A Faster Algorithm for Computing Straight Skeletons*

Siu-Wing Cheng[1], Liam Mencel[2], and Antoine Vigneron[2]

[1] The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
scheng@cse.ust.hk
[2] King Abdullah University of Science and Technology
Thuwal 23955-6900, Saudi Arabia
{antoine.vigneron,liam.mencel}@kaust.edu.sa

Abstract. We present a new algorithm for computing the straight skeleton of a polygon. For a polygon with n vertices, among which r are reflex vertices, we give a deterministic algorithm that reduces the straight skeleton computation to a motorcycle graph computation in $O(n(\log n)\log r)$ time. It improves on the previously best known algorithm for this reduction, which is randomized, and runs in expected $O(n\sqrt{h+1}\log^2 n)$ time for a polygon with h holes. Using known motorcycle graph algorithms, our result yields improved time bounds for computing straight skeletons. In particular, we can compute the straight skeleton of a non-degenerate polygon in $O(n(\log n)\log r + r^{4/3+\varepsilon})$ time for any $\varepsilon > 0$. On degenerate input, our time bound increases to $O(n(\log n)\log r + r^{17/11+\varepsilon})$.

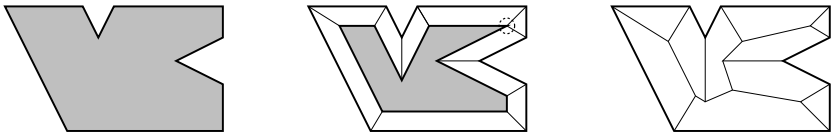
1 Introduction

The straight skeleton of a polygon is defined as the trace of the vertices when the polygon shrinks, each edge moving at the same speed inwards in a perpendicular direction to its orientation. (See Fig. 1.) It differs from the medial axis [7] in that it is a straight line graph embedded in the original polygon, while the medial axis may have parabolic edges. The notion was introduced by Aichholzer et al. [1] in 1995, who gave the earliest algorithm for computing the straight skeleton. However, the concept has been recognized as early as 1877 by von Peschka [20], in his interpretation as projection of roof surfaces.

The straight skeleton has numerous applications in computer graphics. It allows to compute offset polygons [14], which is a standard operation in CAD. Other applications include architectural modelling [19], biomedical image processing [8], city model reconstruction [10], computational origami [11–13] and polyhedral surface reconstruction [2, 9, 15]. Improving the efficiency of straight skeleton algorithms increases the speed of related tools in geometric computing.

The first algorithm by Aichholzer et al. [1] runs in $O(n^2 \log n)$ time, and simulates the shrinking process discretely. Eppstein and Erickson [14] developed the

* Liam Mencel was supported by KAUST base funding. Research supported by the Research Grants Council, Hong Kong, China (project no. 611711).



(a) The input polygon $\mathcal{P}$. (b) An offset of $\mathcal{P}$. (c) Straight skeleton $\mathcal{S}$.

Fig. 1: The straight skeleton is obtained by shrinking the input polygon $\mathcal{P}$.

first sub-quadratic algorithm, which runs in $O(n^{17/11+\varepsilon})$ time. In their work, they proposed motorcycle graphs as a means of encapsulating the main difficulty in computing straight skeletons. Cheng and Vigneron [6] expanded on this notion by reducing the straight skeleton problem in non-degenerate cases to a motorcycle graph computation and a lower envelope computation. This reduction was later extended to degenerate cases by Held and Huber [17]. Cheng and Vigneron gave an algorithm for the lower envelope computation on a non-degenerate polygon with h holes, which runs in $O(n\sqrt{h+1}\log^2 n)$ expected time. They also provided a method for solving the motorcycle graph problem in $O(n\sqrt{n}\log n)$ time. Putting the two together gives an algorithm which solves the straight skeleton problem in $O(n\sqrt{h+1}\log^2 n + r\sqrt{r}\log r)$ expected time, where r is the number of reflex vertices.

Comparison with previous work. Recently, Vigneron and Yan [21] found a faster, $O(n^{4/3+\varepsilon})$-time algorithm for computing motorcycle graphs. It thus removed one bottleneck in straight skeleton computation. In this paper we remove the second bottleneck: We give a faster reduction to the motorcycle graph problem. Our algorithm performs this reduction in deterministic $O(n(\log n)\log r)$ time, improving on the previously best known algorithm, which is randomized and runs in expected $O(n\sqrt{h+1}\log^2 n)$ time [6]. Recently, Bowers independently discovered an $O(n\log n)$-time, deterministic algorithm to perform this reduction in the case of simple polygons, using a very different approach [4].

