Motorcycle Graphs and Straight Skeletons*

Siu-Wing Cheng† Antoine Vigneron ‡

March 17, 2005

Abstract

We present a new algorithm to compute motorcycle graphs. It runs in $O(n\sqrt{n}\log n)$ time when n is the number of motorcycles. We give a new characterization of the straight skeleton of a non–degenerate polygon. For a polygon with n vertices and h holes, we show that it yields a randomized algorithm that reduces the straight skeleton computation to a motorcycle graph computation in expected $O(n\sqrt{h+1}\log^2 n)$ time. Combining these results, we can compute the straight skeleton of a non–degenerate polygon with h holes and with n vertices, among which r are reflex vertices, in $O(n\sqrt{h+1}\log^2 n + r\sqrt{r}\log r)$ expected time. In particular, we can compute the straight skeleton of a non–degenerate polygon with n vertices in $O(n\sqrt{n}\log^2 n)$ expected time.

Key words: Computational Geometry – Randomized Algorithm – Straight Skeleton Medial Axis – Motorcycle graph

1 Introduction

In 1995 Aichholzer et al. [2, 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 and 3). As opposed to the widely used medial axis [13] (see Fig. 2), the straight skeleton has only straight line edges, which is useful when parabolic edges need to be avoided, either because the application requires it or because the software library only handles polygonal figures.

The straight skeleton allows to find offset polygons, known as mitered offset lines, which is a standard operation in computer aided design [20] (note that the medial axis yields offset curves containing circle arcs). It also answers a roof reconstruction problem: given a horizontal section of the walls of a house, find a roof whose faces have same slope and that has one face per wall. (See Fig. 7b.) There could be several possible answers [7] to this problem, the projection of the edges of one of these roofs to the horizontal plane is the straight skeleton of the horizontal section of the walls [3]. These nice properties have been successfully exploited in several applications: polyhedral surface reconstruction from cross–sections [5, 22, 30], biomedical images processing [14], polygon decomposition [31] (in particular, for computer vision applications),

*The work described in this paper has been supported by the Research Grants Council of Hong Kong, China (Project no. HKUST 6088/99E) and by the National University of Singapore (grants R-252-000-130-101 and R-252-000-130-112).

†Dept. of Computer Science, Hong Kong University of Science & Technology, Clear Water Bay, Kowloon, Hong Kong. Email: scheng@cs.ust.hk.

‡Dept. of Computer Science, National University of Singapore, 3, Science Drive 2, Singapore 117 543. Tel: (65) 6874 6802. Fax: (65) 6779 4580. Email: antoine@comp.nus.edu.sg.

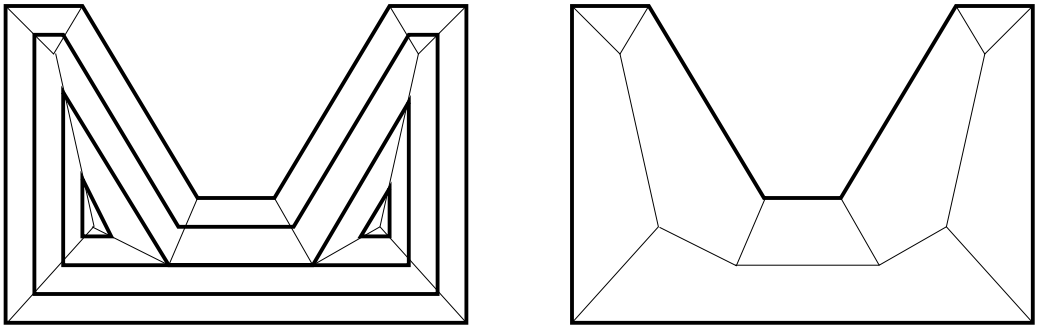

Figure 1: The straight skeleton (on the right) is obtained by shrinking the initial polygon

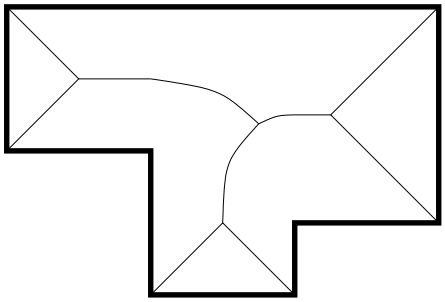

Figure 2: The medial axis of an orthogonal polygon. Note that, even in this simple case, it has parabolic edges.

computational origami [17, 18, 19], computing the city Voronoi diagram [4], morphing between shapes [6], automatic generation of city models [7, 26]. Hence, it is important to design efficient algorithms for computing the straight skeleton, especially since it is currently the bottleneck of some of these applications [5, 14, 31].

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 polygons with holes and brought the space complexity down to $O(n)$. They also show that the straight skeleton cannot be described as the projection of a lower envelope in a similar way as the medial axis. This explains why standard computational geometry techniques such as the randomized incremental construction do not apply directly. Eppstein and Erickson [20] gave the first sub–quadratic algorithm, its 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.

We will give new connections between the straight skeleton and the motorcycle graph problem [20]. This 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, $1 \leqslant i \leqslant n$, 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 called the motorcycle graph (see Fig. 4). Eppstein

