Computational Geometry

Triangulating Polygons
or
Guarding Art Galleries

Lecture #3

Dr. Philipp Kindermann

Winter Semester 2018
Guarding an Art Gallery

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

![Diagram of a simple polygon](image)
Guarding an Art Gallery

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

Diagram of a simple polygon $P$ with a point $c$ inside.
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 \textit{simple} polygon $P$ (i.e., no holes, no self-intersection)...

\begin{itemize}
  \item \textbf{Observation.} Camera $c$ “sees” a star-shaped region
  \item \textbf{Definition.} A pt $q \in P$ is \textit{visible} from $c \in P$ if $\overline{qc} \subseteq P$
  \item \textbf{Aim:} Use few cameras! But minimizing them is NP-hard…
\end{itemize}
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 them 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 them is NP-hard...
Guarding an Art Gallery

Given a \textit{simple} polygon \( P \) (i.e., no holes, no self-intersection)...

\textbf{Observation.} Camera \( c \) “sees” a star-shaped region

\textbf{Definition.} A pt \( q \in P \) is \textit{visible} from \( c \in P \) if \( \overline{qc} \subseteq P \).

\textbf{Aim:} Use few cameras! \textit{But minimizing them 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 $qc \subseteq P$.

Aim: Use few cameras! But minimizing them 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 them 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 them 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 them 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 them 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 $qc \subseteq P$.

**Aim:** Use few cameras! But minimizing them 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 them 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 them 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 $qc \subseteq P$.

Aim: Use few cameras! But minimizing them 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 them 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.
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.

\[
n = 3: \quad 1 \text{ triangle } \checkmark
\]

\( 3, \ldots, n - 1 \to n: \)
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.

\[
\begin{align*}
n &= 3: \quad &1 \text{ triangle} &\checkmark \\
3, \ldots, n - 1 &\rightarrow n:
\end{align*}
\]
Theorem. 1. Every simple polygon can be triangulated.
   2. Any triangulation of a simple polygon with \( n \) vertices consists of \( n - 2 \) triangles.

\[
\begin{align*}
\text{\( n = 3 \):} & \quad \begin{array}{c}
\text{1 triangle}
\end{array} \\
\text{3, \ldots, } n-1 & \rightarrow n:\n\end{align*}
\]
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.

\[
\begin{align*}
n = 3: & \quad \text{1 triangle} \\
3, \ldots, n - 1 \rightarrow n: & \quad \text{[Diagram]}
\end{align*}
\]
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.

\[
\begin{align*}
n = 3: & \quad \text{1 triangle} \\
3, \ldots, n - 1 \rightarrow n: & \quad \text{[Diagrams showing triangulation process]} 
\end{align*}
\]
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.

\[
n = 3: \quad 1 \text{ triangle}
\]

3, \ldots, \( n - 1 \to n \): 

\begin{align*}
&v \quad w \quad u \\
&x \text{ furthest from } uw
\end{align*}
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.

\[ n = 3: \] \hspace{1cm} 1 \text{ triangle} \checkmark

3, \ldots, \( n - 1 \) \rightarrow \( n \):

[Diagram showing triangulation process]

\( x \) 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.

\[
n = 3: \quad \text{1 triangle} \quad \checkmark
\]

3, \ldots, \( n - 1 \to n \):

\[
x \text{ furthest from } uw
\]
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

\[ 3, \ldots, n - 1 \rightarrow n: \]

3 vertices \( \Rightarrow 1 \) 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.

\[
n = 3: \quad 1 \text{ triangle}
\]

3, \ldots, \( n - 1 \rightarrow n: \)

\[
3 \text{ vtcs } \Rightarrow 1 \text{ triangle}
\]
\[
n - 1 \text{ vtcs } \Rightarrow n - 3 \text{ triangles}
\]
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.

\[
n = 3: \quad \text{1 triangle} \quad \checkmark
\]

3, \ldots, \( n - 1 \rightarrow n: \quad \checkmark
\]

\[3 \text{ vtcs} \Rightarrow 1 \text{ triangle}\]

\[n - 1 \text{ vtcs} \Rightarrow n - 3 \text{ triangles}\]

\[\Rightarrow n - 2 \text{ triangles} \quad \checkmark\]
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.

\[ n = 3: \quad \text{1 triangle} \checkmark \]

\[ 3, \ldots, n - 1 \rightarrow n: \]

- 3 vertices $\Rightarrow$ 1 triangle
- $n - 1$ vertices $\Rightarrow n - 3$ triangles
  $\Rightarrow n - 2$ 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 ✓

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

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.

