Altitude Terrain Guarding and Guarding Uni-Monotone Polygons

Stephan Friedrichs Valentin Polishchuk
Christiane Schmidt
max planck institut informatik
LINKÖPING UNIVERSITY

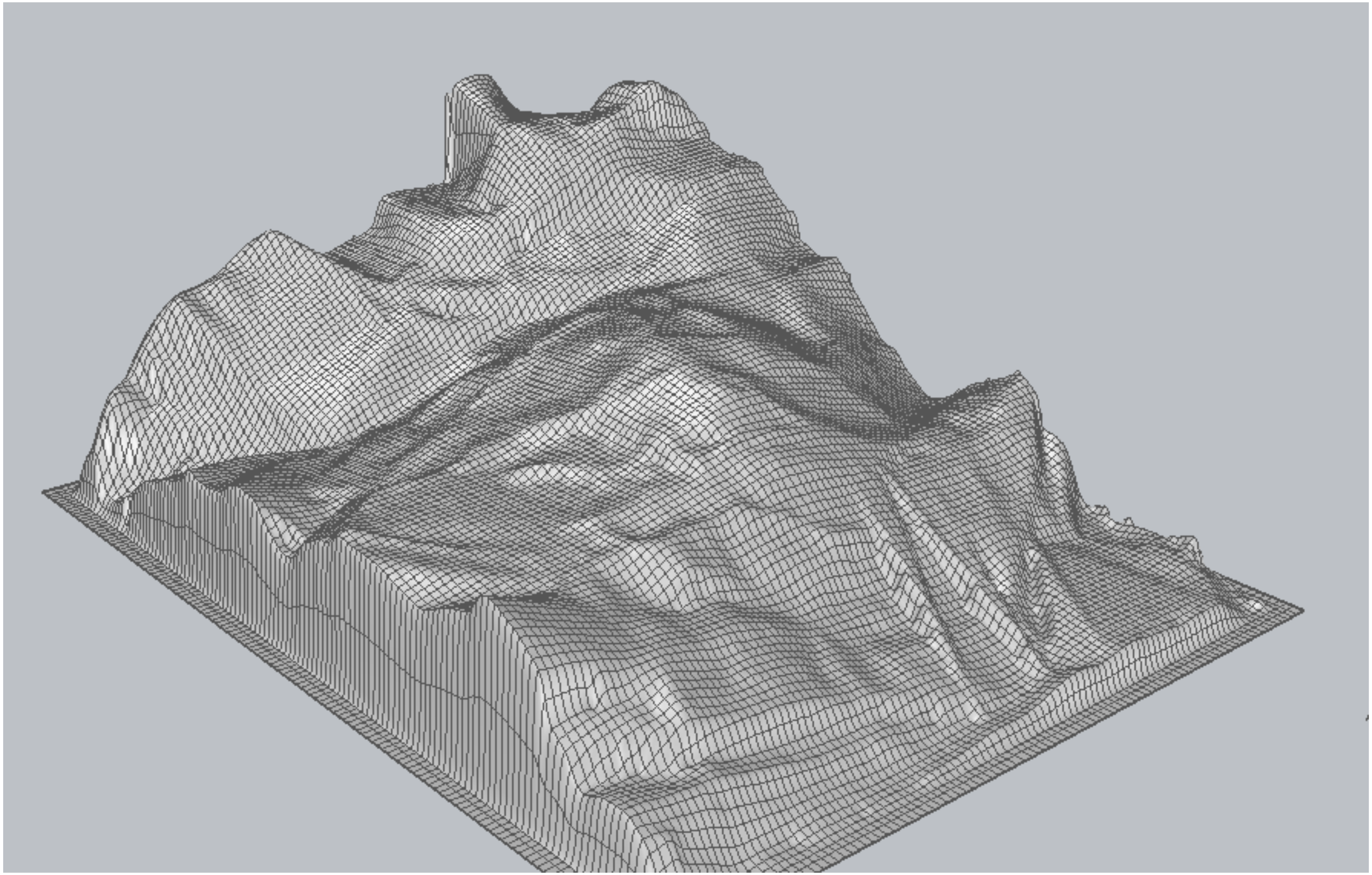


image source: http://2.bp.blogspot.com/-eUnhAo8SfFQ/VDaLij2_csI/AAAAAAAABPM/847rgT6VpQE/s1600/Screen%2BShot%2B2014-05-01%2Bat%2B00.02.54.png
https://www.dronerush.com/best-drones-1977/drone.bmp

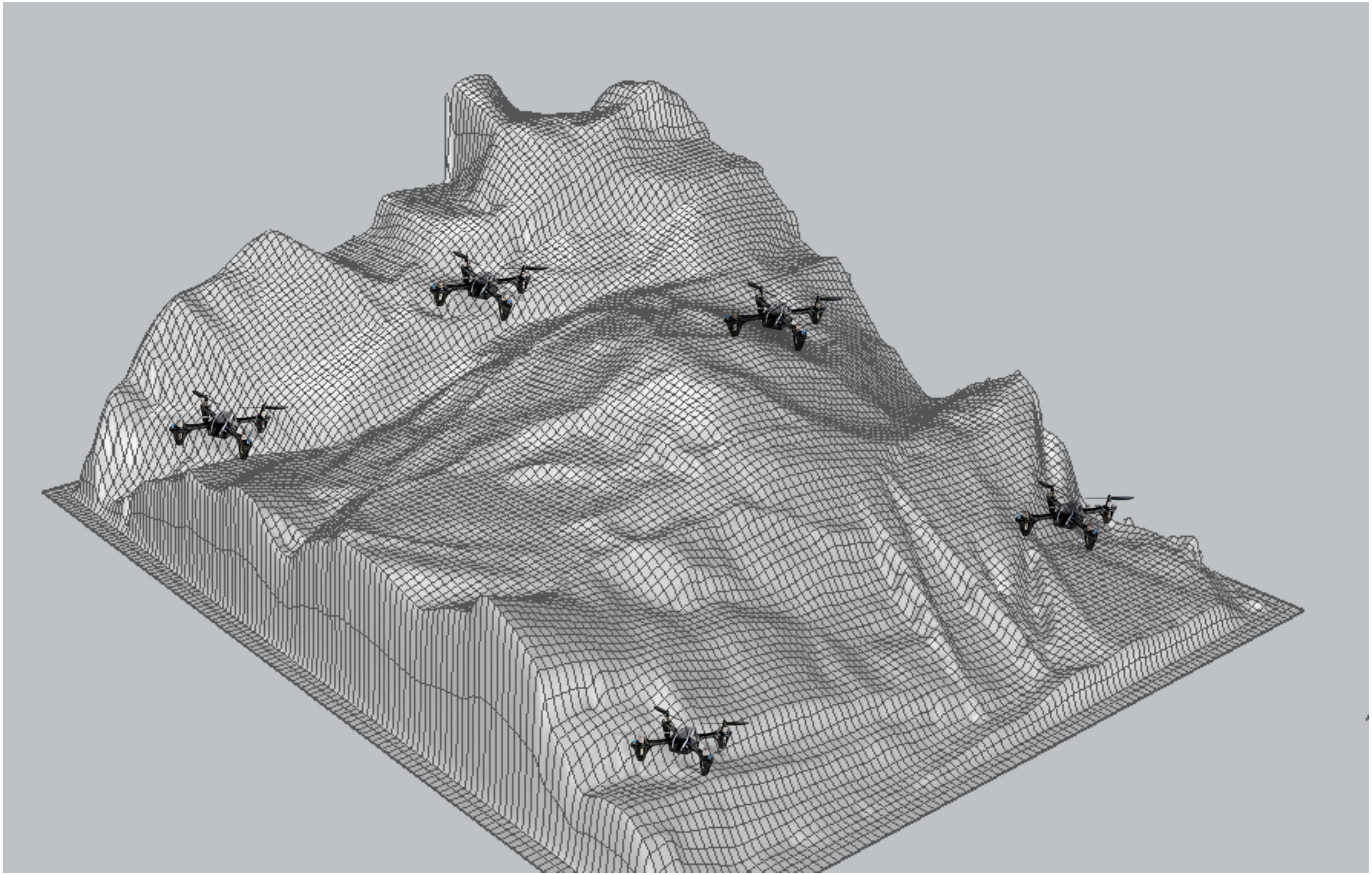


image source: http://2.bp.blogspot.com/-eUnhAo8SfFQ/VDaLij2_csI/AAAAAAAABPM/847rgT6VpQE/s1600/Screen%2BShot%2B2014-05-01%2Bat%2B00.02.54.png
https://www.dronerush.com/best-drones-1977/drone.bmp

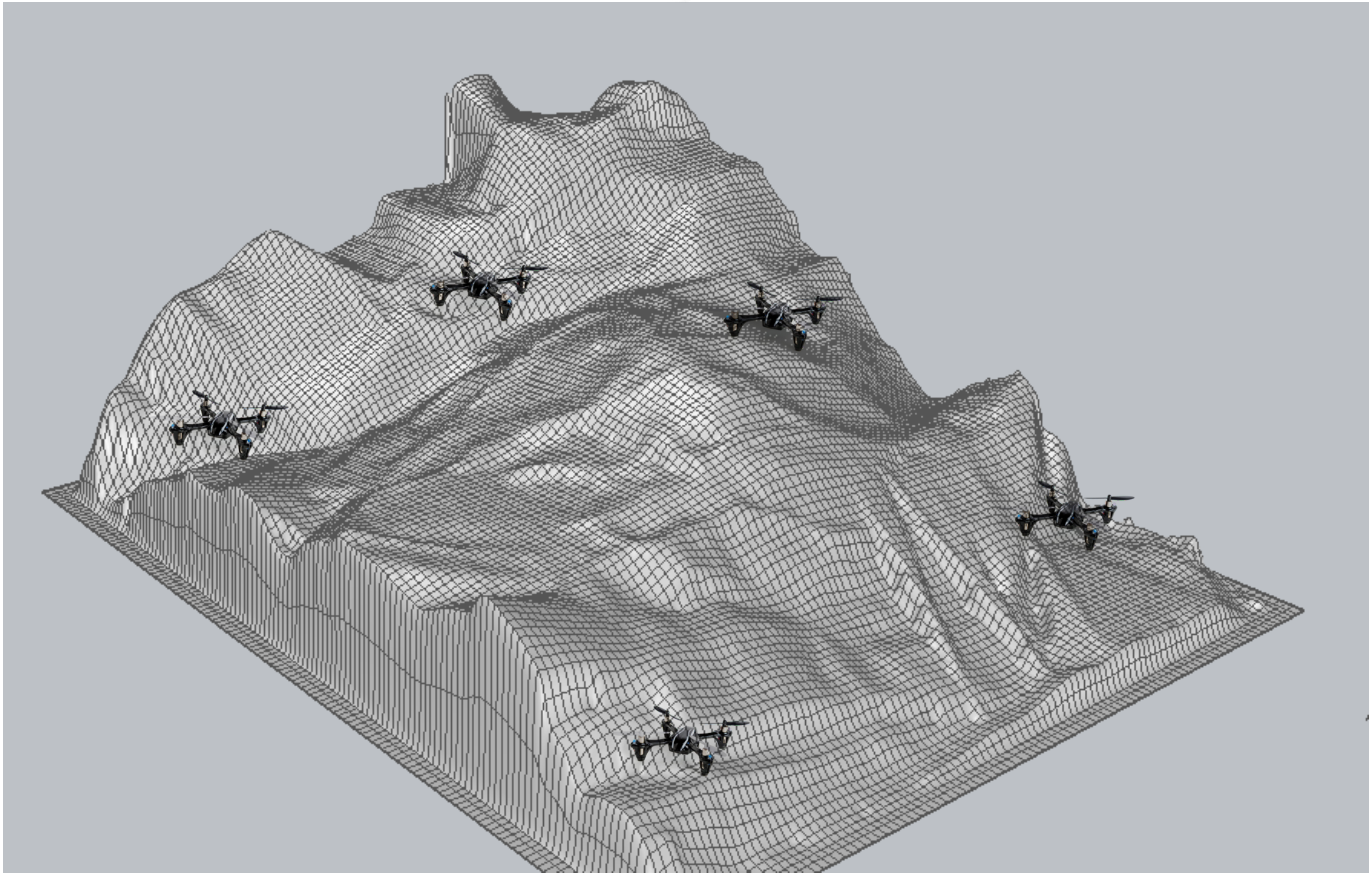


image source: http://2.bp.blogspot.com/-eUnhAo8SfFQ/VDaLij2_csI/AAAAAAAABPM/847rgT6VpQE/s1600/Screen%2BShot%2B2014-05-01%2Bat%2B00.02.54.png
https://www.dronerush.com/best-drones-1977/drone.bmp

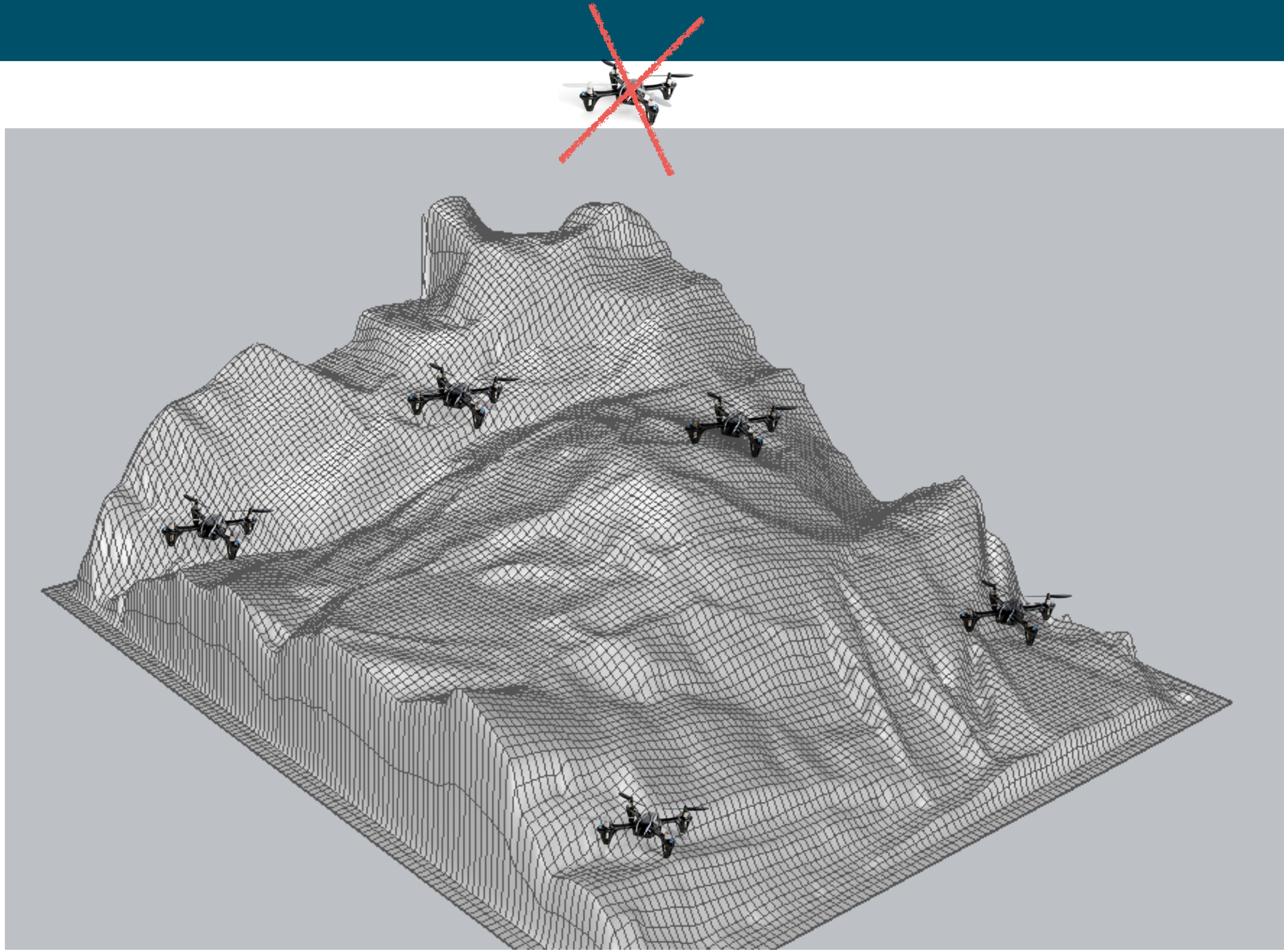


image source: http://2.bp.blogspot.com/-eUnhAo8SfFQ/VDaLij2_csI/AAAAAAAABPM/847rgT6VpQE/s1600/Screen%2BShot%2B2014-05-01%2Bat%2B00.02.54.png
https://www.dronerush.com/best-drones-1977/drone.bmp

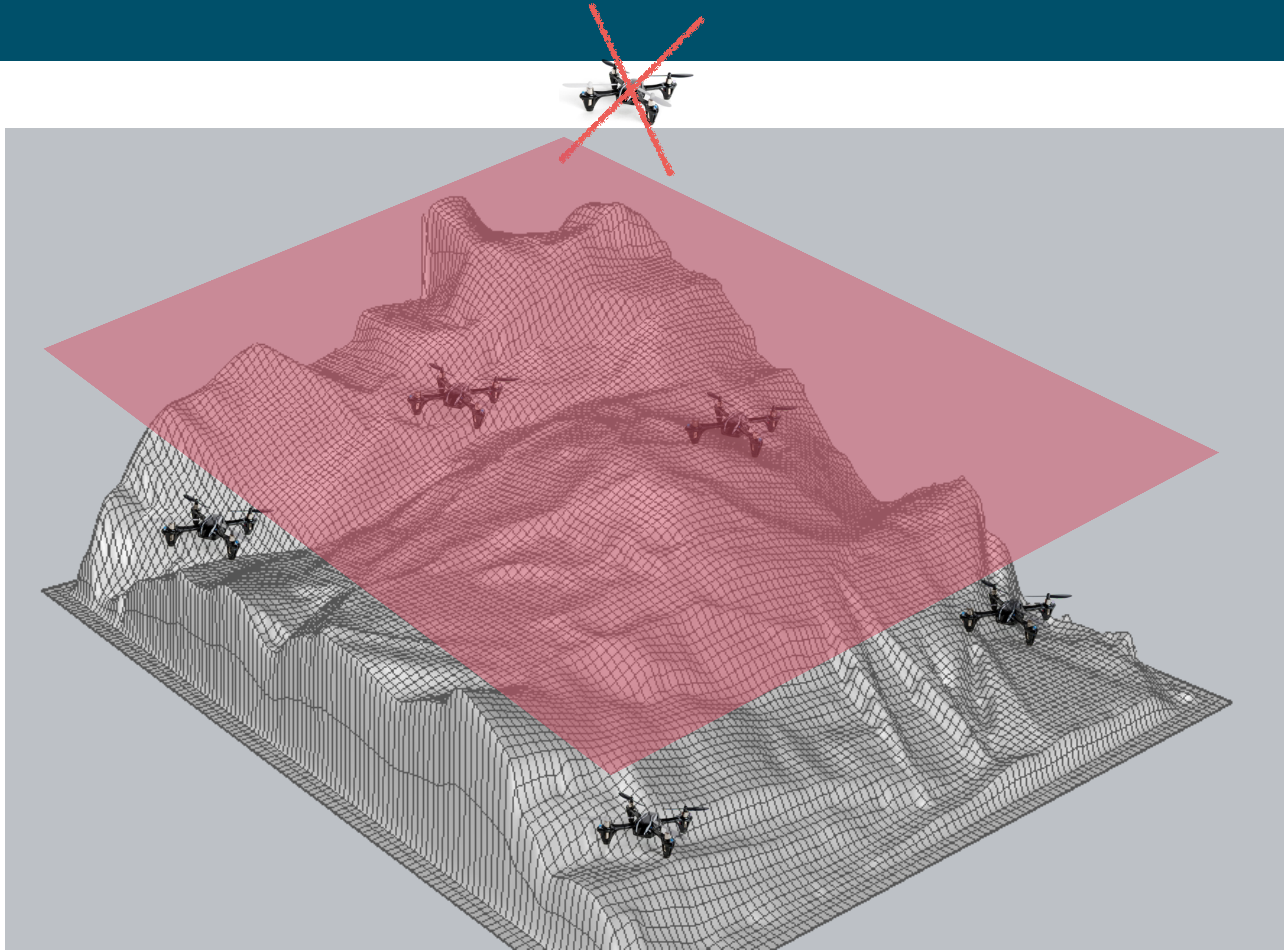


image source: http://2.bp.blogspot.com/-eUnhAo8SfFQ/VDaLij2_csI/AAAAAAAABPM/847rgT6VpQE/s1600/Screen%2BShot%2B2014-05-01%2Bat%2B00.02.54.png
https://www.dronerush.com/best-drones-1977/drone.bmp

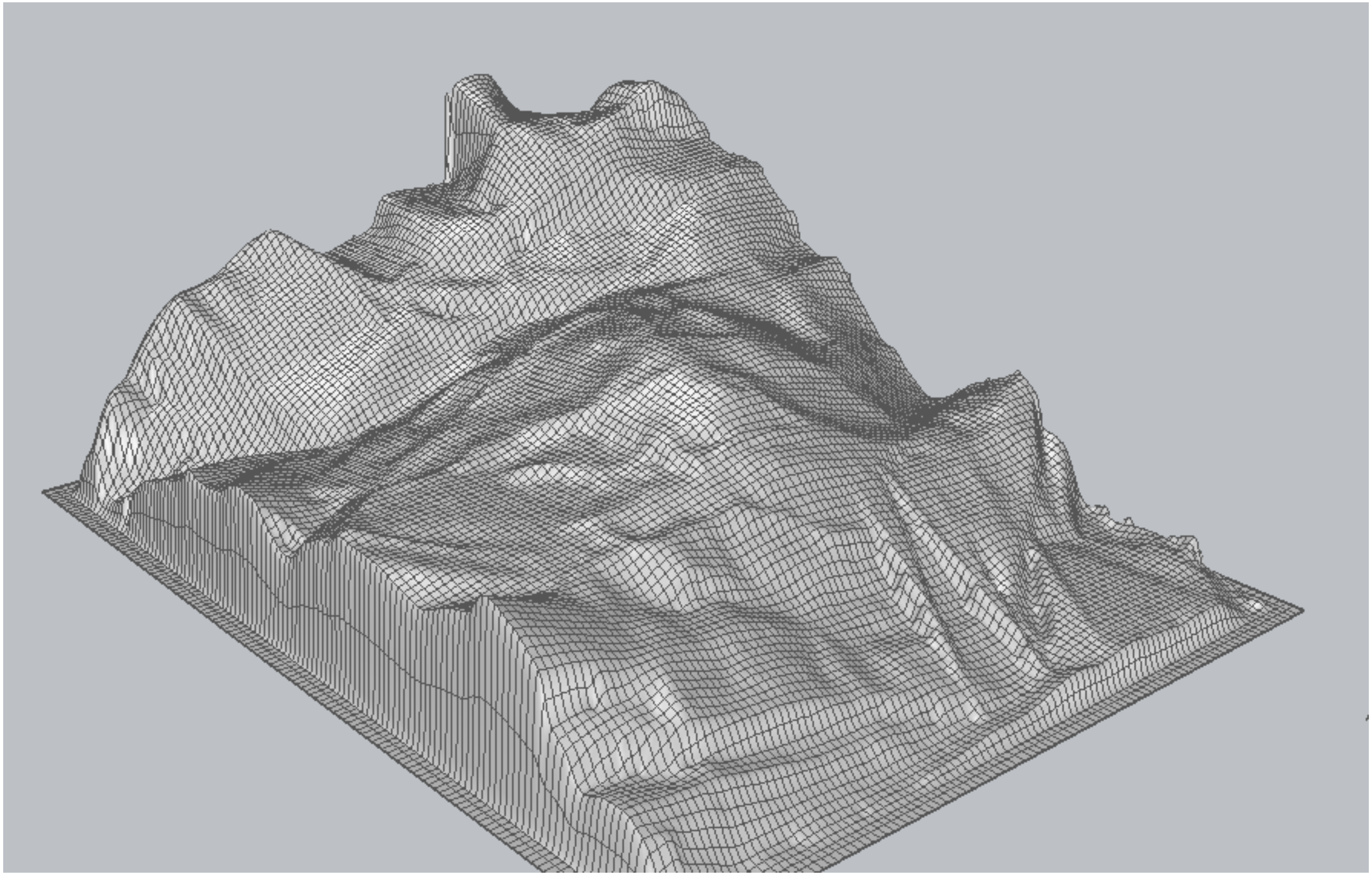


image source: http://2.bp.blogspot.com/-eUnhAo8SfFQ/VDaLij2_csI/AAAAAAAABPM/847rgT6VpQE/s1600/Screen%2BShot%2B2014-05-01%2Bat%2B00.02.54.png

2.5D

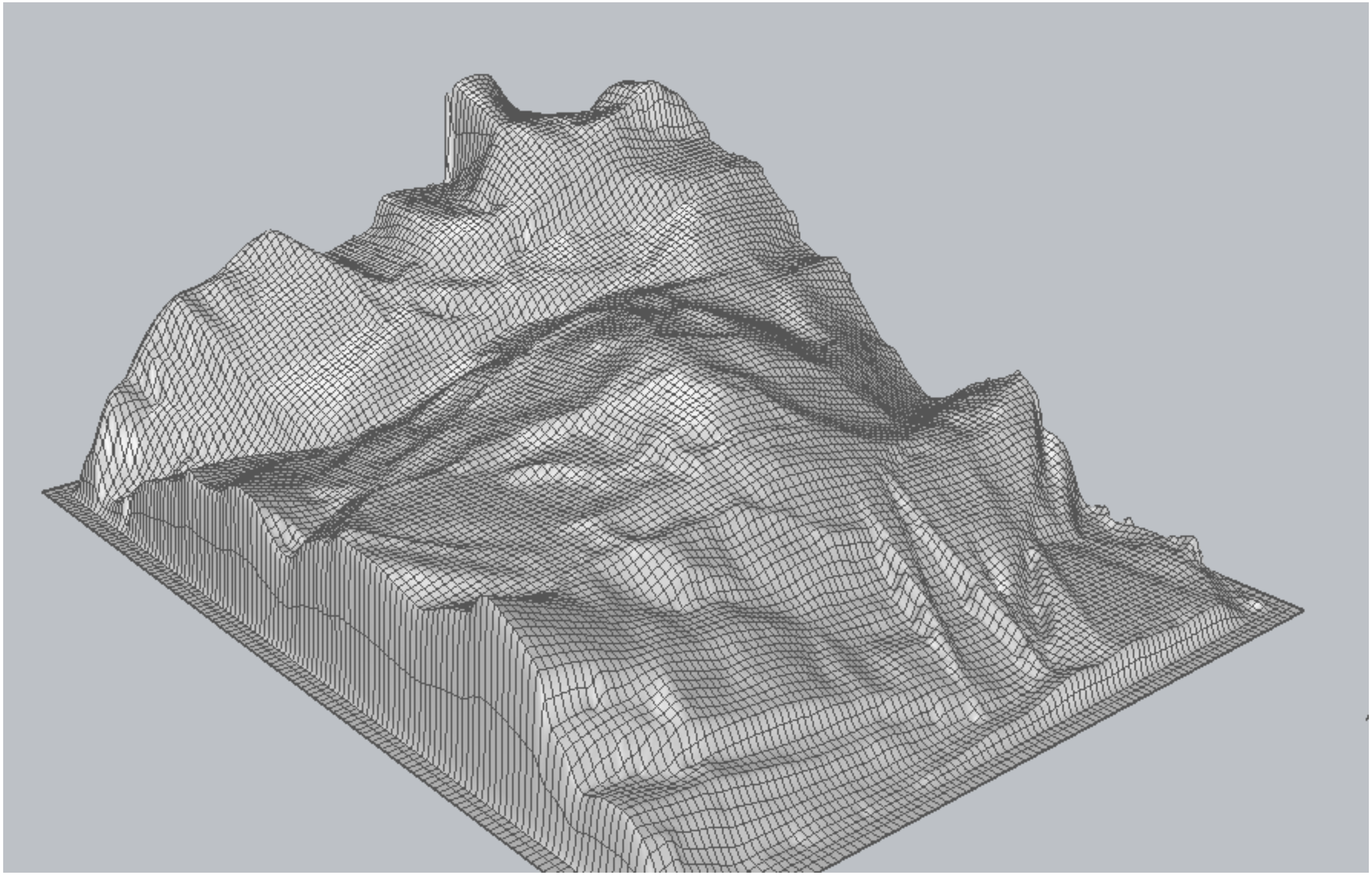
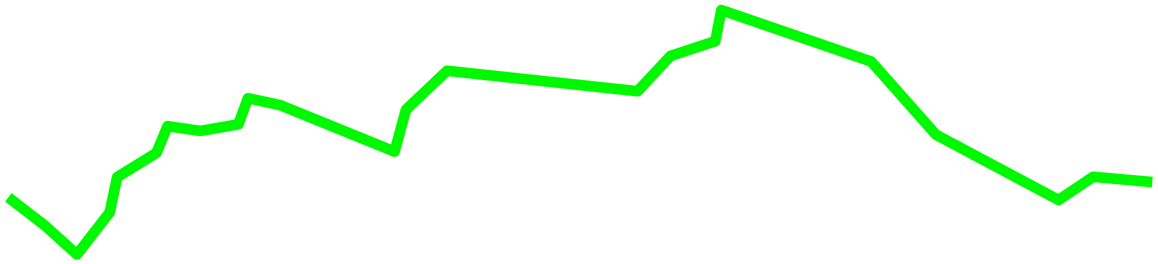


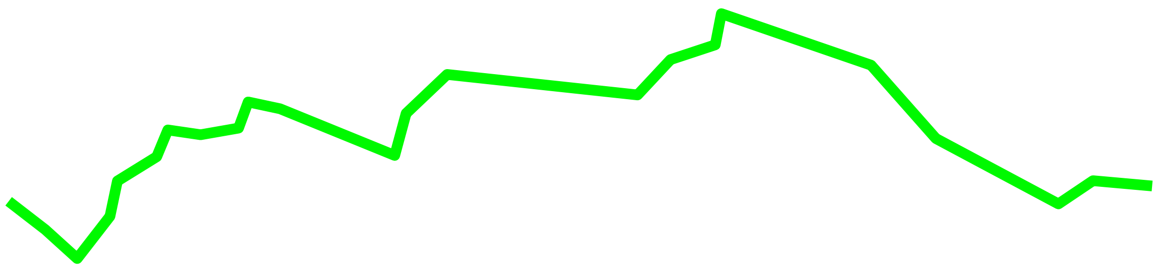
image source: http://2.bp.blogspot.com/-eUnhAo8SfFQ/VDaLij2_csI/AAAAAAAABPM/847rgT6VpQE/s1600/Screen%2BShot%2B2014-05-01%2Bat%2B00.02.54.png

