Algorithmica (2007) 47: 159–182
DOI: 10.1007/s00453-006-1229-7

Algorithmica

Motorcycle Graphs and Straight Skeletons[1]

Siu-Wing Cheng[2] and Antoine Vigneron[3]

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 nondegenerate 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 nondegenerate 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 nondegenerate 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. [3], [4] 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 Figures 1 and 3.) As opposed to the widely used medial axis [14] (see Figure 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 finding offset polygons, known as mitered offset lines, which is a standard operation in computer-aided design [22]. (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 the same slope and that has one face per wall. (See Figure 7(b).) There could be several possible answers to this problem [8]. 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 [4]. These nice properties have been successfully exploited in several applications: polyhedral surface reconstruction from cross sections [6], [24], [33], biomedical image processing [15], polygon decomposition [34] (in particular, for computer vision applications), computational origami [19]–[21], computing the city Voronoi diagram [5], morphing between

[1] 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).

[2] Department of Computer Science, Hong Kong University of Science & Technology, Clear Water Bay, Kowloon, Hong Kong. scheng@cs.ust.hk.

[3] Unité Mathématiques et Informatique Appliquées, INRA, Domaine de Vilvert, F-78352 Jouy-en-Josas cedex, France. antoine.vigneron@polytechnique.org.

Received March 19, 2005; revised August 31, 2005, and October 26, 2005. Communicated by J.-D. Boissonnat. Online publication January 24, 2007.

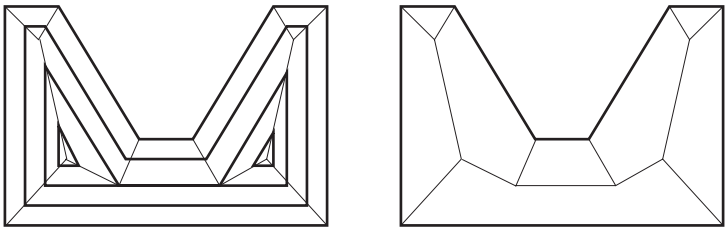

Fig. 1. The straight skeleton (on the right) is obtained by shrinking the initial polygon.

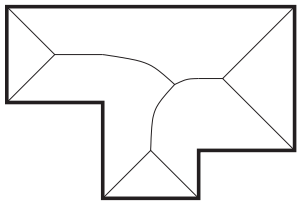

Fig. 2. The medial axis of an orthogonal polygon. Even in this simple case, it has parabolic edges.

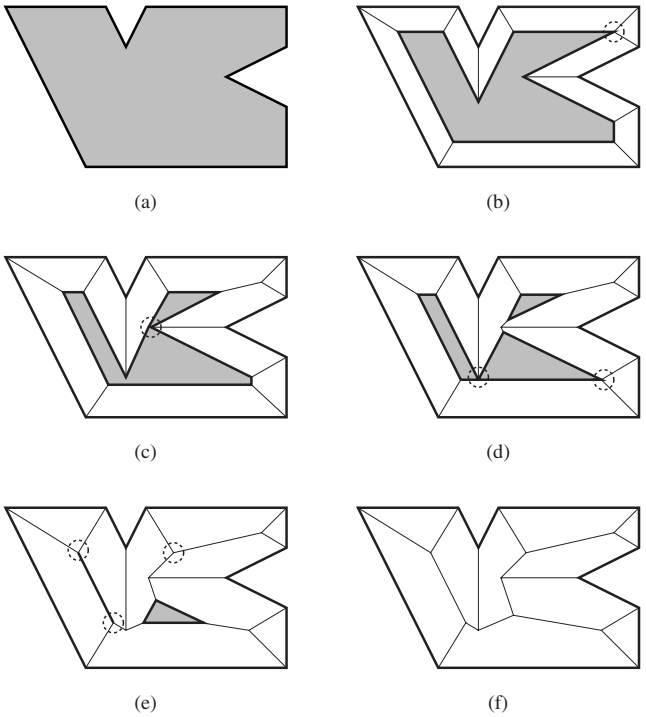

Fig. 3. The input polygon (a). During the shrinking process, an edge can disappear (b); this event is called an edge event. A split event (c) occurs when a reflex vertex hits an edge, splitting the polygon into two parts. Several events may occur simultaneously (d). The straight skeleton is obtained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

shapes [7], and automatic generation of city models [8], [29]. 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 [6], [15], [34].

The first algorithm by Aichholzer et al. [4] 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, Aichholzer and Aurenhammer [3] generalized the straight skeleton to polygons with holes and brought the space complexity down to $O(n)$. They also showed that the straight skeleton cannot be described as the projection of a lower envelope in a similar way as the medial axis. It explains why standard computational geometry techniques such as the randomized incremental construction do not apply directly. Eppstein and Erickson [22] gave the first subquadratic algorithm; its running time is $O(n^{17/11+\varepsilon})$ in the worst case, with a similar space complexity. They also present a reflex sensitive algorithm that runs in $O(n^{1+\varepsilon} + n^{8/11+\varepsilon} r^{9/11+\varepsilon})$ time, where r is the number of reflex (nonconvex) vertices of the polygon.