$n = 3$:  
3 triangles

$x$ furthest from $uw$

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

$n - m + 2$ vtcs $\Rightarrow n - m$ triangles

$3$ vtcs $\Rightarrow 1$ triangle

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

$\Rightarrow n - 2$ 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.

\[ n = 3: \quad \begin{array}{c} 1 \text{ triangle} \end{array} \]

3, \ldots, \( n - 1 \rightarrow n: \)

\[ \begin{align*}
3 \text{ vtcs} & \Rightarrow 1 \text{ triangle} \\
(n-1) \text{ vtcs} & \Rightarrow (n-3) \text{ triangles} \\
& \Rightarrow n-2 \text{ triangles}
\end{align*} \]

\[ x \text{ furthest from } uw \]

\[ \begin{align*}
m \text{ vtcs} & \Rightarrow m-2 \text{ triangles} \\
(n-m+2) \text{ vtcs} & \Rightarrow n-m \text{ triangles} \\
& \Rightarrow n-2 \text{ triangles}
\end{align*} \]
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, $\lceil n/3 \rceil$ 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, \( \lceil n/3 \rceil \) 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]

[dBvKvKO'08]
The Art Gallery Theorem

[Chvátal ’75]

**Theorem.** For surveilling a simple polygon with $n$ vertices, $\lceil n/3 \rceil$ 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]

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

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

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

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

3-color the vtc's
The Art Gallery Theorem

**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

Theorem. For surveilling a simple polygon with \( n \) vertices, \( \lfloor n/3 \rfloor \) cameras are sometimes necessary and always sufficient.
The Art Gallery Theorem

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

3-color the vtcs

dual tree
The Art Gallery Theorem

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
**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

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

3-color the vtcs

Traverse the dual tree
The Art Gallery Theorem

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

3-color the vtc's

Traverse the dual tree
The Art Gallery Theorem

**Theorem.** For surveilling a simple polygon with \( n \) vertices, \( \lceil n/3 \rceil \) cameras are sometimes necessary and always sufficient.

3-color the vtc's

Traverse the dual tree
The Art Gallery Theorem

**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
Theorem. For surveilling a simple polygon with \( n \) vertices, \( \lfloor n/3 \rfloor \) cameras are sometimes necessary and always sufficient.
The Art Gallery Theorem

**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

**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

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

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

3-color the vtc's

Traverse the dual tree
**The Art Gallery Theorem**

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

3-color the vtc's

Traverse the dual tree

Pick “smallest” color
The Art Gallery Theorem

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

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

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

**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, \( \lceil n/3 \rceil \) 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

For surveilling a simple polygon with \( n \) vertices, \( \lceil n/3 \rceil \) 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

| 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 \( \rightarrow \) “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 $\rightarrow$ “nice” pieces, $n'$ vtc $\rightarrow$ $n''$ triangles
The Art Gallery Theorem

**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
\[
O(n \log n)
\]
The Art Gallery Theorem

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 \( \rightarrow \) “nice” pieces, \( n' \) vtc \( \rightarrow \) \( n'' \) triangles
  - \( O(n \log n) \)
  - \( O(n') \)
The Art Gallery Theorem

Theorem. For surveilling a simple polygon with \( n \) vertices, \( \lceil n/3 \rceil \) 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 \( \rightarrow \) “nice” pieces, \( n' \) vtc \( \rightarrow \) \( n'' \) triangles

\( O(n \log n) \) \( \rightarrow \) \( O(n') \)

Definition. A polygon \( P \) is \( y \)-monotone if, for any horizontal line \( \ell \), \( \ell \cap P \) is connected.
The Art Gallery Theorem

**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:**
\[
\begin{align*}
\text{n-vtx polygon} & \rightarrow \text{“nice” pieces, } n' \text{ vtc} & \rightarrow \text{n'' triangles} \\
& \quad \text{running time: } O(n \log n) & \quad \text{running time: } O(n')
\end{align*}
\]

**Definition.** A polygon \( P \) is \( y \)-monotone if, for any horizontal line \( \ell \), \( \ell \cap P \) is connected.
The Art Gallery Theorem

**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 $\rightarrow$ “nice” pieces, $n'$ vtc $\rightarrow$ $n''$ triangles

$O(n \log n)$ $\rightarrow$ $O(n')$

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

**Theorem.** For surveilling a simple polygon with $n$ vertices, $\lceil n/3 \rceil$ 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 $\rightarrow$ “nice” pieces, $n'$ vtc $\rightarrow$ $n''$ triangles