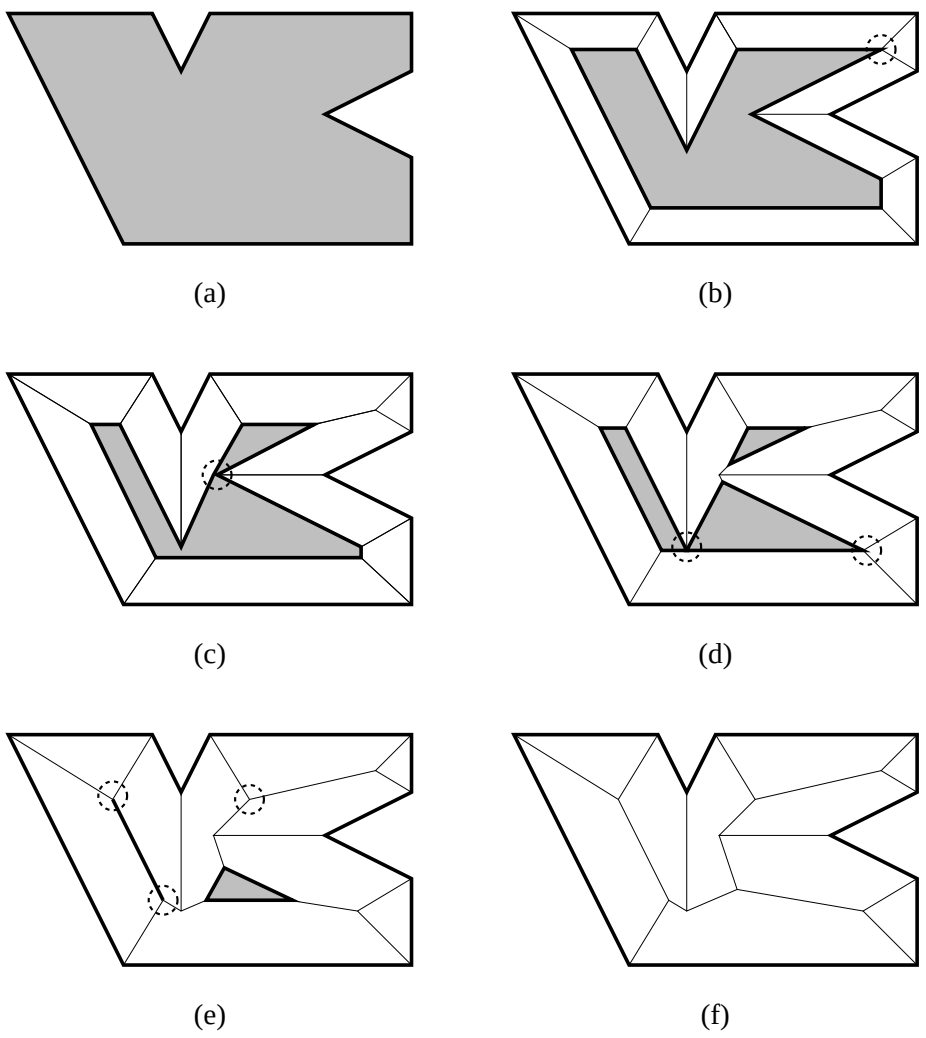

Figure 3: The input polygon (a). During the shrinking process, an edge can disappear (b), this event is called an edge event. A split event (c) occurs when a reflex vertex hits an edge, splitting the polygon into two parts. Several events may occur simultaneously (d). The straight skeleton is obtained as the trace of the reflex vertices (f). Note that in (e), two edges collided during the shrinking process, we keep the trace of this collision in the straight skeleton (f).

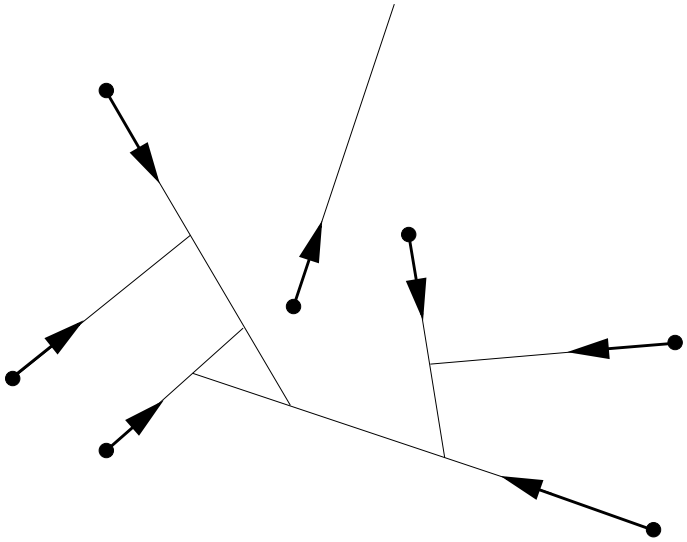

Figure 4: A motorcycle graph

and Erickson [20] solved 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 [20]. We also present a simple randomized algorithm with the 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 non–degenerate polygon with h holes in expected time $O(n\sqrt{h+1}\log^2 n)$ after the motorcycle graph induced by its reflex vertices has been computed. (Our non–degeneracy assumptions are explained in Section 3.2.) This formalizes the idea of Eppstein and Erickson that the motorcycle graph problem captures the most difficult part of the construction of a straight skeleton. Putting everything together, we can compute in $O(n\sqrt{h+1}\log^2 n + r\sqrt{r}\log r)$ expected time the straight skeleton of a non-degenerate polygon. Since $n > r > h$, our algorithm runs in expected time $O(n\sqrt{n}\log^2 n)$. As a comparison, the algorithm by Eppstein and Erickson [20] is slower, but it is deterministic and it can handle degenerate polygons.

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 of them 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. When n is the number of motorcycles, 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 choose 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 2.4). 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. We will show that this algorithm runs in $O(n\sqrt{n}\log n)$ time.

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.

We will make use of *cuttings*: 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 [1, 8, 9, 28, 29]. We will employ a deterministic algorithm presented by Chazelle [8] that runs in $O(n\sqrt{n})$ time. It produces a cutting with $O(n)$ cells and gives the lines intersecting each cell.

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 $t' \in [0, t]$ (i.e., motorcycle T_i crashes into T_j or into the track left by T_j).

All the switching events will be generated during the initialization phase, before the simulation starts. Within our time bounds, we cannot generate all the impact events as there can be a quadratic number of them. But 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, then it intersects its interior. The handling of degenerate cases will be discussed in Section 2.3.

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 [8]. 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 $t \geqslant t'$, then we insert the impact event (i, j, t) into $\mathcal{Q}$. If $t' \geqslant t$, then we insert the impact event (j, i, t') into $\mathcal{Q}$ (see Fig. 5). By

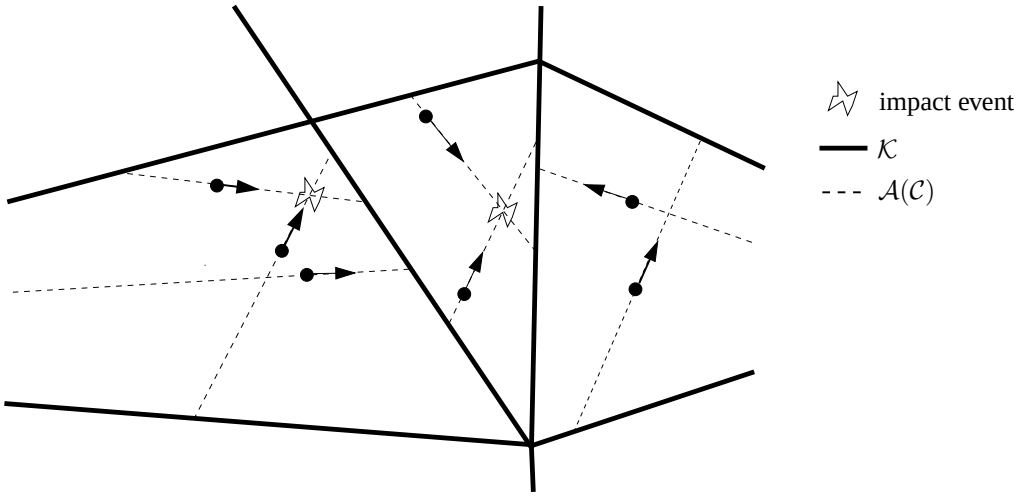

Figure 5: The initialization step. Two impact events are queued here.

the property of cutting, the total size of the local arrangements during this initialization phase is $O(n\sqrt{n})$, so it can be performed in $O(n\sqrt{n}\log n)$ time by plane sweep [16].

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 switching 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** $t_i \geqslant t_j$
15. **then** insert the impact event (i, j, t_i) into $\mathcal{Q}$
16. **if** $t_j \geqslant t_i$
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 the switching events for M_i
22. delete from $\mathcal{Q}$ all impact events involving M_i that occur at points on T_i that are beyond the intersection point $T_i \cap T_j$.

