Computing the Straight Skeleton of an Orthogonal Monotone Polygon in Linear Time*

Günther Eder[1], Martin Held[1], and Peter Palfrader[1]

1 **Universität Salzburg, FB Computerwissenschaften, 5020 Salzburg, Austria**
{geder,held,palfrader}@cs.sbg.ac.at

Abstract

We introduce a simple algorithm to construct the straight skeleton of an n-vertex orthogonal monotone polygon in optimal $\mathcal{O}(n)$ time and space.

1 Introduction

The straight skeleton $\mathcal{S}(\mathcal{P})$ of a simple polygon $\mathcal{P}$ was introduced by Aichholzer et al. [2]. It is the result of a wavefront-propagation process where the edges of $\mathcal{P}$ move inwards at unit speed in a self-parallel manner, forming one or more wavefront polygons whose combinatorics change when wavefront edges collapse or wavefront vertices move into other parts of the wavefront. The straight skeleton is the trace of the vertices of these wavefront polygons over their propagation, cf. Figure 1.

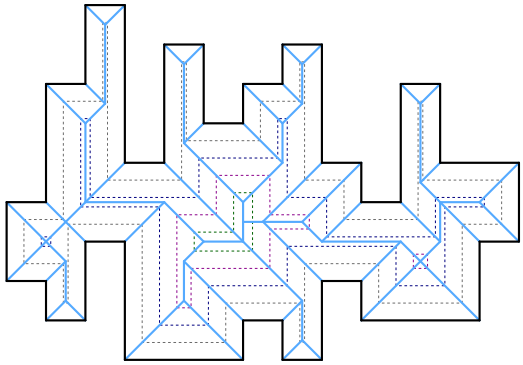

Figure 1 A polygon (black) with its straight skeleton (blue) and some wavefronts (dotted).

The currently best known algorithm for constructing the straight skeleton of unrestricted input is by Eppstein and Erickson [5] and runs in $\mathcal{O}(n^{17/11+\varepsilon})$ time and space for an n-vertex polygon and any $\varepsilon > 0$. In the case of a convex input polygon, the straight skeleton coincides with the medial axis and can be computed in linear time [1]. For monotone polygons, Biedl et al. [3] present an algorithm to compute the straight skeleton in $\mathcal{O}(n \log n)$ time.

In this work we show that an approach which is similar to that of Biedl et al. [3] makes it possible to construct $\mathcal{S}(\mathcal{P})$ in optimal linear time if $\mathcal{P}$ is monotone and orthogonal: We also construct the straight skeleton for each of the two monotone chains of $\mathcal{P}$ separately and then merge them to obtain $\mathcal{S}(\mathcal{P})$. Since $\mathcal{P}$ is orthogonal, $\mathcal{S}(\mathcal{P})$ coincides with the Voronoi diagram of $\mathcal{P}$ in the L_∞-norm [2]. Papadopoulou shows how to compute the L_∞-norm Voronoi diagram of orthogonal planar straight-line graphs in $\mathcal{O}(n \log n)$ time [6].

* Work supported by Austrian Science Fund (FWF): Grant ORD 53-VO.

35th European Workshop on Computational Geometry, Utrecht, The Netherlands, March 18–20, 2019.
This is an extended abstract of a presentation given at EuroCG'19. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

1.1 Preliminaries

Let $\mathcal{P}$ be an x-monotone, axis-aligned orthogonal polygon. For the sake of descriptive simplicity we assume that $\mathcal{P}$ has no vertex with interior angle equal to π. (Our algorithm can be extended to handle such input at no additional computational cost.) Let $\mathcal{C}_l$ and $\mathcal{C}_u$ denote the lower and upper monotone chain of $\mathcal{P}$, respectively. As the leftmost and rightmost edges of $\mathcal{P}$ are vertical, we arbitrarily assign the leftmost edge to $\mathcal{C}_l$ and the rightmost edge to $\mathcal{C}_u$.