$O(n \log n)$ $\rightarrow$ $O(n')$

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

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 \( \rightarrow \) “nice” pieces, \( n' \) vtc \( \rightarrow \) \( n'' \) triangles
  - \( O(n \log n) \) \( \rightarrow \) \( O(n') \)

Definition. A polygon \( P \) is \( y \)-monotone if, for any horizontal line \( \ell \), \( \ell \cap P \) 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

  - regular vertices
Part. a Polygon into Monotone Pieces

Idea: Classify vertices of given simple polygon $P$

- $\text{turn}$ vertices:
  vertical component of walking direction changes
  
  - $\text{start}$ vertex
  
  - $\text{split}$ vertex

- $\text{regular}$ vertices
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
  - *split* vertex
  - *end* vertex
  - *merge* vertex
- *regular* vertices

Lemma: Let $P$ be a simple polygon. Then $P$ is $y$-monotone $\iff P$ 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 vtc.s.
Towards an Algorithm

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

Problem: Diagonals must not cross
Towards an Algorithm

Idea: Add **diagonals** to “destroy” split and merge vtc.s.

Problem: Diagonals must not cross – each other – edges of $P$
Towards an Algorithm

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

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 vtc.

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 vtc.

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 vtc.

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 vtc.

**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 vtc.

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

1) Treating split vertices

Connect $v$ to vertex $w^*$ 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 vtc.s.

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

1) Treating split vertices

Connect $v$ to vertex $w^*$ 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 vtc.

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

1) Treating split vertices

Connect $v$ to vertex $w^*$ 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 $w^*$ 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 vtc.

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

1) Treating split vertices

Connect $v$ to vertex $w^*$ 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 vtc's.

**Problem:** Diagonals must not cross – each other – edges of \( P \)

1) Treating split vertices

Connect \( v \) to vertex \( w^\ast \) 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 vtc.

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

1) Treating split vertices

Connect $v$ to vertex $w^*$ 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 vtc.

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

1) Treating split vertices

Connect $v$ to vertex $w^*$ 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 vtc.

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

**1) Treating split vertices**

Connect $v$ to vertex $w^*$ 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 $\text{helper}(\text{left}(v))$. 
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 $w^*$ 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 $\text{helper}(\text{left}(v))$. 
An Algorithm

2) Treating merge vertices
An Algorithm

2) Treating merge vertices
An Algorithm

2) Treating merge vertices
An Algorithm

2) Treating merge vertices

\[ v \text{ split} \]
An Algorithm

2) Treating merge vertices

[Image of a diagram illustrating the process of treating merge vertices]
An Algorithm

2) Treating merge vertices

[Diagram showing the concept of treating merge vertices with labels 'split' and 'regular']
An Algorithm

2) Treating merge vertices

\textbf{makeMonotone}(polygon }P)\textbf{ }
\begin{align*}
D & \leftarrow \text{DCEL}(V(P), E(P)) \\
Q & \leftarrow \text{priority queue on } V(P) \\
T & \leftarrow \text{empty bin. search tree}
\end{align*}
An Algorithm

2) Treating merge vertices

makeMonotone(polygon P)
\[ \mathcal{D} \leftarrow \text{DCEL}(V(P), E(P)) \]
\[ Q \leftarrow \text{priority queue on } V(P) \]
\[ T \leftarrow \text{empty bin. search tree} \]
An Algorithm

2) Treating merge vertices

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

\{doubly-connected edge list:
  data structure for planar subdivisions
\}
\[ (x, y) \prec (x', y') :\Leftrightarrow \]
\[ y > y' \lor (y = y' \land x < x') \]
An Algorithm

2) Treating merge vertices

\text{makeMonotone}(\text{polygon } P)
\begin{align*}
D & \leftarrow \text{DCEL}(V(P), E(P)) \\
Q & \leftarrow \text{priority queue on } V(P) \\
T & \leftarrow \text{empty bin. search tree} \\
\text{while } Q \neq \emptyset \text{ do} \\
\quad & \begin{cases} 
Q.\text{extractMax}() \\
\text{handleTypeVertex}(v) 
\end{cases} \\
\text{return } DCEL \; D
\end{align*}

\begin{align*}
\{ \text{doubly-connected edge list:} \\
\text{data structure for planar subdivisions} \\
(x, y) \prec (x', y') :\iff \\
y > y' \lor (y = y' \land x < x')
\end{align*}
**An Algorithm**

