Computing Straight Skeletons and Motorcycle Graphs: Theory and Practice

Stefan Huber

June 2011

Submitted to the Faculty of Natural Sciences at the
University of Salzburg in partial fulfillment of the
requirements for the Doctoral degree.

Supervisor: Ao.Univ.Prof. DI Dr. Martin Held
Department of Computer Science
University of Salzburg, Austria

ABSTRACT

The straight skeleton is a geometric structure that is similar to generalized Voronoi diagrams. Straight skeletons were introduced to the field of computational geometry one and a half decades ago. Since then many industrial and academical applications emerged, such as the computation of mitered offset curves, automatic roof construction, solving fold-and-cut problems and the reconstruction of surfaces, to name the most prominent ones. However, there is a significant gap between the most efficient straight-skeleton algorithms and implementations, on one hand, and the best known lower runtime bounds on the other hand. The primary goal of this thesis is the development of an algorithm that is suitable for implementation and efficient in terms of time and space in order to make the advantages of straight skeletons available to real-world applications.

We start with investigations concerning upper and lower bounds on the number of so-called flip events that occur in the triangulation-based straight-skeleton algorithm by Aichholzer and Aurenhammer. In particular, we prove the existence of Steiner triangulations that are free of flip events. This result motivates a novel straight-skeleton algorithm for non-degenerate simple polygons that is based on the so-called motorcycle graph. In order to extend this algorithm to arbitrary planar straight-line graphs, we carefully generalize the motorcycle graph. This generalization leads to practical and theoretical applications: Firstly, we obtain an extension of the alternative characterization of straight skeletons by Cheng and Vigneron to planar straight-line graphs. Secondly, this characterization motivates a straight-skeleton algorithm that is based on 3D graphics hardware. Thirdly, the generalized motorcycle graph leads to a wavefront-type straight-skeleton algorithm for arbitrary planar straight-line graphs. Our algorithm is easy to implement, has a theoretical worst-case time complexity of $O(n^2 \log n)$ and operates in $O(n)$ space. Extensive runtime tests with our implementation Bone exhibit an actual runtime of $O(n \log n)$ on a database containing more than 13 500 datasets of different characteristics. In practice, this constitutes an improvement of a linear factor in time and space compared to the current state-of-the-art straight-skeleton code, which is shipped with the CGAL library. In particular, Bone performs up to 100 times faster than the current CGAL code on datasets with a few thousand vertices, requires significant less memory and accepts more general input.

Our straight-skeleton algorithm motivates the investigation of motorcycle graphs and their practical computation. We start with stochastic considerations of the average length of a motorcycles trace. The results obtained motivate a simple yet fast algorithm that employs geometric hashing. Runtime tests with our implementation Moca exhibit an $O(n \log n)$ runtime on the vast majority of our datasets. Finally, we revisit the geometric relation of straight skeletons and motorcycle graph. We present an algorithm that constructs planar straight-line graphs whose straight skeleton approximates any given motorcycle graph to an arbitrary precision. This algorithm finally leads to a P-completeness proof for straight skeletons of polygons with holes that is based on the P-completeness of motorcycle graphs.

ACKNOWLEDGMENTS

I would like to thank my parents, Stefan and Gabriele, for their unlimited and unconditional support throughout my entire life. Special thanks go to my girlfriend Christina for her constant support and understanding during my doctoral studies.

I also thank my supervisor Martin Held for his guidance and his extensive support. Further, I thank my colleagues and co-workers. In particular I thank Gerhard Mitterlechner and Roland Kwitt for their proofreading and valuable discussions.

Eventually, I thank the Austrian Science Fund (FWF) for supporting this thesis by funding project no. L367-N15.

CONTENTS

1 INTRODUCTION

Assume we are given a simple[1] polygon in the plane which constitutes the outer walls of a house in a ground plan. How can we automatically build a roof such that all faces of the roof have equal slope and no rain drop gets caught in a sink? Assume we are given a polygon with holes[2] which is to be milled out with a numerically controlled machine (NC machine). How do we compute so-called offset curves of the input such that sharp vertices of the input remain sharp for the offset curve?[3] Assume we are given a simple polygon drawn on a paper. How can we determine a series of complete folds of the paper and a subsequent single cut through the folded paper such that one of the resulting paper pieces has the shape of the polygon? These three problems share a common feature: they can be solved using the *straight skeleton*.

Roughly speaking, the straight skeleton of a simple polygon P is a tree-like skeleton structure that is similar to Voronoi diagrams of polygons, but consists of straight-line segments only. The definition of the straight skeleton is based on the so-called wavefront propagation process. In a nutshell, the straight skeleton of P consists of the set of points that are traced out by the vertices of P when the edges of P shrink inwards in a self-parallel manner and with constant speed. During this propagation process, some edges may vanish and the polygon may successively disintegrate into multiple parts, see Figure 1. Straight skeletons of simple polygons were introduced by Aichholzer et al. [AAAG95] and were subsequently generalized to planar straight-line graphs[4] by Aichholzer and Aurenhammer [AA96] about one and a half decades ago. Similar to Voronoi diagrams, straight skeletons possess a heterogeneous plenitude of applications and many of them appeared just in the past decade. The list of applications includes

- the computation of mitered offset curves and motion planning in computer-aided design and computer-aided manufacturing (CAD/CAM) [PC03];
- computing the quickest walking paths in a Manhattan-style city with a fast public transit by means of the city Voronoi diagram; computing critical areas in VLSI circuit models [AAP04, Pap98];
- automatic roof generation, terrain modeling and area collapsing within maps in geographical information systems (GIS) [AAAG95, AA96, LD03, MWH$^+$06, KW11, Hav05, HS08];

1 A polygon is called simple if it is not self-overlapping. That is, each vertex is incident to exactly two segments and two segments may only intersect at their endpoints.

2 A polygon with holes is a simple polygon P, from which a finite number of simple polygons is removed.

3 A robust, easy and well-known way to compute offset curves is based on Voronoi diagrams. However, Voronoi-based offset curves contain circular arcs around vertices that point inside the polygon.

4 A planar straight-line graph consists of a finite set of vertices in the plane which are connected by straight-line edges such that two edges may only intersect in their endpoints.

- solving fold-and-cut problems, computing hinged dissections of polyhedra and related problems in the field of mathematical origami [DO07, DDL98, DDM00, DDLS05];
- shape reconstruction and interpolation of contour lines in three-dimensional space [OPC96, BGLSS04];
- and polygon decompositions [TV04b].

Some of these applications are genuine to straight skeletons, like fold-and-cut problems or automatic roof construction. Other applications are inherited from generalizations of Voronoi diagrams, like many problems in the fields of CAD/CAM, VLSI and GIS.

In contrast to the large number of applications, we perceive a gap between available algorithms and implementations on the one hand and the best known lower time bound on the other hand. At the moment, the best known lower time bound for the computation of straight skeletons is $\Omega(n)$ for an n-vertex polygon and $\Omega(n \log n)$ for an n-vertex planar straight line graph. However, the currently fastest algorithm for arbitrary polygons and planar straight-line graphs is due to Eppstein and Erickson [EE99] and has a theoretical worst-case time complexity of $O(n^{17/11+\epsilon})$, where $\epsilon > 0$ is an arbitrary small number. The only straight-skeleton implementation available is shipped with the CGAL library [CGA] and was implemented by Cacciola [Cac04]. However, our experiments revealed that the underlying algorithm needs $O(n^2 \log n)$ time and $O(n^2)$ space for practical input data.

This thesis deals with theoretical properties and the practical computation of straight skeletons and motorcycle graphs. The goal is to develop a practice-minded straight-skeleton algorithm, which accepts arbitrary planar straight-line graphs as input, is efficient in time and space and easy to implement.

1.1 ORGANIZATION

In this chapter, we start with the definition of straight skeletons, the terrain model and the motorcycle graph in Section 1.2. We continue with a presentation of several applications of straight skeletons in Section 1.3. In Section 1.4, we review known algorithms and implementations of straight skeletons and motorcycle graphs and in Section 1.5, we discuss different approaches by which straight skeletons were generalized.

Chapter 2 is devoted to the computation of straight skeletons. In Section 2.1, we collect known geometric properties of the straight skeleton and present them along with their proofs in a unified formalism based on the definitions in Section 1.2. In Section 2.2, we analyze the triangulation-based algorithm by Aichholzer and Aurenhammer [AA98]. We prove a few results concerning the lower and upper bounds for the number of the so-called flip events and show that Steiner triangulations exist where no flip events occur.

This insight motivates a novel straight-skeleton algorithm for simple non-degenerate[5] polygons that is based on the motorcycle graph in Section 2.3. In order to extend this approach to arbitrary planar straight-line graphs, we carefully generalize the motorcycle graph in Section 2.4. Further, we prove two essential geometric properties for this generalization that are important for our algorithmic approach: (i) the input graph and the motorcycle

5 See Section 1.2.4 for the definition of the non-degeneracy assumption by Cheng and Vigneron.

graph tessellate the plane into convex faces and (ii) the generalized motorcycle graph covers the reflex arcs of the straight skeleton.

In addition, the generalized motorcycle graph permits an extension of Cheng and Vigneron's alternative characterization of straight skeletons to arbitrary planar straight-line graphs. Furthermore, this characterization motivates a straight-skeleton algorithm that employs 3D graphics hardware to approximately compute the straight skeleton.

In Section 2.5, we present a wavefront-type straight-skeleton algorithm for arbitrary planar straight-line graphs. Our implementation Bone runs in $O(n^2 \log n)$ time in the worst-case and uses $O(n)$ space. Experiments exhibit an actual $O(n \log n)$ runtime on real-world input. This constitutes a speed-up by a linear factor in time and space compared to the current state-of-the-art straight-skeleton code that is shipped with the CGAL library.

In Chapter 3 we have a closer look on the motorcycle graph. Our straight-skeleton implementation Bone requires a fast motorcycle graph implementation for practical input. We start our investigations with stochastic considerations of the average trace length in a motorcycle graph in Section 3.2. Motivated by the results we obtained, we present a simple motorcycle graph algorithm, which uses geometric hashing to speed up the computation, see Section 3.3. Runtime tests show an actual runtime of $O(n \log n)$ on practical input for our implementation Moca.

In Section 3.4, we further investigate the geometric relation between the straight skeleton and the motorcycle graph. We first show how to construct a planar straight-line graph whose straight skeleton approximates the motorcycle graph up to any given precision. Further, we present a simple algorithm that computes the motorcycle graph using the straight skeleton. This algorithm finally leads to a LOGSPACE-reduction of the motorcycle graph problem to the straight skeleton problem, which proves the P-completeness of straight skeletons of polygons with holes based on the P-completeness of the motorcycle graph.

1.2 PRELIMINARIES AND DEFINITIONS

1.2.1 The straight skeleton of a simple polygon

Aichholzer et al. [AAAG95] introduced the straight skeleton of simple polygons P by considering a so-called *wavefront-propagation process*. Each edge e of P sends out a wavefront which moves inwards at unit speed and is parallel to e. The wavefront of P can be thought to shrink in a self-parallel manner such that sharp corners at reflex[6] vertices of P remain sharp, see Figure 1. During this propagation process topological changes, so-called *events*, will occur: Edges collapse to zero length and the wavefront may be split into multiple parts. Each wavefront vertex moves along the bisector of two edges of P and, while it moves, it traces out a straight-line segment.

Definition 1.1 (straight skeleton, arcs, nodes, faces)**.** The *straight skeleton* $\mathcal{S}(P)$ of the polygon P comprises the straight-line segments that are traced out by the wavefront vertices.

6 A vertex v of a polygon is called reflex if the angle at v at the polygon side is at least $180°$. In computational geometry one rather speaks of "reflex" vertices than of "concave" vertices.

These straight-line segments and are called the *arcs* of $\mathcal{S}(P)$. The loci of the topological changes of the wavefront are called *nodes*. To each edge e of P belongs a *face* $f(e)$, which consists of all points that are swept by the wavefront edge that is sent out by e.

We illustrate the straight skeleton $\mathcal{S}(P)$ of a simple polygon P in Figure 1. Each node is incident to multiple arcs. The boundary of a face consists of arcs and the vertices of a face are nodes.

Lemma 1.2 ([AAAG95])**.** *The straight skeleton $\mathcal{S}(P)$ of a simple polygon P with n vertices is a tree. It consists $n-2$ nodes and $2n-3$ arcs and tessellates P into n faces.*

For the proof of this lemma we create a single node for each topological change, even if the resulting nodes coincide geometrically. In particular, we assume that each node has degree three.[7] For example, if P is a regular n-gon then $\mathcal{S}(P)$ has a star form, i. e. all arcs meet in the center point of P and multiple nodes with degree three are geometrically coincident.

Proof. The number of faces is n, since P comprises n edges. Next, we note that the face $f(e)$ of an edge e is connected because it is given by the set of points which are swept by a continuously moving wavefront edge. On the other hand, each point within P is reached by the wavefront after some time. Hence, P is tessellated by the faces of $\mathcal{S}(P)$ and $\mathcal{S}(P)$ consists of the boundaries of the faces. From that it follows that $\mathcal{S}(P)$ does not contain a cycle and is therefore a tree. The inner nodes of this tree are of degree 3 and the tree has n leaves. It follows that $\mathcal{S}(P)$ has $n-2$ nodes and $2n-3$ arcs. $\square$

Definition 1.3 (reflex/convex wavefront vertex)**.** A wavefront vertex v is *reflex* if the angle on the side where v propagates to is at least $180°$. Likewise, we define a *convex* wavefront vertex.

Let us discuss the different types of topological events that occur for the wavefront in more detail. We follow Aichholzer et al. [AAAG95], who distinguish two types of events, namely *edge events* and *split events*.

- **Edge event**: An edge event occurs when two neighboring convex vertices u and v of the wavefront meet. This event causes the wavefront edge e, which connects u and v, to collapse to zero length. The wavefront edge e is removed and the vertices u and v are merged into a new convex vertex with its own velocity, see Figure 2 (a).
- **Split event**: A split event occurs when a reflex vertex u of the wavefront meets an edge e of the wavefront. The vertex u splits the entire wavefront into polygonal parts. Each part keeps on propagating for its own, see Figure 2 (b).

  Split events can occur in interesting variations. In Figure 2 (c), a split event for the reflex vertex u and the edge e occurs, while at the same time the edge between u and v collapses to zero length. Note that in this case only one wavefront part remains after the split event. However, the question arises whether we would like to call this event an edge event as well. For subfigure (a) we observe that a local disk around the

7 Strictly speaking, we also have to assume that multi split events — i. e. multiple reflex arcs meeting in a point, see below — do not occur either. If we take multi split events into account then we get at most $n-2$ nodes and at most $2n-3$ arcs.

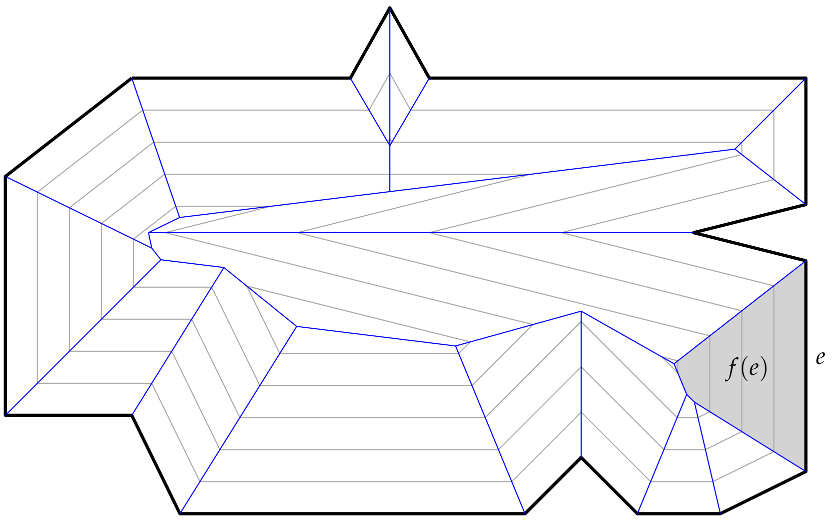


Figure 1: The straight skeleton $\mathcal{S}(P)$ (blue) of a simple polygon P (bold) is defined by a wavefront propagation process where the edges of P move inwards in a self-parallel manner. Five wavefronts at equally spaced points in time are shown in gray. The blue straight-line segments are called the arcs of $\mathcal{S}(P)$ and the common endpoints of arcs are called the nodes of $\mathcal{S}(P)$. We shade the face $f(e)$ of one edge e in light gray.

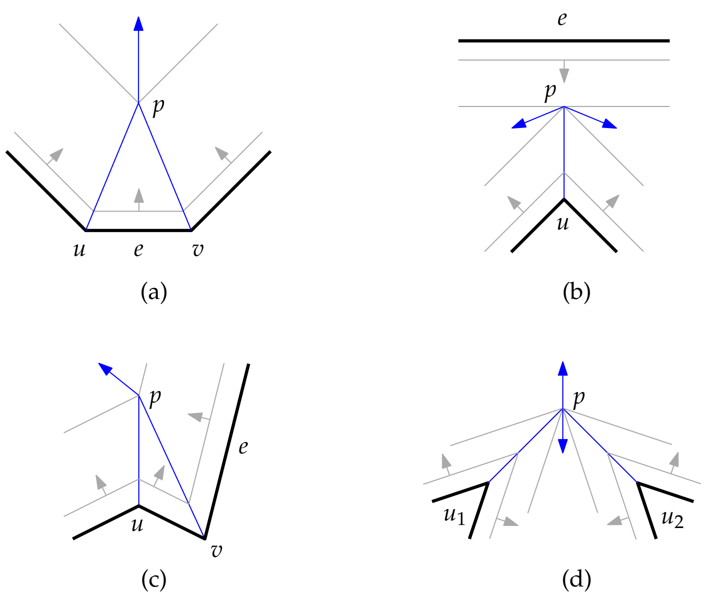


Figure 2: Two types of events occur during the wavefront propagation: edge and split events. (a) illustrates an edge event, (b) a simple split event, (c) a split event with an edge collapse combined, (d) a split event with two reflex vertices u_1 and u_2 involved. The arcs are depicted in blue and the wavefronts in gray.

point p, where the event happened, is tessellated into convex slices by the arcs of the straight skeleton. It is easy to see that this holds in general if u and v are convex. On the other hand, in subfigures (b) and (c), a tessellation of a local disk around p also contains a reflex slice. We would like to use this property as an indicator that a split event happened at p. For this reason, we call the event illustrated by subfigure (c) a split event.[8] As a consequence, edge events describe edge collapses between two convex vertices. Furthermore, if the straight-skeleton face of an edge — interpreted as a polygon — contains a reflex vertex, it has been generated by a split event.

In subfigures (b) and (c) only one reflex vertex u was involved in the split event. However, it could happen that two or more reflex wavefront vertices $u_1, \ldots, u_k$ meet each other in a common point p at the same time, see Figure 2 (d). Such situations occur easily for rectilinear[9] polygons. As a consequence the wavefront is split into k parts. Note that it is possible that a new reflex vertex emerges, as illustrated in Figure 2 (d). We will call events, where two or more reflex vertices meet simultaneously in a point a *multi split event*. Multi split events that cause a new reflex vertex to emerge are called *vertex events*.[10]

We want to remark that the taxonomy of the different classes of events is in flux within the literature of straight skeletons. For example, Tănase [TA09] pursues a different approach by considering three sub-classes of edge events and three sub-classes of split events. Following this taxonomy, the situation in Figure 2 (c) would be called *reflex edge annihilation*.

Using the taxonomy we presented above, the propagation of a wavefront always proceeds as follows: Every edge event reduces the number of wavefront edges by one. A split event splits the wavefront into polygonal parts and implies a hierarchy of nested polygons. Further, a split event reduces the number of reflex vertices at least by one. Even if a vertex event happens, at least two reflex vertices must have been involved in this event. Eventually, all nested parts in the hierarchy become convex polygons. The convex polygons are reduced by a series of edge events. Finally, a polygon vanishes by collapsing either to a point or to a straight-line segment. As an example for the first case one could imagine P to be a triangle, and for the second case a non-equilateral rectangle.

1.2.2 The straight skeleton of a planar straight-line graph

Aichholzer and Aurenhammer [AA96] generalized the concept of straight skeletons to planar straight-line graphs.[11] Assume we are given a planar straight-line graph G with no isolated[12] vertices. The basic idea to define the straight skeleton $\mathcal{S}(G)$ of G remains the same: Every edge e of G sends out two wavefront edges — one at each side of e — which travel at unit speed and stay parallel to the original edge e. However, the terminal[13] vertices of G play an interesting role, since it is a-priori unclear how the wavefront is expected to behave

8 Note that Eppstein and Erickson [EE99] call this event an edge event. Vyatkina [Vya09b] introduced the new name *sticking event*. Aichholzer et al. [AAAG95] did not discuss this case.
9 A polygon is rectilinear if its edges are axis-parallel.
10 This term was introduced by Eppstein and Erickson [EE99].
11 A later version of this paper was published in [AA98].
12 A vertex is called isolated if there are no edges incident to the vertex.
13 A terminal vertex, or "terminal" for short, is a vertex for which the number of incident edges of 1.

at such vertices. Aichholzer and Aurenhammer [AA96] decided that each terminal vertex v sends out a wavefront for it own, which is perpendicular to the single incident edge of v. As a result, the wavefront in the neighborhood of a terminal vertex v forms a rectangular cap around v, see Figure 3.

The terminology of *arcs* and *nodes* remains the same in this general setting. We still have the two types of events — *edge events* and *split events* — which occur during the propagation of the wavefronts and we define $\mathcal{S}(G)$ as the set of points that are traced out by the vertices of the wavefront. Note that a vertex v of G with degree $k \geq 2$ causes k wavefront vertices to emanate from v. Furthermore, every terminal vertex v causes two reflex wavefront vertices to emanate, see Figure 3.

Definition 1.4 (wavefront)**.** We denote by $\mathcal{W}(G,t)$ the *wavefront* of a planar straight-line graph G at some time $t \geq 0$.

We interpret $\mathcal{W}(G,t)$ for a fixed t as a 2-regular graph. A topological event happens when the planarity of $\mathcal{W}(G,t)$ is violated, including the case that two wavefront vertices meet. We interpret $\mathcal{W}(G,0)$ as the wavefront at time zero. While $\mathcal{W}(G,0)$ is geometrically overlapping with G, we assume that $\mathcal{W}(G,0)$ has — as a graph — the same topology as $\mathcal{W}(G,\epsilon)$ for a small $\epsilon > 0$. We observe that $\mathcal{S}(G)$ and G tessellate the plane into *faces* and that every face $f(e)$ belongs to an wavefront edge e: The face consists of all points in the plane which are swept by e, see Figure 3.

UNBOUNDED FACES AND INFINITE ARCS Note that certain faces will be unbounded since every point in the plane is eventually reached by the wavefront. By interpreting $\mathcal{S}(G)$ as a graph, it is reasonable to demand that each face corresponds to a cycle in $\mathcal{S}(G)$. Let us consider the last topological event that happened to the wavefront. After this event, the wavefront is a cycle that circumscribes G and all bounded faces of $\mathcal{S}(G)$. The vertices of this cycle trace the infinite arcs of $\mathcal{S}(G)$. Topologically, we add this cycle to $\mathcal{S}(G)$ and add the corresponding "infinite" arc to each "infinite" vertex of the cycle. Each "infinite" vertex has degree three. This interpretation of $\mathcal{S}(G)$ has practical advantages, e. g., when computing offset curves based on the straight skeleton, see Section 1.3.1.

ISOLATED VERTICES The definition of $\mathcal{S}(G)$ presented above assumes that G does not contain an isolated vertex. Aichholzer and Aurenhammer [AA96] mentioned that an isolated vertex could be approximated by a small straight-line segment. This must be done with some caution since $\mathcal{S}(G)$ does not continuously depend on G. For example, when multiple reflex arcs are incident to a point then a small perturbation of G would lead to a dramatically different straight skeleton $\mathcal{S}(G)$. See more on this issue in Section 2.5.3. However, one could simply define that an isolated vertex v emanates a wavefront which forms an axis-aligned square or any other polygon that has the property that its edges have an orthogonal distance of ϵ at time ϵ. That is, the supporting lines of the wavefront edges at time ϵ are tangential to the disk centered at v with radius ϵ. In fact, Demaine and O'Rourke [DO07] define the wavefronts emanated by an isolated vertex as an axis-aligned square in order to apply the straight skeleton for their fold-and-cut problems, see Section 1.3.3.

TERMINAL VERTICES In the same manner we could also alter the wavefront shape around terminal vertices. Consider a terminal vertex v and the two wavefront edges that emanate from the single incident edge e. Then we could basically place an arbitrary polygonal cap around v which connects the two wavefront edges from e. We only require that the segments of the cap have orthogonal distance ϵ at time ϵ. Nevertheless, using a rectangular cap — as introduced by Aichholzer and Aurenhammer — appears to be the most natural approach.

1.2.3 Roof and terrain model

Aichholzer et al. [AAAG95] presented an interpretation for the straight skeleton of simple polygons, which was extended to planar straight-line graphs by Aichholzer and Aurenhammer [AA96]. It turns out that this interpretation is a versatile tool in proofs of geometric properties of the straight skeleton. Further, it leads to one of the prominent applications of straight skeletons: roof construction and terrain modeling, see Section 1.3.2. The basic idea is to embed the wavefront propagation process in $\mathbb{R}^3$ in the following sense: the first two dimensions represent the plane spatial dimensions and the third dimension reflects the temporal dimension. The wavefront propagation now defines the so-called terrain $\mathcal{T}(G)$ of G as follows.

Definition 1.5 (terrain)**.** The *terrain* $\mathcal{T}(G)$ of G is defined by

$$\mathcal{T}(G) := \bigcup_{t \geq 0} \mathcal{W}(G,t) \times \{t\}. \tag{1.1}$$

Figure 4 illustrates the terrain $\mathcal{T}(G)$ of the graph G that is illustrated in Figure 3. Aichholzer et al. [AAAG95] used the term *roof* resp. *island* to indicate that $\mathcal{T}(P)$ of a simple polygon P has the following two interpretations. Firstly, $\mathcal{T}(P)$ can be interpreted as a particular roof of a house for which P models the footprint of the outer walls. Secondly, one can interpret P as the coastline of an island that has the shape of $\mathcal{T}(P)$. If the surrounding sea floods the island then the rising coastline has the shape of the rising wavefront in $\mathbb{R}^3$. In the case of planar straight-line graphs G Aichholzer and Aurenhammer [AA96] use the term *terrain* for $\mathcal{T}(G)$.

The terrain $\mathcal{T}(G)$ consists of plane facets that have a slope identical to the inverse of the propagation speed of the wavefront edges, which is 1. An edge of $\mathcal{T}(G)$ can either be convex or reflex. In the ordinary sense, we call an edge e of $\mathcal{T}(G)$ convex if the intersection of a small disk at any point in the relative interior of e with the points below $\mathcal{T}(G)$ is always convex. A reflex edge e of $\mathcal{T}(G)$ is defined likewise.

Definition 1.6 (reflex/convex arc, valley, ridge)**.** We call the arcs of $\mathcal{S}(G)$ which are traced out by reflex (convex) wavefront vertices *reflex arcs* (*convex arcs*). We call a reflex edge of $\mathcal{T}(G)$ a *valley* and a convex edge of $\mathcal{T}(G)$ a *ridge*.

Observation 1.7 ([AAAG95, AA98])**.** *The straight skeleton $\mathcal{S}(G)$ is the projection of the valleys and ridges of $\mathcal{T}(G)$ onto the plane $\mathbb{R}^2 \times \{0\}$. Moreover, the valleys correspond to the reflex arcs and the ridges correspond to the convex arcs.*

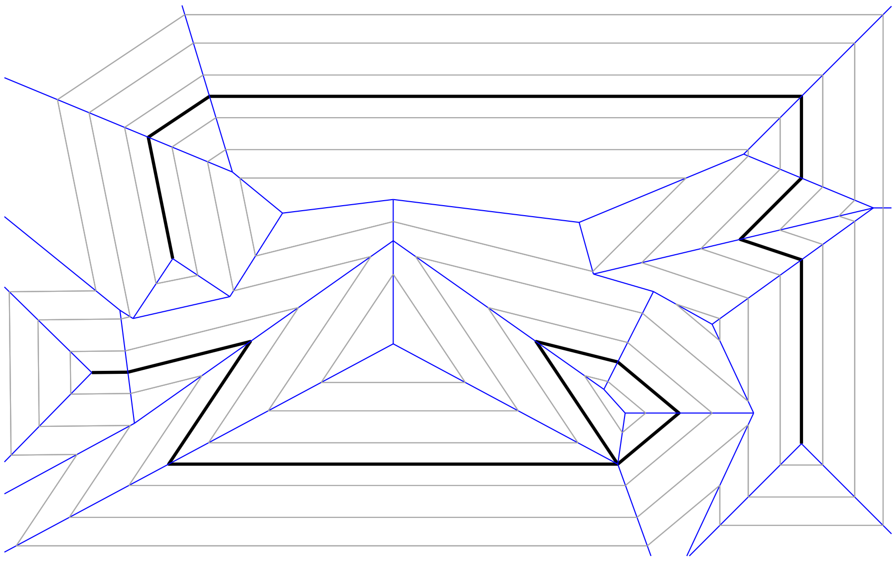


Figure 3: The straight skeleton $\mathcal{S}(G)$ (blue) of the planar straight-line graph G (bold). The wavefronts at three points in time are depicted in gray.

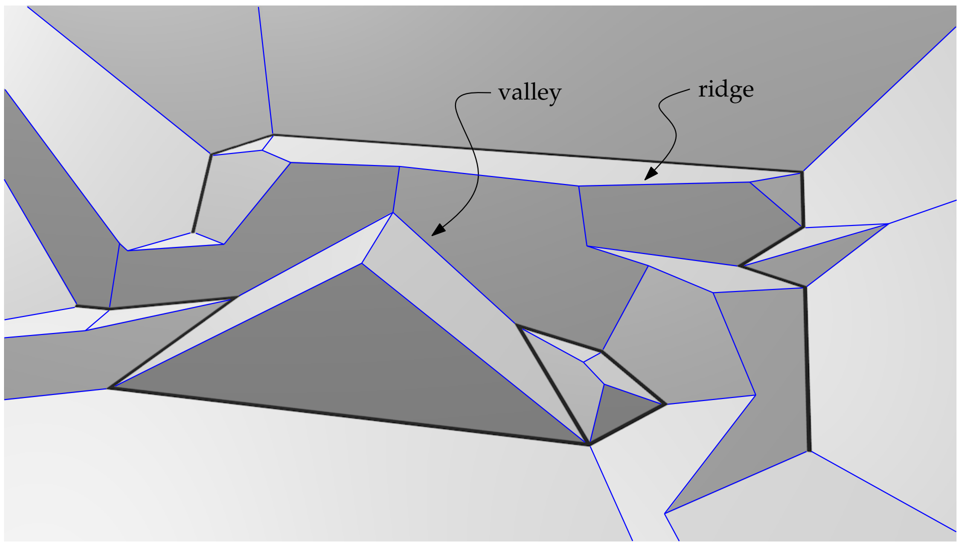


Figure 4: The terrain $\mathcal{T}(G)$ of the graph G which is illustrated in Figure 3. The ridges and valleys are in blue.

Following the notation of Cheng and Vigneron [CV07], we denote by $\hat{a}$ the edge of $\mathcal{T}(G)$ that corresponds to the arc a in $\mathcal{S}(G)$. Analogously, we denote by $\hat{f}(e)$ the facet of $\mathcal{T}(G)$ which corresponds to the face $f(e)$ of $\mathcal{S}(G)$.

ROOFS OF POLYGONS Aichholzer et al. [AAAG95] discussed the roof model of simple polygons P in more detail. They investigated more general roofs R on P which fulfill the property that each facet lies on a plane that contains an edge of P and has slope 1. The question arises whether such an R is equal to $\mathcal{T}(P)$. It turns out that this is not necessarily the case. However, R and $\mathcal{T}(P)$ are equal if all valleys of R are incident to P, or alternatively, if for any point $x \in R$ the path of the steepest descent leads to P. In other words, Aichholzer et al. [AAAG95] showed that among all roofs, $\mathcal{T}(P)$ has the peculiar property that it does not accumulate water when it is raining.

1.2.4 The motorcycle graph

A straight-forward approach to computing the straight skeleton is through the simulation of the propagating wavefront. While edge events can be handled in a relatively efficient way the opposite holds for split events, see Section 1.4.2.1. It turns out to be non-trivial to efficiently determine which reflex wavefront vertex crashes into which wavefront edge. Let us consider for a moment only the simultaneous movement of the reflex wavefront vertices. In order to compute the straight skeleton it is important to know which reflex wavefront vertex is cutting off the trajectory of an other reflex vertex by reaching the common crossing point of their trajectories earlier. In order to extract this sub problem of computing straight skeletons, Eppstein and Erickson [EE99] introduced the so-called motorcycle graph.

A *motorcycle* is a point moving with constant speed on a straight line. Let us consider n motorcycles $m_1, \ldots, m_n$, where each motorcycle m_i has its own constant velocity $v_i \in \mathbb{R}^2$ and a start point $p_i \in \mathbb{R}^2$, with $1 \leq i \leq n$. The trajectory $\{p_i + t \cdot v_i \ : \ t \geq 0\}$ is called the *track* of m_i. While a motorcycle moves it leaves a *trace* behind. When a motorcycle reaches the trace of another motorcycle then it stops driving — it *crashes* —, but its trace remains. Note that it is possible that motorcycles never crash. Following the terminology of Eppstein and Erickson [EE99] such motorcycles are said to have *escaped*. It is easy to see that there can be up to $\binom{n}{2}$ intersections among the motorcycle tracks. However, no two motorcycle traces intersect in both interiors, which leads to at most n intersections among the traces.

Definition 1.8 (motorcycle graph)**.**) The *motorcycle graph* $\mathcal{M}(m_1, \ldots, m_n)$ of the motorcycles $m_1, \ldots, m_n$ is defined by the arrangement of the motorcycles traces.

In Figure 5 (a) we illustrate the motorcycle graph $\mathcal{M}(m_1, \ldots, m_{10})$ of ten motorcycles. Cheng and Vigneron [CV07] presented a straight-skeleton algorithm for simple polygons P, which is based on the motorcycle graph, see Section 1.4.2.4. They first define a motorcycle graph induced by a polygon. The idea is that every reflex vertex v of P defines a motorcycle which starts at v and has the same velocity as the wavefront vertex which corresponds to v.

NON-DEGENERACY ASSUMPTION Cheng and Vigneron [CV07] explicitly exclude the case that vertex events appear for the wavefront of P. The authors call this the *non-degeneracy*

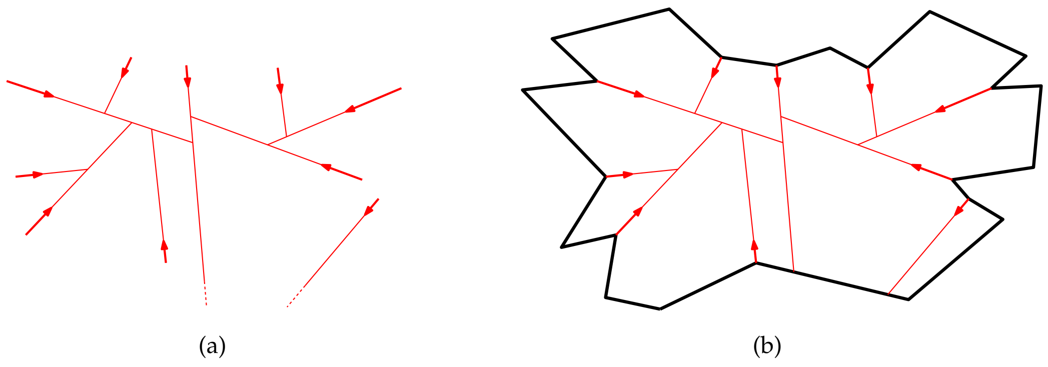


Figure 5: Left: The motorcycle graph $\mathcal{M}(m_1, \ldots, m_{10})$ of ten motorcycles. The velocities are represented by red arrows. Right: The motorcycle graph $\mathcal{M}(P)$, shown in red, induced by a simple polygon P (bold). Each reflex vertex of P emanates a motorcycle.

assumption. Eppstein and Erickson [EE99] remarked that perturbation techniques cannot be applied to transform these "degeneracies" to general cases. A small perturbation would change the straight skeleton drastically, see Section 2.5.3. On the other hand, such situations are very likely to occur, in particular if P contains collinear edges. Figure 1 shows a typical example. (We will be present a generalization of the motorcycle graph to arbitrary planar straight-line graphs in Section 2.5.) For the matter of simplicity we refer by the *non-degeneracy assumption* to the slightly more general assumption that no two motorcycles crash simultaneously into each other.[14]

In order to guarantee the correctness of the algorithm of Cheng and Vigneron [CV07], it is necessary to assume that the motorcycles *run out of fuel* when they reach an edge of P. We cover this circumstance by introducing the alternative concept of *walls*. We assume that the plane contains straight-line segments which model rigid walls. If a motorcycle reaches a wall then it crashes and its trace remains. Following our terminology, we define a motorcycle graph induced by a polygon by specifying the motorcycles and the walls.

Definition 1.9 (motorcycle graph induced by a simple polygon)**.** Let P denote a simple non-degenerate polygon. Each reflex vertex v emanates a motorcycle with the start point v and the same velocity as the corresponding wavefront vertex of v. Further, we consider the edges of P as walls. We denote by $\mathcal{M}(P)$ the resulting motorcycle graph and call $\mathcal{M}(P)$ the *motorcycle graph induced by P*.

Definition 1.10 (arm of a motorcycle)**.** Let m denote a motorcycle of $\mathcal{M}(P)$ that emanates from the vertex v. We call the two wavefront edges that are incident to the reflex wavefront vertex emanated from v, the *arms of m*. The one arm that is left of the track of m is called *left arm of m* and the other arm is called *right arm of m*.

We illustrate the motorcycle graph $\mathcal{M}(P)$ of a sample polygon P in Figure 5 (b). Cheng and Vigneron [CV07] also presented an extension of their algorithm to polygons with holes.

14 To be precise, Cheng and Vigneron assumed that no two valleys meet in a common point. Note that his assumption is slightly more general than the assumption that no vertex event happens: it can happen that two reflex wavefront vertices cause a multi split event such that no reflex vertex is emanated.

The motorcycle graph has to be extended accordingly: One has to introduce additional motorcycles at the convex vertices of the holes — which emanate reflex wavefront vertices within P — and one has to add the edges of the holes as walls, too.

1.3 APPLICATIONS

In the following section, we present several applications that appeared since the introduction of straight skeletons. The application of shape reconstruction by Oliva et al. [OPC96] even appeared at roughly the same time as straight skeletons and, in fact, the authors referred to the straight skeleton by a different name, namely *angular bisector networks* (ABN). In general, straight skeletons inherit many applications from Voronoi diagrams. Two typical applications — computing offset curves and terrain modeling — are immediately connected to the original definition of straight skeletons by Aichholzer et al. [AAAG95] and Aichholzer and Aurenhammer [AA96].

1.3.1 Mitered offset curves and NC-machining

Computational geometry has numerous applications in NC-machining. Computing offset curves is certainly one of the most important operations. Besides NC-machining, offset curves have a lot of further applications, like insetting/outsetting[15] paths in vector graphics editors and CAD software, computing tolerance domains around polygons or polygonal chains (e. g. for approximations within given bounds), computing curves parallel to a given curve (e. g. territorial domains on the sea defined by some fixed distance from a coastline), and so on.

First, we introduce some technical terms related to NC-machining. A workpiece that should be milled by an NC-machine is represented by a simple polygon P, possibly with holes. The tool in operation has the shape of a disk D_r with radius r and the origin as center. In the domain of NC-milling such a polygon P is often called *pocket* and the tool is called *cutter*. If we would move the center of the cutter along the boundary of P, we obviously remove too much material, see Figure 6 (a). The idea is to "shrink" the polygon P and then move the tool along the new boundary such that we obtain the desired workpiece, see Figure 6 (b). In mathematical terms, we want to determine a shape P' such that the Minkowski sum $P' + D_r := \{x + y \ : \ x \in P', y \in D_r\}$ equals P (except for certain portions, e. g., at the convex corners of P). The boundary of P' is called an *offset curve* of P with *offset distance* r. Note that there is only one continuous offset curve in Figure 6 (b) for that particular offset radius. If the tool did not leave the gap empty at the bottom of P then the bottom vertex of the hole would be cut off in this example.

At least two main questions related to computational geometry arise from the task of pocket machining: (i) how does one compute offset curves and (ii) how does one compute tool paths in order to remove the material in the interior of P? Held [Hel91] elaborated the computational problems related to pocket machining. He employs the Voronoi diagram of

15 Inkscape [Ink], a GPL-licensed vector graphics application, uses this terminology.

P in order to compute offset curves, where the offset curves are defined by the Minkowski difference $P - D_r$. Once the Voronoi diagram is available, this technique leads to a remarkably simple, efficient and robust method to compute offset curves, see [Hel91, HLA94]. Held [Hel91] also presented algorithms for two basic strategies in order to compute proper tool paths: *contour parallel* tool paths and *direction parallel* tool paths. Computing sophisticated tool paths is still a vital research area in these days. For example, Held and Spielberger [HS09] recently presented an algorithm that produces spiral tool paths which are suitable for high-speed machining. Computing an offset curve belongs to the first steps in their algorithm.

Voronoi-based offset curves consist of straight-line segments and circular arcs at the reflex vertices of P, see Figure 7 (a). Park and Chung [PC03] pointed out that such offset curves are undesirable if a high machining precision is required. The problem is that while the tool moves around a reflex vertex on a circular arc, it is permanently in touch with the reflex vertex, see Figure 7 (a). Oscillations of the workpiece or the tool during this time period lead to erosions of reflex vertices. Hence, they propose the so-called *mitered offset* curves, which are based on the straight skeleton instead of the Voronoi diagram. The basic idea is that sharp vertices of the polygon P should remain sharp in the offset curves, see Figure 7 (b). (Park and Chung [PC03] also mention that the offset curve based on the straight skeleton leads to long tool paths at sharp reflex vertices of the input. Hence, at reflex vertices with an interior angle of at least 1.5π the offset curves are trimmed by an additional line segment in order to shorten the tool path.)

The Voronoi diagram of simple polygons with holes — or more generally, of a planar straight-line graph — can be computed reliably in practice and implementations with an expected $O(n \log n)$ runtime exist, e. g., the Voronoi package Vroni by Held [Hel01]. In fact, Vroni also solves standard tasks for pocket machining, like computing offset curves, finding the maximum inscribed circle, or determining the medial axis[16]. Recently, Held and Huber [HH09a] extended the algorithms and the implementation behind Vroni to circular arcs. A nice byproduct of this extension is that the offset curves of straight-line polygons that are produced by Vroni, can serve as input to Vroni again.[17] As mentioned in the introduction of this chapter, the situation for the straight skeleton is clearly different. The only implementation available is the code by Cacciola [Cac04] within the CGAL library [CGA]. It accepts simple polygons with holes as input and exhibits a close-to quadratic runtime performance and memory footprint.

Park and Chung [PC03] circumvented the lack of efficient straight-skeleton algorithms by computing the mitered offset curves directly. Note that the mitered offset curve with offset distance t is equal to $\mathcal{W}(P, t)$. If no split event occurred until time t then one could simply compute the Voronoi diagram of P. After that one replaces the circular arcs of the offset curve by straight-line caps in order to obtain the mitered offset, without running into self-intersections of the offset curves. However, self intersections occur if the offset distance t is large enough such that a split event occurred until time t. Park and Chung [PC03] presented a relatively complex algorithm based on the concept of the so-called *pairwise interference detection*. Their algorithm basically puts offset segments parallel to the input edges and removes invalid portions of the offset curves afterwards.

16 The medial axis is a certain subset of the Voronoi diagram.

17 The former version of Vroni was able to handle circular arcs by approximating using straight-line segments.

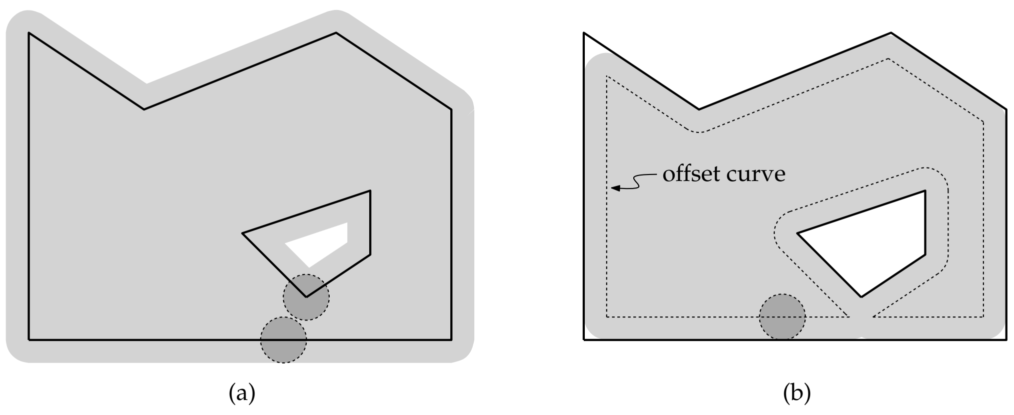


Figure 6: Milling a simple polygon P with a hole (bold). Left: Simply tracing the boundary of P with a tool removes too much material (shaded). Right: Tracing the offset curve (dashed) removes the correct amount of material, except for some remaining material at convex corners.

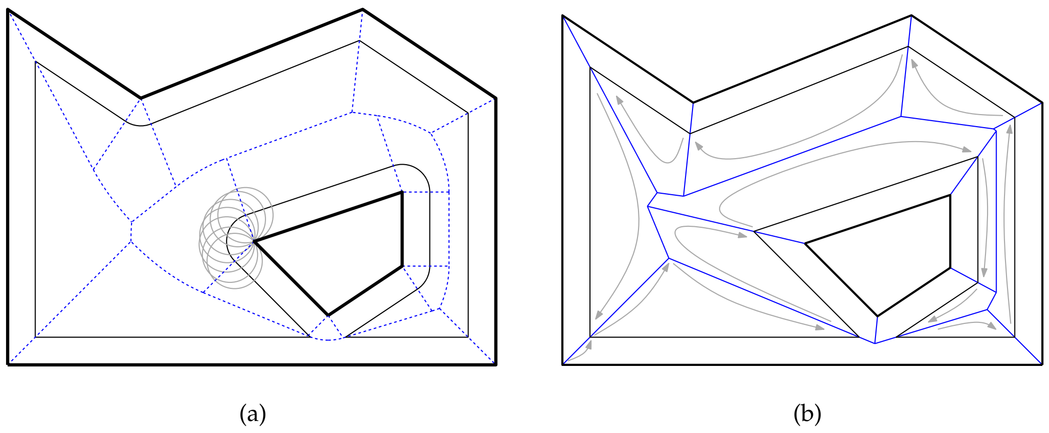


Figure 7: Two different approaches to obtain offset curves (thin) of the input (bold). Left: The approach based on the Voronoi diagram (dashed). The tool is shown in gray for several positions around are reflex vertex. Note that the tool remains in contact with the reflex vertex. Right: The approach based on the straight skeleton (dashed). The order, in which we traverse the straight skeleton to compute the offset curve, is sketched in gray.

Obviously, if the straight skeleton is available, the mitered offset curves are computed in an almost trivial way by simply traversing the straight skeleton in the very same fashion as it is done for Voronoi-based offset curves. In Figure 7 (b) we sketch the order in which the straight skeleton is traversed for a particular offset distance. After we start from an arbitrary input vertex, we basically follow the boundary of one incident straight-skeleton face after the other and stop at each time when we reach the desired offset distance on a straight-skeleton arc.

