Computational Geometry Lecture
Applications of WSPD & Visibility Graphs

Tamara Mchedlidze
25.05.2018
Recall: Well-Separated Pair Decomposition

**Def:** A pair of disjoint point sets $A$ and $B$ in $\mathbb{R}^d$ is called *s-well separated* for some $s > 0$, if $A$ and $B$ can each be covered by a ball of radius $r$ whose distance is at least $sr$. 

![Diagram](image)
Recall: Well-Separated Pair Decomposition

**Def:** A pair of disjoint point sets $A$ and $B$ in $\mathbb{R}^d$ is called \textit{s-well separated} for some $s > 0$, if $A$ and $B$ can each be covered by a ball of radius $r$ whose distance is at least $sr$.

**Def:** For a point set $P$ and some $s > 0$ an \textit{s-well separated pair decomposition} ($s$-WSPD) is a set of pairs $\{\{A_1, B_1\}, \ldots , \{A_m, B_m\}\}$ with

- $A_i, B_i \subset P$ for all $i$
- $A_i \cap B_i = \emptyset$ for all $i$
- $\bigcup_{i=1}^{m} A_i \otimes B_i = P \otimes P$
- $\{A_i, B_i\}$ \textit{s-well separated} for all $i$
Recall: Well-Separated Pair Decomposition

**Def:** A pair of disjoint point sets $A$ and $B$ in $\mathbb{R}^d$ is called *s-well separated* for some $s > 0$, if $A$ and $B$ can each be covered by a ball of radius $r$ whose distance is at least $sr$.

**Def:** For a point set $P$ and some $s > 0$ an *s-well separated pair decomposition* ($s$-WSPD) is a set of pairs $\{\{A_1, B_1\}, \ldots, \{A_m, B_m\}\}$ with
- $A_i, B_i \subset P$ for all $i$
- $A_i \cap B_i = \emptyset$ for all $i$
- $\bigcup_{i=1}^m A_i \otimes B_i = P \otimes P$
- $\{A_i, B_i\}$ $s$-well separated for all $i$

**Thm 3:** Given a point set $P$ in $\mathbb{R}^d$ and $s \geq 1$ we can construct an $s$-WSPD with $O(s^d n)$ pairs in time $O(n \log n + s^d n)$. 
Further Applications of WSPD
Euclidean MST

**Problem:** Given a point set $P$ find a minimum spanning tree (MST) in the Euclidean graph $\mathcal{E}G(P)$. 
Euclidean MST

**Problem:** Given a point set $P$ find a minimum spanning tree (MST) in the Euclidean graph $\mathcal{E}G(P)$.

**Prim:** MST in a graph $G = (V, E)$ can be computed in $O(|E| + |V| \log |V|)$ time.
Euclidean MST

**Problem:** Given a point set $P$ find a minimum spanning tree (MST) in the Euclidean graph $\mathcal{E}G(P)$.

| Prim: MST in a graph $G = (V, E)$ can be computed in $O(|E| + |V| \log |V|)$ time. |
|---------------------------------------------------------------|

- $\mathcal{E}G(P)$ has $\Theta(n^2)$ edges $\Rightarrow$ running time $O(n^2)$
- $(1 + \varepsilon)$-spanner for $P$ has $O(n/\varepsilon^d)$ edges
  $\Rightarrow$ running time $O(n \log n + n/\varepsilon^d)$
Euclidean MST

**Problem:** Given a point set $P$ find a minimum spanning tree (MST) in the Euclidean graph $\mathcal{E}G(P)$.

**Prim:** MST in a graph $G = (V, E)$ can be computed in $O(|E| + |V| \log |V|)$ time.

- $\mathcal{E}G(P)$ has $\Theta(n^2)$ edges $\Rightarrow$ running time $O(n^2)$
- $(1 + \varepsilon)$-spanner for $P$ has $O(n/\varepsilon^d)$ edges
  $\Rightarrow$ running time $O(n \log n + n/\varepsilon^d)$

How good is the MST of a $(1 + \varepsilon)$-spanner?
Euclidean MST

**Problem:** Given a point set \( P \) find a minimum spanning tree (MST) in the Euclidean graph \( \mathcal{E}G(P) \).

**Prim:** MST in a graph \( G = (V, E) \) can be computed in \( O(|E| + |V| \log |V|) \) time.

- \( \mathcal{E}G(P) \) has \( \Theta(n^2) \) edges \( \Rightarrow \) running time \( O(n^2) \)
- \( (1 + \varepsilon) \)-spanner for \( P \) has \( O(n/\varepsilon^d) \) edges \( \Rightarrow \) running time \( O(n \log n + n/\varepsilon^d) \)

How good is the MST of a \( (1 + \varepsilon) \)-spanner?