2) Treating merge vertices

```plaintext
makeMonotone(polygon P)
D ← DCEL(V(P), E(P))
Q ← priority queue on V(P)
T ← empty bin. search tree
while Q ≠ ∅ do
    v ← Q.extractMax()
    type ← type of vertex v
    handleTypeVertex(v)
return DCEL D

handleMergeVertex(vertex v)
e ← edge following v cw
if helper(e) merge vtx then
    D.insert(diag(v, helper(e)))
T.delete(e)
e′ ← T.edgeLeftOf(v)
if helper(e′) merge vtx then
    D.insert(diag(v, helper(e′)))
helper(e′) ← v
```

An Algorithm

2) Treating merge vertices

makeMonotone(polygon P)
\[ D \leftarrow \text{DCEL}(V(P), E(P)) \]
\[ Q \leftarrow \text{priority queue on } V(P) \]
\[ T \leftarrow \text{empty bin. search tree} \]
while \( Q \neq \emptyset \) do
\[ \nu \leftarrow Q.\text{extractMax()} \]
\[ \text{type} \leftarrow \text{type of vertex } \nu \]
\[ \text{handleTypeVertex}(\nu) \]
return DCEL \( D \)

handleMergeVertex(vertex \( \nu \))
\[ e \leftarrow \text{edge following } \nu \ cw \]
if helper(\( e \)) merge vtx then
\[ D.\text{insert}(\text{diag}(\nu, \text{helper}(e))) \]
\[ T.\text{delete}(e) \]
\[ e' \leftarrow T.\text{edgeLeftOf}(\nu) \]
if helper(\( e' \)) merge vtx then
\[ D.\text{insert}(\text{diag}(\nu, \text{helper}(e'))) \]
\[ \text{helper}(e') \leftarrow \nu \]
An Algorithm

2) Treating merge vertices

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

while \( Q \neq \emptyset \) do
\[ v \leftarrow Q.\text{extractMax()} \]
\[ \text{type } \leftarrow \text{type of vertex } v \]
\[ \text{handleTypeVertex}(v) \]

return DCEL \( D \)

handleMergeVertex(vertex \( v \))
\[ e \leftarrow \text{edge following } v \text{ cw} \]
\[ \text{if } \text{helper}(e) \text{ merge vtx then} \]
\[ \quad D.\text{insert(diag}(v, \text{helper}(e))) \]
\[ \quad T.\text{delete}(e) \]
\[ e' \leftarrow T.\text{edgeLeftOf}(v) \]
\[ \text{if } \text{helper}(e') \text{ merge vtx then} \]
\[ \quad D.\text{insert(diag}(v, \text{helper}(e'))) \]
\[ \quad \text{helper}(e') \leftarrow v \]
An Algorithm

2) Treating merge vertices

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

handleMergeVertex(vertex \( v \))
\[ e \leftarrow \text{edge following } v \text{ cw} \]
if helper(e) \text{ merge vtx} then
  \[ D.\text{insert}(\text{diag}(v, \text{helper}(e))) \]
\[ T.\text{delete}(e) \]
\[ e' \leftarrow T.\text{edgeLeftOf}(v) \]
if helper(e') \text{ merge vtx} then
  \[ D.\text{insert}(\text{diag}(v, \text{helper}(e'))) \]
  helper(e') \leftarrow v
```
An Algorithm

2) Treating merge vertices

