International Journal of Computational Geometry
& Applications
Vol. 22, No. 5 (2012) 471–498
© World Scientific Publishing Company
DOI: 10.1142/S0218195912500124

A FAST STRAIGHT-SKELETON ALGORITHM BASED ON GENERALIZED MOTORCYCLE GRAPHS*

STEFAN HUBER
Department of Mathematics, University of Salzburg
A-5020 Salzburg, Austria
shuber@cosy.sbg.ac.at

MARTIN HELD
Department of Computer Sciences, University of Salzburg
A-5020 Salzburg, Austria
held@cosy.sbg.ac.at

Received 6 April 2012
Revised 11 October 2012
Communicated by Christian Knauer

ABSTRACT

This paper deals with the fast computation of straight skeletons of planar straight-line graphs (PSLGs) at an industrial-strength level. We discuss both the theoretical foundations of our algorithm and the engineering aspects of our implementation Bone. Our investigation starts with an analysis of the triangulation-based algorithm by Aichholzer and Aurenhammer and we prove the existence of flip-event-free Steiner triangulations. This result motivates a careful generalization of motorcycle graphs such that their intimate geometric connection to straight skeletons is maintained. Based on the generalized motorcycle graph, we devise a non-procedural characterization of straight skeletons of PSLGs and we discuss how to obtain a discretized version of a straight skeleton by means of graphics rendering. Most importantly, this generalization allows us to present a fast and easy-to-implement straight-skeleton algorithm.

We implemented our algorithm in C++ based on floating-point arithmetic. Extensive benchmarks with our code Bone demonstrate an $O(n \log n)$ time complexity and $O(n)$ memory footprint on 22 300 datasets of diverse characteristics. This is a linear factor better than the implementation provided by CGAL 4.0, which shows an $O(n^2 \log n)$ time complexity and an $O(n^2)$ memory footprint; the CGAL code has been the only fully-functional straight-skeleton code so far. In particular, on datasets with ten thousand vertices, Bone requires about 0.2–0.6 seconds instead of 4–7 minutes consumed by the CGAL code, and Bone uses only 20 MB heap memory instead of several gigabytes. We conclude our paper with a discussion of the engineering aspects and principles that make Bone reliable enough to compute the straight skeleton of datasets comprising a few million vertices on a desktop computer.

Keywords: Straight skeleton; motorcycle graph; reliable implementation.

*Work supported by Austrian FWF Grant L367-N15.

1. Introduction

1.1. *Motivation and preliminaries*

In this paper[a] we deal with the industry-strength computation of straight skeletons of planar straight-line graphs (PSLGs). Straight skeletons are skeletal structures similar to generalized Voronoi diagrams. Unlike Voronoi diagrams of straight-line segments, straight skeletons have the pleasant property of comprising straight-line segments only. Straight skeletons of simple polygons were introduced to computational geometry in the mid-90s by Aichholzer *et al.*[2] and generalized to PSLGs by Aichholzer and Aurenhammer.[3] However, the geometric roots of straight skeletons date back to the 19th century, see Ref. 4 for details. Since the mid-90s, straight skeletons have found numerous applications in science and industry:

- In computer-aided design and manufacturing (CAD/CAM), straight skeletons are employed for computing so-called mitered offset curves. Unlike offset curves based on Minkowski sums of a polygon with a unit disk, mitered offset curves preserve sharp reflex vertices. For instance, Park and Chung[27] use mitered offset curves in order reduce heat and erosion at reflex vertices.
- Oliva *et al.*[26] and Barequet *et al.*[6] presented algorithms for shape reconstruction and contour interpolation of three-dimensional bodies from a family of parallel cross sections, e.g., in medical imaging.
- Haunert and Sester[14] presented a method for topology-preserving area collapsing in geographic maps. They also used the straight skeleton in order to compute centerlines of roads and junctions.
- The straight skeleton is used to automatically generate different types of roofs for buildings, in particular so-called hip roofs.[15,23–25] Besides roof construction, the straight skeleton can also be employed to generate mountain terrains, see Ref. 3.
- Tănase and Veltkamp[29] presented a polygon-decomposition algorithm that decomposes a simple polygon into convex polygons, which is based on straight skeletons.
- Demaine *et al.*[11] investigated the fold-and-cut problem within the field of mathematical origami. They presented an algorithm based on straight skeletons in order to compute so-called crease patterns by which a piece of paper needs to be folded such that a single straight cut with a scissor produces a paper figure given.

Following Aichholzer *et al.*,[2] a straight skeleton is defined by a so-called wavefront propagation process. Consider a simple polygon P. Every edge of P sends out a parallel wavefront edge that moves with unit speed to the interior of P. The wavefront edges of two adjacent polygon edges e_1, e_2 are joined by a wavefront vertex that moves along the angular bisector of e_1 and e_2. We interpret the wavefront at any time t as a 2-regular graph that is denoted by $\mathcal{W}_P(t)$. During the propagation,

[a]A short version of this paper was published in Ref. 22.

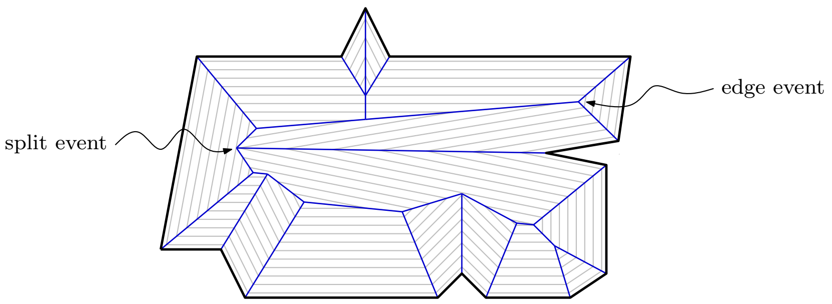


Fig. 1. A bold simple polygon P, wavefronts $\mathcal{W}_P(t)$ in gray for different times t, and the straight skeleton $\mathcal{S}(P)$. Every vertex of $\mathcal{S}(P)$ is created by an event. Every edge of $\mathcal{S}(P)$ lies on the angular bisector of two polygon edges.

topological changes occur in the wavefront, which are classified into two types of events:

- An *edge event* occurs when a wavefront edge shrinks to length zero and vanishes.
- A *split event* occurs when a reflex wavefront vertex meets a wavefront edge and causes a split of the wavefront into parts. A wavefront vertex is called reflex if the angle of the incident wavefront edges on the propagation side is larger than π.

  If multiple split events occur at the same time and location — i.e., when multiple reflex wavefront vertices meet — then we call this a multi split event.[b] Multi split events play a prominent role in the theory of straight skeletons.

The straight skeleton $\mathcal{S}(P)$ of P is defined as the set of loci that are traced out by the wavefront vertices, see Fig. 1. Aichholzer and Aurenhammer[3] generalized this concept to PSLGs G. First, the wavefront propagation is extended to the entire plane as every edge e of G sends out a wavefront copy on both sides. Second, at terminal vertices v of G the wavefront forms a rectangular cap since an additional wavefront edge is sent out perpendicular to the single incident edge of v. Third, if multiple edges of G meet in a vertex v then only the wavefront edges of neighboring edges of G in the cyclic incidence order at v are joined by a wavefront vertex. The straight skeleton $\mathcal{S}(G)$ is again defined as the set of loci traced out by wavefront vertices of $\mathcal{W}_G$, see Fig. 2.

The overlay of $\mathcal{S}(G)$ and G partitions $\mathbb{R}^2$ into polygonal faces. Each face $f(e)$ is swept out by a single wavefront edge e. By $e(t)$ we denote the union of straight-line segments covered by the wavefront edge e at time t. By $\overline{e(t)}$ we denote the supporting line of $e(t)$. In case that $e(0)$ is a single point — e.g., because e emanates from a terminal vertex of G — then we define $\overline{e(0)} := \lim_{t \searrow 0} \overline{e(t)}$. It is well-known that each face $f(e)$ is monotone w.r.t. $\overline{e(0)}$. The edges of $\mathcal{S}(G)$ are called arcs. Every

[b]Also known as *vertex events*, see Ref. 12.

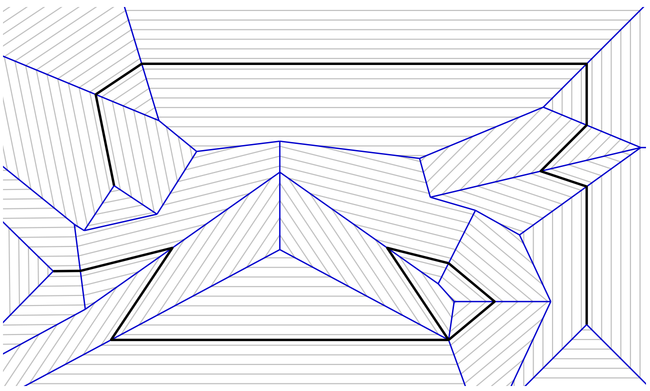


Fig. 2. Gray wavefronts $\mathcal{W}_G(t)$ at different times t and the straight skeleton $\mathcal{S}(G)$ of a bold planar straight-line graph G.

arc lies on the boundary of two faces, say $f(e)$ and $f(e')$, and therefore also on the bisector of $\overline{e(0)}$ and $\overline{e'(0)}$. Thus, the straight skeleton comprises straight-line segments only. The straight skeleton $\mathcal{S}(G)$ is of linear size as it comprises $2n - t + 2$ vertices, where t denotes the number of terminal vertices of G. (In the presence of multi split events, one needs to take the degree of the resulting straight-skeleton vertex into account.)

