# Optimal Polyline Simplification under the Local Fréchet Distance in 2D in (Near-)Quadratic Time

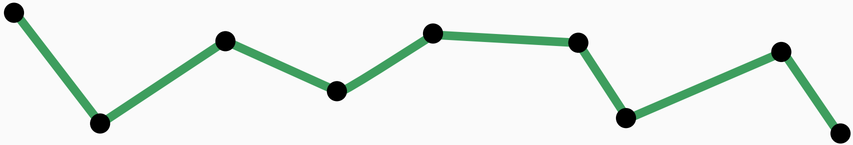
Peter Schäfer, Sabine Storandt, Johannes Zink

Universität Konstanz

Julius-Maximilians-Universität Würzburg

CCCG 2023 Aug 3, Montreal

# Polyline Simplification

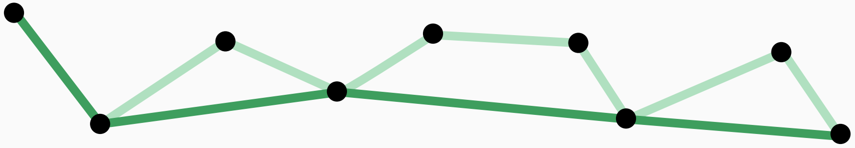


**Problem Setting**

- a sequence of $n$ vertices $p_1, \ldots, p_n$
- **straight** segments
- select a **minimal subset** of vertices (no interpolation!)
- such that a distance **measure** is within a given **threshold** $\delta$:

$$d(P, S) \leq \delta$$

# Polyline Simplification

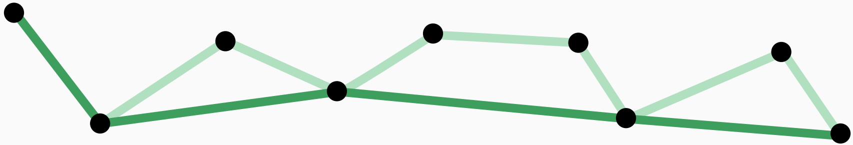


## Problem Setting

- a sequence of $n$ vertices $p_1, \ldots, p_n$
- **straight** segments
- select a **minimal subset** of vertices (no interpolation!)
- such that a distance **measure** is within a given **threshold** $\delta$:

$$d(P,S) \leq \delta$$

# Polyline Simplification

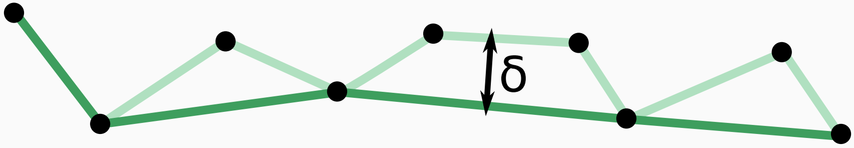


## Problem Setting

- a sequence of $n$ vertices $p_1, \ldots, p_n$
- **straight** segments
- select a **minimal subset** of vertices (no interpolation!)
- such that a distance **measure** is within a given **threshold** $\delta$:

$$d(P, S) \leq \delta$$

# Polyline Simplification

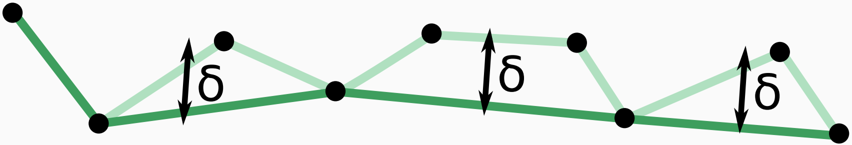



## Problem Setting

- a sequence of $n$ vertices $p_1, \ldots, p_n$
- **straight** segments
- select a **minimal subset** of vertices (no interpolation!)
- such that a distance **measure** is within a given **threshold** $\delta$:

$$d(P, S) \leq \delta$$

# Local Simplification




## Problem Setting

- **Local** Simplification = segment-wise
- distance measure applied to each **segment**
- select **minimal subset** of vertices such that

$$d(P_{[i,i+1]}, S_{[j,k]}) \leq \delta$$

# Distance Measures

## Hausdorff Distance

- minimize maximum distance between two vertices
- works for **any** set of points
- ignores their **order** ☹

## Fréchet Distance

- minimize maximum distance over all **mapping functions**
- recognizes the **course** of the trajectory ☺
- algorithmically challenging ☹

# Distance Measures

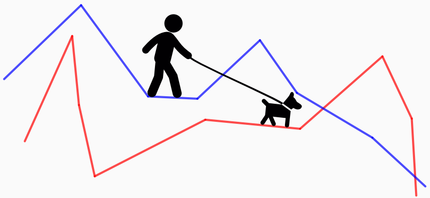
## Hausdorff Distance

- minimize maximum distance between two vertices
- works for **any** set of points
- ignores their **order** ☹

## Fréchet Distance

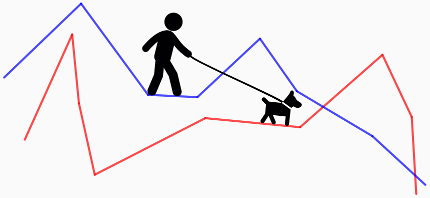
- minimize maximum distance over all **mapping functions**
- recognizes the **course** of the trajectory ☺
- algorithmically challenging ☹

# Fréchet Distance



- it's the **dog-leash** distance
- man and dog walk the curves
- at variable speed (but never backwards)
- find the **minimum** required length of the leash

# Fréchet Distance



**Definition**

Given two parameterized curves $P, Q : [0,1] \longrightarrow \mathbb{R}^2$

$$d_F(P,Q) = \inf_{\sigma,\tau} \max_{\substack{s\in[0,1],\\ t\in[0,1]}} || P(\sigma(s)) - Q(\tau(t)) ||$$

the Fréchet distance is the infimum over all **continuous and increasing** bijections $\sigma, \tau : [0,1] \longrightarrow [0,1]$.

$|| \cdot ||$ is the underlying norm (Euclidean, or other)

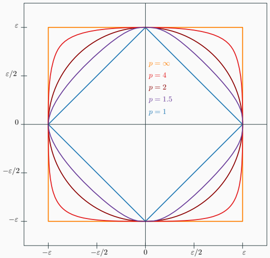
# State of the Art

| local Hausdorff Distance | | local **Fréchet Distance** | |
|---|---|---|---|
| $O(n^3)$ | Imai,Iri '88 | $\mathbf{O(n^3)}$ | Godau '91 |
| $O(n^2 \log n)$ | Melkman,O'Rourke '88 | $O(n^{2.5})$ | Buchin et al.'22 |
| $O(n^2)$ | Chan,Chin '88 | $O(n \log n)$ | *Approximation* |
| | | | Agarwal et al.'05 |

# State of the Art

| local Hausdorff Distance | |
|---|---|
| $O(n^3)$ | Imai,Iri '88 |
| $O(n^2 \log n)$ | Melkman,O'Rourke '88 |
| $O(n^2)$ | Chan,Chin '88 |

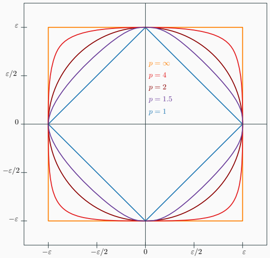
| local **Fréchet Distance** | |
|---|---|
| $\mathbf{O(n^3)}$ | Godau '91 |
| $O(n^{2.5})$ | Buchin et al.'22 |
| $O(n \log n)$ | *Approximation* |
| | Agarwal et al.'05 |
| $\mathbf{O(n^2 \log n)}$ | $L_2$, $L_{p\in(1,\infty)}$ |




**NEW**

# State of the Art

| local Hausdorff Distance | |
|---|---|
| $O(n^3)$ | Imai,Iri '88 |
| $O(n^2 \log n)$ | Melkman,O'Rourke '88 |
| $O(n^2)$ | Chan,Chin '88 |




| local **Fréchet Distance** | |
|---|---|
| $\mathbf{O(n^3)}$ | Godau '91 |
| $O(n^{2.5})$ | Buchin et al.'22 |
| $O(n \log n)$ | *Approximation* |
| | Agarwal et al.'05 |
| $\mathbf{O(n^2 \log n)}$ | $L_2$, $L_{p\in(1,\infty)}$ |
| $\mathbf{O(n^2)}$ | $L_1$, $L_\infty$ |

