

Motorcycle Graphs and Straight Skeletons*

Siu-Wing Cheng[†] Antoine Vigneron[†]

November 22, 2001

Abstract

We present a new algorithm to compute a motorcycle graph. It runs in $O(n\sqrt{n}\log n)$ time when n is the size of the input. We give a new characterization of the straight skeleton of a polygon possibly with holes. For a simple polygon, we show that it yields a randomized algorithm that reduces the straight skeleton computation to a motorcycle graph computation in $O(n\log^2 n)$ time. Combining these results, we can compute the straight skeleton of a non-degenerate simple polygon with n vertices, among which r are reflex vertices, in $O(n\log^2 n + r\sqrt{r}\log r)$ expected time. For a degenerate simple polygon, our expected time bound becomes $O(n\log^2 n + r^{17/11+\epsilon})$.

1 Introduction

In 1995 Aichholzer et al. [3] introduced a new kind of skeleton for a polygon. It is defined as the trace of the vertices when the initial polygon is shrunken, each edge moving at the same speed (see Fig. 1). As opposed to the widely used

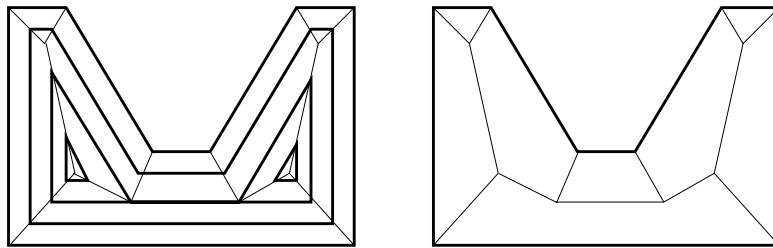


Figure 1: The straight skeleton (on the right) is obtained by shrinking the initial polygon

*The work described in this paper has been supported by the Research Grants Council of Hong Kong, China (Project no. HKUST 6088/99E).

[†]Dept. of Computer Science, Hong Kong University of Science & Technology, Clear Water Bay, Kowloon, Hong Kong. Email: {scheng|antoine}@cs.ust.hk.

medial axis [8], the straight skeleton has only straight line edges, which could be useful when parabolic edges need to be avoided, either because the application requires it or because the software library only handles polygonal figures. A direct application is to compute the roof of a building, knowing that its slope is constant. The projection of the edges of the roof to the horizontal plane is the straight skeleton of an horizontal section of the walls [3]. Another important application is to compute offset lines of a polygon [13] (note that the medial axis yields offset curves containing circle arcs). The straight skeleton may prove useful for other applications of the medial axis, for instance, shape recognition [4] and terrain reconstruction from a river network [2] as well. Recently, Demaine et al. found applications to computational origami [10, 11, 12]. The geometry of the straight skeleton is still not well understood and it is unclear whether the best known algorithm is close to optimal [13]. We contribute to this problem by finding new connections with motorcycle graphs [13] that allow us to design a faster randomized algorithm.

The first algorithm by Aichholzer et al. [3] computes the straight skeleton of a simple polygon with n vertices in $O(n^2 \log n)$ time by running a discrete simulation of the shrinking process. Later on, Aichholzer and Aurenhammer [2] generalized it to polygonal figures in the plane and brought the space complexity down to $O(n)$. They also show that the straight skeleton cannot be described as a lower envelope in a similar way as the medial axis. This explains why standard computational geometry techniques such as randomized incremental construction do not apply directly. In 1997, Eppstein and Erickson [13] gave the first sub-quadratic algorithm whose running time is $O(n^{17/11+\epsilon})$ in the worst case, with a similar space complexity. They also present a reflex sensitive algorithm that runs in $O(n^{1+\epsilon} + n^{8/11+\epsilon} r^{9/11+\epsilon})$ time, where r is the number of reflex (non-convex) vertices of the polygon.

The motorcycle graph problem was proposed by Eppstein and Erickson to capture the most difficult part of the construction of straight skeletons. The input consists of n motorcycles M_i , $1 \leq i \leq n$, and each M_i has an initial position and a fixed velocity. At time 0, all motorcycles move from their initial positions at their fixed velocities. If a motorcycle M_j meets the track left by another motorcycle M_i , then M_j crashes and cannot move any further. If two motorcycles collide, both of them crash and cannot move any more. When all motorcycles have either crashed or moved to infinity, their tracks form a planar graph which is the motorcycle graph (see Fig. 2). Eppstein and Erickson [13] solve the motorcycle graph problem in $O(n^{17/11+\epsilon})$ time, using advanced data structures for maintaining pairwise interaction and for ray shooting.

This work has several contributions. First, we present an algorithm to compute a motorcycle graph that runs in $O(n\sqrt{n} \log n)$ time. It is faster and simpler than the previous best known algorithm [13]. We also present a simple randomized algorithm with same running time on average. Second, we give a new characterization of the straight skeleton of a polygon (possibly with holes). Third, we present an algorithm that computes the straight skeleton of a simple polygon in $O(n \log^2 n)$ expected time from the motorcycle graph induced by its reflex vertices. Our results formalize the idea of Eppstein and Erickson that

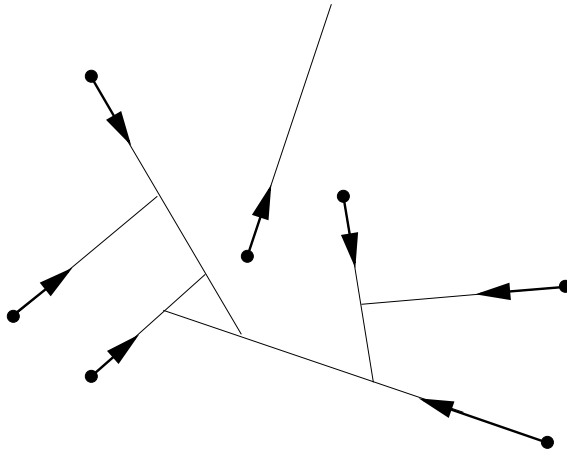


Figure 2: A motorcycle graph

the motorcycle graph problem captures the most difficult part of the construction of a straight skeleton. Putting everything together, we can compute in $O(n \log^2 n + r\sqrt{r} \log r)$ expected time the straight skeleton of a non-degenerate simple polygon. For a degenerate simple polygon, our expected time bound raises to $O(n \log^2 n + r^{17/11+\epsilon})$. As a comparison, the previous best known algorithm by Eppstein and Erickson [13] is slower, but it is deterministic and can handle polygons with holes.

2 Computing a motorcycle graph

A simple approach to compute a motorcycle graph is first to build a list of potential crashes, by considering each pair of motorcycles independently of the rest. Note that there is a quadratic number of potential crashes, but only a linear number actually occur. Then the motorcycle graph can be drawn in chronological order of its crashes by scanning the list of potential crashes in chronological order. This algorithm can easily be implemented to run in $O(n^2 \log n)$ time using a priority queue.

Our algorithm is similar to this simple event-queue algorithm in that we also track the crashes of motorcycles in chronological order. The main difference is that we introduce new events to confine our search. We chose an appropriate partition of the plane (an $1/\sqrt{n}$ -cutting, see Section 2.1, or the partition induced by an $1/\sqrt{n}$ random sample of the support lines, see Section 2.5). Then we run the simple event queue algorithm simultaneously in all the regions. We generate an event each time a motorcycle enters a region, at this point the motorcycle is inserted in the simulation of the new region. As we shall see, this yields an $O(n\sqrt{n} \log n)$ time algorithm for computing a motorcycle graph.

2.1 Preliminaries.

Let p_i and \vec{v}_i denote the initial position and velocity of the motorcycle M_i . The *trajectory* T_i of M_i is the infinite ray that emits from p_i in the direction of \vec{v}_i . The *support line* L_i of M_i is the line containing T_i . At any time t , $T_i(t)$ denotes the point $p_i + t\vec{v}_i$. Note that M_i may crash before reaching $T_i(t)$. A *crossing point* is $T_i \cap T_j$ for some motorcycles M_i and M_j . If no motorcycle M_j reaches the crossing point $T_i \cap T_j$ earlier than M_i , then M_i moves to infinity in the motorcycle graph. Otherwise, M_i crashes at the first crossing point $T_i \cap T_j$ such that M_j reaches it before M_i .