The correctness of this algorithm 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 [16] where each cell is associated with a search structure allowing split operations in logarithmic time (for instance a balanced binary tree [33]). 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. For all i, we maintain the impact events involving M_i in a balanced binary search tree ordered 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 $\mathcal{C} = \mathcal{C}_i$ or $\mathcal{C} = \mathcal{C}_j$ 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.*

2.3 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 T_j and T_k lie on L, and such that $[p_j, p_k]$ contains $L \cap L_i$ and is minimal. 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 a motorcycle 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 $\mathcal{K}$ and new events involving them throughout the simulation. If a motorcycle reaches a vertex v of $\mathcal{K}$, any other motorcycle reaching v afterward will crash at v, so there is only a linear number of such event. We can handle them by 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 motorcycles at any time, which allows to record their status and handling the events involving them without hurting our time bounds.

2.4 A simple randomized algorithm

Our algorithm is simple and implementable, and, using practical planar cuttings algorithm [23], we expect it to beat the naive $O(n^2 \log n)$-time algorithm in practice. A simpler algorithm is given by the following random sampling approach, that has the same running time in expectation. We first choose $\sqrt{n}$ motorcycles uniformly at random, and compute the arrangement $\mathcal{K}^*$ of their support lines. This arrangement $\mathcal{K}^*$ plays the role of the cutting $\mathcal{K}$ of our deterministic algorithm; in fact it is the only modification we make. The expected running time of this algorithm is the expected sum of the sizes of the local arrangements in $\mathcal{K}^*$ multiplied by $O(\log n)$.

There are two kinds of vertices in these local arrangements. First there are the vertices that lie on the boundary of a cell, and correspond to switching events. It is easy to see that there are at most $\sqrt{n}$ such events per motorcycle. We denote by $\mathcal{C}_i$ (resp. $\mathcal{C}'_i$) the initial (resp. final) cell of M_i in $\mathcal{K}^*$. As in the deterministic case, the number of local arrangement vertices that do not lie on the boundary of any cell is bounded by the sum, for all i, of the number n_i of vertices that lie in $L_i \cap (\mathcal{C}_i \cup \mathcal{C}'_i)$. Consider the intersection points of the type $L_i \cap L_j$ for $j \neq i$. We color $L_i \cap L_j$ red if L_j is a line of $\mathcal{K}^*$ and we color it blue otherwise. We denote by n_i the number of blue points that we can reach without crossing a red point when we move along L_i, starting from the initial or the final position of M_i. Since an intersection point is colored red with uniform probability $\sqrt{n}/(n-1)$, we can show easily that the probability that $n_i > c\sqrt{n}$ is $e^{-\Omega(c)}$ (a similar analysis can be found in article [11] Lemma 1.1). So the expected value of this number is $O(\sqrt{n})$ and, by linearity of expectation, the expected total size of the local arrangements is $O(n\sqrt{n})$. It follows that the expected running time of our algorithm is $O(n\sqrt{n}\log n)$.

2.5 Further remarks

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 backward). As a comparison, Eppstein and Erickson [20] 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. It will allow us to develop a faster algorithm for constructing the straight skeleton of a simple polygon (see sections 4 and 5).

3.1 Definition

For convenience, we place ourselves in three dimensional 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 figures 1, 3 and 7 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 types 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. 6). These events are called *edge events*. Second,

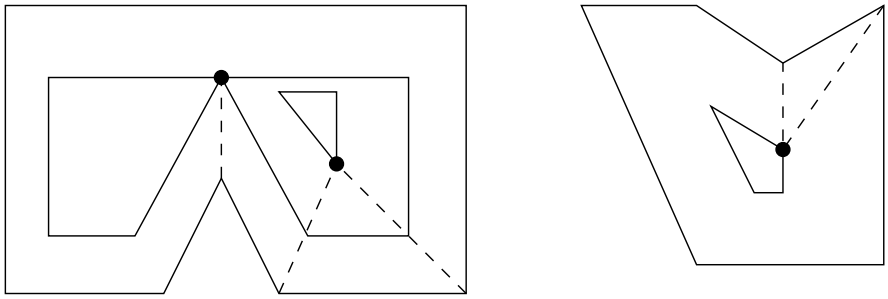

Figure 6: On the left, a split event and an edge event. On the right, an edge event involving a reflex vertex.

a vertex may hit an edge, which splits the polygon into two parts (see Fig. 6). These events are called *split* events.

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 speed, drawing a terrain $\mathcal{T}$ in three dimension (see Fig. 7 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. 7 c,d). We call the reflex edges of $\mathcal{T}$ *valleys*. (See Fig. 8 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.

3.2 Non–degeneracy assumptions

In degenerate cases, several edge or split events can take place simultaneously, and create straight skeleton vertices of degree higher than three. Eppstein and Erickson noticed that these situations can be handled by standard perturbation methods (in [20], section 4.1). So in this paper we assume that edge events and split events occur one at a time.

A third type of events can occur in degenerate cases. Two reflex vertices can collide, giving birth to a new reflex vertex (see Fig. 9). These events are called *vertex events*. They cannot be handled by perturbation methods, because a small change in the input polygon can change the straight skeleton dramatically [20]. In order to avoid vertex events, we will make the following