**Thm 5:** The MST obtained from a \( (1 + \varepsilon) \)-spanner of \( P \) is a \( (1 + \varepsilon) \)-approximation of the EMST of \( P \).
Diameter of $P$

**Problem:** Find the diameter of a point set $P$ (i.e., the pair $\{x, y\} \subset P$ with maximum distance).
Diameter of \( P \)

**Problem:** Find the diameter of a point set \( P \) (i.e., the pair \( \{x, y\} \subset P \) with maximum distance).

*brute-force testing all point pairs \( \Rightarrow \) running time \( O(n^2) \) :-(
  *test distances \( ||\text{rep}(u) \text{rep}(v)|| \) of all ws-pairs \( \{P_u, P_v\} \)
  \( \Rightarrow \) running time \( O(n \log n + s^d n) \) :-)*
Diameter of $P$

**Problem:** Find the diameter of a point set $P$ (i.e., the pair $\{x, y\} \subset P$ with maximum distance).

- brute-force testing all point pairs $\Rightarrow$ running time $O(n^2)$ :-(
- test distances $||\text{rep}(u) \text{rep}(v)||$ of all ws-pairs $\{P_u, P_v\}$ $\Rightarrow$ running time $O(n \log n + s^d n)$ :-)

How good is the computed diameter?
Diameter of $P$

**Problem:** Find the diameter of a point set $P$ (i.e., the pair $\{x, y\} \subset P$ with maximum distance).

- brute-force testing all point pairs $\Rightarrow$ running time $O(n^2)$ :
- test distances $||\text{rep}(u) \text{ rep}(v)||$ of all ws-pairs $\{P_u, P_v\}$$\Rightarrow$ running time $O(n \log n + s^d n)$ :

How good is the computed diameter?

**Thm 6:** The diameter obtained from an $s$-WSPD of $P$ for $s = 4/\varepsilon$ is a $(1 + \varepsilon)$-approximation of the diameter of $P$. 
Closest Pair of Points

**Problem:** Find the pair \( \{x, y\} \subset P \) with minimum distance.
Closest Pair of Points

**Problem:** Find the pair \( \{x, y\} \subset P \) with minimum distance.

- brute-force testing all point pairs \( \Rightarrow \) running time \( O(n^2) \)
- test distances \( ||\text{rep}(u) - \text{rep}(v)|| \) of all ws-pairs \( \{P_u, P_v\} \)
  \( \Rightarrow \) running time \( O(n \log n + s^d n) \)
Closest Pair of Points

**Problem:** Find the pair \( \{x, y\} \subset P \) with minimum distance.

- brute-force testing all point pairs \( \implies \) running time \( O(n^2) \) :-(
- test distances \( ||\text{rep}(u)\text{rep}(v)|| \) of all ws-pairs \( \{P_u, P_v\} \)
  \( \implies \) running time \( O(n \log n + s^d n) \) :-)

**Exercise:** For \( s > 2 \) this actually yields the closest pair.
Discussion

What are further applications of the WSPD?
Discussion

What are further applications of the WSPD?

WSPD is useful whenever one can do without knowing all $\Theta(n^2)$ exact distances in a point set and approximate them instead. One example are force-based layout algorithms in graph drawing, where pairwise repulsive forces of $n$ points need to be calculated.
Discussion

What are further applications of the WSPD?

WSPD is useful whenever one can do without knowing all $\Theta(n^2)$ exact distances in a point set and approximate them instead. One example are force-based layout algorithms in graph drawing, where pairwise repulsive forces of $n$ points need to be calculated.

Why approximate geometrically?
Discussion

What are further applications of the WSPD?

WSPD is useful whenever one can do without knowing all $\Theta(n^2)$ exact distances in a point set and approximate them instead. One example are force-based layout algorithms in graph drawing, where pairwise repulsive forces of $n$ points need to be calculated.

Why approximate geometrically?

On the one hand, this replaces slow computations by faster (but less precise) ones; on the other hand, often the input data are imprecise so that approximate solutions can be sufficient depending on the application.
Discussion

What are further applications of the WSPD?

