Computational Geometry

Triangulating Polygons
or
Guarding Art Galleries

Lecture #2
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)...

![Diagram of a simple polygon](image)
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 $qc \subseteq P$. 
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

*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 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 \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 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)... 

![Diagram showing a complex polygon and a camera's viewpoint](image)

**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 $qc \subseteq P$.

**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)...

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 $qc \subseteq P$.

**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.

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

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

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

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

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

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

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

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

\[ \begin{array}{c} \text{v} \\ \text{w} \\ \text{u} \\ \end{array} \]
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.

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

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

- For \( n = 3 \): One triangle
- For \( 3, \ldots, n - 1 \rightarrow n \):
  - Select vertex \( x \) 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, ..., $n - 1 \rightarrow n$:

- $x$ 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.

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

3, \ldots, \( n - 1 \rightarrow 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.

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

\[
\begin{align*}
n &= 3: & 1 \text{ triangle} & \checkmark \\
3, \ldots, n - 1 & \rightarrow n: & \\
\text{3 vtcs} & \Rightarrow 1 \text{ triangle} & \\
\text{n-1 vtcs} & \Rightarrow n-3 \text{ triangles}
\end{align*}
\]
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.

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

3, \ldots, \( n - 1 \) → \( n \):

- 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.

$n = 3$: \begin{tikzpicture}
\fill[blue!30] (0,0) -- (1,1) -- (1,-1) -- cycle;
\end{tikzpicture} 1 \text{ triangle} \checkmark

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

3 vtcs $\Rightarrow$ 1 triangle

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

$\Rightarrow$ $n - 2$ triangles

$x$ furthest from $uw$
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

$x$ furthest from $uw$

$3, \ldots, n - 1 \rightarrow n$: 3 vtcs $\Rightarrow$ 1 triangle

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

$n - 1$ 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.

$n = 3$: 1 triangle

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

$x$ furthest from $uw$

$3$ vtcs $\Rightarrow$ 1 triangle

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

$\Rightarrow$ $n - 2$ triangles

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

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

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

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

- 3 \( \text{vtcs} \) \( \Rightarrow \) 1 triangle
- \( n - 1 \) \( \text{vtcs} \) \( \Rightarrow \) \( n - 3 \) triangles
- \( \Rightarrow n - 2 \) triangles

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

- \( m \) \( \text{vtcs} \) \( \Rightarrow m - 2 \) triangles
- \( n - m + 2 \) \( \text{vtcs} \) \( \Rightarrow n - m \) triangles
- \( \Rightarrow n - 2 \) triangles

\[ \checkmark \]
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

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

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

**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.

---

[Chvátal ’75]

[dBCvKO’08]
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

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

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

3-color the vtcs

[Chvátal '75]
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
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
Theorem. For surveilling a simple polygon with \( n \) vertices, \( \lceil n/3 \rceil \) cameras are sometimes necessary and always sufficient.

3-color the vtc\( s \)
dual tree
The Art Gallery Theorem

**Theorem.** For surveilling a simple polygon with $n$ vertices, $\left\lfloor n/3 \right\rfloor$ 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.

3-color the vtc's

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
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
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

[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 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.
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 vtics

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
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
The Art Gallery Theorem

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

1. 3-color the vtcs
2. Traverse the dual tree
3. Pick “smallest” color
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

[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

**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:**
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, $\lfloor n/3 \rfloor$ 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

**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

**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

**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

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
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 → \( n'' \) triangles
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
\( O(n \log n) \)
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:**

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

\[ O(n \log n) \rightarrow O(n') \]
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
  - \( O(n \log n) \)
  - \( O(n') \)

**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 → “nice” pieces, \( n' \) vtc → \( n'' \) triangles

\( O(n \log n) \) \hspace{1cm} \( O(n') \)

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:**

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

\[ O(n \log n) \quad O(n') \]

**Definition.** A polygon \( P \) is \( y \)-monotone if, for any horizontal line \( \ell \), \( \ell \cap P \) is connected.
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: $O(n^2)$

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
[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: \( O(n^2) \)

**Faster triangulation in two steps:**

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

**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 $P$

– \textit{turn} vertices:
  vertical component of walking direction changes

– \textit{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

\[
\text{if } \alpha < 180^\circ
\]
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
    - *split* vertex
    - *end* vertex

- *regular* vertices

- if $\alpha < 180^\circ$
- if $\beta > 180^\circ$
- if $\gamma < 180^\circ$
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

If \( \alpha < 180^\circ \)

If \( \beta > 180^\circ \)

If \( \gamma < 180^\circ \)

If \( \delta > 180^\circ \)
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 if $\alpha < 180^\circ$ if $\beta > 180^\circ$ if $\gamma < 180^\circ$ if $\delta > 180^\circ$

$P$ has neither split vertices nor merge vertices.
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'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.