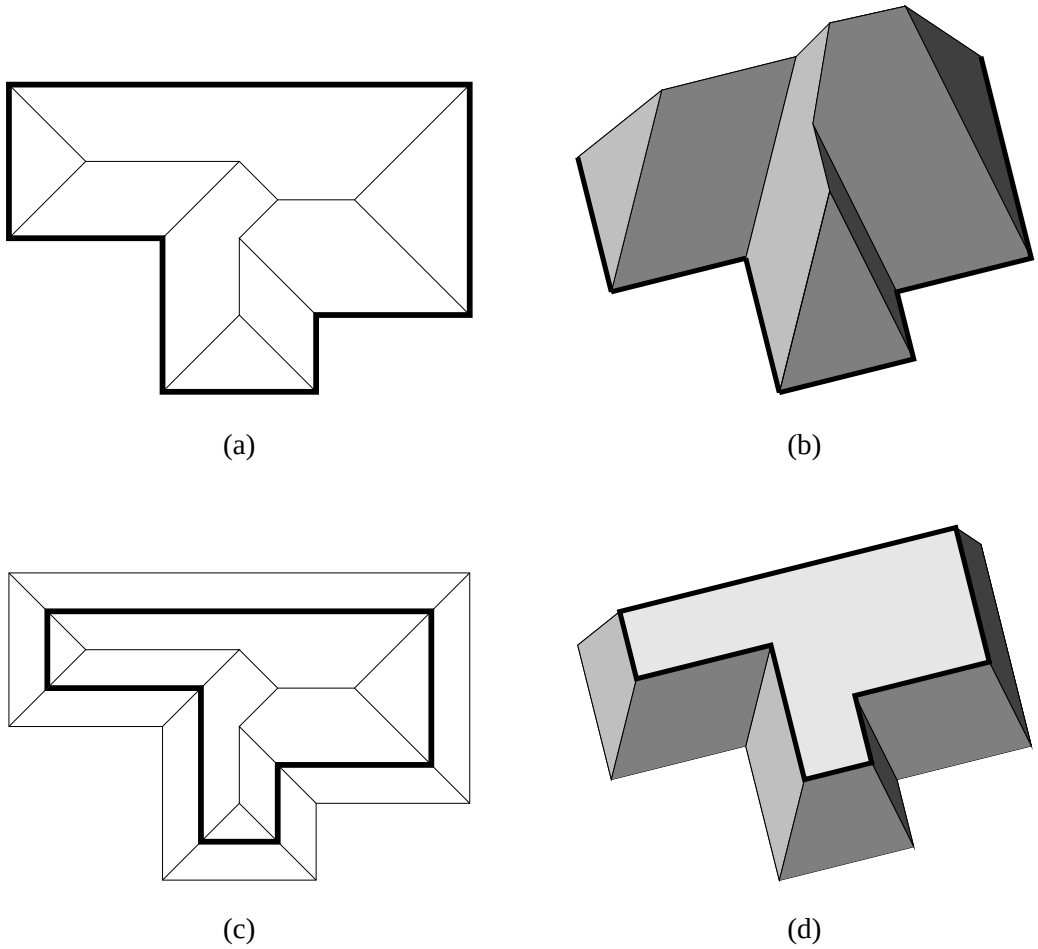

Figure 7: (a) The input polygon $\mathcal{P}$ (thick lines), and its straight skeleton $\mathcal{S}$ (inside). (b) The terrain $\mathcal{T}$ obtained by lifting the subdivision $\mathcal{K}(\mathcal{S})$. (c) The shrunken polygon at time t is a horizontal section of $\mathcal{T}$ at height t. (d) The 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.

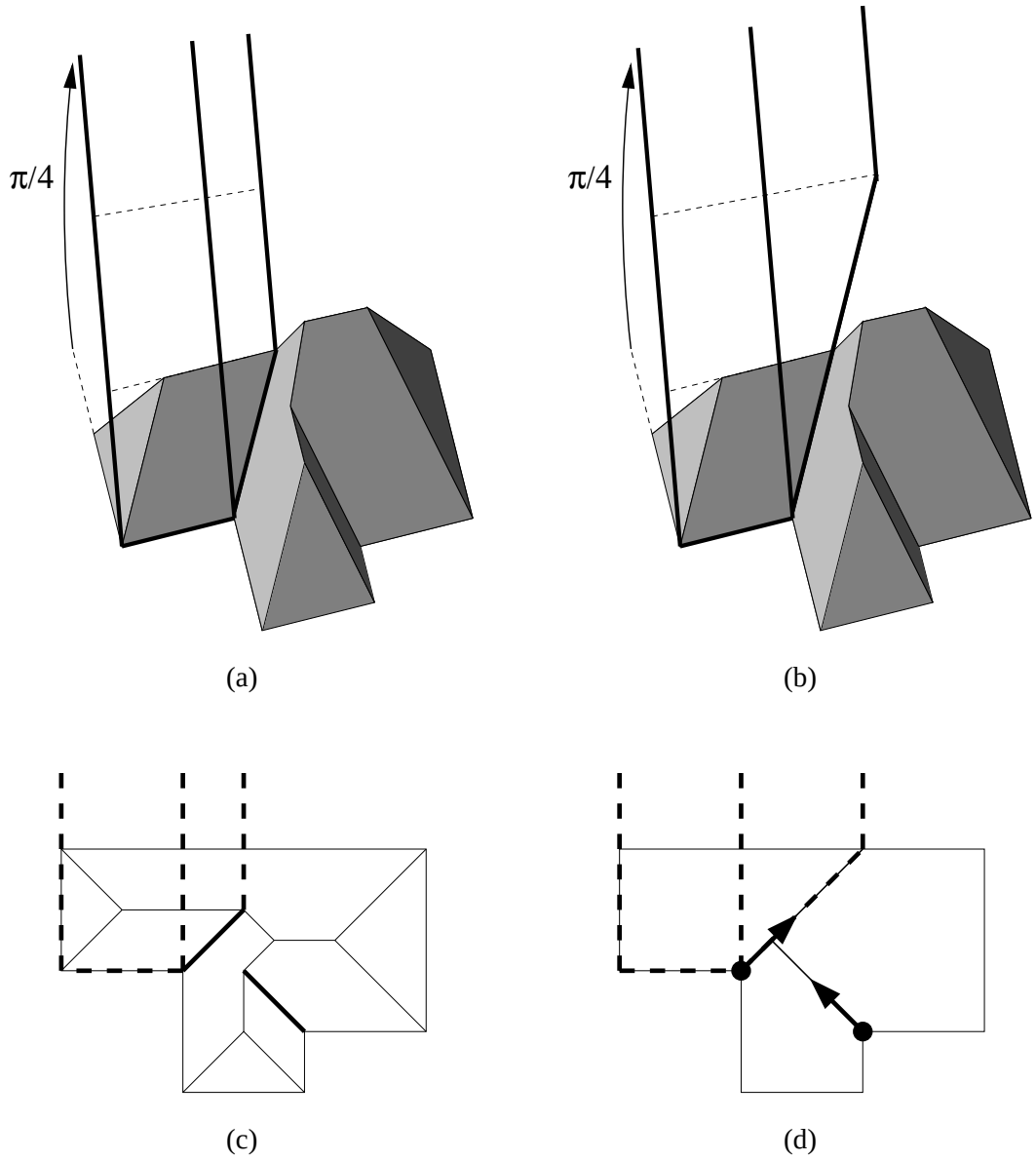

Figure 8: An illustration of the different kinds of slabs we defined, here we only draw the slabs associated with one particular edge that is adjacent to a reflex vertex. (a) The edge slab and the reflex slab. (b) The edge slab and the motorcycle slab. (c) The edge slab, the reflex slab and the two valleys 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.

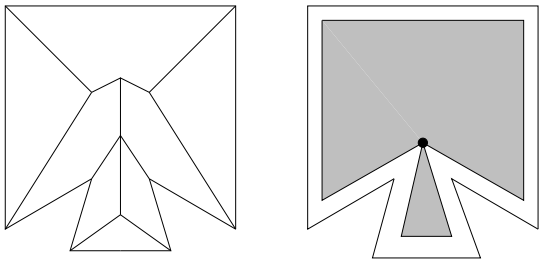

Figure 9: A degenerate roof and a vertex event

non–degeneracy assumption. Every reflex vertex of $\mathcal{P}$ is the lower endpoint of a valley of $\mathcal{T}$. We consider all the half–lines obtained by extending these valleys to $z = +\infty$. We will assume that no two such half–lines intersect. In particular, it means that no two reflex vertices of the input polygon $\mathcal{P}$ will collide (in a vertex event), and since a new reflex vertex can only appear after a vertex event, it means that no vertex event happens during the whole shrinking process.

It follows from our non–degeneracy assumptions that all the valleys of $\mathcal{T}$ are adjacent to reflex vertices of $\mathcal{P}$. Another consequence is that all the vertices of the straight skeleton have degree one or three .

