Computational Geometry
Winter term 2014/15

Triangulating Polygons
or
Guarding Art Galleries

Lecture #3

Prof. Dr. Alexander Wolff
Chair for Informatics I
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 $P$ with a point $C$]
Guarding an Art Gallery

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

**Observation.** Camera $c$ "sees" a star-shaped region
Guarding an Art Gallery

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

**Observation.** Camera \( c \) “sees” a star-shaped region

**Definition.** A pt \( q \in P \) is *visible* from \( c \in P \) if \( \overline{qc} \subseteq P \).
Guarding an Art Gallery

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

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

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

**Aim:** Use few cameras!
Guarding an Art Gallery

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

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

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

**Aim:** Use few cameras!
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!
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!
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!
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!
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)...

\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!

\textbf{Theorem.} 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!

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

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

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

[2 minutes]
The Art Gallery Theorem

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

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

[2 minutes]
The Art Gallery Theorem

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.

[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.

To do: Find algorithm for triangulating a simple polygon!
The Art Gallery Theorem [Chvátal '75]

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

**To do:** Find algorithm for triangulating a simple polygon!

**Brute force:**
The Art Gallery Theorem

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

**To do:** Find algorithm for triangulating a simple polygon!

**Brute force:** follow existence proof, using recursion
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 algorithm 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 algorithm for triangulating a simple polygon!

**Brute force:** follow existence proof, using recursion running time: $O(n^2)$
The Art Gallery Theorem [Chvátal ’75]

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

**To do:** Find algorithm 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 algorithm for triangulating a simple polygon!

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

**Faster triangulation in two steps:**
The Art Gallery Theorem [Chvátal ’75]

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

**To do:** Find algorithm 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 \( \rightarrow \)
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 algorithm 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 algorithm 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} & \quad \rightarrow \quad \text{“nice” pieces, n’ vtc} \quad \rightarrow \quad \text{n’’ triangles} \\
& \quad \rightarrow \quad \text{O(n log n)}
\end{align*}
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 algorithm for triangulating a simple polygon!

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

**Faster triangulation in two steps:**

\[
\begin{align*}
\text{n-vtx polygon} & \quad \rightarrow \quad \text{“nice” pieces, n’ vtc} \quad \rightarrow \quad \text{n’’ triangles} \\
O(n \log n) & \qquad \rightarrow \quad O(n')
\end{align*}
\]
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 algorithm for triangulating a simple polygon!

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

**Faster triangulation in two steps:**

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

**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 algorithm for triangulating a simple polygon!

**Brute force:** follow existence proof, using recursion

running time: \( O(n^2) \)

**Faster triangulation in two steps:**

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

**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, $\left\lfloor n/3 \right\rfloor$ cameras are sometimes necessary and always sufficient. [Chvátal ’75]

To do: Find algorithm 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 algorithm 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 algorithm 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.
Partitioning a Polygon into Monotone Pieces

Idea: Classify vertices of given simple polygon $P$
Partitioning a Polygon into Monotone Pieces

Idea: Classify vertices of given simple polygon $P$
  – turn vertices:
    – regular vertices
Partitioning a Polygon into Monotone Pieces

**Idea:** Classify vertices of given simple polygon $P$

- *turn* vertices:
  
  - vertical component of walking direction changes

- *regular* vertices
Partitioning a Polygon into Monotone Pieces

Idea: Classify vertices of given simple polygon $P$

- *turn* vertices:
  vertical component of walking direction changes
  
  \[ \text{start vertex} \]
  
  if $\alpha < 180^\circ$

- *regular* vertices
Partitioning a Polygon into Monotone Pieces

Idea: Classify vertices of given simple polygon $P$

- **turn** vertices:
  - vertical component of walking direction changes
  
  \[ \begin{align*}
  &\text{start vertex} \\
  &\text{split vertex}
  \end{align*} \]

  - **regular** vertices

\[ \begin{align*}
  &\text{if } \alpha < 180^\circ \\
  &\text{if } \beta > 180^\circ
  \end{align*} \]
Partitioning 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

\[
\begin{align*}
\text{if } \alpha < 180^\circ \\
\text{if } \beta > 180^\circ \\
\text{if } \gamma < 180^\circ
\end{align*}
\]
Partitioning 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 $\delta > 180^\circ$
Partitioning 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 vertices.
Towards an Algorithm

