxmin-solid angle polyhedral surface in a 3d point set can be chosen to be a convex polyhedron, and we show how to find it in time $O(n^2)$.
- For a 3d point set, the maxmin-angle closed polygonal curve (by which we mean a cyclic sequence of line segments meeting end to end) may intersect at points or even entire line segments, so we do not call it a polygon. Even if it is a polygon, it may use points interior to its convex hull, or may be knotted. We show how to find a maxmin-angle closed polygonal curve in time $O(n^3 \log^* n)$, or in time $O\big(n^3(\log n \log\log n)^2\big)$ if repeated points are allowed but repeated line segments are forbidden.

For space reasons we relegate additional results to the full version of this paper, including details on the antimatroid nature of the removals performed by the decremental greedy algorithm, on the NP-hardness of finding 3d maxmin-angle non-self-intersecting curves, and on algorithms for bottleneck cycles in graphs.

Those graph algorithms are used for our 3d polygonal curve algorithms, so for completeness we state them here. For undirected graphs the bottleneck cycle problem has an easy non-decremental linear time algorithm. For other types of graphs, it becomes more complicated:

- In a directed graph with $m$ edges, or a mixed graph with $m$ directed and undirected edges, we show how to find a bottleneck cycle in time $O(m \log^* m)$.
- We also consider *polar graphs* or *switch graphs*, in which each vertex has two poles at which edges attach, and a *regular cycle* must pass through both poles of each of its vertices (Fig. 2). We show how to find a bottleneck regular cycle in such a graph in time $O\big(m(\log m \log\log m)^2\big)$.

### 1.2 Related work

As far as we know the graph bottleneck cycle problems that we study are novel, but bottleneck path and bottleneck spanning tree problems were already studied by Pollack in 1960. The maximum spanning tree follows undirected bottleneck paths, and a variant of Dijkstra's algorithm constructs directed bottleneck paths [23]. Additional algorithms for these problems are known [10,14], and they have several applications [12,24,26].

Connecting given points into curves or surfaces has been studied with the goal of reconstructing an unknown shape from sparse samples [1,3,21], in some cases assuming that the curve is sampled densely enough to cause all angles to be close to $\pi$ [1]. The problems that we study have a similar flavor, but for curves through a subset of points rather than requiring a curve to pass through or near all points. We are unaware of previous work using the max-min-angle optimization criterion for curves and surfaces, but this criterion is well known in computational geometry in the context of Delaunay triangulations, which maximize the minimum angle among triangulations of given planar points [25], and has also been applied to other forms of triangulation [2,16,20]. Maximizing the minimum angle is also important in graph drawing, where the minimum angle at any vertex of a graph drawing is its *angular resolution* [13,18].

Dynamic programming can find convex polygons with vertices in a planar point set, optimizing a broad range of criteria [4,9,11]. However, we do not require the maxmin-angle polygon to be convex (instead this is an emergent property of the result) and our algorithm is simpler and faster than the known dynamic programming methods. It is not obvious how to generalize the dynamic programs for optimal convex polygons to curves or surfaces in higher dimensions.

## 2 Formalization

We define a *monotone bottleneck subset problem* to consist of a set $U$ of elements, and a function $q(x, S)$ that takes as input a pair $(x, S)$ where $x \in S$ and $S \subset U$, and produces as output a real number, the quality of $x$ as a member of $S$. We require $q$ to be monotone: whenever $x \in S \subset T \subset U$, we have $q(x, S) \le q(x, T)$. Intuitively, removing other elements from a subset containing $x$ causes the quality of $x$ to decrease or stay the same. We define the quality $Q(S)$ of a nonempty subset $S$ to be $Q(S) = \min_{x \in S} q(x, S)$, the least quality of a member of $S$, with $Q(\emptyset) = -\infty$, a flag value preventing the empty set from being optimal. Our goal is to find a nonempty subset whose quality is maximum, which we call the *bottleneck subset*. If $x$ is a lowest-quality element of a subset $S$ we call $x$ a *bottleneck element* of $S$.

The *decremental greedy algorithm* for a monotone bottleneck subset problem performs the following steps:

1. Initialize two sets $S$ and $T$ to both equal $U$.
2. While $S$ is not empty, repeat the following steps:
   - If $Q(S) > Q(T)$, set $T = S$.
   - Find any $x$ with $q(x, S) \leq Q(T)$.
   - Remove $x$ from $S$.
3. Return $T$.

