Computing Motorcycle Graphs Based on Kinetic Triangulations*

Willi Mann† Martin Held† Stefan Huber‡

Abstract

We present an efficient algorithm for computing generalized motorcycle graphs, in which motorcycles are allowed to emerge after time zero. Our algorithm applies kinetic triangulations inside of the convex hull of the input, while a plane sweep is used outside of it. Its worst-case complexity is $O((n+f)\log n)$, where $f \in O(n^3)$ denotes the number of flip events that occur in the kinetic triangulation. Outside of the convex hull it runs in $O(n \log n)$ time. In order to reduce the number of flip events we investigate the use of Steiner triangulations. We prove the existence of Steiner triangulations that eliminate all flip events and discuss heuristics for approximating such a Steiner triangulation.

Extensive experiments with our C++ implementation run on thousands of datasets of various characteristics demonstrate a runtime of $c \cdot 10^{-6} \cdot n \log n$ seconds, with $c \leq 4$ for virtually all of our datasets. This constitutes a significant practical improvement over the motorcycle code Moca [Huber&Held 2011], which runs in $O(n \log n)$ time only if the motorcycles are distributed uniformly enough. In particular, our experiments yielded $f \leq 5n$ flip events for all but very few datasets.

1 Introduction

1.1 Motivation and Definitions

A *motorcycle* is a point that moves with constant velocity along a straight-line ray. Consider n motorcycles $m_1, \ldots, m_n$, each of them having a start point $p_i \in \mathbb{R}^2$ and a velocity $v_i \in \mathbb{R}^2$, with $1 \leq i \leq n$. (We assume that at most a constant number of motorcycles share one start point.) The *track* of motorcycle m_i is defined by the ray $\{p_i + t \cdot v_i \ : \ t \geq 0\}$. While a motorcycle moves it leaves a *trace* behind. A motorcycle *crashes* when it reaches the trace of another motorcycle: It stops driving but its trace remains. A motorcycle *escapes* if it never crashes. The motorcycle graph $\mathcal{M}(m_1, \ldots, m_n)$ is defined as the arrangement of all motorcycle traces.

Motorcycle graphs were introduced by Eppstein and Erickson [4] when investigating straight skeletons [1]. The basic motivation behind the definition of the motorcycle graph was to extract the essential sub-problem of computing straight skeletons and to cast it into a separate problem. In fact, it turns out that motorcycle graphs and straight skeletons share a strong relationship: (i) motorcycle graphs can be used to give a non-procedural characterization of straight skeletons [9, 2], (ii) the straight-skeleton algorithm by Huber and Held [9] and the algorithm by Cheng and Vigneron [2] are based on motorcycle graphs, and (iii) the $\mathcal{P}$-completeness of straight skeletons of polygons with holes follows from the $\mathcal{P}$-completeness of motorcycle graphs [7, 4]. In particular, the currently fastest straight-skeleton code Bone [9] employs the motorcycle graph in a preprocessing step.

In this paper, we present an algorithm for computing the motorcycle graph $\mathcal{M}(G)$ that is induced by a planar straight-line graph (PSLG) G. This requires a generalization of the original motorcycle graph, see Fig. 1.

First, we consider rigid *walls* formed by straight-line segments. If a motorcycle meets a wall then it crashes, too. Secondly, if two or more motorcycles $m_1, \ldots, m_k$ crash simultaneously into each other at the point p such that the traces of $m_1, \ldots, m_k$ lie in a half-plane whose boundary contains p then a new motorcycle m is launched from p in the complementary half-plane. In other words, a local disc at p is tessellated into convex slices by the traces of $m, m_1, \ldots, m_k$. To sum up, in generalized motorcycle graphs a motorcycle is specified by a start point, a velocity and a start time. A formal definition of $\mathcal{M}(G)$ involves concepts of straight skeletons, see [9] for further

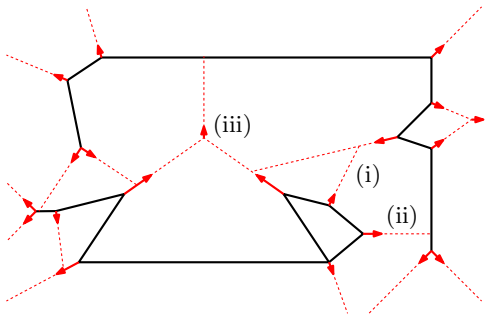

Figure 1: A generalized motorcycle graph. The motorcycles' velocities are depicted by (red) arrows. A motorcycle may crash against (i) another trace or (ii) a wall. (iii) A Motorcycle may be launched after two or more motorcycles crashed into each other.