For an edge e of $\mathcal{P}$, denote by $I(e)$ the half-plane induced by the supporting line $\ell(e)$ of e which locally (close to e) overlaps with the interior of $\mathcal{P}$. For two non-parallel edges e_i and e_j of the polygon, the bisector $b_{i,j}$ is the ray lying on the angular bisector of the supporting lines of e_i and e_j within the common interior region $I(e_i) \cap I(e_j)$. If the edges e_i, e_j are parallel then we use their wavefront edges $e_i(t)$ and $e_j(t)$ to build $b_{i,j}$. (But we will still refer to $b_{i,j}$ as the bisector of e_i and e_j.) If the wavefront edges overlap at a specific time t' then $b_{i,j}$ is the segment formed by the non-empty intersection $e_i(t') \cap e_j(t')$. Otherwise, if $e_i(t')$ and $e_j(t')$ share a common end-point p then $b_{i,j}$ is the ray perpendicular to them that starts at p and lies in the common interior $I(e_i) \cap I(e_j)$. A wavefront vertex that moves along such a bisector is called a *ghost vertex* [4], and we call the resulting straight-skeleton arc a *ghost arc*. It is *unfinished* if its extent is not yet known.

Let $\mathcal{C}$ be an x-monotone polygonal chain. For $e_i \in \mathcal{C}$, let Π_i denote the portion of $I(e_i)$ that is incident at e_i and limited by the two bisectors through the endpoints of e_i. We call Π_i the *half-plane slab* of e_i.

▶ **Lemma 1.1.** *Every face of $\mathcal{S}(\mathcal{C})$ is monotone with respect to its defining edge and is also monotone with respect to a line perpendicular to its defining edge.*

▶ **Corollary 1.2.** *Every face of $\mathcal{S}(\mathcal{C})$ is x-monotone.*

▶ **Lemma 1.3.** *For every edge e_i of $\mathcal{C}$ the straight-skeleton face $f(e_i)$ incident at e_i lies inside of the half-plane slab Π_i.*

2 Computing the Straight Skeleton of a Single Chain

In order to compute the straight skeleton of a polygon, we first construct the straight skeletons of its lower and upper chains individually, and then we merge them; cf. Section 3. We start with describing the construction of the skeleton $\mathcal{S}(\mathcal{C}_l)$ of the lower chain $\mathcal{C}_l := e_1, \ldots, e_{n'}$. (The upper chain $\mathcal{C}_u$ is processed in an identical fashion.) If we extend the leftmost edge e_1 and rightmost edge $e_{n'}$ of $\mathcal{C}_l$ to infinity then the plane is split into two areas. The area which contains $\mathcal{P}$ is tessellated by $\mathcal{S}(\mathcal{C}_l)$ into straight-skeleton faces, with one face $f(e_i)$ being incident at each input edge e_i. As $f(e_i)$ is monotone with respect to the normal of e_i (Lemma 1.1), we can meaningfully split the arcs bounding $f(e_i)$ into a left and a right chain. Each chain is a list of arcs (edges and potentially one ray) that starts in a vertex of e_i and either ends in a ray for unbounded faces or ends when it meets the other chain. If the final arc of a chain is parallel to e_i then it can be assigned arbitrarily to the left or the right chain.

We construct the straight skeleton of $\mathcal{C}_l$ incrementally. As we insert edges of $\mathcal{C}_l$ from left to right, we maintain a partial straight skeleton $\mathcal{S}^*$. We store in $\mathcal{S}^*$ for each input edge e_i the left chain of $f(e_i)$ as a list of arcs. Additionally, $\mathcal{S}^*$ maintains a stack `R` of edges whose faces have a left chain that terminates in a ray and another stack `G` of edges which have faces whose left chain terminates in an unfinished ghost arc, a vertical edge where the second endpoint is not yet known. Figure 2 (left) shows blue and purple rays of `R` and `G`, respectively. Initially, `R` contains e_1, which does not have a left chain as it extends to infinity itself.