WSPD is useful whenever one can do without knowing all \( \Theta(n^2) \) exact distances in a point set and approximate them instead. One example are force-based layout algorithms in graph drawing, where pairwise repulsive forces of \( n \) points need to be calculated.

Why approximate geometrically?

On the one hand, this replaces slow computations by faster (but less precise) ones; on the other hand, often the input data are imprecise so that approximate solutions can be sufficient depending on the application.

Can we achieve the same time bounds with exact computations?
Discussion

What are further applications of the WSPD?

WSPD is useful whenever one can do without knowing all $\Theta(n^2)$ exact distances in a point set and approximate them instead. One example are force-based layout algorithms in graph drawing, where pairwise repulsive forces of $n$ points need to be calculated.

Why approximate geometrically?

On the one hand, this replaces slow computations by faster (but less precise) ones; on the other hand, often the input data are imprecise so that approximate solutions can be sufficient depending on the application.

Can we achieve the same time bounds with exact computations?

In $\mathbb{R}^2$ this is often true, but not in $\mathbb{R}^d$ for $d > 2$. (e.g. EMST, diameter)
Motion planning and Visibility Graphs
Problem: Given a (point) robot at position $p_{\text{start}}$ in an area with polygonal obstacles, find a shortest path to $p_{\text{goal}}$ avoiding obstacles.
**Problem:** Given a (point) robot at position $p_{\text{start}}$ in a area with polygonal obstacles, find a shortest path to $p_{\text{goal}}$ avoiding obstacles.
Problem: Given a (point) robot at position $p_{\text{start}}$ in a area with polygonal obstacles, find a shortest path to $p_{\text{goal}}$ avoiding obstacles.
**Lemma 1:** For a set $S$ of disjoint open polygons in $\mathbb{R}^2$ and two points $s$ and $t$ not in $S$. 

![Diagram](image-url)
Shortest Paths in Polygonal Areas

**Lemma 1:** For a set $S$ of disjoint open polygons in $\mathbb{R}^2$ and two points $s$ and $t$ not in $S$ each shortest $st$-path in $\mathbb{R}^2 \setminus \bigcup S$ is a polygonal path whose internal vertices are vertices of $S$. 

\[ S \quad \text{and} \quad t \]
Lemma 1: For a set $S$ of disjoint open polygons in $\mathbb{R}^2$ and two points $s$ and $t$ not in $S$ each shortest $st$-path in $\mathbb{R}^2 \setminus \bigcup S$ is a polygonal path whose internal vertices are vertices of $S$. 
Lemma 1: For a set $S$ of disjoint open polygons in $\mathbb{R}^2$ and two points $s$ and $t$ not in $S$, each shortest $st$-path in $\mathbb{R}^2 \setminus \bigcup S$ is a polygonal path whose internal vertices are vertices of $S$. 

![Diagram showing shortest paths in polygonal areas]
Shortest Paths in Polygonal Areas

Lemma 1: For a set $S$ of disjoint open polygons in $\mathbb{R}^2$ and two points $s$ and $t$ not in $S$ each shortest $st$-path in $\mathbb{R}^2 \setminus \bigcup S$ is a polygonal path whose internal vertices are vertices of $S$.

Proof sketch:
Shortest Paths in Polygonal Areas

**Lemma 1:** For a set $S$ of disjoint open polygons in $\mathbb{R}^2$ and two points $s$ and $t$ not in $S$ each shortest $st$-path in $\mathbb{R}^2 \setminus \bigcup S$ is a polygonal path whose internal vertices are vertices of $S$.

**Proof sketch:**

![Diagram showing shortest path with internal vertices as vertices of polygons]
Shortest Paths in Polygonal Areas

Lemma 1: For a set $S$ of disjoint open polygons in $\mathbb{R}^2$ and two points $s$ and $t$ not in $S$ each shortest $st$-path in $\mathbb{R}^2 \setminus \bigcup S$ is a polygonal path whose internal vertices are vertices of $S$.

Proof sketch:
Shortest Paths in Polygonal Areas

**Lemma 1:** For a set $S$ of disjoint open polygons in $\mathbb{R}^2$ and two points $s$ and $t$ not in $S$, each shortest $st$-path in $\mathbb{R}^2 \setminus \bigcup S$ is a polygonal path whose internal vertices are vertices of $S$.