Any bottleneck element $x$ of $S$ satisfies $q(x, S) \leq Q(T)$, so the algorithm always has an element that it can remove, although we do not require it to remove a bottleneck element at each step. When the bottleneck quality $\beta$ is already known, a simpler decremental algorithm can find the bottleneck subset: repeatedly remove any element of quality less than $\beta$ until all remaining elements have greater value. Call this the *known-$\beta$ algorithm*.

**Theorem 1.** *Every monotone bottleneck subset problem has a unique maximal bottleneck subset. Regardless of the choices they make at each step, the decremental greedy algorithm and known-$\beta$ algorithm both always find and return the maximal bottleneck subset.*

*Proof.* The union $M$ of all bottleneck subsets is a bottleneck subset: each $x \in M$ belongs to a bottleneck subset $X \subset M$, so $q(x, M) \geq q(x, X) \geq \beta$. Because this is true for all elements of $M$, $Q(M) \geq \beta$, the optimal value. $M$ is the unique maximal bottleneck subset, because it is a superset of all other bottleneck subsets.

Because each proper superset $V \supset M$ is not a bottleneck subset, some element $y \in V$ has $q(y, V) < \beta$, and any such $y$ cannot be in $M$ by monotonicity of its quality. Therefore, until $M$ is reached, the known-$\beta$ algorithm can and will remove an element of the complement of $M$. And until $M$ is reached, the decremental greedy algorithm will have $Q(T) < \beta$ and the element $x$ that it removes will have $q(x, S) < \beta$; again, $x \notin M$ by monotonicity of its quality. Therefore, until $M$ is reached, the decremental greedy algorithm can and will remove an element of the complement of $M$.

Once $M$ is reached, the known-$\beta$ algorithm terminates and returns it. The decremental greedy algorithm records $M$ as the set $T$ that it will eventually return; it can never subsequently change $T$ to another set because no other set has better quality. □

Both algorithms make arbitrary choices that can cause them to produce different removal sequences before reaching $U$. Their families of allowed removal sequences form *antimatroids*, structures that formalize the familiar introductory programming concept of a ready list. Antimatroids can be defined as families of sequences generated by a process that repeatedly appends an arbitrary "available" element from a given set, under the constraint that availability is determined by a monotonic function of the elements that have already been appended. Less formally, once an element becomes available, it remains available until it is appended itself, and availability depends only on the set of elements that have been chosen, not on their order. For the known-$\beta$ algorithm, the antimatroid property is straightforward (the property of having quality at most $\beta$ is determined by a monotonic function, as required) but for the decremental greedy algorithm, it requires a proof; see the full version of this paper.

According to the monotonicity that we require our quality measures to satisfy, each element gets worse (prioritized for earlier removal) as other elements around it are removed. This behavior should be contrasted with the quality of elements in max-min matroid problems such as the maximum spanning tree, which can also be solved by a decremental algorithm (remove the minimum-weight non-bridge edge until all remaining edges are bridges). In the decremental maximum spanning tree algorithm, an element (an edge) either keeps its priority (its weight) or gets a better priority (it becomes an unremovable bridge) as the algorithm progresses, rather than getting worse, so the decremental greedy maximum spanning tree algorithm does not fit into our framework.

Finding graph degeneracy is a bottleneck subset problem where $U$ is the set of vertices of a given graph $G$ and $q(x, S) = \deg_{G[S]} x$ is the degree of vertex $x$ in the subgraph induced by $S$. The bottleneck elements of any induced subgraph are its minimum-degree vertices. The bottleneck subset is a set of vertices that induces a subgraph maximizing its minimum degree. The linear-time degeneracy algorithm of Matula and Beck [19] repeatedly removes a minimum-degree vertex, as a special case of the decremental greedy algorithm for this quality. However, even for graph degeneracy, the antimatroid nature of the decremental greedy algorithm appears to be novel. To achieve this antimatroid property, we require a more general algorithm, allowing the removal of any vertex whose degree is at most the current quality bound, rather than the special case that only removes minimum-degree vertices.