A key structure is a $(1/\sqrt{n})$ -cutting. Given n lines, a $(1/\sqrt{n})$ -cutting is a partition of the plane into disjoint triangular cells (possibly unbounded) such that the interior of each cell intersects at most \sqrt{n} lines. Cuttings have been studied extensively, for example, see [1, 5, 6, 17, 18]. We will employ a deterministic algorithm presented by Chazelle [5] that runs in $O(n\sqrt{n})$ time. It produces a cutting of $O(n)$ size and gives the lines intersecting each cell of the cutting.

Let \mathcal{K} be a $(1/\sqrt{n})$ -cutting of the support lines of the motorcycles. We will simulate the movements of motorcycles within \mathcal{K} . During the simulation, a motorcycle M_i is *active* in a cell \mathcal{C} of \mathcal{K} at time t if M_i is in \mathcal{C} at time t , or if M_i has been in \mathcal{C} before time t . Intuitively, M_i can interact with other motorcycles within \mathcal{C} only if it is currently in \mathcal{C} or if it has left a track in \mathcal{C} before, this is why we call it active in this situation.

The simulation will progress in chronological order of two kinds of events.

1. A *switching event* (i, \mathcal{C}, t) happens at the earliest time t such that $T_i(t)$ lies on the boundary of the cell \mathcal{C} (i.e., the first intersection between T_i and \mathcal{C}).
2. An *impact event* (i, j, t) happens at time t when $T_i(t) = T_i \cap T_j = T_j(t')$ for some time $t' \in [0, t]$.

All the switching events will be generated in initialization before the simulation starts. We cannot generate all the impact events as there can be a quadratic number of them. We will generate a subset good enough for our purpose on the fly. To do so, we maintain a *local arrangement* $\mathcal{A}(\mathcal{C})$ for each cell \mathcal{C} in \mathcal{K} . $\mathcal{A}(\mathcal{C})$ is the arrangement of line segments $L_i \cap \mathcal{C}$ for all motorcycles M_i currently active in \mathcal{C} .

To simplify the presentation, we first assume that no two trajectories are collinear and that if a support line intersects a cell, it intersects the interior of the cell. The handling of degenerate cases will be discussed in Section 2.4.

2.2 Algorithm.

We first compute \mathcal{K} in $O(n\sqrt{n})$ time. We then initialize an empty event-queue \mathcal{Q} . The switching events can be obtained by computing the intersections between \mathcal{K} and the trajectories of the motorcycles. There are $O(n\sqrt{n})$ such

intersections and they can be computed in $O(n\sqrt{n})$ time [5]. We insert the corresponding switching events into \mathcal{Q} . Next, we generate the first batch of impact events. For each cell \mathcal{C} in \mathcal{K} , we collect the motorcycles whose initial positions reside in \mathcal{C} and compute $\mathcal{A}(\mathcal{C})$. Each vertex of $\mathcal{A}(\mathcal{C})$ is $L_i \cap L_j$ for some i and j . If $L_i \cap L_j = T_i \cap T_j$, then we compute t and t' such that $T_i(t) = T_j(t') = T_i \cap T_j$. If $t \geq t'$, then we insert the impact event (i, j, t) into \mathcal{Q} . If $t' \geq t$, then we insert the impact event (j, i, t') into \mathcal{Q} (see Fig. 3). By the property of cutting, the total size of the local arrangements during this

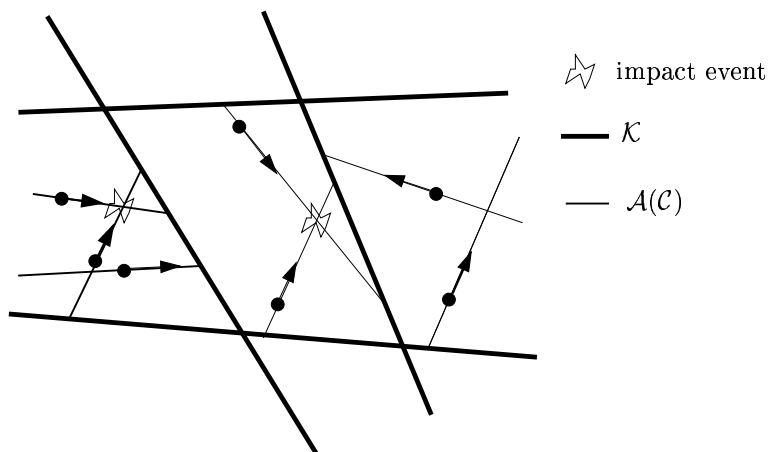


Figure 3: The initialization step. Two impact events are queued here.

initialization phase is $O(n\sqrt{n})$, so it can be performed in $O(n\sqrt{n} \log n)$ time.

In the main loop of the algorithm, we repeatedly extract from \mathcal{Q} and process the event e with the smallest time stamp. If e is a switching event (i, \mathcal{C}, t) , then we update the local arrangement $\mathcal{A}(\mathcal{C})$ by inserting the line segment $L_i \cap \mathcal{C}$. For all the new vertices in $\mathcal{A}(\mathcal{C})$, we compute the associated impact events and insert them into \mathcal{Q} . Otherwise, if e is an impact event (i, j, t) , then M_i crashes at $T_i \cap T_j$ at time t , so we insert the edge connecting p_i and $T_i \cap T_j$ into the motorcycle graph, and delete from \mathcal{Q} all the switching events of M_i and all the impact events involving M_i that occur at points on T_i outside the segment $[p_i, T_i \cap T_j]$.

The following pseudo-code describes our motorcycle graph algorithm.

Algorithm *motorcycle_graph()*

1. /* initialization */
2. compute \mathcal{K}
3. insert all the crossing events into \mathcal{Q}
4. for all \mathcal{C} in \mathcal{K} , compute $\mathcal{A}(\mathcal{C})$ at time $t = 0$
5. for all \mathcal{C} in \mathcal{K} , insert the impact events corresponding to vertices of $\mathcal{A}(\mathcal{C})$ into \mathcal{Q}

```

6. /* main loop */
7. while  $\mathcal{Q}$  is not empty
8.   do extract the next event from  $\mathcal{Q}$ ;
9.   if the next event is a switching event  $(i, \mathcal{C}, t)$ 
10.    then /*  $M_i$  enters the cell  $\mathcal{C}$  at time  $t$ . */
11.     insert  $L_i \cap \mathcal{C}$  into  $\mathcal{A}(\mathcal{C})$ ;
12.     for each vertex of  $\mathcal{A}(\mathcal{C})$  on  $L_i$  that is equal to  $T_i \cap T_j$  for
        some  $j$ 
13.      do compute  $t_i$  and  $t_j$  such that  $T_i(t_i) = T_j(t_j) = T_i \cap T_j$ ;
14.      if  $t_i \geq t_j$ 
15.       then insert the impact event  $(i, j, t_i)$  into  $\mathcal{Q}$ ;
16.      if  $t_j \geq t_i$ 
17.       then insert the impact event  $(j, i, t_j)$  into  $\mathcal{Q}$ ;
18.   if the next event is an impact event  $(i, j, t)$ 
19.    then /*  $M_i$  crashes at  $T_i \cap T_j$  at time  $t$ . */
20.     insert the edge connecting  $p_i$  and  $T_i \cap T_j$  into the motorcycle
        graph;
21.     delete from  $\mathcal{Q}$  all switching events for  $M_i$  (i.e., switching
        events that match  $(i, *, *)$ );
22.     delete from  $\mathcal{Q}$  all impact events involving  $M_i$  that occur at
        points on  $T_i$  outside the interval between  $p_i$  and  $T_i \cap T_j$ 

```

2.3 Analysis.

Correctness follows from the fact that the movements of the motorcycles are simulated in chronological order, and that once a motorcycle crashes, irrelevant switching and impact events involving the motorcycle are deleted.