![Diagram showing shortest paths in polygonal areas]

**Proof sketch:**
Shortest Paths in Polygonal Areas

Lemma 1: For a set $S$ of disjoint open polygons in $\mathbb{R}^2$ and two points $s$ and $t$ not in $S$ each shortest $st$-path in $\mathbb{R}^2 \setminus \bigcup S$ is a polygonal path whose internal vertices are vertices of $S$.

Proof sketch:
Shortest Paths in Polygonal Areas

**Lemma 1:** For a set $S$ of disjoint open polygons in $\mathbb{R}^2$ and two points $s$ and $t$ not in $S$ each shortest $st$-path in $\mathbb{R}^2 \setminus \bigcup S$ is a polygonal path whose internal vertices are vertices of $S$.

Proof sketch:
Shortest Paths in Polygonal Areas

**Lemma 1:** For a set $S$ of disjoint open polygons in $\mathbb{R}^2$ and two points $s$ and $t$ not in $S$, each shortest $st$-path in $\mathbb{R}^2 \setminus \bigcup S$ is a polygonal path whose internal vertices are vertices of $S$.

**Proof sketch:**
Shortest Paths in Polygonal Areas

**Lemma 1:** For a set $S$ of disjoint open polygons in $\mathbb{R}^2$ and two points $s$ and $t$ not in $S$ each shortest $st$-path in $\mathbb{R}^2 \setminus \bigcup S$ is a polygonal path whose internal vertices are vertices of $S$.

Proof sketch:
Shortest Paths in Polygonal Areas

**Lemma 1:** For a set $S$ of disjoint open polygons in $\mathbb{R}^2$ and two points $s$ and $t$ not in $S$, each shortest $st$-path in $\mathbb{R}^2 \setminus \bigcup S$ is a polygonal path whose internal vertices are vertices of $S$.

Proof sketch:
Shortest Paths in Polygonal Areas

**Lemma 1:** For a set $S$ of disjoint open polygons in $\mathbb{R}^2$ and two points $s$ and $t$ not in $S$ each shortest $st$-path in $\mathbb{R}^2 \setminus \bigcup S$ is a polygonal path whose internal vertices are vertices of $S$.

**Proof sketch:**
Shortest Paths in Polygonal Areas

**Lemma 1:** For a set $S$ of disjoint open polygons in $\mathbb{R}^2$ and two points $s$ and $t$ not in $S$, each shortest $st$-path in $\mathbb{R}^2 \setminus \bigcup S$ is a polygonal path whose internal vertices are vertices of $S$.

**Proof sketch:**
Lemma 1: For a set $S$ of disjoint open polygons in $\mathbb{R}^2$ and two points $s$ and $t$ not in $S$, each shortest $st$-path in $\mathbb{R}^2 \setminus \bigcup S$ is a polygonal path whose internal vertices are vertices of $S$.

Proof sketch:
Visibility Graph

Given a set $S$ of disjoint open polygons...
Visibility Graph

Given a set $S$ of disjoint open polygons...

...with point set $V(S)$.
Visibility Graph

Given a set $S$ of disjoint open polygons...

Def.: Then $G_{\text{vis}}(S) = (V(S), E_{\text{vis}}(S))$ is the visibility graph of $S$ with $E_{\text{vis}}(S) = \{uv \mid u, v \in V(S) \text{ and } u \text{ sees } v\}$ and $w(uv) = |uv|$. 

...with point set $V(S)$. 

Def.: Then $G_{\text{vis}}(S) = (V(S), E_{\text{vis}}(S))$ is the visibility graph of $S$ with $E_{\text{vis}}(S) = \{uv \mid u, v \in V(S) \text{ and } u \text{ sees } v\}$ and $w(uv) = |uv|$. 

Given a set $S$ of disjoint open polygons...
Visibility Graph

Given a set $S$ of disjoint open polygons...

...with point set $V(S)$.

**Def.:** Then $G_{\text{vis}}(S) = (V(S), E_{\text{vis}}(S))$ is the visibility graph of $S$ with $E_{\text{vis}}(S) = \{uv \mid u, v \in V(S) \text{ and } u \text{ sees } v\}$ and $w(uv) = |uv|$. 

Where $u$ sees $v : \iff uv \cap \bigcup S = \emptyset$
Visibility Graph

Given a set $S$ of disjoint open polygons...