*Work supported by Austrian FWF Grant L367-N15.

†FB Computerwissenschaften, Universität Salzburg, A–5020 Salzburg, Austria, {wmann,held}@cosy.sbg.ac.at

‡FB Mathematik, Universität Salzburg, A–5020 Salzburg, Austria, shuber@cosy.sbg.ac.at

details. For the reader to be able to follow this paper it suffices to note that not all motorcycles are known a priori, and that the motorcycles may crash into a total of $O(n)$ walls.

1.2 Prior Work

For the original setting of the motorcycle graph problem, the algorithm by Cheng and Vigneron [2] achieves the best worst-case complexity: it runs in $O(n\sqrt{n}\log n)$ time. In a preprocessing, they compute a $1/\sqrt{n}$-cutting on the supporting lines of the motorcycle tracks. However, this requires to know all motorcycles in advance and hence their algorithm is not applicable to generalized motorcycle graphs.

The algorithm by Eppstein and Erickson [4] is applicable to generalized motorcycle graphs. It employs various efficient closest-pair data structures in a hierarchical fashion. By a clever trade-off between time and space they achieve an $O(n^{17/11+\epsilon})$ time and space complexity. However, their algorithm is far too complex for an actual implementation.

A fairly simple recent algorithm by Huber and Held [8] uses a $\sqrt{n}\times\sqrt{n}$-grid to speed up computation. If the motorcycles are distributed uniformly enough then a motorcycle crosses only $O(1)$ grid cells on average, which leads to an $O(n\log n)$ runtime. The resulting motorcycle-graph code Moca has become to be known as the fastest implementation. While Moca works nicely for most datasets, it requires up to $O(n^2\sqrt{n}\log n)$ time for some contrived inputs, and $O(n^2\log n)$ time for densely sampled convex bodies.

1.3 Our Contribution

In Sec. 2 we present a novel motorcycle-graph algorithm for the computation of $\mathcal{M}(G)$ that consists of two phases. The first phase computes $\mathcal{M}(G)$ within the convex hull $CH(G)$ of the walls and the start points of all motorcycles. It is based on a kinetic triangulation, akin to [1]. Its worst-case time complexity is $O((n+f)\log n)$, where $f\in O(n^3)$ denotes the number of flip events that occur in the kinetic triangulation. The second phase uses a plane-sweep algorithm to compute $\mathcal{M}(G)$ outside of $CH(G)$ in time $O(n\log n)$. Thus, in the worst case the total complexity is $O(n^3\log n)$. However, no input is known that causes a runtime of more than $O(n^2\log n)$.

As the time complexity strongly depends on the number f of flip events, we investigate the use of Steiner triangulations for reducing f. In fact, we prove that Steiner triangulations exist for which no flip event occurs and for which our algorithm would take $O(n\log n)$ time. This motivates the search for practical heuristics to approximate such a Steiner triangulation.

We implemented our algorithm in C++ and report on implementational and numerical aspects. Extensive benchmarks on several thousand datasets clearly demonstrate an $O(n\log n)$ runtime. In particular, our experiments yielded $f\le 5n$ flip events for virtually all datasets. Additional experiments showed that our heuristics reduce the number of flip events by 20% on average. As our algorithm does not rely on a roughly uniform distribution of the motorcycles this constitutes a major practical improvement compared to the algorithm that drives Moca.

2 Computing the Generalized Motorcycle Graph

2.1 Computation Inside of Convex Hull $CH(G)$

To compute $\mathcal{M}(G)$ within $CH(G)$ we need to determine which motorcycle crashes into which trace or wall. (Motorcycles which start on the boundary of $CH(G)$ and do not move inwards are considered in the second phase of our algorithm, see Sec. 2.2.) The basic idea is to use a kinetic triangulation such that every crash event is indicated by the collapse of a triangle in the triangulation. This approach is motivated by the straight-skeleton algorithm by Aichholzer and Aurenhammer [1]. Thus, we start by computing the constrained Delaunay triangulation T within $CH(G)$, where the start points of the initial motorcycles and the endpoints of the walls form the vertices and the walls form the constrained diagonals of T.

