On Polyhedra Induced by Point Sets in Space[*]

Pankaj K. Agarwal[†] Ferran Hurtado[‡] Godfried T. Toussaint[§] Joan Trias[¶]

Abstract

It is well known that one can always *polygonize* a set S of $n \geq 3$ points in the plane (not all on a line), i.e., construct a simple polygon P whose vertices are precisely the given points in S. For example, the shortest circuit through S is a simple polygon. In 1994 Branko Grünbaum showed that an analogous theorem holds in $\mathbb{R}^3$. More precisely, if S is a set of $n \geq 4$ points in $\mathbb{R}^3$ (not all of which are coplanar) then it is always possible to *polyhedronize* S, i,e., construct a simple (sphere-like) polyhedron P such that the vertices of P are precisely the given points in S. Grünbaum's constructive proof may yield *Schönhardt* polyhedra that cannot be triangulated. In this paper several alternative algorithms are proposed for constructing such polyhedra induced by a set of points, which may always be triangulated, and which enjoy several other useful properties as well. Such properties include polyhedra that are star-shaped, have Hamiltonian skeletons, and admit efficient point-location queries. We show that *polyhedronizations* with a variety of such useful properties can be computed efficiently in $O(n \log n)$ time. Furthermore, we show that a tetrahedralized, xy-monotonic, polyhedronization of S can be computed in time $O(n^{1+\varepsilon})$, for any $\varepsilon > 0$.

1 Introduction

In 1964 Hugo Steinhaus posed the following problem [24]: Consider a set S of $n \geq 3$ points in the plane such that no three of them are collinear. Is it always possible to find a closed polygon with n non-intersecting sides whose vertices are these n points? He proceeded to give a clever proof that this is true. His proof removes an extreme point p of S and, by induction, assumes that the remaining $n-1$ points admit such a polygon. Next, an edge e of this polygon that is completely visible from

[*]Work by P.A. is supported by NSF under grants CCR-00-86013, EIA-98-70724, EIA-01-31905, and CCR-02-04118, and by a grant from the U.S.–Israel Binational Science Foundation. Work by G.T. is supported by NSERC Grant no. OGP0009293 and FCAR Grant no. 93-ER-0291. Work by F.H. and J.T. is supported by grants MEC MTM2006-01267, DURSI 2005SGR00692, DURSI 2001SGR00224, and MCYT BFM2003-0368.

[†]Department of Computer Science, Box 90129, Duke University, Durham NC 27708-0129, USA; pankaj@cs.duke.edu.

[‡]Departament de Matemática Aplicada II, Universitat Politécnica de Catalunya (UPC), Jordi Girona, 1-3, 08034, Barcelona, Spain; ferran.hurtado@upc.edu

[§]School of Computer Science, 3480 University Street, McGill University, Montreal, Quebec H3A 2A7, Canada; godfried@cs.mcgill.ca

[¶]Departament de Matemática Aplicada II, Universitat Politécnica de Catalunya (UPC), Jordi Girona, 1-3, 08034, Barcelona, Spain. joan.trias@upc.edu

p is found, p is connected to the endpoints of e, and e is removed. A direct implementation of this proof yields an $O(n^3)$ time algorithm for constructing the required polygon.

Independently, in 1966 Michael Gemignani [8] posed the following similar problem: given $n \geq 3$ points in the plane, not all lying on the same line, are they the vertices of a simple closed polygonal chain and, if so, produce a witness, i.e., construct one. Note that the problem posed by Gemignani is more general than the version posed by Steinhaus, since Gemignani assumes only that not *all* the points are collinear, whereas Steinhaus assumes no *three* points are collinear. Indeed, the induction proof of Steinhaus does not hold if one only assumes that not all points are collinear. Gemignani also conjectured that the *shortest* closed route through the points must be one of these simple polygons. This conjecture had in fact been proved one year earlier by Quintas and Supnick [19]. Later it was also proved independently in [20], but finding such a shortest circuit is difficult. This is the well known *Euclidean Travelling Salesman Problem* and is NP-complete [14]. In a later paper Gemignani [7] gave a much simpler proof (than the one given in [8]), which yields a star-shaped polygon. Gemignani's proof yields directly an algorithm that runs in $O(n \log n)$ time. This bound is optimal since Shamos [22] proved an $\Omega(n \log n)$ lower bound on this problem. In 1994 Branko Grünbaum [12] gave an alternate simple proof of Gemignani's problem that yields a *monotonic* polygon. Furthermore, Grünbaum's proof can also be easily implemented in $O(n \log n)$ time.

Different types of polygonizations are of interest in a variety of disciplines where they serve different functions. Clearly for traveling-salesperson-type problems we are interested in polygonizations that have a short, if not minimum, length. In pattern recognition we are often interested in polygonizations that characterize, in a "nice" periosteal manner, the boundary of a shape [17], [28]. One may be interested in polygonizations as data structures that afford simple insertions and deletions of points from S [1]. In computational geometry it may also be the case that there exists a simple solution to a problem for polygons that may also be the solution to the problem in which the input is the set of vertices of the polygon, i.e., a set of points. If the right kind of polygonization can be found efficiently then a simple solution for polygons may yield a simple solution for point sets. A notable example here is Graham's convex-hull algorithm that applies the Graham scan to a *star-shaped* polygonization of the points [11].

The star polygonization in Graham's algorithm has the nice property that it is star-shaped from a known point in its kernel (namely the origin o). Therefore the polygon can be triangulated with a simple and practical linear-time algorithm [30]. This is an attractive property of a polygonization because a triangulated polygon is useful for the efficient computation of many geometric properties [27], [29], [13]. A triangulated polygon is also useful because the dual graph of the triangulation is a tree, and this tree can be easily used to guide efficient search in the polygon. For precisely the same reason, another attractive property of a polygon is the admissibility of a good *thin* triangulation [23]. A thin triangulation is one that minimizes the number of nodes of degree three in the dual tree and can be computed in $O(n^3)$ time using $O(n^2)$ space [23]. Clearly an even more attractive property of a polygon is that of admitting a triangulation whose dual is a chain. Such polygons are called *serpentine*. The disadvantage of Graham's star polygonization is that it may yield a polygon that is not serpentine, and therefore it may require $O(n^3)$ time and $O(n^2)$ space to compute a thin triangulation for it. However, a simple modification of the Graham polygonization not only has this serpentine property but a serpentine triangulation is generated during polygonization at no extra cost. Instead of picking, as the origin, a point in the interior of the convex hull, we select a point of S on the convex hull of S, such as the point with minimum y-coordinate. We call such a polygonization a *fan polygonization* of a point set.

Another desirable and useful property of a polygon is its *monotonicity*. Neither the star nor the fan polygonization methods are guaranteed to yield monotonic polygons. However a monotonic polygonization can be easily obtained in $O(n \log n)$ time as follows. It should be noted that in the 1970's this polygonization was used in several variations of Graham's convex-hull algorithm [3], [4], and in 1994 Grünbaum [12] offered the following proof which assumes only that not all points of S are collinear.

First rotate S so that no two points have the same x-coordinate. Next, find the points of S with minimum and maximum x-coordinates, say a and b, respectively. Construct a line L through a and b and determine which of the $n-2$ points lie above L (call these S_1) and which below (call these S_2). Since not all points of S lie on a line, at least one of S_1 and S_2 must be non-empty. Assume without loss of generality that it is S_1. Sort the points in $S_1 \cup \{a, b\}$ by x-coordinate and connect adjacent points by edges. If $S \setminus S_1 \neq \emptyset$, do the same for points in $S \setminus S_1$. Otherwise connect a to b.

