Computational Geometry · Lecture
Polygon Triangulation

Tamara Mchedlidze
3.5.2018
The Art-Gallery-Problem

**Task:** Install a number of cameras in an art gallery so that every part of the gallery is visible to at least one of them.
The Art-Gallery-Problem

**Task:** Install a number of cameras in an art gallery so that every part of the gallery is visible to at least one of them.
The Art-Gallery-Problem

**Task:** Install a number of cameras in an art gallery so that every part of the gallery is visible to at least one of them.

**Assumption:** Art gallery is a simple polygon $P$ with $n$ corners (no self-intersections, no holes)
The Art-Gallery-Problem

**Task:** Install a number of cameras in an art gallery so that every part of the gallery is visible to at least one of them.

**Assumption:** Art gallery is a *simple* polygon $P$ with $n$ corners (no self-intersections, no holes)

**Observation:** each camera observes a star-shaped region
The Art-Gallery-Problem

**Task:** Install a number of cameras in an art gallery so that every part of the gallery is visible to at least one of them.

**Assumption:** Art gallery is a *simple* polygon $P$ with $n$ corners (no self-intersections, no holes)

**Observation:** each camera observes a star-shaped region

**Definition:** Point $p \in P$ is *visible* from $c \in P$ if $cp \in P$
The Art-Gallery-Problem

**Task:** Install a number of cameras in an art gallery so that every part of the gallery is visible to at least one of them.

**Assumption:** Art gallery is a *simple* polygon $P$ with $n$ corners (no self-intersections, no holes)

**Observation:** each camera observes a star-shaped region

**Definition:** Point $p \in P$ is *visible* from $c \in P$ if $cp \in P$

**Goal:** Use as few cameras as possible!
The Art-Gallery-Problem

**Task:** Install a number of cameras in an art gallery so that every part of the gallery is visible to at least one of them.

**Assumption:** Art gallery is a *simple* polygon $P$ with $n$ corners (no self-intersections, no holes)

**Observation:** each camera observes a star-shaped region

**Definition:** Point $p \in P$ is *visible* from $c \in P$ if $cp \in P$

**Goal:** Use as few cameras as possible!

$→$ The number depends on the number of corners $n$ and on the shape of $P$
The Art-Gallery-Problem

**Task:** Install a number of cameras in an art gallery so that every part of the gallery is visible to at least one of them.

**Assumption:** Art gallery is a *simple* polygon $P$ with $n$ corners (no self-intersections, no holes)

**Observation:** each camera observes a star-shaped region

**Definition:** Point $p \in P$ is *visible* from $c \in P$ if $cp \in P$

**Goal:** Use as few cameras as possible!

→ The number depends on the number of corners $n$ and on the shape of $P$
Problem Simplification

Observation: It is easy to guard a triangle
Problem Simplification

**Observation:** It is easy to guard a triangle

**Idea:** Decompose $P$ into triangles and guard each of them
Problem Simplification

**Observation:** It is easy to guard a triangle

**Idea:** Decompose $P$ into triangles and guard each of them

**Theorem 1:** Each simple polygon with $n$ corners admits a triangulation; any such triangulation contains exactly $n - 2$ triangles.
Problem Simplification

Observation: It is easy to guard a triangle

Idea: Decompose $P$ into triangles and guard each of them

Theorem 1: Each simple polygon with $n$ corners admits a triangulation; any such triangulation contains exactly $n - 2$ triangles.

The proof implies a recursive $O(n^2)$-Algorithm!
Problem Simplification

Observation: It is easy to guard a triangle

Idea: Decompose $P$ into triangles and guard each of them

Theorem 1: Each simple polygon with $n$ corners admits a triangulation; any such triangulation contains exactly $n - 2$ triangles.

- $P$ could be guarded by $n - 2$ cameras placed in the triangles
Problem Simplification

Observation: It is easy to guard a triangle

Idea: Decompose $P$ into triangles and guard each of them

Theorem 1: Each simple polygon with $n$ corners admits a triangulation; any such triangulation contains exactly $n - 2$ triangles.

- $P$ could be guarded by $n - 2$ cameras placed in the triangles

Can we do better?
Problem Simplification

Observation: It is easy to guard a triangle

Idea: Decompose $P$ into triangles and guard each of them

Theorem 1: Each simple polygon with $n$ corners admits a triangulation;
any such triangulation contains exactly $n - 2$ triangles.