1.3.2 Building roofs and generating terrains

Automatic roof generation is an important task in 3D modeling, e. g., as part of an automatic city generator in a 3D modeling software. Let us consider a simple polygon P that represents the footprint of a building. How can we generate a realistic roof on top of the outer walls? A simple method to obtain a proper roof is to compute the roof model $\mathcal{T}(P)$ based on straight skeletons, see Section 1.2.3.

Laycock and Day [LD03] picked up this method and developed heuristic approaches in order to generate a larger variety of differently looking roofs. The original approach by interpreting the terrain $\mathcal{T}(P)$ above a simple polygon P as a roof results in the so-called *hip roof*. See Figure 8 for an example. Laycock and Day also describe how they obtain so-called *gable roofs*, *mansard roofs*, *gambrel roofs*, and *Dutch roofs*. For example, in order to generate a mansard roof, they consider the offset t until the first edge or split event happens and determine the offset at $0.85 \cdot t$, which is $\mathcal{W}(P, 0.85 \cdot t)$. The mansard roof consists of the original facets of $\mathcal{T}(P)$ which are restricted up to an height of $0.85 \cdot t$ and a top facet parallel to the ground plane. Straight skeletons became a general tool in order to generate sophisticated and realistically looking roofs, see [MWH$^+$06, KW11, Hav05] for further examples.

The automatic generation of mountain terrains in the neighborhood of waters is very similar to roof construction. Assume we are given the shape of a river or a lake and we want to model the surrounding terrain. If the boundary of this shape is given by a planar straight-line graph G then the terrain $\mathcal{T}(G)$ gives a realistic model for the surrounding terrain. In Figure 9 we show the generated terrain that illustrates the an part of the river Danube called "Schlögener Schlinge" in Austria. The generated terrain gives indeed a good approximation of the actual real-world scene.

The 3D models of Figure 8 and Figure 9 were generated by our straight-skeleton implementation Bone, see Section 2.5.3. Our implementation can be used to export the terrain $\mathcal{T}(G)$ of a planar straight-line graph G in a file, which is further processed by the free 3D modeling software Blender [Ble].

1.3.3 Mathematical origami and the fold-and-cut problem

Straight skeletons possess an interesting application in the field of mathematical origami: the *fold-and-cut problem*. Let us consider a simple polygon P drawn on a sheet of paper. (The polygon P could illustrate a flower or an animal.) We are allowed to apply a sequence of folds of the paper along straight lines. Finally, we take a scissor and cut the folded paper along a straight line into pieces. Which sequence of folds and which line for the final cut

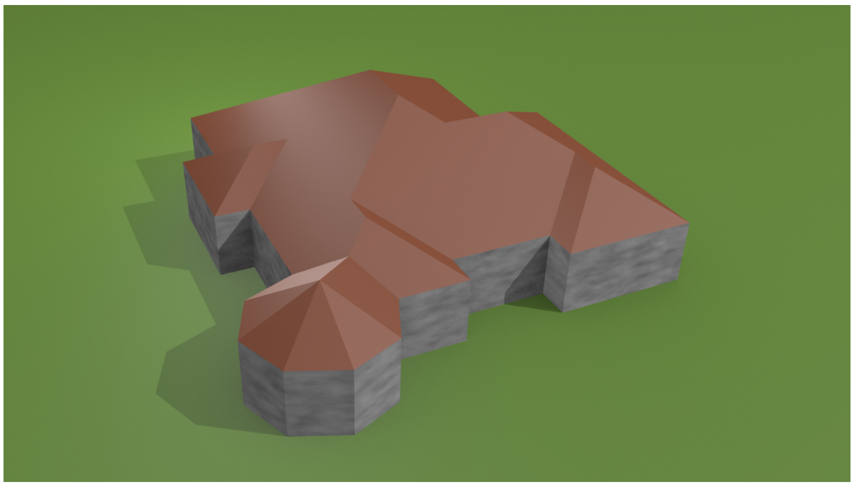


Figure 8: A hip roof generated by our straight-skeleton implementation Bone from the polygon that forms the footprint of the walls. Every facet of this roof has an identical slope.

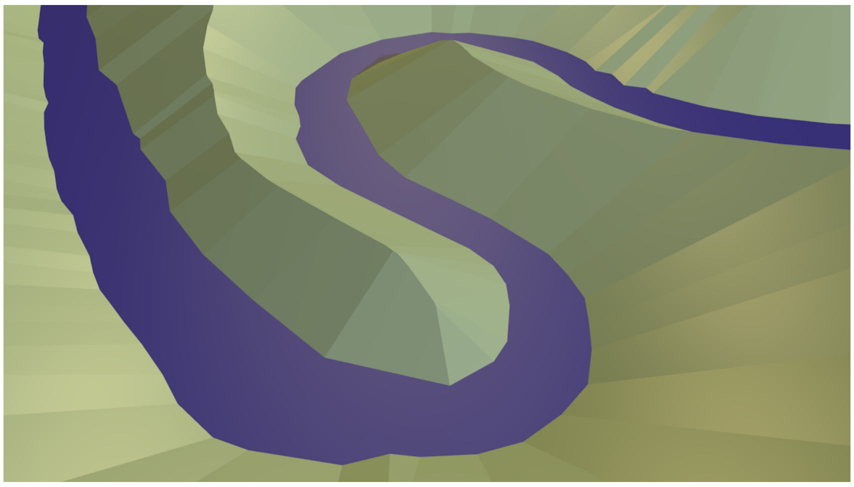


Figure 9: A terrain generated from the boundary of the blue river by our straight-skeleton implementation Bone. The figure illustrates the "Schlögener Schlinge"', which is a part of the river Danube in Austria (48°26′ 10″ N 13°51′ 50″ E). The boundary of the river is based on data obtained from OpenStreetMap [OSM].

do we have to apply in order to obtain a piece of paper that has the shape *P*? This problem, among others, is discussed in the book by Demaine and O'Rourke [DO07] in an illustrative and detailed fashion. They summarize the fold-and-cut problem as follows: (i) which shapes can be produced by such a fold-and-cut sequence and (ii) how can we compute a corresponding fold-and-cut sequence if the shape is given?

As described in [DO07], Chapter 17, the basic idea is to align the edges of *P* along straight lines by folding according to a so-called *crease pattern*. More precisely, the question whether a given polygon can be produced by a fold-and-cut sequence is equivalent to the question whether a crease pattern exists such that the edges of *P* can be arranged on a straight line and that any other point of the paper does not lie on this straight line. A cut through this line cuts exactly at the edges of *P*. The *universality result* states that every planar straight-line graph *G* can be cut by an appropriate fold-and-cut sequence [DDL98].

Demaine et al. [DDL98] presented an algorithm to compute crease patterns based on the straight skeleton. The input to their algorithm is a planar straight-line graph *G*. The motivation to use straight skeletons is that folding a paper at the bisector of two edges aligns both edges on a common straight line. Note that the arcs of the straight skeleton lie, by definition, on the bisectors of the defining pair of edges of *G*. The basic idea is that the straight skeleton almost poses an appropriate crease pattern (depending whether an arc is reflex or convex, the paper is folded in one or the other direction). As elaborated in [DO07], additional creases, so-called *perpendicular creases*, need to be introduced to obtain the final crease pattern for a given input graph *G*. Demaine et al. [DDL98] proved that a certain class of planar straight-line graphs can be obtained with a single cut using their crease-pattern algorithm based on straight skeletons. Crease patterns for arbitrary planar straight-line graphs can be computed using an algorithm based on disk packings. However, the crease patterns based on straight skeletons tend to be simpler [DO07].

Straight skeletons have also been applied to further problems in the field of mathematical origami and also to polyhedral wrapping problems, see [DDM00] for example.

1.3.4 Shape reconstruction and contour interpolation

Oliva et al. [OPC96] introduced an alternative version of the medial axis, which they called *angular bisector network* (ABN) and which turns out to be exactly the straight skeleton. Their motivation for this skeleton structure originates from the problem of 3D-surface reconstruction, for which they need to have a skeleton comprising straight-line segments only. They consider a sequence of cross-sections of a 3D-surface that lie on parallel planes. Each cross section consists of nested non-intersecting polygons defining so-called material and non-material domains. The problem is to compute a reconstruction of the original 3D-surface based on the set of cross sections. This is a typical problem in medical imaging.

The problem mentioned above is considered to be difficult in the presence of complex branches of the surface. Oliva et al. [OPC96] interpret the problem as an interpolation task between two consecutive sections. In the first step the polygonal shapes on two consecutive sections are projected on a parallel plane. Then they consider the symmetric difference of both shapes. This difference consists of nested polygons and each polygon comprises edges of both cross sections. In the next step they compute the straight skeleton of the difference shape and build a triangulation of the straight-skeleton faces. This triangulation is lifted to 3D in order to pose a patch of the 3D-surface between the two cross sections. The nodes of the straight skeleton lie on an intermediate layer between the two cross sections.

Barequet et al. [BGLSS04] presented a very similar approach. In contrast to [OPC96], they consider a different triangulation scheme and assign different heights to the vertices between the cross sections.

1.3.5 Polygon decomposition

Tănase and Veltkamp [TV04b] proposed an approach to polygon decompositions based on straight skeletons, where the split events define how the polygon is decomposed into sub polygons. They consider a reflex wavefront vertex v that led to a split event by crashing into the wavefront edge e. Then the polygon is decomposed by (i) the arc a that is traced out by v and (ii) the projection line of the endpoint of a that is orthogonal to e, until the boundary of $f(e)$ is hit. These two paths can be interpreted in the terrain model as two possible paths of a raindrop that starts at the endpoint of the lifted arc $\hat{a}$, see Section 1.2.3. Decompositions of this type form the first phase of their algorithm. In the second phase, further local simplifications of the decomposition are applied.

The decomposition steps of the first phase are similar to the randomized partitioning used by Cheng and Vigneron [CV07]. However, Cheng and Vigneron [CV07] do not investigate polygon decomposition schemes per se, but used their partition in order to devise a randomized straight-skeleton algorithm, see Section 1.4.2.4.

1.3.6 Area collapsing in geographic maps and centerlines of roads

In geographic information systems a common task is the simplification of maps by collapsing certain areas. Assume we are given a map of a landscape, including rivers, streets and municipalities. The rivers and streets are given as polygonal areas. The problem of area collapsing asks for a method to collapse a certain polygonal area A (e. g., a river or a street) to a one-dimensional structure by dividing A among its neighboring areas.

Haunert and Sester [HS08] introduced a method based on the straight skeleton $\mathcal{S}(A)$ of the polygon A. Each straight-skeleton face $f(e)$ is merged with the neighboring area that shares the edge e with A. In order to assign larger portions of A to larger neighboring areas, they employ the weighted straight skeleton, cf. Section 1.5.2. The weighted straight skeleton allows them to adjust the sizes of the straight-skeleton faces in A accordingly. Their method respects certain topological constraints, e. g., it maintains connectivity. For instance, if a river joins a lake then it is desirable that the collapsed river is still connected to the lake.

Haunert and Sester [HS08] also discuss the problem of computing the centerlines of roads. They consider a road network of a city, where each road is given by the polygonal area it occupies. How can one obtain a one-dimensional representation such that each road is represented by a polygonal chain? Haunert and Sester [HS08] compute the so-called centerline of a road by, roughly speaking, considering those straight-skeleton arcs which are not defined by two adjacent boundary edges of the road area. Special care is taken in the

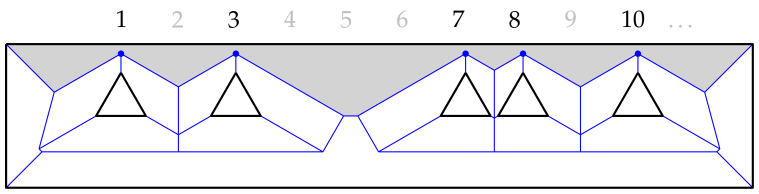


Figure 10: Sorting natural numbers can be reduced to the computation of the straight skeletons (blue) of a polygon with holes (bold). The nodes that are depicted by small disks correspond to the input numbers $\{1, 3, 7, 8, 10\}$.

presence of road junctions. If three or more roads meet in a junction it is rather unlikely that the four centerlines meet in a common straight-skeleton node. Haunert and Sester [HS08] presented a heuristic approach to detect such junctions and to connect the centerlines of the joining roads with a common junction node.

1.4 PRIOR WORK

1.4.1 Runtime bounds for the straight skeleton

To the best of our knowledge, the currently best known lower bound to compute the straight skeleton of a simple polygon with n vertices is $\Omega(n)$. In the convex case a simulation of the propagating wavefront gives us an almost optimal algorithm using $O(n \log n)$ time, see Section 1.4.2.1. For the more general case of monotone polygons, Das et al. [DMN+10] presented an $O(n \log n)$ algorithm. However, for arbitrary simple polygons, no algorithm that is even reasonably close to linear is known. Currently, the fastest algorithm is due to Eppstein and Erickson [EE99] with a worst-case runtime of $O(n^{17/11+\epsilon})$. This poses a high contrast to the situation for Voronoi diagrams, for which Chin and Snoeyink [CSW99] presented an optimal linear-time algorithm. Note that in the convex case the Voronoi diagram and the straight skeleton are coinciding and the algorithm by Chin and Snoeyink solves the straight-skeleton problem in optimal time as well. However, for convex polygons also a simpler Voronoi algorithm is known due to Aggarwal et al. [AGSS87].

For planar straight-line graphs and for polygons with holes, the best known lower bound is $\Omega(n \log n)$. This is easy to see, since we can simply reduce the sorting problem to straight skeletons. Assume n distinct natural numbers $a_1, \ldots, a_n$ are given, which have to be sorted. For any a_k, we place a small equilateral triangle that is hinged with its top vertex at the coordinates $(a_k, 0)$. The length of the edges of the triangles are considered to be less than 1, say 0.9. Then we put a box around the triangles such that the top vertices of the box have small y-coordinates, say 0.1. The box and the triangles form a simple polygon P with holes with $3n + 4$ vertices and can be constructed in $O(n)$ time. After computing $\mathcal{S}(P)$ we consider the face $f(e)$ of the top edge e of P that is within P. It is easy to see that the nodes that appear as reflex vertices of $f(e)$ correspond to the input $a_1, \ldots, a_n$ and occur in sorted order along the x-axis, see Figure 10. A simple traverse of $f(e)$ gives us the sorted sequence in linear time.

1.4.2 Algorithms for computing straight skeletons and motorcycle graphs

1.4.2.1 *Aichholzer et al., 1995*

When Aichholzer et al. [AAAG95] introduced straight skeletons of simple polygons P, they also presented an algorithm, which simulates the propagation of the wavefront in a discrete manner. Their algorithm maintains a priority queue Q which contains all potential edge events for each wavefront edge of P, prioritized by their occurrence time. Assume for a moment that P is convex, which means that only edge events occur. The algorithm fetches the earliest event from Q and processes it accordingly. That is, the incident vertices of the affected edge e are merged to a single vertex v. Note that the velocity of v is given by the two incident wavefront edges. Hence, the collapsing times of the two incident edges of v got invalid and one has to (i) re-compute them and (ii) update the priority queue accordingly. A single edge event affects only a constant number of entries within Q and can therefore be handled in $O(\log n)$ time. For convex polygons P this algorithm computes the straight skeleton in $O(n \log n)$ time.

In the presence of split events the situation gets more complicated. Efficiently determining the first split turns out to be non-trivial. Let v denote a reflex vertex of P. The problem is to determine in which wavefront edge the corresponding reflex wavefront vertex will crash. Simple ray-shooting does not solve the problem. The wavefront edge e, which is hit by the ray, could move off the ray during the wavefront propagation. Maintaining the wavefront edge that is hit by the ray does not solve the problem either: another reflex wavefront vertex from aside could cross the ray at any position. However, detecting such incidents seems to be costly.

The idea of Aichholzer et al. [AAAG95] is the following: Consider two consecutive edge events at time t' and $t > t'$ and assume that the wavefront is free of self-intersections until time t'. The wavefront becomes self-intersecting until t if and only if one or more split events happened in the time interval $[t', t]$. More importantly, Aichholzer et al. [AAAG95] were able to show that these split events can be determined from the wavefront $\mathcal{W}(P, t)$. In particular, they showed that the corresponding split events can be found and processed in $O(n \log n)$ time.

Testing whether a self-intersection is present at the k-th edge event is done in linear time using the triangulation algorithm by Chazelle [Cha91]. Applying exponential searching, the first split event can be determined at the costs of $O(\log k)$ intersection tests, i. e., in $O(n \log k)$ time, where the first split event happened just before the k-th edge event. Applying this algorithm recursively on each sub-polygon of $\mathcal{W}(P, t)$ after the first split event leads to a total runtime of $O(n^2 \log n)$. Denoting by $r \in O(n)$ the number of reflex vertices, one can further refine the runtime analysis to $O(nr \log n)$.

1.4.2.2 *Aichholzer and Aurenhammer, 1996*

Aichholzer and Aurenhammer [AA96, AA98] presented an algorithm for planar straight-line graphs which is based on a kinetic triangulation. Their algorithm accepts a planar straight-line graph G with n vertices as input. The basic idea is again to simulate the prop-

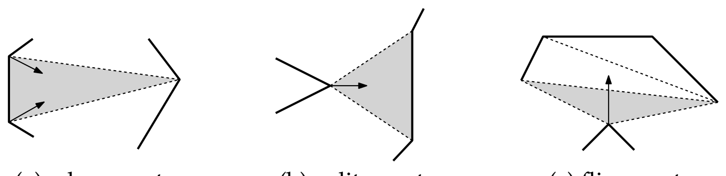


Figure 11: The three types of topological changes during the propagation of the triangulation.

agating wavefront, but to exploit the topological changes in a kinetic triangulation in order to determine the edge and split events.

The algorithm starts with an initial triangulation of G and keeps the area $\bigcup_{t' \geq t} \mathcal{W}(G, t')$ triangulated for all times $t \geq 0$. Aichholzer and Aurenhammer [AA96] also include infinite triangles to the triangulation such that the entire plane is triangulated at time zero. During the propagation of the wavefront, the triangulation keeps the area $\bigcup_{t' \geq t} \mathcal{W}(G, t')$ triangulated and at certain points in time the triangulation may change its topology: triangles collapse to a point or to a line. Whenever such an event occurs local modifications have to be applied in order to maintain a proper triangulation. The essential observation is that edge and split events of the wavefront correspond to a collapse of a triangle. Unfortunately, not every triangle collapse corresponds to an event of the wavefront. Aichholzer and Aurenhammer [AA96] distinguish the following types of events, as illustrated in Figure 11.

- **Edge event**: A triangle collapsed due to an edge event of the wavefront. The corresponding edge e collapsed to zero length and the one triangle having e as an edge collapsed with it.
- **Split event**: A triangle collapsed because a split event in the wavefront happened. In the simple case a reflex wavefront vertex moves into a wavefront edge e and the one triangle having e as an edge collapses thereby. In case of a multi split event two or more reflex wavefront vertices meet in a point and again certain triangles collapse.
- **Flip event**: A triangle collapsed because a wavefront vertex crosses an inner triangulation diagonal. The affected diagonal has to be flipped[18] in the triangulation in order to maintain a valid triangulation. This event does not immediately correspond to a topological change of the wavefront.

The algorithm of Aichholzer and Aurenhammer computes $\mathcal{S}(G)$ by keeping track of the topological changes within the triangulation. This again involves a priority queue Q containing the events. The runtime complexity depends on the number of events that occurred. The number of edge and split events corresponds to the number of nodes of $\mathcal{S}(G)$. According to Lemma 2.4, this number[19] is in $O(n)$. Each edge and split event involves the adaption of a certain number of wavefront vertices and, as a consequence, the re-calculation of the collapsing times of up to $O(n)$ triangles that are incident to these vertices. Summarizing, all edge and split events can be handled in $O(n^2 \log n)$ time.

18 If we remove the considered edge just before the flip event happens then we get a convex quadrilateral which can be triangulated in two ways. By an edge flip we mean that we triangulate the quadrilateral the other way.

19 Actually, in order to prove this Lemma, Aichholzer and Aurenhammer count the number of triangles in the triangulations, see the proof of Lemma 2.4.

However, finding a sophisticated upper bound for the number of flip events appears to be harder. First of all, a single flip event is handled in $O(\log n)$ time since it only involves the two triangles that share the affected triangulation diagonal. The upper bound of $O(n^3)$ for the number of flip events is relatively easy to see. Each wavefront vertex moves along a straight-line with constant speed. Consider the vertices p, q, r of a triangle Δ and denote by $p(t), q(t), r(t)$ their positions at time t, respectively. We denote by $u, v, w \in \mathbb{R}^2$ the velocities of p, q, r. The triangle Δ collapses if and only if the points p, q, r get collinear. That is

$$\begin{vmatrix} p_x(t) & q_x(t) & r_x(t) \\ p_y(t) & q_y(t) & r_y(t) \\ 1 & 1 & 1 \end{vmatrix} = \begin{vmatrix} p_x(0) + t \cdot u_x & q_x(0) + t \cdot v_x & r_x(0) + t \cdot w_x \\ p_y(0) + t \cdot u_y & q_y(0) + t \cdot v_y & r_y(0) + t \cdot w_y \\ 1 & 1 & 1 \end{vmatrix} = 0, \tag{1.2}$$

where the x- and y-coordinates are denoted by subscriptions. This quadratic equation in t is fulfilled for exactly zero, one, two or all values of t. As a consequence, a single triangle with vertices p, q, r collapses at most twice. Since we have $\binom{n}{3}$ possible triples of vertices the number of flip events is bounded by $O(n^3)$. Summarizing, the runtime complexity of the algorithm is in $O((n^2 + k) \log n)$, where $k \in O(n^3)$ denotes the number of flip events.

Interestingly, to the best of our knowledge, no input is known that exceeds a quadratic number of flip events. We will revisit the open question concerning this gap of a linear factor in Section 2.2. For some typical input data Aichholzer and Aurenhammer [AA96] observe an actual runtime close to $O(n \log n)$. However, no published runtime statistics are known.

1.4.2.3 *Eppstein and Erickson, 1999*

Eppstein and Erickson [EE99] presented a straight-skeleton algorithm for simple polygons, which is applicable to planar straight-line graphs and which can even be extended to compute the weighted straight skeleton, cf. Section 1.5.2. Let us recall the discussion in Section 1.4.2.1 concerning the determination of the split events. As elaborated by Aichholzer et al. [AAAG95], a simple ray-shooting from reflex vertices does not solve the problem of finding the next split event. However, one could consider for every propagating wavefront edge e the triangle (possibly infinite in size) that is bound by e at time zero and the trajectories of its incident wavefront vertices. The idea is to keep track of those pairs of reflex vertices v and wavefront edges e, where the potential split event of v and e takes place within the corresponding triangle of e. The next split event is given by closest pair in terms of the earliest split-event time.

Eppstein and Erickson [EE99] pursued this approach by employing very powerful closest-pair data structures. First of all, Eppstein and Erickson lift the problem to $\mathbb{R}^3$ in the same fashion as it is done for the terrain model $\mathcal{T}(G)$. Every reflex vertex v of $\mathcal{W}(G, 0)$ defines a ray in $\mathbb{R}^3$ starting at v and supports the corresponding ridge in $\mathcal{T}(G)$. At every initial wavefront edge e a corresponding (lifted) triangle, which supports $\hat{f}(e)$, is considered. The two other edges of the triangle are given by the supporting rays of the lifted trajectories of the incident wavefront vertices, see Figure 12. Note that if we consider the unweighted straight skeleton then the triangles have slope 1, whereas the rays have a slope of at most 1.

Eppstein and Erickson [EE99] consider a sweep plane, which sweeps $\mathbb{R}^3$ along the z-axis. Whenever the sweep plane reaches the top of a triangle, an edge event happens. When the intersection of a ray with a triangle is reached a split happened, including multi split

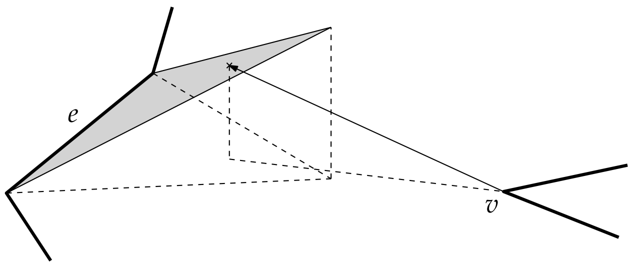


Figure 12: The triangle of an edge e and a ray of the reflex vertices v, as considered by the algorithm by Eppstein and Erickson [EE99]. The lowest intersection of a ray with a triangle gives the next split event.

events. For every edge and split event a constant number of triangles and rays is removed and added. Every multi split event involving k reflex wavefront vertices leads to the deletion and insertion of $O(k)$ triangles and rays. Eppstein and Erickson [EE99] maintain the triangles in a priority queue Q, prioritized by the height of their top vertex. The total costs for maintaining Q is in $O(n \log n)$. The essential part of the algorithm consists of determining the ray-triangle intersections.

Central aspects in the algorithm by Eppstein and Erickson [EE99] are range searching techniques that allow the application of time-space trade-offs. Assume we are in possession of a data structure that supports fast queries but has a high space consumption and, conversely, a data structure that has high query times but low space bounds. The idea is to mix those two data structure in a hierarchical fashion in order to obtain characteristics for query time and space consumption between the two original data structures. A survey by Agarwal and Erickson [AE99] discusses these techniques in detail.

Eppstein and Erickson [EE99] denote by s the space required by a data structure and express the time complexity as a function in s. One step towards their straight-skeleton algorithm is a data structure which itself is based on several other data structures and allows lowest intersection queries among n rays and triangles within $O(n^{1+\epsilon}/s^{1/4})$ time, for any $\epsilon > 0$. Another main ingredient is a data structure by Agarwal and Matoušek [AM94] in order to answer ray shooting queries, again in $O(n^{1+\epsilon}/s^{1/4})$ time. By balancing time and space complexities, Eppstein and Erickson finally obtain a straight-skeleton algorithm which uses $O(n^{8/5+\epsilon})$ time and space. They further note that this algorithm is also applicable in order to compute the weighted straight skeleton.

For the unweighted case, Eppstein and Erickson [EE99] prove Theorem 2.7 in order to devise a slightly faster algorithm, see Section 2.1. In this case, all triangles of the algorithm have identical slope 1. This fact and the theorem mentioned above can be exploited in order to use more efficient data structures for intersection queries between rays and triangles. Using these refinements the algorithm by Eppstein and Erickson [EE99] has a theoretical worst-case time and space complexity of $O(n^{1+\epsilon} + n^{8/11+\epsilon}r^{9/11+\epsilon})$ for the unweighted straight skeleton of planar straight-line graphs, where r denotes the number of reflex vertices.

Eppstein and Erickson also present an approach that uses a simpler data structure by Eppstein [Epp00] in order to maintain the closest-pair between a ray and a triangle. This data structure uses a quadtree structure on the matrix of the pairwise distances. Using this data structure, the weighted straight skeleton can be computed in $O(n \log n + nr)$ time and $O(nr)$ space and the unweighted straight skeleton can be computed in $O(n \log n + nr)$ time and $O(n + r^2)$ space. However, no implementations of these algorithms are known and, in particular, no runtime statistics have been published.

Similar techniques as sketched above are also used by Eppstein and Erickson [EE99] in order to compute the motorcycle graph of n motorcycles $m_1, \ldots, m_n$. The motorcycle graph was introduced in their paper in order to describe the essential sub problem of computing straight skeletons, see Section 1.2.4. The idea is that each motorcycle m_i defines a tilted ray in $\mathbb{R}^3$ and a vertical curtain above the ray. The intersection of a ray of the motorcycle m_i with a curtain of a motorcycle m_j corresponds to a crash of m_i into the trace of m_j. Their algorithm runs in $O(n^{17/11+\epsilon})$ time and space.

1.4.2.4 *Cheng and Vigneron, 2002*

Cheng and Vigneron [CV02, CV07] were the first to exploit the relationship between motorcycle graphs and straight skeletons in order to compute the straight skeleton. Their straight-skeleton algorithm accepts non-degenerate polygons P with holes as input.

First of all, Cheng and Vigneron compute the motorcycle graph $\mathcal{M}(P)$ induced by P. Let $m_1, \ldots, m_n$ denote n motorcycles. Further, we denote by p_i the start point and by v_i the velocity of the motorcycle m_i. They observed that the tracks, on which the n motorcycles drive, can have $\Theta(n^2)$ pairwise intersections, but only $O(n)$ among them actually correspond a crash. The challenge is to geometrically separate motorcycles that do not interact. At the first sight this appears to be difficult, since a single motorcycle can be very fast and cross the tracks of many motorcycles. The central idea of Cheng and Vigneron is to employ so-called ϵ-cuttings.

Chazelle [Cha04] explains ϵ-cuttings as follows. Let H denote a set of n hyperplanes in $\mathbb{R}^d$. An ϵ-cutting is a tessellation of $\mathbb{R}^d$ into non-overlapping simplices such that each simplex intersects at most ϵn hyperplanes of H. Chazelle [Cha93] presented an algorithm that runs in optimal time $O(n\epsilon^{1-d})$ and constructs a cutting of optimal size $O(\epsilon^{-d})$.

Cheng and Vigneron [CV07] use $1/\sqrt{n}$-cuttings in $\mathbb{R}^2$ on the supporting lines of the motorcycle tracks. Hence, every triangle of the cutting intersects at most $\sqrt{n}$ motorcycle tracks. The cutting of $O(n)$ size can be computed in $O(n\sqrt{n})$ time by Chazelle's algorithm [Cha93]. Once the cutting is computed, Cheng and Vigneron [CV07] simulate the movement of the motorcycles within the cutting. They distinguish two types of events:

- Switch event: A motorcycle reaches the boundary of a cutting-triangle and migrates to a neighboring triangle.
- Crash[20] event: A motorcycle crashes into the trace of another motorcycle.

The motorcycle graph can be computed independently within each cell. A priority queue Q contains each event and the events are processed in chronological order. First, they compute all potential switch events in advance and insert them all into the priority queue Q in $O(n\sqrt{n}\log n)$ time. Note that a single motorcycle may indeed cross $\omega(\sqrt{n})$ simplices, but the total number of switch events among all motorcycles is in $O(n\sqrt{n})$.[21] In order to man-

20 The original phrasing of Cheng and Vigneron was "impact event".
21 By $\omega(f(n))$ we denote the set $\Omega(f(n)) \setminus \Theta(f(n))$ following the notation of $o(f(n))$ versus $O(f(n))$.

age the potential crash events, Cheng and Vigneron maintain for each cutting-triangle an arrangement on the tracks of each motorcycle that is or has been visiting the triangle. Initially, each arrangement in a cutting triangle C is built for the motorcycles that start within C. Using a standard plane-sweep algorithm — e. g., Bentley-Ottmann [BO79] — this can be achieved in $O(n\sqrt{n}\log n)$ time.

When a switch event occurs for the motorcycle m_i they first check whether m_i has already crashed. If it is still driving they insert the track of m_i into the arrangement of the new triangle C. Cheng and Vigneron use a binary tree structure on each cell in each arrangement, which enables them to insert the track of m_i in $O(k\log n)$ time, where k denotes the number of new arrangement vertices introduced. For each such arrangement vertex they add a potential crash event into Q, depending on which of both motorcycles involved reaches the vertex earlier. Cheng and Vigneron argue that the total number of arrangement vertices is bounded by $O(n\sqrt{n})$. In the triangle C, they consider an arrangement vertex v on the intersection of the tracks of m_i and m_j. Either m_i or m_j started or crashed within C. Cheng and Vigneron charge m_i with v, if m_i started or crashed within C and likewise for m_j. In order to bound the total number of arrangement vertices they count for each motorcycle m_k the number of vertices for which m_k has been charged. In order to do so they only have to consider the cutting triangles where m_k started resp. crashes. We get at most $2\sqrt{n}$ vertices for each motorcycle and $O(n\sqrt{n})$ in total. Hence, all switch events can be handled in $O(n\sqrt{n}\log n)$ time.

When a crash event is processed for m_i, Cheng and Vigneron check whether m_i has not crashed already. If it is still driving, the motorcycle m_i definitely crashed at the arrangement vertex for which the current crash event has been created. The total number of potential crash events is bounded by the total number of arrangement vertices, which is $O(n\sqrt{n})$, as elaborated above. A single crash event is easily handled in $O(\log n)$ time and hence the total costs for all crash events is in $O(n\sqrt{n}\log n)$.

The space complexity for all arrangements is bounded by $O(n\sqrt{n})$. Summarizing, Cheng and Vigneron compute the motorcycle graph of n motorcycles in $O(n\sqrt{n}\log n)$ time and $O(n\sqrt{n})$ space[22]. This is a slight improvement compared to the algorithm by Eppstein and Erickson [EE99].

Strictly speaking, the discussed algorithm computes $\mathcal{M}(m_1,\ldots,m_n)$ but does not consider the edges of P as walls, i. e., they do not compute $\mathcal{M}(P)$ as defined in Section 1.2.4. However, Cheng and Vigneron note that their algorithm can handle motorcycles that *run out of fuel* when they reach the boundary of P. In order to bound the length of the motorcycle traces accordingly, one could apply the ray-shooting algorithm within a polygon by Chazelle et al. [CEG⁺91], which supports a single ray shooting in $O(\log n)$ time.

Cheng and Vigneron [CV07] also mention a randomized and simpler version of their algorithm, where the supporting lines of the traces of $\sqrt{n}$ randomly chosen motorcycles could be used instead of an $1/\sqrt{n}$-cutting. Furthermore, they mention a simpler cutting algorithm in the plane by Har-Peled [HP00]. Nevertheless, no implementations or runtime statistics are known for their motorcycle graph algorithm.

In order to compute the straight skeleton of a simple non-degenerate polygon P with n vertices, Cheng and Vigneron [CV07] exploit Theorem 2.11. The fact that $\mathcal{M}(P)$ covers

22 Cheng and Vigneron did not discuss space consumption in their work.

the reflex arcs of $\mathcal{S}(P)$ allows them to devise a randomized algorithm for the computation of $\mathcal{S}(P)$. In a nutshell, the algorithm chooses random sites on the straight skeleton which induce a partition of P. The idea is to compute $\mathcal{S}(P)$ on each part of the partition in a divide-and-conquer fashion, as described in the following.

They first choose a point p on a convex arc of $\mathcal{S}(P)$ and project p vertically onto the terrain $\mathcal{T}(P)$. The projection point $\hat{p}$ lies on a ridge of $\mathcal{T}(P)$ and defines two or three paths on $\mathcal{T}(P)$ that follow the steepest descent and correspond to the raindrop traces in Section 1.2.3. Cheng and Vigneron consider a set E of such ridge points and consider the vertical projections of the corresponding raindrop paths onto the plane. These projected paths induce a partition of P, which Cheng and Vigneron call *canonical partition of P induced by E*. (Note that these paths do not cross, but may merge in a valley of $\mathcal{T}(P)$.) Cheng and Vigneron observed that the canonical partition of P induced by E can be computed recursively. Consider a polygon that is partition by E_1 and a cell C of that partition that is further partitioned by E_2. They proved that the resulting partition is equal to the partition of P induced by $E_1 \cup E_2$.

Cheng and Vigneron presented a method to compute the intersection points $L \cap \mathcal{S}(P)$, for a line L parallel to the y-axis, by computing a vertical slice of $\mathcal{T}(P)$ above L. The resulting points E are used in order to subdivide P according to the partition scheme described above. Cheng and Vigneron keep on subdividing cells of the partition scheme using this method. Once the cells have reached a certain size, they compute the straight skeleton on this cell by a brute force method. The way how Cheng and Vigneron select the vertical line L involves randomness. Finally, Cheng and Vigneron are able to compute the straight skeleton of a simple non-degenerate polygon P in expected $O(n \log^2 n)$ time if the motorcycle graph is already given. Since the motorcycle graph $\mathcal{M}(P)$ can be computed in $O(r\sqrt{r}\log r)$ time they obtain a straight-skeleton algorithm that runs in $O(n \log^2 n + r\sqrt{r}\log r)$ time.

Cheng and Vigneron [CV07] extended their algorithm to non-degenerate polygons with holes. If h denotes the number of holes their algorithm computes the straight skeleton in $O(n\sqrt{h}\log^2 n + r\sqrt{r}\log r)$ expected time.

1.4.2.5 *Felkel and Obdržálek, 1999*

Felkel and Obdržálek [FO99] briefly presented a wavefront-type straight-skeleton algorithm for simple polygons that is based on the algorithm of Aichholzer et al. [AAAG95]. As discussed in Section 1.4.2.1, the simulation of the edge events is simple, but handling split events is the challenge for a wavefront-type straight-skeleton algorithm.

They again maintain a priority queue of edge events and split events. Felkel and Obdržálek [FO99] propose the following procedure in order to determine the split event for a reflex vertex v of P. For each edge e they determine a point p_e on the bisector ray emanated from v which has equal orthogonal distance to e and the incident edges of v. The edge e defines a triangle Δ_e (possibly infinite) that is bounded by e and the bisector rays emanated from the incident vertices of e. If p_e does not lie in Δ_e then they ignore p_e. Among all edges e of P, with p_e lying in Δ_e, they select the one point p_e whose distance from v is smallest. They add a split event for v at this point into our priority queue.

This algorithm takes $O(nr + n\log n)$ time in order to compute the straight skeleton. Unfortunately, the procedure which computes the next split event does not necessarily determine

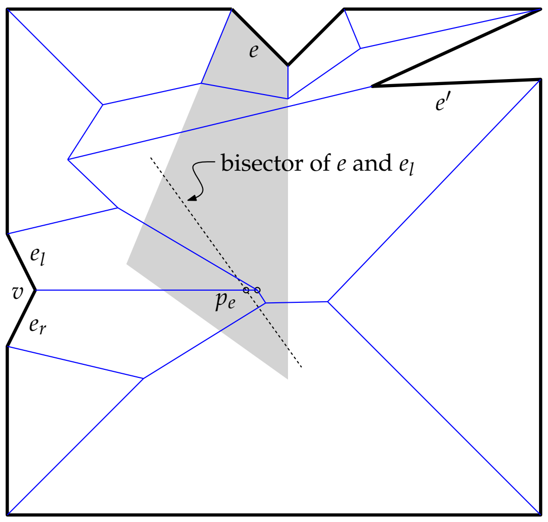


Figure 13: The point p_e does not mark the split event of the reflex vertex v. The vertex v actually causes a crash with the edge e', which happens after v passed p_e.

the correct split event. In Figure 13 we give an example where the split event of a vertex v is not computed correctly. Different issues concerning the correctness are mentioned in Yakersberg [Yak04].

1.4.3 Implementations

To the best of our knowledge the only published runtime statistics are due to Felkel and Obdržálek [FO99], who published the runtime consumption of their code for seven datasets. Beside the implementation of Felkel and Obdržálek, the only implementation available is the straight-skeleton code by Cacciola [Cac04] that is shipped with CGAL [CGA]. The algorithm behind the CGAL implementation was originally based on the algorithm by Felkel and Obdržálek [FO99]. However, Cacciola has significantly adapted the original algorithm in order to get it to work correctly.[23] The current version, CGAL 3.8, can compute the straight skeleton of polygons with holes. Unfortunately, no details on the implemented algorithm are published. In Section 2.5.4, we present experimental results that yield an $O(n^2 \log n)$ runtime performance and an $O(n^2)$ memory footprint for the CGAL implementation.

1.4.4 Summary

The best known lower bounds for the straight skeleton of simple polygons is $\Omega(n)$ resp. $\Omega(n \log n)$ for simple polygons with holes and planar straight-line graphs. This poses a significant gap to the fastest algorithms known so far, see Table 1. The fastest algorithm is due to Eppstein and Erickson [EE99] with a runtime of $O(n^{1+\epsilon} + n^{8/11+\epsilon} r^{9/11+\epsilon})$ for pla-

23 Based on personal e-mail correspondence with the author in 2010.

Algorithm	Time	Space	PSLG	Impl.
[AAAG95]	$O(nr \log n)$	$O(n)$	No	Yes
[AA96]	$O(n^3 \log n)$	$O(n)$	Yes	Yes
[EE99]	$O(n^{1+\epsilon} + n^{8/11+\epsilon} r^{9/11+\epsilon})$	$O(n^{1+\epsilon} + n^{8/11+\epsilon} r^{9/11+\epsilon})$	Yes	No
[CV02]	exp. $O(n \log^2 n + r\sqrt{r} \log r)$		No	No

Table 1: Summary of known straight-skeleton algorithms and their time and space complexities. The input contains n vertices and r denotes the number of reflex wavefront vertices. The column "PSLG" denotes whether the algorithm accepts a planar straight-line graph as input. The column "Impl." denotes whether the algorithm is suitable for implementation from a practical point of view.

Algorithm	Time	Space	Impl.
[EE99]	$O(n^{17/11+\epsilon})$	$O(n^{17/11+\epsilon})$	No
[CV02]	$O(n\sqrt{n} \log n)$	$O(n\sqrt{n})$	No

Table 2: Summary of known motorcycle-graph algorithms and their time and space complexities for n motorcycles. The column "Impl." denotes whether the algorithm is suitable for implementation from a practical point of view.

nar straight-line graphs. For simple non-degenerate polygons Cheng and Vigneron [CV07] presented an algorithm with a slightly better expected runtime of $O(n \log^2 n + r\sqrt{r} \log r)$.

On the implementation side there is the straight-skeleton code by CGAL which accepts polygons with holes as input. It is a wavefront-type algorithm which is roughly based on the algorithms due to [AAAG95, FO99]. The only straight-skeleton algorithm which is suitable for implementation and accepts planar straight-line graphs as input was presented by Aichholzer and Aurenhammer [AA98]. However, no implementation is available and no runtime statistics have been published.

1.5 GENERALIZATIONS AND RELATED PROBLEMS

1.5.1 Linear axis

The linear axis is a variation of the straight skeleton for simple polygons P introduced by Tănase and Veltkamp [TV04a]. The difference between the linear axis and the straight skeleton is a more general definition of the initial wavefront at reflex vertices of P.

The idea is that a reflex vertex v of P does not emanate a single reflex wavefront vertex but k reflex wavefront vertices $v_1, \dots, v_k$, with $1 \leq k$. The wavefront edges connecting two consecutive vertices v_i, v_{i+1} propagate with unit speed and for each v_i the two incident edges span identical angles, see Figure 14. The skeleton structure which results from this initial wavefront and the ordinary straight-skeleton wavefront propagation is called *linear axis*. For

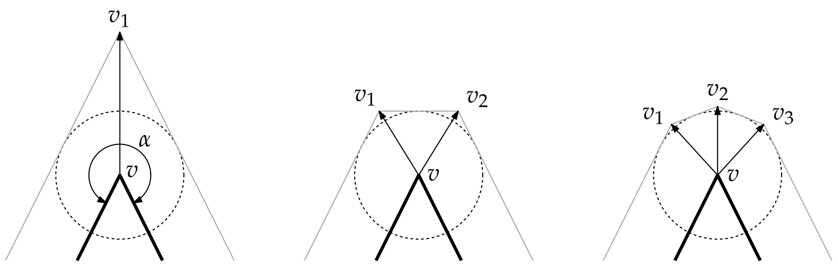


Figure 14: The wavefront (gray) of the linear axis at a reflex vertex v of the input polygon (bold) for different numbers k of emanated wavefront vertices $v_1, \dots, v_k$, with $k = 1, 2, 3$. The angles between consecutive wavefront edges are equal.

$k = 1$ the linear axis is identical to the straight skeleton. Denoting by $\alpha \in [\pi, 2\pi)$ the interior angle at the vertex v of P, the speeds of the reflex wavefront vertices are given by

$$\frac{1}{\cos \frac{\alpha - \pi}{2k}}. \tag{1.3}$$

Note that this expression tends towards 1 for large k. In particular, the speeds are bounded by $\sqrt{2}$ for $k \geq 2$, whereas for $k = 1$ the reflex wavefront vertices can get arbitrarily fast when α gets close to 2π. The larger k gets, the better the wavefront approximates a circular arc. Tănase and Veltkamp [TV04a] proved that the linear axis converges to the medial axis as k grows for all reflex vertices. The concept of the linear axis has later been generalized to planar straight-line graphs by Vyatkina [Vya09a].

Since the linear axis follows the same wavefront propagation process as the straight skeleton, one can basically apply the algorithms known for straight skeletons in order to compute the linear axis. For $k \in O(1)$ at all reflex vertices, we obtain the same time and space complexities. However, Tănase and Veltkamp showed that if k is chosen large enough such that the medial axis and the linear axis differ in a sufficiently small amount then the linear axis can be obtained from the medial axis in linear time.

Of course, we can further generalize the linear axis by not requiring that consecutive wavefront edges in Figure 14 have identical angles. Basically, we can emanate any polygonal chain as wavefront as long as the wavefront edges have orthogonal distance to v that is equal to the time elapsed so far. We also refer to the discussion in Section 1.2.2 on various types of wavefront polygons emanated at isolated vertices and terminal vertices of the input graph.

The advantages of the linear axis compared to the straight skeleton are all, more or less, related to a reduced speed of the reflex vertices. From a practical point of view, we expect that the linear axis is easier to compute in terms of numerical stability. Tănase and Veltkamp [TV04a] mention that the polygon decomposition application from Section 1.3.5 would also benefit if the speed of the reflex wavefront vertices is limited. Finally, we want to review the mitered offset application for NC-machining, see Section 1.3.1. A tool path based on the straight skeleton can lead to long paths at very sharp reflex vertices. The offset curve based on the linear axis would give us a shorter path while still avoiding a continuous contact of the tool with reflex vertices, as it would be the case for the medial axis.

1.5.2 Weighted straight skeleton

Eppstein and Erickson [EE99] were the first to mention the weighted straight skeleton, where the wavefront edges are allowed to propagate at different speeds. The geometry of the weighted straight skeleton has not yet been systematically investigated and, in particular, its relation to the motorcycle graph is unclear. However, one can observe that basic properties of the unweighted straight skeleton do not carry over to the weighted case. For example, in the unweighted case every straight-skeleton face is monotone w. r. t. its defining wavefront edge, cf. Lemma 2.3 in Section 2.1. As Figure 15 (a) illustrates, this is not necessarily the case for the weighted straight skeleton. Moreover, important theorems concerning alternative characterizations of the straight skeleton do not hold for the weighted case. Eppstein and Erickson [EE99] presented a simple polygon for which Theorem 2.7 does not hold in the weighted case, see Figure 16. The same figure also serves as a counter-example to Theorem 2.11.

Beside the fact that certain properties do not carry over from the unweighted straight skeleton, we observe that the definition of the weighted straight skeleton is tricky in the presence of parallel input segments. Figure 15 shows two wavefront edges e_1, e_2 that propagate with unit speed and e_3 that propagates with speed 2. Let us rotate the edge e_1 around its right endpoint such that e_1 and e_3 become parallel in both subfigures. In the limit we obtain two identical inputs in both subfigures, but in subfigure (a) the straight-skeleton arc between e_1 and e_3 is pointing to the left, whereas in subfigure (b) this arc is pointing to the right. This example poses a singularity in the naive wavefront-based definition of the weighted straight skeleton. Also note that in subfigure (a) the arc between e_1 and e_3 is convex, but the arc encloses an angle greater than $\pi/2$ with e_3. Analogously for subfigure (b), where this arc is reflex but e_3 and the arc encloses an angle less than $\pi/2$. For the unweighted straight skeleton, however, we could characterize reflex and convex arcs by the angle enclosed with its defining wavefront edges.