While the resulting polygon is monotonic in the x-direction and a simple linear-time triangulation algorithm exists for monotone polygons [26], this procedure may yield monotonic polygons that are not serpentine and therefore computing a thin triangulation for them may still require $O(n^3)$ time and $O(n^2)$ space, as in the case of star polygonizations.

The planar polygonization problem can be generalized in at least two ways to 3-dimensional space. We can ask for a closed polygonal chain that is "simple" in the sense that it is not knotted. This is the 3D-polygonization problem. This problem can be solved using the planar polygonization procedures by suitably projecting the points of S onto a plane and then "lifting" the planar polygonization obtained back into space. In the more interesting generalization we can ask for a simple polyhedron the vertices of which are the given point set.[1] We call this problem the *polyhedronization* problem. Surprisingly this problem has received scant attention in this general setting. However the following special case is a well-known problem in solid modeling and has received much attention in medical applications concerning the reconstruction of solids [6]: we are given two simple polygons P and Q of n and m vertices, respectively, each on one of two parallel planes in space, and it is desired to find a simple polyhedron that has the two polygons as faces and whose vertices are precisely the vertices of the two polygons. Clearly, if the two given polygons are *convex*, this is always possible as it suffices to compute the convex hull of the union of the two polygons. Furthermore such a polyhedronization can be computed in $O(n + m)$ time by using the "rotating caliper" technique [25]. O'Rourke and Subramanian [18] have shown that such a polyhedronization is not always possible for arbitrary simple polygons. Finally, if a judiciously-placed "Steiner vertex" is permitted then such a polyhedronization always exists [9].

In this paper we study various methods for generating, in polynomial time, polyhedronizations that have a variety of desirable properties: monotonicity, star-shapedness, admitting a tetrahedralization (triangulation), possibly with nice dual structure, possessing a good 1-skeleton from the viewpoint of graph theory -a neat departure from the 2-dimensional case-, and affording fast point-location queries. It is worth emphasizing that optimizing the area or perimeter of a polygonization

[1]A polyhedron in $\mathbb{R}^3$ is *simple* if its interior is homeomorphic to an open ball in $\mathbb{R}^3$.

are problems known to be NP-hard. Now, if we take a point set S in $\mathbb{R}^3$ such that all its points but one (p) lie on a plane, the only way to polyhedronize S is to polygonize the points of S with p removed, and subsequently to join p to all the $n-1$ vertices of the polygonization. Therefore optimizing properties such as volume or total edge-length of polyhedronizations is also NP-hard. The 3D-polygonization problem is easily solved along the way in that a polyhedronization with the property that its 1-skeleton admits a Hamiltonian yields a 3D-polygonization if one of its Hamiltonians is reported.

The paper is organized as follows. Section 2 presents a new $O(n \log n)$-time polygonization method that combines the desirable properties of both the monotonic and fan polygonizations, i.e., it computes a polygonization that is: monotonic, serpentine, and triangulated in a serpentine manner at no extra cost. Section 3 presents two versions of an algorithm for computing a tetrahedralized, xy-monotonic, polyhedronization of a set of points in $\mathbb{R}^3$. The straight-forward implementation of this algorithm runs in time $O(n^2)$, but the running time can be improved to $O(n^{1+\varepsilon})$ by using an appropriate data structure [2]. We then present in Section 4 several algorithms for computing a star-shaped polyhedronization of a set of points in $\mathbb{R}^3$. Along the way, we discuss the properties of the 1-skeletons our methods yield. Finally we conclude by mentioning a few open problems in Section 5.

While some of our constructions are delicate and require fine technical tuning, others are simpler; we have preferred to include all of them, as this is to the best of our knowledge, the first systematic exposition of this topic.

2 Serpentine Monotonic Polygonizations

We present here a new method of polygonization in the plane that achieves many desirable properties, and that, surprisingly, does not extend to $\mathbb{R}^3$. The main idea is simple and consists of sorting all the points along some direction such as the x-axis, creating a triangle from the first three points, and subsequently processing one point at a time in the sorted list, creating a new triangle that is "glued" to a suitable visible edge of the existing polygonization. This algorithm may be viewed as a modification of the method of Steinhaus. Computational efficiency is obtained by presorting the points, which dispenses with the time-consuming search for a visible edge, in the construction of Steinhaus, and yields the desirable properties to boot. The algorithm is described more formally in Fig. 1. The correctness of Algorithm-1 is established by the following simple lemma.

Lemma 2.1 *At step i, for $i > j$, of Algorithm-1, i.e., when p_i is incorporated, at least one of the edges of Q_{i-1} adjacent to p_{i-1} is entirely visible from p_i. Assuming Q_{i-1} is monotonic and triangulated in a serpentine manner, Q_i also satisfies this property.*

Proof: Let $x(q)$ be the x-coordinate of q. We use the notation b_i and a_i for the points found before and after p_i, respectively, in a counterclockwise traversal of the boundary of Q_i. We consider two cases depending on whether $x(p_{i-1}) > x(p_{i-2})$ or $x(p_{i-1}) = x(p_{i-2})$.

Case 1. $x(p_{i-1}) > x(p_{i-2})$ (refer to Fig. 2 (a)). Because of the monotonicity of Q_{i-1} (which we are inductively assuming), the open region of points q with $x(a_{i-1}) < x(q) < x(p_{i-1})$ that are

```
Algorithm-1
INITIALIZATION

    (1) $\langle p_1, p_2, \ldots, p_n\rangle$: Input points sorted in lexicographic order
        of their coordinates;
    (2) $p_j$: First point in the list not lying on line $p_1p_2$;
    (3) $T_j = \triangle p_1p_{j-1}p_j$; $Q_j = T_j$;

ITERATION

for $i = j + 1$ to $n$ do
    $p_{i-1}p_k$: Edge of $T_{i-1}$ adjacent to $p_{i-1}$ visible from $p_i$;
    $T_i = \triangle p_kp_{i-1}p_i$, $Q_i = Q_{i-1} \cup T_i$;
    end for

FINALIZATION

if $j \geq 3$, then
    for $i = j - 1$ downto $2$ do
        Add $\triangle p_{i-1}p_ip_j$ to $Q_n$;
        end for
    end if
```

Figure 1. Algorithm for computing a serpentine polygonization.

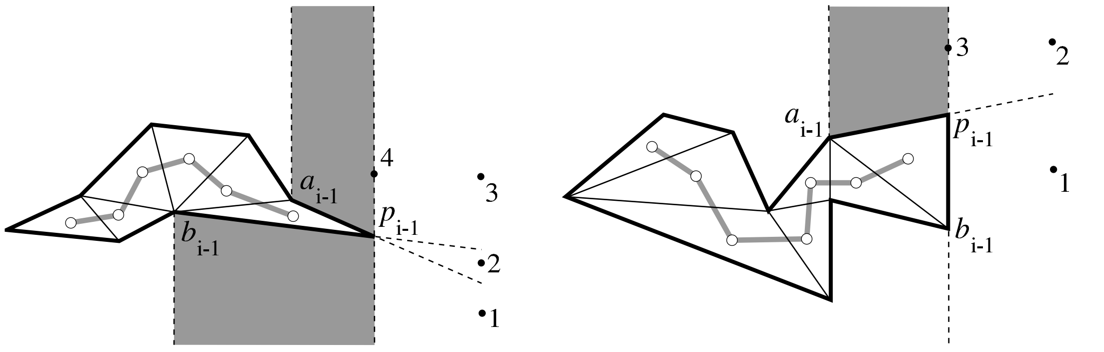


Figure 2. The two cases in the proof of Lemma 2.1.

above the segment $a_{i-1}p_{i-1}$ does not contain any of the points $p_1 \dots p_n$. The same happens with the similar region below the segment $b_{i-1}p_{i-1}$. These regions are shown shaded in the figure. Now the possible positions for p_i are essentially as labeled 1 to 4 in the figure; from 1 and 2 it may be attached to b_{i-1} and p_{i-1}, from 2, 3 and 4 it may be connected to a_{i-1} and p_{i-1}. Therefore we always obtain a new empty triangle which can be glued to Q_{i-1}. It is also clear that the resulting polygon Q_i is for all situations monotonic and has a serpentine triangulation given by the sequence of incrementally glued triangles.