Once an edge e_i has been inserted into $\mathcal{S}^*$, the left chain of $f(e_i)$ in $\mathcal{S}^*$ can be modified only in two specific ways: If the left chain of $f(e_i)$ ends in a ray, this ray may be replaced by a bounded segment. If the left chain of $f(e_i)$ ends in an unfinished ghost arc, this arc may be replaced by a bounded, finished ghost arc, or it may be replaced by a bounded, finished ghost arc followed by another bounded segment. In Figure 2, the ray $b_{2,3}$ is replaced by a bounded segment and the ghost arc a' is terminated at the intersection with arc a. As we

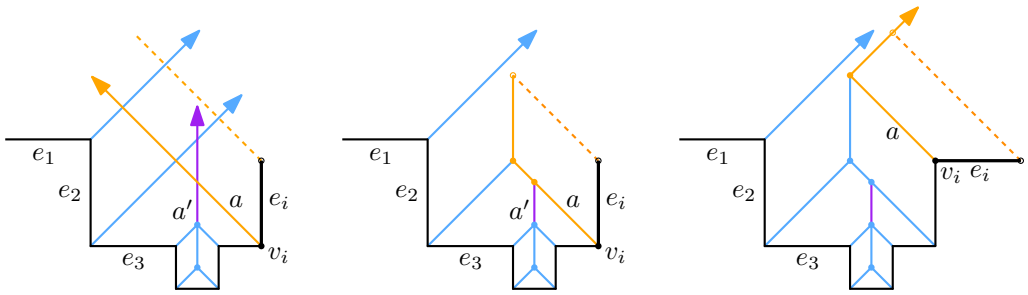

Figure 2 (Left, Center) Inserting edge e_i; (Right) Incrementing i and adding the next edge e_i; $\mathcal{S}^*$, including rays, in blue; (Unfinished) Ghost arcs in purple, and arcs added due to e_i in orange.

insert edge e_i, we have to build the left chain of its face. We do this iteratively, starting at the left vertex of e_i. This chain starts with an arc that lies on a bisector with direction vector either $\binom{-1}{1}$ or $\binom{1}{1}$. We continue to append arc segments, starting each arc where the previous segment terminated. We stop when we append a ghost arc (which we will complete later), when we append a ray, or in some cases when we appended a bounded (vertical) arc segment. Note that all arcs lie on bisectors with direction vectors $\binom{-1}{1}$, $\binom{0}{1}$, or $\binom{1}{1}$. The direction of the initial arc segment is given by the bisector of e_i and its predecessor e_{i-1}. For all subsequent arcs, the direction depends on e_i and the edge on top of the stack `R`.

▶ **Lemma 2.1.** *No arc inserted into $\mathcal{S}^*$ with direction $\binom{1}{1}$ can intersect any arc of $\mathcal{S}^*$ and, thus, is a ray which escapes to infinity.*

Arcs with Direction $\binom{1}{1}$ Therefore, if the left chain of $f(e_i)$ ever includes an arc on a $\binom{1}{1}$-bisector, this arc will be a $\binom{1}{1}$-ray because nothing in the current $\mathcal{S}^*$ can intersect it. Thus, this arc terminates the left chain. We add e_i with this chain to $\mathcal{S}^*$ and also put e_i on the corresponding stack `R`.

