Motorcycle Graphs: Stochastic Properties Motivate an Efficient Yet Simple Implementation

STEFAN HUBER and MARTIN HELD, Universität Salzburg

In this article, we study stochastic properties of a geometric setting that underpins random motorcycle graphs and use it to motivate a simple but very efficient algorithm for computing motorcycle graphs. An analysis of the mean trace length of n random motorcycles suggests that, on average, a motorcycle crosses only a constant number of cells within a $\sqrt{n} \times \sqrt{n}$ rectangular grid, provided that the motorcycles are distributed sufficiently uniformly over the area covered by the grid. This analysis motivates a simple algorithm for computing motorcycle graphs: We use the standard priority-queue–based algorithm and enhance it with geometric hashing by means of a rectangular grid. If the motorcycles are distributed sufficiently uniformly, then our stochastic analysis predicts an $O(n \log n)$ runtime. Indeed, extensive experiments run on 22,000 synthetic and real-world datasets confirm a runtime of less than $10^{-5} n \log n$ seconds for the vast majority of our datasets on a standard PC. Further experiments with our software, Moca, also confirm the mean trace length and average number of cells crossed by a motorcycle, as predicted by our analysis. This makes Moca the first implementation that is efficient enough to be applied in practice for computing motorcycle graphs of large datasets. Actually, it is easy to extend Moca to make it compute a generalized version of the original motorcycle graph, thus enabling a significantly larger field of applications.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Geometrical problems and computations*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Geometric hashing, motorcycle graph, stochastic analysis

ACM Reference Format:
Huber, S. and Held, M. 2011. Motorcycle graphs: Stochastic properties motivate an efficient yet simple implementation. ACM J. Exp. Algor. 16, 3, Article 1.3 (August 2011), 17 pages.
DOI = 10.1145/1963190.2019578 http://doi.acm.org/10.1145/1963190.2019578

1. INTRODUCTION

1.1. Motivation

A motorcycle is a point moving on a straight line with a given start point and a fixed velocity. Consider n motorcycles $m_1, \ldots, m_n$ and denote by v_i the start point and by s_i the velocity of the motorcycle m_i. We call the ray $\{v_i + t \cdot s_i \ : \ t \geq 0\}$ the track of m_i. While each motorcycle drives, it leaves a trace behind. However, when a motorcycle m_i reaches the trace of another motorcycle m_j, then it stops driving, but the trace remains. We say that m_i crashed into m_j. The arrangement of those traces is commonly known as motorcycle graph [Eppstein and Erickson 1999] (Figure 1). Not every motorcycle necessarily crashes into another motorcycle. Motorcycles that never crash are said to "escape." Note that there can be up to $\Theta(n^2)$ intersections among motorcycle tracks,

This work is supported by Austrian FWF Grant L367-N15.
Author's addresses: S. Huber and M. Held, FB Computerwissenschaften, Universität Salzburg, 5020 Salzburg, Austria; email: {shuber, held}@cosy.sbg.ac.at.

DOI 10.1145/1963190.2019578 http://doi.acm.org/10.1145/1963190.2019578

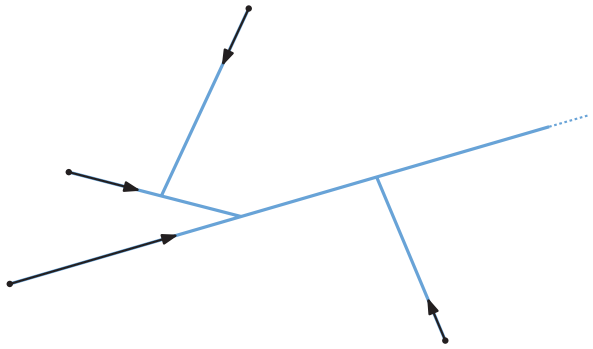

Fig. 1. A motorcycle graph of four motorcycles specified by their start points and velocities.

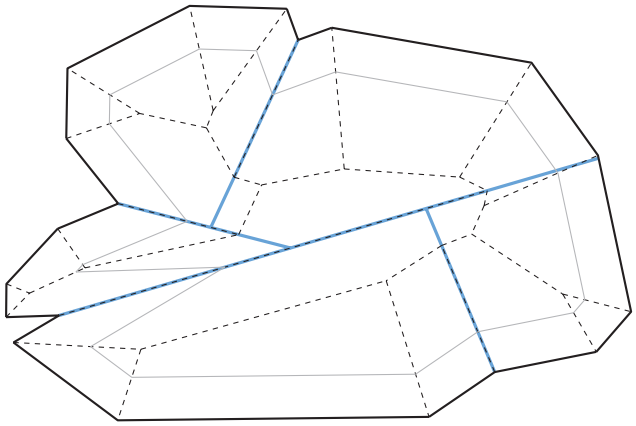

Fig. 2. A simple polygon and its straight skeleton (dashed). The reflex vertices of the polygon define the start points of the motorcycles. Their velocities are determined by the speed of the offset propagation and, therefore, depend on the interior angles at the reflex vertices. One sample offset polygon is depicted by thin grey lines. The corresponding motorcycle graph is superimposed and is identical to one given in Figure 1, but restricted to the interior of the polygon.

but there are at most n intersections among the traces. Moreover, two traces never intersect in the interior of both, since either one motorcycle crashed into the other or vice versa.

Motorcycle graphs bear a close relation to straight skeletons for several reasons: First, it can be shown that those parts of the straight skeleton of a simple polygon which emanate from reflex[1] vertices are a subset of the motorcycle graph, where the motorcycles correspond to the moving reflex vertices during the offset propagation process [Cheng and Vigneron 2007] (Figure 2). Second, the fastest algorithm known for computing the straight skeleton of simple polygons [Cheng and Vigneron 2007] uses the motorcycle graph as a preprocessing step. Furthermore, the straight-skeleton algorithm given in Eppstein and Erickson [1999] exploits the same technique as for the motorcycle graph algorithm given in that paper. Recently, Huber and Held [2011] presented a straight-skeleton algorithm for planar straight-line graphs, which is based on the motorcycle graph.

[1]A vertex is reflex if the interior angle is greater than π.

Straight skeletons, on the other hand, are a structure similar to Voronoi diagrams and, therefore, share a large number of applications, with straight-line offsetting of simple polygons being just one illustrative example [Aichholzer et al. 1995], (see Figure 2). But the list of applications also includes roof and terrain generation, mathematical origamis, polygon decomposition, surface reconstruction, and a few more applications in geographical information systems (GIS).