2.5D

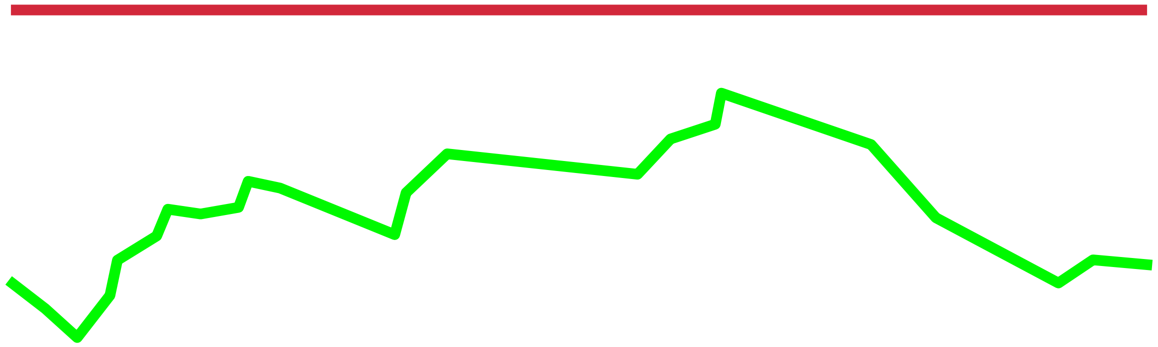


1.5D

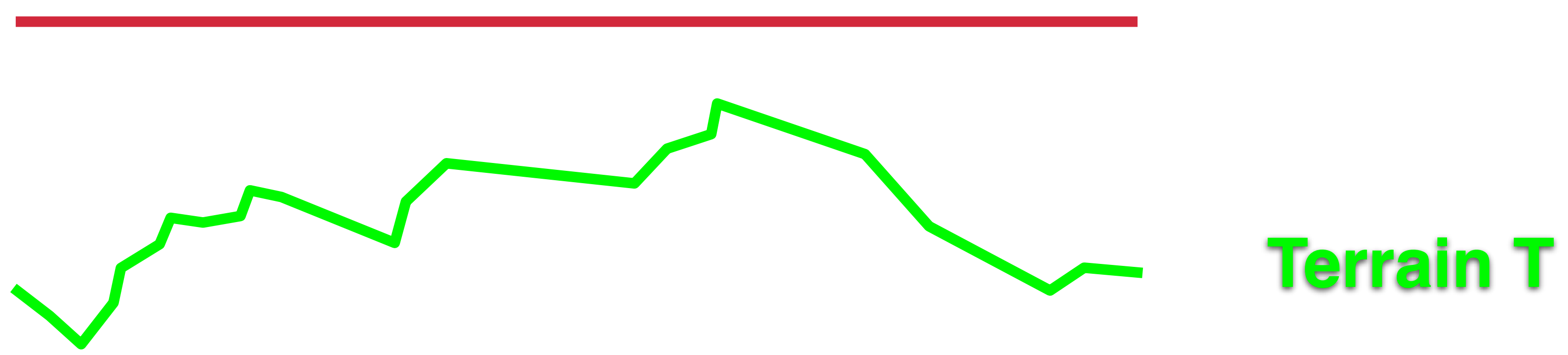
Altitude Terrain Guarding Problem



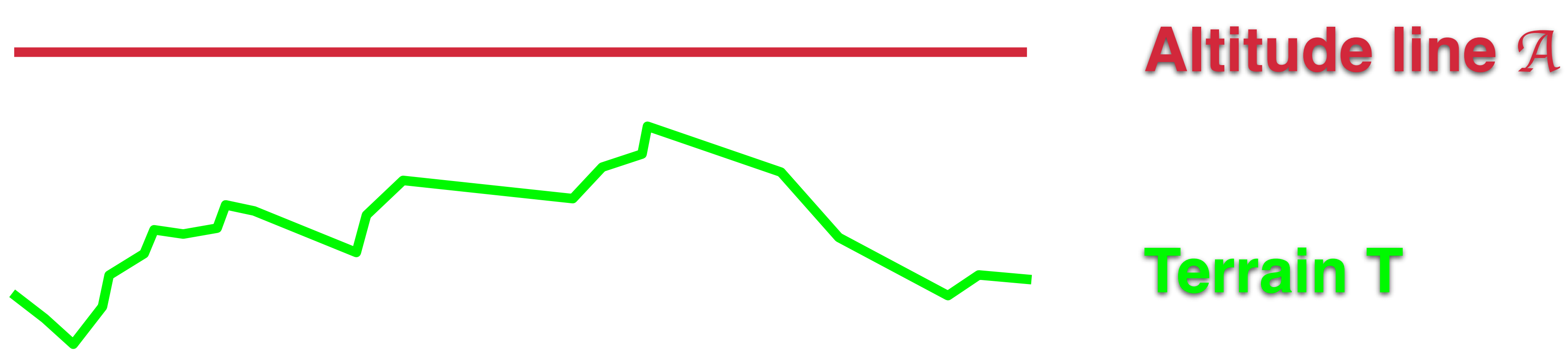
Altitude Terrain Guarding Problem



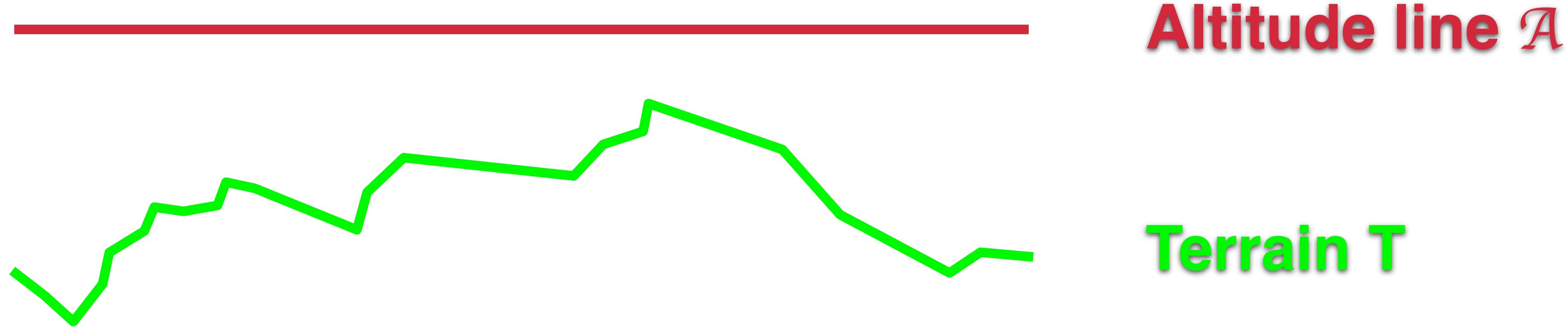
Altitude Terrain Guarding Problem



Altitude Terrain Guarding Problem

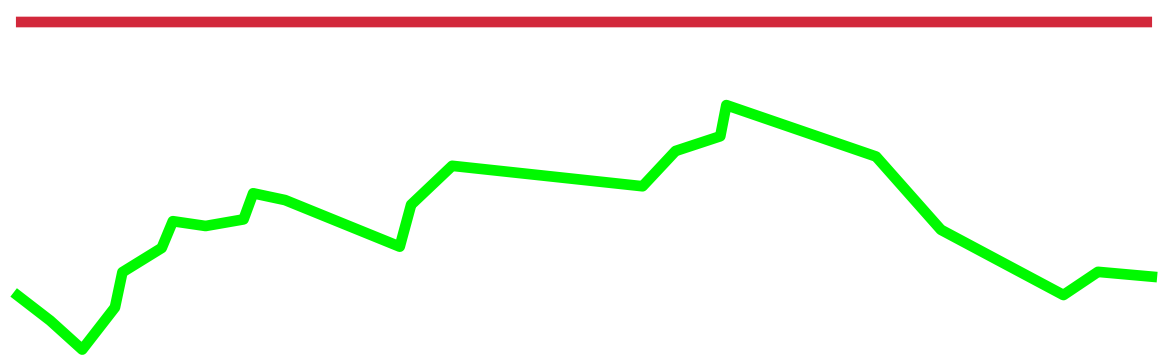


Altitude Terrain Guarding Problem



x-monotone chain of line segments in R^2
defined by its vertices $V(T) = \{ v_1 , \ldots , v_n\}$

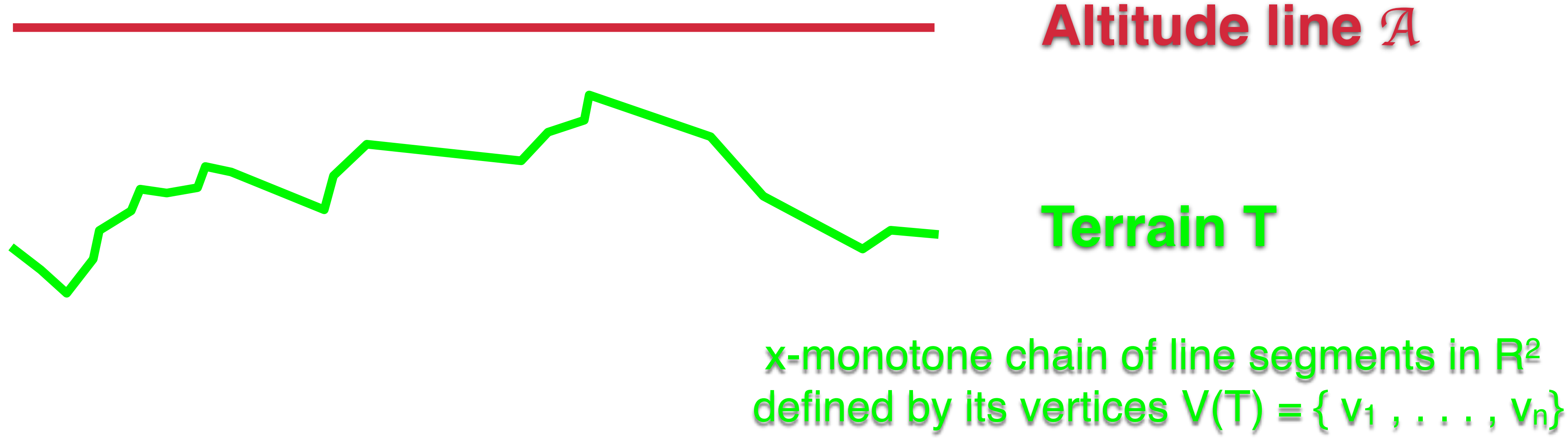
Altitude Terrain Guarding Problem



x-monotone chain of line segments in R^2
defined by its vertices $V(T) = \{ v_1 , . . . , v_n \}$

Monotonicity

Altitude Terrain Guarding Problem

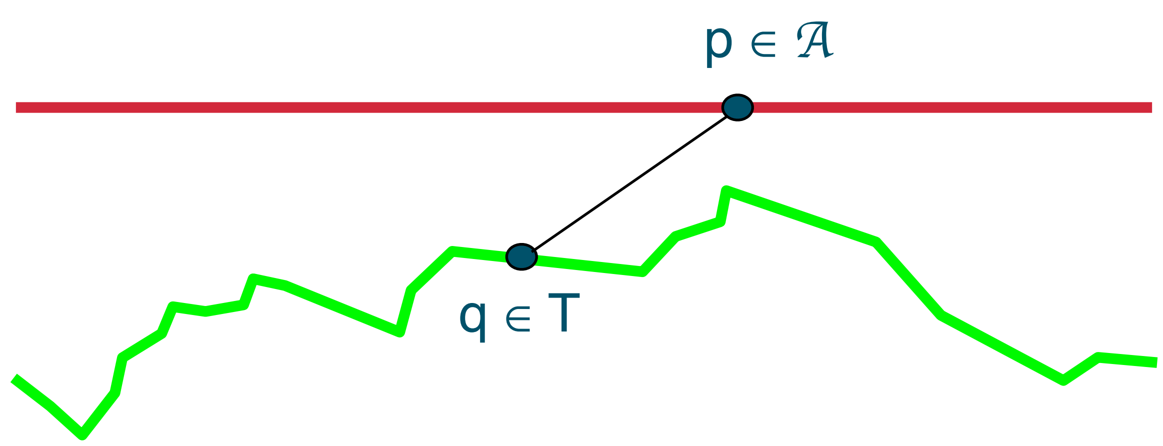


Monotonicity

➡ Points on T are totally ordered wrt to x-coordinate: $p<q$.

Altitude Terrain Guarding Problem

A point p ∈ $\mathcal{A}$ sees or covers q ∈ T if and only if pq is nowhere below T (i.e. pq lies on or above T).



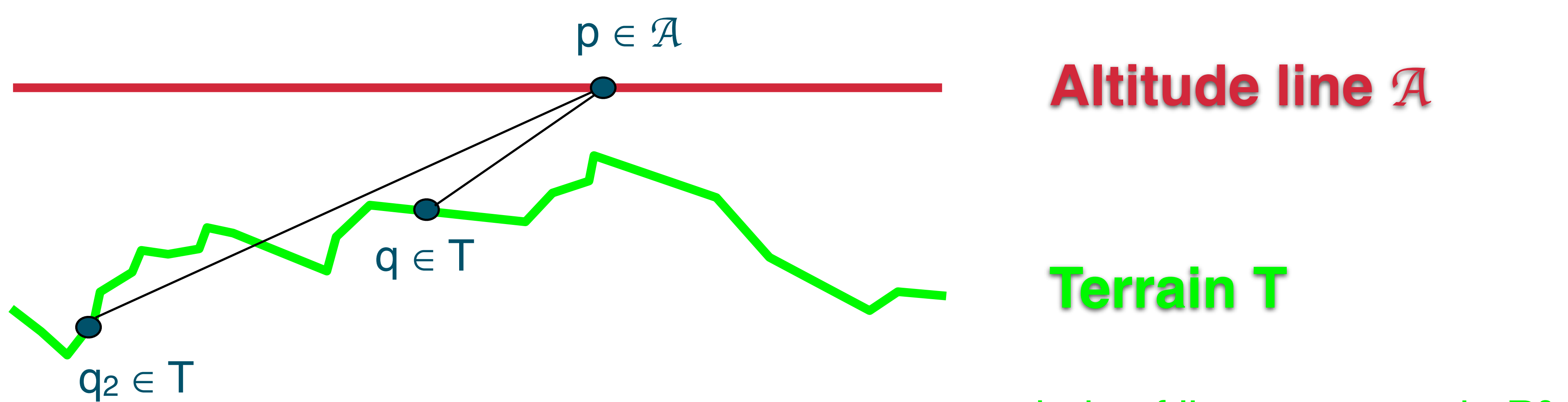
Altitude line $\mathcal{A}$

Terrain T

x-monotone chain of line segments in R^2
defined by its vertices V(T) = { v_1 , . . . , v_n}

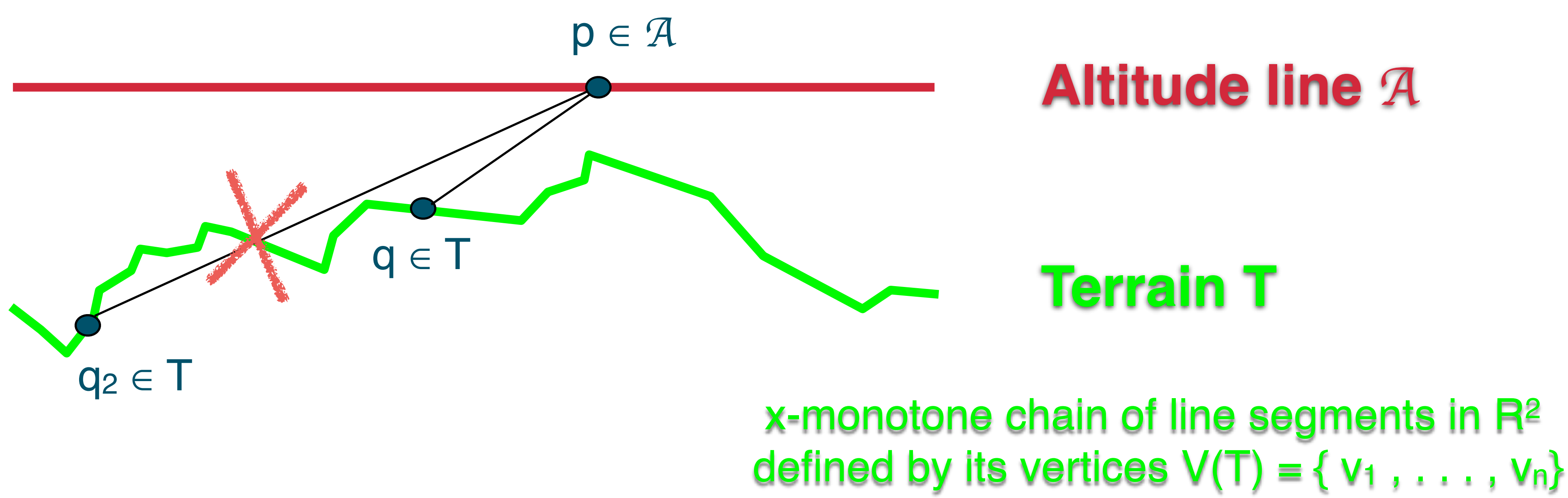
Altitude Terrain Guarding Problem

A point $p \in \mathcal{A}$ sees or covers $q \in T$ if and only if pq is nowhere below T (i.e. pq lies on or above T).



Altitude Terrain Guarding Problem

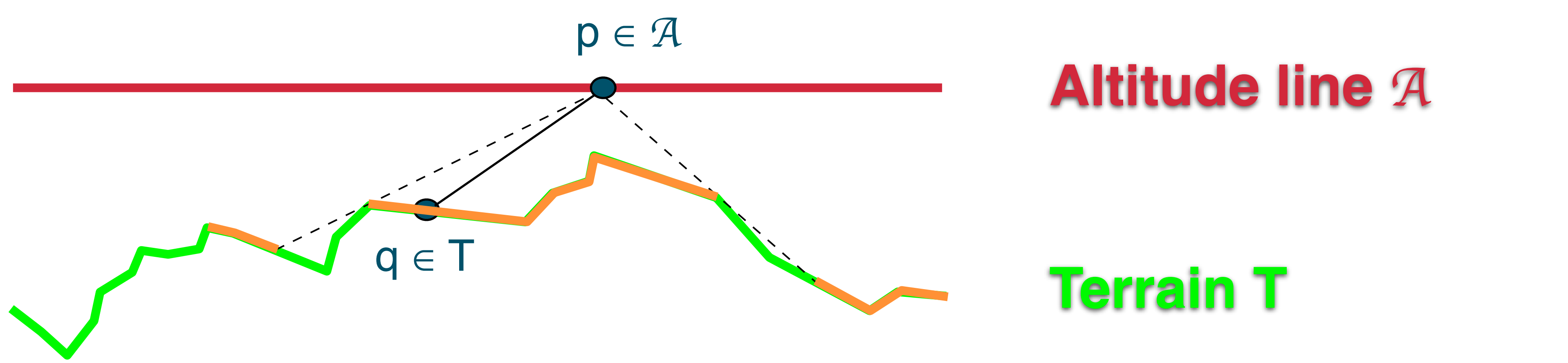
A point $p \in \mathcal{A}$ sees or covers $q \in T$ if and only if pq is nowhere below T (i.e. pq lies on or above T).



Altitude Terrain Guarding Problem

A point $p \in \mathcal{A}$ sees or covers $q \in T$ if and only if pq is nowhere below T (i.e. pq lies on or above T).

$V_T(p)$ is the visibility region of p with $V_T(p) := \{q \in T \mid p \text{ sees } q\}$.

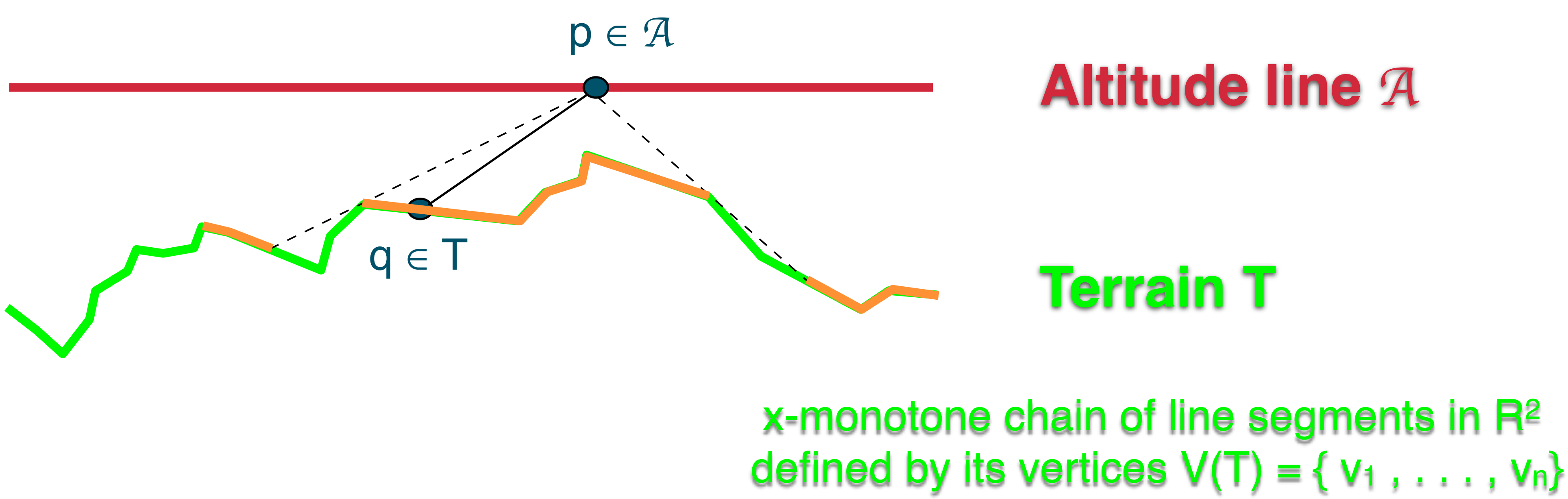


Altitude Terrain Guarding Problem

A point $p \in \mathcal{A}$ sees or covers $q \in T$ if and only if pq is nowhere below T (i.e. pq lies on or above T).

$V_T(p)$ is the visibility region of p with $V_T(p) := \{q \in T \mid p \text{ sees } q\}$.

For $G \subseteq A$ $V_T(G) := \bigcup_{g \in G} V_T(g)$.

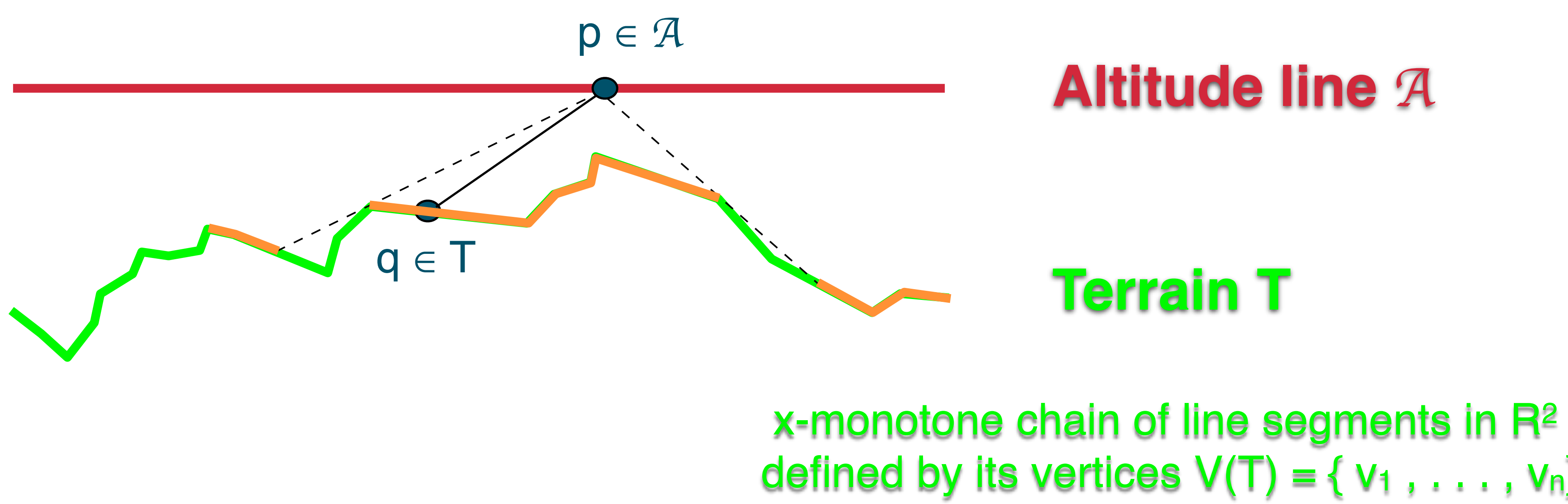


Altitude Terrain Guarding Problem

A point $p \in \mathcal{A}$ sees or covers $q \in T$ if and only if pq is nowhere below T (i.e. pq lies on or above T).

$V_T(p)$ is the visibility region of p with $V_T(p) := \{q \in T \mid p \text{ sees } q\}$.

For $G \subseteq A$ $V_T(G) := \bigcup_{g \in G} V_T(g)$.



Altitude Terrain Guarding Problem (ATGP) ATGP(T,$\mathcal{A}$)

Given: a terrain T and an altitude line $\mathcal{A}$.

A minimum set of guards that see all of T.

(Formally: A guard set $G \subset \mathcal{A}$ is optimal w.r.t. ATGP(T,$\mathcal{A}$) if G is feasible, that is, $T \subseteq V_T(G)$, and

$|G| = OPT(T,\mathcal{A}) := \min\{|C| \mid C \subset \mathcal{A}$ is feasible w.r.t. ATGP(T,$\mathcal{A}$)$\}$.)

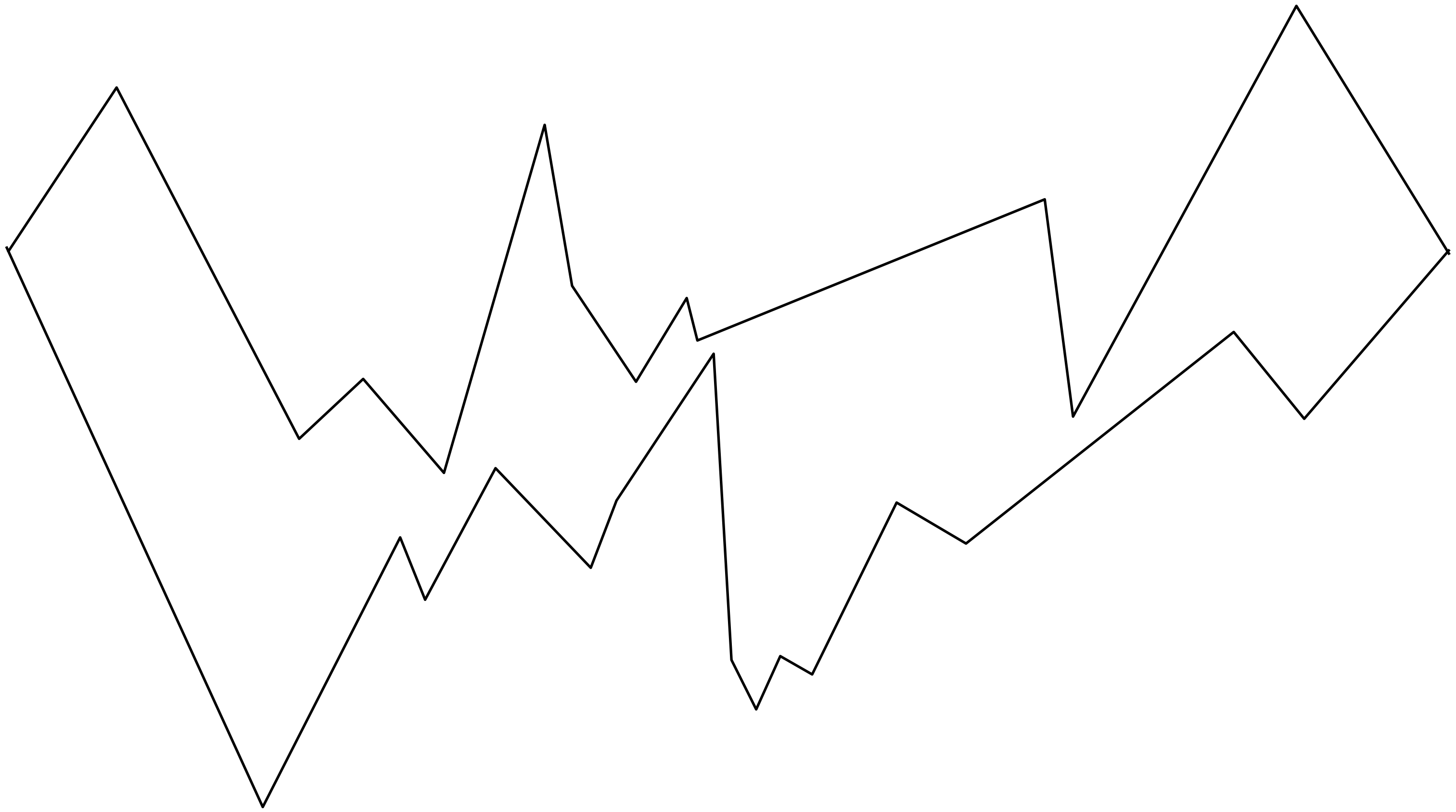
AGP in uni-monotone polygons

uni-monotone polygon?

AGP in uni-monotone polygons

uni-monotone polygon?

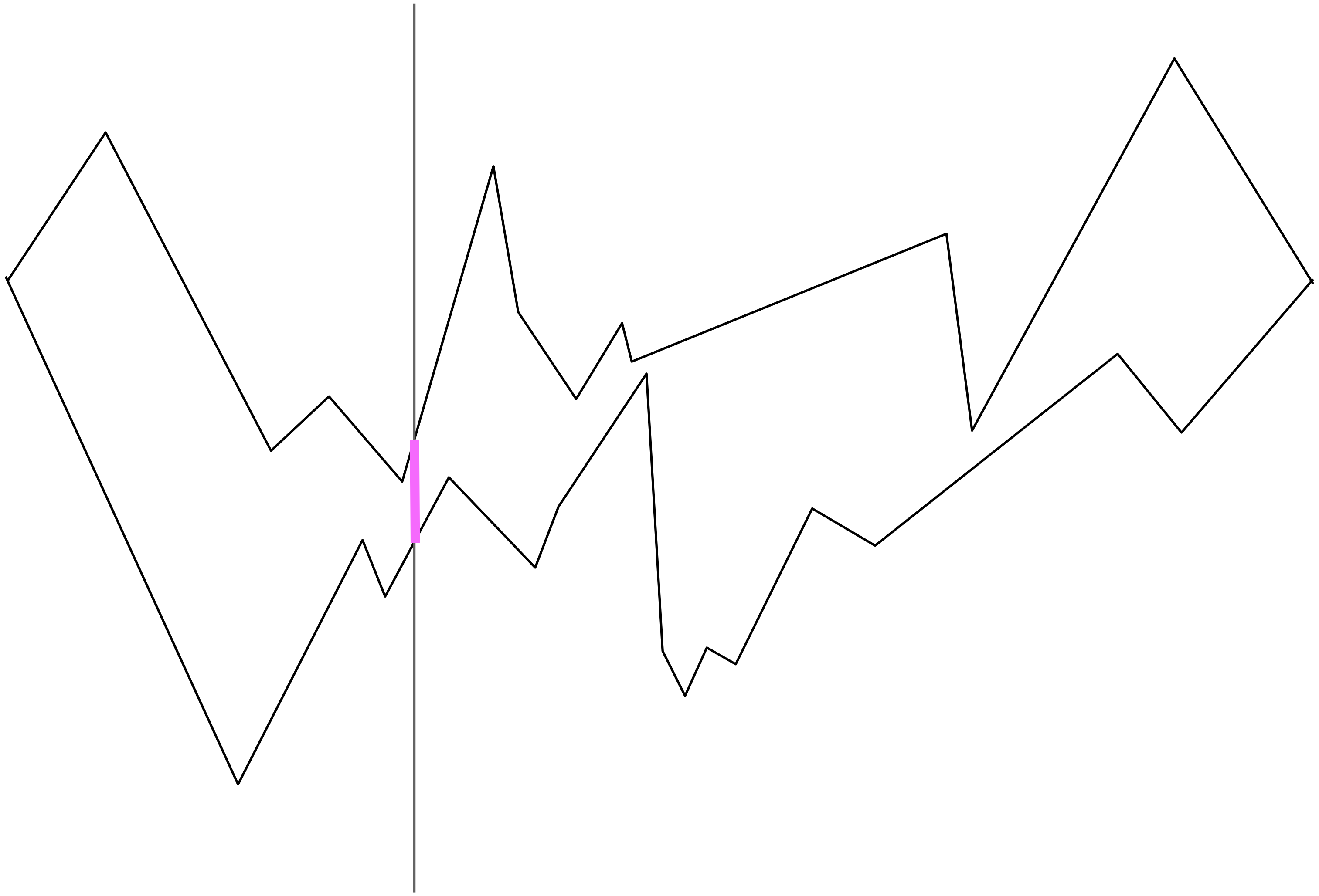
monotone polygon



AGP in uni-monotone polygons

uni-monotone polygon?