Case 2. $x(p_{i-1}) = x(p_{i-2})$ (refer to Fig. 2 (b)). Now the situation is as shown in the figure, and we have essentially the possible positions 1, 2 and 3 for p_i, which are treated as in the preceeding paragraph.

□

The FINALIZATION step of the algorithm is necessary only when p_1, p_2 and p_3 are collinear, in which case the initial triangle $p_1p_{j-1}p_j$ is decomposed into a path of sub-triangles. Since the complexity of the algorithm is dominated by the sorting step, we obtain the following result:

Theorem 2.2 *A triangulated, serpentine, monotonic, polygonization of a set of n points in the plane, not all of them collinear, can be computed in $O(n \log n)$ time.*

At first glance it may appear that this *on-line* algorithm extends to $\mathbb{R}^3$ by "gluing" a new tetrahedron to one of the three faces incident on the last point of the polyhedron constructed thus far. Unfortunately, it may happen that none of these three faces is completely visible from the new point to be inserted, and therefore the method fails. An example of a set of points for which this procedure fails is shown in Fig. 3.

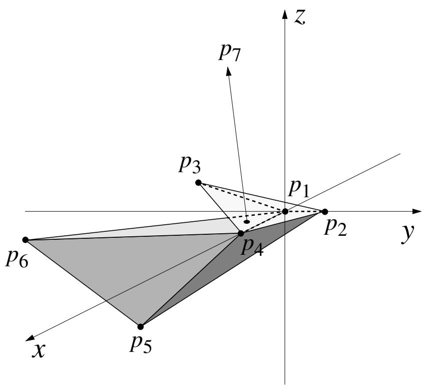


Figure 3. Illustrating the polyhedron constructed from the first three tetrahedra and the line on which the seventh point lies.

First consider the six points ordered by increasing x-coordinate: $p_1 = (0,0,0)$, $p_2 = (0,1,0)$, $p_3 = (0,-3,1)$, $p_4 = (1,0,0)$, $p_5 = (2,-5/2,-3)$ and $p_6 = (3,-4,1)$. The first, second, and third tetrahedra glued in the construction are given, respectively, by $p_1p_2p_3p_4, p_1p_2p_4p_5, p_1p_4p_5p_6$.

When viewed from the top ($+z$ direction) the projection of p_5 on the xy plane lies in the interior of the projection of the triangle $p_1p_6p_4$. Therefore the outer normals of faces $p_4p_5p_6$ and $p_1p_5p_6$ are pointing in the negative z direction. Furthermore, any point p_7 above the planes $p_1p_4p_6$, $p_1p_5p_6$ and $p_4p_5p_6$ cannot see faces $p_4p_5p_6$ and $p_1p_5p_6$. Finally, if p_7 is high enough it will not see face $p_1p_4p_6$ either, and if p_7 lies on a nearly vertical line slightly slanted towards the positive x axis, its x-coordinate can be made to be larger than that of p_6, as required.

3 Monotonic Polyhedronizations

Let S be a set of $n \geq 4$ points in $\mathbb{R}^3$ in general position, i.e., no four of them are coplanar (which we are assuming for the rest of the paper) . In 1994 Branko Grünbaum [12] outlined a constructive proof that S can always be polyhedronized. However, he was concerned neither with the properties of the polyhedronization nor with its computational complexity. As it turns out, his idea leads to an xy-monotonic polyhedronization, as defined below. In this section we present a simplification of his approach and show that it can be efficiently computed. We also present a method for computing a tetrahedralized, xy-monotonic, polyhedronization of S. A naïve implementation of this method takes quadratic time, but the running time can be improved to $O(n^{1+\varepsilon})$, for any $\varepsilon > 0$, by using a data structure of Agarwal and Matoušek [2].

Definition 3.1 A polyhedron is *xy-monotonic* if its intersection with every line parallel to the z-axis is either empty or a connected interval.

In other words, an xy-monotonic polyhedron $\mathbb{P}$ is bounded from above and below by polyhedral terrains. We refer to the upper (resp. lower) boundary of $\mathbb{P}$, i.e., the portion of its boundary that is visible from $z = +\infty$ (resp. $z = -\infty$) as the *upper terrain* (resp. *lower terrain*) of $\mathbb{P}$. The polygonal cycle that forms the common boundary of the lower and upper terrains of $\mathbb{P}$ is called its *shadow boundary*. Monotonic polyhedra are ubiquitous in geographic information systems and manufacturing applications, and admit efficient point-location queries by deciding whether a query point is above or below the terrains, which can be quickly achieved after performing point location in the projections of the terrains on the xy-plane. Hereafter we use the simpler term *monotonic* to mean xy-monotonic.

Theorem 3.1 *A monotonic polyhedronization of a set S of n points in general position in $\mathbb{R}^3$ can be computed in $O(n \log n)$ time.*

Proof: First rotate S so that it has a *regular* projection on the xy-plane, i.e., no two points of S lie on a vertical line. Next compute the convex hull $\text{conv}(S)$ of S. If no point of S lies inside $\text{conv}(S)$, we are done because $\text{conv}(S)$ is a monotonic polyhedron. Otherwise, let $\text{conv}_L(S)$ and $\text{conv}_U(S)$ be the lower and upper convex hull of S, i.e., the lower and upper terrains of $\text{conv}(S)$, respectively. Let B be the shadow boundary of $\text{conv}(S)$. Let $S_U \subset S$ be the set of points that are not vertices of $\text{conv}_L(S)$. Triangulate the xy-projection of the set $S_U \cup B$ without creating edges that have both endpoints on B, and lift each triangle to the points that projected onto its vertices. By gluing along B this terrain with $\text{conv}_L(S)$ we obtain the desired monotonic polyhedron.

A regular projection of n points can be computed in $O(n \log n)$ time using the algorithm of Gomez et al. [10]. Similarly, $\text{conv}(S)$ and a triangulation of the projection of $S_U \cup B$ can also be computed in $O(n \log n)$ time. Since $O(n)$ time is sufficient for the lifting step, the overall running time of the algorithm is $O(n \log n)$. $\square$

A drawback of the preceding construction is that the resulting polyhedronization may not admit a tetrahedralization, as is demonstrated in what follows. It is well known that a polyhedron may not always be triangulated. Lennes [16] was the first to exhibit such an *indecomposable* polyhedron. Lennes's counter-example contains seven vertices. In 1928 Schönhardt [21] strengthened this result by constructing a polyhedron with only six vertices that does not admit any diagonals, and proved that no indecomposable polyhedron exists with less than six vertices. In 1948 Bagemihl [5] generalized Schönhardt's result to polyhedra with any number $n > 6$.