NEW

# Algorithm by Imai and Iri

- proceeds in two phases

- First phase
  - valid shortcuts (brute force)
  - build shortcut graph


- Second phase
  - shortest path in graph
  - optimal simplification

→ total running time $O(n^3)$

# Algorithm by Imai and Iri

- proceeds in two phases

- First phase
  - valid shortcuts (brute force)
  - build shortcut graph


- Second phase
  - shortest path in graph
  - optimal simplification

→ total running time $O(n^3)$

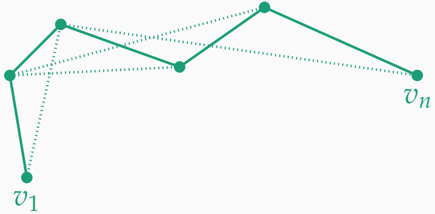
# Algorithm by Imai and Iri

- proceeds in two phases

- First phase
  - valid shortcuts (brute force)
  - build shortcut graph

- Second phase
  - shortest path in graph
  - optimal simplification

→ total running time $O(n^3)$



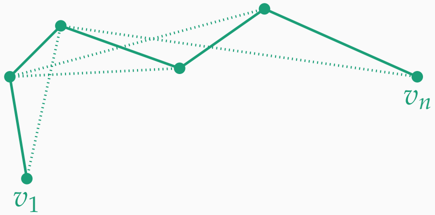
# Algorithm by Imai and Iri

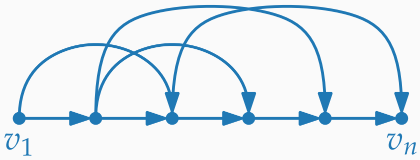
- proceeds in two phases

- First phase
  - valid shortcuts (brute force)
  - build shortcut graph

- Second phase
  - shortest path in graph
  - optimal simplification

→ total running time $O(n^3)$



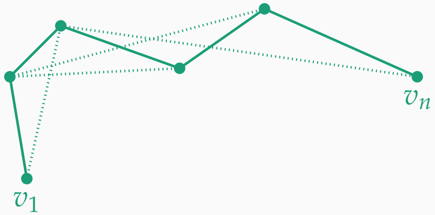
# Algorithm by Imai and Iri

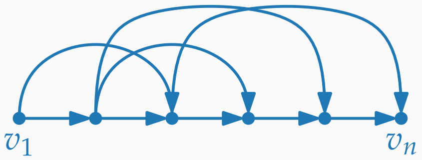
- proceeds in two phases

- First phase
  - valid shortcuts (brute force)
  - build shortcut graph

- Second phase
  - shortest path in graph
  - optimal simplification

→ total running time $O(n^3)$



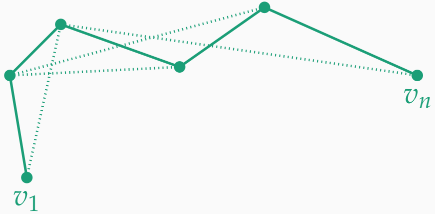
# Algorithm by Imai and Iri

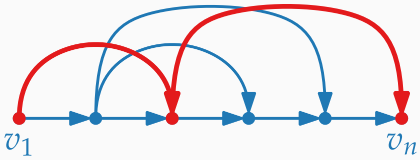
- proceeds in two phases

- First phase
  - valid shortcuts (brute force)
  - build shortcut graph

- Second phase
  - shortest path in graph
  - optimal simplification

→ total running time $O(n^3)$



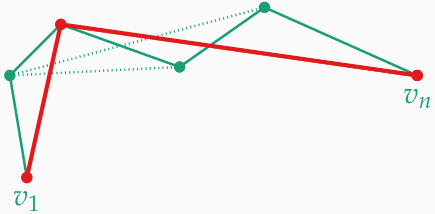
# Algorithm by Imai and Iri

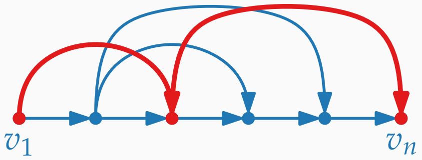
- proceeds in two phases

- First phase
  - valid shortcuts (brute force)
  - build shortcut graph

- Second phase
  - shortest path in graph
  - optimal simplification

→ total running time $O(n^3)$



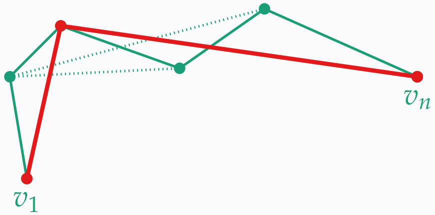
# Algorithm by Imai and Iri

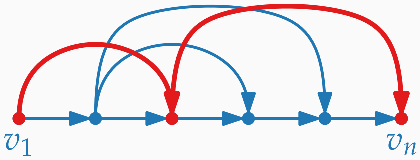
- proceeds in two phases

- First phase $O(n^3)$
  - valid shortcuts (brute force)
  - build shortcut graph $O(n^2)$

- Second phase
  - shortest path in graph
  - optimal simplification

→ total running time $O(n^3)$

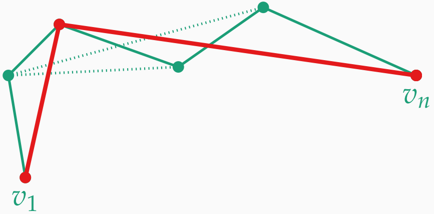


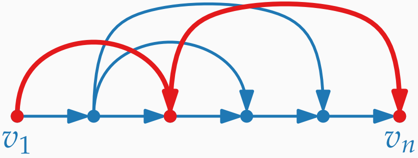



# Algorithm by Imai and Iri

- proceeds in two phases
- First phase $O(n^3)$
  - valid shortcuts (brute force)
  - build shortcut graph $O(n^2)$
- Second phase $O(n^2)$
  - shortest path in graph
  - optimal simplification

→ total running time $O(n^3)$

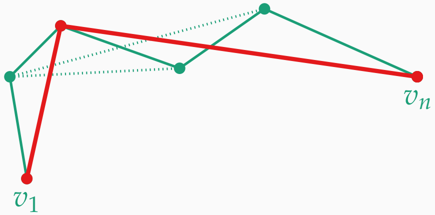


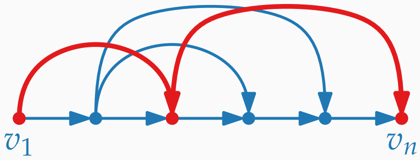



# Algorithm by Imai and Iri

- proceeds in two phases
- First phase $O(n^3)$
  - valid shortcuts (brute force)
  - build shortcut graph $O(n^2)$
- Second phase $O(n^2)$
  - shortest path in graph
  - optimal simplification

→ total running time $O(n^3)$



# Algorithm by Melkman & O'Rourke

- for each vertex $v_i,\ i \in \{1, \ldots, n\}$
  - traverse each subsequent vertex $v_j,\ j > i$
  - while maintaining a **cone**
  - and a **wave front**


- $\overline{v_i v_j}$ is a **valid** shortcut

  $\Leftrightarrow$

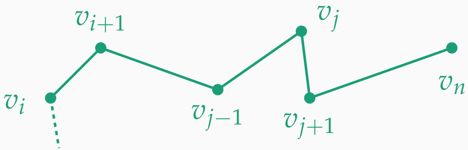
  $v_j$ is **inside** the cone and **behind** the wave front

- can be updated **incrementally** ☺

- why is the wave front used at all?

# Algorithm by Melkman & O'Rourke

- for each vertex $v_i,\ i \in \{1, \ldots, n\}$
  - traverse each subsequent vertex $v_j,\ j > i$
  - while maintaining a **cone**
  - and a **wave front**




- $\overline{v_i v_j}$ is a **valid** shortcut

  $\Leftrightarrow$

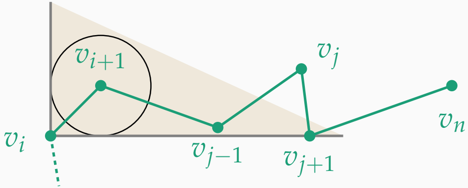
  $v_j$ is **inside** the cone and **behind** the wave front

- can be updated **incrementally** ☺

- why is the wave front used at all?

# Algorithm by Melkman & O'Rourke

- for each vertex $v_i,\ i \in \{1, \dots, n\}$
  - traverse each subsequent vertex $v_j,\ j > i$
  - while maintaining a **cone**
  - and a **wave front**




- $\overline{v_i v_j}$ is a **valid** shortcut

  $\Leftrightarrow$

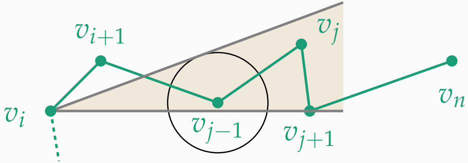
  $v_j$ is **inside** the cone and **behind** the wave front

- can be updated **incrementally** ☺

- why is the wave front used at all?

# Algorithm by Melkman & O'Rourke

- for each vertex $v_i,\ i \in \{1, \ldots, n\}$
  - traverse each subsequent vertex $v_j,\ j > i$
  - while maintaining a **cone**
  - and a **wave front**




- $\overline{v_i v_j}$ is a **valid** shortcut

  $\Leftrightarrow$

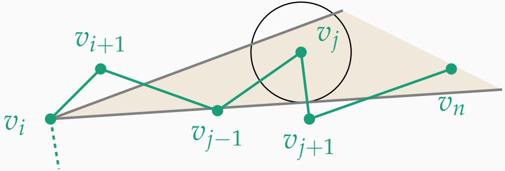
  $v_j$ is **inside** the cone and **behind** the wave front

- can be updated **incrementally** ☺

- why is the wave front used at all?

