Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Chair for
INFORMATICS I
Efficient Algorithms and
Knowledge-Based Systems

Institute for Informatics

Computational Geometry

Triangulating Polygons
or
Guarding Art Galleries

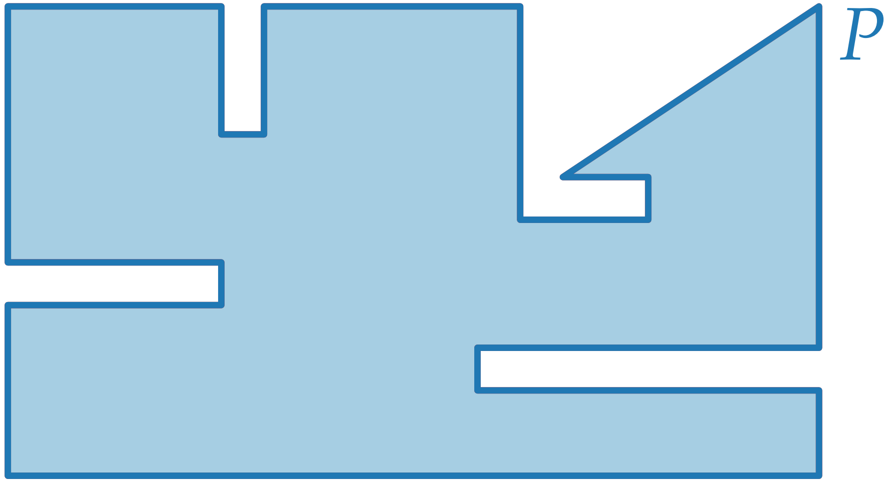
Lecture #2

Thomas van Dijk

Winter Semester 2019/20

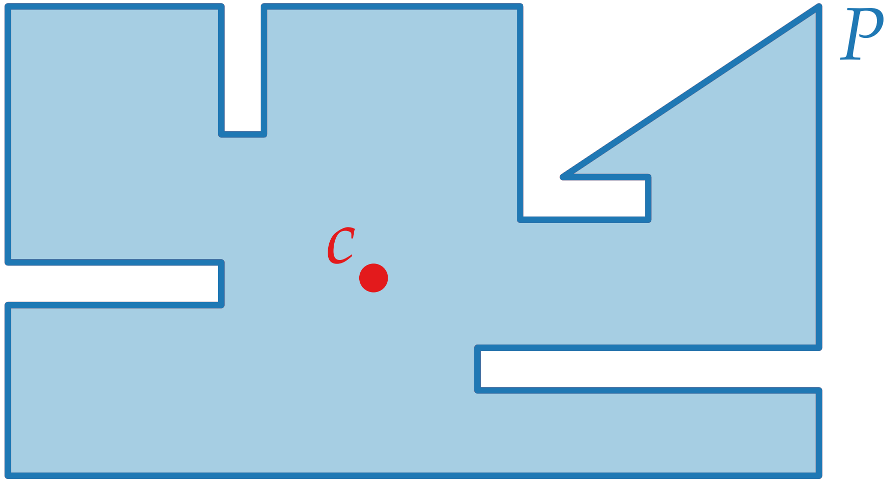
Guarding an Art Gallery

Given a *simple* polygon P (i.e., no holes, no self-intersection)...



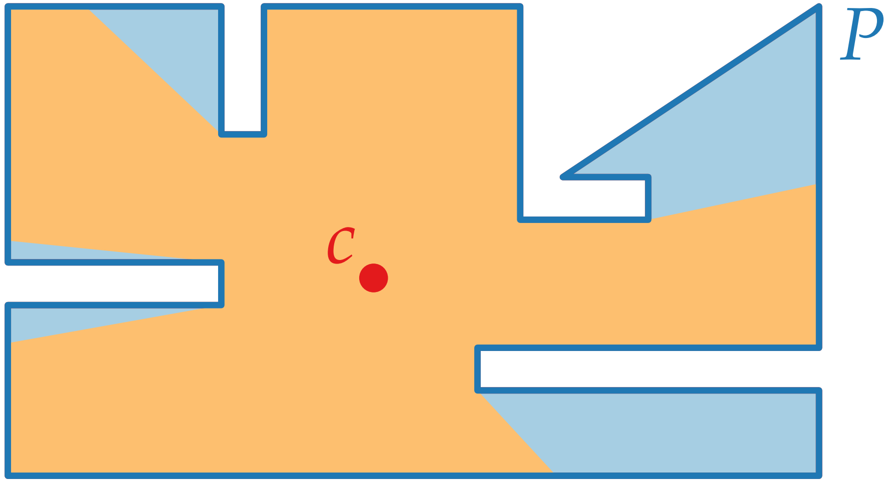
Guarding an Art Gallery

Given a *simple* polygon P (i.e., no holes, no self-intersection)...



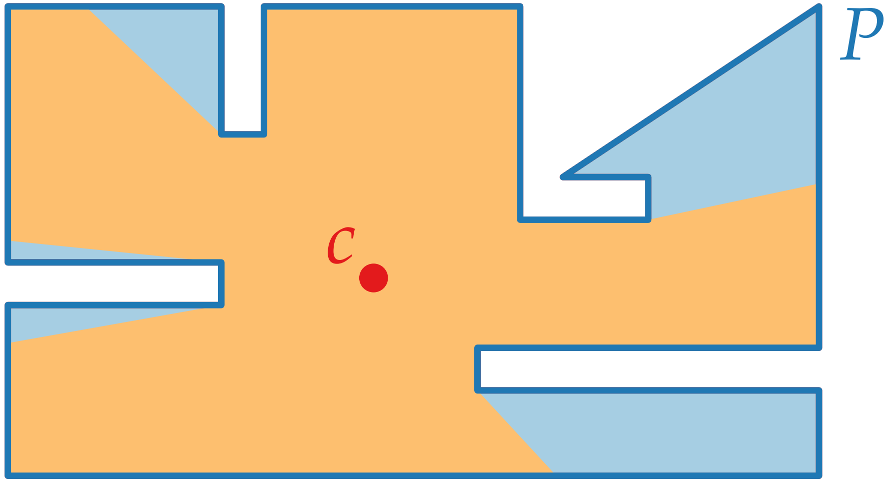
Guarding an Art Gallery

Given a *simple* polygon P (i.e., no holes, no self-intersection)...



Guarding an Art Gallery

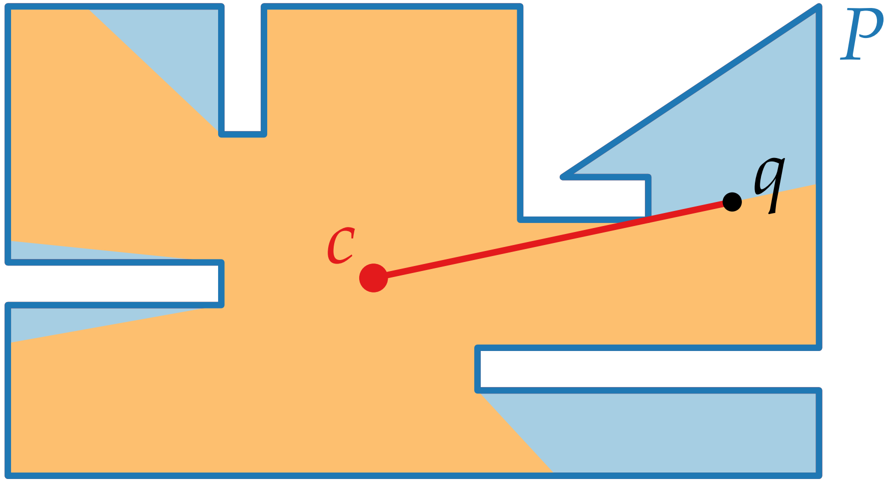
Given a *simple* polygon P (i.e., no holes, no self-intersection)...



Observation. Camera c "sees" a star-shaped region

Guarding an Art Gallery

Given a *simple* polygon P (i.e., no holes, no self-intersection)...

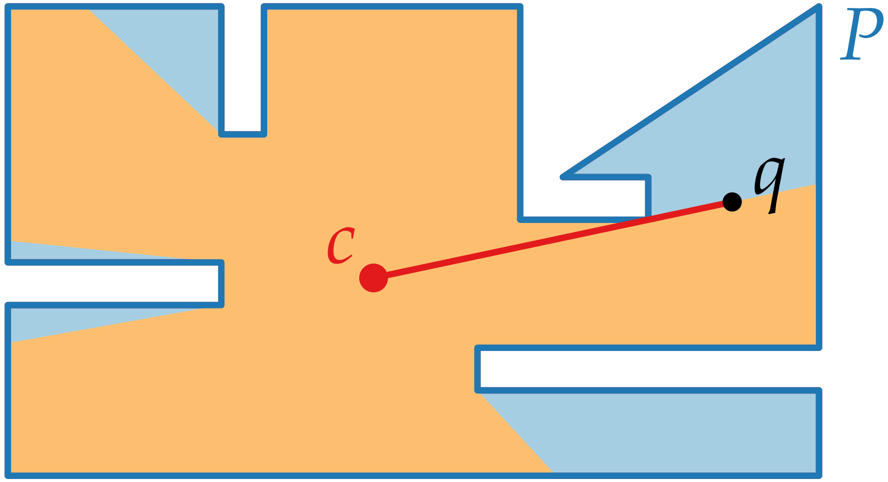


Observation. Camera c “sees” a star-shaped region

Definition. A pt $q \in P$ is *visible* from $c \in P$ if $\overline{qc} \subseteq P$.

Guarding an Art Gallery

Given a *simple* polygon P (i.e., no holes, no self-intersection)...



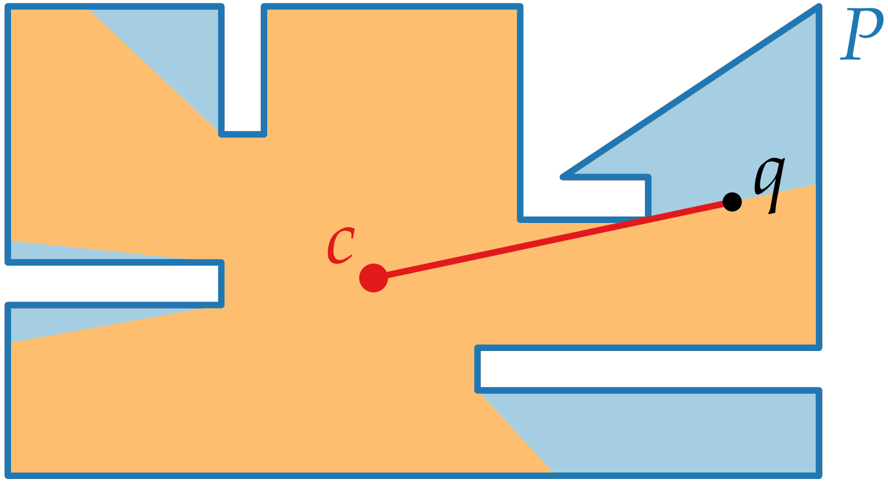
Observation. Camera c “sees” a star-shaped region

Definition. A pt $q \in P$ is *visible* from $c \in P$ if $\overline{qc} \subseteq P$.

Aim: Use few cameras!

Guarding an Art Gallery

Given a *simple* polygon P (i.e., no holes, no self-intersection)...



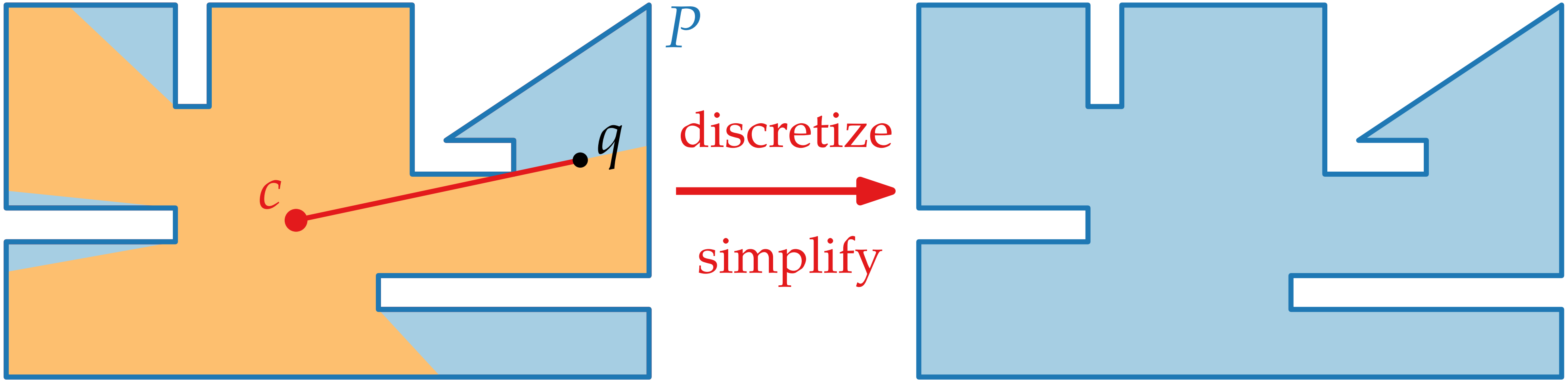
Observation. Camera c “sees” a star-shaped region

Definition. A pt $q \in P$ is *visible* from $c \in P$ if $\overline{qc} \subseteq P$.

Aim: Use few cameras! *But minimizing this is NP-hard...*

Guarding an Art Gallery

Given a *simple* polygon P (i.e., no holes, no self-intersection)...



discretize

simplify

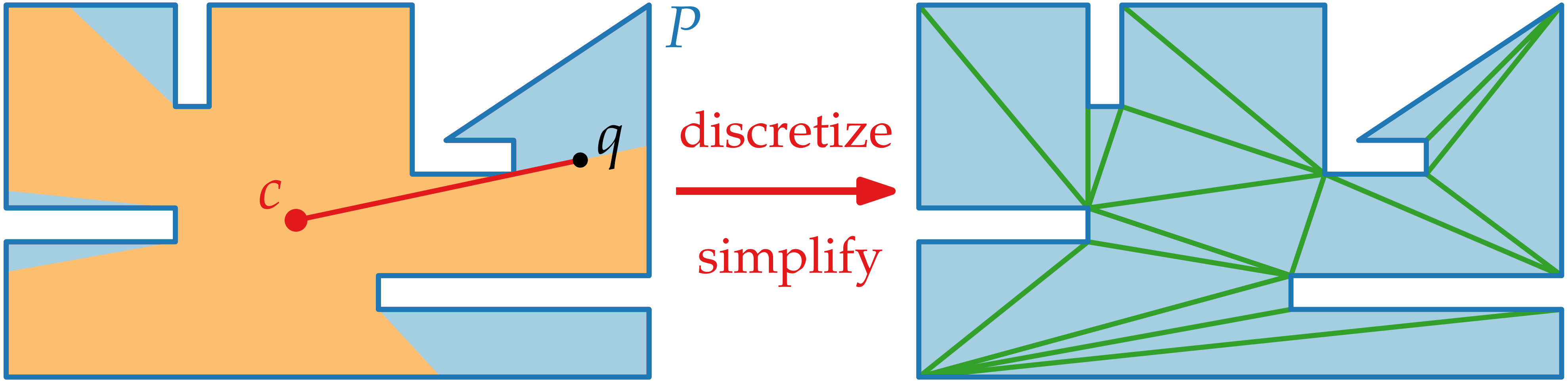
Observation. Camera c “sees” a star-shaped region

Definition. A pt $q \in P$ is *visible* from $c \in P$ if $\overline{qc} \subseteq P$.

Aim: Use few cameras! *But minimizing this is NP-hard...*

Guarding an Art Gallery

Given a *simple* polygon P (i.e., no holes, no self-intersection)...



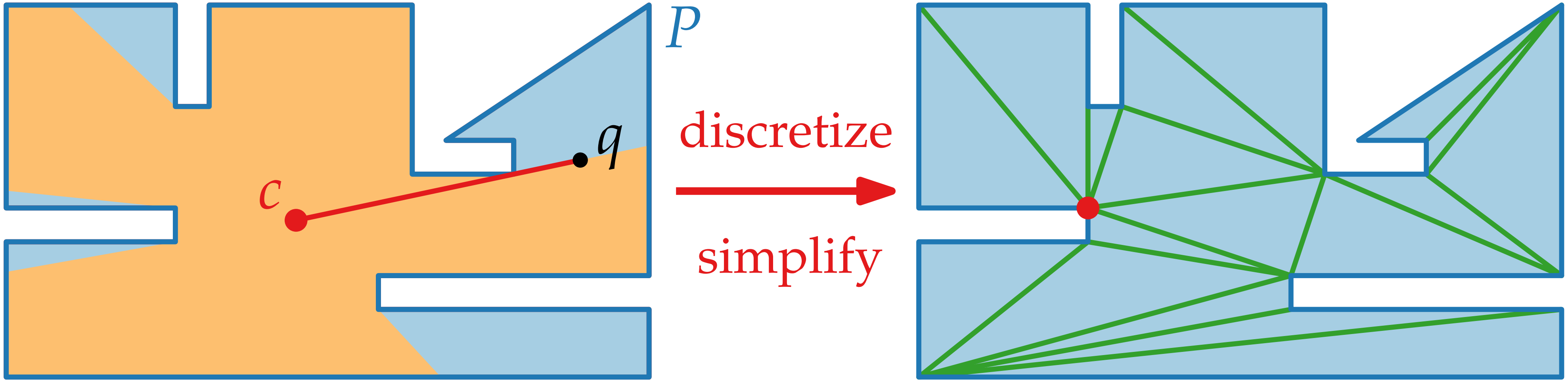
Observation. Camera c “sees” a star-shaped region

Definition. A pt $q \in P$ is *visible* from $c \in P$ if $\overline{qc} \subseteq P$.

Aim: Use few cameras! *But minimizing this is NP-hard. . .*

Guarding an Art Gallery

Given a *simple* polygon P (i.e., no holes, no self-intersection)...



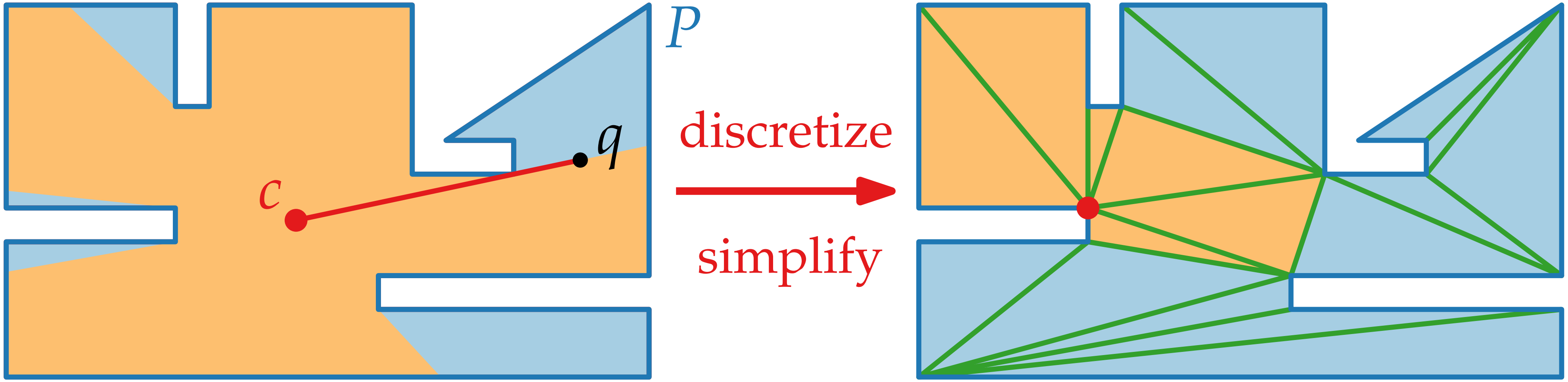
Observation. Camera c “sees” a star-shaped region

Definition. A pt $q \in P$ is *visible* from $c \in P$ if $\overline{qc} \subseteq P$.

Aim: Use few cameras! *But minimizing this is NP-hard…*

Guarding an Art Gallery

Given a *simple* polygon P (i.e., no holes, no self-intersection)...



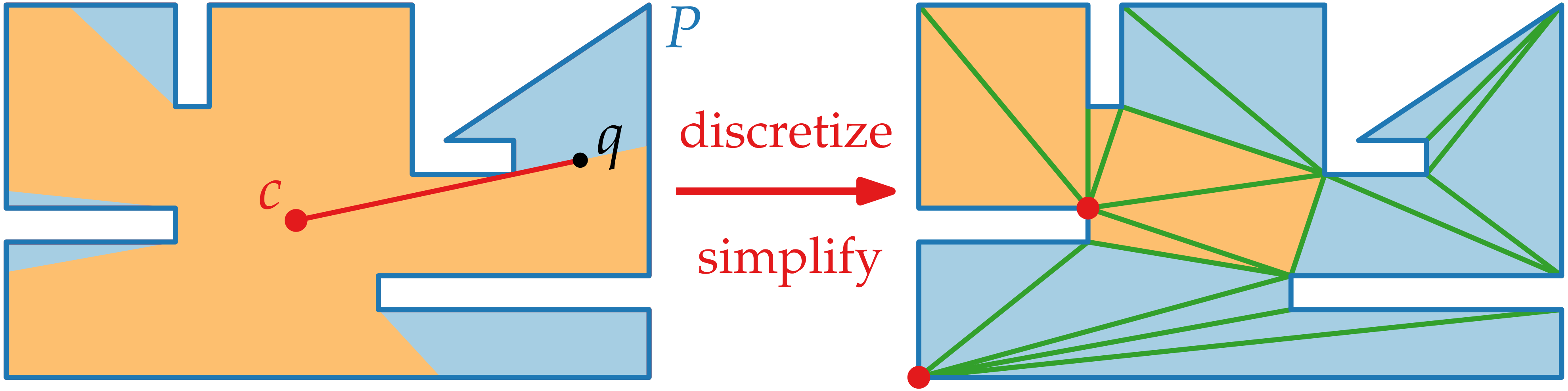
Observation. Camera c “sees” a star-shaped region

Definition. A pt $q \in P$ is *visible* from $c \in P$ if $\overline{qc} \subseteq P$.

Aim: Use few cameras! *But minimizing this is NP-hard. . .*

Guarding an Art Gallery

Given a *simple* polygon P (i.e., no holes, no self-intersection)...



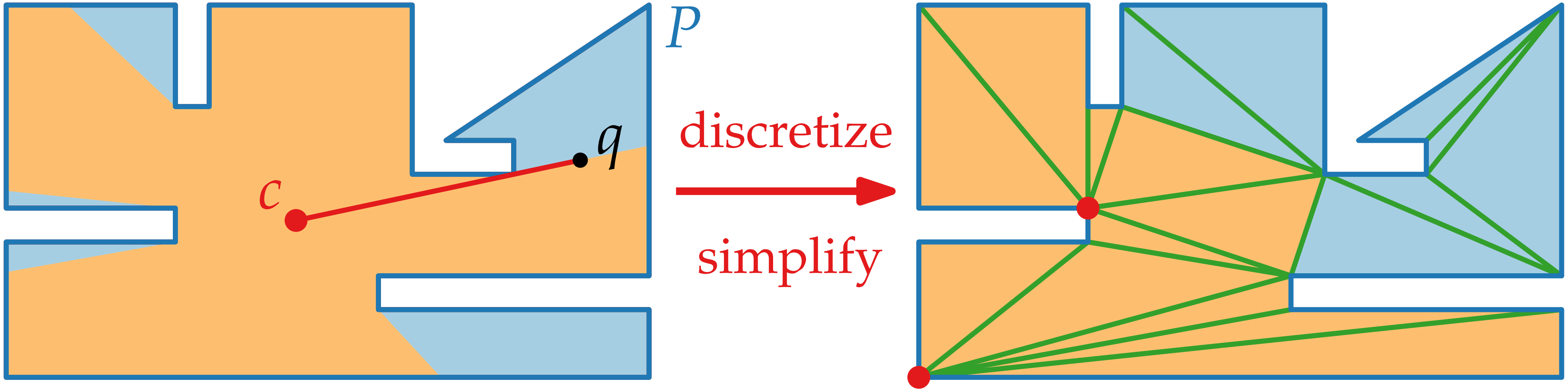
Observation. Camera c “sees” a star-shaped region

Definition. A pt $q \in P$ is *visible* from $c \in P$ if $\overline{qc} \subseteq P$.

Aim: Use few cameras! *But minimizing this is NP-hard…*

Guarding an Art Gallery

Given a *simple* polygon P (i.e., no holes, no self-intersection)...



discretize

simplify

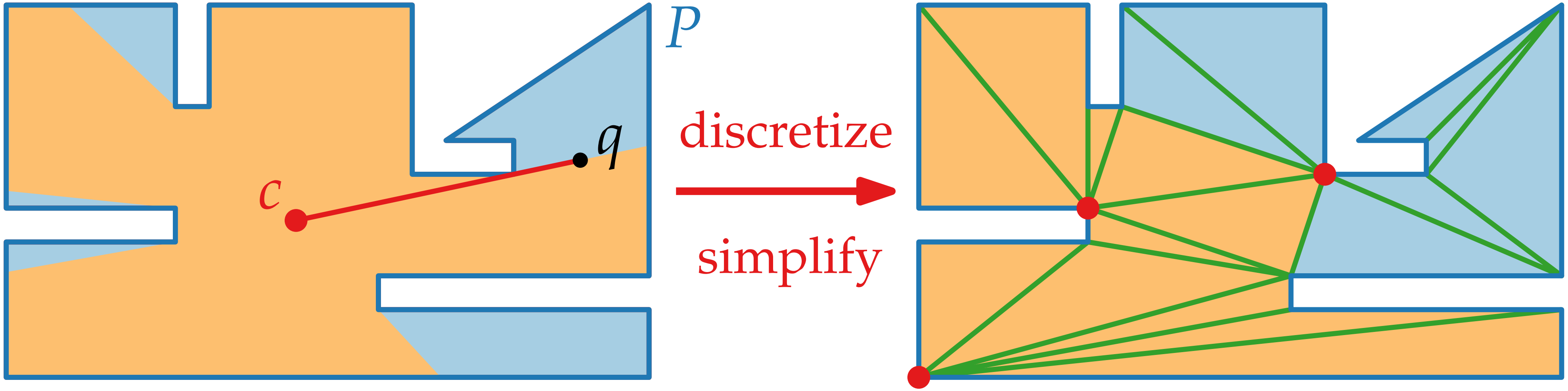
Observation. Camera c "sees" a star-shaped region

Definition. A pt $q \in P$ is *visible* from $c \in P$ if $\overline{qc} \subseteq P$.

Aim: Use few cameras! *But minimizing this is NP-hard. . .*

Guarding an Art Gallery

Given a *simple* polygon P (i.e., no holes, no self-intersection)...



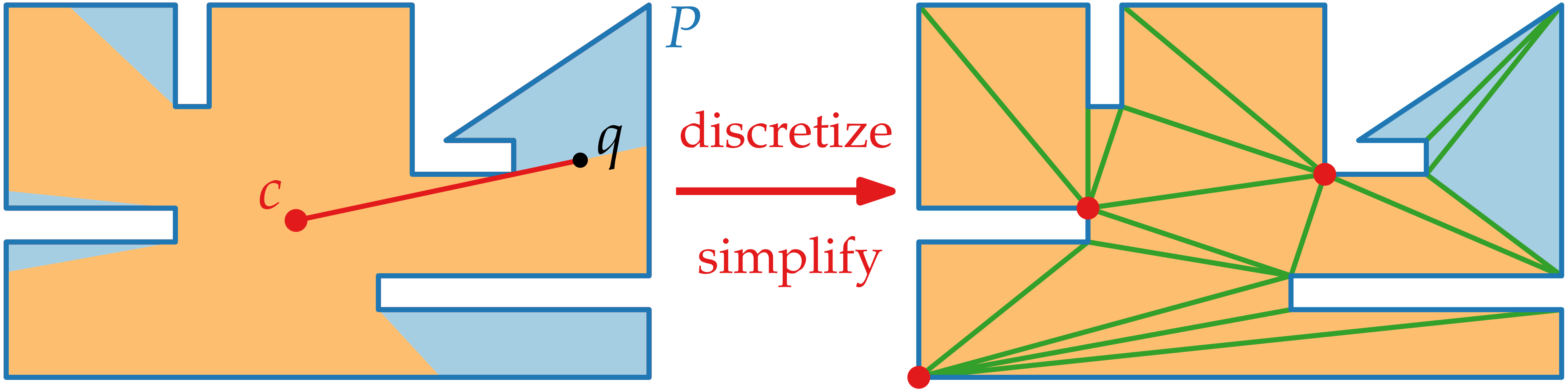
Observation. Camera c “sees” a star-shaped region

Definition. A pt $q \in P$ is *visible* from $c \in P$ if $\overline{qc} \subseteq P$.

Aim: Use few cameras! *But minimizing this is NP-hard…*

Guarding an Art Gallery

Given a *simple* polygon P (i.e., no holes, no self-intersection)...



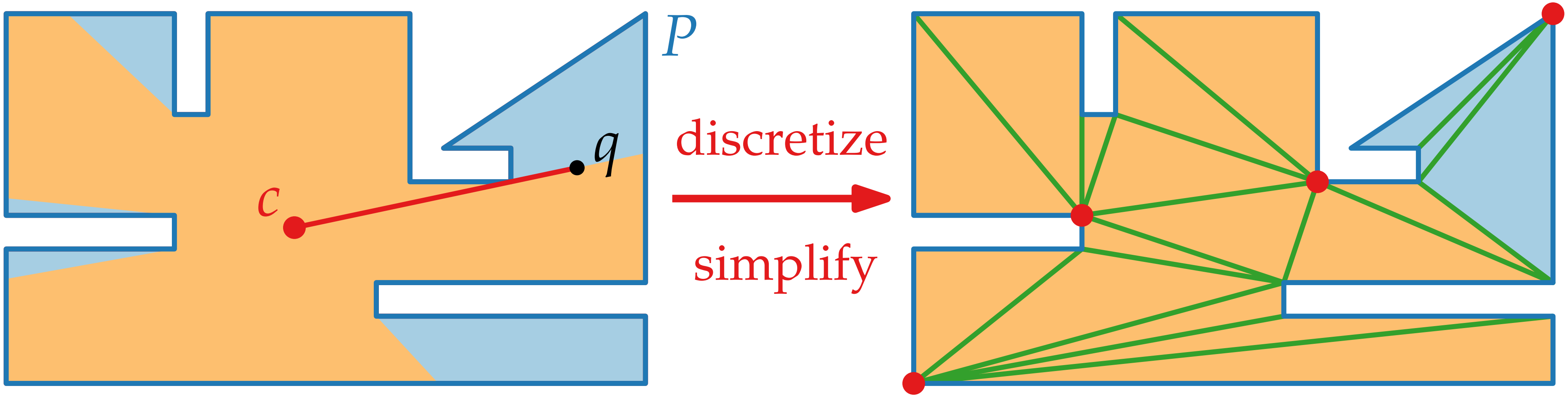
Observation. Camera c "sees" a star-shaped region

Definition. A pt $q \in P$ is *visible* from $c \in P$ if $\overline{qc} \subseteq P$.

Aim: Use few cameras! *But minimizing this is NP-hard...*

Guarding an Art Gallery

Given a *simple* polygon P (i.e., no holes, no self-intersection)...



discretize

simplify

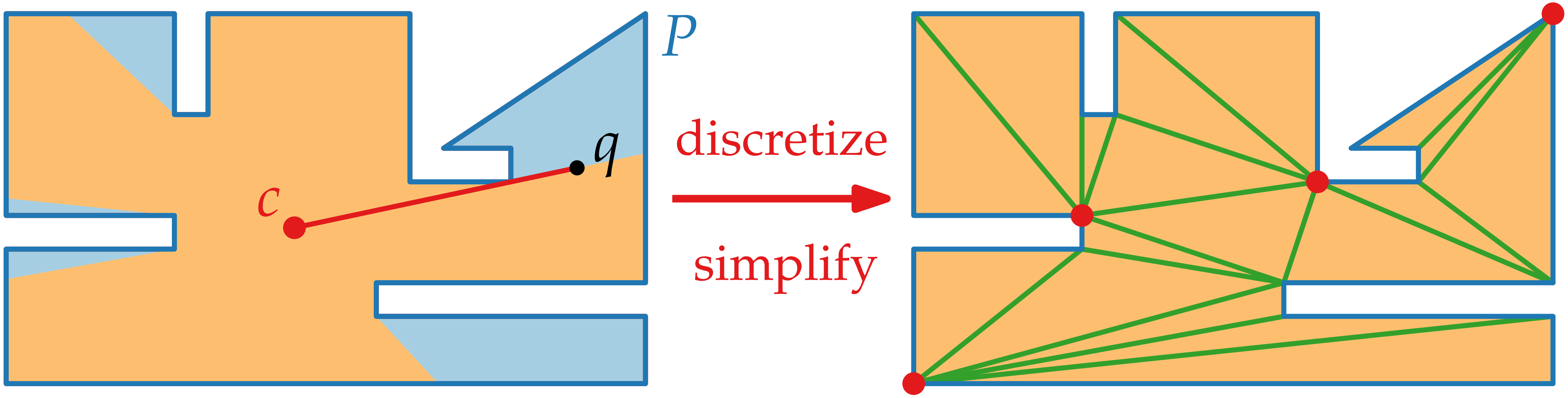
Observation. Camera c “sees” a star-shaped region

Definition. A pt $q \in P$ is *visible* from $c \in P$ if $\overline{qc} \subseteq P$.

Aim: Use few cameras! *But minimizing this is NP-hard…*

Guarding an Art Gallery

Given a *simple* polygon P (i.e., no holes, no self-intersection)...



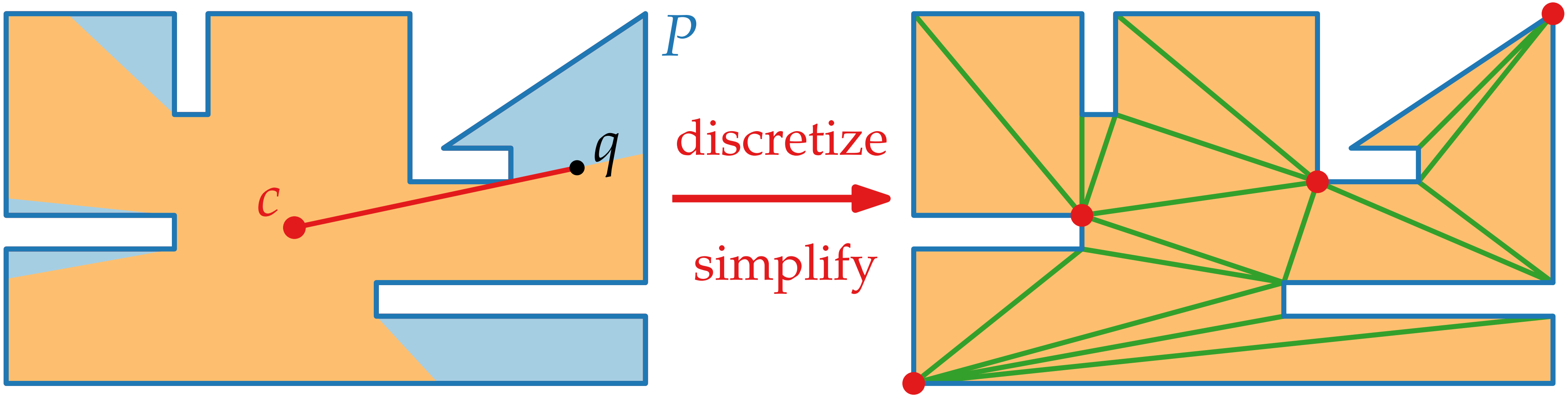
Observation. Camera c “sees” a star-shaped region

Definition. A pt $q \in P$ is *visible* from $c \in P$ if $\overline{qc} \subseteq P$.

Aim: Use few cameras! *But minimizing this is NP-hard. . .*

Guarding an Art Gallery

Given a *simple* polygon P (i.e., no holes, no self-intersection)...



Observation. Camera c “sees” a star-shaped region

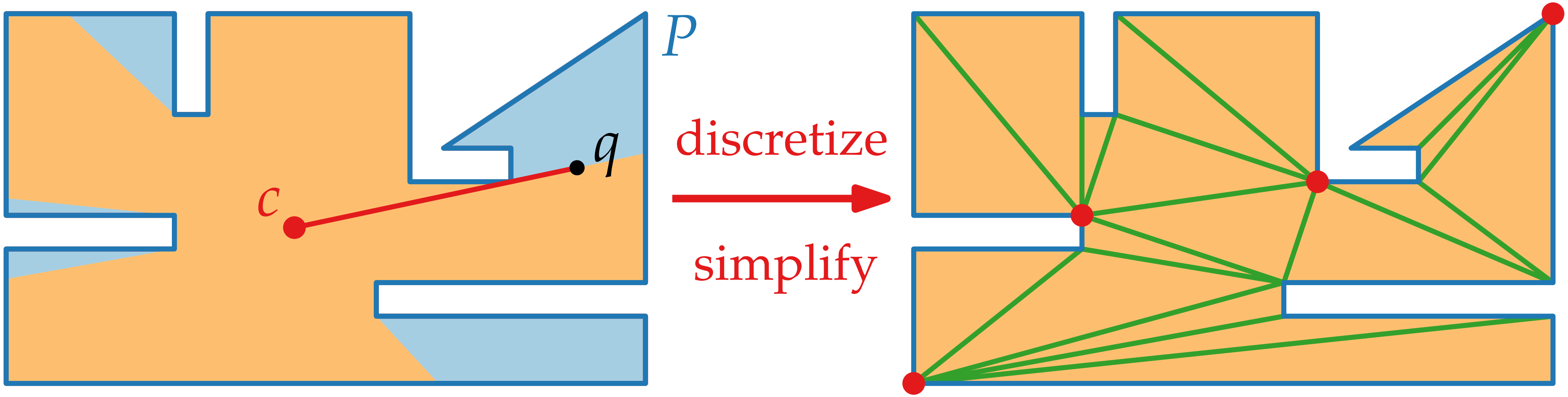
Definition. A pt $q \in P$ is *visible* from $c \in P$ if $\overline{qc} \subseteq P$.

Aim: Use few cameras! *But minimizing this is NP-hard...*

Theorem. 1. Every simple polygon can be triangulated.

Guarding an Art Gallery

Given a *simple* polygon P (i.e., no holes, no self-intersection)...



Observation. Camera c "sees" a star-shaped region

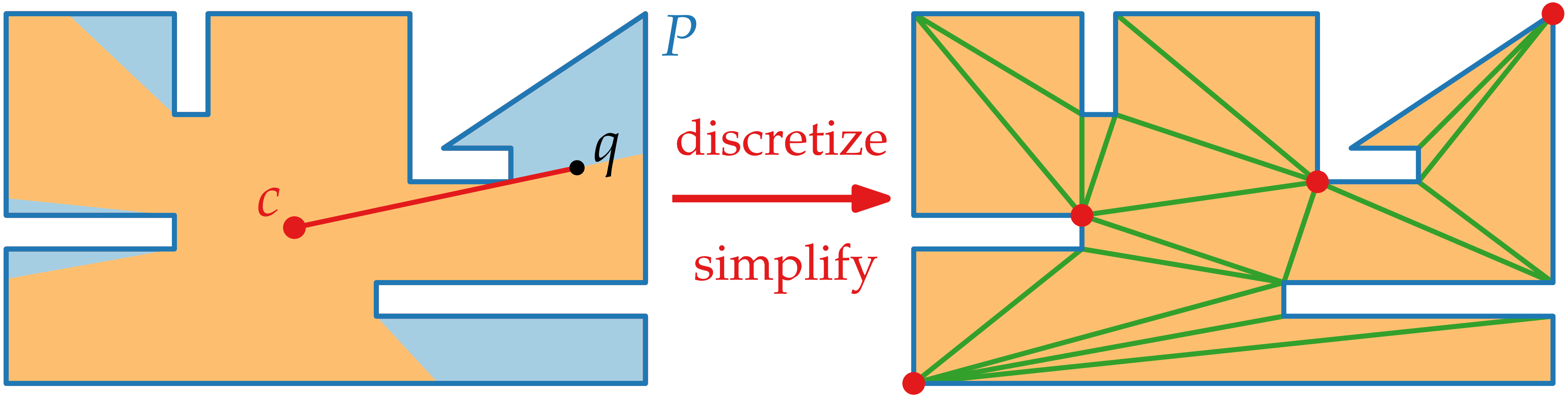
Definition. A pt $q \in P$ is *visible* from $c \in P$ if $\overline{qc} \subseteq P$.

Aim: Use few cameras! *But minimizing this is NP-hard. . .*

Theorem.
1. Every simple polygon can be triangulated.
2. Any triangulation of a simple polygon with n vertices consists of $n - 2$ triangles.

Guarding an Art Gallery

Given a *simple* polygon P (i.e., no holes, no self-intersection)...



Observation. Camera c "sees" a star-shaped region

Definition. A pt $q \in P$ is *visible* from $c \in P$ if $\overline{qc} \subseteq P$.

Aim: Use few cameras! *But minimizing this is NP-hard. . .*

Theorem.
1. Every simple polygon can be triangulated.
2. Any triangulation of a simple polygon with n vertices consists of $n - 2$ triangles.

How can we prove these?

Existence of Triangulation

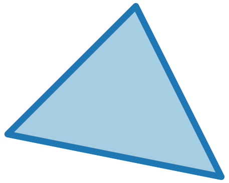
Theorem.

1. Every simple polygon can be triangulated.
2. Any triangulation of a simple polygon with n vertices consists of $n - 2$ triangles.

Existence of Triangulation

Theorem.

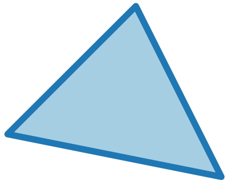
1. Every simple polygon can be triangulated.
2. Any triangulation of a simple polygon with n vertices consists of $n - 2$ triangles.

$n = 3$:  1 triangle ✓

Existence of Triangulation

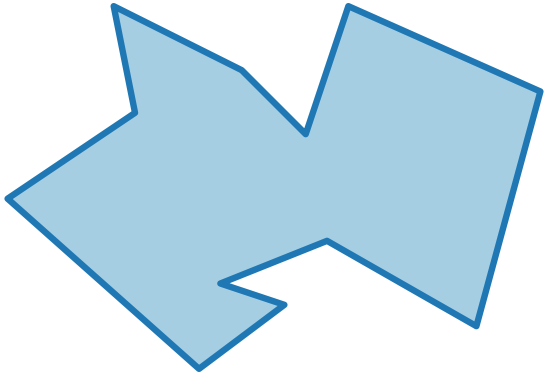
Theorem.
1. Every simple polygon can be triangulated.
2. Any triangulation of a simple polygon with n vertices consists of $n-2$ triangles.

$n = 3$:



1 triangle ✓

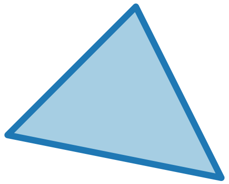
$3, \ldots, n-1 \rightarrow n$:



Existence of Triangulation

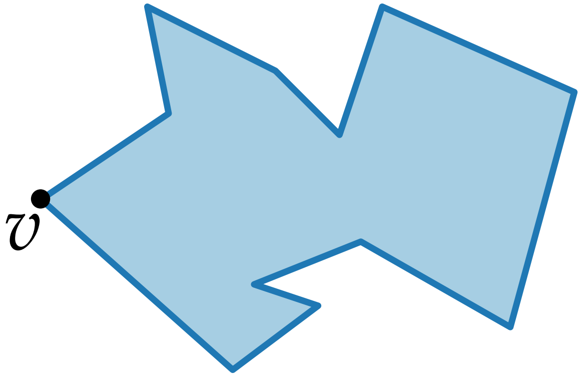
Theorem.
1. Every simple polygon can be triangulated.
2. Any triangulation of a simple polygon with n vertices consists of $n - 2$ triangles.

$n = 3$:



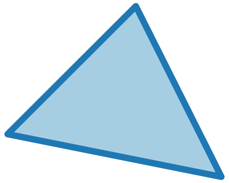
1 triangle ✓

$3, \ldots, n - 1 \rightarrow n$:

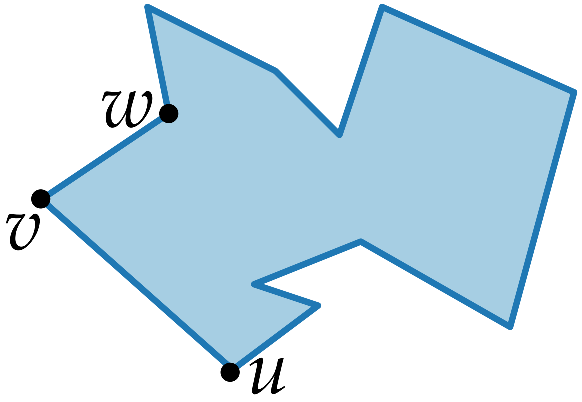


Existence of Triangulation

Theorem.
1. Every simple polygon can be triangulated.
2. Any triangulation of a simple polygon with n vertices consists of $n-2$ triangles.

$n = 3$:  1 triangle ✓

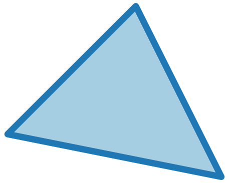
$3, \ldots, n-1 \rightarrow n$:



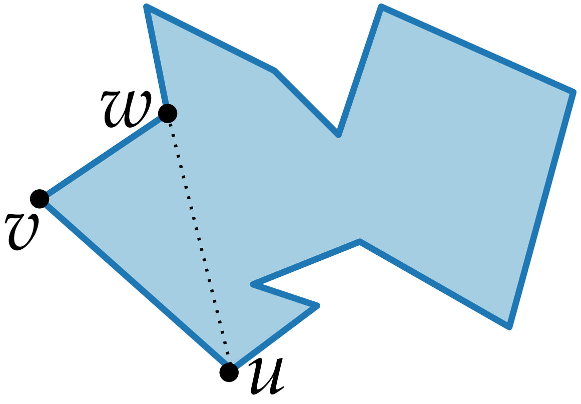
Existence of Triangulation

Theorem.

1. Every simple polygon can be triangulated.
2. Any triangulation of a simple polygon with n vertices consists of $n-2$ triangles.

$n = 3$:  1 triangle ✓

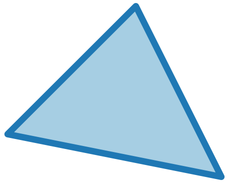
$3, \ldots, n-1 \rightarrow n$:



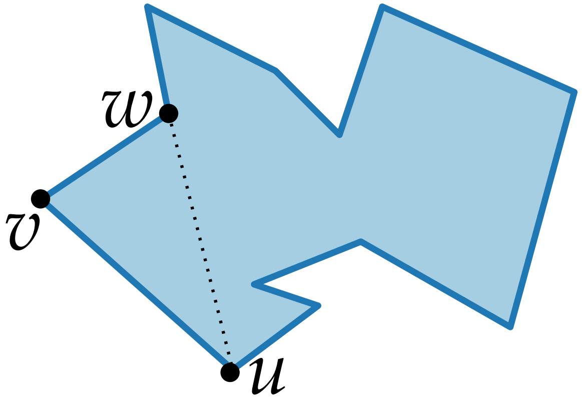
Existence of Triangulation

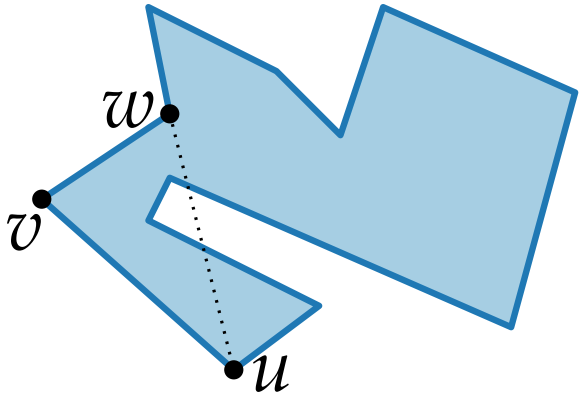
Theorem. 1. Every simple polygon can be triangulated.