As explained before, there are $O(n\sqrt{n})$ switching events and they can be found in $O(n\sqrt{n})$ time. So initializing \mathcal{Q} with the switching events takes $O(n\sqrt{n} \log n)$ time. This also bounds the total time spent on extracting and deleting switching events during the simulation. It remains to bound the total time spent on updating the local arrangements of the cells as well as inserting and deleting impact events.

We maintain a local arrangement in a doubly connected edge list where each cell is associated with a search structure allowing split operations in logarithmic time (for instance a balanced binary tree [21]). Thus, updating a local arrangement can be done in $O(\log n)$ per new vertex generated. Each impact event corresponds to a vertex of some local arrangement and inserting or deleting an impact event clearly takes $O(\log n)$ time. We keep a dictionary of impact events involving M_i sorted along T_i . Thus, when M_i crashes, we can identify in $O(\log n)$ time the range of impact events along T_i which are to be deleted. So it suffices to bound the total size of the local arrangements at the end of the simulation.

For all motorcycle M_i , the cell of \mathcal{K} that contains its starting point p_i is denoted by \mathcal{C}_i and the cell where M_i crashes is denoted by \mathcal{C}'_i . Each vertex of a local arrangement $\mathcal{A}(\mathcal{C})$ is $L_i \cap L_j$ for some i and j . We charge this vertex to the

motorcycle M_i (resp. M_j) if it lies within $C_i \cup C'_i$ (resp. $C_j \cup C'_j$). Note that a vertex may be charged twice, now we want to prove that each vertex is charged at least once. Let $v = L_i \cap L_j$ be a vertex of the local arrangement of a cell \mathcal{C} , if $\mathcal{C} = C_i$ or $\mathcal{C} = C_j$ then it is charged to M_i or M_j . Otherwise, $v = T_i \cap T_j$ and therefore, M_i and M_j cannot both cross v . On the other hand, both M_i and M_j are active in \mathcal{C} , so they must have entered \mathcal{C} at some point of the simulation. Therefore, M_i or M_j (or both) crashes within \mathcal{C} , and thus v has been charged to M_i or M_j .

So each motorcycle will be charged with intersections on its support line in the first and last cells that contain the motorcycle in the simulation, and each vertex of a local arrangement is charged once or twice. Since at most \sqrt{n} support lines intersects a cell, each motorcycle is charged at most $2\sqrt{n}$ times. So the total size of the local arrangements at the end of the simulation is $O(n\sqrt{n})$. It follows that we spend $O(n\sqrt{n} \log n)$ time on updating the local arrangements of the cells as well as inserting and deleting impact events.

Theorem 1 *Given the initial positions and velocities of n motorcycles, the motorcycle graph can be computed in $O(n\sqrt{n} \log n)$ time.*

2.4 Degenerate cases.

If several motorcycles are allowed to share the same support line L , then there may be a linear number of motorcycles in the same cell. Note that it does not increase the size of the local arrangements. Moreover, a motorcycle M_i whose trajectory T_i crosses L can be possibly involved in only two crashes along L , namely crashes with motorcycles M_j and M_k such that $[p_j, p_k]$ contains $L \cap L_i$. This means that, with simple modifications, our algorithm can handle aligned motorcycles within the same time bound.

Another type of degeneracy that we did not handle is when motorcycles reaches a vertex or follows an edge of \mathcal{K} . In this case, we need to maintain information about the status of vertices and edges of \mathcal{K} across the simulation. If a motorcycle reaches a vertex v of \mathcal{K} , any other motorcycle reaching v afterwards will crash there, so there is only a linear number of such event. We can handle them by to maintaining one flag per vertex of \mathcal{K} . An edge of \mathcal{K} , on the other hand, may contain several motorcycles moving at the same time, we can get around this problem by cutting each edge of \mathcal{K} at each initial point p_i it contains. The resulting sub-edges can contain at most two motorcycle at any time, which allows to record their status and handling they are involved in without hurting our time bounds.

2.5 Further remarks.

Our algorithm is simple and implementable, and, using practical planar cuttings algorithm [14], we expect it to beat the naive $O(n^2 \log n)$ algorithm for instances of a few hundreds motorcycles. However, the most promising approach in practice is the following randomized version of our algorithm, with the same

running time in expectation: take a random sample of \sqrt{n} support lines whose arrangement will play the role of our cutting. The analysis of this algorithm is straightforward.

We also note that our algorithms handle without any difficulty the case where a motorcycle may run out of fuel at some point and stop, and the case where the speed of a motorcycle can vary (but it cannot move backwards). As a comparison, Eppstein and Erickson [13] algorithm can handle dynamic insertion of motorcycles, but it requires a constant speed for each motorcycle.

3 Geometry of the straight skeleton

In this section, we present a new characterization of the straight skeleton of a polygon (possibly with holes). This characterization reduces the construction of the straight skeleton to the construction of a motorcycle graph and a lower envelope. There are two implications. First, it justifies that the motorcycle graph problem is indeed the most difficult part of the straight skeleton construction. Second, it allows us to develop a faster algorithm for constructing the straight skeleton of a simple polygon (see section 4).

3.1 Definition

For convenience, we place ourselves in a 3 dimension euclidean space, and use the standard convention that the height of a point is its third coordinate z and the two other coordinates are denoted by x and y . We consider a polygon \mathcal{P} , possibly with holes, lying in the horizontal plane $z = 0$.

The straight skeleton \mathcal{S} of \mathcal{P} is a straight line graph embedded in the interior of \mathcal{P} , each vertex of \mathcal{P} being adjacent to an edge of \mathcal{S} . It is defined by means of a shrinking process [2, 3]. The edges of \mathcal{P} move towards its interior, at unit speed, remaining parallel to their initial position (see Fig. 1 and Fig. 5 c). The traces of the vertices of the polygon during this shrinking process form the edges of the straight skeleton. The straight skeleton \mathcal{S} induces a subdivision of \mathcal{P} which we denote by $\mathcal{K}(\mathcal{S})$.

During the shrinking process, two main type of events may occur that change the combinatorial structure of the polygon. First, the length of an edge may decrease to 0, and thus this edge disappears from the shrinking polygon (see Fig. 4). These events are called *edge events*. Second, a vertex may hit an edge which splits the polygon into two parts (see Fig. 4). These events are called *split events*. Other kinds of event may occur in degenerate cases, for instance several edge or vertex events can take place simultaneously, or two reflex vertices can collide. These events will be considered in the full version of this paper, here we will assume that only vertex and split events occur and that they occur one at a time.

Another way to look at the shrinking process is to consider time as a third dimension, which means that the shrinking polygon also moves vertically at unit

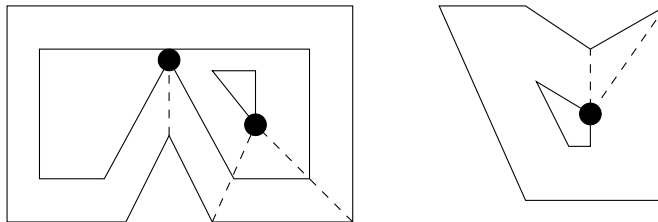


Figure 4: On the left, a split event and an edge event. On the right, an edge event involving a reflex vertex.

speed, drawing a terrain \mathcal{T} in three dimension (see Fig. 5 b). Then \mathcal{S} is the vertical projection of the edges of the terrain \mathcal{T} .

The edges and faces of \mathcal{T} are the lifted versions of the edges and faces of $\mathcal{K}(\mathcal{S})$. Each face of \mathcal{T} makes an angle $\pi/4$ with horizontal. A horizontal cut of \mathcal{T} at height $z = t$ is the shrunken version of \mathcal{P} at time t (see Fig. 5 c,d). We call the reflex edges of \mathcal{T} *valleys*. (See Fig. 6 c, here there are only two valleys and they are adjacent to the reflex vertices of \mathcal{P} .) So the edges of \mathcal{S} corresponding to valleys are the traces of the reflex vertices in the shrinking process. Note that our general position assumption implies that no two reflex vertices collide, and so no new reflex vertex is produced during the shrinking process. It follows that all the valleys of \mathcal{T} are adjacent to reflex vertices of \mathcal{P} .