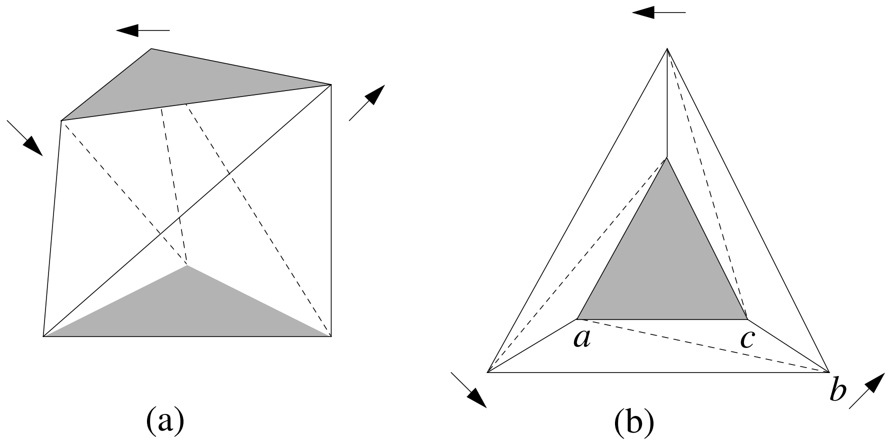


Figure 4. Constructing a polyhedral terrain that cannot be tetrahedralized.

First we show how to modify Schönhardt's construction to create a terrain that cannot be tetrahedralized. Then this object is embedded into a larger terrain to create an example for which the polyhedron constructed by Grünbaum's algorithm cannot be tetrahedralized. Schönhardt's six-vertex polyhedron is constructed by starting with a triangular prism (refer to Fig. 4 (a) where the top and bottom are shown shaded) and the top face is "twisted" in the direction shown by some small amount. The three side faces (rectangles consisting of two triangles each) cannot remain planar, and so "buckle" inwards along the diagonals to produce non-coplanar triangular faces. No two non-adjacent vertices of this polyhedron are internally visible from each other.

To create a terrain polyhedron that cannot be tetrahedralized, start with a prism that has a base larger than the top (see Fig. 4(b) where the top is shown shaded). Then add three points that form an even larger triangular base below the base of the prism, in such a way that the vertices of the base of the prism lie in the interior (strictly) of the convex hull of all nine points. By twisting the newly inserted base as before and selecting an appropriate triangulation of the projection of $S_U \cup B$ on the xy-plane, we obtain the desired polyhedron that cannot be tetrahedralized. Note that here $\mathrm{conv}_L(S)$ and B are both one and the same triangle.

We show next an alternative, somewhat more complicated, construction that does admit a tetrahedralization. For a point $p \in \mathbb{R}^3$, let p^* denote its xy-projection, and for a set $A \subset \mathbb{R}^3$, let $A^* = \{p^* \mid p \in A\}$.

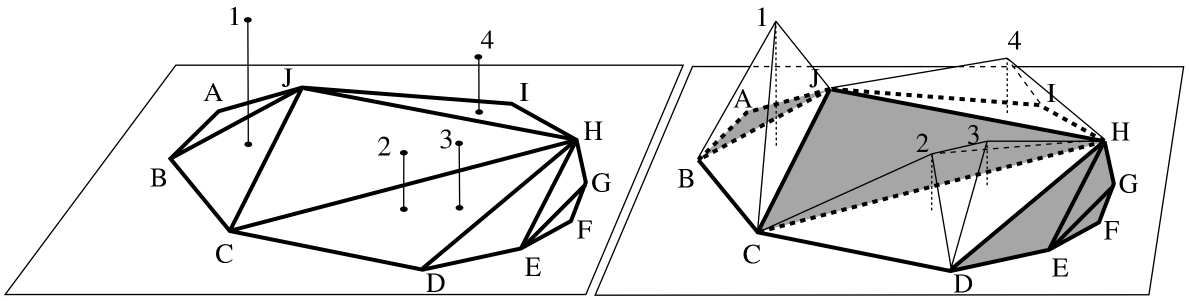


Figure 5. Illustrating point insertion in the proof of Lemma 3.2.

Lemma 3.2 *Let P be a triangulated convex polygon with vertex set V lying on the xy-plane, and let S be a point set in $\mathbb{R}^3$ lying above the xy-plane such that every point of S^* lies in the interior of P. Then it is possible to construct a tetrahedralized monotonic polyhedron with vertex set $V \cup S$ such that its lower terrain is P.*

Proof: The idea is to incrementally construct an upper terrain Π on top of P, each of whose faces is a triangle. At the end of the process P and Π will share exactly the boundary of P. The construction will also give a tetrahedralization of the monotonic polyhedron bounded by Π and P. The algorithm consists of two stages: (1) point insertion and (2) "skin" extension.

POINT INSERTION. This stage constructs an upper terrain by inserting the points of S one by one. Initially, Π is the same as P, so initially each face of Π is a triangle. Let Π be the terrain constructed so far. Let $\mathbb{P}$ be the polyhedron bounded by Π and P. For a triangle $xyz \in \Pi$, let $S_{xyz} \subseteq S$ be the set of poins such that S^*_{xyz} lies in the interior of $\triangle xyz$. In fact the construction will ensure that all points in S_{xyz} lie above $\triangle xyz$. Let xyz be a triangle in Π for which $S_{xyz} \neq \emptyset$ (see Fig. 5 a). Let u be the first point of S_{xyz} that xyz meets as we translate xyz in $(+z)$-direction. We delete xyz from Π and add the triangles uxy, uyz, and uzx to Π. The process is repeated until all points in S have been incorporated. Notice that at each step one tetrahedron is "glued" to $\mathbb{P}$. This stage produces a tetrahedralized monotonoic polyhedron.

If P is a triangle, then we are done, since $\mathbb{P}$ is simple in this case. Otherwise $\mathbb{P}$ may not be simple because some of the triangles and edges of P may also belong to Π (in which case the interior of $\mathbb{P}$ is not homemorphic to a ball). An example illustrating this situation is shown in Fig. 5 (b). The triangles of P that belong to Π, i.e., the triangles of P that are visible from the top $(z = +\infty)$ are drawn with grey interior and thick edges. Triangles that have been added to Π later are shown with white interior and thin edges. The same scenario, as seen from above, is shown in Fig. 6(a).

SKIN EXTENSION. In this stage we continue adding tetrahedra, one by one, to $\mathbb{P}$, keeping it monotonic, until none of the edges (and triangles) of P appear on its upper terrain Π. We mark all edges of Π that belong to P.

Suppose Π has at least one marked edge. Since $S \neq \emptyset$, there is a marked edge p_1p_2 on Π so that at least one of the triangles adjacent to p_1p_2 is not a triangle of P. Let $p_1p_2p_3$ and $p_2p_1p_4$ be the two triangles in Π adjacent to p_1p_2, and without loss of generality assume that $p_4 \in S$, i.e., $p_2p_1p_4$ is not a triangle of P. There are two cases to consider:

Case (i) $p_3 \in P$ (e.g. the diagonal BJ in Fig. 6(a)), i.e., $p_1p_2p_3$ is a triangle of P. Triangles $p_1^*p_2^*p_3^*$ and $p_2^*p_1^*p_4^*$ form a convex quadrilateral because $p_1^*p_2^*$ is a segment connecting two vertices of $\mathrm{conv}(P \cup S^*)$. We add the tetrahedron $p_1p_2p_3p_4$ to $\mathbb{P}$ and update Π by removing the triangle $p_2p_1p_4$ and the edge p_1p_2 from Π and adding the triangles $p_1p_4p_3$ and $p_4p_2p_3$ and the edge p_3p_4. In the example of diagonal BJ in Fig. 6(a), triangle $1JB$ and edge BJ are removed; triangles $B1A$, $A1J$ and edge $A1$ are added; see Fig. 6(b).