- $P$ could be guarded by $n - 2$ cameras placed in the triangles
- $P$ can be guarded by $\approx n/2$ cameras placed on the diagonals
Problem Simplification

**Observation:** It is easy to guard a triangle

**Idea:** Decompose $P$ into triangles and guard each of them

**Theorem 1:** Each simple polygon with $n$ corners admits a triangulation; any such triangulation contains exactly $n - 2$ triangles.

- $P$ could be guarded by $n - 2$ cameras placed in the triangles
- $P$ can be guarded by $\approx n/2$ cameras placed on the diagonals
Problem Simplification

**Observation:** It is easy to guard a triangle

**Idea:** Decompose $P$ into triangles and guard each of them

**Theorem 1:** Each simple polygon with $n$ corners admits a triangulation; any such triangulation contains exactly $n - 2$ triangles.

- $P$ could be guarded by $n - 2$ cameras placed in the triangles
- $P$ can be guarded by $\approx n/2$ cameras placed on the diagonals
- $P$ can be observed by even smaller number of cameras placed on the corners
Problem Simplification

**Observation:** It is easy to guard a triangle

**Idea:** Decompose $P$ into triangles and guard each of them

**Theorem 1:** Each simple polygon with $n$ corners admits a triangulation; any such triangulation contains exactly $n - 2$ triangles.

- $P$ could be guarded by $n - 2$ cameras placed in the triangles
- $P$ can be guarded by $\approx n/2$ cameras placed on the diagonals
- $P$ can be observed by even smaller number of cameras placed on the corners
The Art-Gallery-Theorem [Chvátal ’75]

**Theorem 2:** For a simple polygon with $n$ vertices, $\left\lfloor n/3 \right\rfloor$ cameras are sometimes necessary and always sufficient to guard it.
The Art-Gallery-Theorem [Chvátal ’75]

**Theorem 2:** For a simple polygon with \( n \) vertices, \( \lfloor n/3 \rfloor \) cameras are sometimes necessary and always sufficient to guard it.

**Proof:**
- Find a simple polygon with \( n \) corners that requires \( \approx n/3 \) cameras!

Discuss it with your neighbour for 2 minutes
The Art-Gallery-Theorem [Chvátal ’75]

Theorem 2: For a simple polygon with $n$ vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient to guard it.

Proof:

- Find a simple polygon with $n$ corners that requires $\approx n/3$ cameras!
The Art-Gallery-Theorem [Chvátal ’75]

**Theorem 2:** For a simple polygon with \( n \) vertices, \( \lfloor n/3 \rfloor \) cameras are sometimes necessary and always sufficient to guard it.

**Proof:**

- Find a simple polygon with \( n \) corners that requires \( \approx n/3 \) cameras!

- Sufficiency on the board

[Diagram of a simple polygon with cameras placed at strategic points to guard it.]
The Art-Gallery-Theorem [Chvátal ’75]

**Theorem 2:** For a simple polygon with $n$ vertices, $\lfloor n/3 \rfloor$ cameras are sometimes necessary and always sufficient to guard it.

**Proof:**
- Find a simple polygon with $n$ corners that requires $\approx n/3$ cameras!
- Sufficiency on the board

**Conclusion:** Given a triangulation, the $\lfloor n/3 \rfloor$ cameras that guard the polygon can be placed in $O(n)$ time.
The Art-Gallery-Theorem [Chvátal ’75]

Theorem 2: For a simple polygon with \( n \) vertices, \( \lfloor n/3 \rfloor \) cameras are sometimes necessary and always sufficient to guard it.

Proof:

- Find a simple polygon with \( n \) corners that requires \( \approx \frac{n}{3} \) cameras!

- Sufficiency on the board

[Chvátal ’75]

Conclusion: Given a triangulation, the \( \lfloor n/3 \rfloor \) cameras that guard the polygon can be placed in \( O(n) \) time.