1.2. *Prior work*

The brute-force approach to a straight skeleton computation would be to simply find the chronological order of all edge and split events. While edge events are rather simple to handle — one puts every edge in a priority queue, where the collapse time is the priority — the split events require significantly more effort. In order to find the next split event one could test every reflex wavefront vertex against every wavefront edge for a potential hit. This would require $O(n^3 \log n)$ time in total.

Aichholzer *et al.*[2] presented a method to handle all split events of a simple polygon in $O(n^2 \log n)$, which is also the total runtime of their algorithm. For straight skeletons, it became common to refine the runtime analysis by taking the number $r \in O(n)$ of reflex wavefront vertices into account. Then the refined complexity bound is $O(nr \log n)$ for the algorithm by Aichholzer *et al.*[2]

Aichholzer and Aurenhammer[3] presented an algorithm for PSLGs that is based on kinetic triangulations. The algorithm works as follows: One keeps the area triangulated that has not yet been swept by the wavefront at time t, which is $\bigcup_{t' \geq t} \mathcal{W}_G(t')$. Every edge and split event is indicated by the collapse of a triangle in this kinetic triangulation. Hence, the problem has been reduced to simulate a kinetic triangulation, in which the vertices move along straight lines with constant velocity. The simplicity of this algorithm is traded for a new type of event: *flip events*. A flip event happens when a reflex wavefront vertex hits an inner diagonal of the triangulation. This diagonal needs to be flipped for the triangulation to remain correct. Unfortunately, the best upper bound known so far for the number of flip events is $O(n^2r)$, even though no example is known that exceeds $\Omega(nr)$ flip events. Nonetheless, $O(n^2r\log n)$ is currently the best known worst-case time complexity for this algorithm. Aichholzer and Aurenhammer[3] mention that tests carried out on a few datasets suggest that an implementation would probably perform better on many datasets. From a practical point of view, this algorithm appears promising due to its simplicity. However, if multiple events occur concurrently then it is easily seen that flip events may lead to a loop in the event handling code such that the same event sequence is repeated over and over again. Consequently, the basic algorithm would not terminate for such an input. Hence, for a real-world implementation an additional non-trivial problem remains open for the approach by Aichholzer and Aurenhammer[3] in order to obtain an implementation that is guaranteed to terminate.

Eppstein and Erickson[12] were the first to present a sub-quadratic algorithm. Their algorithm accepts PSLGs as input and is heavily based on efficient closest-pair data structures that are combined in a hierarchical fashion to trade off time and space. They managed to design an algorithm with a theoretical $O(n^{1+\epsilon} + n^{8/11+\epsilon}r^{9/11+\epsilon}) \subseteq O(n^{17/11+\epsilon})$ time and space complexity. The entire algorithm with all its sub-algorithms is too complicated to be suitable for an implementation.

Cheng and Vigneron[10] presented a slightly faster algorithm for simple non-degenerate polygons with holes. A dataset is non-degenerate if no multi-split events occur. For example, the polygon in Fig. 1 is degenerate in this sense and most datasets of the real world, such as CAD drawings, are likely to be degenerate as well. Their algorithm first computes the so-called motorcycle graph induced by a simple polygon. Based on the motorcycle graph they present a randomized algorithm with an expected runtime of $(n\log^2 n + r\sqrt{r}\log r)$ for a simple polygon. In order to compute the motorcycle graph they rely on $1/\sqrt{r}$-cuttings, which makes the overall algorithm very complicated to implement. In fact, no implementation is known to exist so far.

Indeed, the progress on implementations has been very limited so far. Felkel and Obdržálek[13] gave a brief description of a simple straight-skeleton algorithm that would run in $O(nr + n\log n)$ time. They also implemented a prototype of their algorithm and tested the runtime on seven datasets to confirm the runtime analysis. However, it turned out that the underlying algorithm is flawed, see Ref. 30 for details.

Cacciola[7] implemented the straight-skeleton package that is shipped with the CGAL library.[9] The underlying algorithm is based on the algorithm by Felkel and Obdržálek,[13] but was modified significantly by Cacciola[8] in order to work correctly. However, no details of the algorithm implemented have been published. The current implementation in CGAL 4.0 is based on exact arithmetic. It accepts polygons with holes as input. The worst-case runtime complexity is $O(n^2\log n)$.

1.3. *Our contribution*

In this paper we present the theoretical foundations and the practical developments which led to the first straight-skeleton implementation that is able to actually compute the straight skeleton of PSLGs comprising more than a few ten thousand vertices on a desktop computer. We start with investigations concerning the number of flip events in the triangulation-based algorithm of Aichholzer and Aurenhammer[3] and show that Steiner triangulations exist which are free of flip-events.

These findings motivated us to generalize the concept of a motorcycle graph and to extend the geometric relationship between motorcycle graphs and straight skeletons from non-degenerate polygons to arbitrary PSLGs. This generalization turns out to be very fruitful. First, we present a non-procedural characterization of the straight skeleton of PSLGs, which is a problem investigated since the introduction of straight skeletons. Second, this characterization motivates a straight-skeleton algorithm by means of graphics hardware. Third, we present a wavefront-based straight-skeleton algorithm for arbitrary PSLGs.

Extensive benchmarks on 22 300 datasets show that our floating-point C++ implementation Bone runs in $O(n \log n)$ time on virtually all datasets. A comparison with the implementation provided by CGAL shows that Bone is by a linear factor faster and, even more importantly, by a linear factor more memory efficient. That is, for datasets with ten thousand vertices our code requires between 0.2–0.6 seconds and about 20 MB heap memory. The CGAL code, on the other hand, takes 4–7 minutes and 3–10 GB heap memory for datasets of that size. Moreover, our code is stable enough to cope with datasets containing up to a few million vertices.

Finally, we also report on engineering aspects of our implementation Bone, discuss the main challenges and present our solutions and principles for coping with them.

2. An Analysis of the Triangulation-Based Approach

The triangulation-based algorithm by Aichholzer and Aurenhammer[3] is a promising candidate for a practical implementation due to its simplicity. However, the best known upper bound on its worst-case complexity is $O(n^3 \log n)$, because the best known upper bound on the number of flip events is $O(n^3)$. Taking the number $r \in O(n)$ of reflex wavefront vertices and the number $k \in O(n^2 r)$ of flip events into account, we can express the complexity as $O((n^2 + k) \log n)$. It is noteworthy that even if we would manage to reduce k to $o(n^2)$ we would obtain a worst-case runtime no better than $O(n^2 \log n)$.

In fact, Fig. 3 illustrates a convex polygon for which no flip event and no split event occurs, but still the edge events consume in total $\Theta(n^2 \log n)$ time. The reason is that $e_1, \ldots, e_k$ lead to edge events in the given order. Hence, for the edge event of e_i all the diagonals incident to the former vertices of $e_1, \ldots, e_i$ are collected and now incident to a single vertex. Consequently, we increase the computational effort for every following edge event as we need to reschedule every incident triangle

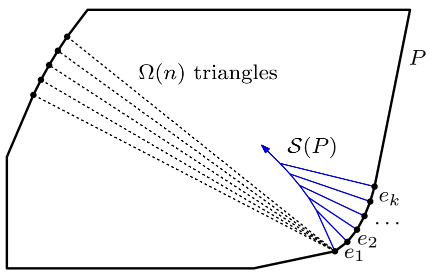


Fig. 3. A convex polygon and (a portion of) a triangulation for which the triangulation-based algorithm requires $\Theta(n^2 \log n)$ time.

in the priority queue after an edge event. Also note that it is not necessary that $\Omega(n)$ diagonals are incident to the bottom vertex of e_1. It would suffice that $\Omega(n)$ diagonals are collected during the sequence of edge events such that each of $\Omega(n)$ edge events needs to reschedule $\Omega(n)$ triangles.

Since the number k of flip events is currently the leading term in the time complexity, we will investigate flip events in more detail. In Ref. 20, we showed different approaches to prove the lower bound of $\Omega(n^2)$ on the number of flip events. In particular, we showed that polygons exist for which every possible initial triangulation leads to $\Omega(n^2)$ flip events and even re-triangulating at favorable points in time does not save enough flip events such that the costs for repeated triangulations were justified.

In other words, ordinary triangulations are in general not flexible enough to reduce the number of flip events to $o(n^2)$. It appears natural to ask whether one can use Steiner triangulations in order to achieve a reduction of flip events. In Ref. 20 we proved that for every simple polygon P we can find a Steiner triangulation which employs $O(n)$ Steiner points and is entirely free of flip events. A more general version of the theorem and a simpler proof is presented in the following.

Theorem 1. *Every planar straight-line graph G with n vertices admits a triangulation with $O(n)$ Steiner points that is free of flip events.*

Proof. The straight skeleton $\mathcal{S}(G)$ of G comprises $O(n)$ inner nodes. We add each node of $\mathcal{S}(G)$ as Steiner vertex and each edge of $\mathcal{S}(G)$ as Steiner edge. It remains to properly triangulate the faces of $\mathcal{S}(G)$. Let $f(e)$ denote an arbitrary face of $\mathcal{S}(G)$ for a wavefront edge e of G. It is known that $f(e)$ is monotone w.r.t. $\overline{e(0)}$, and that the monotone chain of $f(e)$ that contains $e(0)$ is convex. We call this chain the lower convex chain. Reflex vertices of $f(e)$ can only occur in the opposite monotone chain. At each reflex vertex v in $f(e)$ we tessellate $f(e)$ by a line that is perpendicular to $\overline{e(0)}$. That is, we split the edge of the lower convex chain that is hit by the perpendicular line by a Steiner point v' and insert a vertical edge e' between v and v', see Fig. 4. In total this leads to at most $r \in O(n)$ additional Steiner points because any reflex vertex in a face corresponds to a reflex wavefront

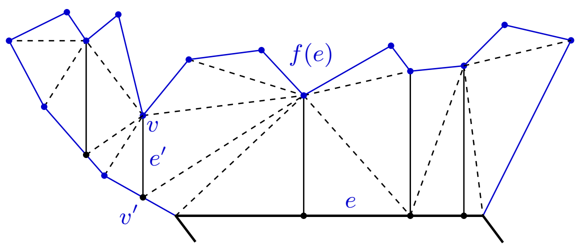


Fig. 4. The monotone face $f(e)$ is tessellated into convex parts by edges that are incident to reflex vertices of $f(e)$ and perpendicular to $\overline{e(0)}$. Every convex part is triangulated arbitrarily.

vertex. Note that these vertical edges are completely contained in $f(e)$ as $f(e)$ is monotone. Hence we end up with a tessellation of $f(e)$ into convex parts. Each of these convex parts is triangulated in some arbitrary fashion.