3.3 Other characterizations

Eppstein and Erickson [20] 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. 8 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.

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. 8 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. 8).*

Proof: Suppose that the lemma is false. So there exists an edge of $\widehat{\mathcal{G}}$ that is shorter than or equal to its corresponding valley. Among all such edges of $\widehat{\mathcal{G}}$, we choose an 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. 7 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. We first show that $\mathcal{T}_t \cap S$ is convex. Remember that the only reflex edges of $\mathcal{T}_t$ are the valleys. So, if there was a point x on the boundary of $\mathcal{T}_t \cap S$ where $\mathcal{T}_t \cap S$ is locally concave, then either a valley of $\mathcal{T}_t$ would cross $\mathcal{T}_t$ at x, or two valleys would meet at x. The first case is impossible because the valley would be longer than its corresponding edge in $\widehat{\mathcal{G}}$. The second case is impossible too by our non–degeneracy assumptions (see Section 3.2).

Therefore $\mathcal{T}_t \cap S$ is a convex chain. Since $\widehat{f}$ is tangent to $\mathcal{T}_t$ at its lower endpoint, $\widehat{f}$ is on or above $\mathcal{T}_t \cap S$. By our non–degeneracy assumptions, $\widehat{f}$ and $\widehat{e}$ cannot intersect, and since the higher endpoint of $\widehat{e}$ is on $\mathcal{T}$, then $\widehat{f}$ is above the higher endpoint of $\widehat{e}$. This contradicts the fact that e crashes into f. □

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 $\mathcal{T}$. □

We are now ready to prove our characterization. Remember that $\mathcal{P}$ is non–degenerate in the sense of Section 3.2.

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 $\widehat{e}$. Lemma 1 implies that $S(\nu) \subseteq S(\widehat{e})$. By Theorem 2, it suffices to prove that each point in $S(\widehat{e}) \setminus S(\nu)$ is on or above $\mathcal{T}$. Consider a point p in $\widehat{e} \setminus \nu$. Let r be a halfline that starts at p, has unit slope, and lie on a motorcycle slab bounded below by $\widehat{e}$. 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 $\mathcal{T}$. □

4 Computing the straight skeleton of a simple polygon

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. By our non–degeneracy assumptions (see Section 3.2), the straight skeleton $\mathcal{S}$ is an unrooted binary tree and 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 $\widehat{p}$ be the corresponding point on $\mathcal{T}$. The point p is a *ridge point* if $\widehat{p}$ does not lie in the interior of a reflex edge (valley). Given a set E of ridge points, for each $p \in E$, $\widehat{p}$ 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*

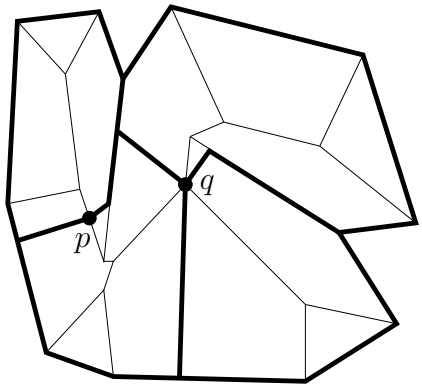

Figure 10: The canonical partition induced by $\{p, q\}$

partition of $\mathcal{P}$ induced by E (see Fig. 10). If E is empty, we take the interior of $\mathcal{P}$ to be 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 $[1 \ldots n_{\mathcal{C}}]$ 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 $edges(\mathcal{C})$: the polygon edge and at most two paths of steepest descent that consist of at most two edges each (one in the interior of the face and one inside a valley).

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 $O(\sum_i n_{\mathcal{C}_i}) = O(n_{\mathcal{C}} + |E|)$.

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 a particular 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 f, pointers to its three defining faces in $faces(\mathcal{C})$.*

Proof: Let $\widehat{f}$ be the lifted version of f on $\mathcal{T}$. We compute $\widehat{f}$ 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 [24]. By Property 1, the region in $S \cap \mathcal{C}$ below this lower envelope is $\widehat{f}$. 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 P. Note that both $f \cap \partial\mathcal{C}$ and the boundary of the projected lower envelope are monotone with respect to the direction of e, so we can compute their arrangement in $O(n_{\mathcal{C}})$ time. The face of this arrangement that is bounded by e is f. □

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.11). 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 choose the centroid whose coordinates are smallest in lexicographical 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 $faces(\mathcal{C})$. 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 on. 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 $O(n_{\mathcal{C}})$. □

Let L be a line parallel to the y–axis that goes through $v(f)$ (see Fig. 11). 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 $\widehat{\mathcal{K}}$ 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 $H \cap \widehat{\mathcal{K}}$ in $O(n_{\mathcal{C}} \log n_{\mathcal{C}})$ time. Let v be a vertex of $H \cap \widehat{\mathcal{K}}$. 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 $faces(\mathcal{C})$ adjacent to e. The projection of v is an interior ridge point if and only if 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 $\widehat{f_1}$ and $\widehat{f_2}$ 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

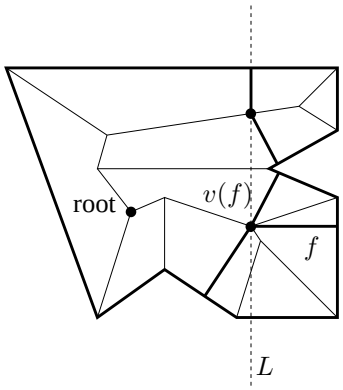

Figure 11: The divide step. Here the initial cell $\mathcal{C}$ is the whole polygon. A face f is chosen at random. The line L is parallel to the y–axis and passes through the internal node $v(f)$. The interior ridge points of L (here, only two) define the canonical partition that will be used to divide $\mathcal{C}$.

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. □

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 [10, 25].

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 $skeleton(D_{\mathcal{P}})$.

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. pick a face f of $\mathcal{K}(\mathcal{S} \cap \mathcal{C})$ uniformly at random in $faces(\mathcal{C})$
5. compute an explicit description of f using Lemma 4
6. identify $v(f)$ using Lemma 5
7. **if** $v(f)$ is the root of $\mathcal{S} \cap \mathcal{C}$
8. **then** let $E = \{v(f)\}$;
9. compute the partition information of $\mathcal{C}$ induced by E
10. **else** let L be the line parallel to the y–axis that contains $v(f)$
11. 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
12. subdivide $\mathcal{C}$ into cells $\{\mathcal{C}_1, \mathcal{C}_2, \ldots \mathcal{C}_k\}$ with respect to E using Lemma 3
13. **until** $\max_{1 \leqslant i \leqslant k} n_{\mathcal{C}_i} \leqslant 3n_{\mathcal{C}}/4$
14. recursively compute $\mathcal{S} \cap \mathcal{C}_i = skeleton(D_{\mathcal{C}_i})$ for all $1 \leqslant i \leqslant k$
15. compute the union U_E for all i of the edges of $\mathcal{C}_i$ that belong to a valley and are not on the boundary of $\mathcal{C}$
16. return $\mathcal{S} \cap \mathcal{C} = E \cup U_E \cup (\mathcal{S} \cap \mathcal{C}_1) \cup (\mathcal{S} \cap \mathcal{C}_2) \cup \cdots \cup (\mathcal{S} \cap \mathcal{C}_k)$

In line 7, we can tell whether $v(f)$ is the root in constant time as described in the proof of Lemma 5. In line 9, the computation can be done in constant time as described in the proof of

Lemma 6. Line 15 is necessary to recover the edges of the skeleton that are on the boundary of some cell $\mathcal{C}_i$. This can happen when the boundary of $\mathcal{C}_i$ follows a valley (see for instance Figure 10). This union, as well as the result of line 16, can be composed in $O(\sum_{1 \leqslant i \leqslant k} n_{\mathcal{C}_i}) = O(n_{\mathcal{C}} + |E|) = O(n_{\mathcal{C}})$ time using the partition information.