Using known algorithms for computing motorcycle graphs, our reduction yields faster algorithms for computing the straight skeleton. In particular, using the algorithm by Vigneron and Yan [21], we can compute the straight skeleton of a non-degenerate polygon in $O(n(\log n)\log r + r^{4/3+\varepsilon})$ time for any $\varepsilon > 0$. On degenerate input, we use Eppstein and Erickson's algorithm, and our time bound increases to $O(n(\log n)\log r + r^{17/11+\varepsilon})$. For simple polygons whose coordinates are $O(\log n)$-bit rational numbers, we can compute the straight skeleton in $O(n\log^2 n)$ time using the motorcycle graph algorithm by Vigneron and Yan [21] (even in degenerate cases). Table 1 summarizes the previously known results and compares with our new algorithm.

Our approach. We use the known reduction to a lower envelope of slabs in 3D [6, 17]: First a motorcycle graph induced by the input polygon is computed, and then this graph is used to define a set of slabs in 3D. The lower envelope of these

	Previously best known	This paper
Arbitrary polygon	$O(n^{8/11+\varepsilon}r^{9/11+\varepsilon})$ [14]	$O(n(\log n)\log r + r^{17/11+\varepsilon})$
Non-degenerate polygon	$O^*(n\sqrt{r}\log^2 n)$ [6]	$O(n(\log n)\log r + r^{4/3+\varepsilon})$
Simple pol., arbitrary	$O^*(n\log^2 n + r^{17/11+\varepsilon})$ [6, 14]	$O(n(\log n)\log r + r^{17/11+\varepsilon})$
Simple pol., $O(\log n)$ bits	$O^*(n\log^2 n)$ [6, 21]	$O(n\log^2 n)$

Table 1: O^* denotes the expected time bound of a randomized algorithm, and O is for deterministic algorithms. To make the comparison easier, we replaced the number of holes h with r, as $h = O(r)$.

slabs is a terrain, whose edges vertically project to the straight skeleton on the horizontal plane. (See Section 2.)

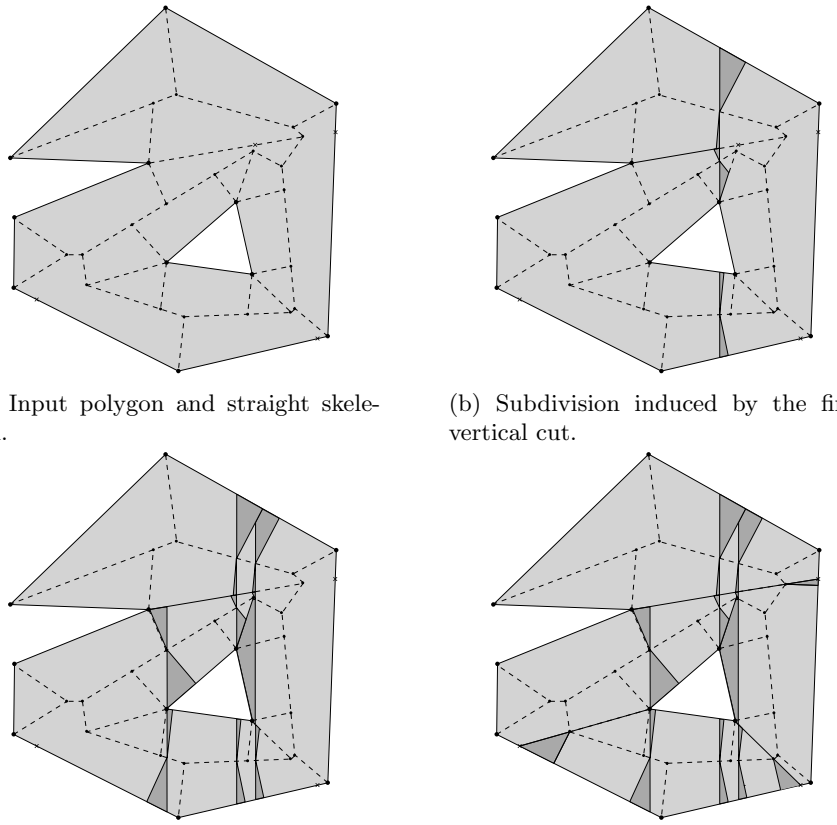
The difficulty is that these slabs may cross, and in general their lower envelope is a non-convex terrain, so known algorithms for computing lower envelopes of triangles would be too slow for our purpose. Our approach is thus to remove non-convex features: We compute a subdivision of the input polygon into convex cells such that, above each cell of this subdivision, the terrain is convex. Then the portion of the terrain above each cell can be computed efficiently, as it reduces to computing a lower envelope of planes in 3D. The subdivision is computed recursively, using a divide and conquer approach, in two stages.

During the first stage (Section 3), we partition using vertical lines, that is, lines parallel to the y-axis. At each step, we pick the vertical line ℓ through the median motorcycle vertex in the current cell. We first cut the cell using ℓ, and we compute the restriction of the terrain to the space above ℓ, which forms a polyline. It can be computed in near-linear time, as it reduces to computing a lower envelope of line segments in the vertical plane through ℓ. Then we cut the cell using the steepest descent paths from the vertices of this polyline. (See Fig. 2b.) We recurse until the current cell does not contain any vertex of the motorcycle graph. (See Fig. 2c.)