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's.

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 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{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 vertices.

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:
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 $\text{left}(w) = \text{left}(v)$.

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 $w^*$ 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 $w^*$ 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 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 \( \text{left}(w) = \text{left}(v) \).

Think of a sweep-line algorithm:

Connect \( v \) to \( \text{helper}(\text{left}(v)) \).
An Algorithm

2) Treating merge vertices

(left(v), ℓ)
An Algorithm

2) Treating merge vertices
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

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

2) Treating merge vertices

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

\{doubly-connected edge list: data structure for planar subdivisions\}
An Algorithm

2) Treating merge vertices

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

\[
\begin{align*}
(x, y) \prec (x', y') & : \iff \\
y > y' & \lor (y = y' \land x < x')
\end{align*}
\]
An Algorithm

2) Treating merge vertices

```
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 ∈ start, split, end, merge, regular
  handleVertex_{type}(v)
return DCEL D
```

```
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

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

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

handleVertex_{\text{merge}}(vertex \( v \))

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

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

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

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

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

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

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

helper(e') \leftarrow v

return DCEL \( D \)
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
\[ v \leftarrow Q.\text{extractMax()} \]
\[ \text{type } \leftarrow \text{type of vertex } v \]
\[ \text{handleVertex}_{\text{merge}}(v) \]

return \( \text{DCEL } D \)

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

An Algorithm

2) Treating merge vertices

```
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
    handleVertex_{merge}(v)
return DCEL D
```

```
handleVertex_{merge}(vertex v)
e ← edge following v ccw
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)
\[ \mathcal{D} \leftarrow \text{DCEL}(V(P), E(P)) \]
\[ Q \leftarrow \text{priority queue on } V(P) \]
\[ T \leftarrow \text{empty bin. search tree} \]
while \( Q \neq \emptyset \) do
\[ v \leftarrow Q.\text{extractMax}() \]
\[ \text{type } \leftarrow \text{type of vertex } v \]
\[ \text{handleVertex}_{\text{merge}}(v) \]
return \( \text{DCEL } \mathcal{D} \)

handleVertex_{\text{merge}}(vertex v)
\[ e \leftarrow \text{edge following } v \text{ ccw} \]
if helper(e) merge vtx then
\[ \mathcal{D}.\text{insert}(\text{diag}(v, \text{helper}(e))) \]
\[ T.\text{delete}(e) \]
e′ ← T.edgeLeftOf(v)
if helper(e′) merge vtx then
\[ \mathcal{D}.\text{insert}(\text{diag}(v, \text{helper}(e'))) \]
helper(e′) ← v

while \( Q \neq \emptyset \) do
v ← Q.extractMax()
type ← type of vertex v
handleVertex_{\text{type}}(v)
return \( \text{DCEL } \mathcal{D} \)
An Algorithm

2) Treating merge vertices

\begin{align*}
\text{makeMonotone}(\text{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} \\
\text{while } Q \neq \emptyset \text{ do} & \\
& \quad v \leftarrow Q.\text{extractMax}() \\
& \quad \text{type } \leftarrow \text{type of vertex } v \\
& \quad \text{handleVertex}_{\text{merge}}(v) \\
\text{return } \text{DCEL } D
\end{align*}

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

2) Treating merge vertices

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

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

2) Treating merge vertices

\textbf{makeMonotone}(polygon P)

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

\textbf{handleVertex}_{\text{merge}}(vertex v)

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

2) Treating merge vertices

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

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

2) Treating merge vertices

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

\[
\text{while } Q \neq \emptyset \text{ do}
\]
\[
\quad v \leftarrow Q.\text{extractMax()}
\]
\[
\quad \text{type } \leftarrow \text{type of vertex } v
\]
\[
\quad \text{handleVertex}_{\text{merge}}(v)
\]
\[
\text{return } \text{DCEL } D
\]

\[
\text{handleVertex}_{\text{merge}}(\text{vertex } v)
\]
\[
\quad e \leftarrow \text{edge following } v \text{ ccw}
\]
\[
\quad \text{if } \text{helper}(e) \text{ merge vtx then}
\]
\[
\quad \quad D.\text{insert(diag}(v, \text{helper}(e)))
\]
\[
\quad T.\text{delete}(e)
\]
\[
\quad e' \leftarrow T.\text{edgeLeftOf}(v)
\]
\[
\quad \text{if } \text{helper}(e') \text{ merge vtx then}
\]
\[
\quad \quad D.\text{insert(diag}(v, \text{helper}(e')))
\]
\[
\quad \quad \text{helper}(e') \leftarrow v
\]
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
\[ v \leftarrow Q.\text{extractMax()} \]
\[ \text{type} \leftarrow \text{type of vertex } v \]
handleVertex\text{type}(v)
\]
return DCEL \( D \)