3.2 Other characterizations

Eppstein and Erickson [13] expressed \mathcal{T} as the lower envelope of a collection of slabs making angle $\pi/4$ with horizontal. Each edge e of \mathcal{P} defines an *edge slab*, bounded below by e and on the sides by rays perpendicular to e . Each reflex vertex v incident to the edges e and e' defines two *reflex slabs*. One reflex slab is bounded below by the valley incident to v and bounded on the sides by rays perpendicular to e (see Fig. 6 a,c). The definition of the other reflex slab is similar with e replaced by e' .

Theorem 2 (Eppstein and Erickson) *The terrain \mathcal{T} is the restriction of the lower envelope of the edge slabs and the reflex slabs to the space vertically above the polygon.*

We give a new characterization similar to Theorem 2, except that we do not need to know the valleys of \mathcal{T} to define the slabs, but only the motorcycle graph induced by the reflex vertices of \mathcal{P} . Each reflex vertex of \mathcal{P} is associated with a motorcycle whose velocity is the velocity of the reflex vertex in the shrinking process that generates the straight skeleton (this speed is the inverse of the sine of half the exterior angle at the reflex vertex). Each motorcycle runs out of fuel when it meets the boundary of \mathcal{P} . We denote by \mathcal{G} the motorcycle graph of this set of motorcycles. In this section we will assume that \mathcal{P} is non-degenerate in

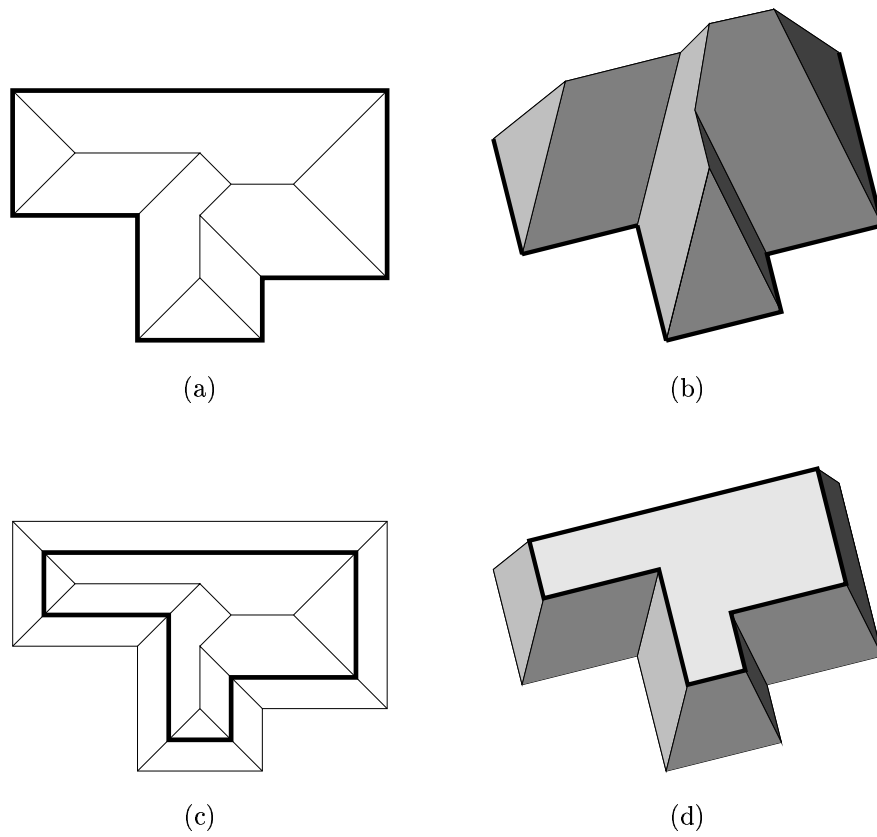


Figure 5: (a) The input polygon \mathcal{P} (thick lines), and its straight skeleton \mathcal{S} (inside). (b) The terrain \mathcal{T} obtained by lifting the subdivision $\mathcal{K}(\mathcal{S})$. (c) The shrunken polygon at time t is a horizontal section of \mathcal{T} at height t . (d) The terrain \mathcal{T}_t is the restriction of \mathcal{T} to the vertical interval $[0, t]$, its top face is the shrunken version of \mathcal{P} at time t .

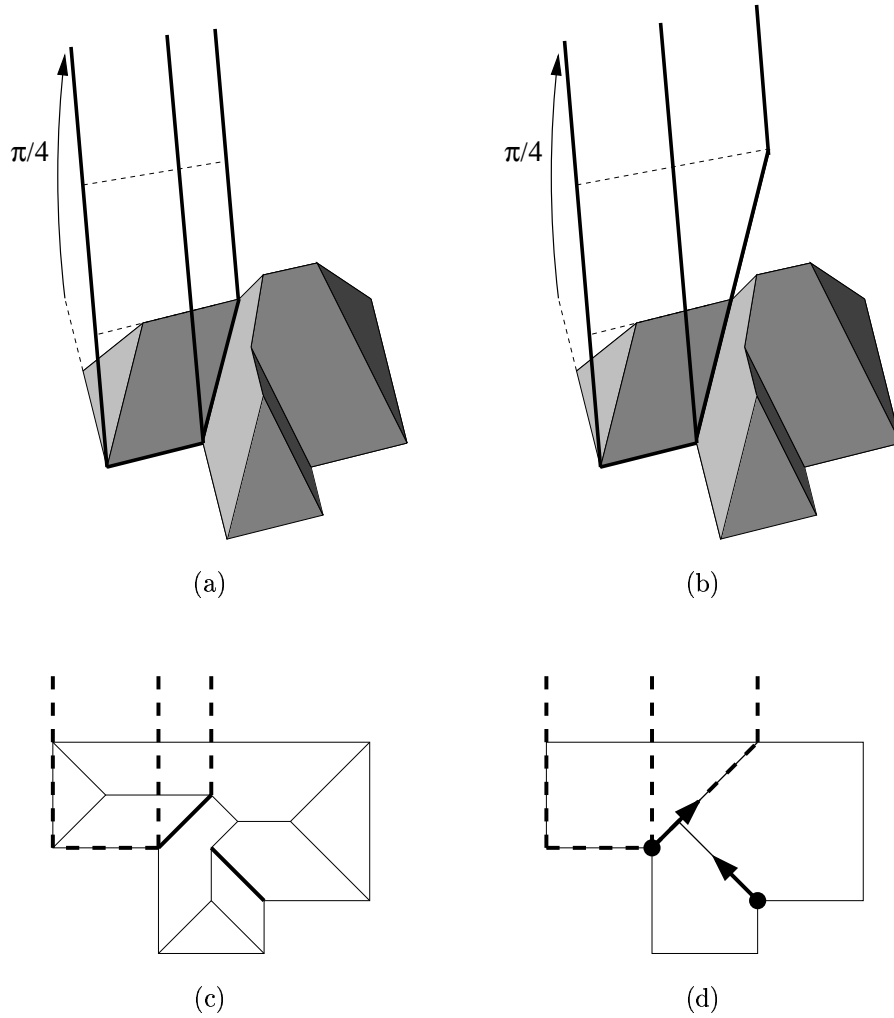


Figure 6: An illustration of the different kinds of slabs we defined, here we only draw the slabs associated with one particular edge that is adjacent to a reflex vertex. (a) The edge slab and the reflex slab. (b) The edge slab and the motorcycle slab. (c) The edge slab, the reflex slab and the two valleys of \mathcal{T} seen from above. (d) The motorcycle graph \mathcal{G} associated with \mathcal{P} and the boundaries of the edge slab and the motorcycle slab seen from above. Note that the edges of $\widehat{\mathcal{G}}$ are longer than the associated valleys, and thus a motorcycle slab contains the associated reflex slab.

the sense that no two motorcycles of \mathcal{G} collide, that is, no two motorcycles reach the same point at the same time.