The first step ensures that the cells of the subdivision are convex. However, valleys (non-convex edges) may still enter the interior of the cells. So our second stage (Section 4) recursively partitions cells using lines that split the set of valleys of the current cell, instead of vertical lines. (See Fig. 2d.) As the first stage results in a partition where the restriction of the motorcycle graph to any cell is outerplanar, we can perform this subdivision efficiently by divide and conquer.

Each time we partition a cell, we know which slabs contribute to the child cells, as we know the terrain along the vertical plane through the cutting line. In addition, we will argue via careful analysis that our divide and conquer approach avoids slabs being used in too many iterations, and hence the algorithm completes in $O(n(\log n)\log r)$ time.

Due to space limitation, some proofs are missing from this extended abstract. A more detailed description of our algorithm, as well as the missing proofs, can be found in the full version of this paper [5]. We state here our main result:



(a) Input polygon and straight skeleton.

(b) Subdivision induced by the first vertical cut.

(c) Result of the vertical subdivision.

(d) Final subdivision.

Fig. 2: Example of subdivision computed by our algorithm.

Theorem 1. *Given a polygon $\mathcal{P}$ with n vertices, r of which being reflex vertices, and given the motorcycle graph induced by $\mathcal{P}$, we can compute the straight skeleton of $\mathcal{P}$ in $O(n(\log n)\log r)$ time.*

Our algorithm does not handle weighted straight skeletons [14] (where edges move at different speeds during the shrink process), because the reduction to a lower envelope of slabs does not hold in this case.

2 Notations and Preliminaries

The input polygon is denoted by $\mathcal{P}$. A *reflex vertex* of a polygon is a vertex at which the internal angle is more than π. $\mathcal{P}$ has n vertices, among which r are reflex vertices. We work in $\mathbb{R}^3$ with $\mathcal{P}$ lying flat in the xy-plane. The z-axis becomes analogous to the time dimension. We say that a line, or a line segment, is *vertical*, if it is parallel to the y-axis, and we say that a plane is vertical if

it is orthogonal to the xy-plane. The boundary of a set A is denote by ∂A. We denote by $\overline{pq}$ the line segment with endpoints p, q.

Terrain. At any time, the horizontal plane $z = t$ contains a snapshot of $\mathcal{P}$ after shrinking for t units of time. While the shrinking polygon moves vertically at unit speed, faces are formed as the trace of the edges, and these faces make an angle $\pi/4$ with the xy-plane. The surface formed by the traces of the edges is the *terrain* $\mathcal{T}$. (See Fig. 3 a.) The traces of the vertices of $\mathcal{P}$ form the set of edges of $\mathcal{T}$. An edge e of $\mathcal{T}$ is *convex* if there is a plane through e that is above the two faces bounding e. The edges of $\mathcal{T}$ corresponding to the traces of the reflex vertices will be referred to as *valleys*. Valleys are the only non-convex edges on $\mathcal{T}$. The other edges, which are convex, are called *ridges*. The *straight skeleton* $\mathcal{S}$ is the graph obtained by projecting the edges and vertices of $\mathcal{T}$ orthogonally onto the xy-plane. We also call valleys and ridges the edges of $\mathcal{S}$ that are obtained by projecting valleys and ridges of $\mathcal{T}$ onto the xy-plane.

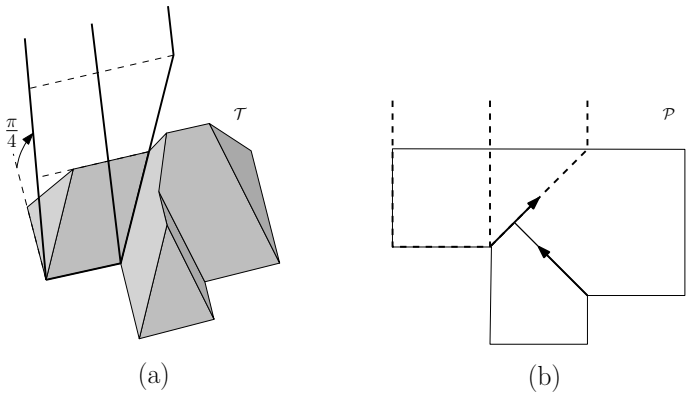


Fig. 3: Illustration of the two different types of slabs. (a) The terrain $\mathcal{T}$, an edge slab and motorcycle slab. This terrain has two valleys, adjacent to the two reflex vertices of the polygon. (b) The motorcycle graph associated with $\mathcal{P}$ and the boundaries of the edge slab and the motorcycle slab viewed from above.