*C*ase (ii). $p_3 \in S$ (e.g. the edge CJ in Fig. 6(c)). In this case we add the tetrahedron $p_1p_2p_3p_4$ to $\mathbb{P}$ and update Π by removing the triangles $p_1p_2p_3$, $p_2p_1p_4$ and the edge p_1p_2, and adding the triangles $p_1p_4p_3$ and $p_3p_4p_2$ and the edge p_3p_4. For example, triangles $CJ1$, $CJ2$ and edge CJ in Fig. 6(c) are deleted, and triangles $1C2$, $1J2$ and edge 12 are added, as shown in Fig. 6(d).

The process is repeated until no marked edges in Π remain; $\mathbb{P}$ is the desired polyhedron. At each step a vertical line may pierce only once the upper terrain Π and only once the lower terrain P, which proves the claimed monotonicity. On the other hand, the incremental addition of tetrahedra, one by one, provides a tetrahedralization of $\mathbb{P}$. $\square$

Remark. The preceding lemma also applies when P is a lower convex hull instead of a plane triangulated polygon.

A direct implementation of the preceeding constructive proof leads to an algorithm with $O(n^2)$ time complexity. Next, we show that the construction of Lemma 3.2 can be carried out in $O(n^{1+\varepsilon})$ time, for any $\varepsilon > 0$. The skin-extension stage of the above construction can easily be carried out in linear time, so it suffices to describe an algorithm for the point-insertion stage that takes $O(n^{1+\varepsilon})$ time, for any $\varepsilon > 0$. Let $\Delta = xyz$ be a triangle constructed in the upper terrain of Π during the first stage, and let $S_\Delta \subseteq S$ be the set of points that lies above Δ, i.e., the ray from a point in S_Δ in $(-z)$-direction intersects Δ. We preprocess S_Δ into a dynamic data structure $\Psi(S_\Delta)$ by Agarwal and Matoušek [2] so that the following three operations can be performed efficiently:

UPDATE. Insert a point into S_Δ, or delete a point from S_Δ.

LP QUERY. Let h be a plane lying below all the points of S_Δ. Return the first point met by h as we translate it in $(+z)$-direction.

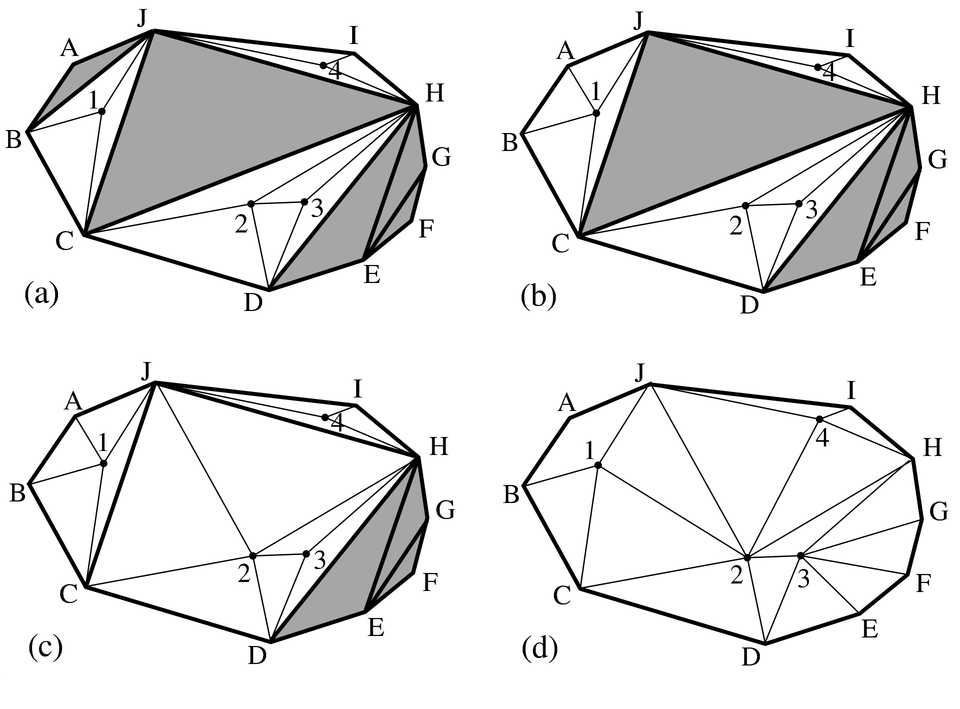


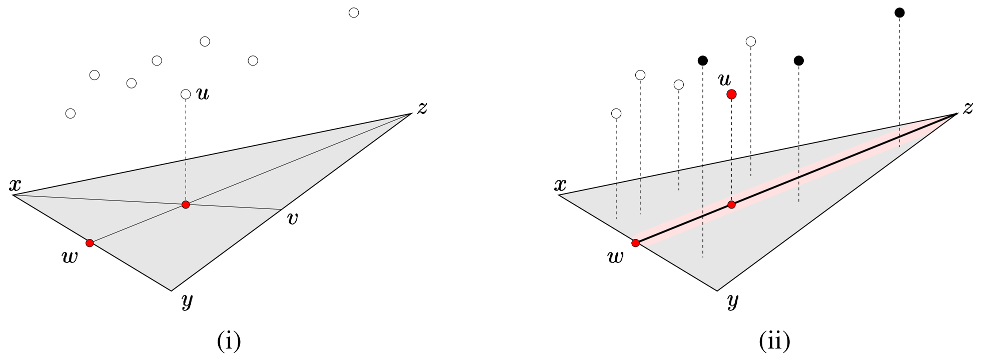
Figure 6. Illustrating skin extension in the proof of Lemma 3.2.

HALFSPACE QUERY. Return the points of $S_\triangle$ lying inside a query halfspace g.

Agarwal and Matoušek have shown that an update operation can be performed in $O(n^\varepsilon)$ time, and that an LP-query can be answered in $O(\log n)$ time. They also showed that, for a query halfspace g, a subset of k points in $g \cap S$ can be reported in $O(\log n + k)$ time in an incremental manner. That is, one can stop the query procedure at any time and if k points were reported, the time spent is $O(\log n + k)$.

Equipped with this data structure, we incorporate a point into Π as follows. Let $\Delta = xyz$ be a triangle for which $S_\triangle \neq \emptyset$. Let h be the plane supporting Δ. We find in $O(\log n)$ time the first point $u \in S_\triangle$ that h intersects as we translate it in $(+z)$-direction. Next, we add the triangles $\triangle xyu$, $\triangle yzu$, and $\triangle zxu$ to Π and remove xyz from Π. We also construct the data structures $\Psi(S_{xyu})$, $\Psi(S_{yzu})$, and $\Psi(S_{zxu})$. The latter is accomplished in four steps as follows. Let h_{uz} (resp. h_{ux}, h_{uy}) be the verical plane passing through u and z (resp. x, y).