**Idea:** Add diagonals to “destroy” split and merge vertices.
Towards an Algorithm

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

**Problem:** Diagonals must not cross
Towards an Algorithm

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

Problem: Diagonals must not cross
- each other
- edges of \( P \)
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
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
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
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
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
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 $\text{left}(w) = \text{left}(v)$. 
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 $\text{left}(w) = \text{left}(v)$.
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 $\text{left}(w) = \text{left}(v)$.
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 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 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 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:

Connect $v$ to $\text{helper}(\text{left}(v))$. 
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 left($w$) = left($v$).

Think of a sweep-line algorithm:

Connect $v$ to helper(left($v$)).
An Algorithm

2) Treating merge vertices

$\ell \left( v \right)$
An Algorithm

2) Treating merge vertices
An Algorithm

2) Treating merge vertices
An Algorithm

2) Treating merge vertices
An Algorithm

2) Treating merge vertices
An Algorithm

2) Treating merge vertices
An Algorithm

2) Treating merge vertices

\texttt{makeMonotone}(\text{polygon } P)

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

2) Treating merge vertices

makeMonotone(polygon \( P \))

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

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

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

2) Treating merge vertices

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

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

2) Treating merge vertices

\[ \text{makeMonotone}(\text{polygon } P) \]

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

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

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

\[ \text{while } Q \neq \emptyset \text{ do} \]

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

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

\[ \text{handleTypeVertex}(v) \]

\[ \text{return } \text{DCEL } \mathcal{D} \]

- \textit{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

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

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

2) Treating merge vertices

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

while \(Q \neq \emptyset\) do

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

return DCEL \(D\)

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

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

2) Treating merge vertices

```
\text{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} \\
\textbf{while } Q \neq \emptyset \textbf{ do} \\
\quad v \leftarrow Q.\text{extractMax()} \\
\quad \text{type} \leftarrow \text{type of vertex } v \\
\quad \text{handleTypeVertex}(v) \\
\textbf{return } \text{DCEL } D
```

```
\text{handleMergeVertex(vertex } v) \\
\quad e \leftarrow \text{edge following } v \text{ cw} \\
\quad \textbf{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 \textbf{if } \text{helper}(e') \text{ merge vtx then} \\
\quad \quad D.\text{insert(diag}(v, \text{helper}(e'))) \\
\quad \text{helper}(e') \leftarrow v
```
An Algorithm

2) Treating merge vertices

\begin{itemize}
\item \textbf{makeMonotone}(polygon \( P \))
\item \( \mathcal{D} \leftarrow \text{DCEL}(V(P), E(P)) \)
\item \( Q \leftarrow \) priority queue on \( V(P) \)
\item \( \mathcal{T} \leftarrow \) empty bin. search tree
\item \textbf{while} \( Q \neq \emptyset \) \textbf{do}
\item \hspace{1em} \( v \leftarrow Q.\text{extractMax}() \)
\item \hspace{1em} type \leftarrow \text{type of vertex } v
\item \hspace{1em} \textbf{handleTypeVertex}(v)
\item \textbf{return} \text{DCEL } \mathcal{D}
\end{itemize}

\begin{itemize}
\item \textbf{handleMergeVertex}(vertex \( v \))
\item \( e \leftarrow \) edge following \( v \) cw
\item \textbf{if} helper(\( e \)) merge vtx \textbf{then}
\item \hspace{1em} \( \mathcal{D}.\text{insert(diag}(v, \text{helper}(e))) \)
\item \( \mathcal{T}.\text{delete}(e) \)
\item \( e' \leftarrow \mathcal{T}.\text{edgeLeftOf}(v) \)
\item \textbf{if} helper(\( e' \)) merge vtx \textbf{then}
\item \hspace{1em} \( \mathcal{D}.\text{insert(diag}(v, \text{helper}(e'))) \)
\item \( \text{helper}(e') \leftarrow v \)
\end{itemize}
An Algorithm

2) Treating merge vertices

\textbf{makeMonotone}(polygon \( P \))

