Visibility Graphs

Outline:

I. Shortest path

II. Construction of the visibility graph

III. Finding visible vertices

IV. Shortest path for a translational robot

I. Configuration Space C

robot $\longmapsto$ *point* $\in C \subseteq \mathbb{R}^d$

I. Configuration Space C

robot $\longmapsto$ *point* $\in C \subseteq \mathbb{R}^d$ ← # degrees of freedom of the robot

obstacle $\longmapsto$ *C-obstacle* $\subseteq \mathbb{R}^d$

I. Configuration Space C

robot $\longmapsto$ *point* $\in C \subseteq \mathbb{R}^d$

degrees of freedom of the robot

obstacle $\longmapsto$ *C-obstacle* $\subseteq \mathbb{R}^d$

set of configurations at which the robot collides with the obstacle

I. Configuration Space C

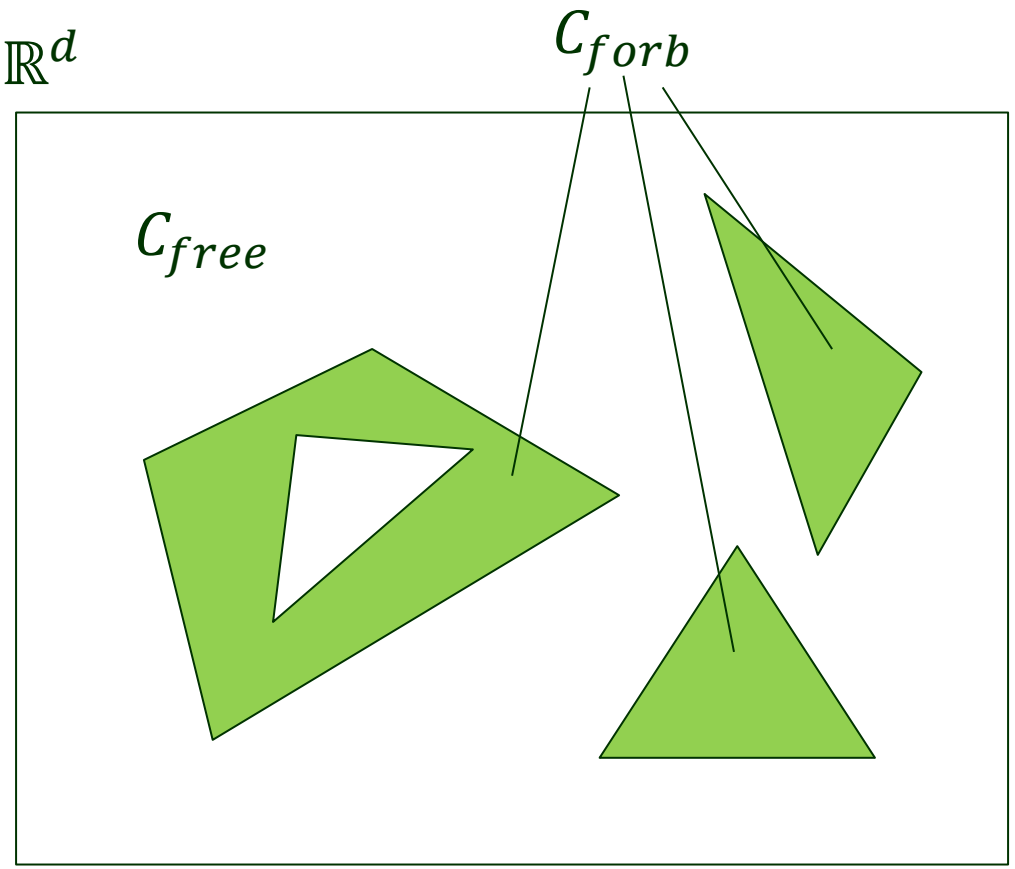
robot $\longmapsto$ *point* $\in C \subseteq \mathbb{R}^d$ ← # degrees of freedom of the robot

obstacle $\longmapsto$ *C-obstacle* $\subseteq \mathbb{R}^d$

↑ set of configurations at which the robot collides with the obstacle

C_{forb}: union of all C-obstacles

C_{free}: set of free configurations



I. Configuration Space C

robot $\longmapsto$ *point* $\in C \subseteq \mathbb{R}^d$ ← # degrees of freedom of the robot

obstacle $\longmapsto$ *C-obstacle* $\subseteq \mathbb{R}^d$

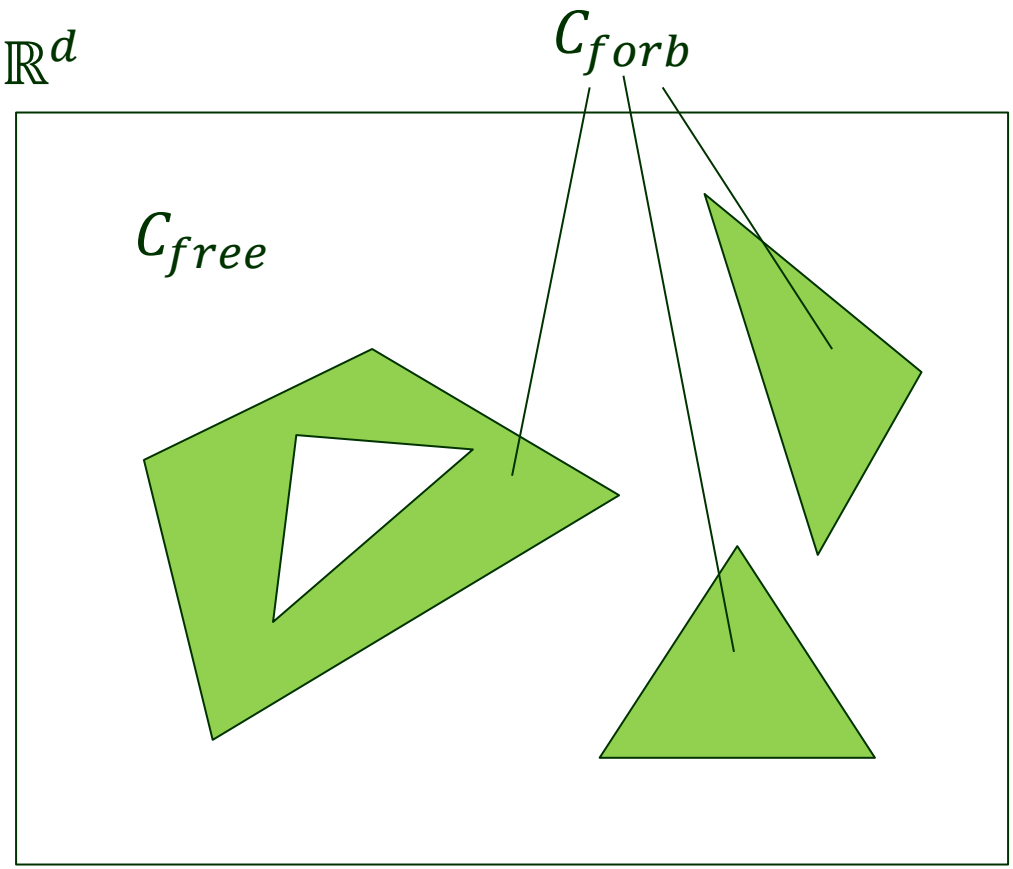
↑ set of configurations at which the robot collides with the obstacle

C_{forb}: union of all C-obstacles

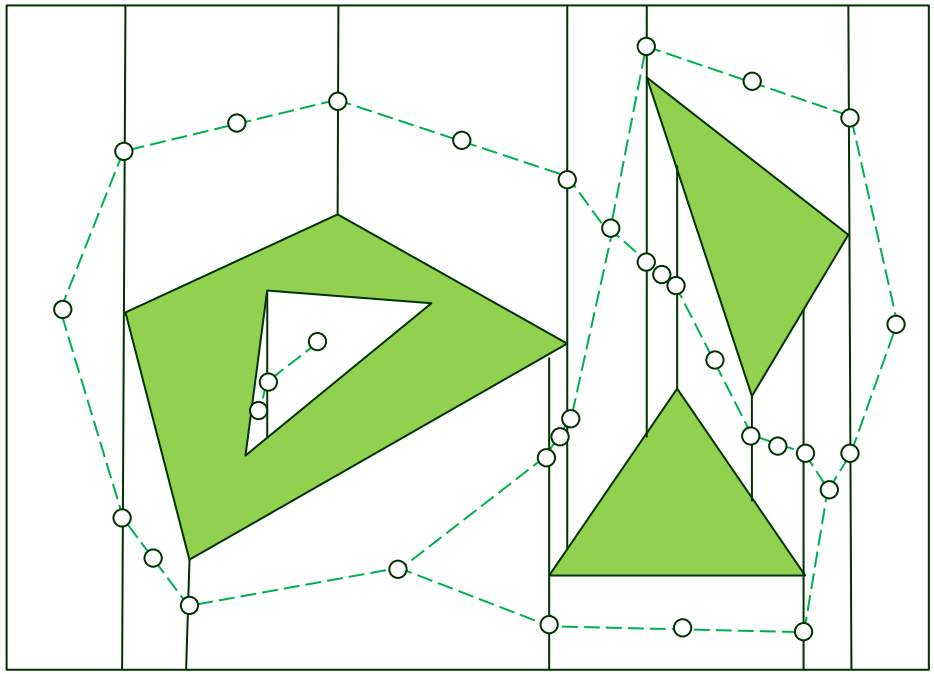
C_{free}: set of free configurations

$$C_{forb} \cup C_{free} = C$$

$$C_{forb} \cap C_{free} = \emptyset$$



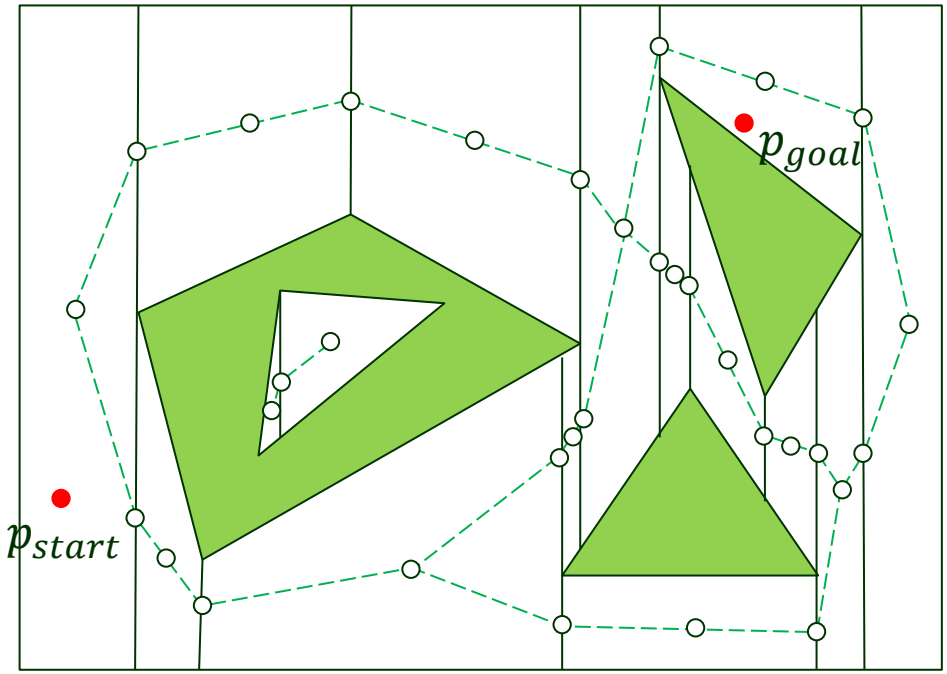
Roadmap Construction



Use the trapezoidal map of C_{free}.

$O(n \log n)$ expected time

Motion Planning for a Point Robot

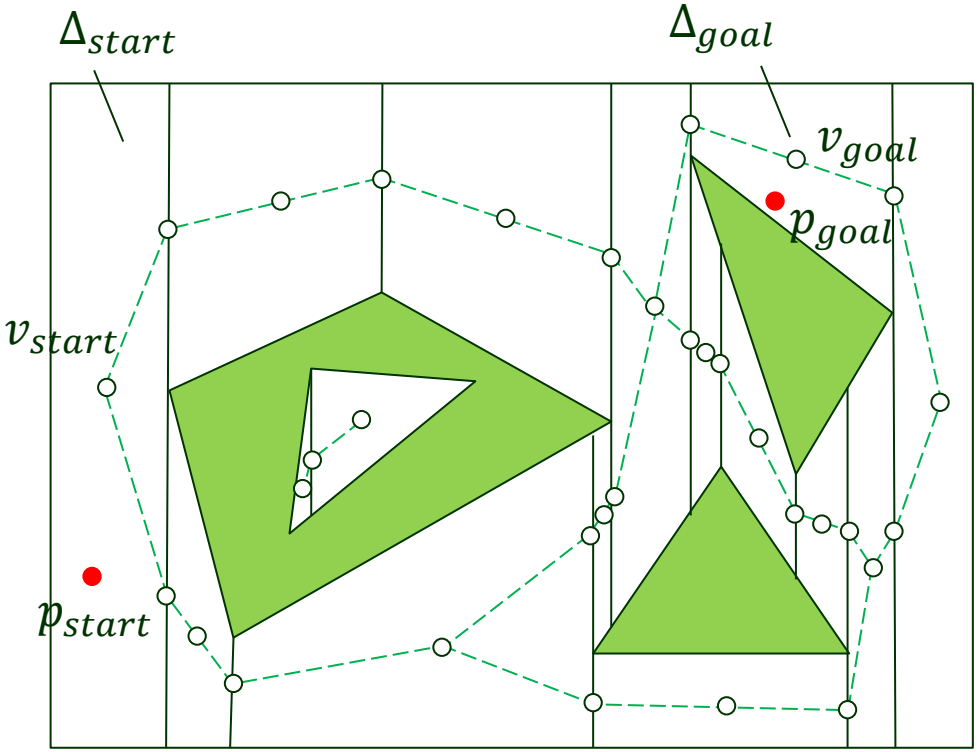


Input: starting position p_{start}
goal position p_{goal}

Breadth-first search over the roadmap.

$O(n)$

Motion Planning for a Point Robot

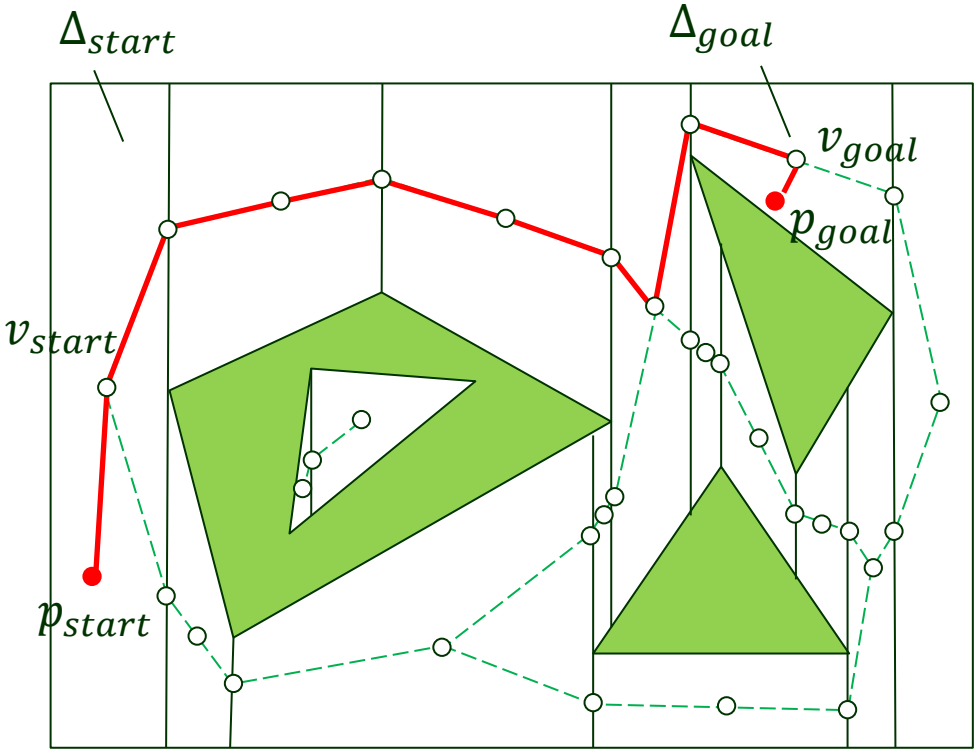


Input: starting position p_{start}
goal position p_{goal}

Breadth-first search over the roadmap.

$O(n)$

Motion Planning for a Point Robot

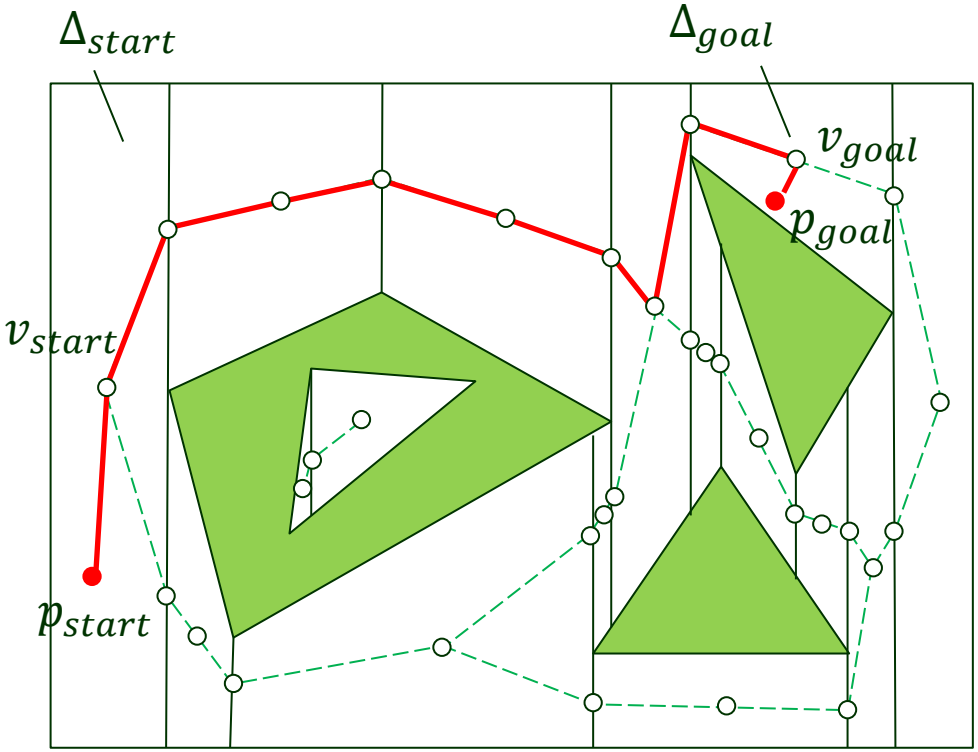


Input: starting position p_{start}
goal position p_{goal}

Breadth-first search over the roadmap.