I. We perform four half-space queries on $\Psi(S_\triangle)$ simulatenously. More precisely, we query $\Psi(S_\triangle)$ with both (open) halfspaces bounded by h_{uz} and by h_{ux} and report the points lying in each halfspace in a lock-step manner, i.e., return one point from each halfspace at a time, until we have reported all points of $S_\triangle$ lying in one of the four halfspaces. Suppose we report all the points lying in one of the halfspaces bounded by h_{uz}. Let w be the intersection point of the edge xy with h_{uz}. The points of $S_\triangle$ lying in the halfspace bounded by h_{uz} and containing x (resp. y) form the set S_{zxw} (resp. S_{yzw}). The total time spent by the above query procedure is $O(\log n + \min\{|S_{zxw}|, |S_{yzw}|\})$. Suppose $|S_{zxw}| \le |S_{ywz}|$. We delete the points of S_{zxw} from $\Psi(S_\Delta)$ and insert them into a new data structure, which is initially empty, to obtain $\Psi(S_{zxw})$. At the end of this step, which takes $O(\min\{|S_{zxw}|+1, |S_{yzw}|+1\} \cdot n^\varepsilon)$ time, we have $\Psi(S_{zxw})$ and $\Psi(S_{yzw})$ at our disposal.



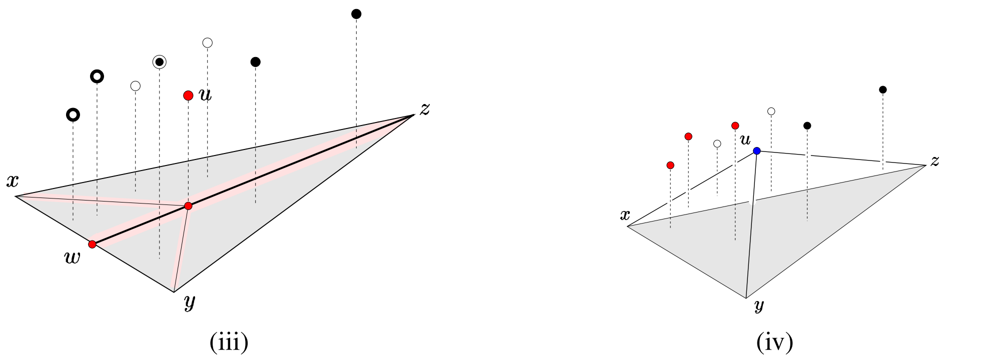


Figure 7. Incorporating a point u: (i) Querying S_Δ with four halfspaces bounded by the planes h_{ux} and h_{uz}; segments vx and wz denote $h_{ux} \cap \Delta$ and $h_{uz} \cap \Delta$, respectively. (ii) $\triangle xyz$ is partitioned into two triangles by segment wz, filled (hollow) circles denote poins in S_{yzw} (resp. S_{xwz}). (iii) Hollows circles with thick boundary denote points in S_{xwu}, and filled circles with double boundary denote points in S_{yuw}. (iv) Three new triangles xyu, yzu, and zxu.

II. Next, by querying $\Psi(S_{zxw})$ with the two halfspaces bounded by h_{xu} in the same manner as in Step I (but only performing two instead of four halfspace queries), we construct, in time $O(\min\{|S_{zxu}|+1, |S_{xwu}|+1\} \cdot n^\varepsilon)$, the data structures $\Psi(S_{zxu})$ and $\Psi(S_{xwu})$.

III. Similarly, by querying $\Psi(S_{yzw})$ with h_{yu}, we construct, in time $O(\min\{|S_{yzu}|+1, |S_{yuw}|+1\} \cdot n^\varepsilon)$, the data structures $\Psi(S_{yzu})$ and $\Psi(S_{yuw})$.

IV. By merging $\Psi(S_{xwu})$ and $\Psi(S_{yuw})$ in $O(\min\{|S_{yuw}|+1, |S_{xwu}|+1\} \cdot n^\varepsilon)$ time, we obtain $\Psi(S_{xyu})$, as desired.

Remark. In Step I, if we had reported all the points of S_Δ in a halfspace bounded by h_{ux} (instead of a halfspace bounded by h_{uz}), then in Step I we would have split S_Δ into two sets S_{xyv} and S_{zxv} by the plane h_{ux}. Step II and III would have constructed the structures $\Psi(S_{xyu})$, $\Psi(S_{uyv})$, $\Psi(S_{zxu})$, and $\Psi(S_{uvz})$. Finally, Step IV would have constructed $\Psi(S_{yzu})$ by merging $\Psi(S_{uyv})$ and $\Psi(S_{uvz})$.

Let κ_u be the number of delete operations that were performed in Steps I–III while adding u to Π. The number of insertions in Steps I–III is also bounded by κ_u. In Step IV, while constructing $\Psi(S_{xyu})$, we perform $\min\{|S_{xwu}|, |S_{yuw}|\}$ insertions to construct $\Psi(S_{xyu})$. Since $S_{xwu} \subseteq S_{zxw}$ and $S_{yuw} \subseteq S_{yzw}$, we conclude that

$$\min\{|S_{xwu}|, |S_{yuw}|\} \leq \min\{|S_{zxw}|, |S_{yzw}|\} \leq \kappa_u.$$

Hence, the total number of insertions performed in Steps I–IV is at most $2\kappa_u$. Finally, since halfspace queries were performed in lock-step manners, the total time spent in answering these queries is $O(\log n + \kappa_u)$. The overall running time of the algorithm is is thus $O(\sum_{u \in S}(1+\kappa_u)n^{\varepsilon})$. It thus suffices to bound $\sum_{u\in S} \kappa_u$.

We use a charging scheme to bound $\sum_u \kappa_u$. Initially, we assign $2\log_{3/2} n$ credits to each point of S. We use one unit of credit to pay for each delete operation. The charging scheme maintains the following invariant during the algorithm:

($\star$) *For any* $\triangle \in \Pi$, *each point in* $S_\triangle$ *has at least* $2\log_{3/2}|S_\triangle|$ *credits.*

This invariant ensures that the credit of each point remains positive throughout the algorithm, therefore the number of deletions is at most $\sum_u \kappa_u = O(n \log n)$, which in turn implies that the overall running time of the point-insertion stage is $O(n^{1+\varepsilon'})$ for any $\varepsilon' > \varepsilon$. Let $\chi(p)$ denote the number of credits the point p currently has. We now describe the charging scheme.

If $\max\{|S_{xyu}|, |S_{zxu}|, |S_{yzu}|\} \leq 2|S_\triangle|/3$, we charge one credit to each point whenever it is deleted in Steps I–III, to pay for its deletion. Note that each point is deleted at most twice. Assume that the invariant ($\star$) was maintained before u was inserted, and let $p \in S_{xyu}$ be a point that is deleted (at most twice) in Steps I–III. Then

$$\chi(p) \geq 2\log_{3/2}|S_\triangle| - 2 = 2\log_{3/2}(2|S_\triangle|/3) \geq 2\log_{3/2}|S_{xyu}|,$$

i.e., ($\star$) is satisfied after incorporating u as well. The same is true for points in S_{zxu} and S_{yzu}.

If $|S_{xyu}| \leq 2|S_\triangle|/3$, then we again charge one unit of credit to each point whenever it is deleted. If a point p in S_{zxu} was deleted (at most twice) in Steps I–III, then $|S_{zxu}| \leq |S_\triangle|/2$, and we can again argue that $\chi(p) \geq 2\log_{3/2}|S_{zxu}|$. The same argument holds for the points in S_{yzu}. By assumption, $|S_{xyu}| \leq 2n/3$, therefore each point in this set can also pay one unit of credit whenever it was deleted.