Def.: Then $G_{\text{vis}}(S) = (V(S), E_{\text{vis}}(S))$ is the visibility graph of $S$ with $E_{\text{vis}}(S) = \{uv \mid u, v \in V(S) \text{ and } u \text{ sees } v\}$ and $w(uv) = |uv|$. Where $u$ sees $v : \iff uv \cap \bigcup S = \emptyset$

Define $S^* = S \cup \{s, t\}$ and $G_{\text{vis}}(S^*)$ analogously.
Visibility Graph

Given a set $S$ of disjoint open polygons...

...with point set $V(S)$.

Def.: Then $G_{vis}(S) = (V(S), E_{vis}(S))$ is the visibility graph of $S$ with $E_{vis}(S) = \{uv \mid u, v \in V(S) \text{ and } u \text{ sees } v\}$ und $w(uv) = |uv|$.

Where $u$ sees $v :\iff \overline{uv} \cap \bigcup S = \emptyset$

Define $S^* = S \cup \{s, t\}$ and $G_{vis}(S^*)$ analogously.

Lemma 1 \(\Rightarrow\) A shortest $st$-path in $\mathbb{R}^2$ avoiding obstacles in $S$ is equivalent to a shortest $st$-path in $G_{vis}(S^*)$. 
Algorithm

ShortestPath($S, s, t$)

**Input:** Obstacles $S$, points $s, t \in \mathbb{R}^2 \setminus \bigcup S$

**Output:** Shortest collision-free $st$-path in $S$

1. $G_{\text{vis}} \leftarrow \text{VisibilityGraph}(S \cup \{s, t\})$
2. foreach $uv \in E_{\text{vis}}$ do $w(uv) \leftarrow |uv|$
3. return Dijkstra($G_{\text{vis}}, w, s, t$)
Algorithm

ShortestPath($S, s, t$)

- **Input:** Obstacles $S$, points $s, t \in \mathbb{R}^2 \setminus \bigcup S$
- **Output:** Shortest collision-free $st$-path in $S$

1. $G_{vis} \leftarrow \text{VisibilityGraph}(S \cup \{s, t\})$
2. foreach $uv \in E_{vis}$ do $w(uv) \leftarrow |uv|$
3. return Dijkstra($G_{vis}, w, s, t$)

$n = |V(S)|, m = |E_{vis}(S)|$

$O(m)$

$O(n \log n + m)$
Algorithm

ShortestPath($S, s, t$)

**Input:** Obstacles $S$, points $s, t \in \mathbb{R}^2 \setminus \bigcup S$

**Output:** Shortest collision-free $st$-path in $S$

1. $G_{vis} \leftarrow \text{VisibilityGraph}(S \cup \{s, t\})$
2. **foreach** $uv \in E_{vis}$ **do** $w(uv) \leftarrow |uv|$
3. **return** $\text{Dijkstra}(G_{vis}, w, s, t)$

$$n = |V(S)|, m = |E_{vis}(S)|$$

$$O(n \log n + m)$$

$$O(n^2 \log n)$$

**Thm 1:** A shortest $st$-path in an area with polygonal obstacles with $n$ edges can be computed in $O(n^2 \log n)$ time.
Computing a Visibility Graph

VisibilityGraph($S$)

**Input:** Set of disjoint polygons $S$

**Output:** Visibility graph $G_{\text{vis}}(S)$

1. $E \leftarrow \emptyset$
2. foreach $v \in V(S)$ do
3.     $W \leftarrow \text{VisibleVertices}(v, S)$
4.     $E \leftarrow E \cup \{vw \mid w \in W\}$
5. return $(V(S), E)$
Computing Visible Nodes

VisibleVertices\((p, S)\)
Computing Visible Nodes

VisibleVertices\((p, S)\)
Computing Visible Nodes

VisibleVertices(p, S)
Computing Visible Nodes

VisibleVertices\((p, S)\)

**Problem:** Given \(p\) and \(S\), find in \(O(n \log n)\) time all nodes that \(p\) sees in \(V(S)\)!
Computing Visible Nodes

VisibleVertices\((p, S)\)

\[ r \leftarrow \{ p + (k, 0) \mid k \in \mathbb{R}_0^+ \} \]
Computing Visible Nodes