## 3 Geometric applications

### 3.1 Polygons in 2d

**Lemma 2.** *Let $S$ be a finite set of points in the plane. Then there exists a convex polygon $P$ with vertices in $S$ that maximizes the minimum angle among all closed polygonal curves (allowing repeated vertices and edges) with vertices in $S$.*

*Proof.* Because $S$ determines finitely many angles, the max-min angle among closed polygonal curves exists. Let $W$ be any closed polygonal curve through $S$ attaining this angle, and let $P$ be the convex hull of $W$. Then,

compared to $W$, $P$ may omit some vertices and may increase the angle of the others that remain; both of these changes can only increase the sharpest angle in $P$, relative to the sharpest angle in $W$. Therefore, $P$ is also a max-min angle closed polygonal curves, as was stated to exist in the lemma. □

**Theorem 3.** *Let $S$ be a finite set of points in the plane. Then in time $O(n \log n)$ we can find a convex polygon $P$ with vertices in $S$ that maximizes the minimum angle among all closed polygonal curves in $S$.*

*Proof.* We perform the following steps:

1. Initialize a dynamic convex hull data structure to contain all points of $S$.
2. Initialize parameters $\theta$ to the sharpest angle of the hull, and $s$ to zero. These parameters will store the best angle found so far, and the number of removed points corresponding to that best angle.
3. Initialize an empty list $L$ of removed points.
4. While the current convex hull is non-degenerate (it has more than two vertices), repeat the following steps:
   - Find and remove from the dynamic hull the vertex with the sharpest angle (choosing arbitrarily any vertex of sharpest angle in case of ties) and append this vertex to $L$.
   - Set $\theta$ to the maximum of its previous value and the sharpest angle of the current hull, and if the result is an increase in $\theta$ then set $s$ to the current length of $L$.
5. Return the convex hull of the point set obtained from $S$ by removing the first $s$ points of $L$.

This is a decremental greedy algorithm where the quality of a point is its interior angle, if it is a convex hull vertex, or $2\pi$ otherwise. This quality is monotonic and the algorithm finds a max-min angle convex polygon by Theorem 1. This polygon is also a max-min angle closed polygonal curve by Theorem 2.

The sharpest angle in the current hull can be maintained during this algorithm using a priority queue of the current convex hull vertices and their angles, updated whenever the dynamic convex hull structure adds a vertex to the hull or changes the neighbor of an existing vertex. This structure requires $O(n)$ updates over the course of the algorithm and therefore takes $O(n \log n)$ time. Decremental or fully dynamic convex hull data structures that take $O(\log n)$ time per update are also known [5, 15], leading to an $O(n \log n)$ time bound for that part of the algorithm as well. □

### 3.2 Polyhedra in 3d

In this section we seek a polyhedral surface, with vertices at a subset of a given point sets, maximizing the solid angle interior to the surface as viewed from any point (or vertex) of the surface. To avoid definitional issues we consider only non-self-intersecting surfaces. Analogously to the results of the previous section, we have:

**Lemma 4.** *Among non-self-intersecting polyhedral surfaces having vertices at a subset of given points in $\mathbb{R}^3$, there exists a convex polyhedron that maximizes the minimum solid angle.*

*Proof.* As in Theorem 2, consider any polyhedral surface that maximizes the minimum solid angle, and take its convex hull. This can only remove vertices and improve the solid angle at the remaining vertices, so it must also maximize the minimum solid angle. □

As in the two-dimensional case, we will need a data structure for decremental convex hulls. Chan has studied this problem [7, 8], but his algorithms do not represent the hull explicitly, instead using an implicit representation that allows only extreme point queries. In our case, we need to find the solid angles of the vertices of the hull, not possible with Chan's structure. Another data structure, of Buchin and Mulzer [6] allows hulls of any subset of an input point set to be computed in randomized expected time $O\big(n(\log \log n)^2\big)$ on a word RAM. Instead, we use a simpler method for maintaining three-dimensional convex hulls explicitly, in time $O(n)$ per point deletion. Such a data structure was described by Overmars [22] (Theorem 6.4.1.6, p. 90), but to keep the presentation self-contained we outline a simplified version. The simplified data structure consists of a balanced binary search tree for the $x$-coordinate order of the given points, together with explicitly represented hulls of the sets of not-yet-deleted points in each subtree. After each deletion, each changed hull can be computed by merging two child hulls in time linear in its subtree. The total time for all merges adds in a geometric series to $O(n)$ per deletion. The space for this data structure is $O(n \log n)$. The version of Overmars improves the space to $O(n \log \log n)$ by storing only the hulls of large subsets of points, and makes the data structure fully dynamic rather than decremental using weight-balanced trees rather than static balanced trees, but we do not need those advances.