monotone polygon



AGP in uni-monotone polygons

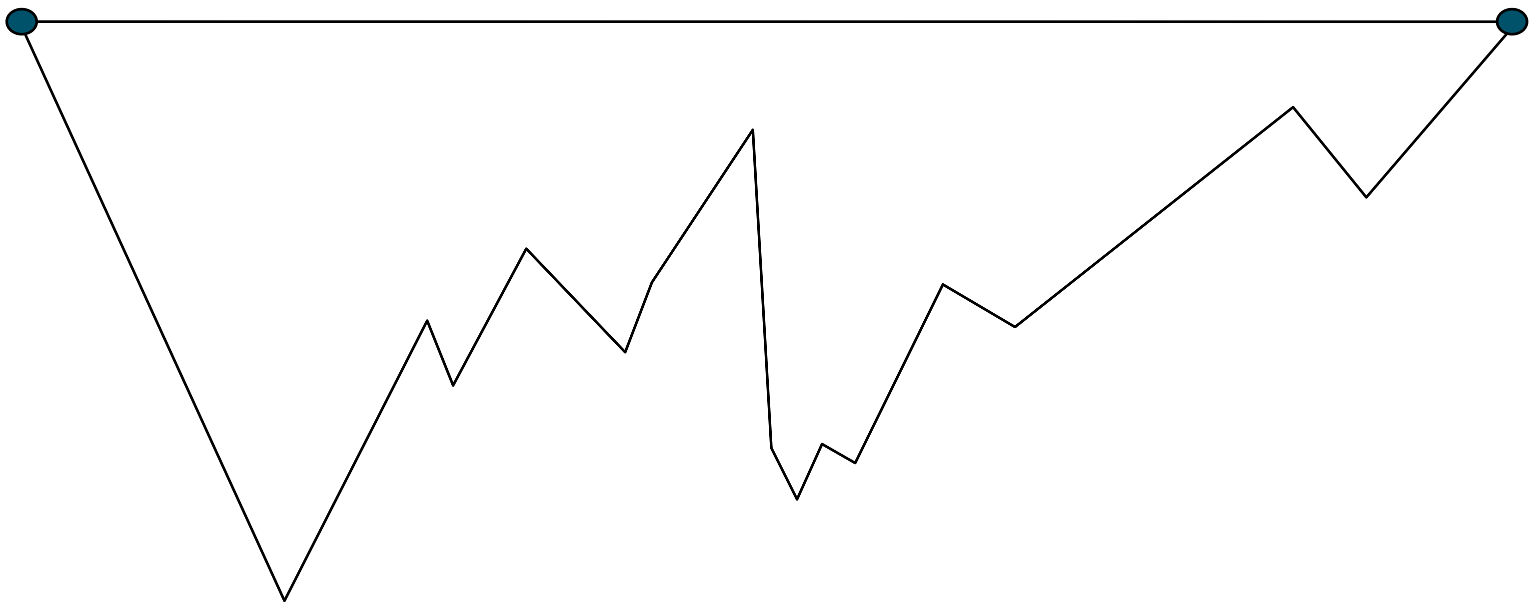
uni-monotone polygon?

monotone polygon

AGP in uni-monotone polygons

uni-monotone polygon?

uni-monotone polygon



AGP in uni-monotone polygons

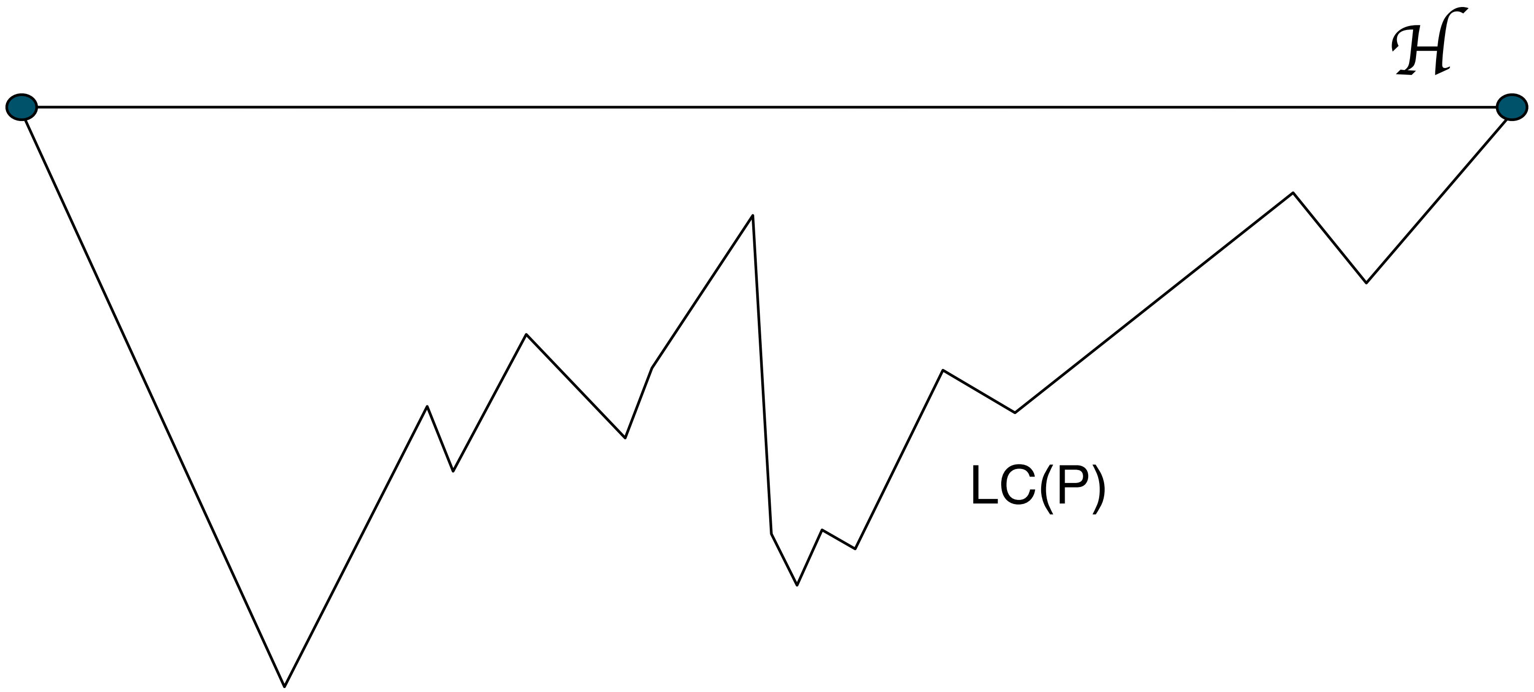
uni-monotone polygon?

uni-monotone polygon

AGP in uni-monotone polygons

uni-monotone polygon?

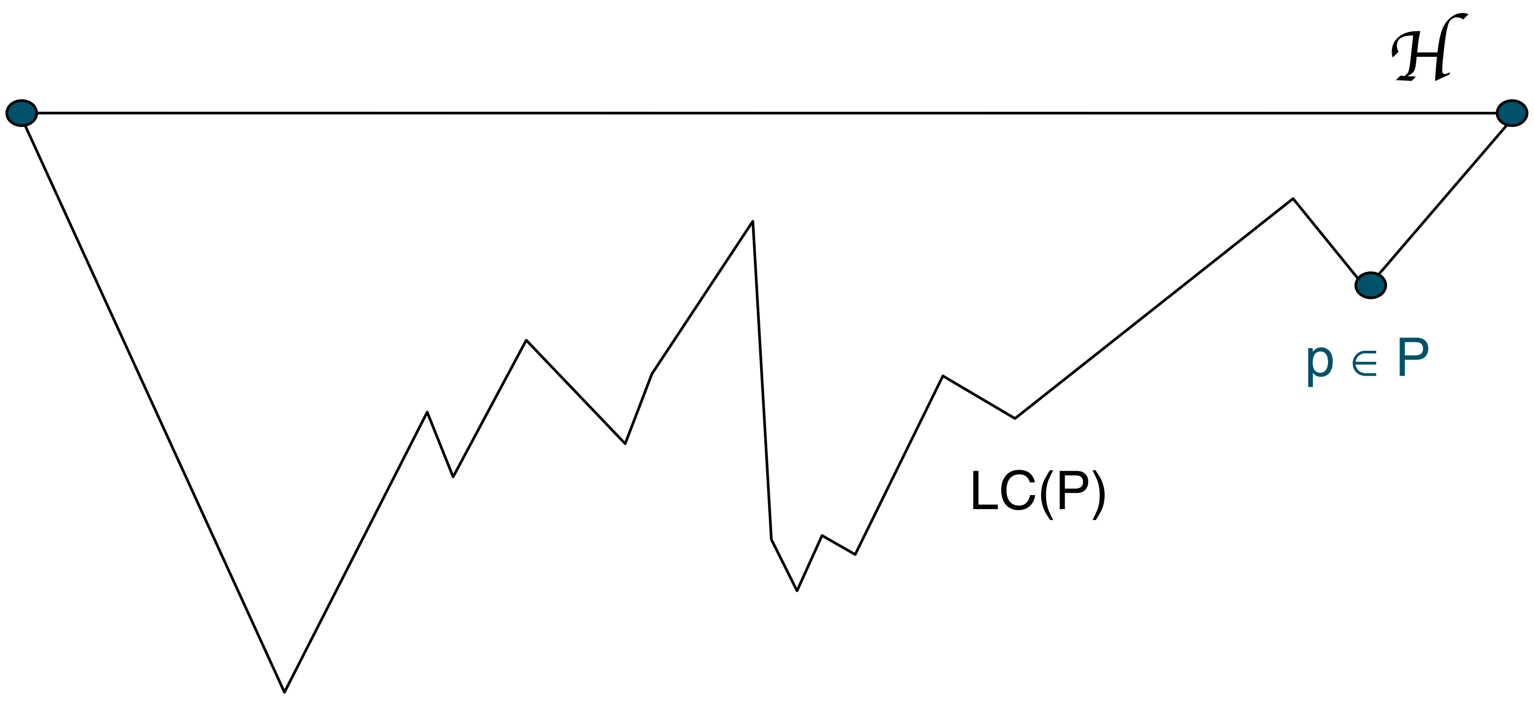
uni-monotone polygon



AGP in uni-monotone polygons

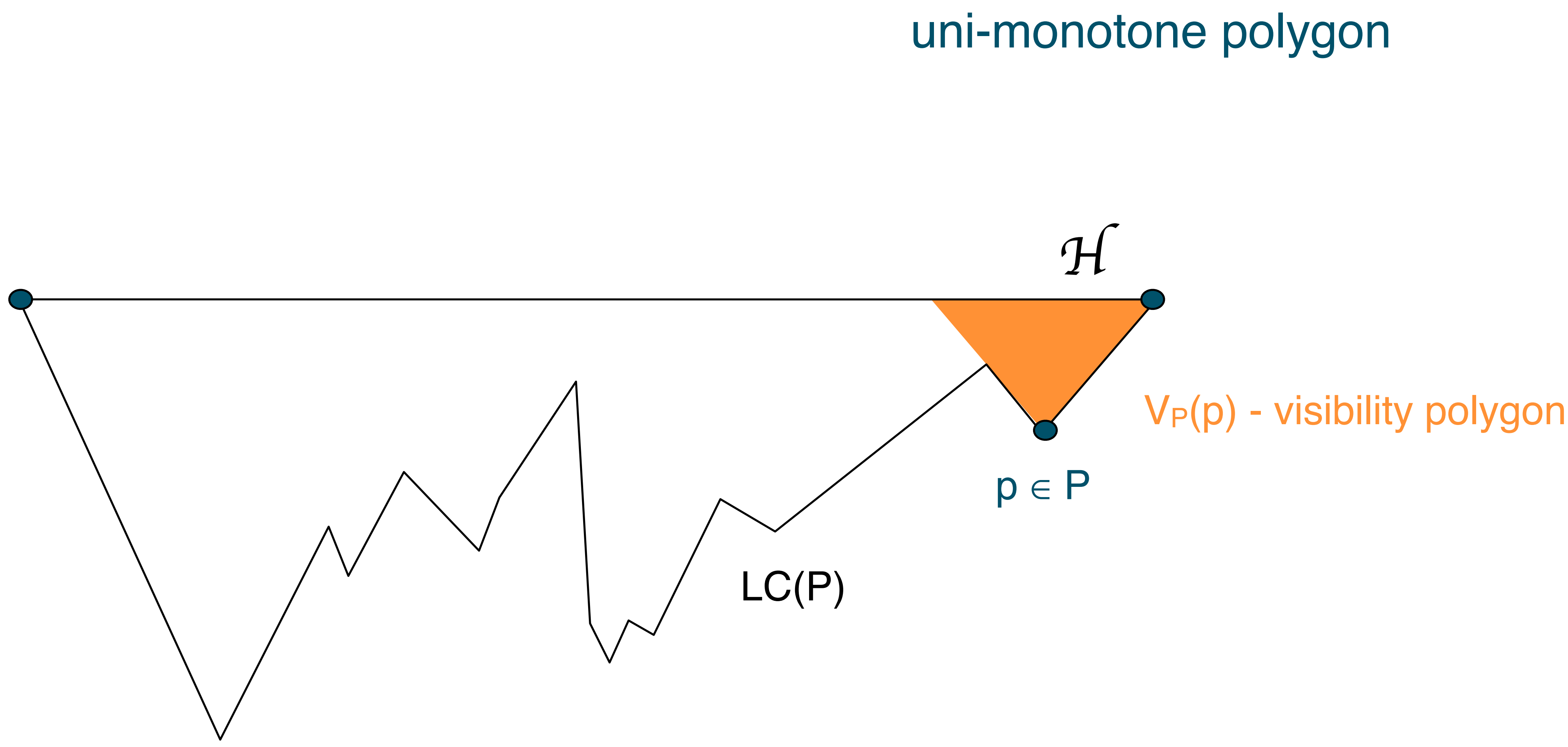
uni-monotone polygon?

uni-monotone polygon



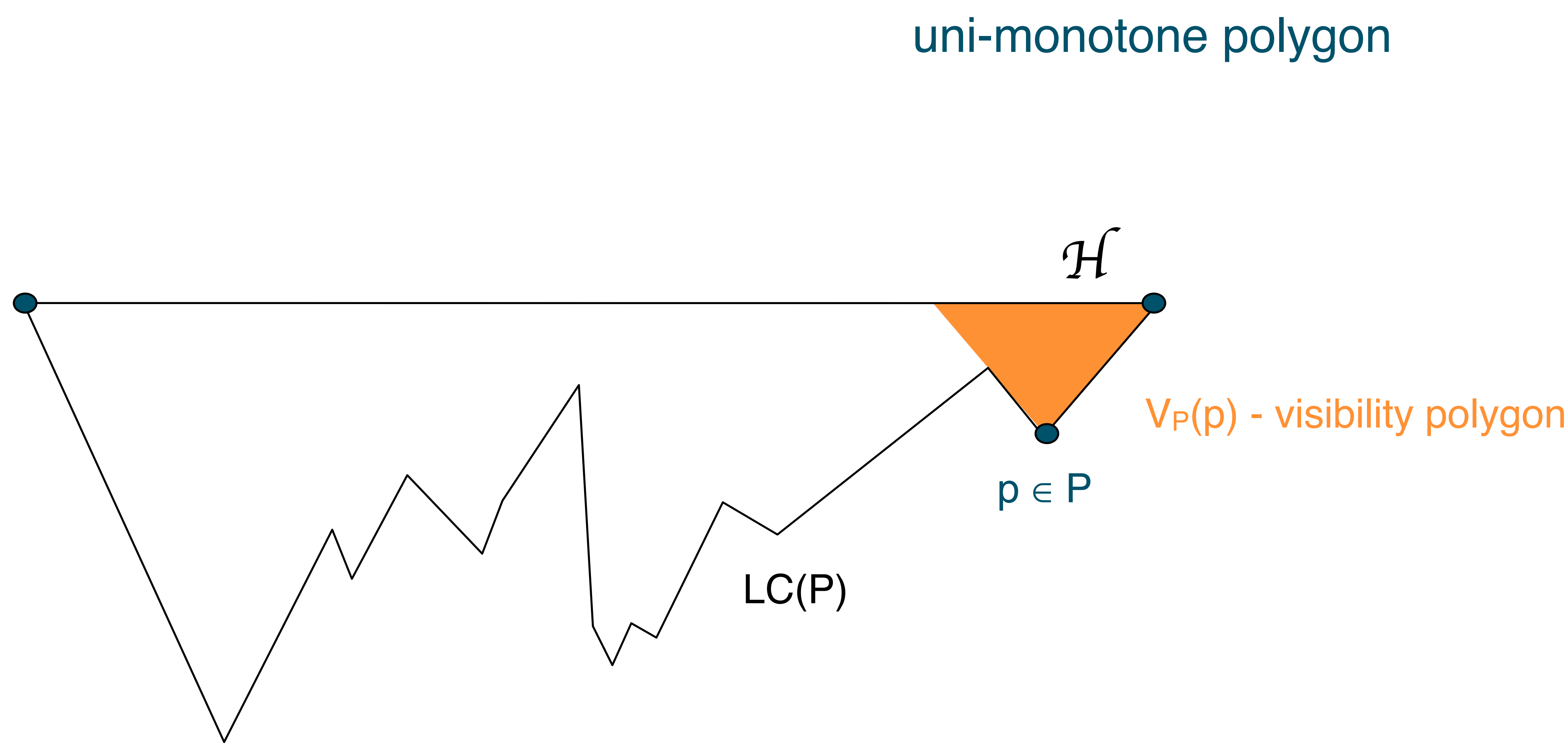
AGP in uni-monotone polygons

uni-monotone polygon?



AGP in uni-monotone polygons

uni-monotone polygon?



Formally:

Art Gallery Problem (AGP) AGP(G,W)

Given: a polygon P and sets of guard candidates and points to cover $G, W \subseteq P$.

A minimum guard set $C \subseteq G$ that covers W (that is, $W \subseteq V_P(C)$).

We want to solve AGP(P,P).

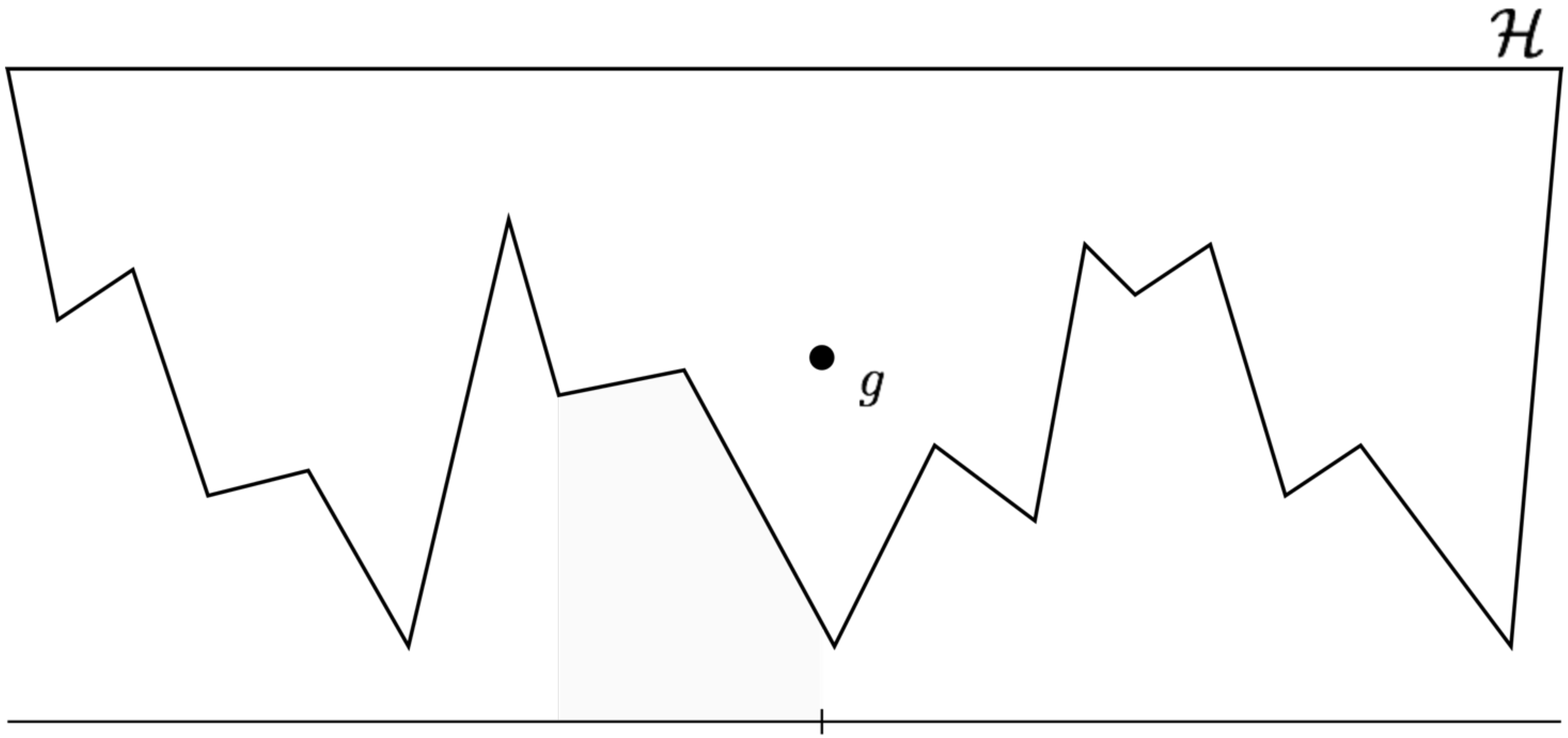
AGP in uni-monotone polygons

If we want to solve the AGP for a uni-monotone polygon, w.l.o.g. we can restrict our guards to be located on $\mathcal{H}$.

AGP in uni-monotone polygons

If we want to solve the AGP for a uni-monotone polygon, w.l.o.g. we can restrict our guards to be located on $\mathcal{H}$.

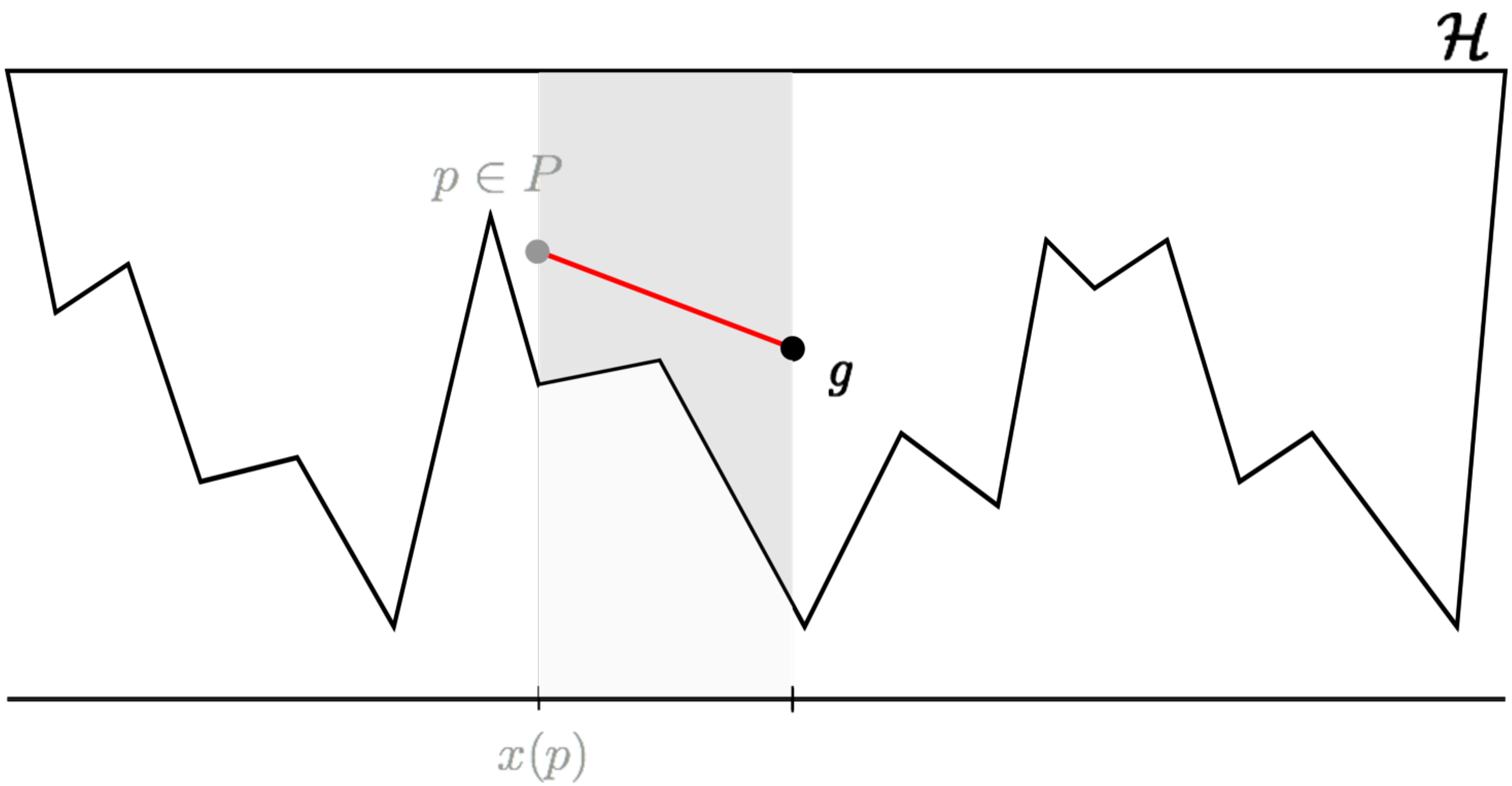
Proof: Consider any optimal guard set G, let g∈G be a guard not located on $\mathcal{H}$



AGP in uni-monotone polygons

If we want to solve the AGP for a uni-monotone polygon, w.l.o.g. we can restrict our guards to be located on $\mathcal{H}$.

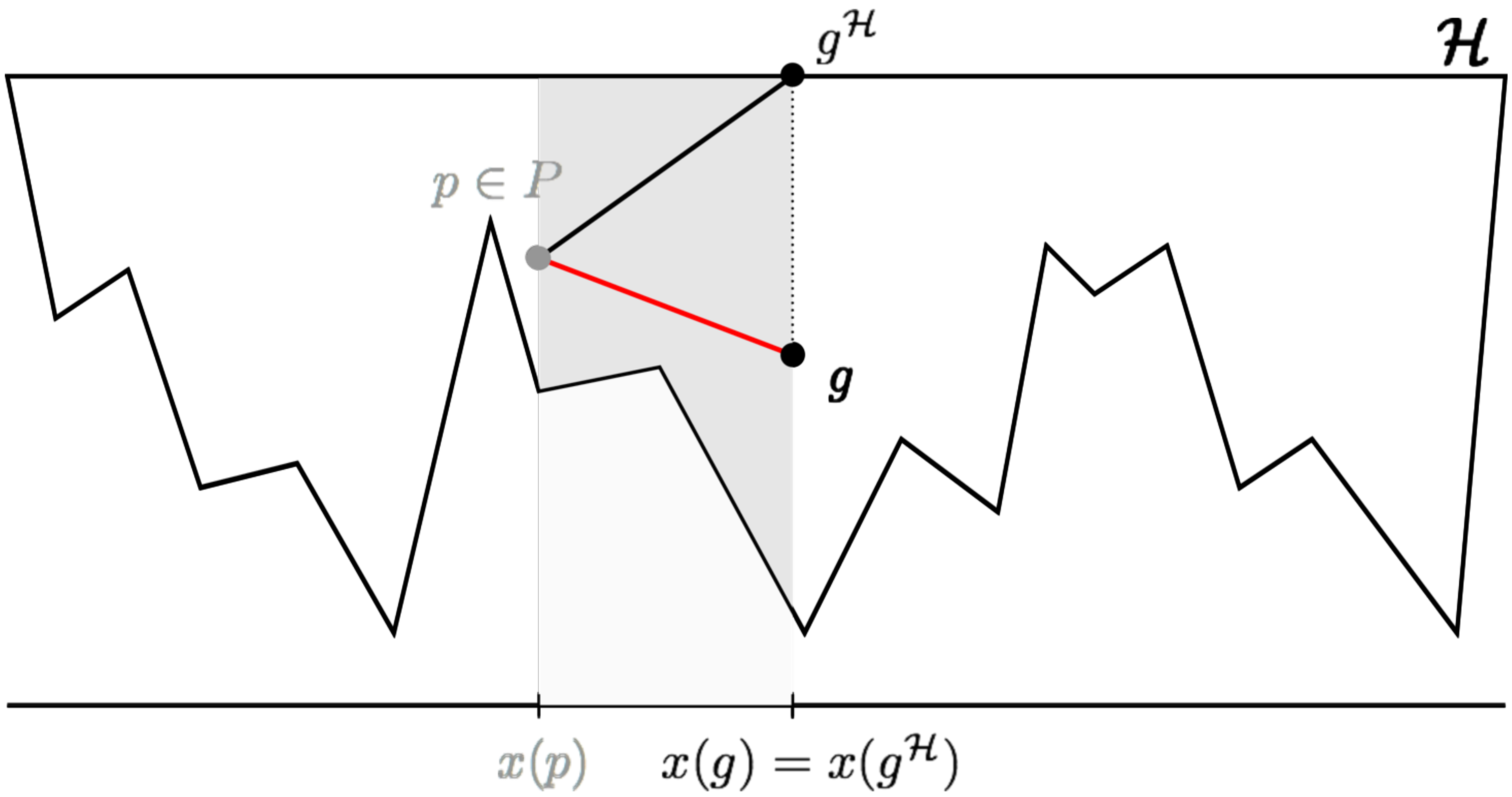
Proof: Consider any optimal guard set G, let g∈G be a guard not located on $\mathcal{H}$



AGP in uni-monotone polygons

If we want to solve the AGP for a uni-monotone polygon, w.l.o.g. we can restrict our guards to be located on $\mathcal{H}$.

Proof: Consider any optimal guard set G, let g∈G be a guard not located on $\mathcal{H}$



AGP in uni-monotone polygons

Let P be a uni-monotone polygon, let G be a guard set with $g \in \mathcal{H}$ $\forall g \in G$ that covers $LC(P)$, that is, $LC(P) \subset V_P(G)$. Then G covers all of P, that is, $P \subseteq V_P(G)$.

AGP in uni-monotone polygons

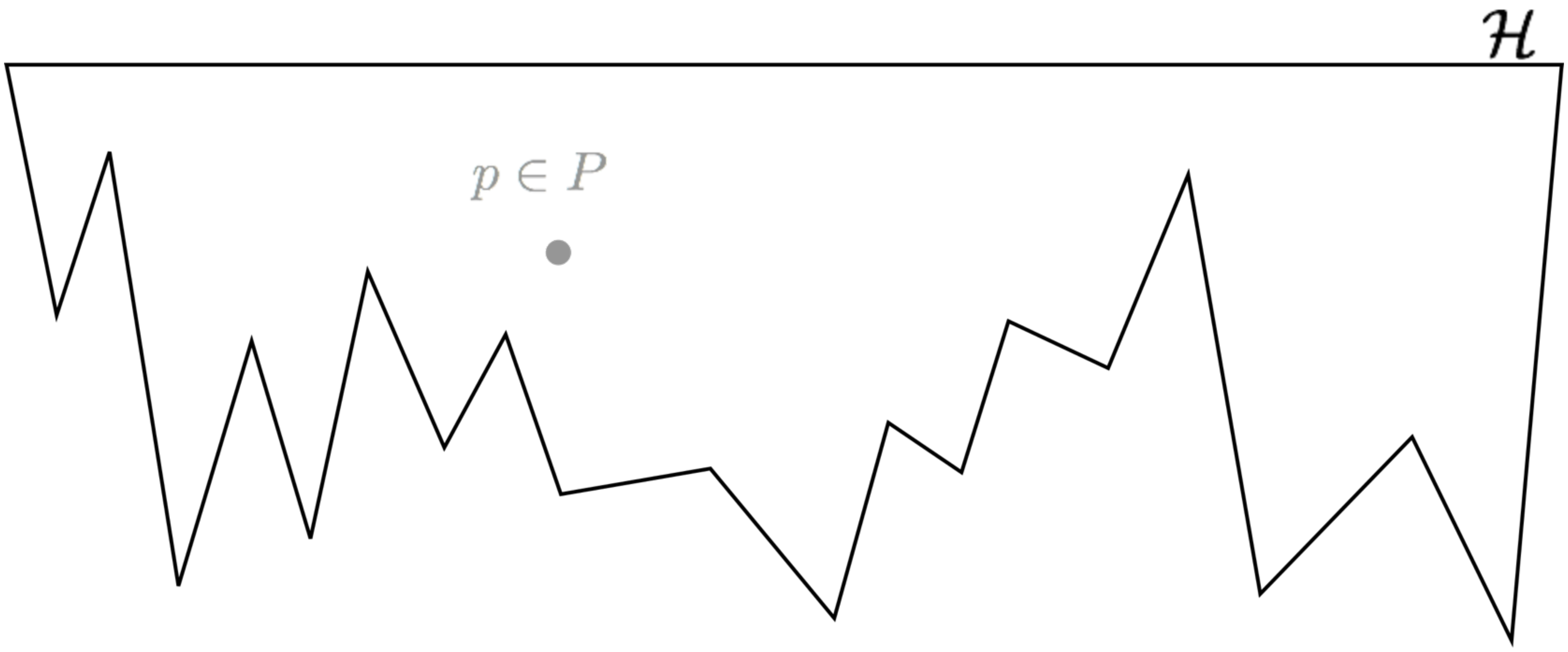
Let P be a uni-monotone polygon, let G be a guard set with $g \in \mathcal{H}\ \forall g \in G$ that covers $LC(P)$, that is, $LC(P) \subset V_P(G)$. Then G covers all of P, that is, $P \subseteq V_P(G)$.