Lemma 1.11. *Let v denote a wavefront vertex incident to two edges e_a and e_b, where e_a propagates with speed $a > 0$ and e_b propagates with speed $b > 0$. Further assume that e_a and e_b span an angle of $\alpha \in (0, 2\pi)$ at the side where the wavefront propagates to. Then the trajectory of v encloses with e_b an angle α_b and the wavefront vertex v has a speed of $\frac{1}{\sin \alpha_b}$, where*

$$\alpha_b = \frac{\pi}{2} - \arctan \frac{\cos\alpha + \frac{a}{b}}{\sin\alpha}. \tag{1.4}$$

Proof. Let us consider Figure 17. Simple trigonometry leads to $x = -a \cdot \sin\alpha$, $y = a \cdot \cos\alpha$, $z = \frac{b+y}{\tan(2\pi-\alpha)}$ and finally to $\alpha_b = \pi/2 + \arctan\frac{x+z}{b}$. □

In the unweighted case, i. e. $a = b$, we obtain $\alpha_b = \alpha/2$ as expected. If we consider a/b fixed but less than 1 we observe the following behavior of α_b at $\alpha = \pi$.

$$\lim_{\alpha \nearrow \pi} \alpha_b = \pi \qquad \lim_{\alpha \searrow \pi} \alpha_b = 0. \tag{1.5}$$

Both equations correspond to the two cases illustrated in Figure 15, with v denoting the wavefront vertex on the arc between e_1 and e_3 and $e_b = e_3$. The left equation is related to Figure 15 (a) while the right equation describes the situation in Figure 15 (b).

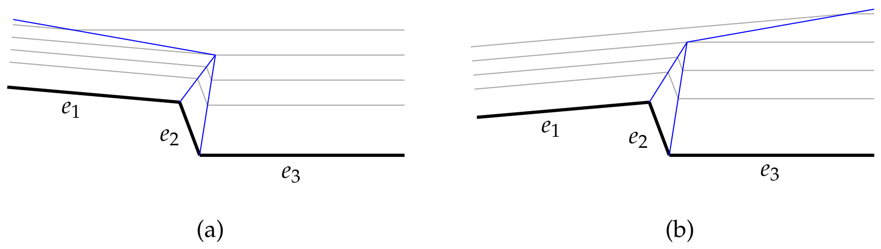


Figure 15: The weighted straight skeleton of three input edges. The edges e_1 and e_2 propagate at unit speed, but e_3 propagates with speed 2. If we rotate e_1 such that e_1 and e_3 become parallel, we obtain two different straight skeletons in the limit for the two subfigures.

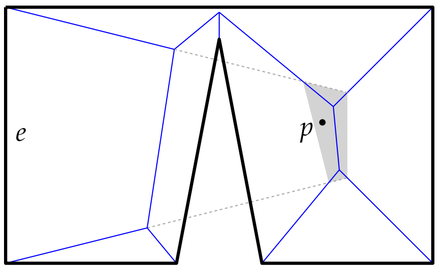


Figure 16: Assume all edges propagate with unit speed, but e has speed 4. Then the edge slab of e cuts off the shaded area from the terrain, i. e. the edge slab reaches below $\mathcal{T}(P)$ at point p. (This figure is based on Figure 6 in [EE99].)

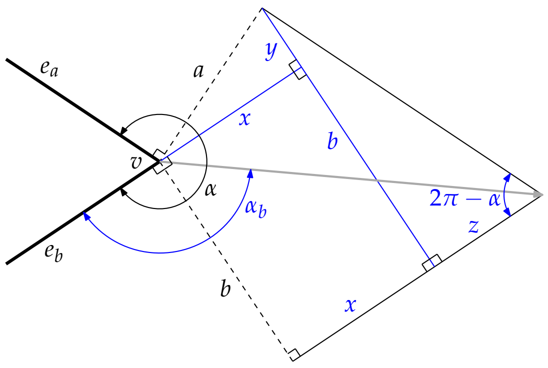


Figure 17: The speed and direction (gray arrow) of a vertex v which is incident to two edges e_a and e_b. The edge e_a is propagating with speed a and e_b is propagating with speed b.

As mentioned in Section 1.4.2.3, Eppstein and Erickson [EE99] presented an algorithm for the weighted straight skeleton of planar straight-line graphs with n vertices which uses $O(n^{8/5+\epsilon})$ time and space.

1.5.3 Straight skeleton of polyhedra in $\mathbb{R}^3$

Straight skeletons of three-dimensional polyhedra have first been mentioned by Demaine et al. [DDLS05], who investigated so-called *hinged dissections* of polyhedra. A dissection of two polyhedra is a tessellation of the first polyhedron into polyhedral pieces such that the second polyhedron can be built by the pieces of the first one. Demaine et al. [DDLS05] discuss so-called hinged dissections which are special dissections where the polyhedral pieces are hinged together at points or edges. It is an open question whether there exists a hinged dissection for every pair of polyhedra. However, the straight skeleton of polyhedra is used by Demaine et al. in order to compute hinged dissections for a certain class of polyhedra.

The straight skeleton $\mathcal{S}(P)$ of a three-dimensional polyhedron P is defined by a wavefront propagation process, where the faces of P move inwards with unit speed in a self-parallel fashion. The set of points which are traced by the intersections of two adjacent faces is called the straight skeleton. Demaine et al. mention that $\mathcal{S}(P)$ gives a decomposition of P into cells such that one cell belongs to exactly one face of P. Further, Demaine et al. cite a personal communication with J. Erickson, in which it is mentioned that the straight skeleton in $\mathbb{R}^3$ is no longer uniquely defined, because at certain points in time one can make multiple decisions to continue the offset propagation.

Barequet et al. [BEGV08] were the first to investigate algorithms for the straight skeleton of polyhedra P of certain types. Firstly, they mention a lower bound of $\Omega(n^2)$ for the size of $\mathcal{S}(P)$ of polyhedra with n vertices. Note that the medial axis and the straight skeleton are identical for a convex polyhedron and the lower bound of $\Omega(n^2)$ due to Held [Hel94] can be applied. Secondly, Barequet et al. mention $O(n^{3+\epsilon})$ as the best known upper bound due to Sharir [Sha94].

Furthermore, Barequet et al. [BEGV08] presented a straight-skeleton algorithm for a certain class of polyhedra. First, they start with polyhedra made of voxels, i. e. axis-parallel cubes of identical size. Second, an extension to polyhedra with axis-parallel edges is presented. In the latter case the complexity of the straight skeleton can be bound to $O(n^2)$. Barequet et al. [BEGV08] present two algorithms for n-vertex polyhedra with axis-parallel edges. The first runs in $O(n^2 \log n)$ time, the second in $O(k \log^{O(1)} n)$ time, where k denotes the size of the straight skeleton. Putting both algorithms together results in a runtime of $O(\min\{n^2 \log n, k \log^{O(1)} n\})$.

An interesting result of Barequet et al. [BEGV08] is that general simple n-vertex polyhedra exist, where the complexity of the straight skeleton is super-quadratic, namely $\Omega(n^2\alpha^2(n))$, where $\alpha(.)$ denotes the extremely slowly growing inverse Ackermann function. Furthermore, Barequet et al. shed more light on the ambiguity issue of straight skeletons of polyhedra.

1.5.4 City Voronoi diagrams

The straight skeleton was used by Aichholzer et al. [AAP04] in order to compute the so-called city Voronoi diagram which is presented as follows. Let C denote a planar straight-line graph with c vertices and edges that are axis-parallel. The graph C models a public transit system like the subway. The problem is to find the shortest path between two vertices x and$y$ in the plane, where the distance measure on $\mathbb{R}^2 \setminus C$ is given by the Manhattan metric. Furthermore, one may enter and leave the public transit at any point of C with zero delay. The public transit is assumed to move on the edges of C with speed $v > 1$.

Aichholzer et al. [AAP04] define by $Q_C(x, y)$ the temporal distance according to the quickest path from x to y. This distance measure Q_C induces a Voronoi diagram on a set S of n vertices, which is called the *city Voronoi diagram* $\mathcal{V}_C(S)$ *of* S *w. r. t.* C. Aichholzer et al. show that $\mathcal{V}_C(S)$ is a subset of the additively and multiplicatively weighted[24] straight skeleton of an input which is based on S and C. This enables Aichholzer et al. [AAP04] to compute the city Voronoi diagram by using straight-skeleton algorithms. Interestingly, Aichholzer et al. were able to show that the input to the straight-skeleton algorithm has sufficiently nice properties such that the framework of abstract Voronoi diagrams of Klein [Kle89] can be applied. Note that for general input it is not possible to interpret the straight skeleton as abstract Voronoi diagram, see Section 2.1. This observation finally leads to an algorithm which uses $O(n \log n + c^2 \log c)$ time and $O(n + c)$ optimal space.

24 We use the term "multiplicatively weighted" if the wavefront edges propagate at different speeds. We use the term "additively weighted" if the some wavefront edges are allowed to start propagating after some time.

2 COMPUTING THE STRAIGHT SKELETON

In order to make straight skeletons applicable in practice we seek an algorithm which is (i) easy to implement and (ii) exhibits an actual runtime that is relatively close to linear in the input size. Fortune started his introduction [For00] to the 27th volume of *Algorithmica* with the following words:

> It is notoriously difficult to obtain a practical implementation of an abstractly described geometric algorithm. This difficulty arises in part from the conceptual complexity of many geometric algorithms, which often use sophisticated data structures or require other complex algorithms as subroutines. The difficulty also arises because many algorithms are designed and described to achieve good asymptotic behavior in the worst case, ignoring behavior in more realistic situations.

In this chapter we present a novel straight-skeleton algorithm which is (i) easy to implement, (ii) exhibits a runtime that is close to linear in practice, (iii) accepts planar straight-line graphs as input, and (iv) has a lower worst-case runtime complexity than the triangulation-based algorithm by Aichholzer and Aurenhammer [AA98].

We start with an investigation of the number of flip events that occur for the algorithm by Aichholzer and Aurenhammer [AA98] in Section 2.2. We first show a few results regarding the gap between $\Omega(n^2)$ and $O(n^3)$ for the worst-case number of flip events and prove that we can exploit Steiner triangulations in order to completely avoid all flip events. This insight motivates an algorithm for straight skeletons of non-degenerate polygons that is based on motorcycle graphs, see Section 2.3. In order to generalize this algorithm to arbitrary planar straight-line graphs we introduce a generalization of the motorcycle graph in Section 2.4 and present an extension of our straight-skeleton algorithm in Section 2.5. Extensive runtime experiments in Section 2.5.4 reveal that our straight-skeleton implementation Bone exhibits an actual runtime consumption of $O(n \log n)$ in practice. Most results in the Sections 2.2 to 2.5 have been presented in the following publications:

- [HH10b] S. Huber and M. Held. Straight Skeletons and their Relation to Triangulations. In *Proc. 26th Europ. Workshop Comput. Geom.*, pages 189–192, Dortmund, Germany, Mar 2010
- [HH10a] S. Huber and M. Held. Computing Straight Skeletons of Planar Straight-Line Graphs Based on Motorcycle Graphs. In *Proc. 22nd Canad. Conf. Comput. Geom. (CCCG 2010)*, pages 187–190, Winnipeg, Canada, Aug 2010
- [HH11c] S. Huber and M. Held. Theoretical and Practical Results on Straight Skeletons of Planar Straight-Line Graphs. In *Proc. 27th Annu. ACM Sympos. Comput. Geom.*, Paris, France, to be published 2011

2.1 GEOMETRIC PROPERTIES OF THE STRAIGHT SKELETON

In this section we build a collection of geometric properties of straight skeletons and their relation to motorcycle graphs. Most of the following geometric properties were presented in [AAAG95, AA98, EE99, CV07]. Throughout this section, we denote by G a planar straight-line graph. It turns out that the terrain $\mathcal{T}(G)$ is often useful to prove various properties of the straight skeleton and its relation to motorcycle graphs. In many cases the proofs become much easier when the scene considered is lifted to $\mathbb{R}^3$. The following lemma is a simple but useful tool for proofs of this kind.

Lemma 2.1. *Consider two distinct points p, q on $\mathcal{T}(G)$. Then the straight line through p and q has a slope of at most* 1.

Proof. We project p and q onto the plane $\mathbb{R}^2 \times \{0\}$ and denote the results by p' and q'. The intersection of $\mathcal{T}(G)$ with a vertical curtain above $[p' q']$ results in a plane polygonal chain that starts at p and ends at q. All segments of this chain have a slope of at most 1, because each segment results from the intersection of the vertical curtain with a facet of $\mathcal{T}(G)$ and all facets have exactly slope 1. As a consequence, the supporting line $\overline{pq}$ through p and q has also a slope of at most 1. □

Definition 2.2. We denote by $e(t)$ the set of points that are occupied by the wavefront edge e at time t and by $\overline{e(t)}$ we denote the supporting line of $e(t)$. If e is emanated by a terminal vertex or an isolated vertex v of G then we define $\overline{e(0)} := \lim_{t \searrow 0} \overline{e(t)}$.

Note that the propagating wavefront edge e can be split at certain points in time. That is, $e(t)$ can be expressed as the union of straight-line segments. Furthermore, we get that $f(e) = \bigcup_{t \geq 0} e(t)$. For a terminal vertex v we obtain that $\overline{e(0)}$ is equal to the supporting line of v that is perpendicular to the single incident edge of v.

Aichholzer et al. [AAAG95] proved the following lemma for general bisector graphs of simple polygons instead of straight skeletons. These graphs correspond to generalized roofs as discussed in Section 1.2.3. Aichholzer and Aurenhammer [AA96] presented this lemma for planar straight-line graphs G. We rephrase their original proof using Lemma 2.1.

Lemma 2.3 ([AA96]). *Let e be a wavefront edge of G. The face $f(e)$ is monotone w. r. t. $\overline{e(0)}$.*

Proof. It has to be shown that an intersection of $f(e)$ with a line perpendicular to $\overline{e(0)}$ is connected. Assume that this is not the case for a line l that is perpendicular to $\overline{e(0)}$. One can find two points x and y on the boundary of $f(e)$ such that the open segment (x, y) is not contained in $f(e)$. We lift the problem to $\mathbb{R}^3$ and put a vertical curtain on $[xy]$, intersect it with $\mathcal{T}(G)$ and obtain a polygonal chain L. Let us denote its endpoints with $\hat{x}$ and $\hat{y}$, which are contained in $\hat{f}(e)$. The supporting line of $\hat{x}, \hat{y}$ has a slope of exactly 1, but note that L is not contained in $\hat{f}(e)$ by assumption. Hence, L contains at least one segment which has a slope that is greater than 1. But this is a contradiction to Lemma 2.1. □

Lemma 2.4 ([AA96]). *The straight skeleton $\mathcal{S}(G)$ has exactly $2n + t - 2$ number of nodes, where t denotes the number of terminals in G.*

Similar to Lemma 1.2, the lemma holds if the degree of all nodes is three. In the presence of multi split events the degree of the resulting nodes have to be taken into account accordingly.

The proof of Aichholzer and Aurenhammer is based on the analysis of their algorithm, see Section 1.4.2.2. Recall that their algorithm computes a constrained triangulation of G and then simulates the wavefront by maintaining the triangulation. A topological event of the wavefront corresponds to a decrease of the number of triangles by one. Note that the so-called flip events leave the number of triangles invariant. Therefore, the number of events happening to the wavefront corresponds to the number of triangles collapsed. The number of triangles that remain after the last event happened corresponds to the number of unbounded arcs resp. infinite nodes of $\mathcal{S}(G)$. Aichholzer and Aurenhammer [AA96] argue that the number of initial triangles equals the number of nodes, including the infinite nodes.

Proof. One observes that the initial triangulation tessellates the plane into $2n - 2$ triangles. Let us consider an embedding of G onto the three-dimensional sphere, in a local neighborhood of the north pole. We identify the north pole with the origin of the plane and we consider the south pole as an additional vertex which models the locus at infinity. Then an induction-type argument easily shows that any triangulation of $n \geq 3$ vertices plus the infinite vertex comprises $2n - 2$ triangles. Finally, each terminal vertex gives rise to an additional triangle in order to take the additional wavefront edges at terminal vertices into account. Summarizing, the initial triangulation comprises $2n - 2 + t$ triangles. □

THE STRAIGHT SKELETON AS A VORONOI DIAGRAM An obvious yet essential question is whether the straight skeleton can be interpreted as a generalized or abstract Voronoi diagram. The benefit of such a connection is obvious: Voronoi diagrams have been extensively studied in the past decades and there are efficient algorithms for a wide variety of generalizations of Voronoi diagrams [Kle89, KMM93, Yap87, AS95]. Aichholzer and Aurenhammer [AA96] considered the abstract Voronoi framework by Klein [Kle89] in order to investigate this question. The idea of Klein is that one does not consider a distance function in order to define the Voronoi region of each input site, but to define mutual bisector curves between each pair of input sites. Klein [Kle89] gave a set of axioms that need be fulfilled by this set of bisectors, in order to define an abstract Voronoi diagram. The generalized Voronoi diagrams of points, straight-line segments and circular arcs are covered by this framework as well. In addition, generalizations to other distance functions can be represented within this framework. Recently, Klein et al. [KLN09] revisited this framework and presented a conciser set of axioms for the bisector system.

Unfortunately, Aichholzer and Aurenhammer [AA96] pointed out that the framework of Klein cannot be applied to define straight skeletons. Let us consider a bisector between two input sites a, b motivated by the straight skeleton. That is, the bisector is the set of points that are reached at the same time by the wavefronts of a and b. The claim is that the resulting bisector system does not fulfill the axioms of [KLN09]. Following the notation of [KLN09] we denote by $J(a, b)$ the bisector between the site a and b. The bisector tessellates the plane into two halves. The half which belongs to a is denoted by $D(a, b)$ and the other by $D(b, a)$. The Voronoi region $\mathcal{V}(a)$ of a is defined by the intersection $\bigcap_{s \neq a} D(a, s)$ among

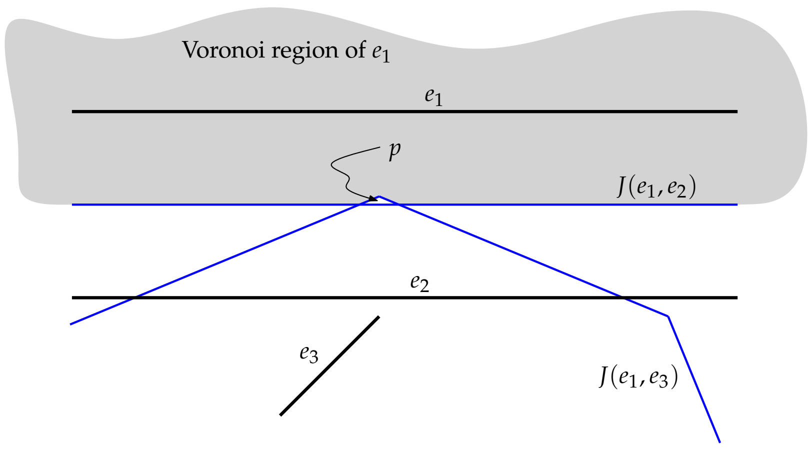


Figure 18: An attempt to define $\mathcal{S}(G)$ using abstract Voronoi diagrams. The bisectors $J(e_1,e_2)$ and $J(e_1,e_3)$ are depicted in blue. The attempt fails because e_3 influences the Voronoi region of e_1. In particular, the point p is not contained in the Voronoi region of e_1.

all input sites s. See Figure 18 for an example. We observe that the Voronoi region of e_1 is influenced by the presence of e_3. However, the straight-skeleton wavefronts of e_3 are blocked by e_2. Note that if we would remove e_2 then $J(e_1,e_3)$ would pose a correct part of the resulting straight skeleton. The basic reason for this strange behavior is that the bisectors motivated by straight skeletons do not fulfill the axioms of Klein et alii. In particular, their axioms include that $D(e_1,e_2) \cap D(e_2,e_3) \subset D(e_1,e_3)$, which is clearly not the case in our example: The point p is contained in the set at the left-hand side but not in the right-hand side. Moreover, p is not contained in any Voronoi region at all.

The concept of abstract Voronoi diagrams by Klein et al. [Kle89, KLN09] distills the very essence of Voronoi diagrams from an abstract point of view. Having this interpretation in mind, straight skeletons are essentially different to Voronoi diagrams, even though they share common properties at the first sight. Nevertheless, Aichholzer and Aurenhammer [AA98] mentioned that for special case of rectilinear polygons P, the Voronoi diagram of P in the L^∞-space and the straight skeleton of P coincide.

THE TERRAIN MODEL AND PIECEWISE-LINEAR FUNCTIONS Aichholzer et al. [AAAG95] investigated whether partially defined linear distance functions could be used in order to obtain an alternative, non-procedural way of defining the straight skeleton $\mathcal{S}(G)$. More precisely, the idea is to define for each wavefront edge e a real-valued linear function $d_e(.)$ on a subset of $\mathbb{R}^2$ such that the lower envelope of the graphs of these functions forms the terrain $\mathcal{T}(G)$. For a point $p \in \mathbb{R}^2$, the value $d_e(p)$ would indicate the time when p is hit by the wavefront edge e. Aichholzer et al. already showed that the domain of d_e cannot only depend on e: In Figure 18, the domain of d_{e_3} must somehow take care for the presence of e_2. We can summarize this insight as follows:

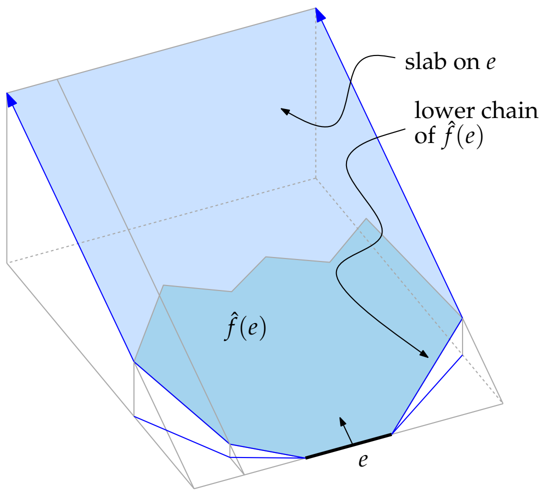


Figure 19: The slabs in $\mathbb{R}^3$ are bounded from below by the lower chain of $\hat{f}(e)$.

Observation 2.5 ([AAAG95])**.** *If one attempts to define a partially linear function for each wavefront edge e of G such that the lower envelope of their graphs is identical to $\mathcal{T}(G)$ then the domain of each function d_e cannot only depend on the defining wavefront edge e.*

Let us consider the face $f(e)$ of a wavefront edge e of G. By Lemma 2.3, the boundary of $f(e)$ consists of two monotonic chains with respect to $\overline{e(0)}$.

Definition 2.6 (lower/upper chain of a face)**.** The *lower chain* and the *upper chain* of $f(e)$ are the two monotonic chains w. r. t. $\overline{e(0)}$ that constitute the boundary of $f(e)$. The lower chain contains $e(0)$.

For the lower chain C of a face $f(e)$, we denote by $\hat{C}$ the projection of C onto $\mathcal{T}(G)$. For each wavefront edge e we define an infinite slab that lies on the supporting plane of $\hat{f}(e)$ and is bounded from below by $\hat{C}$ and by two rays at both endpoints of $\hat{C}$ that are perpendicular to $\overline{e(0)}$, see Figure 19. Eppstein and Erickson [EE99] proved the following theorem.

Theorem 2.7 ([EE99])**.** *The lower envelope of all slabs of the wavefront edges of G is identical to $\mathcal{T}(G)$.*

Proof. We first note that for any wavefront edge e of G, the lifted face $\hat{f}(e)$ is contained in the slab that we defined for e. Hence, it remains to show that the terrain $\mathcal{T}(G)$ is not above the lower envelope of the slabs at any point.

Assume that p is a point on $\mathcal{T}(G)$ that indeed lies above the lower envelope. Hence, we can project p vertically onto the lower envelope and obtain a point p'. Then we can project p' along the steepest descent of the slab, where p' lies on, until we hit a point q. Clearly, q and p are on $\mathcal{T}(G)$. Note that $\overline{qp'}$ has slope 1 and, thereby, $\overline{qp}$ must have slope greater than 1. This is a contradiction to Lemma 2.1. $\square$

In the original work by Eppstein and Erickson [EE99] the theorem above is presented for polygonal input, but it is claimed that it also extends to planar straight-line graphs. Further, they actually used the following set of slabs: For every edge e they defined an *edge slab*, which is bounded from below by e and by two perpendicular rays at each endpoint of e. For every reflex vertex v of G they defined two *reflex slabs* each of which is bounded from below by the valley incident to v and, for each, two rays that are perpendicular to the two incident edges of v. Strictly speaking, in their original phrasing they did not consider that vertex events could happen, i. e., that a valley of $\mathcal{T}(G)$ is not incident to a vertex of G. Nevertheless, the basic idea of their proof also works in the general case, which is based on a careful consideration of the relative positions of the moving wavefront edges. We presented a proof that is based on Lemma 2.1 instead. Our proof appears to be simpler and immediately applies to arbitrary planar straight-line graphs G.

As Eppstein and Erickson [EE99] mentioned, Theorem 2.7 leads to a nice alternative interpretation of the straight skeleton $\mathcal{S}(G)$, based on a partially defined linear functions. The following corollary formulates this interpretation, where d denotes the infimum distance, that is $d(A, B) := \inf_{x \in A, y \in B} d(x, y)$ for two point sets $A, B \subset \mathbb{R}^2$.

Corollary 2.8. *Let e and e' denote two wavefront edges. For any point $p \in f(e)$, whose orthogonal projection line onto $\overline{e'(0)}$ intersects the lower chain of $f(e')$, holds $d(p, \overline{e(0)}) \leq d(p, \overline{e'(0)})$.*

This result enables us to define a point set that is similar to the *cone of influence* in the theory of Voronoi diagrams, cf. [Hel91]. That is, for each wavefront edge e we define a set $\mathcal{CI}(e)$ by the projection of the slab of e onto the plane. Next, we define a distance function d_e between a wavefront edge e and a point p by

$$d_e(p) := \begin{cases} d(\overline{e(0)}, p) & \text{if } p \in \mathcal{CI}(e) \\ \infty & \text{otherwise.} \end{cases} \tag{2.1}$$

Using these distance functions d_e, we can rephrase the previous corollary to

$$f(e) = \bigcap_{e' \in E} \{p \in \mathbb{R}^2 \ : \ d_e(p) \leq d_{e'}(p)\}, \tag{2.2}$$

where E denotes the set of wavefront edges. At the first sight, this result seems to be an alternative characterization of $\mathcal{S}(G)$. However, we want to remark that the sets $\mathcal{CI}(e)$ depend on the length of the reflex arcs of $\mathcal{S}(G)$. In other words, the representation of the faces via (2.2) does not serve as an alternative characterization of straight skeletons.

Eppstein and Erickson [EE99] remarked that Corollary 2.8 does not pose a contradiction to Observation 2.5: The domain of the distance function d_e depends on the length of certain reflex arcs and, therefore, does not only depend on the wavefront edge e.

Lemma 2.9. *The lower chain of the face $f(e)$ of a wavefront edge e is convex.*

Proof. The lemma asserts that the interior angle at each vertex of the lower chain C of $f(e)$ is at most 180°. The segment $e(0)$ encloses with its incident arcs an angle less than 180°, because these arcs lie on the bisectors of e and adjacent edges of e. Hence, it suffices to show that the interior angle at any straight skeleton node on C is at most 180°. We assume the contrary: Suppose that there is a node v on C whose interior angle is reflex. Without loss of generality, we may assume that v is (i) closest to $e(0)$ and (ii) on the left part of C w. r. t. $e(0)$, see Figure 20.

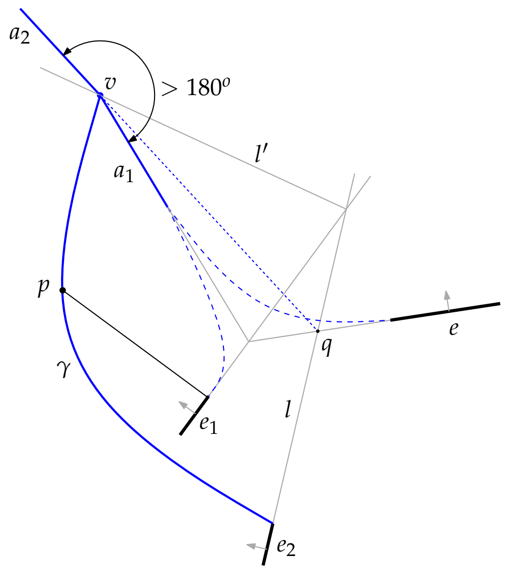


Figure 20: The vertex v on the lower chain of $f(e)$ cannot be reflex.

We denote by a_1 and a_2 the arcs on C that are incident to v such that the sub-chain from a_1 to $e(0)$ is convex. Further, we denote by e_1 and e_2 the other two wavefront edges whose faces contain the arcs a_1 and a_2, respectively. The supporting line of a_2 intersects $\overline{e(0)}$ at a point q. There exists a supporting line l of q such that a_2 lies on the bisector of l and $\overline{e}$. Hence, the edge e_2 lies on l, by construction. Further, we note that $e_2(0)$ lies behind $e(0)$ and also behind $e_1(0)$ w. r. t. the corresponding propagation direction.

Since e_2 reaches v after some time, the lower chain of $f(e_2)$ contains a polygonal chain γ that connects $e_2(0)$ and v. We note that γ is monotone w. r. t. e_2 by Lemma 2.3. Furthermore, γ is monotone w. r. t. the propagation direction of e_2 because the points on γ are swept by the propagating edge e_2. The curve γ contains a point p that can be projected orthogonally onto $e_1(0)$. Corollary 2.8 implies that $d(p, \overline{e_2(0)}) \leq d(p, \overline{e_1(0)})$ because $p \in f(e_2)$. Let us denote by l' the bisector between e_1 and $_2$. That is, l' consists of all points $\overline{e_1(t)} \cap \overline{e_2(t)}$, with $t \geq 0$, and all points left to l' are first reached e_1. However, since p is left to l' it follows that $d(p, \overline{e_1(0)}) < d(p, \overline{e_2(0)})$, which is a contradiction. □

Lemma 2.10. *The lower chain of a face $f(e)$ consists of $e(0)$ and reflex straight-skeleton arcs.*

The following theorem was proved by Cheng and Vigneron [CV07]. It establishes an essential connection between the motorcycle graph and the straight skeleton.

Theorem 2.11 ([CV07])**.** *Let P denote a simple non-degenerate polygon P. The reflex arcs of $\mathcal{S}(P)$ are covered by $\mathcal{M}(P)$.*

Proof. We slightly rephrase the proof in [CV07] in order to fit our setting. First, we declare that $\mathcal{S}(P)$ and $\mathcal{M}(P)$ are restricted to the interior of the polygon P rather than being defined on the whole plane.

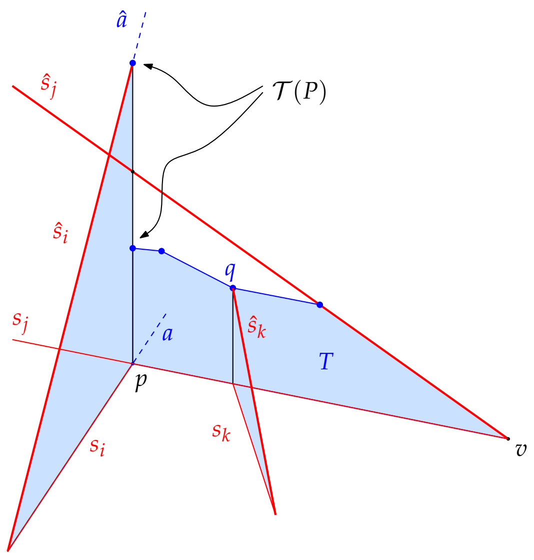


Figure 21: Reflex arcs are covered by motorcycle traces. The set T turns out to be convex which leads to a contradiction concerning the height of $\mathcal{T}(P)$ above p.

Cheng and Vigneron lift the problem to $\mathbb{R}^3$ by means of the terrain model. Let $m_1, \ldots, m_r$ denote the motorcycles from $\mathcal{M}(P)$, where r denotes the number of reflex vertices of P. Further, let s_i denote the trace of the motorcycle m_i and let $\hat{s}_i$ denote the lifted trace of $\hat{m}_i$ by interpreting the third spatial dimension as the time. Recall that each motorcycle m_i starts at a reflex wavefront vertex v of $\mathcal{W}(P, 0)$ and m_i and v have the same velocity. Hence, the valley $\hat{a}$ that belongs to v has the same inclination as the lifted motorcycle trace $\hat{m}_i$.

Now assume that the statement is false and there is indeed a reflex arc a of $\mathcal{S}(P)$ that is only partially covered by the motorcycle trace s_i. Since a continues on the track of m_i, the motorcycle m_i crashed into another motorcycle, say m_j. Let p denote the intersection point of s_i and s_j and let t^* denote the height of $\mathcal{T}(P)$ above p. Among all reflex arcs that are only partially covered, a is chosen such that t^* is lowest. Hence, up to the height t^*, all valleys of $\mathcal{T}(P)$ are covered by tilted motorcycle traces.

The trace s_j starts at a reflex vertex v of P and contains p. Let T denote the intersection of a vertical curtain that is put on the segment $[vp]$ with the set of points below $\mathcal{T}(P)$, see Figure 21. The claim is that T is convex. Assume, to the contrary, that T contains a reflex vertex on its top chain. Let q denote such a reflex vertex that is closest to v. The vertex q cannot be at height zero, which would mean that m_j crashed into a wall — that is, an edge of P — at q. Note that T and $\hat{s}_j$ start with the same inclination, which means that q is below or just at the same height as $\hat{s}_j$. Because q is a reflex vertex of T, there exists a valley at q, and because each valley is covered by a motorcycle trace until height t^*, there is a tilted motorcycle trace $\hat{s}_k$ at q. So either s_j crashed into s_k or the valleys corresponding to m_j and m_k meet at q. Both cases are a contradiction to the initial assumptions. It follows that T is convex and hence below $\hat{s}_j$.

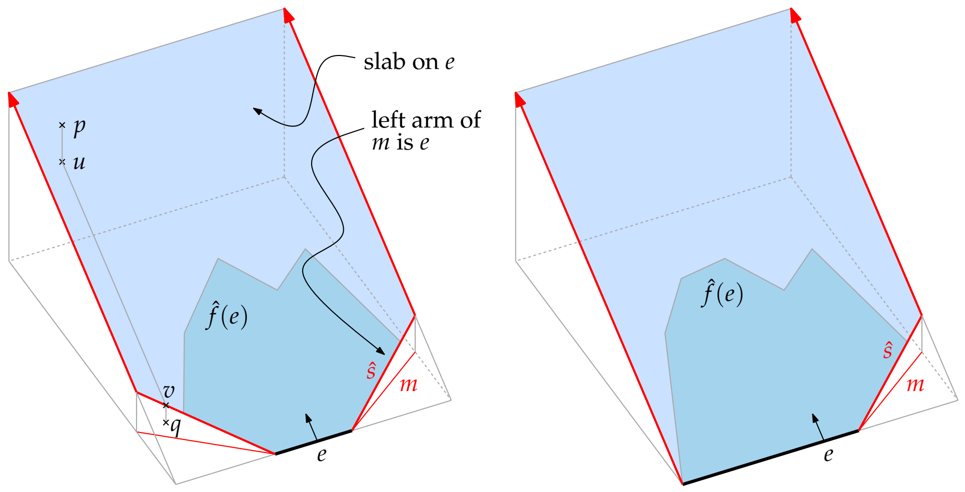


Figure 22: The slab at e is bounded from below by e and the tilted motorcycle traces of motorcycles that have e as an arm. Left: two motorcycles have e as an arm. Right: Only one motorcycle has e as an arm.

The height of $\mathcal{T}(P)$ at p is once given by the valley $\hat{a}$, on one hand, and by the height of T at p, on the other hand. But this is a contradiction, because $\hat{a}$ is strictly above $\hat{s}_j$, which itself is above T. □

The following corollary is a byproduct of the previous proof.

Corollary 2.12 ([CV07]). *Let P denote a simple non-degenerate polygon P. The tilted traces $\hat{s}$ of $\mathcal{M}(P)$ are above or just at the same height as $\mathcal{T}(P)$.*

Let us revisit the slab construction from Theorem 2.7. Cheng and Vigneron [CV07] presented a different slab construction scheme that is based on the motorcycle graph. For the matter of consistency, we reformulate their construction as follows.

Let P denote a simple non-degenerate polygon. Each motorcycle m in $\mathcal{M}(P)$ starts from a reflex wavefront vertex v in $\mathcal{W}(P, 0)$, which is incident to two wavefront edges. Recall that we call the one wavefront edge that is left to the track of m the left arm of m and the other wavefront edge the right arm of m. To each wavefront edge e, there can exist one motorcycle whose right arm is e and one motorcycle whose left arm is e, see Figure 22.

Definition 2.13 (lower envelope)**.** Let $\hat{C}$ denote the union of $e(0)$ and the tilted traces of the motorcycles that have e as an arm. For each wavefront edge e we define a slab that lies on the supporting plane of $\hat{f}(e)$ and is bounded from below by $\hat{C}$ and two rays at the endpoints of $\hat{C}$ that are perpendicular to $\overline{e(0)}$. The *lower envelope* of these slabs is denoted by $L(P)$.

Lemma 2.14 ([CV07]). *For a simple non-degenerate polygon P holds $L(P) = \mathcal{T}(P)$.*

Proof. Consider a point p on $\mathcal{T}(P)$. Because p sits on a tilted face $\hat{f}(e)$ of a wavefront edge, it follows that p is also contained in the corresponding slab of and, thereby, not below $L(P)$. It remains to show that a point p of $\mathcal{T}(P)$ is not above $L(P)$. Assume, to the contrary, that p is above any slab of an wavefront edge e, see Figure 22. One can project p down onto the slab and obtain the point u. Then one can project u along the steepest descent of the slab until the lower boundary of the slab is hit at the point v. By Corollary 2.12, the point v is above or just on $\mathcal{T}(P)$. Hence, one can project v onto $\mathcal{T}(P)$ and obtain q, which is below v or identical to v. The slope of the line $\overline{pq}$ is at most 1, by Lemma 2.1. On the other hand, the slope of $\overline{pq}$ is strictly greater than the slope of $\overline{uv}$, which is exactly 1. This is a contradiction. □

Cheng and Vigneron [CV07] pursued the approach of Eppstein and Erickson to define their set of slabs. They reuse the term *edge slab* from [EE99] and, instead of reflex slabs, they introduce the so-called *motorcycle slabs*. The approach presented above, however, extends more naturally to arbitrary planar straight-line graphs, see Section 2.4.

Let us revisit the discussion concerning an alternative characterization of straight skeletons after Corollary 2.8. Recall that the lower envelope of the slabs of Eppstein and Erickson [EE99] does not lead to an alternative characterization of straight skeletons, because the sizes of the slabs depend on the length of reflex arcs. The slabs of Cheng and Vigneron [CV07], however, do indeed lead to an alternative characterization of straight skeletons: Their slabs only depend on the motorcycle graph. Again, a single slab is not only locally defined by the corresponding wavefront edge. Nevertheless, the contributions of Cheng and Vigneron [CV07] only apply to non-degenerate polygons, whereas the results of Eppstein and Erickson [EE99] apply to arbitrary planar straight-line graphs.

We want to give three reasons why Theorem 2.11 appears to be important. Firstly, the motorcycle graph is an extraction of the "essential sub problem" of computing straight skeletons. Hence, it is important to know the geometric relation between straight skeletons and motorcycle graphs. Secondly, the motorcycle graph helps to devise algorithms to compute the straight skeleton. Thirdly, the motorcycle graph helps to devise an alternative characterization of straight skeletons. Aichholzer and Aurenhammer [AA96] already remarked that it is desirable to find a non-procedural definition of straight skeletons. For this reasons we consider our generalization of Theorem 2.11 to arbitrary planar straight-line graphs in Section 2.4 as important.

Cheng and Vigneron [CV07] presented a randomized straight-skeleton algorithm for simple non-degenerate polygons P, which is based on Theorem 2.11, see Section 1.4.2.4. They also present an extension of their algorithm to non-degenerate polygons P with holes. It is easy to see that Theorem 2.11 holds for these slightly more general polygons as well, after generalizing the definition of the motorcycle graph induced by P accordingly.

2.2 THE TRIANGULATION-BASED APPROACH

In this section, we study the number of flip events that occur in the triangulation-based straight-skeleton algorithm by Aichholzer and Aurenhammer [AA98]. The upper bound of $O(n^3)$ is easy to see, cf. Section 1.4.2.2. However, to the best of our knowledge, no n-vertex polygon or planar straight-line graph is known that exceeds a quadratic number of flip events. This circumstance constitutes a gap by a linear factor and the question remains

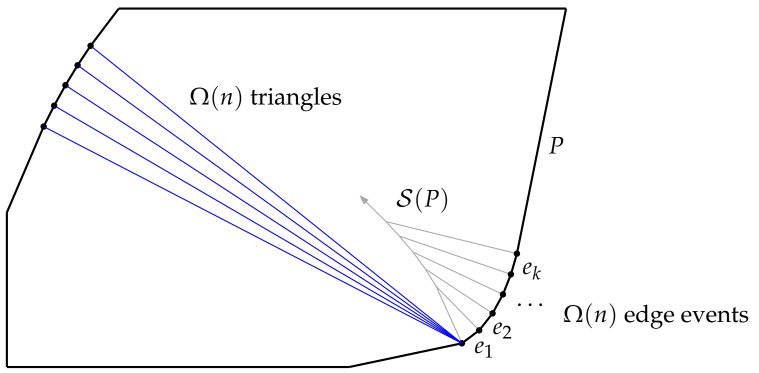


Figure 23: A convex polygon P with n vertices that causes a $\Theta(n^2 \log n)$ runtime for the triangulation-base straight-skeleton algorithm. Triangulation diagonals are shown in blue. Part of the straight skeleton is depicted in gray. The edge events for the edges $e_1, \ldots, e_k$, with $k \in \Omega(n)$, occur in the order $e_1, \ldots, e_k$. For each edge event, the collapsing times of $\Omega(n)$ incident triangles have to be updated, which consumes $\Theta(n^2 \log n)$ time in total.

open, whether the number of flip events is actually bound by $O(n^2)$. Besides, note that processing edge and split events already takes $O(n^2 \log n)$ time: Even though there are only $\Theta(n)$ edge and split events in total, a single event requires up to $O(n \log n)$ time, because the collapsing times of $O(n)$ triangles may need an update in the priority queue. In fact, even a modest convex polygon can lead to a runtime consumption of $\Theta(n^2 \log n)$, as illustrated in Figure 23.

2.2.1 The number of reappearances of diagonals

In Section 1.4.2.2, we explained the $O(n^3)$ bound for the number of flip events, by observing that three constantly moving points do not get collinear more than twice and every flip event corresponds to a collinearity of a triple of vertices. But note that not every collinearity does necessarily correspond to a flip event. Let us consider a triangulation diagonal between two vertices A and B. This diagonal may disappear because another vertex S crosses this diagonal during the propagation process. However, it could be possible that the diagonal is restored by subsequent flip events as illustrated in Figure 24. In order to have the diagonal AB restored, the vertex S has to back off such that A and B see each other again, as illustrated in Step 2. Hence, the vertices A, B and S got collinear twice and S cannot cause a flip event for AB again. In other words, S cannot flip the diagonal AB twice. Nevertheless, Figure 24 does not prove that the diagonal AB can be actually restored. We just considered the necessary topological changes and not the proper geometric setting that leads to these topological changes. How often can a single diagonal, say AB, actually reappear? If this number would be in $O(1)$ then we would conclude that the number of flip events is in fact in $O(n^2)$, because there are only $\binom{n}{2}$ pairs of vertices that can be connected by a diagonal. Unfortunately, the following lemma does not even state that a single diagonal can reappear $\Omega(n)$ times, but $\Omega(n)$ diagonals can each reappear $\Omega(n)$ times, which leads to $\Omega(n^2)$ flip events in total.

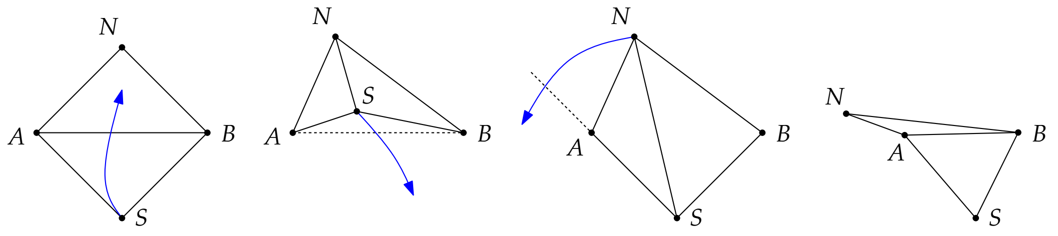


Figure 24: A sequence of flip events which leads to the reappearance of the diagonal AB between the vertices A and B.

Lemma 2.15 ([HH10b]). *There exist polygons P with n vertices and triangulations T of P such that $\Omega(n)$ diagonals each reappear $\Omega(n)$ times during the wavefront propagation.*

Proof. We prove the statement in three steps. In each step we give a geometric setting of moving vertices, for which a specific sequence of flip events occurs. At first, we describe how a single diagonal AB reappears twice. In the second step, we extend the construction such that AB reappears $\Omega(n)$ times. In the third step, we show how k diagonals $A_1B, \dots, A_kB$ reappear each $\Omega(n)$ times, with $k \in \Omega(n)$.

Let us consider six vertices A, B, S_0, S_1, N_0, N_1. The goal is to carefully construct start points and velocities for these six vertices such that the topological transitions, as illustrated in Figure 25, are executed. We denote by $V(t)$ the position of a vertex V at time t. The basic building block of this proof works as follows: We want that the vertex S_0 flips the diagonal AB and after that backs off such that the vertices A and B see each other again. All vertices are bound to move with constant speed. We let A and B drive vertically upwards and we assume that $A(0)$ is at the origin and $B(0)$ at $(1,0)$. We denote by v_A and v_B the speeds of A and B and we further assume that $v_A = 2$ and $1 < v_B/v_A \ll 2$.

The movement of the vertex S_0 is completely determined by choosing the loci of $S_0(1)$ and $S_0(1+\Delta)$, where $\Delta \in (0, 1/4)$ is fixed. The locus of $S_0(1)$ is chosen to be on the straight-line segment $[A(1)B(1)]$ and $S_0(1+\Delta)$ is chosen to be on $[A(1+\Delta)B(1+\Delta)]$. We further require that S_0 moves to the north-east, i. e. that $S_0(1+\Delta)$ lies strictly to the right to $S_0(1)$. The determinant $\det(A(t), B(t), S_0(t))$, which is

$$\begin{vmatrix} A_x(t) & B_x(t) & S_{0,x}(t) \\ A_y(t) & B_y(t) & S_{0,y}(t) \\ 1 & 1 & 1 \end{vmatrix}, \tag{2.3}$$

is a quadratic polynomial in t and its sign corresponds to the orientation of the three vertices. (The x- and y-coordinates of the vertices A, B, S_0 are denoted by subscripts.) We observe that for some $t > 1 + \Delta$ the points $A(t), B(t), S_0(t)$ are in clockwise position. This is because the vertex S_0 moves to north-east and $v_B/v_A > 1$. Knowing that $\det(A(t), B(t), S_0(t))$ is a quadratic polynomial with roots $\{1, 1+\Delta\}$, and by observing that $A(t), B(t), S_0(t)$ are in clockwise position for some $t > 1 + \Delta$, we conclude that the triangle $A(t), B(t), S_0(t)$ is clockwise for all $t < 1$, counter-clockwise for all $t \in (1, 1+\Delta)$ and clockwise again for all $t > 1 + \Delta$. In other words, we constructed a vertex S_0 which indeed executes the first two transitions in Figure 25.