Motorcycle graph. Our algorithm for computing the straight skeleton assumes that a motorcycle graph induced by $\mathcal{P}$ is precomputed [6]. This graph is defined as follows. A motorcycle is a point moving at a fixed velocity. We place a motorcycle at each reflex vertex of $\mathcal{P}$. The velocity of a motorcycle is the same as the velocity of the corresponding reflex vertex when $\mathcal{P}$ is shrunk, so its direction is the bisector of the interior angle, and its speed is $1/\sin(\theta/2)$, where θ is the exterior angle at the reflex vertex. (See Fig. 4a.)

The motorcycles begin moving simultaneously. They each leave behind a track as they move. When a motorcycle collides with either another motorcycle's

(a) (b)

Fig. 4: Motorcycle graph.

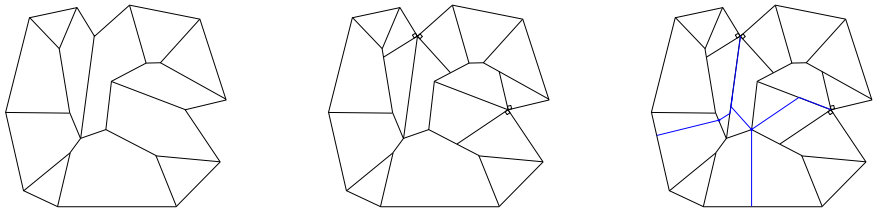
track or the boundary of $\mathcal{P}$, the colliding motorcycle halts permanently. (In degenerate cases, a motorcycle may also collide head-on with another motorcycle, but for now we rule out this case.) After all motorcycles stop, the tracks form a planar graph called the *motorcycle graph induced by* $\mathcal{P}$. (see Fig. 4b.)

General position assumptions. To simplify the description and the analysis of our algorithm, we assume that the polygon is in general position. No edge of $\mathcal{P}$ or $\mathcal{S}$ is vertical. No two motorcycles collide with each other in the motorcycle graph, and thus each valley is adjacent to some reflex vertex. Each vertex in the straight skeleton graph has degree 1 or 3. Our results, however, generalize to degenerate polygons.

Lifting map. The *lifted* version $\hat{p}$ of a point $p \in \mathcal{P}$ is the point on $\mathcal{T}$ that is vertically above p. In other words, $\hat{p}$ is the point of $\mathcal{T}$ that projects orthogonally to p on the xy-plane. We may also apply this transformation to a line segment s in the xy-plane, then $\hat{s}$ is a polyline in $\mathcal{T}$. We will abuse notation and denote by $\hat{\mathcal{G}}$ a lifted version of $\mathcal{G}$ where the height of a point is the time at which the corresponding motorcycle reaches it. Then the lifted version $\hat{e}$ of an edge e of $\mathcal{G}$ does not lie entirely on $\mathcal{T}$, but it contains the corresponding valley, and the remaining part of $\hat{e}$ lies above $\mathcal{T}$ [6]. (See Fig. 3a.)

Given a point $\hat{p}$ that lies in the interior of a face f of $\mathcal{T}$, there is a unique steepest descent path from $\hat{p}$ to the boundary of $\mathcal{P}$. This path consists either of a straight line segment orthogonal to the base edge e of f, or it consists of a segment going straight to a valley, and then follows this valley. (In degenerate cases, the path may follow several valleys consecutively.) If $\hat{p}$ is on a ridge, then two such descent paths from p exist, and if $\hat{p}$ is a convex vertex, then there are three such paths. (See Fig. 5c.)

Reduction to a lower envelope. Following Eppstein and Erickson [14], Cheng and Vigneron [6], and Held and Huber [17], we use a construction of the straight skeleton based on the lower envelope of a set of three-dimensional slabs. Each edge e of $\mathcal{P}$ defines an *edge slab*, which is a 2-dimensional half-strip at an angle of $\pi/4$ to the xy-plane, bounded below by e and along the sides by rays perpendicular to e. (See Fig. 3.) We say that e is the *source* of this edge slab.



(a) The skeleton $\mathcal{S}$. (b) The skeleton $\mathcal{S}'$. (c) Descent paths.

Fig. 5: The polygon $\mathcal{P}$, its skeletons, and descent paths.

For each reflex vertex $v = e \cap e'$, where e and e' are edges of $\mathcal{P}$, we define two *motorcycle slabs* making angles of $\pi/4$ to the xy-plane. One motorcycle slab is bounded below by the edge of $\hat{\mathcal{G}}$ incident to v and is bounded on the sides by two rays from each end of this edge in the ascent direction of e. The other motorcycle slab is defined similarly with e replaced by e'. The *source* of a motorcycle slab is the corresponding edge of $\hat{\mathcal{G}}$. Cheng and Vigneron [6] proved the following result, which was extended to degenerate cases by Huber and Held [16]:

Theorem 2. *The terrain $\mathcal{T}$ is the restriction of the lower envelope of the edge slabs and the motorcycle slabs to the space vertically above the polygon.*

Our algorithm constructs a graph $\mathcal{S}'$, which is obtained from $\mathcal{S}$ by adding two edges at each reflex vertex v of $\mathcal{P}$ going inwards and orthogonally to each edge of $\mathcal{P}$ incident to v. (See Fig. 5b.) We also include the edges of $\mathcal{P}$ into $\mathcal{S}'$. It means that each face f of $\mathcal{S}'$ corresponds to exactly one slab. More precisely, a face is the vertical projection of $\mathcal{T} \cap \sigma$ to the xy-plane for some slab σ. By contrast, in the original straight skeleton $\mathcal{S}$, a face incident to a reflex vertex corresponds to one edge slab and one motorcycle slab.