Proof: Assume $p \in P$, $p \notin LC(P)$, $p \notin V_P(G)$

AGP in uni-monotone polygons

Let P be a uni-monotone polygon, let G be a guard set with $g \in \mathcal{H}\ \forall g \in G$ that covers $LC(P)$, that is, $LC(P) \subset V_P(G)$. Then G covers all of P, that is, $P \subseteq V_P(G)$.

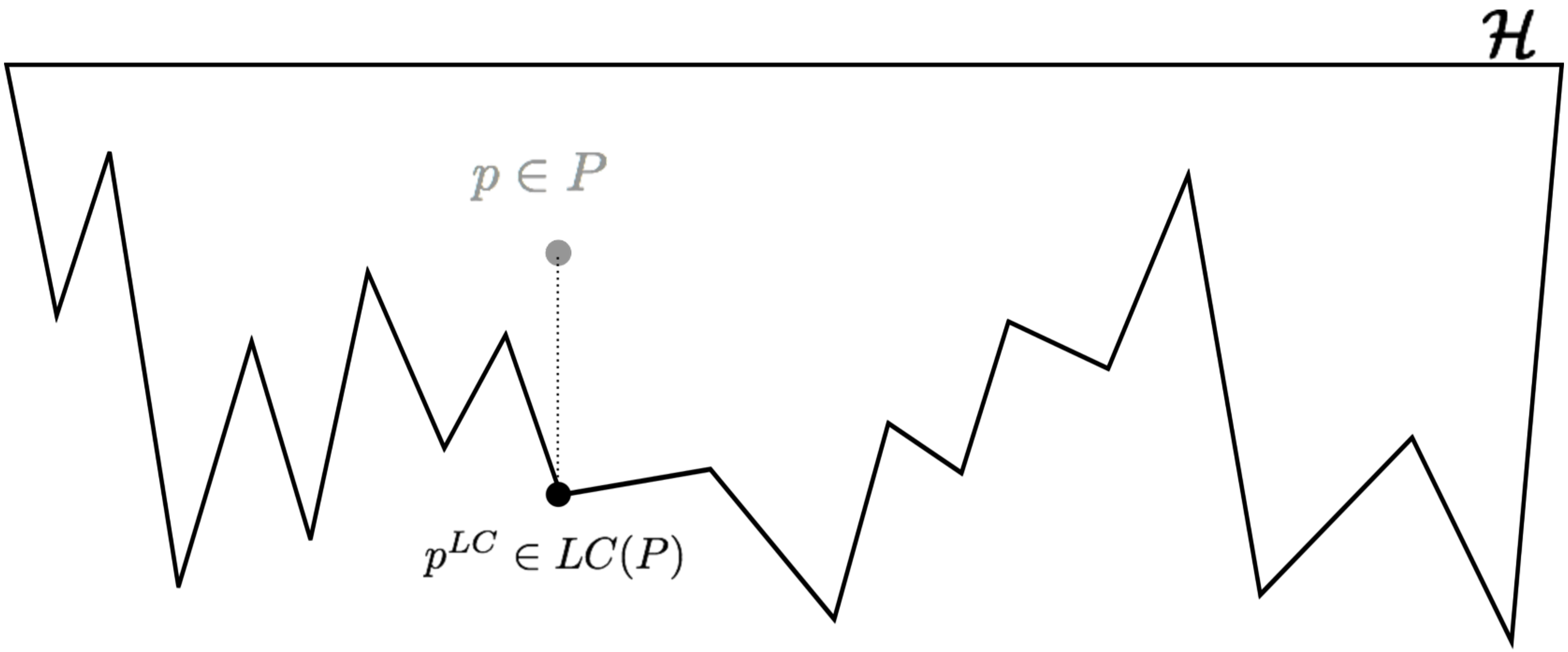
Proof: Assume $p \in P$, $p \notin LC(P)$, $p \notin V_P(G)$



AGP in uni-monotone polygons

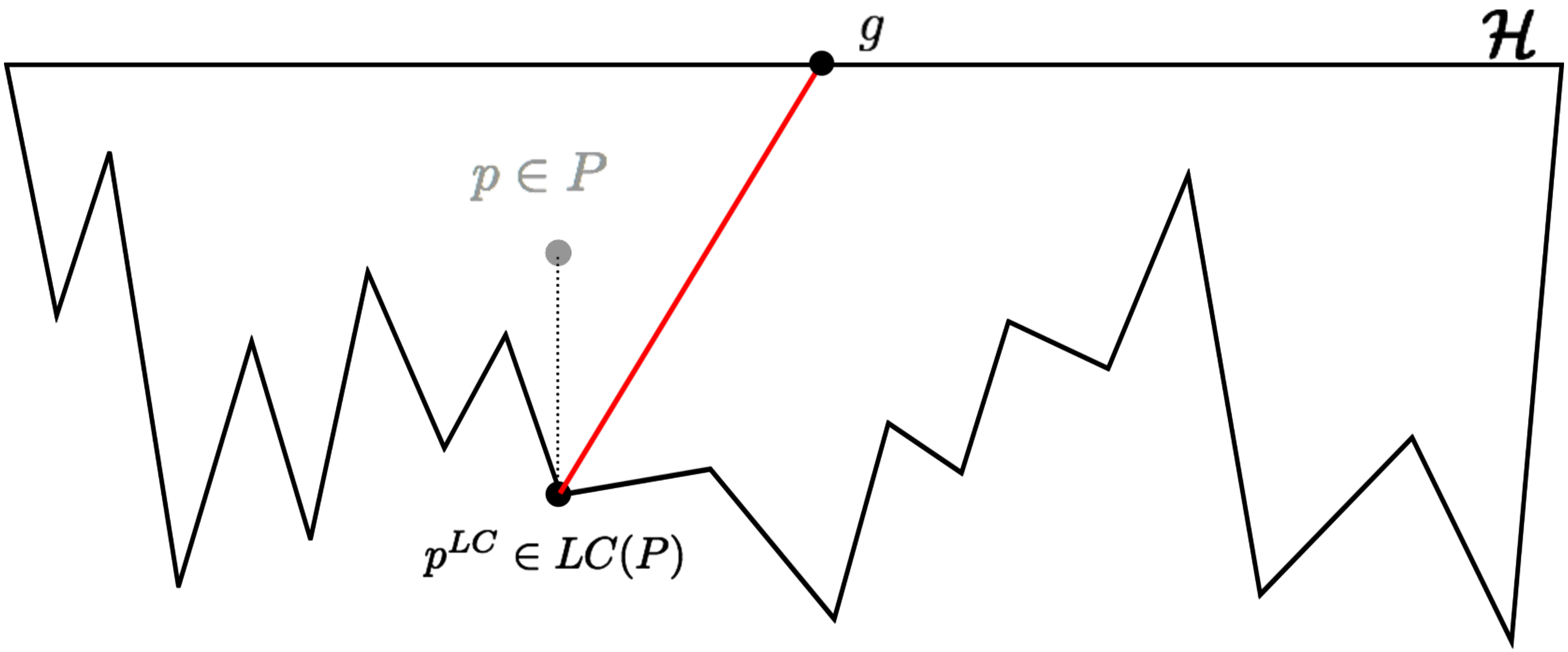
Let P be a uni-monotone polygon, let G be a guard set with $g \in \mathcal{H}\ \forall g \in G$ that covers LC(P), that is, $LC(P) \subset V_P(G)$. Then G covers all of P, that is, $P \subseteq V_P(G)$.

Proof: Assume $p \in P$, $p \notin LC(P)$, $p \notin V_P(G)$



Let P be a uni-monotone polygon, let G be a guard set with $g \in \mathcal{H}\ \forall g \in G$ that covers LC(P), that is, $LC(P) \subset V_P(G)$. Then G covers all of P, that is, $P \subseteq V_P(G)$.

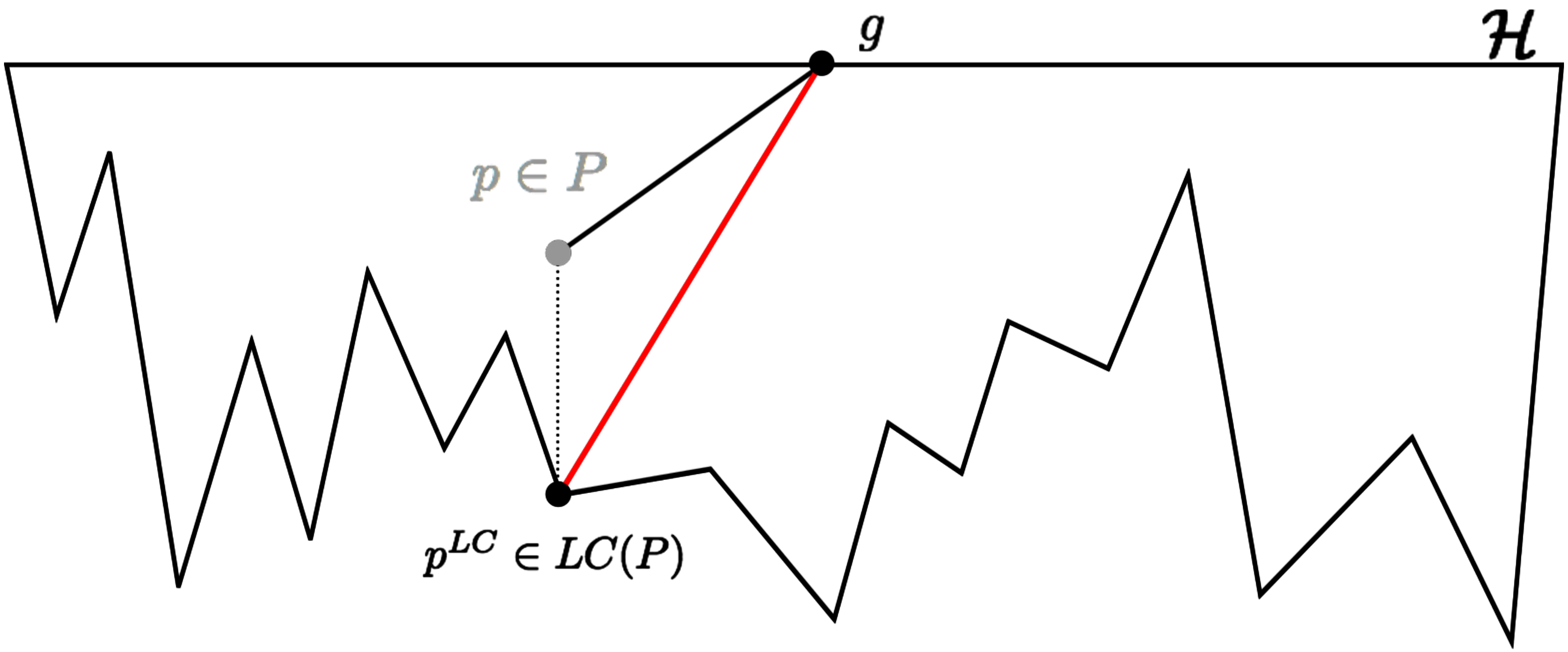
Proof: Assume $p \in P$, $p \notin LC(P)$, $p \notin V_P(G)$



AGP in uni-monotone polygons

Let P be a uni-monotone polygon, let G be a guard set with $g \in \mathcal{H}\ \forall g \in G$ that covers $LC(P)$, that is, $LC(P) \subset V_P(G)$. Then G covers all of P, that is, $P \subseteq V_P(G)$.

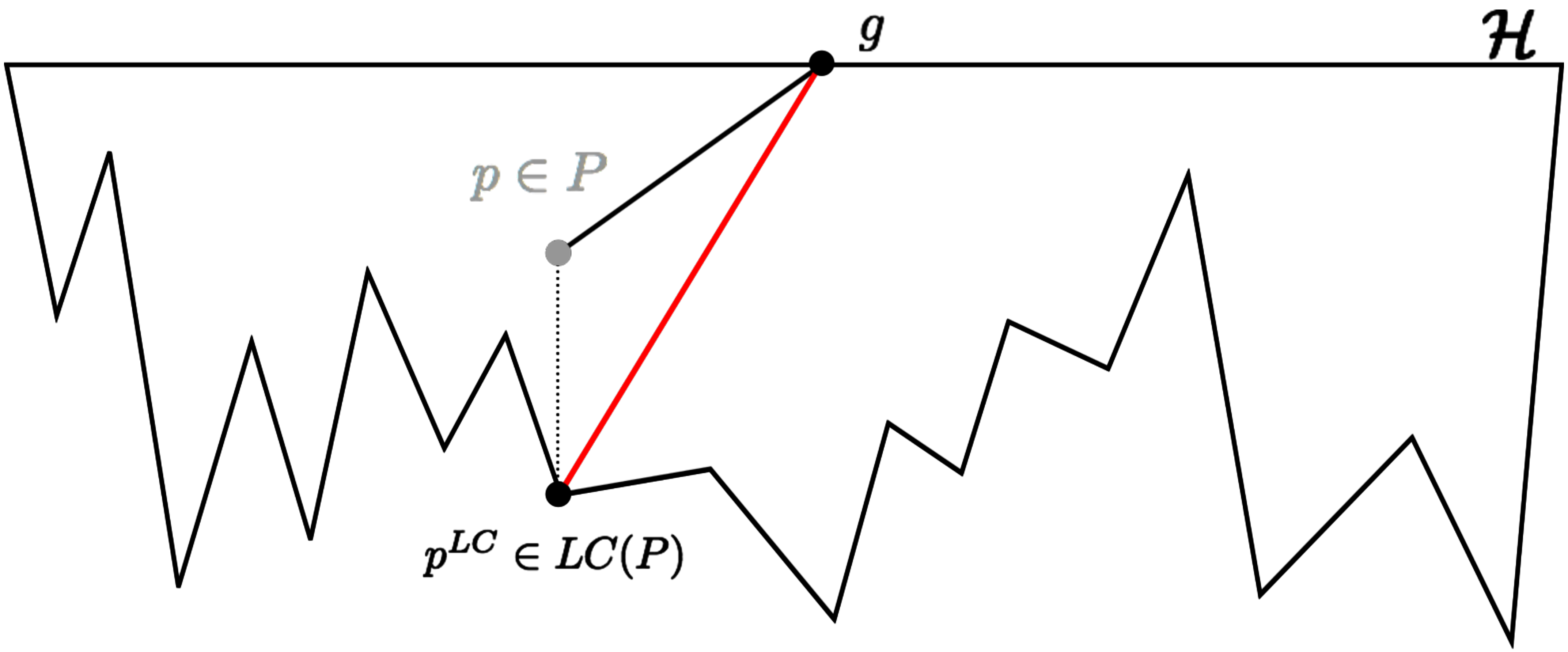
Proof: Assume $p \in P$, $p \notin LC(P)$, $p \notin V_P(G)$



AGP in uni-monotone polygons

Let P be a uni-monotone polygon, let G be a guard set with $g \in \mathcal{H}\ \forall g \in G$ that covers $LC(P)$, that is, $LC(P) \subset V_P(G)$. Then G covers all of P, that is, $P \subseteq V_P(G)$.

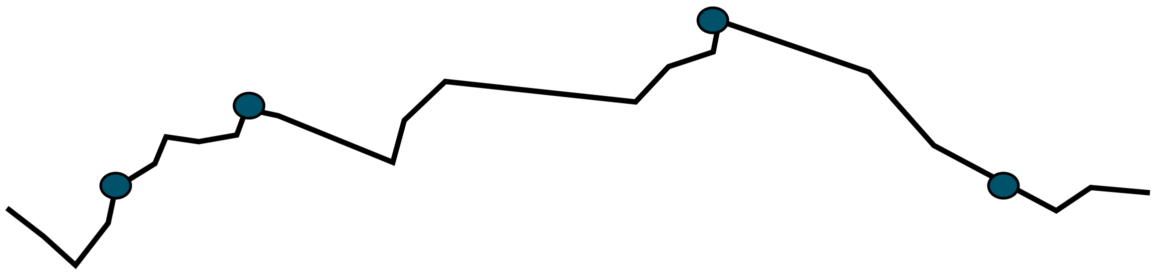
Proof: Assume $p \in P$, $p \notin LC(P)$, $p \notin V_P(G)$



➜ATGP and AGP for uni-monotone polygons equivalent

AGP-TGP-ATGP

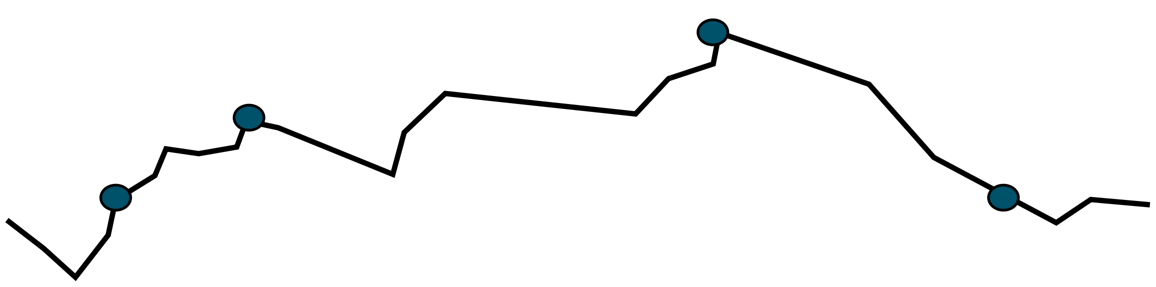
Art Gallery Problem (AGP)
Given: a polygon P.
A minimum guard set that covers P.



Terrain Guarding Problem (TGP)
Given: a terrain T.
Find: minimum number of guards located **on** T.

AGP-TGP-ATGP

Art Gallery Problem (AGP)
Given: a polygon P.
A minimum guard set that covers P.

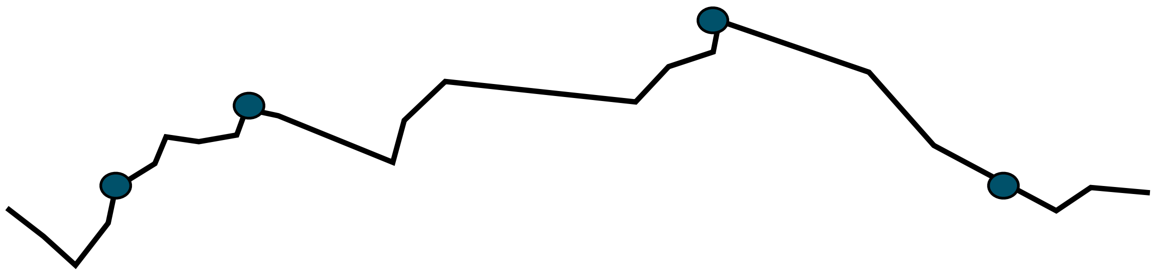


Terrain Guarding Problem (TGP)
Given: a terrain T.
Find: minimum number of guards located **on** T.

$\boxed{\text{NP-hard}}$

Art Gallery Problem (AGP)
Given: a polygon P.
A minimum guard set that covers P.

NP-hard, even in monotone polygons



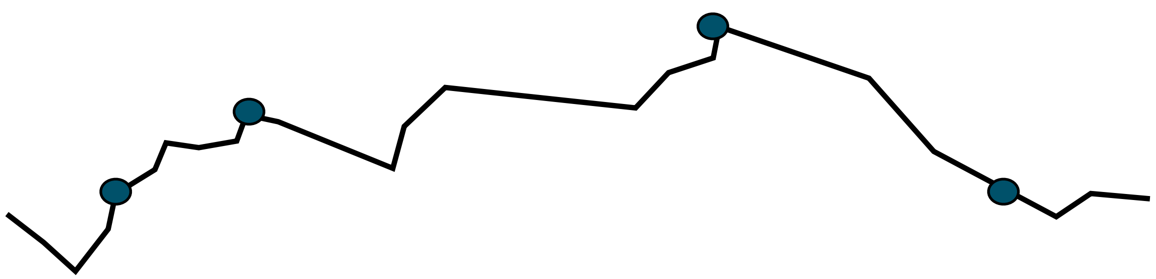
Terrain Guarding Problem (TGP)
Given: a terrain T.
Find: minimum number of guards located **on** T.

NP-hard

Altitude Terrain Guarding Problem (ATGP)
Given: a terrain T and an altitude line $\mathcal{A}$.
A minimum set of guards that see all of T.

Art Gallery Problem (AGP)
Given: a polygon P.
A minimum guard set that covers P.

NP-hard, even in monotone polygons

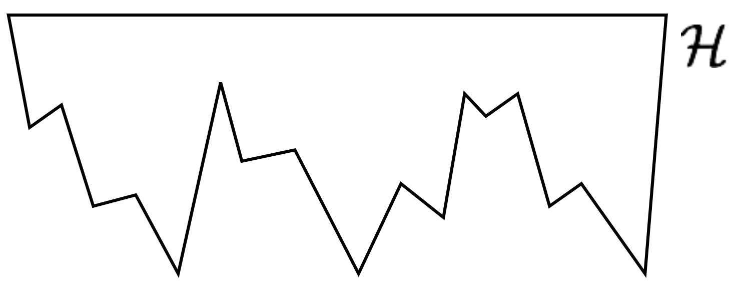


Terrain Guarding Problem (TGP)
Given: a terrain T.
Find: minimum number of guards located **on** T.

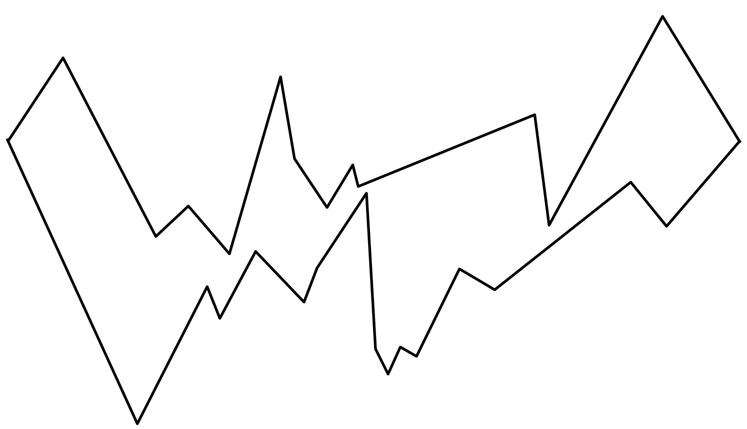
NP-hard

AGP-TGP-ATGP

Altitude Terrain Guarding Problem (ATGP)
Given: a terrain T and an altitude line $\mathcal{A}$.
A minimum set of guards that see all of T.

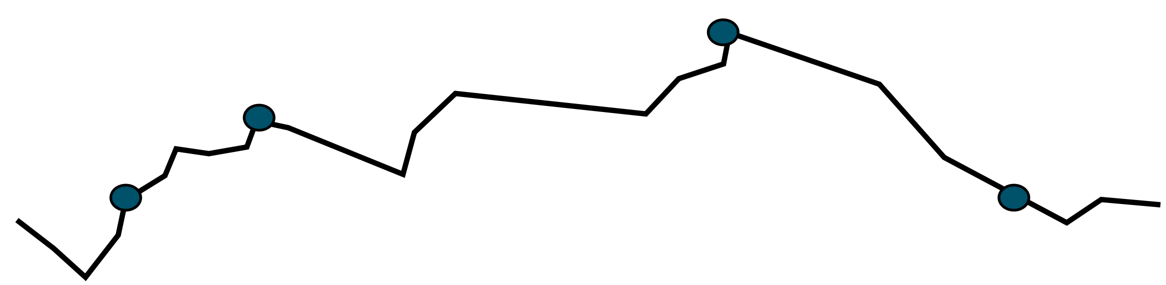


Art Gallery Problem (AGP) in Uni-Monotone Polygons
Given: a uni-monotone polygon P.
A minimum guard set that covers P.



Art Gallery Problem (AGP)
Given: a polygon P.
A minimum guard set that covers P.

NP-hard, even in monotone polygons

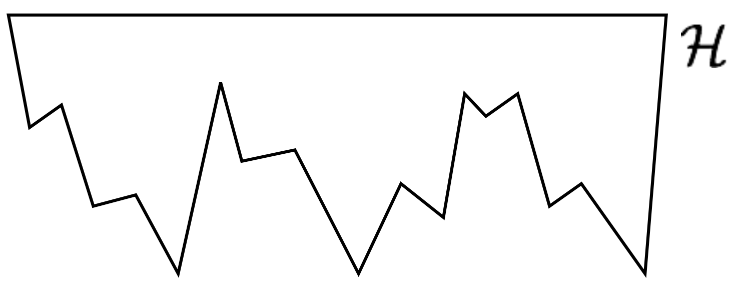


Terrain Guarding Problem (TGP)
Given: a terrain T.
Find: minimum number of guards located **on** T.

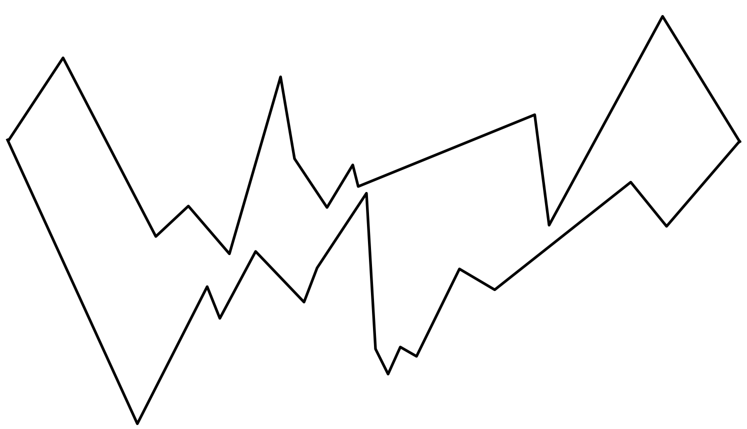
NP-hard

AGP-TGP-ATGP

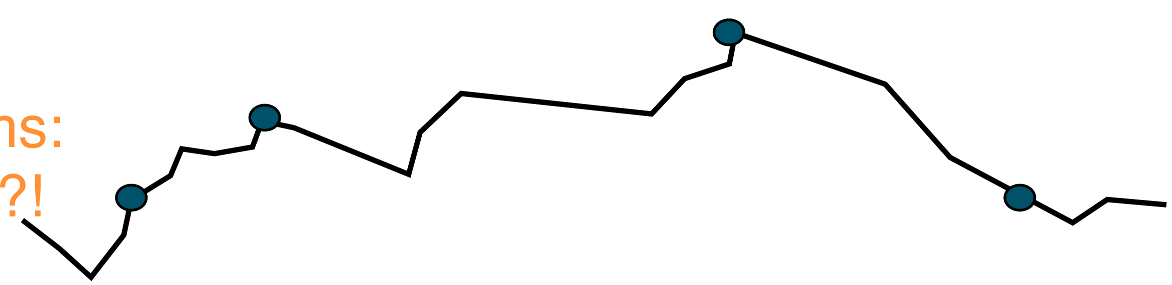
Altitude Terrain Guarding Problem (ATGP)
Given: a terrain T and an altitude line $\mathcal{A}$.
A minimum set of guards that see all of T.



Art Gallery Problem (AGP) in Uni-Monotone Polygons
Given: a uni-monotone polygon P.
A minimum guard set that covers P.



Our Problems:
“Inbetween”?!



Art Gallery Problem (AGP)
Given: a polygon P.
A minimum guard set that covers P.

NP-hard, even in monotone polygons

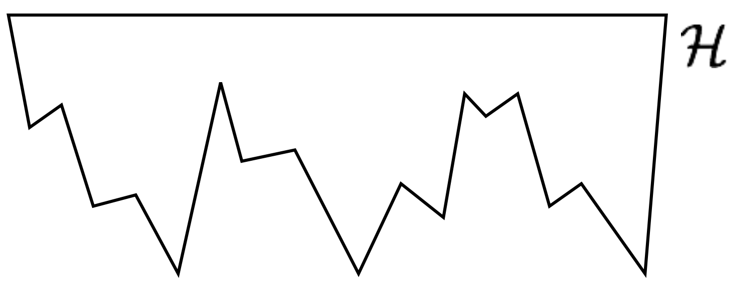
Terrain Guarding Problem (TGP)
Given: a terrain T.
Find: minimum number of guards located **on** T.

NP-hard

AGP-TGP-ATGP

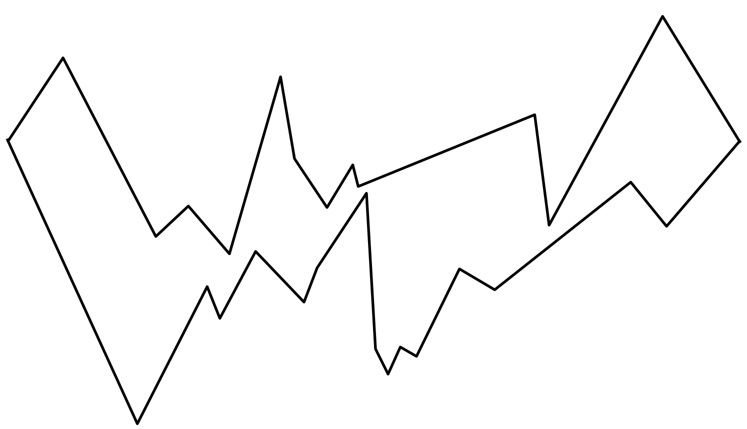
Both polytime

Altitude Terrain Guarding Problem (ATGP)
Given: a terrain T and an altitude line $\mathcal{A}$.
A minimum set of guards that see all of T.



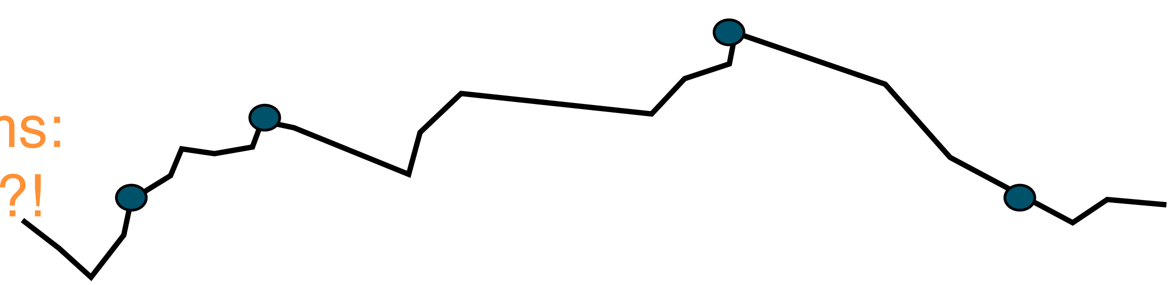
Art Gallery Problem (AGP) in Uni-Monotone Polygons
Given: a uni-monotone polygon P.
A minimum guard set that covers P.

Our Problems:
“Inbetween”?!



Art Gallery Problem (AGP)
Given: a polygon P.
A minimum guard set that covers P.

NP-hard, even in monotone polygons



Terrain Guarding Problem (TGP)
Given: a terrain T.
Find: minimum number of guards located **on** T.