$O(n)$

Motion Planning for a Point Robot



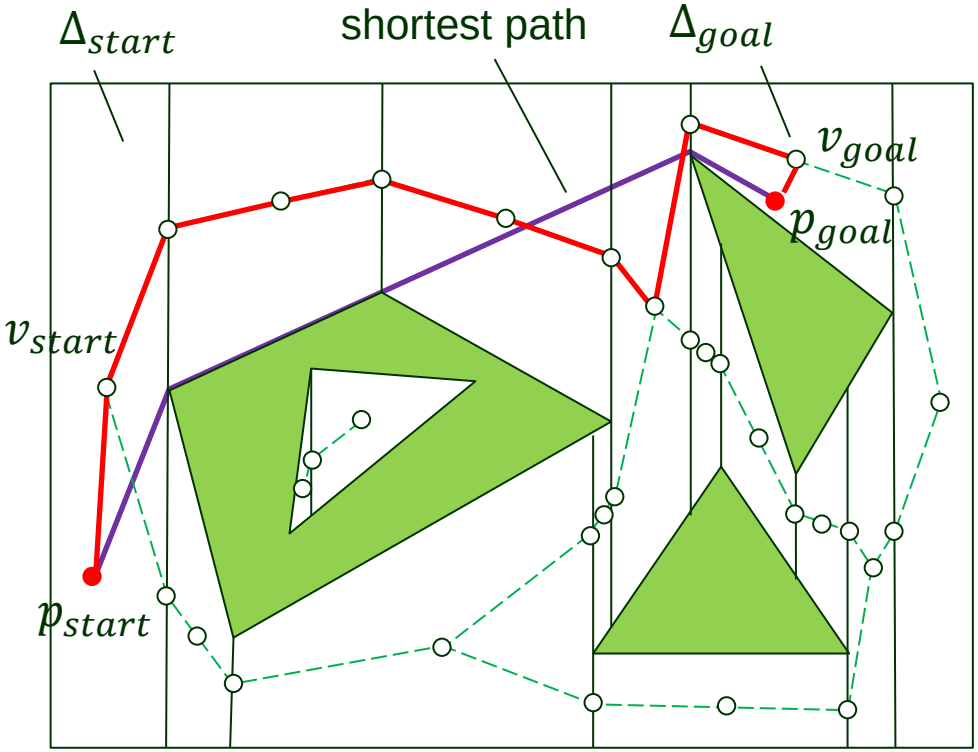
Input: starting position p_{start}
goal position p_{goal}

Breadth-first search over the roadmap.

$O(n)$

♣ The path has smallest number of arcs from the roadmap but is often not the shortest.

Motion Planning for a Point Robot



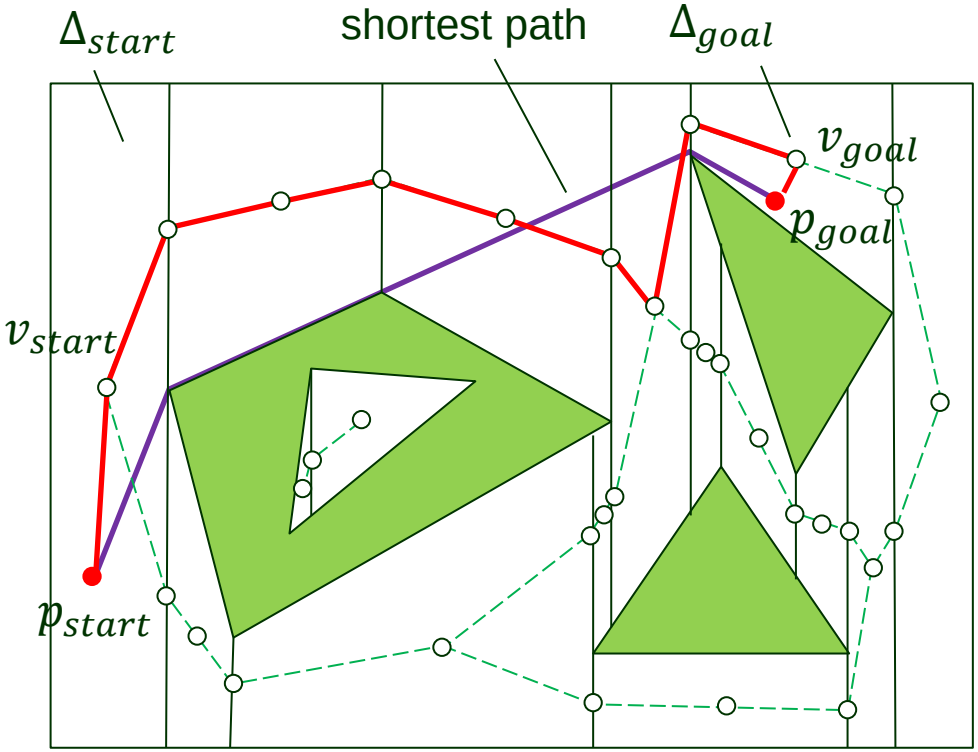
Input: starting position p_{start}
goal position p_{goal}

Breadth-first search over the roadmap.

$O(n)$

♣ The path has smallest number of arcs from the roadmap but is often not the shortest.

Motion Planning for a Point Robot



Input: starting position p_{start}
goal position p_{goal}

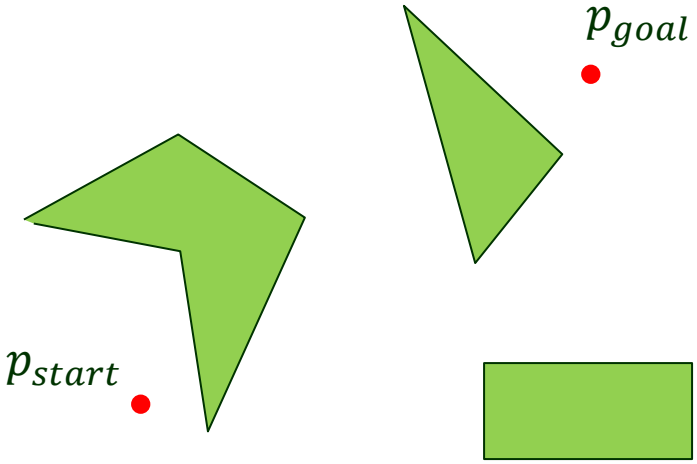
Breadth-first search over the roadmap.

$O(n)$

- The path has smallest number of arcs from the roadmap but is often not the shortest.
- Weight each arc by its length and apply Dijkstra's algorithm.

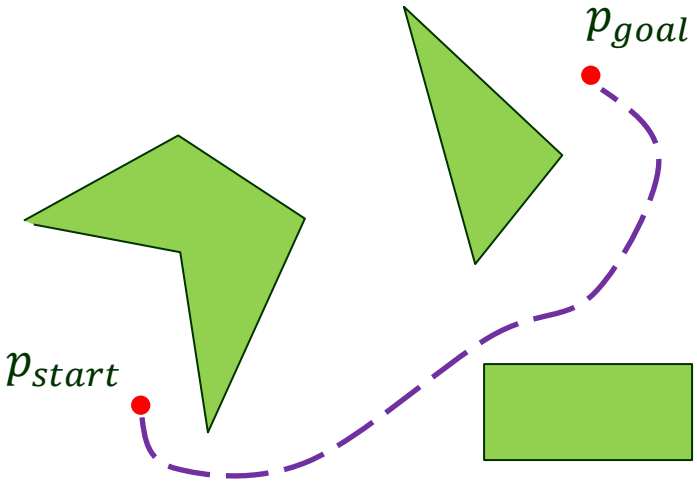
  Still often not the shortest path length.

Shortest Path



S: a set of disjoint polygonal obstacles.

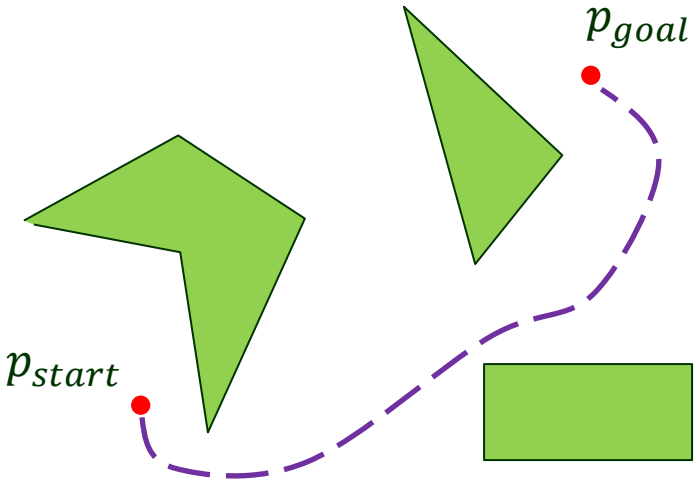
Shortest Path



S: a set of disjoint polygonal obstacles.

- Think of the path as a rubber band with endpoints fixed at p_{start} and p_{goal}.
- It has been stretched to avoid all the obstacles.

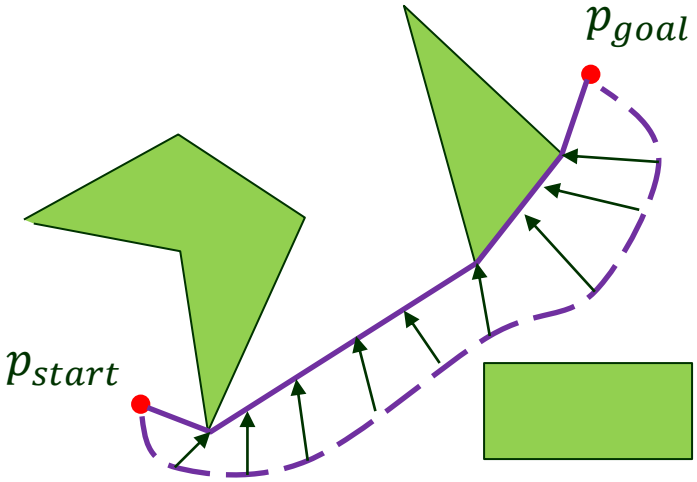
Shortest Path



S: a set of disjoint polygonal obstacles.

- Think of the path as a rubber band with endpoints fixed at p_{start} and p_{goal}.
- It has been stretched to avoid all the obstacles.
- When the band is released, it will contract as much as possible:

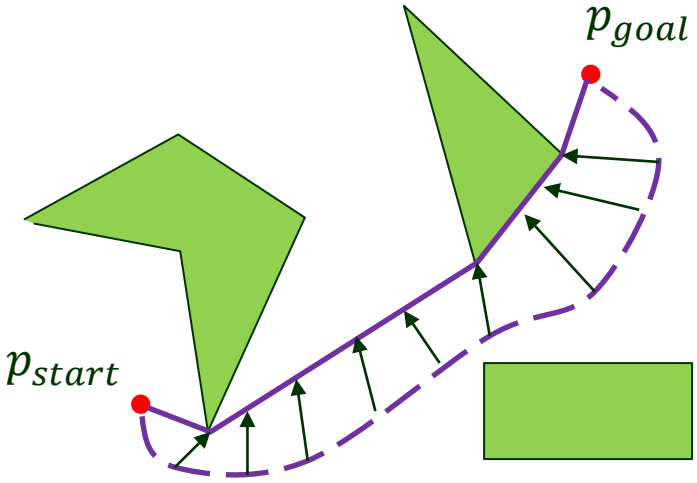
Shortest Path



S: a set of disjoint polygonal obstacles.

- Think of the path as a rubber band with endpoints fixed at p_{start} and p_{goal}.
- It has been stretched to avoid all the obstacles.
- When the band is released, it will contract as much as possible:

Shortest Path



S: a set of disjoint polygonal obstacles.

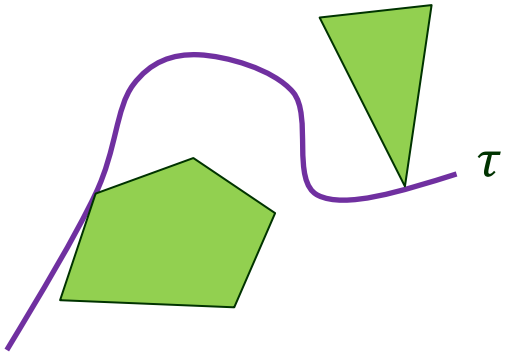
- Think of the path as a rubber band with endpoints fixed at p_{start} and p_{goal}.
- It has been stretched to avoid all the obstacles.
- When the band is released, it will contract as much as possible:
  - following some obstacle boundaries
  - moving along straight segments in the free space.

Geometry of a Shortest Path

Lemma 1 The shortest path is a polygonal path whose inner vertices are also vertices of some obstacles in S.

Geometry of a Shortest Path

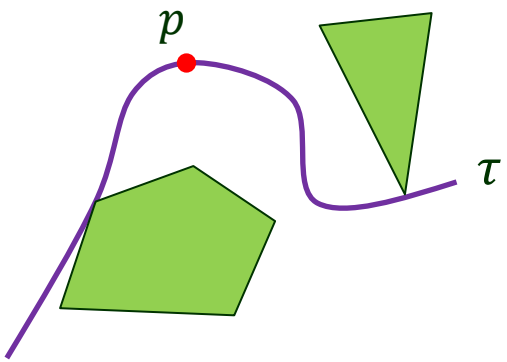
Lemma 1 The shortest path is a polygonal path whose inner vertices are also vertices of some obstacles in S.



Proof Suppose a shortest path τ is not polygonal.

Geometry of a Shortest Path

Lemma 1 The shortest path is a polygonal path whose inner vertices are also vertices of some obstacles in S.



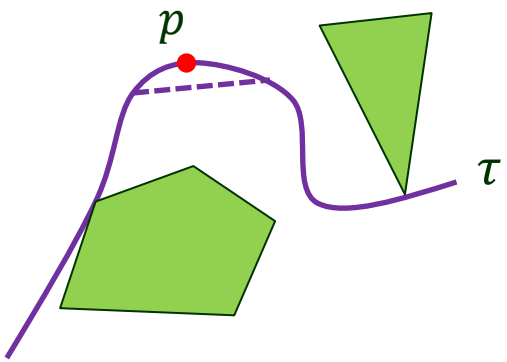
Proof Suppose a shortest path τ is not polygonal.

⇩

There exist $p \in \tau$ in the free space such that τ locally is not a line segment containing p.

Geometry of a Shortest Path

Lemma 1 The shortest path is a polygonal path whose inner vertices are also vertices of some obstacles in S.



Proof Suppose a shortest path τ is not polygonal.

⇩

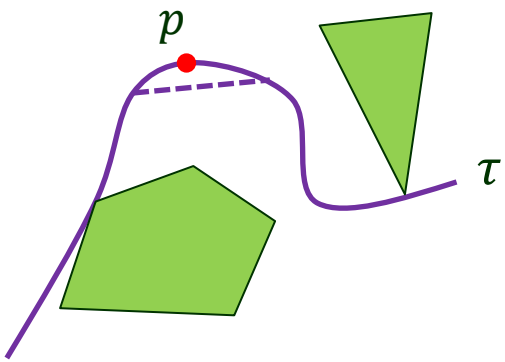
There exist $p \in \tau$ in the free space such that τ locally is not a line segment containing p.

⇩

But τ can be locally shortened around p. Contradiction.

Geometry of a Shortest Path

Lemma 1 The shortest path is a polygonal path whose inner vertices are also vertices of some obstacles in S.



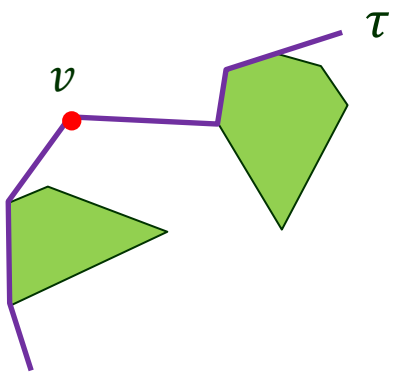
Proof Suppose a shortest path τ is not polygonal.

⇩

There exist $p \in \tau$ in the free space such that τ locally is not a line segment containing p.

⇩

But τ can be locally shortened around p. Contradiction.

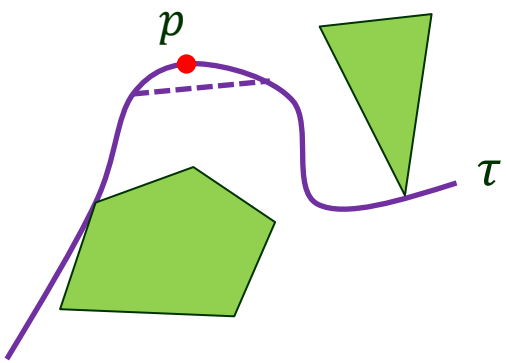


A vertex v of τ with $v \neq p_{start}, p_{goal}$ cannot lie in the interior of

- the free space, or

Geometry of a Shortest Path

Lemma 1 The shortest path is a polygonal path whose inner vertices are also vertices of some obstacles in S.



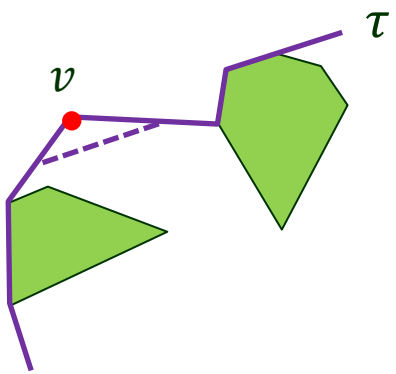
Proof Suppose a shortest path τ is not polygonal.