2. Any triangulation of a simple polygon with n vertices consists of $n - 2$ triangles.

$n = 3$:  1 triangle ✓

$3, \ldots, n - 1 \rightarrow n$:

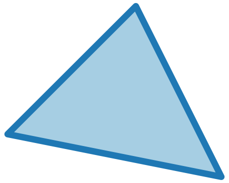




Existence of Triangulation

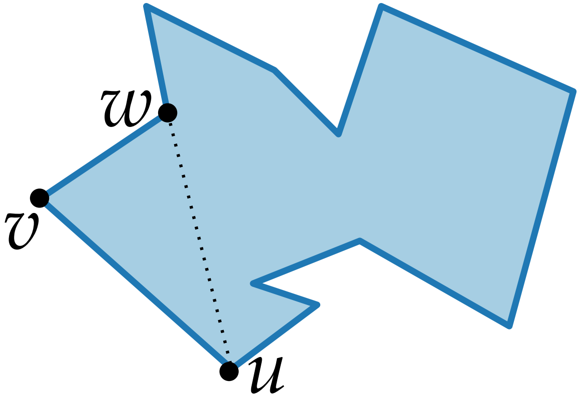
Theorem.
1. Every simple polygon can be triangulated.
2. Any triangulation of a simple polygon with n vertices consists of $n - 2$ triangles.

$n = 3$:

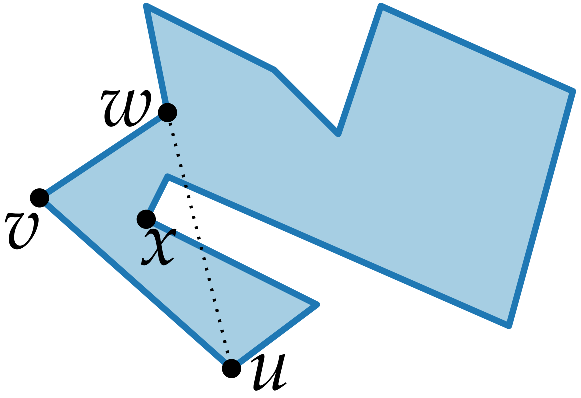


1 triangle ✓

$3, \ldots, n - 1 \rightarrow n$:



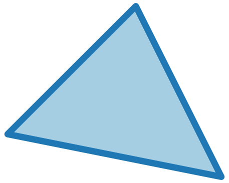
x furthest from uw



Existence of Triangulation

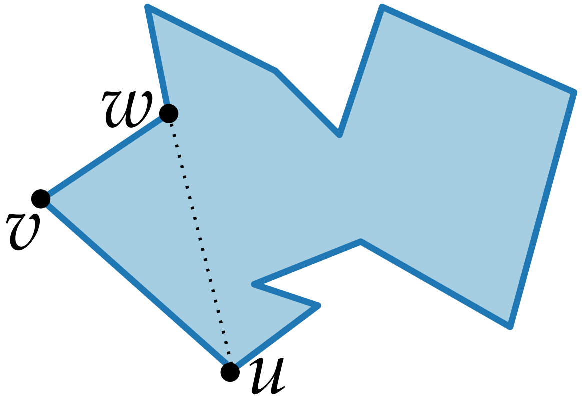
Theorem.
1. Every simple polygon can be triangulated.
2. Any triangulation of a simple polygon with n vertices consists of $n-2$ triangles.

$n = 3$:

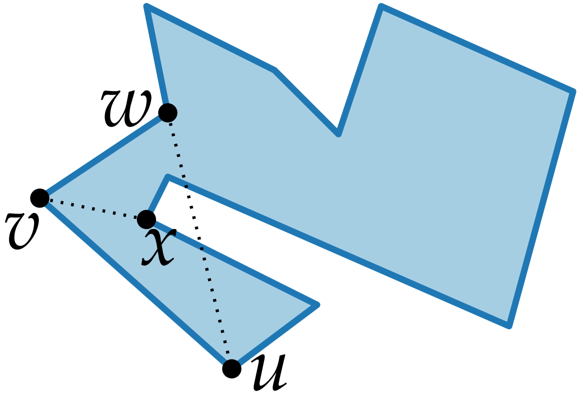


1 triangle ✓

$3, \ldots, n-1 \to n$:



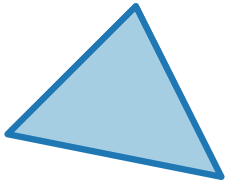
x furthest from uw



Existence of Triangulation

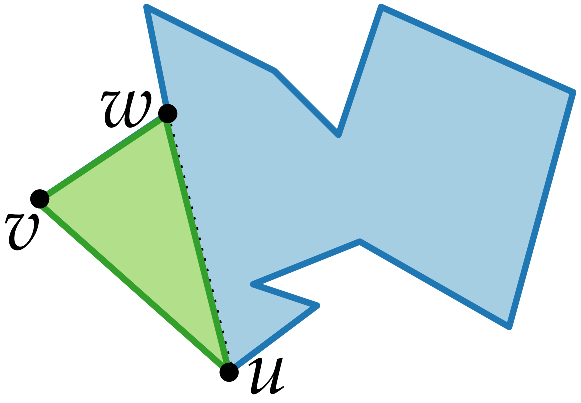
Theorem.
1. Every simple polygon can be triangulated.
2. Any triangulation of a simple polygon with n vertices consists of $n - 2$ triangles.

$n = 3$:

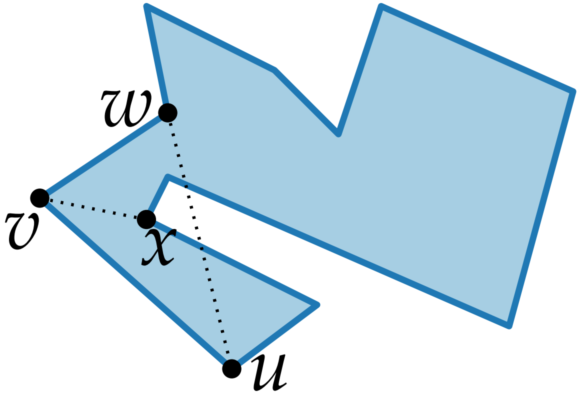


1 triangle ✓

$3, \ldots, n-1 \rightarrow n$:



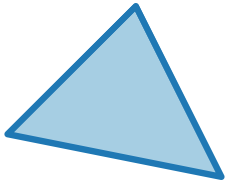
x furthest from uw



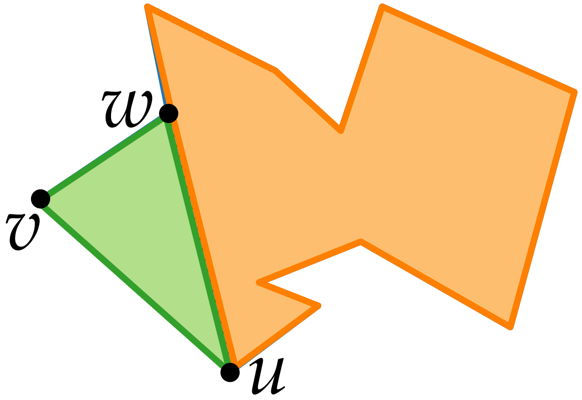
Existence of Triangulation

Theorem. 1. Every simple polygon can be triangulated.

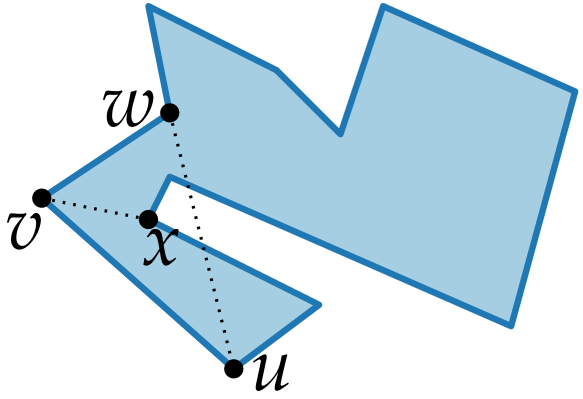
2. Any triangulation of a simple polygon with n vertices consists of $n-2$ triangles.

$n = 3$:  1 triangle ✓

$3, \ldots, n-1 \to n$:

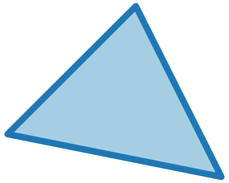


x furthest from uw

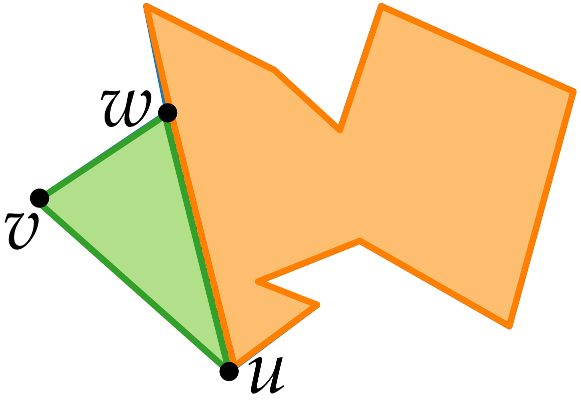


Existence of Triangulation

Theorem. 1. Every simple polygon can be triangulated.
2. Any triangulation of a simple polygon with n vertices consists of $n-2$ triangles.

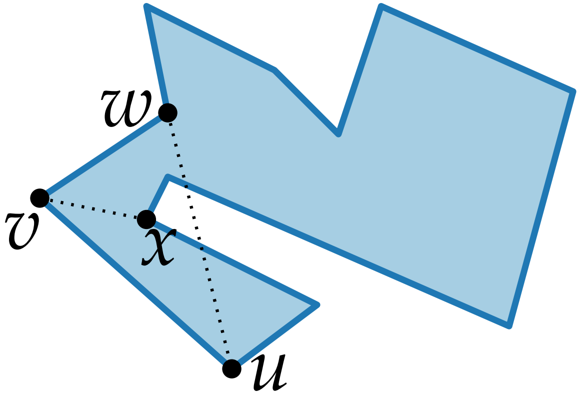
$n = 3$:  1 triangle ✓

$3, \ldots, n-1 \to n$:



3 vtcs ⇒ 1 triangle

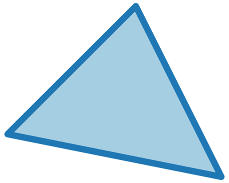
x furthest from uw



Existence of Triangulation

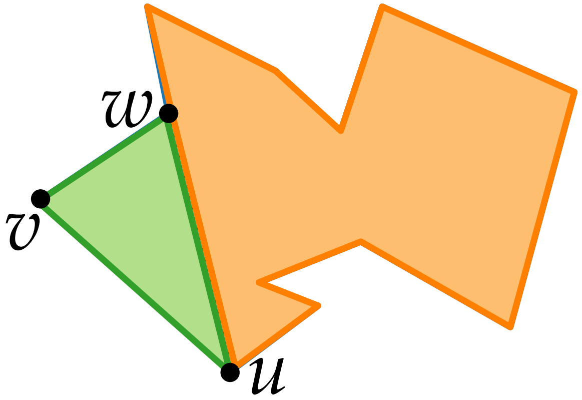
Theorem. 1. Every simple polygon can be triangulated.
2. Any triangulation of a simple polygon with n vertices consists of $n - 2$ triangles.

$n = 3$:



1 triangle ✓

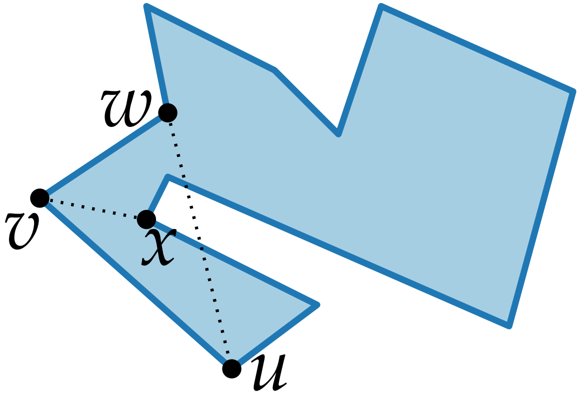
$3, \ldots, n-1 \to n$:



3 vtcs ⇒ 1 triangle

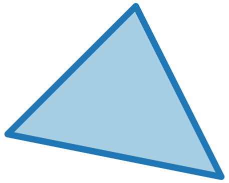
$n-1$ vtcs ⇒ $n-3$ triangles

x furthest from uw

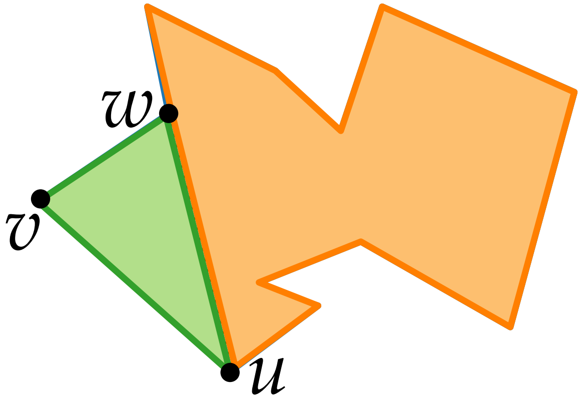


Existence of Triangulation

Theorem.
1. Every simple polygon can be triangulated.
2. Any triangulation of a simple polygon with n vertices consists of $n-2$ triangles.

$n = 3$:  1 triangle ✓

$3, \ldots, n-1 \to n$:



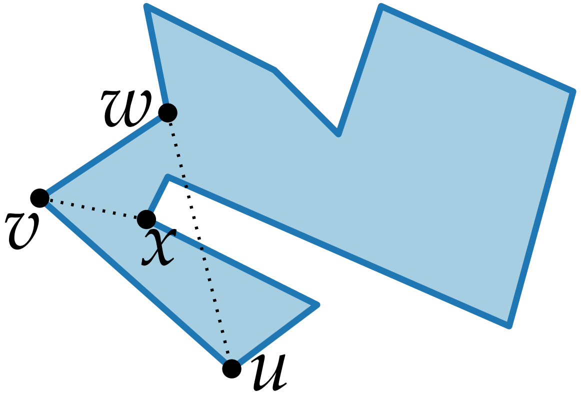
3 vtcs ⇒ 1 triangle

$n-1$ vtcs ⇒ $n-3$ triangles

⇒ $n-2$ triangles

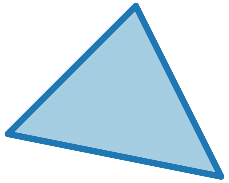
✓

x furthest from uw

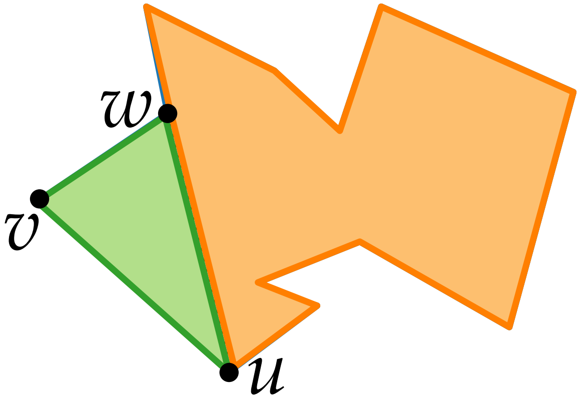


Existence of Triangulation

Theorem.
1. Every simple polygon can be triangulated.
2. Any triangulation of a simple polygon with n vertices consists of $n-2$ triangles.

$n = 3$:  1 triangle ✓

$3, \ldots, n-1 \to n$:



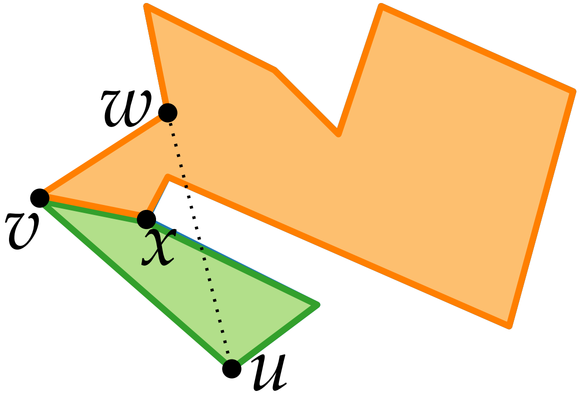
3 vtcs ⇒ 1 triangle

$n-1$ vtcs ⇒ $n-3$ triangles

⇒ $n-2$ triangles

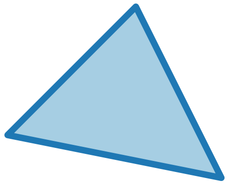
✓

x furthest from uw

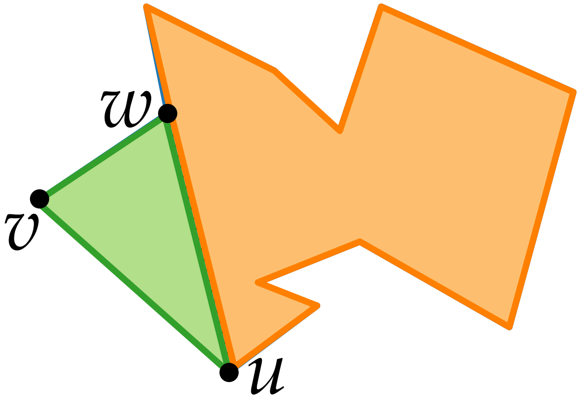


Existence of Triangulation

Theorem.
1. Every simple polygon can be triangulated.
2. Any triangulation of a simple polygon with n vertices consists of $n - 2$ triangles.

$n = 3$:  1 triangle ✓

$3, \ldots, n - 1 \to n$:



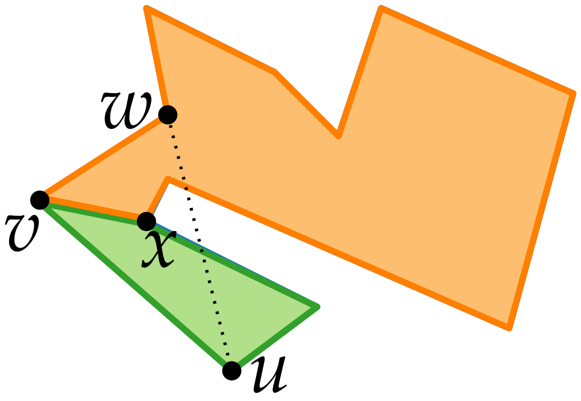
3 vtcs $\Rightarrow$ 1 triangle

$n{-}1$ vtcs $\Rightarrow$ $n{-}3$ triangles

$\Rightarrow$ $n{-}2$ triangles

✓

x furthest from uw

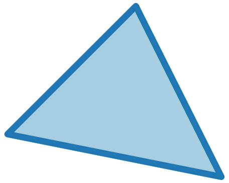


m vtcs $\Rightarrow$ $m{-}2$ triangles

Existence of Triangulation

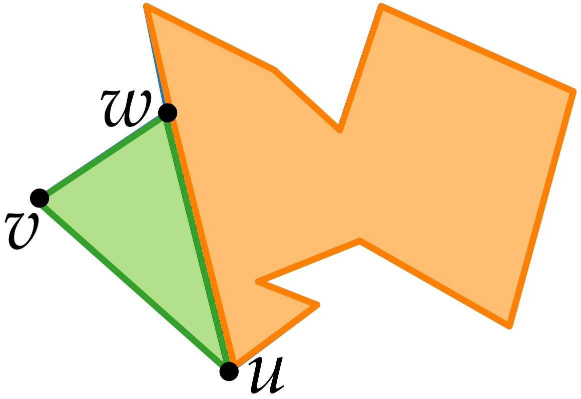
Theorem.
1. Every simple polygon can be triangulated.
2. Any triangulation of a simple polygon with n vertices consists of $n-2$ triangles.

$n = 3$:



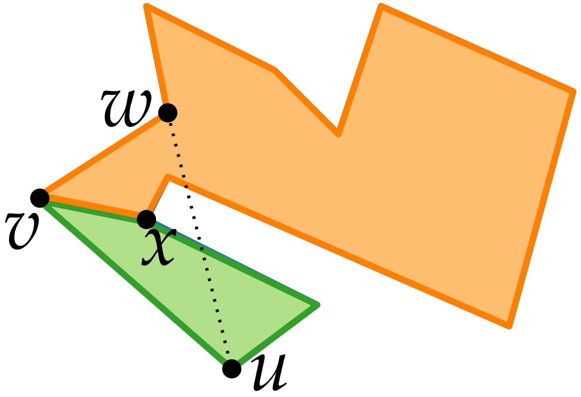
1 triangle ✓

$3, \ldots, n-1 \to n$:



3 vtcs ⇒ 1 triangle
$n-1$ vtcs ⇒ $n-3$ triangles
⇒ $n-2$ triangles
✓

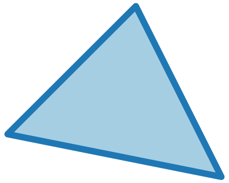
x furthest from uw



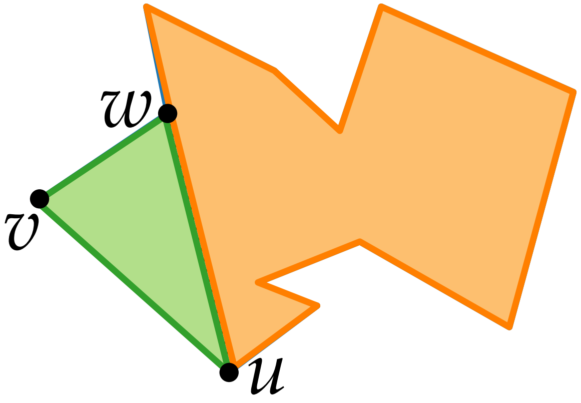
m vtcs ⇒ $m-2$ triangles
$n-m+2$ vtcs ⇒ $n-m$ triangles

Existence of Triangulation

Theorem.
1. Every simple polygon can be triangulated.
2. Any triangulation of a simple polygon with n vertices consists of $n-2$ triangles.

$n = 3$:  1 triangle ✓

$3, \ldots, n-1 \to n$:



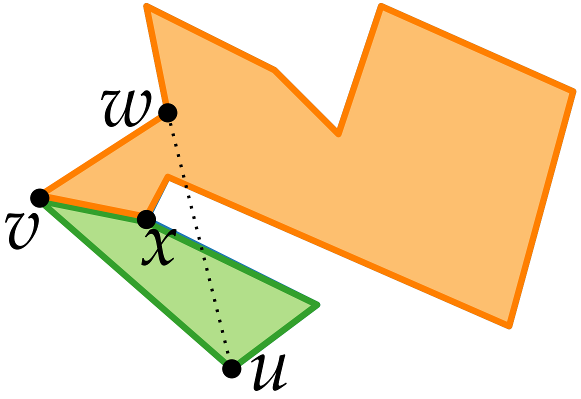
3 vtcs ⇒ 1 triangle

$n-1$ vtcs ⇒ $n-3$ triangles

⇒ $n-2$ triangles

✓

x furthest from uw



m vtcs ⇒ $m-2$ triangles

$n-m+2$ vtcs ⇒ $n-m$ triangles

⇒ $n-2$ triangles

✓

The Art Gallery Theorem [Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

The Art Gallery Theorem

[Chvátal '75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

Exercise. Find, for arbitrarily large n, a polygon with n vertices, where $\approx n/3$ cameras are necessary.

[2 minutes]

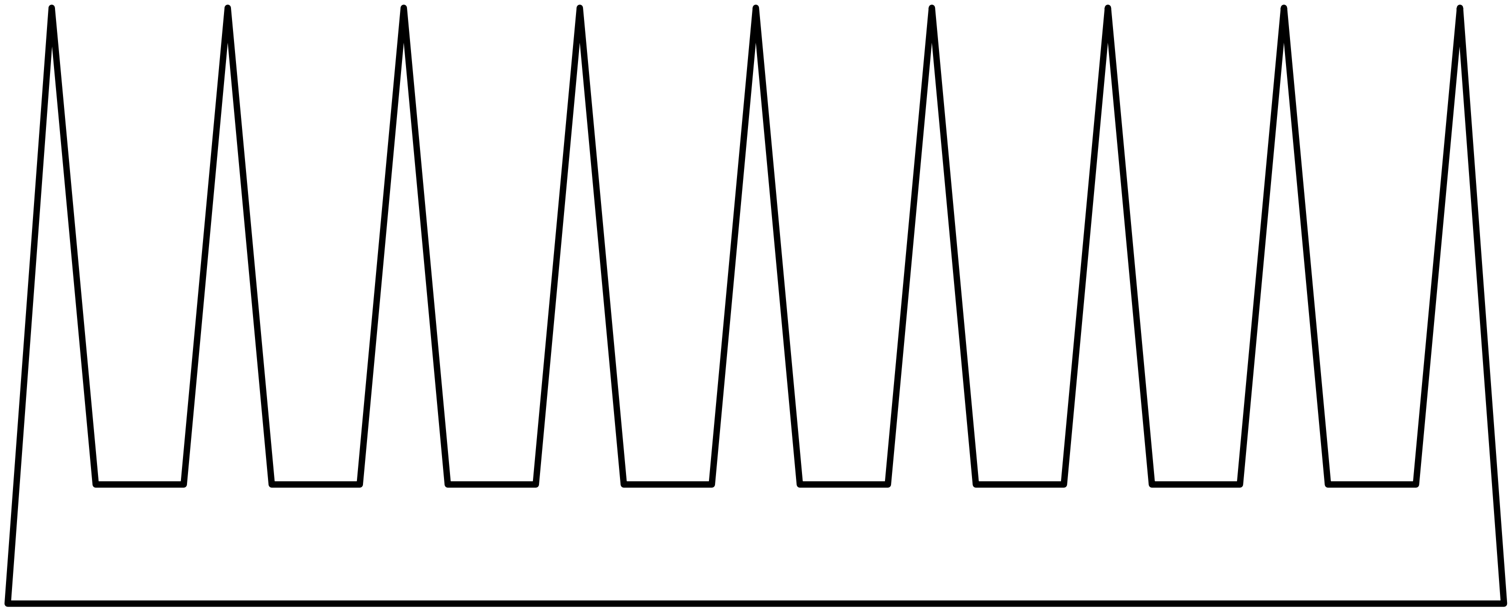
The Art Gallery Theorem

[Chvátal '75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

Exercise. Find, for arbitrarily large n, a polygon with n vertices, where $\approx n/3$ cameras are necessary.

[2 minutes]



[dBCvKO'08]

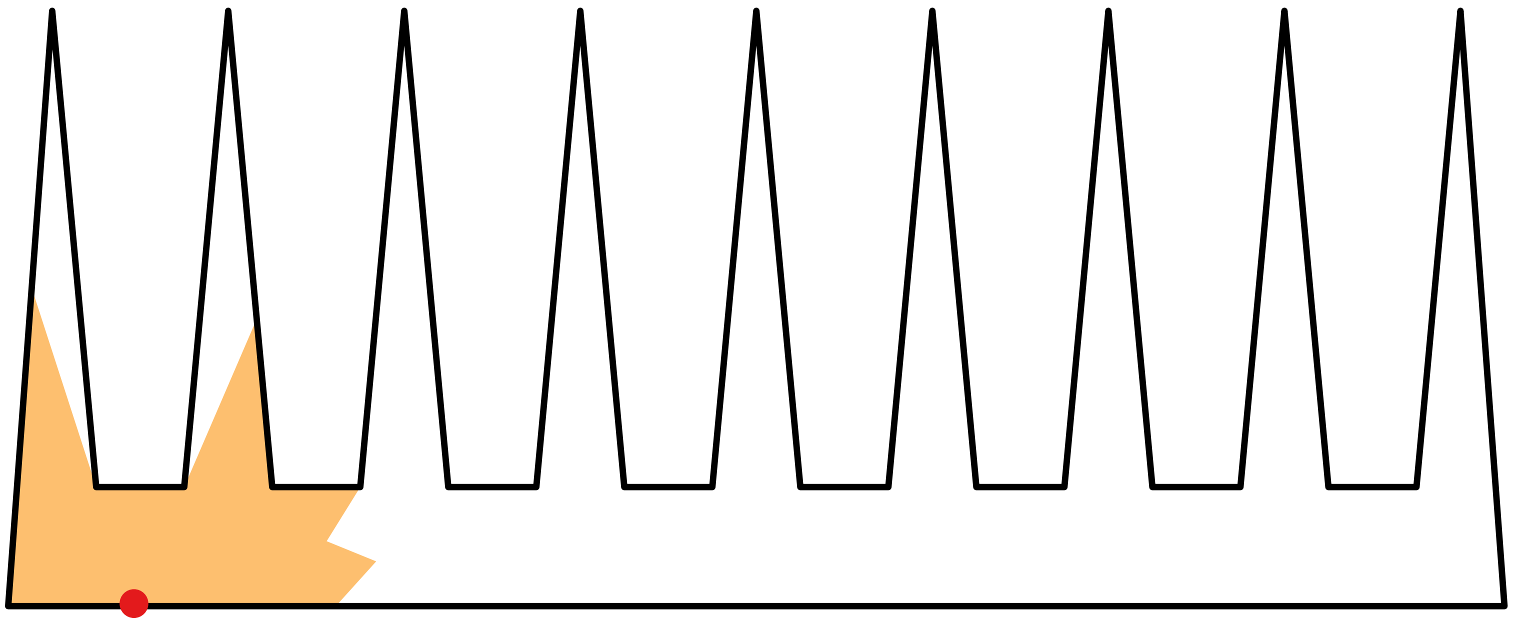
The Art Gallery Theorem

[Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

Exercise. Find, for arbitrarily large n, a polygon with n vertices, where $\approx n/3$ cameras are necessary.

[2 minutes]



[dBCvKO’08]

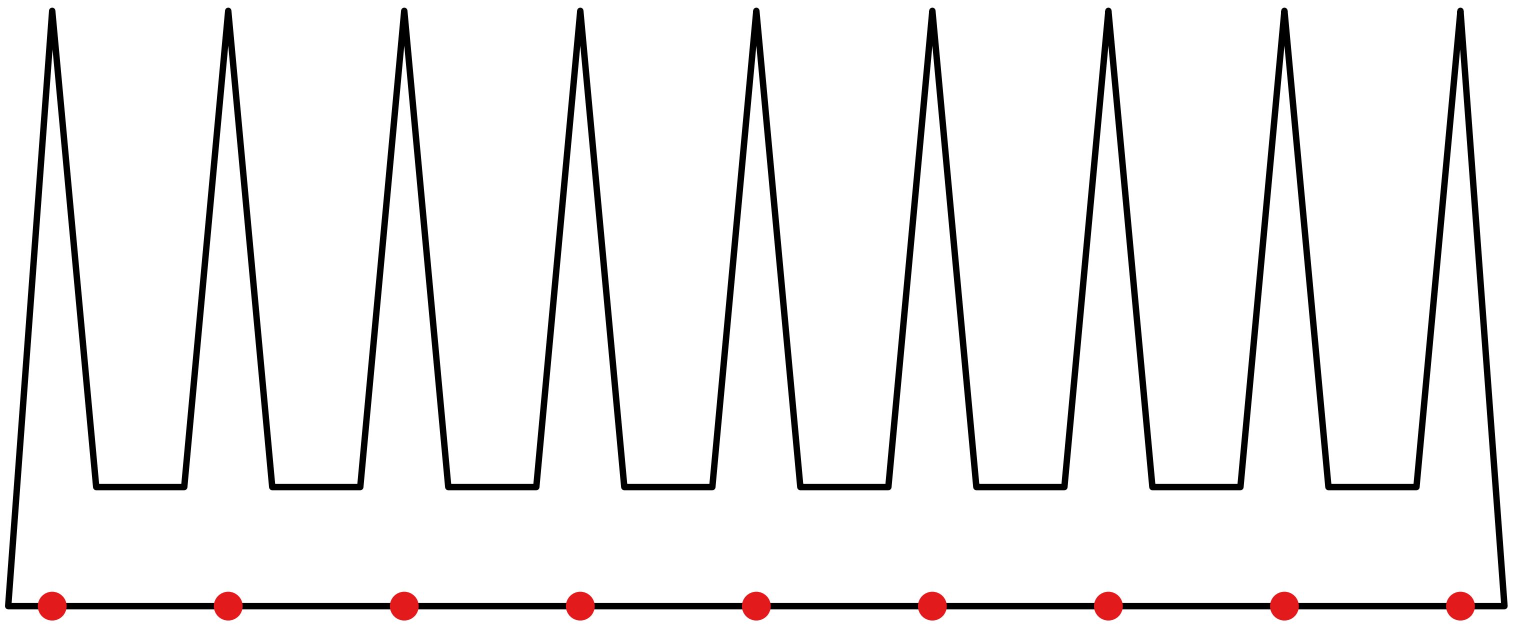
The Art Gallery Theorem

[Chvátal '75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

Exercise. Find, for arbitrarily large n, a polygon with n vertices, where $\approx n/3$ cameras are necessary.

[2 minutes]



[dBCvKO'08]

The Art Gallery Theorem

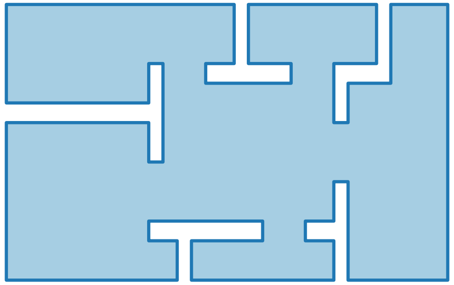
[Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

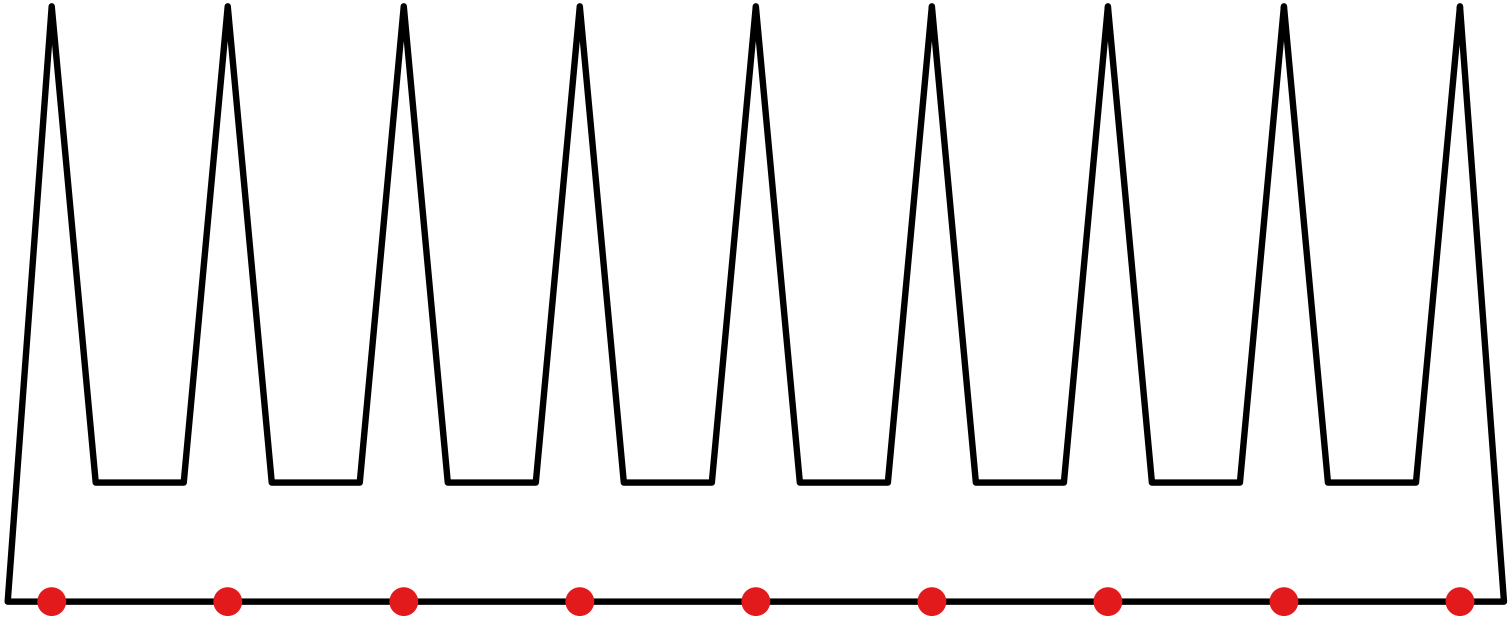
rectilinear

Exercise. Find, for arbitrarily large n, a polygon with n vertices, where $\approx n/3$ cameras are necessary.

$n/4$



[dBCvKO’08]



The Art Gallery Theorem

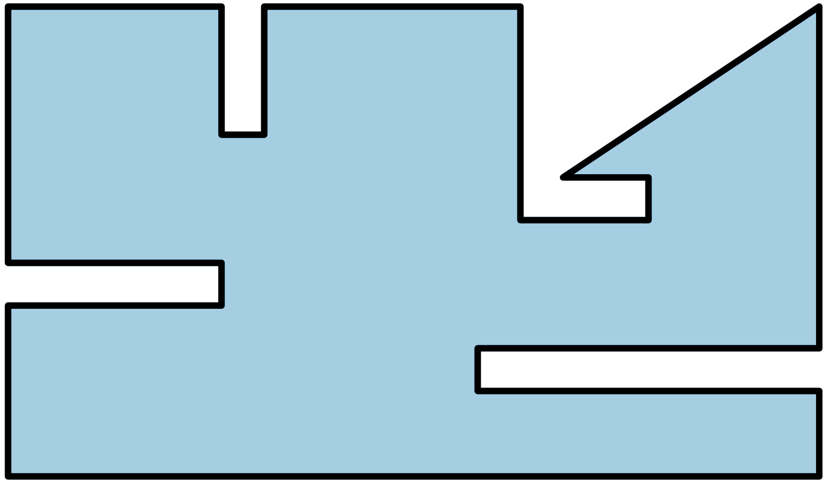
[Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

The Art Gallery Theorem

[Chvátal '75]

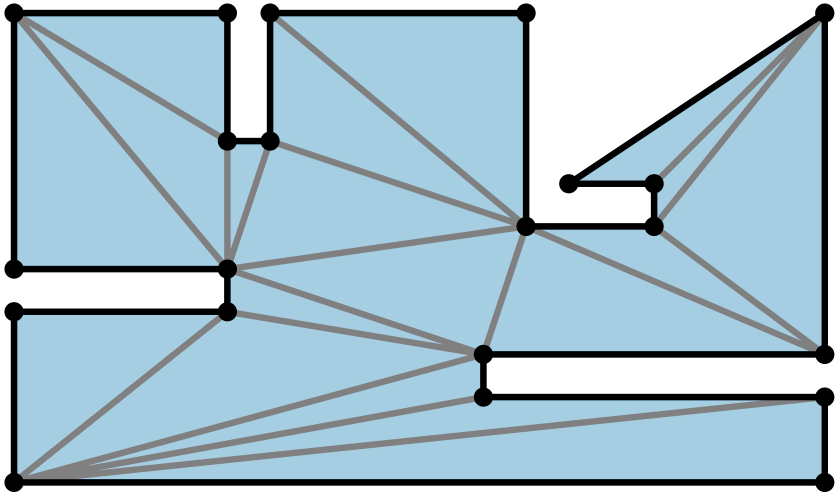
Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.



The Art Gallery Theorem

[Chvátal '75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

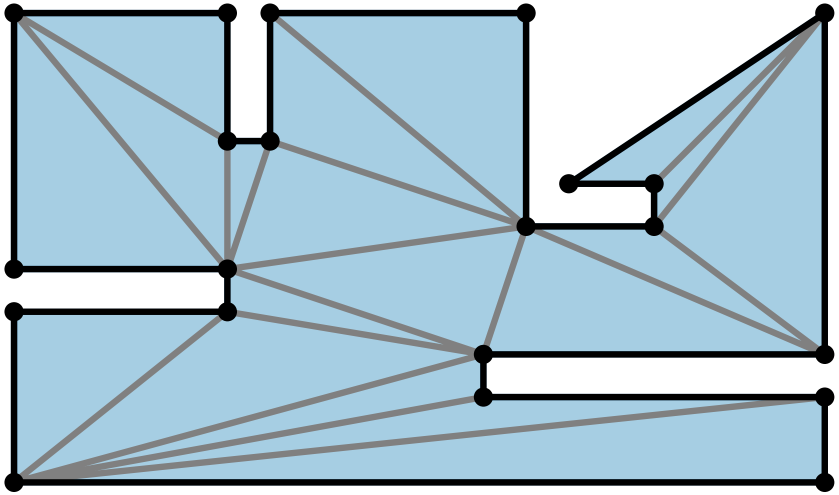


The Art Gallery Theorem

[Chvátal '75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

3-color the vtcs

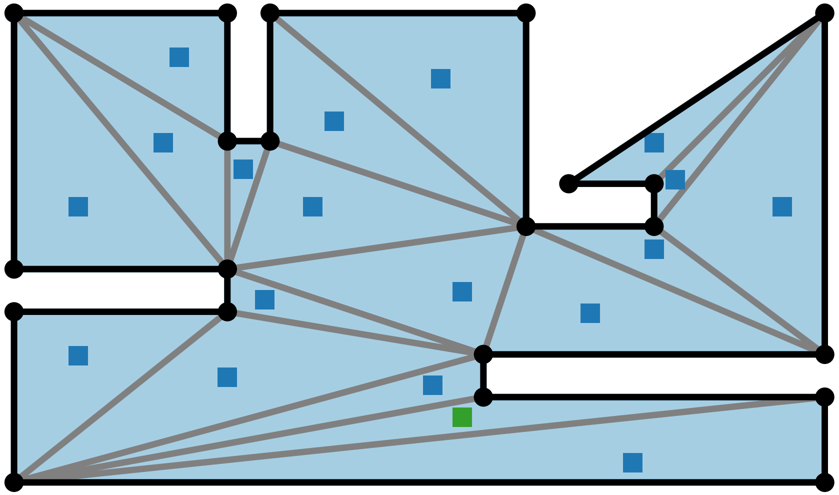


The Art Gallery Theorem

[Chvátal '75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

3-color the vtcs

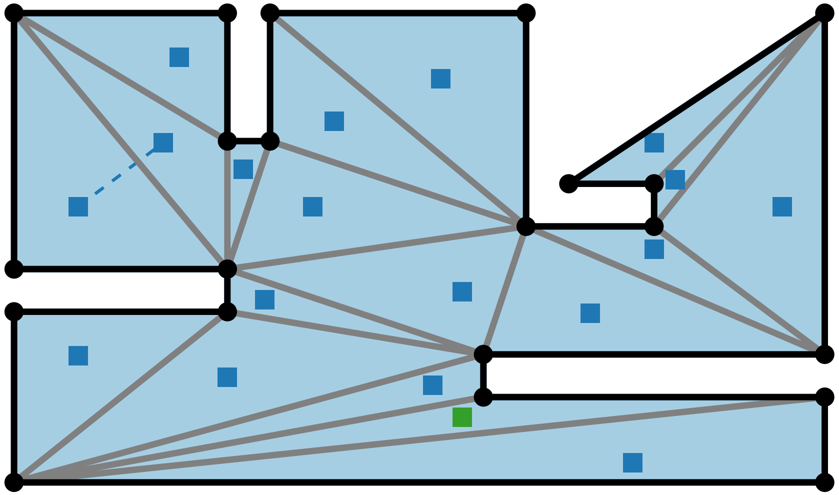


The Art Gallery Theorem

[Chvátal '75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

3-color the vtcs

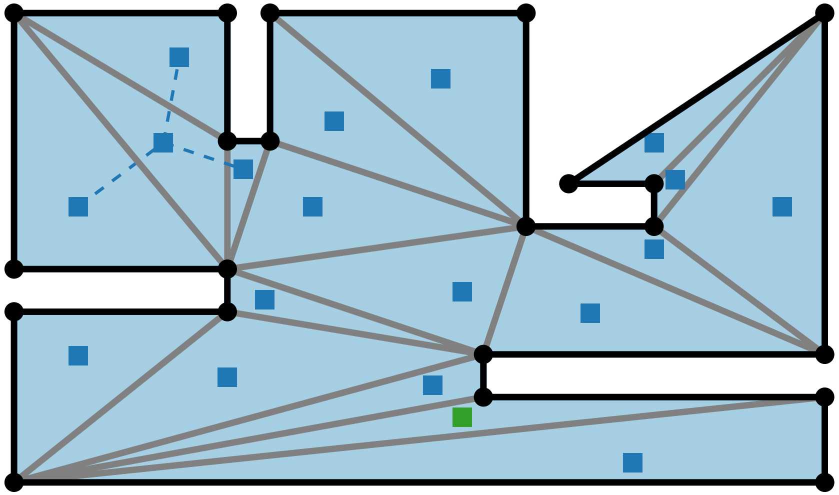


The Art Gallery Theorem

[Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

3-color the vtcs



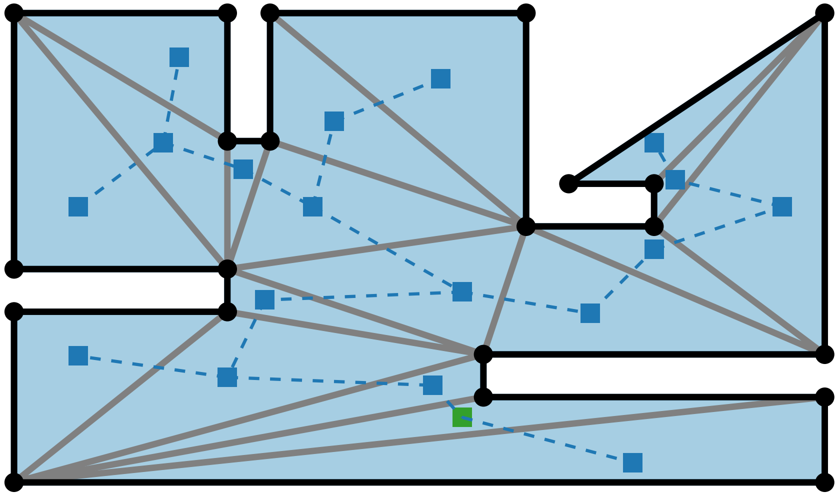
The Art Gallery Theorem

[Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

3-color the vtcs

dual tree



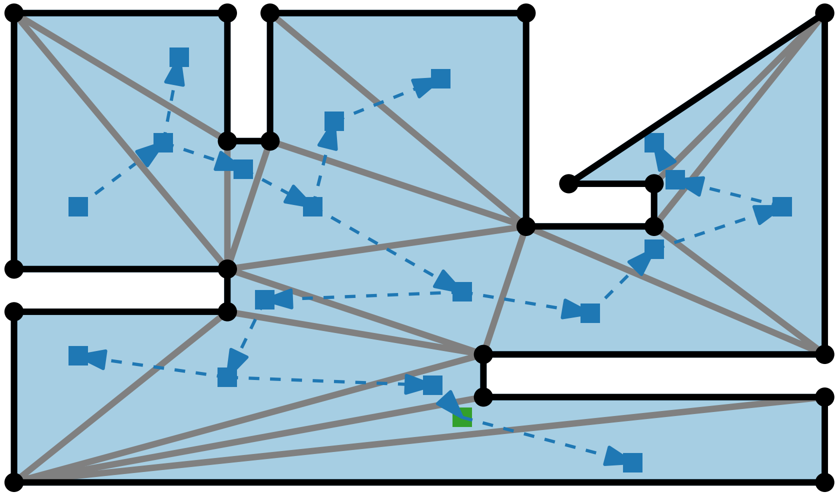
The Art Gallery Theorem

[Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

3-color the vtcs

dual tree



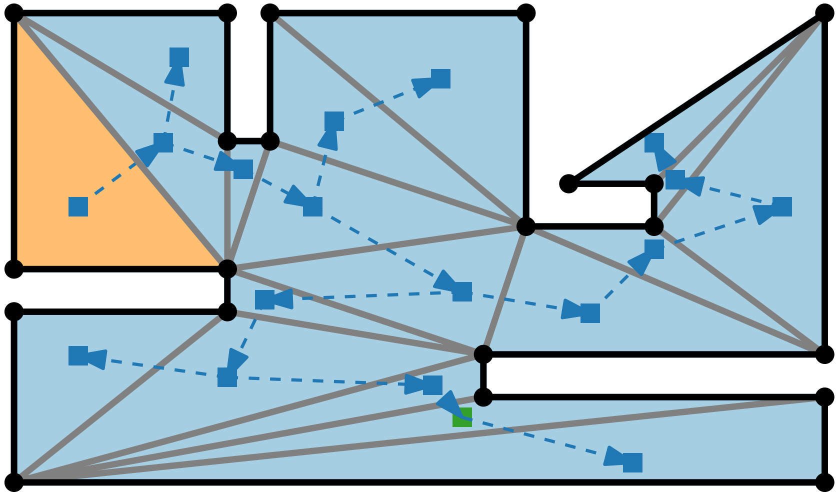
The Art Gallery Theorem

[Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

3-color the vtcs

dual tree



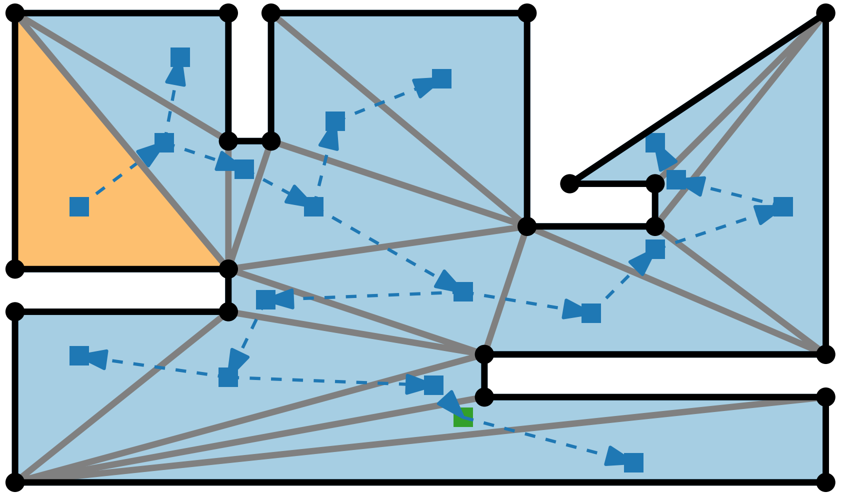
The Art Gallery Theorem

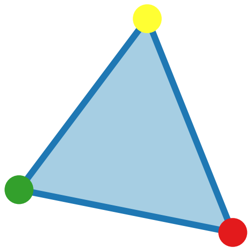
[Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