```plaintext
makeMonotone(polygon P)
D ← DCEL(V(P), E(P))
Q ← priority queue on V(P)
T ← empty bin. search tree
while Q ≠ ∅ do
    v ← Q.extractMax()
    type ← type of vertex v
    handleTypeVertex(v)
handleMergeVertex(vertex v)
e ← edge following v cw
if helper(e) merge vtx then
    D.insert(diag(v, helper(e)))
T.delete(e)
e' ← T.edgeLeftOf(v)
if helper(e') merge vtx then
    D.insert(diag(v, helper(e')))
helper(e') ← v
return DCEL D
```
makeMonotone(polygon P)

\[ \mathcal{D} \leftarrow \text{DCEL}(V(P), E(P)) \]
\[ Q \leftarrow \text{priority queue on } V(P) \]
\[ T \leftarrow \text{empty bin. search tree} \]

while \( Q \neq \emptyset \) do

\[ v \leftarrow Q.\text{extractMax()} \]
\[ \text{type} \leftarrow \text{type of vertex } v \]
\[ \text{handleTypeVertex}(v) \]

return \( \text{DCEL } \mathcal{D} \)

handleMergeVertex(vertex v)

\[ e \leftarrow \text{edge following } v \text{ cw} \]
if helper(e) merge vtx then

\[ \mathcal{D}.\text{insert(diag}(v, \text{helper}(e))) \]
\[ T.\text{delete}(e) \]
\[ e' \leftarrow T.\text{edgeLeftOf}(v) \]
if helper(e') merge vtx then

\[ \mathcal{D}.\text{insert(diag}(v, \text{helper}(e')))) \]
\[ \text{helper}(e') \leftarrow v \]
An Algorithm

2) Treating merge vertices

\(\text{makeMonotone}(\text{polygon } P)\)
\[
\begin{align*}
\mathcal{D} &\leftarrow \text{DCEL}(V(P), E(P)) \\
Q &\leftarrow \text{priority queue on } V(P) \\
\mathcal{T} &\leftarrow \text{empty bin. search tree}
\end{align*}
\]
\[
\text{while } Q \neq \emptyset \text{ do}
\begin{align*}
\nu &\leftarrow Q.\text{extractMax}() \\
\text{type} &\leftarrow \text{type of vertex } \nu \\
\text{handleTypeVertex}(\nu)
\end{align*}
\]
\[
\text{return } \text{DCEL } \mathcal{D}
\]

\(\text{handleMergeVertex}(\text{vertex } \nu)\)
\[
\begin{align*}
e &\leftarrow \text{edge following } \nu \text{ cw} \\
\text{if helper}(e) \text{ merge vtx then}
\begin{align*}
\mathcal{D}.\text{insert}(\text{diag}(\nu, \text{helper}(e))) \\
\mathcal{T}.\text{delete}(e)
\end{align*}
\]
\[
e' &\leftarrow \mathcal{T}.\text{edgeLeftOf}(\nu) \\
\text{if helper}(e') \text{ merge vtx then}
\begin{align*}
\mathcal{D}.\text{insert}(\text{diag}(\nu, \text{helper}(e'))) \\
\text{helper}(e') &\leftarrow \nu
\end{align*}
\]
An Algorithm

2) Treating merge vertices

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

\[
\text{handleMergeVertex(vertex } v) \\
\text{ } e \leftarrow \text{edge following } v \text{ cw} \\
\text{if helper}(e) \text{ merge vtx then} \\
\text{ } D.\text{insert(diag}(v, \text{helper}(e))) \\
\text{ } \mathcal{T}.\text{delete}(e) \\
\text{ } e' \leftarrow \mathcal{T}.\text{edgeLeftOf}(v) \\
\text{if helper}(e') \text{ merge vtx then} \\
\text{ } D.\text{insert(diag}(v, \text{helper}(e')))) \\
\text{helper}(e') \leftarrow v
\]
An Algorithm

2) Treating merge vertices

makeMonotone(polygon $P$)
\[
\begin{align*}
D & \leftarrow \text{DCEL}(V(P), E(P)) \\
Q & \leftarrow \text{priority queue on } V(P) \\
T & \leftarrow \text{empty bin. search tree}
\end{align*}
\]
while $Q \neq \emptyset$
\[
\begin{align*}
& v \leftarrow Q.\text{extractMax()} \\
& \text{type} \leftarrow \text{type of vertex } v \\
& \text{handleTypeVertex}(v)
\end{align*}
\]
return DCEL $D$

handleMergeVertex(vertex $v$)
\[