In this paper we give new connections between the straight skeleton and the motorcycle graph problem [22]. 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 $M_1, M_2, \ldots, M_n$ where 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 called the motorcycle graph. (See Figure 4.) Eppstein and Erickson [22] solved the motorcycle graph problem in $O(n^{17/11+\varepsilon})$ time, using advanced data structures for maintaining pairwise interaction and for ray shooting.

Our work has several contributions. First, we present an algorithm to compute a motorcycle graph in $O(n\sqrt{n} \log n)$ time. It is faster and simpler than the previous best known algorithm [22]. 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

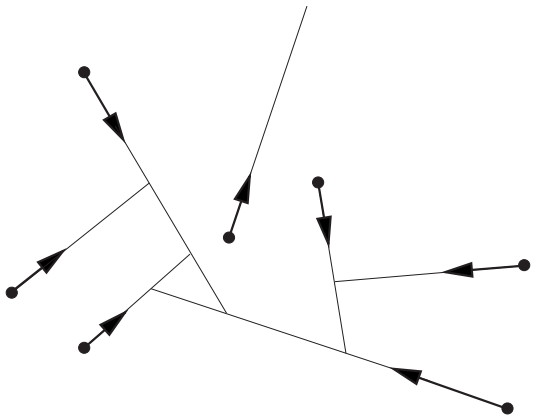

Fig. 4. A motorcycle graph.

polygon (possibly with holes). Third, we present an algorithm that computes the straight skeleton of a nondegenerate 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 nondegeneracy assumptions are explained in Section 3.2.) This reduction 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 nondegenerate 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 [22] 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 begins by building a list of potential crashes, each pair of motorcycles being considered independently of the rest. There can be 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.

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: either a $1/\sqrt{n}$-cutting (Section 2.1), or the partition induced by a random sample of $\sqrt{n}$ support lines (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 direction $\vec{v}_i$. 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 earliest crossing point $T_i \cap T_j$ such that M_j reaches it before M_i.

We will make use of *cuttings* [18]: 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], [9], [10], [30], [31]. We will employ a deterministic algorithm presented by Chazelle [9] 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 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; therefore we call it active in this situation.

The simulation progresses 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 $t' \leqslant t$ (i.e., motorcycle T_i crashes into T_j or into the track left by T_j).

All the switching events are 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. However, we can generate a subset of the impact events that includes all the actual collisions by maintaining 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 $\mathcal{C}$, together with the edges of $\mathcal{C}$.

To simplify the presentation, we first assume that no two trajectories are collinear. 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 $\mathcal{Q}$. We obtain the switching events 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 [9]. 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 Figure 5.) By the definition of cuttings, 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 [18].

In the main loop of the algorithm, we repeatedly extract from $\mathcal{Q}$ and process the event e with the smallest time stamp. This event e may not happen if one motorcycle involved has crashed earlier. We say that e is *relevant* if it does happen. It can be checked in

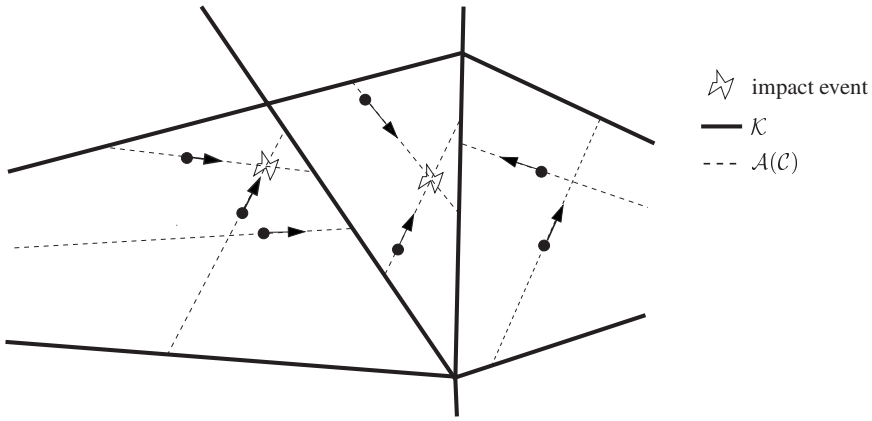

Fig. 5. The initialization step. Two impact events are queued here.

constant time as follows. If e is a switching event $(i, \mathcal{C}, t)$, we only need to check that motorcycle i has not crashed yet. Otherwise, if e is an impact event (i, j, t), we only need to check that motorcycle i has not crashed yet and that motorcycle j has reached or gone beyond the impact point $T_i \cap T_j$.