⇩

There exist $p \in \tau$ in the free space such that τ locally is not a line segment containing p.

⇩

But τ can be locally shortened around p. Contradiction.

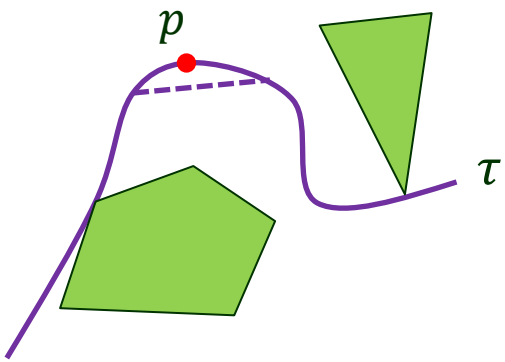


A vertex v of τ with $v \neq p_{start}, p_{goal}$ cannot lie in the interior of

- the free space, or

Geometry of a Shortest Path

Lemma 1 The shortest path is a polygonal path whose inner vertices are also vertices of some obstacles in S.



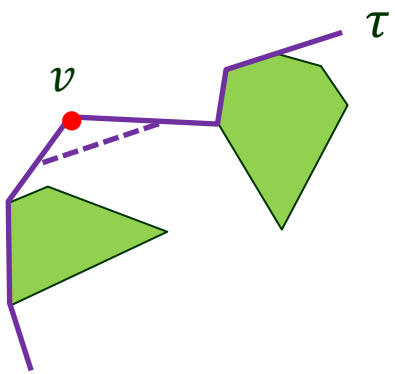
Proof Suppose a shortest path τ is not polygonal.

⇩

There exist $p \in \tau$ in the free space such that τ locally is not a line segment containing p.

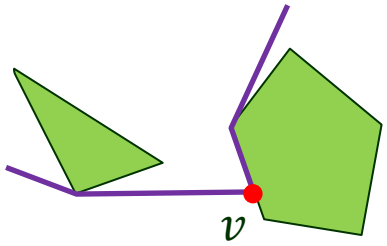
⇩

But τ can be locally shortened around p. Contradiction.



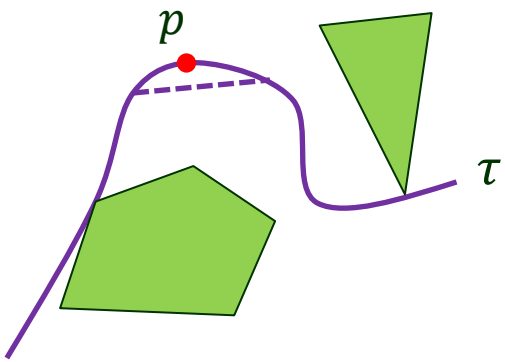
A vertex v of τ with $v \neq p_{start}, p_{goal}$ cannot lie in the interior of

- the free space, or
- an obstacle edge



Geometry of a Shortest Path

Lemma 1 The shortest path is a polygonal path whose inner vertices are also vertices of some obstacles in S.



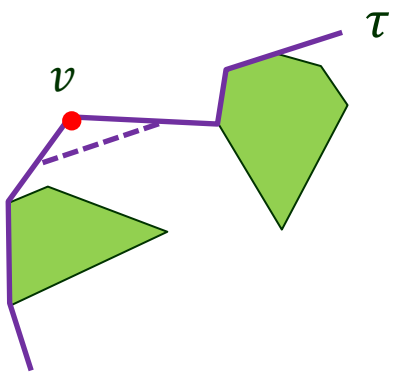
Proof Suppose a shortest path τ is not polygonal.

⇩

There exist $p \in \tau$ in the free space such that τ locally is not a line segment containing p.

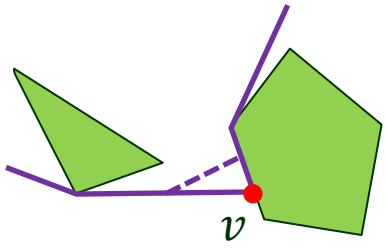
⇩

But τ can be locally shortened around p. Contradiction.



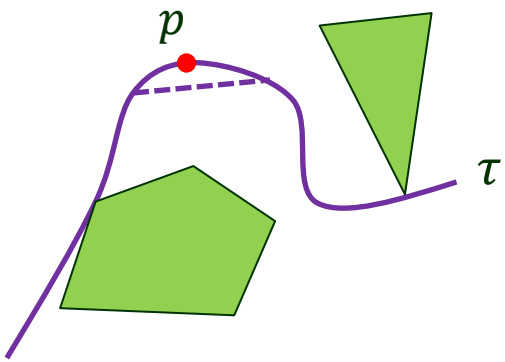
A vertex v of τ with $v \neq p_{start}, p_{goal}$ cannot lie in the interior of

- the free space, or
- an obstacle edge



Geometry of a Shortest Path

Lemma 1 The shortest path is a polygonal path whose inner vertices are also vertices of some obstacles in S.



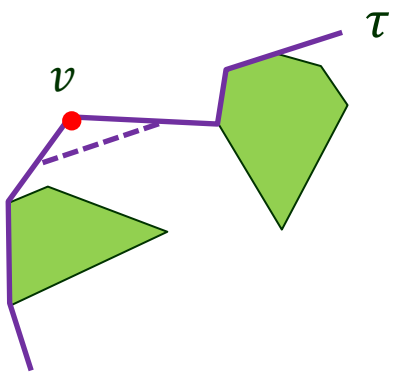
Proof Suppose a shortest path τ is not polygonal.

⇩

There exist $p \in \tau$ in the free space such that τ locally is not a line segment containing p.

⇩

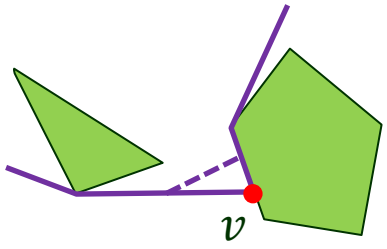
But τ can be locally shortened around p. Contradiction.



A vertex v of τ with $v \neq p_{start}, p_{goal}$ cannot lie in the interior of

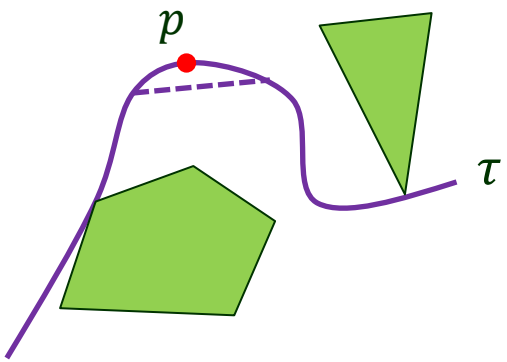
- the free space, or
- an obstacle edge

Otherwise, τ can be locally shortened.



Geometry of a Shortest Path

Lemma 1 The shortest path is a polygonal path whose inner vertices are also vertices of some obstacles in S.



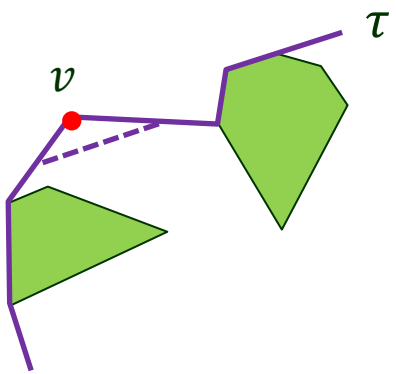
Proof Suppose a shortest path τ is not polygonal.

⇩

There exist $p \in \tau$ in the free space such that τ locally is not a line segment containing p.

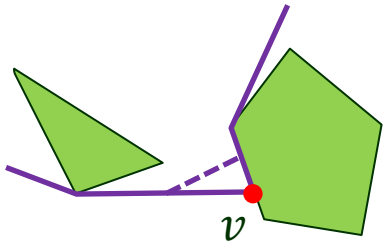
⇩

But τ can be locally shortened around p. Contradiction.



A vertex v of τ with $v \neq p_{start}, p_{goal}$ cannot lie in the interior of

- the free space, or
- an obstacle edge

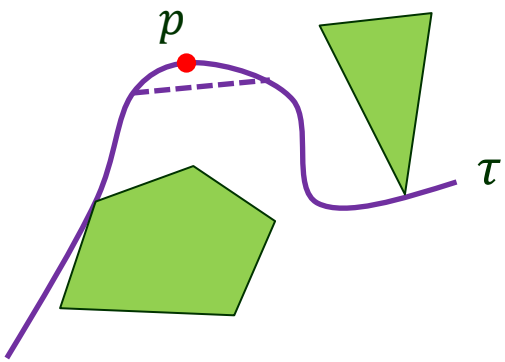


Otherwise, τ can be locally shortened.

Every inner vertex τ is an obstacle vertex.

Geometry of a Shortest Path

Lemma 1 The shortest path is a polygonal path whose inner vertices are also vertices of some obstacles in S.



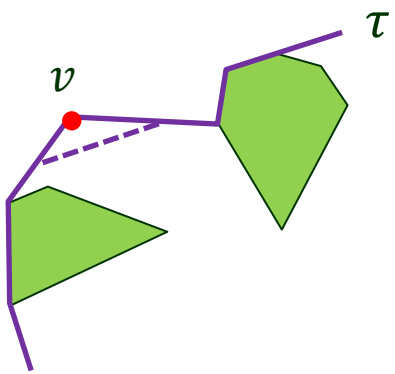
Proof Suppose a shortest path τ is not polygonal.

⇩

There exist $p \in \tau$ in the free space such that τ locally is not a line segment containing p.

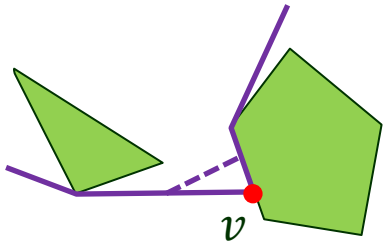
⇩

But τ can be locally shortened around p. Contradiction.



A vertex v of τ with $v \neq p_{start}, p_{goal}$ cannot lie in the interior of

- the free space, or
- an obstacle edge



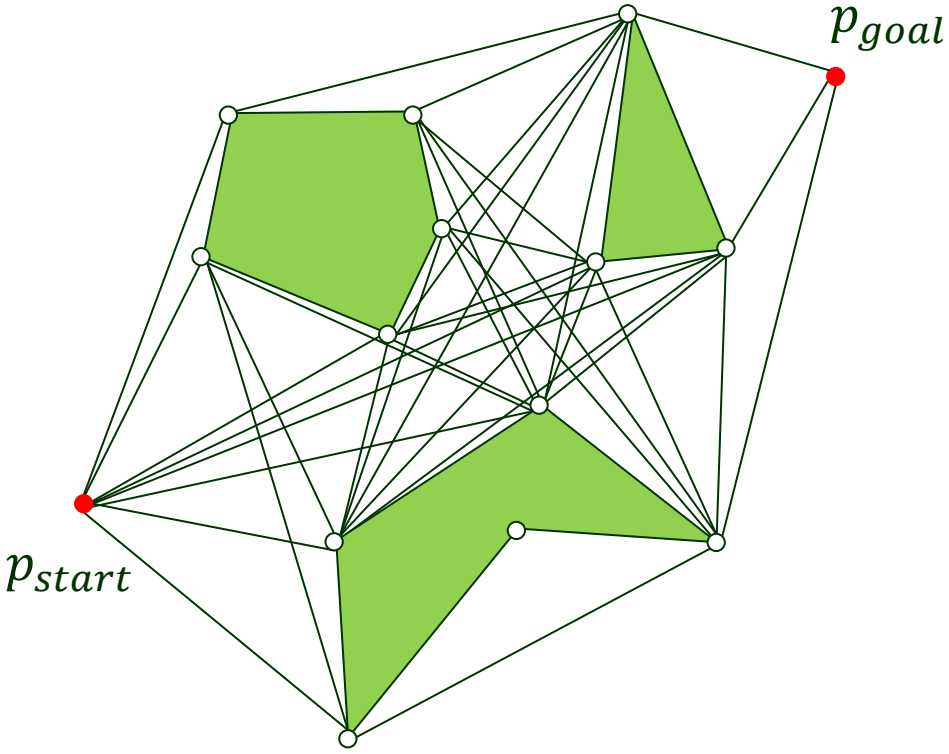
Otherwise, τ can be locally shortened.

Every inner vertex τ is an obstacle vertex.

□

II. Visibility Graph

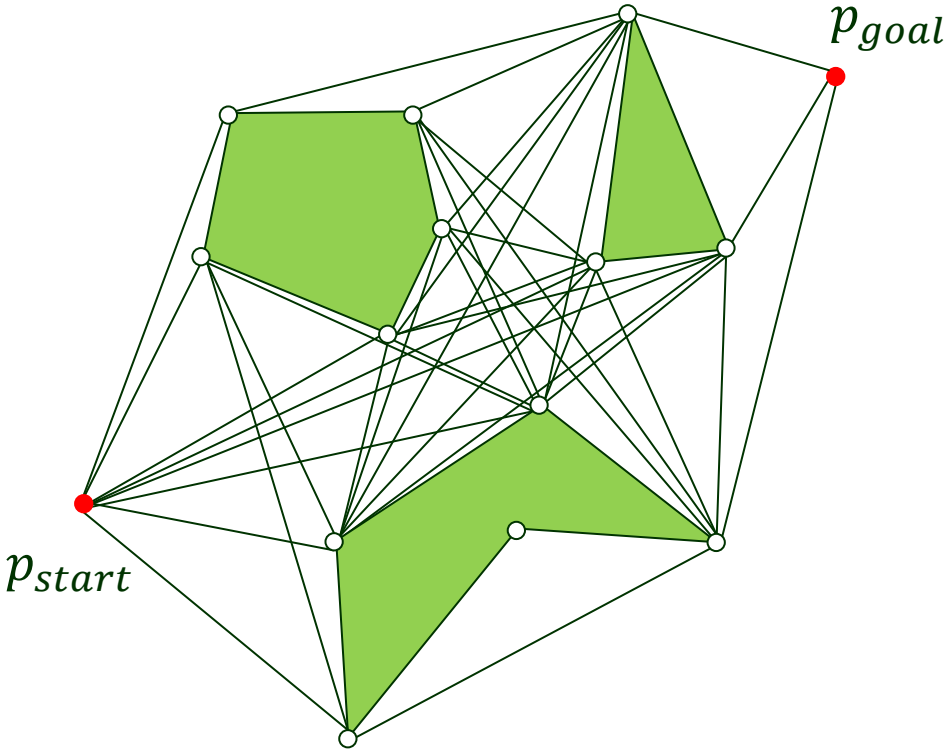
A road map for finding the shortest path.



$$\mathcal{G}_{vis}(S \cup \{p_{start}, p_{goal}\})$$

II. Visibility Graph

A road map for finding the shortest path.

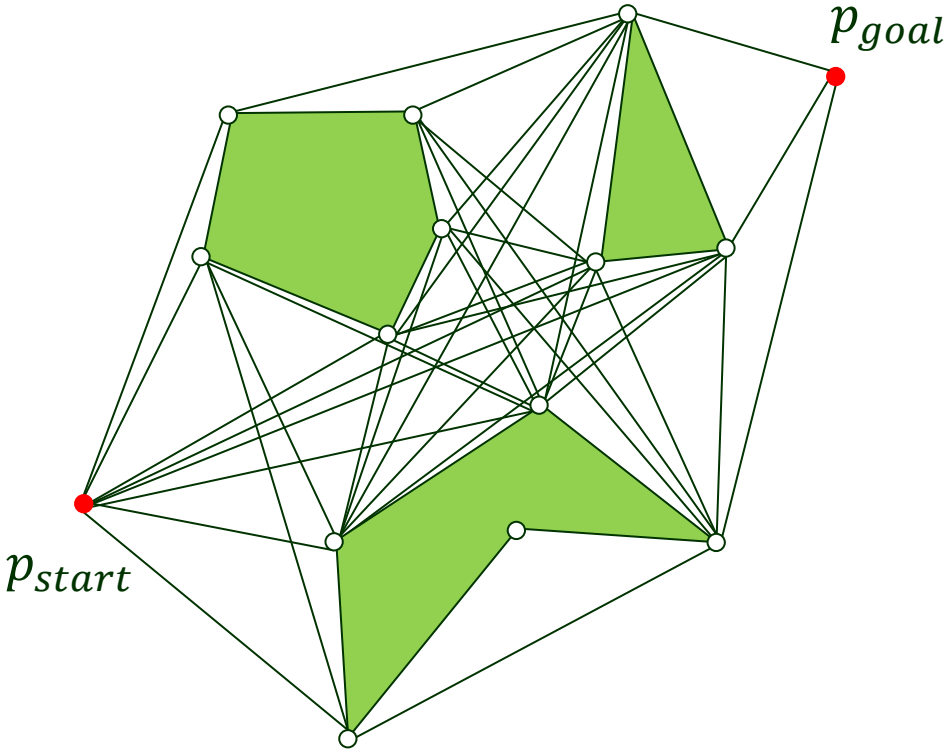


$\mathcal{G}_{vis}(S \cup \{p_{start}, p_{goal}\})$

- Vertex set $S \cup \{p_{start}, p_{goal}\}$

II. Visibility Graph

A road map for finding the shortest path.