VisibleVertices\((p, S)\)

\[ r \leftarrow \{p + (k, 0) \mid k \in \mathbb{R}_0^+\} \]

\[ I \leftarrow \{e \in E(S) \mid e \cap r \neq \emptyset\} \]
Computing Visible Nodes

VisibleVertices\( (p, S) \)

\[ r \leftarrow \{ p + (k, 0) \mid k \in \mathbb{R}^+ \} \]

\[ I \leftarrow \{ e \in E(S) \mid e \cap r \neq \emptyset \} \]

\[ \mathcal{T} \leftarrow \text{balancedBinaryTree}(I) \]
Computing Visible Nodes

VisibleVertices\((p, S)\)

\[
r \leftarrow \{ p + (k, 0) \mid k \in \mathbb{R}_0^+ \}
\]

\[
I \leftarrow \{ e \in E(S) \mid e \cap r \neq \emptyset \}
\]

\[
\mathcal{T} \leftarrow \text{balancedBinaryTree}(I)
\]

\[
w_1, \ldots, w_n \leftarrow \text{sort } V(S) \text{ in cyclic order around } p
\]
Computing Visible Nodes

VisibleVertices\((p, S)\)

\[
r \leftarrow \{p + (k, 0) \mid k \in \mathbb{R}^+_0\}
\]

\[
I \leftarrow \{e \in E(S) \mid e \cap r \neq \emptyset\}
\]

\[
\mathcal{T} \leftarrow \text{balancedBinaryTree}(I)
\]

\[
w_1, \ldots, w_n \leftarrow \text{sort } V(S) \text{ in cyclic order around } p
\]

\[v \prec v' :\Leftrightarrow \]

\[
\angle v < \angle v' \text{ or } \]

\[
(\angle v = \angle v' \text{ and } |pv| < |pv'|)
\]
Computing Visible Nodes

VisibleVertices\((p, S)\)

\[ r \leftarrow \{ p + (k, 0) \mid k \in \mathbb{R}_0^+ \} \]

\[ I \leftarrow \{ e \in E(S) \mid e \cap r \neq \emptyset \} \]

\[ \mathcal{T} \leftarrow \text{balancedBinaryTree}(I) \]

\( w_1, \ldots, w_n \leftarrow \text{sort } V(S) \text{ in cyclic order around } p \)

\[ v \prec v' : \iff \]

\[ \angle v < \angle v' \text{ or } \]

\[ (\angle v = \angle v' \text{ and } |pv| < |pv'|) \]
Computing Visible Nodes

\[ \text{VisibleVertices}(p, S) \]

\[ r \leftarrow \{ p + (k, 0) \mid k \in \mathbb{R}^+_0 \} \]

\[ I \leftarrow \{ e \in E(S) \mid e \cap r \neq \emptyset \} \]

\[ \mathcal{T} \leftarrow \text{balancedBinaryTree}(I) \]

\[ w_1, \ldots, w_n \leftarrow \text{sort } V(S) \text{ in cyclic order around } p \]

\[ v \prec v' \iff \angle v < \angle v' \text{ or } (\angle v = \angle v' \text{ and } |pv| < |pv'|) \]

\textit{Sweep method with rotation}
Computing Visible Nodes

\[ \text{VisibleVertices}(p, S) \]

\[ r \leftarrow \{ p + (k, 0) \mid k \in \mathbb{R}_{0}^{+} \} \]

\[ I \leftarrow \{ e \in E(S) \mid e \cap r \neq \emptyset \} \]

\[ T \leftarrow \text{balancedBinaryTree}(I) \]

\[ w_1, \ldots, w_n \leftarrow \text{sort } V(S) \text{ in cyclic order around } p \]
Computing Visible Nodes

\[ \text{VisibleVertices}(p, S) \]

\[ r \leftarrow \{ p + (k, 0) \mid k \in \mathbb{R}_0^+ \} \]

\[ I \leftarrow \{ e \in E(S) \mid e \cap r \neq \emptyset \} \]

\[ \mathcal{T} \leftarrow \text{balancedBinaryTree}(I) \]

\[ w_1, \ldots, w_n \leftarrow \text{sort } V(S) \text{ in cyclic order around } p \]

\[ W \leftarrow \emptyset \]

\[ \text{for } i = 1 \text{ to } n \text{ do} \]

\[ \quad \text{if } \text{Visible}(p, w_i) \text{ then} \]

\[ \quad \quad W \leftarrow W \cup \{ w_i \} \]