Can we do better than \( O(n^2) \) described before?
Proof of Art-Gallery-Theorem: Overview

3-step process:

• Step 1: Decompose $P$ into $y$-monotone polygons

**Definition:** A polygon is $y$-monotone, if for any horizontal line $\ell$, the intersection $\ell \cap P$ is connected.
3-step process:

• Step 1: Decompose $P$ into $y$-monotone polygons

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

The two paths from the topmost to the bottommost point bounding the polygon, never go upward.
Proof of Art-Gallery-Theorem: Overview

3-step process:

• Step 1: Decompose $P$ into $y$-monotone polygons

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

The two paths from the topmost to the bottommost point bounding the polygon, never go upward.
Proof of Art-Gallery-Theorem: Overview

3-step process:

• Step 1: Decompose $P$ into $y$-monotone polygons

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

• Step 2: Triangulate the resulting $y$-monotone polygons
Proof of Art-Gallery-Theorem: Overview

3-step process:

• Step 1: Decompose $P$ into $y$-monotone polygons

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

![Diagram showing y-monotone polygons](image)

• Step 2: Triangulate the resulting $y$-monotone polygons

• Step 3: use DFS to color the vertices of the polygon
Partition into $y$-monotone Polygons

**Idea:** Five different types of vertices
Partition into $y$-monotone Polygons

**Idea:** Five different types of vertices

- *Turn vertices:*
Partition into $y$-monotone Polygons

**Idea:** Five different types of vertices

- *Turn vertices:* vertical change in direction
Partition into $y$-monotone Polygons

**Idea:** Five different types of vertices

- *Turn vertices:* vertical change in direction
  - *start vertices*

\[
\text{if } \alpha < 180^\circ
\]
Partition into $y$-monotone Polygons

**Idea:** Five different types of vertices

- **Turn vertices:** vertical change in direction
  - **start vertices**
  - **split vertices**

$$\begin{align*}
\alpha &< 180^\circ \\
\beta &> 180^\circ
\end{align*}$$
Partition into $y$-monotone Polygons

**Idea:** Five different types of vertices

- **Turn vertices:**
  vertical change in direction
  - **start vertices**
  - **split vertices**
  - **end vertices**

- **if $\alpha < 180^\circ$**
- **if $\beta > 180^\circ$**
- **if $\gamma < 180^\circ$**
Partition into $y$-monotone Polygons

**Idea:** Five different types of vertices

- *Turn vertices:* vertical change in direction
  - *start vertices*
  - *split vertices*
  - *end vertices*
  - *merge vertices*

\[ \begin{align*}
\text{if } \alpha < 180^\circ \\
\text{if } \beta > 180^\circ \\
\text{if } \gamma < 180^\circ \\
\text{if } \delta > 180^\circ 
\end{align*} \]
Partition into $y$-monotone Polygons

**Idea:** Five different types of vertices

- *Turn vertices:* vertical change in direction
  - *start vertices*  
  - *split vertices*  
  - *end vertices*  
  - *merge vertices*  
- *regular vertices*
Lemma 1: A polygon is $y$-monotone if it has no split vertices or merge vertices.
Lemma 1: A polygon is $y$-monotone if it has no split vertices or merge vertices.

Proof: On the blackboard
Characterization

Lemma 1: A polygon is $y$-monotone if it has no split vertices or merge vertices.

Proof: On the blackboard

⇒ We need to eliminate all split and merge vertices by using diagonals
Lemma 1: A polygon is $y$-monotone if it has no split vertices or merge vertices.

Proof: On the blackboard

$\Rightarrow$ We need to eliminate all split and merge vertices by using diagonals

Observation: The diagonals should neither cross the edges of $P$ nor the other diagonals
Ideas for Sweep-Line-Algorithm

1) Diagonals for the split vertices
Ideas for Sweep-Line-Algorithm

1) Diagonals for the split vertices
   • compute for each vertex $v$ its left adjacent edge $\text{left}(v)$ with respect to the horizontal sweep line $\ell$
Ideas for Sweep-Line-Algorithm

1) Diagonals for the split vertices
   - compute for each vertex $v$ its left adjacent edge $\text{left}(v)$ with respect to the horizontal sweep line $\ell$
Ideas for Sweep-Line-Algorithm

1) Diagonals for the split vertices

- compute for each vertex $v$ its left adjacent edge $\text{left}(v)$ with respect to the horizontal sweep line $\ell$
- connect split vertex $v$ to the nearest vertex $w$ above $v$, such that $\text{left}(w) = \text{left}(v)$
Ideas for Sweep-Line-Algorithm