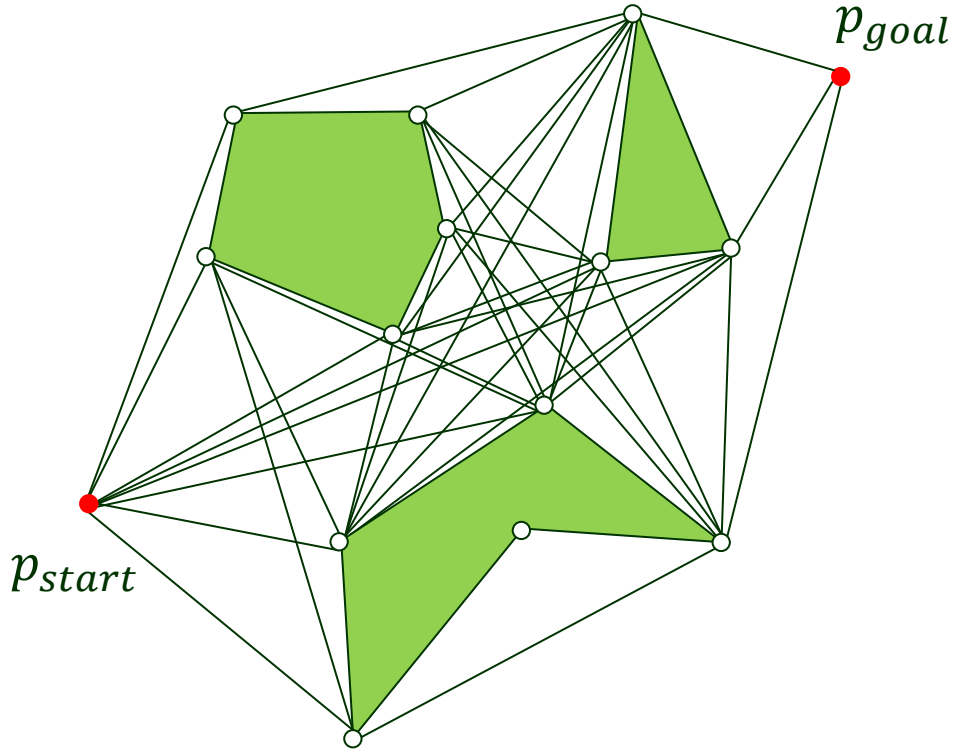


$\mathcal{G}_{vis}(S \cup \{p_{start}, p_{goal}\})$

- Vertex set $S \cup \{p_{start}, p_{goal}\}$
- Edge (v, w) exists if v and w can see each other – it's called a *visibility edge*.

II. Visibility Graph

A road map for finding the shortest path.

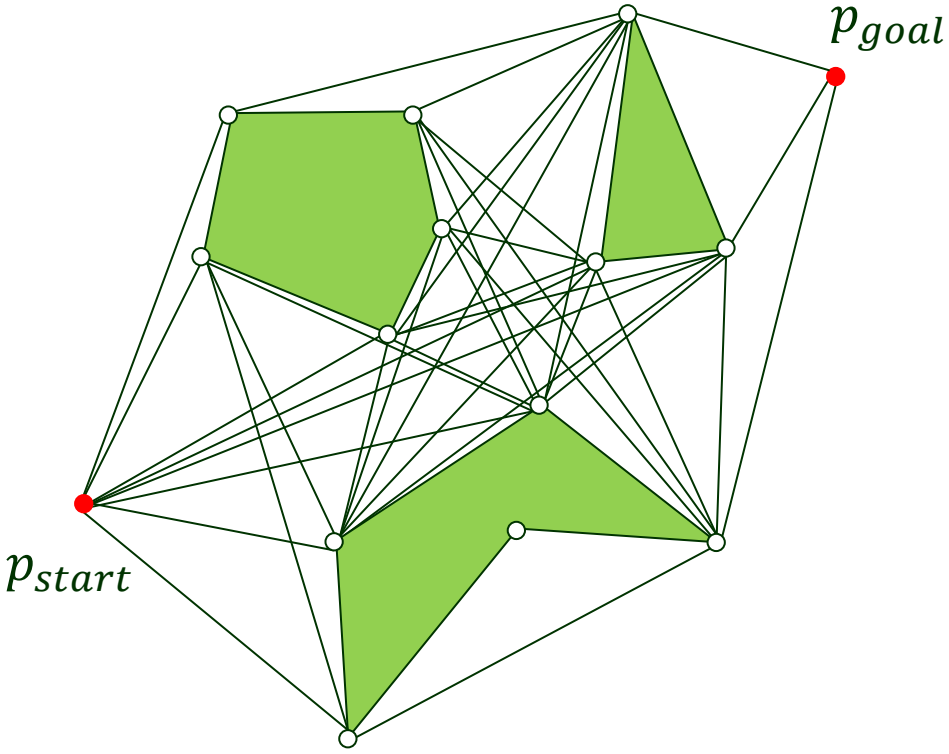


$\mathcal{G}_{vis}(S \cup \{p_{start}, p_{goal}\})$

- Vertex set $S \cup \{p_{start}, p_{goal}\}$
- Edge (v, w) exists if v and w can see each other – it's called a *visibility edge*.
- Every obstacle edge is in $\mathcal{G}_{vis}$ because its endpoints see each other.

II. Visibility Graph

A road map for finding the shortest path.



$\mathcal{G}_{vis}(S \cup \{p_{start}, p_{goal}\})$

- Vertex set $S \cup \{p_{start}, p_{goal}\}$
- Edge (v, w) exists if v and w can see each other – it's called a *visibility edge*.
- Every obstacle edge is in $\mathcal{G}_{vis}$ because its endpoints see each other.

Corollary 2 The shortest path $p_{start} \rightsquigarrow p_{goal}$ consists of edges in $\mathcal{G}_{vis}$.

Shortest Path Algorithm

1. Construct the visibility graph.

Shortest Path Algorithm

1. Construct the visibility graph.

$O(n^2 \log n)$ (algorithm to be described)

Shortest Path Algorithm

1. Construct the visibility graph.

   $O(n^2 \log n)$ (algorithm to be described)

2. Assign each edge (v, w) the weight $\| \overline{vw} \|$.

Shortest Path Algorithm

1. Construct the visibility graph.

   $O(n^2 \log n)$ (algorithm to be described)

2. Assign each edge (v, w) the weight $\| \overline{vw} \|$.

3. Run Dijkstra's algorithm.

Shortest Path Algorithm

1. Construct the visibility graph.

$O(n^2 \log n)$ (algorithm to be described)

2. Assign each edge (v, w) the weight $\| \overline{vw} \|$.

3. Run Dijkstra's algorithm.

$$O(|V| \log |V| + |E|) = O(n \log n + n^2) = O(n^2)$$

Shortest Path Algorithm

1. Construct the visibility graph.

   $O(n^2 \log n)$ (algorithm to be described)

2. Assign each edge (v, w) the weight $\| \overline{vw} \|$.

3. Run Dijkstra's algorithm.

$$O(|V| \log |V| + |E|) = O(n \log n + n^2) = O(n^2)$$

Theorem 3 A shortest path $p_{start} \rightsquigarrow p_{goal}$ among a set of polygonal obstacles with n edges can be computed in $O(n^2 \log n)$ time.

Constructing the Visibility Graph

It suffices to describe how to construct $\mathcal{G}_{vis}(S)$.

Brute-force strategy:

Test, for every pair of vertices, whether the line segment connecting them interests any obstacle.

Constructing the Visibility Graph

It suffices to describe how to construct $\mathcal{G}_{vis}(S)$.

Brute-force strategy:

Test, for every pair of vertices, whether the line segment connecting them interests any obstacle.

$O(n^2)$ pairs

$O(n)$ time per test

$O(n^3)$ running time

Constructing the Visibility Graph

It suffices to describe how to construct $\mathcal{G}_{vis}(S)$.

Brute-force strategy:

Test, for every pair of vertices, whether the line segment connecting them interests any obstacle.

$O(n^2)$ pairs

$O(n)$ time per test

$O(n^3)$ running time

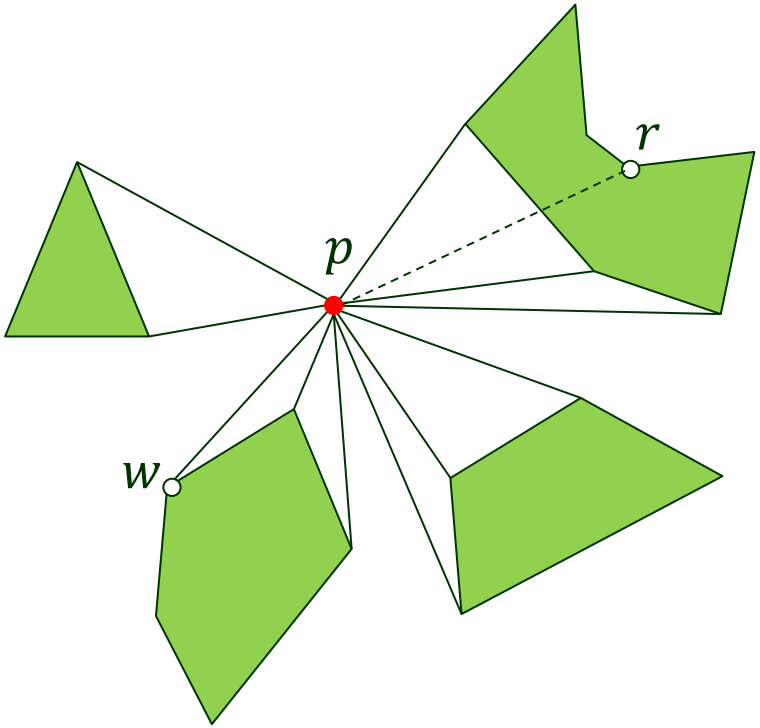
Improvement: Concentrate on one vertex at time to identify all visible vertices from it.

Improved Algorithm

VisibilityGraph(S)

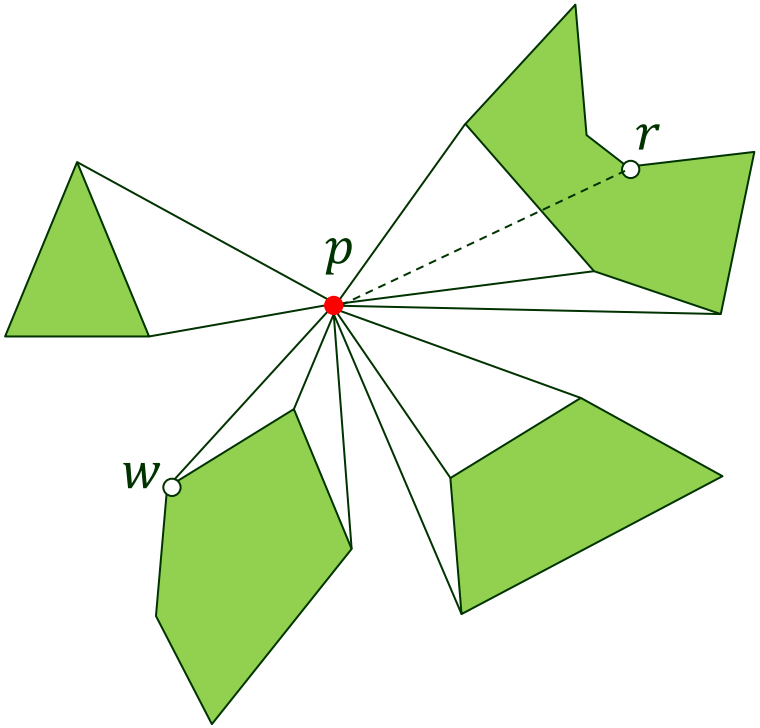
1. $V \leftarrow$ vertices of S
2. $E \leftarrow \emptyset$
3. initialize a graph $\mathcal{G} = (V, E)$
4. for all vertices $v \in V$
5. do $W \leftarrow$ VisibleVertices(v, S)
6. for every vertex $w \in W$
7. add the edge (v, w) to E
8. return $\mathcal{G}$

III. Visible Vertices from a Point



`VisibleVertices`(v, S) is made no more difficult if we consider v as a generally positioned point p in the free space instead of a vertex of some obstacle.

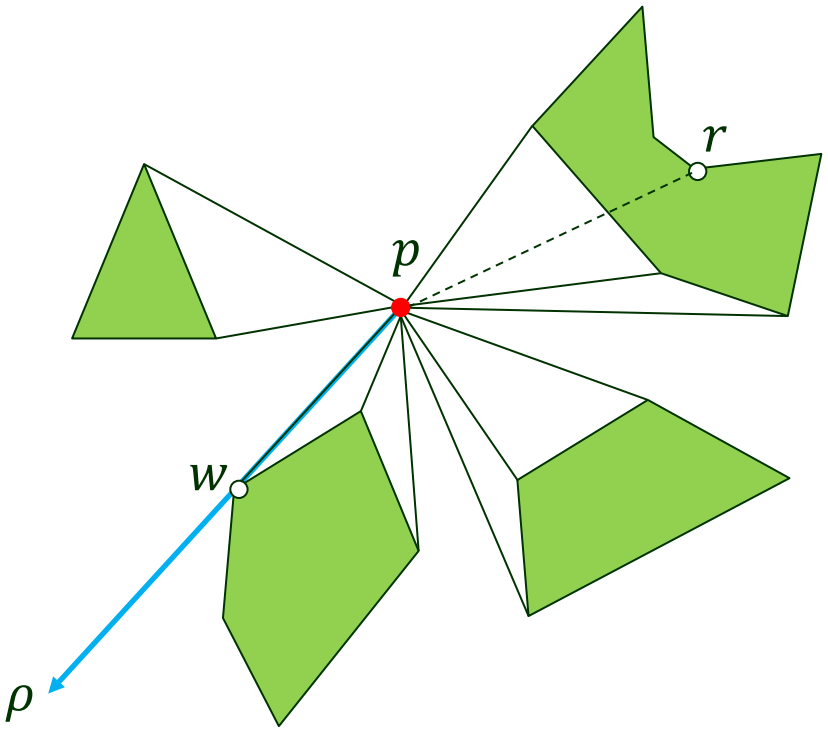
III. Visible Vertices from a Point



`VisibleVertices`(v, S) is made no more difficult if we consider v as a generally positioned point p in the free space instead of a vertex of some obstacle.

- A vertex w is *visible* from p if $\overline{wp}$ does not intersect the interior of any obstacle.

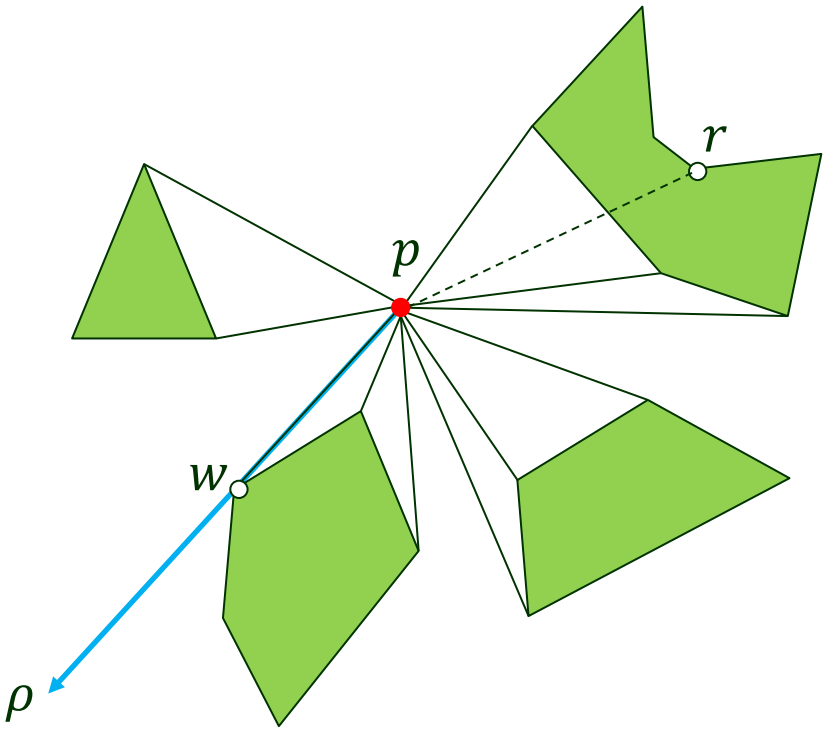
III. Visible Vertices from a Point



`VisibleVertices`(v, S) is made no more difficult if we consider v as a generally positioned point p in the free space instead of a vertex of some obstacle.

- A vertex w is *visible* from p if $\overline{wp}$ does not intersect the interior of any obstacle.
- Consider the half-line ρ starting at p and passing through w.

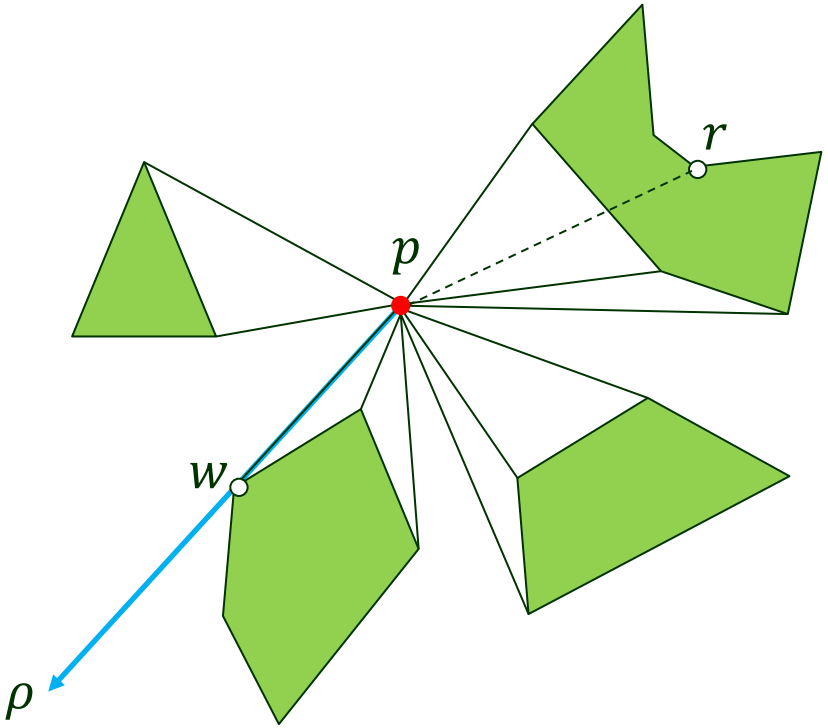
III. Visible Vertices from a Point



`VisibleVertices`(v, S) is made no more difficult if we consider v as a generally positioned point p in the free space instead of a vertex of some obstacle.

- A vertex w is *visible* from p if $\overline{wp}$ does not intersect the interior of any obstacle.
- Consider the half-line ρ starting at p and passing through w.
  - If ρ does not hit any obstacle edge before w, then w is visible.

