A Practice-Minded Approach to Computing Motorcycle Graphs

Stefan Huber    Martin Held

Universität Salzburg
FB Computerwissenschaften
Salzburg, Austria

16–18 March, EuroCG09, Brussels
What is a motorcycle graph?

- We define a **motorcycle** $m$ as a triple $(p, s, t^*) \in \mathbb{R}^2 \times \mathbb{R}^2 \times [0, \infty)$, where $p$ is the start point, $t^*$ is the start time and $s$ is the speed vector.
- Consider $n$ motorcycles $m_1, \ldots, m_n$, with $m_i = (p_i, s_i, t_i^*)$. Each motorcycle leaves a trace behind and crashes when reaching the trace of another motorcycle.
What is a motorcycle graph?

- We define a **motorcycle** \( m \) as a triple \((p, s, t^*) \in \mathbb{R}^2 \times \mathbb{R}^2 \times [0, \infty)\), where \( p \) is the start point, \( t^* \) is the start time and \( s \) is the speed vector.

- Consider \( n \) motorcycles \( m_1, \ldots, m_n \), with \( m_i = (p_i, s_i, t_i^*) \). Each motorcycle leaves a trace behind and crashes when reaching the trace of another motorcycle.
What is a motorcycle graph?

- We define a **motorcycle** $m$ as a triple $(p, s, t^*) \in \mathbb{R}^2 \times \mathbb{R}^2 \times [0, \infty)$, where $p$ is the start point, $t^*$ is the start time and $s$ is the speed vector.

- Consider $n$ motorcycles $m_1, \ldots, m_n$, with $m_i = (p_i, s_i, t_i^*)$. Each motorcycle leaves a trace behind and crashes when reaching the trace of another motorcycle.
What is a motorcycle graph?

- We define a **motorcycle** $m$ as a triple $(p, s, t^*) \in \mathbb{R}^2 \times \mathbb{R}^2 \times [0, \infty)$, where $p$ is the start point, $t^*$ is the start time and $s$ is the speed vector.

- Consider $n$ motorcycles $m_1, \ldots, m_n$, with $m_i = (p_i, s_i, t_i^*)$. Each motorcycle leaves a trace behind and crashes when reaching the trace of another motorcycle.
Prior work

- **Trivial brute-force algorithm**
  - Find $O(n)$ crashes in chronological order. Testing each against each takes $O(n^2)$ time for each crash.
  - Using a priority queue results in an $O(n^2 \log n)$ algorithm, instead of $O(n^3)$.

Eppstein and Erickson, 1999

Very complicated $O(n^{17/11 + \epsilon})$ algorithm.

Transformed problem to intersecting 3D faces and considered closest-pair problems.

Cheng and Vigneron, 2002

Induced a partitioning of the plane by $1/\sqrt{n}$-cuttings and exploited arrangements on each cutting-cell, resulting in an $O(n^{\sqrt{n} \log n})$ algorithm.

Too complicated for an actual implementation.

Stefan Huber, Martin Held

An Approach to Computing Motorcycle Graphs
Prior work

- **Trivial brute-force algorithm**
  - Find $O(n)$ crashes in chronological order. Testing each against each takes $O(n^2)$ time for each crash.
  - Using a priority queue results in an $O(n^2 \log n)$ algorithm, instead of $O(n^3)$.

- **Eppstein and Erickson, 1999**
  - Very complicated $O(n^{17/11+\epsilon})$ algorithm.
  - Transformed problem to intersecting 3D faces and considered closest-pair problems.
Prior work

• **Trivial brute-force algorithm**
  - Find $O(n)$ crashes in chronological order. Testing each against each takes $O(n^2)$ time for each crash.
  - Using a priority queue results in an $O(n^2 \log n)$ algorithm, instead of $O(n^3)$.

• **Eppstein and Erickson, 1999**
  - Very complicated $O(n^{17/11+\epsilon})$ algorithm.
  - Transformed problem to intersecting 3D faces and considered closest-pair problems.

• **Cheng and Vigneron, 2002**
  - Induced a partitioning of the plane by $1/\sqrt{n}$-cuttings and exploited arrangements on each cutting-cell, resulting in an $O(n\sqrt{n} \log n)$ algorithm.
  - Too complicated for an actual implementation.
Current situation

Summary

- No “close-to linear” algorithm is known.
- No sub-quadratic implementation is known.
Basic idea

Replace the $\frac{1}{\sqrt{n}}$-cutting of Cheng and Vigneron’s algorithm by a regular rectangular grid and drop the arrangements.

With other words: We apply geometric hashing to the straightforward algorithm.
Basic idea

Replace the $1/\sqrt{n}$-cutting of Cheng and Vigneron’s algorithm by a regular rectangular grid and drop the arrangements.

With other words: We apply geometric hashing to the straightforward algorithm.

Main question

Consider $n$ motorcycles on a $h \times h$ hash. In the worst case, this leads to $O(n \cdot h)$ crossings of motorcycles with the grid-lines. Do we have a chance to obtain a good performance in practice?
Basic algorithm

Discrete event simulation of the moving motorcycles:

1. Crash event: a motorcycle crashes into a trace.
2. Switch event: a motorcycle leaves one grid cell and enters a neighboring grid cell.