In the next step, we insert at each start point p_i of an initially present motorcycle a duplicate vertex q_i, which represents the moving motorcycle. Thus, q_i will move away from p_i according to the speed vector v_i. (We get a function linear in t for the movement of q_i.) Initially, $q_i := p_i$. We call q_i a moving triangulation vertex. We also regard it as one of k moving triangle vertices if k triangles are incident at q_i. (This distinction will be useful in the complexity analysis, see Sec. 2.3.)

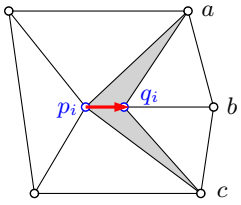

Figure 2: Illustration of initial split

In general, q_i will move into the interior of precisely one triangle $\Delta(p_i,b,a)$, and we replace $\Delta(p_i,b,a)$ by the two degenerate triangles $\Delta(p_i,q_i,a)$ and $\Delta(p_i,b,q_i)$, and by $\Delta(q_i,b,a)$. (All triangle vertices are always kept in counter-clockwise (CCW) order.) If q_i moves along the edge (p_i,b) of the non-degenerate triangles $\Delta(p_i,b,a)$ and $\Delta(p_i,c,b)$, see Fig. 2, then we replace these two triangles by the two degenerate triangles $\Delta(p_i,q_i,a)$ and $\Delta(p_i,c,q_i)$, and by $\Delta(q_i,b,a)$ and $\Delta(q_i,c,b)$. Similarly if both vertices of an edge correspond to start points of motorcycles and, thus, degenerate triangles are already present. In any case, this initial split of all moving triangle vertices from their start points results in the generation of only a constant number of new triangles per vertex.

We note that we may deviate from these strict rules as

long as the topology is not violated and no wall is altered. For instance, if q_i of Fig. 2 would move into the interior of $\Delta(p_i, b, a)$ but rather close to the edge (p_i, b) then we could still use the split depicted if (p_i, b) is no wall, thus avoiding to split off the sliver triangle $\Delta(p_i, b, q_i)$.

We start the event processing after all moving triangulation vertices have been split from their start vertices. A priority queue maintains a list of collapsing triangles as events, sorted by their collapse time. The three main types of events are flip event, crash event, and stop event. We discuss these events below. After all motorcycles have stopped moving, no further triangulation vertex moves and no triangle collapses. Thus, at that point in time the priority queue is empty and we can continue with the second part of the algorithm, see Sec. 2.2.

A *flip event* occurs when the vertex a of the triangle $\Delta(a, b, c)$ ends up on the edge (b, c) which is not a wall, motorcycle trace or edge of $CH(G)$. Within the quadrilateral formed with its neighbor $\Delta(b, d, c)$ on the other side of the edge (b, c), the diagonal is flipped such that the triangles $\Delta(a, d, c)$ and $\Delta(b, d, a)$ are generated, cf. Fig. 3. As no triangulation vertex changes its speed, all that remains to do is to update the priority queue by rescheduling the collapse events of the two triangles.

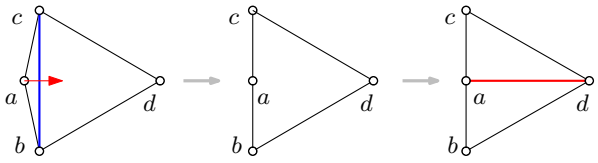

Figure 3: Handling of flip event.

A *crash event* occurs when a motorcycle, i.e., a moving triangulation vertex, reaches the trace of another motorcycle or a wall. A crash events manifests itself as a triangle collapse, which is handled similar to a flip event, except that the moving vertex needs to be halted. Of course, halting a moving vertex requires to re-schedule all triangles attached to this vertex. As a special case we get a vertex collapse if two vertices of a triangle become coincident. Vertex collapses lead to the removal of the edge between two vertices and their incident triangles, and the two vertices are fused to one. This can be seen as the reverse of the initial split, recall Fig. 2.

A *stop event* occurs when a motorcycle reaches $CH(G)$. The handling of this event is very similar in concept to crash events. The only difference is that stopped motorcycles are awakened again in the second part of the algorithm.