3-color the vtcs

dual tree





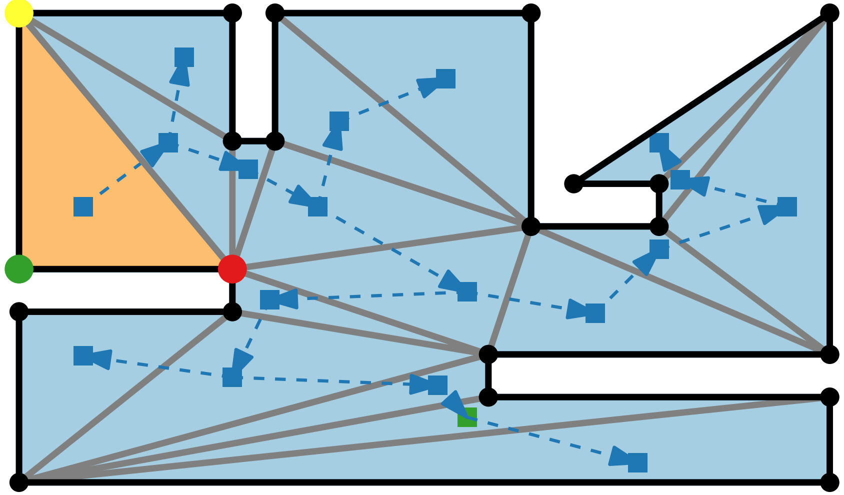
The Art Gallery Theorem

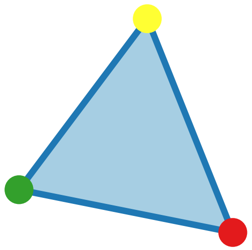
[Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

3-color the vtcs

dual tree





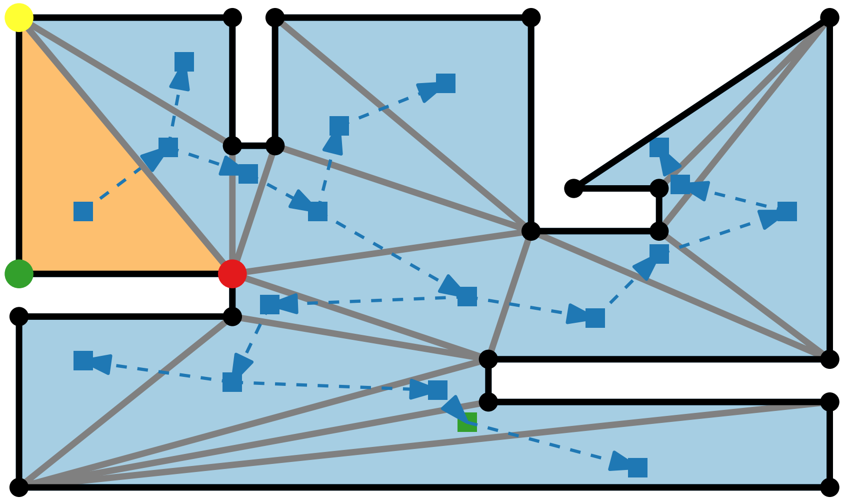
The Art Gallery Theorem

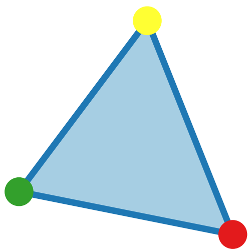
[Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

3-color the vtcs

Traverse the dual tree



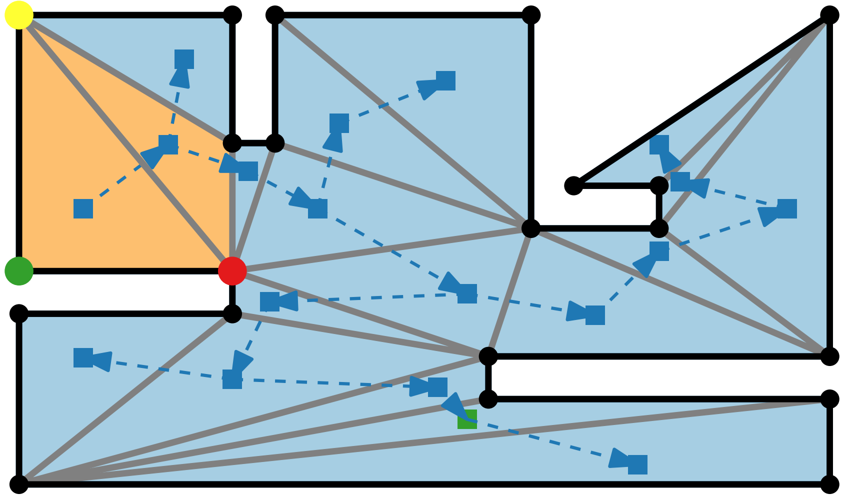


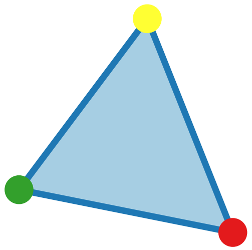
The Art Gallery Theorem [Chvátal '75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

3-color the vtcs

Traverse the dual tree





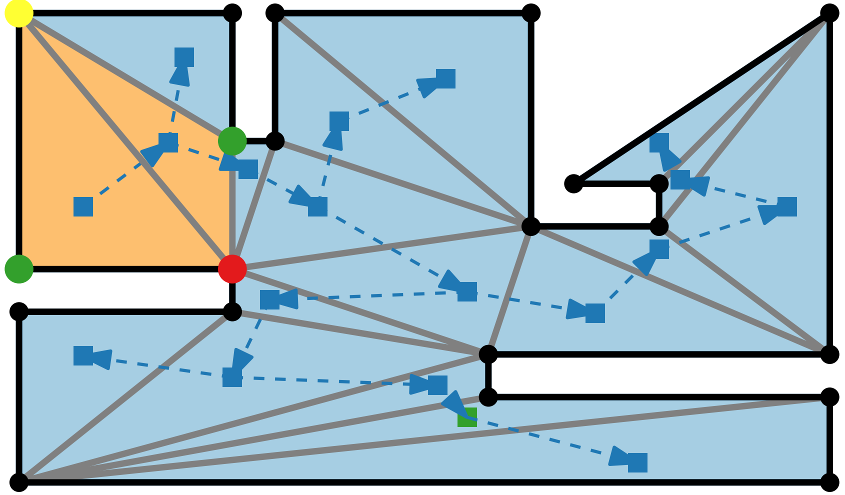
The Art Gallery Theorem

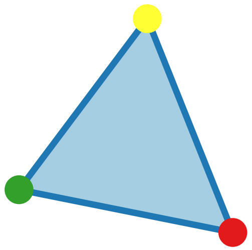
[Chvátal '75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

3-color the vtcs

Traverse the dual tree





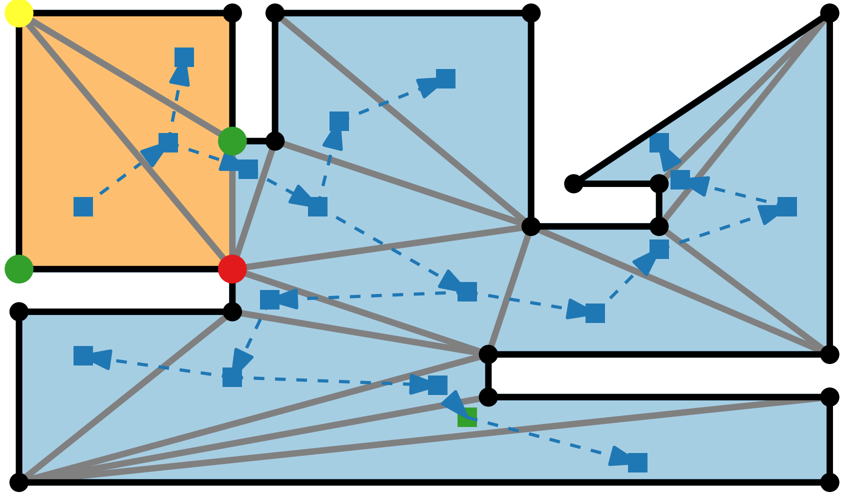
The Art Gallery Theorem

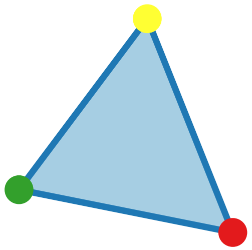
[Chvátal '75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

3-color the vtcs

Traverse the dual tree





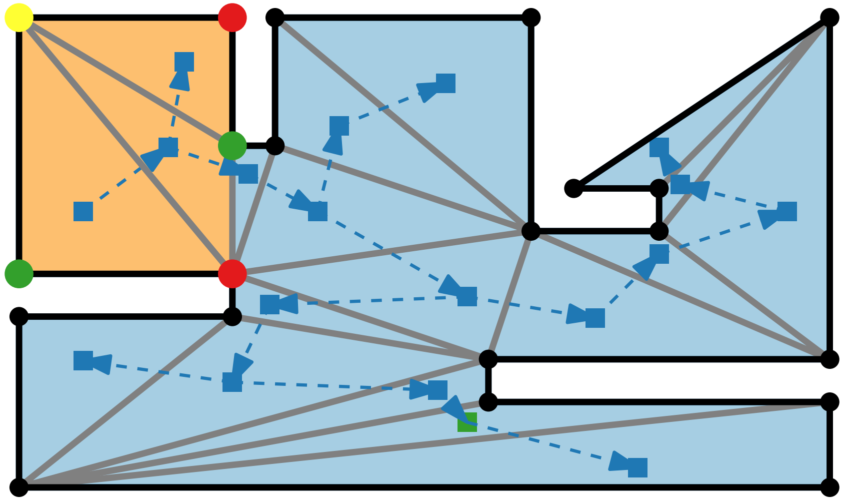
The Art Gallery Theorem

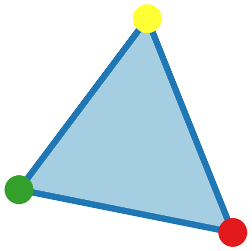
[Chvátal '75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

3-color the vtcs

Traverse the dual tree





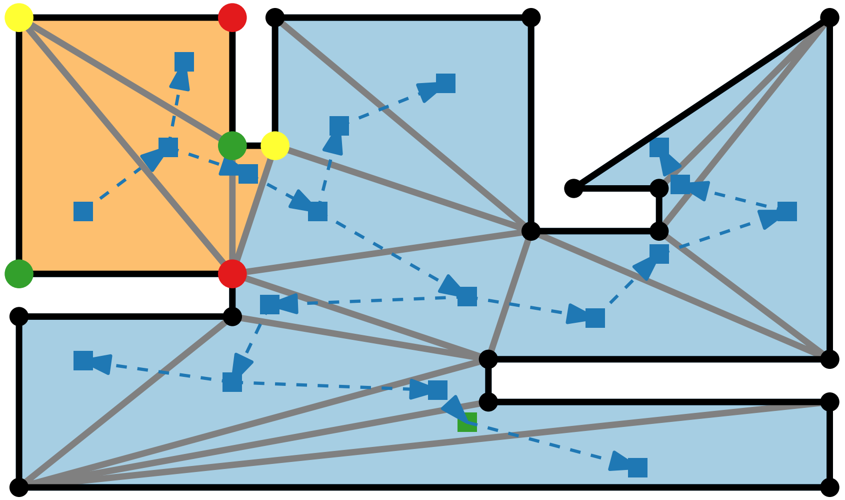
The Art Gallery Theorem

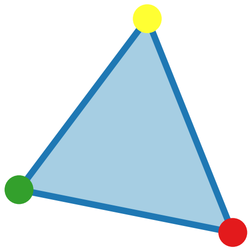
[Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

3-color the vtcs

Traverse the dual tree





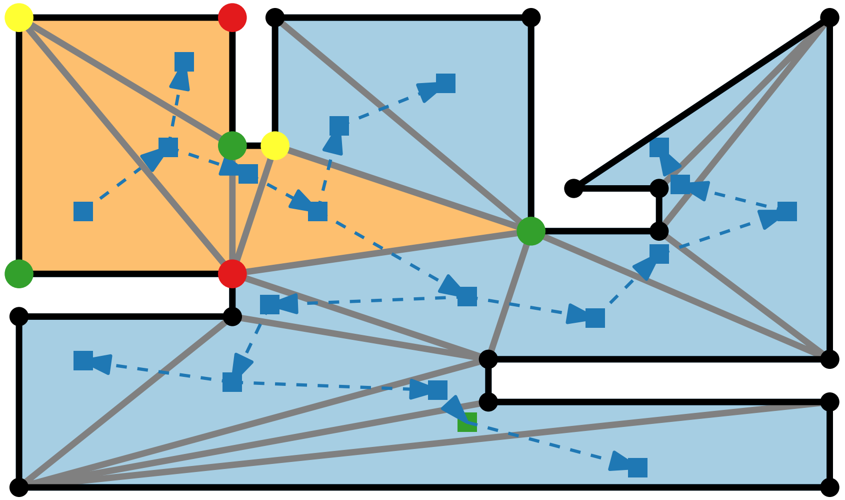
The Art Gallery Theorem

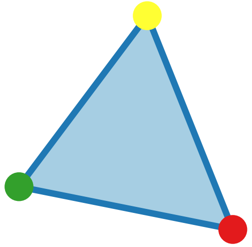
[Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

3-color the vtcs

Traverse the dual tree





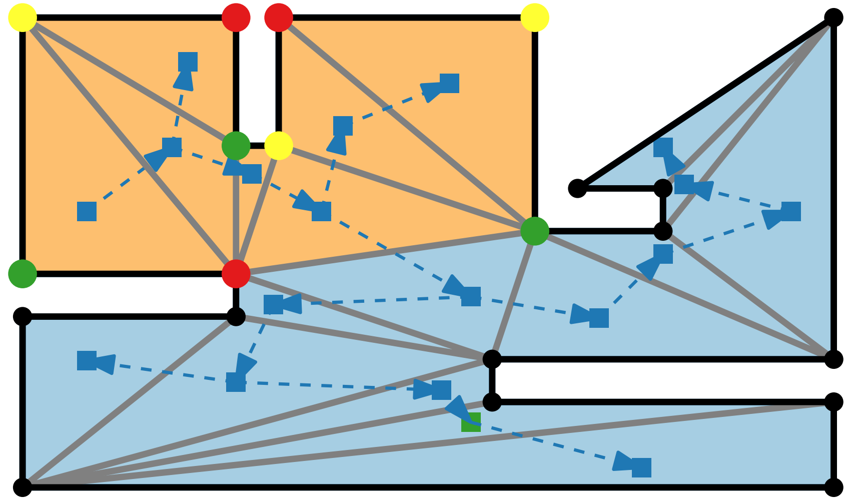
The Art Gallery Theorem

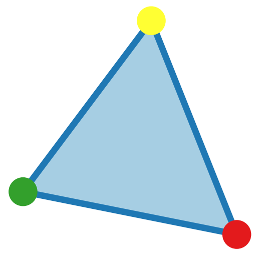
[Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

3-color the vtcs

Traverse the dual tree





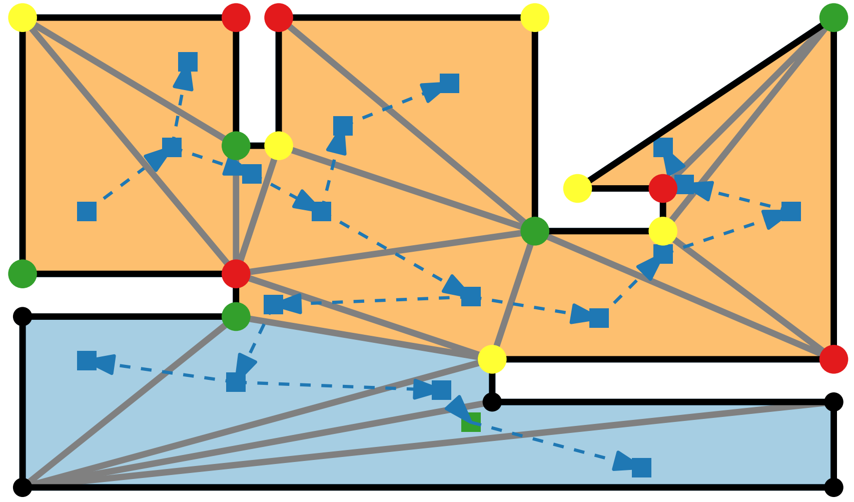
The Art Gallery Theorem

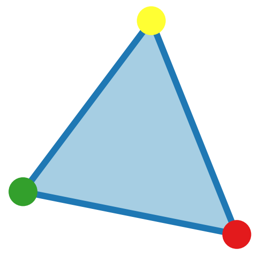
[Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

3-color the vtcs

Traverse the dual tree





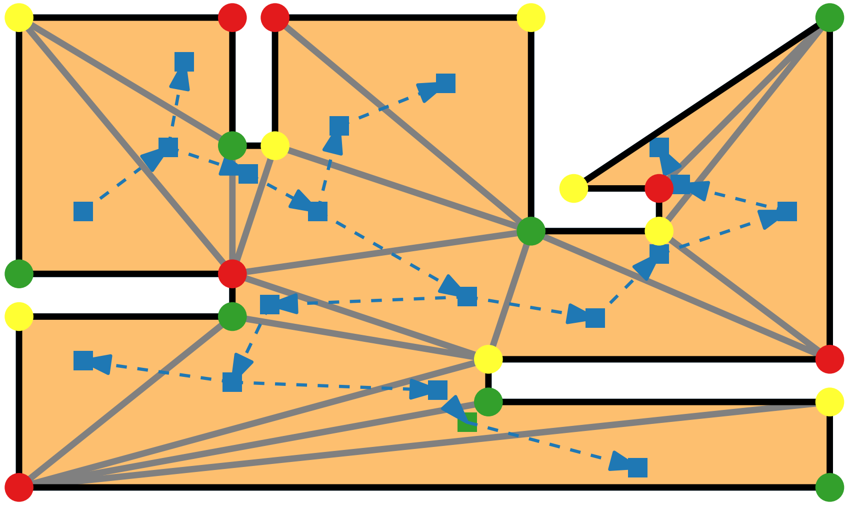
The Art Gallery Theorem

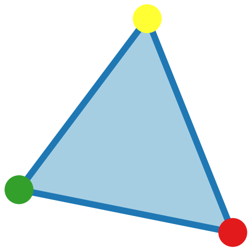
[Chvátal '75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

3-color the vtcs

Traverse the dual tree





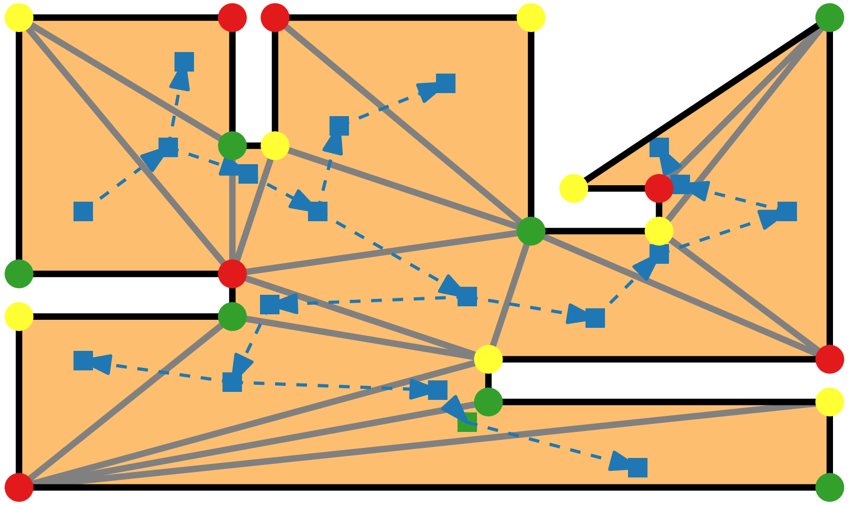
The Art Gallery Theorem

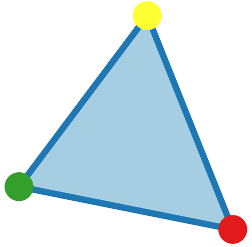
[Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

3-color the vtcs

Traverse the dual tree





Pick “smallest” color

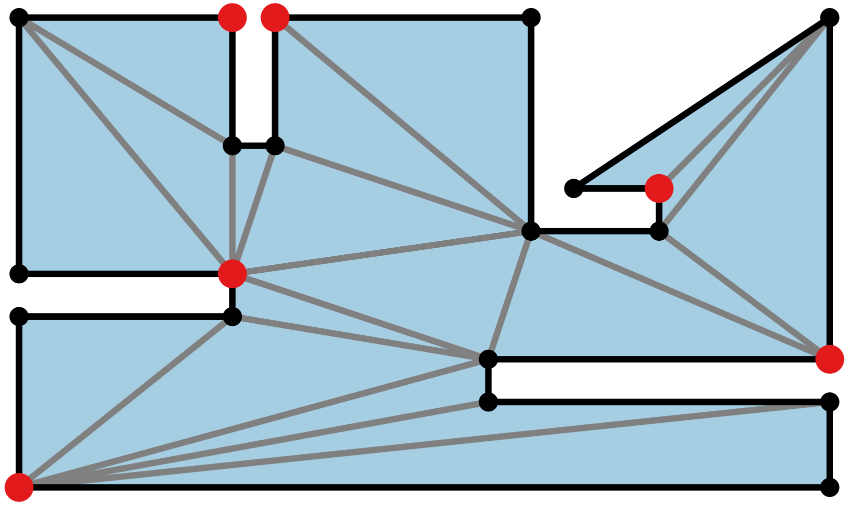
The Art Gallery Theorem

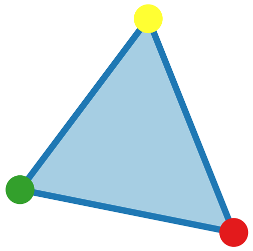
[Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

3-color the vtcs

Traverse the dual tree





Pick “smallest” color

The Art Gallery Theorem [Chvátal '75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

To do: Find algo. for triangulating a simple polygon!

The Art Gallery Theorem [Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

To do: Find algo. for triangulating a simple polygon!

Brute force:

The Art Gallery Theorem [Chvátal '75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

To do: Find algo. for triangulating a simple polygon!

Brute force: follow existence proof, using recursion

The Art Gallery Theorem [Chvátal '75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

To do: Find algo. for triangulating a simple polygon!

Brute force: follow existence proof, using recursion

running time:

The Art Gallery Theorem [Chvátal '75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

To do: Find algo. for triangulating a simple polygon!

Brute force: follow existence proof, using recursion
running time: $O(n^2)$

The Art Gallery Theorem

[Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

To do: Find algo. for triangulating a simple polygon!

Brute force: follow existence proof, using recursion running time: $O(n^2)$

Faster triangulation in two steps:

The Art Gallery Theorem [Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

To do: Find algo. for triangulating a simple polygon!

Brute force: follow existence proof, using recursion running time: $O(n^2)$

Faster triangulation in two steps:

n-vtx polygon

The Art Gallery Theorem

[Chvátal '75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

To do: Find algo. for triangulating a simple polygon!

Brute force: follow existence proof, using recursion
running time: $O(n^2)$

Faster triangulation in two steps:

n-vtx polygon → "nice" pieces, n' vtc →

The Art Gallery Theorem [Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

To do: Find algo. for triangulating a simple polygon!

Brute force: follow existence proof, using recursion
running time: $O(n^2)$

Faster triangulation in two steps:

n-vtx polygon $\longrightarrow$ “nice” pieces, n' vtc $\longrightarrow$ n'' triangles

The Art Gallery Theorem

[Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

To do: Find algo. for triangulating a simple polygon!

Brute force: follow existence proof, using recursion
running time: $O(n^2)$

Faster triangulation in two steps:

n-vtx polygon $\xrightarrow[O(n \log n)]{}$ “nice” pieces, n' vtc $\xrightarrow{}$ n'' triangles

The Art Gallery Theorem [Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

To do: Find algo. for triangulating a simple polygon!

Brute force: follow existence proof, using recursion
running time: $O(n^2)$

Faster triangulation in two steps:

n-vtx polygon $\xrightarrow[O(n \log n)]{}$ “nice” pieces, n' vtc $\xrightarrow[O(n')]{}$ n'' triangles

The Art Gallery Theorem [Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

To do: Find algo. for triangulating a simple polygon!

Brute force: follow existence proof, using recursion

running time: $O(n^2)$

Faster triangulation in two steps:

n-vtx polygon $\xrightarrow[O(n \log n)]{}$ “nice” pieces, n' vtc $\xrightarrow[O(n')]{}$ n'' triangles

Definition. A polygon P is *y-monotone* if, for any horizontal line ℓ, $\ell \cap P$ is connected.

The Art Gallery Theorem [Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

To do: Find algo. for triangulating a simple polygon!

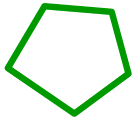
Brute force: follow existence proof, using recursion

running time: $O(n^2)$

Faster triangulation in two steps:

n-vtx polygon $\xrightarrow[O(n \log n)]{}$ “nice” pieces, n' vtc $\xrightarrow[O(n')]{}$ n'' triangles

Definition. A polygon P is *y-monotone* if, for any horizontal line ℓ, $\ell \cap P$ is connected.



The Art Gallery Theorem

[Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

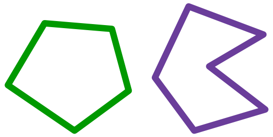
To do: Find algo. for triangulating a simple polygon!

Brute force: follow existence proof, using recursion
running time: $O(n^2)$

Faster triangulation in two steps:

n-vtx polygon $\xrightarrow[O(n \log n)]{}$ “nice” pieces, n' vtc $\xrightarrow[O(n')]{}$ n'' triangles

Definition. A polygon P is *y-monotone* if, for any horizontal line ℓ, $\ell \cap P$ is connected.



The Art Gallery Theorem

[Chvátal ’75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

To do: Find algo. for triangulating a simple polygon!

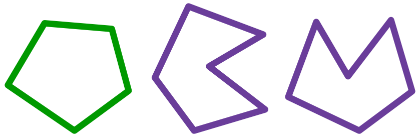
Brute force: follow existence proof, using recursion

running time: $O(n^2)$

Faster triangulation in two steps:

n-vtx polygon $\xrightarrow[O(n \log n)]{}$ “nice” pieces, n' vtc $\xrightarrow[O(n')]{}$ n'' triangles

Definition. A polygon P is *y-monotone* if, for any horizontal line ℓ, $\ell \cap P$ is connected.



The Art Gallery Theorem

[Chvátal '75]

Theorem. For surveilling a simple polygon with n vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient.

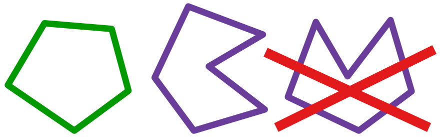
To do: Find algo. for triangulating a simple polygon!

Brute force: follow existence proof, using recursion
running time: $O(n^2)$

Faster triangulation in two steps:

n-vtx polygon $\xrightarrow[O(n \log n)]{}$ "nice" pieces, n' vtc $\xrightarrow[O(n')]{}$ n'' triangles

Definition. A polygon P is *y-monotone* if, for any horizontal line ℓ, $\ell \cap P$ is connected.



Part. a Polygon into Monotone Pieces

Idea: Classify vertices of given simple polygon P

Part. a Polygon into Monotone Pieces

Idea: Classify vertices of given simple polygon P

– *turn* vertices:

– *regular* vertices

Part. a Polygon into Monotone Pieces

Idea: Classify vertices of given simple polygon P

– *turn* vertices:

vertical component of walking direction changes

– *regular* vertices

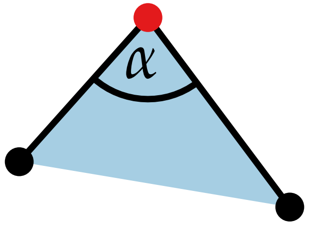
Part. a Polygon into Monotone Pieces

Idea: Classify vertices of given simple polygon P

– *turn* vertices:

vertical component of walking direction changes

- *start* vertex



if $\alpha < 180°$

– *regular* vertices

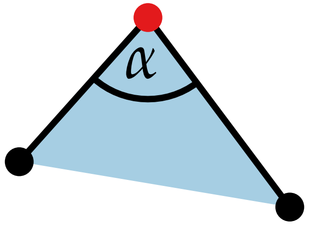
Part. a Polygon into Monotone Pieces

Idea: Classify vertices of given simple polygon P

– *turn* vertices:

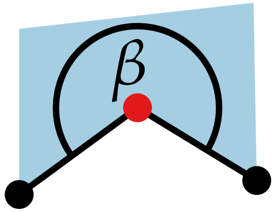
vertical component of walking direction changes

- *start* vertex



if $\alpha < 180^\circ$

- *split* vertex



if $\beta > 180^\circ$

– *regular* vertices

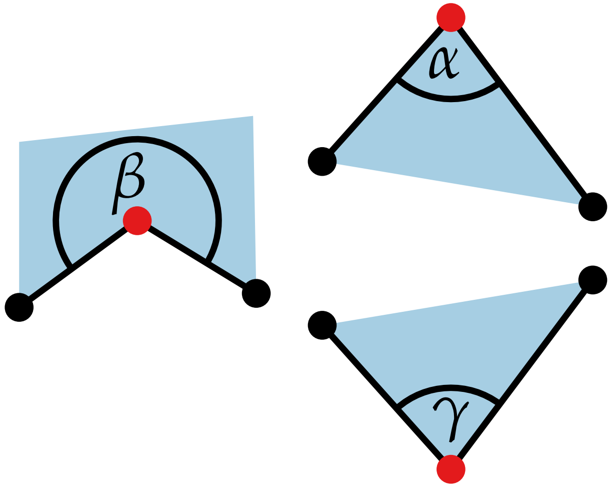
Part. a Polygon into Monotone Pieces

Idea: Classify vertices of given simple polygon P

– *turn* vertices:

vertical component of walking direction changes

- *start* vertex if $\alpha < 180^\circ$
- *split* vertex if $\beta > 180^\circ$
- *end* vertex if $\gamma < 180^\circ$



– *regular* vertices

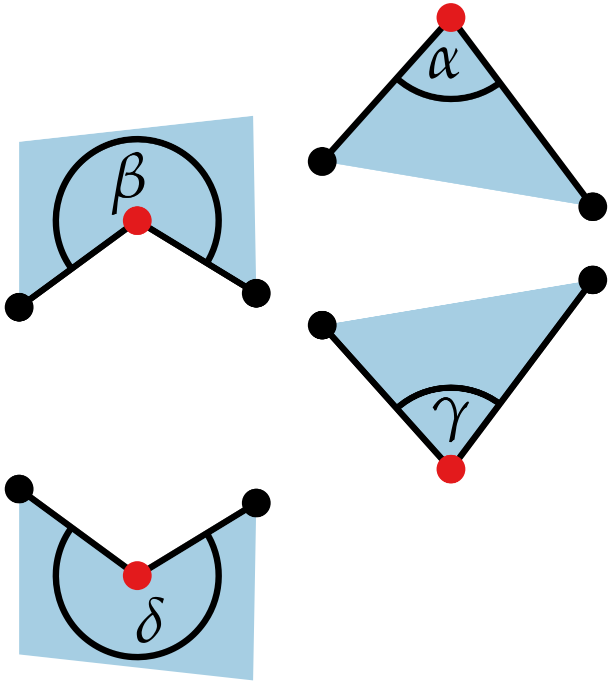
Part. a Polygon into Monotone Pieces

Idea: Classify vertices of given simple polygon P

– *turn* vertices:

vertical component of walking direction changes

- *start* vertex if $\alpha < 180^\circ$
- *split* vertex if $\beta > 180^\circ$
- *end* vertex if $\gamma < 180^\circ$
- *merge* vertex if $\delta > 180^\circ$



– *regular* vertices

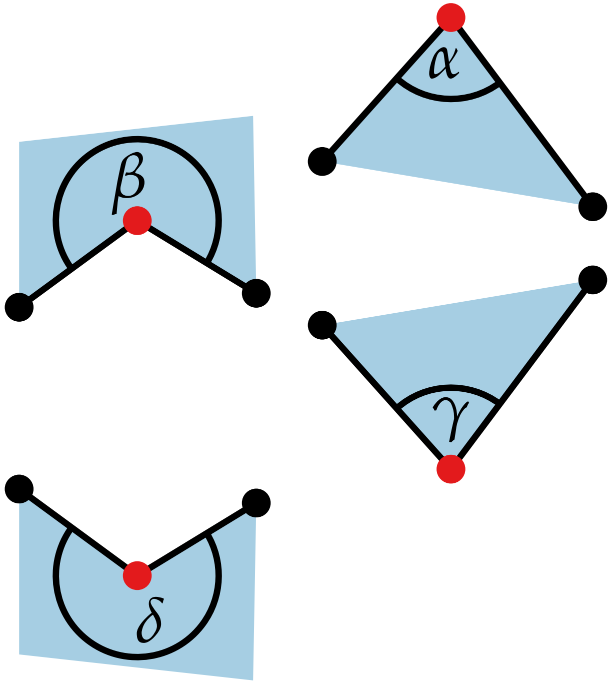
Part. a Polygon into Monotone Pieces

Idea: Classify vertices of given simple polygon P

– *turn* vertices:

vertical component of walking direction changes

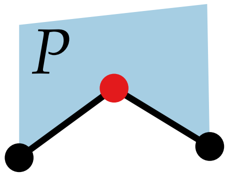
- *start* vertex if $\alpha < 180^\circ$
- *split* vertex if $\beta > 180^\circ$
- *end* vertex if $\gamma < 180^\circ$
- *merge* vertex if $\delta > 180^\circ$

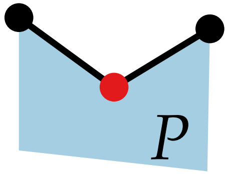


– *regular* vertices

Lemma: Let P be a simple polygon. Then P is y-monotone $\Leftrightarrow$ P has neither split vertices nor merge vertices.

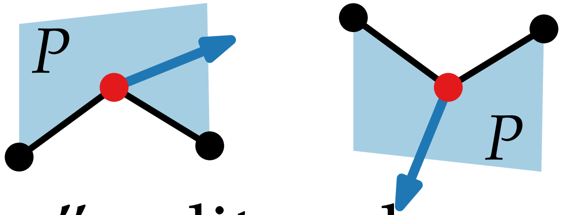
Towards an Algorithm





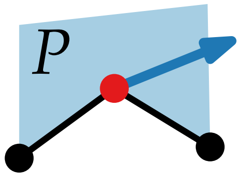
Idea: Add diagonals to “destroy” split and merge vtcs.

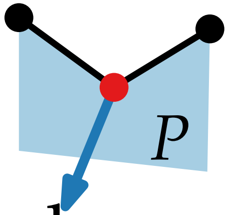
Towards an Algorithm



Idea: Add diagonals to “destroy” split and merge vtcs.

Towards an Algorithm

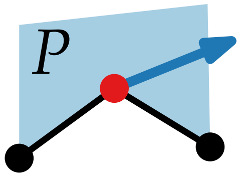


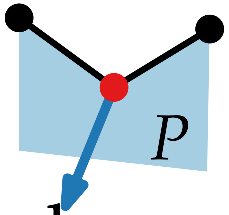


Idea: Add diagonals to “destroy” split and merge vtcs.

Problem: Diagonals must not cross:

Towards an Algorithm

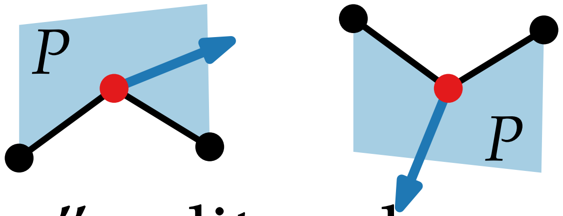




Idea: Add diagonals to “destroy” split and merge vtcs.

Problem: Diagonals must not cross:
– each other
– edges of P

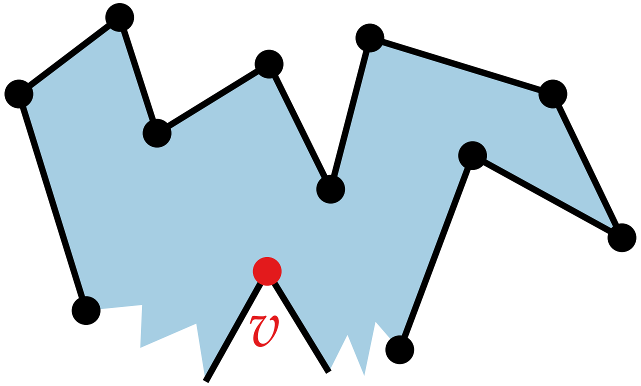
Towards an Algorithm



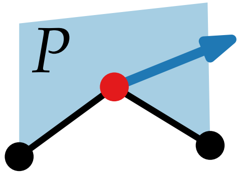
Idea: Add diagonals to “destroy” split and merge vtcs.

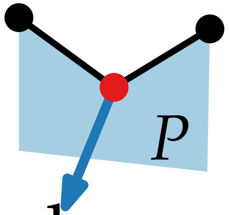
Problem: Diagonals must not cross:
- each other
- edges of P

1) Treating split vertices



Towards an Algorithm

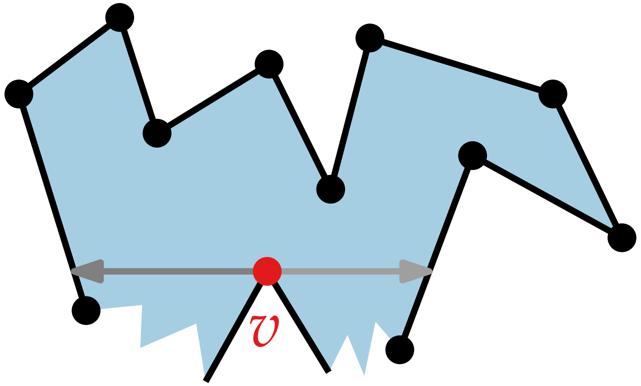




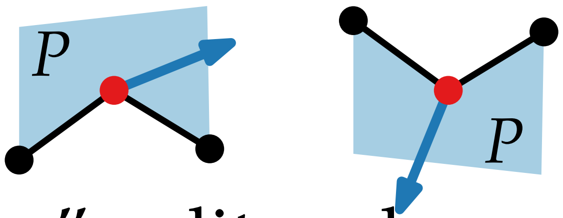
Idea: Add diagonals to “destroy” split and merge vtcs.

Problem: Diagonals must not cross:
- each other
- edges of P

1) Treating split vertices



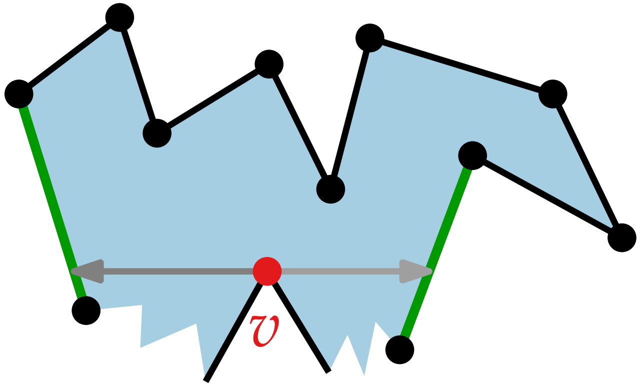
Towards an Algorithm



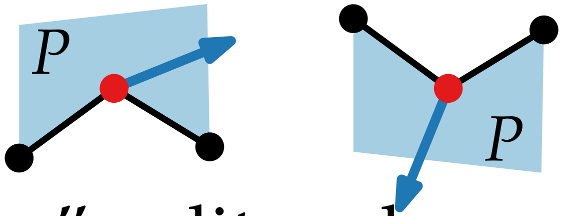
Idea: Add diagonals to “destroy” split and merge vtcs.

Problem: Diagonals must not cross:
- each other
- edges of P

1) Treating split vertices



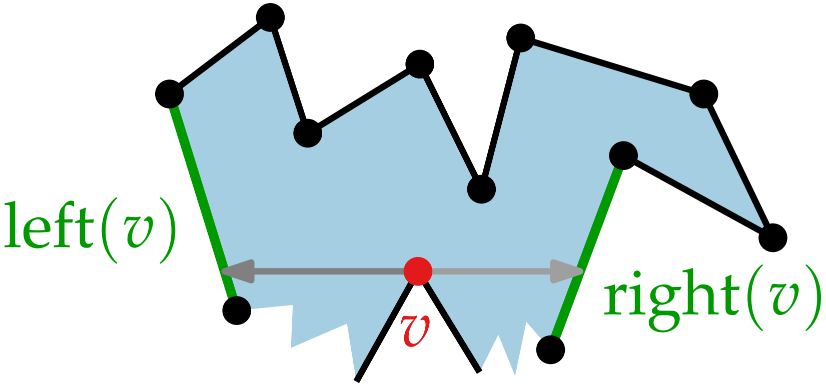
Towards an Algorithm



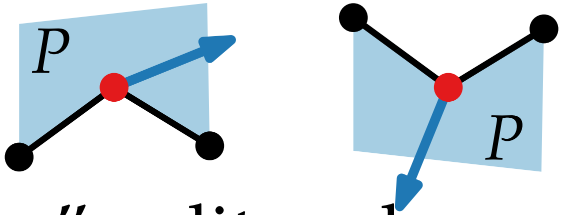
Idea: Add diagonals to "destroy" split and merge vtcs.

Problem: Diagonals must not cross:
– each other
– edges of P

1) Treating split vertices



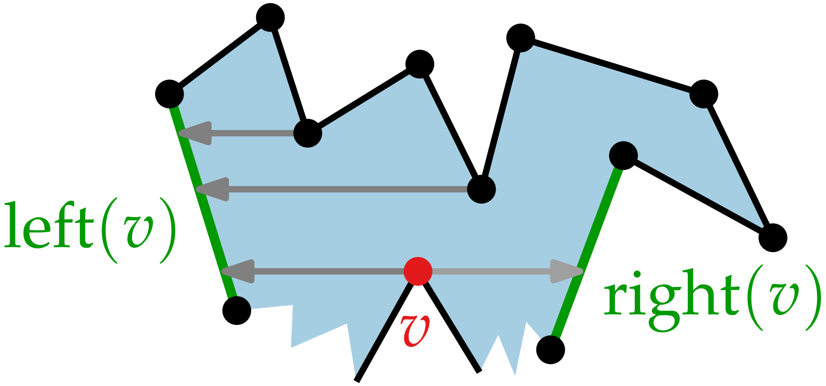
Towards an Algorithm



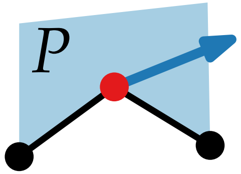
Idea: Add diagonals to “destroy” split and merge vtcs.

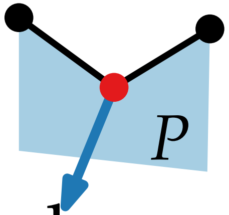
Problem: Diagonals must not cross:
- each other
- edges of P

1) Treating split vertices



Towards an Algorithm

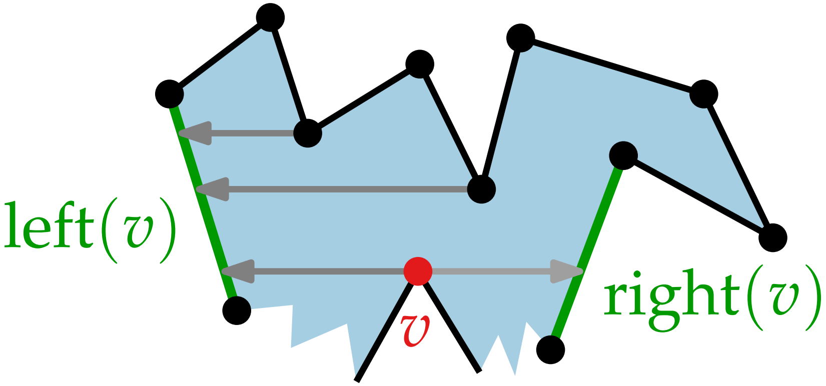




Idea: Add diagonals to "destroy" split and merge vtcs.

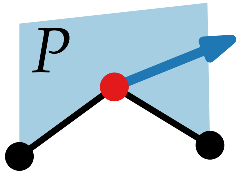
Problem: Diagonals must not cross:
– each other
– edges of P

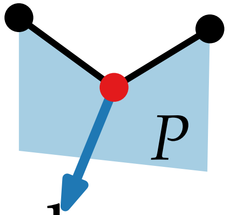
1) Treating split vertices



Connect v to vertex $w^\star$ having minimum y-coordinate among all vertices w above v and with $\text{left}(w) = \text{left}(v)$.

Towards an Algorithm

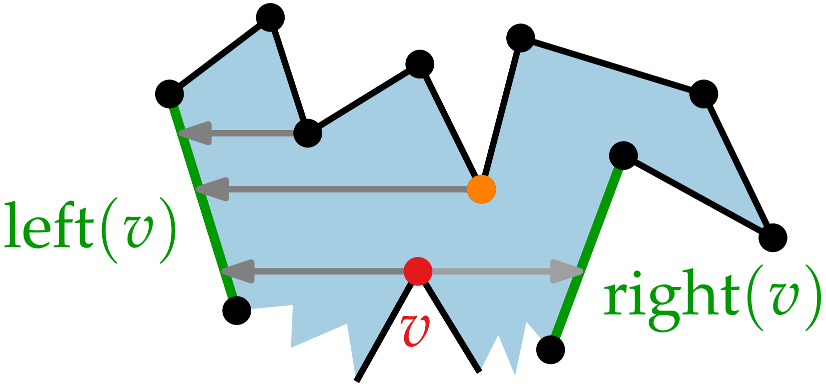




Idea: Add diagonals to "destroy" split and merge vtcs.

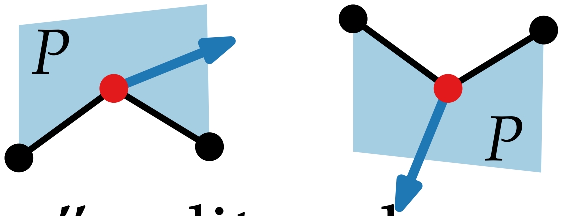
Problem: Diagonals must not cross:
– each other
– edges of P

1) Treating split vertices



Connect v to vertex $w^\star$ having minimum y-coordinate among all vertices w above v and with $\text{left}(w) = \text{left}(v)$.

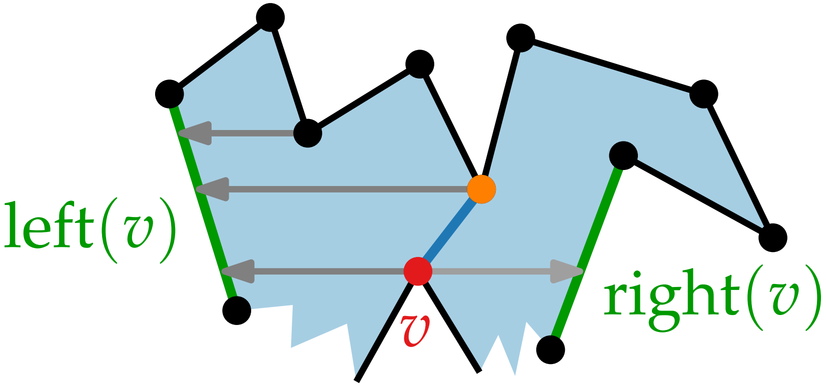
Towards an Algorithm



Idea: Add diagonals to “destroy” split and merge vtcs.