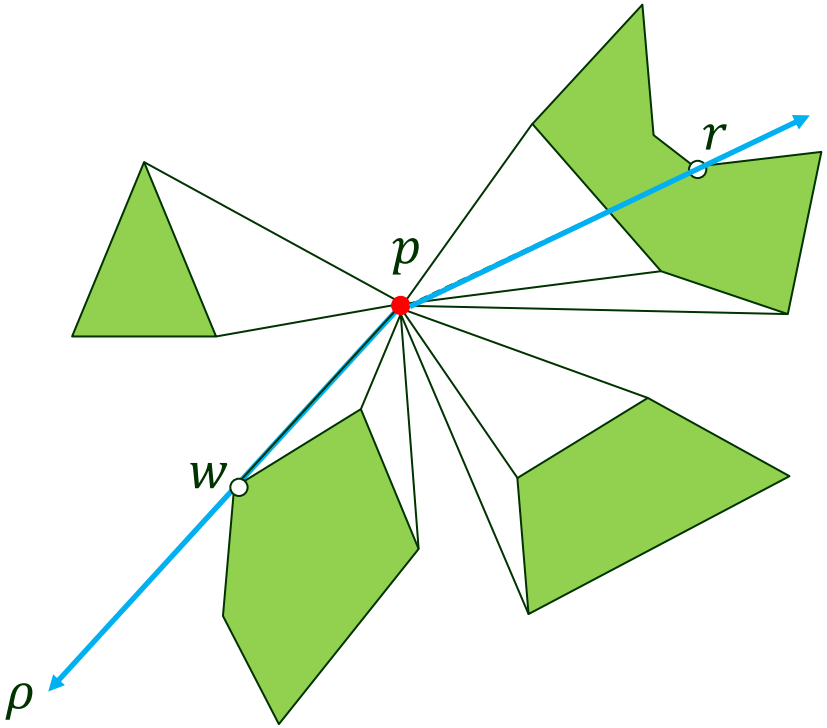
III. Visible Vertices from a Point



`VisibleVertices`(v, S) is made no more difficult if we consider v as a generally positioned point p in the free space instead of a vertex of some obstacle.

- A vertex w is *visible* from p if $\overline{wp}$ does not intersect the interior of any obstacle.
- Consider the half-line ρ starting at p and passing through w.
  - If ρ does not hit any obstacle edge before w, then w is visible.
  - Otherwise, it is invisible.

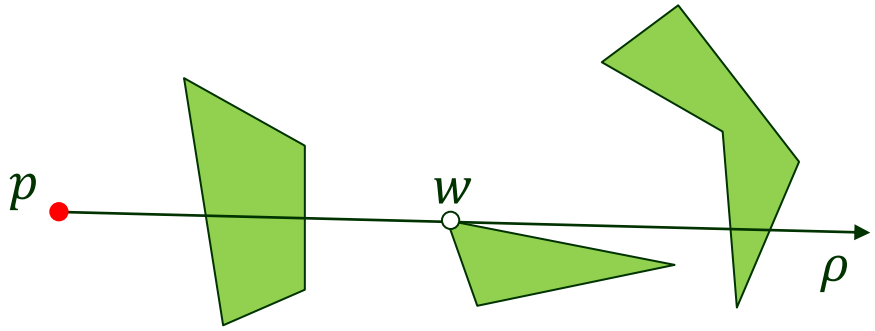
III. Visible Vertices from a Point



`VisibleVertices`(v, S) is made no more difficult if we consider v as a generally positioned point p in the free space instead of a vertex of some obstacle.

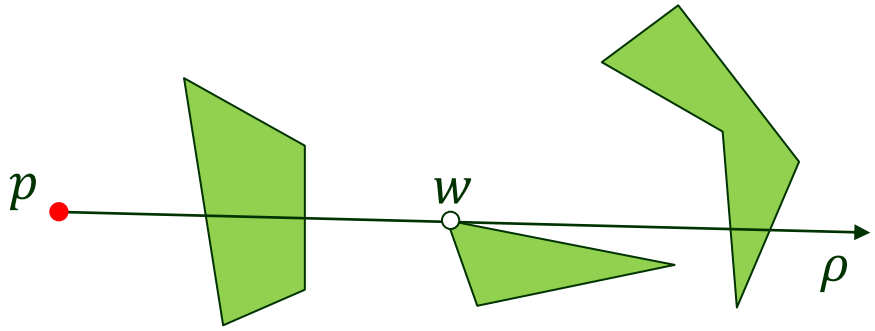
- A vertex w is *visible* from p if $\overline{wp}$ does not intersect the interior of any obstacle.
- Consider the half-line ρ starting at p and passing through w.
  - If ρ does not hit any obstacle edge before w, then w is visible.
  - Otherwise, it is invisible.

Visibility Checking



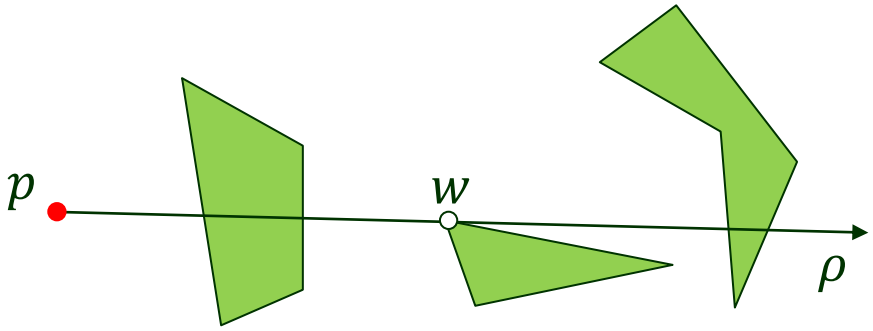
- Check visibility via binary search over edges intersected by the ray.

Visibility Checking

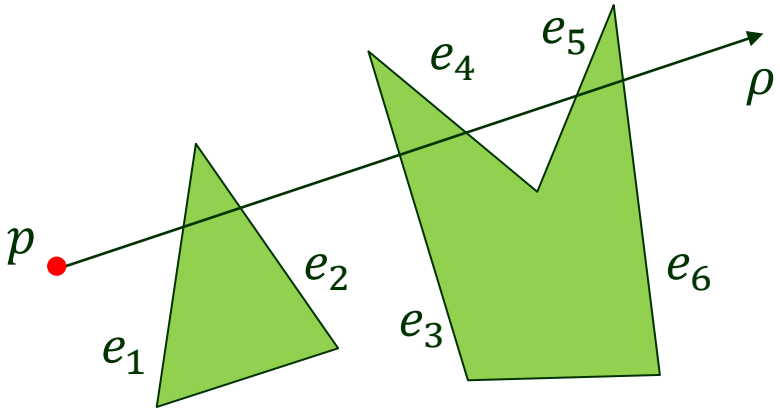


- Check visibility via binary search over edges intersected by the ray.
- Maintain the edges being intersected by ρ in the balanced binary search tree $\mathcal{T}$.

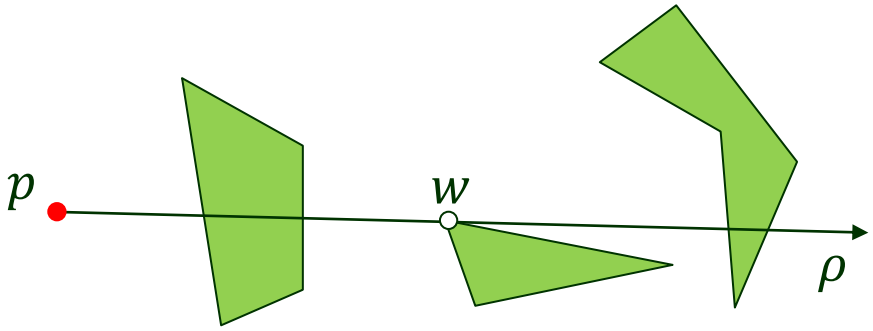
Visibility Checking



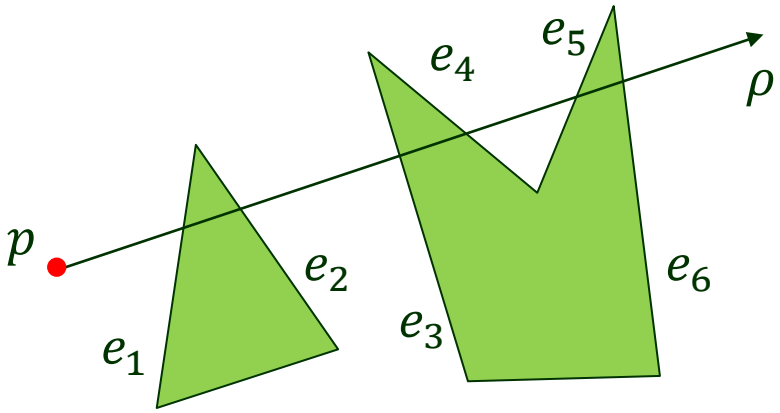
- Check visibility via binary search over edges intersected by the ray.
- Maintain the edges being intersected by ρ in the balanced binary search tree $\mathcal{T}$.

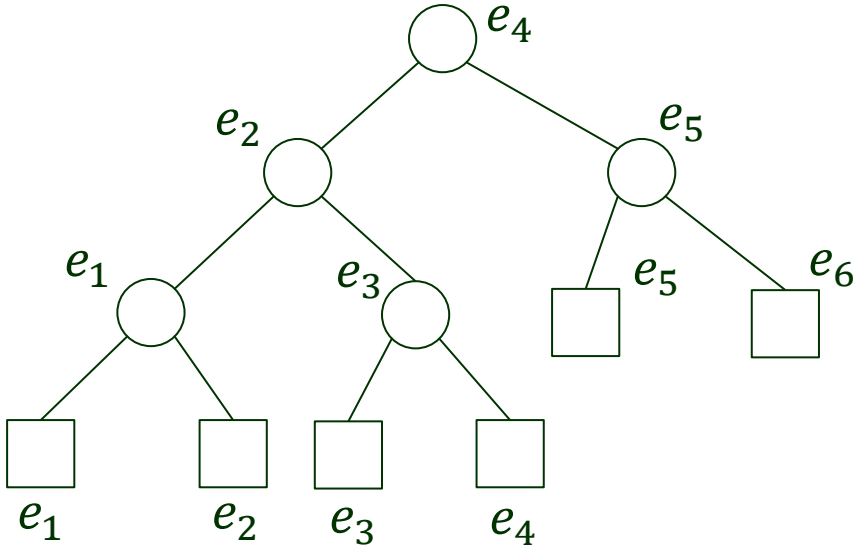


Visibility Checking

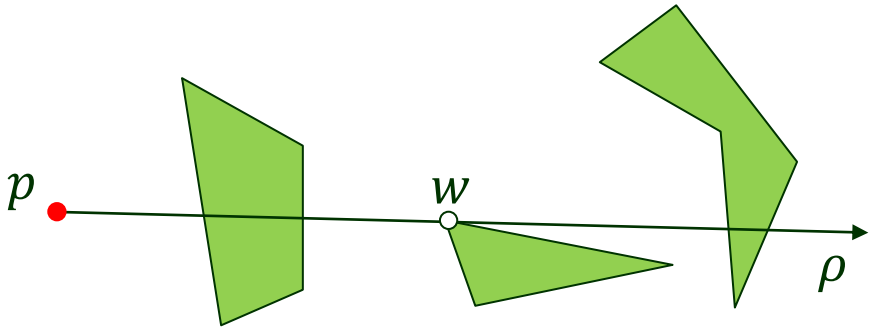


- Check visibility via binary search over edges intersected by the ray.
- Maintain the edges being intersected by ρ in the balanced binary search tree $\mathcal{T}$.

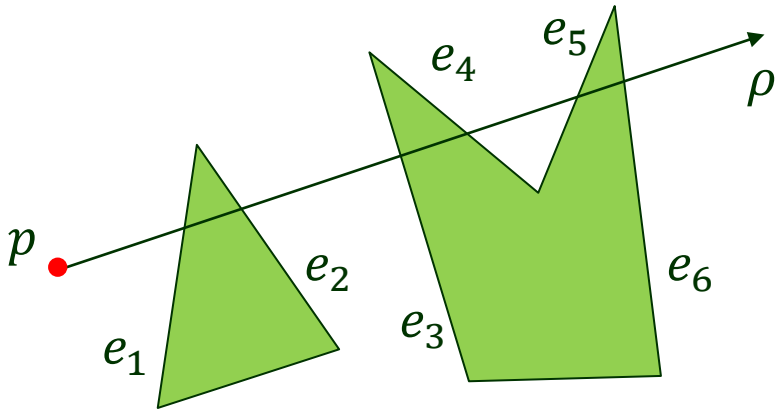


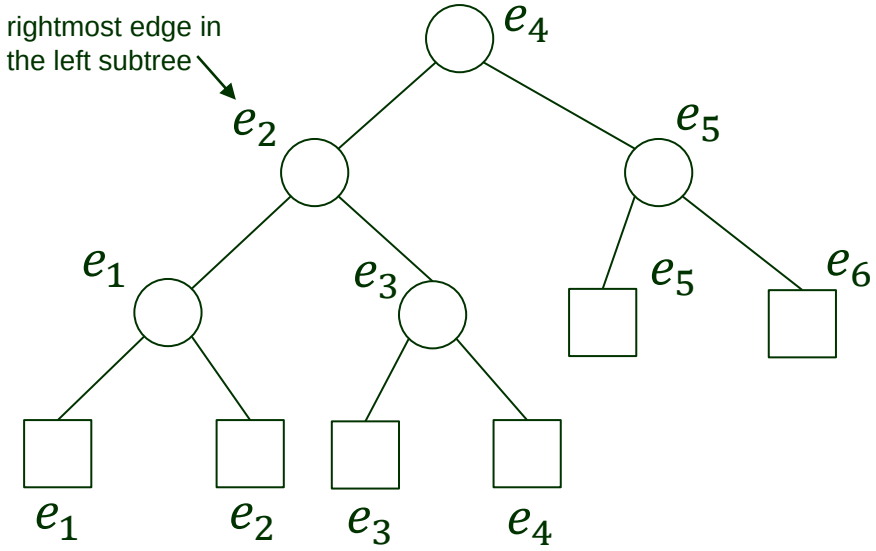


Visibility Checking

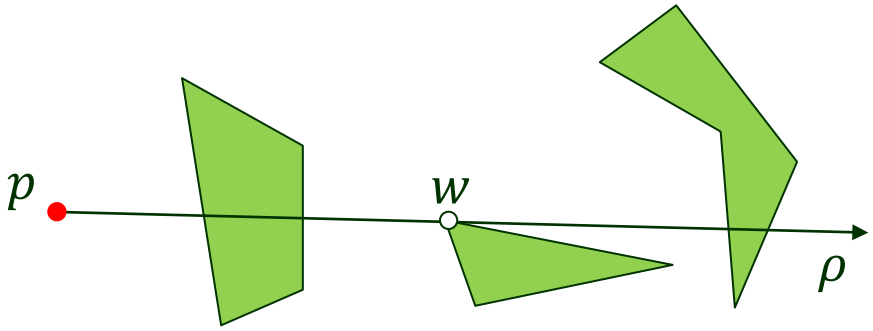


- Check visibility via binary search over edges intersected by the ray.
- Maintain the edges being intersected by ρ in the balanced binary search tree $\mathcal{T}$.

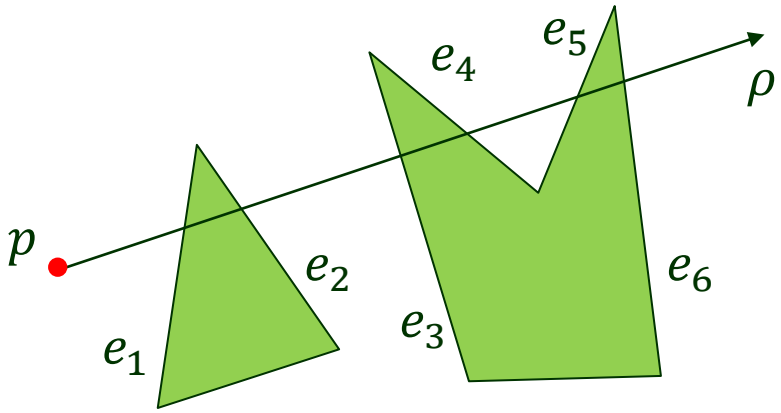


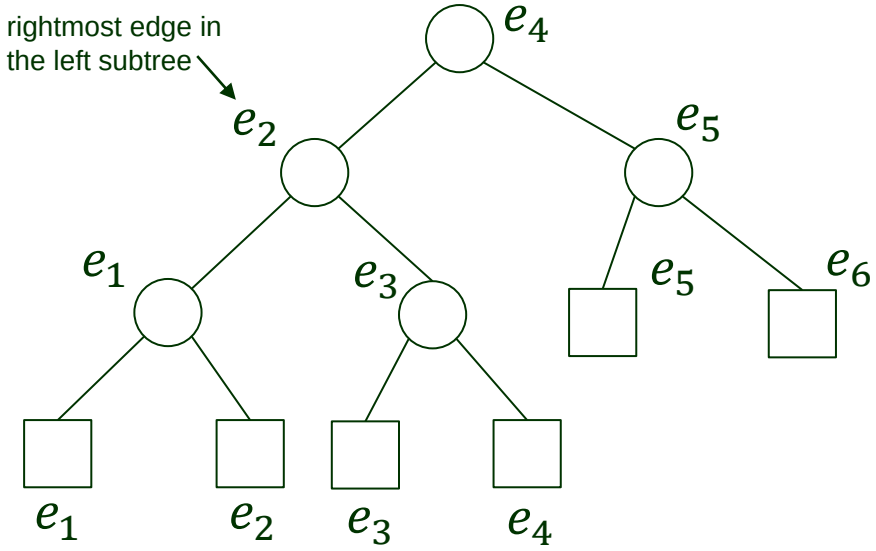


Visibility Checking



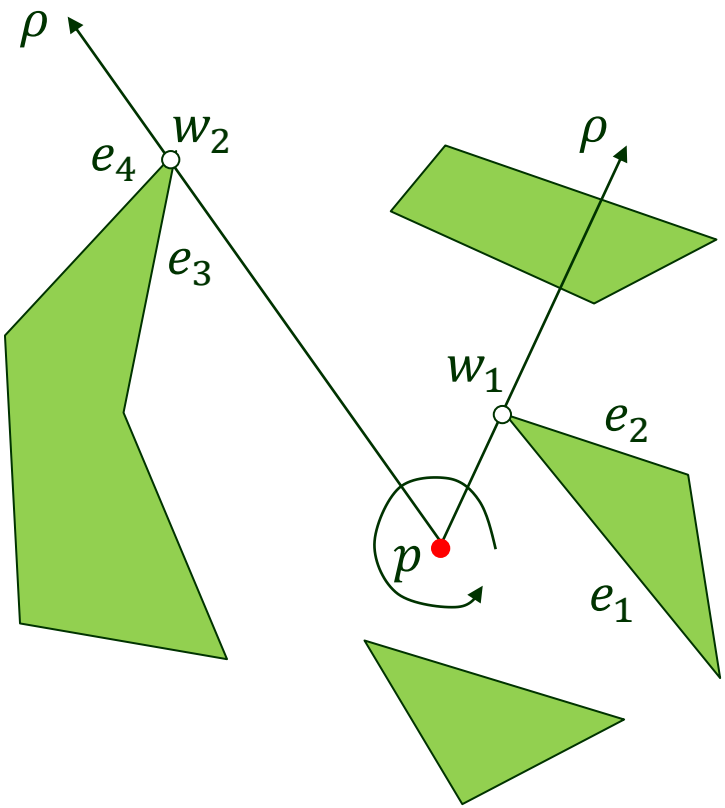
- Check visibility via binary search over edges intersected by the ray.
- Maintain the edges being intersected by ρ in the balanced binary search tree $\mathcal{T}$.