2.2 Computation Outside of Convex Hull $CH(G)$

We apply a generalized plane sweep, whose front is given by increasingly larger copies of $CH(G)$. This can be seen as a generalization of the approach sketched by Erickson [5] for motorcycles whose speed vectors do not span more than 180°. As the front advances the common vertex of two neighboring edges of $CH(G)$ moves along their outwards angular bisector. Hence, the exterior of $CH(G)$ is partitioned into individual sweep regions by the bisectors, with one region per edge of $CH(G)$.

All motorcycles that start on $CH(G)$ and move away from it or that were stopped during the first phase, when reaching $CH(G)$, are stored in cyclic order in a doubly-linked circular list. This list represents the front of the sweep. We do not need to maintain the front as a search tree since the only case where a new motorcycle needs to be inserted into the front after the initial set-up happens at a joint crash position of two or more motorcycles; in this case we have handles to the position, and do not need to search for the correct insert position.

During the sweep only one type of event needs to be handled: A *crash event* occurs when a motorcycle track intersects the motorcycle track of a neighboring motorcycle, where "neighboring" is defined relative to the sorted cyclic order of motorcycles in the front of the sweep. The motorcycle that is second at the intersection position is stopped. If two or more motorcycles meet in the same intersection position p at the same time, they are all stopped and a new motorcycle that moves further away from $CH(G)$ is inserted at p. (Recall that a local disc at p is tessellated into convex slices by the traces of the motorcycles according to our definition of $\mathcal{M}(G)$.)

The priority queue is sorted in increasing order of the distances of the event positions to $CH(G)$. Hence, all events are handled in the order as they occur relative to the front of the sweep, which is not necessarily the chronological order. In order to compute the distance to $CH(G)$ we use bisection on $CH(G)$ for determining the sweep region that contains the event position.

2.3 Complexity Analysis

Complexity of computation inside of $CH(G)$. The initial constrained Delaunay triangulation within $CH(G)$ contains $O(n)$ triangles and can be computed in $O(n \log n)$ time [3]. For each of the n initial motorcycles we create two new degenerate triangles in the initial split. Afterwards, one new motorcycle is only created if at least two motorcycles have crashed. Hence, the overall number of motorcycles (resp. moving triangulation vertices) and the number of triangles created at the start times of all motorcycles are in $O(n)$.

For each initial motorcycle we have to determine the triangle(s) that the motorcycle runs into. Since at most a constant number of motorcycles is allowed to share a starting point, we can determine all those triangles in $O(n)$ time by means of brute-force searches around each start vertex. And since the events for only a constant number of triangles need to be stored in the priority queue, the priority queue after all initial splits can be

set up in $O(n \log n)$ time.

While flip events do not affect the number of moving triangulation vertices, every flip may increase the number of moving triangle vertices by two: Assume that a and d of Fig. 3 are moving triangle vertices, while b and c do not move. Since after the flip a, d are counted as moving triangle vertices for each triangle on either side of (a, d), the number of moving triangle vertices has increased by two. Hence, f edge flips increase the number of moving triangle vertices by at most $2f$.

If a crash or stop event occurs for a moving triangulation vertex q then we have to re-schedule all triangles incident at q. Since we crash or stop a moving triangulation vertex at most once, the overall number of triangles that need to be re-scheduled equals the overall number of moving triangle vertices, which is in $O(n + f)$. Thus, the complexity of updating the priority queue for handling all crash and stop events is $O((n + f) \log n)$, which also models the worst-case complexity of the entire first phase of our algorithm.

Complexity of computation outside of $CH(G)$. The distance of one event position from $CH(G)$ can be determined in $O(\log n)$ time. The priority queue that stores the events defined by neighboring motorcycles is initialized in $O(n \log n)$ time. The handling of a crash event takes $O(k \log n)$ time, where k denotes the number of motorcycles stopped. Since each crash reduces the number of active motorcycles by at least one, the complexity of handling all crash events is $O(n \log n)$. Summarizing, the total complexity of the second phase of the algorithm is $O(n \log n)$.