Besides straight skeletons, other problems are also related to (possibly generalized versions of) motorcycle graphs. Obviously, motorcycle graphs have strong relations to ray shooting problems. Ishaque et al. [2009] present an $O(n \log^2 n + m \log m)$ algorithm for m repetitive ray shooting-and-insertion operations in the plane among a set of polygonal obstacles of total size n. As shown in Section 3.1, our algorithm computes a generalized version of motorcycle graphs: Arbitrary start times of the motorcycles are allowed, the motorcycles need not be known a priori, and we support rigid walls where motorcycles may crash against. In this sense, the general motorcycle graph problem is also a generalization of the ray shooting-and-insertion problem. The algorithm by Czyzowicz et al. [1989] for solving an art gallery problem makes use of a straight-line structure of growing segments and, thus, is also related to motorcycle graphs. (Their structure can be interpreted as a motorcycle graph where all motorcycles drive at the same speed.) In Eppstein et al. [2008], motorcycle graphs on quadrilateral meshes are considered. The motorcycle graph is used to obtain a canonical partition of a mesh and to find isomorphisms between such meshes efficiently or to find a compressed representation of the original mesh. Even though the motorcycles do not drive on a plane but on the edges of a mesh, the basic concept for the interaction between moving motorcycles remains the same. We refer to Eppstein et al. [2009] for more details on topological matches and graph isomorphisms of quadrilateral meshes.

1.2. Prior Work

Motorcycle graphs can be computed by a straightforward brute-force algorithm: One simply determines crashes of motorcycles in chronological order by pairwise checking for the very next crash. By employing a priority queue for maintaining the chronologically ordered potential crash events, the complexity of this algorithm can be brought from $O(n^3)$ to $O(n^2 \log n)$. In the remainder of this article, we will refer to this algorithm as the standard priority-queue–based algorithm.

The algorithms by Eppstein and Erickson [1999] and Cheng and Vigneron [2007] are the only two subquadratic algorithms for computing motorcycle graphs known so far. Eppstein and Erickson lift the problem to $\mathbb{R}^3$ by tilting the n motorcycle traces at their start point upward, according to their velocities, and then reduce the problem to closest-pair queries. Their algorithm was the first one with subquadratic time complexity, namely $O(n^{17/11+\epsilon})$. The algorithm and its subalgorithms are quite complicated, and no implementation is known.

The key idea of the approach of Cheng and Vigneron [2007] is as follows: There are $O(n^2)$ many intersections of the tracks of the motorcycles, but only $O(n)$ of them realize a crash. Consequently, Cheng and Vigneron reduce the complexity of interactions among the motorcycles by employing a so-called $1/\sqrt{n}$-cutting, which is a partition of $\mathbb{R}^2$ into a set of simplices. This cutting has the powerful property that no more than $O(\sqrt{n})$ rays intersect a single simplex. (In Chazelle [1993], an algorithm is presented which can compute a cutting of size $O(n)$ in $O(n\sqrt{n})$ time.) Basically, the algorithm by Cheng and Vigneron is a discrete simulation of the movement of the motorcycles on the cutting. The simulation handles two types of events: crash events and switch events. A crash event indicates a crash of a motorcycle into the trace left behind by another motorcycle, while a switch event indicates a switch of a motorcycle from one simplex of the cutting

to a neighboring one. Hence, there are $O(n)$ crash events, and it can be shown that there are not more than $O(n\sqrt{n})$ switch events. In the course of the simulation, crash and switch events are inserted into a priority queue and the algorithm iteratively fetches the earliest event and processes it. Cheng and Vigneron manage to process a crash event in $O(\sqrt{n}\log n)$ time and a switch event in amortized $O(\log n)$ time, leading to an algorithm consuming $O(n\sqrt{n}\log n)$ time and $O(n)$ space in total. To obtain these complexities, they introduce arrangements of motorcycle traces on every simplex that support inserting a new trace in $O(\log n)$ amortized time. Implementing the algorithms for the $1/\sqrt{n}$-cutting and the local arrangements might be easier than tackling the algorithm by Eppstein and Erickson, but it would definitely require a lot of nontrivial work. In any case, we are not aware of any implementation of the algorithm by Cheng and Vigneron.

In summary, no algorithm for computing motorcycle graphs is known that realizes the easy-to-prove $\Omega(n\log n)$ lower bound. Furthermore, the two known subquadratic algorithms are too complicated to be implemented.[2] That is, no competitive algorithm and implementation exists that is efficient enough to be applied in practice for computing motorcycle graphs of large datasets.

1.3. Our Contribution

The contribution of this article is twofold. In Section 2, we use a stochastic analysis of the geometric setting that underpins random motorcycle graphs to estimate the mean trace length of a motorcycle. It turns out that, on average, we may expect a motorcycle to cross only a constant number of cells within a $\sqrt{n}\times\sqrt{n}$ rectangular grid that covers the unit square, provided that the n start points and directions of the motorcycles are distributed sufficiently uniformly. Equivalently, the mean length of a motorcycle trace can be expected to be in $\Theta(1/\sqrt{n})$.

These results motivate our second contribution. In Section 3, we present a simple yet competitive algorithm for computing motorcycle graphs: We use the priority-queue–based algorithm and enhance it with geometric hashing by means of a rectangular grid. If the start points of n motorcycles are distributed sufficiently uniformly, then our stochastic analysis predicts an $O(n\log n)$ runtime. Indeed, extensive experiments with our implementation Moca confirm an $O(n\log n)$ expected complexity. For our tests, we generated motorcycles from polygonal data by firing off rays at the vertices, with the input size n ranging from a few hundred to several millions of motorcycles. This is motivated by our straight-skeleton implementation Bone [Huber and Held 2011] and the straight-skeleton algorithm by Cheng and Vigneron [2007]. For both algorithms the motorcycle graph induced by polygonal input (planar straight-line graphs in general) is a key ingredient (see Figure 2). In a nutshell, if we are given a simple polygon, then we define for each reflex vertex v a motorcycle that starts at v, drives on the bisector of the incident edges and has a speed given by $\frac{1}{\sin\alpha/2}$, where α is the angle spanned by the edges incident to v. The velocity of this motorcycle corresponds to the propagation speed of the vertex v, when the incident edges are considered to move with unit speed and in a self-parallel fashion. See details in Section 4.2. (The definition of the straight skeleton is based on such a "wavefront-propagation" process.)

Our polygonal database consists of about 22,000 datasets containing both (highly) contrived data and real-world data (e.g., medical scans, CAD models, GIS maps of roads and river networks, polygonal outlines of fonts). It turns out that Moca processes the vast majority of our datasets within less than $10^{-5}n\log n$ seconds on a standard PC. It is

[2]In personal communication with David Eppstein and Siu-Wing Cheng, we learned that they also do not know of implementations of their algorithms.

interesting to note that this bound on the runtime holds even for contrived data where the vertices (and, thus, also the motorcycles) are distributed highly nonuniformly, such as for very fine approximations of spline curves.