\begin{align*}
e & \leftarrow \text{edge following } v \text{ cw} \\
& \text{if helper}(e) \text{ merge vtx then} \\
& \quad D.\text{insert(diag}(v, \text{helper}(e))) \\
& \quad T.\text{delete}(e) \\
& e' \leftarrow T.\text{edgeLeftOf}(v) \\
& \text{if helper}(e') \text{ merge vtx then} \\
& \quad D.\text{insert(diag}(v, \text{helper}(e'))) \\
& \quad \text{helper}(e') \leftarrow v
\end{align*}
\]
An Algorithm

2) Treating merge vertices

\[
\text{makeMonotone}(\text{polygon } P) \\
\text{D} \leftarrow \text{DCEL}(V(P), E(P)) \\
\text{Q} \leftarrow \text{priority queue on } V(P) \\
\text{T} \leftarrow \text{empty bin. search tree} \\
\text{while } Q \neq \emptyset \text{ do} \\
\quad \text{v} \leftarrow Q.\text{extractMax}() \\
\quad \text{type} \leftarrow \text{type of vertex } v \\
\quad \text{handleTypeVertex}(v) \\
\text{return } \text{DCEL } D
\]

\[
\text{handleMergeVertex}(\text{vertex } v) \\
\text{e} \leftarrow \text{edge following } v \text{ cw} \\
\text{if helper}(e) \text{ merge vtx then} \\
\quad \text{D}.\text{insert}(\text{diag}(v, \text{helper}(e))) \\
\quad \text{T}.\text{delete}(e) \\
\quad \text{e}' \leftarrow \text{T}.\text{edgeLeftOf}(v) \\
\text{if helper}(e') \text{ merge vtx then} \\
\quad \text{D}.\text{insert}(\text{diag}(v, \text{helper}(e'))) \\
\quad \text{helper}(e') \leftarrow v
\]
An Algorithm

2) Treating merge vertices

`makeMonotone(polygon P)`

\[ D \leftarrow \text{DCEL}(V(P), E(P)) \]

\[ Q \leftarrow \text{priority queue on } V(P) \]

\[ T \leftarrow \text{empty bin. search tree} \]

while \( Q \neq \emptyset \) do

\[ v \leftarrow Q.\text{extractMax()} \]

\[ \text{type} \leftarrow \text{type of vertex } v \]

handleTypeVertex(\( v \))

return \( \text{DCEL } D \)

`handleMergeVertex(vertex v)`

\[ e \leftarrow \text{edge following } v \text{ cw} \]

if \( \text{helper}(e) \text{ merge vtx} \) then

\[ D.\text{insert(diag}(v, \text{helper}(e)))) \]

\[ T.\text{delete}(e) \]

\[ e' \leftarrow T.\text{edgeLeftOf}(v) \]

if \( \text{helper}(e') \text{ merge vtx} \) then

\[ D.\text{insert(diag}(v, \text{helper}(e'))) \]

\[ \text{helper}(e') \leftarrow v \]
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

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

- angle in $P > 180^\circ$
- reflex vtc

- 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*.

angle in \( P \) > 180°

reflex vtc

convex vtc

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*.

Our funnels are special:

- angle in \( P \) > 180°
- reflex vtc
- convex vtc
- 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*.

Our funnels are special: just 1 chain!

angle in $P$ $>$ 180°

reflex vtc

convex vtc

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*.

Our funnels are special: just 1 chain!

Easy!
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 \( S \); \( S.push(u_1); S.push(u_2) \)
\textbf{for} \( j \leftarrow 3 \) \textbf{to} \( n - 1 \) \textbf{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


\[
\text{draw diagonals from } u_n \text{ to all vtc on } S \text{ except first and last one}
\]
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
    $v \leftarrow S$.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 and last one
Algorithm