**Theorem 5.** *Let $S$ be a finite set of points in $\mathbb{R}^3$. Then in time $O(n^2)$ we can find a convex polyhedron $P$ with vertices in $S$ that maximizes the minimum solid angle among all non-self-intersecting polyhedral surfaces having vertices at a subset of $S$.*

*Proof.* We perform the following steps:

1. Initialize a dynamic convex hull data structure for the points of $S$.
2. Initialize parameters $\theta$ to the sharpest solid angle of the hull, and $s$ to zero. These parameters will store the best angle found so far, and the number of removed points corresponding to that best angle.
3. Initialize a list $L$ of removed points to an empty list.
4. While the current convex hull is non-degenerate (it has nonzero volume), repeat the following steps:
   - Find and remove the vertex with the sharpest solid angle (choosing arbitrarily any vertex of sharpest angle in case of ties) and append this vertex to $L$.
   - Set $\theta$ to the maximum of its previous value and the sharpest angle of the current hull, and if the result is an increase in $\theta$ then set $s$ to the current length of $L$.
   - Update the hull of the remaining points.
5. Return the convex hull of the point set obtained from $S$ by removing the first $s$ points of $L$.

Using the dynamic hull data structure outlined above, each iteration of the main loop takes time $O(n)$, and the whole algorithm takes time $O(n^2)$. □

### 3.3 Closed polygonal curves in 3d

To model the search for a closed polygonal curves through a given system of line segments in 3d, allowing repeated vertices but without allowing any line segment to be repeated, we use *polar graphs*. These are graphs in which the edges are undirected, but are attached to one of two *poles* of each vertex. A *regular cycle* in such a graph is a simple cycle for which the two edges incident to each vertex are attached to different poles: if the cycle enters a vertex via one pole, it must exit via the other pole. In the full version of this paper we describe a decremental greedy algorithm for finding a minmax-weight regular cycle in a polar graph with $m$ edges in time $O(m(\log m \log\log m)^2)$. For a given set $S$ of points in $\mathbb{R}^3$, we may define a weighted polar graph $G(S)$, as follows (Fig. 2):

- The vertices of $G(S)$ are the line segments determined by pairs of points in $S$.
- The two poles of each vertex of $G(S)$ are the two endpoints of the corresponding line segment.
- The edges of $G(S)$ connect pairs of line segments that form a three-point polygonal chain and are weighted by the angle at the middle point of the chain.

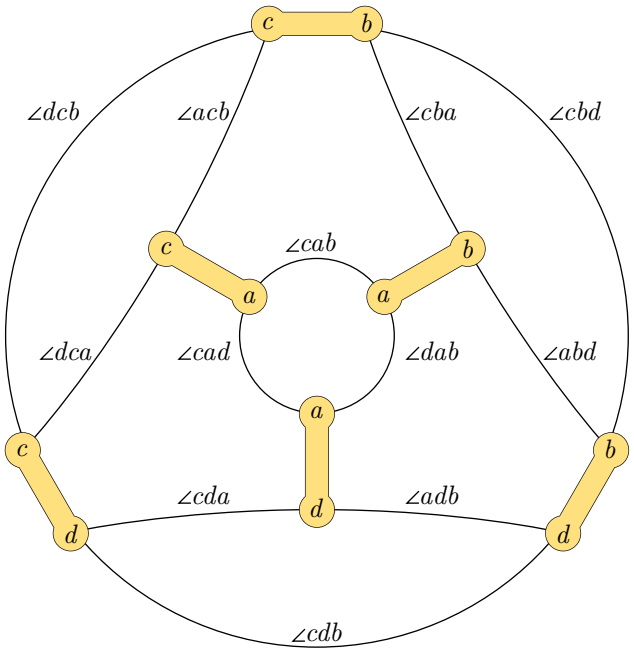


Figure 2: A polar graph with six vertices representing the line segments determined by pairs of four points $a$, $b$, $c$, and $d$, and 12 edges labeled by the angles determined by triples of points. The two poles of each vertex are labeled by the points. A regular cycle must enter and exit the vertices it visits via opposite poles.