# Algorithm by Melkman & O'Rourke

- for each vertex $v_i,\ i \in \{1, \ldots, n\}$
  - traverse each subsequent vertex $v_j,\ j > i$
  - while maintaining a **cone**
  - and a **wave front**




- $\overline{v_i v_j}$ is a **valid** shortcut

  $\Leftrightarrow$

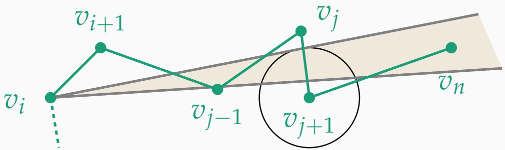
  $v_j$ is **inside** the cone and **behind** the wave front

- can be updated **incrementally** ☺

- why is the wave front used at all?

# Algorithm by Melkman & O'Rourke

- for each vertex $v_i,\ i \in \{1, \ldots, n\}$
  - traverse each subsequent vertex $v_j,\ j > i$
  - while maintaining a **cone**
  - and a **wave front**




- $\overline{v_i v_j}$ is a **valid** shortcut

  $\Leftrightarrow$

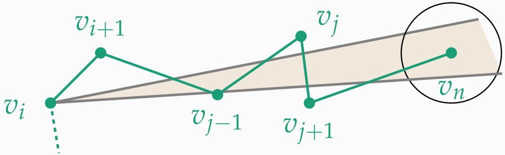
  $v_j$ is **inside** the cone and **behind** the wave front

- can be updated **incrementally** ☺

- why is the wave front used at all?

# Algorithm by Melkman & O'Rourke

- for each vertex $v_i,\ i \in \{1, \ldots, n\}$
  - traverse each subsequent vertex $v_j,\ j > i$
  - while maintaining a **cone**
  - and a **wave front**




- $\overline{v_i v_j}$ is a **valid** shortcut

  $\Leftrightarrow$

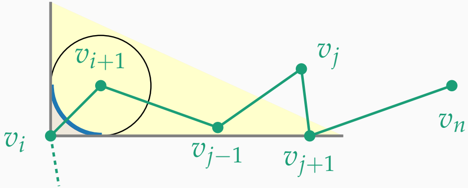
  $v_j$ is **inside** the cone and **behind** the wave front

- can be updated **incrementally** ☺

- why is the wave front used at all?

# Algorithm by Melkman & O'Rourke

- for each vertex $v_i,\ i \in \{1, \ldots, n\}$
  - traverse each subsequent vertex $v_j,\ j > i$
  - while maintaining a **cone**
  - and a **wave front**




- $\overline{v_i v_j}$ is a **valid** shortcut

  $\Leftrightarrow$

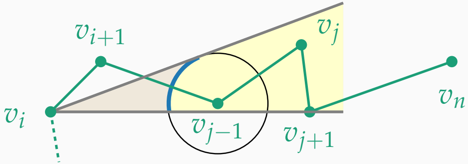
  $v_j$ is **inside** the cone and **behind** the wave front

- can be updated **incrementally** ☺

- why is the wave front used at all?

# Algorithm by Melkman & O'Rourke

- for each vertex $v_i,\ i \in \{1, \ldots, n\}$
  - traverse each subsequent vertex $v_j,\ j > i$
  - while maintaining a **cone**
  - and a **wave front**




- $\overline{v_i v_j}$ is a **valid** shortcut

  $\Leftrightarrow$

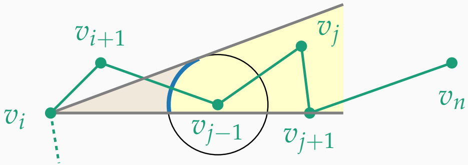
  $v_j$ is **inside** the cone and **behind** the wave front

- can be updated **incrementally** ☺

- why is the wave front used at all?

# Algorithm by Melkman & O'Rourke

- for each vertex $v_i,\ i \in \{1, \ldots, n\}$
  - traverse each subsequent vertex $v_j,\ j > i$
  - while maintaining a **cone**
  - and a **wave front**




- $\overline{v_i v_j}$ is a **valid** shortcut

  $\Leftrightarrow$

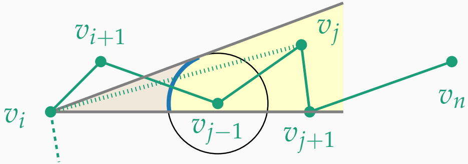
  $v_j$ is **inside** the cone and **behind** the wave front

- can be updated **incrementally** ☺

- why is the wave front used at all?

# Algorithm by Melkman & O'Rourke

- for each vertex $v_i,\ i \in \{1, \ldots, n\}$
  - traverse each subsequent vertex $v_j,\ j > i$
  - while maintaining a **cone**
  - and a **wave front**




- $\overline{v_i v_j}$ is a **valid** shortcut

  $\Leftrightarrow$

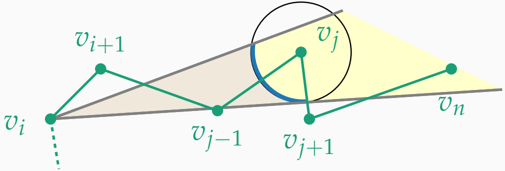
  $v_j$ is **inside** the cone and **behind** the wave front

- can be updated **incrementally** ☺

- why is the wave front used at all?

# Algorithm by Melkman & O'Rourke

- for each vertex $v_i,\ i \in \{1, \dots, n\}$
  - traverse each subsequent vertex $v_j,\ j > i$
  - while maintaining a **cone**
  - and a **wave front**




- $\overline{v_i v_j}$ is a **valid** shortcut

  $\Leftrightarrow$

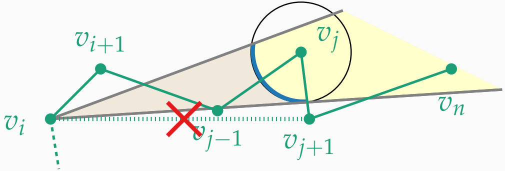
  $v_j$ is **inside** the cone and **behind** the wave front

- can be updated **incrementally** ☺

- why is the wave front used at all?

# Algorithm by Melkman & O'Rourke

- for each vertex $v_i,\ i \in \{1, \ldots, n\}$
  - traverse each subsequent vertex $v_j,\ j > i$
  - while maintaining a **cone**
  - and a **wave front**




- $\overline{v_i v_j}$ is a **valid** shortcut

  $\Leftrightarrow$

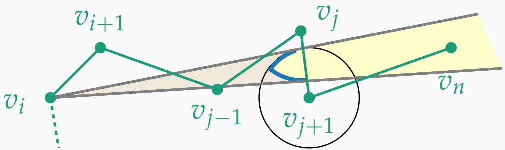
  $v_j$ is **inside** the cone and **behind** the wave front

- can be updated **incrementally** ☺

- why is the wave front used at all?

# Algorithm by Melkman & O'Rourke

- for each vertex $v_i,\ i \in \{1, \ldots, n\}$
  - traverse each subsequent vertex $v_j,\ j > i$
  - while maintaining a **cone**
  - and a **wave front**




- $\overline{v_i v_j}$ is a **valid** shortcut

  $\Leftrightarrow$

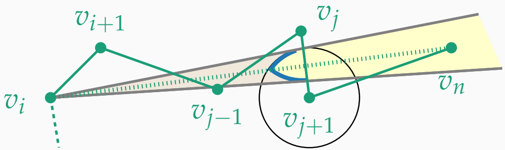
  $v_j$ is **inside** the cone and **behind** the wave front

- can be updated **incrementally** ☺

- why is the wave front used at all?

# Algorithm by Melkman & O'Rourke

- for each vertex $v_i, \ i \in \{1, \ldots, n\}$
  - traverse each subsequent vertex $v_j, \ j > i$
  - while maintaining a **cone**
  - and a **wave front**




- $\overline{v_i v_j}$ is a **valid** shortcut

  $\Leftrightarrow$

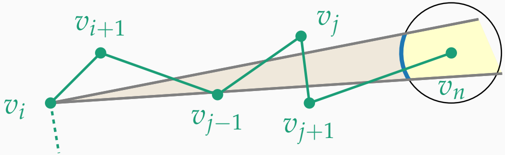
  $v_j$ is **inside** the cone and **behind** the wave front

- can be updated **incrementally** ☺

- why is the wave front used at all?

# Algorithm by Melkman & O'Rourke

- for each vertex $v_i,\ i \in \{1, \ldots, n\}$
  - traverse each subsequent vertex $v_j,\ j > i$
  - while maintaining a **cone**
  - and a **wave front**




- $\overline{v_i v_j}$ is a **valid** shortcut

  $\Leftrightarrow$

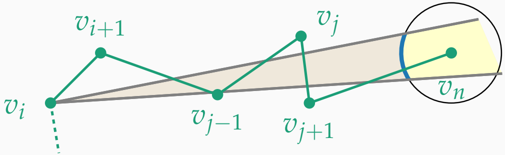
  $v_j$ is **inside** the cone and **behind** the wave front

- can be updated **incrementally** ☺

- why is the wave front used at all?