```
TriangulateMonotonePolygon(Polygon P as circular vertex list)
  merge left and right chain → seq. u₁, . . . , uₙ with y₁ ≥ . . . ≥ yₙ
  Stack S; S.push(u₁); S.push(u₂)
  for j ← 3 to n − 1 do
    if uₗ and S.top() lie on different chains then
      while not S.empty() do
        v ← S.pop()
        if not S.empty() then draw diag. (uₗ, v)
    else
      v ← S.pop()
      while not S.empty() and uₗ sees S.top() do
        v ← S.pop()
        draw diagonal (uₗ, v)
        S.push(v); S.push(uₗ)
  draw diagonals from uₙ to all vtc on S except first and last one
```

Algorithm

\textbf{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\((u_2)\)

\textbf{for} \( j \leftarrow 3 \) \textbf{to} \( n-1 \) \textbf{do}

\hspace{1em} \textbf{if} \( u_j \) and \( S\).top() \textbf{lie on different chains} \textbf{then}

\hspace{2em} \textbf{while} \textbf{not} \( S\).empty() \textbf{do}

\hspace{3em} \( v \leftarrow S\).pop()

\hspace{4em} \textbf{if not} \( S\).empty() \textbf{then} draw diag. \((u_j, v)\)

\hspace{1em} \textbf{else}

\hspace{2em} \textbf{while} \textbf{not} \( S\).empty() \textbf{and} \( u_j \) \textbf{sees} \( S\).top() \textbf{do}

\hspace{3em} \( v \leftarrow S\).pop()

\hspace{4em} draw diagonal \((u_j, v)\)

\hspace{3em} \( S\).push\((v)\);

\hspace{3em} \( S\).push\((u_j)\)

\hspace{1em} \textbf{draw diagonals from} \( u_n \) \textbf{to all} vtc \textbf{on} \( S \) \textbf{except first and last one}
Algorithm

\textbf{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)  \\
\textbf{for} \ j \leftarrow 3 \ \textbf{to} \ n-1 \ \textbf{do}  \\
\hspace{1em} \textbf{if} \ u_j \ \textbf{and} \ S.top() \ \textbf{lie on different chains} \ \textbf{then}  \\
\hspace{2em} \textbf{while} \ \textbf{not} \ S.empty() \ \textbf{do}  \\
\hspace{3em} v \leftarrow S.pop()  \\
\hspace{3em} \textbf{if} \ \textbf{not} \ S.empty() \ \textbf{then} \ \textbf{draw} \ \text{diag.} \ (u_j, v)  \\
\hspace{1em} \textbf{else}  \\
\hspace{2em} v \leftarrow S.pop()  \\
\hspace{2em} \textbf{while} \ \textbf{not} \ S.empty() \ \textbf{and} \ u_j \ \textbf{sees} \ S.top() \ \textbf{do}  \\
\hspace{3em} v \leftarrow S.pop()  \\
\hspace{3em} \textbf{draw} \ \text{diagonal} \ (u_j, v)  \\
\hspace{2em} S.push(v); \ S.push(u_j)  \\
\hspace{1em} \textbf{draw} \ \text{diagonals} \ \text{from} \ u_n \ \textbf{to} \ \text{all} \ \text{vtc} \ \text{on} \ S \ \text{except first and last one}
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
    $v \leftarrow S$.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 and last one
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
    \( v \leftarrow S \).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 and last one
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
    \( v \leftarrow S.pop() \)
    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 and last one
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

else
Algorithm

**TriangulateMonotonePolygon** (*Polygon* *P* as circular vertex list)
merge left and right chain → seq. *u*₁, . . . , *u*ₙ with *y*₁ ≥ . . . ≥ *y*ₙ
Stack *S*; *S*.push(*u*₁); *S*.push(*u*₂)
for *j* ← 3 to *n* − 1 do
    if *u*ₗ and *S*.top() lie on different chains then
        while not *S*.empty() do
            *v* ← *S*.pop()
            if not *S*.empty() then draw diag. (*u*ₗ, *v*)
        end while
    else
        *v* ← *S*.pop()
        while not *S*.empty() and *u*ₗ sees *S*.top() do
            *v* ← *S*.pop()
            draw diagonal (*u*ₗ, *v*)
        end while
    end if
end for
draw diagonals from *u*ₙ to all vtc on *S* except first and last one
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


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. \( (u_j, v) \)
      \( S\).push(\( u_{j-1} \)); \( S\).push(\( u_j \))
  else
    \( S\).push(\( u_{j-1} \)); \( S\).push(\( u_j \))

draw diagonals from \( u_n \) to all vtc on \( S \) except first and last one
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. \((u_j, v)\)
      \( S\).push\((u_{j-1})\); \( S\).push\((u_j)\)
  else
    \( u_j \) sees \( S\).top()
    \( v \leftarrow S\).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 and last one
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. $(u_j, v)$
      $S$.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 and last one
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 $\rightarrow$ 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. $(u_j, v)$

$S$.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 $(u_j, v)$

draw diagonals from $u_n$ to all vtc on $S$ except first and last one
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 $(u_j, v)$
    draw diagonals from $u_n$ to all vtc on $S$ except first and last one
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 diagonal $(u_j, v)$
      $S.push(u_{j-1})$; $S.push(u_j)$
  else
    $v \leftarrow S.pop()$
    while not $S.empty()$ and $u_j$ sees $S.top()$ do
      $v \leftarrow S.pop()$
      draw diagonal $(u_j, v)$
    draw diagonals from $u_n$ to all vtc on $S$ except first and last one
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 \( \text{diag. } (u_j, v) \)
      \( S.push(u_{j-1}); S.push(u_j) \)
  else
    \( v \leftarrow S.pop() \)
    while not \( S.empty() \) and \( u_j \) sees \( S.top() \) do
      \( v \leftarrow S.pop() \)
      draw diagonal \( (u_j, v) \)

draw diagonals from \( u_n \) to all vtc on \( S \) except first and last one
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. \( u_1, \ldots, u_n \) 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 diagonal \( (u_j, v) \)
      \( S.push(u_{j-1}); S.push(u_j) \)
  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. $u_1, \ldots, u_n$ 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 $(u_j, v)$

draw diagonals from $u_n$ to all vtc on $S$ except first and last one
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. \( u_1, \ldots, u_n \) 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 and last one.
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($u_j$)