Now we assume that e is relevant. 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.

The following pseudo-code describes our motorcycle graph algorithm:

Algorithm *motorcycle_graph* ()

```
1. /* initialization */
2. compute K
3. insert all the switching events into Q
4. for all C in K, compute A(C) at time t = 0
5. for all C in K, insert the impact events corresponding to vertices of A(C)
   into Q
6. /* main loop */
7. while Q is not empty
8.     do extract the next event e from Q;
9.        if e is relevant
10.          then
11.             if e is a switching event (i, C, t)
12.                then /* M_i enters the cell C at time t. */
13.                   insert L_i ∩ C into A(C)
14.                   for each vertex of A(C) on L_i that is equal to T_i ∩ T_j
                      for some j
15.                      do compute t_i and t_j such that T_i(t_i) = T_j(t_j) =
                         = T_i ∩ T_j
16.                         if t_i ⩾ t_j
17.                            then insert the impact event (i, j, t_i) into Q
18.                         if t_j ⩾ t_i
19.                            then insert the impact event (j, i, t_j) into Q
20.             if e is an impact event (i, j, t)
21.                then /* M_i crashes at T_i ∩ T_j at time t. */
22.                   insert the edge connecting p_i and T_i ∩ T_j into the motor-
                      cycle graph.
```

The correctness of this algorithm follows from the fact that the movements of the motorcycles are simulated in chronological order.

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. The total time spent on extracting and deleting switching events during the simulation is also $O(n\sqrt{n}\log n)$. 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 [18] where the boundary of each cell is stored in a search structure that allows split operations in logarithmic time (for instance, a balanced binary tree [36]). Since each cell is convex, the search structure also allows reporting of the intersections between a line and the cell boundary in logarithmic time. Thus, updating a local arrangement can be done in $O(\log n)$ time 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. So it suffices to bound the total size of the local arrangements at the end of the simulation.

For each motorcycle M_i, let $\mathcal{C}_i$ denote the cell of $\mathcal{K}$ that contains its starting point p_i and let $\mathcal{C}'_i$ denote the cell where M_i crashes. 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 $\mathcal{C}_i \cup \mathcal{C}'_i$ (resp. $\mathcal{C}_j \cup \mathcal{C}'_j$). We may charge a vertex twice. Next, we prove that we charge each vertex 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 we charge this vertex 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 is charged to M_i or M_j.

In all, we charge each motorcycle 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 intersect 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 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. Thus, 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 events. 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 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 us to record their status and handle the events involving them without hurting our time bounds.

2.4. *A Simple Randomized Algorithm.* Our algorithm is simple and implementable. Using a practical planar-cutting algorithm [25], 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 boundaries of cells, and they 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 $L_i \cap L_j$. 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 $\sqrt{n}/(n-1)$, the probability that $n_i > c\sqrt{n}$ can easily be shown to be $e^{-\Omega(c)}$. (A similar analysis can be found in an article by Chazelle et al. [12, Lemma 1.1].) So the expected value of this number is $O(\sqrt{n})$ and, by the 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, as well as the algorithm by Eppstein and Erickson [22], handle without any difficulty the case where a motorcycle may run out of fuel at some point and stop. As opposed to Eppstein and Erickson's algorithm, our algorithms also handle the case where the speed of a motorcycle can vary (but it cannot move backward). On the other hand, Eppstein and Erickson's algorithm can handle dynamic insertion of motorcycles (at the current time of the simulation), but our algorithms cannot handle this case efficiently.

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. *Definitions.* 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 incident on an edge of $\mathcal{S}$. It is defined by means of a shrinking process [3], [4]. The edges of $\mathcal{P}$ move toward its interior at unit speed while remaining parallel

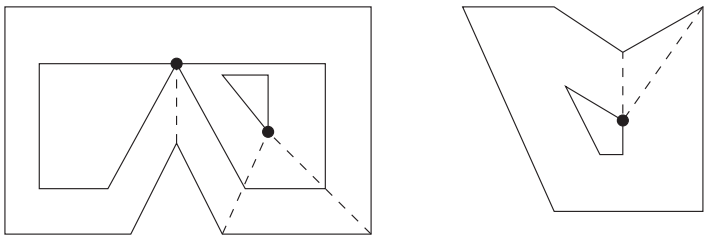

Fig. 6. On the left, a split event and an edge event. On the right, an edge event involving a reflex vertex.