1) Diagonals for the split vertices

- compute for each vertex $v$ its left adjacent edge $\text{left}(v)$ with respect to the horizontal sweep line $\ell$
- connect split vertex $v$ to the nearest vertex $w$ above $v$, such that $\text{left}(w) = \text{left}(v)$
Ideas for Sweep-Line-Algorithm

1) Diagonals for the split vertices

• compute for each vertex $v$ its left adjacent edge $\text{left}(v)$ with respect to the horizontal sweep line $\ell$

• connect split vertex $v$ to the nearest vertex $w$ above $v$, such that $\text{left}(w) = \text{left}(v)$
Ideas for Sweep-Line-Algorithm

1) Diagonals for the split vertices

- compute for each vertex $v$ its left adjacent edge $\text{left}(v)$ with respect to the horizontal sweep line $\ell$
- connect split vertex $v$ to the nearest vertex $w$ above $v$, such that $\text{left}(w) = \text{left}(v)$
- for each edge $e$ save the bottommost vertex $w$ such that $\text{left}(w) = e$; notation $\text{helper}(e) := w$

\[ v \]
\[ \text{left}(v) \]
\[ w \]
\[ \ell \]
\[ e \]
\[ \text{helper}(e) \]
Ideas for Sweep-Line-Algorithm

1) Diagonals for the split vertices

• compute for each vertex \( v \) its left adjacent edge \( \text{left}(v) \) with respect to the horizontal sweep line \( \ell \)

• connect split vertex \( v \) to the nearest vertex \( w \) above \( v \), such that \( \text{left}(w) = \text{left}(v) \)

• for each edge \( e \) save the bottommost vertex \( w \) such that \( \text{left}(w) = e \); notation \( \text{helper}(e) := w \)

• when \( \ell \) passes through a split vertex \( v \), we connect \( v \) with \( \text{helper}(\text{left}(v)) \)
Ideas for Sweep-Line-Algorithm

2) Diagonals for merge vertices
   • when the vertex $v$ is reached, we set $\text{helper}(\text{left}(v)) = v$
Ideas for Sweep-Line-Algorithm

2) Diagonals for merge vertices

- when the vertex $v$ is reached, we set $\text{helper}(\text{left}(v)) = v$
- when we reach a split vertex $v'$ such that $\text{left}(v') = \text{left}(v)$ the diagonal $(v, v')$ is introduced
Ideas for Sweep-Line-Algorithm

2) Diagonals for merge vertices

- when the vertex $v$ is reached, we set $\text{helper}(\text{left}(v)) = v$
- when we reach a split vertex $v'$ such that $\text{left}(v') = \text{left}(v)$ the diagonal $(v, v')$ is introduced
- in case we reach a regular vertex $v'$ such that $\text{helper}(\text{left}(v'))$ is $v$ the diagonal $(v, v')$ is introduced
Ideas for Sweep-Line-Algorithm

2) Diagonals for merge vertices

- when the vertex $v$ is reached, we set $\text{helper}(\text{left}(v)) = v$

- when we reach a split vertex $v'$ such that $\text{left}(v') = \text{left}(v)$ the diagonal $(v, v')$ is introduced

- in case we reach a regular vertex $v'$ such that $\text{helper}(\text{left}(v')) = v$ the diagonal $(v, v')$ is introduced

- if the end of $v'$ of $\text{left}(v)$ is reached, then the diagonal $(v, v')$ is introduced
Algorithm MakeMonotone(P)

**MakeMonotone**(Polygon $P$)

$\mathcal{D} \leftarrow$ doubly-connected edge list for $(V(P), E(P))$

$Q \leftarrow$ priority queue for $V(P)$ sorted lexicographically; $\mathcal{T} \leftarrow \emptyset$ (binary search tree for sweep-line status)

while $Q \neq \emptyset$ do

\begin{align*}
  v & \leftarrow Q\text{.nextVertex}() \\
  Q\text{.deleteVertex}(v) \\
  \text{handleVertex}(v)
\end{align*}

return $\mathcal{D}$
Algorithm MakeMonotone(P)