This makes Moca the first implementation that is efficient enough to be applied in practice for computing motorcycle graphs of large real-world datasets. Actually, it is easy to extend Moca to make it compute a generalized version of the original motorcycle graph, thus enabling a significantly larger field of applications: Moca is able to cope with simultaneous crashes and motorcycles that do not all start driving at the same time, and it can handle rigid walls, where motorcycles may crash against. Moreover, it would be easy to extend the algorithm even further, for example, by considering motorcycles that run out of fuel or drive along more general curves.

2. STOCHASTIC PROPERTIES OF MOTORCYCLE GRAPHS

2.1. Number of Intersections of Bounded Rays

We start with gaining insight into the mean trace length of a motorcycle for a given distribution of a set of motorcycles. Let us recall that there can be up to $\Theta(n^2)$ intersections among the tracks of the motorcycles, but only $O(n)$ intersections among their traces. We consider n motorcycles starting in the unit square with uniformly distributed start points $v_1, \ldots, v_n$, unit speed and direction angles $\varphi_1, \ldots, \varphi_n$ uniformly distributed on $[0, 2\pi)$. Directly computing the expectation of the trace length of a motorcycle appears to be difficult due to the stochastic dependencies on the other motorcycles. However, it seems that the motorcycle traces do not get too long (on average), since we know that two traces do not intersect in the interiors of each other.

In order to simplify the original question for the mean trace length, we instead investigate the number of intersections among rays with finite lengths, where the lengths are chosen at random within $[0, 0.2]$ according to some density function. (Bounding the length of the rays to 0.2 is a technicality that simplifies the analysis.) The idea is that if the number of expected intersections of these rays is bounded, then the mean length of the bounded rays can be bounded as well.

THEOREM 2.1. *Let $v_1, \ldots, v_n$ be n points, which are uniformly i.i.d.*[3] *on the unit square $[0, 1]^2$, and $\varphi_1, \ldots, \varphi_n$ be n angles i.i.d. on $D := \{\delta_1, \ldots, \delta_d\}$, with $d \in \mathbb{N}$ and $\delta_i \in [0, 2\pi)$ occurs with probability p_i and $\sum_i p_i = 1$. Further, let $L_1, \ldots, L_n$ be i.i.d. on $[0, 0.2]$ according to a probability density function f.*

For every $i \in \{1, 2, \ldots, n\}$, consider a bounded ray $T_i \subset \mathbb{R}^2$, which starts at v_i, has direction angle φ_i, and length L_i. We denote by

$$I = \sum_{i=2}^{n} 1_{T_1 \cap T_i \neq \emptyset}$$

the number of intersections of T_1 with $T_2, \ldots, T_n$, where 1_P denotes the indicator function of the predicate P. Then,

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

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

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

[3] A common shorthand for "independent and identically distributed."

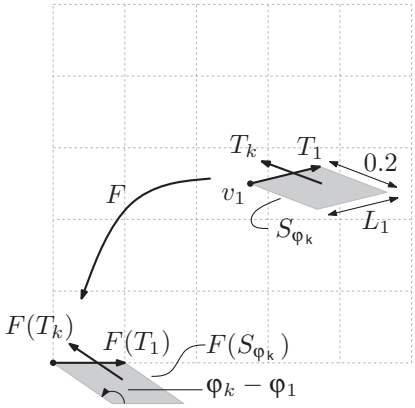

Fig. 3. The big 5×5 grid illustrates $[0,1]^2$ with the origin at the left bottom point. We see that $\lambda(S_{\varphi_k}) = L_1 \cdot 0.2 \cdot |\sin(\varphi_k - \varphi_1)|$, where λ denotes the Lebesgue measure.

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

$$E[I] = \sum_{i=1}^{d} P(\varphi_1 = \delta_i)\, E[I \mid \varphi_1 = \delta_i] = \sum_{i=1}^{d} p_i\, E[I \mid \varphi_1 = \delta_i]. \tag{3}$$

Consider the ray T_1 fixed. In order to have a ray T_k intersect T_1, the start point v_k of T_k needs to start in a certain area whose shape depends on the direction φ_k of T_k. For some arbitrary direction φ, we denote this area by a parallelogram S_φ, hinged at v_1 (Figure 3):

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

The mapping

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

models the translation of v_1 to the origin and subsequent clockwise rotation by the angle φ_1. Hence, $F(T_1)$ starts at the origin and points rightwards, (see Figure 3).

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

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

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

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

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

and, therefore,

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

where A_i denotes the event that $\varphi_1 = \delta_i$ and $v_1 \in [0.4, 0.6]^2$. Summing up over all i according to Equation (3) gives

$$\frac{1}{25} \cdot \sum_{i=1}^{d} p_i E[I|A_i] \;\leq\; E[I] \;\leq\; 1 \cdot \sum_{i=1}^{d} p_i E[I|A_i]. \tag{4}$$

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

$$\begin{aligned} E[I|A_i] &= \sum_{j=1}^{d} E[I_j|A_i] \\ &= \sum_{j=1}^{d} 25 \int_{[0.4,0.6]^2} \int_0^{0.2} \sum_{n_1+\cdots+n_d=n-1} \\ &\qquad \binom{n-1}{n_1, \ldots, n_d} p_1^{n_1} \cdots p_d^{n_d} E[I_j|B_{i,j,n_j}] \, \mathrm{d}\, f(L_1) \mathrm{d} v_1. \end{aligned} \tag{5}$$

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

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

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

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

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

Next, we substitute y_2' by $z := y_2'/|\sin(\delta_j - \delta_i)|$ and get

$$\begin{aligned} E[I_j|A_{i,j,l}] &= l\frac{|\sin(\delta_j - \delta_i)|}{\lambda(S_{\delta_j})}L_1\int_0^{0.2} P(L_2 \geq z)\,\mathrm{d}z \\ &= l\frac{|\sin(\delta_j - \delta_i)|L_1}{\lambda(S_{\delta_j})}E[L_2] \\ &= 5lE[L_1]. \end{aligned} \tag{6}$$

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

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

In the final step, we plug this result into Equation (5) and get

$$\begin{aligned} E[I|A_i] = 25\int_{[0.4,0.6]^2}\mathrm{d}v_1 \cdot E[L_1]\int_0^{0.2} L_1\,\mathrm{d}f(L_1)\cdot\sum_{j=1}^{d}|\sin(\delta_i - \delta_j)| \\ \sum_{n_1+\cdots+n_d=n-1} n_j\binom{n-1}{n_1,\ldots,n_d}p_1^{n_1}\cdots p_d^{n_d}. \end{aligned} \tag{7}$$