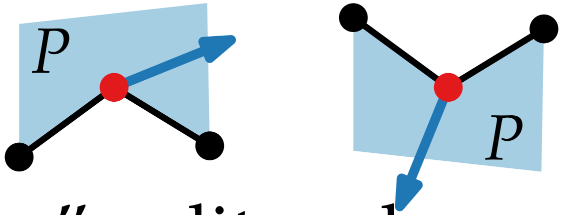
Problem: Diagonals must not cross:
– each other
– edges of P

1) Treating split vertices



Connect v to vertex $w^\star$ having minimum y-coordinate among all vertices w above v and with $\mathrm{left}(w) = \mathrm{left}(v)$.

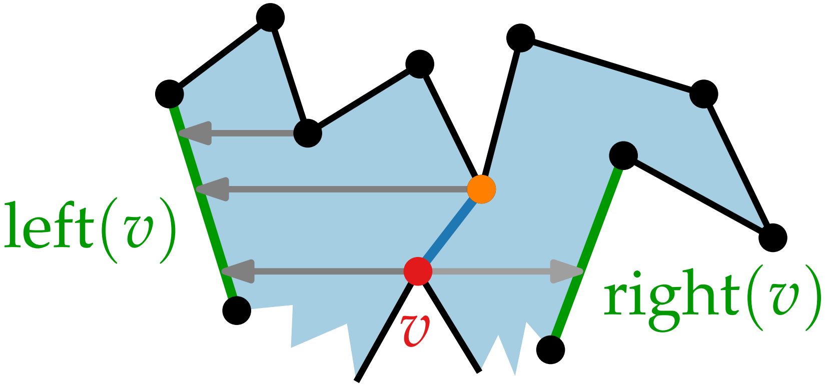
Towards an Algorithm



Idea: Add diagonals to "destroy" split and merge vtcs.

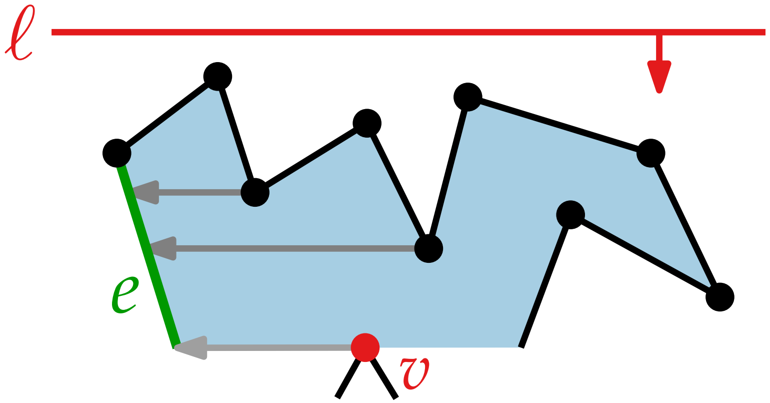
Problem: Diagonals must not cross:
- each other
- edges of P

1) Treating split vertices

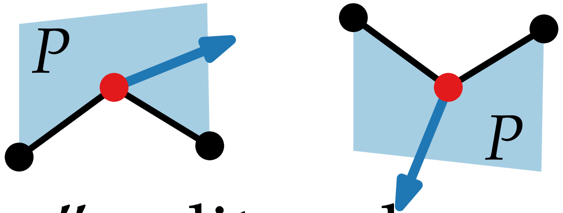


Connect v to vertex $w^\star$ having minimum y-coordinate among all vertices w above v and with $\text{left}(w) = \text{left}(v)$.

Think of a sweep-line algorithm:



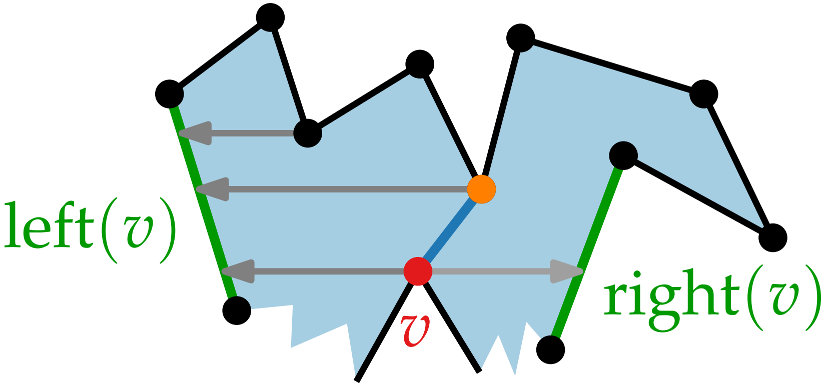
Towards an Algorithm



Idea: Add diagonals to “destroy” split and merge vtcs.

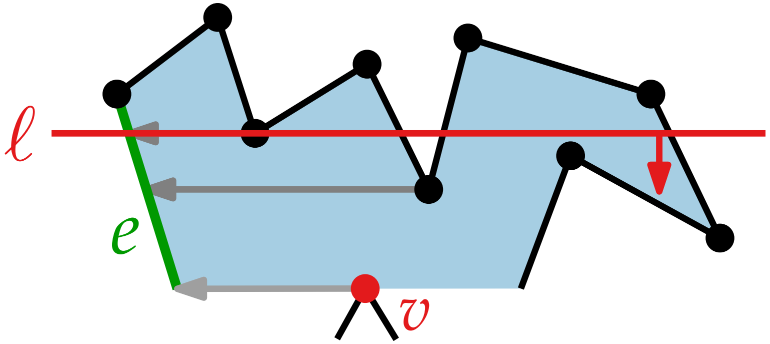
Problem: Diagonals must not cross:
- each other
- edges of P

1) Treating split vertices

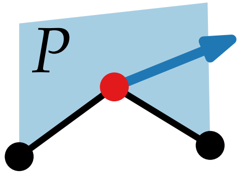


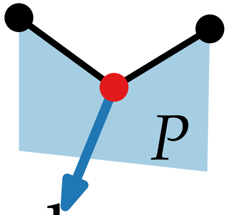
Connect v to vertex $w^\star$ having minimum y-coordinate among all vertices w above v and with left(w) = left(v).

Think of a sweep-line algorithm:



Towards an Algorithm

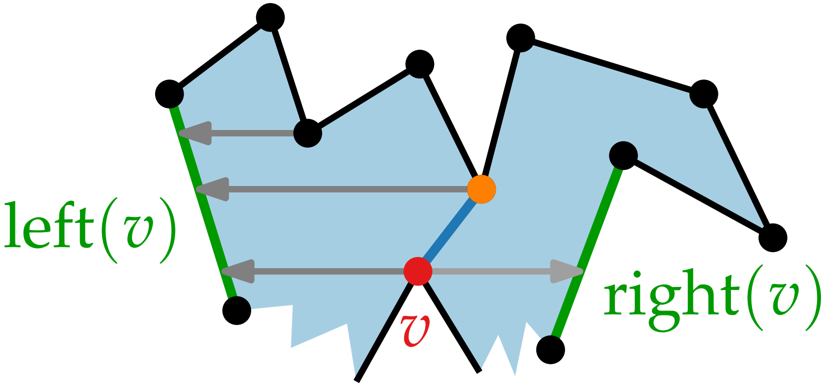




Idea: Add diagonals to “destroy” split and merge vtcs.

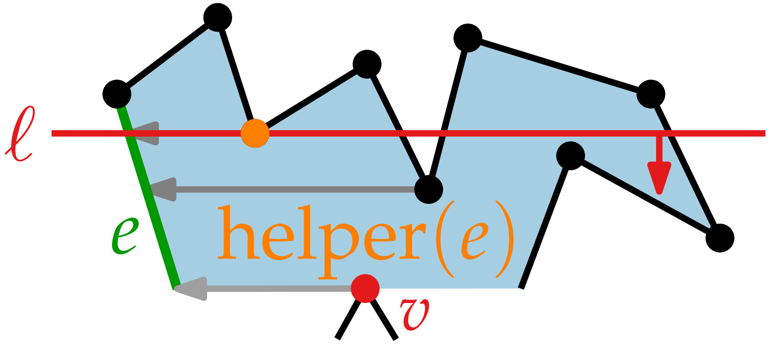
Problem: Diagonals must not cross:
– each other
– edges of P

1) Treating split vertices

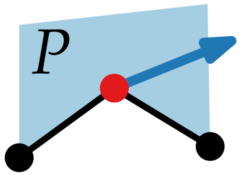


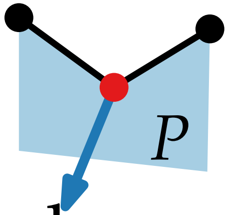
Connect v to vertex $w^\star$ having minimum y-coordinate among all vertices w above v and with $\text{left}(w) = \text{left}(v)$.

Think of a sweep-line algorithm:



Towards an Algorithm

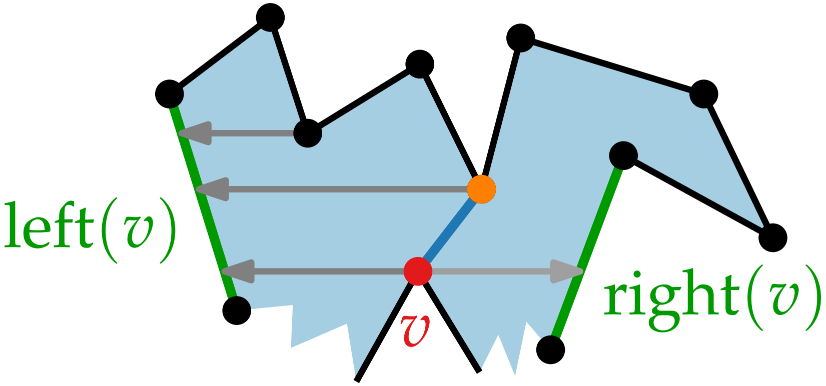




Idea: Add diagonals to “destroy” split and merge vtcs.

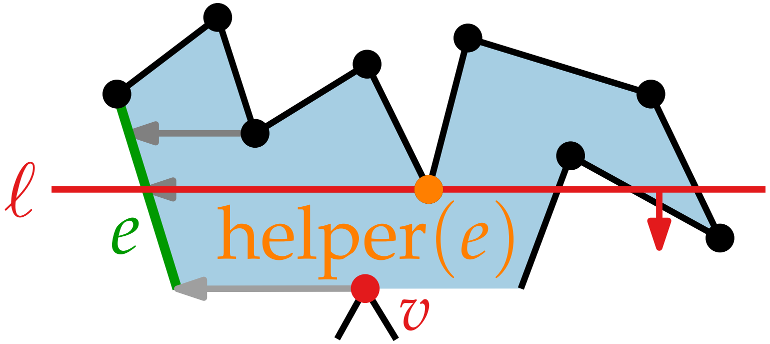
Problem: Diagonals must not cross:
– each other
– edges of P

1) Treating split vertices

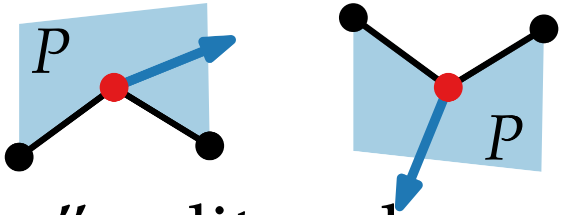


Connect v to vertex $w^\star$ having minimum y-coordinate among all vertices w above v and with left(w) = left(v).

Think of a sweep-line algorithm:



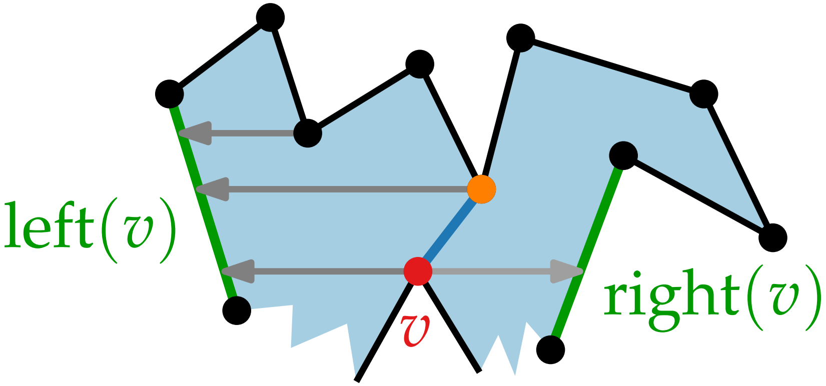
Towards an Algorithm



Idea: Add diagonals to “destroy” split and merge vtcs.

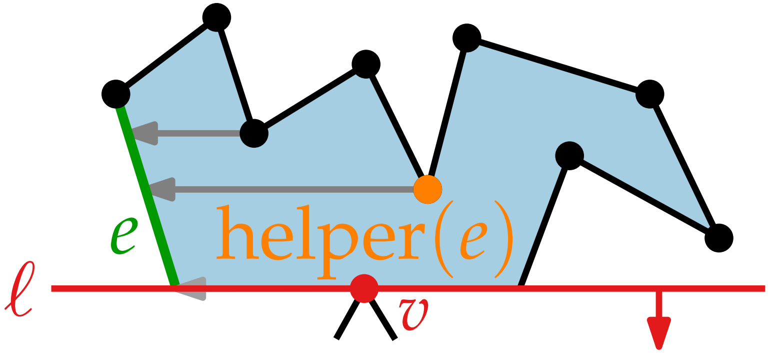
Problem: Diagonals must not cross:
- each other
- edges of P

1) Treating split vertices

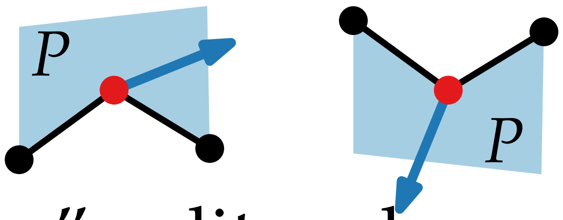


Connect v to vertex $w^\star$ having minimum y-coordinate among all vertices w above v and with left(w) = left(v).

Think of a sweep-line algorithm:



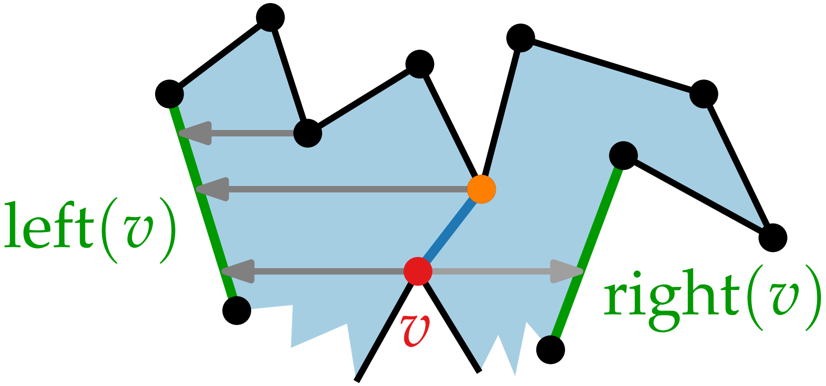
Towards an Algorithm



Idea: Add diagonals to “destroy” split and merge vtcs.

Problem: Diagonals must not cross:
- each other
- edges of P

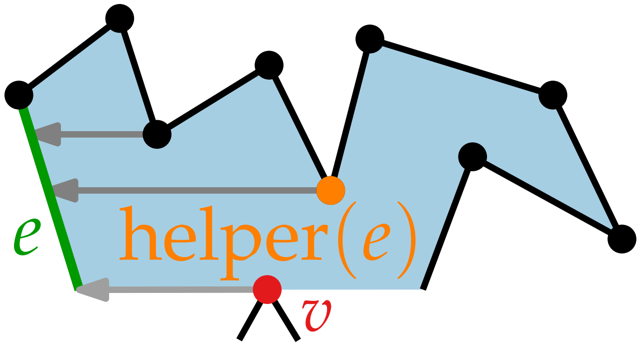
1) Treating split vertices



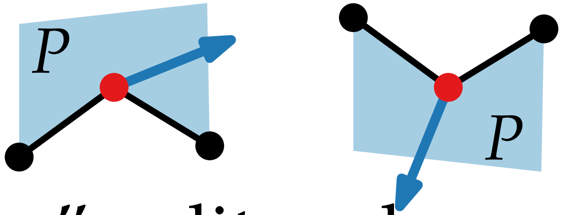
Connect v to vertex $w^\star$ having minimum y-coordinate among all vertices w above v and with $\text{left}(w) = \text{left}(v)$.

Think of a sweep-line algorithm:

Connect v to helper(left(v)).



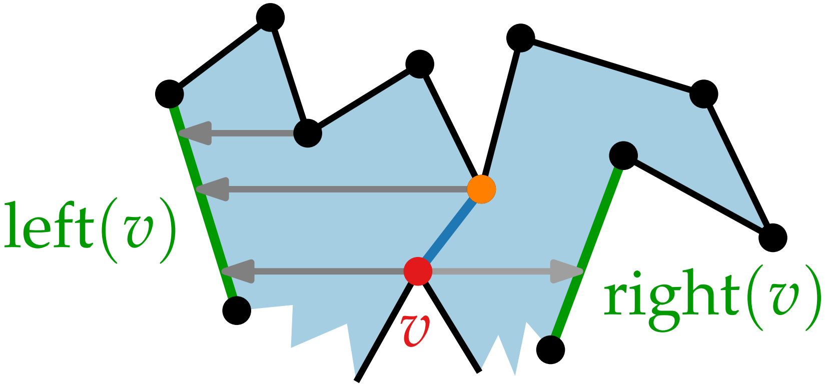
Towards an Algorithm



Idea: Add diagonals to "destroy" split and merge vtcs.

Problem: Diagonals must not cross:
- each other
- edges of P

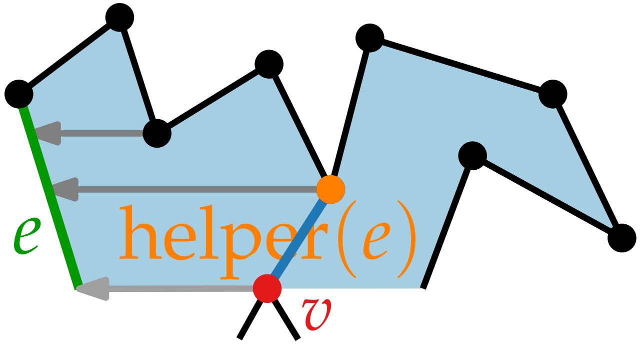
1) Treating split vertices



Connect v to vertex $w^\star$ having minimum y-coordinate among all vertices w above v and with left$(w) = $ left(v).

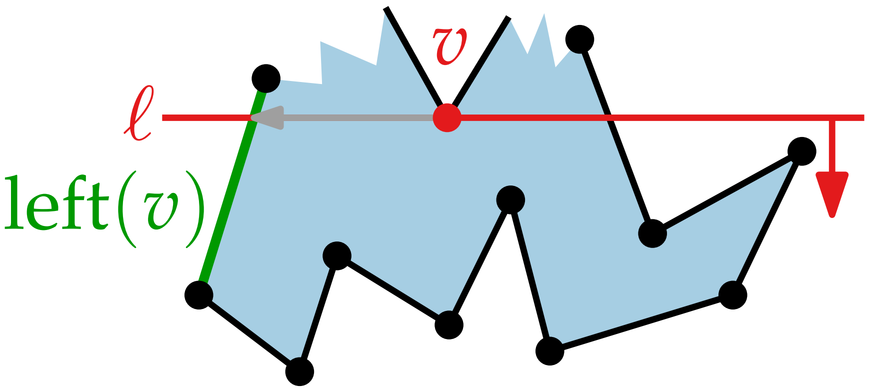
Think of a sweep-line algorithm:

Connect v to helper(left(v)).



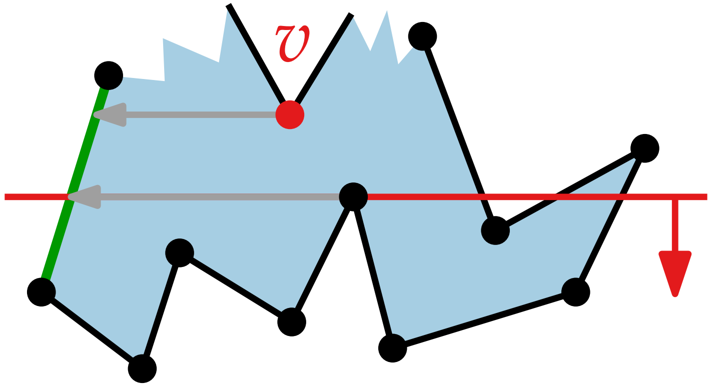
An Algorithm

2) Treating merge vertices



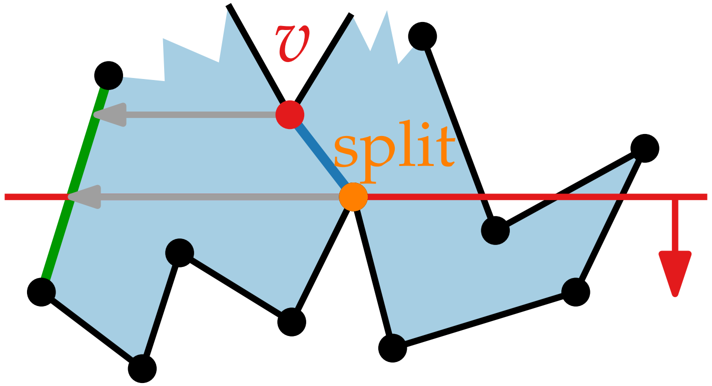
An Algorithm

2) Treating merge vertices



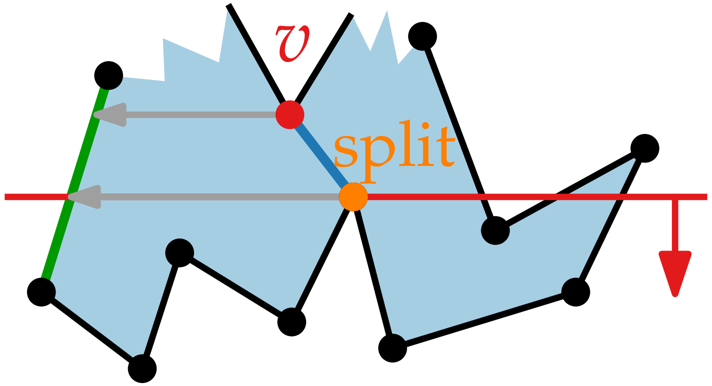
An Algorithm

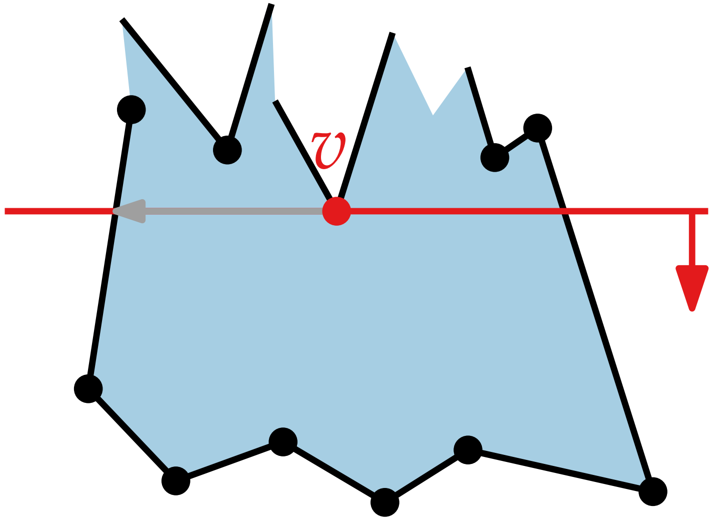
2) Treating merge vertices



An Algorithm

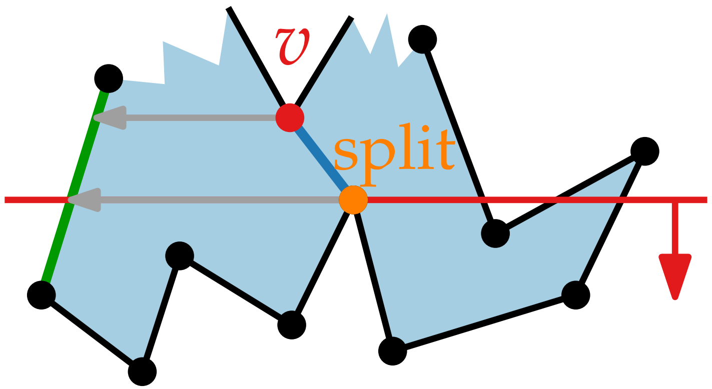
2) Treating merge vertices

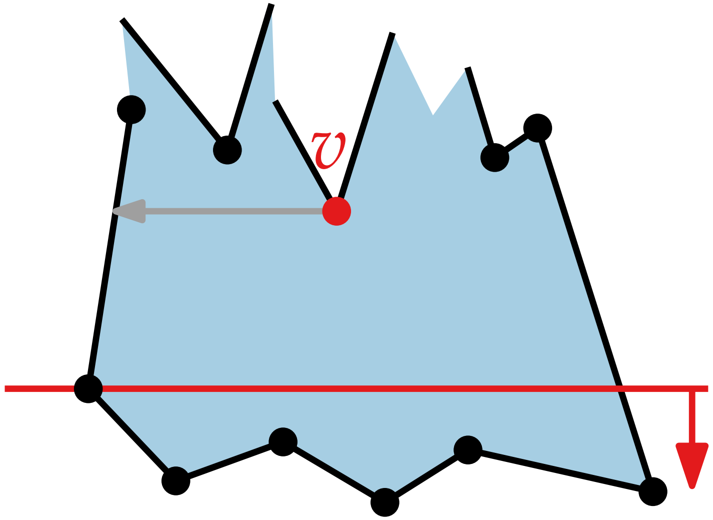




An Algorithm

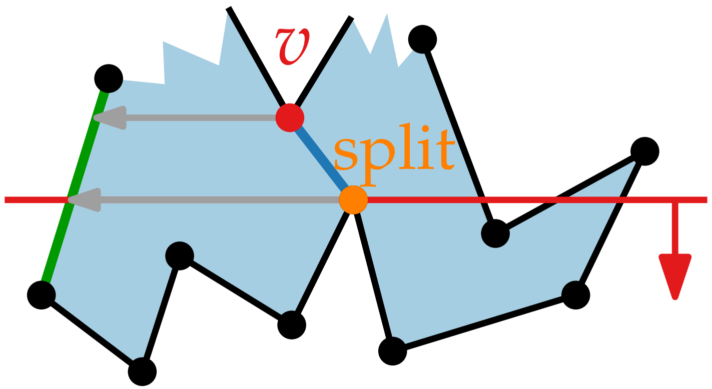
2) Treating merge vertices

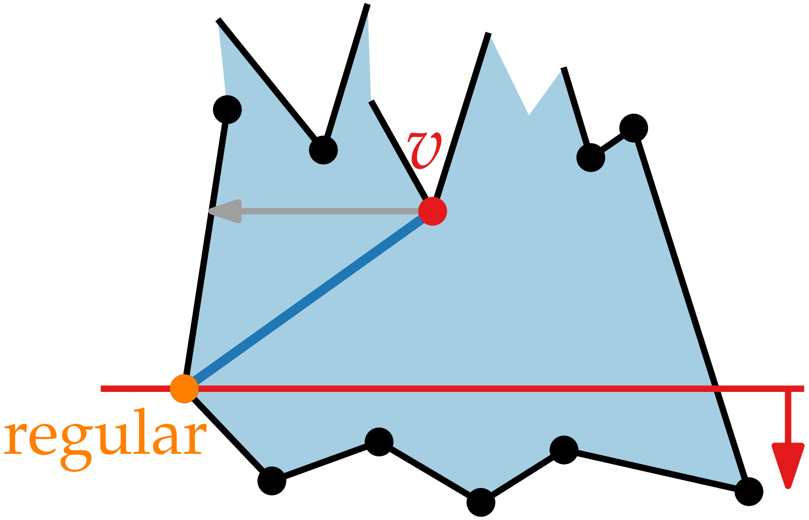




An Algorithm

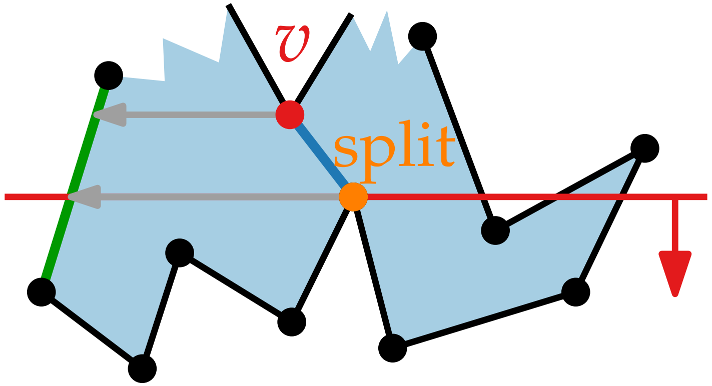
2) Treating merge vertices

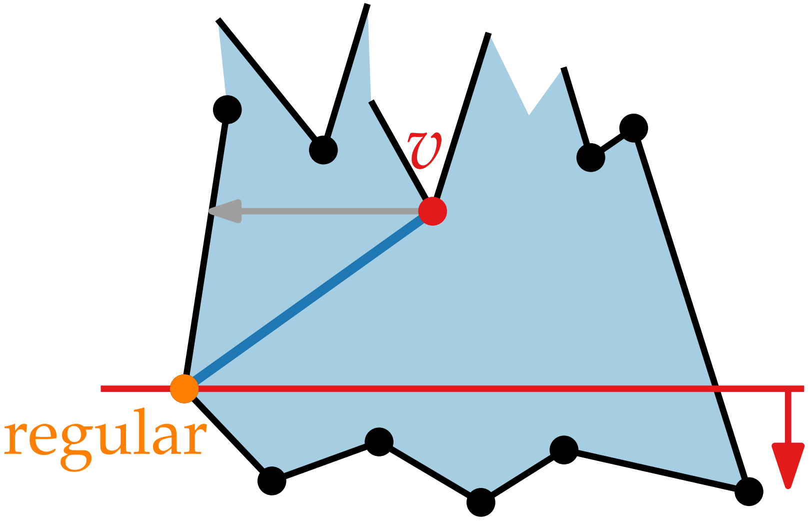




An Algorithm

2) Treating merge vertices

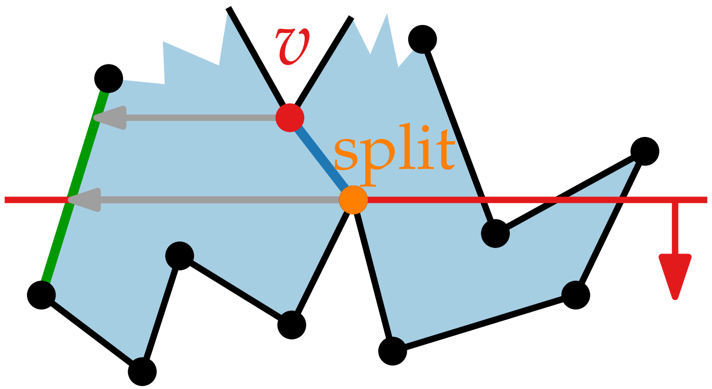


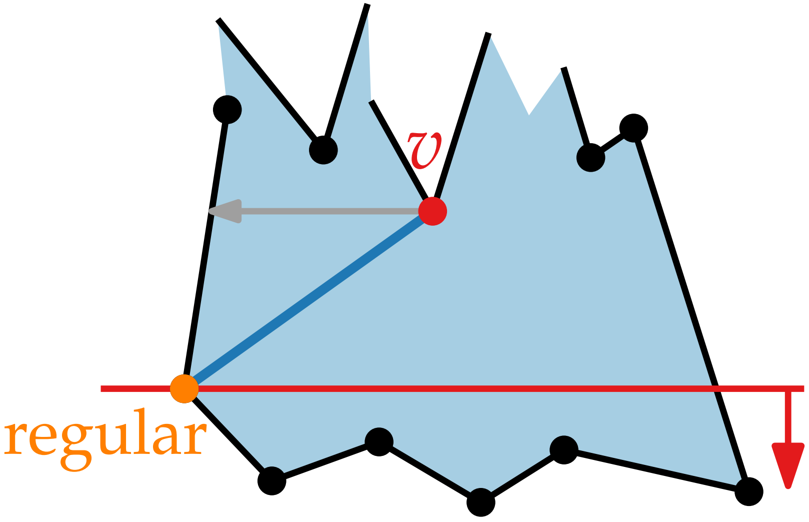


```
**makeMonotone**(polygon $P$)
$\mathcal{D} \leftarrow \text{DCEL}(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue on $V(P)$
$\mathcal{T} \leftarrow$ empty bin. search tree
```

An Algorithm

2) Treating merge vertices



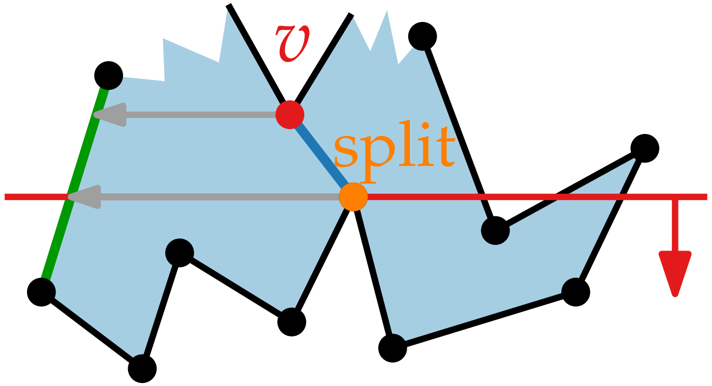


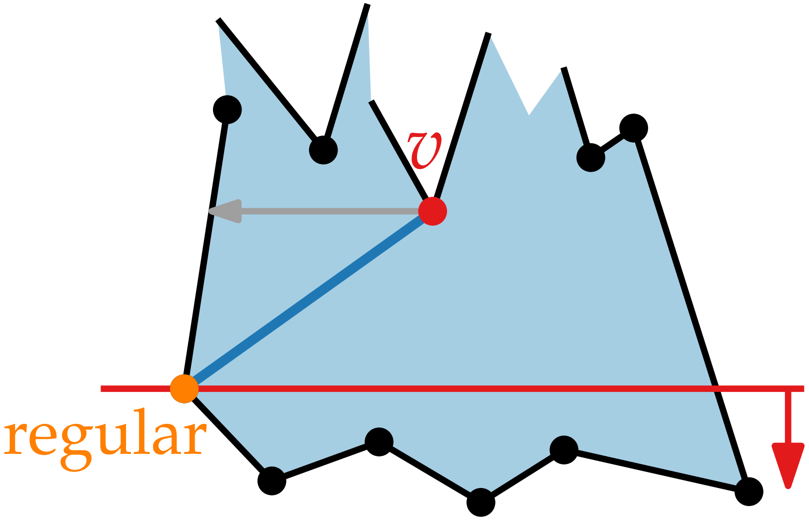
```
**makeMonotone(**polygon $P$**)**
$\mathcal{D} \leftarrow \mathrm{DCEL}(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue on $V(P)$
$\mathcal{T} \leftarrow$ empty bin. search tree
```

doubly-connected edge list:
data structure for planar subdivisions

An Algorithm

2) Treating merge vertices





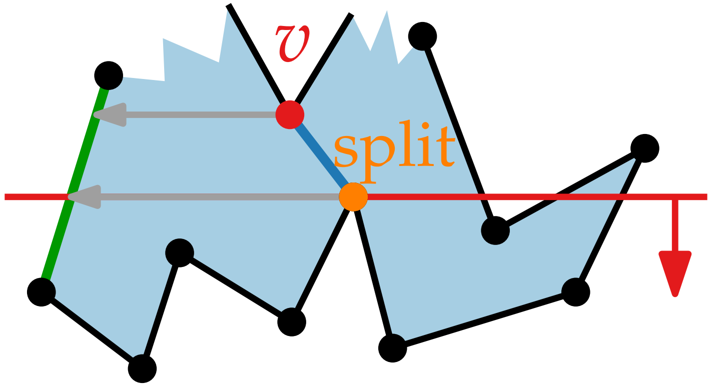
```
makeMonotone(polygon $P$)
$\mathcal{D} \leftarrow$ DCEL$(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue on $V(P)$
$\mathcal{T} \leftarrow$ empty bin. search tree
```

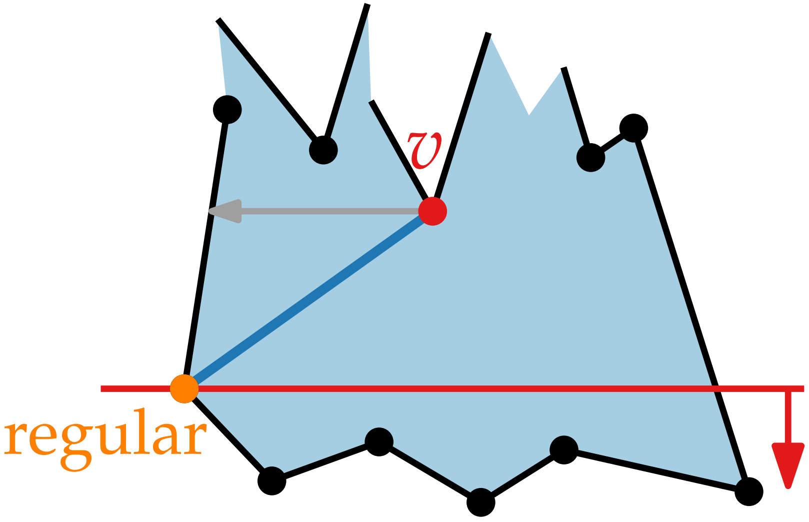
doubly-connected edge list:
data structure for planar subdivisions

$(x,y) \prec (x',y') \;:\Leftrightarrow$
$y > y' \;\vee\; (y = y' \wedge x < x')$

An Algorithm

2) Treating merge vertices

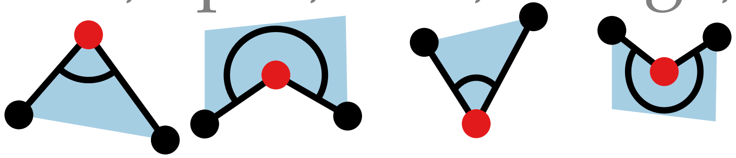




```
makeMonotone(polygon $P$)
$\mathcal{D} \leftarrow$ DCEL$(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue on $V(P)$
$\mathcal{T} \leftarrow$ empty bin. search tree
while $\mathcal{Q} \neq \emptyset$ do
    $v \leftarrow \mathcal{Q}$.extractMax()
    type $\leftarrow$ type of vertex $v \in$ start, split, end, merge, regular
    handleVertex$_{\text{type}}(v)$
return DCEL $\mathcal{D}$
```

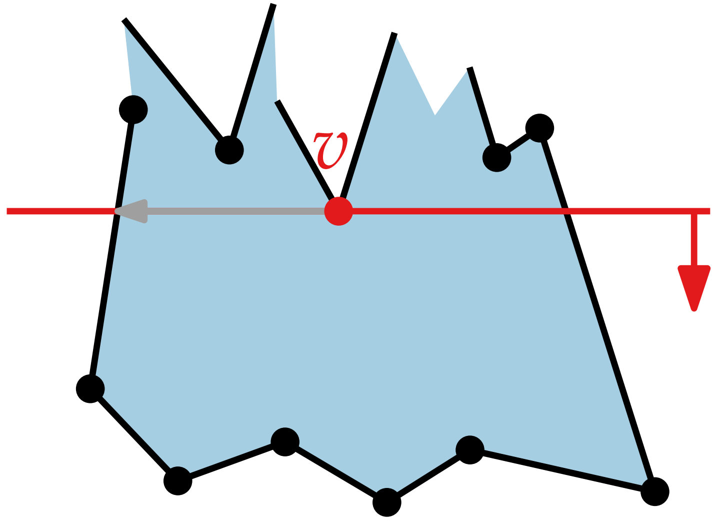
doubly-connected edge list:
data structure for planar subdivisions

$$(x,y) \prec (x',y') \;:\Leftrightarrow\; y > y' \;\vee\; (y = y' \wedge x < x')$$



An Algorithm

2) Treating merge vertices

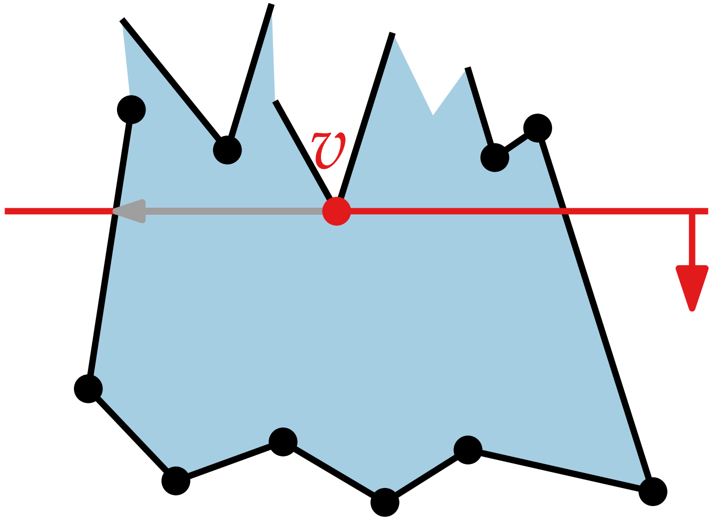


```
makeMonotone(polygon $P$)
$\mathcal{D} \leftarrow$ DCEL$(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue on $V(P)$
$\mathcal{T} \leftarrow$ empty bin. search tree
while $\mathcal{Q} \neq \emptyset$ do
    $v \leftarrow \mathcal{Q}$.extractMax()
    type $\leftarrow$ type of vertex $v$
    handleVertex$_{\text{type}}(v)$
return DCEL $\mathcal{D}$
```

```
handleVertex$_{\text{merge}}$(vertex $v$)
$e \leftarrow$ edge following $v$ ccw
if helper$(e)$ merge vtx then
    $\mathcal{D}$.insert(diag($v$, helper$(e)$))
$\mathcal{T}$.delete$(e)$
$e' \leftarrow \mathcal{T}$.edgeLeftOf$(v)$
if helper$(e')$ merge vtx then
    $\mathcal{D}$.insert(diag($v$, helper$(e')$))
helper$(e') \leftarrow v$
```

An Algorithm

2) Treating merge vertices

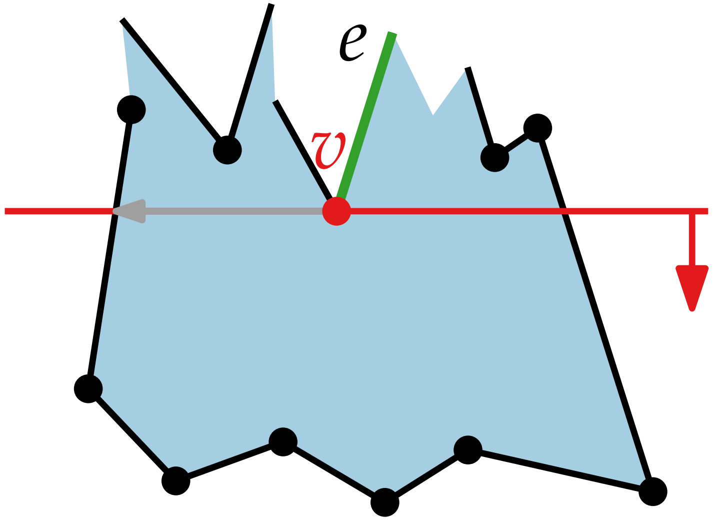


```
makeMonotone(polygon $P$)
$\mathcal{D} \leftarrow$ DCEL$(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue on $V(P)$
$\mathcal{T} \leftarrow$ empty bin. search tree
while $\mathcal{Q} \neq \emptyset$ do
    $v \leftarrow \mathcal{Q}$.extractMax()
    type $\leftarrow$ type of vertex $v$
    handleVertex$_{\text{type}}(v)$
return DCEL $\mathcal{D}$
```

```
handleVertex$_{\text{merge}}$(vertex $v$)
$e \leftarrow$ edge following $v$ ccw
if helper($e$) merge vtx then
    $\mathcal{D}$.insert(diag($v$, helper($e$)))
$\mathcal{T}$.delete($e$)
$e' \leftarrow \mathcal{T}$.edgeLeftOf($v$)
if helper($e'$) merge vtx then
    $\mathcal{D}$.insert(diag($v$, helper($e'$)))
helper($e'$) $\leftarrow v$
```

An Algorithm

2) Treating merge vertices

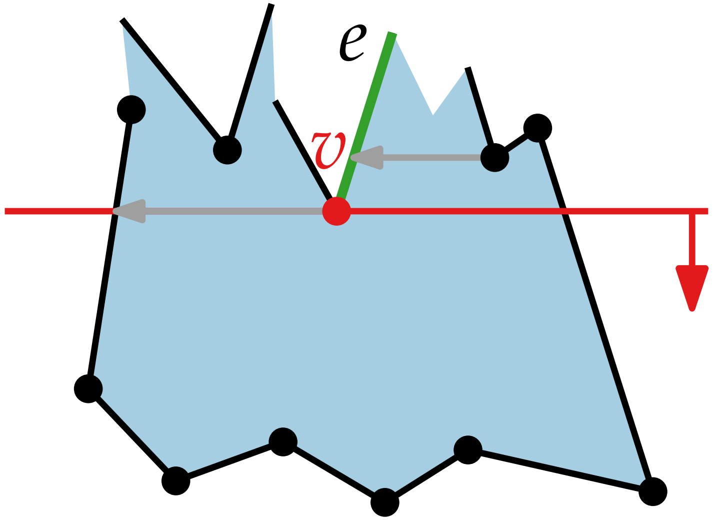


```
makeMonotone(polygon $P$)
$\mathcal{D} \leftarrow$ DCEL($V(P), E(P)$)
$\mathcal{Q} \leftarrow$ priority queue on $V(P)$
$\mathcal{T} \leftarrow$ empty bin. search tree
while $\mathcal{Q} \neq \emptyset$ do
    $v \leftarrow \mathcal{Q}$.extractMax()
    type $\leftarrow$ type of vertex $v$
    handleVertex$_{\text{type}}(v)$
return DCEL $\mathcal{D}$
```

```
handleVertex$_{\text{merge}}$(vertex $v$)
$e \leftarrow$ edge following $v$ ccw
if helper($e$) merge vtx then
    $\mathcal{D}$.insert(diag($v$, helper($e$)))
$\mathcal{T}$.delete($e$)
$e' \leftarrow \mathcal{T}$.edgeLeftOf($v$)
if helper($e'$) merge vtx then
    $\mathcal{D}$.insert(diag($v$, helper($e'$)))
helper($e'$) $\leftarrow v$
```

An Algorithm

2) Treating merge vertices

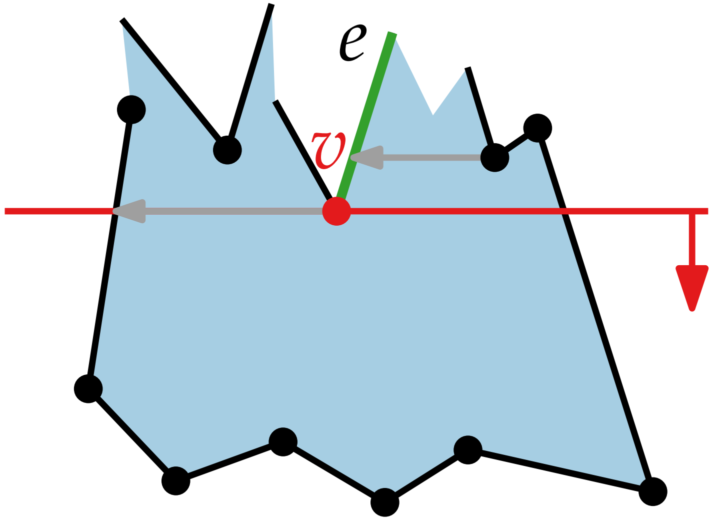


```
makeMonotone(polygon $P$)
$\mathcal{D} \leftarrow$ DCEL$(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue on $V(P)$
$\mathcal{T} \leftarrow$ empty bin. search tree
while $\mathcal{Q} \neq \emptyset$ do
    $v \leftarrow \mathcal{Q}$.extractMax()
    type $\leftarrow$ type of vertex $v$
    handleVertex$_{\text{type}}(v)$
return DCEL $\mathcal{D}$
```

```
handleVertex$_{\text{merge}}$(vertex $v$)
$e \leftarrow$ edge following $v$ ccw
if helper($e$) merge vtx then
    $\mathcal{D}$.insert(diag($v$, helper$(e)$))
$\mathcal{T}$.delete$(e)$
$e' \leftarrow \mathcal{T}$.edgeLeftOf$(v)$
if helper$(e')$ merge vtx then
    $\mathcal{D}$.insert(diag($v$, helper$(e')$))
helper$(e') \leftarrow v$
```