MakeMonotone(Polygon P)

\[ D \leftarrow \text{doubly-connected edge list for } (V(P), E(P)) \]
\[ Q \leftarrow \text{priority queue for } V(P) \text{ sorted lexicographically; } T \leftarrow \emptyset \text{ (binary search tree for sweep-line status)} \]

while \( Q \neq \emptyset \) do
\[ v \leftarrow Q\text{.nextVertex()} \]
\[ Q\text{.deleteVertex}(v) \]
\[ \text{handleVertex}(v) \]

return \( D \)

handleStartVertex(vertex \( v \))

\[ T \leftarrow \text{add the left edge } e \]
\[ \text{helper}(e) \leftarrow v \]
Algorithm MakeMonotone(P)

**MakeMonotone**(Polygon \(P\))

\[
\mathcal{D} \leftarrow \text{doubly-connected edge list for } (V(P), E(P)) \\
\mathcal{Q} \leftarrow \text{priority queue for } V(P) \text{ sorted lexicographically}; \mathcal{T} \leftarrow \emptyset \text{ (binary search tree for sweep-line status)}
\]

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

\[
v \leftarrow \mathcal{Q}.\text{nextVertex}() \\
\mathcal{Q}.\text{deleteVertex}(v) \\
\text{handleVertex}(v)
\]

return \(\mathcal{D}\)

**handleStartVertex**(vertex \(v\))

\[
\mathcal{T} \leftarrow \text{add the left edge } e \\
\text{helper}(e) \leftarrow v
\]

**handleEndVertex**(vertex \(v\))

\[
e \leftarrow \text{left edge} \\
\text{if isMergeVertex(helper(e)) then} \\
\mathcal{D} \leftarrow \text{add edge } (\text{helper}(e), v) \\
\text{remove } e \text{ from } \mathcal{T}
\]
Algorithm MakeMonotone(P)

\textbf{MakeMonotone}(Polygon \( P \))

\( D \leftarrow \) doubly-connected edge list for \((V(P), E(P))\)
\( Q \leftarrow \) priority queue for \( V(P) \) sorted lexicographically; \( T \leftarrow \emptyset \) (binary search tree for sweep-line status)

\textbf{while} \( Q \neq \emptyset \) \textbf{do}
  \begin{align*}
  v &\leftarrow Q\text{.nextVertex}() \\
  Q\text{.deleteVertex}(v) \\
  \text{handleVertex}(v)
  \end{align*}

\textbf{return} \( D \)

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

\begin{align*}
  e &\leftarrow \text{Edge to the left of } v \text{ in } T \\
  D &\leftarrow \text{add edge } (\text{helper}(e), v) \\
  \text{helper}(e) &\leftarrow v \\
  T &\leftarrow \text{add the right edge } e' \text{ of } v \\
  \text{helper}(e') &\leftarrow v
  \end{align*}
Algorithm MakeMonotone(P)

MakeMonotone(Polygon $P$)

$D \leftarrow$ doubly-connected edge list for $(V(P), E(P))$
$Q \leftarrow$ priority queue for $V(P)$ sorted lexicographically; $T \leftarrow \emptyset$ (binary search tree for sweep-line status)

while $Q \neq \emptyset$ do

$v \leftarrow Q$.nextVertex()
$Q$.deleteVertex($v$)
handleVertex($v$)

return $D$

handleMergeVertex(vertex $v$)

$e \leftarrow$ right edge
if isMergeVertex(helper($e$)) then
$D \leftarrow$ add edge (helper($e$), $v$)
remove $e$ from $T$

$e' \leftarrow$ edge to the left of $v$ in $T$
if isMergeVertex(helper($e'$)) then
$D \leftarrow$ add edge (helper($e'$), $v$)

helper($e'$) $\leftarrow$ $v$
Algorithm MakeMonotone(P)

MakeMonotone(Polygon P)

\[ \mathcal{D} \leftarrow \text{doubly-connected edge list for } (V(P), E(P)) \]
\[ \mathcal{Q} \leftarrow \text{priority queue for } V(P) \text{ sorted lexicographically; } \mathcal{T} \leftarrow \emptyset \text{ (binary search tree for sweep-line status)} \]

\[ \text{while } \mathcal{Q} \neq \emptyset \text{ do} \]
\[ \quad v \leftarrow \mathcal{Q} . \text{nextVertex}() \]
\[ \quad \mathcal{Q} . \text{deleteVertex}(v) \]
\[ \quad \text{handleVertex}(v) \]