NP-hard

We show:

We show:

- A polytime algorithm for AGTP and AGP in uni-monotone polygons

We show:

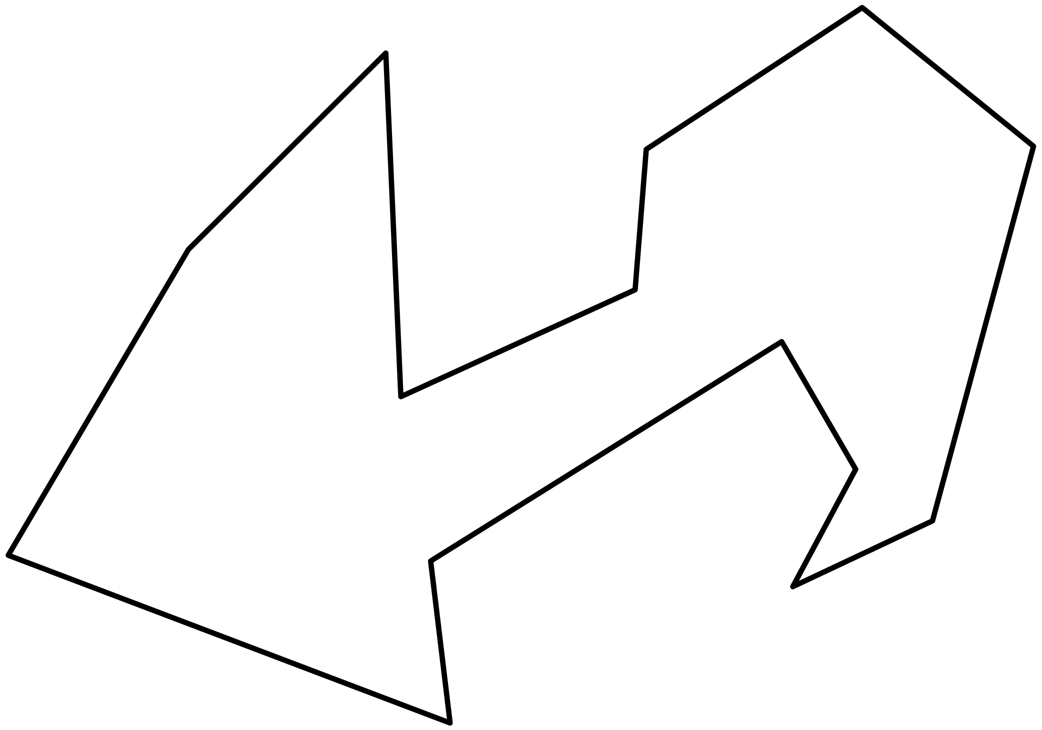
- A polytime algorithm for AGTP and AGP in uni-monotone polygons
- Uni-monotone polygons are perfect - first non-trivial class

We show:

- A polytime algorithm for AGTP and AGP in uni-monotone polygons
- Uni-monotone polygons are **perfect ?** - first non-trivial class

We show:
- A polytime algorithm for AGTP and AGP in uni-monotone polygons
- Uni-monotone polygons are **perfect ?** - first non-trivial class

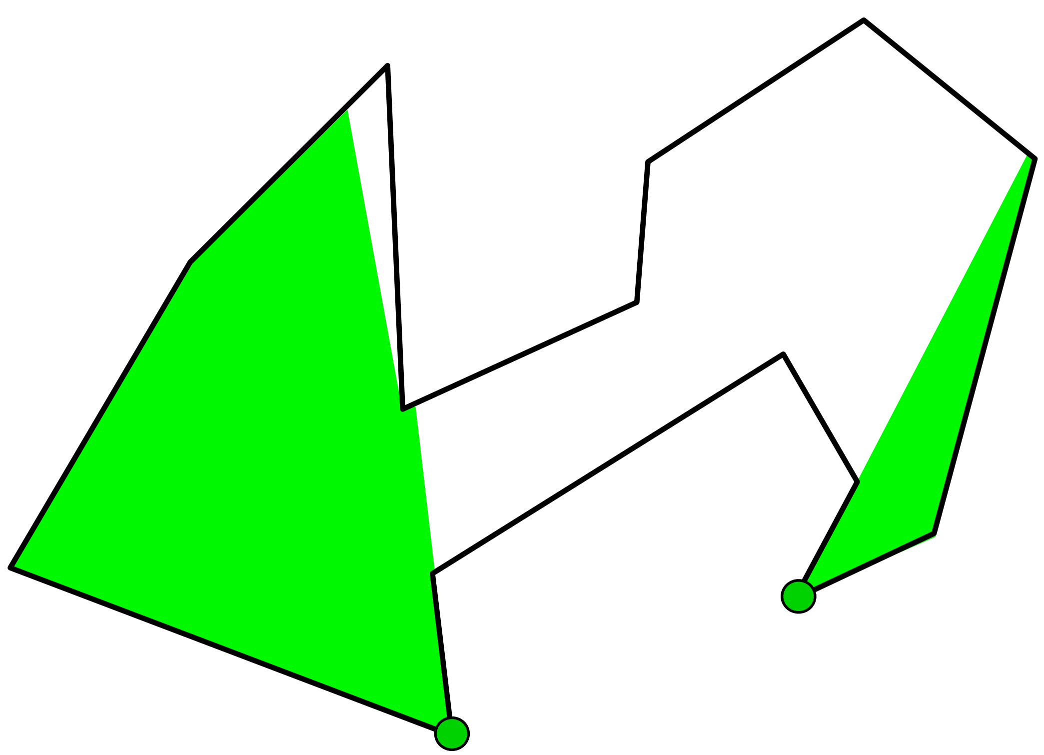
A set $W \subset P$ ($W \subset T$) is a ***witness set*** if $\forall w_i \neq w_j \in W$ we have $V_P(w_i) \cap V_P(w_j) = \emptyset$.



We show:

- A polytime algorithm for AGTP and AGP in uni-monotone polygons
- Uni-monotone polygons are **perfect ?** - first non-trivial class

A set $W \subset P$ ($W \subset T$) is a ***witness set*** if $\forall w_i \neq w_j \in W$ we have $V_P(w_i) \cap V_P(w_j) = \emptyset$.

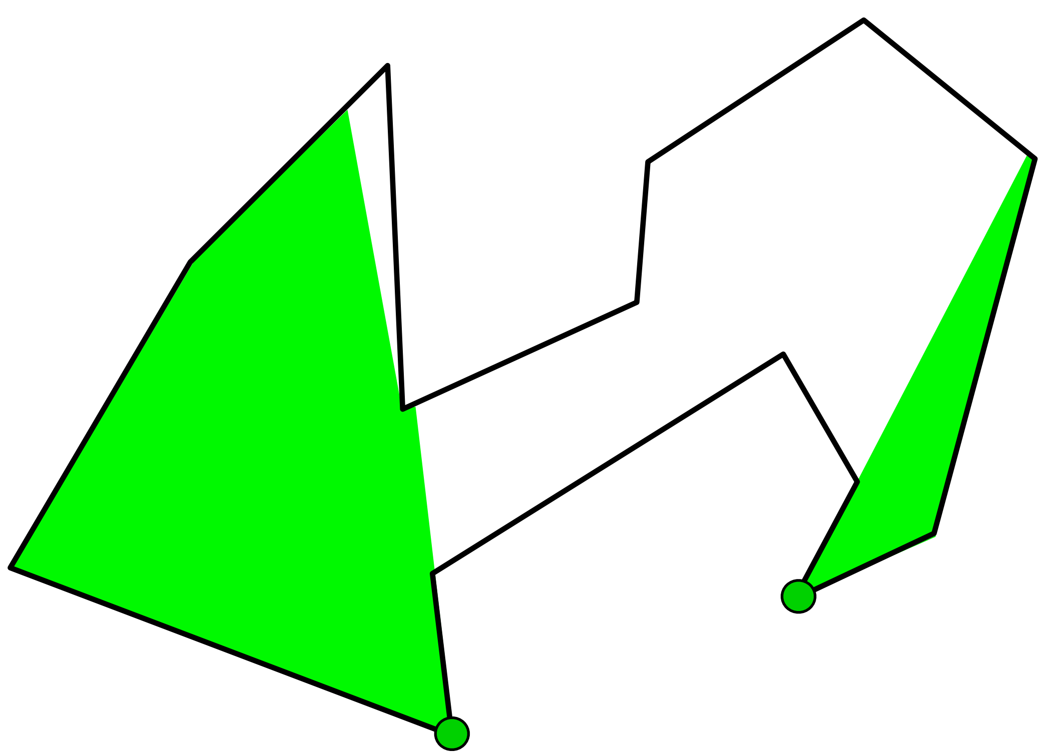


We show:

- A polytime algorithm for AGTP and AGP in uni-monotone polygons
- Uni-monotone polygons are **perfect ?** - first non-trivial class

A set $W \subset P$ ($W \subset T$) is a ***witness set*** if $\forall w_i \neq w_j \in W$ we have $V_P(w_i) \cap V_P(w_j) = \emptyset$.

A ***maximum witness set*** W_{opt} is a witness set of maximum cardinality, $|W_{opt}| = \max\{|W|: \text{witness set } W\}$.



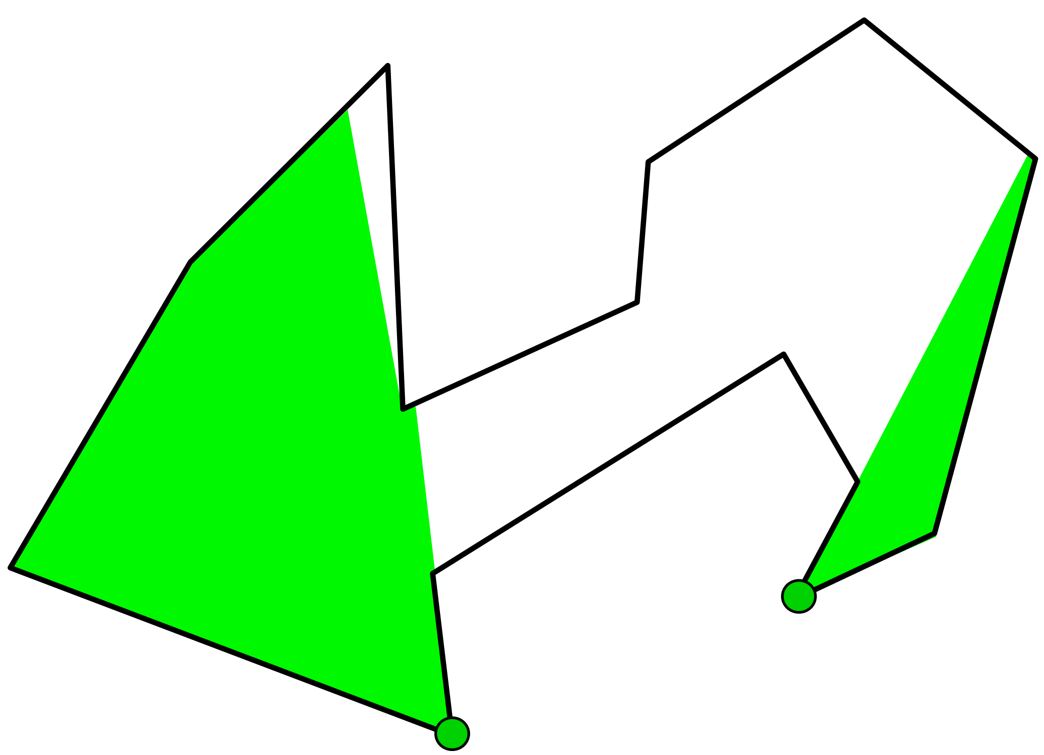
We show:

- A polytime algorithm for AGTP and AGP in uni-monotone polygons
- Uni-monotone polygons are **perfect ?** - first non-trivial class

A set $W \subset P$ ($W \subset T$) is a ***witness set*** if $\forall w_i \neq w_j \in W$ we have $V_P(w_i) \cap V_P(w_j) = \emptyset$.

A ***maximum witness set*** W_{opt} is a witness set of maximum cardinality, $|W_{opt}| = \max\{|W|: \text{witness set } W\}$.

A polygon class $\mathcal{P}$ is ***perfect*** if



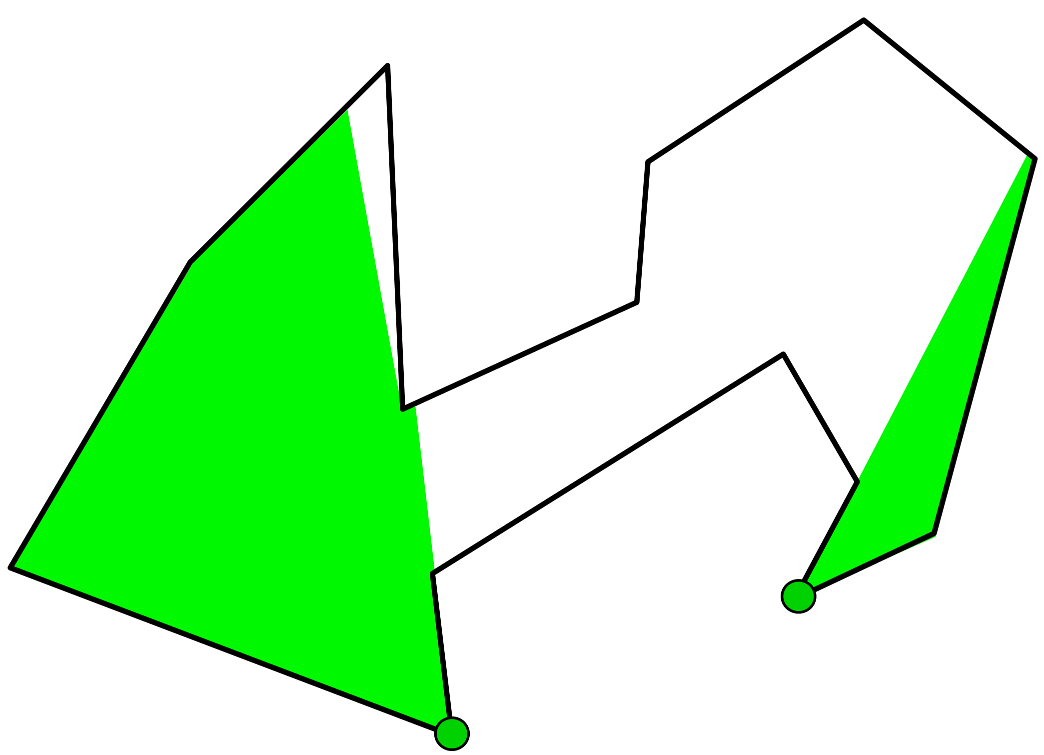
We show:
- A polytime algorithm for AGTP and AGP in uni-monotone polygons
- Uni-monotone polygons are **perfect ?** - first non-trivial class

A set $W \subset P$ ($W \subset T$) is a ***witness set*** if $\forall w_i \neq w_j \in W$ we have $V_P(w_i) \cap V_P(w_j) = \emptyset$.

A ***maximum witness set*** W_{opt} is a witness set of maximum cardinality, $|W_{opt}| = \max\{|W|: \text{witness set } W\}$.

A polygon class $\mathcal{P}$ is ***perfect*** if

cardinality of an optimum guard set = cardinality of a maximum witness set $\forall P \in \mathcal{P}$

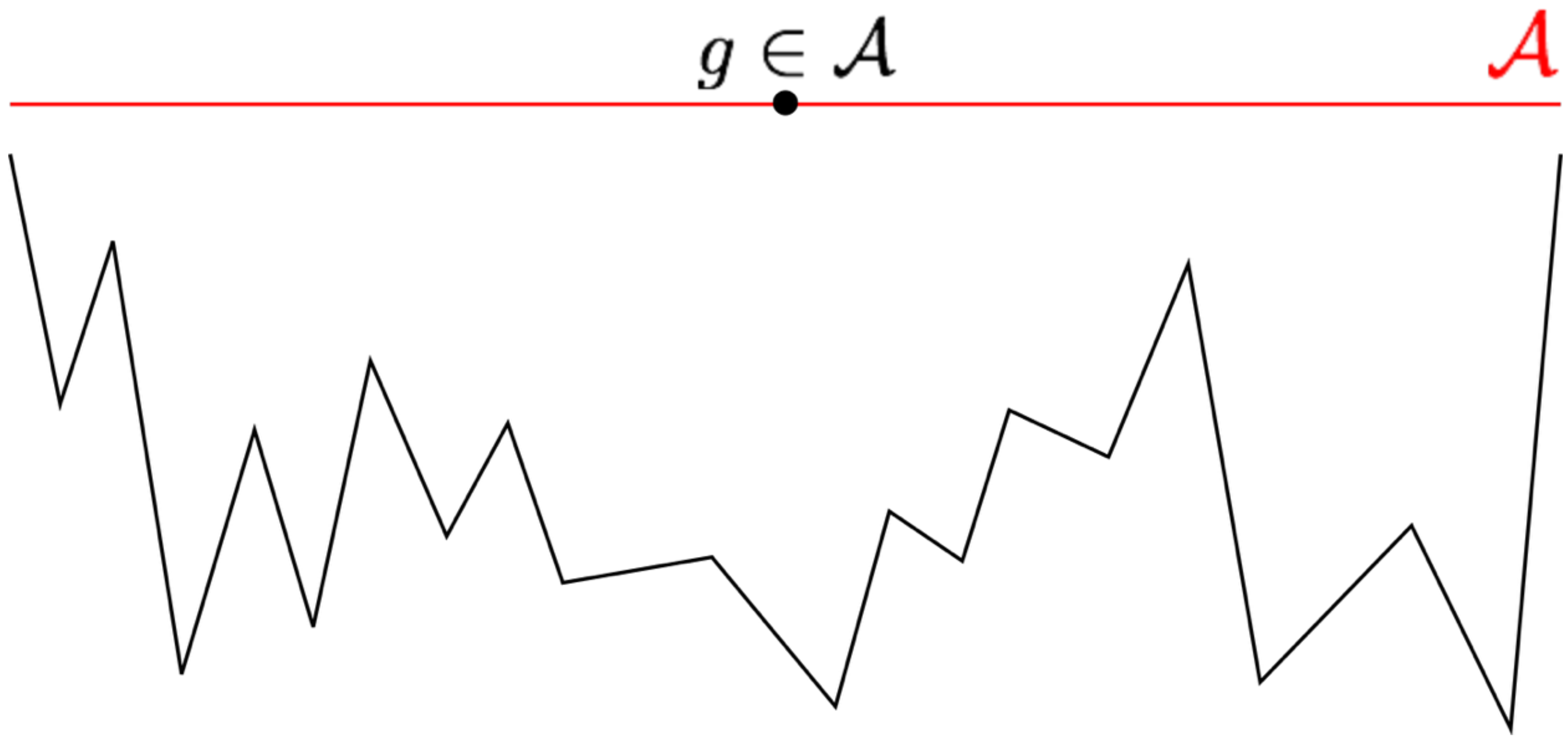


Altitude Terrain Guarding Problem

Guards to the left “don’t help”:

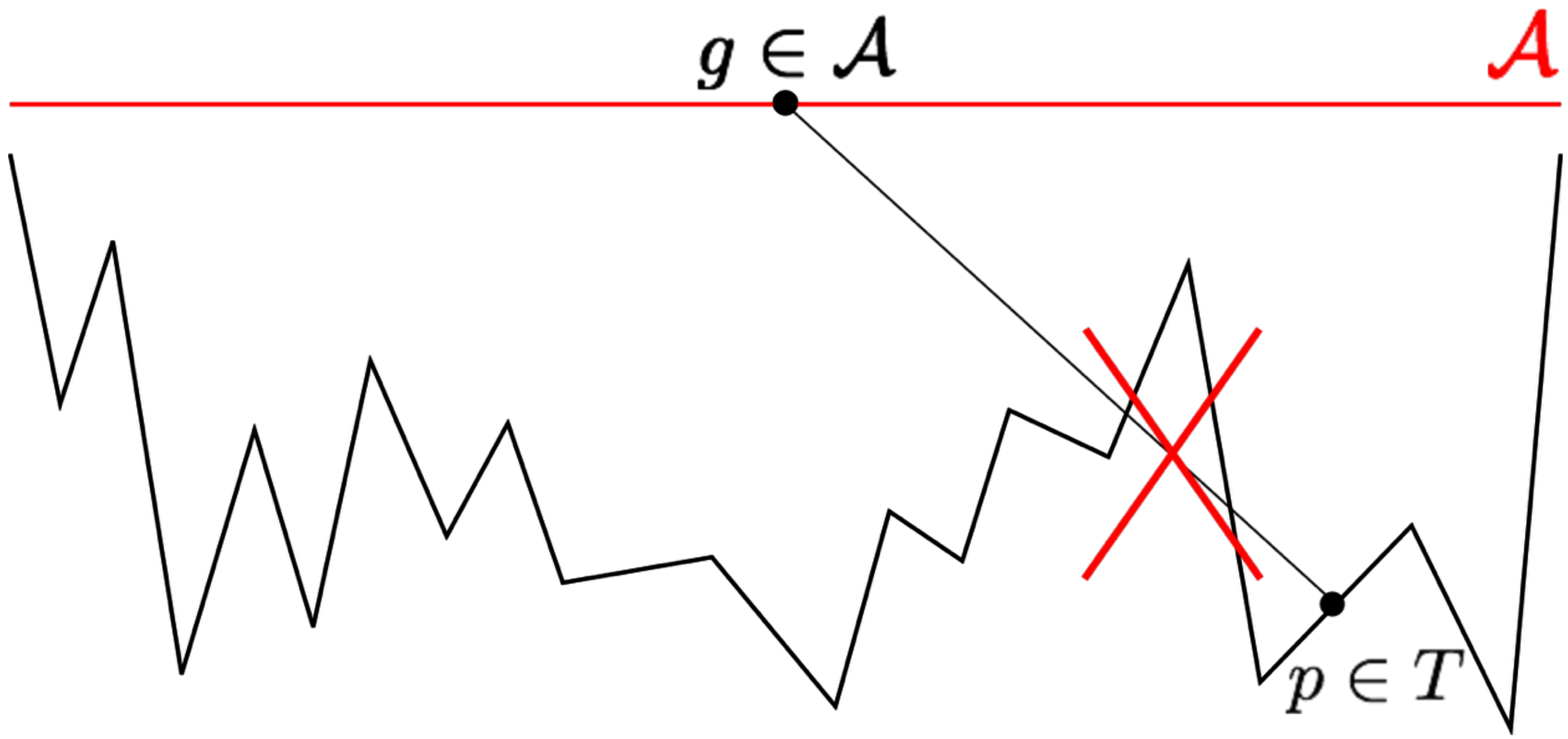
Altitude Terrain Guarding Problem

Guards to the left "don't help":



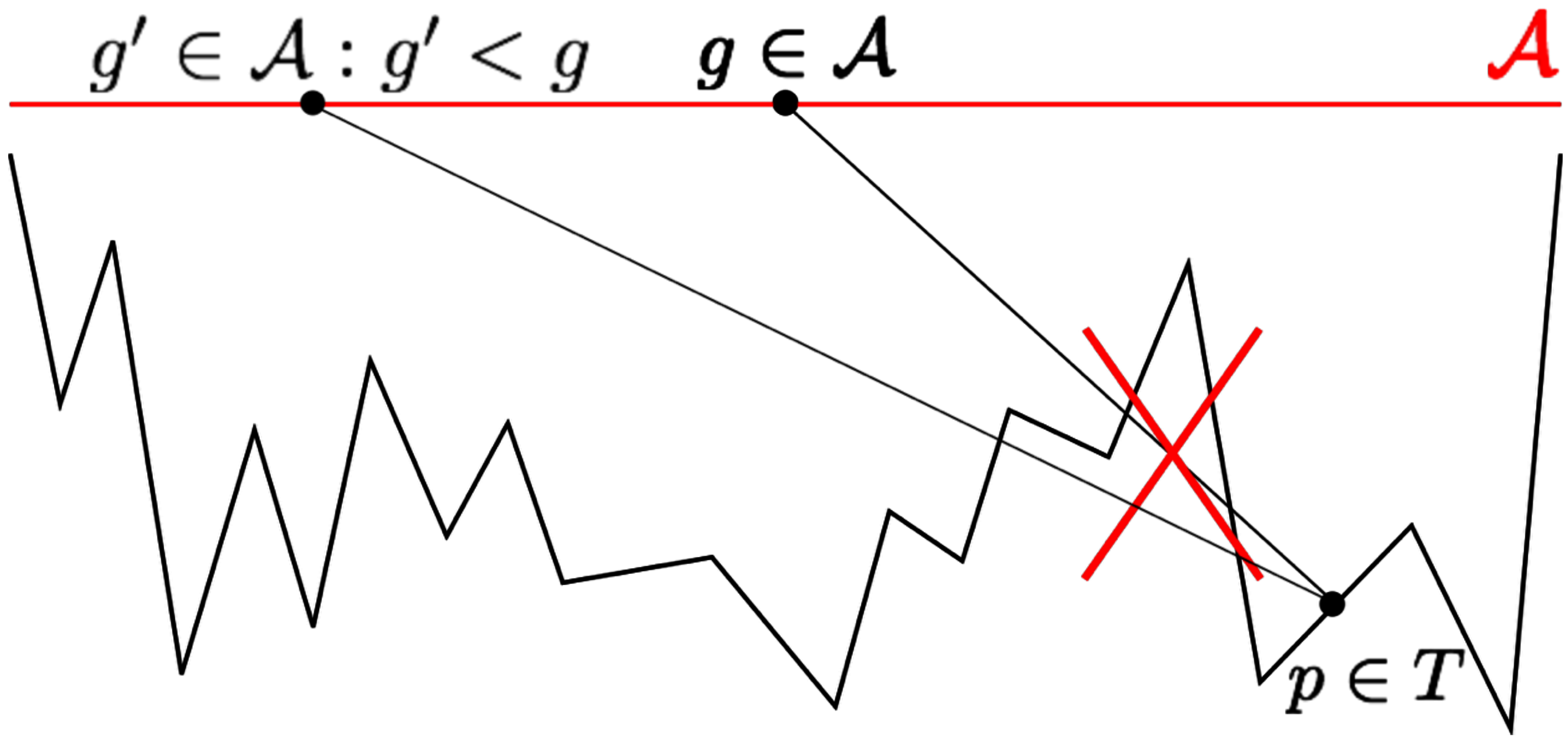
Altitude Terrain Guarding Problem

Guards to the left “don’t help”:



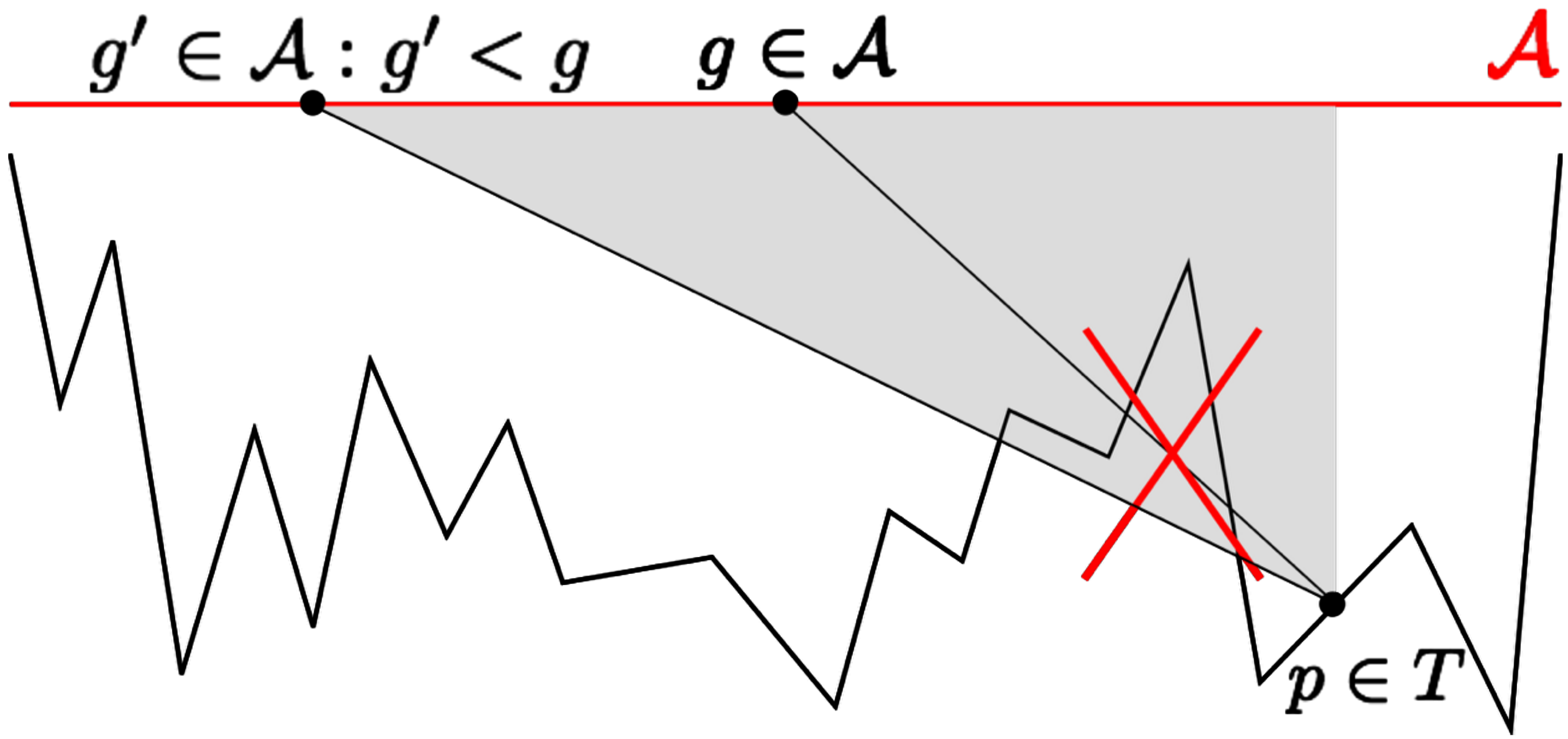
Altitude Terrain Guarding Problem

Guards to the left "don't help":



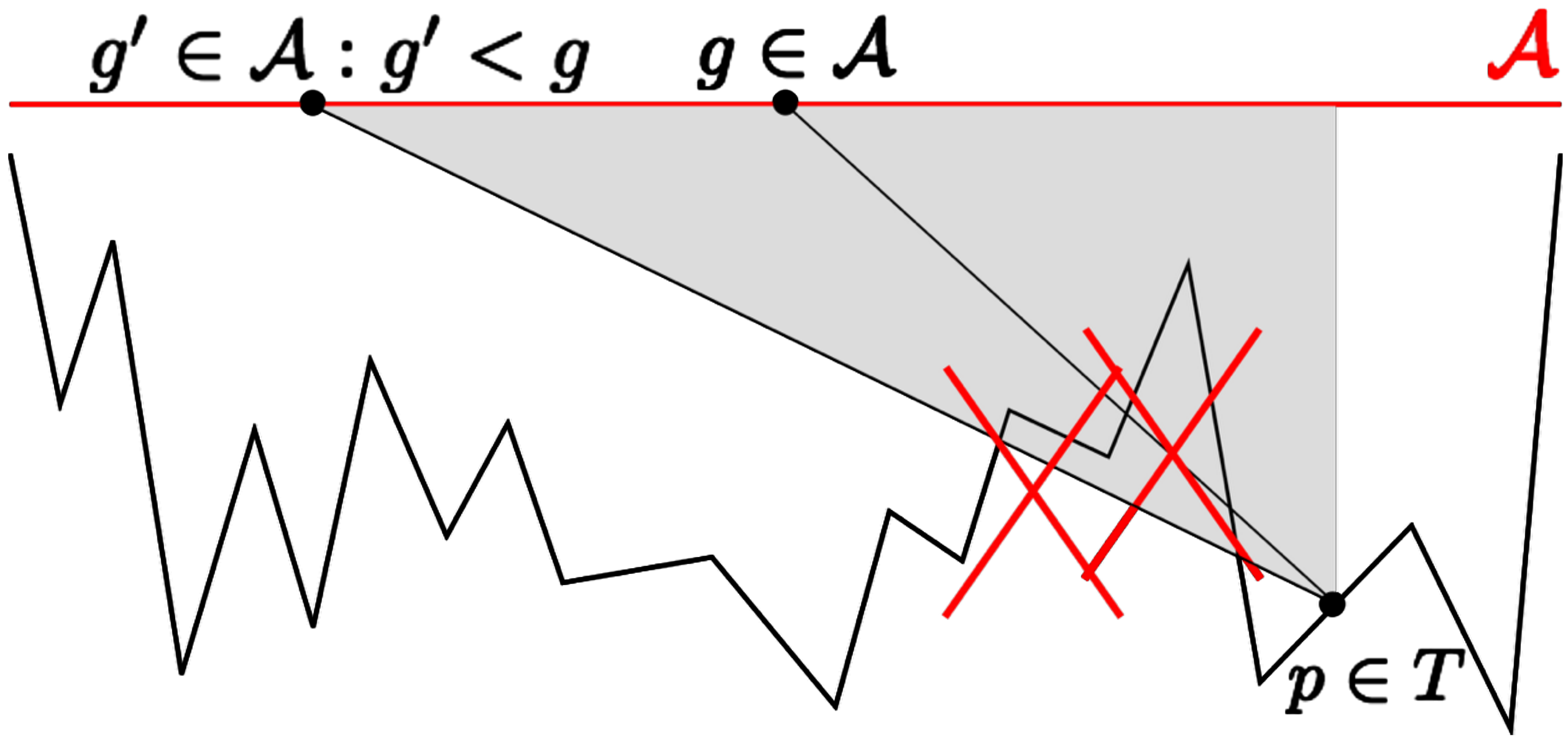
Altitude Terrain Guarding Problem

Guards to the left “don't help”:

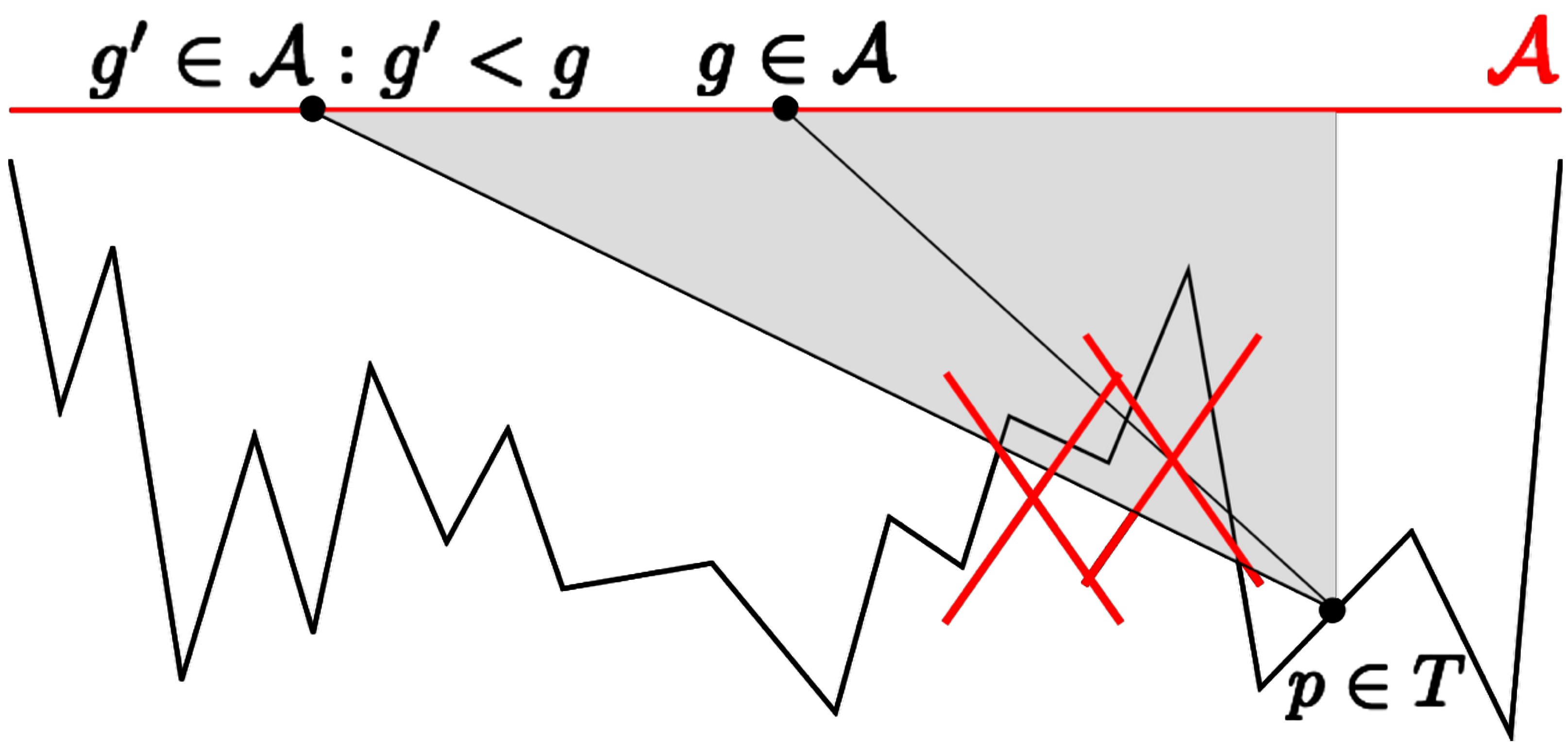


Altitude Terrain Guarding Problem

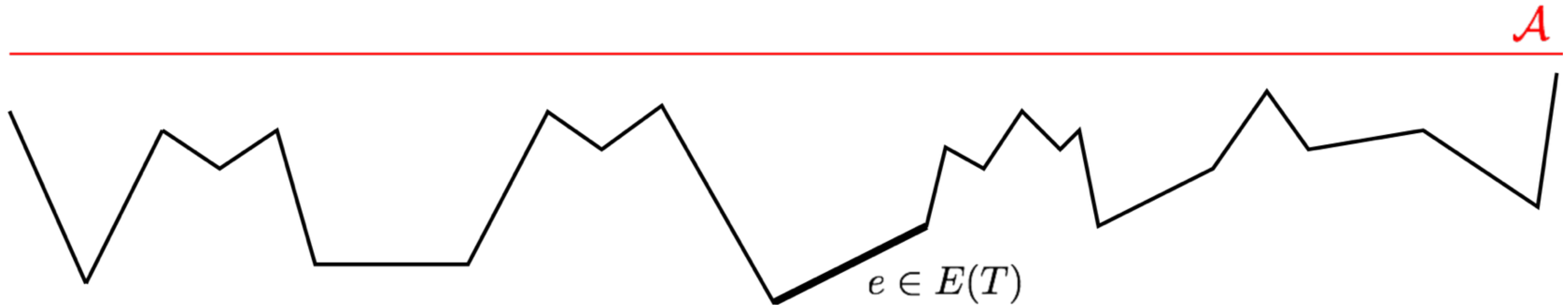
Guards to the left "don't help":



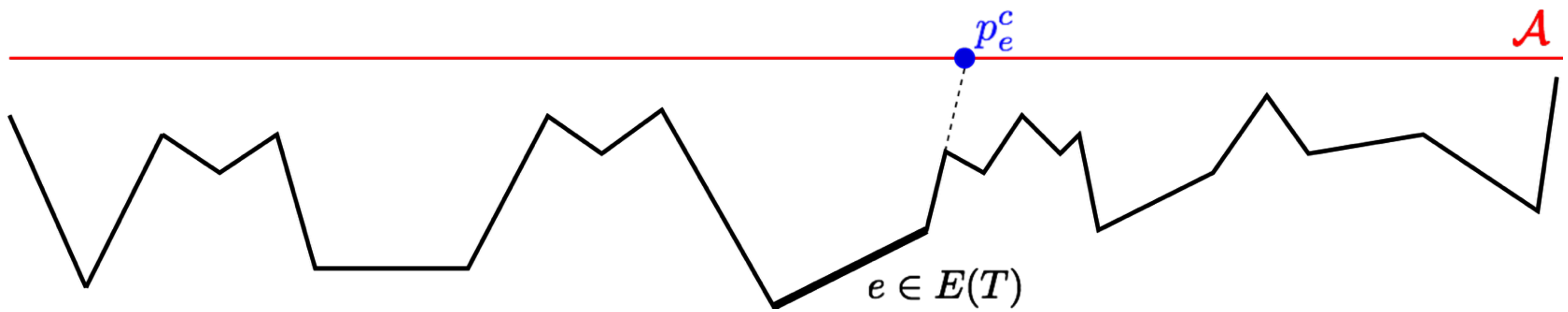
Guards to the left “don’t help”:



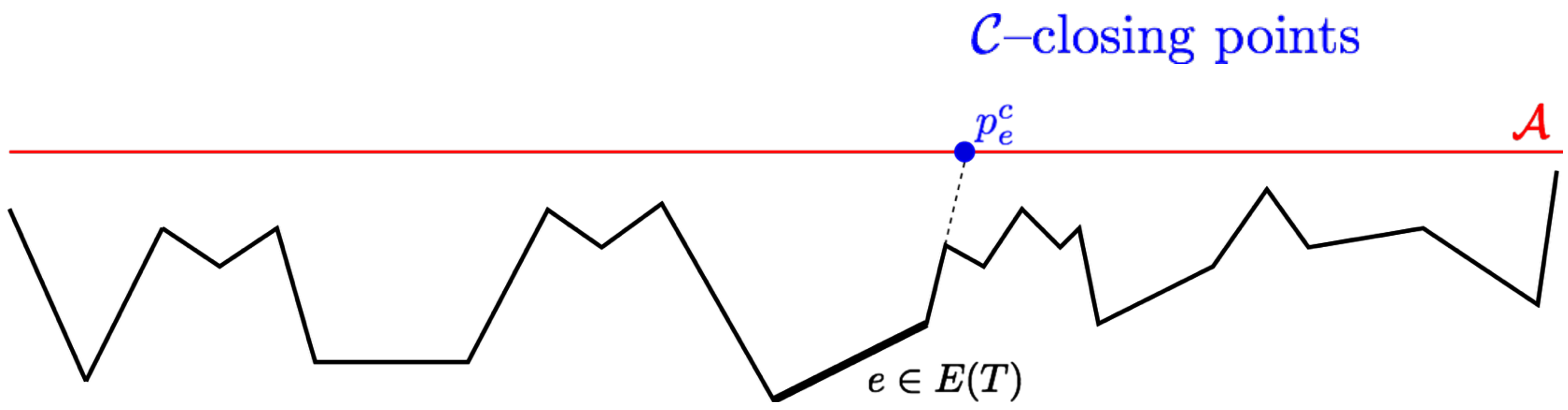
Sweep Algorithm



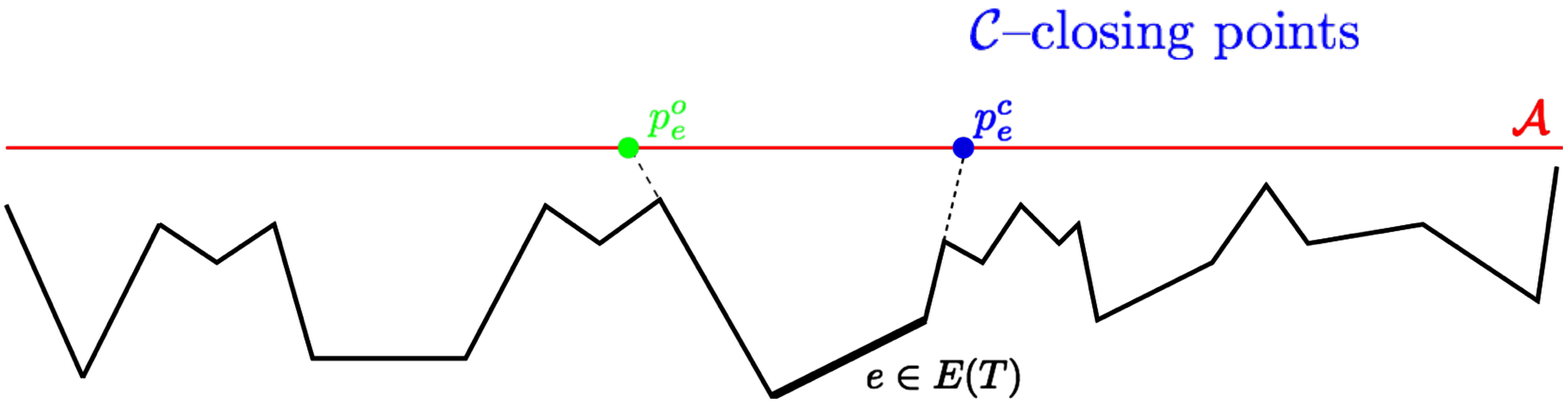
Sweep Algorithm



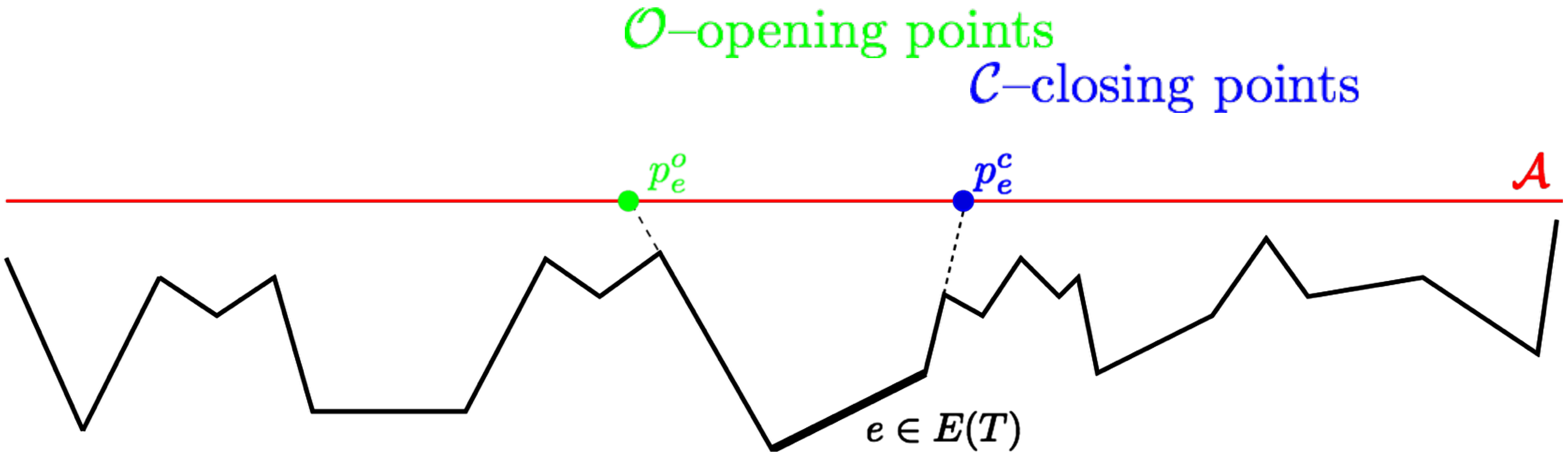
Sweep Algorithm



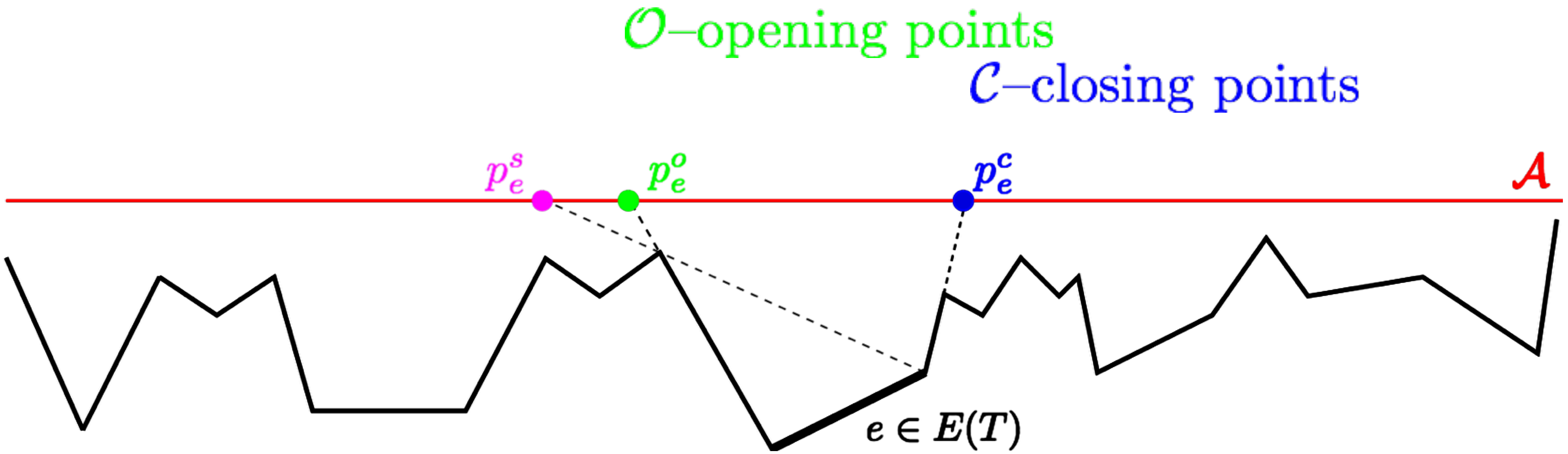
Sweep Algorithm



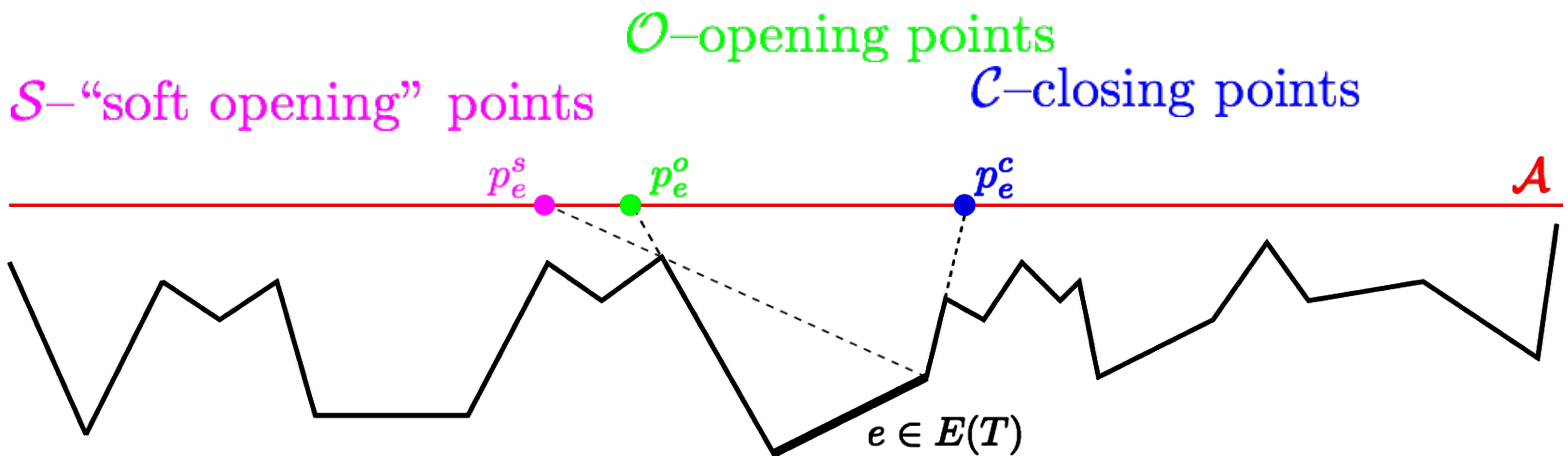
Sweep Algorithm



Sweep Algorithm

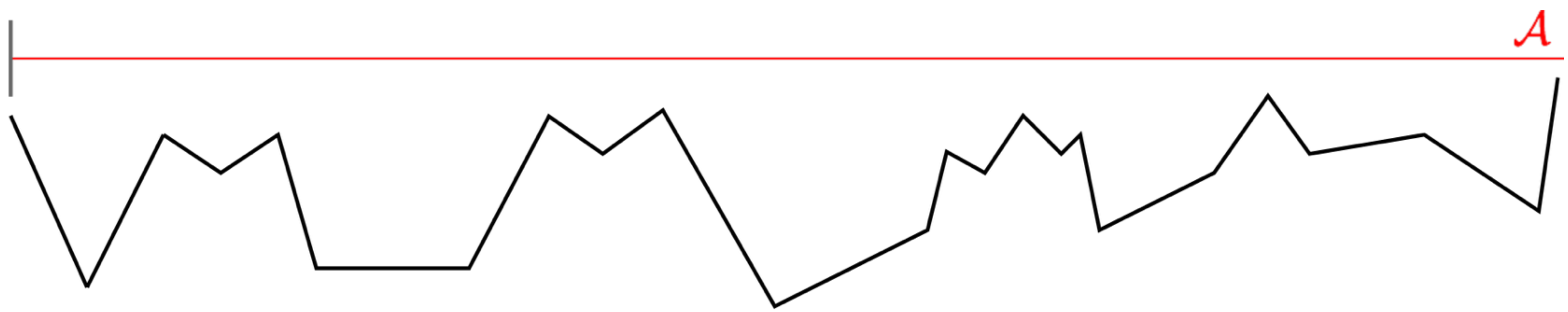


Sweep Algorithm



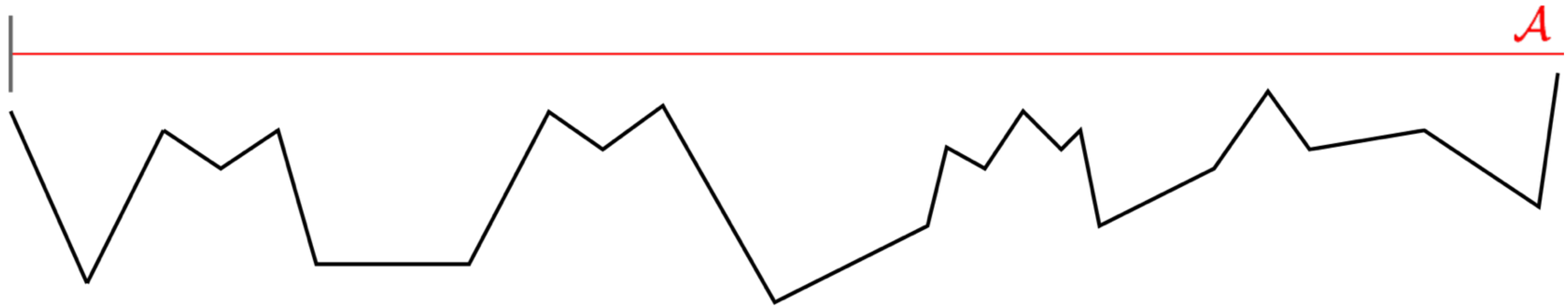
Sweep Algorithm

- Start: empty set of guards $G = \varnothing$; at leftmost point of $\mathcal{A}$; all edges in E(T) are completely unseen (E_g=E(T)).



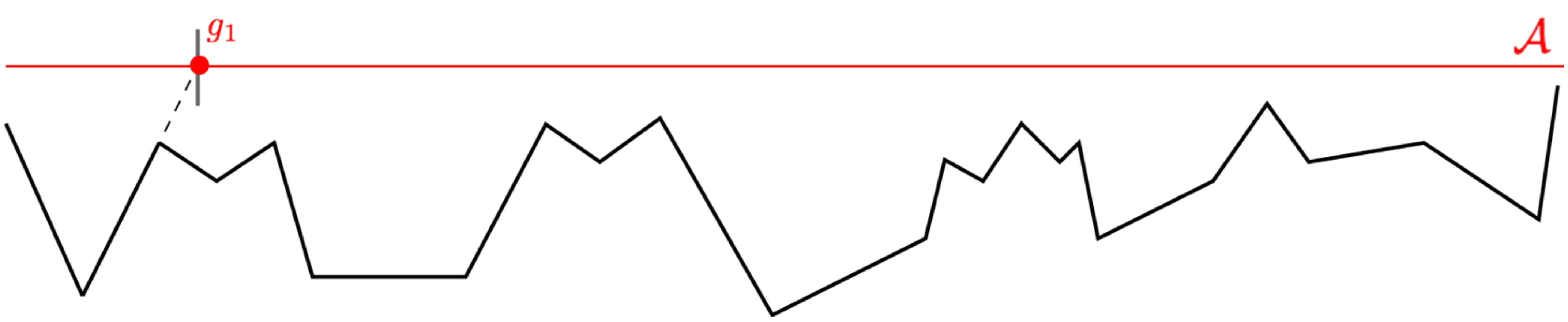
Sweep Algorithm

- Start: empty set of guards $G = \varnothing$; at leftmost point of $\mathcal{A}$; all edges in E(T) are completely unseen (E_g=E(T)).
- Sweep along $\mathcal{A}$ from left to right



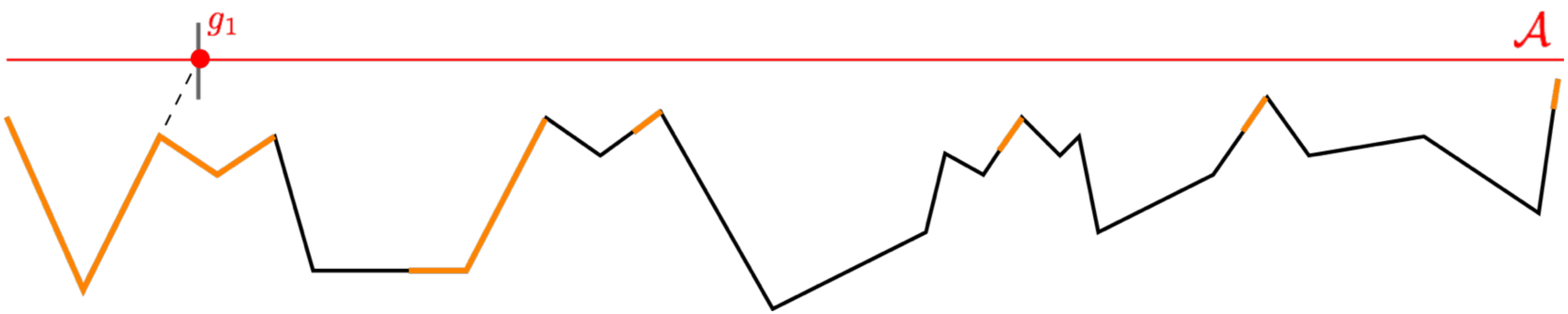
Sweep Algorithm

- Start: empty set of guards G = ∅ ; at leftmost point of $\mathcal{A}$; all edges in E(T) are completely unseen (E_g=E(T)).
- Sweep along $\mathcal{A}$ from left to right
- Place a guard g_i whenever we could no longer see all of an "edge" e if we would move more to the right - first point in $\mathcal{C}$.



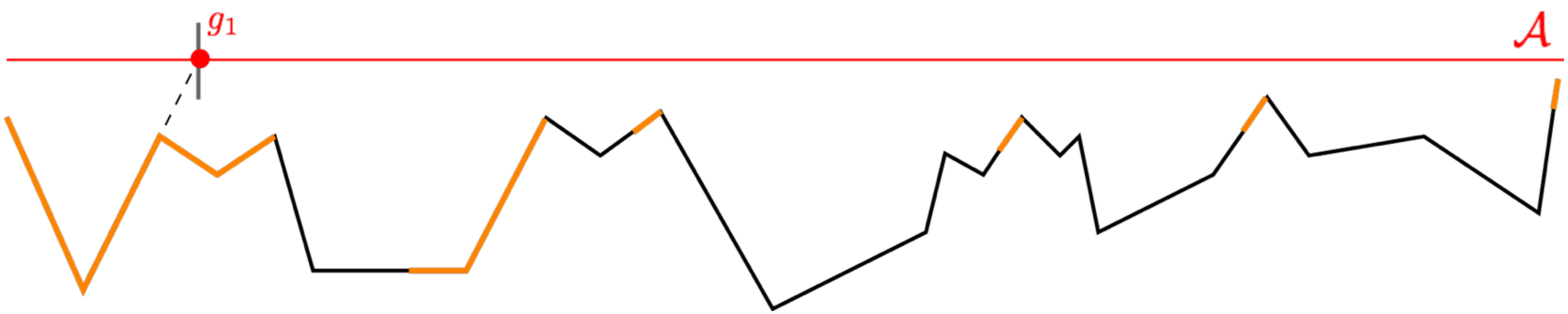
Sweep Algorithm

- Start: empty set of guards $G = \varnothing$; at leftmost point of $\mathcal{A}$; all edges in E(T) are completely unseen (E_g=E(T)).
- Sweep along $\mathcal{A}$ from left to right
- Place a guard g_i whenever we could no longer see all of an “edge” e if we would move more to the right - first point in $\mathcal{C}$.
- Compute $V_T(g_i)$



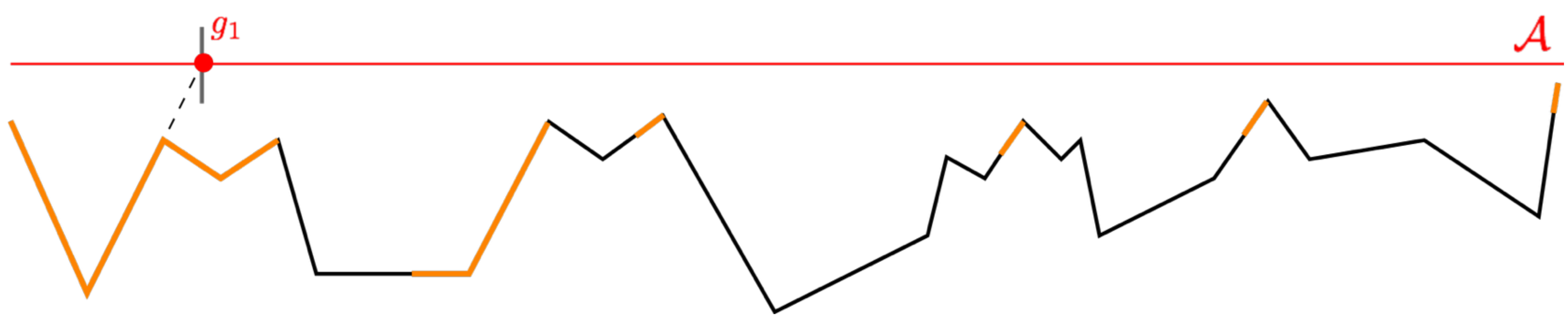
Sweep Algorithm

- Start: empty set of guards $G = \varnothing$; at leftmost point of $\mathcal{A}$; all edges in $E(T)$ are completely unseen (E_g=$E(T)$).
- Sweep along $\mathcal{A}$ from left to right
- Place a guard g_i whenever we could no longer see all of an "edge" e if we would move more to the right - first point in $\mathcal{C}$.
- Compute $V_T(g_i)$
- Remove all completely seen edges from E_g



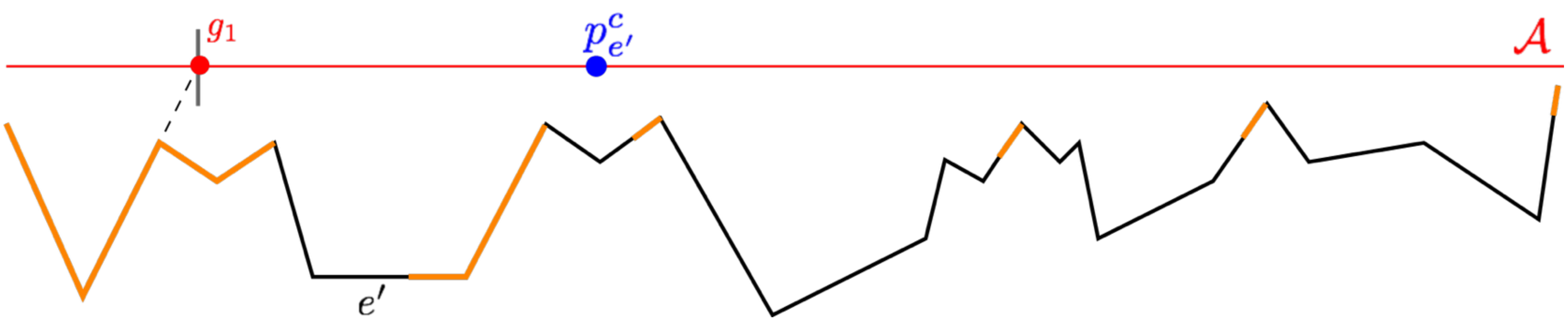
Sweep Algorithm

- Start: empty set of guards $G = \varnothing$; at leftmost point of $\mathcal{A}$; all edges in E(T) are completely unseen (E_g=E(T)).
- Sweep along $\mathcal{A}$ from left to right
- Place a guard g_i whenever we could no longer see all of an "edge" e if we would move more to the right - first point in $\mathcal{C}$.
- Compute $V_T(g_i)$
- Remove all completely seen edges from E_g
- For each edge e = {v,w} partially seen by g_i: split edge, keep the open interval that is not yet guarded —>new "edge" $e' \subset e$



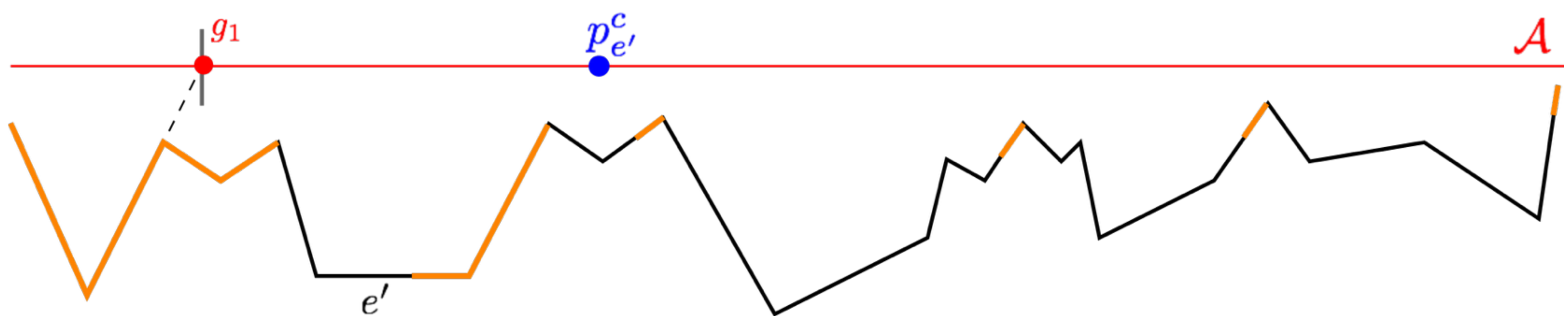
Sweep Algorithm

- Start: empty set of guards G = ∅ ; at leftmost point of $\mathcal{A}$; all edges in E(T) are completely unseen (E_g=E(T)).
- Sweep along $\mathcal{A}$ from left to right
- Place a guard g_i whenever we could no longer see all of an "edge" e if we would move more to the right - first point in $\mathcal{C}$.
- Compute $V_T(g_i)$
- Remove all completely seen edges from E_g
- For each edge e = {v,w} partially seen by g_i: split edge, keep the open interval that is not yet guarded —>new "edge" e'⊂e