We now argue that no flip event occurs for this triangulation. During the wavefront propagation the face $f(e)$ is swept by $e(t)$. We declare that the Steiner vertices that reside on the lower chain of $f(e)$ and which are not yet swept by the wavefront remain still. At the moment when a such a Steiner vertex v' gets in contact with the wavefront it moves with the wavefront along the perpendicular edge e' until v' hits the reflex vertex v on the upper chain of $f(e)$ at which e' is incident and vanishes. At any time of the wavefront propagation the tessellation of $f(e)$ induced by the perpendicular edges consists of convex cells. As a consequence, no flip event is caused. Also note that the reflex wavefront vertices are prevented from causing flip events as they move on diagonals of the Steiner triangulation. $\square$

This theorem tells us, that in principle, it is possible to get rid of all flip events using Steiner triangulations. The question that remains is: How can we construct Steiner triangulations with a small number of flip events without knowing the straight skeleton? This question is our main motivation for generalizing the motorcycle graph in the following section.

3. Generalizing the Motorcycle Graph

3.1. *Preparations*

The motorcycle graph is a geometric structure introduced by Eppstein and Erickson.[12] A motorcycle is a point that moves in $\mathbb{R}^2$ with constant velocity along a straight line. Consider n motorcycles $m_1, \dots, m_n$, each having a start point p_i and a velocity v_i, with $1 \leq i \leq n$. While a motorcycle drives, it leaves a trace behind. A motorcycle *crashes* — i.e., it stops driving, but its trace remains — when it reaches the trace of another motorcycle. A motorcycle *escapes* if it never crashes. The *motorcycle graph* $\mathcal{M}(m_1, \dots, m_n)$ is defined as the arrangement of all traces after infinite time. Further, the *track* of m_i is defined as the infinite ray $\{p_i + tv_i : t \geq 0\}$.

Cheng and Vigneron[10] observed a nice geometric relationship between a non-degenerate polygon and a motorcycle graph. Let us first explain what a motorcycle graph $\mathcal{M}(P)$ induced by a non-degenerate simple polygon P is: For every reflex vertex v of P we construct a motorcycle that starts at v and whose velocity is equal to the velocity of the wavefront vertex emanated by v. Further, we consider the edges of P as solid walls. That is, if a motorcycle reaches an edge of P then it crashes, too. The motorcycle graph resulting from this setting is denoted by $\mathcal{M}(P)$.[c]

Theorem 2 (Cheng and Vigneron). *For a non-degenerate simple polygon P the arcs of $\mathcal{S}(P)$ that are incident to a reflex vertex of P are covered by $\mathcal{M}(P)$.*

Let us revisit Theorem 1. For ordinary triangulations, the flip events are caused by reflex wavefront vertices meeting inner triangulation diagonals. For the Steiner triangulation constructed in the proof of Theorem 1, the reflex wavefront vertices move along triangulation diagonals. Hence, they are prevented from causing flip events. The key idea is now the following: If we would use the motorcycle graph $\mathcal{M}(P)$ instead of the straight skeleton $\mathcal{S}(P)$ in order to construct our Steiner triangulation, we would get the very same effect: By Theorem 2, the motorcycle traces are at least as long as the arcs of $\mathcal{S}(P)$ that are traced out by the reflex wavefront vertices. Hence, the reflex wavefront vertices are again prevented from causing any flip event.

However, in this paper we aim to develop a practical algorithm that is (i) implementable and (ii) works for arbitrary PSLGs and not only for non-degenerate polygons. In order to do so, we require a more general motorcycle graph $\mathcal{M}(G)$ induced by an arbitrary PSLG G. We demand the following properties of $\mathcal{M}(G)$:

(1) $\mathcal{M}(G)$ has to cover all reflex arcs of $\mathcal{S}(G)$, where an arc is called *reflex* if it is traced out by a reflex wavefront vertex. Note that once we do not only consider non-degenerate input, reflex arcs exist that are not incident to a vertex of G. That is, it can happen that reflex wavefront vertices are born at multi split events. As a consequence, we may need some motorcycles that start later than the rest.

(2) The overlay of G and $\mathcal{M}(G)$ has to induce a convex tessellation of $\mathbb{R}^2$. This requirement will be used in order to devise a straight-skeleton algorithm. This is trivial for non-degenerate polygons, but becomes non-trivial for arbitrary PSLGs G.

3.2. *Motorcycle graph induced by a PSLG*

In addition to a start point and a velocity, a motorcycle will now also have a start time. As in the previous section, we define the motorcycle graph $\mathcal{M}(G)$ induced by a PSLG G by declaring the set of walls and the set of motorcycles. The set of walls is given by the set of straight-line edges of G. For the matter of simplicity,

[c]Cheng and Vigneron did not introduce the concept of walls and did not distinguish as strictly between motorcycle graphs and motorcycle graphs induced by polygons, as we do here. However, our formalism comes handy for our further generalization of motorcycle graphs.

we define the movement of a motorcycle m by two wavefront edges e and e' as follows: The position of a driving motorcycle m at time t is given by the intersection $\overline{e(t)} \cap \overline{e'(t)}$. Hence, if we know the start point and the two defining wavefront edges of a motorcycle then we also know its velocity and its start time. We call the defining wavefront edge left to the track of m the *left arm* of m and the other one the *right arm*. The set of motorcycles of $\mathcal{M}(G)$ are defined as follows:

(1) For each reflex wavefront vertex v in the initial wavefront[d] $\mathcal{W}_G(0)$ we start a motorcycle m at $v(0)$, where $v(t)$ denotes the position of v at time t. The arms of m are the two incident wavefront edges of v, see Fig. 5(a). Hence, one motorcycle belongs to every reflex arc of $\mathcal{S}(G)$ that originates from G. At terminal vertices of G we launch two motorcycles, see Fig. 5(b).
(2) If two or more motorcycles crash simultaneously at a point p then we consider a local disc D around p. The disc D is tessellated into slices by motorcycle traces established up to the current simulation time. If one of the slices is non-convex then we start a new motorcycle as follows.

    Denote by $m_1, \ldots, m_k$ the motorcycle that crashed at p such that (i) their traces appear counter-clockwise around p and (ii) the traces of m_1 and m_k bound the convex slice of D. We distinguish two cases:

    (a) The left arm of m_1 and the right arm m_k span a reflex angle. Then we start at p a motorcycle m whose left arm is the left arm m_1 and whose right arm is the right arm of m_k, see Fig. 5(c).
    (b) The left arm of m_1 and the right arm m_k span a convex angle. Then we start at p a motorcycle m whose left and right arm is inherited from m_k, see Fig. 5(d). Hence, m continues the movement of m_k.[e]

    We call $m_1, \ldots, m_k$ the ancestor of m. In particular, we call m_1 the left ancestor and m_k the right ancestor. The left-most ancestor chain of m is recursively defined by the trace of m and the left-most ancestor chain of m_1 until no further left-most ancestors exist. Likewise we define the right-most ancestor chain of m.

The motorcycle graph $\mathcal{M}(G)$ induced by G is defined as the motorcycle graph that results from the above set of motorcycles and walls. Figure 6 depicts $\mathcal{M}(G)$ induced by the graph shown in Fig. 2.

[d]We interpret $\mathcal{W}_G(0)$ as a 2-regular graph that forms a hull around G, i.e., $\mathcal{W}_G(0)$ and $\mathcal{W}_G(\epsilon)$ have the same topology for a sufficiently small $\epsilon > 0$. However, geometrically, G and $\mathcal{W}_G(0)$ overlap each other.
[e]Lemma 3 is fundamental to our straight-skeleton algorithm and it requires that we launch a new motorcycle. In addition, our proof of Theorem 3 uses the property that motorcycles span an angle of at least 90° with their defining arms. If we would let m move on the bisector of the according wavefront edges as in Case (2a) then we would introduce motorcycles that break this assumption. Our simple solution is to continue the moment of m_k, but m_1 would work just as good. Since it can be shown that the corresponding reflex straight-skeleton arcs do not reach p anyhow, it has no influence to the output of our straight-skeleton algorithm in Sec. 4.

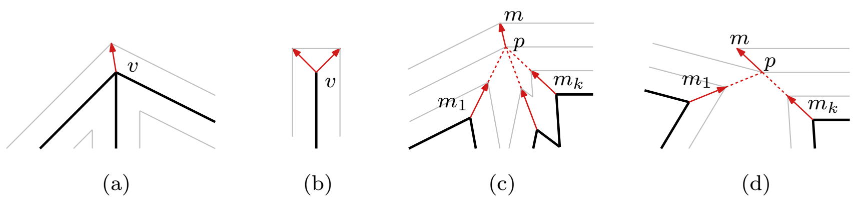


Fig. 5. The procedure by which motorcycles are launched: (a) start a motorcycle for a reflex vertex in $\mathcal{W}_G(0)$, (b) at terminal vertices of G we start two motorcycles and (c, d) launch a new motorcycle when multiple motorcycles crash simultaneously into each other.