to their initial positions. (See Figures 1, 3, and 7(c).) The traces of the polygon vertices during this shrinking process form the edges of $\mathcal{S}$. 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 shrinking polygon. First, the length of an edge may decrease to 0, and thus this edge disappears from the shrinking polygon. (See Figure 6.) These events are called *edge events*. Second, a vertex may hit an edge and the polygon splits into two parts afterward. (See Figure 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, tracing out a terrain $\mathcal{R}$ in three dimensions. We call $\mathcal{R}$ the *roof* of $\mathcal{P}$. (See Figure 7(b).) Then $\mathcal{S}$ is the vertical projection of the edges of the roof $\mathcal{R}$. The edges and faces of $\mathcal{R}$ are the lifted versions of the edges and faces of $\mathcal{K}(\mathcal{S})$. Each face of $\mathcal{R}$ makes an angle $\pi/4$ with the horizontal. A horizontal cut of $\mathcal{R}$ at height $z = t$ is the shrunken version of $\mathcal{P}$ at time t. (See Figure 7(c),(d).) We call the reflex edges of $\mathcal{R}$ *valleys*. (See Figure 8(c). In the figure 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. *Nondegeneracy 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 [22, Section 4.1]. So in this paper we assume that edge events and split events occur one at a time.

A third type of event can occur in degenerate cases. Two reflex vertices can collide, giving birth to a new reflex vertex. (See Figure 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 [22]. In order to avoid vertex events, we will make the following nondegeneracy assumption. Every reflex vertex of $\mathcal{P}$ is the lower endpoint of a valley of $\mathcal{R}$. We consider all the half-lines obtained by extending these valleys to $z = +\infty$. We will assume that no two such half-lines intersect. It means that no two reflex vertices of $\mathcal{P}$ will collide (in a vertex event). Since a new reflex vertex can only appear after a vertex event, it implies that no vertex event happens during the whole shrinking process.

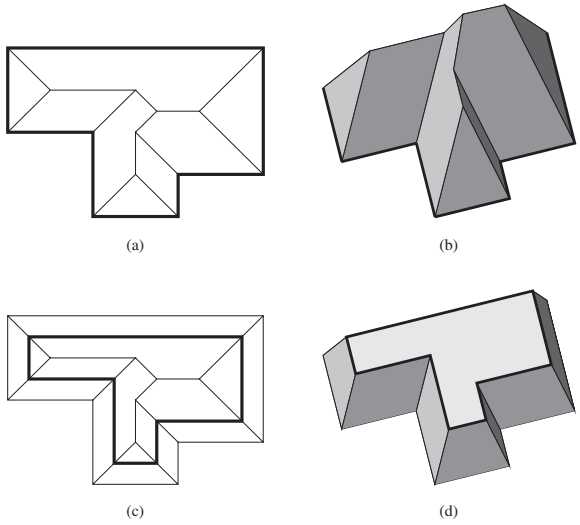

Fig. 7. (a) The input polygon $\mathcal{P}$ (thick lines) and its straight skeleton $\mathcal{S}$ (inside). (b) The roof $\mathcal{R}$ obtained by lifting the subdivision $\mathcal{K}(\mathcal{S})$. (c) The shrunken polygon at time t is a horizontal section of $\mathcal{R}$ at height t. (d) The roof $\mathcal{R}_t$ is the restriction of $\mathcal{R}$ to the vertical interval $[0, t]$. The top face of $\mathcal{R}_t$ is the shrunken version of $\mathcal{P}$ at time t.

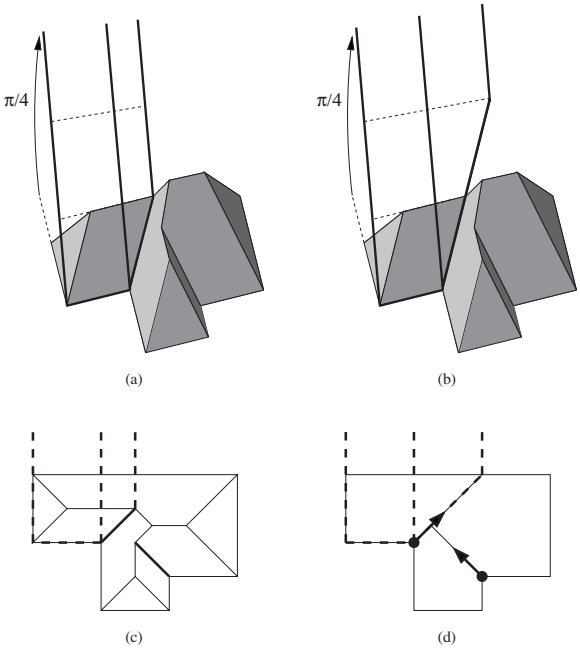

Fig. 8. An illustration of the different kinds of slabs. We show only the slabs associated with one particular edge adjacent to a reflex vertex. (a) The edge slab and the reflex slab. (b) The edge slab and the motorcycle slab. (c) The edge slab, the reflex slab, and the two valleys of $\mathcal{R}$ seen from above. (d) The motorcycle graph $\mathcal{G}$ associated with $\mathcal{P}$, the edge slab, and the motorcycle slab seen from above. The edges of $\widehat{\mathcal{G}}$ are longer than the associated valleys, thus a motorcycle slab contains the corresponding reflex slab.