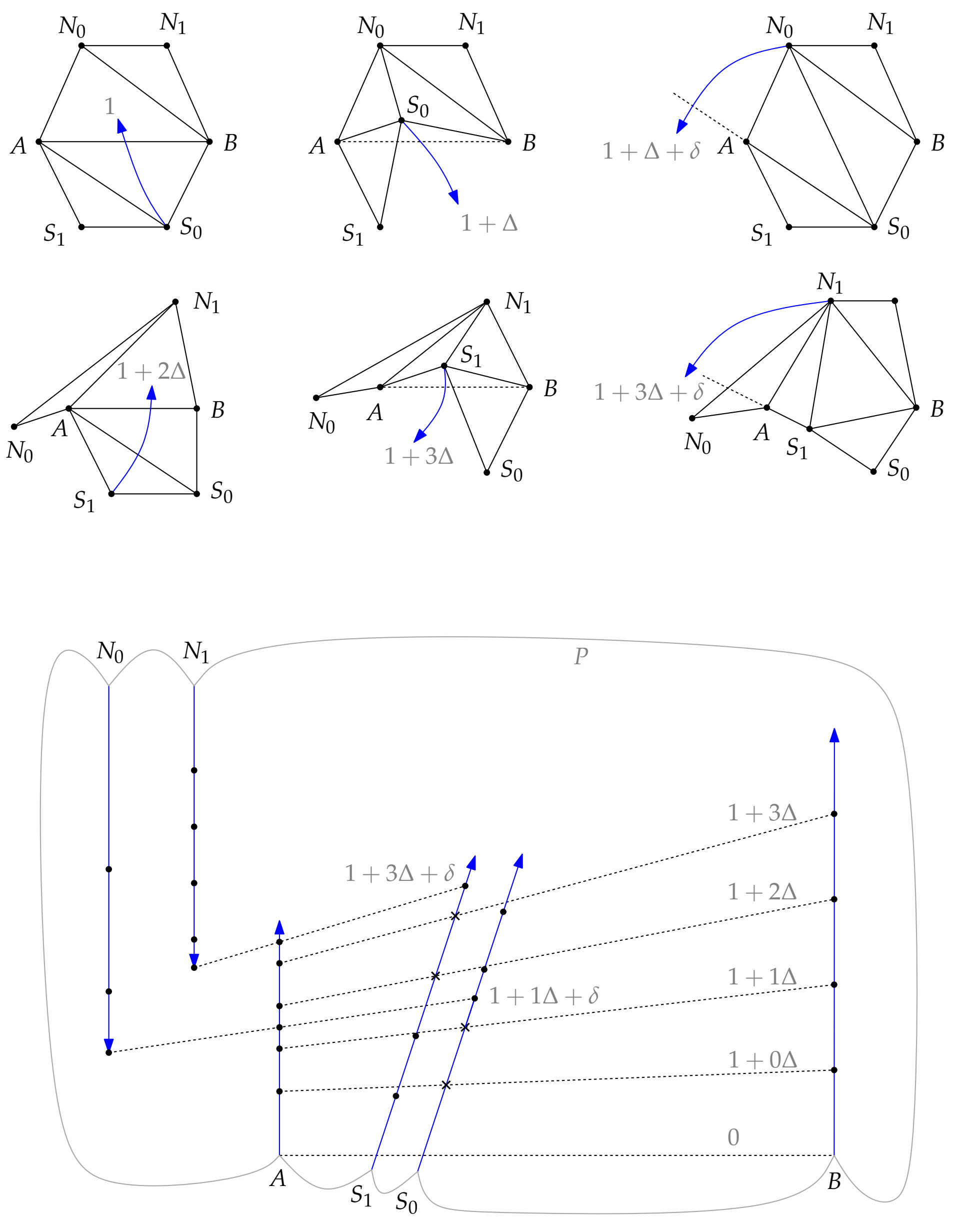


Figure 25: Step 1 of the proof of Lemma 2.15: The diagonal AB reappears twice since the geometric setting at the bottom produces the sequence of flip events at the top. Bottom: Every black dot denotes the position of a vertex at the time given which is depicted in gray. A cross mark indicates collinearity for A and B with S_0 resp. S_1. Top: Each figure illustrates a topological transition. The second reappearance is caused by the sixth transition.

According to our desired sequence of transitions, we need a vertex N_0 such that $N_0(t)$ gets collinear with $A(t)$ and $S_0(t)$ for some $t > 1 + \Delta$, say at $t = 1 + \Delta + \delta$, where $\delta \in (0, \Delta/2)$ is fixed. (In the sequel, we have to choose δ small enough.) We place a vertex N_0 that moves southwards and parallel to A. Furthermore, we choose $N_{0,y}(0) = 8$ such that $N_0(0)$ is strictly above $B(1 + 4\Delta)$. Next, we request that $N_0(1 + \Delta + \delta)$ is on the supporting line $\overline{A(1 + \Delta + \delta)\, B(1 + \Delta + \delta)}$. Hence, at time $1 + \Delta + \delta$ the vertex N_0 causes the recreation of the diagonal AB by flipping the diagonal N_0S_0.

In order to conclude the first step of the proof, we repeat the life-cycle of the diagonal AB, by using the vertices S_1, N_1 instead of S_0, N_0. We place the vertex S_1 between A and S_0 and let S_1 move parallel to S_0. The locus of $S_1(t)$ is chosen to be on $[A(t)B(t)]$ for $t \in \{1 + 2\Delta, 1 + 3\Delta\}$. Applying the same argument as for S_0, we observe that the triangle $A(t), B(t), S_1(t)$ is clockwise for all $t < 1 + 2\Delta$, counter-clockwise for all $t \in (1 + 2\Delta, 1 + 3\Delta)$ and clockwise again for all $t > 1 + 3\Delta$. Note that S_0 is already behind $\overline{A(t)B(t)}$ for $t > 1 + \Delta$. Hence, S_1 causes a flip event for the diagonal AB at $t = 1 + 2\Delta$ and falls back at $t = 1 + 3\Delta$ such that the vertices A and B see each other again for $t > 1 + 3\Delta$. Also note that the introduction of S_1 does not interfere with the transitions caused by S_0 and N_0, since S_1 is behind the supporting line of A, B, S_0 at $t = 1 + \Delta$. Hence, by choosing δ small enough, S_1 is also behind the supporting line of A and S_0 at $t = 1 + \Delta + \delta$.

Finally, we place a vertex N_1 between N_0 and A which again moves southwards and parallel to A. We assume that N_0 and N_1 start from the same horizontal line and we require that $N_1(1 + 3\Delta + \delta)$ is collinear with $A(1 + 3\Delta + \delta)$ and $S_1(1 + 3\Delta + \delta)$. Hence, the diagonal AB is recreated again by an edge flip of the diagonal N_1S_1.

To sum up, we are able to construct a geometric setting of moving vertices such that the diagonal between the vertices A and B reappears twice. However, in order to finish Step 1 of the proof, we have to guarantee that there is a polygon P and a triangulation T of P that realizes the sequence of transitions in Figure 25. First, we note that we can choose Δ very small such that the speeds of N_0 and N_1 get arbitrarily close. Next, we note that if we choose $v_B/v_A > 1$ but arbitrarily close to 1 then the points $S_1(0)$ and $S_2(0)$ approach the supporting line of $A(0)$ and $B(0)$. Hence, we can construct a polygon P, as in Figure 25, such that only the vertices A, S_1, S_0, B, N_1, N_0 of P are reflex and connected by convex chains. By choosing the incident polygon edges of each of these vertices accordingly, we obtain the desired velocities for the propagating wavefront vertices. The initial triangulation T of P contains the edges given in Figure 25 and the remaining faces can be triangulated arbitrarily.

For Step 2 we extend our construction by adding vertices $S_2, \dots, S_m$ from right to left between S_1 and A, with $m \in \Omega(n)$. We repeat the construction scheme presented above for these new vertices. Furthermore, we choose $\Delta < 1/2m+2$ in order to make the new vertices fit. Likewise we add vertices $N_2, \dots, N_m$ from left to right between N_1 and A. In general, for each $i \in \{0, \dots, m\}$ the vertex S_i causes a flip event for the diagonal AB at time $1 + 2i\Delta$ and falls back behind $\overline{A(t)\, B(t)}$ at time $t = 1 + (2i + 1)\Delta$. At time $t = 1 + (2i + 1)\Delta + \delta$ the vertex N_i gets collinear with $A(t)$ and $S_i(t)$ and causes the recreation of the diagonal AB.

The basic idea of Step 3 is to rename the vertex A to A_1 and to carefully place copies $A_2, \dots, A_k$ of A_1 from right to left. In the following, we only consider a single reappearance cycle for each diagonal $A_1B, \dots, A_kB$, caused by flip events due to vertex S_i and the subsequent restoration due to vertex N_i, for an arbitrary $i \in \{0, \dots, m\}$. We illustrated the topo-

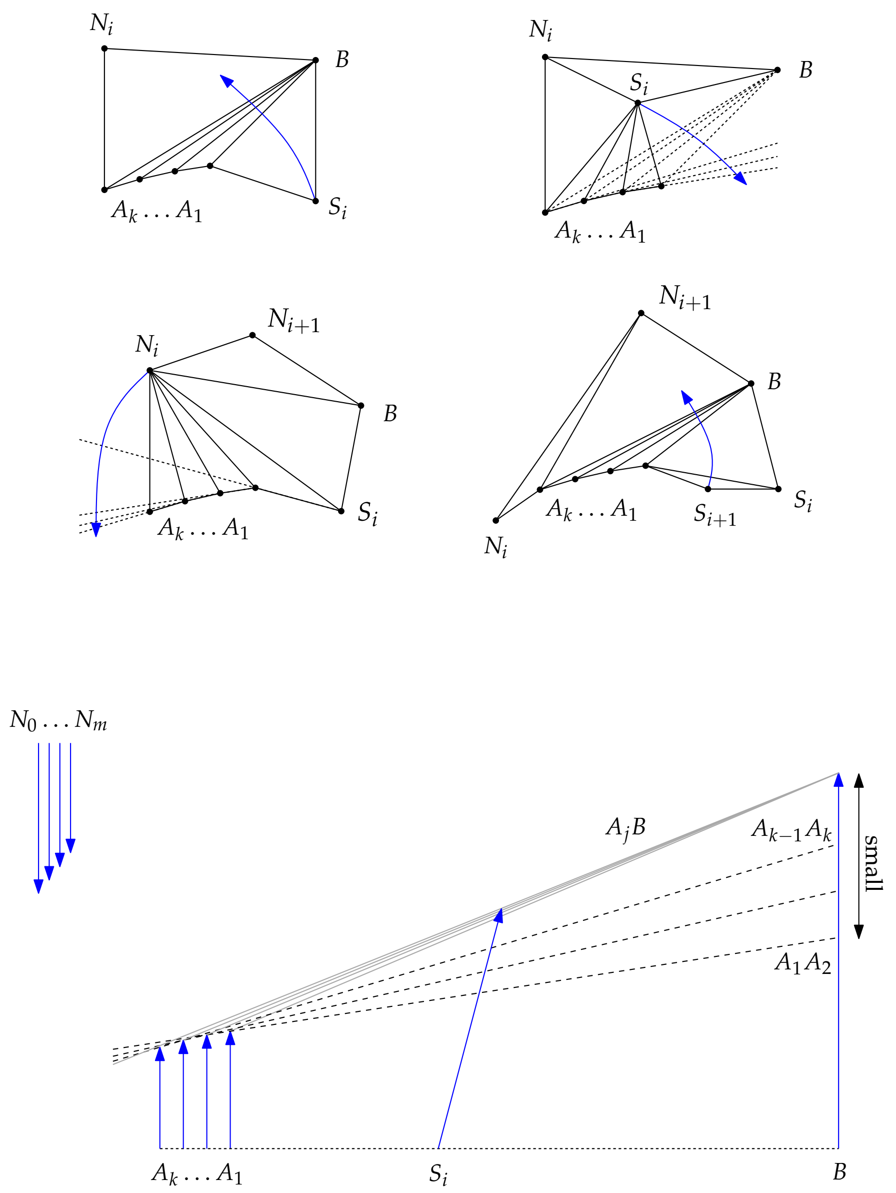


Figure 26: Step 3 of the proof of Lemma 2.15: We place copies of A_1 in order to obtain diagonals $A_1B, \dots, A_kB$. By arranging the vertices $A_2, \dots, A_k$ almost collinear with A_1B the diagonals $A_2B, \dots, A_kB$ behave like the diagonal A_1B from Step 2.

logical transitions in Figure 26. Let us assume that the points $B(0), A_1(0), \ldots, A_k(0)$ are arranged on a horizontal line, see Figure 26. Furthermore, assume that $B(1), A_1(1), \ldots, A_k(1)$ are almost on a straight line, by arranging $A_2(1), \ldots, A_k(1)$ accordingly. However, for any $j \in \{1, \ldots, k-1\}$ each $A_{j+1}(1)$ shall properly see the vertex $B(1)$ and every supporting line $\overline{A_j(1)A_{j+1}(1)}$ shall intersect the trajectory of B in an ascending order. Moreover, the intersection points shall be below but arbitrarily close to $B(1)$, see Figure 26. The idea is to arrange $A_2(1), \ldots, A_k(1)$ very close to $\overline{A_1(1)\, B(1)}$ such that the diagonals $A_1B, \ldots, A_kB$ behave almost the same as the single diagonal AB from Step 2. (Note that if $B, A_1, \ldots, A_k$ are collinear at time 1 then they are collinear at all times, due to the intercept theorems.)

We recall from Step 2 that $\det(A_1(t), B(t), S_i(t))$ is a quadratic polynomial in t with roots $\{1 + 2i\Delta, 1 + (2i+1)\Delta\}$. Furthermore, $A_1(t), B(t), S_i(t)$ is in clockwise position for all $t < 1 + 2i\Delta$, in counter-clockwise position for $t \in (1 + 2i\Delta, 1 + (2i+1)\Delta)$ and again in clockwise position for $t > 1 + (2i+1)\Delta$. Consider $\tau \in (0, \delta)$ to be fixed. It follows that the triangle $A_1(t)B(t)S_i(t)$ is in a strict counter-clockwise position for $t = 1 + 2i\Delta + \tau$. Recall that S_i crosses the diagonal A_1B at $t = 1 + 2i\Delta$. We now choose $B(1), A_1(1), \ldots, A_k(1)$ as close to collinear as possible such that S_i also crosses the diagonal A_jB before $t = 1 + 2i\Delta + \tau$ for all $j \in \{1, \ldots, k\}$. That is, S_i causes a flip event for each diagonal $A_1B, \ldots, A_kB$ within a temporal tolerance of τ after $t = 1 + 2i\Delta$. Hence, the first topological transition in Figure 26 happened after $t = 1 + 2i\Delta + \tau$.

According to the second transition in Figure 26, we need S_i to fall back behind the former diagonals $A_1B, \ldots, A_kB$. However, in addition we require that S_i also falls behind the supporting lines of $\overline{A_j\, A_{j+1}}$ for all $j \in \{1, \ldots, k-1\}$. This circumstance leads to flip events of the diagonals $A_2S_i, \ldots, A_{k-1}S_i$ such that all $A_1, \ldots A_k$ are connected to N_i. From Step 2 we know that S_i falls behind A_1B at time $t = 1 + (2i+1)\Delta$. For Step 3 we choose $B(1), A_1(1), \ldots, A_k(1)$ as close to collinear as necessary such that S_i falls behind the supporting lines $\overline{A_j\, A_{j+1}}$ until $t = 1 + (2i+1)\Delta + \tau$.

In order to execute the third transition in Figure 26, we finally require that N_i crosses the supporting lines $\overline{A_j\, A_{j+1}}$ until $t = 1 + (2i+1)\Delta + \delta + \tau$. After that, all diagonals $A_1B, \ldots, A_kB$ are restored and we completed the reappearance cycle caused by S_i and N_i.

To sum up, we can arrange $A_2(1), \ldots, A_k(1)$ as close as necessary to $\overline{A_1(1)\, B(1)}$ such that the topological transitions in Figure 26 are executed for all S_i, N_i, with $0 \leq i \leq m$, within a temporal tolerance of $\tau \in (0, \delta)$. Furthermore, we observe that at time zero the vertices $B, A_1, \ldots, A_k$ are on a horizontal line. In order to enable diagonals $A_1B, \ldots, A_kB$ in the initial triangulation, we start our simulation at time $-\epsilon$, for a sufficiently small $\epsilon > 0$. $\square$

2.2.2 Good triangulations and bad polygons

As a byproduct of Lemma 2.15, we obtain that polygons P with n vertices and corresponding triangulations exist for which $\Omega(n^2)$ flip events occur. Besides the actual shape of the polygon, this result also hinges on the specific initial triangulation. If the initial triangulation from Lemma 2.15 would not contain the diagonals $A_1B, \ldots A_kB$, but, for example, the diagonals $S_0N_m, \ldots, S_mN_m$ then we would easily avoid the occurrence of $\Omega(n^2)$ flip events. Can we always find for a polygon P a "good" initial triangulation, for which only a few

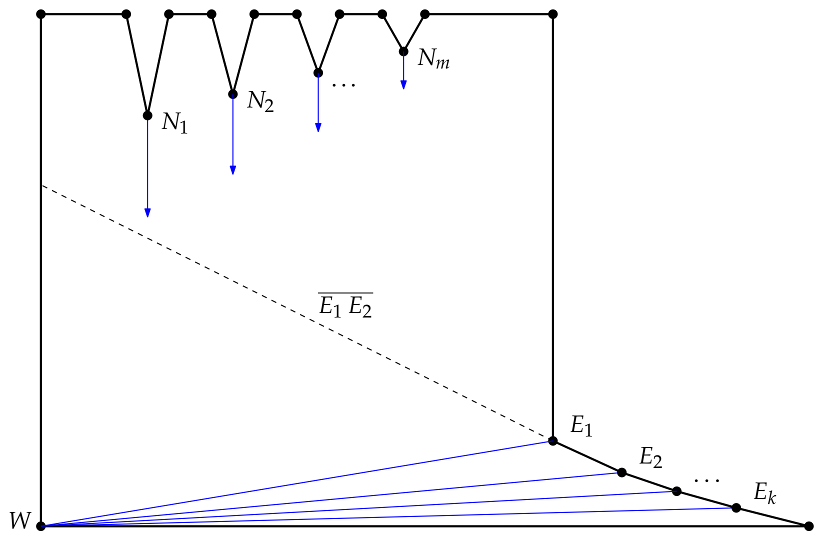


Figure 27: An n-vertex polygon for which any triangulation leads to $\Omega(n^2)$ flip events.

number of flip events occur — say, at most $o(n^2)$ or even $O(n)$? Or, conversely, are there "bad" polygons, for which any triangulation ends up with a large number of flip events?

Lemma 2.16 ([HH10b]). *There exist polygons P with n vertices, for which any triangulation leads to $\Omega(n^2)$ flip events.*

Proof. Let us consider the polygon illustrated in Figure 27. The polygon has $k \in \Omega(n)$ vertices $E_1, \dots, E_k$ that form a reflex chain. The supporting line $\overline{E_1 E_2}$ is chosen in a way such that $E_2, \dots, E_k$ only see the vertex W (except for their neighboring vertices, of course). Hence, any triangulation for P necessarily contains the diagonals $E_1W, \dots, E_kW$. The idea is that each of these $\Omega(n)$ diagonals is flipped $\Omega(n)$ times by vertices $N_1, \dots, N_m$, with $m \in \Omega(n)$. We choose the vertex N_i fast enough such that it leads to a split event with the bottom polygon edge before N_{i+1} crosses $\overline{E_1 E_2}$. Furthermore, we make the vertices $N_1, \dots, N_m$ fast enough such that the split events for $N_1, \dots, N_m$ happen before any other split or edge event happens.

If we choose the speeds of $N_1, \dots, N_m$ accordingly then we observe that N_1 leads to $\Omega(n)$ flip events with $E_1W, \dots, E_kW$ and we obtain diagonals $N_1E_1, \dots N_1, E_k$. Next, the vertex N_2 flips the diagonals $N_1E_1, \dots, N_1E_k$ and we obtain diagonals $N_2E_1, \dots, N_2E_k$. (In the meanwhile N_1 could have caused a split event already. However, the corresponding diagonals remain incident to a wavefront vertex emanated from this split event and will be flipped by N_2.) At the end, each vertex $N_1, \dots, N_m$ will flip $\Omega(n)$ diagonals and we obtain $\Omega(n^2)$ flip events in total. □

In order to reduce the number of flip events it seems reasonable to try to re-triangulate the wavefront $\mathcal{W}(P, t)$ at favorable moments. In other words, we could try to pay the costs for computing a new triangulating in exchange to the costs caused for a certain number of flip events. However, we want to remark that one re-triangulation of the polygon constructed in

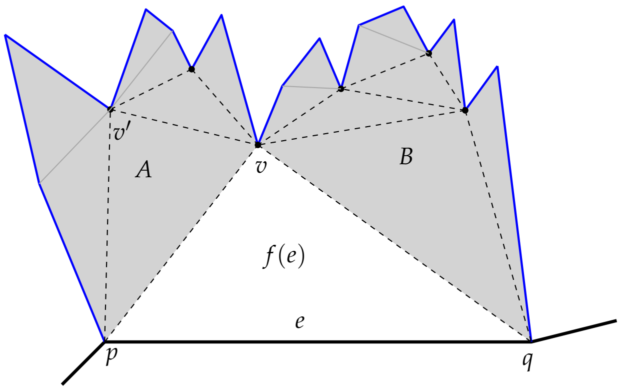


Figure 28: A recursive triangulation scheme for the face $f(e)$ which is free of flip events.

the proof of Lemma 2.16 at any point in time saves at most $O(n)$ flip events: If N_i is already below $\overline{E_1 E_2}$ then all vertices $N_1, \ldots, N_{i-1}$ already caused $\Omega((i-1)n)$ flip events and N_{i+1} is still above $\overline{E_1 E_2}$. Hence, we avoid at most $O(n)$ flip events that would be caused by N_i itself and pay the costs for a re-triangulation.

2.2.3 Steiner triangulations without flip events

Lemma 2.16 tells us that ordinary triangulations do not allow a strategy to keep the number of flip events below $O(n^2)$ for arbitrary polygons. However, we observe that if we use Steiner[1] points, we could reduce the number of flip events significantly for the polygon illustrated in Figure 27. Nevertheless, introducing Steiner points to the initial triangulation in the algorithm by Aichholzer and Aurenhammer [AA98] obtrudes several questions: (i) how do Steiner points interact with the wavefront, (ii) how do we determine the loci of the Steiner points and (iii) how effective is the introduction of Steiner points in order to reduce the number of flip events? Regarding the first question, we assume that the Steiner points keep still. Furthermore, it seems natural to maintain the property that the triangulation keeps the area $\mathbb{R}^2 \setminus \bigcup_{t' \leq t} \mathcal{W}(P, t')$ triangulated for any time $t \geq 0$. We give an answer to the remaining two questions by the following lemma.

Lemma 2.17 ([HH10b])**.** *Every simple polygon P with n vertices admits a triangulation that employs at most $n-2$ Steiner points and is free of flip events.*

Proof. We prove the statement by first presenting a proper set of Steiner points and a proper triangulation. In the sequel we show that no flip events occur for that particular triangulation.

First, we consider the straight skeleton $\mathcal{S}(P)$ of P, which has $n-2$ inner nodes. Each inner node serves as a Steiner point in our triangulation. Next, we add the arcs of $\mathcal{S}(P)$ as diagonals to the triangulation. It remains to properly triangulate the faces of $\mathcal{S}(P)$.

1 Steiner points are additional points at favorable loci and which can be used for the triangulation.

Let $f(e)$ denote an arbitrary face of $\mathcal{S}(P)$ for a wavefront edge e of P. We triangulate $f(e)$ in a recursive manner as follows. Let us denote by p and q the endpoints of e. If we consider $f(e)$ as a polygon we know due to Lemma 2.3 that $f(e)$ is monotone and due to Lemma 2.9 that reflex vertices only appear in the upper chain of $f(e)$. If $f(e)$ is convex then we triangulate it arbitrarily. Otherwise we denote by v a reflex vertex of $f(e)$ with minimum orthogonal distance to $\overline{e}$, see Figure 28. We note that the line segments $[pv]$ and $[qv]$ are completely contained in $f(e)$. Otherwise there would be another reflex vertex of $f(e)$ whose orthogonal distance to e would be less than for v. We insert pv and qv as diagonals to the triangulation. In the sequel we call pv and qv the two diagonals that belong to v.

The triangle pqv tessellates $f(e)$ into (at most) two parts. We denote by A the part to which the diagonal pv belongs to and by B the part to which qv belongs. In both parts A and B we proceed recursively: If the part A is convex we triangulate A arbitrarily. Otherwise we denote by v' the reflex vertex of A with minimum orthogonal distance to $\overline{e}$. We add pv' and vv' as diagonals. The triangle pvv' tessellates A into (at most) two parts and so on.

It remains to show that the triangulation presented above is free of flip events. First of all, we observe that during the wavefront propagation all vertices of the wavefront move on the diagonals of the triangulation. Hence, no reflex wavefront vertex can cause a flip event. However, we also have to guarantee that no triangulation diagonal is moving over a Steiner point. More precisely, we have to exclude the case that a triangulation diagonal moves over a Steiner point that appears as a reflex vertex of the polygon $f(e)$. However, if we consider the propagating wavefront within $f(e)$ then we only need to consider the wavefront edge e, which is sweeping $f(e)$ in a self-parallel manner and is split at any reflex vertex of $f(e)$. If there would exist a diagonal that moves over a reflex vertex v of $f(e)$ then it would follow that this diagonal crosses at least one of the two diagonals that belong to v. □

The proof of the above Lemma obviously does not provide a new straight-skeleton algorithm since we employ the straight skeleton in order to construct an appropriate Steiner triangulation. Nevertheless, the lemma tells us at least that there exist triangulations that are free of flip events. Consequently, we can ask for Steiner triangulations which are free of flip events and which are constructed without the help of straight skeletons. We also want to remark that Lemma 2.17 is not restricted to polygons only and we can easily apply the very same proof to planar straight-line graphs as well.

Corollary 2.18. *Every planar straight-line graph G with n vertices admits a triangulation that employs $O(n)$ Steiner points and is free of flip events.*

2.3 A NOVEL WAVEFRONT-TYPE APPROACH

2.3.1 Motivation

In the previous section we learned that the use of Steiner vertices can be advantageous in order to reduce the number of flip events in the triangulation-based straight-skeleton algorithm by Aichholzer and Aurenhammer [AA98]. However, the following question remained open: How do we find a good set of Steiner points without knowing the straight skeleton?

In the original approach a flip event occurs when a reflex wavefront vertex crosses a triangulation diagonal. These flip events are avoided for the Steiner triangulation presented in the proof of Lemma 2.17, because the reflex wavefront vertices move *along* triangulation diagonals. The idea is to find a set of Steiner points and a triangulation such that the trajectories of the reflex wavefront vertices are covered by triangulation diagonals.

Assume we are given a non-degenerate polygon P. (Recall, P is non-degenerate if no two motorcycles of $\mathcal{M}(P)$ crash simultaneously at the same point.) By Theorem 2.11 we know that the motorcycle graph $\mathcal{M}(P)$ covers the reflex arcs of the straight skeleton $\mathcal{S}(P)$. We consider the endpoints of the motorcycle traces of $\mathcal{M}(P)$ as Steiner points and the traces as diagonals of the triangulation. Theorem 2.11 guarantees that a reflex wavefront vertex does not move beyond its corresponding motorcycle trace. Hence, we obtained a similar situation as in Lemma 2.17, where no flip events are caused by reflex wavefront vertices. In the following we will discuss these two questions: (i) what happens when the wavefront meets a Steiner point and (ii) how do we triangulate the remaining faces?

2.3.2 The extended wavefront and a novel straight-skeleton algorithm

Consider a motorcycle trace that starts at a reflex wavefront vertex v of $\mathcal{W}(P,0)$ and ends at an edge of P at point p. By our construction above, we obtain a Steiner point at p and a triangulation diagonal covering the motorcycle trace that starts at v and ends in p. We define that while the wavefront propagates, the Steiner point at p accompanies the moving intersection of $\mathcal{W}(P,t)$ and the motorcycle trace. Metaphorically speaking, the point p "surfs" on the wavefront, along the motorcycle trace, towards v. Analogously, if a motorcycle trace starts at v and crashes into another motorcycle trace at point p then the Steiner point at p keeps still until the wavefront $\mathcal{W}(P,t)$ meets p. From that moment, p starts moving on the motorcycle trace towards v. To sum up, we maintain the intersection of the motorcycle graph $\mathcal{M}(P)$ and the propagating wavefront $\mathcal{W}(P,t)$ and we refer to the additional points as Steiner vertices. More precisely, we call a Steiner vertex, which has not yet been met by the wavefront, a *resting Steiner vertex* and we call a Steiner vertex, which already moves on the intersection of a motorcycle trace and the wavefront, a *moving Steiner vertex*. Every resting Steiner vertex lies on the intersection of two motorcycle traces and eventually becomes a moving Steiner vertex when it is reached by the wavefront, see Figure 29.

Definition 2.19 (extended wavefront)**.** Let P be a simple non-degenerate polygon. We denote by $\mathcal{M}(P,t)$ those parts of $\mathcal{M}(P)$ which have not been swept by $\mathcal{W}(P,t')$ for $t' < t$. The *extended wavefront* $\mathcal{W}^*(P,t)$ is defined as the overlay of $\mathcal{W}(P,t)$ and $\mathcal{M}(P,t)$ by splitting the edges of $\mathcal{W}(P,t)$ at the intersection points accordingly.

Figure 29 illustrates the extended wavefront $\mathcal{W}^*(P,t)$ for a simple polygon P. In the following, we want to remark that P denotes the filled polygon and not only its boundary.

Lemma 2.20. *We denote by P a simple non-degenerate polygon. Let p be a point in the relative interior of $\mathcal{M}(P)$. Then a local disk around p is tessellated into convex slices by $\mathcal{M}(P)$.*

Proof. Since P is non-degenerate it follows that p is also in the relative interior of a motorcycle trace. □

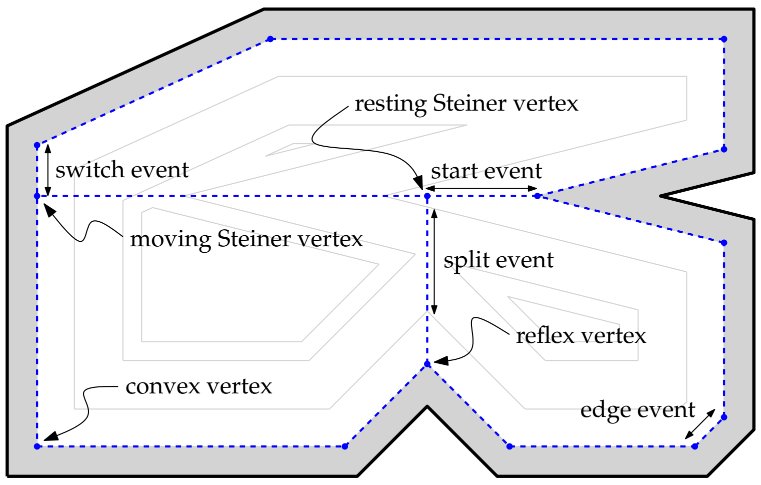


Figure 29: A non-degenerate polygon (bold) and the extended wavefront (dashed) after some time. The area already swept by the wavefront is shaded. The offset curves for three further points in time have been depicted in gray.

Lemma 2.21. *For any $t \geq 0$ the set $P \setminus \bigcup_{t' \in [0,t]} \mathcal{W}^*(P,t')$ consists of open convex faces only.*

Proof. Lemma 2.20 and the fact that at each reflex vertex of P the interior angle is halved by a motorcycle trace, proves the lemma. □

We now tackle the remaining question concerning the triangulation of the faces induced by the extended wavefront. Recall that we regard the nodes of the motorcycle graph as Steiner vertices and the edges of the motorcycle graph as triangulation diagonals. It follows by Lemma 2.21 that during the propagation of the extended wavefront $\mathcal{W}^*(P,t)$ only neighboring vertices on $\mathcal{W}^*(P,t)$ can meet, since the resulting faces are at any time convex. Hence, we can simply determine all topological changes that occur to $\mathcal{W}^*(P,t)$ by just considering neighboring vertices. In other words, there is no need for a triangulation in order to identify the topological changes.

Summarizing, our approach to compute straight skeletons of simple non-degenerate polygons is not to simulate the wavefront $\mathcal{W}(P,t)$, but to simulate the extended wavefront $\mathcal{W}^*(P,t)$. As mentioned in Section 1.4.2.1, the difficult problem when simulating the original wavefront $\mathcal{W}(P,t)$ is the determination of the split events. In our approach, however, split events are easily determined: A moving Steiner vertex meets a reflex wavefront vertex and both move on the same motorcycle trace. Moreover, every topological change in the extended wavefront $\mathcal{W}^*(P,t)$ is indicated by the collision of two neighboring vertices. In other words, each topological change corresponds to an edge of $\mathcal{W}^*(P,t)$ that collapses to zero length and vice versa. Our algorithm maintains a priority queue Q that contains for each topological change an event. We fetch all events in chronological order and process them. That means, we maintain the extended wavefront depending on the type of event that happened. We distinguish the following types of events. (A detailed discussion is given in

Section 2.5.) Note that the different types of events are easily distinguished by the types of vertices involved.

- **(Classical) edge event**: Two convex vertices u and v meet. We merge u and v to a single convex vertex and recompute the collapsing times of the two incident edges of $\mathcal{W}^*(P,t)$. Further, we add the straight-skeleton arcs that have been traced out by u and v to the growing straight-skeleton structure. As a special case, we check whether a whole triangle of $\mathcal{W}^*(P,t)$ collapsed by this edge event.
- **(Classical) split event**: A reflex vertex u and a moving Steiner vertex v meet at point p and both move against each other. We add the straight-skeleton arc that has been traced out by u. In general, $\mathcal{W}^*(P,t)$ is split into two parts. We create corresponding convex vertices that start at p connect them with the resulting parts.
- **Start event**: A reflex vertex or a moving Steiner vertex u meets a resting Steiner vertex v. The vertex v becomes a moving Steiner vertex and slides along one incident edge of u. Technically, we have to split an incident edge of u accordingly.
- **Switch event**: A convex vertex u meets a moving Steiner vertex or a reflex Steiner vertex v. The convex vertex u migrates from one convex face of $\mathcal{W}^*(P,t)$ to a neighboring one, by jumping over v. Technically, the vertex v splits the opposite edge of u and we have to recompute the speed of v.
- All remaining combinations of vertices meeting are guaranteed not to happen. For instance, a convex vertex will never meet a resting Steiner vertex, because a resting Steiner vertex can only have reflex vertices or Steiner vertices as neighbors.

The correctness of the algorithm follows directly from Lemma 2.21 and Theorem 2.11. The latter guarantees that reflex vertices stay within their corresponding motorcycle traces, i. e. they lead to a split event before they would move beyond the extended wavefront.

2.3.3 Runtime analysis and conclusion

Each event involves the recomputation of the collapsing times for a certain amount of edges of the extended wavefront and subsequent modifications in the priority queue Q. Note that each vertex of the extended wavefront has degree two or three. Hence, every event involves only a constant amount of modifications and, thereby, can be handled in $O(\log n)$ time.

We know that we have $O(n)$ edge events and $O(r)$ split and start events in total, where $r \in O(n)$ denotes the number of reflex vertices of P. The number of switch events is bound by $O(nr)$, because a convex vertex can meet a moving Steiner vertex only once. Note that the size of the extended wavefront is linear and the priority queue Q contains at most as many events as the number of edges of the extended wavefront. Hence, our algorithm runs in $O(n)$ space.

Lemma 2.22. *Let P denote a simple non-degenerate polygon with n vertices, where r vertices are reflex. If the motorcycle graph $\mathcal{M}(P)$ is known then our algorithm runs in $O((n+k)\log n)$ time and $O(n)$ space, where $k \in O(nr) \subset O(n^2)$ denotes the number of switch events that occurred.*

For real-world input, it seems very unlikely that $\Omega(n^2)$ switch events actually occur. This would mean that $\Omega(n)$ moving Steiner vertices actually meet with $\Omega(n)$ convex vertices.

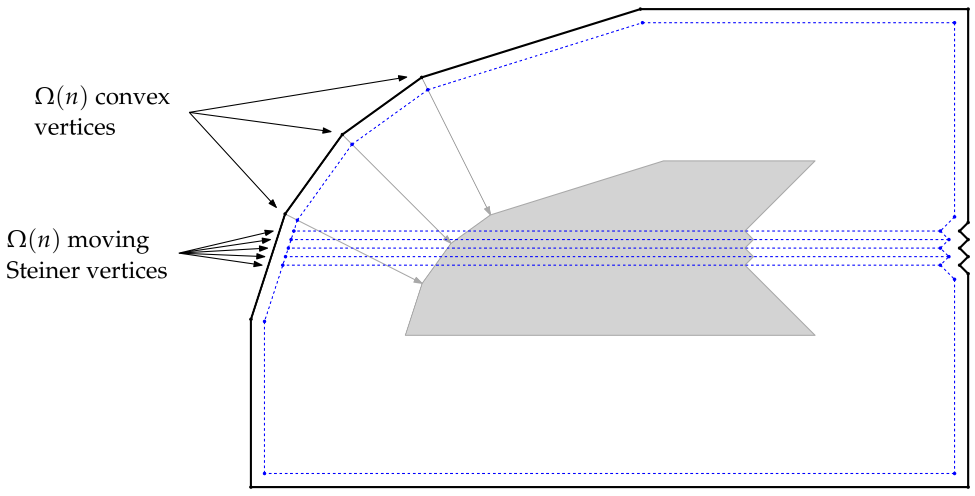

$\Omega(n)$ convex vertices
$\Omega(n)$ moving Steiner vertices

Figure 30: A polygon P for which $\Omega(n^2)$ switch events occur.

This observation is also confirmed by extensive runtime tests in Section 2.5.4. However, a worst-case example, for which $\Theta(n^2)$ switch events actually occur, can be constructed, see Figure 30. Note that we have to take care that the traces of the motorcycles are almost parallel within the whole polygon. Furthermore, we require that the trajectories of convex vertices do not intersect before the $\Omega(n^2)$ switch events occur. That is, the trajectories more or less approach the same point. In other words, this worst-case example is highly contrived.

The motorcycle graph $\mathcal{M}(P)$ of a polygon P with r reflex vertices can be computed in $O(r^{17/11+\epsilon})$ time and space in theory, using the algorithm by Eppstein and Erickson [EE99], cf. Section 1.4.2.3, and in $O(r\sqrt{r}\log r)$ time, using the algorithm by Cheng and Vigneron [CV07], cf. Section 1.4.2.4. In practice, the motorcycle graph can be computed in $O(r^2 \log r)$ time by a straight-forward algorithm enhanced with a priority queue. In Section 3.3, we will present our motorcycle graph implementation Moca, which exhibits an $O(r \log r)$ runtime for practical input.

Summarizing, the algorithm presented in this section has a worst-case runtime complexity of $O(n^2 \log n)$, while it is still easy to implement. This constitutes an improvement by a linear factor compared to the $O(n^3 \log n)$ worst-case complexity of the algorithm by Aichholzer and Aurenhammer [AA98]. However, their algorithm is capable of computing the straight skeleton of planar straight-line graphs instead of non-degenerate polygons.

In the following sections we will extend our approach to arbitrary planar straight-line graphs. In the first place this means that we get rid of the non-degeneracy assumption. In order to achieve this we will need to generalize the motorcycle graph accordingly. As a byproduct, this will also result in an alternative characterization of straight skeletons of planar straight-line graphs. Furthermore, the generalized motorcycle graph will allow us to extend the algorithmic approach to planar straight-line graphs.

2.4 A GENERALIZED MOTORCYCLE GRAPH

2.4.1 Motivation and definition

In order to extend the algorithm that was presented in the previous section to arbitrary planar straight-line graphs, we have to generalize the motorcycle graph from simple non-degenerate polygons to arbitrary planar straight-line graphs. While a generalization to non-degenerate planar straight-line graph appears to be straight-forward, the main challenge is to eliminate the need for the assumption of Cheng and Vigneron [CV07]. That is, we need to allow that two motorcycles may crash simultaneously into each other. No matter how the actual generalization is finally realized, we demand the following two properties to hold for the motorcycle graph $\mathcal{M}(G)$ of a planar straight-line graph G:

1. The motorcycle graph $\mathcal{M}(G)$ has to cover the reflex arcs of $\mathcal{S}(G)$ and
2. the overlay of the motorcycle graph $\mathcal{M}(G)$ and G has to yield a convex tessellation of the plane.

Following the definition of the motorcycle graph induced by a simple polygon in Section 1.2.4, we have to specify the motorcycles and the walls that are induced by a given planar straight-line graph G. First of all, we consider the edges of G to be walls. Secondly, we shoot for each reflex wavefront vertex v of $\mathcal{W}(G,0)$ a motorcycle that starts at v and has the same velocity as v. Note that if G forms a polygon then we obtain motorcycles inside and outside of the polygon. Let us consider a planar straight-line graph G, for which a vertex event occurs at point p. This implies that at least two motorcycles m and m' crashed simultaneously into each other at p. Since a new reflex straight-skeleton arc emanates at p, we need to start a new motorcycle which is going to cover this new straight-skeleton arc. Hence, it will be necessary to generalize the concept of motorcycle graphs such that new motorcycles can emerge at certain moments in time. As a consequence, our generalized motorcycles are specified by a start point, a velocity and, in addition, a start time.

For the matter of simplicity we will rephrase the procedure, by which we obtain the set of motorcycles induced by a planar straight-line graph G. Let us denote by $e(t)$ the straight-line segment occupied by a wavefront edge e at time t. We demand that two wavefront edges e and e' of G belong to each motorcycle m and the position of m at time t is given by the intersection $\overline{e(t)} \cap \overline{e'(t)}$. We call the wavefront edge left to the trace of m the *left arm of* m and the other wavefront edge the *right arm of* m. Obviously, by specifying the start point and the two arms of a motorcycle m, we implicitly specified the start time of m, too.

For any reflex wavefront vertex v in $\mathcal{W}(G,0)$ we define a motorcycle m that starts at v and whose arms are the two incident wavefront edges of v, see Figure 31 (a). Hence, the motorcycle m has the same velocity as v by construction. Furthermore, we note that each terminal vertex of G gives rise to two motorcycles by definition, see Figure 31 (b). If two or more motorcycles $m_1, \ldots, m_k$ crash simultaneously at a point p then we consider a local disk D around p. This disk is tessellated into slices by the motorcycle traces. If all slices are convex then we ignore this event. In particular, if another motorcycle reached p earlier, the disk is tessellated into convex slices. In the opposite case, there is exactly one non-convex slice. We may assume that (i) the traces of $m_1, \ldots, m_k$ appear in counter-clockwise order at

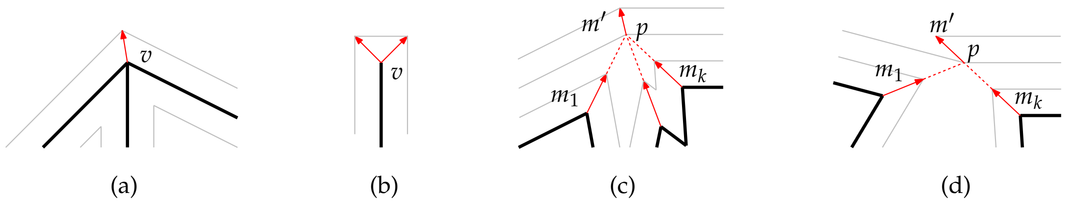


Figure 31: Four different situations for which motorcycles are launched. (a) each reflex wavefront vertex at time zero gives rise to one motorcycle. (b) a terminal vertex gives rise to two motorcycles. (c,d) if two or more motorcycles crash at p such that the traces tessellate a local disc into non-convex slices then we launch a new motorcycle.

p and (ii) the non-convex slice is bounded by the traces of m_1 and m_k. We distinguish the following two cases:

1. The left arm of m_1 and the right arm of m_k span a reflex angle (at the side which has not yet been swept by the wavefront at this time). Then we launch a new motorcycle m', which starts at p and whose arms are given by the left arm of m_1 and the right arm of m_k, see Figure 31 (c). This case is necessary in order to cover reflex straight-skeleton arcs that emanate from a vertex event.
2. The left arm of m_1 and the right arm of m_k span a convex angle. Then we shoot a new motorcycle m' which continues the movement of m_k. That is, m' starts at p and has the same pair of arms as m_k, see Figure 31 (d).

   Letting the motorcycle m' continue the movement of m_k appears to be somewhat arbitrary at the first sight. However, by this definition, we obtain the nice property that the arms of a motorcycle always span a reflex angle at the propagation side. This property, however, turns out to be useful in the proof of Theorem 2.26.

   Also note that Lemma 2.25 becomes trivial for the case illustrated in Figure 31 (d). Indeed, Lemma 2.25 would be also trivial in the case that is illustrated in Figure 31 (c) if we would launch an additional motorcycle that continues the movement of m_k. However, it is unclear whether Lemma 2.24 would remain true.

We note that we do not have to simulate the wavefront in order to apply the above case distinction: it suffices to know the propagation direction of the arms of the motorcycles. Furthermore, it could happen that an arm of a motorcycle already vanished in the wavefront propagation process while the motorcycle still moves. The terminology of *arms* turns out to be advantageous in the subsequent analysis.

In both cases of the previous case distinction, we call $m_1, \ldots, m_k$ the ancestors of m'. In particular, we call m_k the right-most ancestor and m_1 the left-most ancestor. Note that m_1 resp. m_k can again have ancestors. We define the right-most ancestor chain of m' as the union of the trace of m', the trace of m_k, the trace of the right-most ancestor of m_k and so on. The left-most ancestor chain is defined likewise.

Definition 2.23 (motorcycle graph induced by a planar straight-line graph)**.** Let G denote a planar straight-line graph. We consider the edges of G as walls and consider the set of motorcycles as elaborated above. The *motorcycle graph* $\mathcal{M}(G)$ *induced by* G is defined as the arrangement of the resulting motorcycle traces.

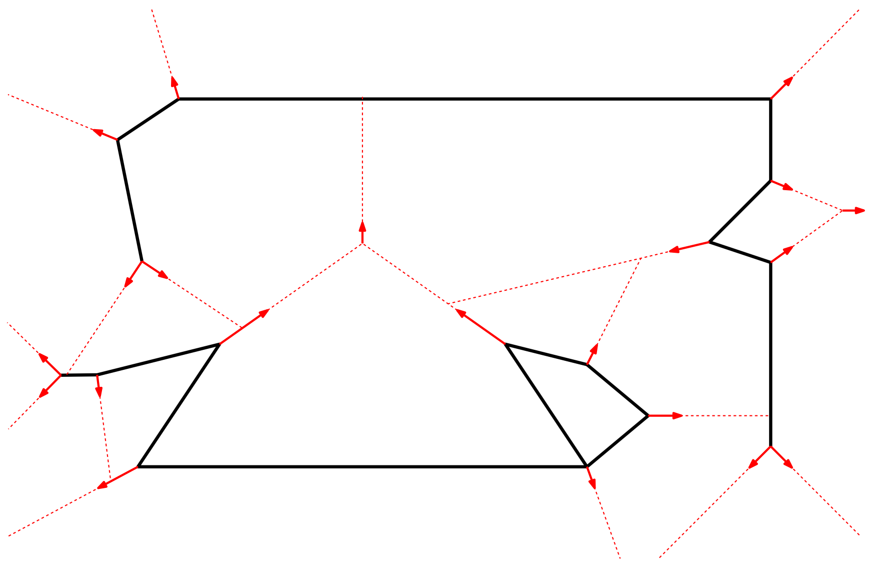


Figure 32: The motorcycle graph $\mathcal{M}(G)$, depicted in red, which is induced by the planar straight-line graph G (bold).