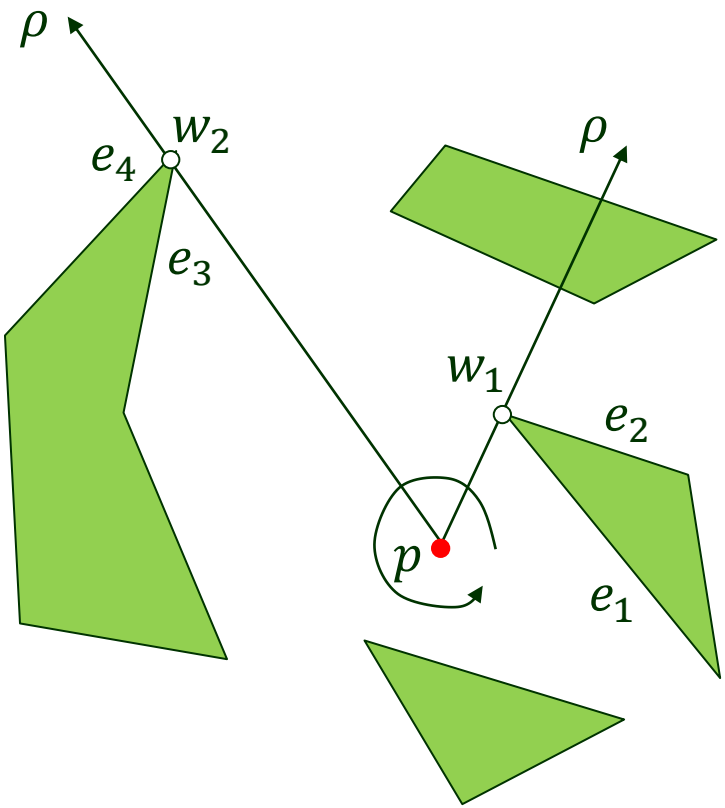
Status structure T

Rotational Sweep



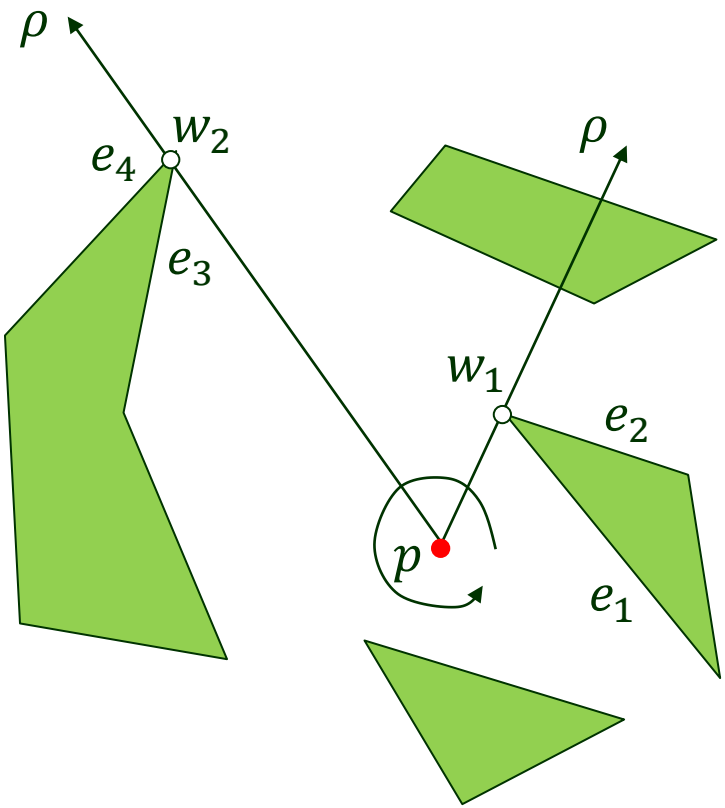
- Treat the vertices in cyclic order around p.

Rotational Sweep



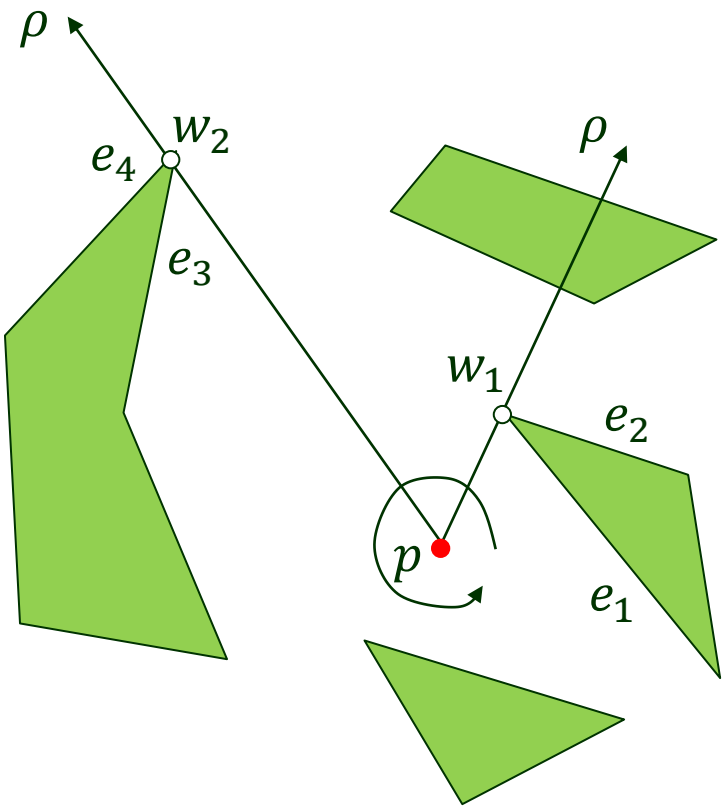
- Treat the vertices in cyclic order around p.
- Sweep the plane by rotating the ray ρ. At each vertex stop,

Rotational Sweep



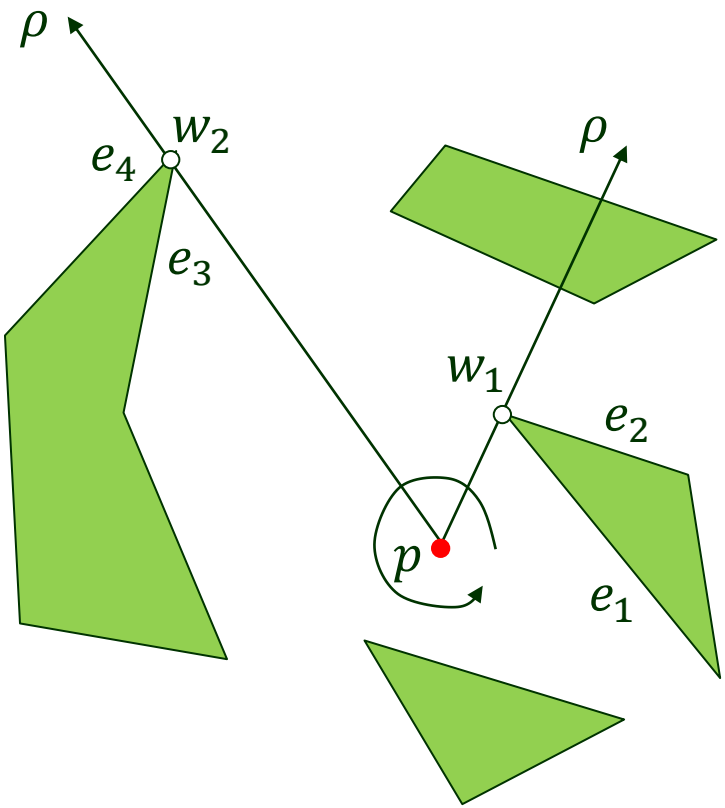
- Treat the vertices in cyclic order around p.
- Sweep the plane by rotating the ray ρ. At each vertex stop,
  - decide if the current vertex w is visible from p by searching in $\mathcal{T}$;

Rotational Sweep



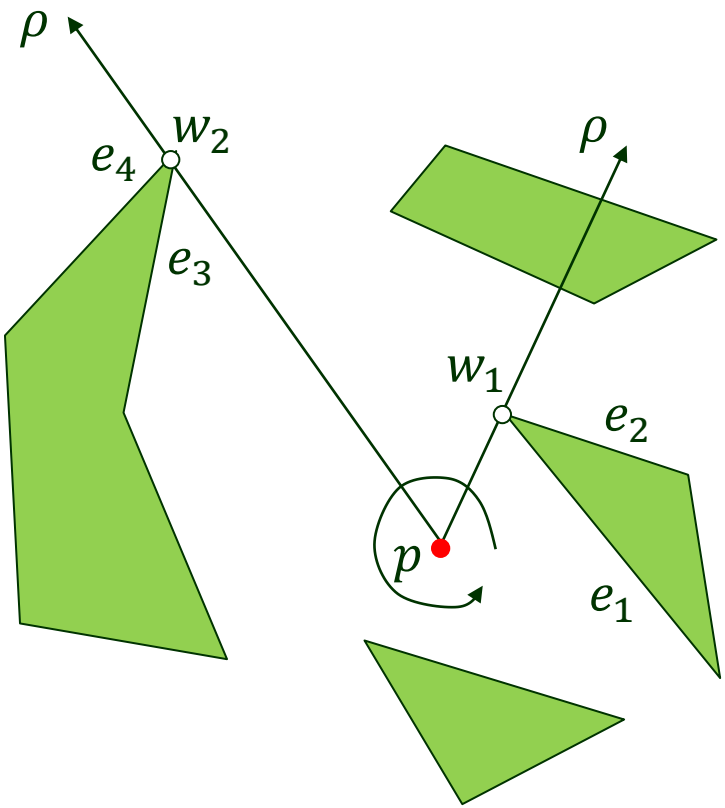
- Treat the vertices in cyclic order around p.
- Sweep the plane by rotating the ray ρ. At each vertex stop,
  - decide if the current vertex w is visible from p by searching in $\mathcal{T}$; $O(\log n)$

Rotational Sweep



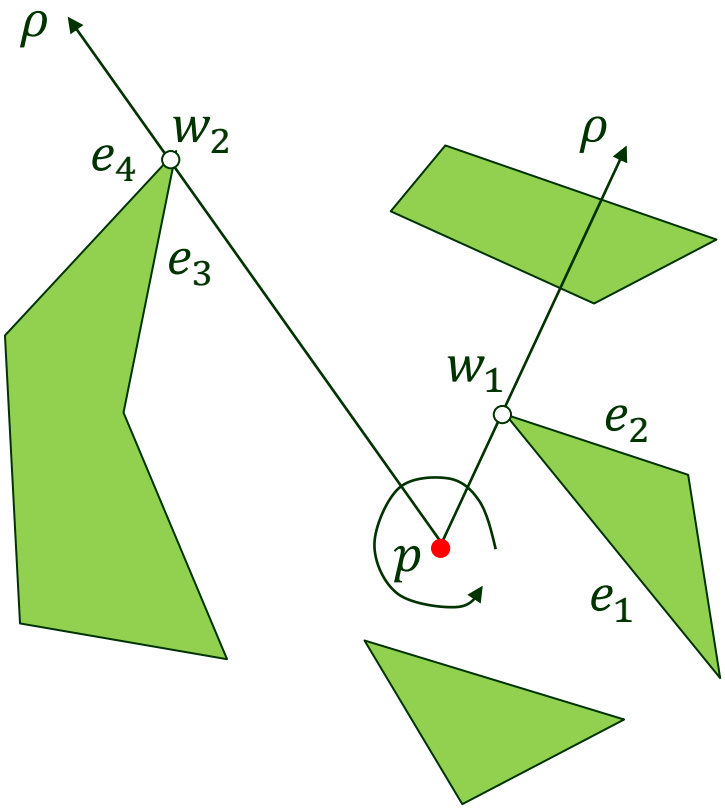
- Treat the vertices in cyclic order around p.
- Sweep the plane by rotating the ray ρ. At each vertex stop,
  - decide if the current vertex w is visible from p by searching in $\mathcal{T}$; $O(\log n)$
  - update $\mathcal{T}$ by inserting/deleting obstacle edges incident to w.

Rotational Sweep



- Treat the vertices in cyclic order around p.
- Sweep the plane by rotating the ray ρ. At each vertex stop,
  - decide if the current vertex w is visible from p by searching in $\mathcal{T}$; $O(\log n)$
  - update $\mathcal{T}$ by inserting/deleting obstacle edges incident to w. $O(\log n)$

Rotational Sweep



- Treat the vertices in cyclic order around p.
- Sweep the plane by rotating the ray ρ. At each vertex stop,
  - decide if the current vertex w is visible from p by searching in $\mathcal{T}$; $O(\log n)$
  - update $\mathcal{T}$ by inserting/deleting obstacle edges incident to w. $O(\log n)$

Delete e_1 and e_2 at w_1.

Insert e_3 and e_4 at w_2.

Algorithm for Finding Visible Vertices

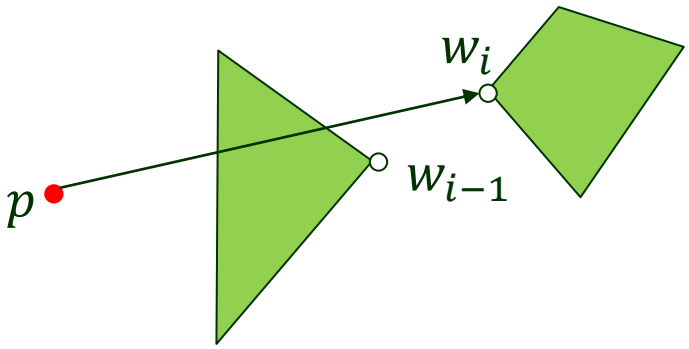
```
VisibleVertices(p, S)

1.  sort the obstacle vertices into a list w_1, w_2, ..., w_n by
    polar angle with respect to p, breaking tie by distance
2.  ρ ← half-line in the positive x-axis direction
3.  initialize a balanced BST T to store obstacle edges
    intersected by ρ
4.  W ← ∅
5.  for i ← 1 to n
6.     do if Visible(w_i) // to be described
7.           then W ← W ∪ {w_i}
8.                insert into T incident obstacle edges at w_i
                     that are entering the sweep line status
9.                delete from T those incident edges that
                     are leaving the status
10. return W
```

Back to Visibility Checking

Take a closer look at checking if a vertex is visible during the sweep.

This is carried out by `Visible(`w_i`).`



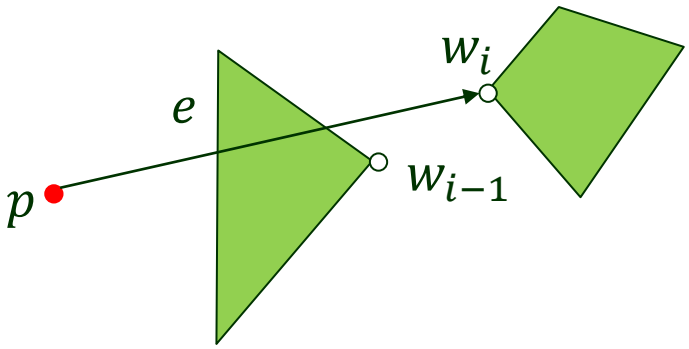
Non-degenerate case:

w_{i-1} does not lie on the directed edge $\overrightarrow{pw_i}$ (part of ρ), i.e., $w_{i-1} \notin \overrightarrow{pw_i}$.

Back to Visibility Checking

Take a closer look at checking if a vertex is visible during the sweep.

This is carried out by `Visible(`w_i`).`



Non-degenerate case:

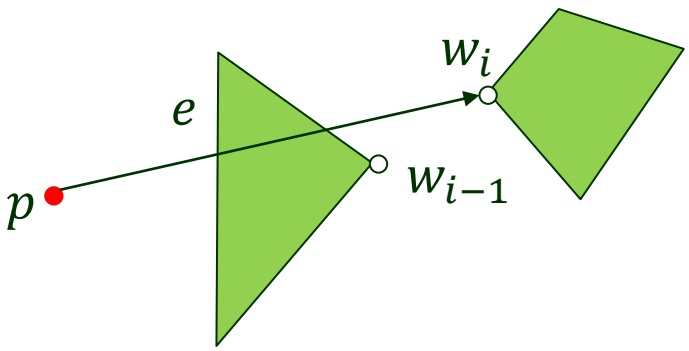
w_{i-1} does not lie on the directed edge $\overrightarrow{pw_i}$ (part of ρ), i.e., $w_{i-1} \notin \overrightarrow{pw_i}$.