# Algorithm by Melkman & O'Rourke

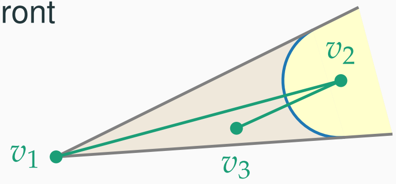
- for each vertex $v_i,\ i \in \{1,\ldots,n\}$
  - traverse each subsequent vertex $v_j,\ j > i$
  - while maintaining a **cone**
  - and a **wave front**




- $\overline{v_i v_j}$ is a **valid** shortcut

  $\Leftrightarrow$

  $v_j$ is **inside** the cone and **behind** the wave front

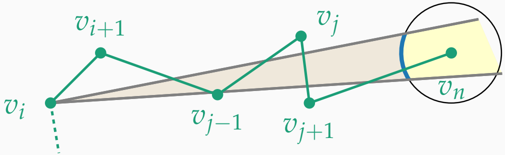



- can be updated **incrementally** ☺

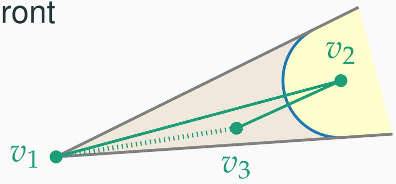
- why is the wave front used at all?

# Algorithm by Melkman & O'Rourke

- for each vertex $v_i,\ i \in \{1, \dots, n\}$
  - traverse each subsequent vertex $v_j,\ j > i$
  - while maintaining a **cone**
  - and a **wave front**




- $\overline{v_i v_j}$ is a **valid** shortcut
  $\Leftrightarrow$
  $v_j$ is **inside** the cone and **behind** the wave front

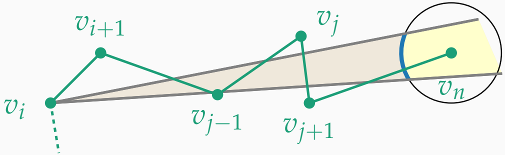



- can be updated **incrementally** ☺

- why is the wave front used at all?

# Algorithm by Melkman & O'Rourke

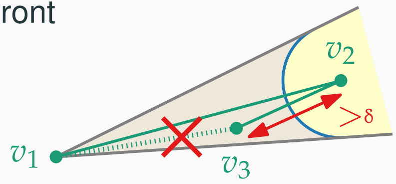
- for each vertex $v_i,\ i \in \{1, \ldots, n\}$
  - traverse each subsequent vertex $v_j,\ j > i$
  - while maintaining a **cone**
  - and a **wave front**




- $\overline{v_i v_j}$ is a **valid** shortcut

  $\Leftrightarrow$

  $v_j$ is **inside** the cone and **behind** the wave front




- can be updated **incrementally** ☺

- why is the wave front used at all?

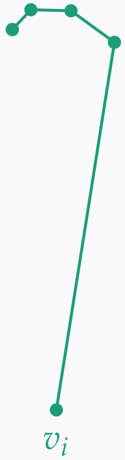
# The Wave Front

- is a sequence of circular arcs
- how complex can it be?
- each vertex contributes $\leq 1$ arc → $O(n)$ arcs

→ stored in a binary tree

- find intersections: $O(\log n)$
- update: $O(\log n)$, amortized

→ total time $\mathbf{O(n^2 \log n)}$



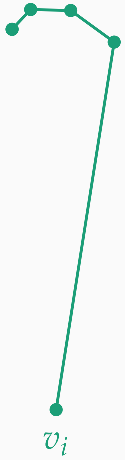
# The Wave Front

- is a sequence of circular arcs
- how complex can it be?
- each vertex contributes $\leq 1$ arc → $O(n)$ arcs

→ stored in a binary tree

- find intersections: $O(\log n)$
- update: $O(\log n)$, amortized

→ total time $\mathbf{O(n^2 \log n)}$



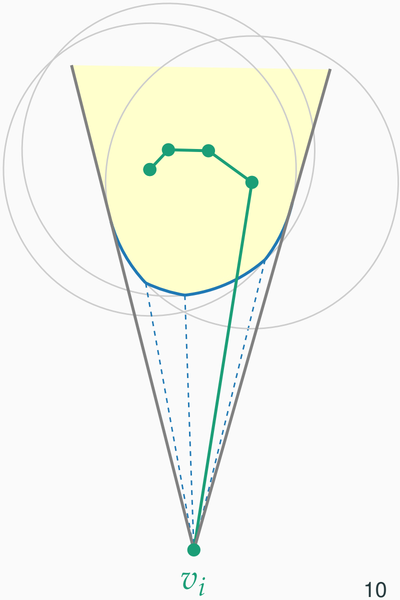
# The Wave Front

- is a sequence of circular arcs
- how complex can it be?
- each vertex contributes $\leq 1$ arc → $O(n)$ arcs

→ stored in a binary tree

- find intersections: $O(\log n)$
- update: $O(\log n)$, amortized

→ total time $\mathbf{O(n^2 \log n)}$



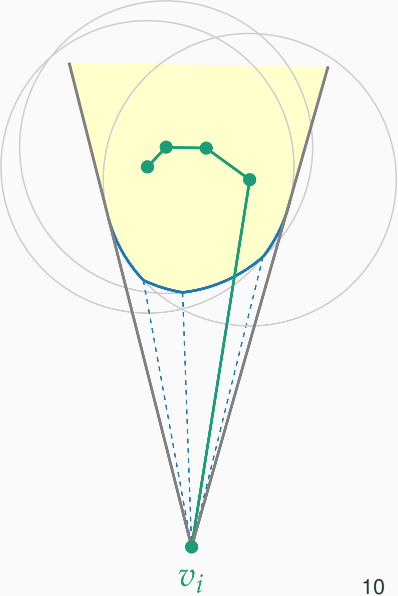
# The Wave Front

- is a sequence of circular arcs
- how complex can it be?
- each vertex contributes $\leq 1$ arc → $O(n)$ arcs

→ stored in a binary tree

- find intersections: $O(\log n)$
- update: $O(\log n)$, amortized

→ total time $\mathbf{O(n^2 \log n)}$



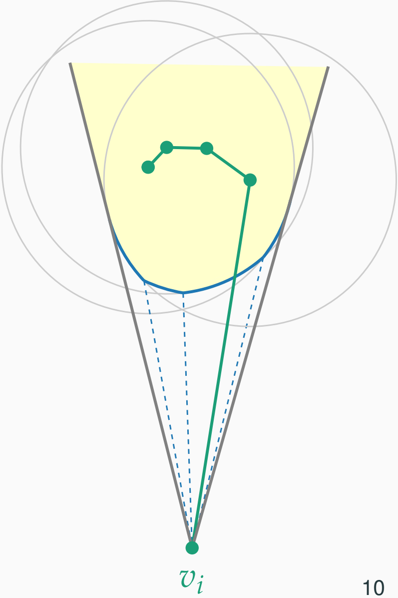
# The Wave Front

- is a sequence of circular arcs
- how complex can it be?
- each vertex contributes $\leq 1$ arc → $O(n)$ arcs

→ stored in a binary tree

- find intersections: $O(\log n)$
- update: $O(\log n)$, amortized

→ total time $\mathbf{O(n^2 \log n)}$



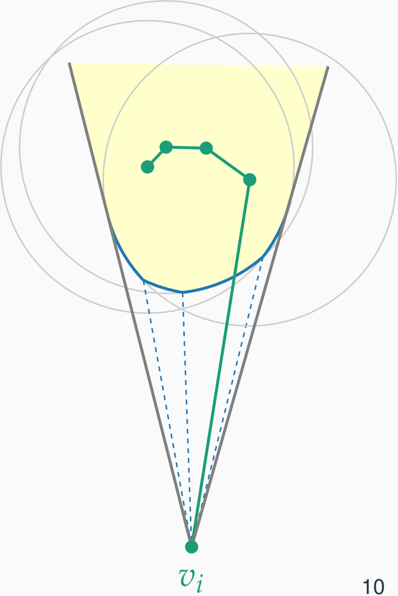
# The Wave Front

- is a sequence of circular arcs
- how complex can it be?
- each vertex contributes $\leq 1$ arc → $O(n)$ arcs

→ stored in a binary tree

- find intersections: $O(\log n)$
- update: $O(\log n)$, amortized

→ total time $\mathbf{O(n^2 \log n)}$



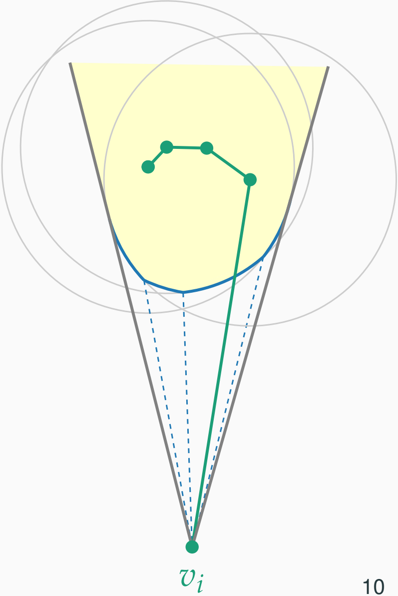
# The Wave Front

- is a sequence of circular arcs
- how complex can it be?
- each vertex contributes $\leq 1$ arc → $O(n)$ arcs