3 Computing the Vertical Subdivision

In this section, we describe and we analyze the first stage of our algorithm, where the input polygon $\mathcal{P}$ is recursively partitioned using vertical cuts. The corresponding procedure is called Divide-Vertical. It results in a subdivision of the input polygon $\mathcal{P}$, such that any cell of this subdivision has the following property: It does not contain any vertex of $\mathcal{G}$ in its interior, or it is contained in the union of two faces of $\mathcal{S}'$.

3.1 Subdivision Induced by a Vertical Cut

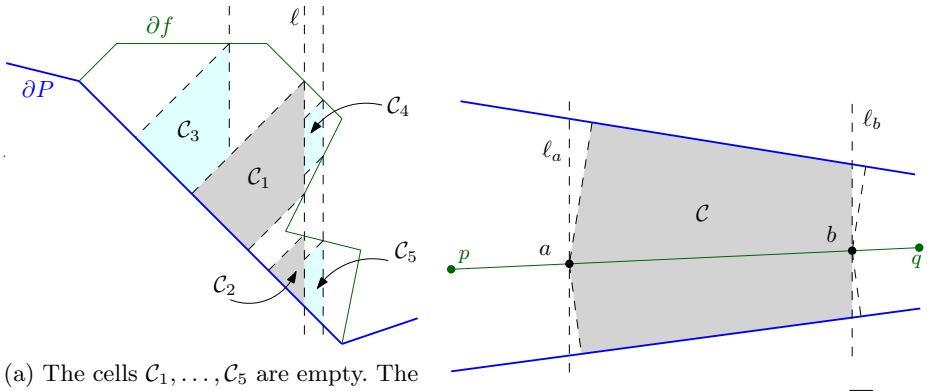
At any step of the algorithm, we maintain a planar subdivision $\mathcal{K}(\mathcal{P})$, which is a partition of the input polygon $\mathcal{P}$ into polygonal cells. Each cell is a polygon, hence it is connected. A cell $\mathcal{C}$ in the current subdivision $\mathcal{K}(\mathcal{P})$ may be further subdivided as follows.

Let ℓ be a vertical line through a vertex of $\mathcal{G}$. We assume that ℓ intersects $\mathcal{C}$, and hence $\mathcal{C} \cap \ell$ consists of several line segments $s_1, \ldots, s_q$. These line segments are introduced as new boundary edges in $\mathcal{K}(\mathcal{P})$; they are called the *vertical edges* of $\mathcal{K}(\mathcal{P})$. They may be further subdivided during the course of the algorithm, and we still call the resulting edges vertical edges.

We then insert non-vertical edges along steepest descent paths, as follows. Note that we are able to efficiently compute the intersection $\mathcal{S}' \cap \ell$ without knowing $\mathcal{S}'$. To do this, we make use of an algorithm by Hershberger [18] and compute the lower envelope of the slabs restricted to the vertical plane through ℓ, using $O(n \log n)$ time. The points at which this envelope changes angle are precisely the points on $\mathcal{T}$ which project onto $\mathcal{S}' \cap \ell$. Each intersection point $p \in s_j \cap \mathcal{S}'$ has a lifted version $\hat{p}$ on $\mathcal{T}$. By our non-degeneracy assumptions, there are at most three steepest descent paths to $\partial\mathcal{C}$ from $\hat{p}$. The vertical projections of these paths onto $\mathcal{C}$ are also inserted as new edges in $\mathcal{K}(\mathcal{P})$. The resulting partition of $\mathcal{C}$ is the *subdivision induced by* ℓ. (See Fig. 2.)

We denote by $\mathcal{C}_1, \mathcal{C}_2, \ldots$ the cells of $\mathcal{K}(\mathcal{P})$ that are constructed during the course of the algorithm. Let ℓ_i^- and ℓ_i^+ denote the vertical lines through the leftmost and rightmost point of $\mathcal{C}_i$, respectively. When we perform one step of the subdivision, each new cell lies entirely to the left or to the right of the splitting line, and thus by induction, any vertical edge of a cell $\mathcal{C}_i$ either lies in ℓ_i^- or ℓ_i^+.

An *empty cell* is a cell of $\mathcal{K}(\mathcal{P})$ whose interior does not overlap with $\mathcal{S}'$. (See Fig. 6a.) Thus an empty cell is entirely contained in a face of $\mathcal{S}'$. Another type of



(a) The cells $\mathcal{C}_1, \ldots, \mathcal{C}_5$ are empty. The first cut is performed along ℓ.