In Figure 32 we have illustrated the motorcycle graph $\mathcal{M}(G)$ for a planar straight-line graph G. Figure 3 on page 9 illustrates the corresponding straight skeleton $\mathcal{S}(G)$.

2.4.2 Geometric properties of the generalized motorcycle graph

Lemma 2.24. *The motorcycle graph $\mathcal{M}(G)$ of a planar straight-line graph G contains $O(n)$ motorcycle traces.*

Proof. Let $r \in O(n)$ denote the number of reflex wavefront vertices in $\mathcal{W}(G, 0)$. Each motorcycle that does not start from a vertex of G starts from the crash point of at least two other motorcycles. Hence, we obtain that $\mathcal{M}(G)$ comprises at most $2r - 1$ motorcycles traces. $\square$

Lemma 2.25 ([HH11c]). *Let p denote a point in the relative interior of $\mathcal{M}(G)$. A local disk around p is tessellated into convex slices by the traces of $\mathcal{M}(G)$.*

Proof. If p is in the relative interior of a trace, the lemma is trivially true. Let us assume that p is the endpoint of $k \geq 2$ motorcycle traces. We denote the motorcycles that crashed at p by $m_1, \ldots, m_k$. Consider a local disk D around p. If D is tessellated into convex slices by the traces of $m_1, \ldots, m_k$ then our assertion holds again. Assume that there is a non-convex slice. We may assume that the motorcycles are numbered in a cyclic order such that (i) their corresponding traces appear counter-clockwise around p and (ii) the trace of m_1 and m_k bound the reflex slice.

If the left arm of m_1 and the right arm of m_k span a convex angle then there is a motorcycle m which continues the movement of m_k and the assertion holds again. So, assume that the

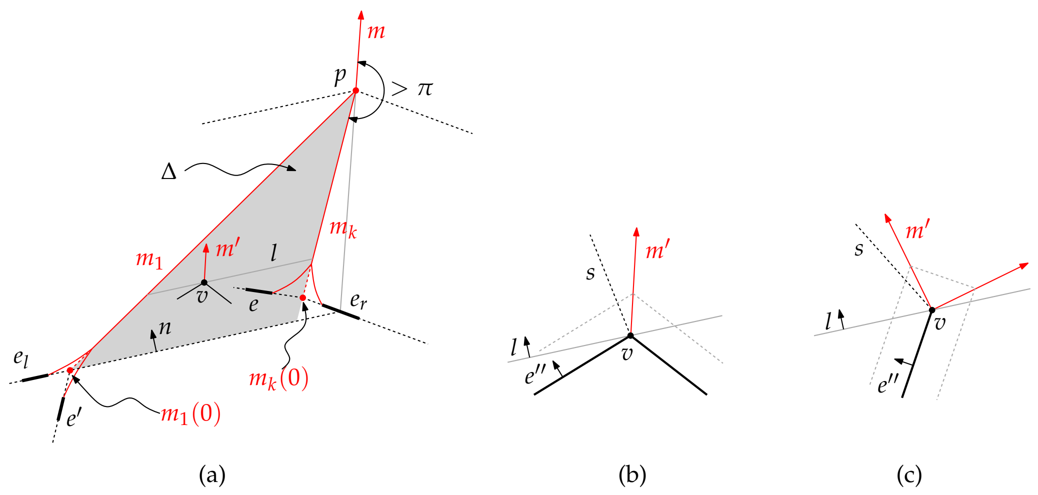


Figure 33: Convex tessellation at simultaneous crashes. (a) if m and m_k span a reflex angle then the launch of m is impossible. (b–c) a close-up of the vertex v, where v is (b) a non-terminal vertex and (c) a terminal vertex.

left arm of m_1 and the right arm of m_k span a reflex angle. Hence, there is a motorcycle m which started from p and shares its left arm with m_1 and its right arm with m_k. We have to prove that (i) the traces of m_1 and m span a convex angle and (ii) the traces of m_k and m span a convex angle. Without loss of generality, we assume that m and m_k span an angle greater than π. (The other case is symmetric.) The basic idea is to prove that under this assumption, the existence of m is contradicted because (i) m_1 or m_k crashes before reaching p or (ii) m_1 and m_k reach p after another motorcycle was at p before.

We denote by e_l and e_r the left and right arm of m, respectively. We note that m shares its right arm with m_k and its left arm with m_1. Further, we denote by e' the right arm of m_1 and by e the left arm of m_k, see Figure 33 (a). Neither m_1 nor m_k need to start at time zero. We denote by R the right-most ancestor chain of m_1, which contains the trace of m_1 and ends at an endpoint of $e'(0)$. (Recall that we denote by $e(t)$ the straight-line segment occupied by e at time t.) Likewise, we denote by L the left-most ancestor chain of motorcycle traces of m_k.

First we show that $e(0)$ is above $\overline{e_l(0)}$, i. e., $e(0)$ is on the propagation side of $\overline{e_l(0)}$: The point $m_1(0)$ must lie on $\overline{e_l(0)}$. (If m_1 does not start at time zero, we interpret $m_1(0)$ as the extrapolation of its movement before its actual starting time.) Likewise, $m_k(0)$ must lie on $\overline{e_r(0)}$. By assumption, $m_k(0)$ lies left to the speed ray of m and right to the speed ray of m_1. Hence, the arms of m_k span a smaller angle (on the propagation side of the motorcycle) than the one of m. It follows that $e(0)$ lies above $\overline{e_l(0)}$.

We denote by Δ the triangle enclosed by $\overline{e_l(0)}$ and the supporting lines of the traces of m_1 and m_k. Among all vertices of G within Δ, we denote by v a vertex that has maximum orthogonal distance to $\overline{e_l(0)}$. Note that at least the endpoints of $e(0)$ are within Δ. The one endpoint that is not incident to L has the greater orthogonal distance of both. We denote by n the propagation vector of e_l and by l the parallel line of e_l through v, see Figure 33 (a). The vector n has unit length and is orthogonal to e_l. If there are multiple vertices of G that lie on l then we may assume that v is the left-most on l (that is, the closest to the trace of m_1).

Consider l to be a sweep line moving parallel according to the speed vector n. We show that there is always a motorcycle m' that emanates from v and which is at any time in front of the sweep line l (or just coinciding). In order to see that we distinguish two cases:

- The vertex v has two or more incident edges. Since no incident edge of v lies above l we have a motorcycle m' emanating from v, see Figure 33 (b). We denote by e'' the left arm of m'. Note that $e''(0)$ is not collinear with l; otherwise there would be another vertex of G on l which is left to v.

  We denote by s the bisector of l and e'', which consists of those points that are reached by the propagating supporting line $\overline{e''(t)}$ and the sweep line l at the same time. Any point right to s is at first reached by $\overline{e''(t)}$. Since the motorcycle m' is at any time right to s we see that m' is always in front of the moving line l.
- The vertex v is a terminal vertex. There are two motorcycles emanating from v. We denote by m' the one whose speed vector has the greater inner product with n, see Figure 33 (c). We denote by e'' the arm of m' which is parallel to the incident edge of v. (W.l.o.g. we may assume that this is the left arm. The other case is symmetric.)

  As in the first case, we denote by s the bisector of the moving line l an e''. Since m' is right to s (or just overlapping) we see that m' is never behind the moving line l.

To sum up, there is always a motorcycle m' that never falls behind the sweep line l. Next, we note that l intersects R and L in their relative interiors. We conclude the proof with the following case distinction:

1. Assume that m' reaches R or L. We first see that m_1 and m_k always stay strictly behind the sweep line l: This is easy to see for m_1 since $m_1(0)$ starts behind the sweep line l and e_l propagates with the same speed as l. (Note that $m_1(t)$ and $\overline{e_l(t)}$ are coinciding.) We assume for a moment that m_k would overtake l at some time. Since m_k started behind the sweep line l, it follows that m_k reaches p before the sweep line does. However, this is a contradiction, because m_k cannot reach p before m_1 does.

   Since m_1 and m_k stay behind the sweep line l and since m' is always in front of (or just coinciding with) l, it follows that (i) m_1 crashes into m', or (ii) m_k crashes into m', or (iii) m_1 and m_k reach p but m' was at p before. In any case, the launch of m from p is avoided.
2. Assume that m' does not reach R or L. Hence, m' crashed within Δ. The motorcycle m' did not crash in a wall. (Otherwise, there would have been a vertex of G within Δ that has a greater orthogonal distance to $\overline{e_l(0)}$.) Hence, m' crashed into the trace of a motorcycle m'', which must have started below l, because no motorcycle could have come through R or L. Hence m'' is faster than m' w.r.t. direction n. That is, m'' is always in front of the sweep line l after m' crashed into m''. We consider m'' as the new m' and apply again our case analysis. Since there is only a finite number of motorcycles, we eventually end up in Case 1 and obtain a contradiction.

□

Theorem 2.26 ([HH11c]). *The reflex arcs of $\mathcal{S}(G)$ are covered by $\mathcal{M}(G)$.*

Proof. The proof consists of two steps. In the first step we state and prove an essential claim, which basically states that the tilted motorcycle traces of $\mathcal{M}(G)$ are above the terrain $\mathcal{T}(G)$. This proposition is applied later on in a proof by contradiction for the actual theorem.

(1) Let m be a motorcycle and $p \in \mathbb{R}^2$ a point on the trace of m. If all valleys of $\mathcal{T}(G)$ are covered by tilted motorcycle traces up to the height of $\hat{m}$ at p then the height of $\hat{m}$ is greater or equal to the height of $\mathcal{T}(G)$ at p. Equality is attained if and only if there is a valley of $\mathcal{T}(G)$ that corresponds to $\hat{m}$ and which exists until p.

We denote by e the right arm of m and we denote by $m_1, \ldots, m_k$ the motorcycles of the right-most ancestor chain of m such that $m_1(0)$ is incident to $e(0)$ and m_k equals m, see Figure 34 for $k = 2$. Furthermore, we denote by p_i the endpoint of the trace of m_i. We arrive at the following observations:

- The motorcycles $m_1, \ldots, m_k$ share the same right arm e. As a consequence, the tilted traces $\hat{m}_1, \ldots, \hat{m}_k$ lie on a plane, namely the supporting plane of the terrain face $\hat{f}(e)$ of the wavefront edge e.
- The angle between $e(0)$ and the trace of m_1 and between the traces of m_i and m_{i+1}, with $1 \leq i \leq k-1$, are at most 180° by Lemma 2.25. However, for any $1 \leq i \leq k$ the motorcycle m_i spans with its right arm e an angle of at least 90° by definition of the motorcycles.

Let us denote by $\mathcal{T}$ the polygonal chain that is defined by the intersection of $\mathcal{T}(G)$ with a vertical curtain that is put on the union of the motorcycle traces of $m_1, \ldots, m_k$. Claim (1) states that the height of $\hat{m}_k$ is greater than or equal to the height of $\mathcal{T}$ at p. In order to prove this claim it suffices to show that the slope of $\mathcal{T}$ is at any interior point of a trace of m_i at most the slope of the tilted trace $\hat{m}_i$. The following proof is an induction-type proof. We show (i) that $\mathcal{T}$ is convex within the interior of motorcycle traces, and (ii) that the slope constraint is maintained when migrating from one trace to the next.

(i) Due to the existence of m_1 there is a reflex wavefront vertex that started from $m_1(0)$. The wavefront vertex traces a reflex straight skeleton arc and has the same speed as m_1 by our definition of the motorcycles. Hence $\mathcal{T}$ and $\hat{m}_1$ start with the same slope. Let us consider the part T of $\mathcal{T}$ that lies above the trace of m_1. If there is a reflex vertex in T then we consider the one whose projection q on the plane is closest to $m_1(0)$. Obviously there would be a valley of $\mathcal{T}(G)$ at q. By assumption there would also be a motorcycle trace covering this valley at q. Since $\hat{m}_1$ is above (or just at the same height as) this trace, it follows that m_1 would have crashed at q. This is a contradiction. Hence the slope of $\mathcal{T}$ is non-increasing above the trace of m_1. The same arguments suffice to show that $\mathcal{T}$ is non-increasing above the trace of m_i if $\mathcal{T}$ was below $\hat{m}_i$ at p_{i-1}.

(ii) Next, we show that if the slope of $\mathcal{T}$ is less than or equal to the slope of $\hat{m}_i$ before p_i, then the slope of $\mathcal{T}$ is less than or equal to the slope of $\hat{m}_{i+1}$ after p_i. We denote by e' the wavefront edge which defines $\mathcal{T}$ after p_i.

The slope of $\mathcal{T}$ before p_i can be expressed by the angle between the trace of m_i and e'. That is, the slope increases monotonically as the corresponding angle increases. Likewise, we can express the slope of $\mathcal{T}$ after p_i by the angle between the trace of m_{i+1} and e'. Moreover, we can express the slopes of $\hat{m}_i$ resp. $\hat{m}_{i+1}$ by the angle between m_i and e resp. m_{i+1} and e. Hence, we can rephrase our assertion: If the angle between m_i

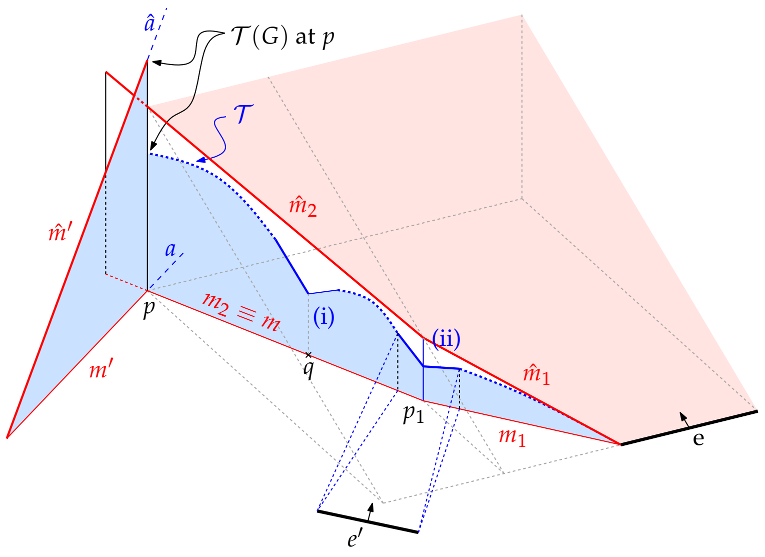


Figure 34: Proof of Theorem 2.26. The terrain $\mathcal{T}(G)$ is always below the tilted motorcycle traces $\hat{m}_k$. The proof shows that the slope of the terrain (blue) is at any point at most the slope of the tilted motorcycle traces (red) since the situations (i) and (ii) (shown in solid red and described in the proof) do not occur.

and e' is smaller than the angle between m_i and e then the angle between m_{i+1} and e' is smaller than the angle between m_{i+1} and e.

Let us consider Figure 35. We denote by l the bisector between e and e' on the left side and by r the bisector on the right side. Hence, we have to prove that m_{i+1} lies right to l and left to r. Our premise states that the angle between e' and m_i is less than or equal to the angle between e and m_i. We denote by e_l the left arm of m_i and by t_i the time when e reaches p_i. Assume that we rotate e' counter-clockwise around p_i until e' is parallel with $\overline{e(t_i)}$. Then l is falling onto $\overline{e(t_i)}$ and r is perpendicular to $\overline{e(t_i)}$. Vice versa, assume that we rotate e' clock-wise around p_i until e' is parallel with e_l. Then l is on the supporting line of m_i and r is on the bisector of m_i and $\overline{e(t_i)}$. We shaded the valid domains for l and r in Figure 35 accordingly. Since the angle between m_i and m_{i+1} is convex, m_{i+1} is right to the domain of l. Since m_{i+1} and e enclose an angle of at least 90°, m_{i+1} is left to the domain of r. Summarizing, for every position of e', which conforms to our initial assumption, m_{i+1} encloses a smaller angle with e' than with e.

Combining arguments (i) and (ii) yields an induction-type proof for Claim (1). Basically, we showed that the distance between $\mathcal{T}$ and the tilted motorcycle traces is (not necessarily strictly) monotonically increasing. If $\mathcal{T}$ and the tilted motorcycle traces are overlapping until p then equality for the height of $\hat{m}$ and $\mathcal{T}(G)$ at p is attained. If $\mathcal{T}$ leaves the tilted motorcycle traces at some point then $\mathcal{T}(G)$ is strictly below $\hat{m}$ at p.

We now return our attention to Figure 34 and use Claim (1) in a proof by contradiction of Theorem 2.26. Assume that there is a reflex arc a in $\mathcal{S}(G)$ that is only partially covered by a motorcycle trace m'. Hence, m' crashed into a motorcycle m. We denote by p the crashing

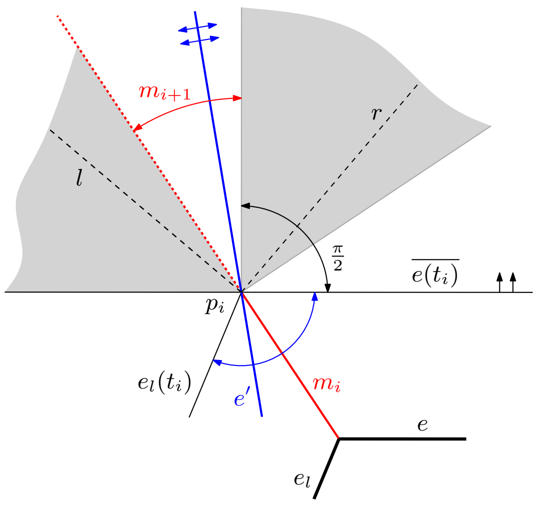


Figure 35: Proof of Theorem 2.26, case (ii). The shaded areas depict the valid domains for l and r and m_{i+1} is at any time right to l and left to r.

point of m'. Without loss of generality we assume that the height of $\hat{m}'$ at p is lowest. (By this assumption we can assume that a is at least partially covered. If a would not be covered at all, then a was not incident to G and at least one of its reflex ancestor arcs was not covered completely.) Hence, all valleys of $\mathcal{T}(G)$ are covered by motorcycle traces up to the height of $\hat{m}'$ at p. By Claim (1) we know that $\mathcal{T}(G)$ is below $\hat{m}$ at p. On the other hand, we know that $\mathcal{T}(G)$ has the same height as $\hat{m}'$ at p. (See the left side of Figure 34.) This contradiction finally concludes the proof. □

Theorem 2.26 extends Theorem 2.11 by Cheng and Vigneron [CV07]. In their proof they assumed that no two motorcycles crashed into each other. The idea of their proof is incorporated into the proof of Case (i) of Claim (1). Claim (1) is a useful tool for its own, so we can cast it to the following corollary.

Corollary 2.27. *Let m be a motorcycle of $\mathcal{M}(G)$ and $p \in \mathbb{R}^2$ a point on the trace of m. The height of $\hat{m}$ is greater or equal to the height of $\mathcal{T}(G)$ at p. Equality is attained if and only if the valley of $\mathcal{T}(G)$, which corresponds to m, exists until p.*

2.4.3 The lower envelope based on the generalized motorcycle graph

Let us revisit the discussion concerning the alternative characterization of the straight skeleton $\mathcal{S}(G)$, by using a lower-envelope representation of the terrain $\mathcal{T}(G)$ from Section 2.1. Recall that Theorem 2.7 by Eppstein and Erickson [EE99] provides a lower-envelope representation for $\mathcal{T}(G)$, but their slabs depend on the length of the reflex arcs of the straight skeleton. On the other hand, Theorem 2.14 by Cheng and Vigneron [CV07] provides a lower-envelope representation of $\mathcal{T}(P)$ that does not depend on the straight skeleton, but their representation can only be applied to simple non-degenerate polygons P with holes.

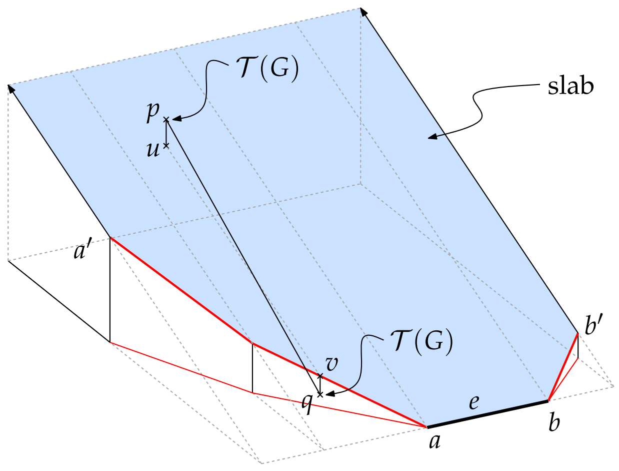


Figure 36: The shaded area illustrates the slab defined by the wavefront edge e. The slab lies on the supporting plane of $\hat{f}(e)$ and is bounded from below by the tilted motorcycle traces that have e as an arm.

Summarizing, no lower-envelope representation is known for the terrain $\mathcal{T}(G)$ of planar straight-line graphs G, which does not depend on the straight skeleton.

In the following we extend the slab construction, which is based on the motorcycle graph, to the generalized motorcycle graph, see Figure 36.

Definition 2.28 (lower envelope)**.** Let e denote a wavefront edge of a planar straight-line graph G and let a and b denote the endpoints of $e(0)$. If there is a motorcycle that starts at a and has the right arm e then we consider the whole chain of tilted motorcycles traces, starting at a and ending at a', whose right arms are e. If there is no such motorcycle then we define $a' := a$. Likewise we consider the chain of tilted motorcycle traces starting at b and ending at b' whose left arms are e. We define a slab for each e that is contained in the supporting plane of the terrain face $\hat{f}(e)$ and which is bounded from below by e and the motorcycle traces mentioned above. At the endpoints a' and b' the slab is bounded by rays which are perpendicular to $\overline{e(0)}$. We denote by $L(G)$ the *lower envelope* of the these slabs.

Theorem 2.29 ([HH11c])**.** *The lower envelope $L(G)$ is identical to $\mathcal{T}(G)$.*

Proof. Each face $\hat{f}(e)$ of $\mathcal{T}(G)$ is contained in the corresponding slab of e. It follows that no point of $L(G)$ is above $\mathcal{T}(G)$ and it remains to show that no point of $\mathcal{T}(G)$ is above $L(G)$. Assume, to the contrary, that a point $p \in \mathcal{T}(G)$ is above a slab of an edge e, see Figure 36. We project p down to the slab and denote the projection point by u. Then we project u down along the steepest descent of the slab until we hit e or one of the tilted motorcycle traces and obtain the point v. If v is on a tilted trace then Corollary 2.27 implies that we can project v down to $\mathcal{T}(G)$ and obtain a point q. If v is on $e(0)$, we set $q := v$. Since the line between u and v has slope 1, the slope of $\overline{pq}$ is greater than 1. This is a contradiction to Lemma 2.1. □

As mentioned above, Theorem 2.29 extends Theorem 2.14 by Cheng and Vigneron [CV07] to planar straight-line graphs G. However, Theorem 2.29 also extends Theorem 2.7 by Eppstein and Erickson [EE99] since $L(G)$ is based on $\mathcal{M}(G)$ and does not depend on the length of the reflex arcs of $\mathcal{S}(G)$. This difference, however, is essential when we attempt to compute $\mathcal{S}(G)$ via a lower-envelope computation, see below. Besides, Theorem 2.29 provides an alternative, non-procedural way to define $\mathcal{S}(G)$ as the lower envelope of partially linear functions, as discussed in Section 2.1.

COMPUTING $\mathcal{S}(G)$ USING GRAPHICS HARDWARE Theorem 2.29 admits a simple method to render $\mathcal{T}(G)$ without the knowledge of $\mathcal{S}(G)$. One first computes the motorcycle graph $\mathcal{M}(G)$ by a conventional algorithm (on the CPU) and then constructs the slabs as illustrated in Figure 36. We paint each slab with a different color. By rendering the set of slabs, while looking at them from below, one obtains an image which shows $L(G)$. Hence, by employing techniques described by Hoff et al. [HCK+99], one can compute the straight skeleton of planar straight line graphs using graphics hardware. The idea is that each face $\hat{f}(e)$ corresponds to a set of pixels of the same color.

2.5 THE GENERAL WAVEFRONT-TYPE ALGORITHM

2.5.1 Details of the general algorithm

Let us recall the basic building blocks of the simple algorithm of Section 2.3. Firstly, we require that the motorcycle graph covers the reflex arcs of the straight skeleton, which is guaranteed by Theorem 2.26. Secondly, we require that the overlay of the motorcycle graph and the input graph results in a convex tessellation of the plane.

Definition 2.30 (extended wavefront)**.** Let G denote a planar straight-line graph. We define by $\mathcal{M}(G,t)$ those parts of $\mathcal{M}(G)$ which have not been swept by $\mathcal{W}(G,t')$ for $t' < t$. The *extended wavefront* $\mathcal{W}^*(G,t)$ is defined as the overlay of $\mathcal{W}(G,t)$ and $\mathcal{M}(G,t)$ by splitting the edges of $\mathcal{W}(G,t)$ at the intersection points accordingly.

Lemma 2.31. *For any $t \geq 0$ the set $\mathbb{R}^2 \setminus \bigcup_{t' \in [0,t]} \mathcal{W}^*(G,t')$ consists of open convex faces only.*

Proof. The assertion follows directly from Lemma 2.25 and the fact that each reflex angle at a reflex wavefront vertex is split by a motorcycle trace. □

We adopt the terms *resting Steiner vertex* and *moving Steiner vertex* from Section 2.3. In addition to these terms we refer to a vertex of $\mathcal{W}^*(G,t)$ that marks the crash of two or more motorcycles as *multi Steiner vertex*, see Figure 37.

The basic algorithm from Section 2.3 remains the same. Lemma 2.31 guarantees that any topological change in the extended wavefront is indicated by the collapse of an edge of the extended wavefront to zero length. We start with the initial extended wavefront $\mathcal{W}^*(G,0)$. For each edge e of $\mathcal{W}^*(G,0)$ we insert an event into a priority queue Q if the collapsing time of e is positive. The events are prioritized by their time of occurrence. After the initialization we fetch from Q one event after the other and process it. That is, for each event we apply local modifications of the extended wavefront and maintain the priority queue Q accordingly. (Note that if we use a maximizing heap as the underlying data structure for Q we may also

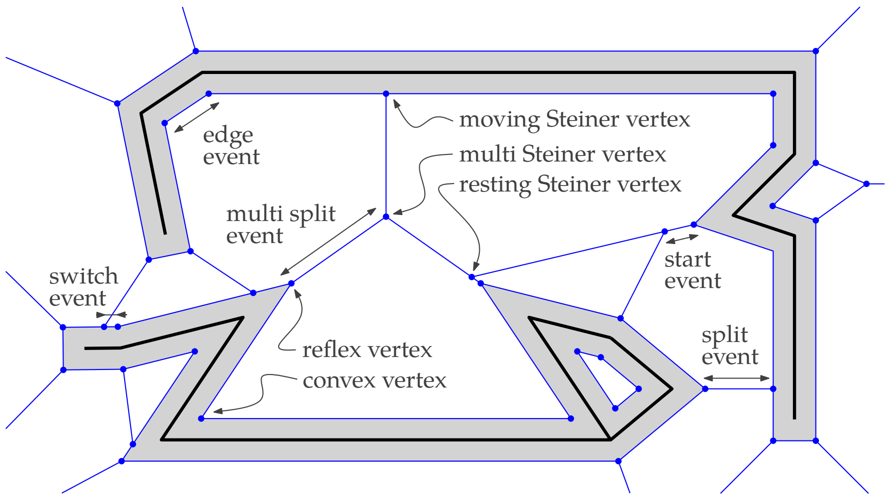


Figure 37: A planar straight-line graph (bold) and the extended wavefront (dashed) after some time. The area already swept by the wavefront is shaded.

remove any element in $O(\log n)$ time if we already have a pointer to the element.) In the following we are discussing the different types of events.

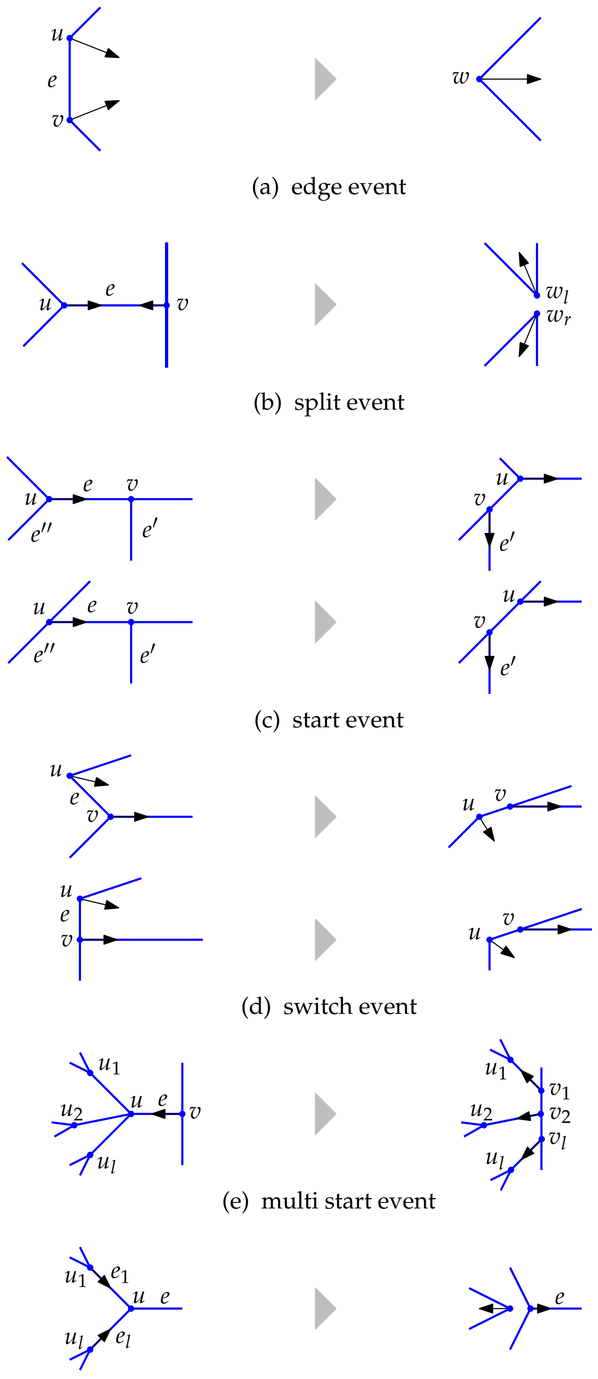
- **(Classical) edge event**: The edge e between two convex vertices u and v collapsed, see Figure 38 (a). We remove e from the graph and merge the vertices u and v to a single convex vertex w. The two edges incident to w determine the new velocity of w. Consequently, we need to recompute the collapsing times of the two incident edges of w and adapt the entries in Q.

  Finally, we add the two convex straight-skeleton arcs that were traced out by u and v to the straight-skeleton graph. Altering $O(1)$ entries in Q costs us $O(\log n)$ time.

- **(Classical) split event**: The edge e between a reflex vertex u and a moving Steiner vertex v collapsed, see Figure 38 (b). Note that v is the Steiner vertex that corresponds to the crash of the motorcycle whose trace covers the reflex arc that is traced out by u.

  We denote by u_l resp. v_l the two vertices which are incident to u resp. v and left to the trajectory of u. The vertices u_r, v_r are denoted likewise. We remove the edge e and merge the vertices u_l and v_l to a new convex vertex w_l. Its velocity is determined by the two incident edges. As a special case we check whether the two incident edges of w_l are parallel. This means that the entire convex face of $\mathcal{W}^*(G,t)$, to which w_l belongs, collapsed at the time when the split event occurred. The right side, with the vertices u_r and v_r, is processed likewise.

  Since u crashed at this split event, we need to add the reflex arc that is traced out by u, to our straight-skeleton structure. Altering $O(1)$ entries in Q costs us again $O(\log n)$ time.



(a) edge event

(b) split event

(c) start event

(d) switch event

(e) multi start event

(f) multi split event

Figure 38: Different types of events that occur during the propagation of the extended wavefront.

- **Start event**: The edge e that connects a resting Steiner vertex v and a reflex vertex or a moving Steiner vertex u collapsed, see Figure 38 (c). The start event occurs since the vertex u moves along the edge e, which lies on the motorcycle trace that corresponds to the vertex u. The vertex v remained on this motorcycle trace and now becomes a moving Steiner vertex by sliding along one of the two other incident edges of u, say e''. While the velocity of u remains the same, we have to assign a new velocity to v and recompute the collapsing times of the incident edges. We consider the situation where e'' and e' overlap as a special case. In this case the entire convex part collapsed to zero area.

  A start event does not immediately contribute to the straight-skeleton structure. However, altering the $O(1)$ entries in Q costs us $O(\log n)$ time.

- **Switch event**: The edge e between a convex vertex u and a reflex vertex or a moving Steiner vertex v collapsed, see Figure 38 (d). That means that the vertex u migrates from one convex face of the extended wavefront to a neighboring one, by jumping over the vertex v. As a consequence, the vertex v now splits the opposite edge incident to u. While the vertex u maintains its velocity, we need to recompute the velocity for the vertex v.

  Similar to start events, a switch event does not directly contribute to the straight skeleton. Again it requires $O(\log n)$ time to update the corresponding entries in the priority queue Q.

- **Multi start event**: The edge e that connects a moving Steiner vertex v and a multi Steiner vertex u collapsed, see Figure 38 (e). That means that the multi split event that corresponds to the multi Steiner vertex u will not happen, because the wavefront swept over the vertex u before.

  Let us denote by $u_1, \ldots, u_l$ the other vertices that are incident to u. Assume that the edges $e_1, \ldots, e_l$, which connect u with $u_1, \ldots, u_l$, respectively, appear in counter-clockwise order at u such that the edge e is between e_1 and e_l in the mentioned order. We remove the vertices u and v and introduce new moving Steiner vertices $v_1, \ldots, v_l$ such that v_i moves towards u_i, with $1 \leq i \leq l$. The new vertices $v_1, \ldots, v_l$ are aligned on the supporting line of the two other edges that are incident to v. We add new edges uv_i for $1 \leq i \leq l$ and $v_i v_{i+1}$ for $1 \leq i \leq l-1$. Besides u, we have two additional vertices that were adjacent to v. We connect these two vertices to v_1 and v_l, respectively.

  The multi start event does not directly contribute to the straight-skeleton structure. Computing the collapsing times of the new edges and adapting the collapsing times of old edges requires $O(l \log n)$ time.

- **Multi split event**: The edges $e_1, \ldots, e_l$ that connect the reflex vertices $v_1, \ldots, v_l$ with the multi Steiner vertex u collapsed at the same time, see Figure 38 (f). We assume that $e_1, \ldots, e_l$ appear in counter-clockwise order at u. The multi Steiner vertex u exists, since two or more motorcycles simultaneously crashed at this location. Note that it is possible that a new motorcycle was emanated from this location and hence there could be an additional edge e incident to u. This edge e lies on the motorcycle trace of the emanated motorcycle, see Figure 31. If this is the case we assume that e is between e_1 and e_l. Also note that in this case the angle from e_l to e_1 is reflex.

Each consecutive pair $u_i u_{i+1}$, with $1 \leq i \leq l-1$, gives rise to a new convex vertex similar to an ordinary split event. This new convex vertex is incident to two edges that were formerly the right arm of u_i and the left arm of u_{i+1}, respectively. Similar to the ordinary split event, we check whether the two incident edges are overlapping. In this case, the whole convex face of the extended wavefront collapsed as the multi split event happened.

If the angle from e_l to e_1 is also convex then there is no additional edge e incident to u and we proceed as above. Otherwise we have an additional edge e incident to u. If the right arm of u_l and the left arm of e_1 span a convex angle than the edge e and the old edge e_l are collinear, cf. Figure 31 (d). We remove the edge e and proceed as above. Otherwise, the right arm of u_l and the left arm of e_1 span a reflex angle. Then the edge e lies on their bisector and we introduce a new reflex vertex that drives on the edge e, cf. Figure 31 (c).

Each u_i, with $1 \leq i \leq l$, traced out a reflex straight-skeleton arc, which we add to our straight-skeleton structure. Computing the collapsing times of the new edges and altering the corresponding entries in Q takes $O(l \log n)$ time.

After the last event occurred, the extended wavefront has the shape of a polygon circumscribing the graph G. From each vertex v of this polygon that has a reflex angle on the outer side, there is an incident edge which reaches to infinity and which corresponds to a motorcycle that escaped. We add these infinite edges as reflex arcs to our straight skeleton structure and connect the infinite end nodes in a circular manner. This allows us to assume that the boundary of any straight-skeleton face $f(e)$, including the unbounded faces, connects one endpoint of e with the other endpoint of e. We also refer to the discussion on unbounded faces and infinite arcs in Section 1.2.2.

2.5.2 Runtime analysis

Let us assume that the motorcycle graph $\mathcal{M}(G)$ of G is already given. In order to create the extended wavefront $\mathcal{W}^*(G,0)$ at time zero we first create $\mathcal{W}(G,0)$: We start with an edge e and a vertex v incident to e. We walk around the corresponding connected component by setting e to the next counter-clockwise edge of e at v and by setting v to the opposite vertex of the updated e. We continue this procedure until we again arrive at our starting edge. At each step, we duplicate the current edge e and consider its duplication as the wavefront edge at one side of e. At any time when we wrap around at a terminal vertex, we add an additional wavefront edge. After we arrive again at the starting edge, we repeat the whole procedure for any edge of G that has not yet a duplicate on either side. This allows us to create $\mathcal{W}(G,0)$ in $O(n \log n)$ time for an n-vertex graph G. Next, we insert $\mathcal{M}(G)$ into $\mathcal{W}(G,0)$ in order to obtain $\mathcal{W}^*(G,0)$. Hence, every motorcycle that crashed into a wall — that is, into an edge e of G — splits the corresponding wavefront duplicate of the input edge e. We need to take into account that multiple motorcycles may crash into the same side of an input edge e. We sort the motorcycles along e and split the corresponding wavefront edge accordingly. To sum up, we can construct $\mathcal{W}^*(G,0)$ in $O(n \log n)$ time if $\mathcal{M}(G)$ is known.

The number of edge events is in $O(n)$ and the number of split and start events is in $O(r) \subset O(n)$, where r denotes the number of reflex wavefront vertices in the initial wavefront $\mathcal{W}(G, 0)$. However, as discussed in Section 2.3, the number k of switch events is in $O(nr) \subset O(n^2)$. Each multi start event and multi split event is handled in $O(l \log n)$ time, where the number l corresponds to the number of motorcycles that gave rise to this event. The sum of all values l among the multi start and multi split events is therefore in $O(r) \subset O(n)$.

Lemma 2.32. *Let G denote a planar straight-line graph G with n vertices, where r denotes the number of reflex wavefront vertices in the initial wavefront $\mathcal{W}(G, 0)$. If $\mathcal{M}(G)$ is known then our algorithms runs in $O((n + k) \log n)$ time and $O(n)$ space, where $k \in O(nr) \subset O(n^2)$ denotes the number of switch events occurred.*

As already mentioned in Section 2.3.3, the $O(n^2)$ bound for the number of switch events is tight. However, we want to recall that it appears to be very unlikely that more than $\Omega(n)$ switch events actually occur for practical applications and hence an $O(n \log n)$ runtime can be expected in real world. In Section 2.5.4 we present extensive experimental results on 13 500 datasets of different characteristics that demonstrate an $O(n \log n)$ runtime on virtually all of our datasets.

In order to compute the motorcycle graph $\mathcal{M}(G)$, we need an algorithm that supports the dynamic insertion of motorcycles during the simulation. Hence, we cannot apply the algorithm by Cheng and Vigneron, cf. Section 1.4.2.4, which needs to compute the $1/\sqrt{r}$-cutting a-priori. However, the algorithm by Eppstein and Erickson [EE99], cf. Section 1.4.2.3, employs dynamic data structures which are capable of adding new motorcycles during the simulation. Hence, in theory, we can compute the motorcycle graph $\mathcal{M}(G)$ in $O(r^{17/11+\epsilon})$ time and space. In Section 3.3 we will discuss a practical algorithm that runs in $O(r \log r)$ time on average, under the assumption that the motorcycles are distributed uniformly enough.

2.5.3 Details of the implementation Bone

We casted our algorithm into the implementation Bone. Bone is written in C++ and uses ordinary double-precision floating-point arithmetic according to IEEE-754. For standard data structures, such as stacks, queues, maps (red-black trees), etc., we use the Standard Template Library. The motorcycle graph $\mathcal{M}(G)$ is computed by our motorcycle graph code Moca, see Section 3.3. One central component of Bone is a kinetic straight-line graph that assigns to each vertex a speed ray (the position at time zero and a velocity). Furthermore, each non-isolated vertex has an index to an incident edge and each edge has, for both endpoints, references to the next clockwise and counter-clockwise edges. The kinetic graph models the extended wavefront and does not maintain planarity. It is in the responsibility of the event handling code to maintain the proper topology of the kinetic graph. However, this graph structure allows us to easily remove edges and reconnect vertices, which is frequently done in the event handling procedures.

For the actual implementation it turned out that is advantageous to introduce a new type of vertex, the so-called *multi convex vertex*. In a degenerate case, it could happen that a convex wavefront vertex and a moving Steiner vertex move along the same trajectory. For instance, consider a symmetric non-convex 4-gon as input, where the motorcycle within the 4-gon crashes exactly into the opposite convex vertex. In this case it is important, for practical reasons, to merge the convex vertex and the Steiner vertex into a multi convex vertex.

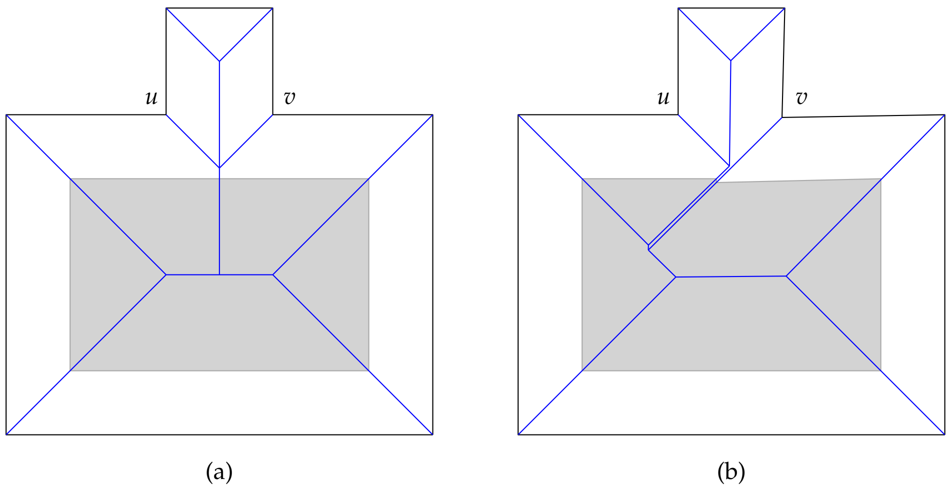


Figure 39: An arbitrarily small perturbation of a vertex v significantly changes the straight skeleton. Left: v is involved in a vertex event with the vertex u. Right: v has been slightly dislocated to the interior of the left polygon. Since v is slightly faster than u, the vertex event is avoided and the reflex arc incident to v is dramatically longer. The corresponding offset polygons are depicted in gray. Note that the offset polygon in the right subfigure contains a reflex vertex at the top.

For real-world implementations of geometric algorithms, it is advisable to implement some a posteriori checks that test whether the result fulfills some basic but necessary properties in order to be correct. BONE follows that strategy and provides a posteriori checks that test whether the boundary of the faces are connected.

DISCONTINUITY AND ϵ-BASED COMPARISONS We already mentioned the unpleasant property of the straight skeleton $\mathcal{S}(G)$ to be very sensitive for changes on the input G multiple times. (For instance, this fact made it necessary in introduce the non-degeneracy assumption by Cheng and Vigneron, cf. Section 1.2.4.) Eppstein and Erickson [EE99] presented a simple example, where a vertex event occurs and for which an arbitrary small perturbation at certain input vertices leads to a significantly different straight skeleton. We illustrate a very similar example in Figure 39. Note that dislocating the vertex v in the right subfigure introduces a proper reflex vertex in the gray offset polygon, while in the left subfigure the offset polygon has a rectangular shape with the top edge split into two collinear edges.

The fact that the straight skeleton $\mathcal{S}(G)$ does not continuously depend on the input graph G — say, in terms of the Hausdorff distance — has important practical implications. First of all, we cannot apply perturbation techniques in order to resolve special cases in various algorithms, as already mentioned by Eppstein and Erickson [EE99]. Secondly, we have to be cautious when we consider the limit of a sequence of straight skeletons. For instance, image we define the wavefront emanated by an isolated vertex by approximating the vertex by small squares or small segments. (Recall the discussion on the wavefront emanated from a isolated vertex in Section 1.2.2.) The limit of the straight skeletons does not necessarily result in the desired structure and may depend on the actual sequence of input graphs. Thirdly, the question arises how to determine whether two motorcycles or two reflex vertices meet each other at exactly the same time.

- If we use ordinary double-precision floating-point arithmetic then we need to apply ϵ-based comparisons. That is, we define a small positive constant ϵ and in order to test whether the numbers a and b are equal we tests whether $|a - b| \leq \epsilon$.
- In order to avoid well-known problems introduced by ϵ-based comparisons, one could apply so-called exact predicates. The decision whether two values a and b are equal is based on the expression trees for the values a and b and the concept of so-called root-bounds [Yap04]. That is, one can precisely decide whether $|a - b|$ is zero, by evaluating the term $|a - b|$ with finite but arbitrary precision. The root bounds tell how precisely the term needs to be evaluated.

Let us consider the application of roof construction, cf. Section 1.2.3. Assume that we implement a roof-construction software based on straight skeletons, which reads the wall footprint of a house from an input file. Since the straight skeleton is sensitive to small errors on the input we would require that the input file is capable of carrying exact coordinates. This basically means that we need to save expression trees of all coordinates in the input files. However, this seems to be infeasible for real-world applications:

- The file size would skyrocket and, moreover, the expression trees tend to grow in size as the input gets more and more complex, because the coordinates of new vertices tend to depend on the positions of previous vertices.
- Standard file formats, like the DXF format, do not support the embedding of expression trees for the coordinates in a standardized way.
- We would require that common CAD software uses exact arithmetic kernels for their own calculations (e. g., snapping to intersection points). In the end, we would require that the entire work-flow uses exact arithmetic.

However, if we assume that the coordinates in the input files are imprecise due to the limited precision or ordinary floating-point numbers, we are basically forced to resort to ϵ-based comparisons for real-world applications. Otherwise, it is virtually impossible to construct non-trivial input data, for which we intend that a vertex event happens. Assume we would like to construct a symmetric wall footprint as in Figure 39 (a), which would produce a roof, where two valleys meet in a common endpoint. However, due to numerical imprecisions, we instead obtain a roof that is based on Figure 39 (b).