In the course of simulation, the algorithm iteratively extracts the next event from a priority queue and processes it.
Input data and data structures

Our input consists of:

- A set $M := \{m_1, \ldots, m_n\}$ of motorcycles.
  No need to know $M$ a-priori: new motorcycles may emerge, if their start time is in the future.

- A set $W$ of line segments representing walls, where motorcycles may crash against.

We maintain the following data structures:

- A priority queue $Q$ of pending events.
- For every motorcycle $m$, a binary search tree $C[m]$, where $C[m]$ holds potential future crash events of $m$.
- A geometric hash $H$ for tracking the motorcycle traces, using a $h \times h$ grid.
- A geometric hash $G$ for the wall-segments, using the same grid geometry as $H$. 

Stefan Huber, Martin Held
An Approach to Computing Motorcycle Graphs
Our input consists of:

- A set \( M := \{m_1, \ldots, m_n\} \) of motorcycles.
  No need to know \( M \) a-priori: new motorcycles may emerge, if their start time is in the future.
- A set \( W \) of line segments representing walls, where motorcycles may crash against.

We maintain the following data structures:

- A priority queue \( Q \) of pending events.
- For every motorcycle \( m \) a binary search trees \( C[m] \), where \( C[m] \) holds potential future crash events of \( m \).
- A geometric hash \( H \) for tracking the motorcycle traces, using a \( h \times h \) grid.
- A geometric hash \( G \) for the wall-segments, using the same grid geometry as \( H \).
Basic algorithm

1: procedure MCGRAF(motorcycles $M$, walls $W$)
2: $Q, C, H \leftarrow$ initialize empty
3: $G \leftarrow$ geometric hash with all $w \in W$
4: for all $m \in M$ do
5: \hspace{1em} INSERTMC($m$) ▷ Adds an empty binary tree $C[m]$ to $C$
6: \hspace{1em} INSERTSW($m$) ▷ Inserts an initial switch-event of $m$ to $Q$
7: end for
8: while not $Q$.empty() do ▷ Process all events $e$
9: \hspace{1em} $e \leftarrow Q$.pop()
10: \hspace{1em} HANDLE($e$)
11: \hspace{1em} Switch-event: attach motorcycle to new grid cell, add next switch-event to $Q$, maintain $C$.
12: \hspace{1em} Crash-event: clean-up stale future crash-events in $C$.
13: end while
14: end procedure
Let $k$ be the maximum number of motorcycles in a hash cell. Processing a switch- resp. crash-event can be done in $O(k \log n)$ time.

There are $O(n)$ crash-events and $O(n \cdot h)$ switch-events and we choose $h \in \Theta(\sqrt{n})$. Hence, the worst case complexity is

$$O(nkh \log n) \subseteq O(n^2 \sqrt{n} \log n).$$
Worst case

Ω(n) motorcycles cross Ω(h) hash-cells in a narrow strip that is O(1) cells thick. Further, no other motorcycle is allowed to cross this strip before.

...what is the expected runtime?
We denote by $S := [0, 1]^2$ the unit square covered by a $h \times h$ grid.

**Lemma**

Let $R = (p, \varphi) \in S \times [0, 2\pi)$ be a uniformly distributed ray, starting at $p$, with direction angle $\varphi$. Further, let $C$ be a cell of a $h \times h$ grid on $S$. The probability that $R$ intersects $C$ is $\Theta(1/h)$. 
We denote by \( S := [0, 1]^2 \) the unit square covered by a \( h \times h \) grid.

**Lemma**

Let \( R = (p, \varphi) \in S \times [0, 2\pi) \) be a uniformly distributed ray, starting at \( p \), with direction angle \( \varphi \). Further, let \( C \) be a cell of a \( h \times h \) grid on \( S \). The probability that \( R \) intersects \( C \) is \( \Theta(1/h) \).

**Theorem**

Consider \( n \) random rays distributed within \( S \). The expected number of rays intersecting a specific cell is in \( \Theta(n/h) \).
Expected runtime

Observation

Consider \( n \) random motorcycles within \( S \) and let \( h \in \Theta(\sqrt{n}) \). A motorcycle trace has a mean length proportional to \( 1/\sqrt{n} \). Hence, it intersects \( O(1) \) cells in average. This leads to an \( O(n \log n) \) expected runtime.
Experimental setup

- A data set consist of polygonal chains.
- For every inner vertex of a chain, we define a motorcycle in “straight skeleton” manner. The chains are considered being walls.
- We ran our implementation MOCA on 22,000 thousand data sets, consisting of real-world\(^1\) data and contrived data.

\(^1\)Medical scans, GIS maps, outlines of fonts, CAD/CAM models, etc.
Experimental results

Actual runtime in seconds divided by the number $n$ of motorcycles for each of the 22,000 data sets.

Least-square fit reveals an average run time of $5.05 \cdot 10^{-6} n \log n$ seconds on our computer.
Conclusion

- Easy-to-implement algorithm.
- Surprisingly good performance and competitive in practice.
- Algorithm can be extended easily to more general motorcycle graph problems: motorcycles running out of fuel, curved traces, partial/temporal/conditional invisible traces, etc.