Next, we use

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

and, therefore, see that

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

Finally, by using $E[L_1] = \int_0^{0.2} L_1\,\mathrm{d}f(L_1)$, we get

$$E[I|A_i] = E[L_1]^2(n-1)\cdot\sum_{j=1}^{d} p_j|\sin(\delta_i - \delta_j)|. \tag{8}$$

Using this result in Equation (4) finally proves the assertions of the theorem. □

Choosing the lengths $L_1,\ldots,L_n$ on the interval $[0, 0.2]$ was a technical twist that allowed us to assume that if a bounded ray T_1 starting in $[0.4, 0.6]^2$ is intersected by another bounded ray T_2, then T_2 started definitely in $[0, 1]^2$. Therefore, it holds that

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

where A denotes the event that T_1 started in $[0.4, 0.6]^2$. Note that the left inequality follows by the law of total expectation. In order to prove Theorem 2.1, it remained to

show that $E[I|A] = \Delta \cdot E[L_1]^2(n-1)$. However, if we distribute $L_1, \ldots, L_n$ on an interval $[0, \epsilon]$, with $\epsilon < 0.25$, the argument from earlier easily extends to

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

where A denotes the event that $v_1 \in [2\epsilon, 1-2\epsilon]^2$. Our proof for Theorem 2.1 basically showed $E[I|A] = \Delta \cdot E[L_1]^2(n-1)$, and it is not affected by the earlier generalization. As a consequence, it turns out that Theorem 2.1 can be actually generalized to

$$(1-4\epsilon)^2\Delta \cdot E[L_1]^2(n-1) \;\leq\; E[I] \;\leq\; \Delta \cdot E[L_1]^2(n-1). \tag{9}$$

2.2. Implications for Motorcycle Graphs

Consider n motorcycles with start points $v_1, \ldots, v_n \in [0,1]^2$ and n direction angles $\varphi_1, \ldots, \varphi_n \in [0, 2\pi)$ that are chosen according to Theorem 2.1. The motorcycles are assumed to drive at the same speed. We compute the motorcycle graph of this setting and record all the trace lengths. We can repeat this experiment a number of times and keep on recording the trace lengths. The samples recorded can be used to obtain an approximation $\hat{f}$ of the density of the trace lengths of a motorcycle graph with n motorcycles. Using that $\hat{f}$ in Theorem 2.1 establishes the relation (Equation (9)) between $E[I]$ and $E[L_1]$. Unfortunately, both $E[I]$ and $E[L_1]$ are unknown.

However, for increasingly larger values of n, the vast majority of motorcycles does not reach the boundary of $[0,1]^2$ but crashes against other traces. Hence, as the number n of motorcycles increases, the trace lengths shrink in the average case,[4] and for sufficiently large n, the vast majority of motorcycles can be expected to have a trace length less than some constant ϵ smaller than 0.25. Since there are at most n crashes, a motorcycle trace may be assumed to intersect two other traces on average: The motorcycle itself crashes into another trace, and a second motorcycle crashes into the considered trace, which suggests $E[I] = 2$. (The assertion $E[I] = 2$ can also be verified experimentally; note that $E[I] \in O(1)$ would be good enough for our subsequent runtime analysis.)

Assuming $E[I] = 2$ and small ϵ in the inequalities (Equation (9)) leads to the following approximation for the mean trace length

$$E[L_1] \;\approx\; \sqrt{\frac{2}{(n-1)\sum_{i,j=1}^{d} p_i p_j |\sin(\delta_i - \delta_j)|}}. \tag{10}$$

Of course, the assumption that $L_1, \ldots, L_n$ are independently distributed is not justified for the actual motorcycle graph problem. However, in Section 4, we are able to substantiate this approximative formula for the mean trace length of motorcycle graphs by providing sound experimental evidence.

3. SIMPLE PRACTICE-MINDED ALGORITHM

In the following, we present a simple algorithm for computing motorcycle graphs that exhibits an $O(n \log n)$ runtime for real-world input. Basically, we take the algorithm by Cheng and Vigneron, drop the arrangements on the cells of the $1/\sqrt{n}$-cutting and replace the cutting by a simple rectangular grid. In other words, we apply ordinary geometric hashing to the priority-queue–based algorithm. Our motivation is a simple trade-off: We lose the deterministic subquadratic runtime behavior and gain a simple implementation that can be expected to run in $O(n \log n)$ time for sufficiently uniformly distributed input.

[4] Of course, the motorcycles must not all drive in parallel directions.

3.1. Algorithm

Actually, the algorithm implemented computes a generalized motorcycle graph. First, it can handle solid walls against which motorcycles may crash. By a crash against a wall, we mean that the motorcycle stops driving when it reaches the wall but its trace remains. Second, our motorcycles need not all start at the same time. Also, new motorcycles may emerge in the course of computation as long as the start times lie in the future. This means that the set of motorcycles need not be known a priori. Both generalizations have applications in order to compute straight skeletons. Cheng and Vigneron [2007] presented a straight-skeleton algorithm for simple polygons with holes that uses motorcycle graph. Here, motorcycles are halting when they reach the boundary of the polygon, which is interpreted as "running out of fuel." In our terms, we would have considered the edges of the polygons as walls. The straight-skeleton algorithm for planar straight-line graphs by Huber and Held [2011] also employs the motorcycle graph, where the edges of the input graph are considered as walls. Furthermore, in this application so-called vertex events of the straight skeleton give rise to motorcycles that are not known a priori.

The input to our algorithm is a set $M = \{m_1, \ldots, m_n\}$ of motorcycles and a set $W = \{w_1, \ldots, w_u\}$ of rigid walls. A wall is modeled as a straight-line segment and a motorcycle m_i is a triple (v_i, s_i, t_i^*), where $v_i \in [0, 1]^2$ is the start point, $s_i \in \mathbb{R}^2$ is the velocities, and $t_i^* \in [0, \infty)$ is the start time. Let us further note that we restrict our computation of the motorcycle graph to the unit square $[0, 1]^2$. (This restriction can be waived; see Section 4.3.) For this reason—and due to practical numerical advantages—we scale the input such that the bounding box of the input is a proper subset of the unit square $[0, 1]^2$. (The restriction of the computation to $[0, 1]^2$ can be simply achieved within our framework by adding four dummy walls that form the boundary of $[0, 1]^2$.)