→ stored in a binary tree

- find intersections: $O(\log n)$
- update: $O(\log n)$, amortized

→ total time $\mathbf{O(n^2 \log n)}$



# Our Modifications

- **order** of vertices matters for Fréchet distance

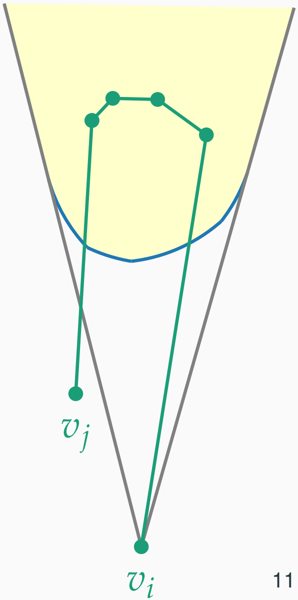
→ narrow cone s.t. $\delta$-circle contains the **whole wave front**

- determine $\leq 2$ intersections

→ can be done in $O(\log n)$

- narrow the cone

- correctness, for a shortcut $\overline{v_i v_k}$: map each intermediate $v_j$ to the intersection of $v_j$'s wave front and $\overline{v_i v_k}$



# Our Modifications

- **order** of vertices matters for Fréchet distance

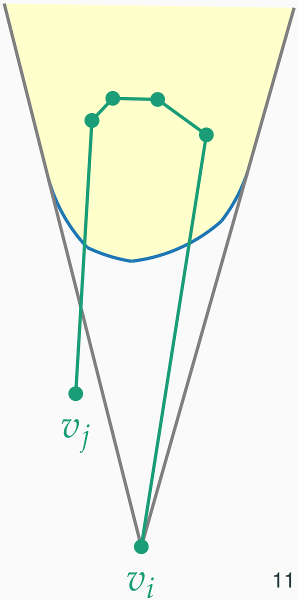
→ narrow cone s.t. $\delta$-circle contains the **whole wave front**

- determine $\leq 2$ intersections

→ can be done in $O(\log n)$

- narrow the cone

- correctness, for a shortcut $\overline{v_i v_k}$:
  map each intermediate $v_j$ to the intersection of $v_j$'s wave front and $\overline{v_i v_k}$



# Our Modifications

- **order** of vertices matters for Fréchet distance

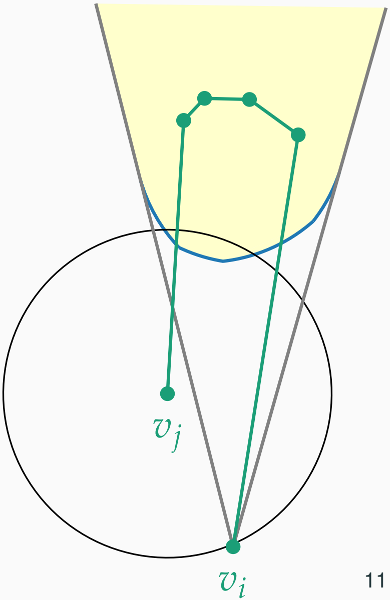
→ narrow cone s.t. $\delta$-circle contains the **whole wave front**

- determine $\leq 2$ intersections

→ can be done in $O(\log n)$

- narrow the cone

- correctness, for a shortcut $\overline{v_i v_k}$: map each intermediate $v_j$ to the intersection of $v_j$'s wave front and $\overline{v_i v_k}$



# Our Modifications

- **order** of vertices matters for Fréchet distance

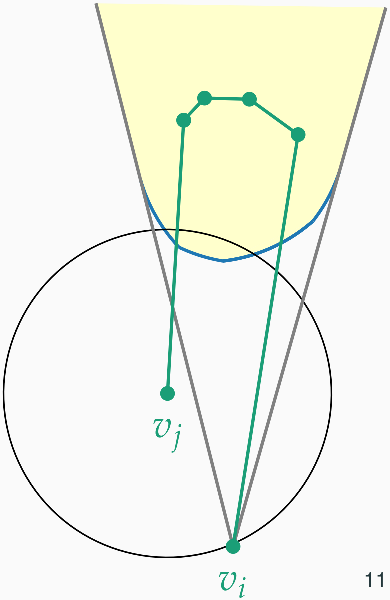
→ narrow cone s.t. $\delta$-circle contains the **whole wave front**

- determine $\leq 2$ intersections

→ can be done in $O(\log n)$

- narrow the cone

- correctness, for a shortcut $\overline{v_i v_k}$: map each intermediate $v_j$ to the intersection of $v_j$'s wave front and $\overline{v_i v_k}$



# Our Modifications

- **order** of vertices matters for Fréchet distance

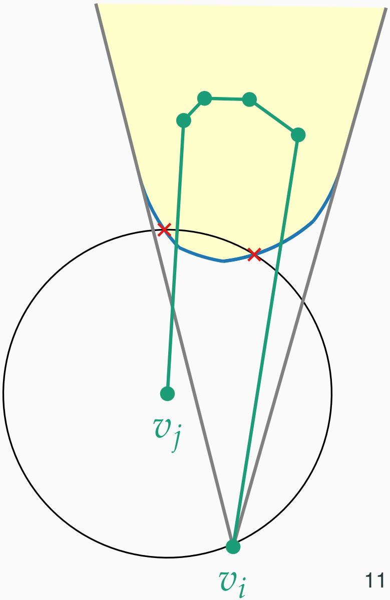
→ narrow cone s.t. $\delta$-circle contains the **whole wave front**

- determine $\leq 2$ intersections

→ can be done in $O(\log n)$

- narrow the cone

- correctness, for a shortcut $\overline{v_i v_k}$:
  map each intermediate $v_j$ to the intersection of $v_j$'s wave front and $\overline{v_i v_k}$



# Our Modifications

- **order** of vertices matters for Fréchet distance

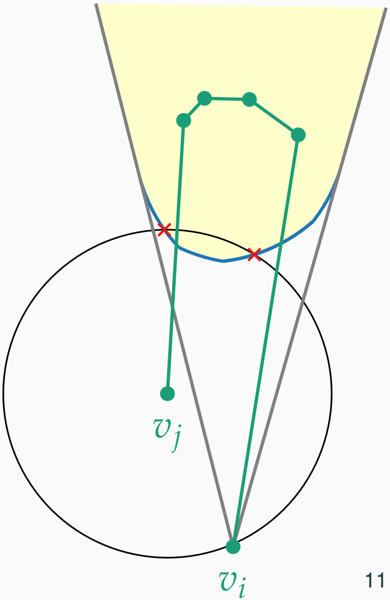
→ narrow cone s.t. $\delta$-circle contains the **whole wave front**

- determine $\leq 2$ intersections

→ can be done in $O(\log n)$

- narrow the cone

- correctness, for a shortcut $\overline{v_i v_k}$: map each intermediate $v_j$ to the intersection of $v_j$'s wave front and $\overline{v_i v_k}$



# Our Modifications

- **order** of vertices matters for Fréchet distance

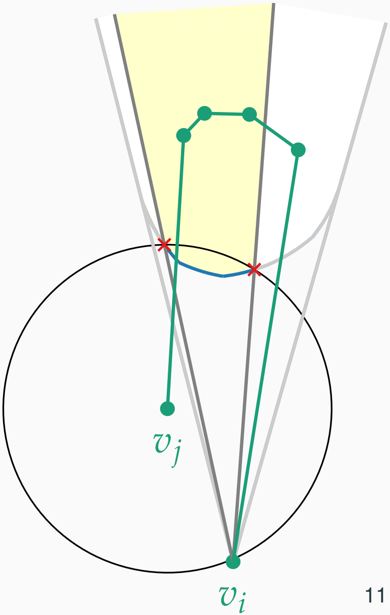
→ narrow cone s.t. $\delta$-circle contains the **whole wave front**

- determine $\leq 2$ intersections

→ can be done in $O(\log n)$

- narrow the cone

- correctness, for a shortcut $\overline{v_i v_k}$: map each intermediate $v_j$ to the intersection of $v_j$'s wave front and $\overline{v_i v_k}$



# Our Modifications

- **order** of vertices matters for Fréchet distance

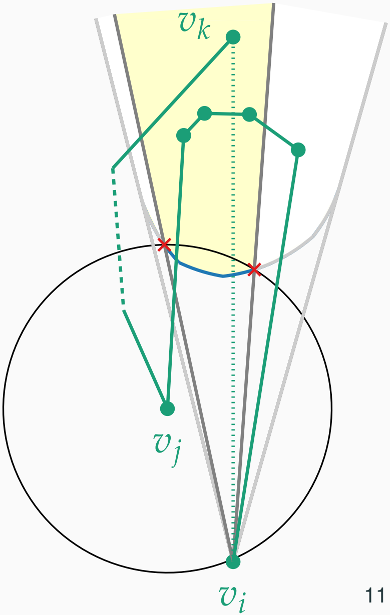
→ narrow cone s.t. $\delta$-circle contains the **whole wave front**