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 14. Each individual step takes $O(n_{\mathcal{C}} \log n_{\mathcal{C}})$ time. We show that $\max_{1 \leqslant i \leqslant k} n_{\mathcal{C}_i} \leqslant 3n_{\mathcal{C}}/4$ holds with probability at least $1/3$. Hence, the expected number of iterations of the loop is at most three. We distinguish between two cases, they both make use of the fact that, by elementary graph theory, $\mathcal{K}(\mathcal{S} \cap \mathcal{C})$ has exactly $n_{\mathcal{C}} - 2$ internal nodes.

Case 1: $v(f)$ is the root. After cutting at $v(f)$, each subtree obtained has at most $(n_{\mathcal{C}} - 2)/2$ internal nodes. It follows that each $\mathcal{C}_i$ has at most $n_{\mathcal{C}}/2 + 1$ faces. For $n_{\mathcal{C}} \geqslant 20$, $n_{\mathcal{C}}/2 + 1 < 3n_{\mathcal{C}}/4$.

Case 2: $v(f)$ is not the root. Remember that $\mathcal{K}(\mathcal{S} \cap \mathcal{C})$ has exactly $n_{\mathcal{C}} - 3$ non-root internal nodes, and that $v(f)$ is chosen uniformly at random among them. Let $I = (M_1, M_2, \ldots M_{n_{\mathcal{C}}-3})$ denote this list of non-root internal nodes in increasing order of x coordinates. We rewrite this list as the concatenation of three lists $I = I_1 I_2 I_3$ of equal size $(n_{\mathcal{C}} - 3)/3$. With probability $1/3$, $v(f)$ falls in I_2. We assume it is the case. If the root is on the left of L, then the nodes of I_3 are not on the same side of L as the root. Similarly, if the root is on the right of L, the nodes of I_1 are not on the same side of L as the root. Thus, with probability $1/3$, at least $(n_{\mathcal{C}} - 3)/3$ internal nodes are not on the same side of L as the root. When $n_{\mathcal{C}} \geqslant 20$, it implies that more than $n_{\mathcal{C}}/4$ internal nodes are not on the same side of L as the root.

Assume that $\mathcal{C}_1$ obtained in line 16 contains the root. We also assume that u is an 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 number of faces in $\mathcal{C}_1$ is at most $n_{\mathcal{C}}$ minus the number of internal nodes that are not on the same side of L as the root. So, with probability $1/3$, this quantity is less than $3n_{\mathcal{C}}/4$.

We are now ready to bound the total expected running time. $skeleton(D_{\mathcal{P}})$ runs in expected $O(n \log n)$ time plus the time taken by the recursive calls in line 14. We examine the recursion tree (see Cormen et al. [15] 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 $skeleton(D_{\mathcal{C}})$ appears at the kth 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 of Section 3.2 (and therefore $\mathcal{S}$ 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 Computing the straight skeleton of a polygon with holes

Let $\mathcal{P}$ be a polygon with h holes. We extend our algorithm to construct $\mathcal{S}$ in $O(n\sqrt{h} \log^2 n)$ expected time given the motorcycle graph. Due to the holes, $\mathcal{S}$ is not a tree. Our strategy is to compute a set E of ridge points such that for each cell $\mathcal{C}$ of the canonical partition of $\mathcal{P}$ induced by E, $\mathcal{S} \cap \mathcal{C}$ is a tree. So we can invoke our algorithm in Section 4 to compute $\mathcal{S} \cap \mathcal{C}$ for each cell $\mathcal{C}$. We assume that $\mathcal{P}$ is non–degenerate as explained in Section 3.2

5.1 Linking the boundaries

We first compute a set of line segments to connect the hole boundaries and the outer boundary of $\mathcal{P}$. We pick a vertex from each hole boundary and the outer boundary. Then we compute a non-self-intersecting spanning tree T of crossing number $O(\sqrt{h})$ to connect these $h+1$ vertices. This can be done in $O(h^{1+\epsilon})$ time [27]. We impose a more convenient structure on T. We duplicate each edge in T and compute an arbitrary Eulerian tour of this graph. This Eulerian tour induces an ordered sequence σ of edges in T and their copies.

5.2 Segment tree

We subdivide the copies of tree edges in σ as follows. We compute the intersections between the tree edges and the boundary of $\mathcal{P}$. We also compute the intersections between the tree edges and the projected boundaries of edge slabs and motorcycle slabs. Each tree edge is subdivided by the intersections on it into a set of intervals. This turns σ into an ordered sequence of intervals. We organize a segment tree [16] on the intervals in σ. We call the intervals associated with the internal nodes and leaves of the segment tree *standard intervals*. Let $I(x)$ denote the standard interval associated with a node x. If an edge slab or motorcycle slab S covers $I(x)$ but S does not cover $I(\text{parent}(x))$, then S is stored at x.

Lemma 8 *There are $O(n\sqrt{h})$ standard intervals. Each slab is stored at $O(\sqrt{h} \log n)$ segment tree nodes.*

Proof: An endpoint of each standard interval is either a tree edge endpoint or an intersection between a tree edge and the boundary of $\mathcal{P}$ or a projected slab boundary. There are $O(h)$ endpoints. Since the spanning tree has crossing number $O(\sqrt{h})$, a boundary edge of $\mathcal{P}$ or a projected slab boundary can intersect at most $O(\sqrt{h})$ tree edges. It follows that there are $O(n\sqrt{h})$ standard interval endpoints and hence $O(n\sqrt{h})$ standard intervals. Each slab intersects σ at $O(\sqrt{h})$ intervals, so it is stored at $O(\sqrt{h} \log n)$ nodes. □

We keep two auxiliary data structures $L(x)$ and $S(x)$ with each node x of the segment tree. $L(x)$ is the lower envelope of the supporting planes of the slabs stored at x. $S(x)$ is the set of slabs stored at the proper descendants of x, i.e., for each slab in $S(x)$, the slab boundary intersects the standard interval associated with x. Since each $L(x)$ can be computed in $|L(x)| \log |L(x)|$ time, by Lemma 8, the auxiliary data structures of all the nodes of the segment tree can be computed in $O(n\sqrt{h}\log^2 n)$ time.