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

\textbf{while} \( \mathcal{Q} \neq \emptyset \) \textbf{do}

\begin{align*}
\mathcal{Q}.\text{extractMax}() & \leftarrow v \\
\text{type} & \leftarrow \text{type of vertex } v \\
\text{handleTypeVertex} & (v)
\end{align*}

\textbf{return} \text{DCEL } \mathcal{D}

\textbf{handleMergeVertex}(vertex \( v \))

\begin{align*}
e & \leftarrow \text{edge following } v \text{ cw} \\
\text{if} \text{ helper}(e) \text{ merge vtx} \text{ then} \\
\mathcal{D}.\text{insert}(\text{diag}(v, \text{helper}(e))) \\
\mathcal{T}.\text{delete}(e) \\
e' & \leftarrow \mathcal{T}.\text{edgeLeftOf}(v) \\
\text{if} \text{ helper}(e') \text{ merge vtx} \text{ then} \\
\mathcal{D}.\text{insert}(\text{diag}(v, \text{helper}(e')))
\end{align*}

\text{helper}(e') & \leftarrow v
An Algorithm

2) Treating merge vertices

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

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

2) Treating merge vertices

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

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

\[ \text{return } \text{DCEL } D \]

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

2) Treating merge vertices

\textbf{makeMonotone}(polygon }P\textbf{)}
\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} \\
\textbf{while } & Q \neq \emptyset \textbf{ do} \\
& v \leftarrow Q.\text{extractMax()} \\
& \text{type } \leftarrow \text{type of vertex } v \\
& \text{handleTypeVertex}(v) \\
\textbf{return } & \text{DCEL } \mathcal{D}
\end{align*}

\textbf{handleMergeVertex}(vertex }v\textbf{)}
\begin{align*}
ed & \leftarrow \text{edge following } v \text{ cw} \\
\textbf{if } & \text{helper}(e) \text{ merge vtx then} \\
& \mathcal{D}.\text{insert(diag}(v, \text{helper}(e))\text{))} \\
& \mathcal{T}.\text{delete}(e) \\
& e' \leftarrow \mathcal{T}.\text{edgeLeftOf}(v) \\
\textbf{if } & \text{helper}(e') \text{ merge vtx then} \\
& \mathcal{D}.\text{insert(diag}(v, \text{helper}(e'))\text{))} \\
& \text{helper}(e') \leftarrow v
\end{align*}
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} \\
\text{while } Q \neq \emptyset \text{ do} \\
&\quad \begin{cases} 
\quad v \leftarrow Q.\text{extractMax}() \\
\quad \text{type } \leftarrow \text{type of vertex } v \\
\quad \text{handleTypeVertex}(v) 
\end{cases} \\
\text{return } \text{DCEL } \mathcal{D}
\end{align*}

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

2) Treating merge vertices

\begin{align*}
\text{makeMonotone}(\text{polygon } P) & \quad \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 } \text{Q} \neq \emptyset \text{ do} & \\
& \quad v \leftarrow \text{Q}.\text{extractMax}() \\
& \quad \text{type} \leftarrow \text{type of vertex } v \\
& \quad \text{handleTypeVertex}(v) \\
\text{return } \text{DCEL } D
\end{align*}

\begin{align*}
\text{handleMergeVertex}(\text{vertex } v) & \\
\text{e} & \leftarrow \text{edge following } v \text{ cw} \\
\text{if } \text{helper(e)} \text{ merge vtx then} & \\
& \quad \text{D}.\text{insert(} \text{diag}(v, \text{helper(e)})\text{)} \\
& \quad \text{T}.\text{delete(e)} \\
& \quad e' \leftarrow \text{T}.\text{edgeLeftOf}(v) \\
& \quad \text{if } \text{helper(e')} \text{ merge vtx then} \\
& \quad \quad \text{D}.\text{insert(} \text{diag}(v, \text{helper(e')})\text{)} \\
& \quad \quad \text{helper(e')} \leftarrow v
\end{align*}
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
**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 $P > 180^\circ$
- reflex vtc
- convex 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 \( P \) \( > 180^\circ \)
- reflex vtc
- convex vtc.