An Algorithm

2) Treating merge vertices

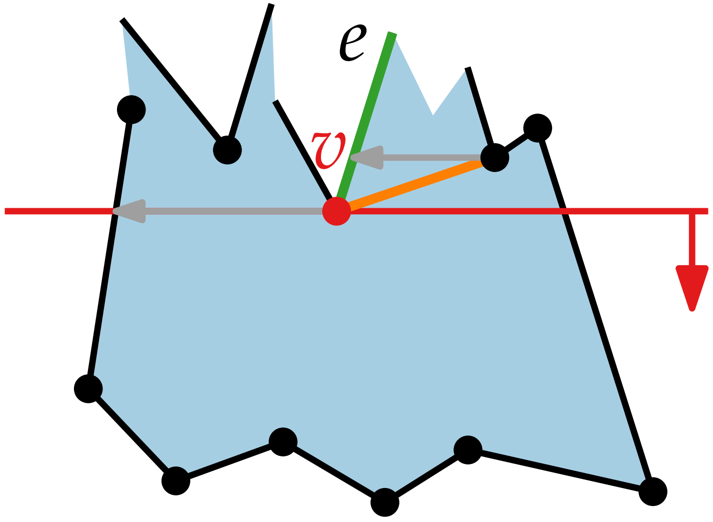


```
makeMonotone(polygon $P$)
$\mathcal{D} \leftarrow$ DCEL$(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue on $V(P)$
$\mathcal{T} \leftarrow$ empty bin. search tree
while $\mathcal{Q} \neq \emptyset$ do
    $v \leftarrow \mathcal{Q}$.extractMax()
    type $\leftarrow$ type of vertex $v$
    handleVertex$_{\text{type}}(v)$
return DCEL $\mathcal{D}$
```

```
handleVertex$_{\text{merge}}$(vertex $v$)
$e \leftarrow$ edge following $v$ ccw
if helper$(e)$ merge vtx then
    $\mathcal{D}$.insert(diag($v$, helper$(e)$))
$\mathcal{T}$.delete$(e)$
$e' \leftarrow \mathcal{T}$.edgeLeftOf$(v)$
if helper$(e')$ merge vtx then
    $\mathcal{D}$.insert(diag($v$, helper$(e')$))
helper$(e') \leftarrow v$
```

An Algorithm

2) Treating merge vertices

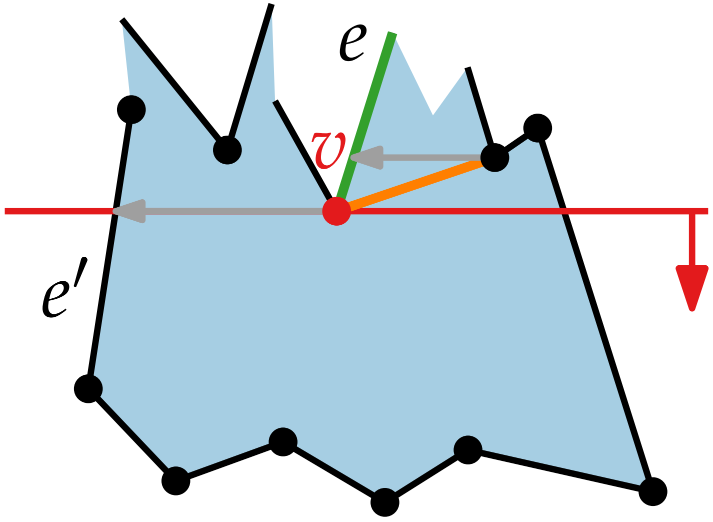


```
makeMonotone(polygon $P$)
$\mathcal{D} \leftarrow$ DCEL$(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue on $V(P)$
$\mathcal{T} \leftarrow$ empty bin. search tree
while $\mathcal{Q} \neq \emptyset$ do
    $v \leftarrow \mathcal{Q}$.extractMax()
    type $\leftarrow$ type of vertex $v$
    handleVertex$_{\text{type}}(v)$
return DCEL $\mathcal{D}$
```

```
handleVertex$_{\text{merge}}$(vertex $v$)
$e \leftarrow$ edge following $v$ ccw
if helper$(e)$ merge vtx then
    $\mathcal{D}$.insert(diag$(v$, helper$(e)$))
$\mathcal{T}$.delete$(e)$
$e' \leftarrow \mathcal{T}$.edgeLeftOf$(v)$
if helper$(e')$ merge vtx then
    $\mathcal{D}$.insert(diag$(v$, helper$(e')$))
helper$(e') \leftarrow v$
```

An Algorithm

2) Treating merge vertices

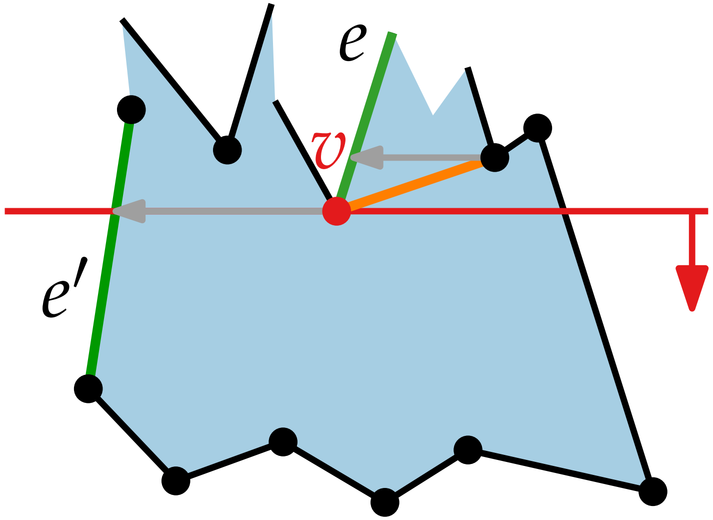


```
makeMonotone(polygon $P$)
$\mathcal{D} \leftarrow$ DCEL$(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue on $V(P)$
$\mathcal{T} \leftarrow$ empty bin. search tree
while $\mathcal{Q} \neq \emptyset$ do
    $v \leftarrow \mathcal{Q}$.extractMax()
    type $\leftarrow$ type of vertex $v$
    handleVertex$_{\text{type}}(v)$
return DCEL $\mathcal{D}$
```

```
handleVertex$_{\text{merge}}$(vertex $v$)
$e \leftarrow$ edge following $v$ ccw
if helper$(e)$ merge vtx then
    $\mathcal{D}$.insert(diag($v$, helper$(e)$))
$\mathcal{T}$.delete$(e)$
$e' \leftarrow \mathcal{T}$.edgeLeftOf$(v)$
if helper$(e')$ merge vtx then
    $\mathcal{D}$.insert(diag($v$, helper$(e')$))
helper$(e') \leftarrow v$
```

An Algorithm

2) Treating merge vertices

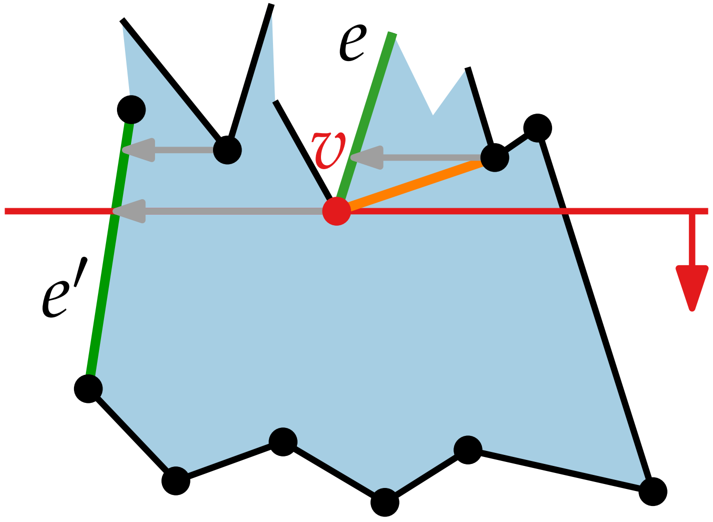


```
**makeMonotone**(polygon $P$)
$\mathcal{D} \leftarrow$ DCEL$(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue on $V(P)$
$\mathcal{T} \leftarrow$ empty bin. search tree
**while** $\mathcal{Q} \neq \emptyset$ **do**
    $v \leftarrow \mathcal{Q}$.extractMax()
    type $\leftarrow$ type of vertex $v$
    handleVertex$_{\text{type}}(v)$
**return** DCEL $\mathcal{D}$
```

```
**handleVertex$_{\text{merge}}$**(vertex $v$)
$e \leftarrow$ edge following $v$ ccw
**if** helper$(e)$ merge vtx **then**
    $\mathcal{D}$.insert(diag($v$, helper$(e)$))
$\mathcal{T}$.delete$(e)$
$e' \leftarrow \mathcal{T}$.edgeLeftOf$(v)$
**if** helper$(e')$ merge vtx **then**
    $\mathcal{D}$.insert(diag($v$, helper$(e')$))
helper$(e') \leftarrow v$
```

An Algorithm

2) Treating merge vertices

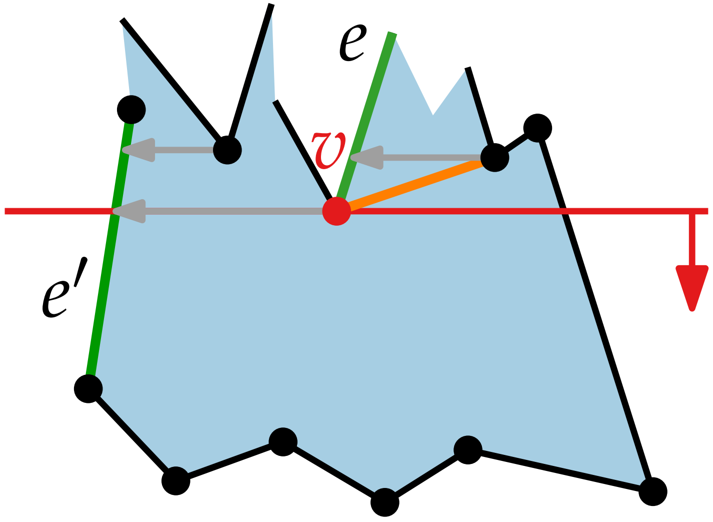


```
makeMonotone(polygon $P$)
$\mathcal{D} \leftarrow$ DCEL($V(P), E(P)$)
$\mathcal{Q} \leftarrow$ priority queue on $V(P)$
$\mathcal{T} \leftarrow$ empty bin. search tree
while $\mathcal{Q} \neq \emptyset$ do
    $v \leftarrow \mathcal{Q}$.extractMax()
    type $\leftarrow$ type of vertex $v$
    handleVertex$_{\text{type}}(v)$
return DCEL $\mathcal{D}$
```

```
handleVertex$_{\text{merge}}$(vertex $v$)
$e \leftarrow$ edge following $v$ ccw
if helper($e$) merge vtx then
    $\mathcal{D}$.insert(diag($v$, helper($e$)))
$\mathcal{T}$.delete($e$)
$e' \leftarrow \mathcal{T}$.edgeLeftOf($v$)
if helper($e'$) merge vtx then
    $\mathcal{D}$.insert(diag($v$, helper($e'$)))
helper($e'$) $\leftarrow v$
```

An Algorithm

2) Treating merge vertices

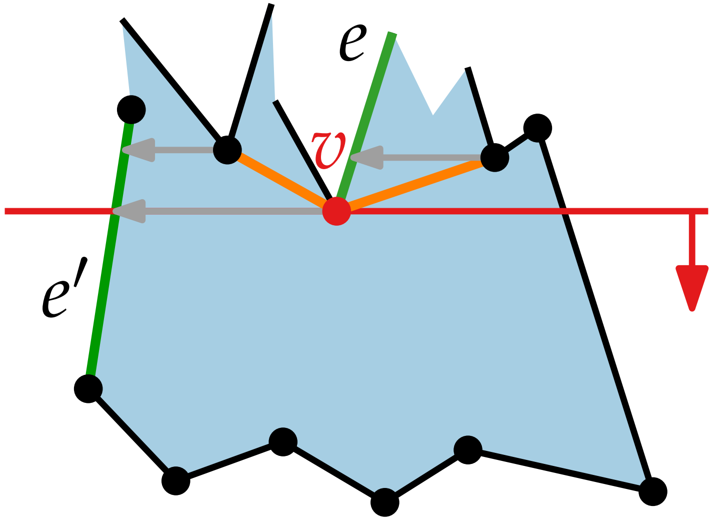


```
makeMonotone(polygon $P$)
$\mathcal{D} \leftarrow$ DCEL$(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue on $V(P)$
$\mathcal{T} \leftarrow$ empty bin. search tree
while $\mathcal{Q} \neq \emptyset$ do
    $v \leftarrow \mathcal{Q}$.extractMax()
    type $\leftarrow$ type of vertex $v$
    handleVertex$_{\text{type}}(v)$
return DCEL $\mathcal{D}$
```

```
handleVertex$_{\text{merge}}$(vertex $v$)
$e \leftarrow$ edge following $v$ ccw
if helper$(e)$ merge vtx then
    $\mathcal{D}$.insert(diag$(v$, helper$(e))$)
$\mathcal{T}$.delete$(e)$
$e' \leftarrow \mathcal{T}$.edgeLeftOf$(v)$
if helper$(e')$ merge vtx then
    $\mathcal{D}$.insert(diag$(v$, helper$(e'))$)
helper$(e') \leftarrow v$
```

An Algorithm

2) Treating merge vertices



```
makeMonotone(polygon $P$)
$\mathcal{D} \leftarrow$ DCEL$(V(P), E(P))$
$\mathcal{Q} \leftarrow$ priority queue on $V(P)$
$\mathcal{T} \leftarrow$ empty bin. search tree
while $\mathcal{Q} \neq \emptyset$ do
    $v \leftarrow \mathcal{Q}$.extractMax()
    type $\leftarrow$ type of vertex $v$
    handleVertex$_{\text{type}}(v)$
return DCEL $\mathcal{D}$
```

```
handleVertex$_{\text{merge}}$(vertex $v$)
$e \leftarrow$ edge following $v$ ccw
if helper$(e)$ merge vtx then
    $\mathcal{D}$.insert(diag($v$, helper$(e)$))
$\mathcal{T}$.delete$(e)$
$e' \leftarrow \mathcal{T}$.edgeLeftOf$(v)$
if helper$(e')$ merge vtx then
    $\mathcal{D}$.insert(diag($v$, helper$(e')$))
helper$(e') \leftarrow v$
```

Analysis

Lemma. makeMonotone() adds a set of non-intersecting diagonals to P such that P is partitioned into y-monotone subpolygons.

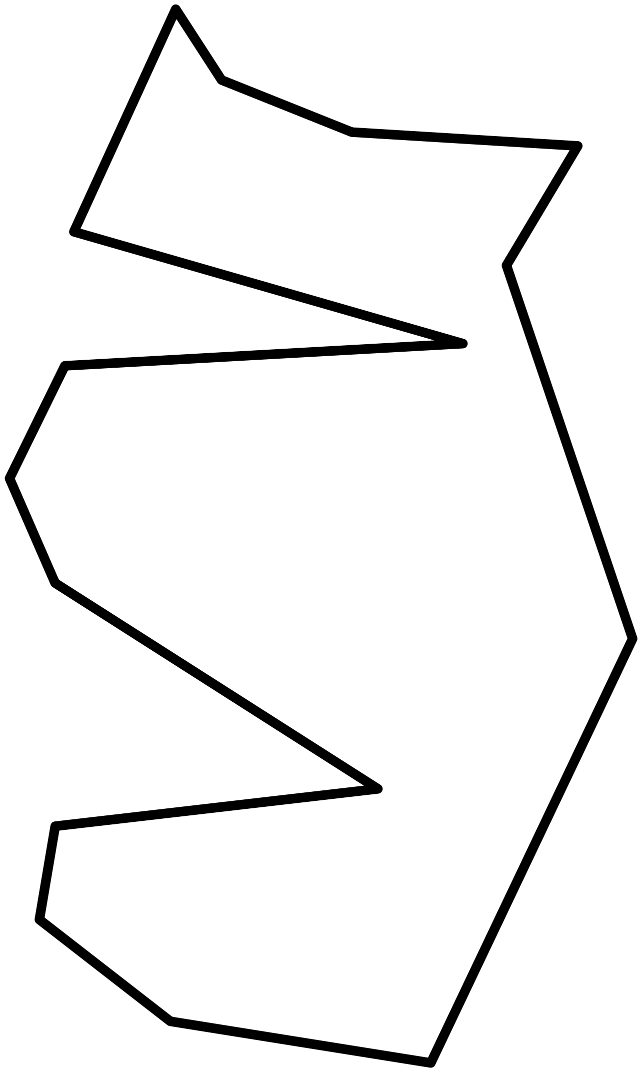
Analysis

Lemma. makeMonotone() adds a set of non-intersecting diagonals to P such that P is partitioned into y-monotone subpolygons.

Lemma. A simple polygon with n vertices can be subdivided into y-monotone polygons in $O(n \log n)$ time.

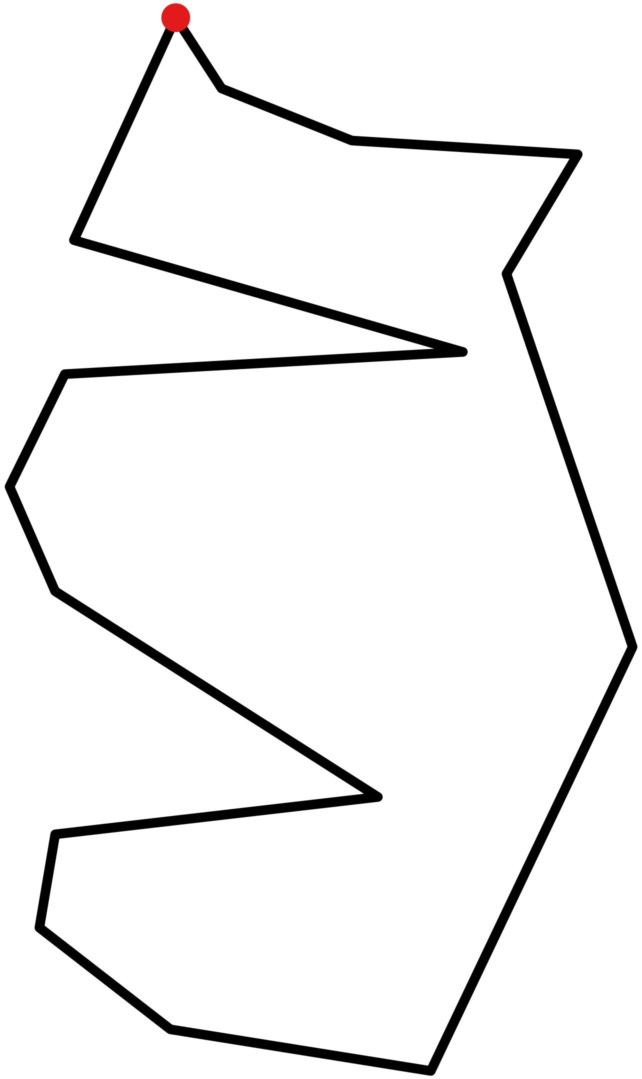
Triangulating a *y*-Monotone Polygon *P*

Approach: greedy, going from top to bottom



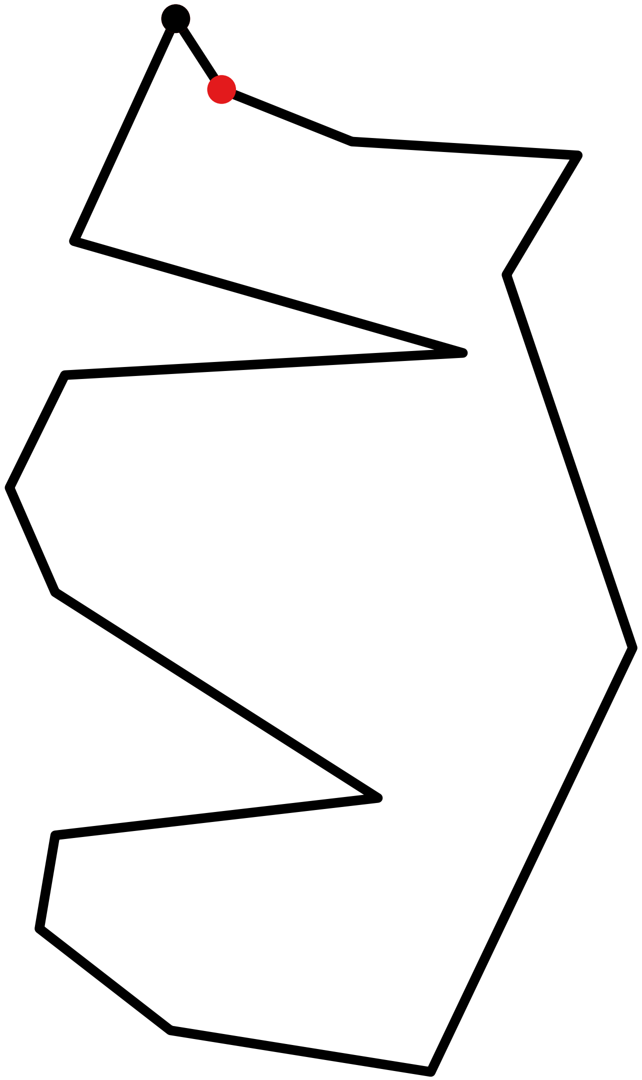
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



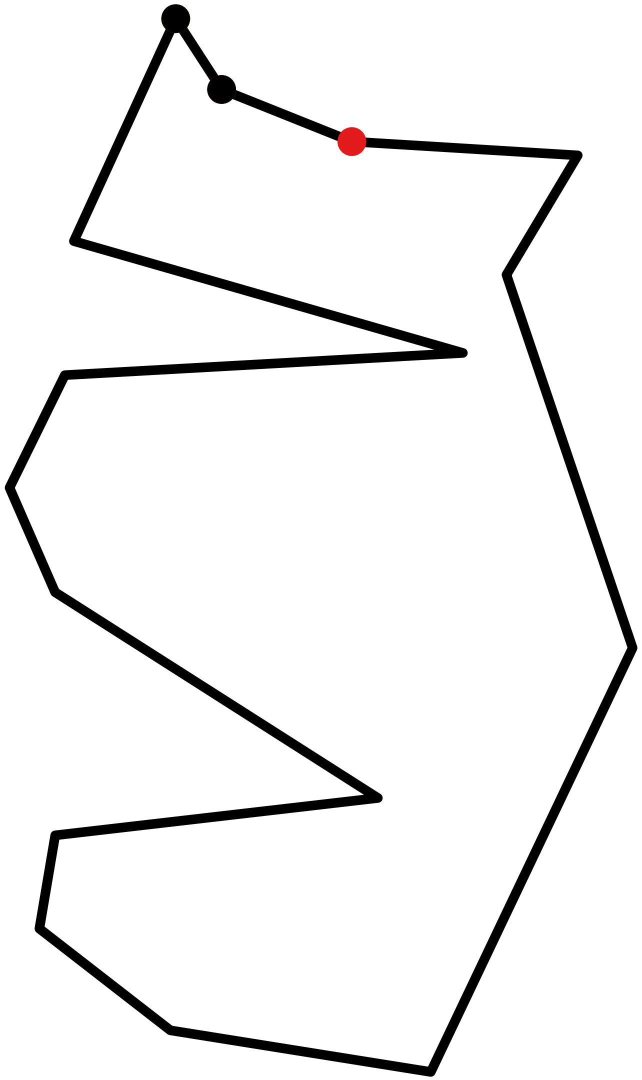
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



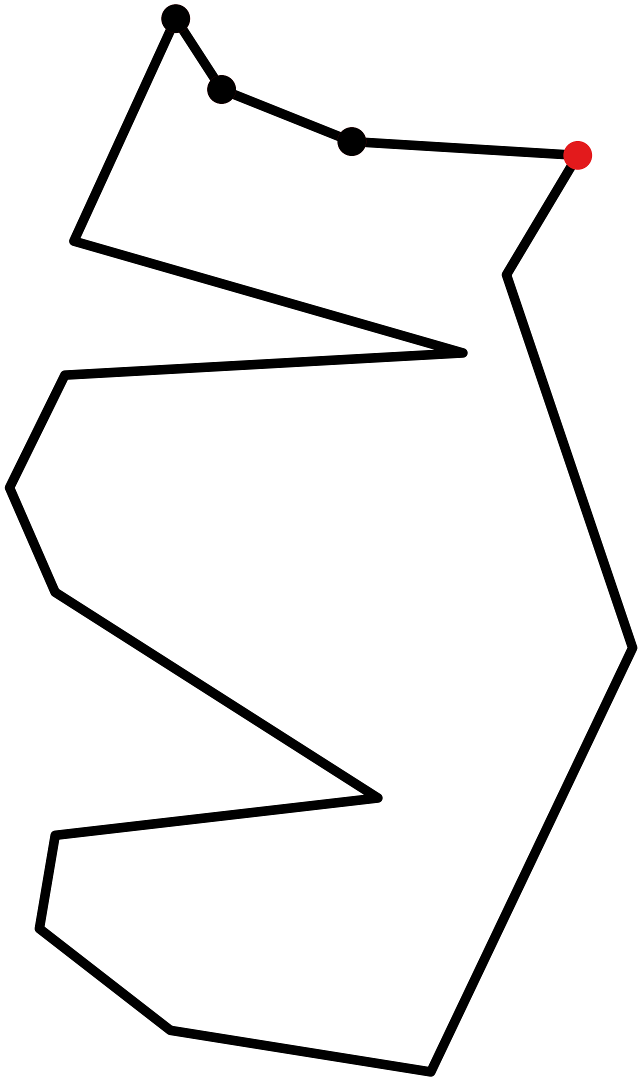
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



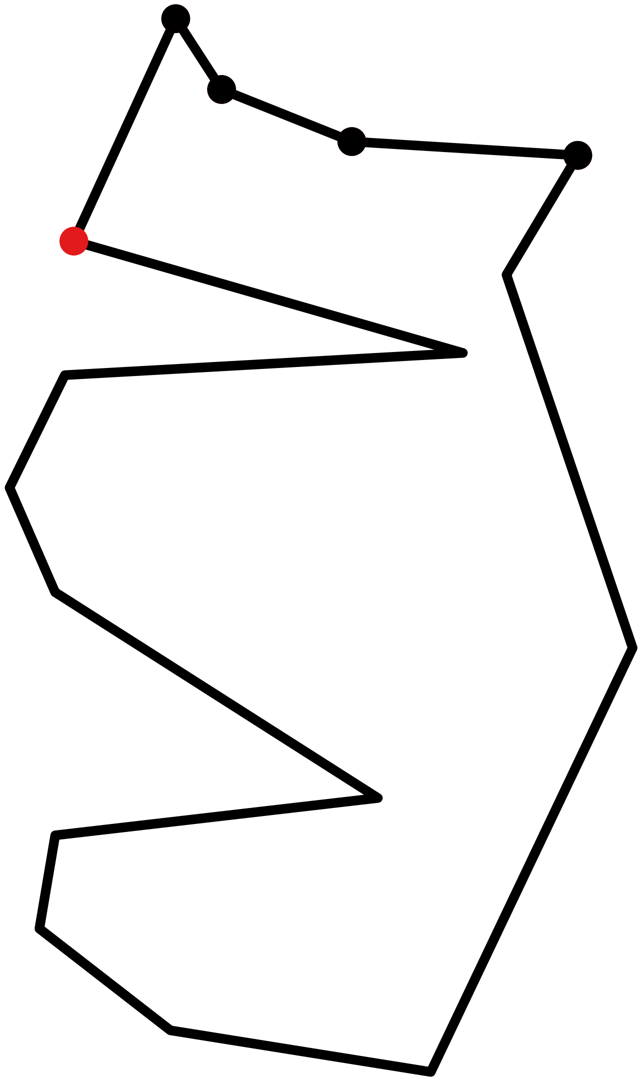
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



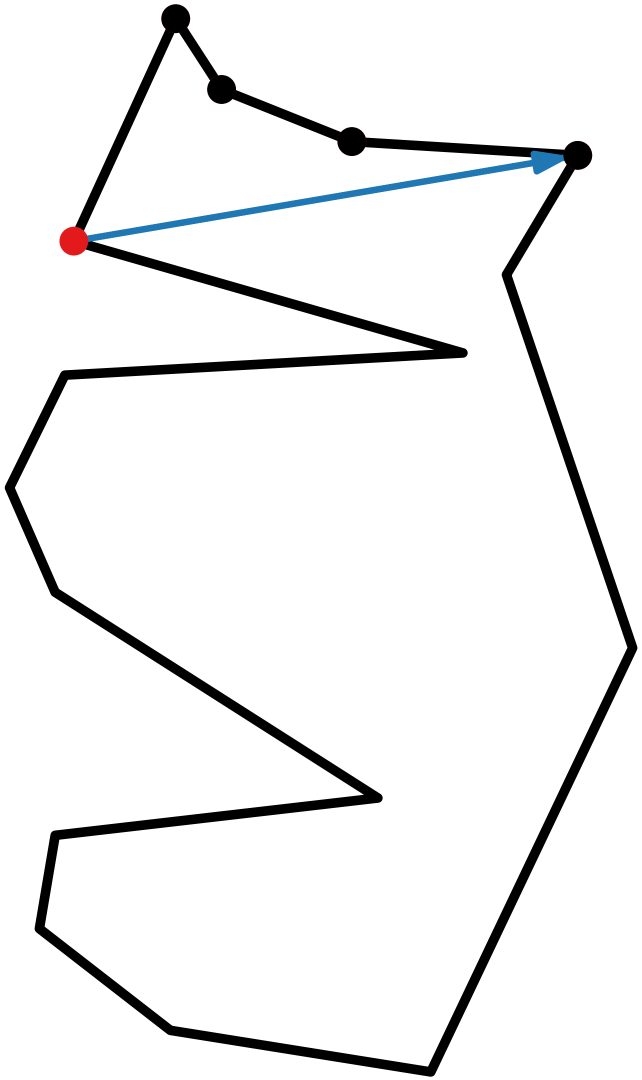
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



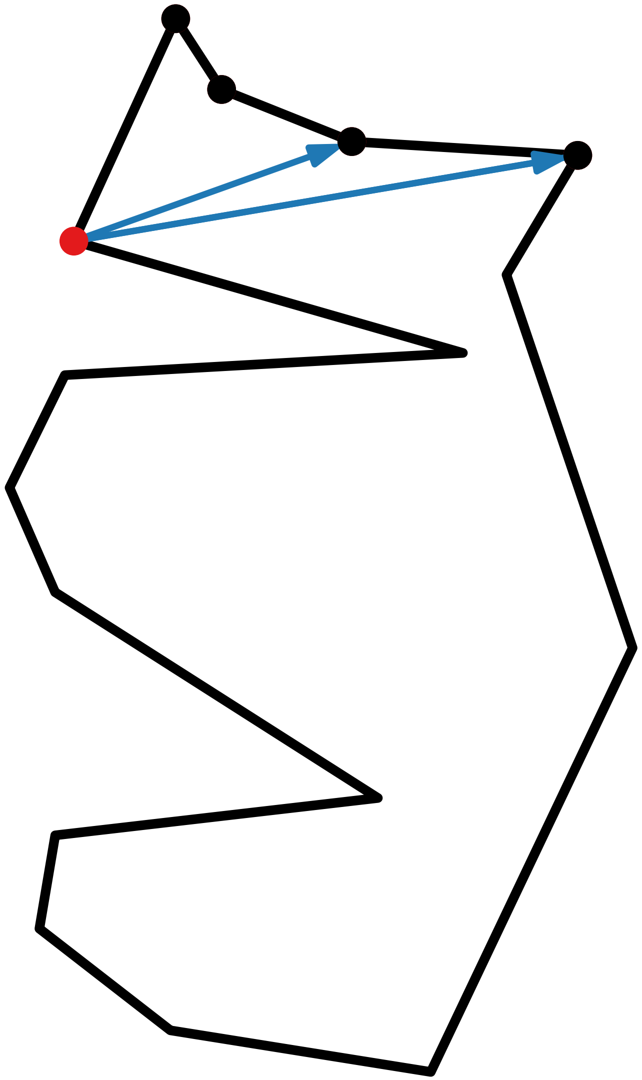
Triangulating a *y*-Monotone Polygon *P*

Approach: greedy, going from top to bottom



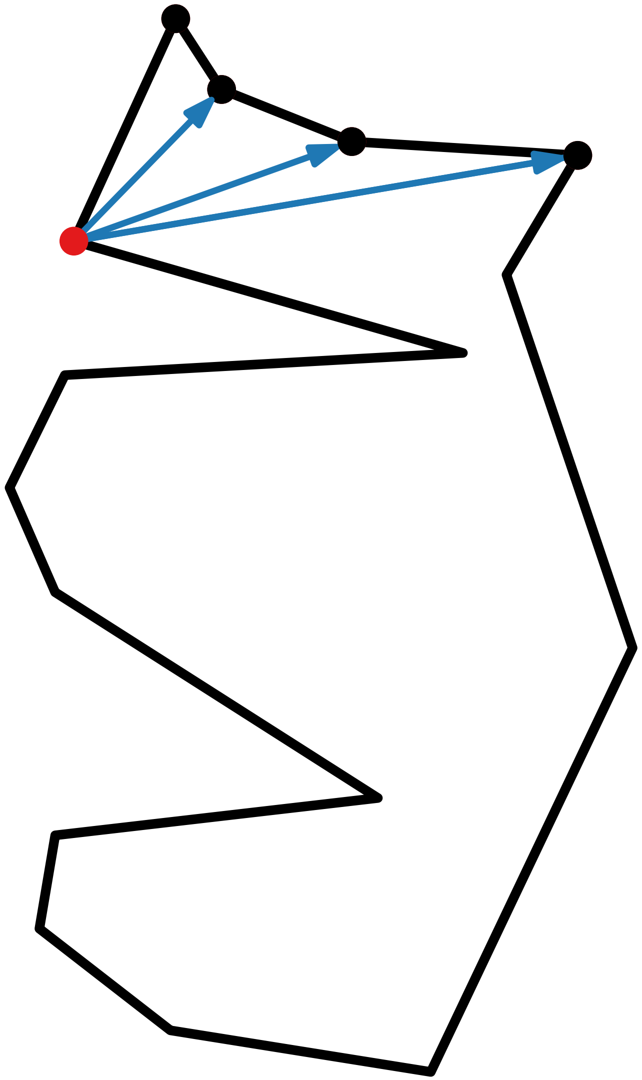
Triangulating a *y*-Monotone Polygon *P*

Approach: greedy, going from top to bottom



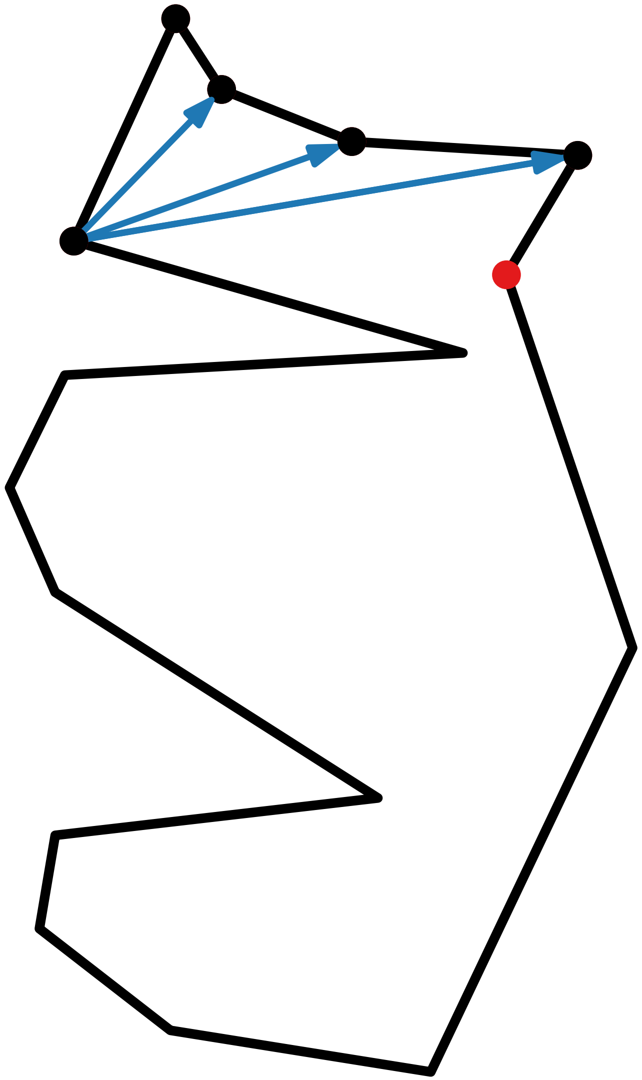
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



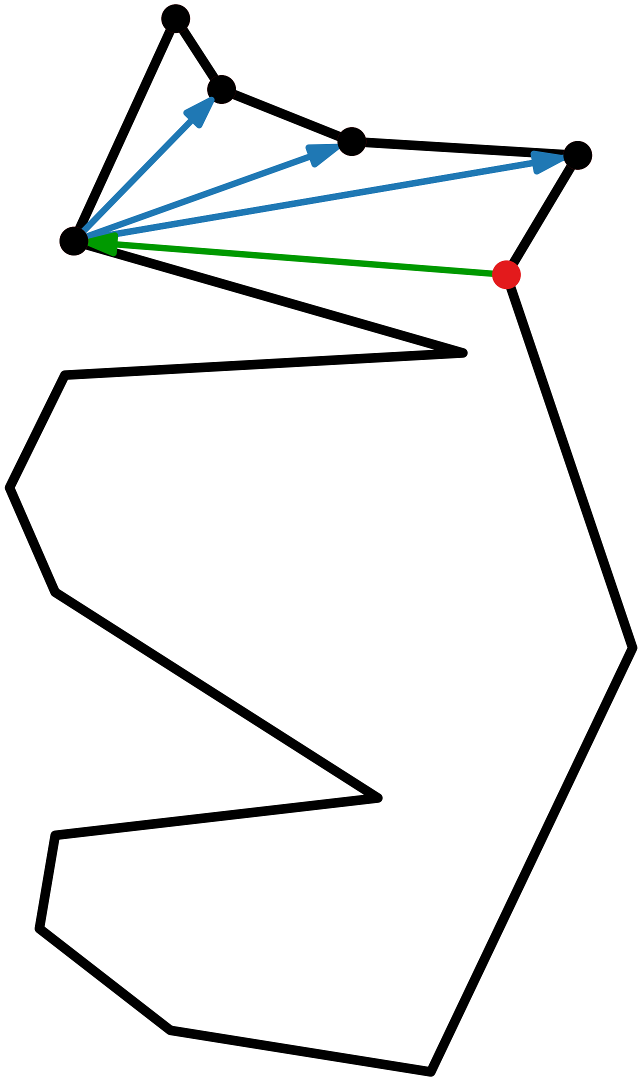
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



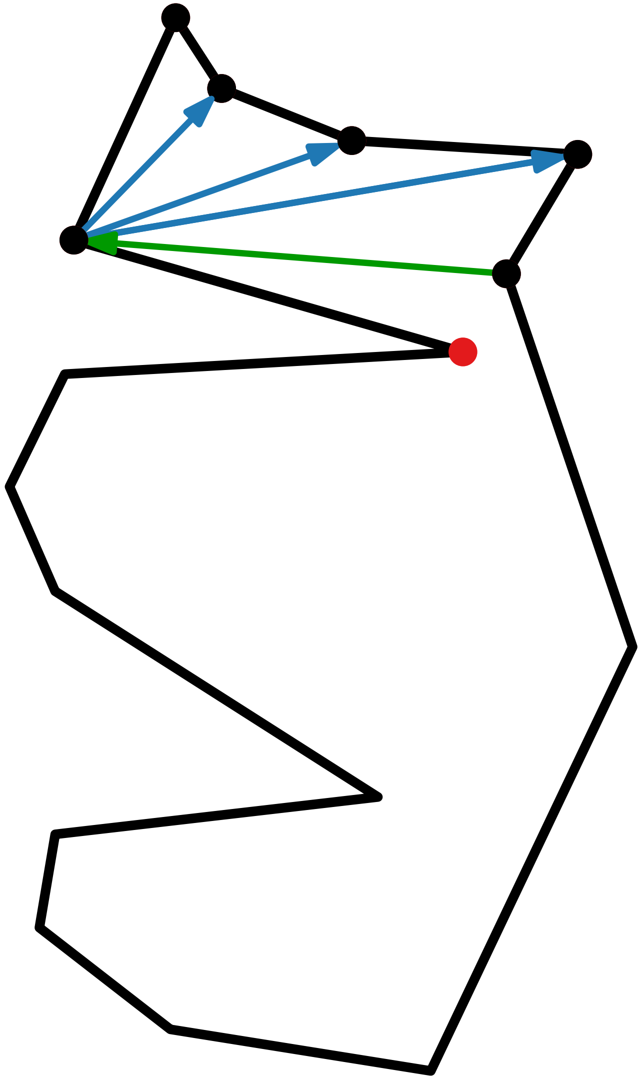
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



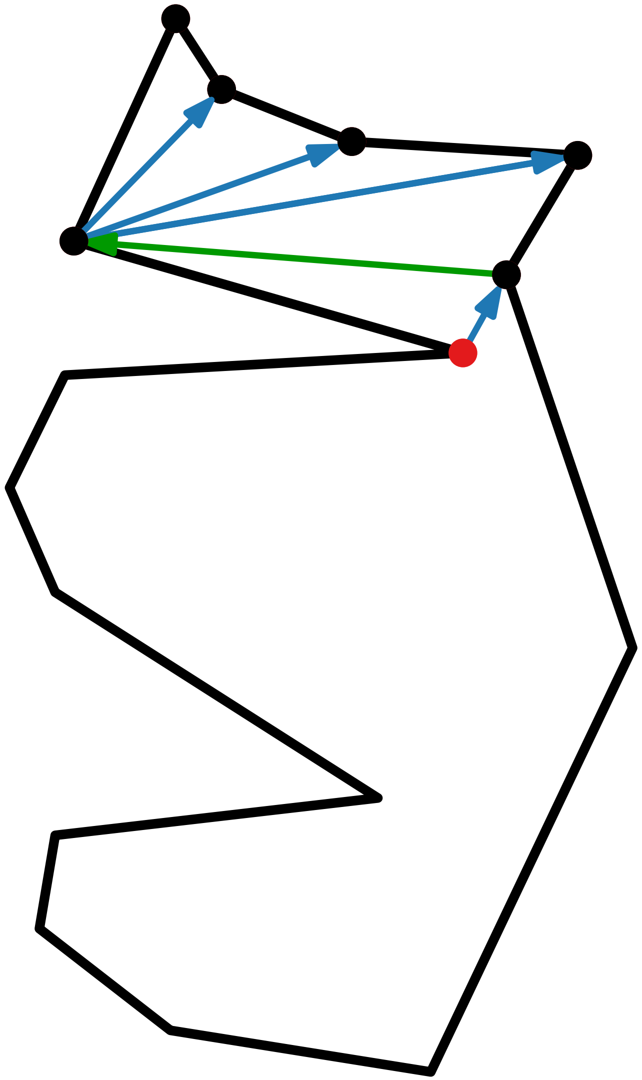
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



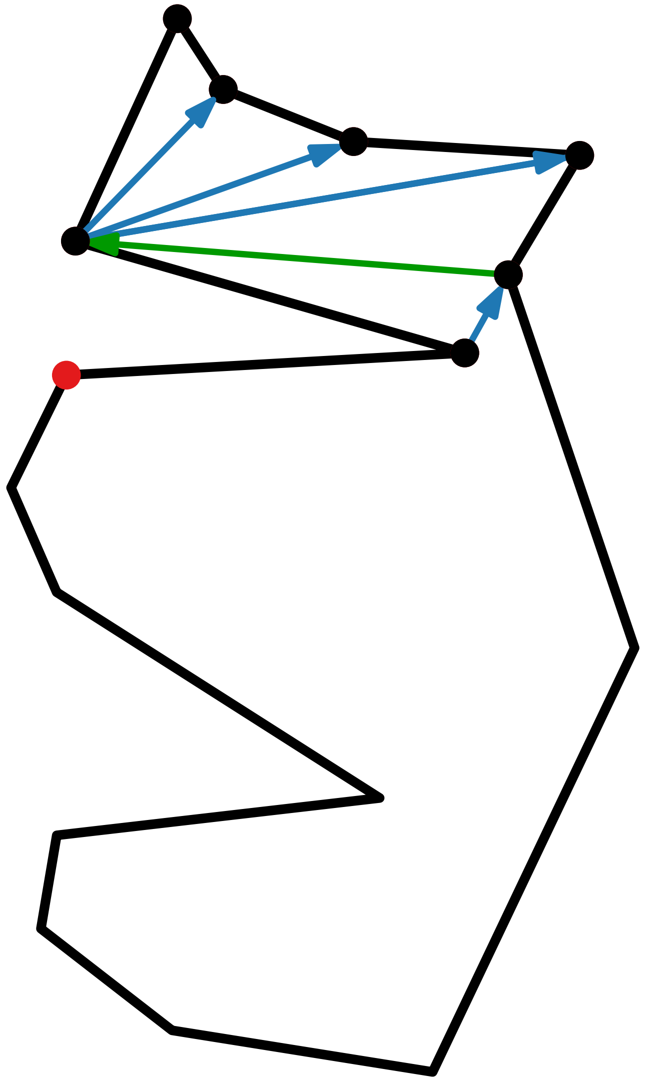
Triangulating a *y*-Monotone Polygon *P*

Approach: greedy, going from top to bottom



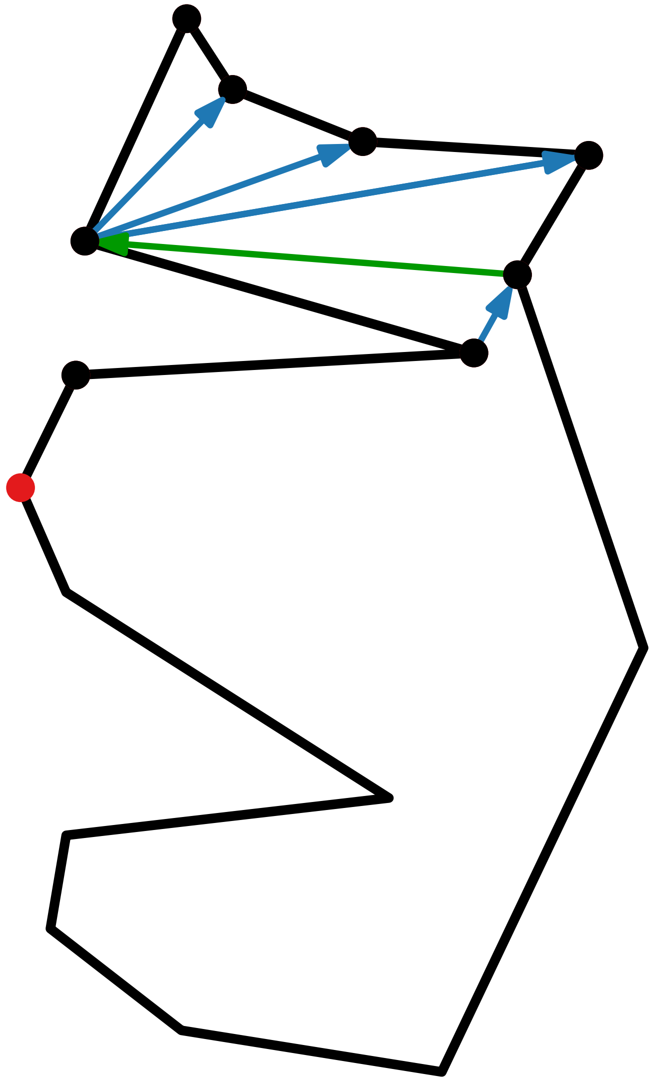
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