5.3 Partition

The edges of T are subdivided by the intersections on them into line segments. We pick those that lie within the interior of $\mathcal{P}$. Among the line segments picked, we select a subset that form a spanning tree of the holes and outer boundaries. We call the selected line segments *links*. We compute the intersections between the straight skeleton $\mathcal{S}$ and each link pq as follows. Let H_{pq} be the vertical slab bounded by vertical lines through p and q. We compute the intersections between pq and $\mathcal{S}$ by computing the intersections between H_{pq} and the terrain corresponding to $\mathcal{S}$. In the proof of Lemma 6, we intersect a vertical slab with the terrain by first computing the intersections with the edge slabs and motorcycle slabs and then finding the lower envelope of the line segments at the intersections. This brute-force approach is too slow for our purposes here because we may waste time examining a slab which does not contribute to any intersection on pq. Instead, we query the segment tree to identify $O(\log n)$ nodes so that pq is equal to the union of the standard intervals associated with these nodes. Let x be one node identified. Let $anc(x)$ denote the set of ancestors of x including x itself. Let H_x denote the vertical slab based at $I(x)$. We compute $H_x \cap \mathcal{S}$ as follows. First, we intersect the slabs in $S(x)$ with H_x and then compute the lower envelope of the line segments at the intersections. Let $chain(x)$ denote the resulting polygonal chain. Second, we compute $H_x \cap L(y)$ for each $y \in anc(x)$. This yields $O(\log n)$ convex chains. Then we compute the lower envelope of $chain(x)$ and $H_x \cap L(y)$ for all $y \in anc(x)$. The projections of the vertices of the resulting lower envelope are the intersections in $I(x) \cap \mathcal{S}$. Note that we also know the defining slabs of the faces of $\mathcal{K}(\mathcal{S})$ that $I(x)$ intersects.

Lemma 9 *In $O(n\sqrt{h}\log^2 n)$ time, we can compute the intersections between $\mathcal{S}$ and the links as well as the the defining slabs of the faces of $\mathcal{K}(\mathcal{S})$ adjacent to these intersections. The number of intersections is $O(n\sqrt{h})$.*

Proof: The correctness is obvious. Since each link is part of an edge of a spanning tree of crossing number $O(\sqrt{h})$, each edge of $\mathcal{S}$ intersects $O(\sqrt{h})$ links. So the number of intersections is $O(n\sqrt{h})$. To analyze the running time, we first bound the total size of chains computed. Let x be a segment tree node identified when we query the segment tree with a link.

We charge the size of the lower envelope $chain(x)$ to slabs in $S(x)$. If a slab S in $S(x)$ is charged again for another segment tree node z, and if z is at the same level as x, then the projected boundary of S intersects $I(z)$. There are $O(\sqrt{h})$ nodes at each level whose standard intervals intersect the projected boundary of S. So S is charged $O(\sqrt{h})$ times at one level and the total charge at S is $O(\sqrt{h}\log n)$. It follows that the total size of $chain(\cdot)$'s computed is $O(n\sqrt{h}\log n)$. For each $y \in anc(x)$, we charge the size of $H_x \cap L(y)$ to the edges of $L(y)$ intersecting H_x and to x. (We need to charge x to take care of the case where $H_x \cap L(y)$ is a single line segment.) The charge accumulated at x is $O(\log n)$. Since each link is decomposed into $O(\log n)$ standard intervals, the $h-1$ links induce $O(h\log^2 n)$ charge to nodes in the segment tree. We analyze the charge at edges of $L(y)$ for all node y. Since each edge of $L(y)$ intersects $O(\sqrt{h})$ links, each edge of $L(y)$ is charged $O(\sqrt{h})$ times. Since the sum of sizes of $L(y)$ for all nodes y at one level

is $O(n)$, the total charge at one level is $O(n\sqrt{h})$. It follows that the total size of convex chains computed is $O(h\log^2 n + n\sqrt{h}\log n) = O(n\sqrt{h}\log n)$.

We are ready to analyze the running time. The lower envelope $chain(x)$ can be computed in time $O(|chain(x)|\log|chain(x)|)$ [24]. Consider computing $H_x \cap L(y)$ for a node y in $anc(x)$. We first locate the face of $L(y)$ vertically above an endpoint of $I(x)$. This can be done by point location in the projection of $L(y)$. Then we simply walk from one face of $L(y)$ to another. This can be done in $O(\log n)$ time per advance if the boundary of each face of $L(y)$ is stored using some balanced tree scheme. So each convex chain $H_x \cap L(y)$ can be computed in $O(|H_x \cap L(y)|\log n)$ time. So the total time needed to compute all the chains is $O(\log n)$ times the total size of all the chains, which is $O(n\sqrt{h}\log^2 n)$. Consider the computation of the lower envelope of $chain(x)$ and $H_x \cap L(y)$ for all $y \in anc(x)$. Using the algorithm in [24], this can be done in $O(m\log m)$, where m is the input size. Summing over all nodes identified for all the links, we get a bound of $O(n\sqrt{h}\log^2 n)$. Hence, the intersections between the links and $\mathcal{S}$ can be found in $O(n\sqrt{h}\log^2 n)$ time. □

5.4 The algorithm

We use Lemma 9 to compute the set of intersections. Among the intersections, we extract the ridge points. If there are more than one ridge point on some edge of $\mathcal{S}$, then we only keep one. Let E be the set of ridge points obtained. We compute the descent paths from the ridge points in E in $O(n\sqrt{h})$ time. This yields a canonical partition and the following result shows that the portion of the straight skeleton inside each cell of this canonical partition is a tree.

Lemma 10 *Let $\mathcal{C}$ be a cell in the canonical partition of $\mathcal{P}$ induced by E. Then $\mathcal{C}$ is simply connected (in other words, its boundary is a single loop) and $\mathcal{S} \cap \mathcal{C}$ is a tree.*

Proof: A descent path cannot cross an edge of $\mathcal{S}$, and thus $\mathcal{S} \cap \mathcal{C}$ is connected. Assume that $\mathcal{S} \cap \mathcal{C}$ is not a tree, which means that it contains a cycle C. So there is a face of $\mathcal{S}$ in the interior of C, and this face necessarily has an edge on the boundary of $\mathcal{P}$, therefore there is a hole H in $\mathcal{P} \cap C$. Without loss of generality, we can assume that there is a spanning tree edge ℓ connecting H to a hole/outer boundary H' outside $\mathcal{C}$. Thus, ℓ intersects an edge e on C. We claim that e is not the projection of a valley. If not, consider the endpoint v of e that is the projection of the lower endpoint of the valley. By our non degeneracy assumptions (Section 3.2), it implies that v is a reflex vertex on the boundary of $\mathcal{P}$. So v has degree 1 in $\mathcal{S}$, contradicting that e lies on the cycle C. Therefore, the descent paths for the ridge point in e cut across C, so C cannot exist as a cycle, a contradiction. Now assume that $\mathcal{C}$ is not simply connected, and hence it contains a hole H. Then we can find a cycle in $\mathcal{S} \cap \mathcal{C}$ by walking along the boundaries of the cells of $\mathcal{S} \cap \mathcal{C}$ that are adjacent to H, a contradiction. □

We walk around the boundary of $\mathcal{C}$ once (the boundary is known given the hole/outer boundaries and the descent paths bounding $\mathcal{C}$) to generate the implicit representation $D_{\mathcal{C}}$ of $\mathcal{K}(\mathcal{S} \cap \mathcal{C})$. Finally, we run $skeleton(D_{\mathcal{C}})$ to compute $\mathcal{S} \cap \mathcal{C}$. Afterward, we return $\mathcal{S} = E \cup \bigcup_{\mathcal{C}} \mathcal{S} \cap \mathcal{C}$. The motorcycle graph can be computed in $O(r\sqrt{r}\log r)$ time by Theorem 1. So the total running time becomes $O(r\sqrt{r}\log r + n\sqrt{h}\log^2 n)$.