chains of reflex vtc

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 $\rightarrow$ sequence $u_1, \ldots, u_n$ with $y_1 \geq \cdots \geq y_n$
Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
Algorithm

`TriangulateMonotonePolygon(Polygon P as circular vertex list)`
merge left and right chain → sequence \( u_1, \ldots, u_n \) with \( y_1 \geq \cdots \geq y_n \)
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 $\rightarrow$ sequence $u_1, \ldots, u_n$ with $y_1 \geq \cdots \geq y_n$

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 → sequence \( u_1, \ldots, u_n \) with \( y_1 \geq \cdots \geq y_n \)
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 \( \rightarrow \) sequence \( u_1, \ldots, u_n \) with \( y_1 \geq \cdots \geq y_n \)

Stack \( S \); \( S.push(u_1); S.push(u_2) \)

for \( j \leftarrow 3 \text{ to } n - 1 \) 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 \( \rightarrow \) sequence \( u_1, \ldots, u_n \) with \( y_1 \geq \cdots \geq y_n \)
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 → sequence \( u_1, \ldots, u_n \) with \( y_1 \geq \cdots \geq y_n \)
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

\textbf{TriangulateMonotonePolygon}(Polygon \ P \ as \ circular \ vertex \ list)

merge left and right chain \rightarrow \ sequence \ u_1, \ldots, u_n \ with \ y_1 \geq \cdots \geq y_n

Stack \ S; \ S.push(u_1); \ S.push(u_2)

for \ j \leftarrow 3 \ to \ n - 1 \ do

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

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

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

\hspace{3em} if \ not \ S.empty() \ then \ draw \ diag. \ (u_j, v)

\hspace{1em} else

\hspace{2em} 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 \( \rightarrow \) sequence \( u_1, \ldots, u_n \) with \( y_1 \geq \cdots \geq y_n \)

Stack \( S \); \( S.push(u_1) \); \( S.push(u_2) \)

**for** \( j \leftarrow 3 \) **to** \( n - 1 \) **do**

\[
\text{if } u_j \text{ and } S.top() \text{ lie on different chains then}
\]

\[
\text{while not } S.empty() \text{ do}
\]

\[
\quad v \leftarrow S.pop()
\]

\[
\quad \text{if not } S.empty() \text{ then draw diag. } (u_j, v)
\]

\[
\text{else}
\]

\[
\quad \text{draw diagonals from } u_n \text{ to all vtc on } S \text{ except first and last one}
\]
**Algorithm**

**TriangulateMonotonePolygon** (Polygon $P$ as circular vertex list)

merge left and right chain $\rightarrow$ sequence $u_1, \ldots, u_n$ with $y_1 \geq \cdots \geq y_n$

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 $\rightarrow$ sequence $u_1, \ldots, u_n$ with $y_1 \geq \cdots \geq y_n$

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

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

if not $S$.empty() then draw diag. $(u_j, v)$

$\quad S$.push($u_{j-1}$); $S$.push($u_j$)

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 \( \rightarrow \) sequence \( u_1, \ldots, u_n \) with \( y_1 \geq \cdots \geq y_n \)
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
    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 $\rightarrow$ sequence $u_1, \ldots, u_n$ with $y_1 \geq \cdots \geq y_n$

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

  draw diagonals from $u_n$ to all vtc on $S$ except first and last one

$u_j$, $u_{j-1}$
Algorithm

**TriangulateMonotonePolygon** (Polygon $P$ as circular vertex list)
merge left and right chain $\rightarrow$ sequence $u_1, \ldots, u_n$ with $y_1 \geq \cdots \geq y_n$

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()

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 $\rightarrow$ sequence $u_1, \ldots, u_n$ with $y_1 \geq \cdots \geq y_n$
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 \( \rightarrow \) sequence \( u_1, \ldots, u_n \) with \( y_1 \geq \cdots \geq y_n \)

Stack \( S; S.\text{push}(u_1); S.\text{push}(u_2) \)

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

\( \text{if } u_j \text{ and } S.\text{top()} \text{ lie on different chains then} \)

\( \text{while not } S.\text{empty()} \text{ do} \)

\( \quad v \leftarrow S.\text{pop()} \)