(b) The wedge $\mathcal{C}$ corresponding to $\overline{ab}$.

Fig. 6: Empty cells and a wedge.

cell, called a *wedge*, will play an important role in the analysis of our algorithm. Let $\overline{pq}$ be a ridge of $\mathcal{S}'$, and let a, b be two points in the interior of $\overline{pq}$. Let ℓ_a and ℓ_b be the vertical lines through a and b, respectively. Consider the subdivision of $\mathcal{P}$ obtained by inserting vertical boundaries along ℓ_a and ℓ_b, and the four descent

paths from a and b. (See Fig. 6b.) The cell of this subdivision containing $\overline{ab}$ is called the wedge corresponding to $\overline{ab}$.

3.2 Data Structure

During the course of the algorithm, we maintain the polygon $\mathcal{P}$ and its subdivision $\mathcal{K}(\mathcal{P})$ in a doubly-connected edge list [3]. So each cell $\mathcal{C}_i$ is represented by a circular list of edges, or several if it has holes. In the following, we show how we augment these chains so that they record incidences between the boundary of $\mathcal{C}_i$ and the faces of $\mathcal{S}'$.

For each cell $\mathcal{C}_i$, let $\mathcal{S}'_i$ be the subdivision of $\mathcal{C}_i$ induced by $\mathcal{S}'$. So the faces of $\mathcal{S}'_i$ are the connected components of $\mathcal{C}_i \setminus \mathcal{S}'$. Let Q denote a circular list of edges that form one component of $\partial\mathcal{C}_i$. We subdivide each vertical edge of Q at each intersection point with an edge of $\mathcal{S}'$. Now each edge e of Q bounds exactly one face f_j of $\mathcal{S}'_i$. We store a pointer from e to the slab σ_j corresponding to f_j. In addition, for each vertex of Q which is a reflex vertex of $\mathcal{P}$, we store pointers to the two corresponding motorcycle slabs. We call this data structure a *face list*. So we store one face list for each connected component of $\partial\mathcal{C}_i$.

We say that a vertex v of the motorcycle graph $\mathcal{G}$ *conflicts* with a cell $\mathcal{C}_i$ of $\mathcal{K}(\mathcal{P})$ if either v lies in the interior of $\mathcal{C}_i$, or v is a reflex vertex of $\partial\mathcal{C}_i$. We also store the list of all the vertices conflicting with each cell $\mathcal{C}_i$. This list V_i is called the *vertex conflict list* of $\mathcal{C}_i$. The size of this list is denoted by v_i. In summary, our data structure consists of:

- A doubly-connected edge list storing $\mathcal{K}(\mathcal{P})$.
- The face lists and the vertex conflict list V_i of each cell $\mathcal{C}_i$.

We say that an edge e of $\mathcal{S}'$ conflicts with the cell $\mathcal{C}_i$ if it intersects the interior of $\mathcal{C}_i$. So any edge of $\mathcal{S}'_i$ that is not on $\partial\mathcal{C}_i$ is of the form $e \cap \mathcal{C}_i$ for some edge e of $\mathcal{S}'$ conflicting with $\mathcal{C}_i$. We denote by c_i the number of edges conflicting with $\mathcal{C}_i$. During the course of the algorithm, we do not necessarily know all the edges conflicting with a cell $\mathcal{C}_i$, and we don't even know c_i, but this quantity will be useful for analyzing the running time. In particular, it allows to bound the size of the data structure for $\mathcal{C}_i$. (The proof is omitted due to space limitation.)

Lemma 1. *If $\mathcal{C}_i$ is non-empty, then the total size of the face lists of $\mathcal{C}_i$ is $O(c_i)$. In particular, it implies that $\partial\mathcal{C}_i$ has $O(c_i)$ edges, and $\mathcal{C}_i$ overlaps $O(c_i)$ faces of $\mathcal{S}'$. On the other hand, if $\mathcal{C}_i$ is empty, then the total size is $O(1)$, and thus $\partial\mathcal{C}_i$ has $O(1)$ edges.*

3.3 Algorithm

Our algorithm partitions $\mathcal{P}$ recursively, using vertical cuts, as in Sect. 3.1. A cell $\mathcal{C}_i$ is subdivided along a vertical cut ℓ through its median conflicting vertex, so the vertex conflict lists of the new cells will be at most half the size of the conflict lists of $\mathcal{C}_i$. When the vertex conflict list of $\mathcal{C}_i$ is empty, we call the procedure

DIVIDE-VALLEY. If $\mathcal{C}_i$ is empty or is a wedge, then we stop subdividing $\mathcal{C}_i$, and it becomes a *leaf cell*.