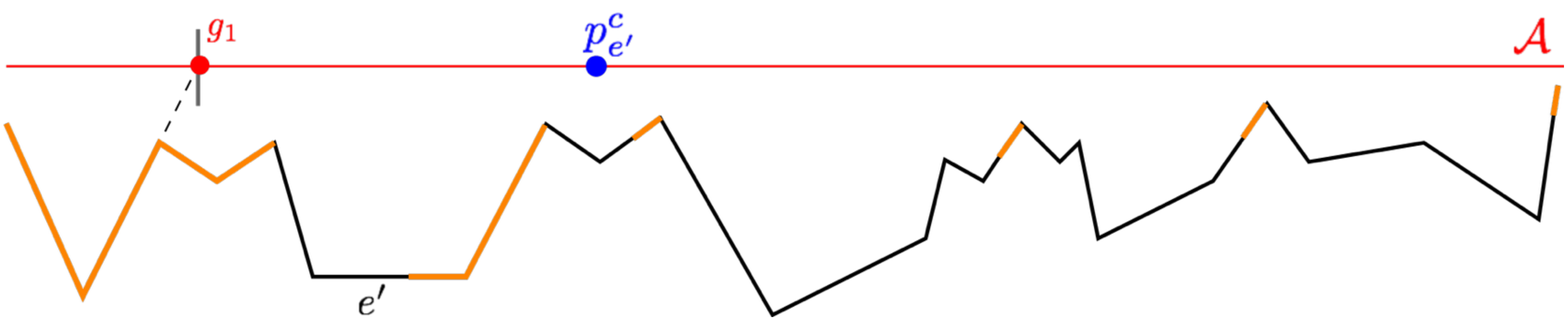
Sweep Algorithm

- Start: empty set of guards $G = \varnothing$; at leftmost point of $\mathcal{A}$; all edges in E(T) are completely unseen (E_g=E(T)).
- Sweep along $\mathcal{A}$ from left to right
- Place a guard g_i whenever we could no longer see all of an "edge" e if we would move more to the right - first point in C.
- Compute $V_T(g_i)$
- Remove all completely seen edges from E_g
- For each edge e = {v,w} partially seen by g_i: split edge, keep the open interval that is not yet guarded —>new "edge" $e' \subset e$
- Delete p_e^c and add $p_{e'}^c$ to C



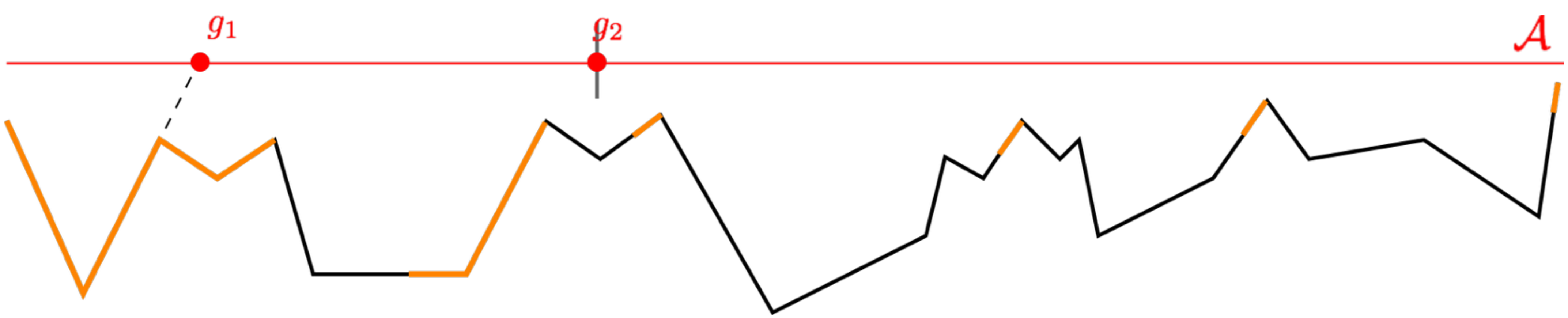
Sweep Algorithm

- Start: empty set of guards $G = \varnothing$; at leftmost point of $\mathcal{A}$; all edges in $E(T)$ are completely unseen (E_g=$E(T)$).
- Sweep along $\mathcal{A}$ from left to right
- Place a guard g_i whenever we could no longer see all of an "edge" e if we would move more to the right - first point in $\mathcal{C}$.
- Compute $V_T(g_i)$
- Remove all completely seen edges from E_g
- For each edge e = {v,w} partially seen by g_i: split edge, keep the open interval that is not yet guarded —>new "edge" $e' \subset e$
- Delete p_e^c and add $p_{e'}^c$ to $\mathcal{C}$
- Delete e from E_g and add e'



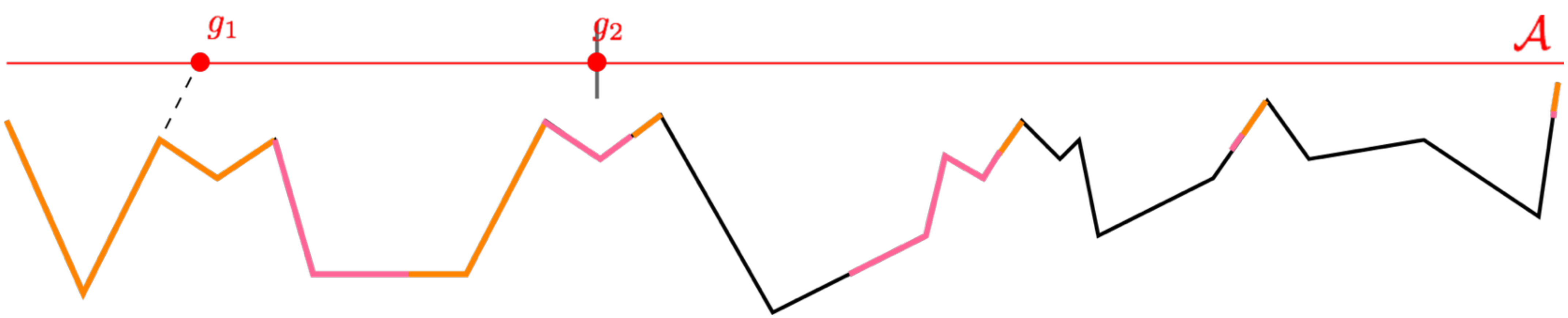
Sweep Algorithm

- Start: empty set of guards $G = \varnothing$; at leftmost point of $\mathcal{A}$; all edges in E(T) are completely unseen (E_g=E(T)).
- Sweep along $\mathcal{A}$ from left to right
- Place a guard g_i whenever we could no longer see all of an “edge” e if we would move more to the right - first point in $\mathcal{C}$.
- Compute $V_T(g_i)$
- Remove all completely seen edges from E_g
- For each edge e = {v,w} partially seen by g_i: split edge, keep the open interval that is not yet guarded —>new “edge” $e' \subset e$
- Delete p_e^c and add $p_{e'}^c$ to $\mathcal{C}$
- Delete e from E_g and add e’



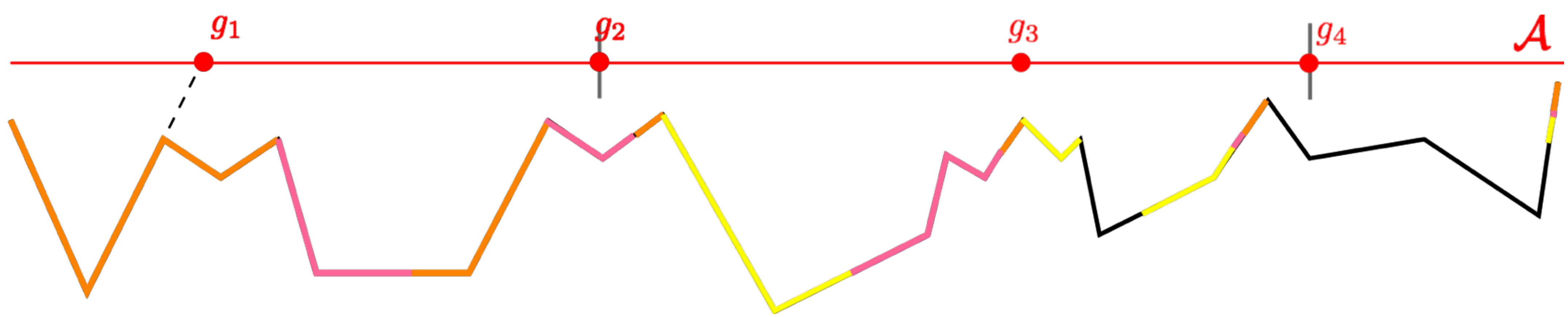
Sweep Algorithm

- Start: empty set of guards $G = \varnothing$; at leftmost point of $\mathcal{A}$; all edges in E(T) are completely unseen (E_g=E(T)).
- Sweep along $\mathcal{A}$ from left to right
- Place a guard g_i whenever we could no longer see all of an "edge" e if we would move more to the right - first point in C.
- Compute $V_T(g_i)$
- Remove all completely seen edges from E_g
- For each edge e = {v,w} partially seen by g_i: split edge, keep the open interval that is not yet guarded —>new "edge" $e' \subset e$
- Delete p_e^c and add $p_{e'}^c$ to C
- Delete e from E_g and add e'



Sweep Algorithm

- Start: empty set of guards $G = \emptyset$; at leftmost point of $\mathcal{A}$; all edges in E(T) are completely unseen (E_g=E(T)).
- Sweep along $\mathcal{A}$ from left to right
- Place a guard g_i whenever we could no longer see all of an “edge” e if we would move more to the right - first point in C.
- Compute $V_T(g_i)$
- Remove all completely seen edges from E_g
- For each edge e = {v,w} partially seen by g_i: split edge, keep the open interval that is not yet guarded —>new “edge” $e' \subset e$
- Delete p_e^c and add $p_{e'}^c$ to C
- Delete e from E_g and add e’



Sweep Algorithm

How do we preprocess our terrain to easily identify the point on $\mathcal{A}$ that we need to add to $\mathcal{C}$ when we split an edge?

Sweep Algorithm

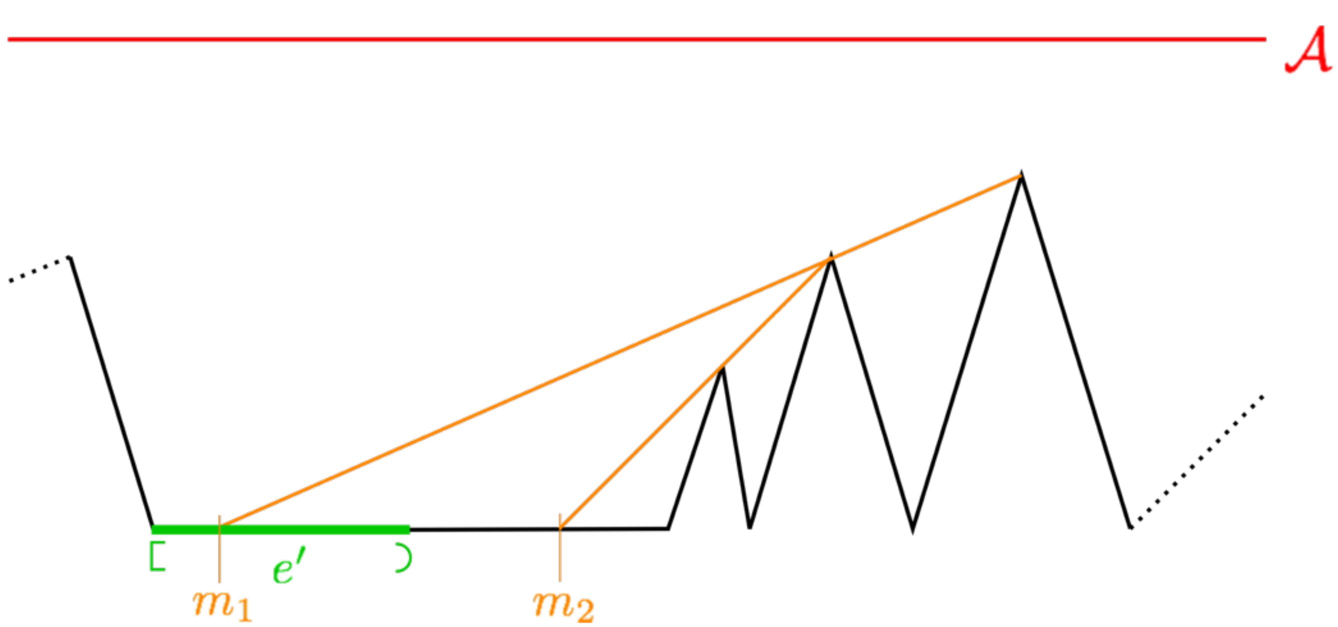
How do we preprocess our terrain to easily identify the point on $\mathcal{A}$ that we need to add to C when we split an edge?

- Sweep rightmost to leftmost vertex

Sweep Algorithm

How do we preprocess our terrain to easily identify the point on $\mathcal{A}$ that we need to add to C when we split an edge?

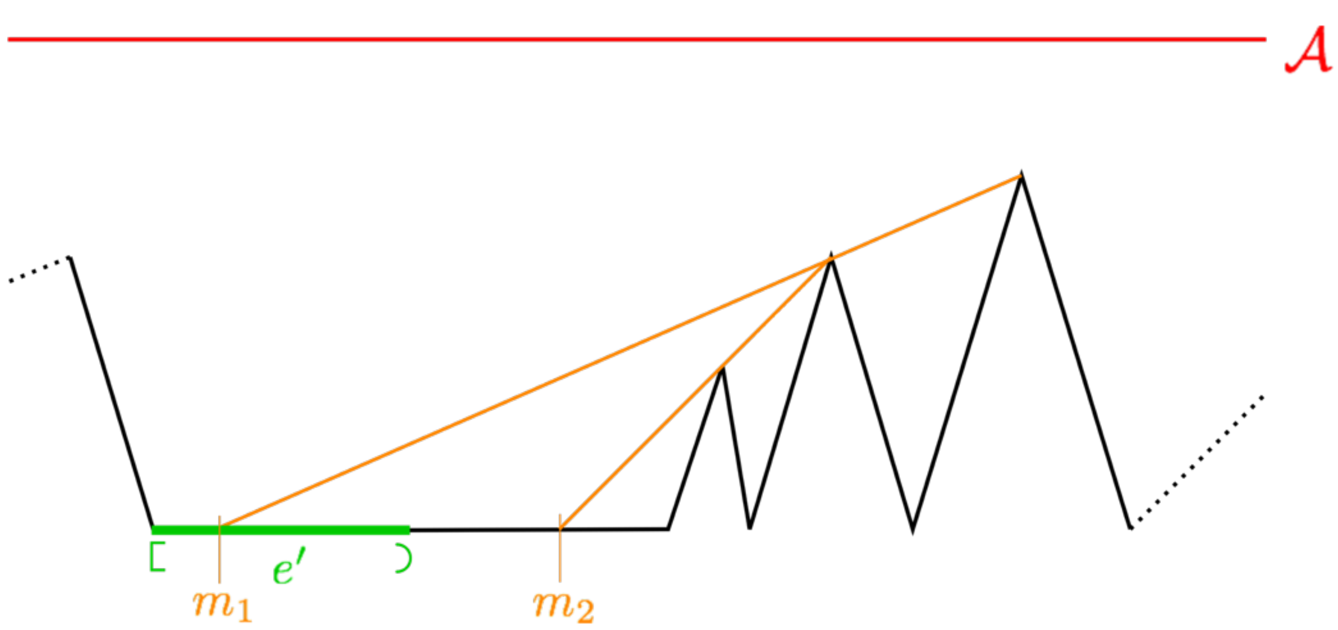
- Sweep rightmost to leftmost vertex
- For each vertex we shoot ray through all vertices to its left



Sweep Algorithm

How do we preprocess our terrain to easily identify the point on $\mathcal{A}$ that we need to add to C when we split an edge?

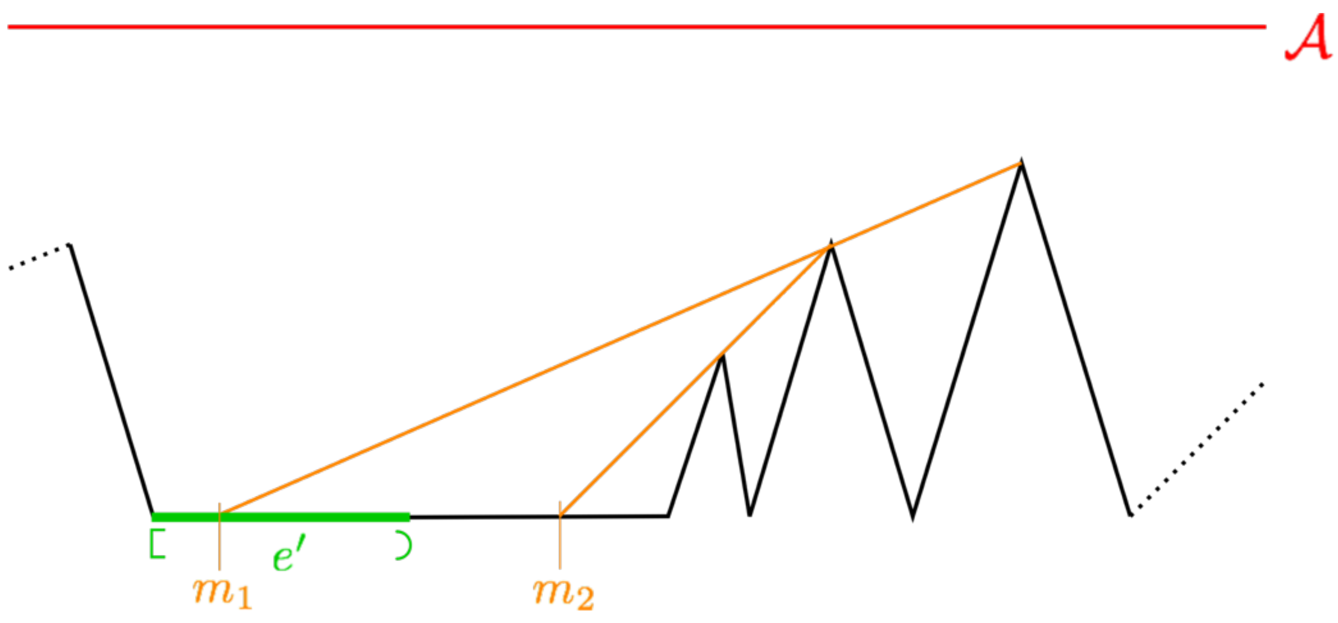
- Sweep rightmost to leftmost vertex
- For each vertex we shoot ray through all vertices to its left
- Where rays hit terrain: *mark points*



Sweep Algorithm

How do we preprocess our terrain to easily identify the point on $\mathcal{A}$ that we need to add to C when we split an edge?

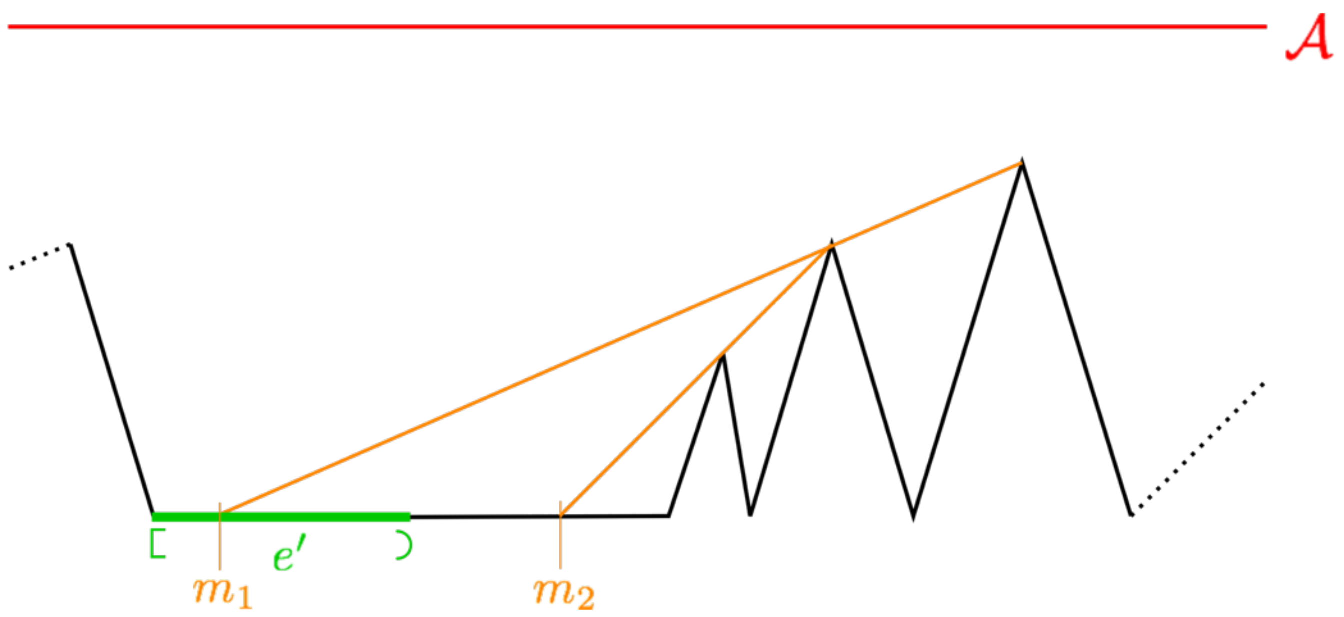
- Sweep rightmost to leftmost vertex
- For each vertex we shoot ray through all vertices to its left
- Where rays hit terrain: *mark points*
- $O(n^2)$ preprocessed intervals



Sweep Algorithm

How do we preprocess our terrain to easily identify the point on $\mathcal{A}$ that we need to add to C when we split an edge?

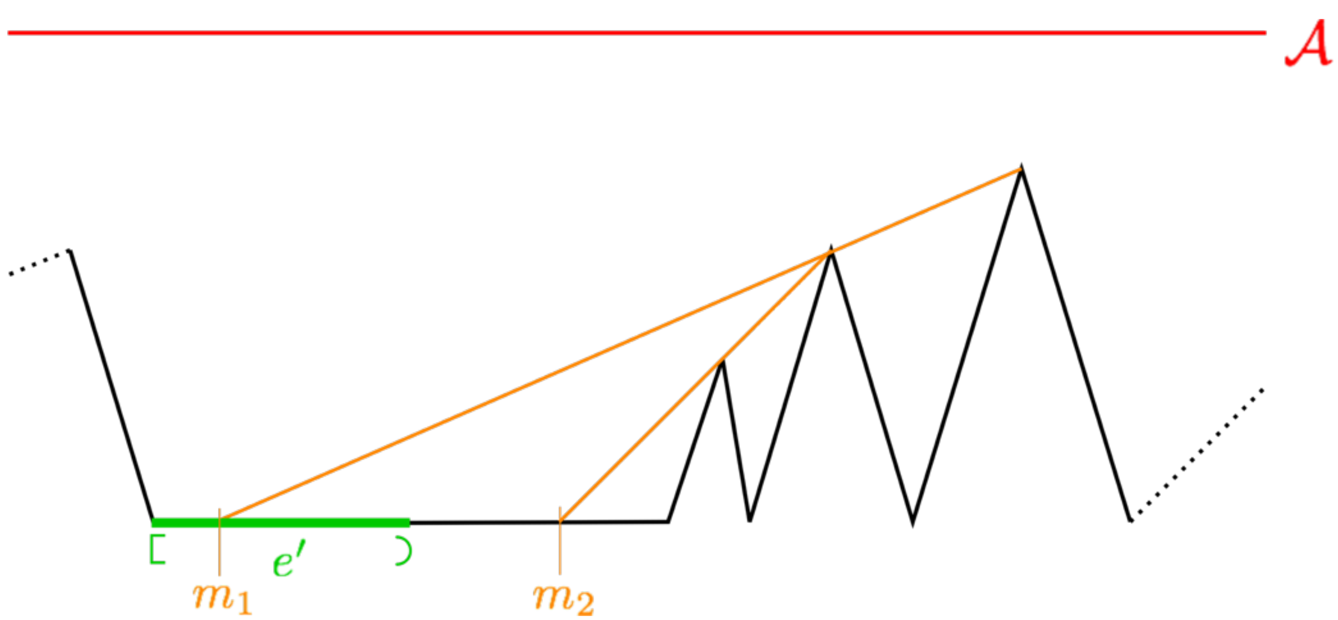
- Sweep rightmost to leftmost vertex
- For each vertex we shoot ray through all vertices to its left
- Where rays hit terrain: *mark points*
- O(n²) preprocessed intervals
- For each mark point *m* remember the rightmost of the two ray-vertices v_m

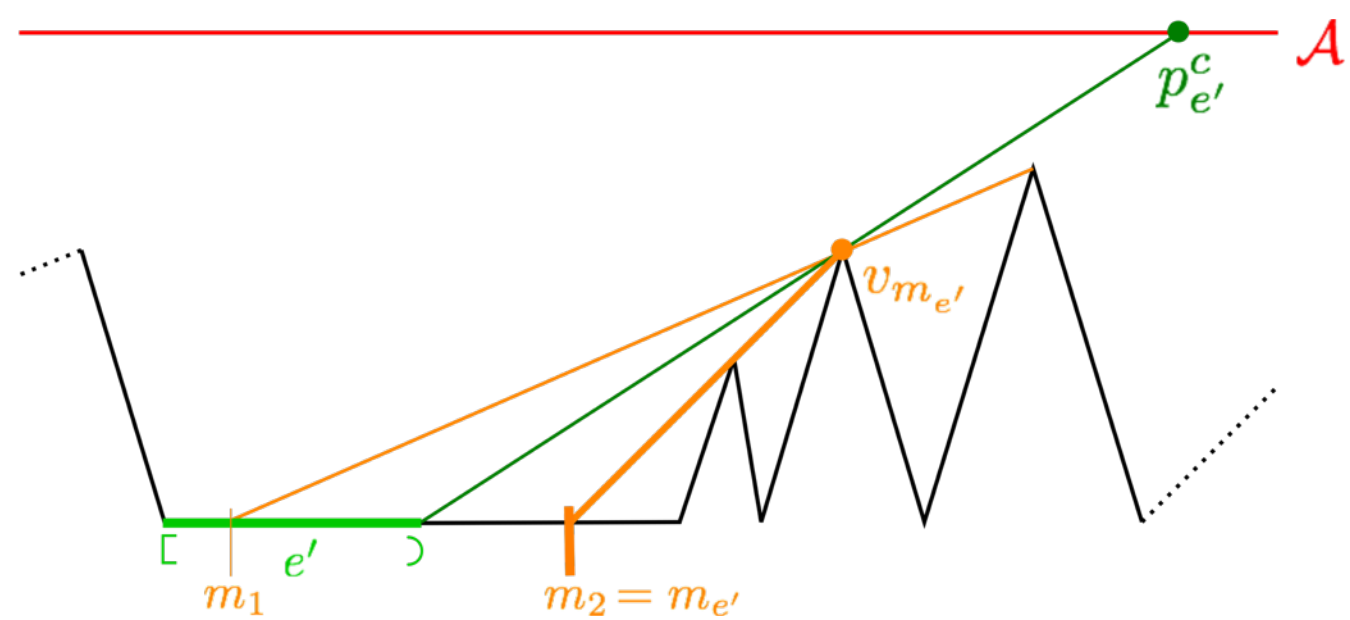


Sweep Algorithm

How do we preprocess our terrain to easily identify the point on $\mathcal{A}$ that we need to add to C when we split an edge?

- Sweep rightmost to leftmost vertex
- For each vertex we shoot ray through all vertices to its left
- Where rays hit terrain: *mark points*
- $O(n^2)$ preprocessed intervals
- For each mark point m remember the rightmost of the two ray-vertices v_m
- When placing guard g splits edge e, and we are left with interval e'⊂e:

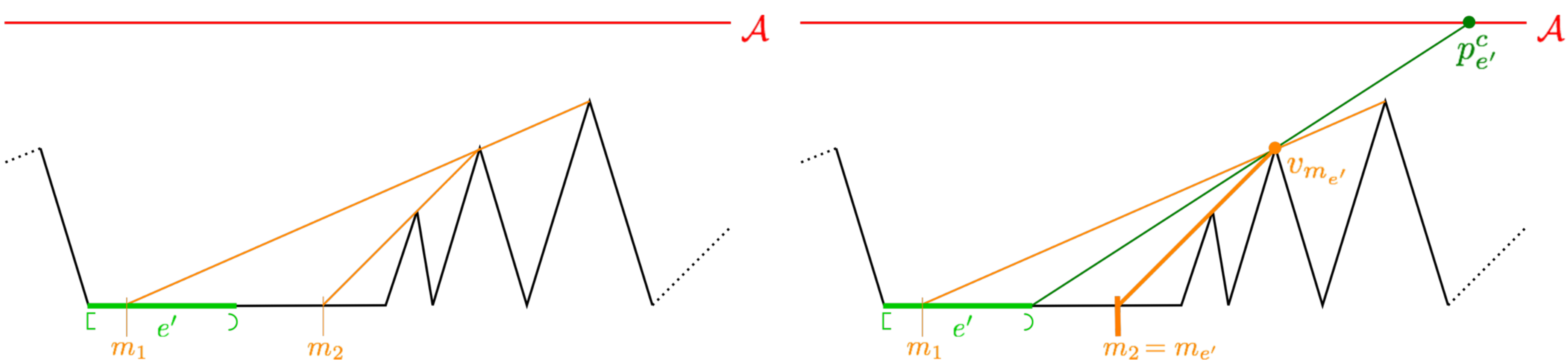




Sweep Algorithm

How do we preprocess our terrain to easily identify the point on $\mathcal{A}$ that we need to add to C when we split an edge?

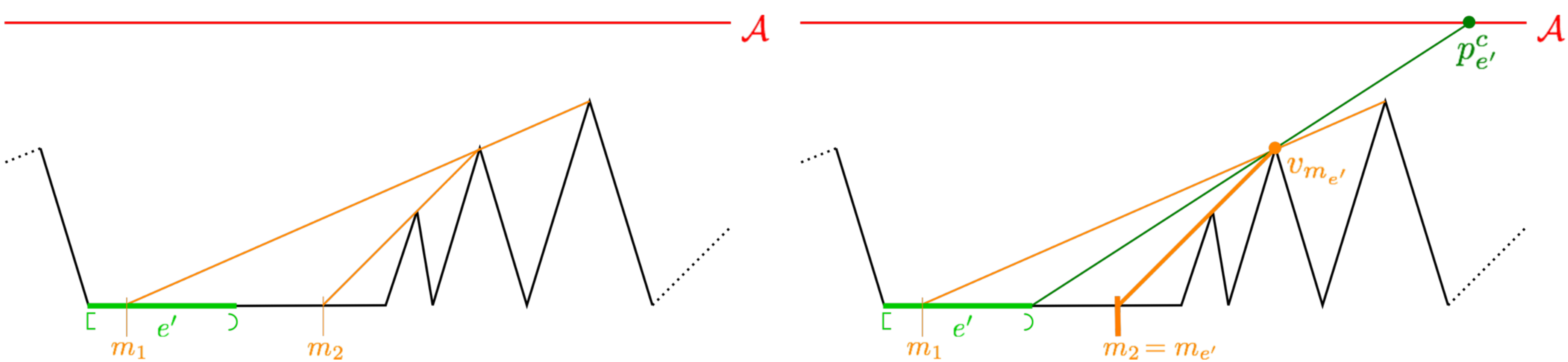
- Sweep rightmost to leftmost vertex
- For each vertex we shoot ray through all vertices to its left
- Where rays hit terrain: *mark points*
- $O(n^2)$ preprocessed intervals
- For each mark point m remember the rightmost of the two ray-vertices v_m
- When placing guard g splits edge e, and we are left with interval $e' \subset e$:
    - Identify mark point, $m_{e'}$, to the right of e'



Sweep Algorithm

How do we preprocess our terrain to easily identify the point on $\mathcal{A}$ that we need to add to C when we split an edge?