Note that for Voronoi diagrams the problem of imprecise input is not as severe as for straight skeletons, because the error introduced to the Voronoi diagrams is, roughly speaking, of the same magnitude as the error embedded in the input values.

2.5.4 Experimental results and runtime statistics

In this subsection we present experimental results for our implementation Bone and the straight-skeleton implementation that is shipped with CGAL 3.8, see Section 1.4.3. As already mentioned, Bone is able to process planar straight-line graphs as input, but the implementation in CGAL can only process polygons with holes. This is the reason why we restrict the following runtime statistics to datasets that contain polygons with holes only.

Since Bone computes the straight skeleton inside and outside the input polygon we call the straight-skeleton routines of CGAL outside the polygon, inside the polygon and also inside the holes, if any exist.[2]

We ran our runtime tests on a about 13 500 datasets, comprising realistic and contrived data of different characteristics, including mechanical designs, printed circuit boards, font outlines, random polygons generated by RPG [AH96], space filling curves, and fractal datasets. The size of the datasets ranges from only a few dozen vertices up to several millions.

Our experiments were carried out on a Linux machine with a 64-bit kernel, running on an Intel Core i7 processor clocked at 3.33 GHz. Note that this processor provides several cores, but neither Bone nor the straight-skeleton implementation from CGAL is multi-threaded. We avoided a falsified system behavior — e. g., caused by the invocation of the out-of-memory killer of Linux, freeing of all file system buffers, etc. — by restricting the memory utilization of Bone resp. the CGAL code to 6 GiB, using the `ulimit` command. Furthermore, we restricted the runtime for a single dataset to 10 minutes. The time consumption in order to compute the straight skeleton was measured by considering the user-space time consumption retrieved by the C-function `getrusage`.

Figure 40 shows four plots with identical axes and in each plot a blue dot represents the runtime of a single dataset. Note that both axes use a logarithmic scale. The x-axis shows the size n of the dataset and the y-axis illustrates the runtime for a dataset in seconds. For illustrative reasons we divided the runtime for each dataset of size n by the factor $n \log n$. Hence, a horizontal line in the plot corresponds to a runtime complexity of $n \log n$ and a line with slope 1 corresponds to a runtime complexity of $n^2 \log n$.

In subfigure (a) we plotted the runtime statistics for Bone. As predicted, Bone exhibits an $n \log n$ runtime behavior on the vast majority of our datasets. All dots in the gray area correspond to an actual runtime of 10 to $30 \cdot n \log n$ µs, where n varies from 60 up to $2 \cdot 10^6$. The motorcycle graph $\mathcal{M}(G)$ is computed by Moca, see Section 3.3. This implementation exhibits an $O(n \log n)$ runtime for well distributed input. We extensively investigate the runtime of Moca in Section 3.3.3. Our theoretical runtime analysis in Section 2.5.2 at first focuses on the assumption that the motorcycle graph is already available. For this reason, we plotted the runtime consumption of Bone, without the time consumed for the computation of the motorcycle graph $\mathcal{M}(G)$, in subfigure (b). We observe that the few outliers in subfigure (a) were virtually all caused by the computation of the motorcycle graph. Hence, an actual runtime of $O(n \log n)$ holds for basically all datasets from our database.

In subfigure (c) we plotted the runtime of CGAL using an arithmetic kernel with exact predicates but inexact constructions. By default, the CORE [Cor] library is employed as the arithmetic backend. Using exact predicates should guarantee that the topology of the straight skeleton is correct, even if the locations of the nodes are not necessarily exact.[3] However, using exact constructions makes the runtime consumption to skyrocket. That is, datasets with a few hundred vertices lead to runtimes in the range of minutes. We see that the runtime consumption of CGAL is mostly in the gray area that represents runtimes between 0.17 to $1.7 \cdot n^2 \log n$ µs. In particular, CGAL exhibits at least a quadratic runtime consumption in practice. We also observe that the runtimes of different datasets of the

2 CGAL requires to specify a maximum offset distance for the straight skeleton outside the polygon. After we scaled the input to the unit square, we set this offset distance to 100.

3 We learned this in a personal communication with F. Cacciola.

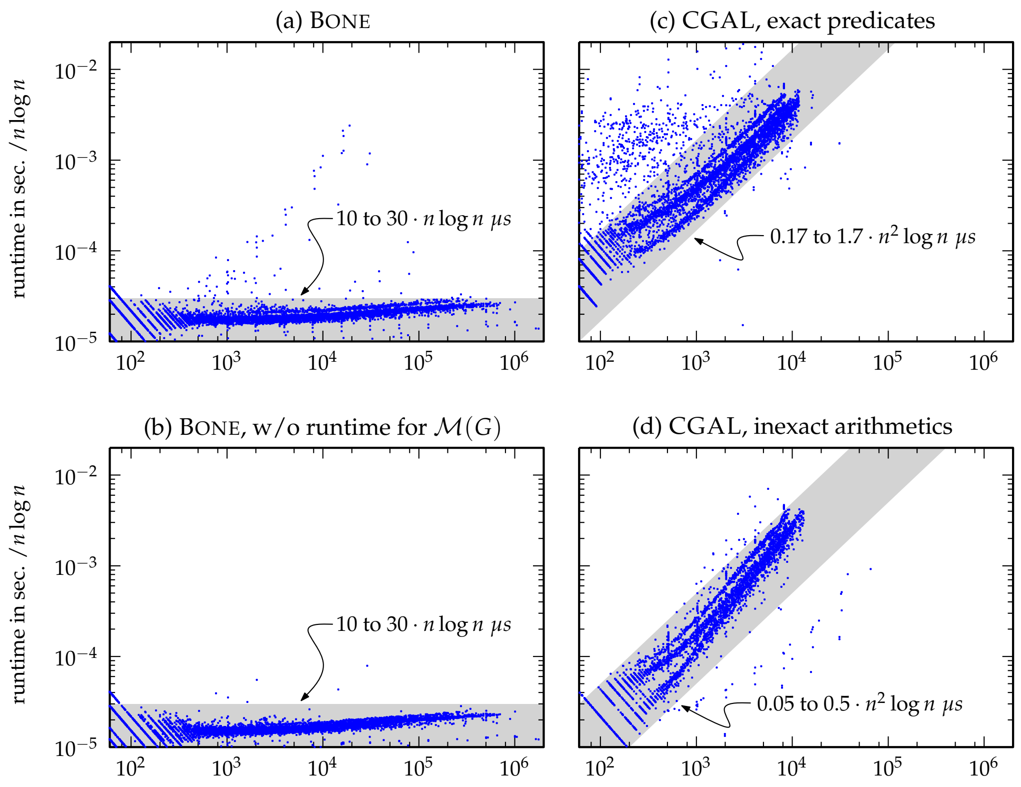


Figure 40: Every point shows the runtime for a single dataset. The x-axis denotes the number n of vertices of the input, ranging from 60 to $2 \cdot 10^6$. For illustrative reasons we divided the actual runtime in seconds by $n \log n$. Hence, a horizontal line corresponds to an $n \log n$ complexity. Points in the shaded areas correspond to runtimes as labeled.

same size vary within two decades. While many datasets with about a hundred vertices need roughly 20 ms, many other datasets lead to runtimes of roughly 20 seconds.

In subfigure (d) we plotted the runtime of CGAL using inexact arithmetics in order to (i) gain a better insight on the runtime penalty due to the exact predicates kernel and (ii) to obtain a better comparison to Bone. We first note that CGAL, with inexact arithmetics, still exhibits at least a quadratic runtime complexity. The vast majority of datasets is processed in 0.05 to $0.5 \cdot n^2 \log n$ µs. However, the big deviations for the runtimes among datasets of the same size vanished. We also observe that using inexact arithmetics increases the performance by a factor of about 3 to 100. However, we want to note that the straight-skeleton implementation of CGAL is not supposed to be used with an inexact predicates kernel.[4]

MEMORY USAGE Bone was able to handle input containing more than 10^6 vertices. However, CGAL could only handle datasets with up to about 10 000 vertices within the time and space constraints mentioned above. Interestingly, also CGAL with inexact arithmetics

4 We refer to a personal mail correspondence with F. Cacciola.

Size n	**Bone**		**CGAL**	
	MB	factor	MB	factor
256	1.44		3.77	
512	2.65	1.8x	13.4	3.5x
1 024	5.06	1.9x	51.1	3.8x
2 048	9.86	1.9x	201	3.9x
4 096	19.5	2.0x	792	3.9x
8 192	38.7	2.0x	3 197	4.0x
16 384	77.1	2.0x	12 600	3.9x

Table 3: The memory usage of Bone and CGAL on random datasets generated by RPG. Each row contains the peak heap size in megabytes and the factor by which the size changed w. r. t. the previous row.

was not able to compute datasets containing significantly more than about 10 000 vertices within the 6 GiB memory limit. We listed the memory footprint of Bone and CGAL on random datasets that were generated by RPG in Table 3. The memory usage was measured by the library libmemusage.so, which is shipped with glibc [GLI]. This library hooks into the `malloc` calls of a process and accounts the memory usage. In Table 3, we print the peak heap size in megabytes and the relative change to the previous row. Note that the actual memory footprint of a process is larger due to ordinary memory fragmentation in the heap and, of course, the program code, libraries and the stack require a few megabytes as well. For instance, the CGAL process required in total almost 20 GiB virtual memory in order to process the last dataset with 16 384 vertices.

We observe that Bone doubles its memory footprint if the input size of the dataset is doubled. In contrast, CGAL quadruples its memory footprint if the input size is doubled, which suggests a quadratic space complexity. As we learned in a personal mail correspondence with F. Cacciola, the author of the straight-skeleton code in CGAL, the algorithm computes potential split events for all pairs of reflex wavefront vertices and wavefront edges and inserts them into a priority queue. This explains the $O(n^2)$ memory footprint and the $O(n^2 \log n)$ runtime.

We tested the reliability of Bone on additional datasets, which do not necessarily form polygons with holes. Bone is also able to sample circular arcs from input files by straight-line segments. This allows us to test Bone on basically the same datasets as we did for Vroni [HH09a]. Furthermore, Bone is able to compute offset-curves based on the straight-skeleton, which turned out to be a versatile development tool in order to find errors in the straight skeleton by visual inspections. Furthermore, Bone can export the terrain $\mathcal{T}(G)$ into a standard file format that can be read by 3D modeling software, like Blender [Ble]. A wrong straight skeleton or significant numerical errors cause visible artifacts like non-planar terrain faces or very odd-looking terrains. Bone also contains many sanity checks for the validity of the wavefront during its propagation simulation and Bone is able to perform simple a posteriori checks on the resulting straight skeleton as mentioned in Section 2.5.3. It turns out that Bone performs well on our datasets in general. However, in order to boost Bone to industrial-strength, we need to implement some fine-tuning in the presence of wavefront-parts that simultaneously collapse in a self-parallel manner. However, the concept of the extended wavefront allows us to handle this issue on each convex faces of the extended wavefront separately, which appears to be a significant advantage.

2.6 SUMMARY

At the beginning of this chapter, the most promising approach towards a straight-skeleton implementation, in terms of implementability and real-world performance, appeared to be the triangulation-based algorithm by Aichholzer and Aurenhammer [AA98]. However, from a theoretical point of view, the best known worst-case time complexity is $O(n^3 \log n)$ for an n-vertex planar straight-line graph. In Section 2.2, we used this circumstance as an entry point to our investigations. After presenting new aspects regarding the number of flip events, we were able to show that Steiner triangulations can be used in order to completely avoid flip events. This insight motivated a new approach to a straight-skeleton algorithm for non-degenerate polygons which is based on motorcycle graphs in Section 2.3. However, in order to generalize this algorithm to arbitrary planar straight-line graphs we had to carefully extend the motorcycle graph such that essential properties, which are required for our algorithm, are preserved, see Section 2.4. This generalization of the motorcycle graph allowed us to extend the approach from Section 2.3 to arbitrary planar straight-line graphs in Section 2.5.

The resulting implementation Bone is able to handle planar straight-line graphs as input and exhibits a runtime of $O(n \log n)$ in practice. Compared to the current state-of-the-art straight-skeleton code, which is shipped with CGAL, this constitutes an improvement of a linear factor in time and space resp. a speed-up of one to two orders of magnitude for medium-sized datasets. Furthermore, CGAL is, in contrast to Bone, only able to handle polygons with holes as input. Our experimental results in Section 2.5.4 also revealed that Bone was able to process datasets containing more than a million vertices, while the implementation in CGAL was not able to handle datasets with significantly more than 10 000 vertices within the limits of 6 GiB memory.

The theoretical worst-case runtime complexity of Bone is $O(n^2 \log n)$, which is an improvement of a linear factor compared to the algorithm of Aichholzer and Aurenhammer [AA98]. However, the circumstances for which the wort-case occurs are easy to investigate for the algorithm underneath Bone, see Section 2.5. In fact, it appears to be very unlikely that the worst case actually happens for practical input, which has been confirmed by our experimental results in Section 2.5.4. Bone computed the straight skeleton for virtually all of our 13 500 datasets in 10 to $30 \cdot n \log n$ µs for datasets containing n vertices. This makes Bone the first straight-skeleton implementation that has been shown to run in $O(n \log n)$ in practice and which accepts planar straight-line graphs as input.

3 MOTORCYCLE GRAPHS

Motorcycle graphs are strongly related to straight skeletons for several reasons. First of all, the motorcycle graph extracts the essential sub problem of computing straight skeletons. Secondly, the motorcycle graph and the straight skeleton possess a strong geometric relation, which is expressed by Theorem 2.11 and Theorem 2.26. Thirdly, motorcycle graphs help us to devise algorithms and implementations for the computation of straight skeletons. At the moment, the fastest algorithm for the straight skeleton of non-degenerate polygons due to Cheng and Vigneron [CV07], see Section 1.4.2.4, and the fastest implementation, Bone, both employ the motorcycle graph. Hence, the investigation of motorcycle graphs is vital in order to obtain fast straight-skeleton algorithms and implementations and it is important to explore the motorcycle graph in order to get a better understanding of straight skeletons.

In this chapter, we develop an implementation for the computation of the generalized motorcycle graph that works fast in practice. We start with a stochastic consideration of the trace lengths in a motorcycle graph in Section 3.2. It turns out that if n motorcycles are distributed well in the unit square then the average trace length is about $O(1/\sqrt{n})$. This insight motivates a motorcycle graph implementation that is based on geometric hashing in Section 3.3. Runtime experiments show that our implementation Moca exhibits an $O(n \log n)$ runtime for most of the 22 000 datasets tested. In Section 3.4, we further investigate the geometric relation between the motorcycle graph and the straight skeleton. The results obtained in this section finally lead to a proof for the P-completeness of the straight skeleton of planar straight-line graphs and polygons with holes that is based on the P-completeness of motorcycle graphs due to Eppstein and Erickson [EE99]. The P-completeness of straight skeletons has important practical implications concerning the application of parallel algorithms. The investigations done in Section 3.2 and Section 3.3 are published in [HH11b]. A preliminary and short version was presented in [HH09b]. The results in Section 3.4 have been submitted for publication [HH11a].

- [HH09b] S. Huber and M. Held. A Practice-Minded Approach to Computing Motorcycle Graphs. In *Proc. 25th Europ. Workshop Comput. Geom.*, pages 305–308, Brussels, Belgium, Mar 2009
- [HH11b] S. Huber and M. Held. Motorcycle Graphs: Stochastic Properties Motivate an Efficient Yet Simple Implementation. *J. Exp. Algorithmics*, 2011. (in press)
- [HH11a] S. Huber and M. Held. Approximating a Motorcycle Graph by a Straight Skeleton. 2011. (submitted for publication)

3.1 PRIOR AND RELATED WORK

3.1.1 Applications of motorcycle graphs and related problems

The most prominent application of motorcycle graphs are straight skeletons. However, motorcycle graphs are also related to other geometric problems. Obviously, there exists a strong connection to ray-shooting problems. For instance, Ishaque et al. [IST09] presented an $O(n \log^2 n + m \log m)$ algorithm for m repetitive ray shooting-and-insertion operations in the plane among a set of polygonal obstacles of total size n. The generalized motorcycle graph problem, for which motorcycles need not all start at the same time, solves this problem in the following way: The polygonal obstacles are replaced by according walls and each ray is replaced by a motorcycle. By choosing the start time of the motorcycles such that no two motorcycles drive at the same time, one can simulate repetitive ray-shootings.

Eppstein and Erickson [EE99] mentioned that the art gallery algorithm by Czyzowicz et al. [CRU89] uses a geometric structure that is related to the motorcycle graph. They consider a set of straight-line segments, where each segment is growing in length and an endpoint of a segment stops propagating when it reaches another segment. The resulting structure can be interpreted as a motorcycle graph: The initial segments are considered as walls and from each endpoint is a motorcycle launched.

Eppstein et al. [EGKT08] consider motorcycle graphs on quadrilateral meshes, which model three-dimensional bodies. The idea is that motorcycles are driving on the edges of the mesh in a discrete manner and a motorcycle stops when it reaches the trace of another motorcycle. It is not exactly the same motorcycle graph problem as introduced by [EE99], but very similar in nature.

3.1.2 Prior work

At the moment, the most efficient motorcycle graph algorithm is due to Cheng and Vigneron [CV07], see Section 1.4.2.4. It computes the motorcycle graph of n motorcycles in $O(n\sqrt{n} \log n)$ time. Let us recall the main idea of their algorithm: Among the $O(n^2)$ intersections of the tracks of the motorcycles only $O(n)$ of them correspond to an actual crash event. Hence, the goal is to geometrically separate those motorcycles that do not interact. Cheng and Vigneron achieved this goal via the employment of $1/\sqrt{n}$-cuttings on the supporting lines of the traces. However, in order to reach the $O(n\sqrt{n} \log n)$ runtime complexity, the algorithm inherently relies on the fact that all motorcycles are known a priori such that the $1/\sqrt{n}$-cutting can be constructed prior to the simulation of the motorcycles movement. As a consequence, this algorithm is not suitable for the computation of the generalized motorcycle graph presented in Section 2.4.

In order to solve the generalized motorcycle graph problem, where motorcycles are allowed to emerge after the simulation started, we are in need of an algorithm that allows the dynamic insertion of motorcycles. Under this requirement, the fastest algorithm is due to Eppstein and Erickson [EE99] and runs in $O(n^{17/11+\epsilon})$ time and space, see Section 1.4.2.3. However, we want to emphasize that this algorithm is not suitable for an actual implementation due to its algorithmic complexity.

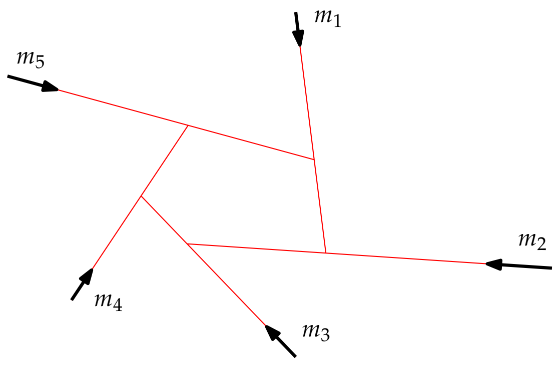


Figure 41: All motorcycles crash into each other.

A straight-forward approach for the actual computation of the motorcycle graph of n motorcycles is to simply simulate their movement in a brute-force manner. We can compute one crash event after the other by pair-wise checks among the motorcycles. By employing a priority queue Q for pending crash events, one obtains an implementation that runs in $O(n^2 \log n)$ time instead of $O(n^3)$. We refer to this algorithm as the standard priority-queue based algorithm.

To sum up, two sub-quadratic algorithms for the computation of motorcycle graphs are known, but only the algorithm by Eppstein and Erickson [EE99] can be used to compute the generalized motorcycle graph problem. Furthermore, no algorithm is known that is suitable for implementation, on one hand, and can be expected to exhibit a significantly sub-quadratic runtime for real-world applications on the other hand.

3.1.3 Geometric properties of the motorcycle graph

We interpret $\mathcal{M}(m_1, \ldots, m_n)$ as a graph and we add the infinite endpoints of the traces of escaped motorcycles to the graph as well. It follows that $\mathcal{M}(m_1, \ldots, m_n)$ contains exactly $2n$ vertices: Each motorcycle contributes a start point and an end point. (If multiple vertices coincide geometrically, we still count them as different vertices.) The number of finite vertices ranges from n to at most $2n$. The lower bound is attained if all motorcycles escape. The upper bound is attained if all motorcycles crash into each other, as illustrated in Figure 41.

Lemma 3.1. *$\mathcal{M}(m_1, \ldots, m_n)$ contains between n and $2n$ edges. The lower bound is attained if all motorcycles escape. The upper bound is attained if all motorcycles crash.*

Proof. The edge set of $\mathcal{M}(m_1, \ldots, m_n)$ results from n motorcycle traces, which are possibly split. Hence, the number of edges is at least n. In order to show the upper bound of $2n$, we denote by n_c the number of crashed motorcycles and by e the number of edges. $\mathcal{M}(m_1, \ldots, m_n)$ contains $n + (n - n_c)$ vertices of degree 1 and n_c vertices of degree 3. (Note that we obtain $n - n^c$ infinite vertices.) We charge each edge by both of its incident vertices. Considering the total number of charges results in $2e = n + (n - n_c) + 3n_c = 2n + 2n_c \leq 4n$ and hence $e \leq 2n$. The bound is tight if $n = n_c$. □

The following lemma is given by Eppstein and Erickson [EE99] without a proof. In the following, a pseudo-forest is a graph whose components are pseudo-trees and a pseudo-tree is a connected graph that contains at most one cycle.

Lemma 3.2 ([EE99]). *$\mathcal{M}(m_1, \dots, m_n)$ is a pseudo-forest.*

Proof. We denote the number of edges of $\mathcal{M}(m_1, \dots, m_n)$ by e. We may assume that the motorcycle graph $\mathcal{M}(m_1, \dots, m_n)$ is connected; the general case is shown analogously. If $\mathcal{M}(m_1, \dots, m_n)$ is not a pseudo-tree we can remove at least two edges without hurting connectedness. Since a connected graph with $2n$ vertices contains at least $2n - 1$ edges, it follows that $e - 2 \geq 2n - 1$. This is a contradiction to Lemma 3.1. □

3.2 STOCHASTIC CONSIDERATIONS OF THE MOTORCYCLE GRAPH

3.2.1 Number of intersections of bounded rays

In order to devise a motorcycle graph algorithm that runs fast in practice, it appears interesting to investigate the average trace length within the motorcycle graph $\mathcal{M}(m_1, \dots, m_n)$ of n motorcycles. Let us recall that the tracks of the motorcycles lead to $O(n^2)$ intersections, whereas the traces produce at most $O(n)$ intersections, namely at precisely those points where a motorcycle crashed into the trace of another.

Let us assume for a moment that the motorcycles drive at unit speed, the start points $v_1, \dots, v_n$ are distributed uniformly in the unit square $[0, 1]^2$ and the $\varphi_1, \dots, \varphi_n$ are distributed uniformly on $[0, 2\pi)$. Directly computing the expectation of the trace length of a single motorcycle trace appears to be complicated due to the stochastic dependencies among the motorcycle traces. However, it seems to be evident that the mean trace length cannot get too large, because two traces are not allowed to intersect in the interior of each other.

Let us consider a regular $\sqrt{n} \times \sqrt{n}$ grid on the unit square. A single motorcycle m_i starts at one grid cell and, while it moves, crosses a specific number of other cells. Each cell contains on average one start point of a motorcycle. The probability that m_i crosses k cells is falling at least exponentially with k if we would flip a coin to decide whether m_i passes the motorcycle that started in each cell entered by m_i. This thought experiment would suggest that a motorcycle does not pass more than $O(1)$ grid cells and, as a consequence, that the average trace length is in $O(1/\sqrt{n})$. However, a simple coin flip does not take into account the stochastic dependencies among the motorcycles.

In order to simplify the original question, we reformulate the problem. Instead of asking for the mean trace length of the motorcycles, we ask for the number of intersection points of bounded rays, where the length of the rays is chosen at random according to the probability density function f. The idea is that if we gain some information on the number of intersections than we also gain information on the mean length of the bounded rays as well.

Theorem 3.3 ([HH11b]). *Let $v_1, \dots, v_n$ denote n points that are uniformly i.i.d.[1] on the unit square $[0, 1]^2$. Further, by $\varphi_1, \dots, \varphi_n$ we denote n angles i.i.d. on $D := \{\delta_1, \dots, \delta_d\}$, with $d \in \mathbb{N}$, such that*

1 A common short-hand for "independent and identically distributed".

$\delta_i \in [0, 2\pi)$ occurs with probability p_i, where $\sum_i p_i = 1$. Next, let $L_1, \dots, L_n$ be i.i.d. on $[0, 0.2]$ according to a probability density function f.

For each $i \in \{1, 2, \dots, n\}$ consider a bounded ray $T_i \subset \mathbb{R}^2$ which starts at v_i, has direction angle φ_i and length L_i. We denote by $I = \sum_{i=2}^{n} 1_{T_1 \cap T_i \neq \emptyset}$ the number of intersections of T_1 with $T_2, \dots, T_n$, where 1_P denotes the indicator function of the predicate P. Then

$$\frac{\Delta}{25} \cdot E[L_1]^2(n-1) \quad \leq \quad E[I] \quad \leq \quad \Delta \cdot E[L_1]^2(n-1), \tag{3.1}$$

holds, where $\Delta := \sum_{i,j=1}^{d} p_i p_j |\sin(\delta_i - \delta_j)|$. Furthermore, for $\Delta > 0$ we obtain

$$E[I] \in \Theta\left(nE[L_1]^2\right). \tag{3.2}$$

(Distributing $L_1, \dots, L_n$ on the interval $[0, 0.2]$ is just a technicality that simplifies the following proof of the theorem.)

Proof. We assume that $p_i > 0$ for all $1 \leq i \leq n$. Hence, Δ is zero if and only if $\delta_i - \delta_j \in \pi\mathbb{Z}$ for all $1 \leq i, j \leq d$. The latter condition means that the supporting lines of the bounded rays are parallel. Hence, two rays intersect with probability zero and the claim of the theorem is trivial. So let us assume $\Delta > 0$. The law of total expectation yields

$$E[I] = \sum_{i=1}^{d} P(\varphi_1 = \delta_i)\, E[I \mid \varphi_1 = \delta_i] = \sum_{i=1}^{d} p_i\, E[I \mid \varphi_1 = \delta_i]. \tag{3.3}$$

Consider the ray T_1 fixed. In order to have a ray T_k intersect T_1 the start point v_k of T_k needs to start in a certain area, whose shape depends on the direction φ_k of T_k. For some arbitrary direction φ we denote this area by a parallelogram S_φ that is hinged at v_1, see Figure 42:

$$S_\varphi := \left\{ v_1 + a \begin{pmatrix} \cos\varphi_1 \\ \sin\varphi_1 \end{pmatrix} - b \begin{pmatrix} \cos\varphi \\ \sin\varphi \end{pmatrix} \;:\; a \in [0, L_1], b \in [0, 0.2] \right\}.$$

The mapping

$$F : \mathbb{R}^2 \to \mathbb{R}^2 : v \mapsto \begin{pmatrix} \cos\varphi_1 & \sin\varphi_1 \\ -\sin\varphi_1 & \cos\varphi_1 \end{pmatrix} \cdot (v - v_1)$$

models the translation of v_1 to the origin and subsequent clockwise rotation by the angle φ_1. Hence, $F(T_1)$ starts at the origin and points rightwards, see Figure 42.

For a ray T_k to intersect T_1, it is necessary that the start point v_k lies is in S_{φ_k}. (Keep in mind that the lengths are restricted to $[0, 0.2]$ and T_k has the direction angle φ_k.) We denote by $(x_i', y_i') := v_i' := Fv_i$ the translated start points v_i, with $1 \leq i \leq n$. Note that T_k intersects T_1 if and only if $F(T_k)$ intersects $F(T_1)$. However, $F(T_k)$ intersects $F(T_1)$ if and only if $v_k' \in F(S_{\varphi_k})$ and $L_k|\sin(\varphi_k - \varphi_1)| \geq |y_k'|$. (The latter condition basically states that T_k is long enough in order to intersect T_1.)

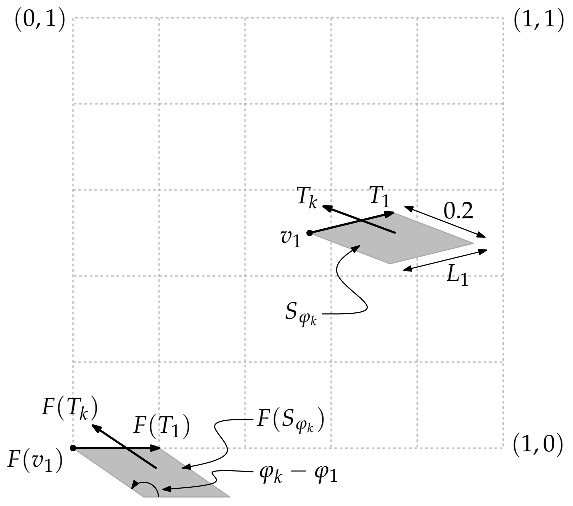


Figure 42: The big 5×5 grid illustrates $[0,1]^2$ with the origin at the left bottom point. We see that $\lambda(S_{\varphi_k}) = L_1 \cdot 0.2 \cdot |\sin(\varphi_k - \varphi_1)|$, where λ denotes the Lebesgue measure.

We note that $S_\varphi \subset [0,1]^2$ holds for all $\varphi_1 \in D$ and $\varphi \in [0,2\pi)$ only if $v_1 \in [0.4,0.6]^2$. Hence, for any $\delta_i \in D$ it follows that

$$E[I \mid \varphi_1 = \delta_i] \leq E[I \mid \varphi_1 = \delta_i, v_1 \in [0.4,0.6]^2].$$

On the other hand, by the law of total expectation we obtain

$$P(v_1 \in [0.4,0.6]^2) \cdot E[I \mid \varphi_1 = \delta_i, v_1 \in [0.4,0.6]^2] \leq E[I \mid \varphi_1 = \delta_i],$$

and therefore

$$\frac{1}{25} \cdot E[I \mid A_i] \quad \leq \quad E[I \mid \varphi_1 = \delta_i] \quad \leq \quad 1 \cdot E[I \mid A_i],$$

where A_i denotes the event that $\varphi_1 = \delta_i$ and $v_1 \in [0.4,0.6]^2$. Summing up over all i according to Equation (3.3) gives

$$\frac{1}{25} \cdot \sum_{i=1}^{d} p_i E[I|A_i] \quad \leq \quad E[I] \quad \leq \quad 1 \cdot \sum_{i=1}^{d} p_i E[I|A_i]. \tag{3.4}$$

In the next step, we analyze $E[I|A_i]$. Let I_j denote the number of intersections caused by rays having a direction angle δ_j. Hence $\sum_{j=1}^{d} I_j = I$. We further denote by $B_{i,j,m} \subseteq A_i$ those events of A_i, where exactly m rays point to direction δ_j. Note that each ray causes intersections independently to each other. Therefore, we can separate the cases according to the distribution of the direction angles $\varphi_2, \dots, \varphi_n$, which is multinomial:

$$\begin{aligned} E[I|A_i] &= \sum_{j=1}^{d} E[I_j|A_i] \\ &= \sum_{j=1}^{d} 25 \cdot \int_{[0.4,0.6]^2} \int_0^{0.2} \sum_{n_1+\dots+n_d=n-1} \\ &\quad \binom{n-1}{n_1,\dots,n_d} p_1^{n_1} \cdots p_d^{n_d} E[I_j|B_{i,j,n_j}] \, \mathrm{d}f(L_1)\mathrm{d}v_1. \end{aligned} \tag{3.5}$$

Next, we analyze $E[I_j|B_{i,j,m}]$. We observe that $E[I_j|A_i]$ is zero for $i = j$. Hence, we may assume $i \neq j$ in the sequel. Recall that we are asking for the expected number of intersections of T_1 with m rays that point in direction δ_j and which are distributed independently. Hence, we may assume that the rays $T_2, \dots, T_{m+1}$ are driving in direction δ_j. Recall that T_k intersects T_1 only if $v_k \in S_{\delta_j}$. Denoting by λ the Lebesgue measure, we obtain

$$E[I_j|B_{i,j,m}] = \sum_{l=0}^{m} \binom{m}{l} \lambda(S_{\delta_j})^l (1-\lambda(S_{\delta_j}))^{m-l} E[I_j|A_{i,j,l}],$$

where $A_{i,j,l} \subseteq B_{i,j,m}$ denotes the event that exactly l of the rays of $B_{i,j,m}$ start within S_{δ_j}.

We now resolve $E[I_j|A_{i,j,l}]$. W.l.o.g. assume that $T_2, \dots, T_{l+1}$ start in S_{δ_j}. Recall the notation $(x'_k, y'_k) := v'_k := F(v_k)$ and recall that we may assume $i \neq j$, since $E[I_j|A_i]$ is zero for $i = j$. Since every ray causes intersections independently, we get

$$\begin{aligned} E[I_j|A_{i,j,l}] &= \sum_{k=2}^{l+1} \frac{1}{\lambda(S_{\delta_j})} \int_{S_{\delta_j}} \int_0^{0.2} 1_{T_k \cap T_1 \neq \emptyset} \, \mathrm{d}f(L_k)\mathrm{d}v_k \\ &= l \frac{1}{\lambda(S_{\delta_j})} \int_{FS_{\delta_j}} \int_0^{0.2} 1_{L_2|\sin(\delta_i-\delta_j)| \geq |y'_2|} \, \mathrm{d}f(L_2)\mathrm{d}v'_2 \\ &= l \frac{1}{\lambda(S_{\delta_j})} L_1 \int_0^{0.2|\sin(\delta_i-\delta_j)|} P(L_2|\sin(\delta_i-\delta_j)| \geq y'_2) \, \mathrm{d}y'_2. \end{aligned}$$

Next, we substitute y'_2 by $z := y'_2/|\sin(\delta_j - \delta_i)|$ and get

$$\begin{aligned} E[I_j|A_{i,j,l}] &= l \frac{|\sin(\delta_j-\delta_i)|}{\lambda(S_{\delta_j})} L_1 \int_0^{0.2} P(L_2 \geq z) \, \mathrm{d}z \\ &= l \frac{|\sin(\delta_j-\delta_i)| L_1}{\lambda(S_{\delta_j})} E[L_2] \\ &= 5lE[L_1]. \end{aligned} \tag{3.6}$$

Since $\sum_{k=0}^{n}\binom{n}{k}p^k(1-p)^{n-k}$ equals 1, we can plug the last result into the expression for $E[I_j|B_{i,j,m}]$ and get

$$\begin{aligned}E[I_j|B_{i,j,m}] &= \sum_{l=0}^{m}\binom{m}{l}\lambda(S_{\delta_j})^l(1-\lambda(S_{\delta_j}))^{m-l}5lE[L_1]\\ &= 5E[L_1]\lambda(S_{\delta_j})m\cdot\sum_{l=1}^{m}\binom{m-1}{l-1}\lambda(S_{\delta_j})^{l-1}(1-\lambda(S_{\delta_j}))^{m-1-(l-1)}\\ &= 5E[L_1]\lambda(S_{\delta_j})m\\ &= mE[L_1]L_1|\sin(\delta_i-\delta_j)|.\end{aligned}$$

In the final step, we plug this result into Equation (3.5) and obtain

$$\begin{aligned}E[I|A_i] &= \sum_{j=1}^{d}25\int_{[0.4,0.6]^2}\int_0^{0.2}\sum_{n_1+\cdots+n_d=n-1}\binom{n-1}{n_1,\ldots,n_d}p_1^{n_1}\cdots p_d^{n_d}\\ &\qquad n_jE[L_1]L_1|\sin(\delta_i-\delta_j)|\,\mathrm{d}f(L_1)\,\mathrm{d}v_1\\ &= 25\int_{[0.4,0.6]^2}\mathrm{d}v_1\cdot E[L_1]\int_0^{0.2}L_1\,\mathrm{d}f(L_1)\cdot\sum_{j=1}^{d}|\sin(\delta_i-\delta_j)|\\ &\qquad\sum_{n_1+\cdots+n_d=n-1}n_j\binom{n-1}{n_1,\ldots,n_d}p_1^{n_1}\cdots p_d^{n_d}.\end{aligned}\tag{3.7}$$

Next we use

$$n_j\binom{n-1}{n_1,\ldots,n_d} = (n-1)\binom{n-2}{n_1,\ldots,n_j-1,\ldots n_d}$$

and therefore see that

$$\sum_{n_1+\cdots+n_d=n-1}n_j\binom{n-1}{n_1,\ldots,n_d}p_1^{n_1}\cdots p_d^{n_d} \quad=\quad (n-1)p_j.$$

Finally, by $E[L_1]=\int_0^{0.2}L_1\,\mathrm{d}f(L_1)$, it follows that

$$E[I|A_i] = E[L_1]^2(n-1)\cdot\sum_{j=1}^{d}p_j|\sin(\delta_i-\delta_j)|.\tag{3.8}$$

Using this result in Equation (3.4) finally proves the assertions of the theorem. □

Choosing the lengths $L_1,\ldots,L_n$ on the interval $[0,0.2]$ was a technical twist. It allowed us to assume that if a bounded ray T_1 starts in $[0.4,0.6]^2$ and is intersected by another bounded ray T_2 then T_2 started definitely in $[0,1]^2$, because the start points of T_1 and T_2 have a distance of at most 0.4. Therefore, it holds that

$$\frac{1}{25}\cdot E[I|A] \le E[I] \le E[I|A],$$

where A denotes the event that T_1 started in $[0.4, 0.6]^2$. Note that the left inequality follows by the law of total expectation. In order to prove Theorem 3.3 it remained to show that $E[I|A] = \Delta \cdot E[L_1]^2(n-1)$. However, if we distribute $L_1, \dots, L_n$ on an interval $[0, \epsilon]$, with any positive $\epsilon < 0.25$, the argument from above easily extends to

$$(1-4\epsilon)^2 \cdot E[I|A] \leq E[I] \leq E[I|A],$$

where A denotes the event that $v_1 \in [2\epsilon, 1-2\epsilon]^2$. Note that the remainder of the proof of Theorem 3.3 is not affected by the above generalization. As a consequence, Theorem 3.3 can actually be generalized to

$$(1-4\epsilon)^2 \Delta \cdot E[L_1]^2(n-1) \leq E[I] \leq \Delta \cdot E[L_1]^2(n-1). \tag{3.9}$$

3.2.2 Implications to the motorcycle graph

Consider n random motorcycles $m_1, \dots, m_n$ that drive at unit speed and where the start points $v_1, \dots, v_n$ are chosen uniformly from the unit square $[0,1]^2$ and the direction angles $\varphi_1, \dots, \varphi_n$ are chosen from a set $D := \{\delta_1, \dots, \delta_d\}$, where $\delta_i \in [0, 2\pi)$ occurs with probability p_i, as in Theorem 3.3. After generating n random motorcycles as described above, we compute the motorcycle graph $\mathcal{M}(m_1, \dots, m_n)$ and record all the trace lengths. We can repeat this experiment a number of times and keep on recording the trace lengths. The samples recorded can be used to obtain an approximation $\hat{f}$ of the density of the trace lengths of a motorcycle graph with n motorcycles. Using the approximate density $\hat{f}$ in Theorem 3.3 establishes the relation (3.9) between the expected number of intersections $E[I]$ and the mean trace length $E[L_1]$ according to the approximate density function $\hat{f}$. Unfortunately, both $E[I]$ and $E[L_1]$ are unknown.

However, for increasingly larger values of n the vast majority of motorcycles does not reach the boundary of $[0,1]^2$, but crashes against other traces. Hence, as the number n of motorcycles increases, the trace lengths shrink in the average case[2], and for sufficiently large n the vast majority of motorcycles can be expected to have a trace length less than some constant ϵ smaller than 0.25. Since there are at most n crashes, a motorcycle trace may be assumed to intersect two other traces on average: the motorcycle itself crashes into another trace and a second motorcycle crashes into the considered trace. This suggests $E[I] = 2$, which can also be verified experimentally. (Actually, $E[I] \in O(1)$ would suffice for our subsequent runtime analysis of our algorithm in Section 3.3.)

Plugging $E[I] = 2$ and a small ϵ in the inequality (3.9) suggests the following approximation for the mean trace length:

$$E[L_1] \approx \sqrt{\frac{2}{(n-1)\sum_{i,j=1}^{d} p_i p_j |\sin(\delta_i - \delta_j)|}}. \tag{3.10}$$

Of course, the assumption that $L_1, \dots, L_n$ are independently distributed is not justified for the actual motorcycle graph problem. However, in Section 3.3.3, we are able to substantiate this approximative formula for the mean trace length of motorcycle graphs by providing sound experimental evidence.

2 Of course, the motorcycles must not all drive in parallel directions.

3.3 A SIMPLE AND PRACTICE-MINDED IMPLEMENTATION

In order to compute the straight skeleton $\mathcal{S}(G)$ of a planar straight-line graph G by using Bone, it is vital that we can compute the motorcycle graph $\mathcal{M}(G)$ fast in practice. The algorithm by Eppstein and Erickson [EE99] would allow us to compute $\mathcal{M}(G)$, but the algorithm is too complicated to be implemented. The algorithm by Cheng and Vigneron [CV07] is easier, but still complicated to implement — e. g., we would need to implement an algorithm for the $1/\sqrt{n}$-cutting — and it needs to know all motorcycles a priori. The standard priority-queue based algorithm is trivial to implement, but exhibits an $O(n^2 \log n)$ runtime in practice. For our purposes, we seek a motorcycle graph implementation that is simple enough to be implemented, runs fast in practice, and which supports the dynamic insertion of new motorcycles.

Our approach is to take the algorithm of Cheng and Vigneron [CV07], drop the arrangements on each cutting cell and to replace the $1/\sqrt{n}$-cutting by a regular $\sqrt{n} \times \sqrt{n}$ grid. In other words, we apply geometric hashing to the standard priority-queue based algorithm. The motivation for our approach is a simple trade-off: we lose the deterministic $O(n\sqrt{n}\log n)$ time complexity, but instead gain an algorithm that (i) is easy to implement and (ii) runs in $O(n \log n)$ time if the motorcycles are sufficiently uniformly distributed.

3.3.1 Details of the algorithm

The input to our algorithm consists of a set $M = \{m_1, \ldots, m_n\}$ of motorcycles and a set $W = \{w_1, \ldots, w_u\}$ of rigid walls. The motorcycles need not all be known a-priori. A wall is modeled as a straight-line segment and a motorcycle m_i is given by a start point v_i, a speed vector s_i and a start time $t_i^* \in [0, \infty)$. Due to practical numerical advantages, we scale the input such that the bounding box of the start points is a proper subset of the unit square $[0,1]^2$. For the matter of simplicity, we first restrict our computation of the motorcycle graph to the unit square $[0,1]^2$. However, this restriction can be waived easily, see Section 3.3.4. (The restriction of the computation to $[0,1]^2$ can be enforced within our framework by adding four dummy walls that form the boundary of $[0,1]^2$.)

Our algorithm maintains two geometric hashes, H_M and H_W, which form regular $\sqrt{n} \times \sqrt{n}$ grids on $[0,1]^2$. While H_M keeps track of the motorcycles, H_W contains the walls of W. We use H_W in order to determine whether a motorcycle crashed against a wall. However, since we consider walls to be rigid, we could have employed more efficient ray-shooting algorithms in terms of worst-case complexity. For the matter of simplicity, we choose the geometric hash for our implementation.

The basic algorithm is a discrete event simulation of the movement of the motorcycles with two types of events: crash events and switch events. A crash event indicates that a motorcycle crashes against another motorcycle or a wall, and a switch event occurs when a motorcycle leaves one grid cell and enters a neighboring one. All events pending are kept in a priority queue Q. Furthermore, for every motorcycle m_i, we maintain a balanced binary search tree $C[m_i]$ that contains potential future crash events of the motorcycle m_i.

The algorithm starts with filling H_W with all walls of W and then invokes `insertMc(`m`)` for each $m \in M$, which inserts a new motorcycle m to our data structures. The main loop of the algorithm extracts one event e from Q after the other and processes them by calling `handle(e)`, depending on the actual type of the event e. If a newly emerging motorcycle m should be inserted at any time of computation then `insertMc(m)` is called. The procedures `insertMc` and `handle` are described in the sequel:

- **`insertMc(motorcycle m)`**: We first create an empty binary search tree $C[m]$ and then insert a switch event for m into Q. The occurrence time of the switch event is set to the start time of m.
- **`handle(switch event e of the motorcycle m)`**: We register m at the cell that m entered and add the subsequent switch event of m to Q, if one exists. Then we check for a potential crash against a wall in the current cell and add the earliest one, if existing, as a crash event to Q. We clear $C[m]$ and for every other motorcycle m' registered in the current cell, we check for an intersection of the tracks of m and m'. For each such intersection, we add a corresponding crash event into $C[m]$ if m' reaches the intersection before m, and into $C[m']$ for the dual case. Note that if we add an event into $C[m']$ that ends up being the earliest in $C[m']$ then we have to update Q accordingly. Finally, we add the earliest crash event of $C[m]$ into Q.
- **`handle(crash event e of the motorcycle m)`**: First, we mark the motorcycle m as crashed and clear $C[m]$. Note that the trace of m ends at the corresponding crash point. Secondly, we remove the remaining switch event of m from Q. (Alternatively, we could leave the event in Q, but check at each switch event whether the current event is still valid.) Then we clean up interactions with other motorcycles m' in the current grid cell: We remove from Q all crash events, where m is involved and which got invalid, because it turned out that m will not reach the location of the potential crash event. Likewise, if $C[m']$ contains such an invalid crash event then it is removed as well.

3.3.2 Runtime analysis

In the subsequent analysis, we ignore the influence of the wall handling and concentrate on the computation of the motorcycle graph only. The procedure `insertMc` is called exactly n times, which takes $O(n \log n)$ time in total. A single crash event or switch event of a motorcycle is handled in $O(k \log n)$ time, where $k \in O(n)$ denotes the number of motorcycles in the cell affected. Note that we can remove an element of Q in $O(\log n)$ time if we have a pointer to the element and if we use, for instance, a maximizing heap to implement Q.

In total we have $O(n)$ crash events and at most $O(n\sqrt{n})$ switch events. Hence, in the worst case, our algorithm runs in $O(n^2\sqrt{n} \log n)$ time. However, it seems very unlikely that the worst case actually happens: it would require that $\Omega(n)$ motorcycles drive across $\Omega(\sqrt{n})$ common grid cells. Hence, those $\Omega(n)$ motorcycles drive virtually parallel along a long strip that is only $O(1/\sqrt{n})$ units thick and, moreover, no other motorcycle is allowed to cross this strip, until the motorcycles crossed a constant fraction of the whole grid.

As we learned in Section 3.2, a motorcycle is visiting $O(1)$ grid cells on average, if the motorcycles are distributed uniformly enough. Consequently, a single grid cell is occupied by $O(1)$ motorcycles on average. Again, the initialization consumes $O(n \log n)$ time in total. However, now a single crash event or switch event is handled in $O(\log n)$ time in the average case. Still, we have $O(n)$ crash events but in the mean observe only $O(1)$ switch events per motorcycle. Summarizing, we may expect a runtime of $O(n \log n)$ for sufficiently uniformly distributed input, as motivated by Section 3.2. We provide sound experimental evidence in Section 3.3.3 that underpins our arguments for an $O(n \log n)$ runtime for uniformly distributed start points, but also demonstrates an $O(n \log n)$ runtime for most of the real-world input, where start points are not necessarily distributed uniformly.