In the induced subdivision, the descent paths cannot cross, and by construction they do not cross the vertical boundary edges. Each edge of $\mathcal{S}'_i$ may create at most three such descent paths, so we create $O(c_i)$ such new descent paths. There are also $O(c_i)$ new vertical edges, so we can update the doubly-connected edge list in time $O(c_i \log c_i)$ by plane sweep. Using an additional $O(v_i \log c_i)$ time, we can update the vertex conflict lists during this plane sweep. The face lists can be updated in overall $O(c_i)$ time. It follows that:

Lemma 2. *We can compute the subdivision of a non-empty cell $\mathcal{C}_i$ induced by a line through its median conflicting vertex, and update our data structure accordingly, in $O((c_i + v_i) \log c_i)$ time.*

3.4 Analysis

Due to space limitation, we only give a sketch of proof for the running time of our algorithm. By Lemma 2, we only need to bound the total size of the conflict lists during the course of the algorithm. As there are only $2r$ motorcycle vertices, and each cell contains at most half as many as its parent, the total size of the vertex conflict lists is $O(r \log r)$.

We also show that the sum of the sizes of the edge conflict lists is $O(n \log r)$. We split into two cases. First, consider the cells containing vertices of $\mathcal{S}'$. At each subdivision, a vertex of $\mathcal{S}'$ in the cell being subdivided moves to a cell whose vertex conflict list has at most half the size of its parent's, so each vertex of $\mathcal{S}'$ is contained in $O(\log r)$ cells throughout the algorithm. Hence we create $O(n \log r)$ such cells. We then consider cells that overlap $\mathcal{S}'$, but none of its vertices. We argue that these cells must be wedges, and that each edge of $\mathcal{S}'$ yields $O(\log r)$ wedges.

Lemma 3. *The vertical subdivision procedure completes in $O(n(\log n) \log r)$ time. The cells of the resulting subdivision are either empty cells, wedges, or do not contain any motorcycle vertex in their interior. They are simply connected, and the only reflex vertices on their boundaries are along valleys.*

4 Cutting Between Valleys

In this section, we describe the second stage of the algorithm. Let $\mathcal{C}_i$ be a cell of $\mathcal{K}(\mathcal{P})$ constructed by DIVIDE-VERTICAL on which we call DIVIDE-VALLEY. The first stage of our algorithm ensures that $\mathcal{C}_i$ is convex and does not contain any reflex vertex in its interior. Let R_i denote the set of valleys that conflict with $\mathcal{C}_i$, and let r_i denote its cardinality. The *extended valley* e' corresponding to a valley $e \in R_i$ is the segment obtained by extending e until it meets the boundary $\partial\mathcal{C}_i$ of the cell. As $\mathcal{C}_i$ does not contain any motorcycle vertex in its interior, the extended valleys of $\mathcal{C}_i$ do not cross. So the extended valleys form an outerplanar graph with outer face $\partial\mathcal{C}_i$. (See Fig. 7.)

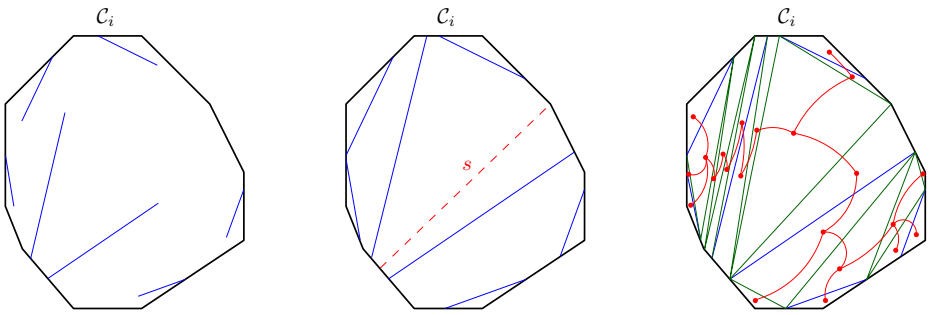


Fig. 7: (Left) The cell $\mathcal{C}_i$ and the conflicting valleys. (Middle) The extended valleys, and a balanced cut. (Right) The triangulation and its dual graph.

If $\mathcal{C}_i$ conflicts with at least one valley, we first construct a *balanced cut*, which is a chord s of $\partial\mathcal{C}_i$ such that there are at most $2r_i/3$ extended valleys on each side of s. (See Fig. 7, middle.) The existence and the algorithm for computing s are explained in the full version of this paper [5]. This balanced cut plays exactly the same role as the vertical edges $s_1, \ldots, s_q$ along the cutting line that were used in DIVIDE-VERTICAL. So we insert s as a new boundary segment, we compute its lifted version $\hat{s}$, and at each crossing between s and $\mathcal{S}'$, intersects the descent paths as new boundary edges.

We repeat this process recursively, and we stop recursing whenever a cell does not conflict with any valley. All the structural results in Sect. 3 still hold, except that now a cell is sandwiched between two balanced cuts, which can have arbitrary orientation, instead of the lines ℓ_i^- and ℓ_i^+.