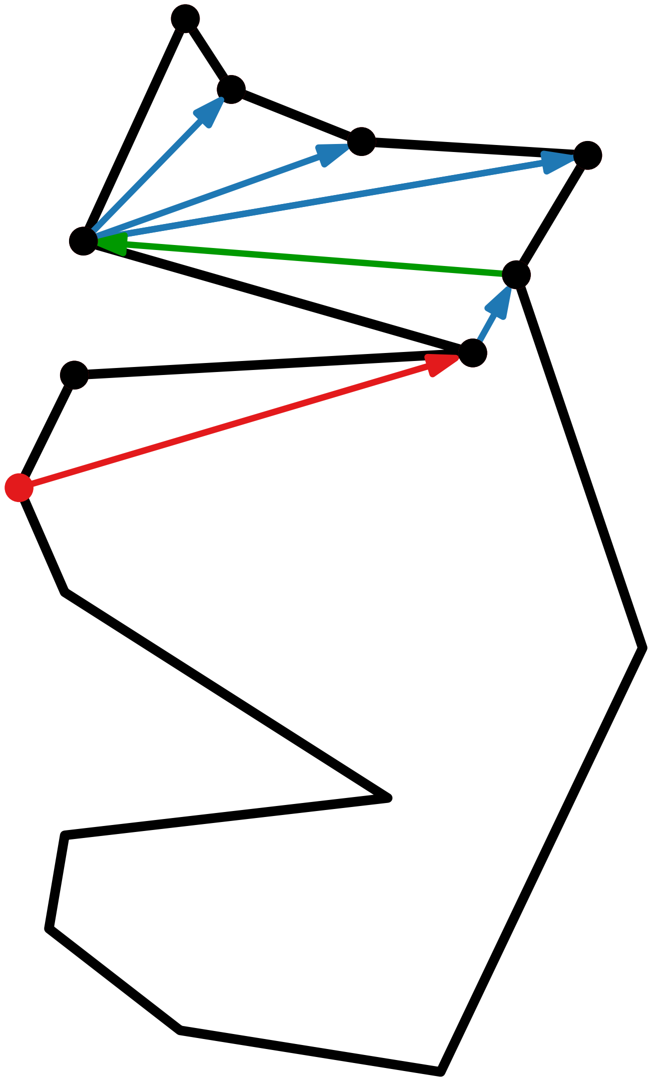
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



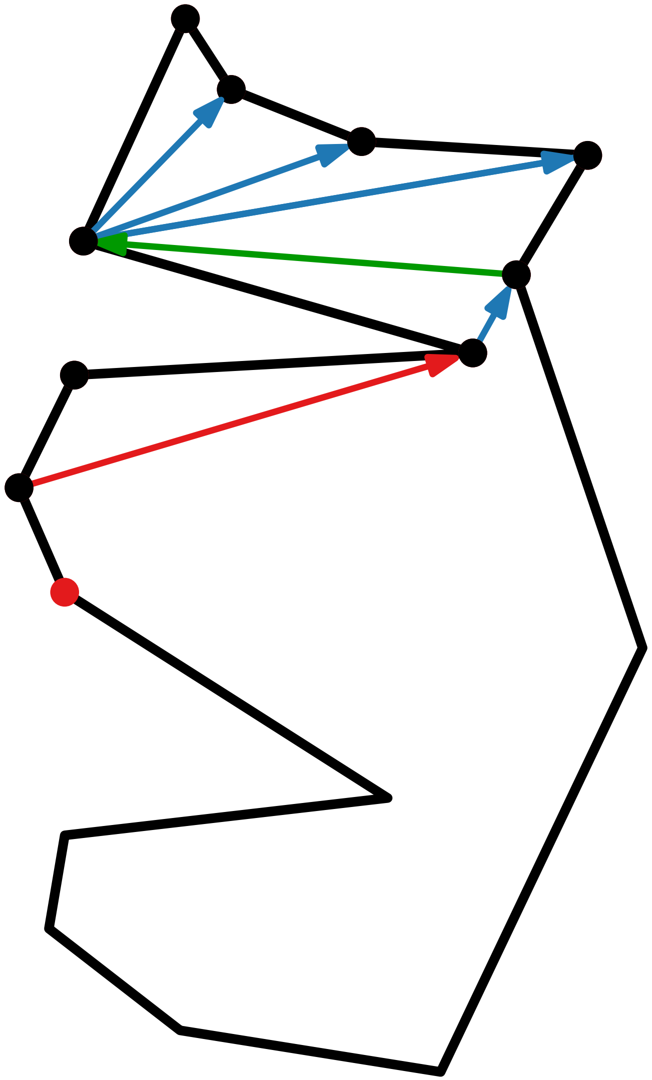
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



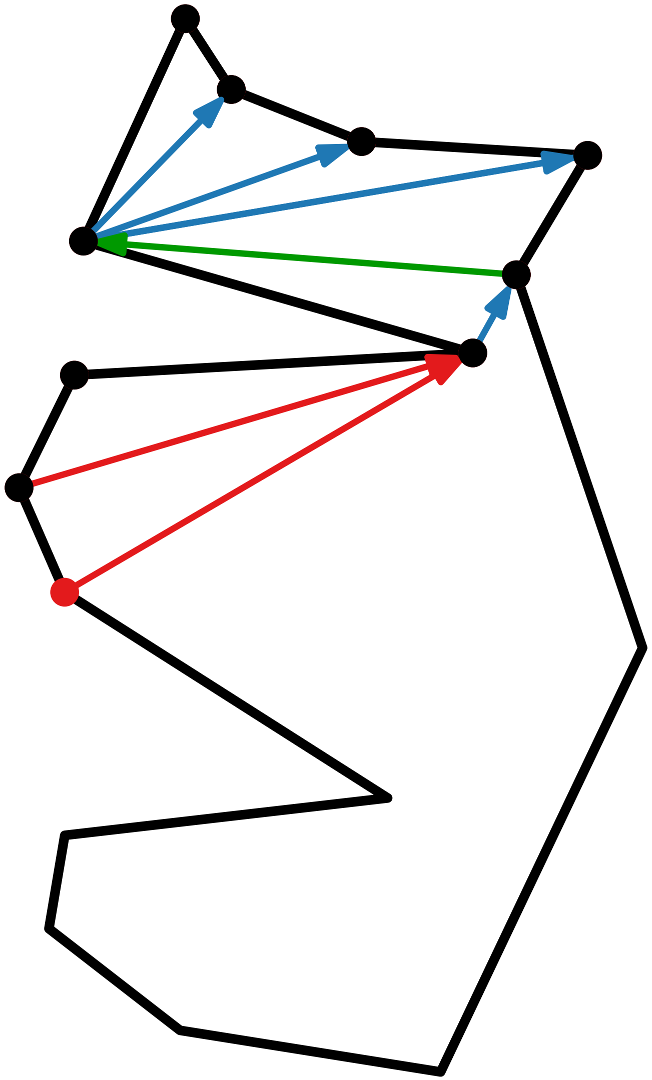
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



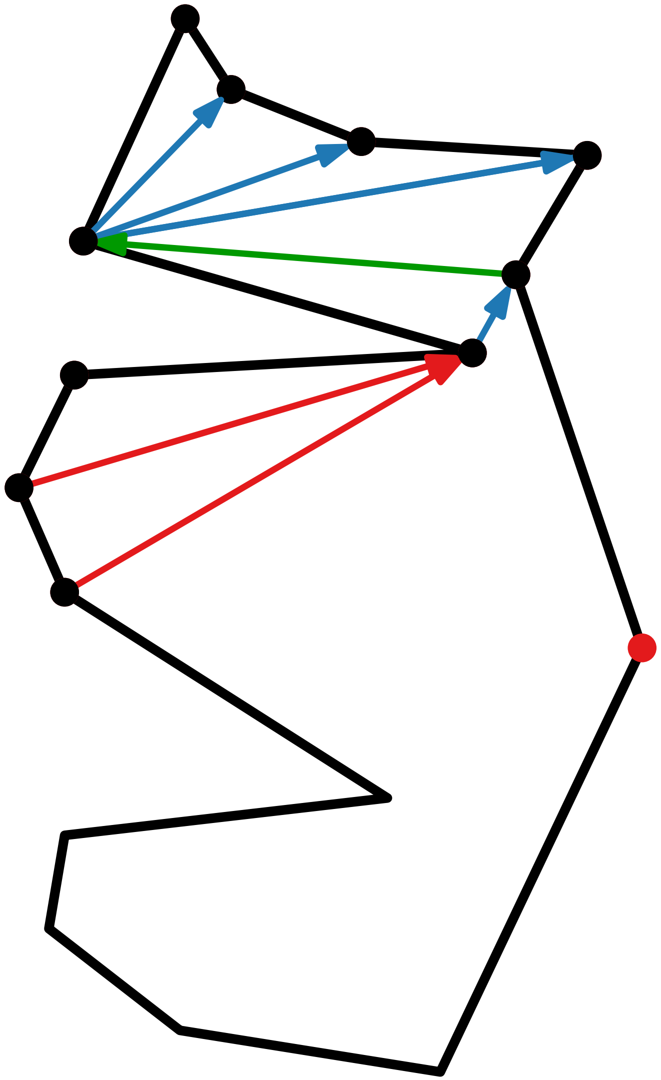
Triangulating a *y*-Monotone Polygon *P*

Approach: greedy, going from top to bottom



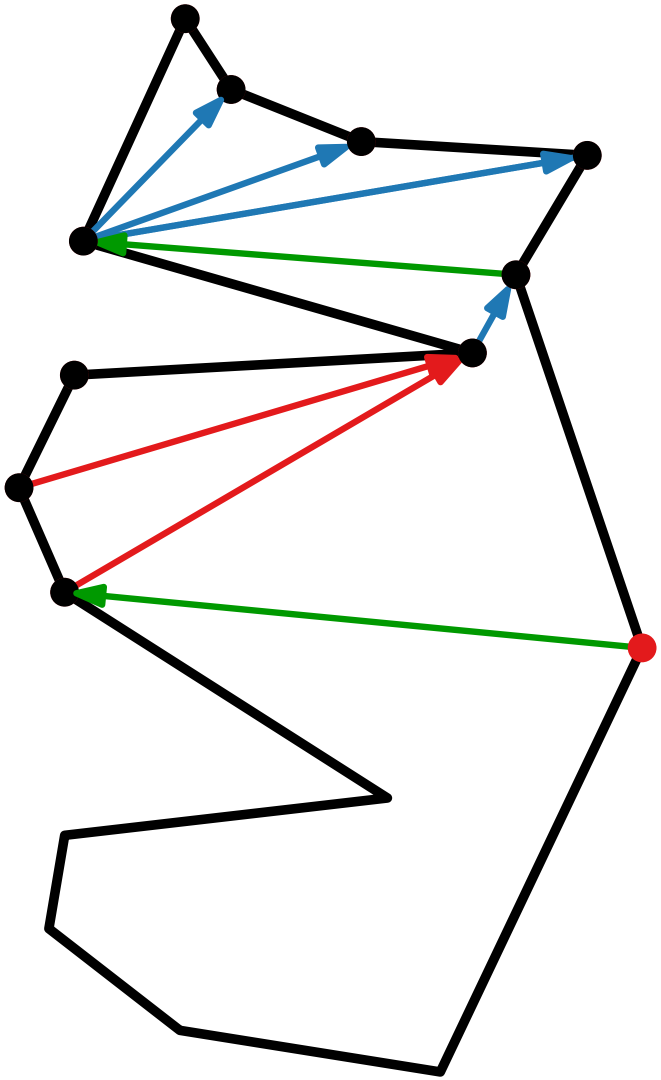
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



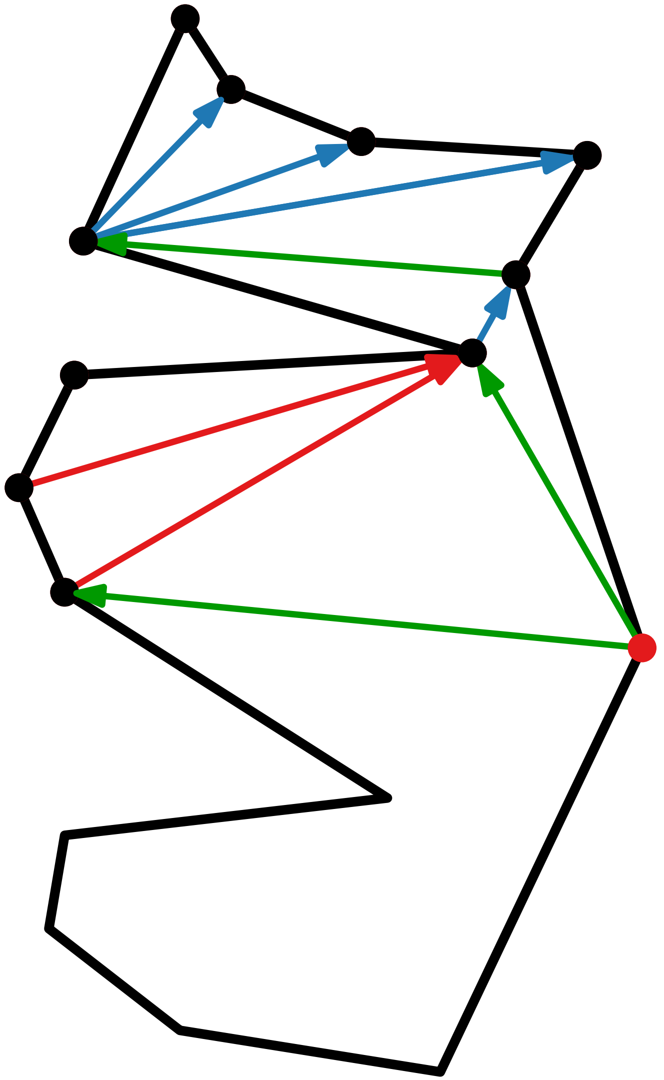
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



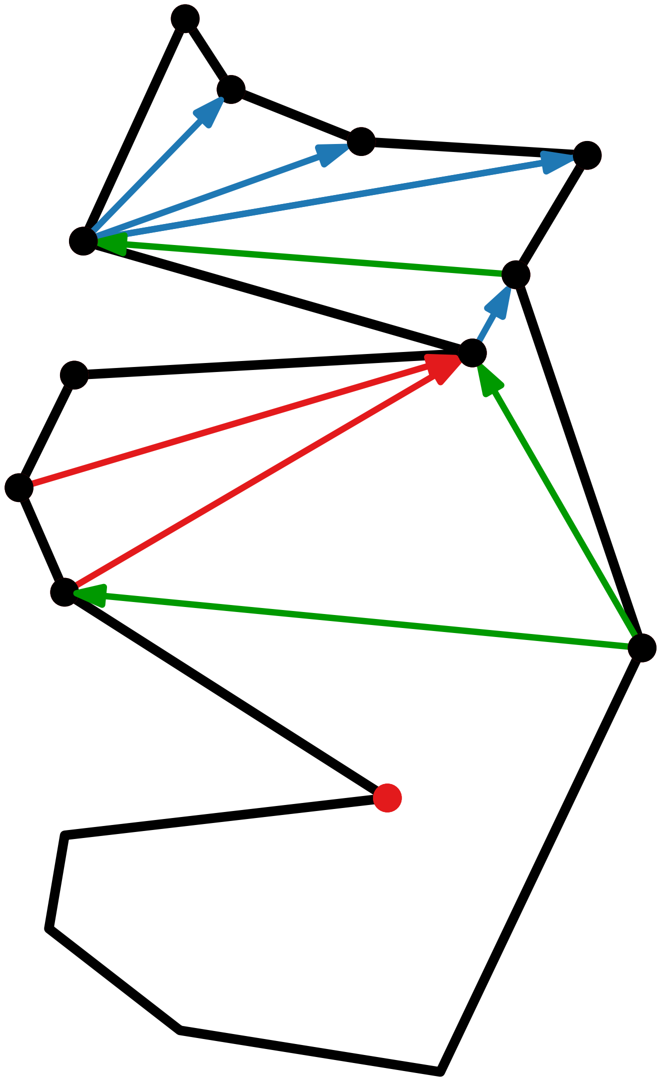
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



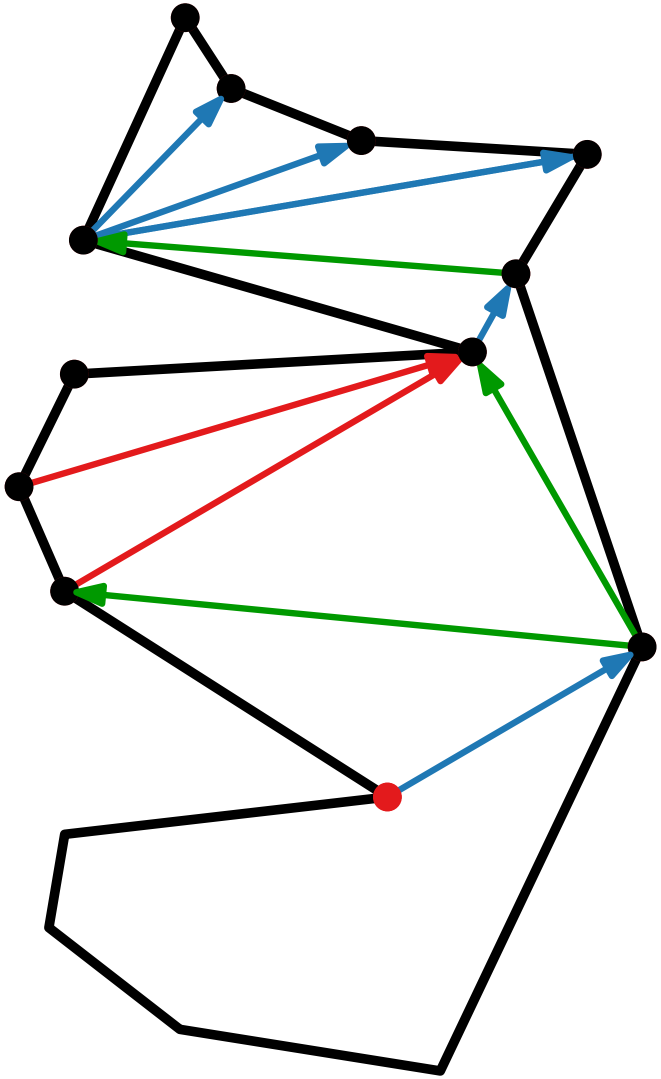
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



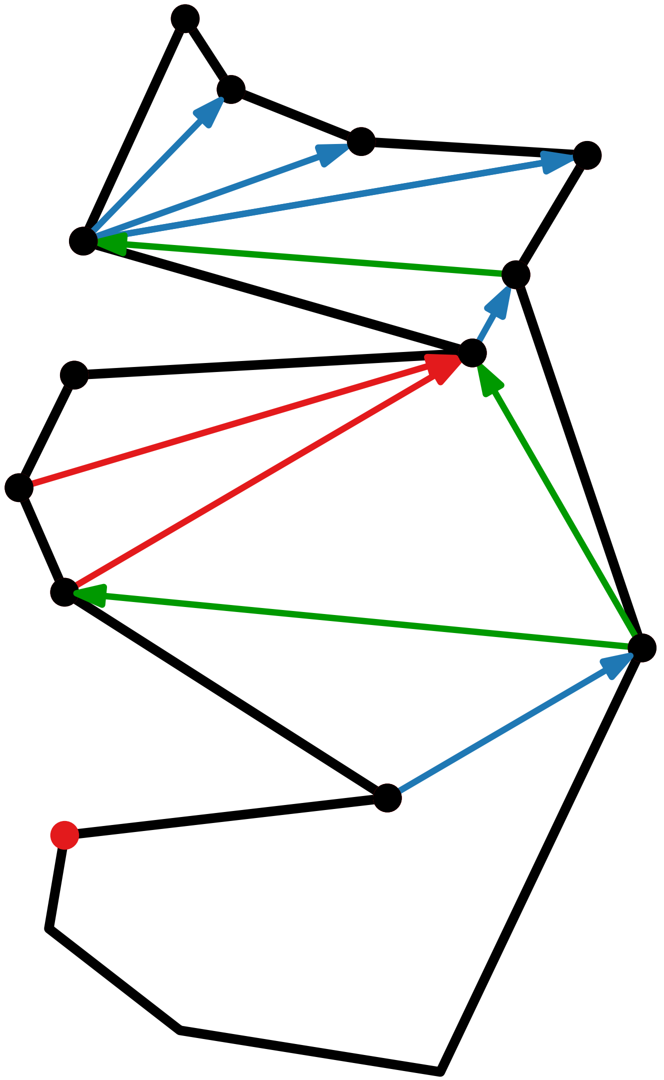
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



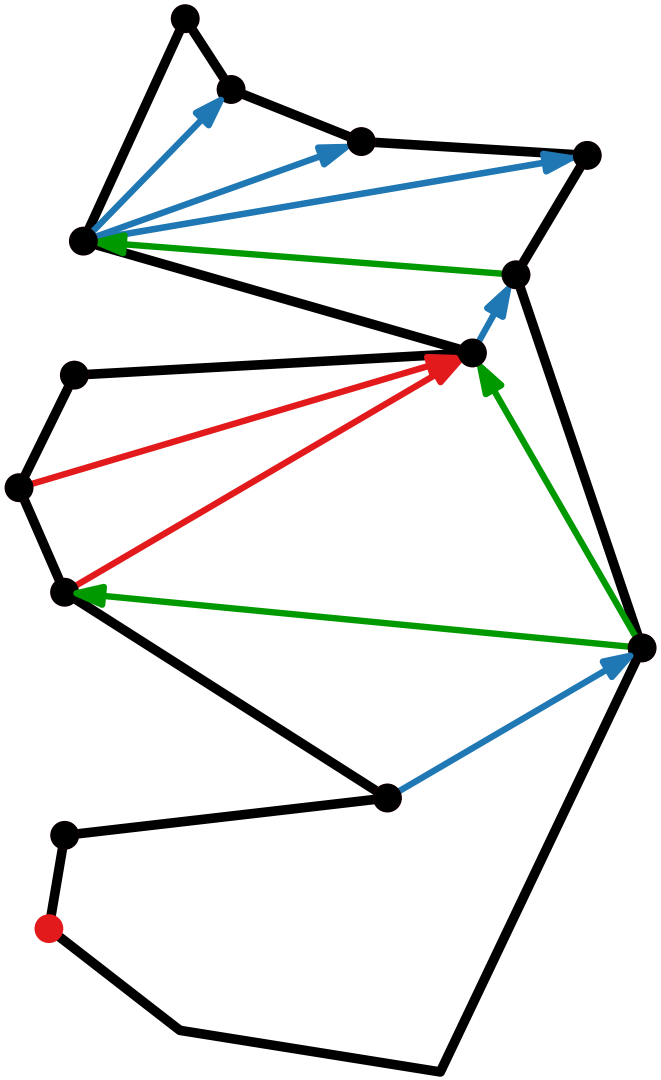
Triangulating a *y*-Monotone Polygon P

Approach: greedy, going from top to bottom



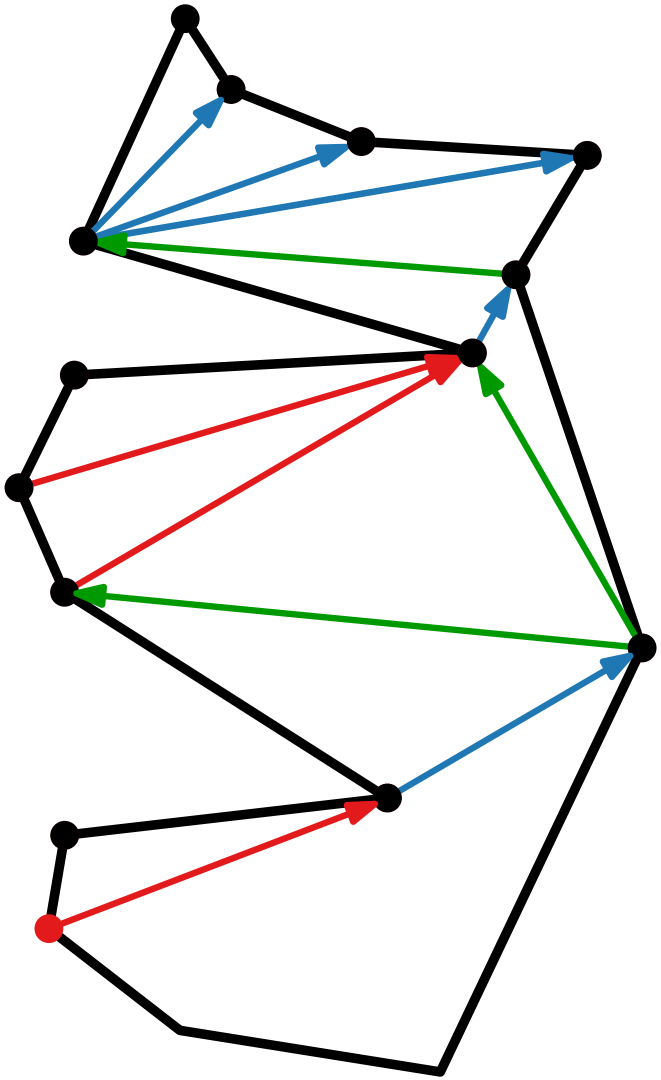
Triangulating a *y*-Monotone Polygon *P*

Approach: greedy, going from top to bottom



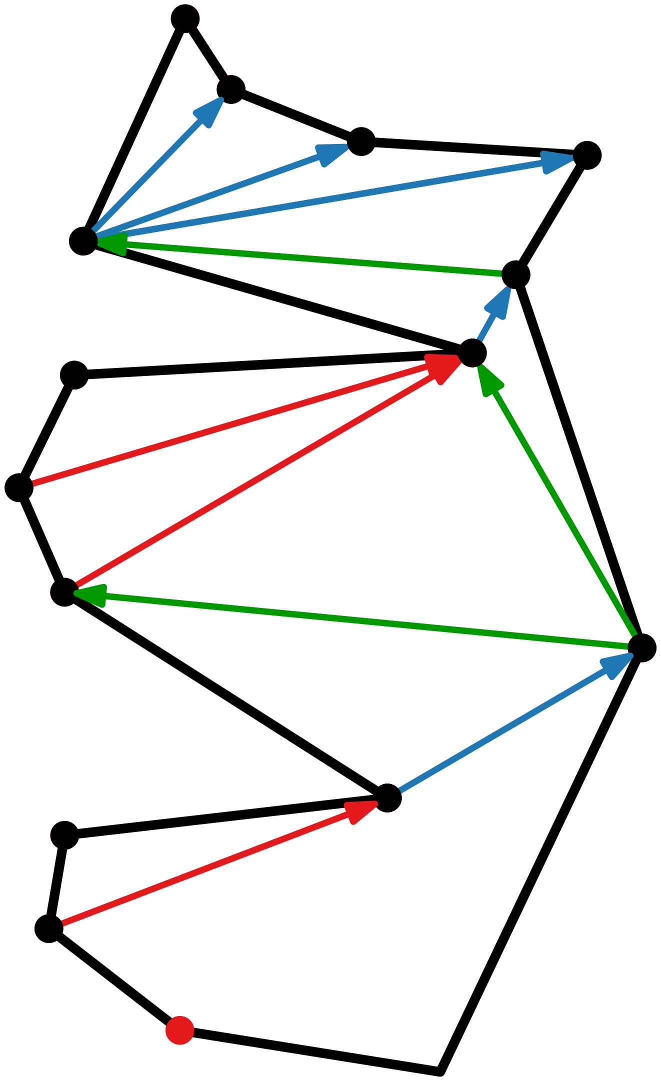
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



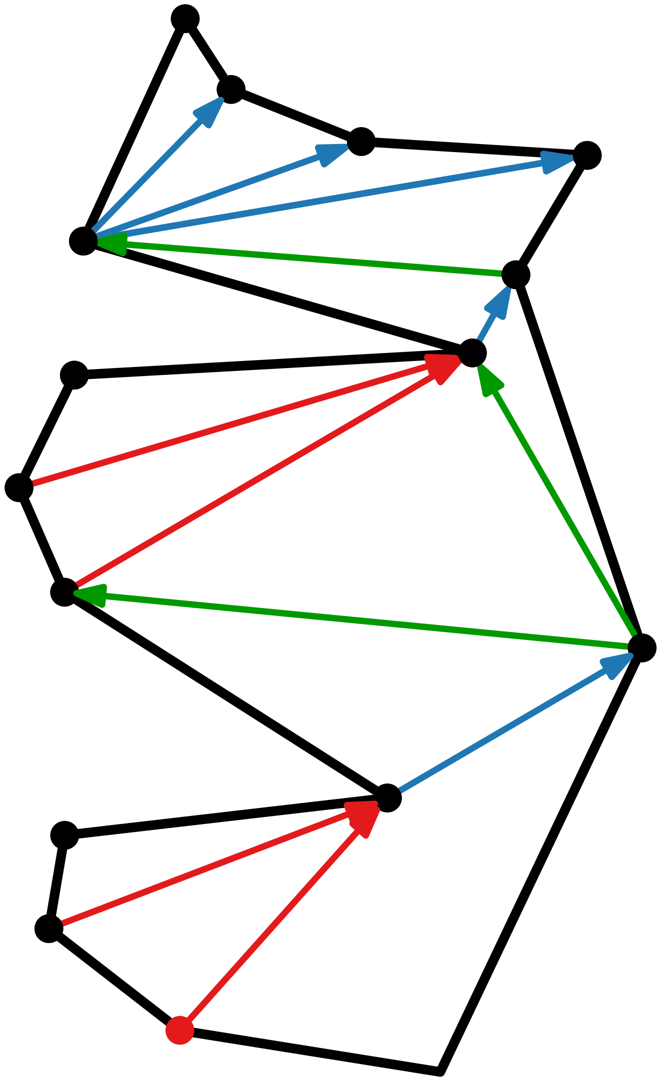
Triangulating a *y*-Monotone Polygon *P*

Approach: greedy, going from top to bottom



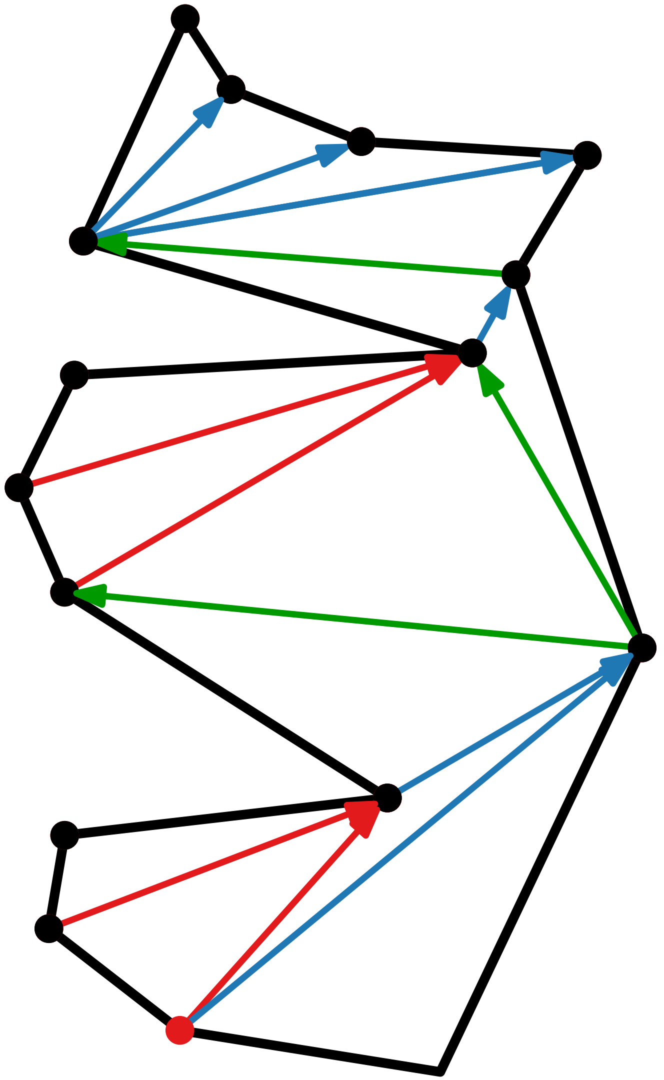
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



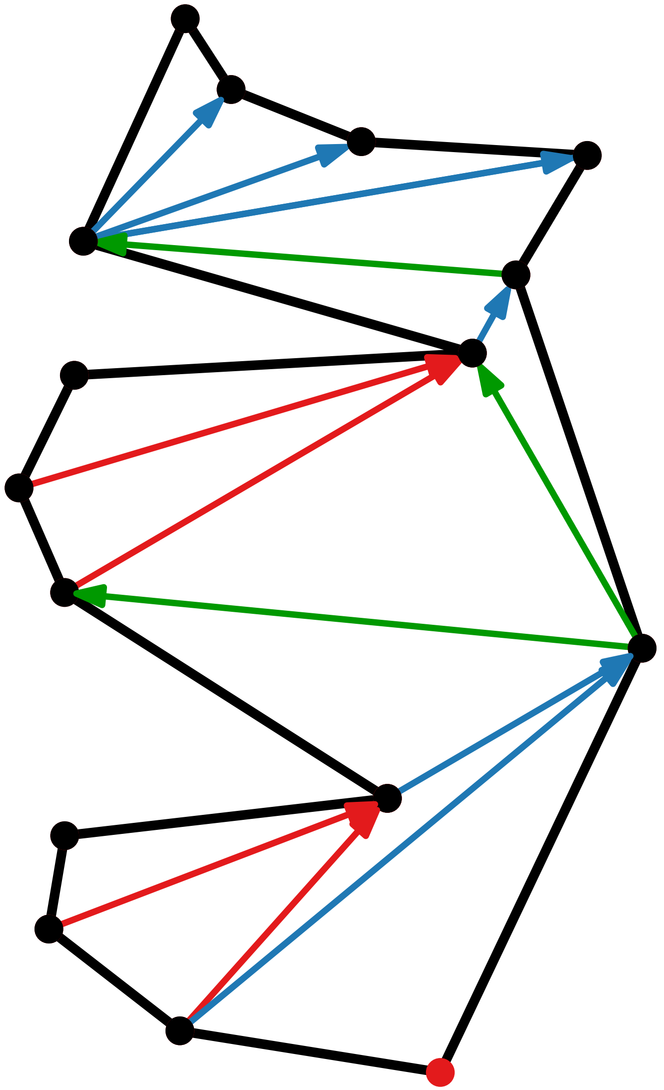
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



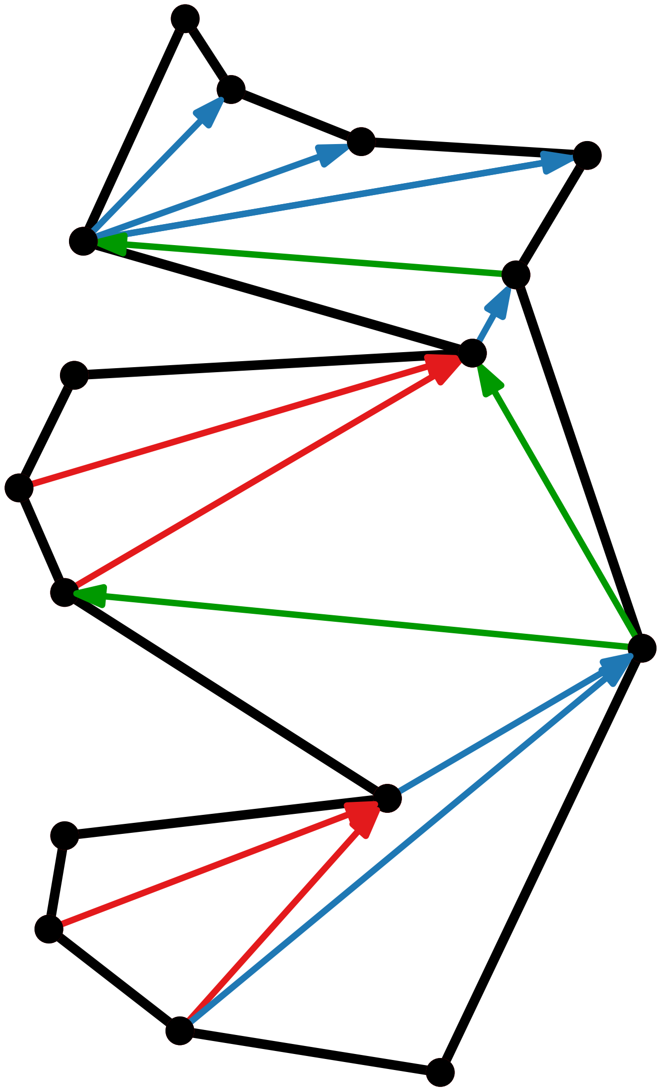
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



Triangulating a y-Monotone Polygon P

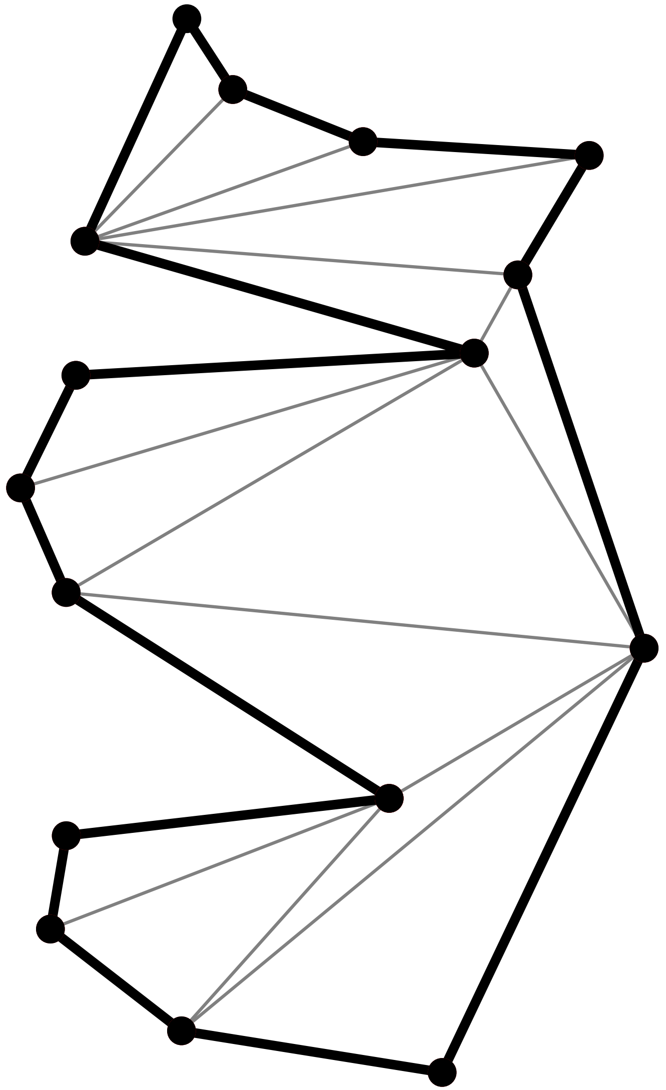
Approach: greedy, going from top to bottom



Invariant?

Triangulating a y-Monotone Polygon P

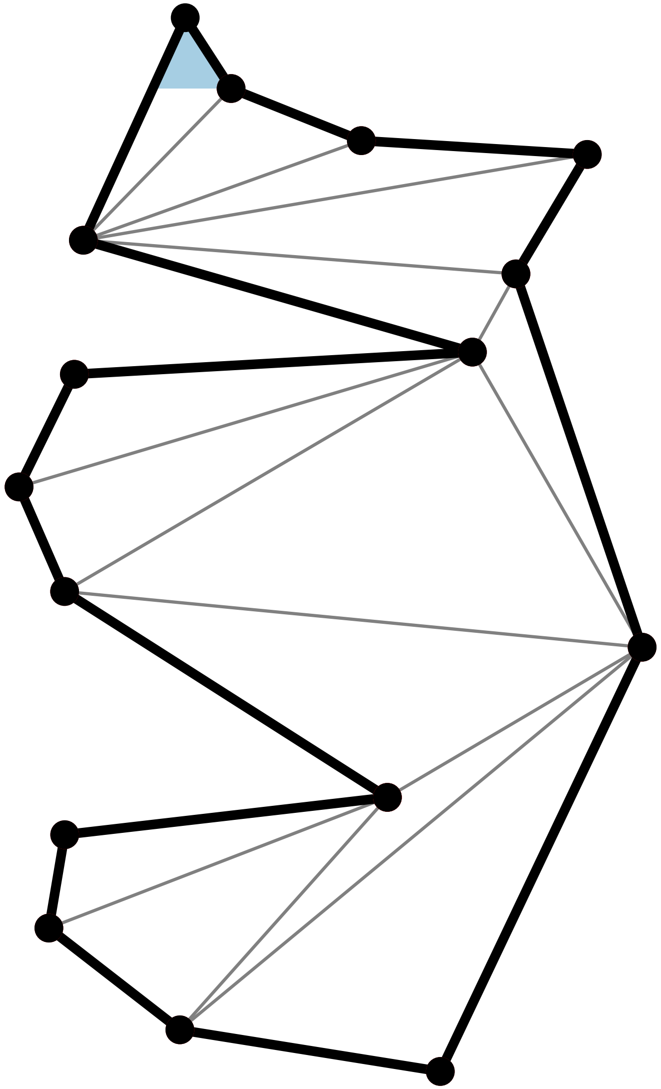
Approach: greedy, going from top to bottom



Invariant?

Triangulating a *y*-Monotone Polygon *P*

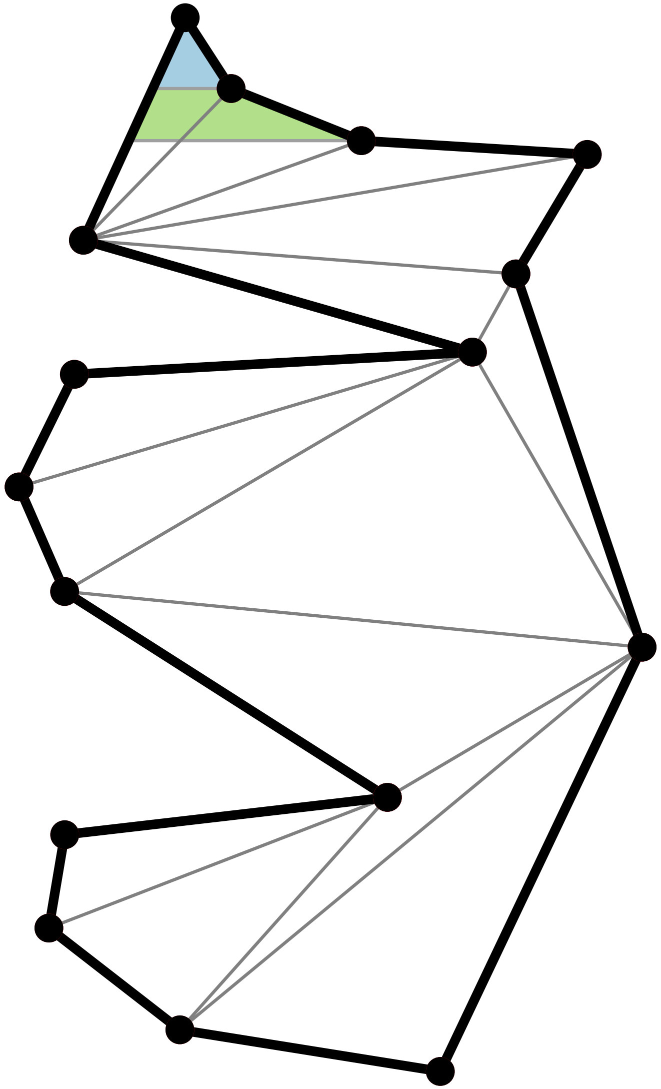
Approach: greedy, going from top to bottom



Invariant?

Triangulating a y-Monotone Polygon P

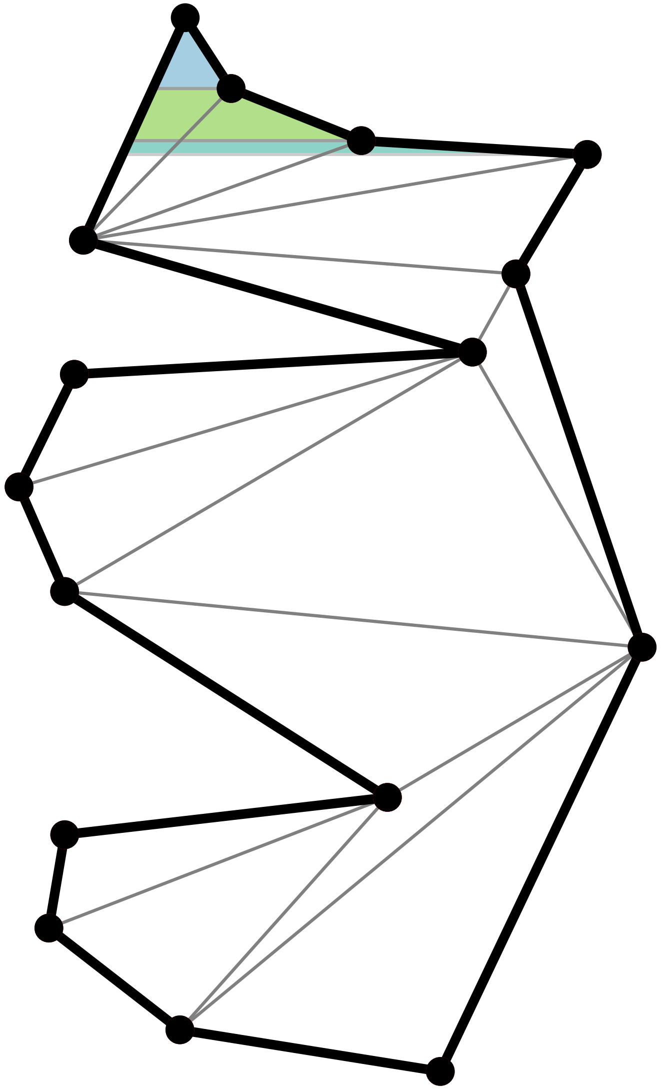
Approach: greedy, going from top to bottom



Invariant?

Triangulating a y-Monotone Polygon P

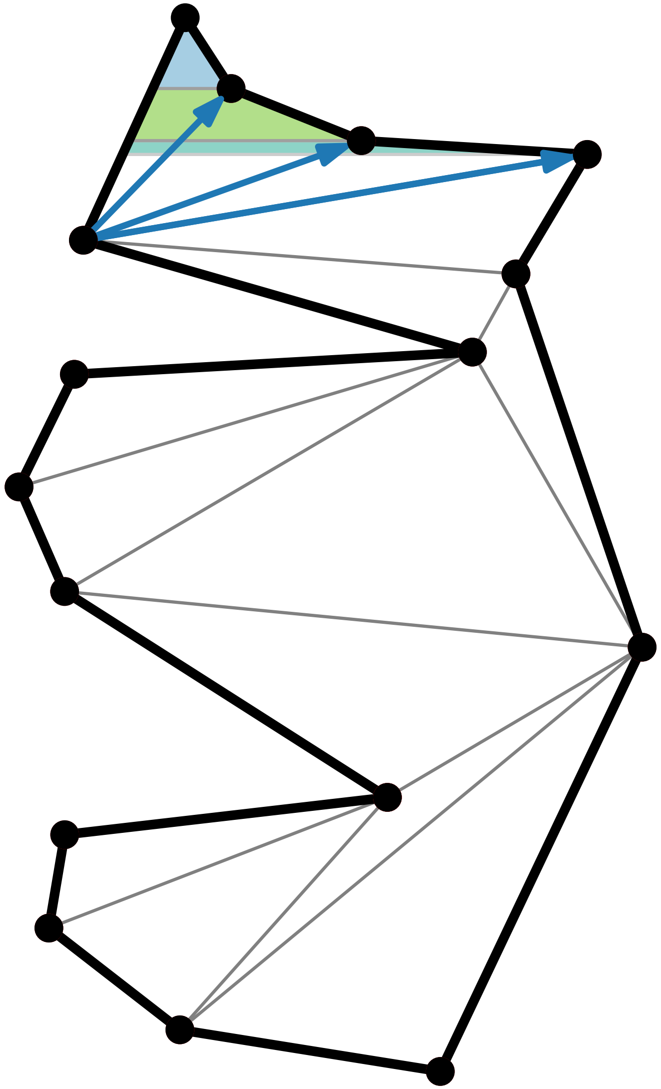
Approach: greedy, going from top to bottom



Invariant?