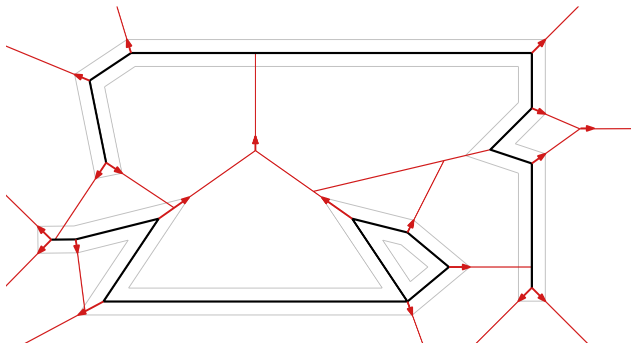


Fig. 6. The motorcycle graph $\mathcal{M}(G)$ induced by the same graph G as in Fig. 2.

Lemma 1. *For each motorcycle m in $\mathcal{M}(G)$ the left and right arm span a reflex angle on the side where m propagates to.*

Lemma 2. *$\mathcal{M}(G)$ contains at most $2r - 1 \in O(n)$ motorcycle traces, where r denotes the number of reflex wavefront vertices in $\mathcal{W}_G(0)$.*

While the two lemmas above are easily seen as correct, the next lemma is non-trivial albeit crucial for our approach.

Lemma 3. *For any point p in the relative interior of $\mathcal{M}(G)$ a local disc D centered at p is tessellated into convex slices by $\mathcal{M}(G)$.*

Proof. For each motorcycle m we define $m(t) := p^* + (t - t^*)v$, where p^*, t^*, v denote the start point, the start time and the velocity of m, respectively.

Assume to the contrary that there is a reflex slice of D. It is easy to see that we only need to check Case (2a) of our motorcycle launch procedure. That is, two or more motorcycles $m_1, \ldots, m_k$ crashed simultaneously at p and a new motorcycle m was launched. The left arm e_l of m_1 and the right arm e_r of m_k span a reflex angle, see Fig. 7(a).

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. Further, we denote by L the left-most ancestor chain of m_k

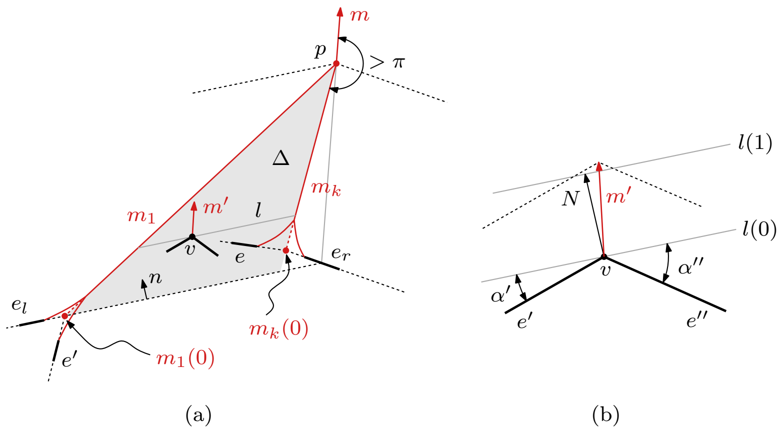


Fig. 7. $\mathcal{M}(G)$ induces a convex tessellation: (a) proof by contradiction: there exists a motorcycle m' cutting off the right-most ancestor chain of m_1 or the left-most ancestor chain of m_k and (b) there exists a motorcycle m' that is always ahead of $l(t)$.

and by R the right-most ancestor chain of m_1. Since (i) $m_k(0)$ is left of m, (ii) e_r is the right arm of m_k and m and e_l is the left arm of m_1 and m and (iii) m_1 and m_k reach p at the same time it holds that $m_k(0) \in \Delta$. In particular, $e(0) \subseteq \Delta$ as $e(0)$ and L cannot intersect.

Hence, there is a vertex of G in the interior of Δ. Let v denote such a vertex with maximum distance to $\overline{e_l(0)}$. Let us denote by n the propagation vector of e_l. Note that $|n| = 1$. Furthermore, let l denote the supporting line of v that is parallel to $\overline{e_l(0)}$. We interpret $l(t)$ as a sweep line propagating with speed n, with $l(0) = l$.

Then there is a motorcycle m' starting at v that is ahead of $l(t)$ or just on $l(t)$ for all $t \geq 0$ until m' crashes: Let e' and e'' denote the left and right arm of m'. Assume w.l.o.g. that the angle α' between l and e' is less than or equal to the angle α'' between l and e''. Then m' is left of the ray $N = v + \lambda n$, with $\lambda \geq 0$, see Figs. 7(b) and 7(c). Consider the direction vector of m' to be arbitrary, but fixed. If $\alpha'' = 0$ then $m'(t) \in l(t)$ for all $t \geq 0$. If $\alpha'' > 0'$ then m_1 moves even faster and $m_1(t)$ is ahead of $l(t)$. We now conclude the proof with the following case analysis:

(1) Assume m' reaches L or R at point p^*. We claim that m' reaches p^* before the motorcycles of L resp. R do, as $m_1(t)$ and $m_k(t)$ always lie strictly behind $l(t)$. This is trivial for m_1 as $e_l(t)$ is behind $l(t)$. On the other hand, $m_k(0)$ starts behind $l(0)$ and if m_k would overtake $l(t)$ then m_k would reach p earlier than m_1 which is a contradiction. As m_1 and all motorcycles of R share a common arm, and m_k and all motorcycles of L share a common arm, $m'(t)$ reaches p^* earlier. This is a contradiction to the birth of m due to m_1 and m_k, which concludes the proof of this case.

(2) Assume that m' does not reach L or R. Then there exists a motorcycle m'' into which m' crashed. As v has maximum distance to $\overline{e_l(0)}$ there exists again

a motorcycle that is always ahead of $l(t)$ and we can repeat this case analysis for this motorcycle. As there are only finitely many motorcycles, we eventually end up in Case (1). $\square$

Corollary 1. *The overlay of* G *and* $\mathcal{M}(G)$ *induces a convex tessellation of* $\mathbb{R}^2$.

Proof. Every reflex angle due to a reflex wavefront vertex in $\mathcal{W}_G(0)$ is split by a trace from $\mathcal{M}(G)$. $\square$

The terrain model. For the following theorem, we need the so-called terrain model, which was introduced by Aichholzer *et al.*[2] The idea is to embed the wavefront propagation into $\mathbb{R}^3$ by considering the z-axis as the time axis. In other words, one considers the set $\mathcal{T}(G) := \bigcup_{t\geq 0} \mathcal{W}_G(t) \times \{t\} \subseteq \mathbb{R}^3$, which is called the *terrain* of G. There is a one-to-one correspondence between the edges of $\mathcal{T}(G)$ and the arcs of $\mathcal{S}(G)$ and between the faces of $\mathcal{T}(G)$ and the faces of $\mathcal{S}(G)$. In particular, we obtain $\mathcal{S}(G)$ from $\mathcal{T}(G)$ by projecting the edges of $\mathcal{T}(G)$ onto the plane. Valleys of $\mathcal{T}(G)$ correspond to reflex arcs of $\mathcal{S}(G)$ and ridges to convex arcs of $\mathcal{S}(G)$.

For every arc a in $\mathcal{S}(G)$ we denote by $\hat{a}$ the corresponding edge in $\mathcal{T}(G)$ and for every straight-skeleton face $f(e)$ we denote by $\hat{f}(e)$ the corresponding face in $\mathcal{T}(G)$. Similarly, we interpret a motorcycle m in the terrain model by defining $\hat{m}(t) = m(t) \times \{t\}$. That is, we obtain a tilted motorcycle trace $\hat{m}$ and the slope is the reciprocal of the speed of m.

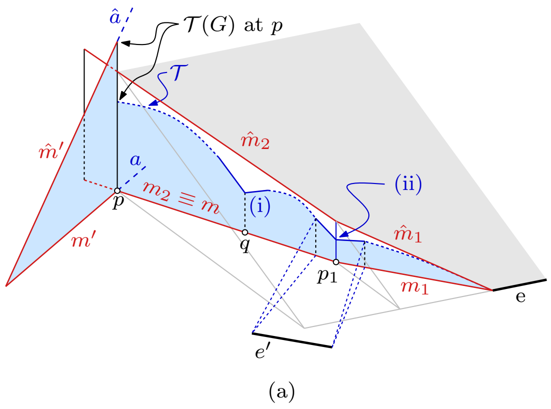
Theorem 3. *The reflex arcs of* $\mathcal{S}(G)$ *are covered by* $\mathcal{M}(G)$.

Proof. We first prove the following claim, which is later used in a proof by contradiction of 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 at least the height of $\mathcal{T}(G)$ at p. Equality is attained if and only if the valley of $\mathcal{T}(G)$, which corresponds to $\hat{m}$, 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 Fig. 8(a) 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, $\hat{m}_1, \ldots, \hat{m}_k$ lie on a plane, namely the supporting plane of the terrain face $\hat{f}(e)$.
- The angle between $e(0)$ and the trace of m_1 and between the traces of m_i and m_{i+1}, for $1 \leq i \leq k-1$, are at most 180° by Corollary 1. Furthermore, for any $1 \leq i \leq k$ the motorcycle m_i spans with its right arm e an angle of at least 90° by Lemma 1.



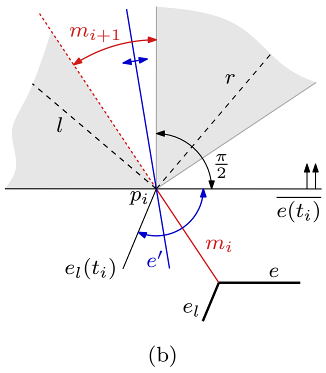


Fig. 8. Tilted motorcycle traces are above the terrain.