- For each vertex and incident edge in $G(S)$, the pole of the vertex to which the edge is attached is the middle point of the three-point polygonal chain that defines the edge.

Then, a regular cycle in this graph corresponds to a closed polygonal curve in 3d, allowing repeated points but not allowing repeated line segments. The minimum edge weight in this cycle is the sharpest angle of the curve. If repeated line segments are to be allowed, we can instead replace the polar graph by an ordinary graph, its *double cover*. This has two copies of each vertex of the polar graph, and two directed edges for each edge of the polar graph, in each direction. Each copy of a vertex is incident to the incoming directed edges for one pole of the vertex and to the outgoing directed edges for the other pole. The resulting directed graph is *skew-symmetric* (the vertex bijection that swaps the two copies of each polar vertex also reverses all the directed edges) and has been used in algorithms for polar graphs. When a directed cycle in the double cover of $G(S)$ uses both copies of a vertex, it corresponds to a closed polygonal curve that uses a line segment twice, in both directions. In the full version of this paper we describe a decremental greedy algorithm for finding a bottleneck directed cycle in a directed graph in time $O(m \log^* m)$.

**Theorem 6.** *We consider maxmin-angle closed polygo-*

*nal curves in a 3d point set that may use points interior to its convex hull, may be knotted, and may self-intersect. If both repeated points and repeated line segments are allowed, we can find such a curve in time $O(n^3 \log^* n)$. If repeated points but not repeated line segments are allowed, we can find it in time $O\big(n^3(\log n \log\log n)^2\big)$.*

*Proof.* We construct $G(S)$, with $O(n^3)$ edges, and then find either a bottleneck directed cycle in the double cover of $G(S)$, or a bottleneck regular cycle in $G(S)$, according to whether we allow or disallow repeated segments, respectively. □

For completeness, we state:

**Theorem 7.** *It is* NP*-complete to find the maxmin-angle simple polygon for points in $\mathbb{R}^3$, for points on a unit sphere given by rational Euler angles.*

We defer the proof to the full version of this paper.

## 4 Conclusions

We have formalized a broad family of decremental greedy problems for monotone bottleneck subset problems, generalizing existing graph degeneracy algorithms, and shown its applicability in developing new algorithms in computational geometry and graphs. We have found simple greedy decremental algorithms for several maxmin-angle problems in geometric optimization: finding a planar polygon through given points maximizing the minimum angle, finding a 3d polyhedral surface through given pints maximizing the minimum solid angle, and finding a (self-intersecting) 3d closed polygonal curve through given points maximizing the minimum angle. On the other hand, constraining the 3d polygonal curve to be non-self-intersecting appears to make the problem computationally infeasible.

There appears to be room for improvement in our polyhedral surface and 3d curve time bounds. It would also be of interest to find other problems to which the same greedy decremental approach applies.

## References


[1] N. Amenta, M. W. Bern, and D. Eppstein. The crust and the $\beta$-skeleton: combinatorial curve reconstruction. *Graphical Models & Image Processing* 60/2(2):125–135, 1998, doi:10.1006/gmip.1998.0465.

[2] N. Amenta, M. W. Bern, and D. Eppstein. Optimal point placement for mesh smoothing. *J. Algorithms* 30(2):302–322, 1999, doi:10.1006/jagm.1998.0984.

[3] D. Attali. $r$-regular shape reconstruction from unorganized points. *Comput. Geom.* 10(4):239–247, 1998, doi:10.1016/S0925-7721(98)00013-3.

[4] D. Avis and D. Rappaport. Computing the largest empty convex subset of a set of points. *Proc. 1st Symp. on Computational Geometry (SOCG '85)*, pp. 161–167. ACM, 1985, doi:10.1145/323233.323255.

[5] G. S. Brodal and R. Jacob. Dynamic planar convex hull. *Proc. 43rd Symp. Foundations of Computer Science (FOCS 2002)*, pp. 617–626. IEEE Computer Society, 2002, doi:10.1109/SFCS.2002.1181985.

[6] K. Buchin and W. Mulzer. Delaunay triangulations in $O\big(\text{sort}(n)\big)$ time and more. *J. ACM* 58(2):1–27, 2011, doi:10.1145/1944345.1944347.