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

handleRegularVertex(vertex v)

\[ \text{if } P \text{ lies locally to the right of } v \text{ then} \]
\[ \quad e, e' \leftarrow \text{above, below edge} \]
\[ \quad \text{if isMergeVertex(helper(e)) then} \]
\[ \quad \quad \mathcal{D} \leftarrow \text{add edge (helper(e), v)} \]
\[ \quad \quad \text{remove } e \text{ from } \mathcal{T} \]
\[ \quad \quad \mathcal{T} \leftarrow \text{add } e'; \text{ helper}(e') \leftarrow v \]
\[ \text{else} \]
\[ \quad e \leftarrow \text{edge to the left of } v \]
\[ \quad \text{add } e \text{ to } \mathcal{T} \]
\[ \quad \text{if isMergeVertex(helper(e)) then} \]
\[ \quad \quad \mathcal{D} \leftarrow \text{add (helper(e), v)} \]
\[ \quad \quad \text{helper(e)} \leftarrow v \]
Analysis

**Lemma 2:** The algorithm MakeMonotone computes a set of crossing-free diagonals of \( P \), which partitions \( P \) into \( y \)-monotone polygons.
Analysis

**Lemma 2:** The algorithm MakeMonotone computes a set of crossing-free diagonals of $P$, which partitions $P$ into $y$-monotone polygons.

**Theorem 3:** A simple polygon with $n$ vertices can be partitioned into $y$-monotone polygons in $O(n \log n)$ time and $O(n)$ space.
Analysis

**Lemma 2:** The algorithm MakeMonotone computes a set of crossing-free diagonals of $P$, which partitions $P$ into $y$-monotone polygons.

**Theorem 3:** A simple polygon with $n$ vertices can be partitioned into $y$-monotone polygons in $O(n \log n)$ time and $O(n)$ space.

- Construct priority queue $Q$: $O(n)$
- Initialize sweep-line status $T$: $O(1)$
- Handle a single event:
  - $Q$.deleteMax: $O(\log n)$
  - Find, remove, add element in $T$: $O(\log n)$
  - Add diagonals to $D$: $O(1)$
- Space: obviously $O(n)$
Proof of Art-Gallery-Theorem: Overview

Three-step procedure:

- **Step 1:** Decompose $P$ in $y$-monotone polygons

  **Definition:** A polygon $P$ is $y$-monotone, if for each horizontal line $\ell$ the intersection $\ell \cap P$ is connected.

- **Step 2:** Triangulate $y$-monotone polygons

- **Step 3:** use DFS to color the triangulated polygon

ToDo!

![Diagram of polygon triangulation process]
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top-down traversal of both sides
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate \( y \)-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing \( y \)-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

Reminder: The left and the right boundary of the polygon have decreasing $y$-coordinates

Approach: Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

Reminder: The left and the right boundary of the polygon have decreasing $y$-coordinates

Approach: Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate \( y \)-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing \( y \)-coordinates.

**Approach:** Greedy, top down traversal of both sides.
Triangulate $y$-monotone Polygon

Reminder: The left and the right boundary of the polygon have decreasing $y$-coordinates

Approach: Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate \( y \)-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing \( y \)-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates.

**Approach:** Greedy, top down traversal of both sides.
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate \( y \)-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing \( y \)-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides

Invariant?
Triangulate \( y \)-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing \( y \)-coordinates

**Approach:** Greedy, top down traversal of both sides

**Invariant?**
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides

Invariant?
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides

**Invariant?**
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates.

**Approach:** Greedy, top down traversal of both sides.

**Invariant?**
Triangulate $y$-monotone Polygon

Reminder: The left and the right boundary of the polygon have decreasing $y$-coordinates

Approach: Greedy, top-down traversal of both sides

Invariant?
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides

The already visited but not triangulated polygon has the shape of a *funnel* (trichter).
Triangulate \( y \)-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing \( y \)-coordinates

**Approach:** Greedy, top down traversal of both sides

The already visited but not triangulated polygon has the shape of a *funnel* (trichter).

- chains of concave vertices

Invariant?

Reminder:

The left and the right boundary of the polygon have decreasing \( y \)-coordinates.
**Triangulate $y$-monotone Polygon**

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides

*Invariant?*

The already visited but not triangulated polygon has the shape of a *funnel* (trichter).