We maintain two geometric hashes, H_M and H_W, which form $\sqrt{n} \times \sqrt{n}$ grids covering $[0, 1]^2$. While H_M keeps track of the motorcycles, H_W contains the walls of W. The basic algorithm is a discrete event simulation of the movement of the motorcycles with two types of events: crash events and switch events. A crash event indicates that a motorcycle crashes against another motorcycle or a wall, and a switch event occurs when a motorcycle leaves one grid cell and enters a neighboring one. All pending events are kept in a priority queue Q. Finally, for every motorcycle m_i, we maintain a balanced binary search tree $C[m_i]$ that contains potential future crash events of the motorcycle m_i.

In the initialization phase of the algorithm, we fill H_W with all walls of W and invoke `insertMc(`m`)` for every $m \in M$ (see the following discussion). Then, until Q is empty, we extract the next event e from Q and process it by calling `handle(`e`)`, depending on the type of the event e. If a newly emerging motorcycle m should be inserted at any time of computation, then `insertMc(`m`)` is called. The procedures `insertMc` and `handle` are described in the sequel.

—`insertMc(motorcycle` m`)`. We first create an empty binary search tree $C[m]$ and then insert a switch event e for m into Q, with the occurrence time of e set to the start time of m.

—`handle(switch event` e `of motorcycle` m`)`. At first, we add m to the newly entered cell c of H_M and add the next switch event of m to Q if one exists. Then, we check for a potential crash against a wall in c and add the earliest one as crash event to Q. Now we clear $C[m]$, and for every other motorcycle m' registered in c, we check for potential future crash events. For every such potential crash, we add a corresponding crash event into $C[m]$ if m' reaches the point of crash before m, and analogously into $C[m']$ for the dual case. Note that if we add an event into $C[m']$, which ends up being the

earliest in $C[m']$, then we have to update Q accordingly. Finally, we add the earliest crash event of $C[m]$ into Q.

—`handle(crash event e of motorcycle m)`. First, we mark the motorcycle m as crashed and clear $C[m]$. Note that the trace of m ends at the corresponding crash point. Second, we remove the possibly remaining switch event of m from Q. Then we clean up interactions with other motorcycles m' in the current grid cell. We remove from Q all crash events where m is involved and which got invalid, because it turned out that m will not reach the affected location. Further, if $C[m']$ contains such an invalid crash event, then it is removed.

3.2. Runtime Analysis

Let us ignore the influence of the wall handling and concentrate only on the computation of the motorcycle graph. The procedure `insertMc` is called exactly n times, each call taking $O(\log n)$ time. The major part of the algorithm consists of event handling. A single (crash, respectively, switch) event is handled in $O(n \log n)$ time. Note that we can remove an event from Q in $O(\log n)$ time if we have a pointer to the affected element.

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

Indeed, Section 2.2 lets us conclude that, for n motorcycles with their start points distributed uniformly, a motorcycle trace has a length of $\Theta(1/\sqrt{n})$, on average—at least, if all motorcycles would drive at the same speed. This implies that the average motorcycle crosses $\Theta(1)$ grid cells. Consequently, we expect that a single grid cell is occupied by $\Theta(1)$ motorcycles. Again, the initialization consumes $O(n \log n)$ time in total. However, now a single (crash, respectively, switch) event is handled in $O(\log n)$ expected time. There are in total still n crash events, but, on average, only $\Theta(1)$ switch events per motorcycle. In summary, we may expect a total complexity of $O(n \log n)$ for sufficiently uniformly distributed input, as motivated by Section 2.2. We provide sound experimental evidence in Section 4 that this bound on the average runtime holds for uniformly distributed start points as well as for almost every real-world input, where start points are not necessarily distributed uniformly and speeds may vary significantly.

4. EXPERIMENTAL RESULTS

Our motorcycle code is called Moca.[5] It is entirely implemented in C++ and makes heavy use of the STL for common data structures like lists, queues and balanced binary trees. All geometric computations are based on ordinary IEEE 745 double precision floating-point arithmetic, following the same general approach used successfully in Held's FIST and VRONI codes [Held 2001a, 2001b]. Moca provides runtime options for a posteriori checks regarding the topology of the computed motorcycle graph. In particular, we check (i) for motorcycle traces with a free end[6] and (ii) for motorcycle traces intersecting in the relative interiors of each other. It turned out that Moca performs reliably. To the best of our knowledge, this is the first competitive motorcycle graph implementation. For this reason, we do not compare our code with other implementations but content ourselves with a discussion of the performance of Moca.

[5]Shorthand for MOTORCYCLE CRASHER.

[6]For example, the crash point must coincide with the trace of another motorcycle.

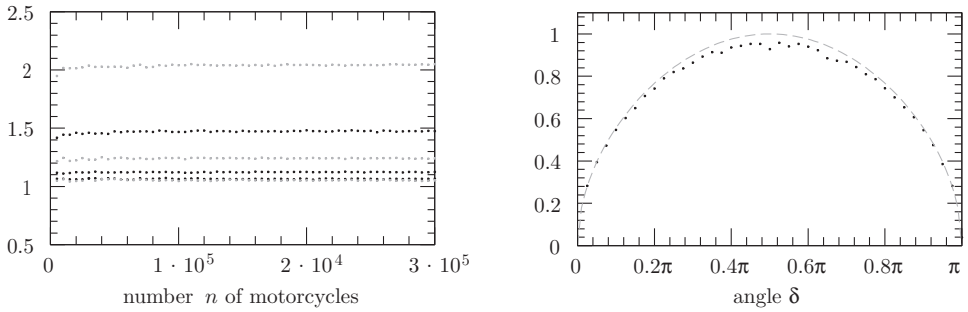

Fig. 4. Two experiments illustrating the mean trace length of motorcycles. A dataset contains motorcycles with uniformly distributed start points and uniformly distributed directions on the set $\{0, \delta\}$. Left, Experiment 1: Every dot depicts the mean trace length of a dataset divided by $2/\sqrt{n-1}$, where n is the number of motorcycles (x-axis). Right, Experiment 2: Every dot depicts the reciprocal of the mean trace length divided by $2/\sqrt{n-1}$. The x-axis illustrates δ and the dashed curve corresponds to $\sqrt{\sin|\delta|}$.

4.1. Experimental Verification of the Stochastic Analysis

We start with substantiating the theoretical results obtained by means of the stochastic analysis of Section 2. We set up two experiments that investigate the dependence of the mean trace length on (i) the number of motorcycles n and (ii) the direction angles δ_i in Equation (10).

We created datasets with n random[7] motorcycles by choosing the start points uniformly in $[0, 1]^2$ and the direction angle uniformly from the set $\{0, \delta\}$. Equation (10) asserts that the mean trace length L for such a dataset is given by

$$E[L] \approx \frac{2}{\sqrt{(n-1)|\sin\delta|}}. \tag{11}$$

For Experiment 1, we created a dataset for every $n \in \{i \cdot 5{,}000 \ : \ 1 \le i \le 60\}$ and $\delta \in \{i\pi/12 \ : \ 1 \le i \le 6\}$. In the left subfigure of Figure 4, we illustrate the normalized mean trace lengths, which, as predicted, are aligned on six[8] horizontal lines, where each line corresponds to a particular value of δ.