\( \quad \text{if not } S.\text{empty()} \text{ then draw diag. } (u_j, v) \)

\( S.\text{push}(u_{j-1}); S.\text{push}(u_j) \)

\( \text{else} \)

\( \quad v \leftarrow S.\text{pop()} \)

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

\( \quad v \leftarrow S.\text{pop()} \)

\( \quad \text{draw diagonal } (u_j, v) \)

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 $\rightarrow$ sequence $u_1, \ldots, u_n$ with $y_1 \geq \cdots \geq y_n$

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 $\rightarrow$ sequence $u_1, \ldots, u_n$ with $y_1 \geq \cdots \geq y_n$
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 $\rightarrow$ sequence $u_1, \ldots, u_n$ with $y_1 \geq \cdots \geq y_n$
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 $\rightarrow$ sequence $u_1, \ldots, u_n$ with $y_1 \geq \cdots \geq y_n$
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 \( \rightarrow \) sequence \( u_1, \ldots, u_n \) with \( y_1 \geq \cdots \geq y_n \)

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 $\rightarrow$ sequence $u_1, \ldots, u_n$ with $y_1 \geq \cdots \geq y_n$
Stack $S$; $S$.push($u_1$); $S$.push($u_2$)
for $j \leftarrow 3$ to $n - 1$ do
  if $u_j$ and $S$.top() lie on different chains then
    while not $S$.empty() do
      $v \leftarrow S$.pop()
      if not $S$.empty() then draw diag. $(u_j, v)$
      $S$.push($u_{j-1}$); $S$.push($u_j$)
  else
    $v \leftarrow S$.pop()
    while not $S$.empty() and $u_j$ sees $S$.top() do
      $v \leftarrow S$.pop()
      draw diagonal $(u_j, v)$
      $S$.push($v$); $S$.push($u_j$)
Algorithm

```
TriangulateMonotonePolygon(Polygon P as circular vertex list)
merge left and right chain \rightarrow sequence u_1, \ldots, u_n with y_1 \geq \cdots \geq y_n
Stack S; S.push(u_1); S.push(u_2)
for j ← 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 diag. (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)
        S.push(v); S.push(u_j)
```

---

**Diagram:**
- **Algorithm Flow:**
  - Start with two chains: left and right.
  - Merge them into a sequence of vertices with increasing y-coordinates.
  - Use a Stack `S` to manage the vertices.
  - Push the first two vertices `u_1` and `u_2` into `S`.
  - For each vertex `u_j` (from 3 to `n-1`):
    - If `u_j` and the top of `S` are on different chains:
      - Pop vertices from `S` until an intersection is found.
      - Draw diagonals as needed.
    - Else (on the same chain):
      - Pop `u_j`.
      - While `u_j` can still see the top of `S`, pop and draw diagonals.
      - Push `v` and `u_j` into `S`.

**Visual Elements:**
- Vertices are represented by black circles.
- Edges are depicted by blue and red arrows, indicating the direction of traversal.
- The diagram illustrates the process of forming triangles by connecting vertices in a monotone polygon sequence.
**Algorithm**

**TriangulateMonotonePolygon** (Polygon $P$ as circular vertex list)

merge left and right chain $\rightarrow$ sequence $u_1, \ldots, u_n$ with $y_1 \geq \cdots \geq y_n$

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 $\rightarrow$ sequence $u_1, \ldots, u_n$ with $y_1 \geq \cdots \geq y_n$

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 → sequence \( u_1, \ldots, u_n \) with \( y_1 \geq \cdots \geq y_n \)
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

Running time?
Algorithm

TriangulateMonotonePolygon(Polygon $P$ as circular vertex list)
merge left and right chain $\rightarrow$ sequence $u_1, \ldots, u_n$ with $y_1 \geq \cdots \geq y_n$
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

Running time? $\Theta(n)$
Summary

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

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

$n$-vtx polygon $\rightarrow$ “nice” pieces $\leftarrow O(n \log n)$

$n' \rightarrow n''$ triangles $O(n')$
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]: $O(n \log \log n)$
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]:
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]: $O(n \log \log n)$
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]: $O(n \log^* n)$
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]: $O(n)$
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