We lift \mathcal{G} to 3D to obtain $\widehat{\mathcal{G}}$, where the height of a point of $\widehat{\mathcal{G}}$ is the time when the corresponding point in \mathcal{G} is reached by the motorcycle. For each edge e of \mathcal{G} , we denote its lifted version by \widehat{e} . At the neighborhood of the reflex vertices, the edges of $\widehat{\mathcal{G}}$ follow valleys of \mathcal{T} . For each reflex vertex $v = e \cap e'$, we define two *motorcycle slabs* making an angle $\pi/4$ with horizontal. One motorcycle slab is bounded below by the edge of $\widehat{\mathcal{G}}$ incident to v and bounded on the sides by rays perpendicular to e (see Fig. 6 b,d). The definition of the other motorcycle slab is similar with e replaced by e' . In the following, we prove that the terrain \mathcal{T} is the lower envelope of edge slabs and motorcycle slabs.

Lemma 1 *For each reflex vertex, the incident valley is shorter than the incident edge in $\widehat{\mathcal{G}}$ (see Fig. 6).*

Proof: Suppose that the lemma is false. So there exists an edge of $\widehat{\mathcal{G}}$ that is shorter or equal to its corresponding valley. Among such edges of $\widehat{\mathcal{G}}$, we choose the edge \widehat{e} whose higher endpoint is lowest. Let t be the height of the higher endpoint of \widehat{e} . We restrict \mathcal{T} to the height interval $[0, t]$ by replacing the parts above the height t with flat patches (see Fig. 5 d). We denote the restriction by \mathcal{T}_t . By minimality of t , no valley of \mathcal{T}_t is longer than its corresponding edge in $\widehat{\mathcal{G}}$.

Since \widehat{e} is not longer than its valley, it does not reach the boundary of \mathcal{P} , so e crashes into an edge f of \mathcal{G} . Let S be the vertical slab with base f . By the definition of $\widehat{\mathcal{G}}$, the higher endpoint of \widehat{e} is directly above \widehat{f} . No valley of \mathcal{T}_t can stab S , otherwise such a valley is not shorter than its corresponding edge in $\widehat{\mathcal{G}}$, a contradiction. Since valleys are the only reflex edges on \mathcal{T}_t , $\mathcal{T}_t \cap S$ is a convex chain. Since \widehat{f} is tangent to \mathcal{T}_t at its lower endpoint, \widehat{f} is above $\mathcal{T}_t \cap S$. This contradicts the fact that the higher endpoint of \widehat{e} is directly above \widehat{f} . \square

Lemma 2 *Each point of $\widehat{\mathcal{G}}$ is on or above \mathcal{T} .*

Proof: By Lemma 1, no edge of \mathcal{G} crosses the projection of a valley to the horizontal plane. So for any edge e of \mathcal{G} , the intersection of \mathcal{T} with the vertical slab with base e is a convex chain. Since \widehat{e} is tangent to \mathcal{T} at the lower endpoint of \widehat{e} , \widehat{e} is on or above \mathcal{T} . \square

We are now ready to prove our characterization. Here \mathcal{P} is non-degenerate in the sense that no two motorcycles collide in its associated motorcycle graph \mathcal{G} .

Theorem 3 *A non-degenerate 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.*

Proof: Let ν be a valley. Let \widehat{e} be its corresponding edge in $\widehat{\mathcal{G}}$. Let $S(\nu)$ denote the union of reflex slabs bounded below by ν . Let $S(\widehat{e})$ denote the union

of motorcycle slabs bounded below by \hat{e} . Lemma 1 implies that $S(\nu) \subseteq S(\hat{e})$. By Theorem 2, it suffices to prove that each point in $S(\hat{e}) \setminus S(\nu)$ is on or above \mathcal{T} . Consider a point p in $\hat{e} \setminus \nu$. Let r be a halfline that starts at p , has unit slope, and lie on a motorcycle slab bounded below by \hat{e} . Let H_r be the halfplane obtained by sweeping r upward and downward to infinity. By Lemma 2, p is on or above \mathcal{T} . \mathcal{T} intersects H_r at a polygonal chain. By Theorem 2, each segment of this polygonal chain has slope with absolute value at most 1. Thus, each point on r is on or above \mathcal{T} . \square

4 Computing the straight skeleton

In this section, we show how to compute the straight skeleton \mathcal{S} of a simple polygon \mathcal{P} after computing the motorcycle graph of the reflex vertices \mathcal{G} . Our algorithm runs in $O(n \log^2 n)$ expected time.

Using directly Theorem 3, we do not know of a lower envelope algorithm that would yield a near-linear running time. However, we can compute in near-linear time a single face of the terrain or a vertical section since it reduces to computing a lower envelope in two dimension. As we shall see later, it allows us to compute the section of the skeleton by a line through a random internal node, we will use this result to design an efficient divide-and-conquer algorithm.

An *unrooted binary tree* is a tree whose nodes have degree one or three, the nodes with degree one are called leaves and the nodes with degree three are called internal nodes. In this section we assume that \mathcal{P} is non-degenerate in the sense that no two motorcycles of \mathcal{G} collide and \mathcal{S} is an unrooted binary tree. By Lemma 1, it implies that no two reflex vertices collide in the shrinking process and therefore all valleys of \mathcal{T} are adjacent to reflex vertices of \mathcal{P} .

4.1 Canonical partition.

Aichholzer et al. [3] showed that the terrain \mathcal{T} has the gradient property, that is, starting at any point of \mathcal{T} in the face adjacent to an edge e of \mathcal{P} , and following the path of steepest descent, we eventually reach the edge e . This also follows directly from the characterizations of \mathcal{T} by means of slabs (theorems 2 and 3): if the starting point lies in the edge slab of e , the path of steepest descent leads directly to e , if it lies in the reflex slab of e (or equivalently its motorcycle slab), the path first reaches a valley that eventually leads to e . The paths of steepest descent have two nice properties. First, two paths of steepest descent cannot cross (but they may merge at some point). Second, a path of steepest descent lies inside (the closure of) a face of \mathcal{T} .

Let p be a point in \mathcal{S} . Let \hat{p} be the corresponding point on \mathcal{T} . The point p is a *ridge point* if \hat{p} does not lie in the interior of a reflex edge (valley). Given a set E of ridge points, for each $p \in E$, \hat{p} defines two or three paths of steepest descent on \mathcal{T} . The projections of the descent paths for all points in E subdivide \mathcal{P} into a collection of cells. This collection of cells is called the *canonical partition of \mathcal{P} induced by E* (see Fig. 7). If E is empty, we take the interior of \mathcal{P} to be

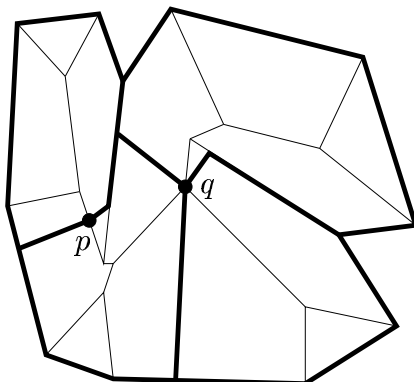


Figure 7: The canonical partition induced by $\{p, q\}$.

the only cell in the canonical partition. Canonical partitions can be recursively constructed. Let \mathcal{C} be a cell in the canonical partition of \mathcal{P} induced by a set E_1 . If E_2 is a set of ridge points in \mathcal{C} , then we can construct the *canonical partition of \mathcal{C} induced by E_2* in the same manner as described previously. The further subdivision of \mathcal{C} yields the canonical partition of \mathcal{P} induced by $E_1 \cup E_2$. This property follows from the fact that two descent paths do not cross.