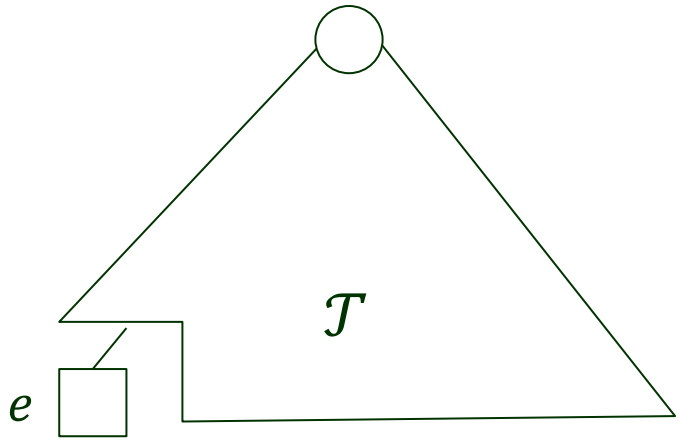
Check if w_i is occluded by the edge e closest to p.

Back to Visibility Checking

Take a closer look at checking if a vertex is visible during the sweep.

This is carried out by `Visible(`w_i`)`.





Non-degenerate case:

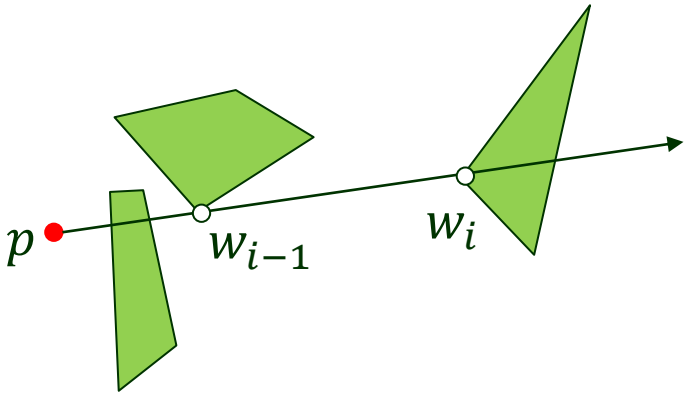
w_{i-1} does not lie on the directed edge $\overrightarrow{pw_i}$ (part of ρ), i.e., $w_{i-1} \notin \overrightarrow{pw_i}$.

Check if w_i is occluded by the edge e closest to p.

e is stored at the leftmost leaf of $\mathcal{T}$.

Handling Degeneracies

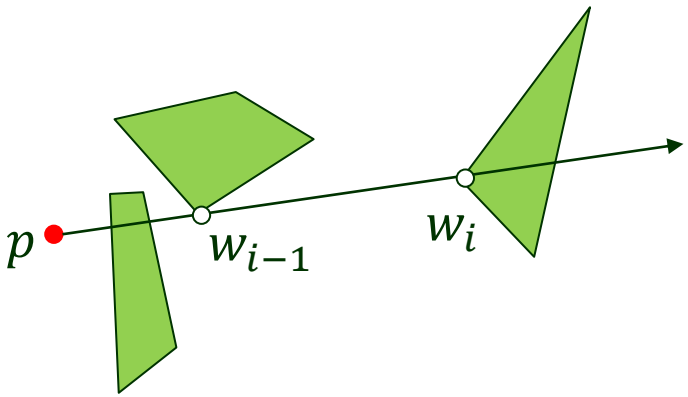
Degenerate case: $w_{i-1} \in \overrightarrow{pw_i}$



- w_{i-1} is not visible

Handling Degeneracies

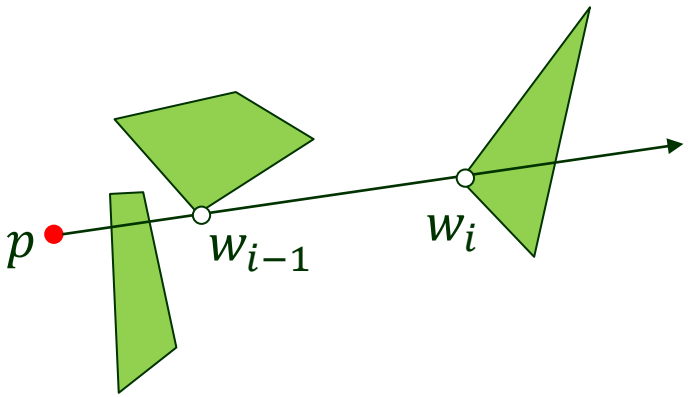
Degenerate case: $w_{i-1} \in \overrightarrow{pw_i}$



- w_{i-1} is not visible $\Longrightarrow$ w_i is not either.

Handling Degeneracies

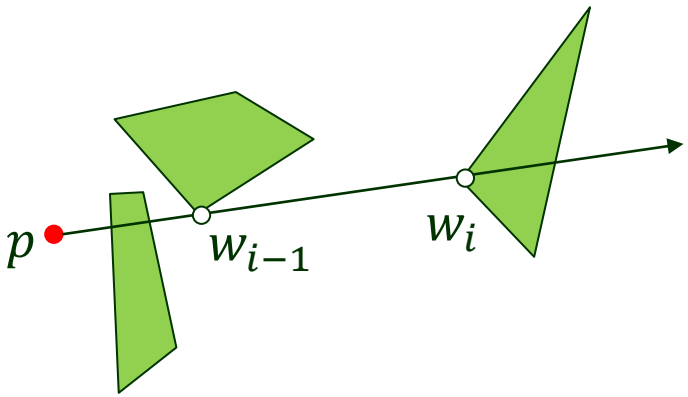
Degenerate case: $w_{i-1} \in \overrightarrow{pw_i}$



- w_{i-1} is not visible $\Longrightarrow$ w_i is not either.
- w_{i-1} is visible

Handling Degeneracies

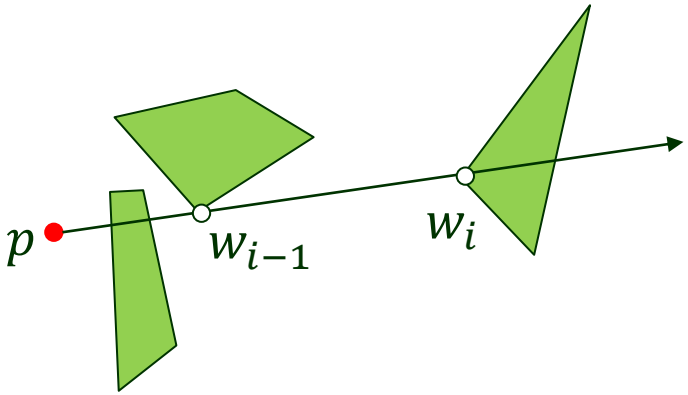
Degenerate case: $w_{i-1} \in \overrightarrow{pw_i}$

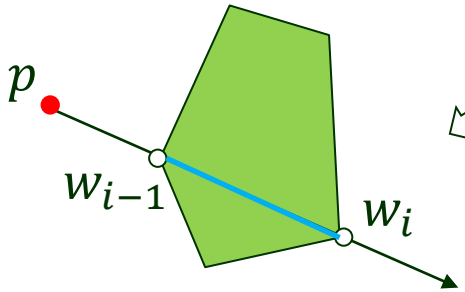


- w_{i-1} is not visible $\Longrightarrow$ w_i is not either.
- w_{i-1} is visible
  - Look at $\overline{w_{i-1}w_i}$. w_i is not visible if

Handling Degeneracies

Degenerate case: $w_{i-1} \in \overrightarrow{pw_i}$

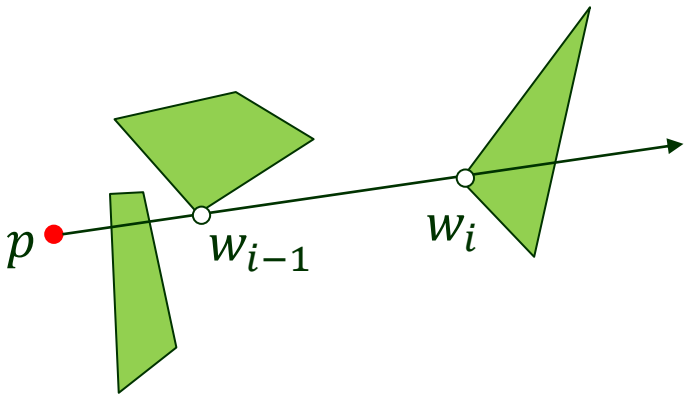




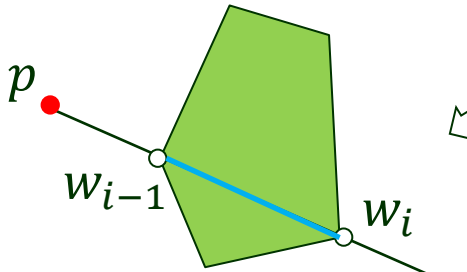
- w_{i-1} is not visible $\Longrightarrow$ w_i is not either.
- w_{i-1} is visible
  - Look at $\overline{w_{i-1}w_i}$. w_i is not visible if
    - lies inside an obstacle (which thus must have w_i and w_{i-1} as vertices), or

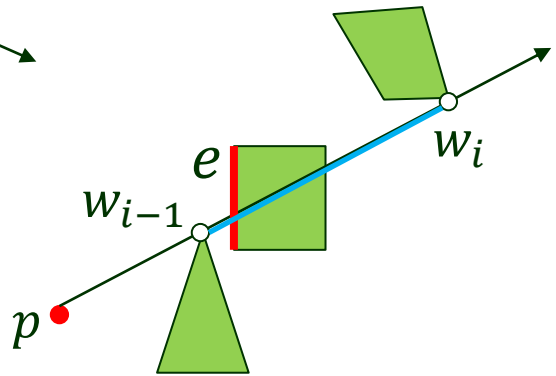
Handling Degeneracies

Degenerate case: $w_{i-1} \in \overrightarrow{pw_i}$



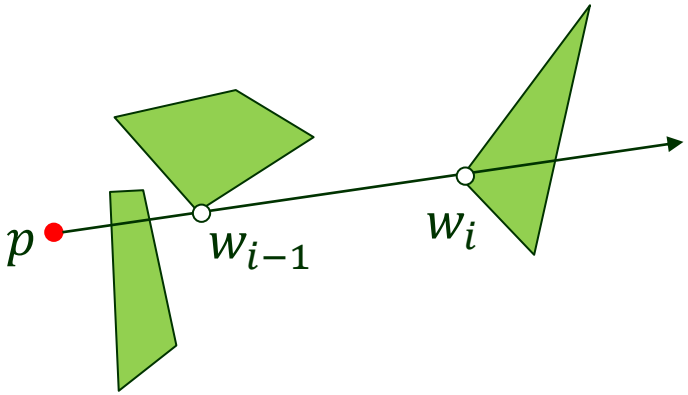
- w_{i-1} is not visible $\Longrightarrow$ w_i is not either.
- w_{i-1} is visible
  - Look at $\overline{w_{i-1}w_i}$. w_i is not visible if
    - ♣ lies inside an obstacle (which thus must have w_i and w_{i-1} as vertices), or
    - ♣ intersects an obstacle edge e in its interior.



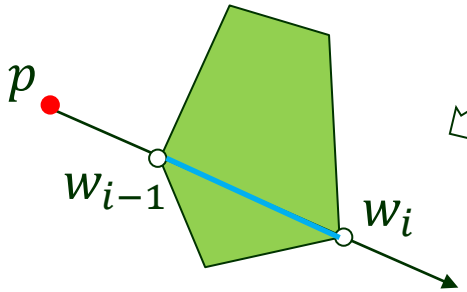


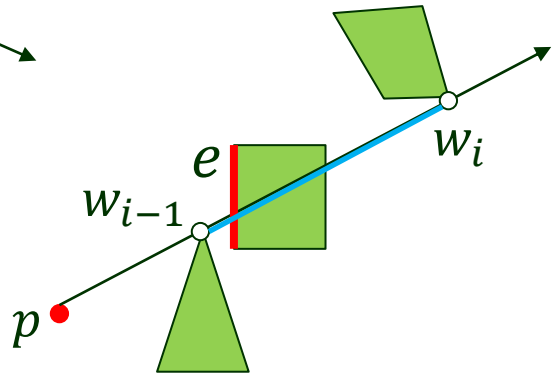
Handling Degeneracies

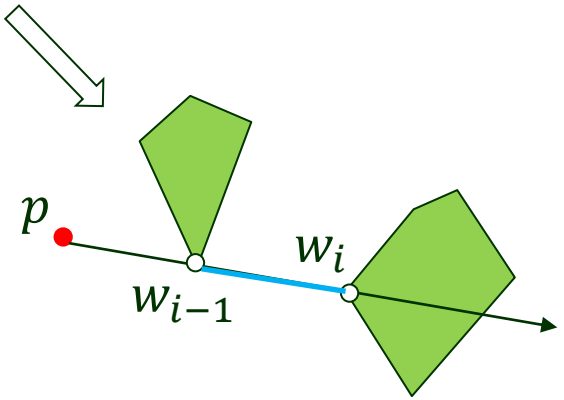
Degenerate case: $w_{i-1} \in \overrightarrow{pw_i}$



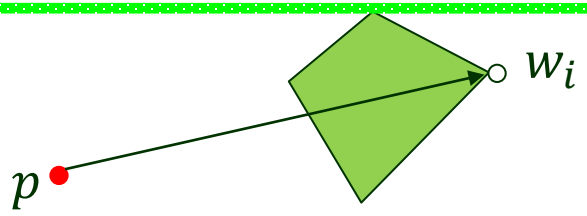
- w_{i-1} is not visible $\Longrightarrow$ w_i is not either.
- w_{i-1} is visible
  - Look at $\overline{w_{i-1}w_i}$. w_i is not visible if
    - ♣ lies inside an obstacle (which thus must have w_i and w_{i-1} as vertices), or
    - ♣ intersects an obstacle edge e in its interior.
  - w_i is visible otherwise.



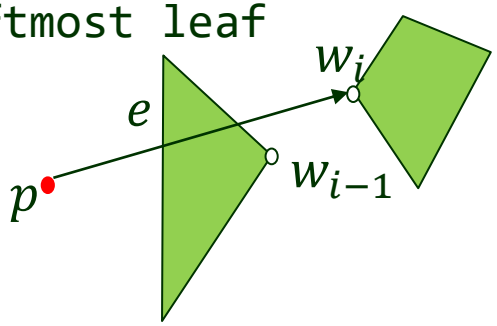


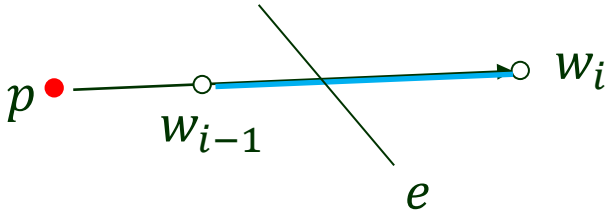


Pseudocode for Visibility Checking




```
Visible(w_i)
1. if pw_i intersects the interior of the obstacle with w_i as a vertex
2.    then return false
3.    else if i = 1 or w_{i-1} is not on the segment pw_i
4.       then search in T for the edge e in the leftmost leaf
5.            if e exists and pw_i intersects e
6.               then return false
7.               else return true
8.            else if w_{i-1} is not visible
9.                    then return false
10.                   else search T for an edge e that intersects w_i w_{i-1}
11.                       if e exists
12.                          then return false
13.                          else return true
```





$O(\log n)$

Time on Finding All Visible Vertices

```
VisibleVertices(p,S)

1.  sort the obstacle vertices into a list w_1,w_2,…,w_n by
    polar angle with respect to p, breaking tie by distance
2.  ρ ← half-line in the positive x-axis direction
3.  initialize a balanced BST T to store obstacle edges
    intersected by ρ
4.  W ← ∅
5.  for i ← 1 to n
6.     do if Visible(w_i)
7.           then W ← W ∪ {w_i}
8.                insert into T incident obstacle edges to w_i
                     that are entering the sweep line status
9.                delete from T those incident edges that
                     are leaving the status
10. return W
```

Time on Finding All Visible Vertices

VisibleVertices(p,S)

1. sort the obstacle vertices into a list $w_1, w_2, \ldots, w_n$ by polar angle with respect to p, breaking tie by distance — $O(n \log n)$
2. $\rho \leftarrow$ half-line in the positive x-axis direction
3. initialize a balanced BST $\mathcal{T}$ to store obstacle edges intersected by ρ
4. $W \leftarrow \emptyset$
5. for $i \leftarrow 1$ to n
6. do if Visible(w_i)
7. then $W \leftarrow W \cup \{w_i\}$
8. insert into $\mathcal{T}$ incident obstacle edges to w_i that are entering the sweep line status
9. delete from $\mathcal{T}$ those incident edges that are leaving the status
10. return W

Time on Finding All Visible Vertices

VisibleVertices(p, S)

1. sort the obstacle vertices into a list $w_1, w_2, \ldots, w_n$ by polar angle with respect to p, breaking tie by distance — $O(n \log n)$
2. $\rho \leftarrow$ half-line in the positive x-axis direction
3. initialize a balanced BST $\mathcal{T}$ to store obstacle edges intersected by ρ — $O(n \log n)$
4. $W \leftarrow \emptyset$
5. for $i \leftarrow 1$ to n
6. do if Visible(w_i)
7. then $W \leftarrow W \cup \{w_i\}$
8. insert into $\mathcal{T}$ incident obstacle edges to w_i that are entering the sweep line status
9. delete from $\mathcal{T}$ those incident edges that are leaving the status
10. return W

Time on Finding All Visible Vertices

VisibleVertices(p, S)

1. sort the obstacle vertices into a list $w_1, w_2, \ldots, w_n$ by polar angle with respect to p, breaking tie by distance — $O(n \log n)$
2. $\rho \leftarrow$ half-line in the positive x-axis direction
3. initialize a balanced BST $\mathcal{T}$ to store obstacle edges intersected by ρ — $O(n \log n)$
4. $W \leftarrow \emptyset$
5. for $i \leftarrow 1$ to n
6. do if Visible(w_i) — $O(\log n)$
7. then $W \leftarrow W \cup \{w_i\}$
8. insert into $\mathcal{T}$ incident obstacle edges to w_i that are entering the sweep line status
9. delete from $\mathcal{T}$ those incident edges that are leaving the status
10. return W

Time on Finding All Visible Vertices

VisibleVertices(p, S)

1. sort the obstacle vertices into a list $w_1, w_2, \ldots, w_n$ by polar angle with respect to p, breaking tie by distance — $O(n \log n)$
2. $\rho \leftarrow$ half-line in the positive x-axis direction
3. initialize a balanced BST $\mathcal{T}$ to store obstacle edges intersected by ρ — $O(n \log n)$
4. $W \leftarrow \emptyset$
5. for $i \leftarrow 1$ to n
6. do if Visible(w_i) — $O(\log n)$
7. then $W \leftarrow W \cup \{w_i\}$
8. insert into $\mathcal{T}$ incident obstacle edges to w_i that are entering the sweep line status
9. delete from $\mathcal{T}$ those incident edges that are leaving the status — $O(\log n)$
10. return W

Time on Finding All Visible Vertices

VisibleVertices(p,S) $O(n \log n)$

1. sort the obstacle vertices into a list $w_1, w_2, \ldots, w_n$ by polar angle with respect to p, breaking tie by distance $O(n \log n)$
2. $\rho \leftarrow$ half-line in the positive x-axis direction
3. initialize a balanced BST $\mathcal{T}$ to store obstacle edges intersected by ρ $O(n \log n)$
4. $W \leftarrow \emptyset$
5. for $i \leftarrow 1$ to n
6. do if Visible(w_i) $O(\log n)$
7. then $W \leftarrow W \cup \{w_i\}$
8. insert into $\mathcal{T}$ incident obstacle edges to w_i that are entering the sweep line status
9. delete from $\mathcal{T}$ those incident edges that are leaving the status $O(\log n)$
10. return W

Time for Visibility Graph Construction

To construct the visibility graph, we call

`VisibleVertices()`

on every obstacle vertex, p_{start}, and p_{goal}.