For Experiment 2, we created datasets for $n = 10{,}000$ and $\delta \in \{i\pi/40 \ : \ 1 \le i \le 40\}$. In the right subfigure of Figure 4, the reciprocal values of the normalized mean trace lengths are plotted. Again, as predicted by Equation (11), the normalized mean trace lengths are aligned on the reference curve $\sqrt{\sin|\delta|}$.

Finally, we used Equation (10) to experimentally verify the mean trace length when considering direction angles uniformly distributed on $[0, 2\pi)$. This can be achieved by uniformly distributing the direction angles on $\{\delta_1, \ldots, \delta_d\}$, with $\delta_i = i\frac{2\pi}{d}$, and subsequently considering $d \to \infty$. This basically results in

$$\begin{aligned} E[L] &\approx \lim_{d\to\infty} d\sqrt{\frac{1}{(n-1)\sum_{i=1}^{d-1}(d-i)|\sin i\frac{2\pi}{d}|}} \\ &= \lim_{d\to\infty} d\sqrt{\frac{1}{(n-1)\cdot d^2\sum_{i=1}^{d-1}(1-\frac{i}{d})|\sin 2\pi\frac{i}{d}|\cdot\frac{1}{d}}} \end{aligned} \tag{12}$$

[7]Random datasets were generated in Python using the module "random."
[8]The bottom two lines, mostly overlap.

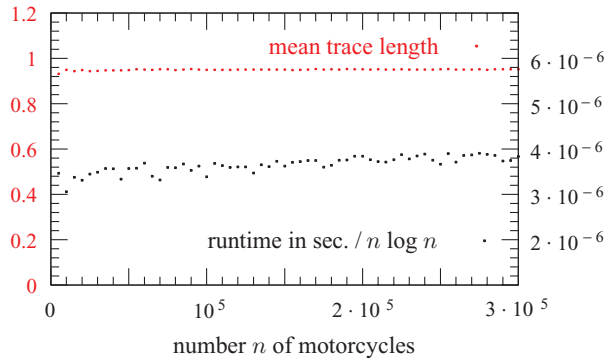

Fig. 5. The runtime of Moca and the mean trace length of random datasets with different numbers n of motorcycles. The start points are uniformly distributed on $[0, 1]^2$ and the directions are uniformly distributed on $[0, 2\pi)$. For illustrative, reasons, we divided the runtime by $n \log n$ and the mean trace length by $\sqrt{\frac{\pi}{n-1}}$.

$$= \sqrt{\frac{1}{(n-1)\int_0^1 (1-x)|\sin 2\pi x|\, dx}}$$
$$= \sqrt{\frac{\pi}{n-1}}. \tag{13}$$

As in Experiment 1, we set up a corresponding experiment. After division by the appropriate factors, we again obtained a plot showing horizontal lines (Figure 5), as predicted by our stochastic analysis.

4.2. Statistics on Contrived and Real-World Data

Our code has been tested and developed on x86 and amd64 Linux distributions, using gcc-4 compilers. The performance benchmarks presented in the sequel were obtained on a 32-bit Debian Linux machine with an Intel E67000 Core 2 Duo processor with 2.66GHz, using 4GB of RAM. Note that Moca does not gain from utilizing multicore machines, and the 32-bit architecture actually limits the memory footprint of a process to roughly 3GB. For time measurement, we used the C library function `getrusage` and summing up user and system time.

For our tests, we obtained motorcycles from straight-line polygonal chains[9] according to the straight skeleton manner: For every triple of noncollinear consecutive vertices v', v'', and v''' in a chain, we generate a motorcycle $m = (v'', s, 0)$. The speed vector s is given by the direction

$$\frac{v'' - v'}{\|v'' - v'\|} + \frac{v'' - v'''}{\|v'' - v'''\|}$$

and scaled such that $\|s\| = 1/\sin(\alpha/2)$, where α is the angle $\angle(v', v'', v''')$ defined by the vertices v', v'', and v'''. This corresponds to the setup in Huber and Held [2011] for computing straight skeletons, and to a generalization of the set-up for motorcycle graphs used by Cheng and Vigneron [2007].

We ran Moca on more than 22,000 datasets, consisting of synthetic and real-world data. Our real-world datasets—obtained from companies, colleagues, and the Web—include polygonal cross-sections of human organs, GIS maps of roads and river

[9]The polygonal chains may be open or closed and need not be simple. One input file can consist of several chains.

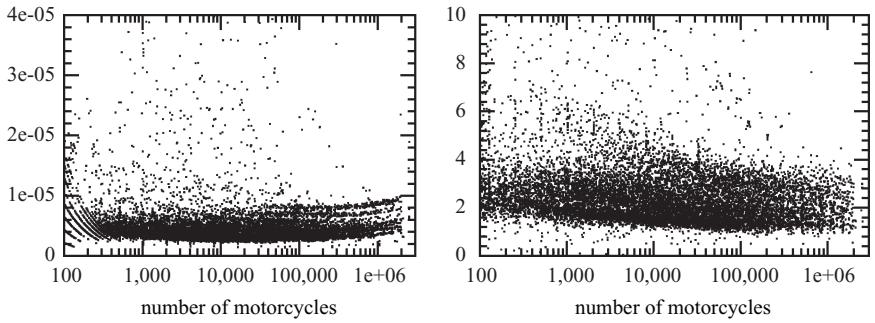

Fig. 6. Left: Runtime (in section, y-axis) divided by $n \log n$, for n motorcycles (x-axis). Right: This plot illustrates for every dataset along the y-axis the mean trace lengths multiplied by $\sqrt{n}$.

networks, polygonal outlines of fonts, and boundaries of workpieces for CNC machining or stereo-lithography, and the like. The synthetic test data was generated by means of RPG [Auer and Held 1996], and a version recently enhanced by the second author, SRPG. The synthetic data also contains contrived data, like extremely fine approximations of smooth curves, where the vertices are distributed highly irregularly.

The left plot of Figure 6 illustrates the actual runtime for every single dataset, with the times divided by $n \log n$ given in seconds on the y-axis. (The number n of motorcycles is shown in a logarithmic scale on the x-axis.) To avoid unreliable timings and other idiosyncrasies of small datasets, we only plot results for test runs with at least 100 motorcycles. As shown, for the vast majority of datasets, Moca exhibits an $O(n \log n)$ runtime behavior: Except for a few outliers, the datasets are processed in at most $10^{-5} n \log n$ seconds. (For the sake of visual clarity, for about 100 datasets, the runtimes are not plotted; processing these datasets takes up to $2 \cdot 10^{-3} n \log n$ seconds.)