One may wonder whether this complexity bound is tight. Consider n motorcycles which have their start points on the x-axis and whose speed vectors have positive y-coordinates. That is, all motorcycles move upwards in the direction of the positive y-axis. If the start points are given in sorted order relative to their x-coordinates then our plane-sweep algorithm requires $O(n \log n)$ time to compute the motorcycle graph. But can one do better? Note that if their sorted order is unknown then a $\Omega(n \log n)$ bound can be shown by a reduction to sorting: Assume that n distinct natural numbers $c_1, \ldots, c_n$ are given. Then we define for each c_i a motorcycle m_i starting at $(c_i, 0)$, with velocity $(-1, 2^{-c_i})$. This guarantees that each motorcycle crashes into the motorcycle starting left to it, except for the left-most motorcycle, which escapes. Hence, we can determine the sorted order of $c_1, \ldots, c_n$ in $O(n)$ time from $\mathcal{M}(m_1, \ldots, m_n)$.

Overall Runtime Complexity. The worst-case complexity is $O((n + f) \log n)$, where f denotes the number of flip events. A flip event requires the area of a triangle to become zero. As all moving triangulation vertices move with constant speeds along rays (until they stop), the signed area of a triangle can be expressed as a quadratic polynomial in time and, hence, a single triangle with three moving vertices can collapse at most at two single points in time. Similarly if one or two vertices have been stopped. Hence, by an argument similar to [1, Lem. 5], we get a trivial $O(n^3)$ bound on the number of flip events. This gives $O(n^3 \log n)$ as total worst-case complexity. However, note that we are not aware of any input that leads to $\omega(n^2)$ flip events. (But one can design inputs that exhibit $\Theta(n^2)$ flip events.)

3 Heuristics for Reducing the Number of Flip Events

It is known that a PSLG G and its motorcycle graph $\mathcal{M}(G)$ partition the plane into a set of convex polygons. Suppose that we overlay $\mathcal{M}(G)$ and G, and triangulate the resulting convex polygons arbitrarily. The edges of $\mathcal{M}(G)$ are called Steiner tracks, and their final points are called Steiner points. Then each motorcycle would move along a triangulation edge, and because no other triangulation edge crosses its track, its movement does not require any edge to be flipped to let it pass through the triangulation. Triangulation edges never leave the convex polygon they were created in because they are always stopped when their incident vertices hit a Steiner point. Thus, for this triangulation our algorithm would be free of flip events.

Of course, the crash positions of the motorcycles are not known to us. But we can try to approximate a portion of the unknown Steiner tracks, in an attempt to reduce the number of flip events by enabling motorcycles to move along triangulation edges.

If Steiner points are present in the triangulation then we handle a point collapse between a moving triangulation vertex (motorcycle) and a Steiner point as follows: We stop the motorcycle at the Steiner point, which is handled like any other vertex collapse, and restart it with the same method as used for the initial split.

In our first heuristic we exploit the average track length of n motorcycles that start from within the unit square, as established in [8]: for each motorcycle we insert a line segment (in the direction of its movement) of length $c/\sqrt{n}$ as Steiner track, for some constant $c > 0$. The second heuristic inserts Steiner tracks of unlimited length for $c \cdot \sqrt{n}$ randomly chosen motorcycles. In both heuristics a Steiner track is terminated at the first intersection if it crosses a wall or intersects $CH(G)$. A intersection between Steiner tracks is resolved by adding an additional Steiner point at the intersection. Due to our choice of the Steiner tracks one may assume for both heuristics that at most $O(n)$ points of intersection need to be added to the input as Steiner points.

Stopping Steiner tracks at walls and $CH(G)$ is done by running a plane sweep twice, once top-down, once bottom-up. Walls and the convex hull segments are in-

serted as normal line segments, but each motorcycle is only inserted in the phase that fits its direction. When a motorcycle first intersects a wall or convex hull segment, it is removed and a Steiner point is placed at the intersection. A third plane sweep is done to resolve intersections between Steiner tracks, also adding Steiner points at the intersections.

4 Implementational Issues

4.1 Simultaneous and Out-of-Order Events

A standard problem of any algorithm that uses a kinetic data structure is the reliable computation of the times when the structure changes. This problem is known as "root sorting", i.e., determining which root of which polynomial occurs first. If root sorting is not guaranteed to be exact, e.g., due to the use of floating-point arithmetic, then some form of out-of-order processing of events is required in order to achieve reliability.