draw diagonals from $u_n$ to all vtc on $S$ except first and last one
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 \( S \); \( S.\text{push}(u_1) \); \( S.\text{push}(u_2) \)
for \( j \leftarrow 3 \) to \( n - 1 \) do
  if \( u_j \) and \( S.\text{top()} \) lie on different chains then
    while not \( S.\text{empty()} \) do
      \( v \leftarrow S.\text{pop()} \)
      if not \( S.\text{empty()} \) then draw diag. \( (u_j, v) \)
      \( S.\text{push}(u_{j-1}) \); \( S.\text{push}(u_j) \)
  else
    \( v \leftarrow S.\text{pop()} \)
    while not \( S.\text{empty()} \) and \( u_j \) sees \( S.\text{top()} \) do
      \( v \leftarrow S.\text{pop()} \)
      draw diagonal \( (u_j, v) \)
      \( S.\text{push}(v) \); \( S.\text{push}(u_j) \)
    draw diagonals from \( u_n \) to all vtc on \( S \) except first
Algorithm

TriangulateMonotonePolygon(Polygon P as circular vertex list)
mmerge 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 diagonal $(u_j, v)$

      $S$.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

Running time?
Algorithm

\textbf{TriangulateMonotonePolygon}(Polygon \ P \text{ 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)
\textbf{for} \ j \leftarrow 3 \ \textbf{to} \ n - 1 \ \textbf{do}
\quad \textbf{if} \ u_j \text{ and } \ S.\text{top()} \text{ lie on different chains} \ \textbf{then}
\quad \quad \textbf{while} \ \textbf{not} \ S.\text{empty()} \ \textbf{do}
\quad \quad \quad v \leftarrow S.\text{pop()}
\quad \quad \quad \textbf{if} \ \textbf{not} \ S.\text{empty() } \textbf{then} \ \text{draw \ diagonal} \ (u_j, v)
\quad \quad S.\text{push}(u_{j-1}); \ S.\text{push}(u_j)
\quad \textbf{else}
\quad \quad v \leftarrow S.\text{pop()}
\quad \quad \textbf{while} \ \textbf{not} \ S.\text{empty()} \ \textbf{and} \ u_j \text{ sees } S.\text{top()} \ \textbf{do}
\quad \quad \quad v \leftarrow S.\text{pop()}
\quad \quad \quad \text{draw \ diagonal} \ (u_j, v)
\quad \quad S.\text{push}(v); \ S.\text{push}(u_j)
\text{draw \ diagonals \ from} \ u_n \ \text{to all vtc on} \ S \ \text{except first}

\textbf{Running time?} \ \Theta(n)
Summary

\[ n\text{-vtx polygon} \rightarrow \text{“nice” pieces, } n' \text{ vtc} \rightarrow n'' \text{ triangles} \]

\[ O(n \log n) \quad O(n') \]
Lemma. A simple polygon with \( n \) vertices can be subdivided into \( y \)-monotone polygons in \( O(n \log n) \) time.
Summary

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

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

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

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

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

<table>
<thead>
<tr>
<th>$n$-vtx polygon</th>
<th>“nice” pieces</th>
<th>$n'$ vtc</th>
<th>$n''$ triangles</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n \log n)$</td>
<td>$O(n')$</td>
<td>$O(n'')$</td>
<td></td>
</tr>
</tbody>
</table>
Summary

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

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

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

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

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

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

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

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

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

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

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

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

Tarjan & van Wyk [1988]:
Summary

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

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

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

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

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

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

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

**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

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

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

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

**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

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

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

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

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

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

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

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

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

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

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]: $O(n)$

Kirkpatrick, Klawe, Tarjan [1992]: $O(n \log^* n)$
### Summary

<table>
<thead>
<tr>
<th>Lemma.</th>
<th>A simple polygon with $n$ vertices can be subdivided into $y$-monotone polygons in $O(n \log n)$ time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>old</td>
<td></td>
</tr>
<tr>
<td>Lemma.</td>
<td>A $y$-monotone polygon with $n$ vertices can be triangulated in $O(n)$ time.</td>
</tr>
<tr>
<td>new</td>
<td></td>
</tr>
<tr>
<td>Lemma.</td>
<td>Subdividing a simple polygon with $n$ vertices by drawing $d$ (pairwise non-crossing) diagonals yields $d + 1$ simple polygons of total complexity $O(n)$.</td>
</tr>
<tr>
<td>homework</td>
<td></td>
</tr>
<tr>
<td>Theorem.</td>
<td>A simple polygon with $n$ vertices can be triangulated in $O(n \log n)$ time.</td>
</tr>
<tr>
<td>Is this it?</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tarjan &amp; van Wyk [1988]: $O(n \log \log n)$</td>
</tr>
<tr>
<td></td>
<td>Clarkson, Tarjan, van Wyk [1989]: $O(n \log^* n)$</td>
</tr>
<tr>
<td></td>
<td>Chazelle [1991]: $O(n)$</td>
</tr>
<tr>
<td></td>
<td>Kirkpatrick, Klawe, Tarjan [1992]: $O(n \log \log n)$</td>
</tr>
<tr>
<td></td>
<td>Seidel [1991]: randomized</td>
</tr>
</tbody>
</table>