- determine $\leq 2$ intersections

→ can be done in $O(\log n)$

- narrow the cone

- correctness, for a shortcut $\overline{v_i v_k}$: map each intermediate $v_j$ to the intersection of $v_j$'s wave front and $\overline{v_i v_k}$



# Our Modifications

- **order** of vertices matters for Fréchet distance

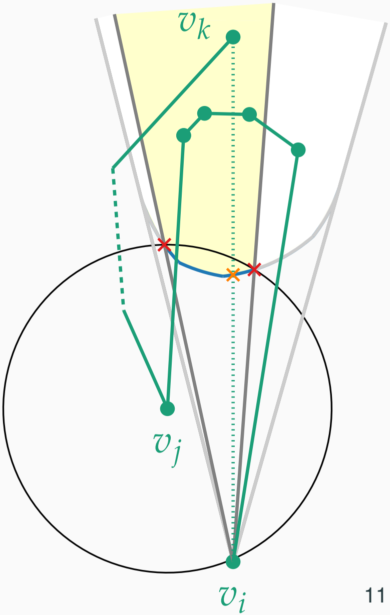
→ narrow cone s.t. $\delta$-circle contains the **whole wave front**

- determine $\leq 2$ intersections

→ can be done in $O(\log n)$

- narrow the cone

- correctness, for a shortcut $\overline{v_i v_k}$: map each intermediate $v_j$ to the intersection of $v_j$'s wave front and $\overline{v_i v_k}$



# Our Modifications

- **order** of vertices matters for Fréchet distance

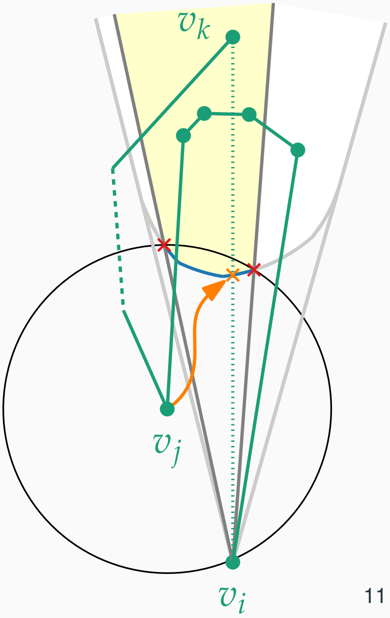
→ narrow cone s.t. $\delta$-circle contains the **whole wave front**

- determine $\leq 2$ intersections

→ can be done in $O(\log n)$

- narrow the cone

- correctness, for a shortcut $\overline{v_i v_k}$: map each intermediate $v_j$ to the intersection of $v_j$'s wave front and $\overline{v_i v_k}$



# Our Modifications

- **order** of vertices matters for Fréchet distance

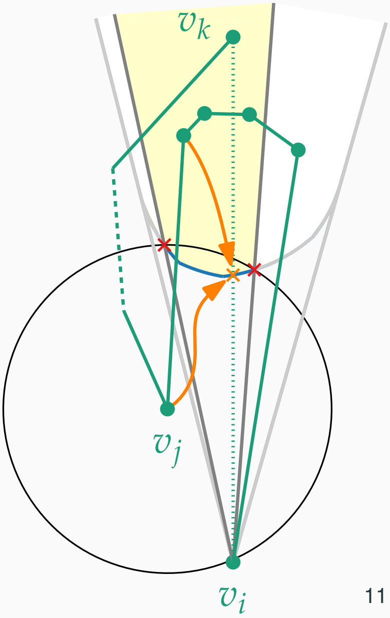
→ narrow cone s.t. $\delta$-circle contains the **whole wave front**

- determine $\leq 2$ intersections

→ can be done in $O(\log n)$

- narrow the cone

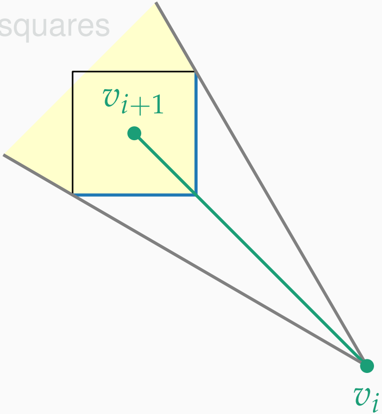
- correctness, for a shortcut $\overline{v_i v_k}$: map each intermediate $v_j$ to the intersection of $v_j$'s wave front and $\overline{v_i v_k}$



# $L_1$ and $L_\infty$ Norms

- sum-norm $L_1$, max-norm $L_\infty$
- observation: the wave front consists of at most **two line segments**
- created by intersecting axis-parallel squares

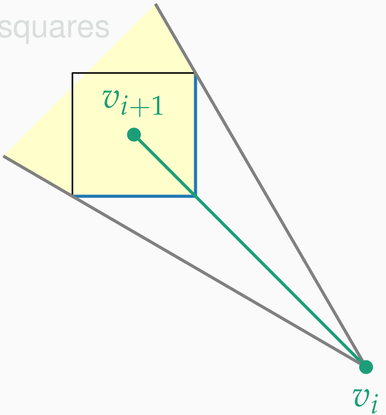
→ total time $\mathbf{O(n^2)}$



# $L_1$ and $L_\infty$ Norms

- sum-norm $L_1$, max-norm $L_\infty$
- observation: the wave front consists of at most **two line segments**
- created by intersecting axis-parallel squares

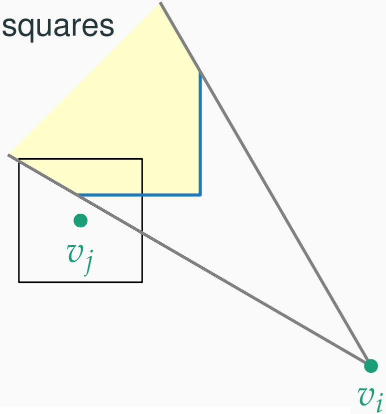
→ total time $\mathbf{O(n^2)}$



# $L_1$ and $L_\infty$ Norms

- sum-norm $L_1$, max-norm $L_\infty$
- observation: the wave front consists of at most **two line segments**
- created by intersecting axis-parallel squares

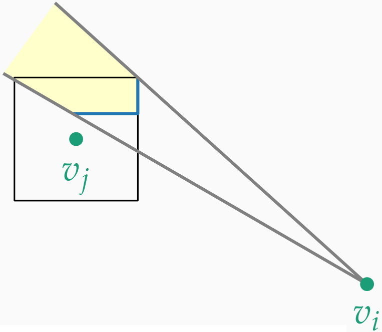
→ total time $\mathbf{O(n^2)}$



# $L_1$ and $L_\infty$ Norms

- sum-norm $L_1$, max-norm $L_\infty$
- observation: the wave front consists of at most **two line segments**
- created by intersecting axis-parallel squares

→ total time $\mathbf{O(n^2)}$



# Constructing Worst-case Instances

- worst case = wavefront size $O(n)$
- we can build worst-case examples ✓

→ $O(n^2 \log n)$ is **tight**

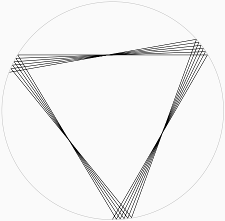
- **but**: worst cases are **contrived**, unlikely to appear in the wild

# Constructing Worst-case Instances

- worst case = wavefront size $O(n)$
- we can build worst-case examples ✓

→ $O(n^2 \log n)$ is **tight**

- **but**: worst cases are **contrived**, unlikely to appear in the wild



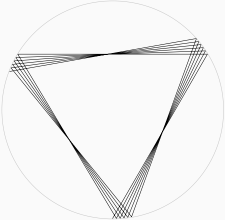
input curve

# Constructing Worst-case Instances

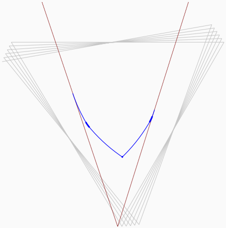
- worst case = wavefront size $O(n)$
- we can build worst-case examples ✓

→ $O(n^2 \log n)$ is **tight**

- **but**: worst cases are **contrived**, unlikely to appear in the wild



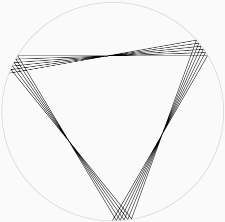
input curve



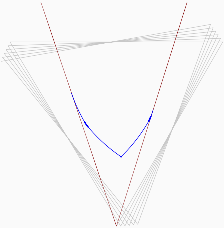
wavefront

# Constructing Worst-case Instances

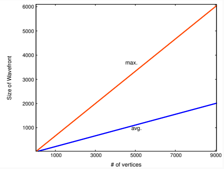
- worst case = wavefront size $O(n)$
- we can build worst-case examples ✓
- → $O(n^2 \log n)$ is **tight**
- **but**: worst cases are **contrived**, unlikely to appear in the wild



input curve



wavefront



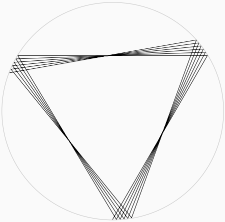

grows linear with $n$

# Constructing Worst-case Instances

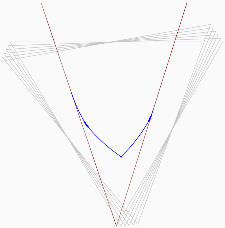
- worst case = wavefront size $O(n)$
- we can build worst-case examples ✓

→ $O(n^2 \log n)$ is **tight**

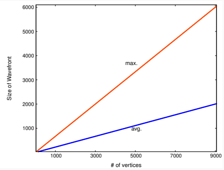
- **but**: worst cases are **contrived**, unlikely to appear in the wild



input curve



wavefront




grows linear with $n$

# Wavefront Size

**Conjecture**

- worst-case is rare
- there is a ‘natural’ tendency to keep wavefronts small

→ total running time close to $\mathbf{O(n^2)}$ ☺

- open question: condition for worst-case instances
- in the mean time...