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. 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 the 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 having the same velocity as m_1 and starting from $m_1(0)$. Hence, the corresponding valley and $\hat{m}_1$ overlap and $\mathcal{T}$ 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) We need to show that if the slope of $\mathcal{T}$ is at most the slope of $\hat{m}_i$ before p_i, then the slope of $\mathcal{T}$ is at most 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 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 Fig. 8(b). 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 $\overline{e(t_i)}$. The valid domains for l and r are shaded accordingly in Fig. 8(b). 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) as 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 Fig. 8(a) and use Claim (1) in a proof by contradiction of Theorem 3. 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 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 Fig. 8(a).) This contradiction finally concludes the proof. □

Theorem 3 extends Theorem 2 by Cheng and Vigneron.[10] The idea of their proof is incorporated into the proof of Case (i) of Claim (1). In the following, Claim (1) will turn out to be a useful tool on its own grounds. Hence, we cast it into the following corollary.

Corollary 2. *Let m be a motorcycle of $\mathcal{M}(G)$ and p a point on its trace. The height of $\hat{m}$ is at least the height of $\mathcal{T}(G)$ at p.*

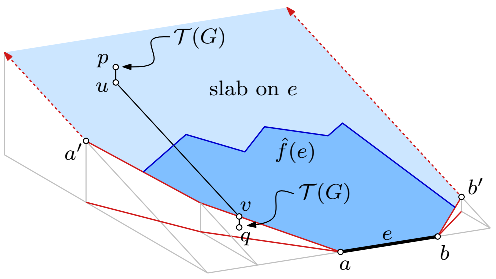


Fig. 9. The lower-envelope slab defined for the wavefront edge e.

3.3. *Lower-envelope characterization of $\mathcal{S}(G)$*

Aichholzer *et al.*[2] stated the problem of finding a non-procedural definition of a straight skeleton. Eppstein and Erickson[12] made the first progress by attempting to characterize $\mathcal{T}(G)$ instead of $\mathcal{S}(G)$. They defined a set of plane slabs (i.e., linear functions $D \to \mathbb{R}$, with $D \subset \mathbb{R}^2$) whose lower envelope is equal to $\mathcal{T}(G)$. However, the definition of their slabs is based on the lengths of the reflex straight-skeleton arcs. Cheng and Vigneron[10] extended this idea and defined their slabs using the motorcycle graph. However, their characterization only works for non-degenerate polygons and not for arbitrary PSLGs. The following construction generalizes the slabs of Cheng and Vigneron and finally yields a non-procedural characterization for arbitrary PSLGs.

Lemma 4. *Let p, q be two distinct points on $\mathcal{T}(G)$. Then the slope of the line $\overline{pq}$ is at most* 1.

Proof. We denote by p' and q' the projections of p and q onto the ground plane. Consider the intersection $\mathcal{T}$ of $\mathcal{T}(G)$ with a vertical curtain erected above the line section $[p', q']$. Then $\mathcal{T}$ is a plane monotone polygonal chain whose sections have a slope of at most 1. Hence the line $\overline{pq}$ has a slope of at most 1. □

The lower envelope. Let e denote a wavefront edge and let a, b be the endpoints of $e(0)$, see Fig. 9. If there is a motorcycle starting at a whose right arm is 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 $a' := a$. Analogously for the chain of tilted motorcycle traces starting at b and ending at b' whose left arms are e. Now we consider the plane slab which lies on the supporting plane $\hat{f}(e)$, and which is bounded below by $e(0)$ and the tilted motorcycle traces mentioned above. At the ends a' and b' the slab is bounded by rays which are perpendicular to $\overline{e(0)}$. Summarizing, at each input edge we have two slabs, one at each side, and for every terminal vertex we have one additional slab. We denote by $L(G)$ the lower envelope of the union of those slabs.

Theorem 4. *The lower envelope $L(G)$ is identical to $\mathcal{T}(G)$.*

Proof. It is easy to see that each face of $\mathcal{T}(G)$ is contained in its corresponding slab of $L(G)$. 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 Fig. 9. 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 get the point v. If v is on a tilted trace then Corollary 2 implies that we can project v down to $\mathcal{T}(G)$ and get to a point q. (Otherwise, $q := v$.) 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 4. $\square$

Computing $\mathcal{S}(G)$ using graphics hardware. Theorem 4 has an immediate practical application: It admits a simple method to render $\mathcal{T}(G)$ without knowing $\mathcal{S}(G)$. One first computes the motorcycle graph $\mathcal{M}(G)$ by a conventional algorithm (on the CPU) and then constructs the slabs as illustrated in Fig. 9. By rendering the set of slabs while looking at them from below, with each slab painted with its own unique color, one obtains an image which corresponds to $L(G)$. Pixels of the same color correspond to the straight-skeleton faces. By employing techniques described by Hoff *et al.*,[19] one can compute $\mathcal{S}(G)$ using graphics hardware.

4. A Wavefront-Type Straight-Skeleton Algorithm

4.1. *Propagating the extended wavefront*

The following algorithm is motivated by flip-event-free Steiner triangulations, whose existence we proved in Theorem 1. Instead of employing the straight skeleton in order to construct such a triangulation, we want to use the motorcycle graph, as motivated by Theorem 3. In fact, in the following it will turn out that we can forget about the triangulation entirely due to Corollary 1, and we will present a wavefront-type algorithm that simulates the propagation of an *extended wavefront* $\mathcal{W}_G^*$.

Definition 1. The extended wavefront $\mathcal{W}_G^*(t)$ is defined by the overlay of $\mathcal{W}_G(t)$ and $\mathcal{M}(G) \cap \bigcup_{t' \geq t} \mathcal{W}_G(t')$.

That is, we add to $\mathcal{W}_G(t)$ the part of $\mathcal{M}(G)$ that has not yet been swept by the wavefront, see Fig. 10. We again interpret $\mathcal{W}_G^*(t)$ as a kinetic PSLG. As the extended wavefront is motivated by Steiner triangulations, we call the vertices of $\mathcal{W}_G^*(t)$ that are not present in $\mathcal{W}_G(t)$ *Steiner vertices*. In addition, we call the Steiner vertices that form the intersection points of $\mathcal{W}_G(t)$ and $\mathcal{M}(G)$ *moving Steiner vertices*, and we call those that have not yet been reached by the wavefront *resting Steiner vertices*. Furthermore, a resting Steiner vertex that corresponds to the simultaneous crash of two or more motorcycles is called a *multi Steiner vertex*.

Lemma 5. *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.*

Proof. This follows immediately from Corollary 1 and Theorem 3. $\square$

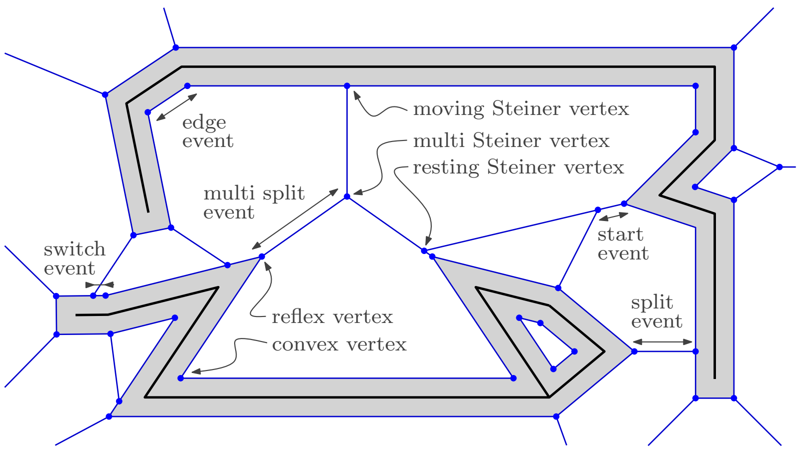


Fig. 10. The extended wavefront $\mathcal{W}^*_G(t)$ of G after some time t. The shaded area was already swept. The remaining white faces are all convex. The different types of vertices and events are labeled accordingly.

An important consequence of this lemma is that during the propagation of $\mathcal{W}^*_G$ only adjacent vertices can meet. In fact, this is the central aspect of our algorithm that ensures its efficiency. For the original wavefront-type algorithm by Aichholzer *et al.*,[2] the authors pointed out that efficiently finding the next split event is the main problem. Using our extended wavefront, a split event is simply indicated by the collapse of an edge of $\mathcal{W}^*_G$ when a reflex wavefront vertex meets a moving Steiner vertex. Thus, we avoid the costly search for the next split event.

The straight-skeleton algorithm. First, we determine the initial extended wavefront[f] $\mathcal{W}^*_G(0)$. Every edge of $\mathcal{W}^*_G(0)$ corresponds to an event and is put in a priority queue Q with the collapse time as priority, if finite and positive. In the main loop we fetch one event after the other in chronological order, apply the corresponding topological changes to $\mathcal{W}^*_G$ and repeat until Q is empty. We distinguish events by the following classification:

- **Edge event:** Two convex vertices u and v meet. We add the convex straight skeleton arcs traced out by u and v. Then we merge u and v to a new convex vertex. As a special case we check whether a whole triangle of the wavefront crashed due to u and v.
- **Split event:** A reflex vertex u meets a moving Steiner vertex v and they are moving towards each other. First, we add the reflex straight skeleton arc which has been traced out by u. Then we consider the wavefront at the left of the edge $e = (u, v)$. If this side collapsed we add corresponding straight skeleton

[f]Again, we interpret $\mathcal{W}^*_G(0)$ as a graph with the same topology as $\mathcal{W}_G(\epsilon)$ for a small enough $\epsilon > 0$.

arcs. Otherwise a new convex vertex emerges, which is connected to the vertices adjacent to u and v lying left to e. We proceed likewise at the right side of e.