Arcs with Direction $\binom{-1}{1}$ If the arc a to be added to the left chain of $f(e_i)$ lies on a $\binom{-1}{1}$-bisector, then this arc may be a ray but it may also be bounded and interact with faces previously inserted. If this is the first arc of the left chain of $f(e_i)$ then we check whether the previous face, $f(e_{i-1})$, has its left chain already completed, terminating in a bounded segment. If this is true then a is an arc from the left vertex of e_i along the bisector to the end of the previous face's chain. Otherwise, or if this is not the first arc of the left chain, we look at the top of our stack `R`. Let e_t be the edge on top of `R`. If the left chain of $f(e_t)$ does not terminate in a $\binom{1}{1}$-ray, then there is no $\binom{1}{1}$-ray in $\mathcal{S}^*$, and a therefore is a $\binom{-1}{1}$-ray which finishes the left chain of this face. An argument similar to the one used in Lemma 2.1, now applied to $\binom{-1}{1}$-direction, shows that a cannot be intersected by any new arc. If `G` is not empty then all unfinished ghost arcs on that stack get turned into finished ghost-arcs which terminate at their intersection points with a. If, however, the left chain of $f(e_t)$ terminates in a $\binom{1}{1}$-ray r then a will intersect r in a point p. We modify $f(e_t)$ by replacing r with a line

segment r' terminating at p, and we remove e_t from R. Furthermore a is a line segment that terminates at p. Lastly, we have to process any elements on G that had been inserted after e_t was processed, as these ghost arcs lie below r' and a. These get popped from G and their unfinished ghost arcs are replaced with arcs terminating where they intersect a or r'.

Arcs with Direction $\binom{0}{1}$ The last possible direction that an arc a of the left chain of e_i may have is $\binom{0}{1}$. Let p be the point where the previous arc ended. Then a is either a ghost arc or a standard vertical arc. In the first case we push e_i onto G. We also store a reference to the edge on top of R at this time, and thereby complete the processing of $f(e_i)$. The latter case is the result of two vertical input segments whose wavefront segments meet. Let e_t be the edge at the top of R. Then a is the line segment from p that is contained in both Π_t and Π_i. If the extent of a is limited by Π_i only, then this finishes the construction of the left chain of $f(e_i)$. Otherwise, the ray of the left chain of $f(e_t)$ touches the end-point of a. We replace that ray with a bounded segment terminating at this intersection and pop e_t from R. If the extent of a was limited by Π_t only, we continue with constructing the next arc of the left chain of $f(e_i)$ whose direction is determined by e_i and the edge now on top of R. Otherwise, if a was limited by both Π_t and Π_i, the construction of the left chain of $f(e_i)$ is also finished.

▶ **Lemma 2.2.** *All arcs created by inserting e_i intersect only rays or ghost arcs of $\mathcal{S}^*$.*

Finalizing $\mathcal{S}(\mathcal{C}_l)$ Once all input edges have been inserted into $\mathcal{S}^*$, we may still have elements on the stack G. For every element on G, we replace the unfinished ghost arc with a segment of finite length that terminates at the $\binom{1}{1}$-ray which it intersects first. We know where to look since we stored a reference to the correct face when we pushed an element onto G. If there is no $\binom{1}{1}$-ray (because the reference was to e_1), then these ghost arcs are finalized as vertical rays that go to infinity. The resulting structure $\mathcal{S}^*$ is now the straight skeleton $\mathcal{S}(\mathcal{C}_l)$.

▶ **Theorem 2.3.** *Our incremental construction computes the straight skeleton of a monotone orthogonal chain of n vertices in $\mathcal{O}(n)$ time and space.*

3 Merging $\mathcal{S}(\mathcal{C}_l)$ and $\mathcal{S}(\mathcal{C}_u)$ into $\mathcal{S}(\mathcal{P})$

While merging the two skeletons we create a polygonal merge chain $\mathcal{M} := a_1, \ldots, a_m$. This chain connects the first (western) vertex v_W of $\mathcal{C}_l$ and $\mathcal{C}_u$ to their last (eastern) vertex v_E. This merge is similar to the algorithm used for merging Voronoi diagrams of point sites [7].