3.3.3 Experimental results and runtime statistics

Our motorcycle code is called Moca[3]. It is implemented in C++ and uses the STL for common data structures like queues, red-black trees and priority queues. All geometric computations are based on ordinary IEEE 745 double-precision floating-point arithmetic. Moca provides runtime options for a posteriori tests to check necessary conditions for the correctness of the resulting motorcycle graph. In particular, we check (i) for motorcycle traces with a free end[4] and (ii) for motorcycle traces intersecting in the relative interiors of each other. To the best of our knowledge, this is the first competitive motorcycle graph implementation. For this reason, we do not compare our code with other implementations, but content ourselves with a discussion on the performance of Moca.

3.3.3.1 *Verification of the stochastic analysis*

We used Moca to collect statistical properties of several datasets, in order to underpin the theoretical results obtained in Section 3.2. We set up three experiments that investigate the dependence of the mean trace length on (i) the number of motorcycles n and (ii) the direction angles δ_i in Equation (3.10).

For the first two experiments, we created datasets with n random motorcycles by choosing the start points uniformly in $[0,1]^2$ and the direction angle uniformly from the set $\{0, \delta\}$. Equation (3.10) asserts that the mean trace length L is given by

$$E[L] \approx \frac{2}{\sqrt{(n-1)|\sin\delta|}}. \tag{3.11}$$

In Experiment 1, we created a dataset for each $n \in \{i \cdot 5\,000 \ : \ 1 \leq i \leq 60\}$ and $\delta \in \{i\pi/12 \ : \ 1 \leq i \leq 6\}$ and used Moca to determine the mean trace length among each dataset. In the left subfigure of Figure 43, each dot depicts the mean trace length of a dataset, where the resulting values were normalized for illustrative reasons, by dividing them by the factor $2/\sqrt{n-1}$. As predicted by Equation 3.11, the plot shows six[5] horizontal lines, where each line corresponds to a particular value of δ. In Experiment 2, we created datasets for $n = 10\,000$ and $\delta \in \{i\pi/40 \ : \ 1 \leq i \leq 40\}$. The reciprocal values of the normalized mean trace lengths are shown in the right subfigure of Figure 43. The normalized mean trace lengths are aligned on the reference curve $\sqrt{|\sin\delta|}$, which matches the estimation provided by Equation (3.11).

3 Short-hand for MOtorcycle CrAsher.
4 Each endpoint of a trace of a motorcycle that did not escape must coincide with the trace of another motorcycle.
5 The two bottom lines mostly overlap.

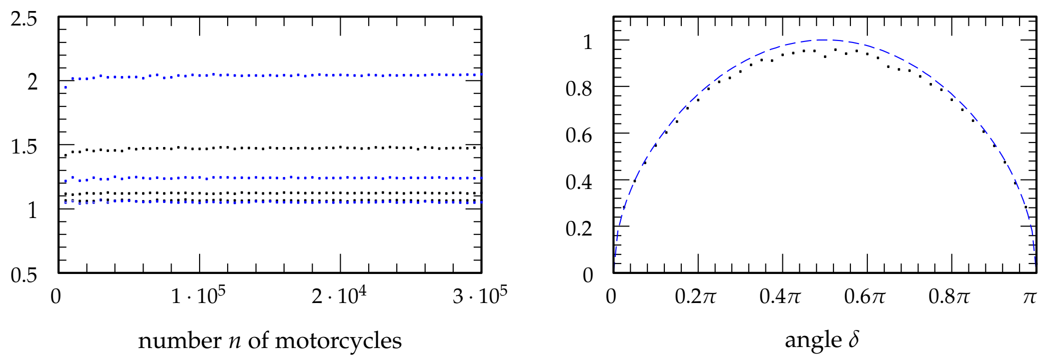


Figure 43: Two experiments illustrating the mean trace length of n motorcycles. A dataset contains motorcycles with uniformly distributed start points and uniformly distributed directions on the set $\{0, \delta\}$. Left, Experiment 1: every dot depicts the mean trace length for different n and δ. The resulting values are divided by $2/\sqrt{n-1}$. Right, Experiment 2: every dot depicts the reciprocal of the mean trace length for a fixed n. The x-axis illustrates δ. The reference curve $\sqrt{|\sin\delta|}$ is shown in blue.

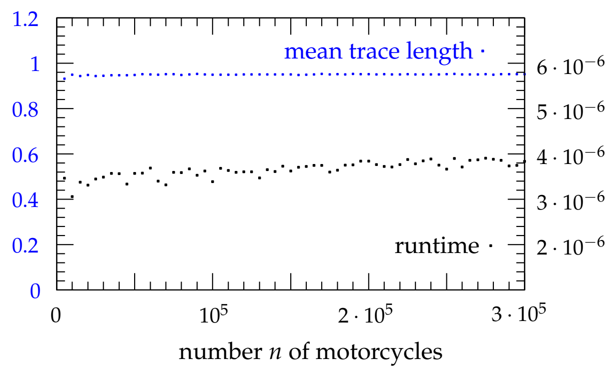


Figure 44: The runtime of Moca and the mean trace length of random dataset containing n motorcycles. The start points are uniformly distributed on $[0,1]^2$ and the directions are uniformly distributed on $[0, 2\pi)$. Each blue dot depicts the mean trace length and each black dot the runtime on a single dataset. For illustrative reasons we divided the runtime by $n \log n$ and the mean trace length by $\sqrt{\frac{\pi}{n-1}}$.

In the third experiment, we considered the mean trace length of datasets where the direction angles are distributed uniformly on $[0, 2\pi)$. This can be achieved by uniformly distributing the direction angles on $\{\delta_1, \ldots, \delta_d\}$, with $\delta_i = i\frac{2\pi}{d}$, and subsequently considering $d \to \infty$. According to Equation (3.10) we obtain:

$$\begin{aligned} E[L] &\approx \lim_{d\to\infty} d\sqrt{\frac{1}{(n-1)\sum_{i=1}^{d-1}(d-i)|\sin i\frac{2\pi}{d}|}} \\ &= \lim_{d\to\infty} d\sqrt{\frac{1}{(n-1)\cdot d^2\sum_{i=1}^{d-1}(1-\frac{i}{d})|\sin 2\pi\frac{i}{d}|\cdot\frac{1}{d}}} \\ &= \sqrt{\frac{1}{(n-1)\int_0^1(1-x)|\sin 2\pi x|\,dx}} \\ &= \sqrt{\frac{\pi}{n-1}}. \end{aligned} \tag{3.12}$$

We again generated random datasets, where n ranges from 5 000 to 300 000 in steps of 5 000. Figure 44 shows the runtimes and the mean trace lengths on these datasets. The runtime has been divided by $n \log n$ and the mean trace lengths have been normalized by the factor $\sqrt{\pi/n-1}$. As predicted, the graph shows two horizontal lines. That is, the runtime of Moca is in $O(n \log n)$ and the mean trace length is approximately $\sqrt{\pi/n-1}$.

3.3.3.2 *Runtime statistics on real world data*

We performed the following runtime tests on a Linux machine with a 32-bit Kernel. We used an Intel E6700 Core 2 Duo processor, clocked at 2.66 GHz, using 4 GiB of memory. Note that the 32-bit architecture limits the memory footprint of Moca to roughly 3 GiB in user space. For time measurement, we used the C library function `getrusage`.

Since the development of Moca is motivated by our straight-skeleton implementation Bone, we consider planar straight-line graphs G as input and test Moca by computing the motorcycle graph $\mathcal{M}(G)$ induced by G. We ran Moca on more than 22 000 datasets, consisting of synthetic and real-world data. Our real-world datasets include polygonal cross-sections of human organs, GIS maps of roads and river networks, polygonal outlines of fonts, and boundaries of work-pieces for CNC machining or stereo-lithography. The synthetic test data was generated by means of RPG [AH96] and an enhanced version of RPG due to Held. The synthetic data also contains contrived data, like extremely fine approximations of smooth curves, where the vertices are distributed highly irregularly.

Figure 45 (a) illustrates the runtime of Moca on each dataset. The runtimes are given in seconds and were divided by $n \log n$ for illustrative reasons. To avoid unreliable timings and other idiosyncrasies of small datasets, we only plot the results of datasets with at least 100 motorcycles. We observe that Moca exhibits a runtime of 2 to $10 \cdot n \log n$ µs for the vast majority of our datasets. About 100 outliers that did not fit into this plot, took up to $2 \cdot n \log n$ ms. A typical dataset of this kind is a sampled ellipse: all motorcycles escape and visit a large number of cells.

The plot in Figure 45 (b) shows the mean trace length, which has been multiplied by $\sqrt{n}$ for a better illustration. We can see that for most datasets in the entire database, the mean

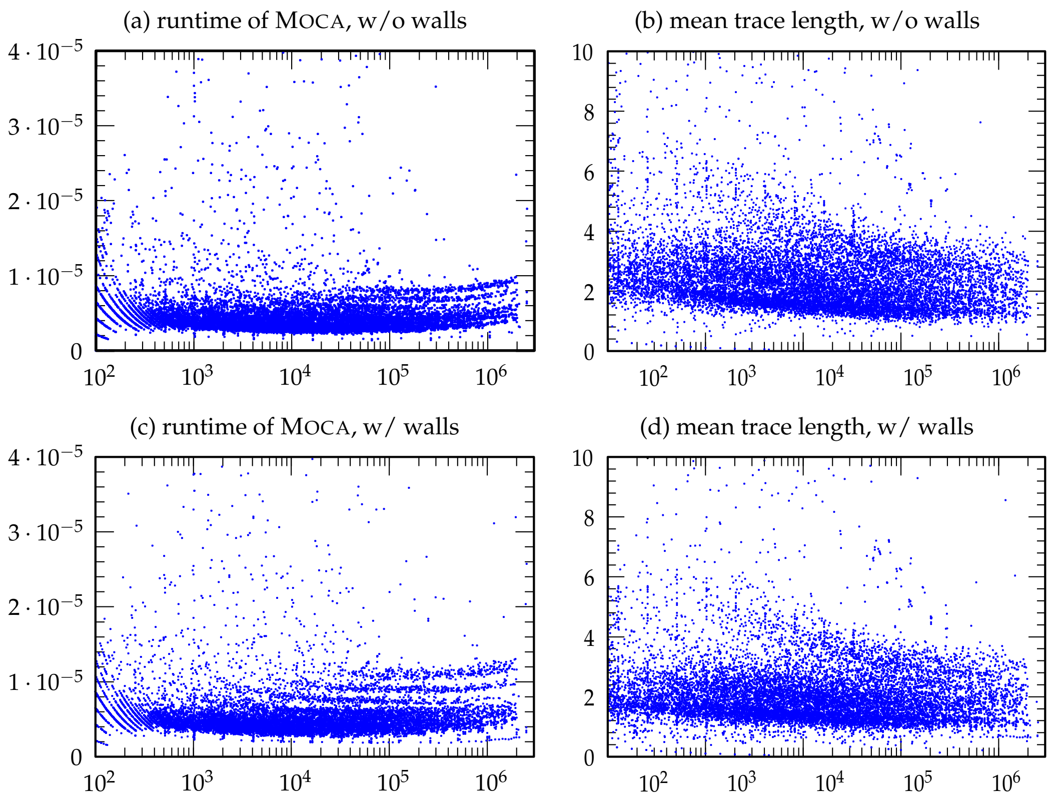


Figure 45: A dot depicts the runtime of Moca resp. the mean trace length of a dataset. The x-axis show the size n of the datasets. (a) Runtime of Moca in seconds, divided by $n \log n$. (b) Mean trace length, multiplied with $\sqrt{n}$. (c, d) Show the plots analogous to (a, b), but with walls inserted.

trace length is between $0.5/\sqrt{n}$ and $6/\sqrt{n}$. This circumstance is the main reason for the good runtime behavior achieved by our implementation. Our theoretical analysis carried out in Section 3.2 is based on the assumption that the start points are distributed uniformly in the unit square. Figure 45 (b) provides experimental evidence that this assumption can be relaxed for real-world input, which still exhibits an average trace length of $O(\sqrt{n})$ for most datasets.

For the test runs of subfigure (a) and (b), we did not insert the edges of the input graphs G as walls. However, additional tests demonstrate that inserting the walls has only a negligible impact on the runtime of Moca resp. the mean trace length of the resulting motorcycle graphs. We illustrated the corresponding runtimes and mean trace lengths in subfigure (c) and (d).

We further investigated the runtime of Moca on random datasets with non-uniformly distributed start points. For this reason, we generated datasets where the start points resp. the x-coordinates of the start points are distributed Gaussian resp. multi-modal Gaussian. By varying the standard deviations, we are able to successively concentrate the start points at specific regions of the unit square. In Figure 46, we plotted the runtime of Moca and the mean trace lengths on datasets, where (a) the start points are distributed Gaussian at the center of the unit square and (b) only the x-coordinates are distributed Gaussian with mean 0.5. As expected, for a decreasing standard deviation the runtime of Moca increases accordingly. Note that the mean trace length decreases as well, but the impact on the runtime due to the condensed start points dominates the result.

3.3.4 Extending the computation beyond the unit square

The computation of the motorcycle graph outside the unit square can be done very efficiently in terms of worst-case complexities. At first, we compute the motorcycle graph inside the unit square. When a motorcycle reaches the boundary of the unit square it is temporarily stopped. After all motorcycles crashed or have been stopped at the boundary of the unit square, we compute the motorcycle graph outside the unit square. This, however, can be done very efficiently: We interpret the boundary of the growing unit square as a sweep-line. Each motorcycle sits on this growing square and whenever two motorcycles m_1, m_2 would exchange their positions — i. e., the square reached an intersection point of the tracks of m_1 and m_2 — then either m_1 crashed into m_2 or vice versa. The crashed motorcycles are removed from the sweep line and the algorithm continues. The algorithm is correct because the motorcycles are, roughly speaking, moving in the same direction as the wavefront and never against it. Using a balanced binary tree structure on the sweep line allows us to compute the motorcycle graph outside the unit square in $O(n \log n)$ time.

This strategy is motivated by the sweep-line algorithm due Eppstein and Erickson [EE99]. Their algorithm computes the motorcycle graph of motorcycles, where the velocities have positive x-coordinates, in $O(n \log n)$ time. Note that our approach also works if we use the convex hull of the start points instead of the boundary of the unit square as initial sweep line. In other words, the motorcycle graph outside the convex hull of the start points can be computed in $O(n \log n)$ time. However, the computation of the motorcycle graph within the convex hull remains complicated.

Note that the approach presented above assumes that no motorcycle emerges during the propagation of the sweep line. However, under specific circumstances we can still allow the launch of new motorcycles: (i) the start point needs to coincide with the current position of the wavefront and (ii) the motorcycle needs to drive to the one side of the wavefront that has not yet been swept. Both conditions are fulfilled by our generalization of the motorcycle graph.

Moca pursues a simple approach to continue the computation of the motorcycle graph outside the unit square, by extending the $2(\sqrt{n} + 1)$ grid lines to infinity. We consider the resulting infinite grid cells as part of the hash and continue the simulation of the moving motorcycles on this grid cells. In order to restrict our computations to the unit square, we add four dummy walls that cover the boundary of the unit square. The impact on the performance of Moca, when the boundary walls are removed, directly depends on the number of motorcycles that do not crash within the unit square, which includes the motorcycles that escape.

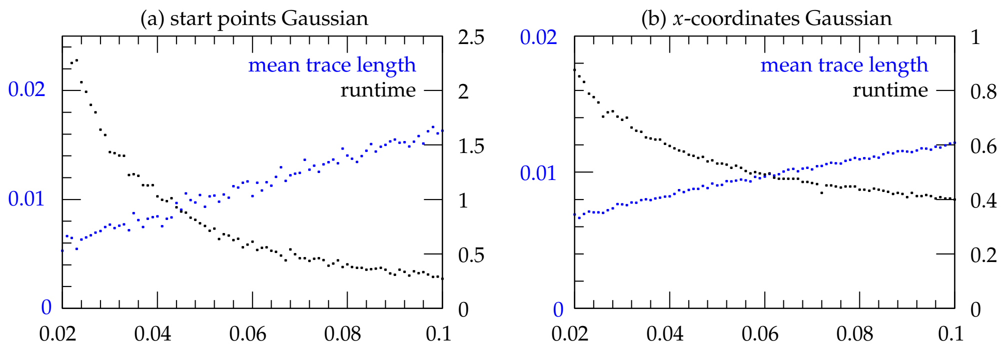


Figure 46: The runtime of Moca and the mean trace length on random datasets, where the direction angles are distributed uniformly on $[0, 2\pi)$. (a) 2 000 motorcycles, where the start points are distributed Gaussian with mean $(0.5, 0.5)$. (b) 10 000 motorcycles, where the x-coordinates of the start points are distributed Gaussian with mean 0.5 and the y-coordinates are distributed uniformly on $[0, 1]$. In both subfigures, the x-axis shows the corresponding standard deviation.

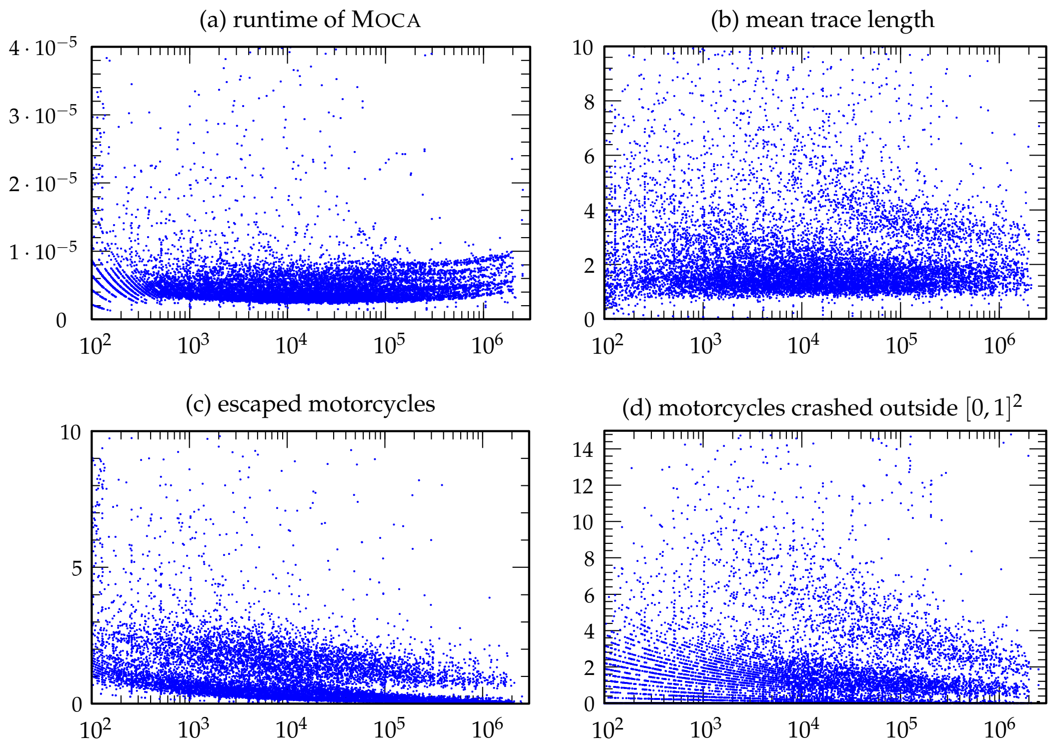


Figure 47: Statistics obtained from Moca when the computation is continued beyond the unit square. The x-axis depicts the number n of motorcycles in the dataset. (a) Runtime, divided by $n \log n$. (b) Mean trace length of the crashed motorcycles, multiplied with $\sqrt{n}$. (c) Number of escaped motorcycles, divided by $\sqrt{n}$. (d) Number of motorcycles crashed outside the unit square, divided by $\sqrt[4]{n}$.

Figure 47 (a) and (b) show the runtime of Moca and the mean trace length, where the computation is continued beyond the unit square. We observe that there is only a little impact on the runtime of Moca and the same holds for the mean trace length. In subfigure (c) we plotted the number of motorcycles that escaped and subfigure (d) shows the number of motorcycles that crashed outside of the unit square. Note that the remaining motorcycles crashed within the unit square. We observe that the number of motorcycles that escaped is roughly $\Theta(\sqrt{n})$ and the number of motorcycles that crashed outside $[0,1]^2$ is approximately $\Theta(\sqrt[4]{n})$. Hence, the vast majority of motorcycles remained within the unit square for most of the datasets in our database. This is the reason why Moca performs well, even though we continue the computation of the motorcycle graph outside the unit square.

3.4 EXTRACTING THE MOTORCYCLE GRAPH FROM THE STRAIGHT SKELETON

In the introduction of this chapter, we mentioned the close relation between straight skeletons and motorcycle graphs. From a geometric point of view, this relation becomes visible by Theorem 2.26, which states that the motorcycle graph covers the reflex arcs of the straight skeleton. In general, it can be observed that the gap between the reflex arcs and the motorcycle traces decreases as the motorcycles resp. the reflex wavefront vertices move faster. From this observation, the question arises whether we can always approximate the motorcycle graph using the straight skeleton. In other words, can we compute the motorcycle graph using a straight-skeleton algorithm?

We think that this question is interesting due to following reasons. Firstly, since motorcycle graphs play an important role in the computation of straight skeletons, it appears to be important to deepen the insight into the geometric relation between motorcycle graphs and straight skeletons. Secondly, if we manage to efficiently reduce the construction problem of motorcycle graphs to straight skeletons then we are able to describe the complexity of one problem in terms of the other problem. In fact, in Section 3.4.3, we present a proof for the P-completeness of straight skeletons that is based on the results of the subsequent sections and the P-completeness of motorcycle graphs due to Eppstein and Erickson [EE99].

3.4.1 Approximating the motorcycle graph by the straight skeleton

We assume that n motorcycles $m_1, \dots, m_n$ are given, where each motorcycle m_i has a start point p_i and a speed vector v_i. All motorcycles start at the same time. In particular, no motorcycles emerge later on. Can we find an appropriate planar straight-line graph G such that the straight skeleton $\mathcal{S}(G)$, resp. a proper subset of $\mathcal{S}(G)$, approximates $\mathcal{M}(m_1, \dots, m_n)$ up to a given tolerance?

It follows from Theorem 2.26 that if we construct G such that at each p_i a reflex wavefront vertex starts moving along the track of m_i with velocity v_i then the reflex arcs of $\mathcal{S}(G)$ that belong to these wavefront vertices approximate the traces of $\mathcal{M}(m_1, \dots, m_n)$ up to some gap. The simplest way in order to obtain such reflex wavefront vertices is to place isosceles

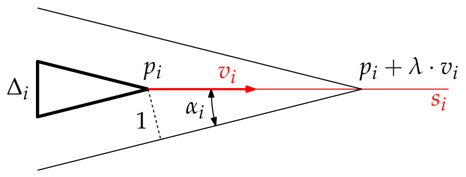


Figure 48: The isosceles triangle Δ_i is placed on the vertex p_i. The interior angle of Δ_i at p_i is $2\alpha_i$ and $\lambda|v_i| = 1/\sin\alpha_i$.

triangles Δ_i at each p_i such that the trace of m_i is bisecting the exterior angle of Δ_i. By setting the angle of Δ_i at p_i accordingly, we can adapt the speed of the corresponding wavefront vertex.

As already mentioned, we observe that the faster m moves the better is its trace approximated by the reflex straight-skeleton arc that is covered by m. But since the speed of $m_1, \ldots, m_n$ is part of the input to our problem setting, we cannot change the speeds of the motorcycles. Moreover, a wavefront vertex has always a speed of at least 1. If a motorcycle has a speed less than 1 then we cannot construct an according Δ_i, as mentioned above. However, it easy to see that if we multiply each speed vector v_i by the same constant $\lambda > 0$ then the motorcycle graph remains the same. Hence, the idea is to place at each p_i an isosceles triangle Δ_i, where the interior angle at p_i equals $2\alpha_i$ and α_i is given by

$$\alpha_i := \arcsin \frac{1}{\lambda|v_i|}. \tag{3.13}$$

The constant λ is chosen large enough such that $\lambda|v_i| > 1$ for all $1 \leq i \leq n$, see Figure 48. The size of Δ_i will be specified later. We denote by u_i the reflex wavefront vertex emanated at p_i. Note that u_i has λ-times the speed of m_i. Furthermore, note that each triangle Δ_i is emanating two additional motorcycles at the other two corners. This leads us to a refined version of our initial question: Can we find λ large enough such that the reflex arcs of $\mathcal{S}(G)$ that are emanated from p_i, approximate $\mathcal{M}(m_1, \ldots, m_n)$ up to a given tolerance?

We denote the trace of each motorcycle m_i by s_i. Recall that D_μ denotes the disk with radius μ and the center at the origin. Since all traces are closed sets, there exists $\mu > 0$ such that the Minkowski sums $s_i + D_\mu$ and $s_j + D_\mu$ are disjoint for all $1 \leq i, j \leq n$, where s_i and s_j are disjoint. For example, let μ be a third of the pairwise infimum distance among all disjoint traces. Note that two traces s_i, s_j are intersecting if and only if m_i crashed into m_j or vice versa. Further, we choose μ small enough such that $p_i + D_\mu$ is disjoint with $s_j + D_\mu$ for all $1 \leq i, j \leq n$, where p_i is disjoint to s_j. We denote by G the planar straight-line graph that consists of the triangles Δ_i, as described above, where the lengths of the arms that are incident to p_i are set to $\mu/2$.

Lemma 3.4. *The wavefronts of Δ_i stay within $s_i + D_\mu$ until time $\mu/4$ if $\lambda \geq \frac{2}{|v_i|}$.*

Proof. From $\lambda \geq 2/|v_i|$ follows that $2\alpha_i$ is at most 60°. Hence, the other two angles of Δ_i are at least 60° and the two additional motorcycles at Δ_i have a speed of at most 2. Since the start points of those motorcycles have a distance of $\mu/2$ from p_i and since they drive at most a distance of $\mu/2$ in time $\mu/4$, they stay within $p_i + D_\mu$.

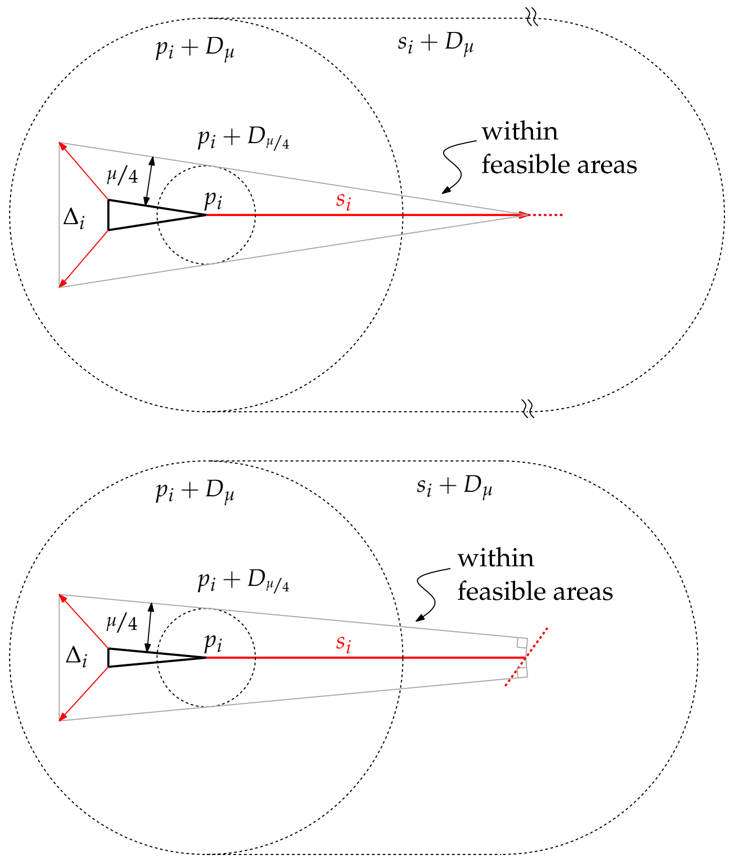


Figure 49: The wavefronts of Δ_i are bounded to $s_i + D_\mu$ until time $\mu/4$. Top: the motorcycle m_i did not crash until time $\mu/4$. Bottom: the motorcycle m_i did crash until time $\mu/4$.

Next, we consider a wavefront edge e that is emanated from Δ_i and propagating to the exterior of Δ_i. Let us recall the slabs of the lower envelope from Section 2.4.3. We call the slab that belongs to e and which is vertically projected onto the plane the *feasible area* of e. The face $f(e)$ is contained in the feasible area of e. We have to proof that $e(t)$ stays within $s_i + D_\mu$ until time $\mu/4$. First, we restrict the feasible area of e to those points that have an orthogonal distance of at most $\mu/4$ to $\overline{e(0)}$, see Figure 49. We distinguish two cases: the motorcycle m_i did crash or did not crash until time $\mu/4$. In both cases the corner points of the restricted feasible areas are contained within $s_i + D_\mu$, the restricted feasible areas are convex and $s_i + D_\mu$ is convex. Hence, the wavefronts of Δ_i until time $\mu/4$ are contained within $s_i + D_\mu$. $\square$

We denote by $\overrightarrow{s_i}$ the ray that starts at p_i in direction v_i and define

$$L := \max_{1 \le i,j \le n} d(p_i, \overrightarrow{s_i} \cap \overrightarrow{s_j}). \tag{3.14}$$

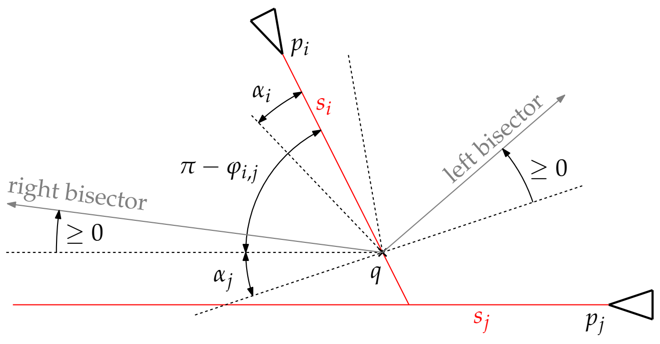


Figure 50: The wavefronts of Δ_i do not cause a crash of the reflex wavefront vertex from p_j.

Note that we may only consider indices i, j for which $\overrightarrow{s_i} \cap \overrightarrow{s_j}$ is not empty. If no such indices i, j exist then we set L to zero. Further, let us denote by $\varphi_{i,j} \in [0, \pi]$ the non-oriented angle spanned by v_i and v_j, with $\varphi_{i,j} = \varphi_{j,i}$. Next we define

$$\Phi := \min_{1 \leq i < j \leq n} \mathbb{R}^+ \cap \{\varphi_{i,j}, \pi - \varphi_{i,j}\}. \tag{3.15}$$

If the corresponding set it is empty — i.e. if all motorcycles drive on parallel tracks — then we set $\Phi := \pi/2$.

Lemma 3.5. *Let m_i denote a motorcycle that crashes into the motorcycle m_j. The wavefronts of Δ_i do not cause a split event for the reflex wavefront vertex u_j until time $\mu/4$ if $\lambda \geq 2/\min_k |v_k| \sin \Phi$.*

Proof. Since $\lambda \geq \frac{2}{|v_k| \sin \Phi}$ holds for any $1 \leq k \leq n$, it follows that

$$\sin \alpha_k \leq \frac{1}{|v_k|\lambda} \leq \frac{1}{2} \sin \Phi \leq \sin \frac{\Phi}{2},$$

because sin is concave on $[0, \pi]$. By further noting that sin is monotone on $[0, \pi/2]$ we see that

$$\alpha_k \leq \frac{\Phi}{2} \qquad \forall\, 1 \leq k \leq n.$$

The motorcycle m_j does not also crash into m_i, since two motorcycles do not crash simultaneously into each other by assumption. If s_i and s_j are collinear then the assertion is either trivial or excluded by assumption. Without loss of generality, we may assume that s_i is right of $\overrightarrow{s_j}$, see Figure 50. We denote by q the endpoint of the reflex straight-skeleton arc that is incident to p_i. Let us consider the left (resp. right) bisector between the left (resp. right) arm of m_i and the right arm of m_j, starting from q.

By Lemma 3.4 it suffices to show that the two arms of m_i do not lead to a split event with u_j until time $\mu/4$: The two additional motorcycles from Δ_i stay within $p_i + D_\mu$. Hence, we only have to consider the arms of m_i.

We conclude the proof by showing that none of both bisectors intersects $\overrightarrow{s_j}$. Let us consider the right bisector. Recall that $\alpha_i, \alpha_j \leq \Phi/2$ and that $\pi - \varphi_{i,j} \geq \Phi$. In the extremal case,

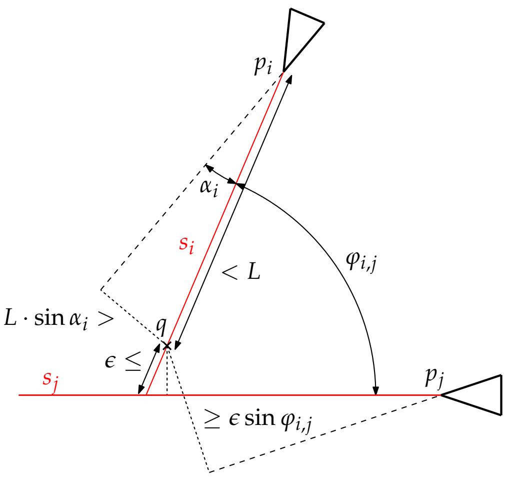


Figure 51: The point q is reached earlier by the left arm of m_i than by the right arm of m_j.

where equality is attained for all three inequalities, the right bisector is just parallel to s_j, but strictly right of $\overrightarrow{s_i}$. In all other cases the bisector rotates clockwise at q such that our assertion is true in general. Analogous arguments hold for the left bisector. Summarizing, the reflex wavefront vertex u_j does not lead to a split event with the wavefronts of Δ_i until time $\mu/4$. □

Lemma 3.6. *Let m_i denote a motorcycle crashing into the motorcycle m_j. For any $\epsilon > 0$ and*

$$\lambda \geq \frac{1}{\min_k |v_k| \cdot \sin \Phi} \cdot \max \left\{ 2, \frac{L}{\min\{\mu/4, \epsilon\}} \right\},$$

the trace s_i is covered up to a gap size ϵ by the reflex arc traced out by u_i.

Proof. We will prove the following: any point q on s_i, whose distance to the endpoint of s_i is at least ϵ, is reached by u_i no later than at time $\mu/4$, see Figure 51. We first show that until time $\mu/4$ the reflex wavefront vertex u_i may only cause a split event with the wavefronts of Δ_j. By Lemma 3.4, we know that until time $\mu/4$, the wavefronts of a triangle Δ_k could only cause a split event with u_i if s_k and s_i intersect. Hence, m_k crashed against s_i. However, by Lemma 3.5 it follows that u_i does not lead to a split event with the wavefronts from Δ_k.

W.l.o.g., we may assume that s_i lies on the right of $\overrightarrow{s_j}$. In order to show that u_i reaches q until time $\mu/4$, it suffices to prove that q has a smaller orthogonal distance to the left arm of m_i than to the right arm of m_j and that the orthogonal distance of q to the left arm of m_i is at most $\mu/4$.

The orthogonal distance of q to the left arm of m_i is at most $L \cdot \sin \alpha_i$. The orthogonal distance of q to the right arm of m_j is at least the orthogonal distance of q to s_j. However, this distance is at least $\epsilon \cdot \sin \varphi_{i,j}$. Summarizing, our assertion holds if

$$L \cdot \sin \alpha_i \leq \min\{\mu/4, \epsilon \sin \varphi_{i,j}\},$$

and thus

$$\lambda \geq \frac{L}{|v_i| \cdot \min\{\mu/4, \epsilon \sin \varphi_{i,j}\}}.$$

Our choice for λ fulfills this condition. The case where s_j and s_i are collinear such that m_i crashes at p_j is similar. The wavefront of Δ_j reaches q no later than at time $\epsilon/2$ and v_i reaches q in at most $\frac{L}{\lambda |v_i|}$ time. $\square$

Let us denote by $\mathcal{S}^*_\lambda(m_1, \ldots, m_n) \subset \mathcal{S}(G)$ the union of the reflex straight-skeleton arcs that are traced out by $u_1, \ldots, u_n$, where G is given as described above. Then we get the following corollary of Lemma 3.6:

Corollary 3.7.

$$\lim_{\lambda \to \infty} \mathcal{S}^*_\lambda(m_1, \ldots, m_n) = \mathcal{M}(m_1, \ldots, m_n)$$

This corollary also includes that a point q on a motorcycle trace s_i of a motorcycle m_i that never crashed, is covered by an arc of $\mathcal{S}^*_\lambda(m_1, \ldots, m_n)$ for large enough λ. However, this is easy to see by applying Lemma 3.4 and Lemma 3.5, and by finally finding λ large enough such that the point q is reached by u_i until time $\mu/4$.

3.4.2 Computing the motorcycle graph

In order to reduce the construction problem of motorcycle graphs to straight skeletons, we have to cope with the remaining gaps between the motorcycle traces s_i and the reflex arcs traced out by u_i, which exist for arbitrary large λ. Hence, the question remains whether m_i actually crashed into m_j (or vice versa), even if the gap between two reflex arcs that are traced out by u_i and u_j, is very small.

In order to decide whether the motorcycle m_i escapes or whether it crashes into a trace s_j, we determine λ large enough such that the following conditions are fulfilled:

- If m_i crashes into a trace s_j then the reflex wavefront vertex u_i leads to a split event until the time $\mu/4$ and the reflex arc that is traced out by u_i has an endpoint in a straight-skeleton face of an edge of Δ_j. (The vertex u_i causes a split event with the right arm of m_j if s_i is right of $\overrightarrow{s_j}$ and the left arm if s_i is left of $\overrightarrow{s_j}$.)
- If m_i escapes then the reflex wavefront vertex u_i did not lead to a split event until the time $\mu/4$.

The following lemma states that such λ exists and provides a sufficient bound.

Lemma 3.8. *Consider $\mathcal{S}(G)$ with*

$$\lambda \geq \frac{\max\left\{2, \frac{8L}{\mu}\right\}}{\min_k |v_k| \cdot \sin \Phi}.$$

Then m_i crashes into s_j if and only if u_i leads to a split event with the wavefront emanated by Δ_j until time $\mu/4$. In particular, m_i escapes if and only if u_i does not lead to a split event until time $\mu/4$.

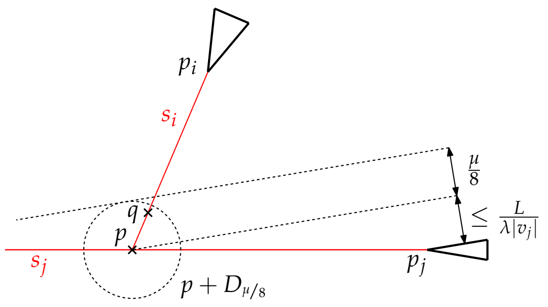


Figure 52: The reflex wavefront vertex u_i causes a split event until time $\mu/4$.

Proof. We distinguish two cases. First, suppose that the motorcycle m_i crashed into the trace s_j, see Figure 52. We may assume without loss of generality that s_i is right of $\overrightarrow{s_j}$. First we note that by our choice of λ we may apply Lemma 3.4. We denote by p the intersection $s_i \cap s_j$. Further, we set $\epsilon := \mu/8$, which allows us to apply Lemma 3.6, because

$$\max\left\{2, \frac{8L}{\mu}\right\} \geq \max\left\{2, \frac{L}{\min\{\mu/4, \epsilon\}}\right\}.$$

Thus, the endpoint q of the reflex arc that is traced out by u_i, has a distance of at most $\mu/8$ to p. On the other hand, u_j reaches p at time $L/\lambda|v_j|$ at the latest. We conclude that the right arm of m_j reaches q at time $L/\lambda|v_j| + \mu/8$ at the latest, which is bounded from above by

$$\frac{L \cdot \mu \cdot \min_k |v_k| \cdot \sin \Phi}{8 \cdot L \cdot |v_j|} + \frac{\mu}{8} \leq \frac{\mu}{4}$$

by our choice of λ. Summarizing, the point q is swept by the wavefront of the right arm of m_j and is reached by u_i until $\mu/4$ time. Hence, u_i must have caused a split event until the requested time by crashing into the wavefront of the right arm of m_j.

For the second case assume that m_i escapes. Lemma 3.4 and Lemma 3.5 imply that u_i does not lead to a split event until time $\mu/4$. □

The previous lemma enables us to compute the motorcycle graph by employing a straight skeleton algorithm. However, in order to apply the lemma, we need to compute appropriate values for L, Φ, μ in order to determine a sufficiently large λ. While L and Φ are already given independent of $\mathcal{M}(m_1, \ldots, m_n)$, the following lemma gives a formula for μ, for which the actual motorcycle graph is not needed to be known. (In the following lemma we take $d(\overrightarrow{s_i}, \emptyset)$ to be infinity.)

Lemma 3.9. *For any two disjoint motorcycle traces s_i and s_j the Minkowski sums $s_i + D_\mu$ and $s_j + D_\mu$ are disjoint for*

$$\mu := \frac{1}{3} \min_{1 \leq i,j,k \leq n} \mathbb{R}^+ \cap \{d(\overrightarrow{s_i}, p_j), d(\overrightarrow{s_i}, \overrightarrow{s_j} \cap \overrightarrow{s_k})\}.$$

Proof. In order to guarantee that the Minkowski sums are disjunct it suffices to show that μ is a lower bound of a third of the minimum of all pairwise infimum distances of disjunct traces s_i and s_j.

Let us consider two disjunct traces s_i and s_j. We choose two points $q_i \in s_i, q_j \in s_j$ for which $d(s_i, s_j) = d(q_i, q_j)$ holds. We may assume that either q_i is an endpoint of s_i or q_j is an endpoint of s_j, because the infimum distance is not uniquely attained for two interior points of s_i and s_j. (If s_k is not bounded, the only endpoint is p_k.) If q_j is the start point of s_j then we have $d(s_i, s_j) = d(s_i, p_j) \geq 3\mu$. If q_j is the opposite endpoint of s_j — and hence s_j is a segment — then s_j crashed into some other motorcycle trace. Hence, there is a trace s_k such that $q_j = s_j \cap s_k$. Again we get $d(s_i, s_j) = d(s_i, q_j) \geq 3\mu$. Analogous arguments hold if q_i is an endpoint of s_i. □

After computing appropriate values for L, Φ and μ for a set of motorcycles $m_1, \ldots, m_n$, we can determine a sufficiently large λ and build the input graph G by constructing the triangles $\Delta_1, \ldots, \Delta_n$ as described. After computing the straight skeleton $\mathcal{S}(G)$, we determine the length of each trace by applying the conditions listed in Lemma 3.8.

3.4.3 Constructing the straight skeleton is P-complete

The concept of P-completeness is similar to the concept of NP-completeness. A problem A from P is said to be P-complete under NC-reductions if any problem in P can be reduced to A in NC time. The complexity class NC comprises all problems that can be efficiently solved in parallel. That is, they can be solved in poly-logarithmic time complexity using a polynomial number of processors. Hence, all P-complete problems cannot be efficiently solved using parallel computers, unless NC = P. In other words, P-complete problems are inherently sequential, provided that NC $\neq$ P. If the latter condition would be wrong then every problem solvable in polynomial time could also be solved efficiently in parallel. To show this it would suffice to find a single P-complete problem that can be solved in NC time. However, is is commonly assumed that NC $\neq$ P.

In order to show that a specific problem A is P-complete it suffices to reduce any P-complete problem to A in NC time. Alternatively, one could also seek for an according LOGSPACE reduction, because LOGSPACE $\subset$ NC. For further details on P-completeness we refer to the book by Greenlaw, Hoover and Ruzzo [GHR95].

Atallah et al. [ACG93] described a framework for geometric reductions of the P-complete Planar Circuit Value problem and used it to prove the P-completeness of several geometric problems. For their framework, they consider specific binary circuits which are organized in layers: an input layer at the top and alternating routing layers and logic layers from top to bottom. The routing layers and logic layers consist of rows of components occupying non-overlapping rectangles. A logic layer consists of a row of $\vee$-gates and a row of $\neg$-gates and a routing layer comprises rows of routing components, namely left shifts, right shifts, fan-out gates and vertical wires. The binary circuit has a single output gate and the Circuit Value problem asks for the output value of this gate when a specific input vector is presented to the input gates. Atallah et al. proved that the Circuit Value problem for circuits of such a specific layout is still P-complete.

Investigating the P-completeness of geometric problems often requires the availability of exact geometric computations, which are not in NC. For instance, they mention the problem of determining whether four points lie on a circle, which is an essential predicate when computing Voronoi diagrams. In order to investigate the P-completeness of geometric problems, Attalah et al. [ACG93] propose that the answers to basic geometric queries are provided by an oracle.

A basic building block for showing that the straight skeleton is P-complete is the construction of the triangles Δ_i. Assume p_i, α_i, v_i, μ, and λ are given. We further assume that an oracle determines the intersection points of two circles with given centers and radii. Then, we can construct Δ_i as follows. We first compute the point $q_i = p_i + \lambda v_i$, which is the position of m_i at time one, see Figure 48. Then we construct the circle C_1 with $[p_i, q_i]$ as diameter and the circle C_2 centered at p_i with radius 1. The two circles C_1, C_2 intersect at two points, say a_i, b_i. The triangle Δ_i^* with vertices a_i, b_i, q_i is an isosceles triangle with angle $2\alpha_i$ at q_i and therefore similar to Δ_i. (Note that p_i, a_i, q_i form a right-angled triangle within Thales' circle C_1.) The length of the arms of Δ_i^* at q_i are at most $\lambda|v_i|$. By scaling the triangle by the factor $\mu/2\lambda|v_i|$ and by translating it accordingly, we get a triangle with the desired geometry. (Strictly speaking, the arms of the constructed triangle are a bit shorter than $\mu/2$, but this is only to our advantage.)

Eppstein and Erickson [EE99] proved that the computation of the motorcycle graph is P-complete by presenting a LOGSPACE-reduction of the Circuit Value problem to the computation of the motorcycle graph. Eppstein and Erickson demonstrated how to translate the Circuit Value problem to the motorcycle graph construction problem by simulating each gadget of the binary circuit using motorcycles. The values 1 and 0 on a wire are represented by the presence or absence of a motorcycle on a track. The original question for the output value of a particular gate of the circuit can be translated to the question whether a specific motorcycle crashes until some distance from its start point. In other words, Eppstein and Erickson proved that the decision problem whether a specific motorcycle crashes until some distance from its start point is P-complete.

Lemma 3.10. *The construction of the straight skeleton of a planar straight-line graph is P-complete under LOGSPACE-reductions.*

Proof. Eppstein and Erickson reduced the Circuit Value problem to a specific motorcycle graph problem. The next step is to reduce the motorcycle graph problem to the straight-skeleton problem: we construct a suitable input graph G that allows us to apply Lemma 3.8 in order to decide whether a specific motorcycle crashes until some distance from its start point.