# Wavefront Size

**Conjecture**

- worst-case is rare
- there is a ‘natural’ tendency to keep wavefronts small
- → total running time close to $\mathbf{O(n^2)}$ ☺

- open question: condition for worst-case instances ?
- in the mean time...

# Wavefront Size

**Conjecture**

- worst-case is rare
- there is a 'natural' tendency to keep wavefronts small

→ total running time close to $\mathbf{O(n^2)}$ ☺

- open question: condition for worst-case instances ?
- in the mean time...

# Wavefront Size

**Conjecture**

- worst-case is rare
- there is a 'natural' tendency to keep wavefronts small

→ total running time close to $\mathbf{O(n^2)}$ ☺

- open question: condition for worst-case instances ?
- in the mean time...

# Wavefront Size

**Conjecture**

- worst-case is rare
- there is a ‘natural’ tendency to keep wavefronts small

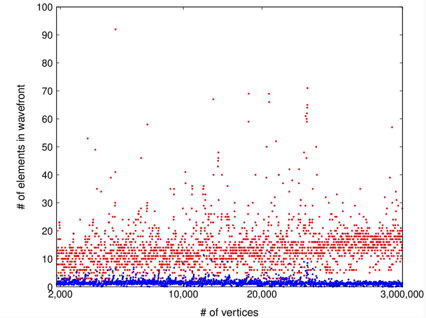
→ total running time close to $\mathbf{O(n^2)}$ ☺

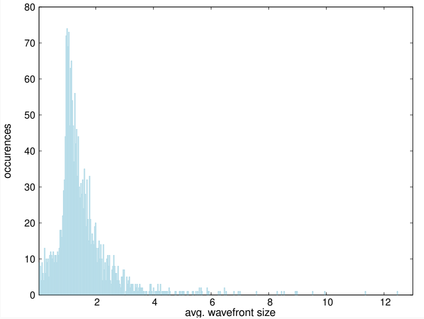
- open question: condition for worst-case instances ?
- in the mean time...
  evaluate on real-world data

# Real-world Instances

- **real-world** data: trajectories from OSM, up to 3 Mio. vertices
- Wave-fronts are always **small**: avg. $\leq 6$, max. $\leq 90$

→ **practical** running-time close to $O(n^2)$
(confirming our conjecture)

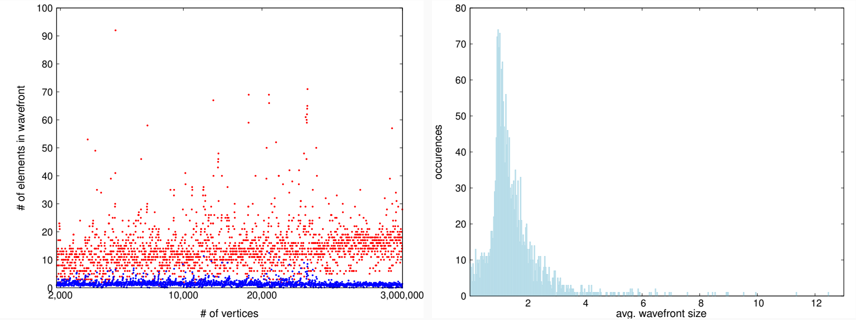


# Real-world Instances

- **real-world** data: trajectories from OSM, up to 3 Mio. vertices
- Wave-fronts are always **small**: avg. $\leq$ **6**, max. $\leq$ 90

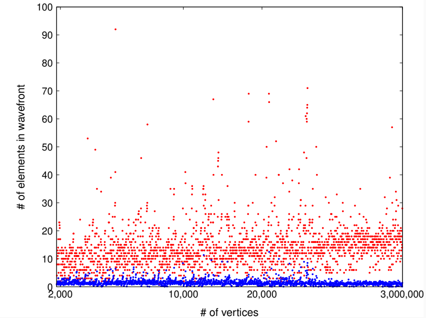
→ **practical** running-time close to $O(n^2)$
(confirming our conjecture)

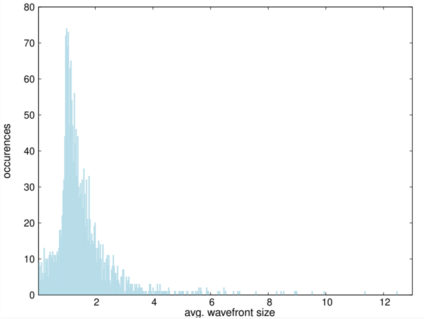


# Real-world Instances

- **real-world** data: trajectories from OSM, up to 3 Mio. vertices
- Wave-fronts are always **small**: avg. $\leq$ **6**, max. $\leq$ 90

→ **practical** running-time close to $O(n^2)$
(confirming our conjecture)

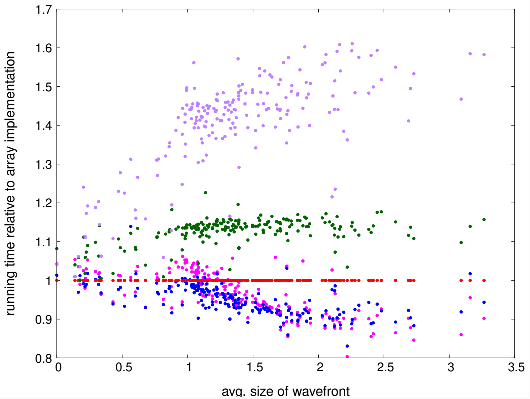


# Container Data Structure

- default: binary **tree**
  - but: left / right decisions are not for free ☹
  - but: wavefronts *are* small ≈ constant

→ try simpler containers:
linked-list, array, skip list, ...



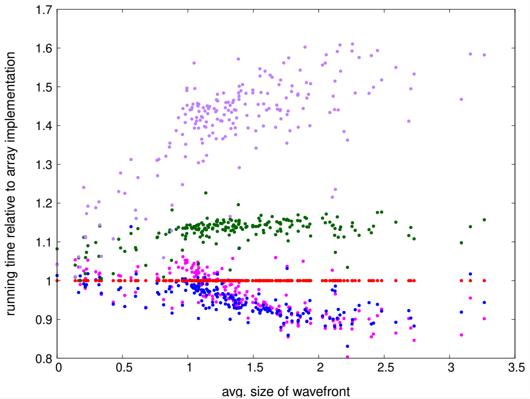

→ the winner is: **linked-list** ☺

– (except on construed worst-case instances)

# Container Data Structure

- default: binary **tree**
  - but: left / right decisions are not for free ☹
  - but: wavefronts *are* small ≈ constant

→ try simpler containers:
linked-list, array, skip list, ...



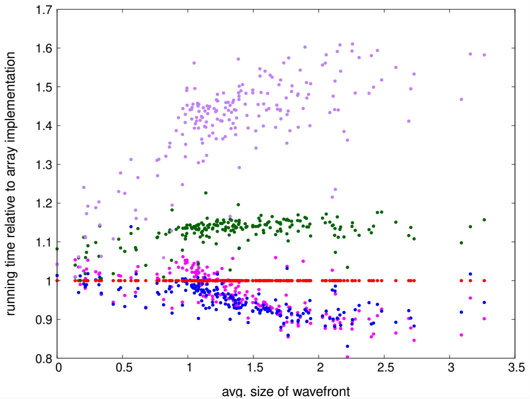

→ the winner is: **linked-list** ☺

- (except on construed worst-case instances)

# Container Data Structure

- default: binary **tree**
  - but: left / right decisions are not for free ☹
  - but: wavefronts *are* small ≈ constant

→ try simpler containers:
linked-list, array, skip list, ...




→ the winner is: **linked-list** ☺

- (except on construed worst-case instances)

# Comparison to other Algorithms

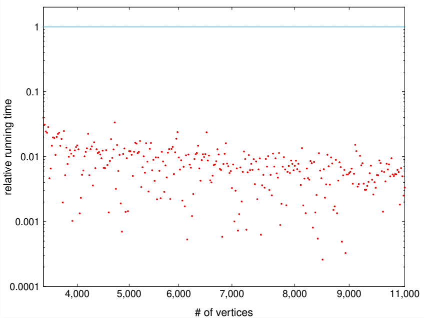
Our algorithm is ...