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

![Diagram of a y-Monotone Polygon](image_url)
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
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?**

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*.
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
- 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**
- **Convex VTC**

![Diagram of triangulation process with labeled angles and VTC types]
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 $P$ > $180^\circ$
- chains of reflex vtc
- convex vtc
- 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!
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 → 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**

`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 → 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

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
\[ \text{if } u_j \text{ and } S.top() \text{ lie on different chains then} \]
\[ \quad \text{while not } S.empty() \text{ do} \]
\[ \quad \quad v \leftarrow S.pop() \]
\[ \quad \quad \text{if not } S.empty() \text{ then draw diag. } (u_j, v) \]
\[ \text{else} \]
\[ \quad \text{draw diagonals from } u_n \text{ to all vtc on } S \text{ except first and last one} \]
Algorithm

\textbf{TriangulateMonotonePolygon}(\text{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}
  \textbf{if} $u_j$ and $S$.top() lie on different chains \textbf{then}
    \textbf{while} not $S$.empty() \textbf{do}
      \textbf{if} not $S$.empty() \textbf{then} draw diag. $(u_j, v)$
      $v \leftarrow S$.pop()
  \textbf{else}
\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($u_j$)
  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
    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

end for

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

\textbf{TriangulateMonotonePolygon}(Polygon \textit{P} as circular vertex list)
merge left and right chain \rightarrow seq. \textit{u}_1, \ldots, \textit{u}_n with \textit{y}_1 \geq \ldots \geq \textit{y}_n
Stack \textit{S}; \textit{S}.push(\textit{u}_1); \textit{S}.push(\textit{u}_2)
\textbf{for} \textit{j} \leftarrow 3 \textbf{ to } \textit{n} - 1 \textbf{ do}
\hspace{1cm} \textbf{if} \textit{u}_j \text{ and } \textit{S}.top() \text{ lie on different chains} \textbf{then}
\hspace{2cm} \textbf{while} \textbf{not} \ \textit{S}.empty() \ \textbf{do}
\hspace{3cm} \textit{v} \leftarrow \textit{S}.pop()
\hspace{4cm} \textbf{if} \textbf{not} \ \textit{S}.empty() \ \textbf{then} \ \text{draw diag. } (\textit{u}_j, \textit{v})
\hspace{2cm} \textbf{else}
\hspace{3cm} \textit{v} \leftarrow \textit{S}.pop()
\hspace{4cm} \textbf{while} \not\textbf{S}.empty() \text{ and } \textit{u}_j \text{ sees } \textit{S}.top() \textbf{do}
\hspace{5cm} \textit{v} \leftarrow \textit{S}.pop()
\hspace{6cm} \text{draw diagonal } (\textit{u}_j, \textit{v})
\hspace{4cm} \textit{S}.push(\textit{v})
\hspace{4cm} \textit{S}.push(\textit{u}_j)
\hspace{2cm} \text{draw diagonals from } \textit{u}_n \text{ to all vtc on } \text{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
    $S$.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 → 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 → 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
    \( \)
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 \( S; S.\text{push}(u_1); S.\text{push}(u_2) \) 
\textbf{for} \( j \leftarrow 3 \) \textbf{to} \( n - 1 \) \textbf{do} 
\hspace{1em} \textbf{if} \( u_j \) and \( S.\text{top}() \) lie on different chains \textbf{then} 
\hspace{2em} \textbf{while} \textbf{not} \( S.\text{empty}() \) \textbf{do} 
\hspace{3em} \( v \leftarrow S.\text{pop}() \) 
\hspace{4em} \textbf{if} \textbf{not} \( S.\text{empty}() \) \textbf{then} draw diag. \( (u_j, v) \) 
\hspace{5em} \( S.\text{push}(u_{j-1}); S.\text{push}(u_j) \) 
\hspace{1em} \textbf{else} 
\hspace{2em} \( v \leftarrow S.\text{pop}() \) 
\hspace{1em} 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 \( (u_j, v) \)
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.pop(u_1); \ S.pop(u_2) \\
\textbf{for} \ j \leftarrow 3 \ \textbf{to} \ n - 1 \ \textbf{do} \\
\quad \textbf{if} \ u_j \ \text{and} \ S.top() \ \text{lie on different chains} \ \textbf{then} \\
\quad \quad \textbf{while} \ \text{not} \ S.empty() \ \textbf{do} \\
\quad \quad \quad v \leftarrow S.pop() \\
\quad \quad \quad \textbf{if} \ \text{not} \ S.empty() \ \textbf{then} \ \text{draw} \ \text{diag.} \ (u_j, v) \\
\quad \quad S.pop(u_{j-1}); \ S.push(u_j) \\
\quad \textbf{else} \\
\quad \quad v \leftarrow S.pop() \\
\quad \quad \textbf{while} \ \text{not} \ S.empty() \ \text{and} \ u_j \ \text{sees} \ S.top() \ \textbf{do} \\
\quad \quad \quad v \leftarrow S.pop() \\
\quad \quad \quad \text{draw} \ \text{diagonal} \ (u_j, v) \\
\quad S.push(v); \ S.push(u_j) \\
\textbf{draw} \ \text{diagonals} \ \text{from} \ u_n \ \text{to} \ \text{all} \ \text{vtc} \ \text{on} \ S \ \text{except} \ \text{first} \ \text{and} \ \text{last} \ \text{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 $\rightarrow$ 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 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
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 \( (u_j, v) \)
  end if
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
            \( v \) ← \( S \).pop()
            if not \( S \).empty() then draw diagonal \( (u_j, v) \)
            \( S \).push\( (u_{j-1}) \); \( S \).push\( (u_j) \)
    else
        \( v \) ← \( S \).pop()
        while not \( S \).empty() and \( u_j \) sees \( S \).top() do
            \( v \) ← \( S \).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 \( \rightarrow \) 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. \((u_j, v)\) \n
\( S.push(u_{j-1}); S.push(u_j) \)