So now we assume that we reach a leaf $\mathcal{C}_i$, which does not conflict with any valley. This cell $\mathcal{C}_i$ must be convex. As valleys are the only reflex edges of $\mathcal{T}$, its restriction $\hat{\mathcal{C}}_i$ above $\mathcal{C}_i$ is convex. Hence, it is the lower envelope of the supporting planes of its faces. These faces are obtained in $O(c_i)$ time from the face lists, and the lower envelope can be computed in $O(c_i \log c_i)$ time algorithm using any optimal 3D convex hull algorithm. We project $\hat{\mathcal{C}}_i$ onto the xy-plane and we obtain the restriction $\mathcal{S}'_i$ of $\mathcal{S}'$ to $\mathcal{C}_i$.

The analysis is similar as for the first stage, except that now the valleys play the role of the vertices of the motorcycle graphs. Our balanced cuts ensure that after each subdivision, the number of conflicting valleys drops by a factor at least $3/2$, so the depth of recursion is still $O(\log r)$. Theorem 1 follows.

Acknowledgments. We thank the anonymous referees for their helpful comments.

References

1. Aichholzer, O., Alberts, D., Aurenhammer, F., Gärtner, B.: A novel type of skeleton for polygons. Journal of Universal Computer Science 1(12), 752–761 (1995)

2. Barequet, G., Goodrich, M., Levi-Steiner, A., Steiner, D.: Straight-skeleton based contour interpolation. Proceedings of the 14th annual ACM-SIAM symposium on Discrete algorithms pp. 119–127 (2003)
3. Berg, M.d., Cheong, O., Kreveld, M.v., Overmars, M.: Computational Geometry: Algorithms and Applications. Springer-Verlag (2008)
4. Bowers, J.: Computing the straight skeleton of a simple polygon from its motorcycle graph in deterministic O(n log n) time. CoRR abs/1405.6260 (2014)
5. Cheng, S.W., Mencel, L., Vigneron, A.: A faster algorithm for computing straight skeletons. CoRR abs/1405.4691 (2014)
6. Cheng, S.W., Vigneron, A.: Motorcycle graphs and straight skeletons. Algorithmica 47(2), 159–182 (2007)
7. Chin, F., Snoeyink, J., Wang, C.A.: Finding the medial axis of a simple polygon in linear time. Discrete and Computational Geometry 21(3), 405–420 (1999)
8. Cloppet, F., Oliva, J., Stamon, G.: Angular bisector network, a simplified generalized voronoi diagram: Application to processing complex intersections in biomedical images. IEEE Transactions on Pattern Analysis and Machine Intelligence 22(1), 120–128 (2000)
9. Coquillart, S., Oliva, J., Perrin, M.: 3d reconstruction of complex polyhedral shapes from contours using a simplified generalized voronoi diagram. Computer Graphics Forum 15(3), 397–408 (1996)
10. Day, A., Laycock, R.: Automatically generating large urban environments based on the footprint data of buildings. Proceedings of the 8th ACM symposium on Solid Modeling and Applications pp. 346–351 (2003)
11. Demaine, E.D., Demaine, M.L., Lubiw, A.: Folding and cutting paper. Revised Papers from the Japan Conference on Discrete and Computational Geometry pp. 104–117 (1998)
12. Demaine, E.D., Demaine, M.L., Lubiw, A.: Folding and one straight cut suffice. Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms pp. 891–892 (1999)
13. Demaine, E.D., Demaine, M.L., Mitchell, J.S.B.: Folding flat silhouettes and wrapping polyhedral packages: New results in computational origami. Proceedings of the 15th Annual ACM Symposium on Computational Geometry pp. 105–114 (1999)
14. Eppstein, D., Erickson, J.: Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. Discrete and Computational Geometry 22(4), 569–592 (1999)
15. Felkel, P., Š. Obdržálek: Straight skeleton implementation. Proceedings of the 14th Spring Conference on Computer Graphics pp. 210–218 (1998)
16. Held, M., Huber, S.: Theoretical and practical results on straight skeletons of planar straight-line graphs. Proceedings of the 27th Symposium on Computational Geometry pp. 171–178 (2011)
17. Held, M., Huber, S.: A fast straight-skeleton algorithm based on generalized motorcycle graphs. International Journal of Computational Geometry and Applications 22(5), 471–498 (2012)
18. Hershberger, J.: Finding the upper envelope of n line segments in O(n log n) time. Information Processing Letters 33(4), 169–174 (1989)
19. Kelly, T., Wonka, P.: Interactive architectural modeling with procedural extrusions. ACM Transactions on Graphics 30(2), 14:1–14:15 (2011)
20. von Peschka, G.: Kotirte Ebenen: Kotirte Projektionen und deren Anwendung; Vorträge. Brno: Buschak and Irrgang (1877)
21. Vigneron, A., Yan, L.: A faster algorithm for computing motorcycle graphs. Proceedings of the 29th Symposium on Computational Geometry pp. 17–26 (2013)