- **significantly** faster than state-of-the-art **Imai-Iri** / **Godau** '91
  (note that Imai-Iri is *always* $\Theta(n^3)$)
- competitive to approx. algorithm **Agarwal** et al.'05

# Comparison to other Algorithms

Our algorithm is ...

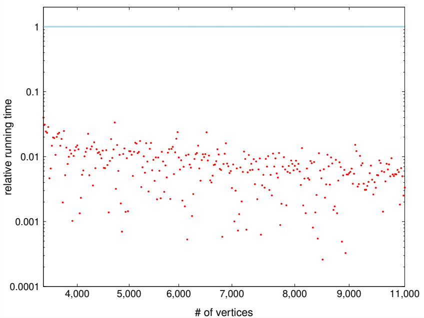
- **significantly** faster than state-of-the-art **Imai-Iri / Godau** '91
  (note that Imai-Iri is *always* $\Theta(n^3)$)
- competitive to approx. algorithm **Agarwal** et al.'05

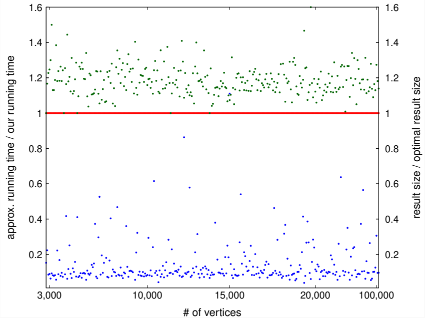


# Comparison to other Algorithms

Our algorithm is ...

- **significantly** faster than state-of-the-art **Imai-Iri** / **Godau** ’91
  (note that Imai-Iri is *always* $\Theta(n^3)$)
- competitive to approx. algorithm **Agarwal** et al.’05

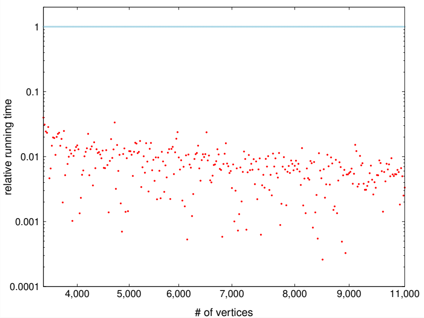


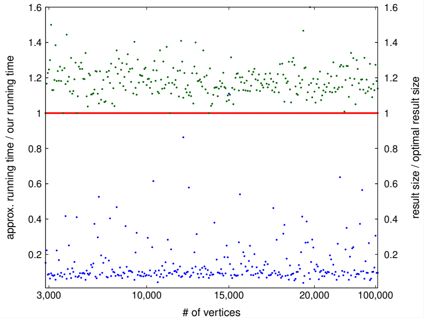



# Comparison to other Algorithms

Our algorithm is ...

- **significantly** faster than state-of-the-art **Imai-Iri** / **Godau** '91
  (note that Imai-Iri is *always* $\Theta(n^3)$)
- competitive to approx. algorithm **Agarwal** et al.'05
  (but we have **exact** results, of course)



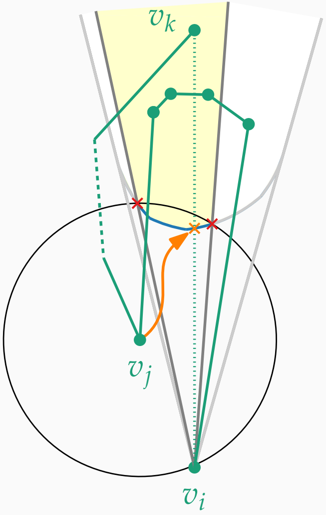



# Conclusion

- **new** algorithm (using some old ideas)
- improves state-of-the-art to $\mathbf{O(n^2 \log n)}$
- even $\mathbf{O(n^2)}$ for $L_1$, $L_\infty$
- bounds are tight, but...
- worst-case is **unlikely** on real-world data

→ practical running time ≈ $\mathbf{O(n^2)}$

- allows for simpler implementation

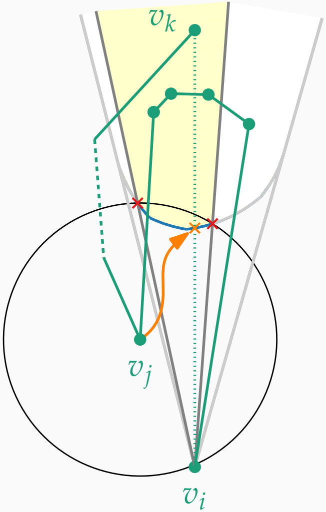


# Conclusion

- **new** algorithm (using some old ideas)
- improves state-of-the-art
  to $\mathbf{O(n^2 \log n)}$
- even $\mathbf{O(n^2)}$ for $L_1$, $L_\infty$
- bounds are tight, but...
- worst-case is **unlikely** on real-world data

→ practical running time $\approx \mathbf{O(n^2)}$

- allows for simpler implementation

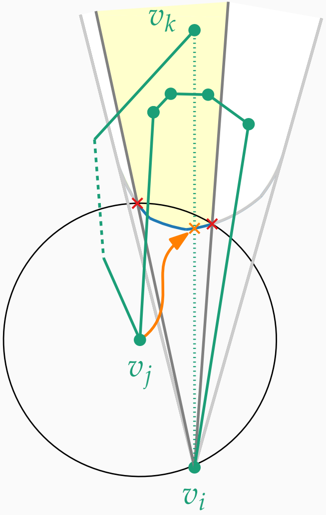


# Conclusion

- **new** algorithm (using some old ideas)
- improves state-of-the-art to $\mathbf{O(n^2 \log n)}$
- even $\mathbf{O(n^2)}$ for $L_1$, $L_\infty$
- bounds are tight, but...
- worst-case is **unlikely** on real-world data

→ practical running time $\approx \mathbf{O(n^2)}$

- allows for simpler implementation

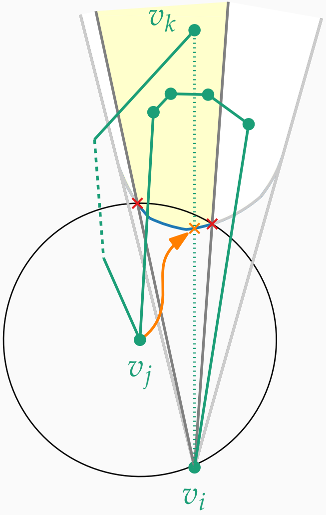


# Conclusion

- **new** algorithm (using some old ideas)
- improves state-of-the-art
  to $\mathbf{O(n^2 \log n)}$
- even $\mathbf{O(n^2)}$ for $L_1$, $L_\infty$
- bounds are tight, but...
- worst-case is **unlikely** on real-world data

→ practical running time $\approx \mathbf{O(n^2)}$

- allows for simpler implementation

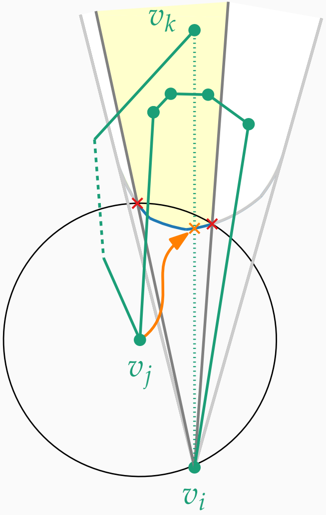


# Conclusion

- **new** algorithm (using some old ideas)
- improves state-of-the-art to $\mathbf{O(n^2 \log n)}$
- even $\mathbf{O(n^2)}$ for $L_1$, $L_\infty$
- bounds are tight, but...
- worst-case is **unlikely** on real-world data

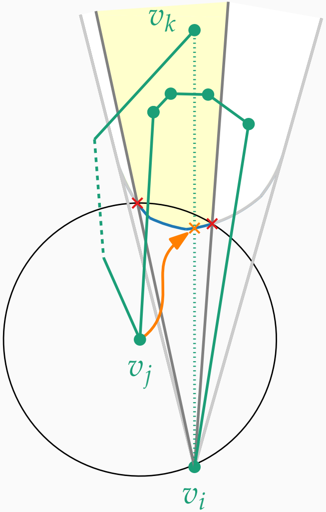
→ practical running time $\approx \mathbf{O(n^2)}$

- allows for simpler implementation



# Conclusion

- **new** algorithm (using some old ideas)
- improves state-of-the-art
  to $\mathbf{O(n^2 \log n)}$
- even $\mathbf{O(n^2)}$ for $L_1$, $L_\infty$
- bounds are tight, but...
- worst-case is **unlikely** on real-world data
- → practical running time $\approx \mathbf{O(n^2)}$
- allows for simpler implementation



# Open Questions

- define worst-case
- lower bounds $< O(n^2)$
- higher dimensions
- norms $L_{p \in (1,\infty)}$
- spherical geometry
- drop endpoints