Theorem 5 *The straight skeleton of a non–degenerate polygon with h holes and n vertices, among which r are reflex vertices, can be computed in $O(n\sqrt{h}\log^2 n + r\sqrt{r}\log r)$ expected time.*

6 Conclusion

As far as we know, no progress has been made on these two problems (computing motorcycle graphs and straight skeletons) since the conference version of this paper [12] was published. However,

- the current time bounds are still far from the only known lower bound $\Omega(n \log n)$,
- the only experimental results that have been published [21] give a quadratic running time,
- and, since then, new applications [4, 5, 6, 31, 32] have been found.

Any improvement on our motorcycle graph algorithm would yield a better time bound for computing the straight skeleton of a non–degenerate simple polygon (or, more generally, with $o(n)$ holes). It would also be interesting to generalize our results to degenerate polygons. It would require, first, to generalize our motorcycle graph algorithm to some cases where a new motorcycle appears after a collision (in order to account for vertex events), and second to show that our new characterization of the straight skeleton can be extended for degenerate polygons using this new type of motorcycle graphs.

References

[1] P. K. Agarwal. Partitioning arrangement of lines, i: An efficient deterministic algorithm. *Discrete Comput. Geom.*, 5:449–483, 1990.

[2] O. Aichholzer and F. Aurenhammer. Straight skeletons for general polygonal figures in the plane. In *Int. Conf. Comput. Comb.*, pages 117–126, 1996.

[3] O. Aichholzer, F. Aurenhammer, D. Alberts, and B. Gärtner. A novel type of skeleton for polygons. *J. Univ. Comp. Sci.*, 1(12):752–761, 1995.

[4] O. Aichholzer, F. Aurenhammer, and B. Palop. Quickest paths, straight skeletons and the city voronoi diagram. *Discrete Comput. Geom.*, 5(1):17–35, 2004.

[5] G. Barequet, M. T. Goodrich, A. Levi-Steiner, and D. Steiner. Straight-skeleton based contour interpolation. In *Proc. 14th ACM-SIAM Symp. Discrete Alg.*, pages 119–127, 2003.

[6] G. Barequet and E. Yakersberg. Morphing between shapes by using their straight skeletons. In *Proc. 19th ACM Symp. Comput. Geom.*, pages 378–379, 2003.

[7] C. Brenner. Towards fully automatic generation of city models. In *Proc. XIXth ISPRS Congress*, volume 33–B3 of *Internat. Arch. Photogramm. Remote Sensing*, pages 85–92, 2000.

[8] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom.*, 9:145–158, 1993.

[9] B. Chazelle and J. Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10(3):229–249, 1990.

[10] B. Chazelle and L. Guibas. Visibility and intersection problems in plane geometry. *Discrete Comput. Geom.*, 4:551–581, 1989.

[11] B. Chazelle, D. Liu, and A. Magen. Sublinear geometric algorithms. In *Proc. 35th ACM Symp. Theory Comput.*, pages 531–540. ACM Press, 2003.

[12] S.-W. Cheng and A. Vigneron. Motorcycle graphs and straight skeletons. In *Proc. 13th Annual ACM-SIAM Symp. Discrete Alg.*, pages 156–165, 2002.

[13] F. Chin, J. Snoeyink, and C. A. Wang. Finding the medial axis of a simple polygon in linear time. *Discrete Comput. Geom.*, 21(3):405–420, 1999.

[14] F. Cloppet, J.-M. Oliva, and G. Stamon. Angular bisector network, a simplified generalized voronoi diagram: Application to processing complex intersections in biomedical images. *IEEE Trans. Pat. Analysis Machine Intel.*, 22(1), 2000.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2nd edition, 2001.

[16] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition, 2000.

[17] E. D. Demaine, M. L. Demaine, and A. Lubiw. Folding and cutting paper. In *Japan Conf. Discrete Comput. Geom.*, pages 104–118, 1998.

[18] E. D. Demaine, M. L. Demaine, and A. Lubiw. Folding and one straight cut suffice. In *Proc. 10th Annu. ACM–SIAM Sympos. Discrete Alg.*, pages 891–892, 1999.

[19] E. D. Demaine, M. L. Demaine, and J. S. B. Mitchell. Folding flat silhouettes and wrapping polyhedral packages: New results in computational origami. In *Proc. 15th ACM Symp. Comput. Geom.*, pages 105–114, 1999.

[20] D. Eppstein and J. Erickson. Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. *Discrete Comput. Geom.*, 22:569–592, 1999.

[21] P. Felkel and Š. Obdržálek. Straight skeleton implementation. In *Proc. 14th Spring Conf. Comp. Graphics*, pages 210–218, 1998.

[22] P. Felkel and Š. Obdržálek. Improvement of Oliva's Algorithm for Surface Reconstruction from Contours. In *Proc. 15th Spring Conf. Comp. Graphics*, pages 254–263, 1999.

[23] S. Har-Peled. Constructing planar cuttings in theory and practice. *SIAM J. Comput.*, 29(6):2016–2039, 2000.

[24] J. Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Inf. Proc. Letters*, 33(4):169–174, 1989.

[25] J. Hershberger and S. Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *J. Algorithms*, 18(3):403–431, 1995.

[26] R. G. Laycock and A. M. Day. Automatically generating large urban environments based on the footprint data of buildings. In *Proc. 8th ACM Symp. Solid model. appl.*, pages 346–351, 2003.

[27] J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8:315–334, 1992.

[28] J. Matoušek. Construction of ϵ-nets. *Discrete Comput. Geom.*, 5:427–448, 1990.

[29] J. Matoušek. Cutting hyperplane arrangements. *Discrete Comput. Geom.*, 6:385–406, 1991.

[30] J.-M. Oliva, M. Perrin, and S. Coquillart. 3D reconstruction of complex polyhedral shapes from contours using a simplified generalized voronoi diagram. *Comp. Graphics Forum*, 15(3):397–408, 1996.

[31] M. Tanase and R. C. Veltkamp. Polygon decomposition based on the straight line skeleton. In *Proc. 19th ACM Symp.Comput. Geom.*, pages 58–67, 2003.

[32] M. Tanase and R. C. Veltkamp. A straight skeleton approximating the medial axis. In *Proc. 12th Euro. Symp. Alg.*, pages 809–821, 2004.

[33] R. Tarjan. *Data Structures and Network Algorithms*. SIAM, Philadelphia, PA, 1983.