Triangulating a y-Monotone Polygon P

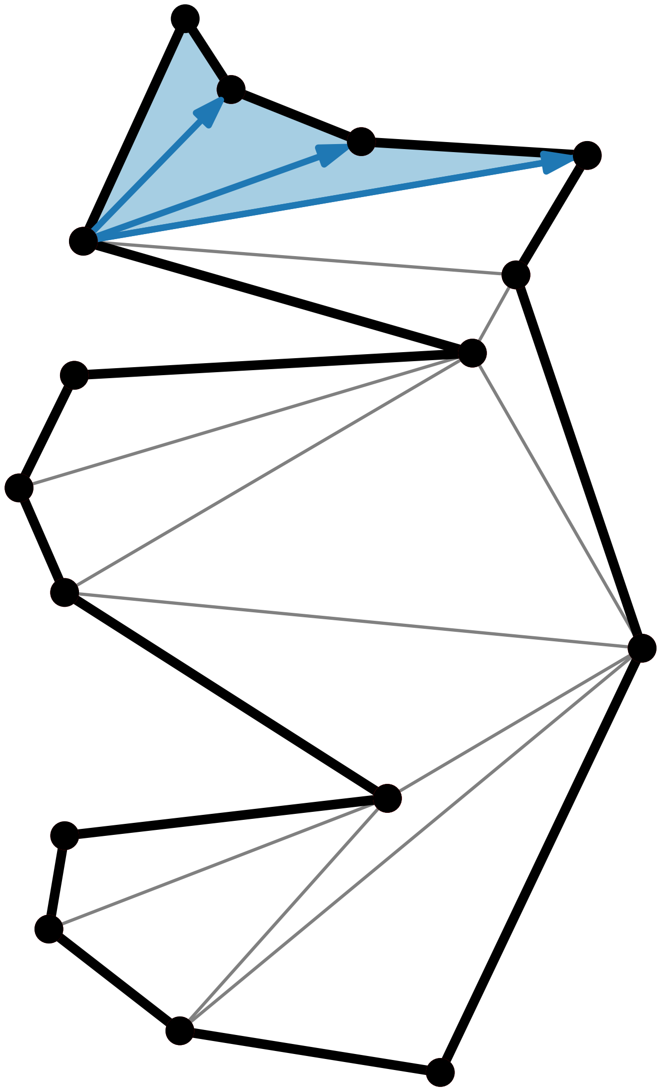
Approach: greedy, going from top to bottom



Invariant?

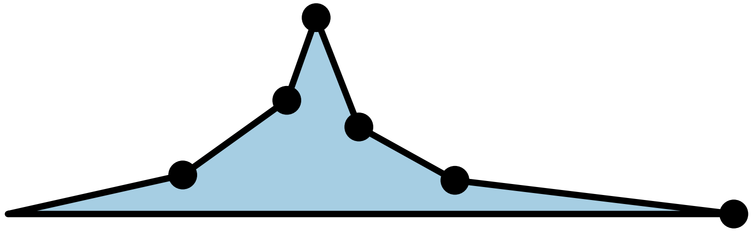
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



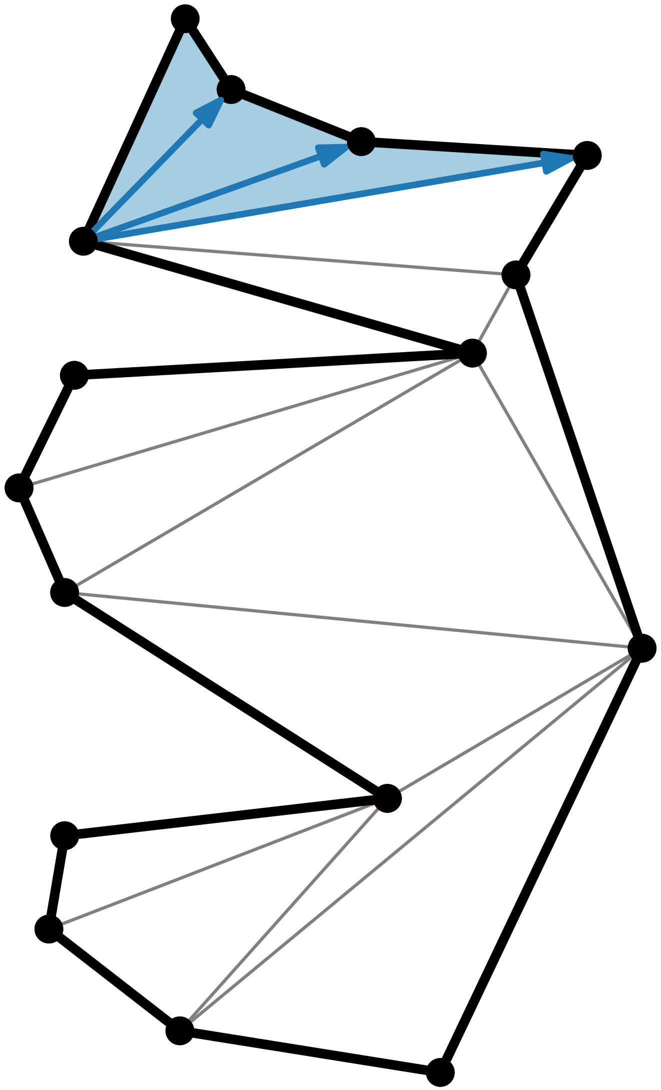
Invariant?

The part of P that we have seen but not yet triangulated is a *funnel*.



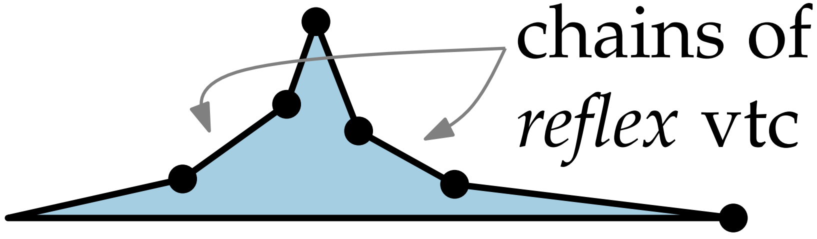
Triangulating a *y*-Monotone Polygon *P*

Approach: greedy, going from top to bottom



Invariant?

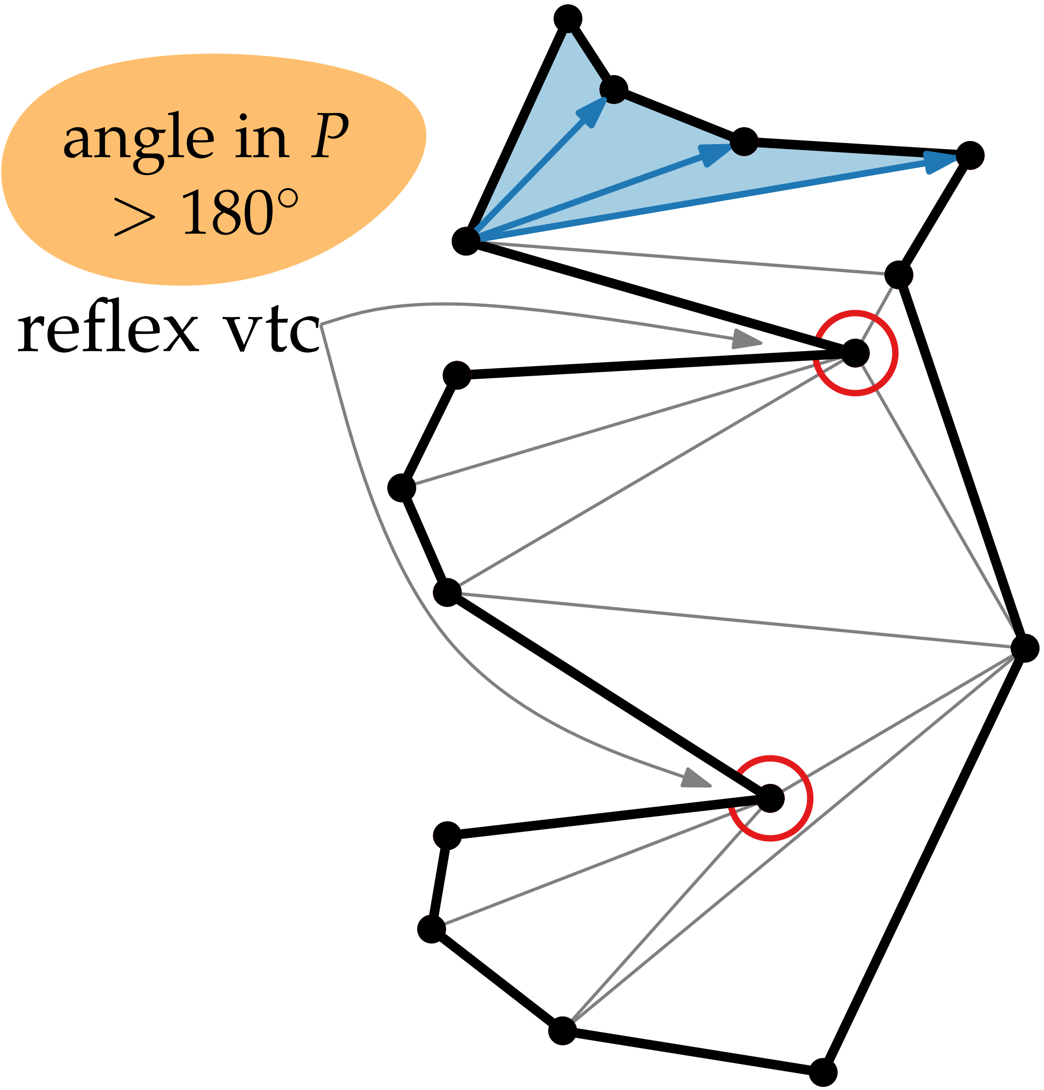
The part of *P* that we have seen but not yet triangulated is a *funnel*.



chains of *reflex* vtc

Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom

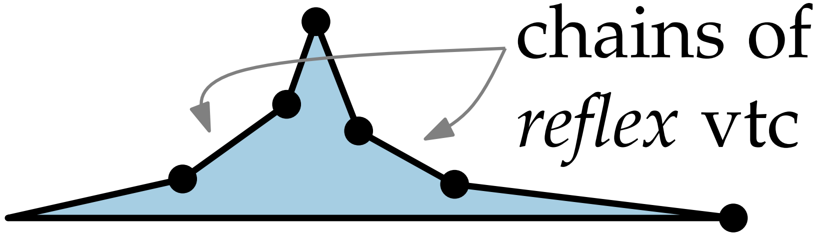


angle in P $> 180°$

reflex vtc

Invariant?

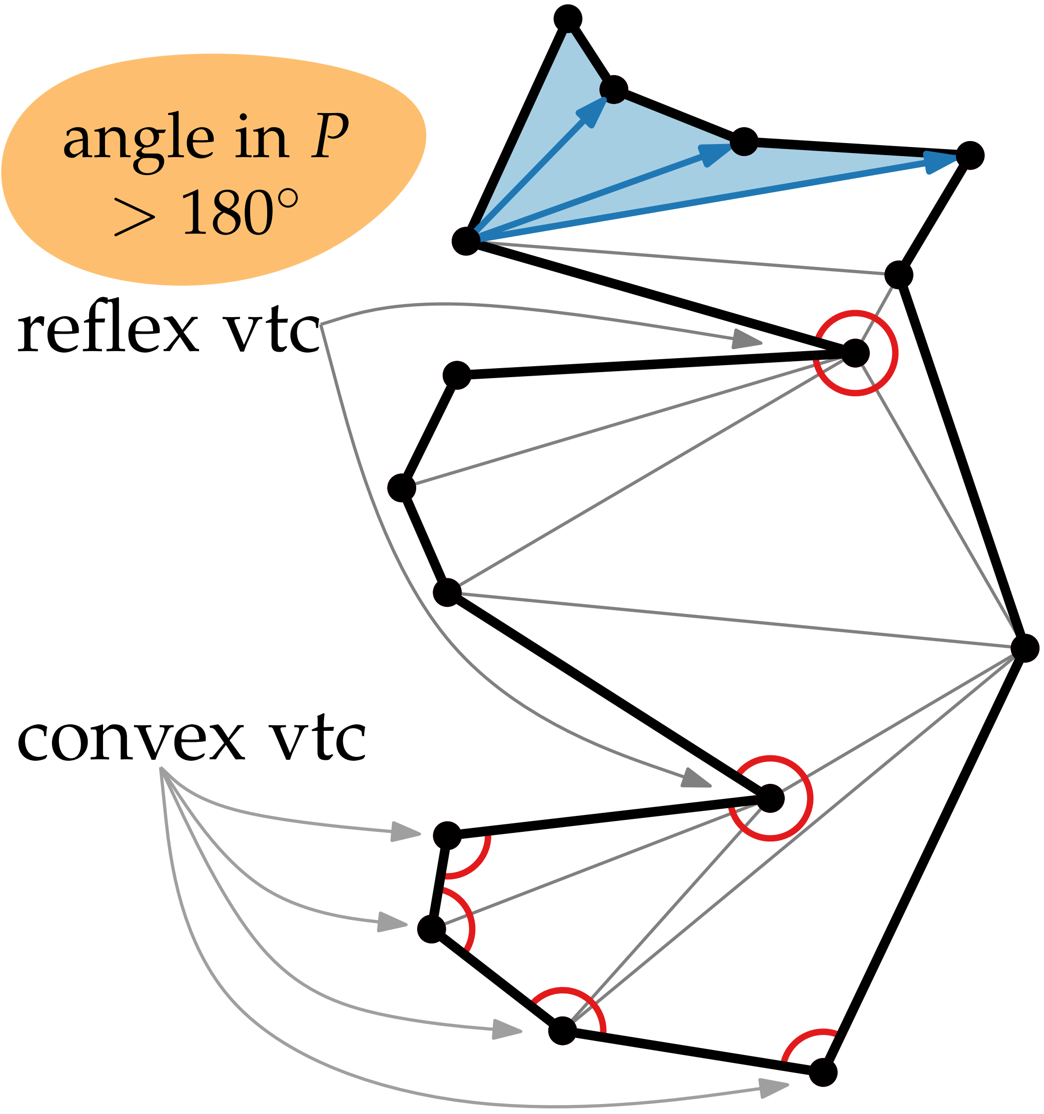
The part of P that we have seen but not yet triangulated is a *funnel*.



chains of *reflex* vtc

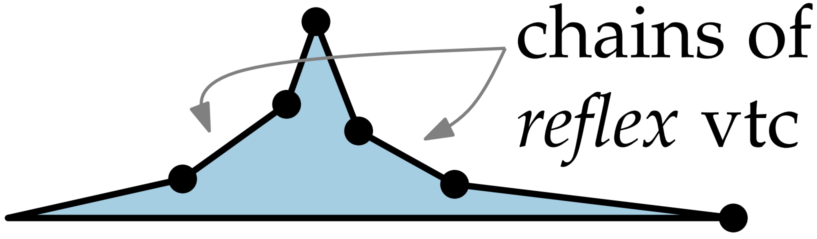
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



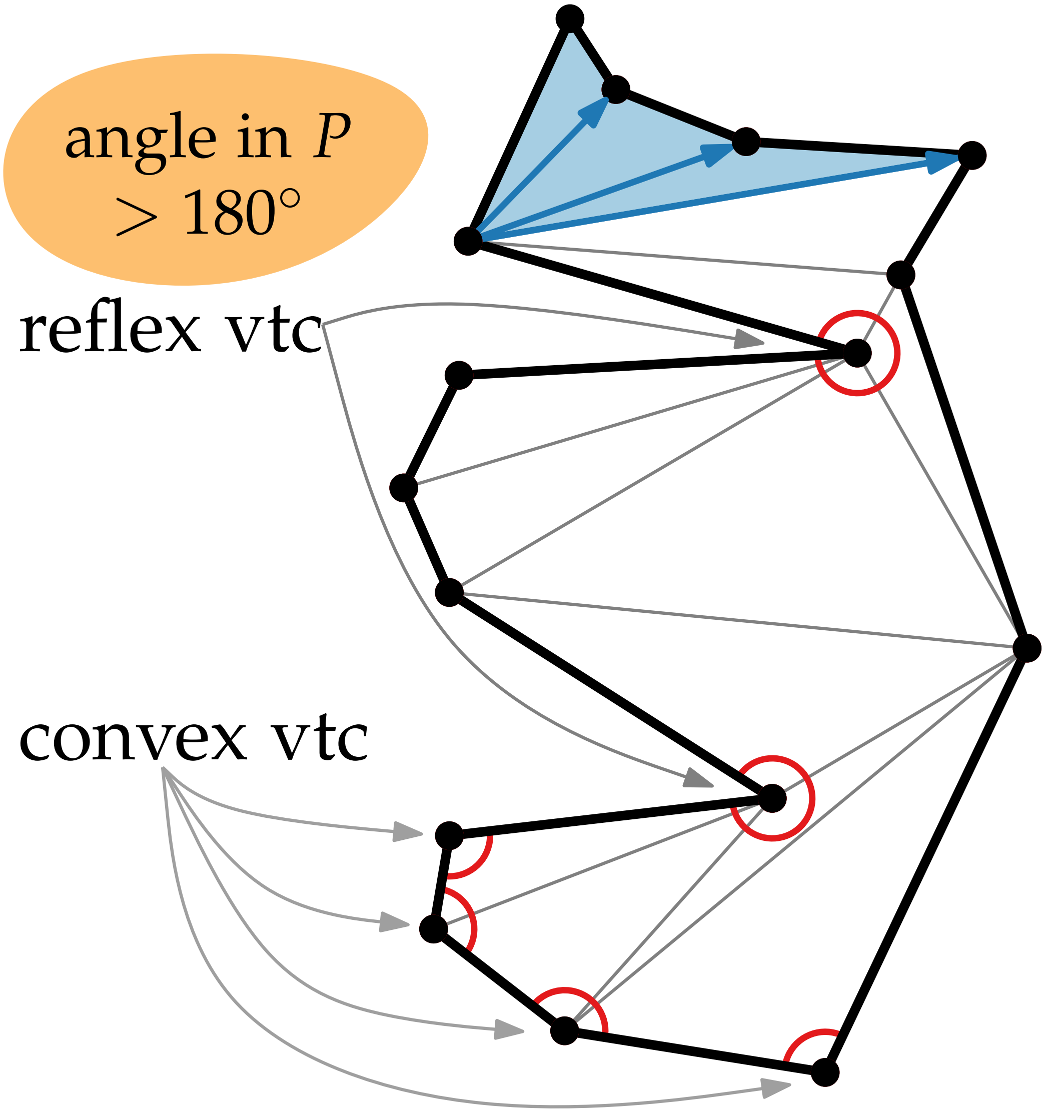
Invariant?

The part of P that we have seen but not yet triangulated is a *funnel*.



Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom



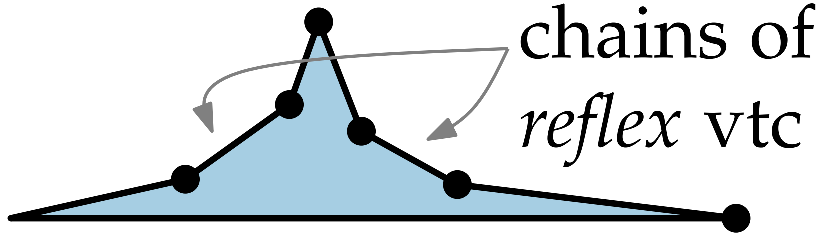
angle in P $> 180°$

reflex vtc

convex vtc

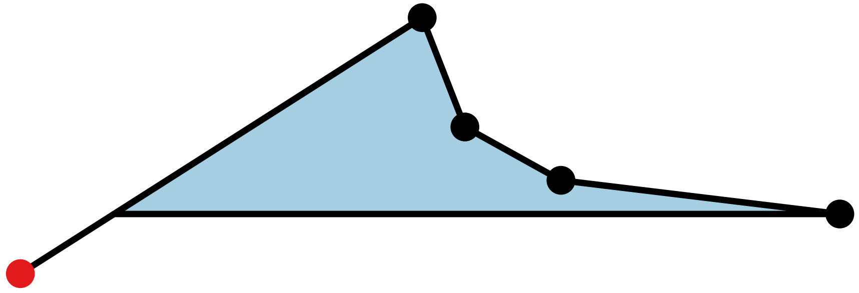
Invariant?

The part of P that we have seen but not yet triangulated is a *funnel*.



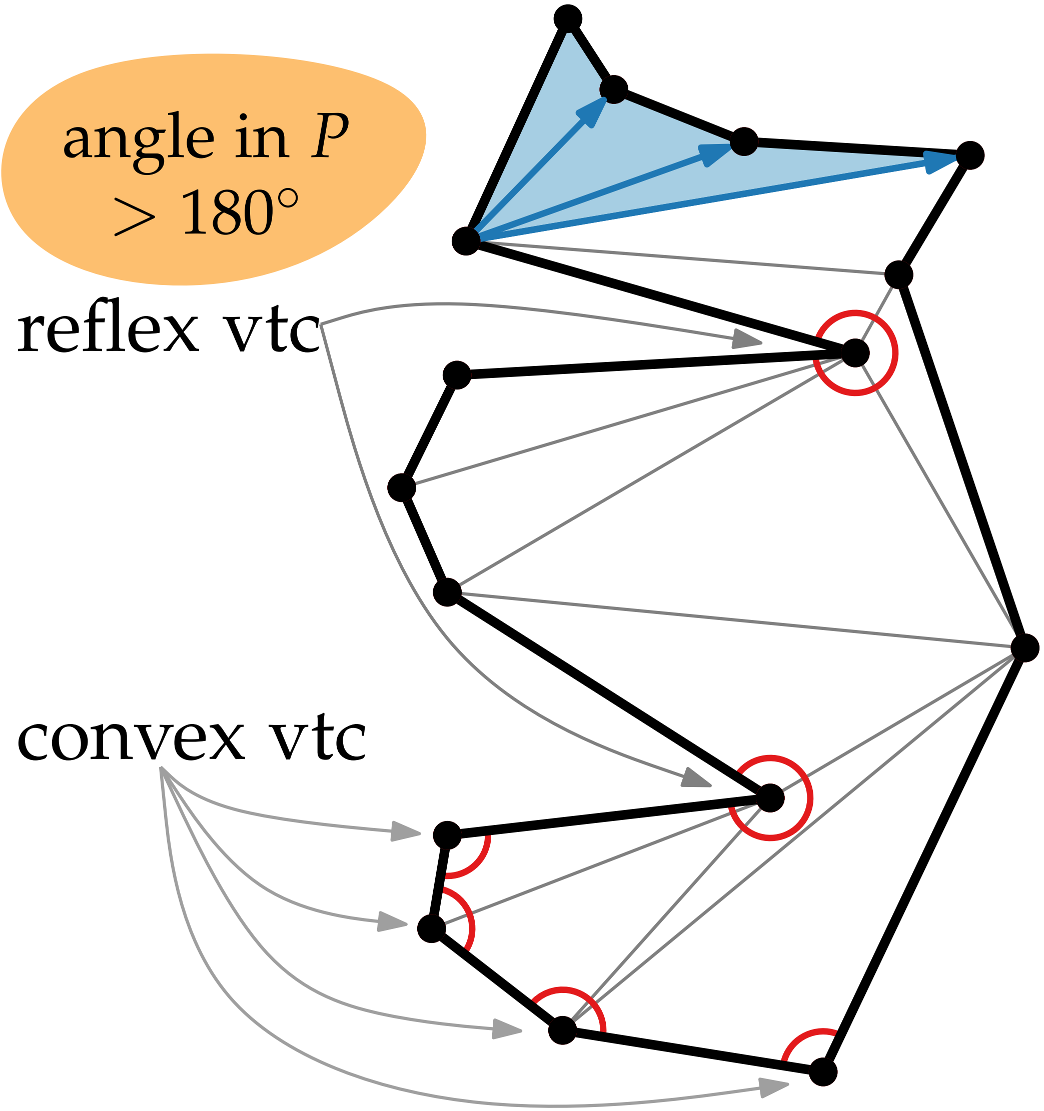
chains of *reflex* vtc

Our funnels are special:



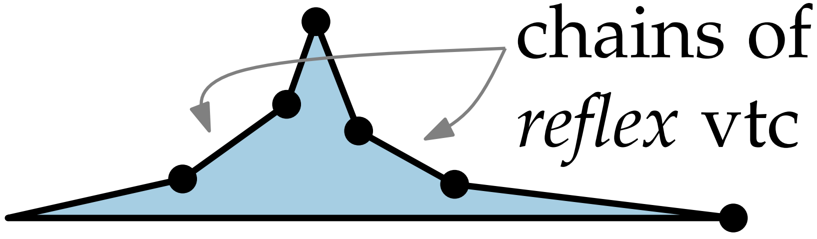
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom

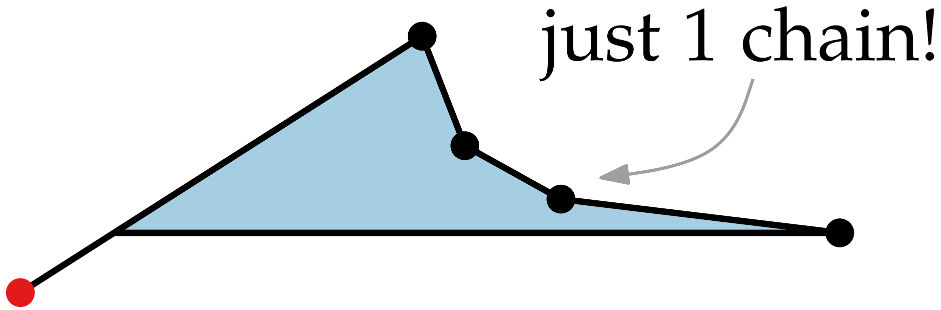


Invariant?

The part of P that we have seen but not yet triangulated is a *funnel*.

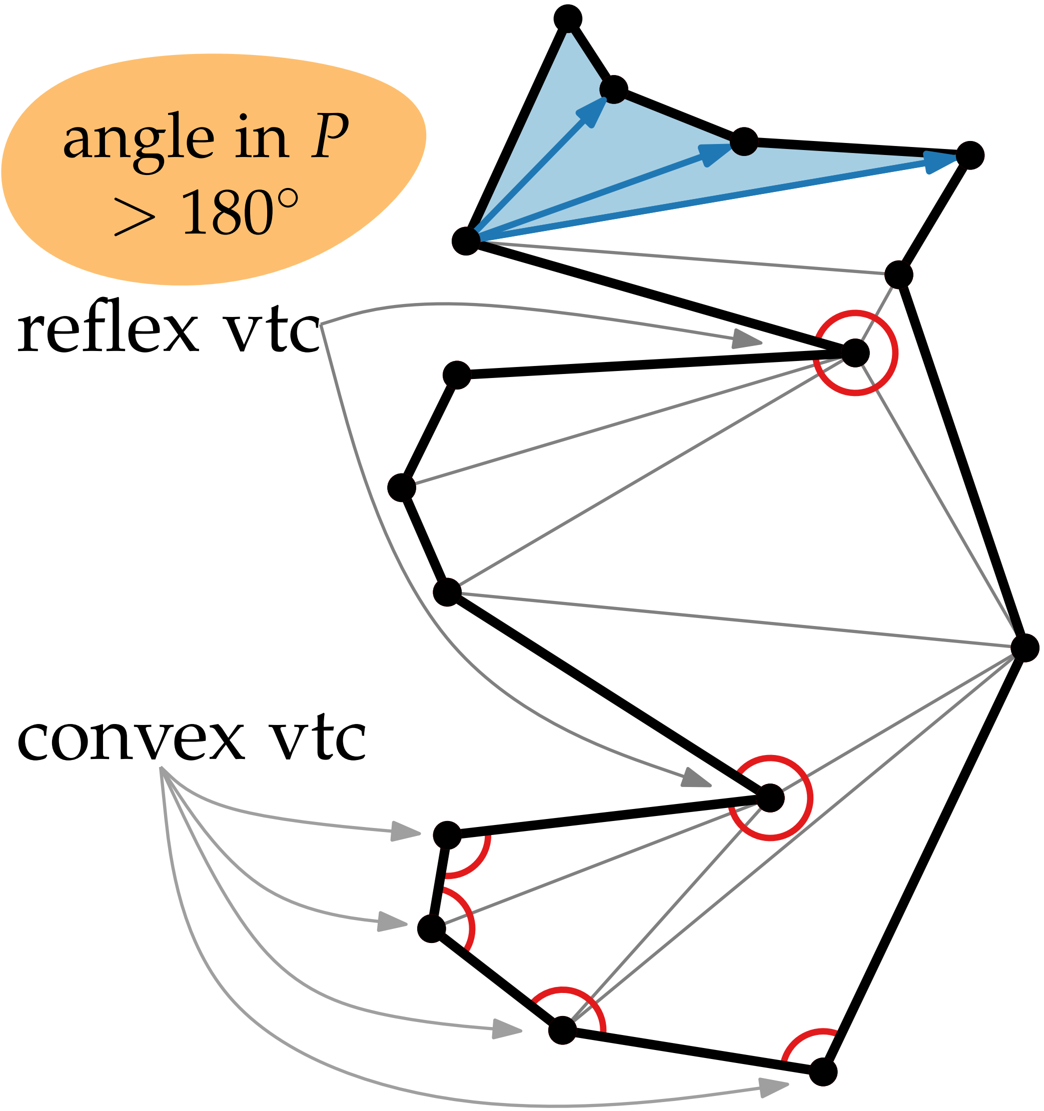


Our funnels are special:



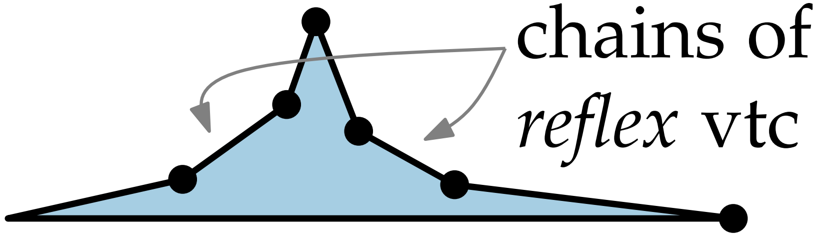
Triangulating a y-Monotone Polygon P

Approach: greedy, going from top to bottom

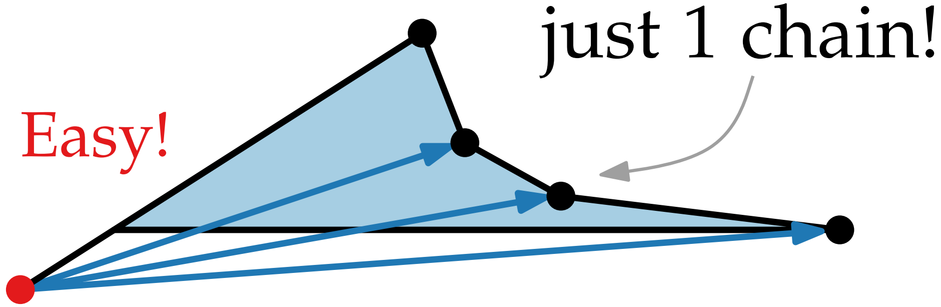


Invariant?

The part of P that we have seen but not yet triangulated is a *funnel*.



Our funnels are special:

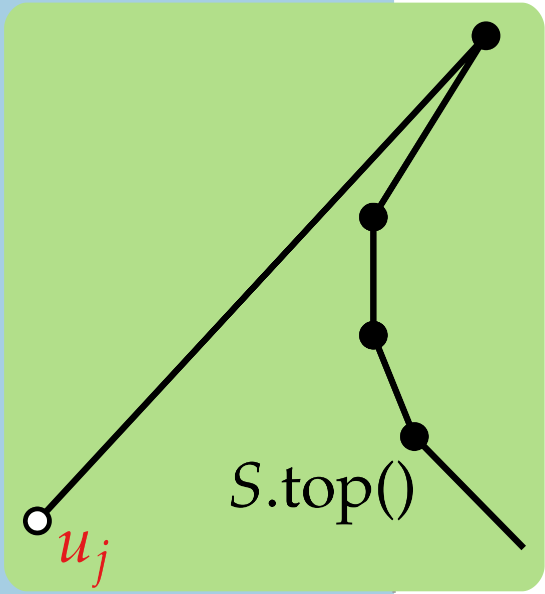


Algorithm

```
**TriangulateMonotonePolygon**(Polygon $P$ as circular vertex list)
    merge left and right chain → seq. $u_1, \ldots, u_n$ with $y_1 \geq \ldots \geq y_n$
    Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
    **for** $j \leftarrow 3$ **to** $n - 1$ **do**
```

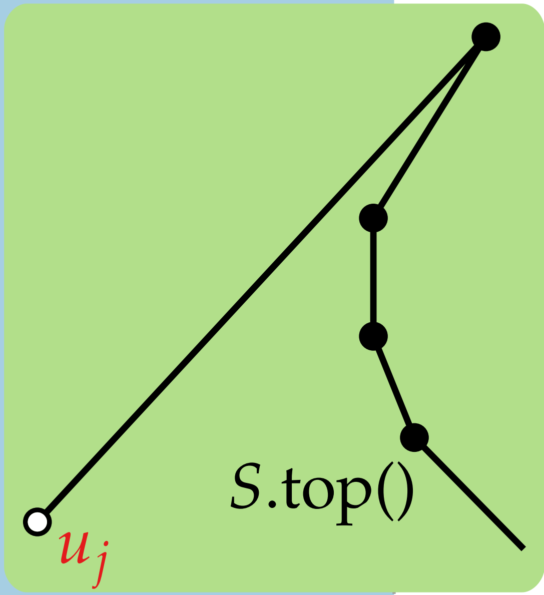
Algorithm

```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \ldots, u_n$ with $y_1 \geq \ldots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n - 1$ do
      if $u_j$ and $S$.top() lie on different chains then

      else

```



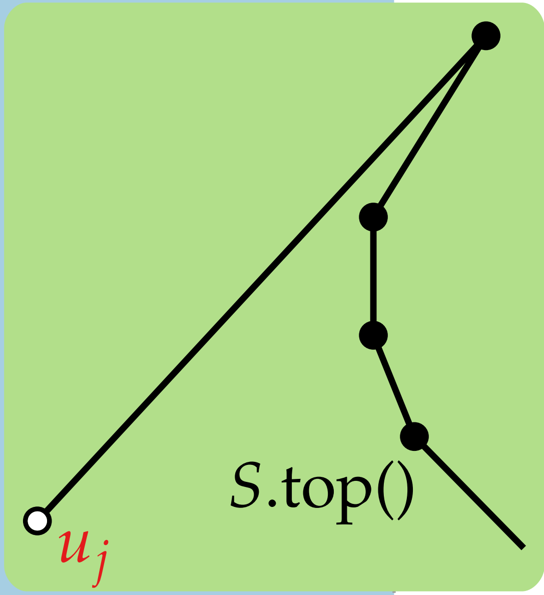
Algorithm

```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \ldots, u_n$ with $y_1 \geq \ldots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n - 1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$

      else
```



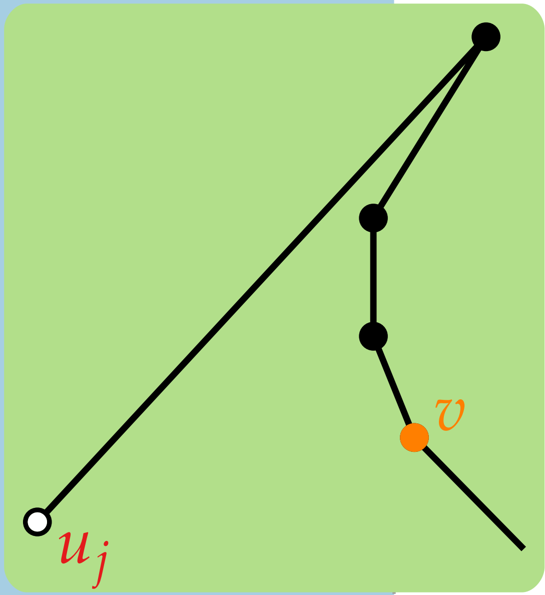
Algorithm

```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \ldots, u_n$ with $y_1 \geq \ldots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n-1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$

      else
```



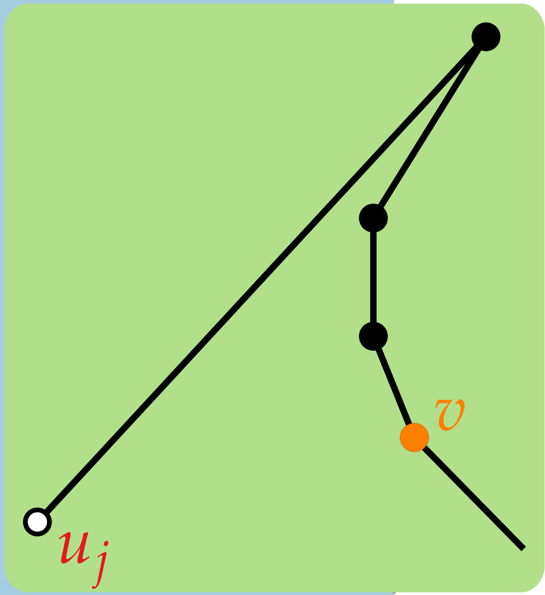
Algorithm

```
**TriangulateMonotonePolygon**(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \ldots, u_n$ with $y_1 \geq \ldots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n - 1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$

      else
```



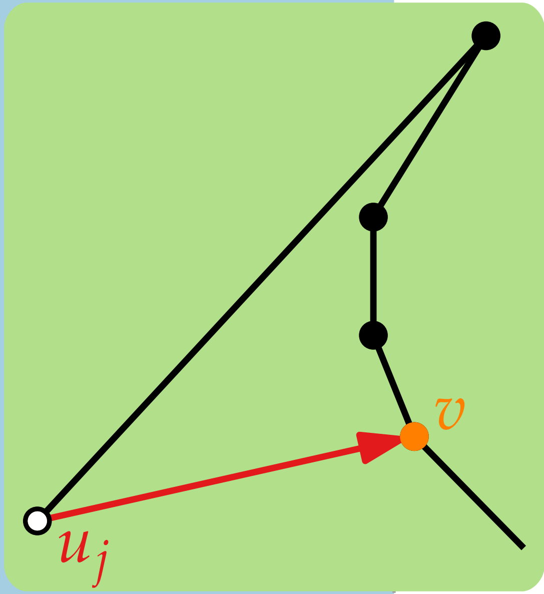
Algorithm

```
**TriangulateMonotonePolygon**(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \ldots, u_n$ with $y_1 \geq \ldots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n-1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$

      else
```



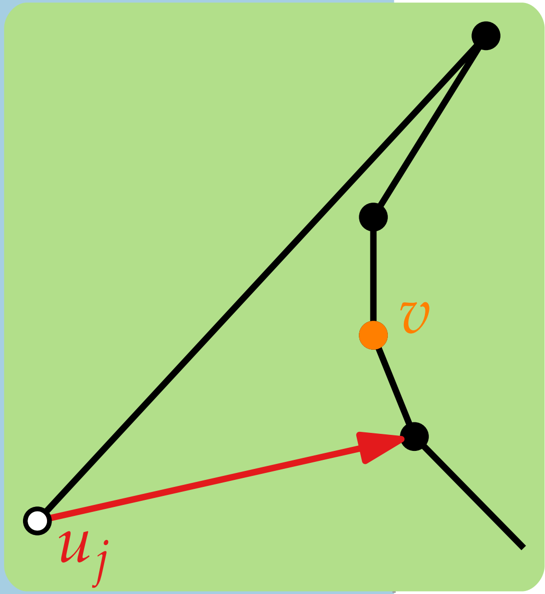
Algorithm

```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain $\to$ seq. $u_1, \dots, u_n$ with $y_1 \geq \dots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n - 1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$

      else
```



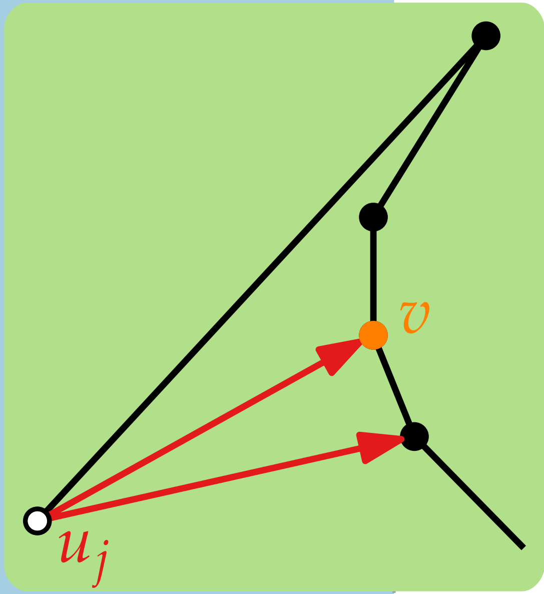
Algorithm

```
**TriangulateMonotonePolygon**(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \ldots, u_n$ with $y_1 \geq \ldots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n-1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$

      else

```



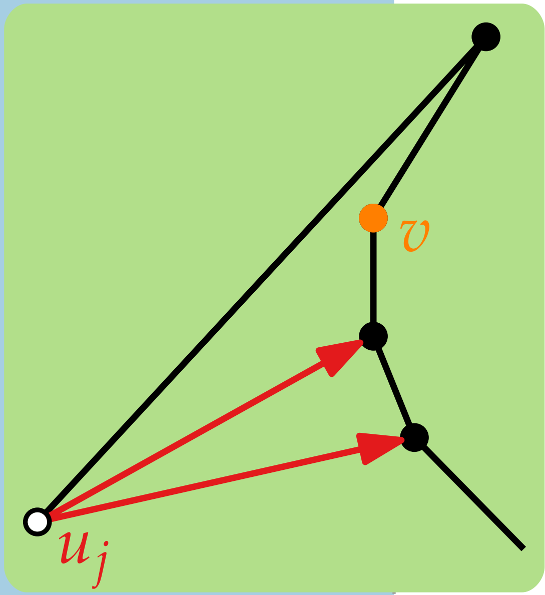
Algorithm

```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain $\rightarrow$ seq. $u_1, \ldots, u_n$ with $y_1 \geq \ldots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n-1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$

      else
```



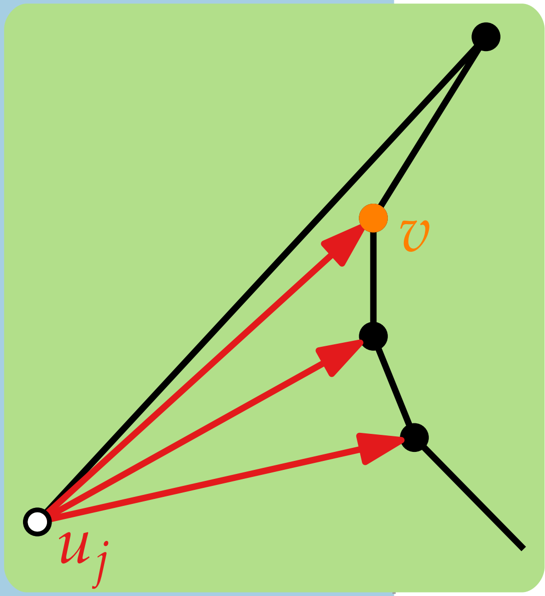
Algorithm

```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \ldots, u_n$ with $y_1 \geq \ldots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n - 1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$

      else
```



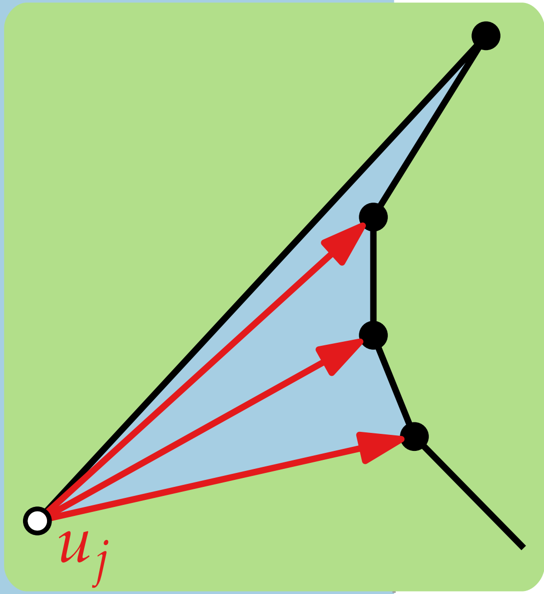
Algorithm

```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \ldots, u_n$ with $y_1 \geq \ldots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n-1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$
      else
```



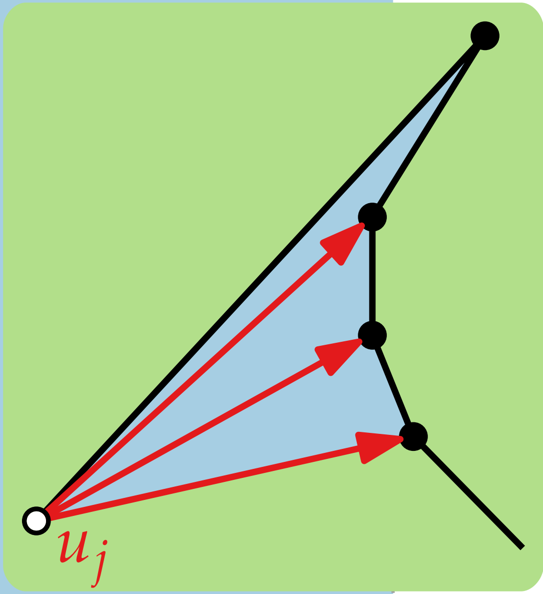
Algorithm

```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \dots, u_n$ with $y_1 \geq \dots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n-1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$

      else
```



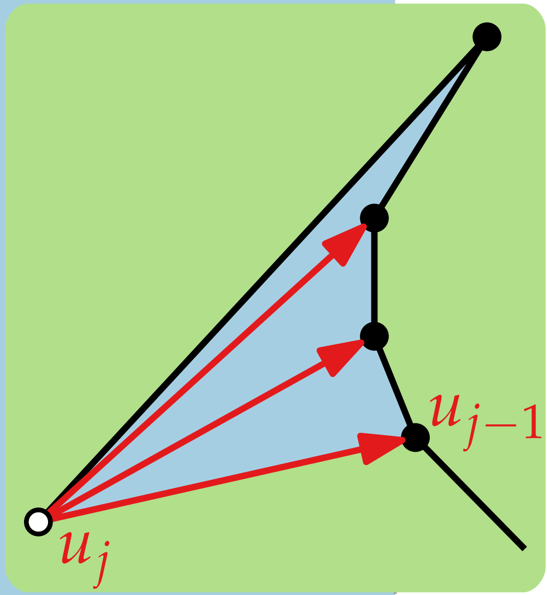
Algorithm

```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \dots, u_n$ with $y_1 \geq \dots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n - 1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$
          $S$.push($u_{j-1}$); $S$.push($u_j$)
      else
```



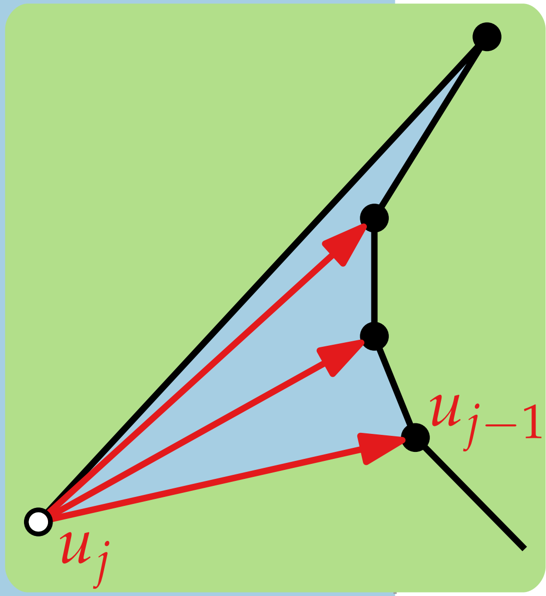
Algorithm

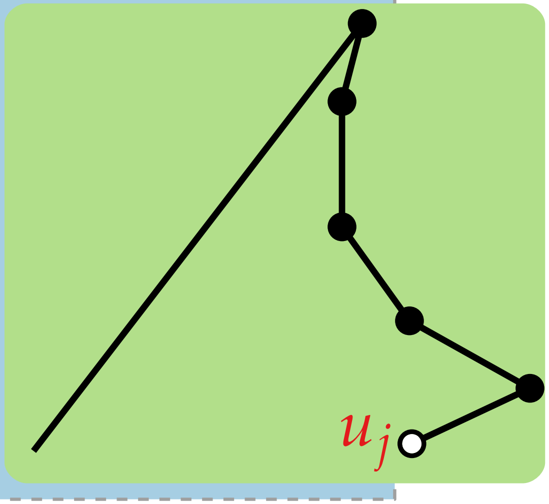
```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
    merge left and right chain → seq. $u_1, \dots, u_n$ with $y_1 \geq \dots \geq y_n$
    Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
    for $j \leftarrow 3$ to $n-1$ do
        if $u_j$ and $S$.top() lie on different chains then
            while not $S$.empty() do
                $v \leftarrow S$.pop()
                if not $S$.empty() then draw diag. $(u_j, v)$
            $S$.push($u_{j-1}$); $S$.push($u_j$)
        else
```