Unless stated otherwise, we will always consider cells of a canonical partition to be open. In particular, it means that for any canonical cell \mathcal{C} , $\mathcal{S} \cap \mathcal{C}$ is an unrooted binary tree whose external edges are half-open. $\mathcal{S} \cap \mathcal{C}$ subdivides \mathcal{C} into several faces. That is, we get a planar subdivision and we denote it by $\mathcal{K}(\mathcal{S} \cap \mathcal{C})$.

4.2 Implicit representation of $\mathcal{K}(\mathcal{S} \cap \mathcal{C})$.

We describe an implicit representation $D_{\mathcal{C}}$ of $\mathcal{K}(\mathcal{S} \cap \mathcal{C})$ for any cell \mathcal{C} in any canonical partition. Note that we have not computed $\mathcal{S} \cap \mathcal{C}$ yet.

$D_{\mathcal{C}}$ stores a circular list $faces(\mathcal{C})$ that implicitly represents the faces of $\mathcal{K}(\mathcal{S} \cap \mathcal{C})$ as follows. For each face of $\mathcal{K}(\mathcal{S} \cap \mathcal{C})$, its lifted version on \mathcal{T} is contained in one edge slab or one edge slab and one motorcycle slab. We call them the *defining slab(s)* of the face. Each face is represented in $faces(\mathcal{C})$ by its defining slab(s). The ordering of faces in $faces(\mathcal{C})$ is the same as their ordering around the boundary of \mathcal{C} . We denote by $n_{\mathcal{C}}$ the number of faces of $\mathcal{K}(\mathcal{S} \cap \mathcal{C})$. Each face is assigned an index in $[1 \dots n_{\mathcal{C}}]$ consistent with the ordering in $faces(\mathcal{C})$. The boundary edges of \mathcal{C} are stored in order in a list $edges(\mathcal{C})$. For each edge e in $edges(\mathcal{C})$, we keep a face pointer to the face in $faces(\mathcal{C})$ that e bounds. Each face in $faces(\mathcal{C})$ also stores edge pointers to its bounding edges in $edges(\mathcal{C})$. Note that each face has at most five bounding edges in $edges(\mathcal{C})$.

Property 1 *Let \mathcal{L} be the lower envelope of the defining slabs of faces in $faces(\mathcal{C})$. The restriction to \mathcal{C} of the vertical projection of the edges of \mathcal{L} is $\mathcal{S} \cap \mathcal{C}$.*

At the top level, there is only one cell which is the interior of \mathcal{P} itself. $D_{\mathcal{P}}$ can easily be initialized in $O(n)$ time by walking around the boundary of \mathcal{P} once. During divide-and-conquer, we will need to subdivide a cell \mathcal{C} further with respect to a set E of ridge points inside \mathcal{C} . We assume that E is given and the following information has been computed.

- The descent paths for each point in E .
- Pointer to the edge in $edges(\mathcal{C})$ that each descent path leads to.
- Pointer to the face in $faces(\mathcal{C})$ intersected by each descent path.

We call the above three records the *partition information of \mathcal{C} induced by E* . Given this information, we can overlay the projection of these descent paths on \mathcal{C} to subdivide \mathcal{C} into $O(|E|)$ smaller cells \mathcal{C}_i . By walking around the boundary of each \mathcal{C}_i , we can compute $faces(\mathcal{C}_i)$ and $edges(\mathcal{C}_i)$. Using standard splitting and concatenation of lists, the total time needed for this is $O(\sum_i |\mathcal{C}_i|) = O(n_{\mathcal{C}} + |E|)$.

Lemma 3 *Given the data structure $D_{\mathcal{C}}$ for a cell \mathcal{C} and the partition information of \mathcal{C} induced by a set E of ridge points, the data structures $D_{\mathcal{C}_i}$ of the cells \mathcal{C}_i 's in the subdivision of \mathcal{C} induced by E can be computed in $O(n_{\mathcal{C}} + |E|)$ time.*

4.3 The divide step.

Our strategy is to divide the problem by, first, taking a line L parallel to the y -axis that passes through a random internal node of the skeleton $\mathcal{S} \cap \mathcal{C}$, and then building the canonical partition induced by a carefully chosen subset of $\mathcal{S} \cap \mathcal{C} \cap L$. Here we show how to find an internal node without knowing the whole skeleton, and we show how to perform the division.

Lemma 4 *Given $D_{\mathcal{C}}$ and a face f in $faces(\mathcal{C})$, the explicit representation of f can be computed in $O(n_{\mathcal{C}} \log n_{\mathcal{C}})$ time. The output includes, for each vertex of this face, pointers to its three defining faces in $faces(\mathcal{C})$.*

Proof: Let \hat{f} be the lifted version of f on \mathcal{T} . We compute \hat{f} and then its projection. We retrieve the defining slab(s) for f . Consider the case where there is one defining slab S of f (the case where there are two can be handled similarly). We first intersect S with the other defining slabs in $faces(\mathcal{C})$ by brute force in $O(n_{\mathcal{C}})$ time. This produces $O(n_{\mathcal{C}})$ line segments on S . Then we compute the lower envelope of these line segments on S in $O(n_{\mathcal{C}} \log n_{\mathcal{C}})$ time [15]. By Property 1, the region in $S \cap \mathcal{C}$ below this lower envelope is \hat{f} . We project this lower envelope onto the plane. We use the edge pointers associated with f in $faces(\mathcal{C})$ to locate the edge e in $edges(\mathcal{C})$ that bounds f and lies on the boundary of \mathcal{P} . We initiate a traversal along the boundary of \mathcal{C} from each endpoint of e to find the two intersections with the projected lower envelope. We trim the projected lower envelope at these two intersections. The trimmed projection and the edge e form the boundary of f . \square

Here we show how we associate a vertex $v(f)$ with each face f in $\mathcal{K}(\mathcal{S} \cap \mathcal{C})$. We root $\mathcal{S} \cap \mathcal{C}$ at its *centroid*. The centroid is a vertex whose removal produces subtrees each of size at most half the size of $\mathcal{S} \cap \mathcal{C}$ (see Fig.8). There may be two centroids, in which case they are adjacent, and we can take any one of them as the root, for instance we chose the centroid whose coordinates are smallest in lexical order. In the rooted $\mathcal{S} \cap \mathcal{C}$, edges are directed from a child to its parent. The rooted $\mathcal{S} \cap \mathcal{C}$ is almost a binary tree, except that the root has three children. For each face f of $\mathcal{K}(\mathcal{S} \cap \mathcal{C})$, we define $v(f)$ to be the vertex of f closest to the root of $\mathcal{S} \cap \mathcal{C}$. Since each non-root internal node u of the rooted $\mathcal{S} \cap \mathcal{C}$ has two children, u is $v(f)$ for exactly one face f of $\mathcal{K}(\mathcal{S} \cap \mathcal{C})$.

Lemma 4 constructs an explicit representation of f . We show how to compute $v(f)$ without knowing other parts of $\mathcal{S} \cap \mathcal{C}$.

Lemma 5 *Let f be a face of $\mathcal{K}(\mathcal{S} \cap \mathcal{C})$. Suppose that we are given an explicit representation of f and for each vertex of f , we are given the pointers to its defining faces in $\text{faces}(\mathcal{C})$. Then we can compute $v(f)$ in $O(n_{\mathcal{C}})$ time.*

Proof: It follows from definition that the two adjacent vertices of $v(f)$ on the boundary of f are children of $v(f)$. Moreover, this condition does not hold for other vertices of f . So it suffices to test this condition. Take a boundary edge e of f that is not a boundary edge of \mathcal{C} . We are given the pointer to the other face f' of $\mathcal{K}(\mathcal{S} \cap \mathcal{C})$ that e is incident to. We retrieve the indices of f and f' in $D_{\mathcal{C}}$. The difference between the two indices modulo $n_{\mathcal{C}}$ tells us the sizes of the two subtrees obtained if we remove e . The root of $\mathcal{S} \cap \mathcal{C}$ lies in the larger subtree. So we have a constant-time procedure to determine the direction of e in the rooted $\mathcal{S} \cap \mathcal{C}$. If the two subtrees have the same size, then the endpoints of e are the centroids of $\mathcal{S} \cap \mathcal{C}$ and we arbitrarily return one to be $v(f)$. In all, we can find $v(f)$ in time linear in the size of f which is $O(n_{\mathcal{C}})$. \square

Let L be a line parallel to the y -axis that goes through $v(f)$ (see Fig. 8). We call a ridge point in $\mathcal{S} \cap \mathcal{C} \cap L$ an *interior ridge point* if it does not lie on an edge of $\mathcal{S} \cap \mathcal{C}$ incident to the boundary of \mathcal{C} . We show how to compute the set E of interior ridge points in $\mathcal{S} \cap \mathcal{C} \cap L$ and the partition information of \mathcal{C} induced by E . Lemma 3 can then be applied to finish the divide step.