▶ **Lemma 3.1** (Lemmas 4 and 5 in Biedl et al. [4])**.** *The polygonal chain $\mathcal{M}$ created by merging $\mathcal{S}(\mathcal{C}_u)$ and $\mathcal{S}(\mathcal{C}_l)$ is x-monotone.*

We use the notion of a bisector between two faces and thereby relate to the bisector between the two edges that define these faces. Let $f_l(i)$ and $f_u(j)$ denote the i-th and j-th face of $\mathcal{S}(\mathcal{C}_l)$ and $\mathcal{S}(\mathcal{C}_u)$, ordered from left to right along each chain, where $0 < i < n'$ and $0 < j < n''$. Clearly, every arc of $\mathcal{M}$ is a portion of a bisector between two such faces, one from $\mathcal{S}(\mathcal{C}_l)$ and one from $\mathcal{S}(\mathcal{C}_u)$; cf. Figure 3 (Left).

We start at the first vertex $p := v_W$ and look at the bisector b between $f_l(1)$ and $f_u(1)$. It starts at p. Let p' denote the intersection closest to p between b and $\mathcal{S}(\mathcal{C}_l)$ as well as b and $\mathcal{S}(\mathcal{C}_u)$. W.l.o.g., we assume that p' is formed between b and an arc a of $\mathcal{S}(\mathcal{C}_l)$. Let $f_l(i)$ denote the second face incident at a, with $1 < i \leq n'$. Then p' forms a node in $\mathcal{S}(\mathcal{P})$ that has the same distance to the edges of $f_u(1)$, $f_l(1)$, and $f_l(i)$. Thus, we let a end at p' and add an arc $a_1 := \overline{pp'}$ to $\mathcal{M}$. This arc also forms an arc in $\mathcal{S}(\mathcal{P})$. For the next incremental

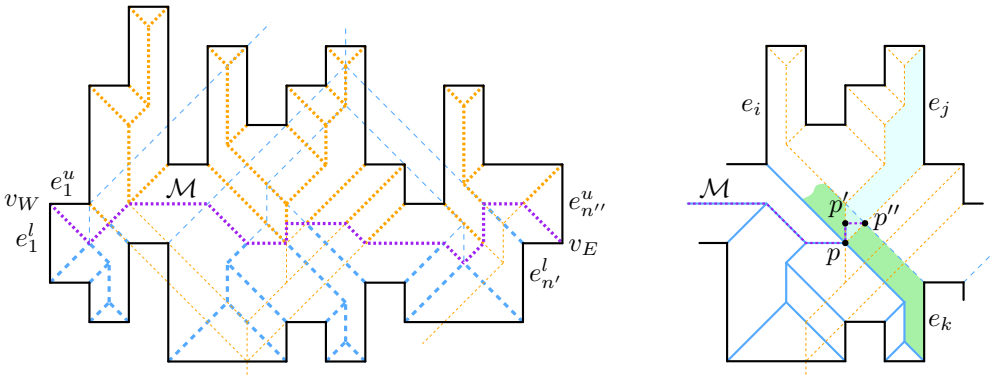

Figure 3 (Left) Monotone orthogonal polygon (black) with the upper (orange) and lower (blue) partial skeleton and the merge line $\mathcal{M}$ (purple); (Right) The merge step creates a vertical arc $\overline{pp'}$.

step, let $p := p'$ and let b denote the bisector between $f_u(1)$ and $f_l(i)$. It starts at p. Again we find the next intersection p' closest to p of b with both skeletons that does not lie left of p. We repeat this process until we arrive at the last vertex v_E.

Note that the intersection between b and one of the skeletons can form a vertical line segment instead of a single point if b coincides with a vertical skeleton arc; cf. Figure 3 (Right). To find the next node p' in this case we have to look at the relevant faces of $\mathcal{S}(\mathcal{C}_l)$ and $\mathcal{S}(\mathcal{C}_u)$. Let s denote the vertical segment formed by such an intersection and let a denote the vertical arc that is intersected. W.l.o.g., we assume that a belongs to $\mathcal{S}(\mathcal{C}_u)$. Let $f_u(i)$ and $f_u(j)$ denote the two faces incident at a. Let $f_l(k)$ together with $f_u(i)$ define b. Thus, the next bisector b' that starts on some point on s is defined by $f_u(j)$ and $f_l(k)$. We observe that both e_j and e_k must be vertical and have the same distance to s, since the bisector between e_i, e_j and between e_i, e_k lies on a common line. Hence, e_j and e_k lie on a common supporting line and their faces in the upper and lower skeleton contain s. We can infer that the wavefront edges $e_j(t)$ and $e_k(t)$ must become adjacent at some point p''. At that point a ghost vertex traces out a horizontal arc. Since we are in the process of merging the two skeletons, this arc is not yet present in either of the two skeletons. Both faces, $f_u(j)$ and $f_l(k)$, are x-monotone. To find p'' we start at their defining input edge and walk along their boundary until we find that intersection. A horizontal line through p'' intersects s and defines the node p' sought.