Finally, consider the case in which $|S_{xyu}| > 2n/3$. Let g_{ux} be the halfspace bounded by h_{ux} that does not contain $\triangle xyu$. Since we split $S_\triangle$ along h_{uz} in the first step instead of splitting along h_{ux}, we observe that

$$\min\{|S_{zxw}|, |S_{yzw}|\} \leq |g_{ux} \cap S_\triangle| \leq |S_{zxu}| + |S_{yzu}|.$$

Therefore we pay for every delete operation in the first step by charging one credit to a point in $S_{zxu} \cup S_{yzu}$. In the second and third steps, the total number of deletions is also bounded by $|S_{zxu}| + |S_{yzu}|$, we again pay for these deletions by charging one unit of credit to a point in these two sets. Since $|S_{zxu}| + |S_{yzu}| \leq n/3$ and we charge at most two units of credit from each point of these sets, invariance ($\star$) is maintained. Putting everything together we conclude the following.

Lemma 3.3 *The construction in the proof of Lemma 3.2 can be completed in $O(n^{1+\varepsilon})$ time, for any $\varepsilon > 0$.*

We now prove the main result of this section.

Theorem 3.4 *Given a set S of n points in $\mathbb{R}^3$, we can compute in time $O(n^{1+\varepsilon})$ a tetrahedralized monotonic polyhedronization of S.*

Proof: Compute the convex hull of S, its shadow boundary B, and the projection of B onto the xy-plane, B^*. Next compute a triangulation of B^*, and lift the triangulation back to $\mathbb{R}^3$. Let Π denote the resulting polyhedral terrain, each of whose face is a triangle. Let $S^+ \subseteq S \setminus B$ (resp. $S^- \subseteq S \setminus B$) be the set of points that lie above (resp. below) Π. The general-position assumption implies that no point of $S \setminus B$ lies on Π. Using Lemma 3.2, we first construct in $O(n^{1+\varepsilon})$ time a tetrahedralized monotonic polyhedronization $\mathbb{P}^+$ of $S^+ \cup B$ whose lower terrain is Π. Next, we construct a tetrahedralized monotonic polyhedronization $\mathbb{P}^-$ of $S^- \cup B$ whose upper terrain is Π. Then we glue $\mathbb{P}^+$ and $\mathbb{P}^-$ together to obtain a tetrahedralized monotonic polyhedronization of S. The total time spent is obviously $O(n^{1+\varepsilon})$. $\square$

4 Star-Shaped Polyhedronizations

In this section we present two methods for polyhedronizing point sets so that the resulting polyhedra are star-shaped from an edge, besides having other good properties. For the sake of completeness, we also describe two other methods, which might be considered as folklore, that lead to star-shaped polyhedra but possibly only from a vertex.

Definition 4.1 A polyhedron Q is star-shaped from a non-exterior point p if for all points $q \in Q$ the line segment pq lies in Q.

Hinge polyhedronizations. The following construction is called *hinge polyhedronization*. Start with any pair of points $x, y \in S$ for which xy is an edge of the convex hull $\text{conv}(S)$; let ℓ be the line containing the edge xy. Sort all the remaining points of S in the order they are encountered when a halfplane bounded by ℓ is rotated around ℓ. Connect all these points in sorted order, thereby obtaining an open polygonal chain. Finally connect every vertex of this chain to both x and y.

Theorem 4.1 *A hinge polyhedronization of a set S of n points in $\mathbb{R}^3$ can be constructed in $O(n \log n)$ time that has the following properties:*

1. *it is star-shaped (fan, edge-visible), serpentine, and Hamiltonian;*
2. *a point-location query can be answered in $O(\log n)$ time.*

Proof: Let $q_1, \ldots, q_{n-2}$ the points in $S \backslash \{x, y\}$ as they appear sorted in the chain. The first two properties follow immediately from the fact that the hinge polyhedronization consists of the union of tetrahedra defined by xy, q_i and q_{i+1}. The path $xq_1 \ldots q_{n-2}yx$ is a Hamiltonian cycle lying on the 1-skeleton of the polyhedron. Angular binary search with the halfplane having its hinge at xy allows point location as claimed.

The selection of an edge from $\operatorname{conv}(S)$ can be done in linear time, and sorting the points in $O(n \log n)$ time, which is the dominating step as the final connections are done in linear time. $\square$

Orange polyhedronizations. We describe here *orange polyhedronizations*, which are a slight modification of hinge polyhedronizations. Start with any pair of points $x, y \in S$ for which xy is *not* an edge of the convex hull $\operatorname{conv}(S)$; let ℓ be the line containing xy. Sort all the remaining points of S in cyclic order they are encountered when a halfplane bounded by ℓ is rotated around ℓ. Connect all these points in sorted order obtaining a *closed* polygonal chain, and finally connect every vertex of this chain to both x and y.

Theorem 4.2 *An orange polyhedronization of a set S of n points in $\mathbb{R}^3$ can be constructed in $O(n \log n)$ time that has the following properties:*

1. *it is star-shaped (from a diagonal) and Hamiltonian;*
2. *it admits a tetrahedralization whose dual is a cycle and has an Eulerian 1-skeleton for even values of n;*
3. *a point-location query can be answered in $O(\log n)$ time.*

Proof: Let $q_1, \ldots, q_{n-2}$ the points in $S \backslash \{x, y\}$ as they appear cyclically sorted in the chain. The first two properties follow immediately from the construction. The path $q_1xq_2yq_3 \ldots q_{n-2}q_1$ is a Hamiltonian cycle lying on the 1-skeleton of the polyhedron. The degrees of x and y in the 1-skeleton of the polyhedron are both $n-2$, while any other vertex has degree 4, therefore for n even all the degrees are even and the graph is Eulerian. Angular binary search with the halfplane having its hinge at xy allows point location as claimed. The computation is done in the same way as for the hinge polyhedronization in $O(n \log n)$ time. $\square$

Note that by deleting an "orange-wedge" tetrahedron from an orange polyhedronization, we obtain an alternate construction of a hinge polyhedron, except that the hinge is no longer a convex-hull edge.

Cone polyhedronizations. A *cone polyhedronization* with apex $q \in \operatorname{conv}(S)$ is constructed as follows: let the apex q be any vertex of $\operatorname{conv}(S)$. Consider a plane H that is parallel to a plane supporting $\operatorname{conv}(S)$ at q but that does not contain q. Let S^* be the perspective projection of $S \backslash \{q\}$ onto the plane H from q. Triangulate S^* and lift every triangle in the triangulation to the corresponding original points in space. Finally, connect to q the points that project to convex-hull vertices of $\operatorname{conv}(S^*)$

Theorem 4.3 *A cone polyhedronization of a set S of n points in $\mathbb{R}^3$ can be constructed in $O(n \log n)$ time that has the following properties:*

1. *it is star-shaped (fan) and admits a tetrahedralization;*
2. *a point-location query can be answered in $O(\log n)$ time.*

Proof: The first two properties are obvious from the construction. Given a query point q, let q^* be the intersection of the line pq with the plane H. If q^* is outside $\text{conv}(S^*)$ then q is outside of the polyhedron. Otherwise, determine which triangle contains q^* and test q for inclusion in the tetrahedron corresponding to this triangle. Kirkpatrick's algorithm allows this to be done within the claimed bound [15].

There are several methods for picking a vertex v of the convex hull and a suitable plane H in $O(n \log n)$ time, for example, by computing the convex hull itself. S^* is then obtained in linear time and triangulated in $O(n \log n)$ time. Finally the triangulation can be preprocessed in $O(n \log n)$ time using linear space to support logarithmic time point-location queries, using Kirkpatrick's algorithm. $\square$

It is worth noticing that the preceding method works as well in $\mathbb{R}^d$. In this case S^* lies on a hyperplane, and once its convex hull is simplicially decomposed the resulting $(d-1)$-simplices can be lifted to S and completed to d-simplices with q.