- **Start event:** A reflex vertex or a moving Steiner vertex u meets a resting Steiner vertex v. In other words, the wavefront reached v, which now becomes a moving Steiner vertex, and one of the incident edges of u other than (u, v) is split by v.
- **Switch event:** A convex vertex u meets a moving Steiner vertex or a reflex vertex v. The convex vertex u is migrating from one convex face to a neighboring one by jumping over v. If v was a reflex vertex then it becomes a moving Steiner vertex and we add corresponding straight skeleton arcs.
- **Multi split event:** Reflex vertices $u_1, \ldots, u_k$ meet simultaneously a multi Steiner vertex u. This event is conceptually similar to the simple split event, except that the wavefront is in general split into multiple parts. First, we add reflex straight skeleton arcs which have been traced out by $u_1, \ldots, u_k$.

  Second, we assume that $u_1, \ldots, u_k$ appear clockwise at u. We consider all consecutive pairs $u_i, u_{1+i \bmod k}$ with $1 \leq i \leq k$. Let e_i denote the edge (u, u_i) and let e_{i+1} denote the edge $(u, u_{1+i \bmod k})$. Then we patch the wavefront for each sector bound by e_i and e_{i+1} as follows. (Note that if $k = 1$ then the one sector spans the whole local disc.)

  We create a new vertex v which patches the ccw-edge e_l of e_i at u_i and the cw-edge e_r of e_{i+1} at $u_{1+i \bmod k}$ together. Also note that additional edges e may have been incident to u between e_i and e_{i+1}. Such an edge e could lie exactly on the trajectory of v, e.g., if v is a reflex wavefront vertex, because e_l and e_r span a reflex angle. In this case e, which was incident to u, simply becomes incident to v. In all other cases, e splits e_l resp. e_r by an additional moving Steiner vertex, depending on whether e lies left or right to the trajectory of v.
- **Multi start event:** A moving Steiner vertex u_1 meets a multi Steiner vertex u. This event can be treated in the same fashion as a multi split event with $k = 1$, except that u_1 is not a reflex vertex. Consequently, there is no straight-skeleton arc that was traced out by u_1. In fact, in our implementation we do not distinguish between a multi start event and a multi split event. Both are handled by the same routine.
- **Remaining events:** If two moving Steiner vertices meet then we can simply remove the corresponding edge. All other events (e.g., a convex vertex meets a resting Steiner vertex) are guaranteed not to occur.

After the last event occurred, the extended wavefront $\mathcal{W}_G^*$ has the shape of a polygon circumscribing G. Each of the remaining wavefront vertices traces an infinite straight-skeleton arc. In our implementation, we add for each infinite arc an infinite straight-skeleton vertex and connect these vertices in a circular fashion. Hence, we obtain the nice property that in our resulting graph structure of $\mathcal{S}(G)$, each face has a boundary that can be traversed, including the geometrically unbounded faces. This comes handy for straight-skeleton applications that need to traverse straight-skeleton faces, like for the computation of mitered offset curves.

4.2. *Runtime analysis*

Let us assume that $\mathcal{M}(G)$ is already given. (We will discuss the computation of the motorcycle graph later.) Computing $\mathcal{W}_G(0)$ is straightforward and can be done easily in at most $O(n \log n)$ time. In order to insert $\mathcal{M}(G)$ into $\mathcal{W}_G(0)$ we need to split edges of $\mathcal{W}_G(0)$ at crash points of motorcycles. As multiple motorcycles may have crashed in a single edge this requires sorting the crash points along wavefront edges. To sum up, the initial extended wavefront $\mathcal{W}^*_G(0)$ can be computed in $O(n \log n)$ time if $\mathcal{M}(G)$ is known.

The number of edge events is in $O(n)$ and the number start events and split events is in $O(r) \subseteq O(n)$. For each of those event we require to change $O(1)$ entries in the priority queue Q. Hence, all edge, split and start events are done in total $O(n \log n)$ time. Each multi split event (including the multi start event) involves the altering of $O(l)$ entries in Q, where l is the degree of the corresponding multi Steiner vertex. Hence, handling all multi split events is also done in total $O(n \log n)$ time. The number k of switch events is bound by $O(nr)$ as a convex vertex may meet the same moving Steiner vertex resp. reflex vertex at most once.

Lemma 6. *If $\mathcal{M}(G)$ is known then our algorithm computes $\mathcal{S}(G)$ of G in time $O((n + k) \log n)$ time and $O(n)$ space, where $k \in O(nr)$ denotes the number of switch events occurring.*

The worst-case runtime bound of $O(nr \log n)$ is tight and a corresponding polygon can be constructed. However, note that if a a convex vertex meets a reflex vertex then a wavefront edge vanishes. Hence, only $O(n)$ such switch events can happen. In order to attain the worst-case, we require $\Omega(r)$ moving Steiner vertices to meet $\Omega(n)$ convex vertices. This basically means that $\Omega(r)$ motorcycles moved all sufficiently parallel until hitting a polygonal chain of walls resulting in a sequence of moving Steiner vertices in $\mathcal{W}_G(0)$. After the sequence of moving Steiner vertices we require $\Omega(n)$ wavefront vertices in convex position that all move towards the $\Omega(r)$ Steiner vertices on the wavefront. Hence, this sequence of convex vertices are not allowed to cause (too many) edge events before producing $\Omega(nr)$ switch events. In other words, the convex vertices are sufficiently precisely lined up on a circular arc. To sum up, the worst-case scenario is highly contrived and unlikely to occur in real-world data. This hypothesis is confirmed impressively by our experiments run on 22 300 datasets of different characteristics; see Sec. 5.

In comparison with the triangulation-based algorithm by Aichholzer and Aurenhammer,[3] our algorithm is by a linear factor faster in the worst-case analysis. In particular, for convex polygons, our algorithm takes in the worst-case at most $O(n \log n)$ time instead of $O(n^2 \log n)$.

Computing the motorcycle graph. Using an enhanced brute-force method, $\mathcal{M}(G)$ can be computed in $O(r^2 \log r)$ time. In theory, the generalized motorcycle graph $\mathcal{M}(G)$ can be computed in $O(r^{17/11+\epsilon})$ time and space using the motorcycle-graph algorithm by Eppstein and Erickson.[12] Note that the $O(r\sqrt{r}\log r)$ motorcycle-graph algorithm by Cheng and Vigneron[10] cannot be applied to the generalized motorcycle graph $\mathcal{M}(G)$ as this algorithm is based on a $1/\sqrt{r}$-cutting of the motorcycle tracks, which all need to be known a-priori. In practice, we use our motorcycle-graph code Moca,[g] see Ref. 21. Moca runs in $O(r \log r)$ time on average, provided that the motorcycles are distributed uniformly enough. However, this is the case for the vast majority of real-world datasets.[21] Also note that outside the convex hull of G, the generalized motorcycle graph $\mathcal{M}(G)$ can be practically computed in $O(r \log r)$ worst-case time, as we showed in Ref. 21. That means, for convex polygons, our straight-skeleton algorithm runs in $O(n \log n)$ worst-case time, taking the computational costs of computing $\mathcal{M}(G)$ into account. (Of course, the straight skeleton and the Voronoi diagram of a convex polygon are identical and, thus, straight skeletons of convex polygons can be computed in $O(n)$ time, see Ref. 1.)

5. Experimental Results

Our implementation Bone is implemented in C++ and uses the STL[h] for ordinary data structures like maps (red-black binary trees), vectors (dynamic arrays) or priority queues. Bone has two numerical backends: by default it uses double-precision floating-point arithmetic and as an alternative it can be linked against the arbitrary precision library MPFR, cf. Ref. 18. Besides computing the straight skeleton $\mathcal{S}(G)$ of PSLGs G, our code can also generate mitered offset curves and export the terrain $\mathcal{T}(G)$, which can be further imported to a 3D modelling software.

We compared the performance of Bone against the straight-skeleton implementation that is shipped with CGAL-4.0, see Sec. 1.2. The CGAL implementation uses exact arithmetic and according to its documentation it is recommended to run the code with the exact-predicates-inexact-constructors kernel. For a polygon with holes we compute the straight skeleton of (i) the interior of the polygon, (ii) of its holes and (iii) the exterior of the polygon. The maximum offset distance for the exterior straight skeleton was set to 100 after the input was scaled to the unit square. (The scaling was done with floating-point arithmetic so that both Bone and the CGAL code get exactly the same input data.)

We tested both implementations against our database containing approximately 22 300 datasets. Our database includes both real-world and contrived data of different characteristics, including CAD/CAM designs, printed-circuit board layouts, font outlines, geographic maps, company logos, random polygons generated by RPG,[5]

[g]The algorithm and the implementation presented in Ref. 21 is general enough to compute the motorcycle graph $\mathcal{M}(G)$ as we introduced it in this paper.
[h]Standard template library.

space filling curves, fractal datasets, sampled ellipses, random star-shaped polygons, families of offset curves, and so on. Some datasets contain also circular arcs, which we approximated by a polygonal chain in our I/O routines. In any case, we considered simple input only, i.e., no line segment intersects another line segment in its relative interior. Furthermore, for tests with CGAL we only considered polygons with holes.

Our test machine runs a 64-bit Linux system on top of a X980 Core i7 processor and 24 GB of main memory. Both Bone and the CGAL code are single-threaded. We ran four parallel benchmark processes at the same time. Each run on a dataset was limited to 15 minutes runtime and 6 GB main memory by means of the `ulimit` command.