We note that our algorithm has to cope with a second problem in addition to the handling of out-of-order events: So far we have ignored the fact that events can occur simultaneously for real-world data. Consider Fig. 4, and suppose that the two vertices a and d are non-moving vertices, while the vertices b and c represent moving motorcycles that meet the edge (a, d) at the same time. Further assume that (a, d) is a normal triangulation edge, rather than a wall or motorcycle trace that would stop the motorcycles b and c. As the collapse times of T_1 and T_2 are identical, it is a matter of chance which flip event is processed first. If the event associated with T_2 is handled first then we flip the diagonal (b, d) to the diagonal (a, c), resulting in the triangles $\Delta(a, b, c)$ and $\Delta(a, c, d)$. If we now happen to choose triangle $\Delta(a, b, c)$ as next triangle to process then that diagonal will just flip back, and we have ended up in a loop. We emphasize that this problem occurs both on floating-point and exact arithmetic.

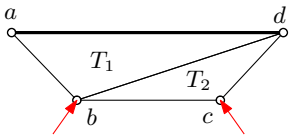

Figure 4: Simultaneous collapse of two triangles.

In order to handle simultaneous and out-of-order events we employ a strategy described in [10]. Roughly, a history of all events processed so far allows to detect a loop. If a loop is encountered then all events of the loop are considered to occur exactly at the same time and replaced by a set of events that guarantee progress.

4.2 Numerical Aspects

As noted, the computation of the collapse times is a challenging problem when using a floating point arithmetic. While the movement of each motorcycle is described by a linear function in time t, the signed area of a triangle with two moving vertices becomes a quadratic function in t.

Since we keep the vertices of all triangles in CCW order, the signed area of a triangles always is positive until the triangle collapses. Note, however, that in the case of a vertex collapse the area of a triangle may be positive again after the collapse time plus some positive epsilon. Fortunately, the degree of the polynomial which describes a vertex collapse can be reduced: We note that the time of a vertex collapse can also be calculated by a linear function, as it corresponds to the minimum of a function modeling the distance of two points moving with constant speeds along two lines. Similarly, all other events where a motorcycle is stopped involve triangles where only one vertex, namely the motorcycle being stopped, is moving. So the collapse times of events that stop motorcycles can be obtained by solving linear equations.

5 Experimental Results

We implemented our algorithm and both heuristics for reducing the number of flip events in C++. Shewchuk's Triangle [11] is employed for computing the initial constrained Delaunay triangulation.

The following tests were conducted on a Intel Core i7 X980 CPU clocked at 3.33 GHz, with Ubuntu 10.04.4 LTS and GCC version 4.4.3. The memory usage was limited to 4.5 GB, and the runtime on each file was limited to 15 minutes by means of the `ulimit` command. We computed generalized motorcycle graphs for both real-world and contrived data of different characteristics. In order to avoid unreliable timings and other idiosyncrasies of small datasets, we only analyze test runs that involved at least 1 000 motorcycles, resulting in a few thousand tests covered by our experiments.

The left plot of Fig. 5 shows the runtimes of our code divided by $n \log n$, where n denotes the number of vertices. The plot shows clearly that the runtime (in seconds) can be modeled by the function $c \cdot 10^{-6} \cdot n \log n$, with $c \leq 4$ for virtually all of our inputs.

This runtime behavior suggests a linear number of flip events, which is confirmed by our tests. The right plot of Fig. 5 shows the number f of flip events divided by n. As can be seen, we get $2n$ flip events on average and $0.8n$ to $5n$ flip events for virtually all inputs. The maximum number of flip events recorded was $39n$ for an input with roughly $n = 60\,000$ motorcycles.

A closer inspection of the test results reveals that an increased runtime of our code is caused by either an abnormal runtime of Triangle [11], which is used to compute the initial constrained Delaunay triangulation, or by an increased number of flip events, or by a combination of both. For instance, for most inputs Triangle consumes about 5–20% of the total runtime, but we witnessed in-