Pyramid polyhedronizations. A *pyramid polyhedronization* is a slight variation of the cone polyhedronization, in which instead of constructing a triangulation of S^* we obtain a planar polygonization $q_1^* q_2^* \ldots q_{n-1}^* q_1^*$ of S^* and lift a triangulation of this polygon. Furthemore, if we use the serpentine triangulation from the preceding section, we obtain a serpentine polyhedronization, which is also Hamiltonian, because $q_1 v q_2 q_3 \ldots q_{n-1} q_1$ is a Hamiltonian cycle in its 1-skeleton. As for point location, Kirpatrick's algorithm can also be used in this context, and in fact is even easier. Therefore we have the following theorem:

Theorem 4.4 *A pyramid polyhedronization of a set S of n points in $\mathbb{R}^3$ can be constructed in $O(n \log n)$ time that has the following properties:*

1. *it is star-shaped (fan) and Hamiltonian;*
2. *it admits a serpentine tetrahedralization;*
3. *a point-location query can be answered in $O(\log n)$ time.*

Proof: The two first properties are obvious from the construction. Given a query point q, let q^* be the intersection of the line pq with the plane H. If q^* is outside $\text{conv}(S^*)$ the q is outside of the polyhedron. Otherwise, we determine which triangle contains q^* and test q for inclusion in the tetrahedron corresponding to this triangle. The complexity follows directly from Kirkpatrick's algorithm. $\square$

Remark. In the preceeding we have considered polygonizations in the plane and polyhedronizations in $\mathbb{R}^3$. However, the problem of polygonizing points in $\mathbb{R}^3$ has also received some attention in the literature. Gemignani [8] gives a construction for polygonizing a set of n points in $\mathbb{R}^3$ that leads to an $O(n^2 \log n)$-time algorithm. As a side benefit of our results, the polyhedronizations that are Hamiltonian solve Gemignani's problem in $O(n \log n)$ time. An alternate $O(n \log n)$ time algorithm can be obtained by first computing a regular projection with the algorithm of Gomez et al. [10], then obtaining a polygonization of the planar projection, and finally lifting the projected polygon back into space.

5 Conclusions

In this paper we have systematically studied various methods for generating polyhedronizations that have a variety of desirable properties: monotonicity, star-shapedness, admitting a tetrahedralization (possibly with nice dual structure), possessing a good 1-skeleton from the viewpoint of graph theory, and affording fast point-location queries.

Some interesting open problems remain, for example, whether it is always possible to construct a polyhedronization with an Eulerian 1-skeleton. On the other hand, all of our methods may yield vertices of high degree, and it is natural to consider which methods would lead to a 1-skeleton graph with bounded degree.

Finally, let us mention that combinatorial problems on counting polyhedronizations may be quite challenging. For example, consider n points in $\mathbb{R}^d$ in convex position; for $d = 2$ they admit only one polyhedronization (polygonization) but this is not the case for $d > 2$, and even obtaining non-trivial bounds in $\mathbb{R}^3$ appears to be a difficult problem.

References

[1] M. Abellanas, J. Garcia, G. Hernandez, F. Hurtado, and O. Serra, Onion polygonizations, *Information Processing Letters*, 57 (1996), 165–173.

[2] P. K. Agarwal and J. Matoušek, Dynamic half-space range reporting and its applications, *Algorithmica*, 13 (1995), 325–345.

[3] S. G. Akl and G. T. Toussaint, A fast convex hull algorithm, *Information Processing Letters*, 7 (1978), 219–222.

[4] A. M. Andrew, Another efficient algorithm for convex hulls in two dimensions, *Information Processing Letters*, 9 (1979), 216–219.

[5] F. Bagemihl, On indecomposable polyhedra, *American Mathematical Monthly*, (1948), 411–413.

[6] J. D. Boissonat, Reconstruction of solids, *Communications of the ACM*, (1985), 46–54.

[7] M. Gemignani, More on finite subsets and simple closed polygonal paths, *Mathematics Magazine*, 39 (1966), 158–160.

[8] M. Gemignani, On finite subsets of the plane and simple closed polygonal paths, *Mathematics Magazine*, (1966), 38–41.

[9] C. Gitlin, J. O'Rourke, and V. Subramanian, On reconstructon of polyhedra from parallel slices, *International Journal of Computational Geometry and Applications*, 6 (1996), 103–122.

[10] F. Gomez, S. Ramaswami, and G. Toussaint, On computing general position views of data in three dimensions, *Journal of Visual Communication and Image Representation*, 12 (2001), 387–400.

[11] R. L. Graham, An efficient algorithm for determining the convex hull of a finite planar set, *Information Processing Letters*, 1 (1972), 132–133.

[12] B. Grünbaum, Hamiltonian polygons and polyhedra, *Geombinatorics*, 3 (1994), 83–89.

[13] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan, Linear time algorithms for shortest path and visibility problems inside triangulated simple polygons, *Algorithmica*, 2 (1987), 209–233.

[14] D. S. Johnson, Computational complexity, in: *The Traveling Salesman Problem* (L. E. L., J. K. Lenstra, A. H. G. Rinnooy, and D. B. Shmoys, eds.), John Wiley, 1985, pp. 37–87.

[15] D. G. Kirkpatrick, Optimal search in planar subdivisions, *SIAM Journal of Computing*, 12 (1983), 28–35.

[16] N. J. Lennes, Theorems on the simple finite polygon and polyhedron, *American Journal of Mathematics*, 33 (1911), 37–62.

[17] J. O'Rourke, H. Booth, and R. Washington, Connect-the-dots: a new heuristic, *Computer Vision, Graphics and Image Processing*, 39 (1987), 258–266.

[18] J. O'Rourke and V. Subramanian, On reconstructing polyhedra from parallel slices, Tech. Rep. 008, Smith College, June 1991.

[19] L. V. Quintas and F. Supnick, On some properties of shortest Hamiltonian circuits, *American Mathematical Monthly*, 72 (1965), 977–980.

[20] D. Sanders, Metric spaces in which minimal circuits cannot self-intersect, *Proc. American Mathematical Society*, Vol. 36, April 1976, pp. 383–387.

[21] E. Schönhardt, Uber die zerlegung von dreieckspolyedern in tetraeder, *Mathematische Annalen*, 98 (1928), 309–312.

[22] M. Shamos, Geometric complexity, *Proceedings of the Seventh ACM Symposium on the Theory of Computing*, 1975, pp. 224–253.

[23] T. Shermer, Computing bushy and thin triangulations, *Computational Geometry: Theory and Applications*, 1 (1991), 115–125.

[24] H. Steinhaus, *One Hundred Problems in Elementary Mathematics*, Dover Publications, Inc., New York, 1964.

[25] G. T. Toussaint, Solving geometric problems with the rotating calipers, *Proc. IEEE MELECON' 83*, May 1983.

[26] G. T. Toussaint, A new linear algorithm for triangulating monotone polygons, *Pattern Recognition Letters*, 2 (1984), 155–158.

[27] G. T. Toussaint, A linear-time algorithm for solving the strong hidden-line problem in a simple polygon, *Pattern Recognition Letters*, 4 (1986), 449–451.

[28] G. T. Toussaint, ed., *Computational Morphology*, North-Holland, 1988.

[29] G. T. Toussaint, Efficient triangulation of simple polygons, *The Visual Computer*, 7 (1991), 280–295.

[30] T. C. Woo and S. Y. Shin, A linear time algorithm for triangulating a point-visible polygon, *ACM Transactions on Graphics*, 4 (1985), 60–70.