- $n + 2$ calls
- $O(n \log n)$ each call

Time for Visibility Graph Construction

To construct the visibility graph, we call

`VisibleVertices()`

on every obstacle vertex, p_{start}, and p_{goal}.

- $n + 2$ calls
- $O(n \log n)$ each call

Construction of the visibility graph takes $O(n^2 \log n)$ time.

IV. Shortest Path for a Translational Robot

R: convex translational robot with constant complexity

S: a set of polygonal obstacles with n edges in total.

IV. Shortest Path for a Translational Robot

R: convex translational robot with constant complexity

S: a set of polygonal obstacles with n edges in total.

Work space

IV. Shortest Path for a Translational Robot

R: convex translational robot with constant complexity

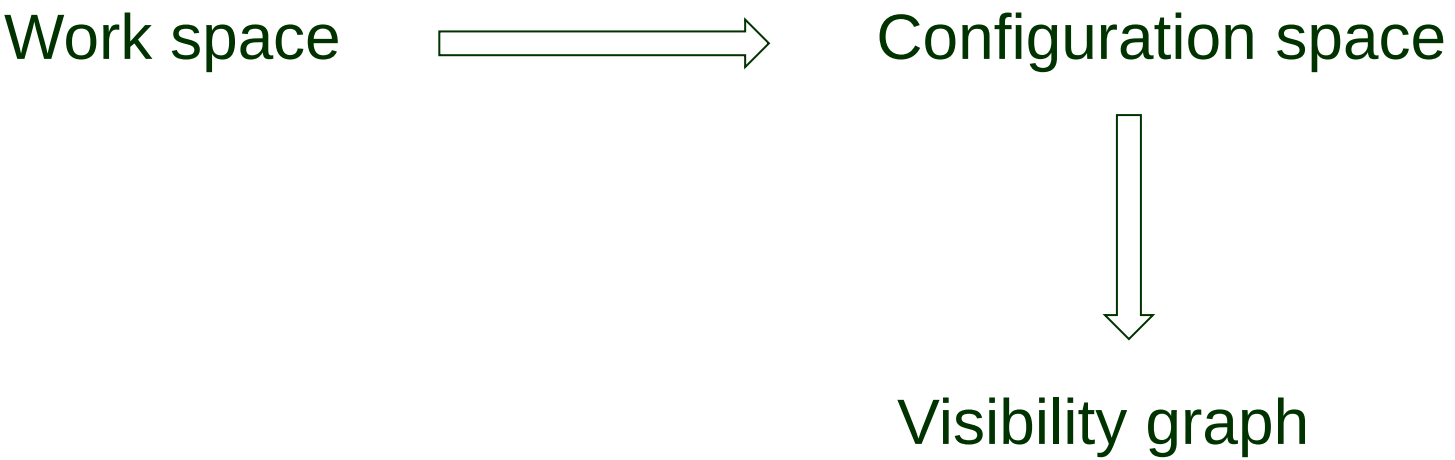
S: a set of polygonal obstacles with n edges in total.

Work space ⟹ Configuration space

IV. Shortest Path for a Translational Robot

R: convex translational robot with constant complexity

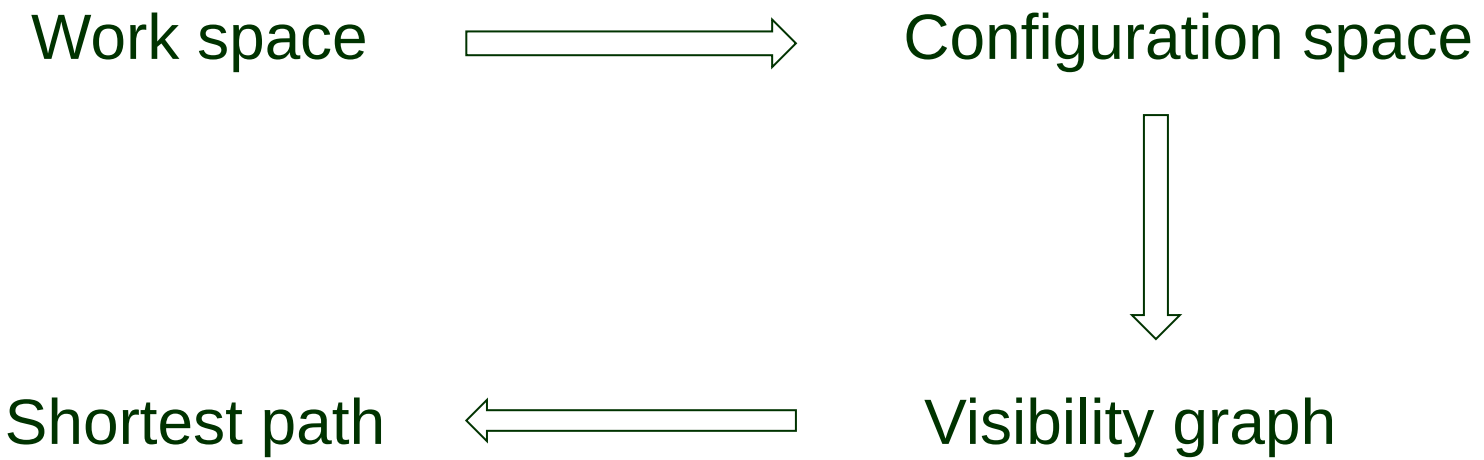
S: a set of polygonal obstacles with n edges in total.



IV. Shortest Path for a Translational Robot

R: convex translational robot with constant complexity

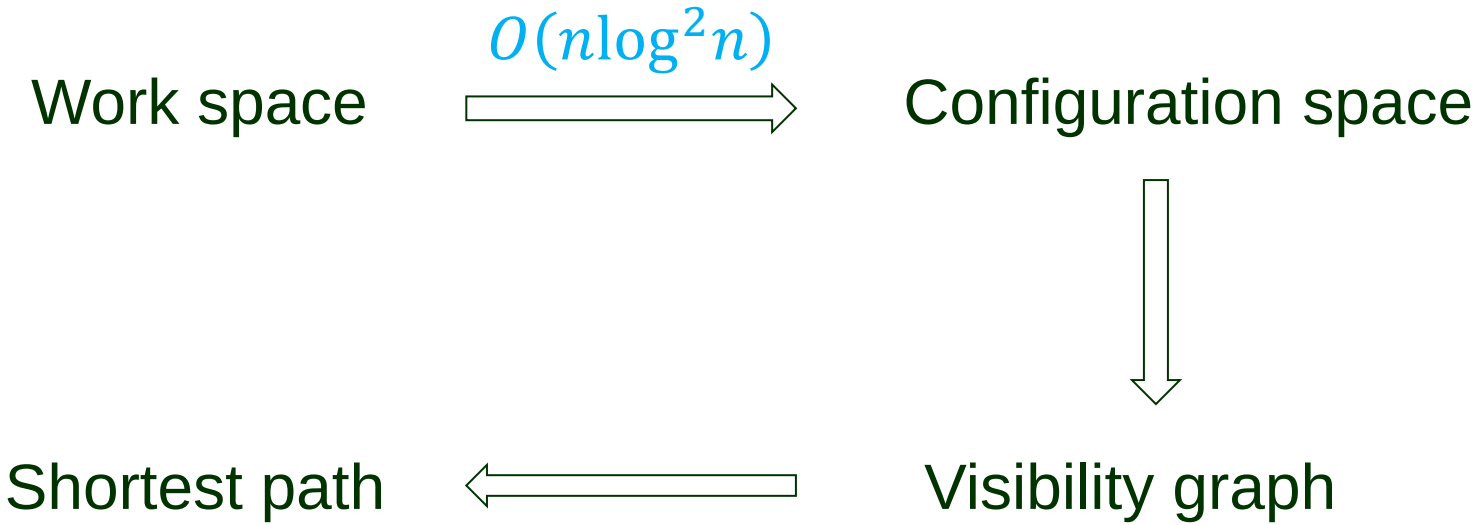
S: a set of polygonal obstacles with n edges in total.



IV. Shortest Path for a Translational Robot

R: convex translational robot with constant complexity

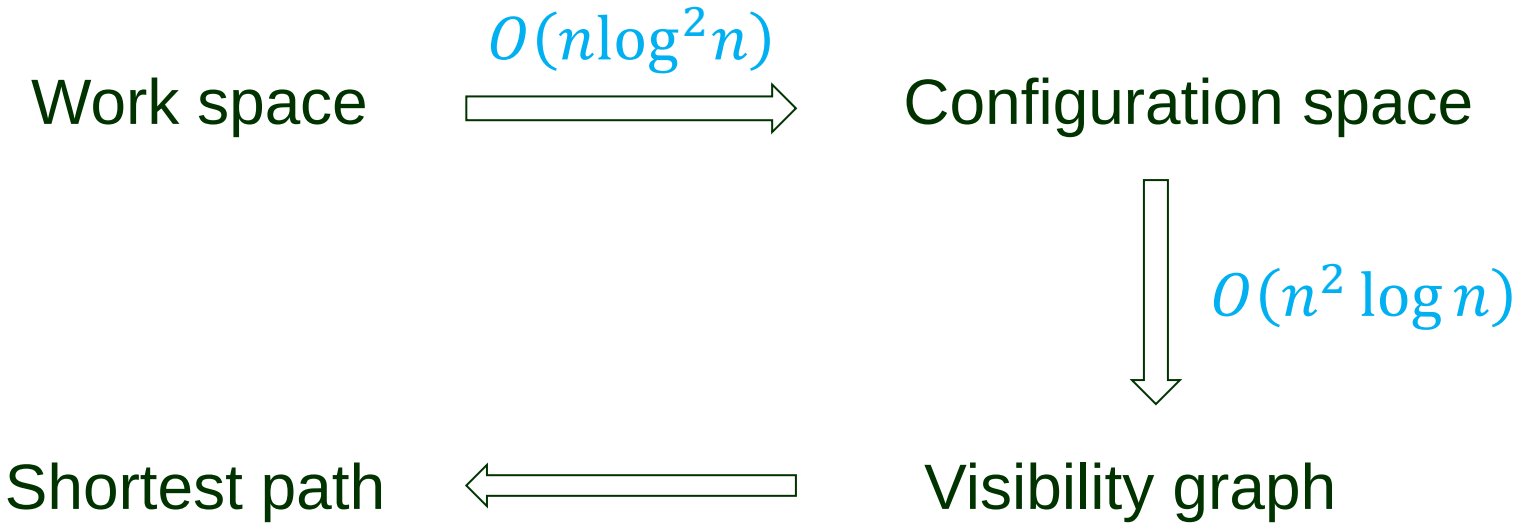
S: a set of polygonal obstacles with n edges in total.



IV. Shortest Path for a Translational Robot

R: convex translational robot with constant complexity

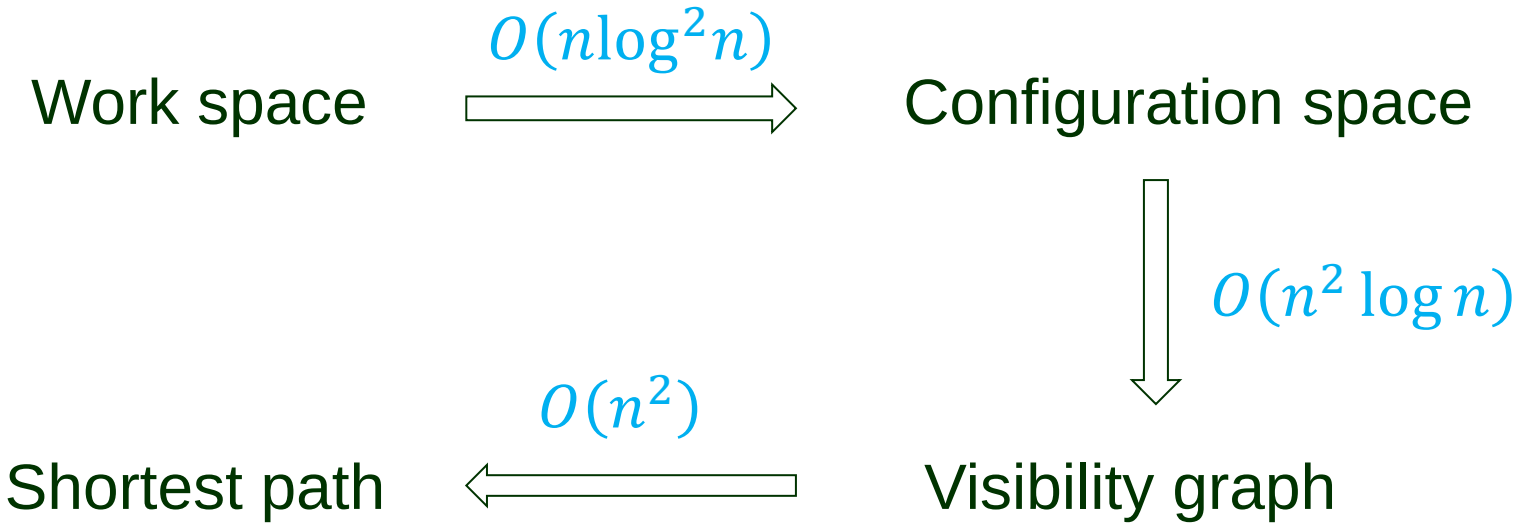
S: a set of polygonal obstacles with n edges in total.



IV. Shortest Path for a Translational Robot

R: convex translational robot with constant complexity

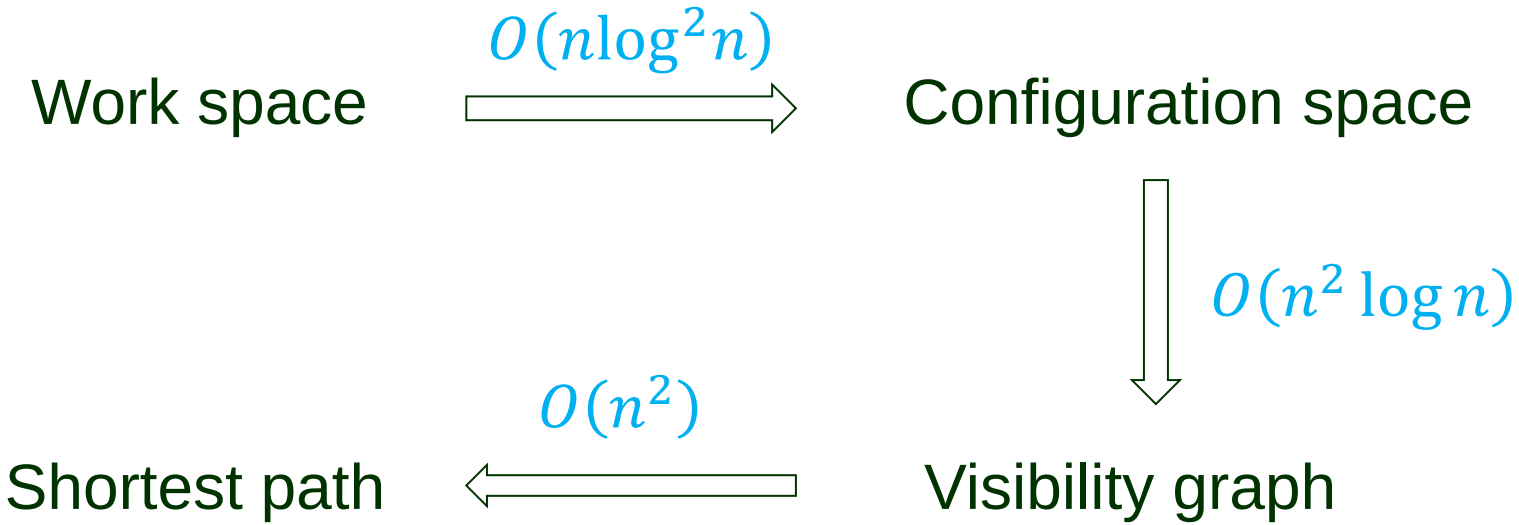
S: a set of polygonal obstacles with n edges in total.



IV. Shortest Path for a Translational Robot

R: convex translational robot with constant complexity

S: a set of polygonal obstacles with n edges in total.



Theorem 4 A shortest collision-free path for R between two placements can be found in $O(n^2 \log n)$ time.