Lemma 6 *Let L be a line. Given $D_{\mathcal{C}}$, the set E of the interior ridge points in $\mathcal{S} \cap \mathcal{C} \cap L$ can be computed in $O(n_{\mathcal{C}} \log n_{\mathcal{C}})$ time. Within the same time bound, we can obtain the partition information of \mathcal{C} induced by E .*

Proof: We go to 3D. Let H be the plane perpendicular to \mathcal{P} that contains L . Let $\hat{\mathcal{K}}$ be the lifted version of $\mathcal{K}(\mathcal{S} \cap \mathcal{C})$ on \mathcal{T} . Like in the proof of Lemma 4, we can compute $H \cap \hat{\mathcal{K}}$ in $O(n_{\mathcal{C}} \log n_{\mathcal{C}})$ time. Let v be a vertex of $H \cap \hat{\mathcal{K}}$. The vertex v projects onto some edge e of $\mathcal{S} \cap \mathcal{C}$. Like in Lemma 4, we get the pointers to the two faces f_1 and f_2 in $\text{faces}(\mathcal{C})$ adjacent to e . The projection of v is an interior ridge point iff f_1 and f_2 are not adjacent along the boundary of \mathcal{C} . This can be tested in constant time using the indices of f_1 and f_2 . Suppose that v is an interior ridge point. Using the defining slabs of f_1 and f_2 , we can compute in constant time the descent paths defined by v . Note that \hat{f}_1 and

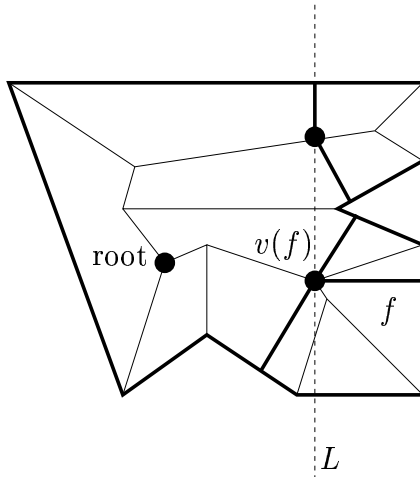


Figure 8: The divide step. Here the initial cell \mathcal{C} is the whole polygon. A face f is chosen at random. The line L is parallel to the y -axis and passes through the internal node $v(f)$. The interior ridge points of L (here, only two) define the canonical partition that will be used to divide \mathcal{C} .

\hat{f}_2 are the faces intersected by the descent paths defined by v . Using the edge pointers stored with f_i , we can retrieve the (at most five) bounding edges of f_i in $edges(\mathcal{C})$. Then we intersect them with the descent paths defined by v to identify the edges in $edges(\mathcal{C})$ that the descent paths lead to. \square

4.4 The straight skeleton algorithm.

We start by computing the point where each motorcycle runs out of fuel (i.e., hits the boundary of P), it is a ray shooting problem that can be solved in $O(n \log n)$ time [7, 16].

The following pseudo-code describes our recursive divide-and-conquer algorithm. The input is $D_{\mathcal{C}}$ for some cell \mathcal{C} and the output is $\mathcal{S} \cap \mathcal{C}$. We first call $skeleton(D_{\mathcal{P}})$. We denote by $\bar{\mathcal{C}}$ the closure of \mathcal{C} , that is, the union of \mathcal{C} and its boundary.

Algorithm $skeleton(D_{\mathcal{C}})$

1. **if** $n_{\mathcal{C}} < 20$
2. **then** compute $\mathcal{S} \cap \mathcal{C}$ by brute force and return the result;
3. **repeat**
4. compute a random face f of $\mathcal{K}(\mathcal{S} \cap \mathcal{C})$ using Lemma 4;
5. identify $v(f)$ using Lemma 5;
6. **if** $v(f)$ is the root of $\mathcal{S} \cap \mathcal{C}$
7. **then** let $E = \{v(f)\}$;

8. compute the partition information of \mathcal{C} induced by E
9. **else** let L be the line parallel to the y -axis that contains $v(f)$;
10. compute the set E of the interior ridge points in $\mathcal{S} \cap \mathcal{C} \cap L$ and the partition information of \mathcal{C} induced by E using Lemma 6;
11. subdivide \mathcal{C} into cells $\{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_k\}$ with respect to E using Lemma 3;
12. **until** $\max_{1 \leq i \leq k} n_{\mathcal{C}_i} \leq 3n_{\mathcal{C}}/4$;
13. recursively call *skeleton*($D_{\mathcal{C}_i}$) for all $1 \leq i \leq k$;
14. return $\mathcal{S} \cap \overline{\mathcal{C}} = E \cup (\mathcal{S} \cap \overline{\mathcal{C}}_1) \cup (\mathcal{S} \cap \overline{\mathcal{C}}_2) \cup \dots (\mathcal{S} \cap \overline{\mathcal{C}}_k)$

In line 6, we can tell whether $v(f)$ is the root in constant time as described in the proof of Lemma 5. In line 8, the computation can be done in constant time as described in the proof of Lemma 6. In line 14, we consider the closure of $\mathcal{S} \cap \overline{\mathcal{C}}$, note that it may not be a tree when it includes sections of valleys that belong to the boundary of \mathcal{C} . These sections can be recovered using *edges*(\mathcal{C}), and thus $\mathcal{S} \cap \overline{\mathcal{C}}$ can be composed in $O(\sum_{1 \leq i \leq k} n_{\mathcal{C}_i}) = O(n_{\mathcal{C}} + |E|) = O(n_{\mathcal{C}})$ time.

Correctness follows from Property 1 and the recursive nature of canonical partitions, if the algorithm terminates. The only uncertainty is the number of iterations of the repeat loop. In the next subsection, we will bound this and hence the total running time.

4.5 Time complexity analysis.

We first bound the expected running time of *skeleton*($D_{\mathcal{C}}$) ignoring the recursive calls at line 13. Each individual step takes $O(n_{\mathcal{C}} \log n_{\mathcal{C}})$ time. We show that $\max_{1 \leq i \leq k} n_{\mathcal{C}_i} \leq 3n_{\mathcal{C}}/4$ holds with probability at least $1/3$. Hence, the expected number of iterations of the loop is at most three.

Case 1: $v(f)$ is the root. There are three descent paths for $v(f)$. Let ρ_i , $1 \leq i \leq 3$, denote the projections of these three descent paths which must lie inside \mathcal{C} . They subdivide \mathcal{C} into three smaller cells. Denote the cell sandwiched by ρ_i and ρ_{i+1} by \mathcal{C}_i . Since $v(f)$ is the root, after cutting at $v(f)$, each subtree obtained has less than $n_{\mathcal{C}}/2$ (non-root) internal nodes. Recall that each face g has a unique vertex representative $v(g)$ and each non-root internal node is $v(g)$ for exactly one face g . It follows that \mathcal{C}_i has less than $n_{\mathcal{C}}/2 + 2$ faces, where the additive factor 2 accounts for the two faces that ρ_i and ρ_{i+1} intersect. For $n_{\mathcal{C}} \geq 20$, $n_{\mathcal{C}}/2 + 2 < 3n_{\mathcal{C}}/4$.