The right plot of Figure 6 shows for every dataset the mean trace multiplied by $\sqrt{n}$ for better illustration. As shown, for most datasets in the entire database, the mean trace length is between $0.5/\sqrt{n}$ and $6/\sqrt{n}$, which is the main reason for the good runtime behavior achieved by our implementation. (A few outliers with mean trace lengths of up to $250/\sqrt{n}$ were not plotted.) This result conforms nicely to the stochastic analysis of Section 2. Our test results provide experimental evidence that the theoretical analysis carried out on the basis of the uniform speed assumption can be relaxed in practice and that the results predicted for uniformly distributed data also hold for data that comes from real-world input.

In our experiments, the polygonal chains were not inserted as walls. However, additional tests demonstrated that inserting or disregarding the walls has hardly any impact on the runtime (Figure 7).

In order to deepen the investigation of the runtime of Moca for random datasets with nonuniformly distributed start points, we also generated datasets where the start points or single coordinates are distributed Gaussian and multimodal Gaussian.[10] We used different standard deviations in order to consider the runtime behavior of Moca on datasets with an increasing accumulation of motorcycles in a small region. In Figure 8, we have exemplarily illustrated two experiments. For the left subfigure, the start points are distributed Gaussian with mean $(0.5, 0.5)$ and standard deviation plotted at the x-axis. For the right subfigure, the start points' y-coordinates are chosen uniformly from $[0, 1]$ and the x-coordinates are distributed Gaussian with mean 0.5 and standard deviation plotted at the x-axis. As expected, the runtime increases significantly for

[10]Since we consider motorcycles starting in $[0, 1]^2$, we drop random results outside of $[0, 1]^2$ and repeatedly generate a random start point.

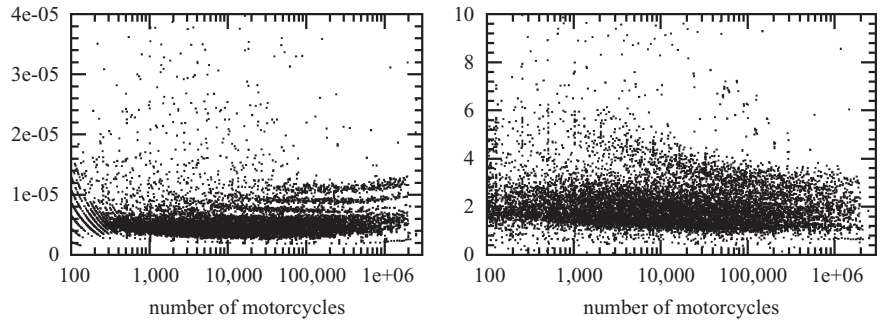

Fig. 7. The runtime (left) and mean trace lengths (right) of Moca on our datasets, as shown in Figure 6, but with input chains inserted as walls.

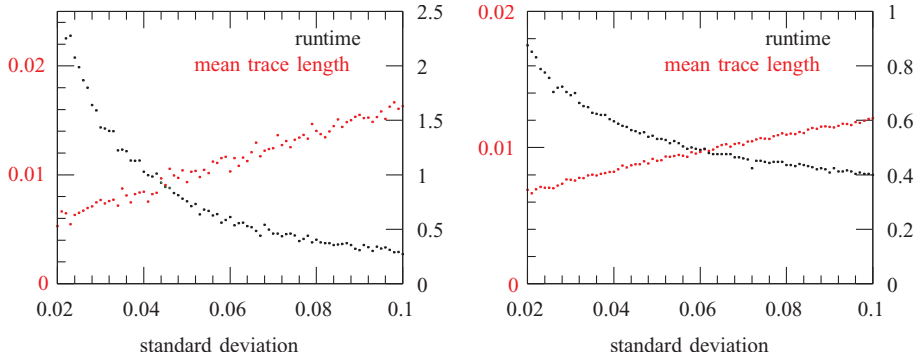

Fig. 8. The runtime of Moca and the mean trace length of random datasets. The direction was chosen uniformly from $[0, 2\pi)$. Left: 2,000 motorcycles with a start point distributed Gaussian with mean $(0.5, 0.5)$ and standard deviation given by the x-axis. Right: 10,000 motorcycles with a start point (p, q), where p is distributed Gaussian with mean 0.5 and standard deviation given by the x-axis and q chosen uniformly from $[0, 1]$.

very small standard deviations, say, less than 0.04. Even though the mean trace length decreases as well for small standard deviations, the fact of condensed start points dominates the runtime behavior.

4.3. Extensions of the Basic Algorithm

4.3.1. Beyond the Unit Square. By simply extending the $2\sqrt{n}+2$ grid segments to infinity, we can cover the entire plane by (possibly unbounded) grid cells and then we simply continue the simulation process outside of the unit square. While each motorcycle crashes if we add dummy segments bounding the unit square, this might not be the case if we remove them. At some instance of time, the last event is removed from the priority queue and all remaining motorcycles that did not crash so far will not crash at all—they "escaped."

We plotted the corresponding runtimes and mean trace lengths on our datasets in Figure 9, with the simulation continued outside the unit square. Compared to Figure 6, there is hardly any impact to Moca's runtime in the average case, since only a relatively small number of motorcycles actually drives outside $[0, 1]^2$. Our experiments confirm this observation. Figure 10 shows the number of motorcycles (divided by $\sqrt{n}$) that never crashed at all and the number of motorcycles (divided by $\sqrt[4]{n}$) that crashed outside of $[0, 1]^2$. The latter observation is related to Erickson's [1998] question: "Is there an efficient algorithm that determines whether a motorcycle escapes?"

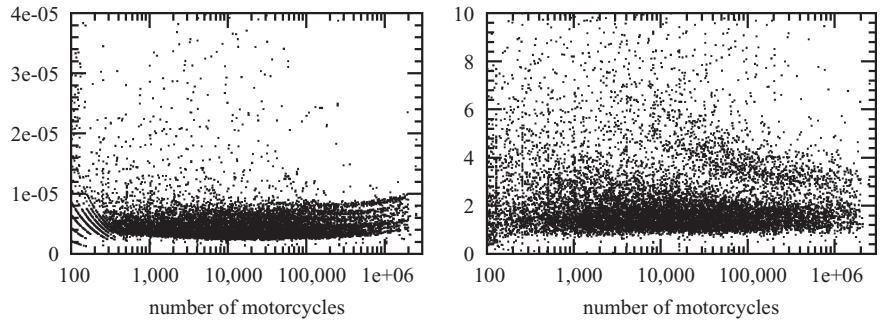

Fig. 9. The runtime (left) and mean trace lengths (right) of Moca on our datasets, as shown in Figure 6, but with computation continued outside the unit square.