Complexity of the Merge At every step a pair of faces $f_l(i)$ and $f_u(j)$ contributes a single arc to $\mathcal{M}$. Upon insertion of this arc we increase at least one of the two indices i, j. Thus, $\mathcal{M}$ has size at most $n' + n''$, which is equal to n. It remains to discuss how to find the next intersection p' efficiently. A new bisector b, defined by the faces $f_l(i)$ and $f_u(j)$, starts at the known point p. Both faces are x-monotone; cf. Corollary 1.2. The monotonicity of a face also holds after the merge since $\mathcal{M}$ is x-monotone as well. We use the x-coordinate of the vertex that traces out b to walk along the boundary of both faces. When an intersection is found we add a node, cross to the neighboring face, and start a new arc. Since $\mathcal{M}$ is x-monotone we can simply continue traversing the face whose boundary was not intersected. Therefore, we traverse every arc of both $\mathcal{S}(\mathcal{C}_l)$ and $\mathcal{S}(\mathcal{C}_u)$ at most once.

▶ **Theorem 3.2.** *The merge process merges $\mathcal{S}(\mathcal{C}_l)$ and $\mathcal{S}(\mathcal{C}_u)$ and obtains $\mathcal{S}(\mathcal{P})$ for an n-vertex monotone orthogonal polygon $\mathcal{P}$ in $\mathcal{O}(n)$ time.*

References

1 Alok Aggarwal, Leonidas J. Guibas, James Saxe, and Peter W. Shor. A Linear-Time Algorithm for Computing the Voronoi Diagram of a Convex Polygon. *Discrete & Computational Geometry*, 4(6):591–604, 1989. doi:10.1007/BF02187749.

2 Oswin Aichholzer, Franz Aurenhammer, David Alberts, and Bernd Gärtner. A Novel Type of Skeleton for Polygons. *Journal of Universal Computer Science*, 1(12):752–761, 1995. doi:10.1007/978-3-642-80350-5_65.

3 Therese Biedl, Martin Held, Stefan Huber, Dominik Kaaser, and Peter Palfrader. A Simple Algorithm for Computing Positively Weighted Straight Skeletons of Monotone Polygons. *Information Processing Letters*, 115(2):243–247, February 2015. doi:10.1016/j.ipl.2014.09.021.

4 Therese Biedl, Martin Held, Stefan Huber, Dominik Kaaser, and Peter Palfrader. Weighted Straight Skeletons in the Plane. *Computational Geometry: Theory and Applications*, 48(2):120–133, 2015. doi:10.1016/j.comgeo.2014.08.006.

5 David Eppstein and Jeff Erickson. Raising Roofs, Crashing Cycles, and Playing Pool: Applications of a Data Structure for Finding Pairwise Interactions. *Discrete & Computational Geometry*, 22(4):569–592, 1999. doi:10.1145/276884.276891.

6 Evanthia Papadopoulou. Critical Area Computation for Missing Material Defects in VLSI Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(5):583–597, May 2001. doi:10.1109/43.920683.

7 Michael I. Shamos and Dan Hoey. Closest-Point Problems. In *Foundations of Computer Science, 1975, 16th Annual Symposium on*, pages 151–162. IEEE, October 1975. doi:10.1109/sfcs.1975.8.