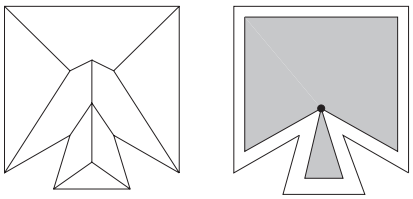

Fig. 9. A degenerate roof and a vertex event.

It follows from our nondegeneracy assumptions that all the valleys of $\mathcal{R}$ are adjacent to reflex vertices of $\mathcal{P}$. Another consequence is that each vertex of the straight skeleton is of degree one or three.

3.3. *Other Characterizations*. Eppstein and Erickson [22] expressed the roof $\mathcal{R}$ 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 Figure 8(a),(c).) The definition of the other reflex slab is similar with e replaced by e'.

THEOREM 2 [22]. *The roof $\mathcal{R}$ is the restriction of the lower envelope of the edge slabs and the reflex slabs to the space vertically above the polygon.*

In this paper, we give a new characterization similar to Theorem 2, except that we do not need to know the valleys of $\mathcal{R}$ 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 reciprocal 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 three dimensions 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 $\mathcal{R}$. 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 Figure 8(b),(d).) The definition of the other motorcycle slab is similar with e replaced by e'. In the following, we prove that the roof $\mathcal{R}$ 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 Figure* 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 $\mathcal{R}$ to the height interval $[0, t]$ by replacing the parts above the height t with flat patches. (See Figure 7(d).) We denote the restriction by $\mathcal{R}_t$. By the minimality of t, no valley of $\mathcal{R}_t$ 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 $\mathcal{R}_t \cap S$ is convex. Remember that the only reflex edges of $\mathcal{R}_t$ are the valleys. So, if there was a locally concave point x on $\mathcal{R}_t \cap S$, then either a valley of $\mathcal{R}_t$ would cross $\mathcal{R}_t$ 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 nondegeneracy assumptions. (See Section 3.2.)

Hence, $\mathcal{R}_t \cap S$ is a convex chain. Since $\widehat{f}$ is tangent to $\mathcal{R}_t$ at its lower endpoint, $\widehat{f}$ is on or above $\mathcal{R}_t \cap S$. Since the higher endpoint of $\widehat{e}$ is on $\mathcal{R}$, $\widehat{f}$ must be above the higher endpoint of $\widehat{e}$ because, by our nondegeneracy assumptions, $\widehat{f}$ and $\widehat{e}$ cannot intersect. It contradicts the fact that e crashes into f. □

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

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 $\mathcal{R}$ with the vertical slab with base e is a convex chain. Since $\widehat{e}$ is tangent to $\mathcal{R}$ at the lower endpoint of $\widehat{e}$, $\widehat{e}$ is on or above $\mathcal{R}$. □

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

THEOREM 3. *A nondegenerate roof $\mathcal{R}$ is the restriction of the lower envelope of the edge slabs and the motorcycle slabs to the space vertically above the polygon.*

PROOF. Let v be a valley. Let $\widehat{e}$ be its corresponding edge in $\widehat{\mathcal{G}}$. Let $S(v)$ denote the union of reflex slabs bounded below by v. Let $S(\widehat{e})$ denote the union of motorcycle slabs bounded below by $\widehat{e}$. Lemma 1 implies that $S(v) \subseteq S(\widehat{e})$. By Theorem 2, it suffices to prove that each point in $S(\widehat{e}) \backslash S(v)$ is on or above $\mathcal{R}$. Consider a point p in $\widehat{e} \backslash v$. Let r be a half-line that starts at p, has unit slope, and lie on a motorcycle slab bounded below by $\widehat{e}$. Let H_r be the half-plane obtained by sweeping r upward and downward to infinity. By Lemma 2, p is on or above $\mathcal{R}$. $\mathcal{R}$ 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 $\mathcal{R}$. □

Without our nondegeneracy assumption, this characterization of the roof is not necessarily true. The problem is that, at a vertex event, a new reflex vertex may appear, and this reflex vertex does not follow the track of a motorcycle. For instance, in Figure 9, the valley that is not adjacent to the polygon boundary does not follow the track of any

motorcycle, and this valley does not appear in the lower envelope of the motorcycle slabs.