else

\( v \leftarrow S.pop() \)

while not \( S.empty() \) and \( u_j \) sees \( S.top() \) do

\( v \leftarrow S.pop() \)

draw diagonal \((u_j, v)\) \n
\( S.push(v); S.push(u_j) \)

draw diagonals from \( u_n \) to all vtc on \( S \) except first and last one
Algorithm

\textbf{TriangulateMonotonePolygon}(Polygon \ P \ as \ circular \ vertex \ list) \\
merge \ left \ and \ right \ chain \ \rightarrow \ \text{seq.} \ u_1, \ldots, u_n \ \text{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 \quad \textbf{while} \ \textbf{not} \ S.\text{empty}() \ \textbf{do} \\
\quad \quad \quad \quad v \leftarrow S.\pop() \\
\quad \quad \quad \quad \textbf{if} \ \textbf{not} \ S.\text{empty}() \ \textbf{then} \ \text{draw \ diagonal} \ \text{\((u_j, v)\)} \\
\quad \quad \quad \quad S.\push(u_{j-1}); \ S.\push(u_j) \\
\quad \textbf{else} \\
\quad \quad v \leftarrow S.\pop() \\
\quad \quad \textbf{while} \ \textbf{not} \ S.\text{empty}() \ \text{and} \ u_j \ \text{sees} \ S.\text{top}() \ \textbf{do} \\
\quad \quad \quad v \leftarrow S.\pop() \\
\quad \quad \quad \text{draw \ diagonal} \ \text{\((u_j, v)\)} \\
\quad \quad \quad S.\push(v); \ S.\push(u_j) \\
\text{draw \ diagonals \ from} \ u_n \ \text{to \ all \ vtc \ on} \ S \ \text{except \ first}
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.

Running time?
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
  \( \text{if } u_j \text{ and } S.top() \text{ lie on different chains } \) then
    while not S.empty() do
      \( v \leftarrow S.pop() \)
      \( \text{if not } S.empty() \text{ then draw diag. } (u_j, v) \)
      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

Running time? \( \Theta(n) \)
Summary

\begin{itemize}
\item \textit{n-vtx polygon} $\rightarrow$ \textit{“nice” pieces, n’ vtc} $\rightarrow$ \textit{n’’ triangles}
\end{itemize}

\text{\textit{O}(n \log n)} $\rightarrow$ \textit{O(n’)}
Summary

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 \( O(n) \).
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?
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]:

<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.

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

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]:
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]: \( O(n) \)
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]: \( O(n) \)  
Kirkpatrick, Klawe, Tarjan [1992]
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]: $O(n)$
Kirkpatrick, Klawe, Tarjan [1992]: $O(n \log n)$
Seidel [1991]: randomized