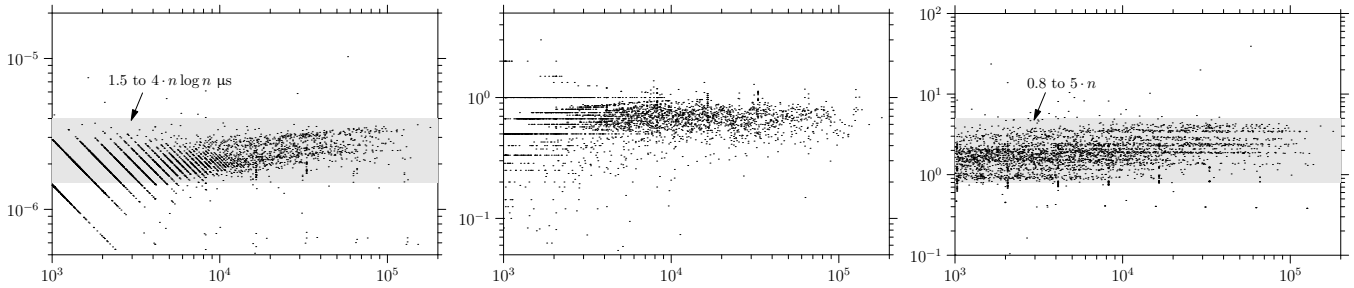

Figure 5: Experimental results for our code without Steiner tracks: In all three plots the x-axis corresponds to the number n of vertices. The left plot shows the runtimes divided by $n \log n$. The middle plot shows the ratio of the runtimes of our code divided by the runtimes of Moca, and the right plot shows the number of flip events divided by n.

puts for which it consumed 85% of the total runtime.

The fact that our code truly runs in $O(n \log n)$ time for most data is also confirmed by a comparison with Moca [8]. The middle plot of Fig. 5 shows the runtimes of our code divided by the runtimes of Moca. On average, our code needs about 32% less runtime than Moca. It is rarely slower than Moca, being at most twice as slow. However, our code is substantially faster for those inputs which cause Moca to consume $O(n^2 \log n)$ time.

Our heuristics for reducing the number of flip events turned out to be a mixed blessing. While a combination of both heuristics did indeed manage to reduce the number of flip events by about 20%, the preprocessing necessary for computing the intersections among the Steiner tracks and with the walls caused the runtime to increase. Apparently, performing plane sweeps is significantly more costly than what our heuristics manage to save by reducing the number of flips.

We also tested our code with the MPFR library [6] for multiple-precision computations, and witnessed an average slow-down by a factor of 25 for an MPFR precision of 212. (Plots are omitted due to lack of space.)

6 Conclusion

We developed and implemented a triangulation-based algorithm for the computation of generalized motorcycle graphs. While its theoretical worst-case time complexity is worse than prior art, our experiments demonstrate that it runs in $O(n \log n)$ time for virtually all inputs. Our new algorithm is an improvement over Moca as it clearly outperforms Moca in our runtime tests and its runtime does not depend on a sufficiently uniform distribution of motorcycles. Our experiments also show that our algorithm requires only $O(n)$ flip events in practice, and that this number can be reduced by the use of Steiner tracks. It remains an interesting problem to come up with more sophisticated methods to place Steiner tracks. After all, if the number of flip events could deterministically be reduced to $O(n)$ then our algorithm would run in optimal $O(n \log n)$ worst-case time.

References

[1] O. Aichholzer and F. Aurenhammer. Straight skeletons for general polygonal figures in the plane. In A. 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.

[2] S.-W. Cheng and A. Vigneron. Motorcycle graphs and straight skeletons. *Algorithmica*, 47:159–182, Feb 2007.

[3] L. Chew. Constrained Delaunay triangulations. *Algorithmica*, 4(1):97–108, 1989.

[4] 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.

[5] J. Erickson. Crashing motorcycles efficiently. `http://compgeom.cs.uiuc.edu/~jeffe/open/cycles.html`.

[6] GNU. The GNU MPFR library. `http://www.mpfr.org/`.

[7] S. Huber and M. Held. Approximating a motorcycle graph by a straight skeleton. In *Proc. 23rd Canad. Conf. Comput. Geom. (CCCG 2011)*, pages 261–266, Toronto, Canada, Aug 2011.

[8] S. Huber and M. Held. Motorcycle graphs: stochastic properties motivate an efficient yet simple implementation. *ACM J. Experimental Algorithmics*, 16:1.3:1.1–1.3:1.17, May 2011.

[9] 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.*, pages 171–178, Paris, France, June 2011.

[10] P. Palfrader, M. Held, and S. Huber. On computing straight skeletons by means of kinetic triangulations. In *Proc. ESA'12*, to appear Sep 2012.

[11] J. Shewchuk. Triangle: engineering a 2D quality mesh generator and Delaunay triangulator. In *1st ACM Workshop Appl. Comput. Geom.*, pages 124–133, Philadelphia, PA, USA, May 1996.