4. Computing the Straight Skeleton of a Simple Polygon. We could compute the straight skeleton $\mathcal{S}$ of a polygon $\mathcal{P}$ by computing the motorcycle graph $\mathcal{G}$, then compute the lower envelope of the edge slabs and motorcycle slabs using a known algorithm. The lower envelope of a set of n (possibly intersecting) triangles in 3D can be computed in time $O(n^{2+\varepsilon})$ using a deterministic algorithm by Agarwal et al. [2] or using the lazy randomized incremental construction [16]. So we would get an $O(n^{2+\varepsilon})$ time bound for computing the straight skeleton. Another approach would be to compute the edges of the straight skeleton one by one, using ray-shooting queries in the set of edge slabs and motorcycle slabs. The data structure of de Berg et al. [17] allows answering ray-shooting queries among n (possibly intersecting) triangles in time $O(n^{3/4}\log n)$ after $O(n^{7/4+\varepsilon})$ preprocessing time. This approach yields a slightly better $O(n^{7/4+\varepsilon})$ time bound for computing a straight skeleton.

In this section, we give a faster randomized algorithm to compute the straight skeleton $\mathcal{S}$, given a simple polygon $\mathcal{P}$ and the associated motorcycle graph $\mathcal{G}$. Our algorithm runs in $O(n\log^2 n)$ expected time. It is based on the following idea: we can compute in $O(n\log n)$ time a vertical slice of the roof, as it can be seen as a lower envelope of line segments in two dimensions. As we shall see later, the ability to quickly construct vertical slices of the roof allows us to compute the section of the skeleton by a line through a random internal node, which allows us to design an efficient divide and conquer algorithm for computing the whole skeleton.

An *unrooted binary tree* is a tree whose nodes are of degree one or three. The nodes of degree one are called leaves and the nodes of degree three are called internal nodes. By our nondegeneracy assumptions (see Section 3.2), the straight skeleton $\mathcal{S}$ is an unrooted binary tree and all valleys of $\mathcal{R}$ are adjacent to reflex vertices of $\mathcal{P}$.

4.1. *Canonical Partition.* Aichholzer et al. [4] showed that the roof $\mathcal{R}$ has the so-called *gradient property*, that is, starting at any point of $\mathcal{R}$ 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 property follows directly from the characterizations of $\mathcal{R}$ 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, and 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 $\mathcal{R}$.

Let p be a point in $\mathcal{S}$. Let $\widehat{p}$ be the corresponding point on $\mathcal{R}$. The point p is a *ridge point* if $\widehat{p}$ does not lie in the interior of a reflex edge (valley). Let E denote a set of ridge points. For each $p \in E$, $\widehat{p}$ defines two or three paths of steepest descent on $\mathcal{R}$. 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 Figure 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

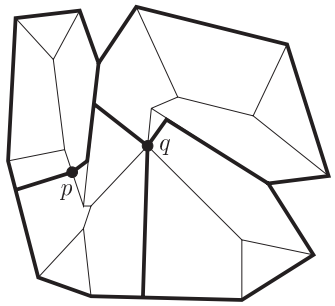

Fig. 10. The canonical partition induced by $\{p, q\}$.

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. Because no two descent paths can cross, this further subdivision of $\mathcal{C}$ yields the canonical partition of $\mathcal{P}$ induced by $E_1 \cup E_2$.

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 $\mathcal{R}$ 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 $[1 \cdots n_{\mathcal{C}}]$ 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.
- A pointer to the edge in $edges(\mathcal{C})$ to which each descent path leads.
- A 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 computation 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 $\mathcal{S} \cap \mathcal{C}$, 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 $\mathcal{R}$. 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. It 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 [26]. 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. 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 bounded by e is f. □

Here we show how we associate a vertex $v(f)$, which we call the *apex*, 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 Figure 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, the centroid whose coordinates are smallest in lexicographical order. In the rooted tree $\mathcal{S} \cap \mathcal{C}$, edges are directed from a child to its parent. The rooted tree $\mathcal{S} \cap \mathcal{C}$ is almost a binary tree, except that the root has three children. For each face f

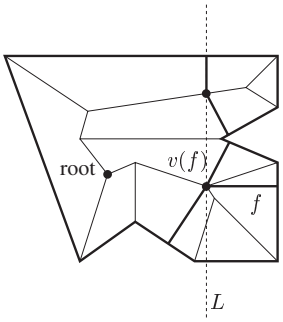

Fig. 11. The divide step. Here the initial cell $\mathcal{C}$ is the whole polygon. A face f is chosen at random. The line L is parallel to the y-axis and passes through the apex $v(f)$ of f. The interior ridge points of L (here, only two) define the canonical partition that will be used to divide $\mathcal{C}$.

of $\mathcal{K}(\mathcal{S} \cap \mathcal{C})$, we define the apex $v(f)$ to be the vertex of f closest to the root of $\mathcal{S} \cap \mathcal{C}$. Since each nonroot internal node u of the rooted tree $\mathcal{S} \cap \mathcal{C}$ has two children, u is the apex of exactly one face 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 the apex $v(f)$ in time linear in the number of vertices of f, which is $O(n_{\mathcal{C}})$.*