Runtime. We measured the time it took Bone and CGAL to compute the straight skeleton, not including time for I/O and similar pre- and post-processing. In Fig. 11 we plot the results. Each dot depicts the runtime of a single dataset with the size of the dataset on the x-axis and the runtime on the y-axis. As predicted in Sec. 4.2, Bone required $c \cdot n \log n$ seconds on virtually all datasets, where $4 \cdot 10^{-6} \leq c \leq 10^{-5}$. Further tests showed that about 20–50% of the total runtime is required by Moca for computing $\mathcal{M}(G)$. Furthermore, all outliers in Fig. 11 that show a quadratic runtime are due to Moca. We also performed runtime tests of Bone and Moca with the MPFR backend. Using the MPFR backend, we basically obtain the same runtime plots, but shifted by a certain factor. If we use 53 bits of precision (which is the size of the mantissa of IEEE 754 double-precision values) then performance drops approximately by a factor of 15 due to the overhead of MPFR. Using $4 \cdot 53 = 212$ resp. $16 \cdot 53 = 848$ bits, the performance drops approximately by a factor of 25 resp. 52. For higher precisions the performance drop increases roughly by a factor of three when the precision is doubled. (An increase by a multiplicative factor of three is plausible since simple tests showed that MPFR consumes roughly $O(k\sqrt{k})$ time for k-bit multiplications.)

The CGAL code required $c \cdot n^2 \log n$ seconds, with $2 \cdot 10^{-7} \leq c \leq 2 \cdot 10^{-6}$ for the majority of datasets. However, for a substantial number of datasets c grows up to roughly 10^{-4}. In other words, for datasets of the same size the runtime may vary within a range of two decades. Further tests using the inexact predicates kernel exhibit that this variation is due to the exact arithmetic backend. To sum up, our code Bone is by a linear factor faster than the CGAL implementation. In particular, for datasets of a moderate size of 10^4, our code is faster by a factor of approximately 300. Even if we run CGAL with an inexact kernel then Bone remains by two orders of magnitudes faster.

Memory footprint. We observed in our benchmarks that the CGAL implementation was not able to compute the straight skeleton of datasets comprising more than about 10^4 vertices within the limits of 15 minutes and 6 GB memory. A closer look revealed that the CGAL code has a very demanding memory footprint. Even if we

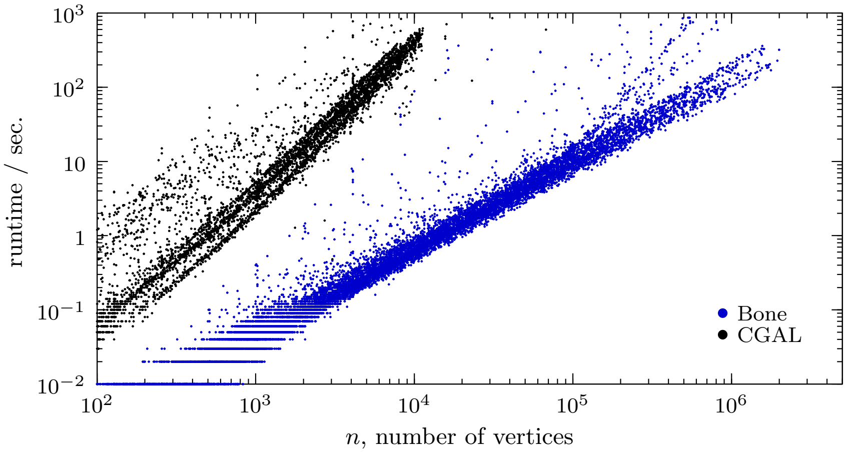


Fig. 11. The runtime of Bone and CGAL on 22 300 datasets. Bone has a $\Theta(n \log n)$ and CGAL a $\Theta(n^2 \log n)$ runtime on virtually all datasets.

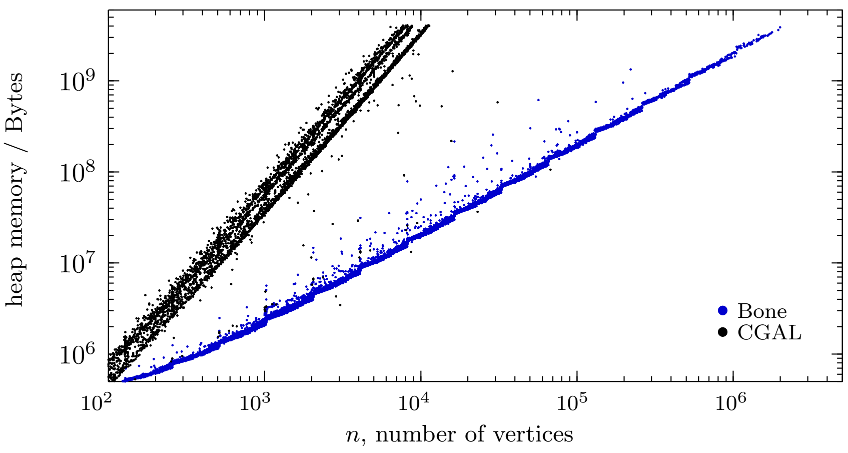


Fig. 12. The peak size of the heap memory for Bone and for CGAL. Bone has a linear and CGAL a quadratic memory footprint.

allowed the CGAL implementation to consume the entire 24 GB of main memory[i] then we could finish at best a random polygon with roughly 16 000 vertices.

This motivated us to use glibc's libmemusage facility in order to track memory allocation and frees. In Fig. 12 we plot the peak size of the heap memory for each dataset. As expected, Bone shows an $\Theta(n)$ memory footprint, while our tests

[i]Our test machine is a headless machine and no desktop environment is running on top of it. Hence, basically the entire memory can be used by CGAL resp. Bone.

of the CGAL implementation exhibit a quadratic memory footprint for basically all datasets. In a private email correspondence with F. Cacciola we learned that the CGAL code employs a priority queue that remembers for a quadratic number of pairs of reflex vertices and edges a potential split event in order to achieve an $O(n^2 \log n)$ runtime. This appears to be the reason why the CGAL code only managed to compute the straight skeleton of datasets with roughly 10^4 vertices, while Bone was able to handle datasets well beyond a million vertices within the 6 GB memory limit.

6. Engineering Details and Principles

Since Bone is based on standard double-precision arithmetic, we need to take special care for the numerical robustness of our computations. In the remainder of his work we summarize the principles we applied in order to make Bone as stable as it is.

First of all, the central data structure of Bone is a kinetic PSLG for the extended wavefront. The graph $\mathcal{W}_G^*$ is implemented as a doubly-linked edge list. The topology of $\mathcal{W}_G^*$ is described as follows: each vertex has a pointer to an incident edge and each edge has a pointer to its two vertices and for each of both ends a CW- and CCW-pointer to the neighboring edges in the circular order around the end point. Geometrically, each vertex has a start point (position at time 0) and a velocity. When adding an edge, it depends on the time t where the new edge is inserted into the circular order around a vertex.

6.1. *Avoiding geometric decisions*

A principle that was successfully applied in the past[16,28] is to prefer topology-only changes to $\mathcal{W}_G^*$ over changes that involve geometric information. For instance, let us consider an edge event for the edge $e = (u_1, u_2)$. Bone needs to remove e, u_1 and u_2, create a new vertex v, and re-link v with the former adjacent vertices v_1, v_2 of u_1 and u_2. The re-linking step involves geometric decisions concerning the circular order at v, v_1 and v_2.

A much more robust way to handle an edge event is the following: remove e, create the vertex v and repot the incident edges of u_1 and u_2 to v such that the topological information at v_1 and v_2 remains unaltered. The topological information at v is trivial, as v has degree two. Hence, an edge event can be handled without depending on any geometric decision. This is particularly important when multiple events happen at the same time — e.g., when adjacent convex vertices are aligned on a circular arc — and hence adjacent vertices of v_1 and v_2 overlap.

We casted the above idea into a more general procedure `patchWavefront(`e_1`,` u_1`,` e_2`,` u_2`,` S`)`. It creates a new vertex v and repots the wavefront edges e_1 and e_2 from u_1 and u_2 to v. Furthermore, S is a possibly empty list of pairs (v', e'), where e' is a Steiner edge incident v'. Depending on whether e' is on, left or right of the trajectory of v, the edge e' is repotted to v or to a moving Steiner vertex that splits e_1 resp. e_2. This procedure `patchWavefront` constitutes the basic building block for all events and reduces the overall code complexity.

6.2. *Special cases*

Like in virtually all abstract descriptions of a geometric algorithm, essential gaps to a real-world implementation often remain open. For the algorithm behind Bone, the following additions have been made:

- **Multi convex vertices.** According to the original description it easily happens that a convex vertex and a neighboring moving Steiner vertex share the same trajectory. (Consider a symmetric non-convex 4-gon as input.) A simple-minded implementation of the algorithm presented is doomed to fail in this case. For our implementation we introduce a new vertex type, *multi convex vertex*, which behaves like both, a moving Steiner vertex and a convex vertex.
- **Parallel collapse with Steiner edges.** According to Lemma 5, our algorithm deals with convex faces, which can collapse to a line. If such a collapse results in a parallel clash of a sequence of collinear wavefront edges with a sequence of collinear Steiner edges, we need to take special care.

  We implemented an extra procedure to resolve this situation. We first traverse the path P of collinear Steiner edges. Note that P in general comprises a sequence of resting Steiner vertices, which have degree three. We remove the Steiner edges along P and incorporate the remaining resting Steiner vertices, which are now of degree one, as moving Steiner vertices to the wavefront edges that clashed against P. In other words, we merge the collapsed face with the neighboring faces along P. If P also contains multi Steiner vertices then we need to handle the according multi start events.