[7] T. M. Chan. A dynamic data structure for 3-D convex hulls and 2-D nearest neighbor queries. *J. ACM* 57(3):1–15, 2010, doi:10.1145/1706591.1706596.

[8] T. M. Chan and K. Tsakalidis. Optimal deterministic algorithms for 2-d and 3-d shallow cuttings. *Discrete Comput. Geom.* 56(4):866–881, 2016, doi:10.1007/s00454-016-9784-4.

[9] V. Chvátal and G. Klincsek. Finding largest convex subsets. *Congressus Numerantium* 29:453–460, 1980, MR608447.

[10] R. Duan and S. Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. *Proc. 20th ACM-SIAM Symp. on Discrete Algorithms (SODA '09)*, pp. 384–391, 2009, https://portal.acm.org/citation.cfm?id=1496813.

[11] D. Eppstein, M. Overmars, G. Rote, and G. J. Woeginger. Finding minimum area $k$-gons. *Discrete Comput. Geom.* 7(1):45–58, 1992, doi:10.1007/BF02187823, MR1134451.

[12] E. Fernández, R. Garfinkel, and R. Arbiol. Mosaicking of aerial photographic maps via seams defined by bottleneck shortest paths. *Operations Research* 46(3):293–304, 1998, doi:10.1287/opre.46.3.293.

[13] M. Formann, T. Hagerup, J. Haralambides, M. Kaufmann, F. T. Leighton, A. Symvonis, E. Welzl, and G. Woeginger. Drawing graphs in the plane with high resolution. *SIAM J. Comput.* 22(5):1035–1052, 1993, doi:10.1137/0222063, MR1237161.

[14] H. N. Gabow and R. E. Tarjan. Algorithms for two bottleneck optimization problems. *J. Algorithms* 9(3):411–417, 1988, doi:10.1016/0196-6774(88)90031-4, MR955149.

[15] J. Hershberger and S. Suri. Applications of a semi-dynamic convex hull algorithm. *BIT* 32(2):249–267, June 1992, doi:10.1007/bf01994880.

[16] B. Joe. Delaunay versus max-min solid angle triangulations for three-dimensional mesh generation. *Int. J. Numer. Methods Eng.* 31(5):987–997, 1991, doi:10.1002/nme.1620310511.

[17] B. Korte, L. Lovász, and R. Schrader. Chapter III: Abstract Convexity – Antimatroids. *Greedoids*, pp. 19–43. Springer-Verlag, 1991, doi:10.1007/978-3-642-58191-5_3.

[18] S. Malitz and A. Papakostas. On the angular resolution of planar graphs. *SIAM J. Discrete Math.* 7(2):172–183, 1994, doi:10.1137/S0895480193242931, MR1271989.

[19] D. W. Matula and L. L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM* 30(3):417–427, 1983, doi:10.1145/2402.322385, MR0709826.

[20] S. A. Mitchell. Approximating the maxmin-angle covering triangulation. *Comput. Geom.* 7(1–2):93–111, 1997, doi:10.1016/0925-7721(95)00046-1.

[21] S. Ohrhallinger, J. Peethambaran, A. D. Parakkat, T. K. Dey, and R. Muthuganapathy. 2D points curve reconstruction survey and benchmark. *Computer Graphics Forum* 40(2):611–632, 2021, doi:10.1111/cgf.142659.

[22] M. H. Overmars. *The Design of Dynamic Data Structures*. Lecture Notes in Computer Science 156. Springer, 1983, doi:10.1007/BFB0014927.

[23] M. Pollack. The maximum capacity through a network. *Operations Research]* 8(5):733–736, 1960, doi:10.1287/opre.8.5.733.

[24] M. Schulze. A new monotonic, clone-independent, reversal symmetric, and Condorcet-consistent single-winner election method. *Social Choice and Welfare* 36(2):267–303, 2011, doi:10.1007/s00355-010-0475-4.

[25] R. Sibson. Locally equiangular triangulations. *The Computer J.* 21(3):243–245, 1978, doi:10.1093/comjnl/21.3.243.

[26] E. Ullah, K. Lee, and S. Hassoun. An algorithm for identifying dominant-edge metabolic pathways. *IEEE/ACM International Conference on Computer-Aided Design (ICCAD 2009)*, pp. 144–150, 2009, https://ieeexplore.ieee.org/document/5361299.