PROOF. It follows from definition that the two vertices adjacent to $v(f)$ 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 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 the apex $v(f)$ of a face f. (See Figure 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 three dimensions. 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 $\mathcal{R}$. As 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 $\mathcal{S} \cap \mathcal{C}$. As in the proof of 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 condition 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 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.* 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. pick a face f of $\mathcal{K}(\mathcal{S} \cap \mathcal{C})$ uniformly at random in *faces*($\mathcal{C}$)
4. compute an explicit description of f using Lemma 4
5. identify the apex $v(f)$ 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 $\{\mathcal{C}_1, \mathcal{C}_2, \ldots \mathcal{C}_k\}$ with respect to E using Lemma 3
12. recursively compute $\mathcal{S} \cap \mathcal{C}_i = \textit{skeleton}(D_{\mathcal{C}_i})$ for all $1 \leqslant i \leqslant k$
13. 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 $\mathcal{C}$
14. return $\mathcal{S} \cap \mathcal{C} = E \cup U_E \cup (\mathcal{S} \cap \mathcal{C}_1) \cup (\mathcal{S} \cap \mathcal{C}_2) \cup \ldots \cup (\mathcal{S} \cap \mathcal{C}_k)$

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. Line 13 is necessary to recover the edges of the skeleton that are on the boundary of some cell $\mathcal{C}_i$. Such edges appear when the boundary of $\mathcal{C}_i$ follows a valley. (See for instance Figure 10.) This union, as well as the result of line 14, 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 section, we will bound this number 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 12. Each individual step takes $O(n_{\mathcal{C}} \log n_{\mathcal{C}})$ time. We first show that $\max_{1 \leqslant i \leqslant k} n_{\mathcal{C}_i} \leqslant 3n_{\mathcal{C}}/4$ holds with probability at least $1/3$. 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 $n_{\mathcal{C}} - 3$ nonroot 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 nonroot 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 14 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 can now apply the same analysis as for quicksort (see the book by Kleinberg and Tardos [28, page 733]), the only difference being that our partitioning step is done in $O(n_{\mathcal{C}} \log n_{\mathcal{C}})$ time instead of $O(n_{\mathcal{C}})$ for quicksort. It yields an overall $O(n \log^2 n)$ expected time bound for our algorithm.

LEMMA 7. *Let $\mathcal{P}$ be a nondegenerate 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.*

Suppose that we are given a simple polygon $\mathcal{P}$ and we want to compute its straight skeleton. First we compute the point where each motorcycle associated with $\mathcal{P}$ runs out of fuel (i.e., hits the boundary of $\mathcal{P}$). It is a ray-shooting problem that can be solved in $O(n \log n)$ time [11], [27]. Then we compute the motorcycle graph $\mathcal{G}$ in $O(n\sqrt{n} \log n)$ time using the algorithm presented in Section 2. By Lemma 4, we obtain the following result when $\mathcal{P}$ is nondegenerate in the sense of Section 3.2 (and therefore $\mathcal{S}$ is an unrooted binary tree).

THEOREM 4. *The straight skeleton of a nondegenerate 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.*

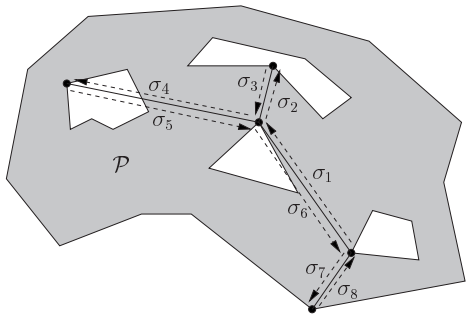

Fig. 12. The Eulerian tour $\sigma = \sigma_1\sigma_2 \cdots \sigma_8$ of the tree T.

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 nondegenerate 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 in $O(h^{1+\varepsilon})$ time a non-self-intersecting spanning tree T of crossing number $O(\sqrt{h})$ to connect these $h+1$ vertices [32]. 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. (See Figure 12.)

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. (See Figure 13(a).) This subdivision turns σ into an ordered sequence of

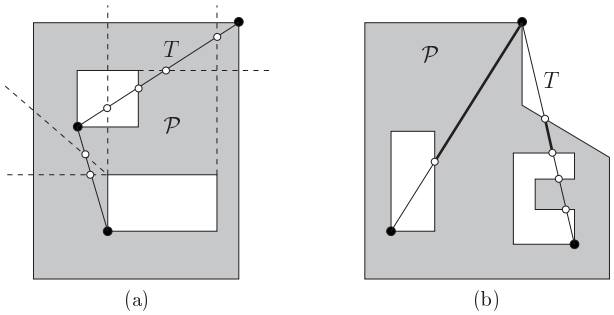

Fig. 13. (a) The edges of the tree T are subdivided by their intersections with the projections of the boundaries of the edge slabs and the motorcycle slabs and by their intersections with the boundary of $\mathcal{P}$. (b) The union of the links is represented by the thick line segments. (We did not show all the vertices in this figure.)