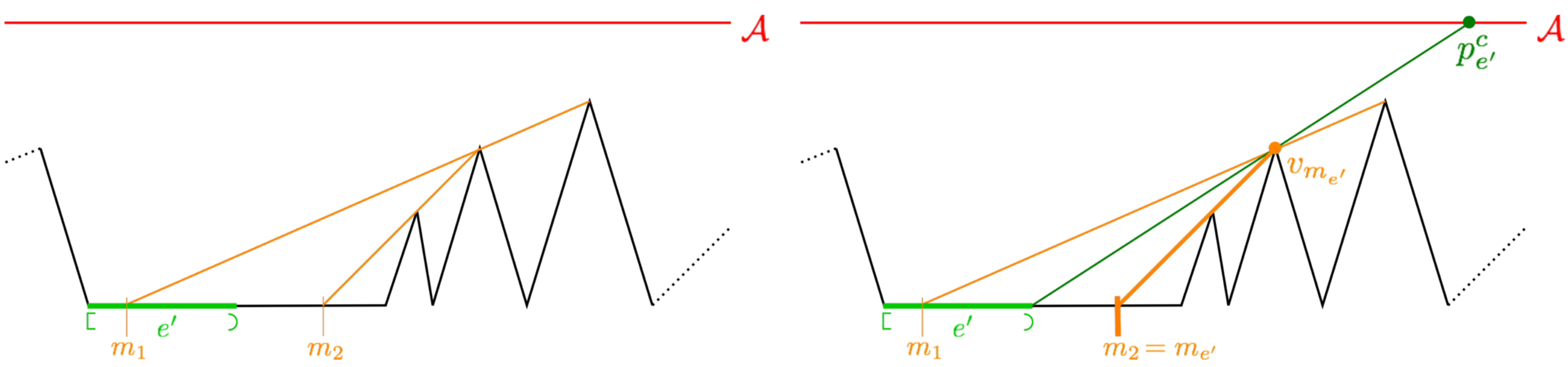
- Sweep rightmost to leftmost vertex
- For each vertex we shoot ray through all vertices to its left
- Where rays hit terrain: *mark points*
- $O(n^2)$ preprocessed intervals
- For each mark point *m* remember the rightmost of the two ray-vertices v_m
- When placing guard g splits edge e, and we are left with interval $e' \subset e$:
  - Identify mark point, $m_{e'}$, to the right of e'
  - Shoot ray *r* from right endpoint of e' through $v_{m_{e'}}$



Sweep Algorithm

How do we preprocess our terrain to easily identify the point on $\mathcal{A}$ that we need to add to C when we split an edge?

- Sweep rightmost to leftmost vertex
- For each vertex we shoot ray through all vertices to its left
- Where rays hit terrain: *mark points*
- $O(n^2)$ preprocessed intervals
- For each mark point m remember the rightmost of the two ray-vertices v_m
- When placing guard g splits edge e, and we are left with interval $e' \subset e$:
    - Identify mark point, $m_{e'}$, to the right of e'
    - Shoot ray r from right endpoint of e' through $v_{m_{e'}}$
    - Intersection point of r and $\mathcal{A}$ is the new closing point



Lemma 1: The set G output by the algorithm is feasible.

Sweep Algorithm

Lemma 1: The set G output by the algorithm is feasible.

Theorem 2: The set G output by the algorithm is optimal.

Sweep Algorithm

Lemma 1: The set G output by the algorithm is feasible.

Theorem 2: The set G output by the algorithm is optimal.

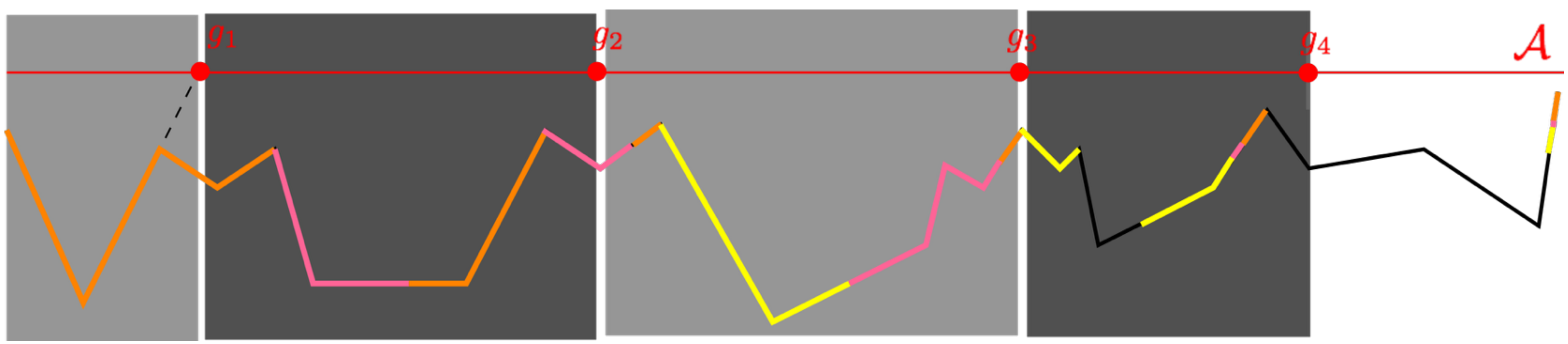
Proof idea: If we can find a witness set with $|W|=|G|$, G is optimal.

Sweep Algorithm

Lemma 1: The set G output by the algorithm is feasible.

Theorem 2: The set G output by the algorithm is optimal.

Proof idea: If we can find a witness set with |W|=|G|, G is optimal.
Let S_i be the strip of all points with x -coordinates between $x(g_{i-1}) + \varepsilon$ and $x(g_i)$.

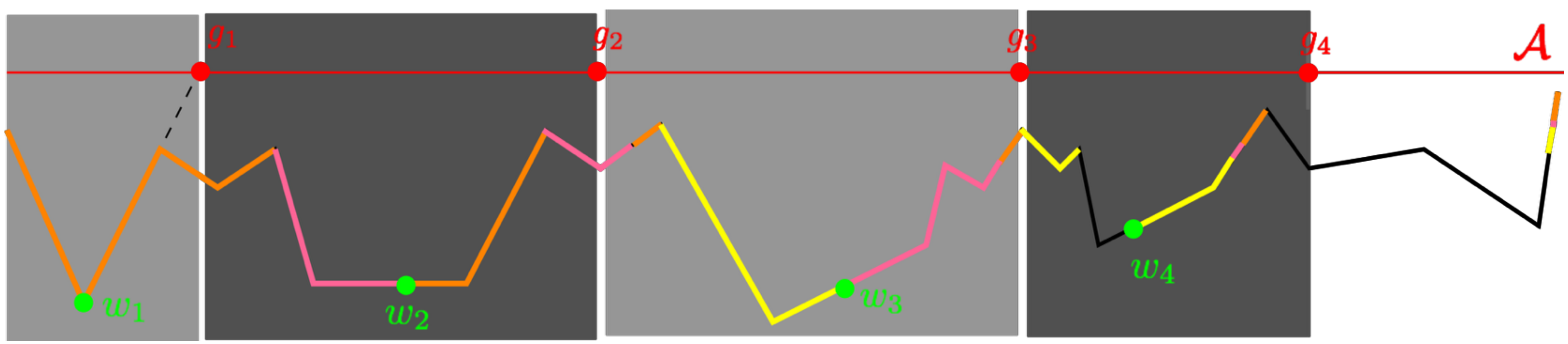


Sweep Algorithm

Lemma 1: The set G output by the algorithm is feasible.

Theorem 2: The set G output by the algorithm is optimal.

Proof idea: If we can find a witness set with |W|=|G|, G is optimal.
Let S_i be the strip of all points with x -coordinates between $x(g_{i-1}) + \varepsilon$ and $x(g_i)$.
We place a witness w_i per guard g_i such that $V_T(w_i) \subseteq S_i$ $\forall i$.

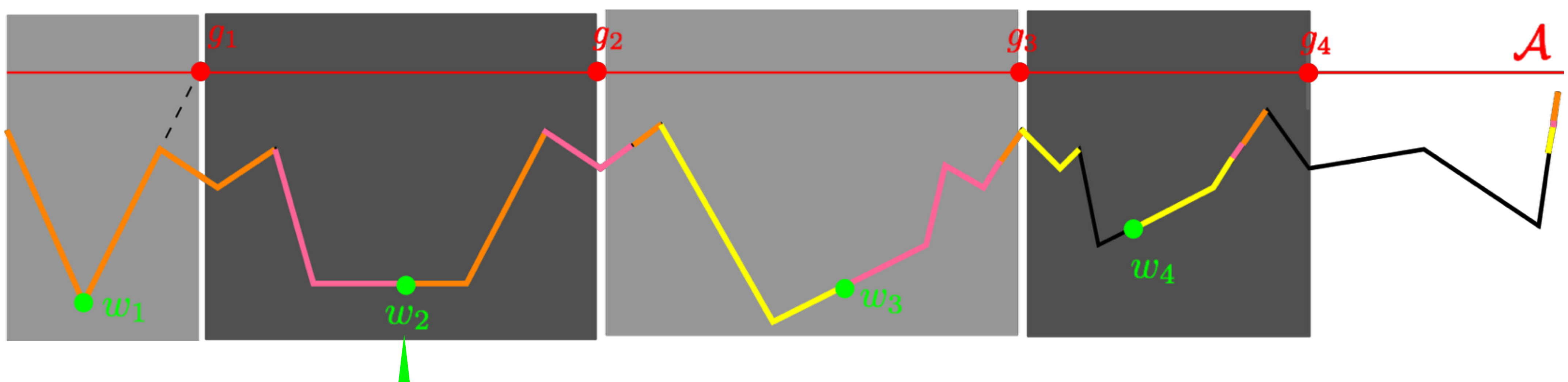


Sweep Algorithm

Lemma 1: The set G output by the algorithm is feasible.

Theorem 2: The set G output by the algorithm is optimal.

Proof idea: If we can find a witness set with |W|=|G|, G is optimal.
Let S_i be the strip of all points with x -coordinates between $x(g_{i-1}) + \varepsilon$ and $x(g_i)$.
We place a witness w_i per guard g_i such that $V_T(w_i) \subseteq S_i$ $\forall i$.



e'=$[v_j, q)$ for some point $q \in e_j$, $q \neq v_{j+1}$

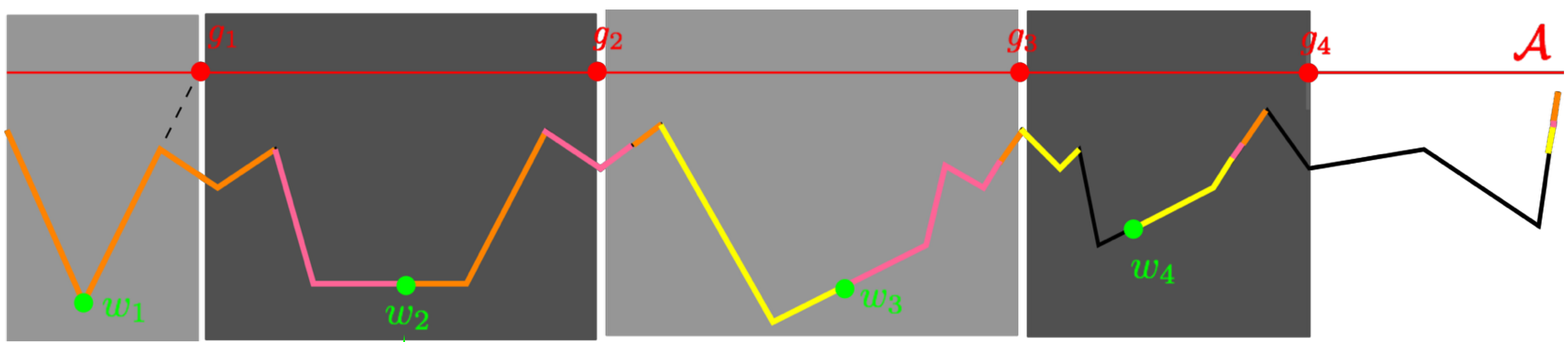
we place witness at q^{ε}, a point ε to the left of q on T

Sweep Algorithm

Lemma 1: The set G output by the algorithm is feasible.

Theorem 2: The set G output by the algorithm is optimal.

Proof idea: If we can find a witness set with |W|=|G|, G is optimal.
Let S_i be the strip of all points with x -coordinates between $x(g_{i-1}) + \varepsilon$ and $x(g_i)$.
We place a witness w_i per guard g_i such that $V_T(w_i) \subseteq S_i$ $\forall i$.



$e' = [v_j, q)$ for some point $q \in e_j$, $q \neq v_{j+1}$

we place witness at q^{ε}, a point ε to the left of q on T

Theorem 3: Uni-monotone polygons are perfect.

Runtime

- Preprocessing: mark points $O(n^2)$

Runtime

- Preprocessing: mark points $O(n^2)$
- Accordingly: mark points from left for opening points (using left vertex of an edge to shoot the ray)

Runtime

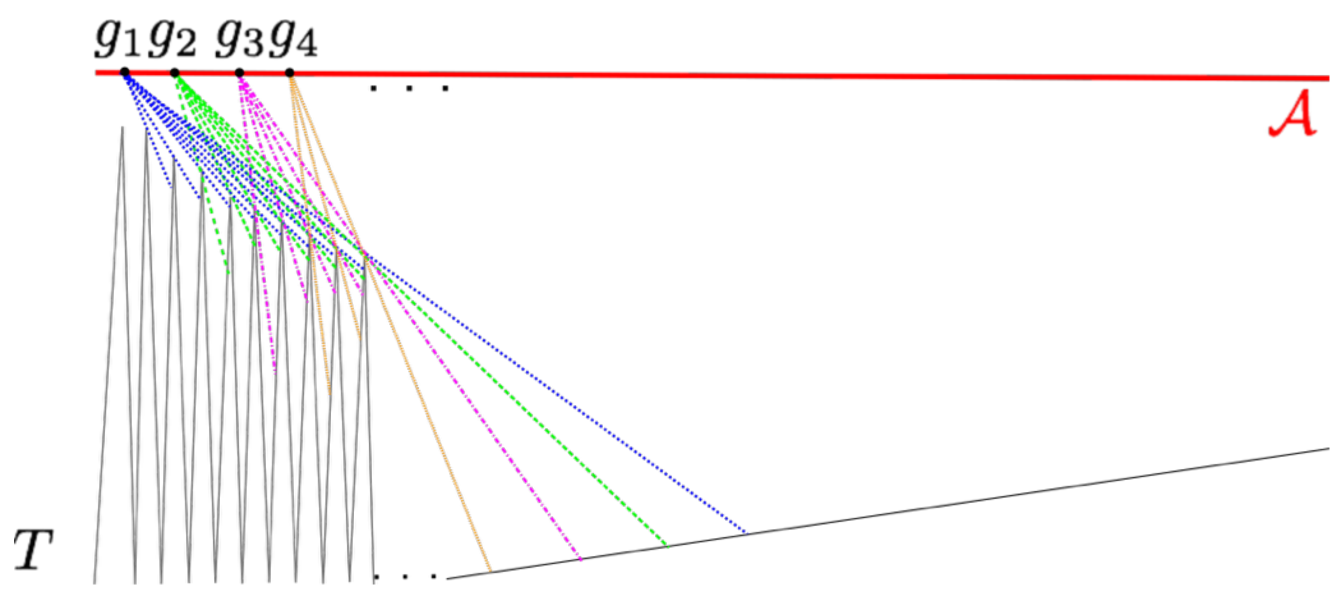
- Preprocessing: mark points $O(n^2)$
- Accordingly: mark points from left for opening points (using left vertex of an edge to shoot the ray)
- and for soft opening points (using right vertex of an edge to shoot the ray)

Runtime

- Preprocessing: mark points $O(n^2)$
- Accordingly: mark points from left for opening points (using left vertex of an edge to shoot the ray)
- and for soft opening points (using right vertex of an edge to shoot the ray)
- Whenever we insert a guard

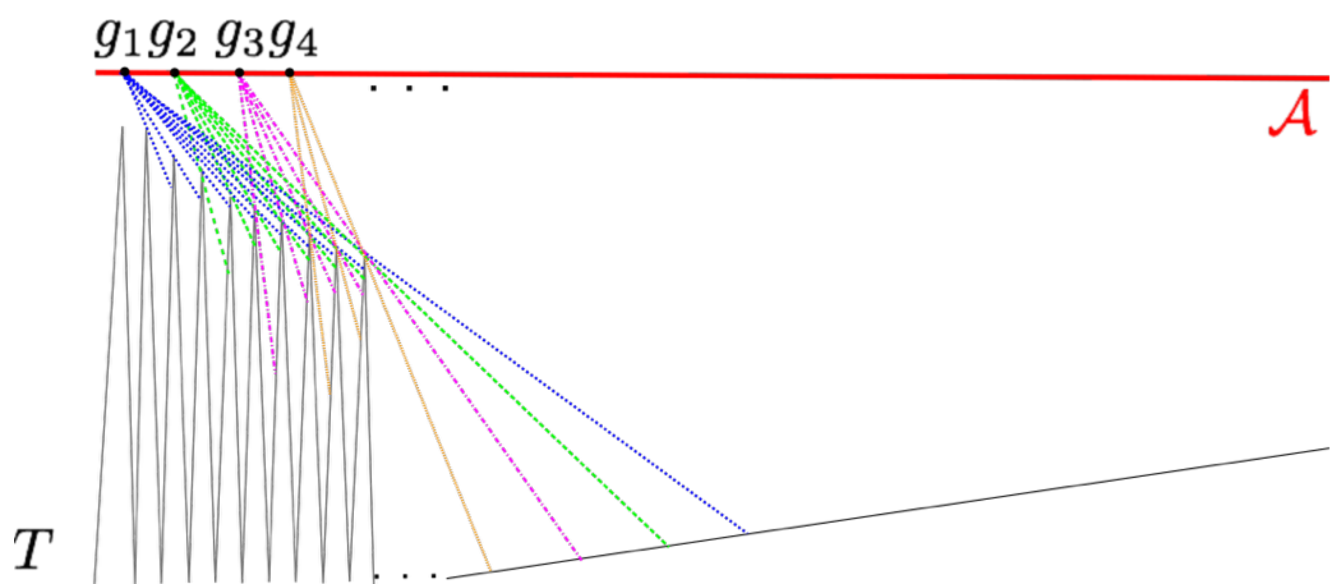
Runtime

- Preprocessing: mark points $O(n^2)$
- Accordingly: mark points from left for opening points (using left vertex of an edge to shoot the ray)
- and for soft opening points (using right vertex of an edge to shoot the ray)
- Whenever we insert a guard
    - Shoot $O(n)$ rays



Runtime

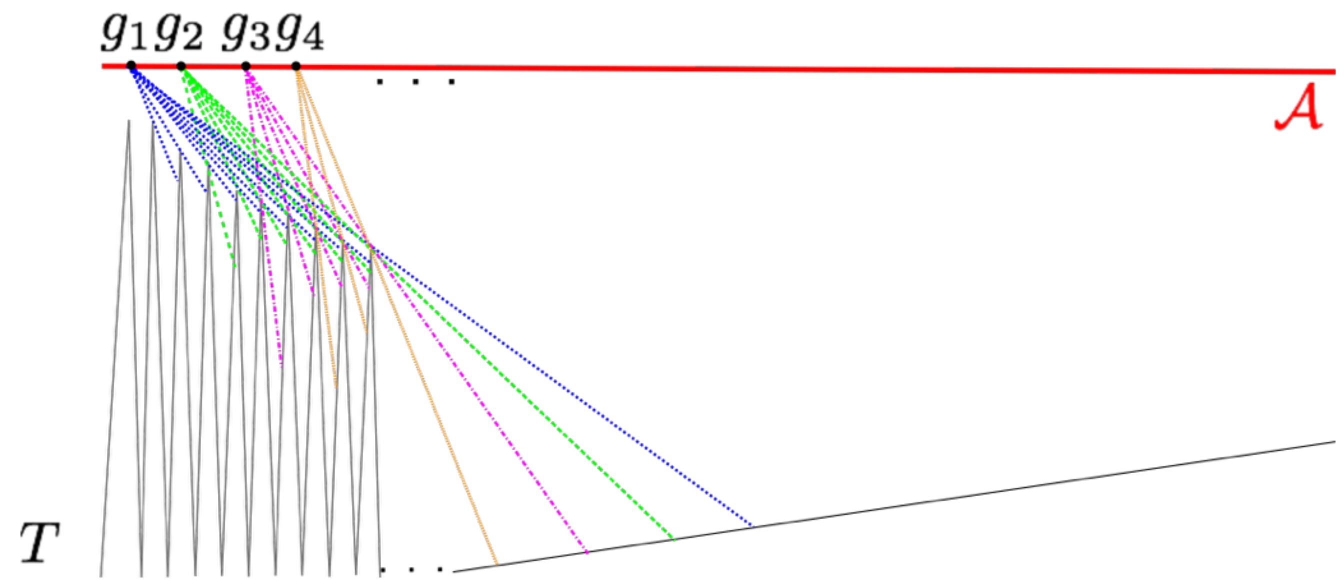
- Preprocessing: mark points $O(n^2)$
- Accordingly: mark points from left for opening points (using left vertex of an edge to shoot the ray)
- and for soft opening points (using right vertex of an edge to shoot the ray)
- Whenever we insert a guard
  - Shoot $O(n)$ rays
  - For each of intersection points r_e: shoot ray through $v_{m_{e'}}$



Runtime

- Preprocessing: mark points $O(n^2)$
- Accordingly: mark points from left for opening points (using left vertex of an edge to shoot the ray)
- and for soft opening points (using right vertex of an edge to shoot the ray)
- Whenever we insert a guard
    - Shoot $O(n)$ rays
    - For each of intersection points r_e: shoot ray through $v_{m_{e'}}$

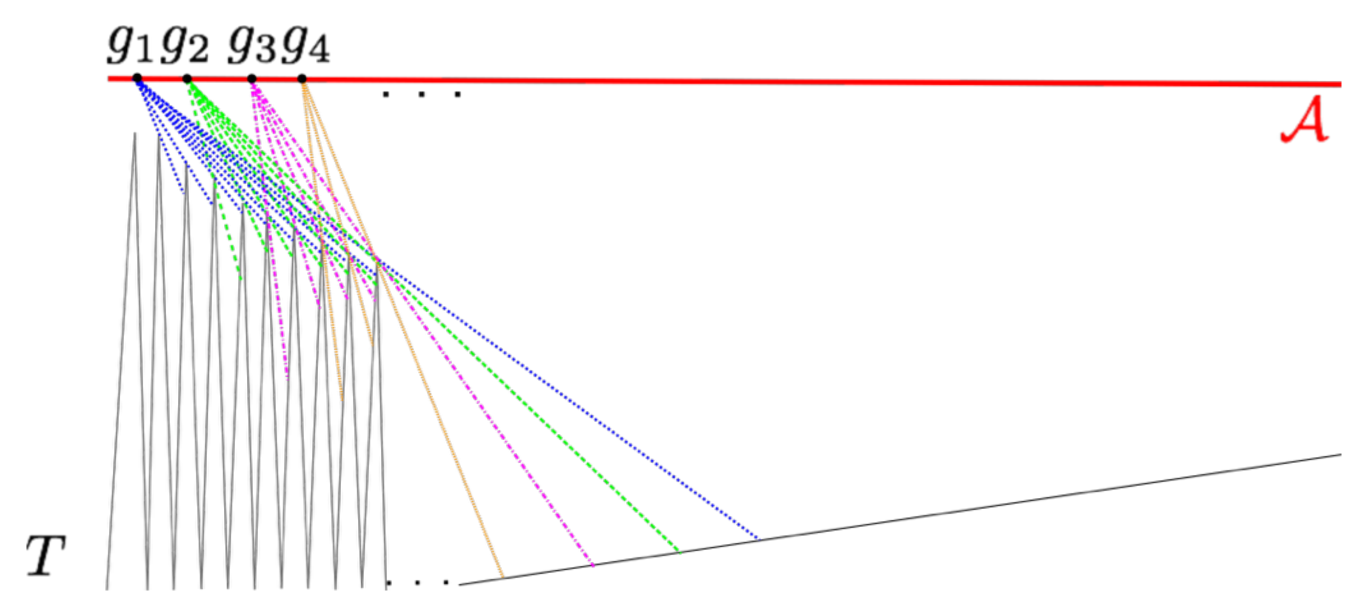
➡ $O(n^2 \log n)$ - not optimized



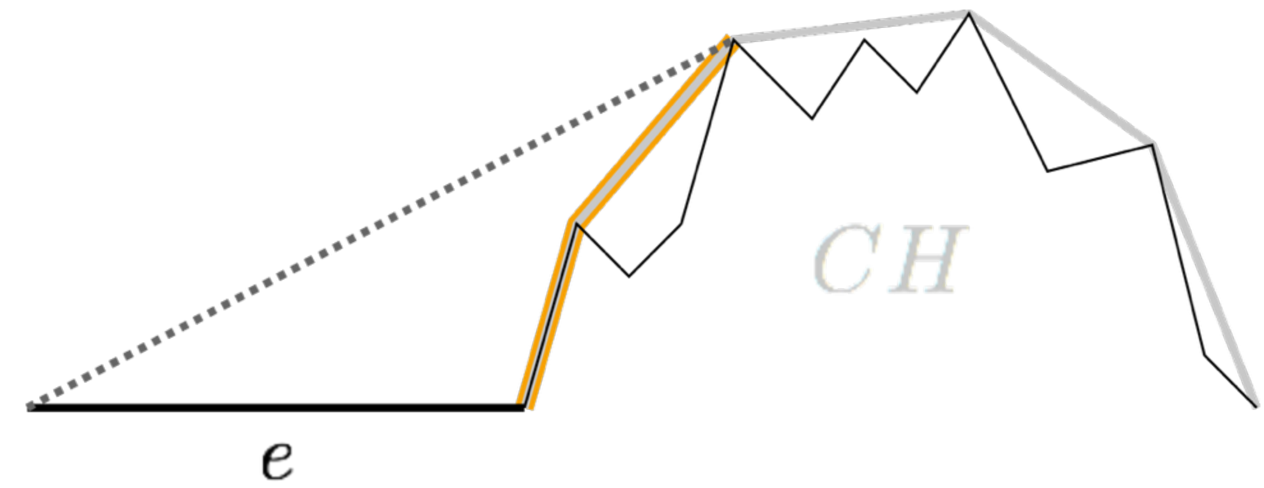
Runtime

- Preprocessing: mark points ~~$O(n^2)$~~ $O(n)$
- Accordingly: mark points from left for opening points (using left vertex of an edge to shoot the ray)
- and for soft opening points (using right vertex of an edge to shoot the ray)
- Whenever we insert a guard
  - Shoot $O(n)$ rays
  - For each of intersection points r_e: shoot ray through $v_{m_{e'}}$

➡ $O(n^2 \log n)$ - not optimized



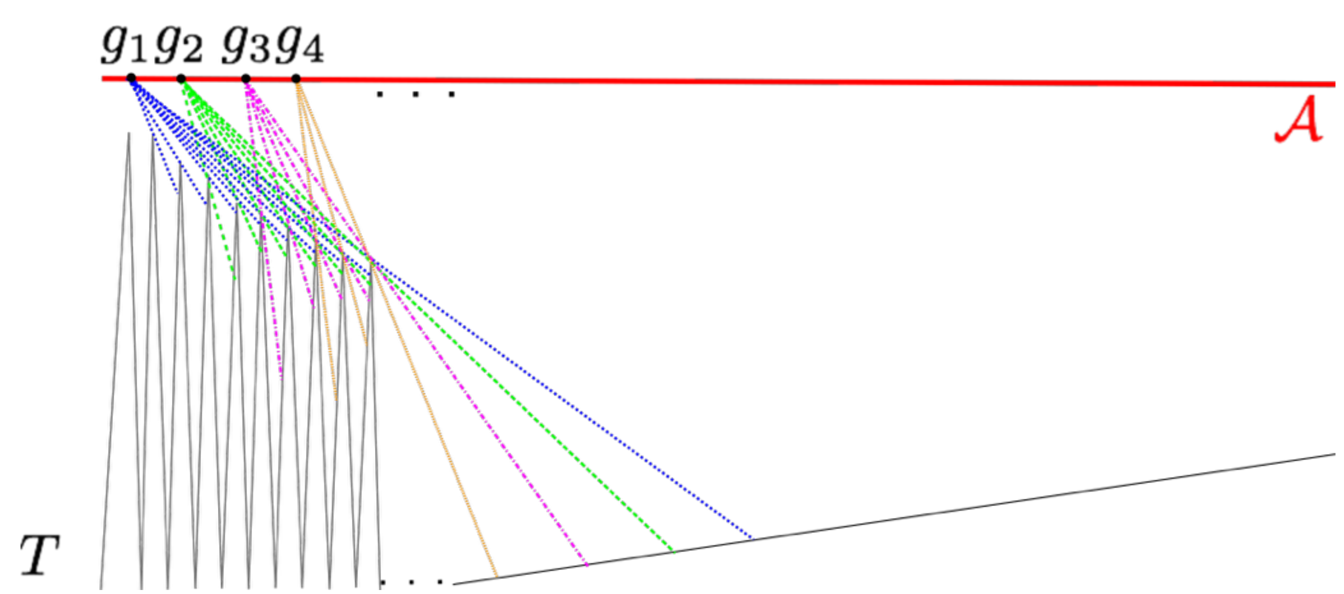
Runtime



O(n)

- Preprocessing: mark points $\cancel{O(n^2)}$
- Accordingly: mark points from left for opening points (using left vertex of an edge to shoot the ray)
- and for soft opening points (using right vertex of an edge to shoot the ray)
- Whenever we insert a guard
  - Shoot O(n) rays
  - For each of intersection points r_e: shoot ray through $v_{m_{e'}}$

➡ $O(n^2 \log n)$ - not optimized

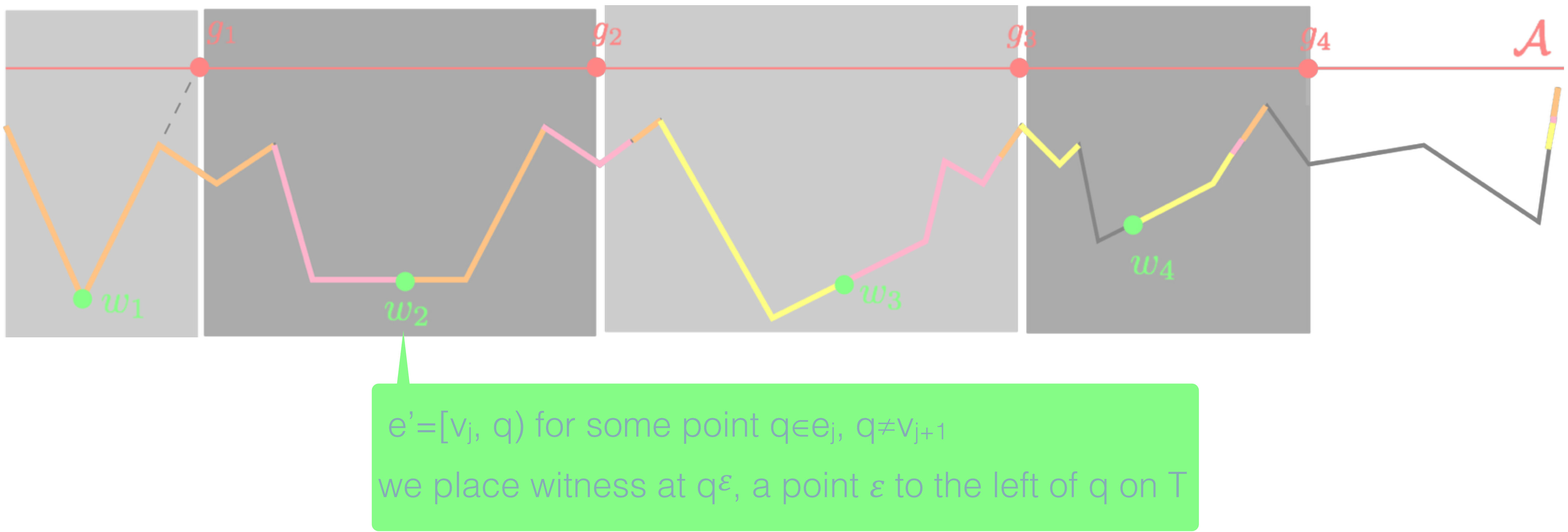


Lemma 1: The set G output by the algorithm is feasible.

Theorem 2: The set G output by the algorithm is optimal.

Proof idea: If we can find a witness set with |W|=|G|, G is optimal.
Let S_i be the strip of all points with x -coordinates between $x(g_{i-1}) + \varepsilon$ and $x(g_i)$.
We place a witness w_i per guard g_i such that $V_T(w_i) \subseteq S_i$ $\forall i$.

Thanks.



Theorem 3: Uni-monotone polygons are perfect.