\[ \quad \text{Add to } \mathcal{T} \text{ edges incident to } w_i: \text{CW from } \overrightarrow{pw_i}^+ \]

\[ \quad \text{Remove from } \mathcal{T} \text{ edges incident to } w_i: \text{CCW from } \overrightarrow{pw_i}^- \]

\[ \text{return } W \]
Computing Visible Nodes

VisibleVertices\( (p, S) \)
\[ r \leftarrow \{ p + (k, 0) \mid k \in \mathbb{R}_0^+ \} \]
\[ I \leftarrow \{ e \in E(S) \mid e \cap r \neq \emptyset \} \]
\[ T \leftarrow \text{balancedBinaryTree}(I) \]

\[ w_1, \ldots, w_n \leftarrow \text{sort } V(S) \text{ in cyclic order around } p \]
\[ W \leftarrow \emptyset \]

\[ \text{for } i = 1 \text{ to } n \text{ do} \]
\[ \quad \text{if Visible}(p, w_i) \text{ then} \]
\[ \quad \quad W \leftarrow W \cup \{ w_i \} \]
\[ \quad \text{Add to } T \text{ edges incident to } w_i: \text{CW from } \overrightarrow{pw_i}^+ \]
\[ \quad \text{Remove from } T \text{ edges incident to } w_i: \text{CCW from } \overrightarrow{pw_i}^- \]

\[ \text{return } W \]
Computing Visible Nodes

`VisibleVertices(p, S)`

1. Set `r ← \{p + (k, 0) \mid k ∈ \mathbb{R}_0^+\}`
2. Set `I ← \{e ∈ E(S) \mid e \cap r \neq \emptyset\}`
3. Set `T ← balancedBinaryTree(I)`
4. Set `w_1, \ldots, w_n ← sort V(S)` in cyclic order around `p`
5. Set `W ← \emptyset`
6. For `i = 1` to `n` do
   - If `Visible(p, w_i)` then
     - Set `W ← W ∪ \{w_i\}`
   - Add to `T` edges incident to `w_i`: CW from \(\overrightarrow{pw_i}\)
   - Remove from `T` edges incident to `w_i`: CCW from \(\overrightarrow{pw_i}\)
8. Return `W`
Visibility Case Analysis

Visible\((p, w_i)\)

\[
\text{if } \overrightarrow{pw_i} \text{ intersects polygon of } w_i \text{ then return false}
\]
Visibility Case Analysis

\textbf{Visible}(p, w_i)

\begin{align*}
\text{if } \overrightarrow{pw_i} \text{ intersects polygon of } w_i & \text{ then return false} \\
\text{if } i = 1 \text{ or } w_{i-1} \not\in \overrightarrow{pw_i} & \text{ then} \\
& \quad \left\{ \begin{array}{l}
\text{e } \leftarrow \text{ edge of leftmost leaf of } T \\
\text{if } e \neq \text{nil and } \overrightarrow{pw_i} \cap e \neq \emptyset & \text{ then return false} \\
\text{else return true}
\end{array} \right.
\end{align*}
Visibility Case Analysis

Visible\((p, w_i)\)

\[
\text{if } \overline{pw_i} \text{ intersects polygon of } w_i \text{ then return false}
\]
\[
\text{if } i = 1 \text{ or } w_{i-1} \not\in \overline{pw_i} \text{ then}
\]
\[
\quad e \leftarrow \text{edge of leftmost leaf of } \mathcal{T}
\]
\[
\quad \text{if } e \neq \text{nil and } \overline{pw_i} \cap e \neq \emptyset \text{ then return false}
\]
\[
\quad \text{else return true}
\]
\[
\text{else}
\]
\[
\quad \text{if } w_{i-1} \text{ is not visible then return false}
\]
\[
\quad \text{else}
\]
\[
\quad \quad e \leftarrow \text{find edge in } \mathcal{T}, \text{ that } \overline{w_{i-1}w_i} \text{ cuts; if } e \neq \text{nil then return false}
\]
\[
\quad \quad \text{else return true}
\]
Summary

**Thm 1:** A shortest $st$-path in an area with polygonal obstacles with $n$ edges can be computed in $O(n^2 \log n)$ time.
Summary

**Thm 1:** A shortest \( st \)-path in an area with polygonal obstacles with \( n \) edges can be computed in \( O(n^2 \log n) \) time.