intervals. We organize a segment tree [18] 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 $O(h)$ tree-edge 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. The boundary of each slab intersects $O(\sqrt{h})$ standard intervals at the segment tree leaves, so each slab 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*. (See Figure 13(b).) 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 roof corresponding to $\mathcal{S}$. In the proof of Lemma 6, we intersect a vertical slab with the roof 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 $y \in anc(x)$; it 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, then the projected boundary of S intersects $I(z)$. There are $O(\sqrt{h})$ nodes at each segment tree 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 segment tree 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 $O(|chain(x)|\log|chain(x)|)$ [26]. 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 $I(x)$; it can be done by point location in the projection of $L(y)$. Then we simply walk from one face of $L(y)$ to another. It 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 $y \in anc(x)$. We can do it in $O(m\log m)$ time [26], 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. It 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 $\mathcal{P} \cap C$. 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 nondegeneracy assumptions (Section 3.2), v is a reflex vertex on the boundary of $\mathcal{P}$. So v is of 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. However, then 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 faces 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 $\mathcal{S} \cap \mathcal{C}$. After repeating the above for all cells in the canonical partition of $\mathcal{P}$, we merge the results to return $\mathcal{S}$ as in lines 13 and 14 of the pseudo-code in Section 4.4. 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 nondegenerate 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 [13] 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 [23] give a quadratic running time, and
- since then, new applications have been found [5]–[7], [34], [35].

Any improvement on our motorcycle graph algorithm would yield a better time bound for computing the straight skeleton of a nondegenerate 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, generalizing our motorcycle graph algorithm to some cases where a new motorcycle appears after a collision (in order to account for vertex events), and second, showing that our new characterization of the straight skeleton can be extended to degenerate polygons using this new type of motorcycle graphs. We were not able to prove that this approach works (which we suggested earlier [13]), but we do not have a counterexample to show that it is flawed.

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

References

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

[2] P. K. Agarwal, O. Schwarzkopf, and M. Sharir. The overlay of lower envelopes and its applications. *Discrete Comput. Geom.*, 15:1–13, 1996.

[3] O. Aichholzer and F. Aurenhammer. Straight skeletons for general polygonal figures in the plane. In *Proc. 2nd Annu. Int. Conf. Comput. Comb.*, volume 1090 of LNCS, pages 117–126, 1996.

[4] 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.

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

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

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

[8] 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.

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

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

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

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

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

[14] 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.

[15] 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. Pattern Anal. Mach. Intell.*, 22(1):120–128, 2000.

[16] M. de Berg, K. Dobrindt, and O. Schwarzkopf. On lazy randomized incremental construction. *Discrete Comput. Geom.*, 14:261–286, 1995.

[17] M. de Berg, D. Halperin, M. Overmars, J. Snoeyink, and M. van Kreveld. Efficient ray shooting and hidden surface removal. *Algorithmica*, 12:30–53, 1994.

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

[19] E. D. Demaine, M. L. Demaine, and A. Lubiw. Folding and cutting paper. In *Proc. Japan Conf. Discrete Comput. Geom.*, volume 1763 of LNCS, pages 104–118, 1998.

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

[21] E. D. Demaine, M. L. Demaine, and J. S. B. Mitchell. Folding flat silhouettes and wrapping polyhedral packages: new results in computational origami. *Comput. Geom.*, 1(16):3–21, 2000.

[22] 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.

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

[24] 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.

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

[26] J. Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Inform. Process. Lett.*, 33(4):169–174, 1989.

[27] J. Hershberger and S. Suri. A pedestrian approach to ray shooting: shoot a ray, take a walk. *J. Algorithms*, 18(3):403–431, 1995.
[28] J. Kleinberg and É. Tardos. *Algorithm Design*. Pearson Studium, Boston, 2006.
[29] 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.
[30] J. Matoušek. Construction of ε-nets. *Discrete Comput. Geom.*, 5:427–448, 1990.
[31] J. Matoušek. Cutting hyperplane arrangements. *Discrete Comput. Geom.*, 6:385–406, 1991.
[32] J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8:315–334, 1992.
[33] J.-M. Oliva, M. Perrin, and S. Coquillart. 3D reconstruction of complex polyhedral shapes from contours using a simplified generalized Voronoi diagram. *Comput. Graphics Forum*, 15(3):397–408, 1996.
[34] M. Tanase and R. C. Veltkamp. Polygon decomposition based on the straight line skeleton. In *Proc. 19th Annu. Symp. Comput. Geom.*, pages 58–67, 2003.
[35] M. Tanase and R. C. Veltkamp. A straight skeleton approximating the medial axis. In *Proc. 12th Euro. Symp. Alg.*, pages 809–821, 2004.
[36] R. Tarjan. *Data Structures and Network Algorithms*. SIAM, Philadelphia, PA, 1983.