Case 2: $v(f)$ is not the root. Assume that \mathcal{C}_1 obtained in line 11 contains the root. Let u be a non-root internal node that is not on the same side of L as the root. Recall that u is $v(g)$ for exactly one face g . We walk from u to the root and let x be the first interior ridge point in E that we encounter. Observe that x is not the root and x lies outside g . Therefore, the projections of the descent paths for x separate the root from u and they do not intersect g . So g lies outside \mathcal{C}_1 . We conclude that the expected number of faces in \mathcal{C}_1 is at most $n_{\mathcal{C}}$ minus the expected number of internal nodes that are not on the same side of L as the root. Since $v(f)$ is not

the root, $v(f)$ is a non-root internal node of $\mathcal{S} \cap \mathcal{C}$ chosen uniformly at random. $\mathcal{K}(\mathcal{S} \cap \mathcal{C})$ has exactly $n_{\mathcal{C}} - 3$ non-root internal nodes. So with probability at least $1/3$, the number of internal nodes that are not on the same side of L as the root is $\geq (n_{\mathcal{C}} - 3)/3$ which is $> n_{\mathcal{C}}/4$ for $n_{\mathcal{C}} \geq 20$. We conclude that \mathcal{C}_1 has at most $3n_{\mathcal{C}}/4$ faces with probability at least $1/3$. As analyzed in case 1, the number of faces in \mathcal{C}_i for $2 \leq i \leq k$ is less than $3n_{\mathcal{C}}/4$.

We are now ready to bound the total expected running time. $\text{skeleton}(D_{\mathcal{P}})$ runs in expected $O(n \log n)$ time plus the time taken by the recursive calls in line 13. We examine the recursion tree (see Cormen et al. [9] pp 66–67). Since the number of faces in a cell drops by a factor of at least $3/4$ between levels, the depth of the recursion tree is $O(\log n)$. Let \mathcal{F}_k denote the family of cells \mathcal{C} such that the call $\text{skeleton}(D_{\mathcal{C}})$ appears at the k th level of the recursion tree. Then the total expected running time is $O(N \log N \log n)$, where N is an upper bound of the total size of cells in \mathcal{F}_k . We have two observations. First, the cells in \mathcal{F}_k form a canonical partition of \mathcal{P} . Second, since we subdivide with respect to interior ridge points in each recursive call, each edge of \mathcal{S} is subdivided at most once in the canonical partition of \mathcal{P} formed by cells in \mathcal{F}_k . It follows that $N = O(n)$.

Lemma 7 *Let \mathcal{P} be a non-degenerate simple polygon with n vertices. Given the motorcycle graph induced by the reflex vertices of \mathcal{P} , the straight skeleton of \mathcal{P} can be computed in $O(n \log^2 n)$ expected time.*

So by Theorem 1, we obtain the following result when \mathcal{P} is non-degenerate in the sense that no two motorcycles of \mathcal{G} collide and \mathcal{S} is an unrooted binary tree:

Theorem 4 *The straight skeleton of a non-degenerate simple polygon with n vertices and r reflex vertices can be computed in $O(n \log^2 n + r\sqrt{r} \log r)$ expected time.*

5 Conclusion and open problems

Some of the general position assumptions that we have made can be removed easily, and Theorems 3 and 4 still hold when we only assume that all the valleys of the terrain are adjacent to a reflex vertex of the initial polygon. In order to achieve full generality, we need to handle the case where two reflex vertices collide during the shrinking process and generate a new reflex vertex. In other words, two valleys are merged into a new one. The motorcycle graph of \mathcal{P} can be defined in a natural way to account for this (see Fig. 9). With this modification, our new characterization holds and our algorithm works. However, the motorcycle graph cannot be computed with our algorithm because it cannot perform dynamic insertion of motorcycles. So we use the algorithm by Eppstein and Erickson [13]. The details will be given in the full version. We just state the main result here:

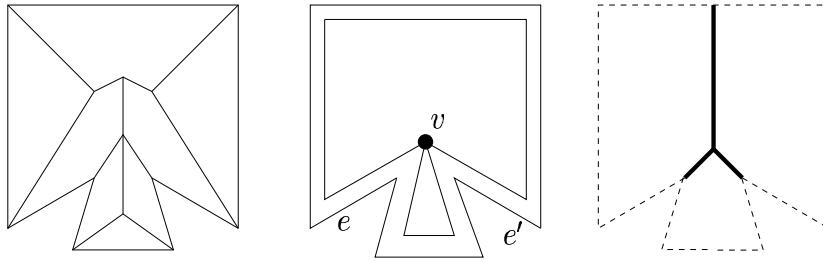


Figure 9: A degenerate roof, a vertex event and the associated motorcycle graph

Theorem 5 *The straight skeleton of a simple polygon with n edges and r reflex vertices can be computed in $O(n \log^2 n + r^{17/11+\epsilon})$ expected time.*

We note that none of our time bounds has been proven to be tight. One way to lower them would be to find an improved algorithm for the generalized motorcycle graph problem where dynamic insertion of motorcycles is allowed. Another possible direction would be to try to extend our algorithm to polygons with holes, since our characterization holds in this case.

References

- [1] P. K. Agarwal. Partitioning arrangement of lines, i: An efficient deterministic algorithm. *Discrete Comput. Geom.*, 5 (1990), 449–483.
- [2] O. Aichholzer and F. Aurenhammer. Straight skeletons for general polygonal figures in the plane. *Proc. 2nd Annu. Int'l. Computing and Combinatorics Conf.*, 1996, 117–126.
- [3] O. Aichholzer, F. Aurenhammer, D. Alberts, and B. Gärtner. A novel type of skeleton for polygons. *The Journal of Universal Computer Science*, 1(1995), 752–761.
- [4] H. Blum. A transformation for extracting new descriptors of shape. In *Models for the Perception of Speech and Visual Form*, (eds. W. Wathen-Dunn), 362–380, 1967, M.I.T. Press.
- [5] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom.*, 9 (1993), 145–158.
- [6] B. Chazelle and J. Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10 (1990), 229–249.
- [7] B. Chazelle and L. Guibas. Visibility and intersection problems in plane geometry. *Discrete Comput. Geom.*, 4 (1989), 551–581.

- [8] F. Chin, J. Snoeyink, and C. A. Wang. Finding the medial axis of a simple polygon in linear time. *Discrete Comput. Geom.*, 21 (1999), 405–420.
- [9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press and McGraw-Hill Book Company, 6th edition, 1992.
- [10] E. D. Demaine, M. L. Demaine, and J. S. B. Mitchell. Folding flat silhouettes and wrapping polyhedral packages: New results in computational origami. *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, 105–114, 1999.
- [11] Erik D. Demaine, Martin L. Demaine, and Anna Lubiw. Folding and cutting paper. *JCDCG*, 104–118, 1998.
- [12] Erik D. Demaine, Martin L. Demaine, and Anna Lubiw. Folding and one straight cut suffice. *Proc. 10th Annu. ACM-SIAM Sympos. Discrete Alg.*, 891–892, 1999.
- [13] D. Eppstein and J. Erickson. Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. *Discrete Comput. Geom.*, 22 (1999), 569–592.
- [14] S. Har-Peled. Constructing planar cuttings in theory and practice. *SIAM J. Comput.*, 29 (2000), 2016–2039.
- [15] J. Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Inform. Process. Lett.*, 33(1989), 169–174.
- [16] J. Hershberger and S. Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *J. Algorithms*, 18(1995), 403–431.
- [17] J. Matoušek. Construction of ϵ -nets. *Discrete Comput. Geom.*, 5 (1990), 427–448.
- [18] J. Matoušek. Cutting hyperplane arrangements. *Discrete Comput. Geom.*, 6 (1991), 385–406.
- [19] J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8 (1992), 315–334.
- [20] E. Papadopoulou and D. T. Lee. The L_∞ Voronoi Diagram of Segments and VLSI Applications. *Internat. J. Comput. Geom. Appl.*, 11 (2001), 503–528.
- [21] R. Tarjan. *Data Structures and Network Algorithms*, chapter 4. SIAM, Philadelphia, PA, 1983.