**Proof:**
- Correctness follows directly from Lemma 1.
Summary

Thm 1: A shortest $st$-path in an area with polygonal obstacles with $n$ edges can be computed in $O(n^2 \log n)$ time.

Proof:
- Correctness follows directly from Lemma 1.
- Running time:
  - $\text{VisibleVertices}$ takes $O(n \log n)$ time per vertex – $n$ calls to $\text{VisibleVertices}$
Summary

**Thm 1**: A shortest \( st \)-path in an area with polygonal obstacles with \( n \) edges can be computed in \( O(n^2 \log n) \) time.

**Proof:**

- Correctness follows directly from Lemma 1.
- Running time:
  - \( \text{VisibleVertices} \) takes \( O(n \log n) \) time per vertex – \( n \) calls to \( \text{VisibleVertices} \).
Discussion

Robots are not single points...
Discussion

Robots are not single points...

For robots modelled by a convex polygon that cannot rotate, we can resize (grow) the polygons representing the obstacles (→ Minkowski Sums, Ch. 13 in [BCKO08]).
Discussion

Robots are not single points...

For robots modelled by a convex polygon that cannot rotate, we can resize (grow) the polygons representing the obstacles (→ Minkowski Sums, Ch. 13 in [BCKO08]).
Discussion

Robots are not single points...

For robots modelled by a convex polygon that cannot rotate, we can resize (grow) the polygons representing the obstacles (→ Minkowski Sums, Ch. 13 in [BCKO08]).
Discussion

Robots are not single points...

For robots modelled by a convex polygon that cannot rotate, we can resize (grow) the polygons representing the obstacles (→ Minkowski Sums, Ch. 13 in [BCKO08]).
Discussion

Robots are not single points...

For robots modelled by a convex polygon that cannot rotate, we can resize (grow) the polygons representing the obstacles (→ Minkowski Sums, Ch. 13 in [BCKO08]).
Robots are not single points...

For robots modelled by a convex polygon that cannot rotate, we can resize (grow) the polygons representing the obstacles (→ Minkowski Sums, Ch. 13 in [BCKO08]).
Discussion

Robots are not single points...

For robots modelled by a convex polygon that cannot rotate, we can resize (grow) the polygons representing the obstacles (→ Minkowski Sums, Ch. 13 in [BCKO08]).
Discussion

Robots are not single points...

For robots modelled by a convex polygon that cannot rotate, we can resize (grow) the polygons representing the obstacles (→ Minkowski Sums, Ch. 13 in [BCKO08]).
Discussion

Robots are not single points...

For robots modelled by a convex polygon that cannot rotate, we can resize (grow) the polygons representing the obstacles (→ Minkowski Sums, Ch. 13 in [BCKO08]).
Discussion

Robots are not single points...

For robots modelled by a convex polygon that cannot rotate, we can resize (grow) the polygons representing the obstacles (→ Minkowski Sums, Ch. 13 in [BCKO08]).

Can we compute faster than $O(n^2 \log n)$?
Discussion

Robots are not single points...

For robots modelled by a convex polygon that cannot rotate, we can resize (grow) the polygons representing the obstacles (→ Minkowski Sums, Ch. 13 in [BCKO08]).

Can we compute faster than $O(n^2 \log n)$?

Yes, by use duality and a simultaneous rotation sweep for all points in the dual. Computing the arrangement, is also in $O(n^2)$. Even though $G_{vis}$ can have $\Omega(n^2)$ edges, the visibility graph can be constructed even faster with an output sensitive $O(n \log n + m)$-time algorithm.

[Ghosh, Mount 1987]
Discussion

Robots are not single points...

For robots modelled by a convex polygon that cannot rotate, we can resize (grow) the polygons representing the obstacles (→ Minkowski Sums, Ch. 13 in [BCKO08]).

Can we compute faster than $O(n^2 \log n)$?

Yes, by use duality and a simultaneous rotation sweep for all points in the dual. Computing the arrangement, is also in $O(n^2)$. Even though $G_{vis}$ can have $\Omega(n^2)$ edges, the visibility graph can be constructed even faster with an output sensitive $O(n \log n + m)$-time algorithm.

[ Ghosh, Mount 1987 ]

If you search only for one shortest Euclidean $st$-path, there is an algorithm with optimal $O(n \log n)$ time.

[Hershberger, Suri 1999]