According to [EE99] all $O(1)$ different types of motorcycle gadgets are arranged in an $n \times n$ grid. Each gadget takes constant space and consists of $O(1)$ motorcycles. To determine a sufficiently large λ, we need bounds on L, Φ and μ. An upper bound on L is the length of the diagonal of the $n \times n$ grid. Further, $\sin \Phi \geq 1/2$ since the direction angles of the motorcycles are all multiples of $\pi/4$. A lower bound on μ can be obtained by applying Lemma 3.9 on each gadget independently and taking the minimum among them. Finally, we build G by modeling each motorcycle (independently from each other) as an isosceles triangle, as described in Section 3.4.1. □

We can easily extend the construction of G to form a polygon with holes, by adding a sufficiently large bounding box to G. As remarked in [EE99], only one motorcycle m may leave the bounding box B of the $n \times n$ grid. The motorcycle m encodes the output of the

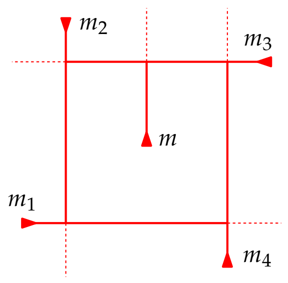


Figure 53: These motorcycle traces cannot be approximated easily by a straight skeleton of a simple polygon.

binary circuit by leaving B if the circuit evaluates to 1 and by crashing within B if the circuit evaluates to 0.

By Lemma 3.8 the reflex wavefront vertex v, which corresponds to m, encodes the output of the binary circuit by leading to a split event until time $\mu/4$ if and only if the circuit evaluates to 0. Lemma 3.4 implies that the wavefront vertices stay within $B + D_\mu$, except possibly v. Hence, we could enlarge B by 2μ at each side and add it to G such that the wavefronts of B do not interfere with the wavefronts of the triangles until time $\mu/4$, except for v. Still, we can determine the output of the binary circuit by checking whether the reflex straight-skeleton arc that corresponds to v, ends within $B + D_\mu$ until time $\mu/4$. (Recall that the end of a reflex straight-skeleton arc marks the place where the reflex wavefront vertex led to a split event.)

Corollary 3.11. *The construction of the straight skeleton of a polygon with holes is* P*-complete under LOGSPACE-reductions.*

Unfortunately, our P-completeness proof cannot be applied easily to simple polygons. Consider the five motorcycles depicted in Figure 53. A simple polygon, whose straight skeleton would approximate the motorcycle traces, would need to connect the start points of m and $m_1, \dots, m_4$. But in order to decide where the red square, formed by the traces of $m_1, \dots, m_4$, can be penetrated by the polygon, while avoiding to stop a motorcycle too early, it would be necessary to know that a specific motorcycle m_i crashes into a specific motorcycle m_j. However, deciding whether a specific motorcycle crashes does not seem much easier than computing the whole motorcycle graph. Hence, it remains open whether the computation of straight skeletons of polygons is P-complete.

4 CONCLUDING REMARKS

The investigations done in this thesis are driven by the lack of an efficient implementation of straight skeletons for real-world purposes that stands opposite to the large number of different industrial and academical applications. At the moment, the state-of-the-art straight-skeleton implementation is shipped with the CGAL library. However, our experiments illustrate that the CGAL implementation is only applicable to real-world problems in a limited manner: Both, the runtime and the memory footprint of the CGAL implementation increase at least quadratically with the input size. While time is more or less an infinite resource, the opposite is true for space. The quadratic memory consumption is an issue that basically renders the implementation in CGAL inapplicable to datasets containing more than ten thousand vertices.

We started our investigations towards an efficient straight-skeleton algorithm with an analysis of the triangulation-based approach by Aichholzer and Aurenhammer [AA98] in Section 2.2. The gap between $\Omega(n^2)$ and $O(n^3)$ for the worst-case number of flip events leads to an open question regarding the time complexity of the algorithm. We present different results regarding this gap and finally prove the existence of Steiner triangulations that are free of flip-events. This result is primarily of theoretical interest, since we use the straight skeleton for the placement of the Steiner points. However, the fact that flip-event-free Steiner triangulations exist led to a novel wavefront-type straight-skeleton algorithm for simple non-degenerate polygons which is based on the motorcycle graph in Section 2.3.

In order to make this algorithm applicable to real-world input we needed to remove the non-degeneracy assumption on the input. In Section 2.4, we carefully generalized the motorcycle graph in order to reflect so-called vertex events for straight skeletons. That is, the simultaneous crash of two or more motorcycles into each other may require to start an additional motorcycle. We demand two essential geometric properties for the generalized motorcycle graph: (i) $\mathcal{M}(G)$ needs to cover the reflex arcs of $\mathcal{S}(G)$, also in the presence of vertex events, and (ii) $G + \mathcal{M}(G)$ needs to induce a convex tessellation of the plane. These two properties allowed us to extend our algorithm to arbitrary planar straight-line graphs G in Section 2.5. Furthermore, the generalization of the motorcycle graph leads to an alternative characterization of the straight skeleton of arbitrary planar straight-line graphs by extending the characterization of Cheng and Vigneron and it motivated a straight-skeleton algorithm that is based on 3D graphics hardware.

The wavefront-type straight-skeleton algorithm presented in Section 2.5 is easy to implement, has a worst-case runtime of $O(n^2 \log n)$ and operates in $O(n)$ space. The resulting implementation Bone uses ordinary double-precision floating-point arithmetic and accepts planar straight-line graphs as input. Our experiments showed an $O(n \log n)$ runtime on 13 500 datasets of different types. This constitutes an improvement of a linear factor in time and space compared to the implementation in CGAL, which only accepts polygons with holes as input. On datasets with several thousand vertices, our implementation is up to two orders of magnitude faster and we need less than 100 megabytes of memory instead of several gigabytes. Moreover, due to our low memory requirements, we are able to compute the straight skeleton of datasets with a million vertices. These circumstances make Bone the fastest straight-skeleton implementation at the moment and the first extensively tested implementation that is capable to process planar straight-line graphs originating from real-world applications.

In order to push Bone to industrial strength, we plan to enhance the numerical stability in the presence of parallel wavefronts that collapse simultaneously. Detecting the simultaneous collapse of larger parts of the wavefront is simplified by exploiting the convex tessellation induced by the motorcycle graph. From an abstract point of view we gain from the fact that the motorcycle graph already provides a sufficient amount of information on the topology of the straight skeleton. Bone is currently extended to use different numerical backends by colleagues, including the arbitrary precision library MPFR [MPF] and the exact geometric computation library CORE [Cor]. The necessary work for MPFR is almost finished and preliminary runtime tests with a floating-point precisions of 212 and 1000 bits show a drop in performance by a factor of approximately 10–20, which is still reasonable for real-world applications.

Even though Bone exhibits an $O(n \log n)$ runtime in practice, the gap between the lower bound of $\Omega(n \log n)$ and the theoretically fastest algorithm by Eppstein and Erickson [EE99], with a theoretical worst-case time complexity of $O(n^{17/11+\epsilon})$, remains open. If the motorcycle graph $\mathcal{M}(G)$ is known, the algorithm behind Bone has an $O((n+k)\log n)$ time complexity, where $k \in O(nr)$ denotes the number of switch events and $r \in O(n)$ denotes the number of reflex wavefront vertices. In order to obtain a worst-case runtime of $O(n \log n)$ it would be necessary to bound the number k of switch events to $O(n)$. One approach towards this goal could be the introduction of additional motorcycles that are launched from convex wavefront vertices of $\mathcal{W}(G, 0)$ such that the number of interactions between moving Steiner vertices and convex wavefront vertices is reduced to $O(n)$. However, in order to extend the basic algorithm behind Bone accordingly, we would require that any extension of the motorcycle graph $\mathcal{M}(G)$ by additional motorcycles needs to maintain the validity of Lemma 2.25 and Theorem 2.26: $G + \mathcal{M}(G)$ needs to induce a convex tessellation and the motorcycle traces need to cover the reflex straight-skeleton arcs.

We also look forward to generalize the algorithms behind Bone in order to compute weighted straight skeletons. In principal, wavefront-type algorithms tend to have a straightforward generalization from the unweighted straight skeleton to the weighted counterpart. However, in order to generalize Bone to weighted straight skeletons, it is necessary to adapt the definition of the motorcycle graph such that Lemma 2.25 and Theorem 2.26 still hold in the weighted case. First of all, the speed and direction of each motorcycle is given by Lemma 1.11 in order to match the speeds of the corresponding reflex wavefront vertices. Assume for a moment that we simply transfer the rules for launching new motorcycles from Section 2.4.1. If we do so, Lemma 2.25 remains true for the Case (d) in Figure 31, but not for Case (c). Note that if a vertex event happens then the reflex straight-skeleton arcs do not need to tessellate a local disk into convex slices. That is, the lower chains of the faces do not need to be convex. In fact, the faces do not even need to be monotone. In order to fix Lemma 2.25, we could launch a second motorcycle that continues the movement of the right ancestor, as in Case (d). However, the question remains whether the number of motorcycles is still in $O(n)$ after this adaption. In the next step, we need to guarantee that Theorem 2.26 remains true for this generalized motorcycle graph. Note that again the tilted motorcycle traces $\hat{m}_1, \ldots, \hat{m}_k$ with the same right arm e lie on the supporting plane of the tilted face $\hat{f}(e)$ and at least the Step (i) in the proof of Theorem 2.26 remains true. We also want to note that a generalization of Theorem 2.26 could also provide a lower envelope characterization of the weighted straight skeleton, which does not exist so far. Recall that Eppstein and Erickson [EE99] showed that the trivial generalization does not work, see Figure 16.

In Chapter 3 we took a closer look at the motorcycle graph. In order to compute straight skeletons fast in practice by means of Bone, we require an implementation for the generalized motorcycle graph that performs well on real-world data. We started with a stochastic analysis of the average trace length in a motorcycle graph in Section 3.2. It turned out that if the motorcycles are distributed uniformly within the unit square then we can expect an average trace length of $\Theta(1/\sqrt{n})$. In other words, if we impose a regular $\sqrt{n} \times \sqrt{n}$ grid on the unit square then we expect that a motorcycle crosses $O(1)$ grid cells on average. This fact motivated a simple pragmatic approach: we enhanced the straight-forward approach based on a priority-queue with geometric hashing in Section 3.3. This measurement reduces the runtime from $O(n^2 \log n)$ to $O(n \log n)$ for input data where the motorcycles are distributed uniformly enough. We performed extensive runtime tests with our implementation Moca. For the vast majority of our datasets Moca runs in $O(n \log n)$ time, which is one of the key reasons for the good overall performance of our straight-skeleton code Bone. Finally, we also used Moca in order to substantiate the theoretical predictions provided by the stochastic analysis.

We plan to further boost the performance of Moca by implementing the $O(n \log n)$ algorithm to compute the motorcycle graph outside the convex hull of the start points, see Section 3.3.4. Furthermore, in order to make the runtime performance of Moca more robust against clustered start points, one may also replace the ordinary rectangular grid by quad trees, which allows a non-uniform tessellation of the plane. Additionally, quad trees would enable us to dynamically increase the tessellation depth at regions where motorcycles accumulate during the propagation process.

Our final contribution concerns the geometric relation of motorcycle graphs and straight skeletons. By Theorem 2.26 we know that the reflex straight-skeleton arcs of $\mathcal{S}(G)$ approximate the motorcycle traces of $\mathcal{M}(G)$ up to a specific extent. Furthermore, one observers that in general the gap between the reflex arcs and the motorcycle traces decreases if the speed of the motorcycle increases. In Section 3.4, we first show that we can construct a planar straight-line graph G whose straight skeleton approximates the motorcycle graph. Based on this result, we present a simple algorithm that computes the motorcycle graph using the straight skeleton. Finally, we show that the resulting algorithm admits a LOGSPACE-reduction of the motorcycle graph problem to the straight-skeleton problem. Consequently, the P-completeness of the motorcycle graph by Eppstein and Erickson [EE99] implies the P-completeness of the straight-skeleton problem for planar straight-line graphs and for polygons with holes. We want to note that Eppstein and Erickson [EE99] were the first to mention that straight skeleton is P-complete, but no proof was given. The P-completeness of straight skeletons has important practical implications: no efficient parallel algorithms exist to compute straight skeletons of planar straight-line graphs and polygons with holes, provided that $\mathsf{P} \neq \mathsf{NC}$. We also note that it is desirable to find an efficient sequential reduction of motorcycle graphs to straight skeletons in order to transfer lower bounds from motorcycle graphs to straight skeletons in the sequential manner.

A NOTATION

$A+B$	The Minkowski sum $A+B=\{x+y\ :\ x\in A, y\in B\}$ for two point sets A and B.
$\overline{pq}, \overline{e}, \overline{\hat{f}(e)}$	The supporting line of the points p,q resp. the edge e and the supporting plane of the lifted face $\hat{f}(e)$.
$[pq]$	The straight-line segment between two points p and q.
$\hat{a}, \hat{s}, \hat{m}, \hat{f}(e)$	The lifted counterpart of the arc a, motorcycle trace s, motorcycle m, straight-skeleton face $f(e)$ in the terrain model.
$d(p,q)$	The Euclidean distance between two points p and q.
$d(A,B)$	The infimum distance $\inf_{x\in A, y\in B} d(x,y)$ for two point sets A and B.
D_r	A disk with radius r and the origin as center.
$e(t)$	The union of the straight-line segments occupied by the wavefront edge e at time t, see Definition 2.2.
$\overline{e(t)}$	The supporting line of $e(t)$. If e is emanated by a terminal vertex or an isolated vertex v of G then we define $\overline{e(0)} := \lim_{t\searrow 0}\overline{e(t)}$, see Definition 2.2.
$f(e)$	The straight-skeleton face of the wavefront edge e, see Definition 1.1.
G	A planar straight-line graph (with no isolated vertices).
$\mathcal{M}(m_1,\ldots,m_n)$	The motorcycle graph of the motorcycles $m_1,\ldots,m_n$, see Definition 1.8.
$\mathcal{M}(P)$	The motorcycle graph induced by the simple polygon P, see Definition 1.9.
$\mathcal{M}(G)$	The motorcycle graph induced by a planar straight-line graph G, see Definition 2.23.
P	A simple polygon in the plane (with holes if mentioned explicitly).
$\mathcal{S}(P)$	The straight skeleton of a simple polygon P, only considered within the polygon P, see Definition 1.1.
$\mathcal{S}(G)$	The straight skeleton of a planar straight-line graph G, see Section 1.2.2.
$\mathcal{T}(G)$	The terrain model corresponding to $\mathcal{S}(G)$, see Definition 1.5.
$\mathcal{W}(G,t)$	The wavefront of G at time $t\geq 0$, see Definition 1.4.

B EXAMPLES

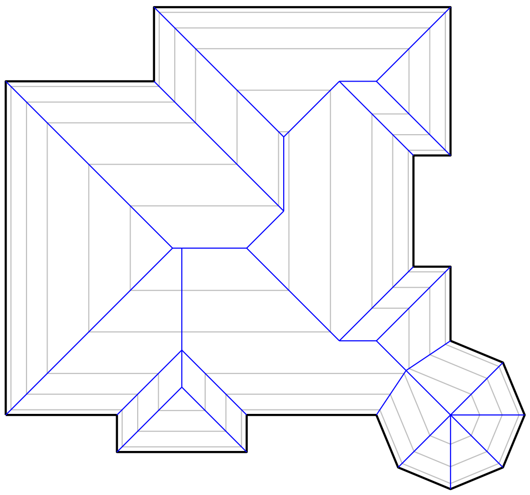


Figure 54: The straight skeleton and offset curves of the polygon that forms the walls of the house that is shown in Figure 8.

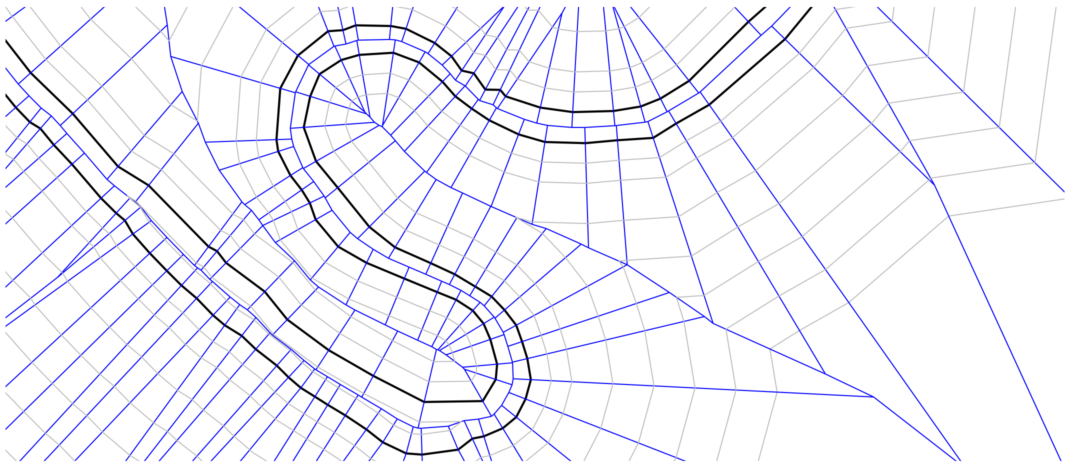


Figure 55: The straight skeleton and offset curves of the river bank of the Danube in Figure 9.

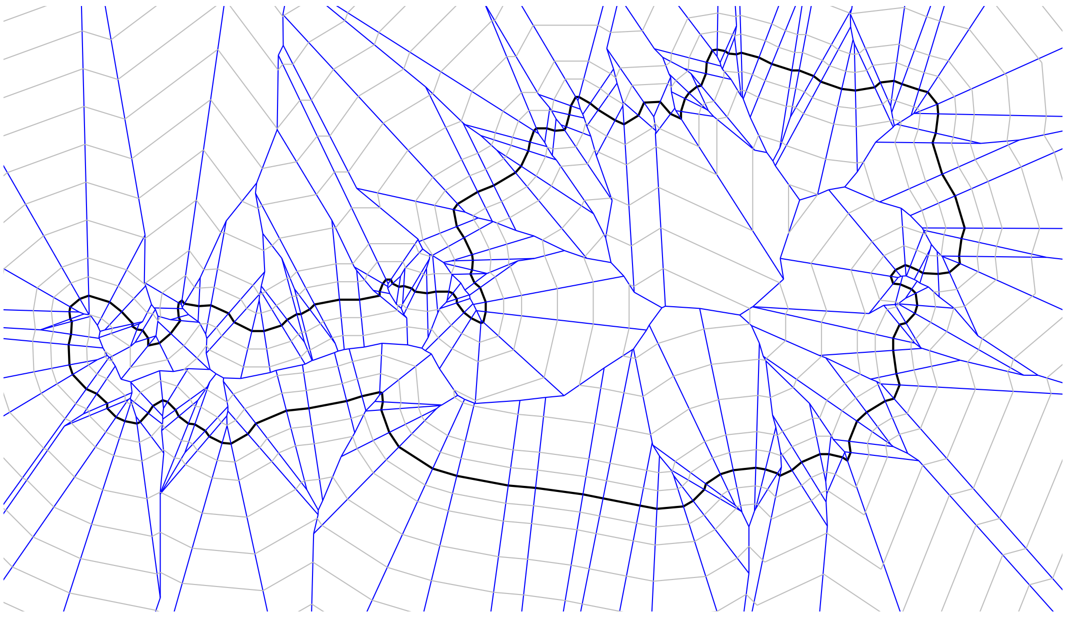


Figure 56: The straight skeleton and offset curves of the outline of Austria.

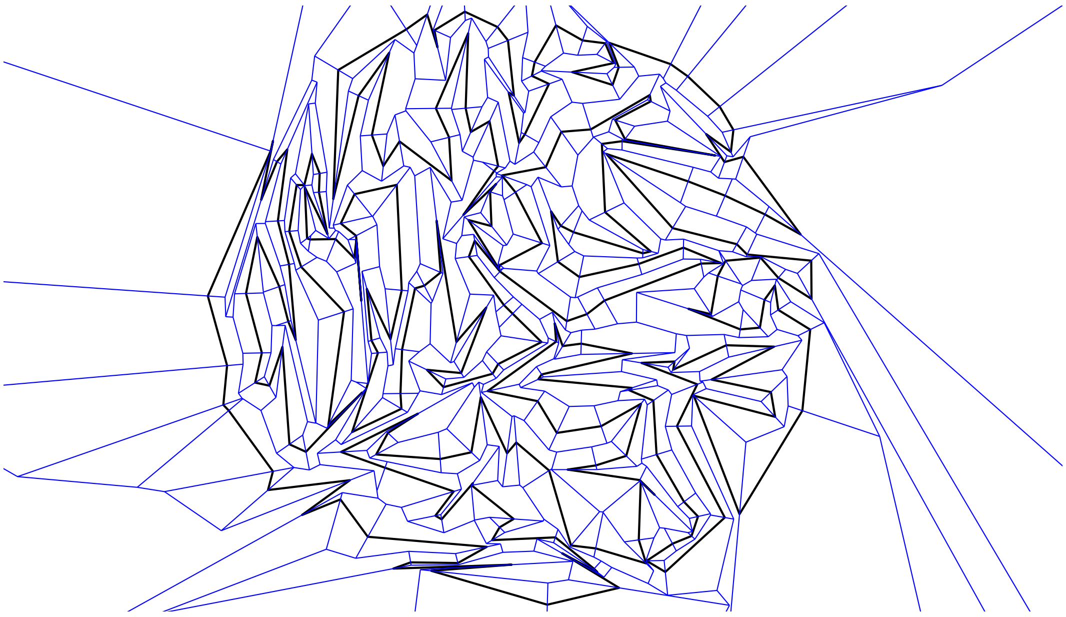


Figure 57: The straight skeleton of a random polygon generated by RPG.

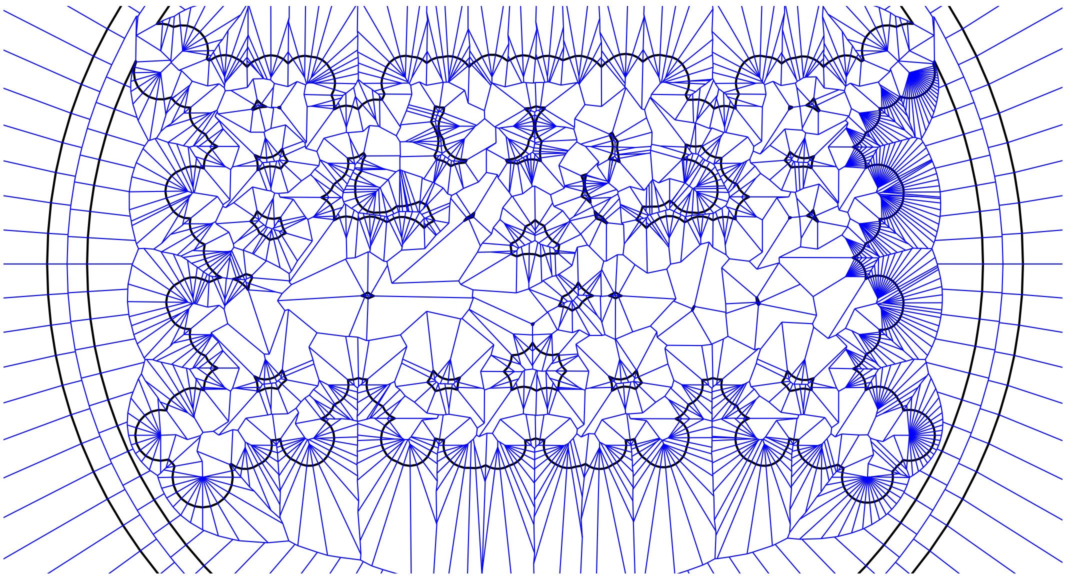


Figure 58: The straight skeleton of a polygon with holes.

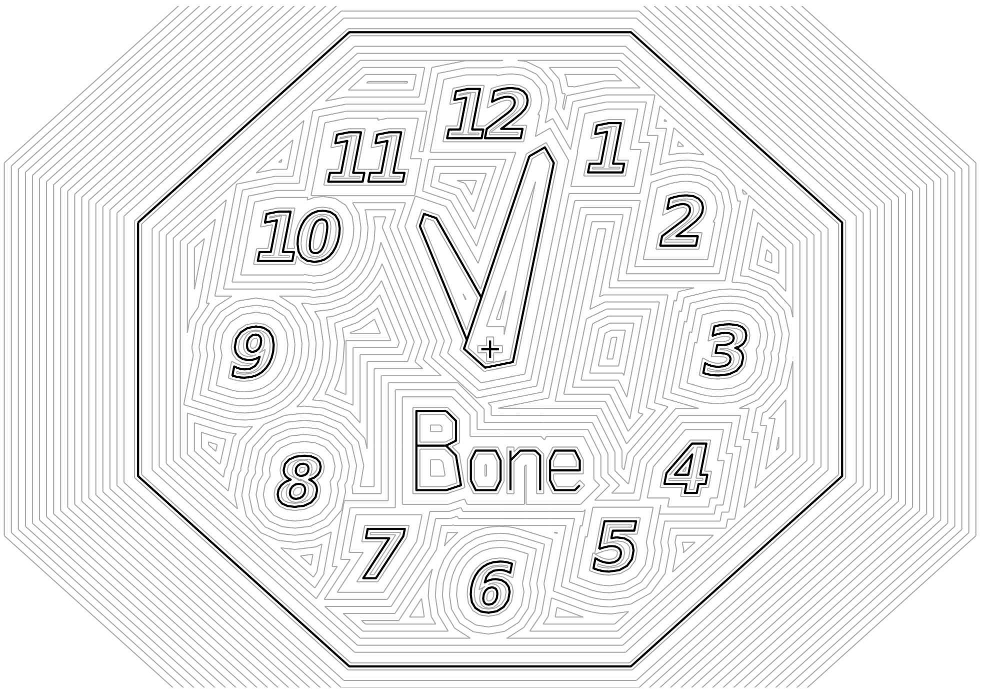


Figure 59: Offset curves based on the straight skeleton of a planar straight-line graph.

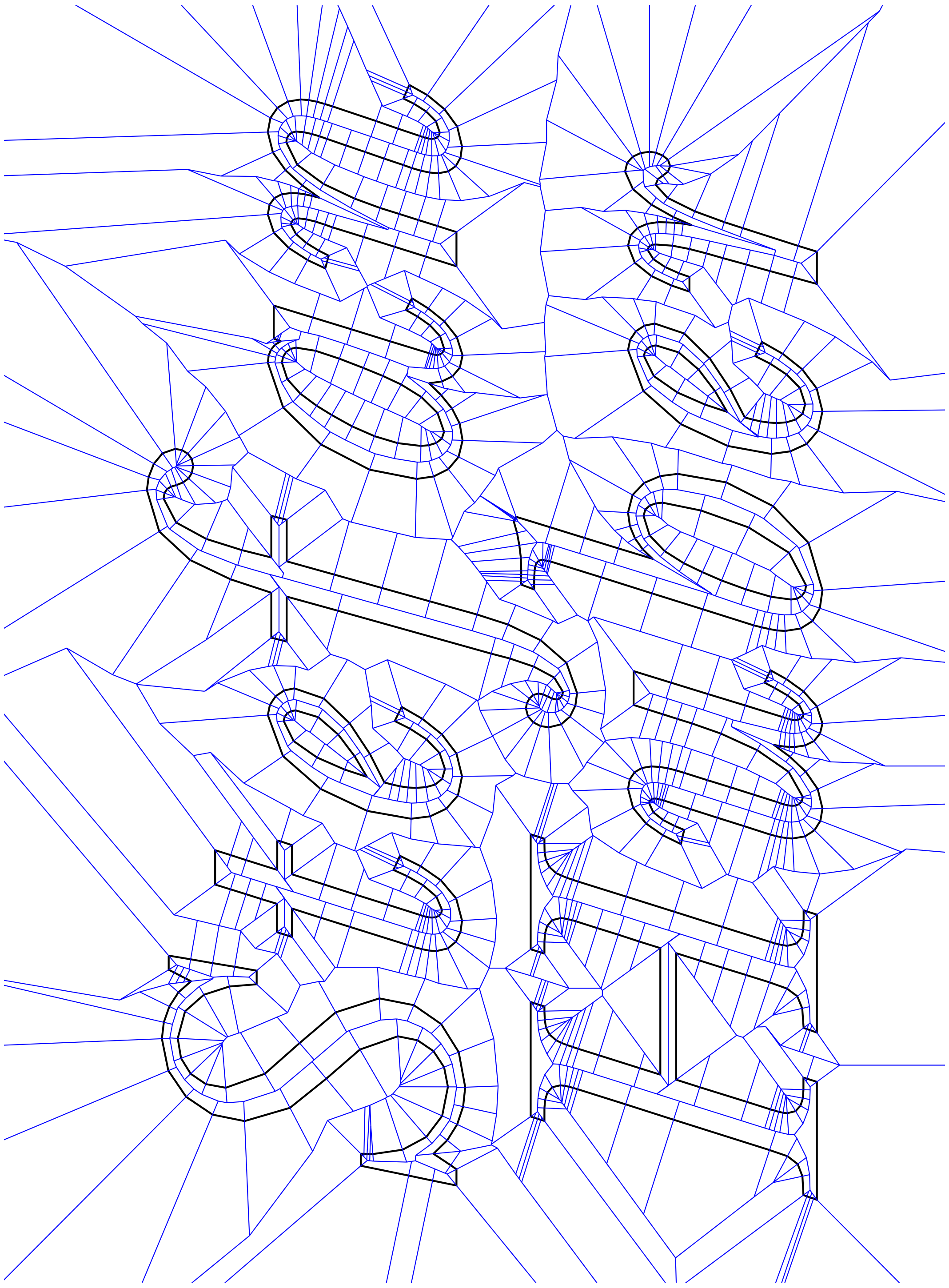


Figure 60: The straight skeleton of a font outline.

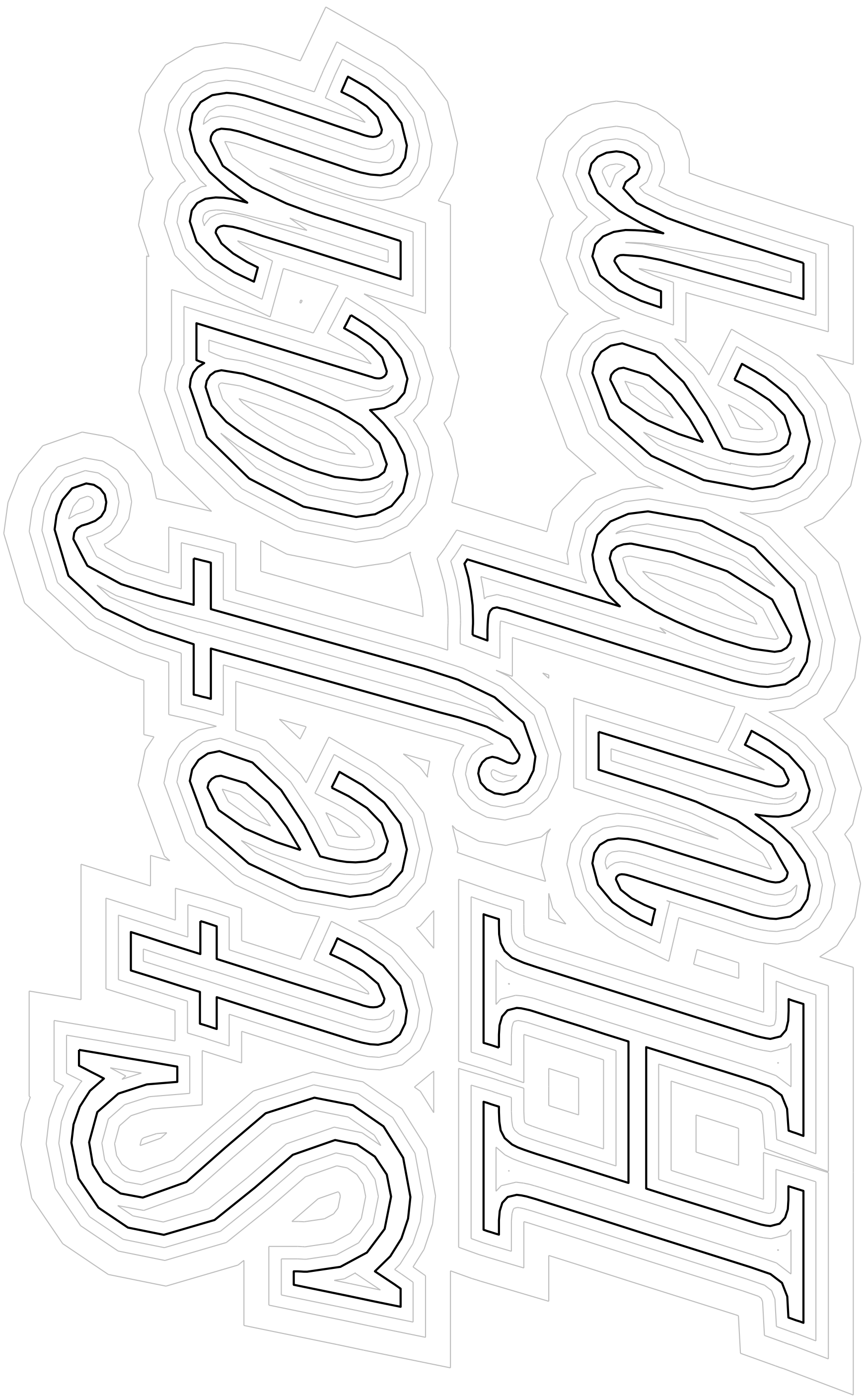


Figure 61: Offset curves of a font outline.

BIBLIOGRAPHY

[AA96] O. Aichholzer and F. Aurenhammer. Straight Skeletons for General Polygonal Figures. In *Proc. 2nd Annu. Internat. Conf. Comput. Combinatorics*, volume 1090 of *Lecture Notes Comput. Sci.*, pages 117–126. Springer-Verlag, 1996.

[AA98] O. Aichholzer and F. Aurenhammer. Straight Skeletons for General Polygonal Figures in the Plane. In A.M. Samoilenko, editor, *Voronoi's Impact on Modern Science, Book 2*, pages 7–21. Institute of Mathematics of the National Academy of Sciences of Ukraine, Kiev, Ukraine, 1998.

[AAAG95] O. Aichholzer, D. Alberts, F. Aurenhammer, and B. Gärtner. Straight Skeletons of Simple Polygons. In *Proc. 4th Internat. Symp. of LIESMARS*, pages 114–124, Wuhan, P.R. China, 1995.

[AAP04] O. Aichholzer, F. Aurenhammer, and B. Palop. Quickest Paths, Straight Skeletons, and the City Voronoi Diagram. *Discrete Comput. Geom.*, 31(1):17–35, 2004.

[ACG93] M.J. Atallah, P.B. Callahan, and M.T. Goodrich. P-complete Geometric Problems. *Internat. J. Comput. Geom. Appl.*, 3(4):443–462, 1993.

[AE99] P. K. Agarwal and J. Erickson. Geometric Range Searching and Its Relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 223 of *Contemporary Mathematics*, pages 1–56. American Mathematical Society, 1999.

[AGSS87] A. Aggarwal, L.J. Guibas, J. Saxe, and P. Shor. A Linear Time Algorithm for Computing the Voronoi Diagram of a Convex Polygon. In *Proc. 19th Annu. ACM Sympos. Theory Comput.*, pages 39–45, 1987.

[AH96] T. Auer and M. Held. Heuristics for the Generation of Random Polygons. In *Proc. Canad. Conf. Comput. Geom. (CCCG'96)*, pages 38–44, Ottawa, Canada, Aug 1996. Carleton University Press.

[AM94] P. Agarwal and J. Matoušek. On Range Searching with Semialgebraic Sets. *Discrete Comput. Geom.*, 11:393–418, 1994.

[AS95] H. Alt and O. Schwarzkopf. The Voronoi Diagram of Curved Objects. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 89–97, Vancouver, BC, Canada, 1995.

[BEGV08] G. Barequet, D. Eppstein, M. T. Goodrich, and A. Vaxman. Straight Skeletons of Three-Dimensional Polyhedra. In *Proc. 16th Annu. Europ. Symp. Algorithms (ESA '08)*, pages 148–160, Karlsruhe, Germany, Sep 2008.

[BGLSS04] G. Barequet, M.T. Goodrich, A. Levi-Steiner, and D. Steiner. Contour Interpolation by Straight Skeletons. *Graph. Models*, 66(4):245–260, 2004.

[Ble] Blender. `http://www.blender.org/`. last checked on May, 2011.

[BO79] J. Bentley and T. Ottmann. Algorithms for Reporting and Counting Geometric Intersections. *IEEE Trans. Comput.*, C-28:643–647, 1979.

[Cac04] F. Cacciola. A CGAL Implementation of the Straight Skeleton of a Simple 2D Polygon with Holes. In *2nd CGAL User Workshop*, Polytechnic Univ., Brooklyn, New York, USA, June 2004.

[CEG$^+$91] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray Shooting in Polygons Using Geodesic Triangulations. In *Proc. 18th Internat. Colloq. Automata Lang. Program.*, number 510 in Lecture Notes Comput. Sci., pages 661–673. Springer-Verlag, 1991.

[CGA] CGAL — Computational Geometry Algorithms Library. `http://www.cgal.org`. accessed on May, 2011.

[Cha91] B. Chazelle. Triangulating a Simple Polygon in Linear Time. *Discrete Comput. Geom.*, 6:485–524, 1991.

[Cha93] B. Chazelle. Cutting Hyperplanes for Divide-and-Conquer. *Discrete Comput. Geom.*, 9:145–158, Apr 1993.

[Cha04] B. Chazelle. Cuttings. In *Handbook of Data Structures and Applications*, pages 25.1–25.10. 1 edition, 2004. Chapman and Hall/CRC Press.

[Cor] CORE library project. `http://cs.nyu.edu/exact/core_pages/`. accessed on May, 2011.

[CRU89] J. Czyzowicz, I. Rival, and J. Urrutia. Galleries, Light Matchings and Visibility Graphs. In *Proc. 1st Workshop Algorithms Data Struct.*, pages 316–324, Ottawa, Canada, Aug 1989.

[CSW99] 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.

[CV02] S.-W. Cheng and A. Vigneron. Motorcycle Graphs and Straight Skeletons. In *Proc. 13th ACM-SIAM Sympos. Discrete Algorithms*, pages 156–165, San Francisco, CA, USA, 2002.

[CV07] S.-W. Cheng and A. Vigneron. Motorcycle Graphs and Straight Skeletons. *Algorithmica*, 47:159–182, Feb 2007.

[DDL98] E. D. Demaine, M. L. Demaine, and A. Lubiw. Folding and Cutting Paper. In *Revised Papers from the Japan Conference on Discrete and Computational Geometry (JCDCG'98)*, volume 1763 of *Lecture Notes Comput. Sci.*, pages 104–117, Tokyo, Japan, Dec 1998.

[DDLS05] E. D. Demaine, M. L. Demaine, J. F. Lindy, and D. L. Souvaine. Hinged Dissection of Polypolyhedra. In *Proc. 9th Workshop Algorithms Data Struct. (WADS 2005)*, volume 3608 of *Lecture Notes Comput. Sci.*, pages 205–217, Waterloo, Ontario, Canada, Aug 2005.

[DDM00] 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. Theory and Appl.*, 16(1):3–21, May 2000.

[DMN$^+$10] G. K. Das, A. Mukhopadhyay, S. C. Nandy, S. Patil, and S. V. Rao. Computing the Straight Skeleton of a Monotone Polygon in $O(n \log n)$ Time. In *Proc. 22nd Canad. Conf. Comput. Geom. (CCCG 2010)*, pages 207–210, Winnipeg, Canada, Aug 2010.

[DO07] E. D. Demaine and J. O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, Jul 2007.

[EE99] 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(4):569–592, 1999.

[EGKT08] D. Eppstein, M.T. Goodrich, E. Kim, and R. Tamstorf. Motorcycle Graphs: Canonical Quad Mesh Partitioning. *Comput. Graph. Forum*, 27(5):1477–1486, Sep 2008.

[Epp00] D. Eppstein. Fast hierarchical clustering and other applications of dynamic closest pairs. *J. Exp. Algorithmics*, 5, Dec 2000.

[FO99] P. Felkel and Š. Obdržálek. Improvement of Oliva's Algorithm for Surface Reconstruction from Contours. In *Proc. 15th Spring Conf. Comput. Graphics*, pages 254–263, Budmerice, Slovakia, Apr 1999.

[For00] Steven Fortune. Introduction. *Algorithmica*, 27(1):1–4, 2000.

[GHR95] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, Apr 1995.

[GLI] GNU C Library. `http://www.gnu.org/software/libc`. accessed on May, 2011.

[Hav05] S. Havemann. *Generative Mesh Modeling*. PhD thesis, TU Braunschweig, Braunschweig, Germany, 2005.

[HCK$^+$99] K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha. Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware. In *Comput. Graphics (SIGGRAPH '99 Proc.)*, pages 277–286, Los Angeles, CA, Aug 1999.

[Hel91] M. Held. *On the Computational Geometry of Pocket Machining*, volume 500 of *Lecture Notes Comput. Sci.* Springer-Verlag, June 1991. ISBN 3-540-54103-9.

[Hel94] M. Held. On Computing Voronoi Diagrams of Convex Polyhedra by Means of Wavefront Propagation. In *Proc. 6th Canad. Conf. Comput. Geom. (CCCG'94)*, pages 128–133, Saskatoon, Saskatchewan, Canada, Aug 1994.

[Hel01] M. Held. VRONI: An Engineering Approach to the Reliable and Efficient Computation of Voronoi Diagrams of Points and Line Segments. *Comput. Geom. Theory and Appl.*, 18(2):95–123, Mar 2001.

[HH09a] M. Held and S. Huber. Topology-Oriented Incremental Computation of Voronoi Diagrams of Circular Arcs and Straight-Line Segments. *Comput. Aided Design*, 41(5):327–338, May 2009.

[HH09b] S. Huber and M. Held. A Practice-Minded Approach to Computing Motorcycle Graphs. In *Proc. 25th Europ. Workshop Comput. Geom.*, pages 305–308, Brussels, Belgium, Mar 2009.

[HH10a] S. Huber and M. Held. Computing Straight Skeletons of Planar Straight-Line Graphs Based on Motorcycle Graphs. In *Proc. 22nd Canad. Conf. Comput. Geom. (CCCG 2010)*, pages 187–190, Winnipeg, Canada, Aug 2010.

[HH10b] S. Huber and M. Held. Straight Skeletons and their Relation to Triangulations. In *Proc. 26th Europ. Workshop Comput. Geom.*, pages 189–192, Dortmund, Germany, Mar 2010.

[HH11a] S. Huber and M. Held. Approximating a Motorcycle Graph by a Straight Skeleton. 2011. (submitted for publication).

[HH11b] S. Huber and M. Held. Motorcycle Graphs: Stochastic Properties Motivate an Efficient Yet Simple Implementation. *J. Exp. Algorithmics*, 2011. (in press).

[HH11c] S. Huber and M. Held. Theoretical and Practical Results on Straight Skeletons of Planar Straight-Line Graphs. In *Proc. 27th Annu. ACM Sympos. Comput. Geom.*, Paris, France, to be published 2011.

[HLA94] M. Held, G. Lukács, and L. Andor. Pocket Machining Based on Contour-Parallel Tool Paths Generated by Means of Proximity Maps. *Comput. Aided Design*, 26(3):189–203, Mar 1994.

[HP00] S. Har-Peled. Constructing Planar Cuttings in Theory and Practice. *SIAM J. Comput.*, 29(6):2016–2039, 2000.

[HS08] J.-H. Haunert and M. Sester. Area Collapse and Road Centerlines based on Straight Skeletons. *GeoInformatica*, 12:169–191, 2008.

[HS09] M. Held and C. Spielberger. A Smooth Spiral Tool Path for High Speed Machining of 2D Pockets. *Comput. Aided Design*, 41(7):539–550, July 2009.

[Ink] Inkscape. `http://inkscape.org/`. accessed on May, 2011.

[IST09] M. Ishaque, B. Speckmann, and C.D. Tóth. Shooting Permanent Rays among Disjoint Polygons in the Plane. In *Proc. 25th Annu. ACM Sympos. Comput. Geom.*, pages 51–60, Aarhus, Denmark, 2009.

[Kle89] R. Klein. *Concrete and Abstract Voronoi Diagrams*, volume 400 of *Lecture Notes in Computer Science*. Springer-Verlag, 1989. ISBN 3-540-52055-4.

[KLN09] R. Klein, E. Langetepe, and Z. Nilforoushan. Abstract Voronoi diagrams revisited. *Comput. Geom. Theory and Appl.*, 42(9):885 – 902, 2009.

[KMM93] R. Klein, K. Mehlhorn, and S. Meiser. Randomized Incremental Construction of Abstract Voronoi Diagrams. *Comput. Geom. Theory and Appl.*, 3(3):157–184, 1993.

[KW11] T. Kelly and P. Wonka. Interactive Architectural Modeling with Procedural Extrusions. *ACM Trans. Graph.*, 2011. accepted, not yet published.

[LD03] R. G. Laycock and A. M. Day. Automatically generating large urban environments based on the footprint data of buildings. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*, SM '03, pages 346–351, New York, NY, USA, 2003. ACM.

[MPF] The GNU MPFR Library. `http://www.mpfr.org/`. accessed on May, 2011.

[MWH$^+$06] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool. Procedural Modeling of Buildings. *ACM Trans. Graph.*, 25:614–623, July 2006.

[OPC96] J.M. Oliva, M. Perrin, and S. Coquillart. 3D Reconstruction of Complex Polyhedral Shapes from Contours Using a Simplified Generalized Voronoi Diagram. *Comput. Graph. Forum*, 15(3):397–408, 1996.

[OSM] OpenStreetMap. `http://www.openstreetmap.org`. accessed on May, 2011.

[Pap98] E. Papadopoulou. L^∞ Voronoi Diagrams and Applications to VLSI Layout and Manufacturing. In *Proc. 9th Annu. Internat. Sympos. Algorithms Comput. (ISAAC*

'98), pages 9–18, London, UK, 1998. Springer-Verlag.

[PC03] S.C. Park and Y.C. Chung. Mitered offset for profile machining. *Comput. Aided Design*, 35(5):501–505, Apr 2003.

[Sha94] M. Sharir. Almost Tight Upper Bounds for Lower Envelopes in Higher Dimensions. *Discrete Comput. Geom.*, 12:327–345, 1994.

[TA09] M. Tănase-Avătavului. *Shape Decomposition and Retrieval*. PhD thesis, Universiteit Utrecht, Faculteit Wiskunde en Informatica, Utrecht, Netherlands, 2009.

[TV04a] M. Tănase and R. C. Veltkamp. A Straight Skeleton Approximating the Medial Axis. In *Proc. 12th Annu. Europ. Symp. Algorithms (ESA '04)*, pages 809–821, Bergen, Norway, Sep 2004.

[TV04b] M. Tănase and R.C. Veltkamp. Straight Line Skeleton in Linear Time, Topologically Equivalent to the Medial Axis. In *Proc. 20th Europ. Workshop Comput. Geom.*, Mar 2004.

[Vya09a] K. Vyatkina. Linear Axis for Planar Straight Line Graphs. In *Proc. of the 15th Australasian Symposium on Computing*, volume 94, pages 139–152, Darlinghurst, Australia, 2009. Australian Computer Society.

[Vya09b] K. Vyatkina. On the Structure of Straight Skeletons. In *Transactions on Computational Science VI*, volume 5730 of *Lecture Notes in Computer Science*, pages 362–379. Springer Berlin / Heidelberg, 2009.

[Yak04] E. Yakersberg. *Morphing Between Geometric Shapes Using Straight-Skeleton-Based Interpolation*. Msc thesis, Technion – Israel Institue of Technology, May 2004.

[Yap87] C.K. Yap. An $O(n \log n)$ Algorithm for the Voronoi Diagram of a Set of Simple Curve Segments. *Discrete Comput. Geom.*, 2(4):365–393, 1987.

[Yap04] C. K. Yap. Robust Geometric Computation. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 41, pages 927–952. Chapmen & Hall/CRC, Boca Raton, FL, 2 edition, 2004.

INDEX