Algorithm

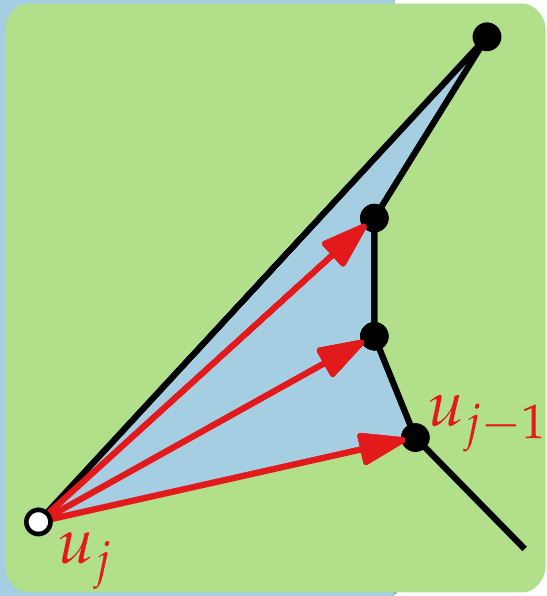
```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \dots, u_n$ with $y_1 \geq \dots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n-1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$
          $S$.push($u_{j-1}$); $S$.push($u_j$)
      else
```

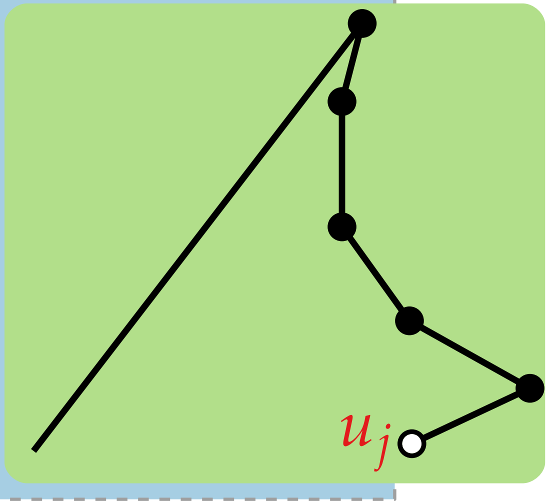




Algorithm

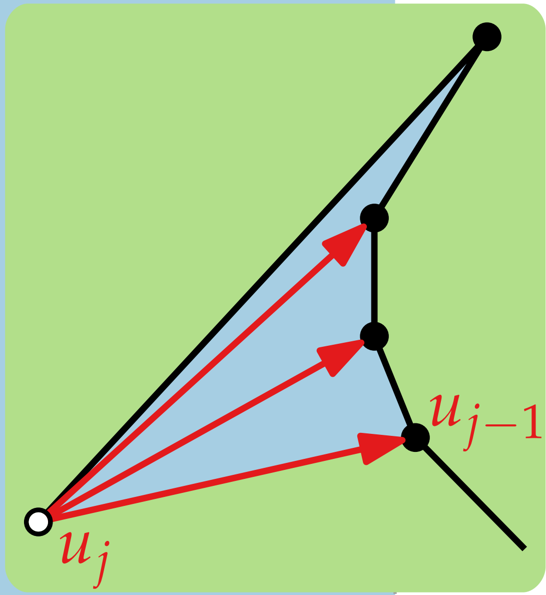
```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \ldots, u_n$ with $y_1 \geq \ldots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n-1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$
          $S$.push($u_{j-1}$); $S$.push($u_j$)
      else
          $v \leftarrow S$.pop()
```

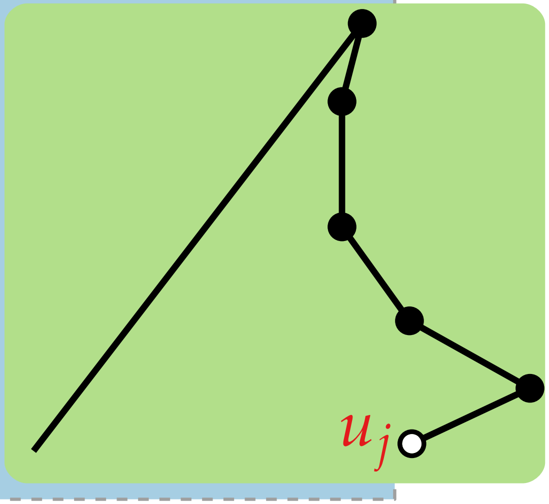




Algorithm

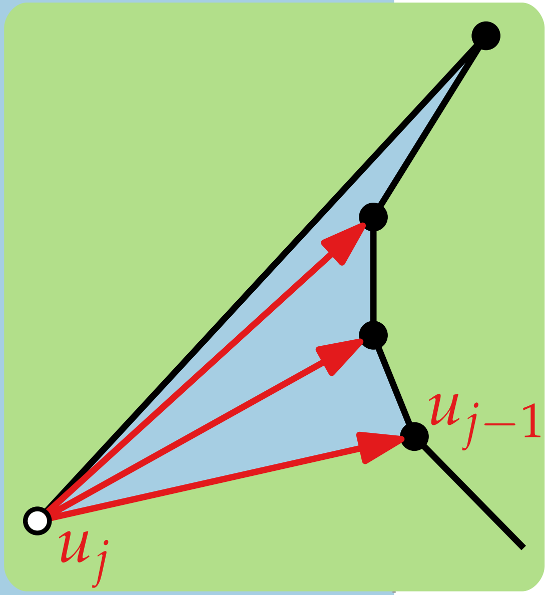
```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \dots, u_n$ with $y_1 \geq \dots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n - 1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$
          $S$.push($u_{j-1}$); $S$.push($u_j$)
      else
          $v \leftarrow S$.pop()
          while not $S$.empty() and $u_j$ sees $S$.top() do
              $v \leftarrow S$.pop()
              draw diagonal $(u_j, v)$
```

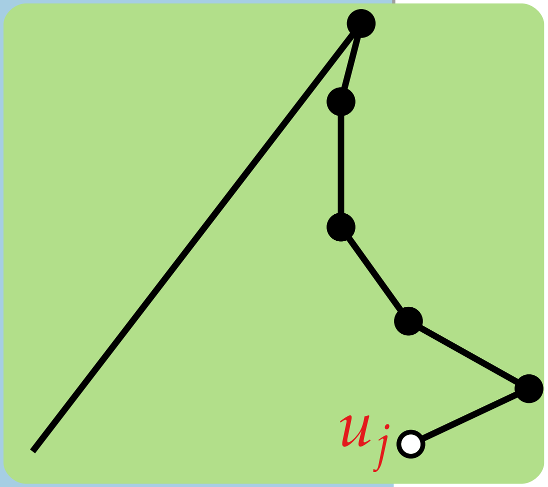




Algorithm

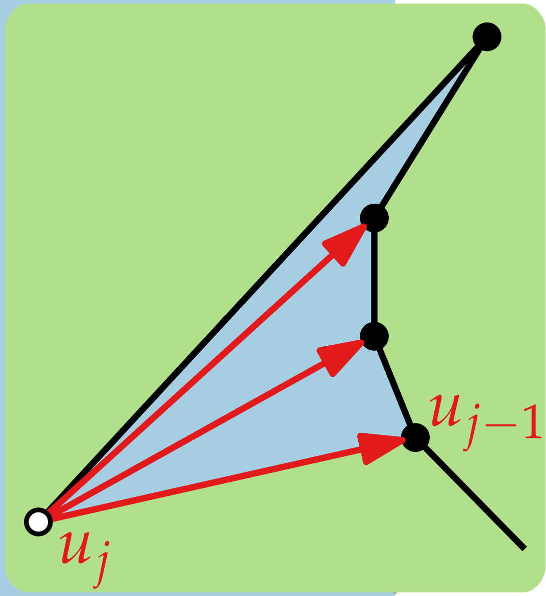
```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \dots, u_n$ with $y_1 \geq \dots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n-1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$
          $S$.push($u_{j-1}$); $S$.push($u_j$)
      else
          $v \leftarrow S$.pop()
          while not $S$.empty() and $u_j$ sees $S$.top() do
              $v \leftarrow S$.pop()
              draw diagonal $(u_j, v)$
```

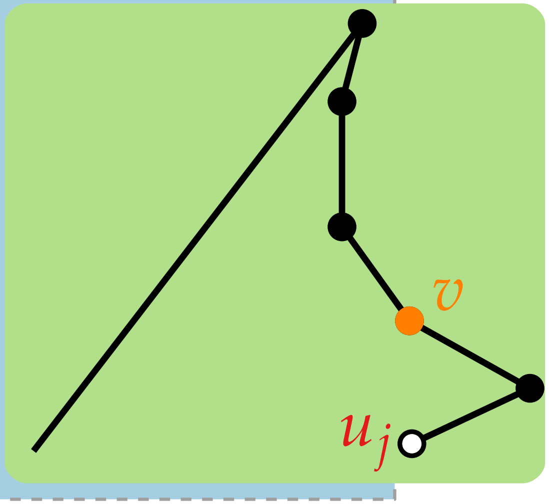




Algorithm

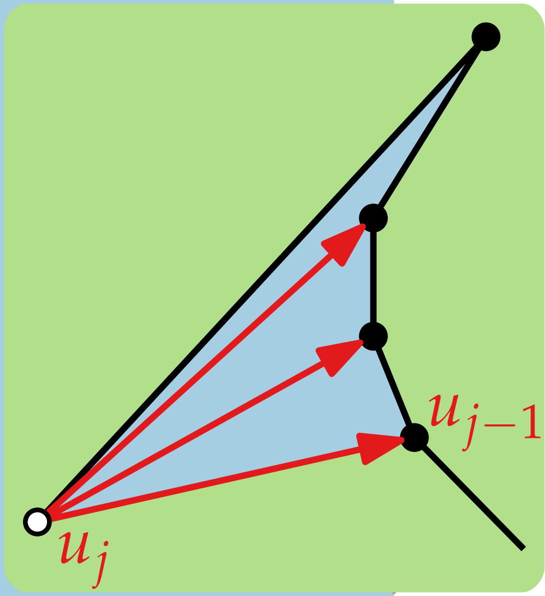
```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \ldots, u_n$ with $y_1 \geq \ldots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n-1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$
          $S$.push($u_{j-1}$); $S$.push($u_j$)
      else
          $v \leftarrow S$.pop()
          while not $S$.empty() and $u_j$ sees $S$.top() do
              $v \leftarrow S$.pop()
              draw diagonal $(u_j, v)$
```

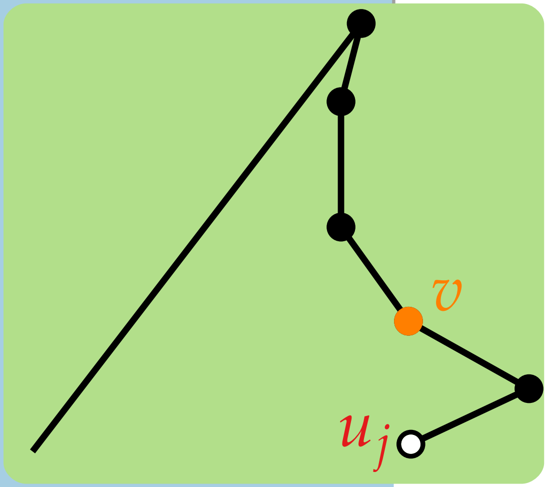




Algorithm

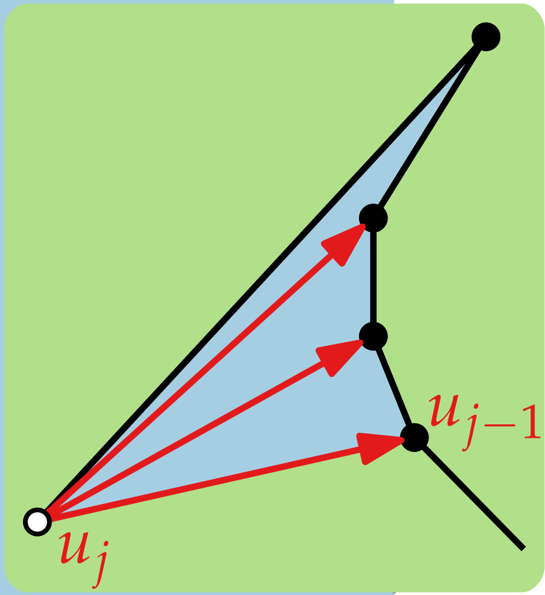
```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \dots, u_n$ with $y_1 \geq \dots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n-1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$
          $S$.push($u_{j-1}$); $S$.push($u_j$)
      else
          $v \leftarrow S$.pop()
          while not $S$.empty() and $u_j$ sees $S$.top() do
              $v \leftarrow S$.pop()
              draw diagonal $(u_j, v)$
```

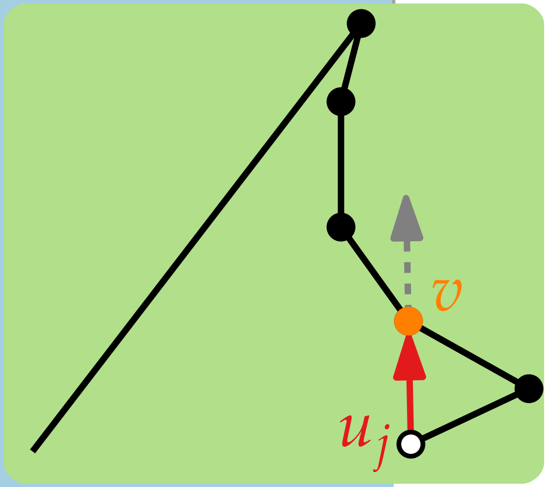




Algorithm

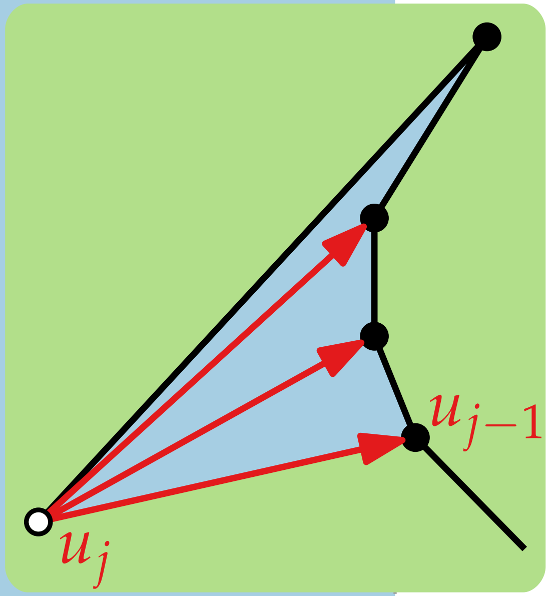
```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \dots, u_n$ with $y_1 \geq \dots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n-1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$
          $S$.push($u_{j-1}$); $S$.push($u_j$)
      else
          $v \leftarrow S$.pop()
          while not $S$.empty() and $u_j$ sees $S$.top() do
              $v \leftarrow S$.pop()
              draw diagonal $(u_j, v)$
```

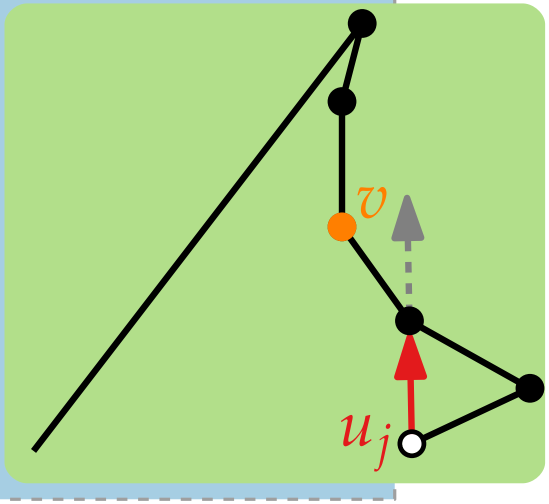




Algorithm

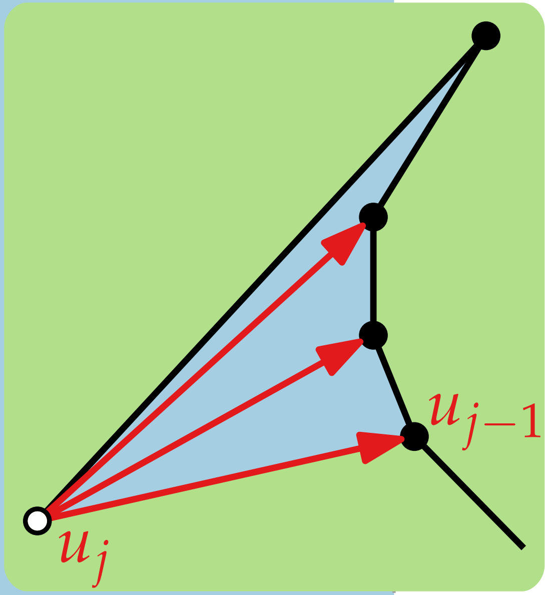
```
**TriangulateMonotonePolygon**(Polygon $P$ as circular vertex list)
    merge left and right chain → seq. $u_1, \ldots, u_n$ with $y_1 \geq \ldots \geq y_n$
    Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
    for $j \leftarrow 3$ to $n - 1$ do
        if $u_j$ and $S$.top() lie on different chains then
            while not $S$.empty() do
                $v \leftarrow S$.pop()
                if not $S$.empty() then draw diag. $(u_j, v)$
            $S$.push($u_{j-1}$); $S$.push($u_j$)
        else
            $v \leftarrow S$.pop()
            while not $S$.empty() and $u_j$ sees $S$.top() do
                $v \leftarrow S$.pop()
                draw diagonal $(u_j, v)$
```

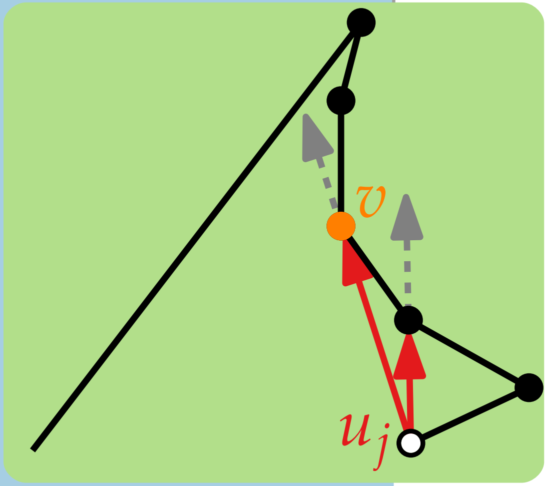




Algorithm

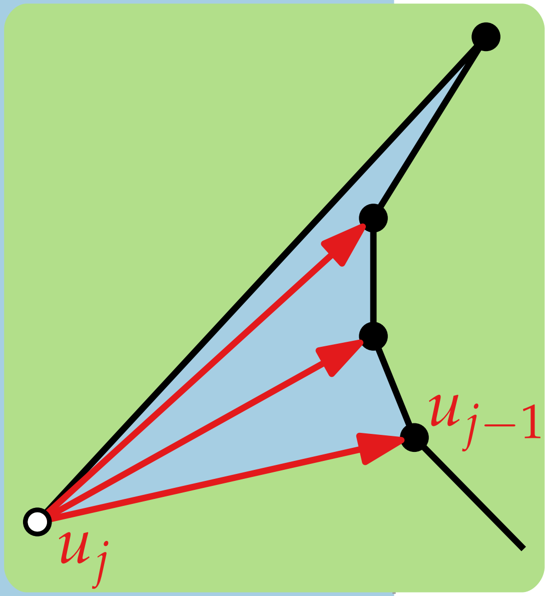
```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \dots, u_n$ with $y_1 \geq \dots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n - 1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$
          $S$.push($u_{j-1}$); $S$.push($u_j$)
      else
          $v \leftarrow S$.pop()
          while not $S$.empty() and $u_j$ sees $S$.top() do
              $v \leftarrow S$.pop()
              draw diagonal $(u_j, v)$
```

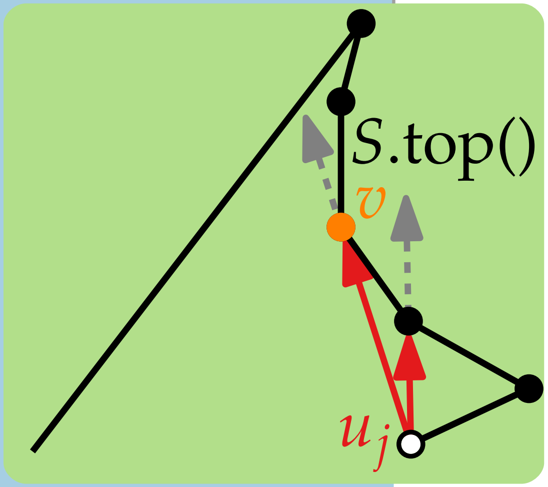




Algorithm

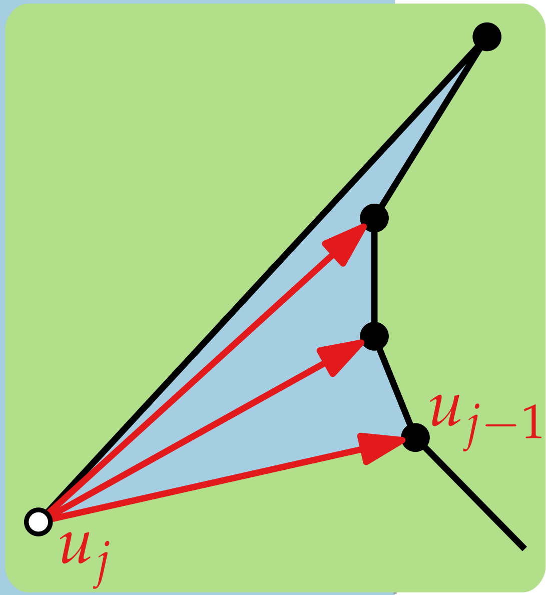
```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \dots, u_n$ with $y_1 \geq \dots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n-1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$
          $S$.push($u_{j-1}$); $S$.push($u_j$)
      else
          $v \leftarrow S$.pop()
          while not $S$.empty() and $u_j$ sees $S$.top() do
              $v \leftarrow S$.pop()
              draw diagonal $(u_j, v)$
```

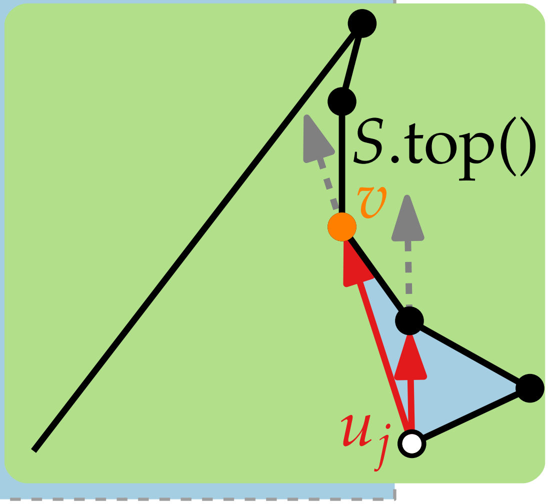




Algorithm

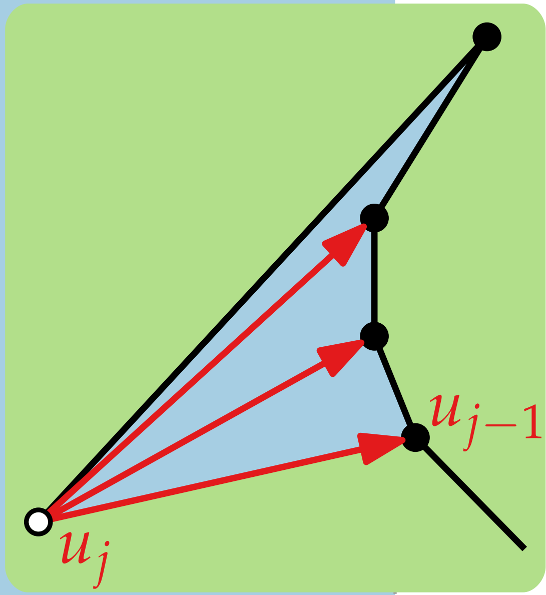
```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \ldots, u_n$ with $y_1 \geq \ldots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n-1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$
          $S$.push($u_{j-1}$); $S$.push($u_j$)
      else
          $v \leftarrow S$.pop()
          while not $S$.empty() and $u_j$ sees $S$.top() do
              $v \leftarrow S$.pop()
              draw diagonal $(u_j, v)$
```

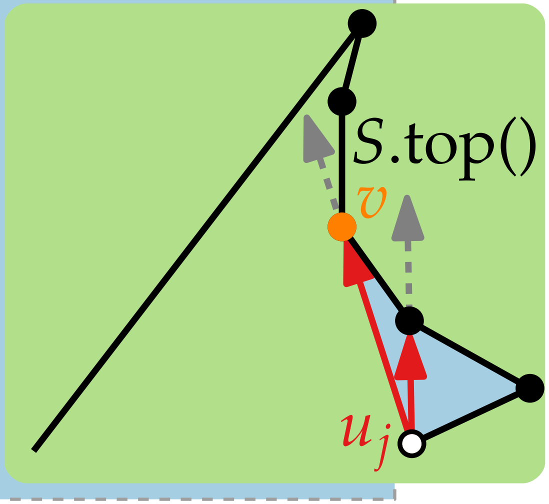




Algorithm

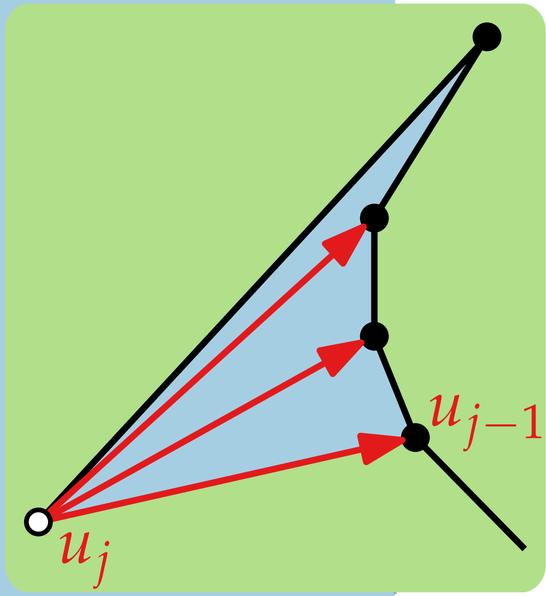
```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \dots, u_n$ with $y_1 \geq \dots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n-1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$
          $S$.push($u_{j-1}$); $S$.push($u_j$)
      else
          $v \leftarrow S$.pop()
          while not $S$.empty() and $u_j$ sees $S$.top() do
              $v \leftarrow S$.pop()
              draw diagonal $(u_j, v)$
          $S$.push($v$); $S$.push($u_j$)
```

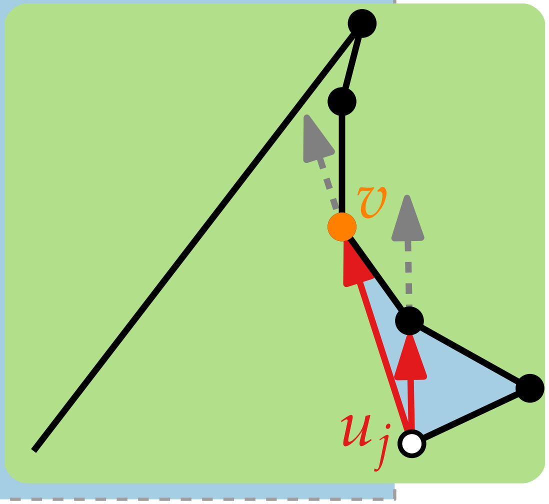




Algorithm

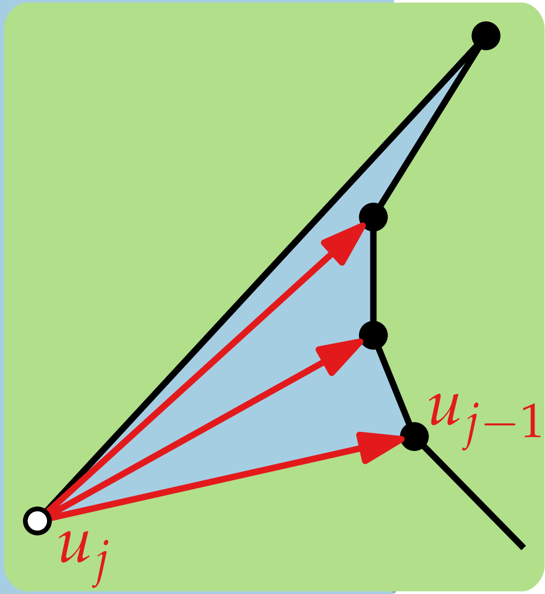
```
**TriangulateMonotonePolygon**(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \ldots, u_n$ with $y_1 \geq \ldots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n-1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$
          $S$.push($u_{j-1}$); $S$.push($u_j$)
      else
          $v \leftarrow S$.pop()
          while not $S$.empty() and $u_j$ sees $S$.top() do
              $v \leftarrow S$.pop()
              draw diagonal $(u_j, v)$
          $S$.push($v$); $S$.push($u_j$)
```

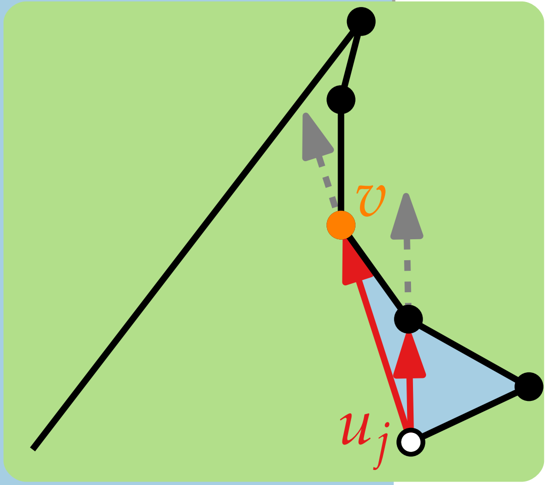




Algorithm

```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \dots, u_n$ with $y_1 \geq \dots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n-1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$
          $S$.push($u_{j-1}$); $S$.push($u_j$)
      else
          $v \leftarrow S$.pop()
          while not $S$.empty() and $u_j$ sees $S$.top() do
              $v \leftarrow S$.pop()
              draw diagonal $(u_j, v)$
          $S$.push($v$); $S$.push($u_j$)
  draw diagonals from $u_n$ to all vtc on $S$ except first
```

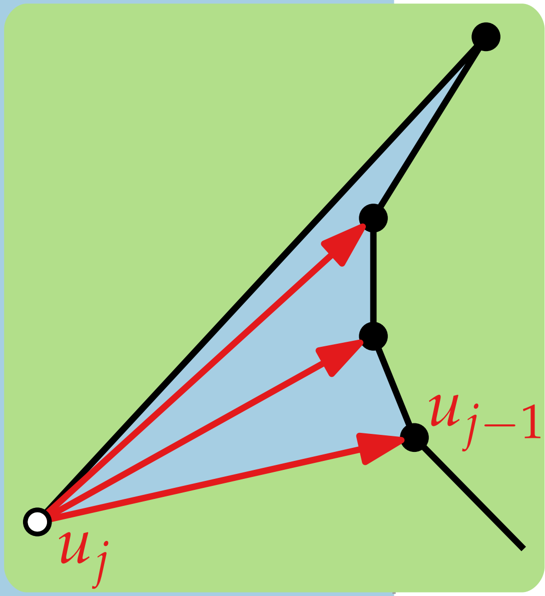


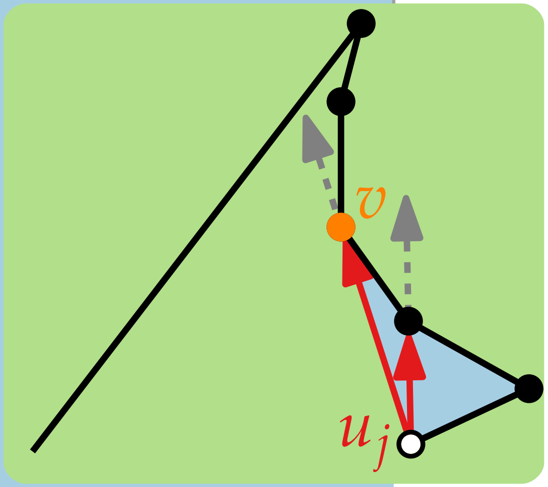


Algorithm

Running time?

```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \ldots, u_n$ with $y_1 \geq \ldots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n - 1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$
          $S$.push($u_{j-1}$); $S$.push($u_j$)
      else
          $v \leftarrow S$.pop()
          while not $S$.empty() and $u_j$ sees $S$.top() do
              $v \leftarrow S$.pop()
              draw diagonal $(u_j, v)$
          $S$.push($v$); $S$.push($u_j$)
  draw diagonals from $u_n$ to all vtc on $S$ except first
```

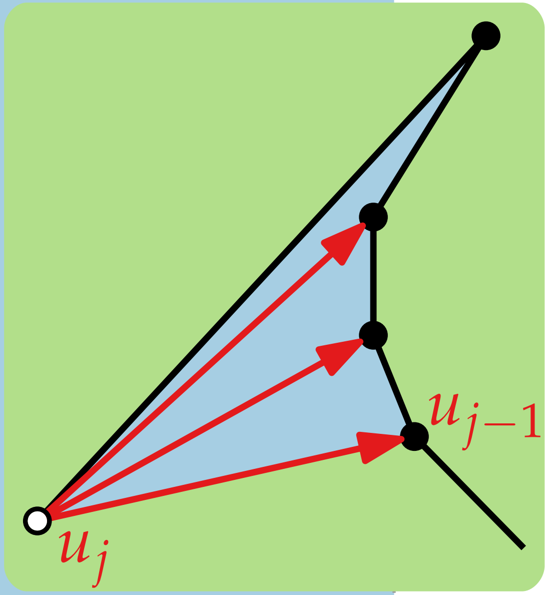


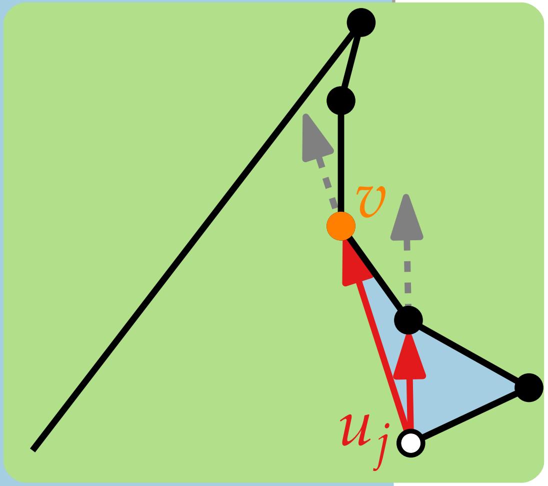


Algorithm

Running time? $\Theta(n)$

```
TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
  merge left and right chain → seq. $u_1, \ldots, u_n$ with $y_1 \geq \ldots \geq y_n$
  Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
  for $j \leftarrow 3$ to $n-1$ do
      if $u_j$ and $S$.top() lie on different chains then
          while not $S$.empty() do
              $v \leftarrow S$.pop()
              if not $S$.empty() then draw diag. $(u_j, v)$
          $S$.push($u_{j-1}$); $S$.push($u_j$)
      else
          $v \leftarrow S$.pop()
          while not $S$.empty() and $u_j$ sees $S$.top() do
              $v \leftarrow S$.pop()
              draw diagonal $(u_j, v)$
          $S$.push($v$); $S$.push($u_j$)
  draw diagonals from $u_n$ to all vtc on $S$ except first
```





Summary

n-vtx polygon $\xrightarrow[O(n \log n)]{}$ “nice” pieces, n' vtc $\xrightarrow[O(n')]{}$ n'' triangles

Summary

n-vtx polygon $\xrightarrow[O(n \log n)]{}$ “nice” pieces, n' vtc $\xrightarrow[O(n')]{}$ n'' triangles

Lemma. A simple polygon with n vertices can be subdivided into y-monotone polygons in $O(n \log n)$ time.

old

Summary

n-vtx polygon $\xrightarrow[O(n \log n)]{}$ “nice” pieces, n' vtc $\xrightarrow[O(n')]{}$ n'' triangles

Lemma. A simple polygon with n vertices can be subdivided into y-monotone polygons in $O(n \log n)$ time.

old

Lemma. A y-monotone polygon with n vertices can be triangulated in $O(n)$ time.

new

Summary

n-vtx polygon $\xrightarrow[O(n \log n)]{}$ "nice" pieces, n' vtc $\xrightarrow[O(n')]{}$ n'' triangles

Lemma. old

A simple polygon with n vertices can be subdivided into y-monotone polygons in $O(n \log n)$ time.

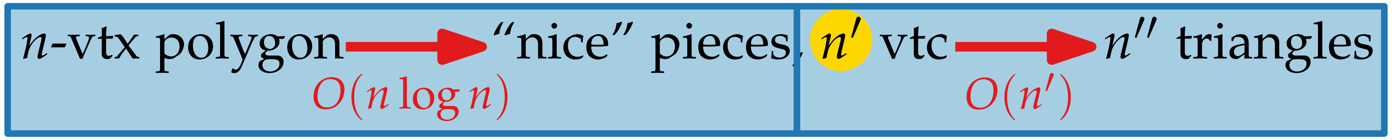
Lemma. new

A y-monotone polygon with n vertices can be triangulated in $O(n)$ time.

Lemma. homework

Subdividing a simple polygon with n vertices by drawing d (pairwise non-crossing) diagonals yields $d+1$ simple polygons of total complexity $O(n)$.

Summary



n-vtx polygon $\xrightarrow[O(n \log n)]{}$ “nice” pieces, n' vtc $\xrightarrow[O(n')]{}$ n'' triangles

Lemma. old

A simple polygon with n vertices can be subdivided into y-monotone polygons in $O(n \log n)$ time.

Lemma. new

A y-monotone polygon with n vertices can be triangulated in $O(n)$ time.

Lemma. home-work

Subdividing a simple polygon with n vertices by drawing d (pairwise non-crossing) diagonals yields $d + 1$ simple polygons of total complexity $O(n)$.

Theorem. A simple polygon with n vertices can be triangulated in $O(n \log n)$ time.

Summary

n-vtx polygon $\xrightarrow[O(n \log n)]{}$ “nice” pieces, n' vtc $\xrightarrow[O(n')]{}$ n'' triangles

Lemma. A simple polygon with n vertices can be subdivided into y-monotone polygons in $O(n \log n)$ time.

old

Lemma. A y-monotone polygon with n vertices can be triangulated in $O(n)$ time.

new

Lemma. Subdividing a simple polygon with n vertices by drawing d (pairwise non-crossing) diagonals yields $d + 1$ simple polygons of total complexity $O(n)$.

home-work

Theorem. A simple polygon with n vertices can be triangulated in $O(n \log n)$ time.

Is this it?

Summary

n-vtx polygon $\xrightarrow[O(n\log n)]{}$ "nice" pieces, n' vtc $\xrightarrow[O(n')]{}$ n'' triangles

Lemma. old

A simple polygon with n vertices can be subdivided into y-monotone polygons in $O(n\log n)$ time.

Lemma. new

A y-monotone polygon with n vertices can be triangulated in $O(n)$ time.

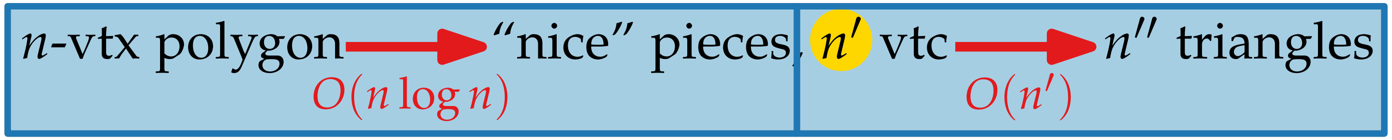
Lemma. homework

Subdividing a simple polygon with n vertices by drawing d (pairwise non-crossing) diagonals yields $d+1$ simple polygons of total complexity $O(n)$.

Theorem. A simple polygon with n vertices can be triangulated in $O(n\log n)$ time.

Is this it? Tarjan & van Wyk [1988]:

Summary



n-vtx polygon $\xrightarrow{O(n \log n)}$ “nice” pieces, n' vtc $\xrightarrow{O(n')}$ n'' triangles

Lemma. old

A simple polygon with n vertices can be subdivided into y-monotone polygons in $O(n \log n)$ time.

Lemma. new

A y-monotone polygon with n vertices can be triangulated in $O(n)$ time.

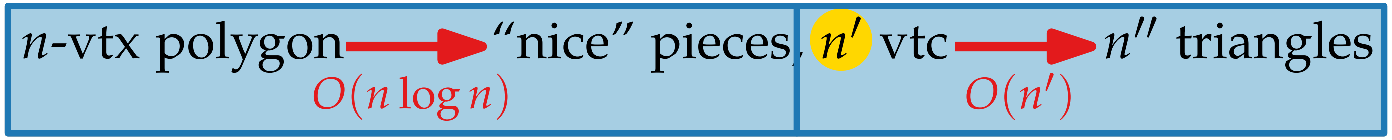
Lemma. homework

Subdividing a simple polygon with n vertices by drawing d (pairwise non-crossing) diagonals yields $d+1$ simple polygons of total complexity $O(n)$.

Theorem. A simple polygon with n vertices can be triangulated in $O(n \log n)$ time.

Is this it? Tarjan & van Wyk [1988]: $O(n \log \log n)$

Summary


n-vtx polygon $\xrightarrow{O(n \log n)}$ "nice" pieces, n' vtc $\xrightarrow{O(n')}$ n'' triangles

Lemma. old

A simple polygon with n vertices can be subdivided into y-monotone polygons in $O(n \log n)$ time.

Lemma. new

A y-monotone polygon with n vertices can be triangulated in $O(n)$ time.

Lemma. home-work

Subdividing a simple polygon with n vertices by drawing d (pairwise non-crossing) diagonals yields $d + 1$ simple polygons of total complexity $O(n)$.

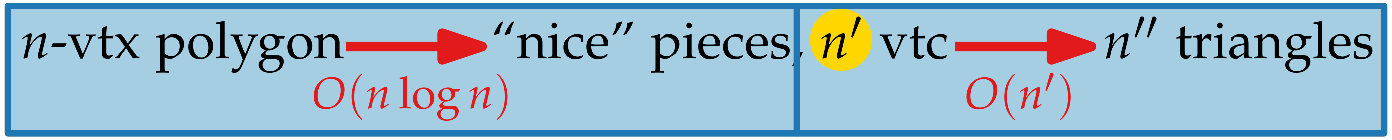
Theorem. A simple polygon with n vertices can be triangulated in $O(n \log n)$ time.

Is this it?

Tarjan & van Wyk [1988]: $O(n \log \log n)$

Clarkson, Tarjan, van Wyk [1989]:

Summary



n-vtx polygon $\xrightarrow{O(n \log n)}$ "nice" pieces, n' vtc $\xrightarrow{O(n')}$ n'' triangles

Lemma. old

A simple polygon with n vertices can be subdivided into y-monotone polygons in $O(n \log n)$ time.

Lemma. new

A y-monotone polygon with n vertices can be triangulated in $O(n)$ time.

Lemma. home-work

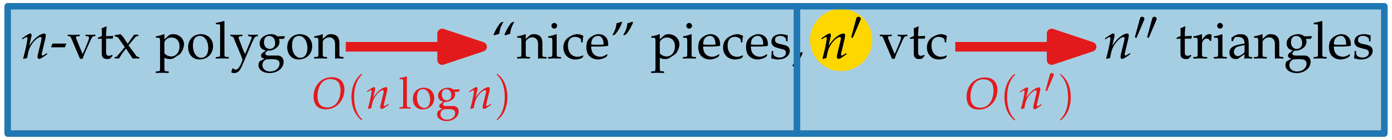
Subdividing a simple polygon with n vertices by drawing d (pairwise non-crossing) diagonals yields $d + 1$ simple polygons of total complexity $O(n)$.

Theorem. A simple polygon with n vertices can be triangulated in $O(n \log n)$ time.

Is this it?

Tarjan & van Wyk [1988]:	$O(n \log \log n)$
Clarkson, Tarjan, van Wyk [1989]:	$O(n \log^* n)$

Summary



n-vtx polygon $\xrightarrow[O(n \log n)]{}$ "nice" pieces, n' vtc $\xrightarrow[O(n')]{}$ n'' triangles

Lemma. old

A simple polygon with n vertices can be subdivided into y-monotone polygons in $O(n \log n)$ time.

Lemma. new

A y-monotone polygon with n vertices can be triangulated in $O(n)$ time.

Lemma. homework

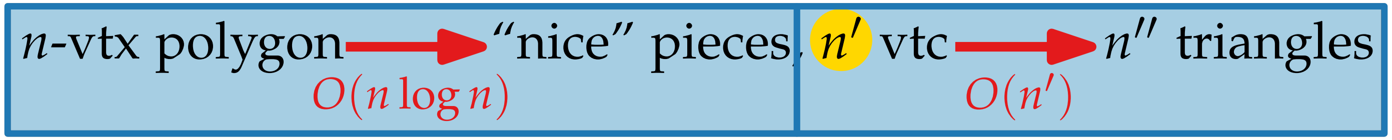
Subdividing a simple polygon with n vertices by drawing d (pairwise non-crossing) diagonals yields $d+1$ simple polygons of total complexity $O(n)$.

Theorem. A simple polygon with n vertices can be triangulated in $O(n \log n)$ time.

Is this it?

Tarjan & van Wyk [1988]:	$O(n \log \log n)$
Clarkson, Tarjan, van Wyk [1989]:	$O(n \log^* n)$
Chazelle [1991]:	

Summary



n-vtx polygon $\xrightarrow{O(n \log n)}$ “nice” pieces, n' vtc $\xrightarrow{O(n')}$ n'' triangles

Lemma. old A simple polygon with n vertices can be subdivided into y-monotone polygons in $O(n \log n)$ time.

Lemma. new A y-monotone polygon with n vertices can be triangulated in $O(n)$ time.

Lemma. home-work Subdividing a simple polygon with n vertices by drawing d (pairwise non-crossing) diagonals yields $d+1$ simple polygons of total complexity $O(n)$.

Theorem. A simple polygon with n vertices can be triangulated in $O(n \log n)$ time.

Is this it?

Tarjan & van Wyk [1988]:	$O(n \log \log n)$
Clarkson, Tarjan, van Wyk [1989]:	$O(n \log^* n)$
Chazelle [1991]:	$O(n)$

Summary

n-vtx polygon $\xrightarrow[O(n \log n)]{}$ "nice" pieces, n' vtc $\xrightarrow[O(n')]{}$ n'' triangles

Lemma. old

A simple polygon with n vertices can be subdivided into y-monotone polygons in $O(n \log n)$ time.

Lemma. new

A y-monotone polygon with n vertices can be triangulated in $O(n)$ time.

Lemma. home-work

Subdividing a simple polygon with n vertices by drawing d (pairwise non-crossing) diagonals yields $d + 1$ simple polygons of total complexity $O(n)$.

Theorem. A simple polygon with n vertices can be triangulated in $O(n \log n)$ time.

Is this it?

Tarjan & van Wyk [1988]:	$O(n \log \log n)$	Kirkpatrick, Klawe, Tarjan [1992]
Clarkson, Tarjan, van Wyk [1989]:	$O(n \log^* n)$	
Chazelle [1991]:	$O(n)$	

Summary

n-vtx polygon $\xrightarrow[O(n \log n)]{}$ "nice" pieces, n' vtc $\xrightarrow[O(n')]{}$ n'' triangles

Lemma. old

A simple polygon with n vertices can be subdivided into y-monotone polygons in $O(n \log n)$ time.

Lemma. new

A y-monotone polygon with n vertices can be triangulated in $O(n)$ time.

Lemma. home-work

Subdividing a simple polygon with n vertices by drawing d (pairwise non-crossing) diagonals yields $d + 1$ simple polygons of total complexity $O(n)$.

Theorem. A simple polygon with n vertices can be triangulated in $O(n \log n)$ time.

Is this it?

Tarjan & van Wyk [1988]:	$O(n \log \log n)$	Kirkpatrick, Klawe, Tarjan [1992]
Clarkson, Tarjan, van Wyk [1989]:	$O(n \log^* n)$	Seidel [1991]: *randomized*
Chazelle [1991]:	$O(n)$	