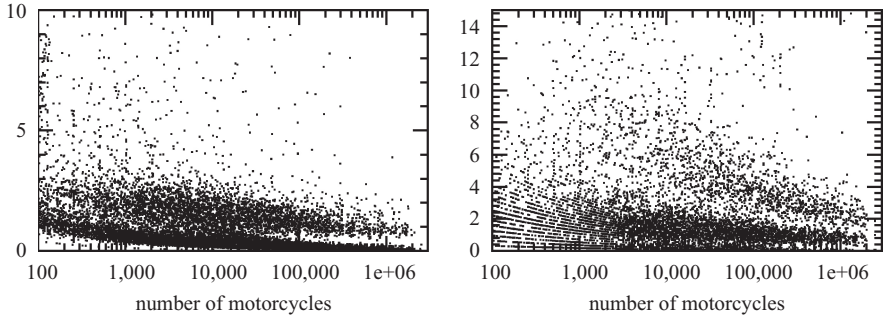

Fig. 10. Left: The number of motorcycles that never crashed divided by $\sqrt{n}$, for n motorcylces (x-axis). Right: The number of motorcycles crashed outside the unit square divided by $\sqrt[4]{n}$.

If we do not allow new motorcycles to emerge, then we can compute the motorcycle graph outside of $[0, 1]^2$ efficiently (also in the worst case) by stopping all motorcycles that reach the boundary of $[0, 1]^2$ until all remaining motorcycles in the interior of $[0, 1]^2$ crashed. Then, we apply a sweep-line algorithm where the sweep line corresponds to the boundary of $[-t, 1 + t]^2$ with increasing $t \in [1, \infty)$. Erickson presented a corresponding $O(n \log n)$ algorithm for motorcycles driving from left to right but not necessarily parallel to the x-axis [Erickson 1998]. In our case, too, the sweep line stops when two neighboring motorcycles on the sweep line exchange their relative positions, thus indicating a crash.

4.3.2. Possible Further Extensions. Since the basic algorithm is very simple, there are many additional ways to extend this algorithm into different directions. First, neither the walls nor the motorcycle traces are conceptionally bounded to form straight lines. The traces even need not be connected (i.e., the motorcycles might be allowed to jump). Second, it would be very easy to equip the motorcycles with a finite amount of fuel or to allow speeds that vary over time. Third, the walls could be conditionally or temporarily permeable. Furthermore, the computation is not restricted to the plane but could also be done on the sphere.

5. CONCLUSION

In this article, we provide both formal and extensive experimental evidence that a simple enhancement of the brute-force algorithm for computing motorcycle graphs will perform nicely in practice: The use of geometric hashing suffices to obtain an $O(n \log n)$ expected-time behavior, even if the motorcycles are placed much more irregularly then

allowed by the theoretical analysis. Tests run with our C++ implementation Moca on contrived and real-world data clearly showed that we may expect most inputs of n motorcycles to be processed in at most $10^{-5} n \log n$ seconds on a standard PC. In particular, we can compute a motorcycle graph of a million motorcycles in a few hundred seconds. This makes Moca the first motorcycle code that is able to cope with large real-world datasets.

This surprisingly good practical performance of an easy-to-implement algorithm for computing motorcycle graphs hinges on the fact that, on average, motorcycles do not cover long distances. Rather, the mean trace length of n (uniformly distributed) motorcycles within the unit square can be modeled by $O(1/\sqrt{n})$. Interestingly, this bound also holds for a large variety of datasets with highly irregular start points and speeds of the motorcycles. That is, our tests provide experimental evidence that our theoretical results—which motivated our implementation—do also hold under less stringent assumptions and, in particular, carry over to real-world data.

Moca actually computes a generalized version of the motorcycle graph. It can handle rigid walls where motorcycles may crash against and new motorcycles may emerge in the course of the computation. It could also be extended to handle curved walls and traces, nonconstant speeds of motorcycles, or a curved playground on which the motorcycles drive. The design and implementation of Moca can be regarded as a first step toward the design of a practical algorithm for computing straight skeletons. In fact, our recent straight-skeleton implementation Bone [Huber and Held 2011] uses Moca and computes the straight skeleton of a planar straight-line graph with n segments in $O(n \log n)$ expected time for real-world input.

REFERENCES

Aichholzer, O., Aurenhammer, F., Alberts, D., and Gärtner, B. 1995. A novel type of skeleton for polygons. *J. Universal Comput. Sci. 1,* 12, 752–761.

Auer, T. and Held, M. 1996. Heuristics for the generation of random polygons. In *Proceedings of the Canadian Conference on Computational Geometry (CCCG'96)*. Carleton University Press, 38–44.

Chazelle, B. 1993. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom. 9,* 2, 145–158.

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

Czyzowicz, J., Rival, I., and Urrutia, J. 1989. Galleries, light matchings and visibility graphs. In *Proceedings of the 1st Workshop Algorithms Data Struct.* 316–324.

Eppstein, D. and Erickson, J. 1999. Raising roofs, crashing cycles, and playing pool: applications of a data structure for finding pairwise interactions. *Discrete Comput. Geom. 22,* 4, 569–592.

Eppstein, D., Goodrich, M., Kim, E., and Tamstorf, R. 2008. Motorcycle graphs: canonical quad mesh partitioning. *Comput. Graph. Forum 27,* 5, 1477–1486.

Eppstein, D., Goodrich, M., Kim, E., and Tamstorf, R. 2009. Approximate topological matching of quad meshes. *Visual Comput. 25,* 8, 771–783.

Erickson, J. 1998. Crashing motorcycles efficiently. http://compgeom.cs.uiuc.edu/~jeffe/open/cycles.html.

Held, M. 2001a. FIST: fast industrial-strength triangulation of polygons. *Algorithmica 30,* 4, 563–596.

Held, M. 2001b. VRONI: An engineering approach to the reliable and efficient computation of voronoi diagrams of points and line segments. *Comput. Geom. Theor. Appl. 18,* 2, 95–123.

Huber, S. and Held, M. 2011. Theoretical and practical results on straight skeletons of planar straight-line graphs. In *Proceedings of the 27th Annual ACM Symposium on Computational Geometry*. ACM, New York, 171–178.

Ishaque, M., Speckmann, B., and Tóth, C. 2009. Shooting permanent rays among disjoint polygons in the plane. In *Proceedings of the 25th Annual ACM Symposium on Computational Geometry*. ACM, New York.

Received April 2010; revised April 2011; accepted April 2011