6.3. *Avoiding the computation of vertex velocities*

In general, it is a bad idea to use the velocity vectors of vertices in order to determine the collapse time of wavefront edges. Figure 13 shows a dataset in which multiple edge events occur at the same time. Bone produces a straight skeleton whose geometry looks like Fig. 13(a) and whose topology looks like Fig. 13(b). Depending on the relative order of edge events at the bottom of the figure, the velocity of v depends on a lot of previously computed velocities. Hence, if we base our computations on vertex velocities, we accumulate numerical errors. Secondly, we need to deal with infinite speeds of wavefront vertices. For example, vertex v in Fig. 13(b) has infinite speed as it traces out an arc between parallel wavefront edges.

In our implementation, we compute the collapse time of e by inspecting its adjacent edges e_1 and e_2. Note that e collapses if u and v meet at the same point p, which means that the trajectories of u and v intersect at p. Hence, we can compute p by determining all points that are equidistant to $\overline{e_1(0)}$, $\overline{e(0)}$ and $\overline{e_2(0)}$. Exactly the same problem occurs when computing Voronoi nodes in a Voronoi diagram and

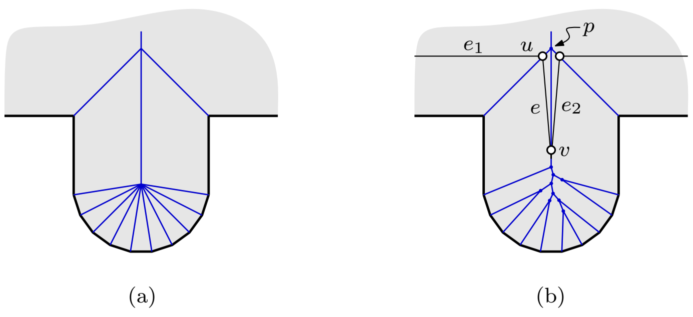


Fig. 13. Using vertex velocities for geometric computations is bad: they may be infinite and accumulate numerical errors from previous results of vertex speeds. Determining collapse times of wavefront edges by computing equidistant points to the supporting lines of three input edges is much more stable. (a) The geometric view. (b) The topological view.

involves computing the roots of a quadratic polynomial, see Ref. 17. Besides gaining better numerical stability, this method also resorts to the original input data instead of accumulating numerical errors of velocities, and it avoids dealing with infinite speeds.

7. Conclusion

In this paper we discuss both the theoretical foundations and practical engineering issues of our straight-skeleton implementation Bone. We start with an analysis of the triangulation-based algorithm by Aichholzer and Aurenhammer.[3] The main result of this analysis is the proof of the existence of Steiner triangulations which are free of so-called flip events. This result motivated us to generalize the motorcycle graph from non-degenerate simple polygons to arbitrary planar straight-line graphs. Our generalization was done in a careful way such that essential geometric relationships between the straight skeleton and the motorcycle graph are preserved.

The generalized motorcycle graph turns out to have important applications. First, we are finally able to give a non-procedural characterization of straight skeletons of planar straight-line graphs. Second, it motivates a straight-skeleton algorithm based on graphics hardware. Third, and most important, it builds the theoretical basis for a novel wavefront-type straight-skeleton algorithm.

We extend the ordinary wavefront by the generalized motorcycle graph. Our novel algorithm is built on the fact that any topological change of the extended wavefront is indicated by the collapse of an edge. The resulting algorithm is simple enough to be implemented. Our C++ implementation Bone was tested in extensive benchmarks on a database with 22 300 datasets. As predicted by the runtime analysis, Bone runs in $O(n \log n)$ time and $O(n)$ space in practice. This constitutes an improvement of a linear factor in time and space compared to the CGAL code. In particular, for datasets with a moderate size of 10^4 vertices, Bone requires 0.2–0.6 seconds using 20 MB of heap memory, while the CGAL code runs 4–7 minutes using

3–10 GB of heap memory. Moreover, Bone is able to handle planar straight-line graphs as input and copes with datasets with a few million vertices on a desktop PC, making Bone the current state-of-the-art implementation for computing straight skeletons.

Tests run on industrial data indicate that Bone is reliable. We discuss engineering aspects and algorithmic details used to boost Bone to industrial strength in the final section of our paper.

References

1. A. Aggarwal, L. J. Guibas, J. Saxe and P. W. Shor, A linear-time algorithm for computing the Voronoi diagram of a convex polygon, *Discr. Comput. Geom.* **4**(6) (1989) 591–604.
2. O. Aichholzer, D. Alberts, F. Aurenhammer and B. Gärtner, Straight skeletons of simple polygons, *Proc. 4th Int. Symp. LIESMARS*, Wuhan, P. R. China (1995), pp. 114–124.
3. O. Aichholzer and F. Aurenhammer, Straight skeletons for general polygonal figures in the plane, *Voronoi's Impact on Modern Science, Book* 2, ed. A. M. Samoilenko (Institute of Mathematics of the National Academy of Sciences of Ukraine, Kiev, Ukraine, 1998), pp. 7–21.
4. O. Aichholzer, H. Cheng, S. L. Devadoss, T. Hackl, S. Huber, B. Li and A. Risteski, What makes a tree a straight skeleton, *Proc. 24th Canad. Conf. Comput. Geom.* (*CCCG'*12), Charlottetown, P.E.I., Canada, August 2012, pp. 267–272.
5. T. Auer and M. Held, Heuristics for the generation of random polygons, *Proc. Canad. Conf. Comput. Geom.* (*CCCG'*96), Ottawa, Canada, August 1996, pp. 38–44.
6. G. Barequet, M. T. Goodrich, A. Levi-Steiner and D. Steiner, Contour interpolation by straight skeletons, *Graph. Models* **66**(4) (2004) 245–260.
7. F. Cacciola, A CGAL implementation of the straight skeleton of a simple 2D polygon with holes, *2nd CGAL User Workshop*, Polytechnic University, Brooklyn, New York, USA, June 2004.
8. F. Cacciola, private email correspondence (2010).
9. CGAL, Computational Geometry Algorithms Library, http://www.cgal.org/.
10. S.-W. Cheng and A. Vigneron, Motorcycle graphs and straight skeletons, *Algorithmica* **47** (2007) 159–182.
11. E. D. Demaine, M. L. Demaine and A. Lubiw, Folding and cutting paper, *Revised Papers from the Japan Conf. Discrete Computat. Geometry* (*JCDCG'*98), Lecture Notes Comput. Sci., Vol. 1763 (Digital Press, Tokyo, Japan, 1998), pp. 104–117.
12. D. Eppstein and J. Erickson, Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions, *Discr. Comput. Geom.* **22**(4) (1999) 569–592.
13. P. Felkel and Š. Obdržálek, Improvement of Oliva's algorithm for surface reconstruction from contours, *Proc. 15th Spring Conf. Comput. Graphics*, Budmerice, Slovakia, April 1999, pp. 254–263.
14. J.-H. Haunert and M. Sester, Area collapse and road centerlines based on straight skeletons, *GeoInformatica* **12** (2008) 169–191.
15. S. Havemann, Generative mesh modeling, PhD. thesis, TU Braunschweig, Braunschweig, Germany (2005).
16. M. Held, VRONI: An engineering approach to the reliable and efficient computation of Voronoi diagrams of points and line segments, *Comput. Geom. Theor. Appl.* **18**(2) (2001) 95–123.

17. M. Held and S. Huber, Topology-oriented incremental computation of Voronoi diagrams of circular arcs and straight-line segments, *Comput. Aided Design* **41**(5) (2009) 327–338.
18. M. Held and W. Mann, An experimental analysis of floating-point versus exact arithmetic, *Proc. 23rd Canad. Conf. Comput. Geom.* (*CCCG* 2011), Toronto, Canada, August 2011, pp. 489–494.
19. K. Hoff *et al.*, Fast computation of generalized Voronoi diagrams using graphics hardware, *Comput. Graphics* (*SIGGRAPH'99 Proc.*), Los Angeles, CA, August 1999, pp. 277–286.
20. S. Huber and M. Held, Straight skeletons and their relation to triangulations, *Proc. 26th Europ. Workshop Comput. Geom.*, Dortmund, Germany, March 2010, pp. 189–192.
21. S. Huber and M. Held, Motorcycle graphs: Stochastic properties motivate an efficient yet simple implementation, *ACM J. Exp. Algorithmics* **16** (2011) 1.3:1.1–1.3:1.17.
22. S. Huber and M. Held, Theoretical and practical results on straight skeletons of planar straight-line graphs, *Proc. 27th Annu. ACM Symp. Comput. Geom.*, Paris, France, June 2011, pp. 171–178.
23. T. Kelly and P. Wonka, Interactive architectural modeling with procedural extrusions, *ACM Trans. Graph.* **30**(2) (2011) 14:1–14:15.
24. R. Laycock and A. Day, Automatically generating roof models from building footprints, *Poster Proc. 11th Int. Conf. Comput. Graphics, Visualizat., Comput. Vision* (2003).
25. P. Müller, P. Wonka, S. Haegler, A. Ulmer and L. Van Gool, Procedural modeling of buildings, *ACM Trans. Graph.* **25** (2006) 614–623.
26. 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) (1996) 397–408.
27. S. C. Park and Y. C. Chung, Mitered offset for profile machining, *Comput. Aided Design* **35**(5) (2003) 501–505.
28. K. Sugihara and M. Iri, Construction of the Voronoi diagram for 'one million' generators in single-precision arithmetic, *Proc. IEEE* **80**(9) (1992) 1471–1484.
29. M. Tănase and R. C. Veltkamp, Straight line skeleton in linear time, topologically equivalent to the medial axis, *Proc. 20th Europ. Workshop Comput. Geom.*, March 2004.
30. E. Yakersberg, Morphing between geometric shapes using straight-skeleton-based interpolation, MSc. thesis, Technion – Israel Institue of Technology, May 2004.