Angle in $P > 180^\circ$

**Invariant?**

The already visited but not triangulated polygon has the shape of a *funnel* (trichter).

chains of concave vertices
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates.

**Approach:** Greedy, top down traversal of both sides.

*Invariant?*

The already visited but not triangulated polygon has the shape of a *funnel* (trichter).

- Angle in $P > 180^\circ$
- Chains of concave vertices
Triangulate \( y \)-monotone Polygon

**Remainder:** The left and the right boundary of the polygon have decreasing \( y \)-coordinates

**Approach:** Greedy, top down traversal of both sides

![Diagram of a polygon with labeled angles and vertices](image)

- **Invariant?** The already visited but not triangulated polygon has the shape of a *funnel* (trichter).
- **In our case:** chains of concave vertices

- **Angle in \( P \):** \( > 180^\circ \)
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates.

**Approach:** Greedy, top down traversal of both sides.

**Invariant?**

The already visited but not triangulated polygon has the shape of a *funnel* (trichter).

In our case:

- Only 1 chain!
- Concave vertices
- Convex vertices

Angle in $P > 180^\circ$
Triangulate $y$-monotone Polygon

**Reminder:** The left and the right boundary of the polygon have decreasing $y$-coordinates

**Approach:** Greedy, top down traversal of both sides

**Invariant?**

The already visited but not triangulated polygon has the shape of a *funnel* (trichter).

In our case:

- **simpler case**
- only 1 chain!
- chains of concave vertices

**Angle in $P$**

$\angle P > 180^\circ$

**concave**

**convex**
Algorithm TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon $P$ as doubly-connected list of edges)

Merge vertices on left and right chains into desc. seq. $\rightarrow u_1, \ldots, u_n$
Algorithm TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon $P$ as doubly-connected list of edges)

Merge vertices on left and right chains into desc. seq. $\rightarrow u_1, \ldots, u_n$

Stack $S \leftarrow \emptyset$; $S$.push($u_1$); $S$.push($u_2$)

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

if $u_j$ and $S$.top() from different paths then

while not $S$.empty() do

$v \leftarrow S$.pop()

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

$S$.push($u_{j-1}$); $S$.push($u_j$)
Algorithm TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon $P$ as doubly-connected list of edges)

Merge vertices on left and right chains into desc. seq. $\rightarrow u_1, \ldots, u_n$
Stack $S \leftarrow \emptyset$; $S$.push($u_1$); $S$.push($u_2$)
for $j \leftarrow 3$ to $n - 1$ do
  if $u_j$ and $S$.top() from different paths then
    while not $S$.empty() do
      $v \leftarrow S$.pop()
      if not $S$.empty() then draw $(u_j, v)$
      $S$.push($u_{j-1}$); $S$.push($u_j$)

Connect $u_n$ to all the vertices in $S$ (except for the first and the last)
Algorithm TriangulateMonotonePolygon

\textbf{TriangulateMonotonePolygon}(Polygon }P\text{ as doubly-connected list of edges)

Merge vertices on left and right chains into desc. seq. \( \rightarrow u_1, \ldots, u_n \)

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

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

\hspace{1em} \textbf{if } u_j \text{ and } S.\text{top}() \text{ from different paths then}

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

\hspace{3em} \textit{\hspace{1em}v \leftarrow S.\text{pop}()}

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

\hspace{3em} S.\text{push}(u_{j-1}); S.\text{push}(u_j)
Algorithm TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon $P$ as doubly-connected list of edges)

Merge vertices on left and right chains into desc. seq. $\rightarrow u_1, \ldots, u_n$
Stack $S \leftarrow \emptyset$; $S$.push($u_1$); $S$.push($u_2$)
for $j \leftarrow 3$ to $n - 1$ do
  if $u_j$ and $S$.top() from different paths then
    while not $S$.empty() do
      $v \leftarrow S$.pop()
      if not $S$.empty() then draw $(u_j, v)$
      $S$.push($u_{j-1}$); $S$.push($u_j$)
Algorithm TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon \( P \) as doubly-connected list of edges)

Merge vertices on left and right chains into desc. seq. \( \rightarrow u_1, \ldots, u_n \)
Stack \( S \leftarrow \emptyset; S.push(u_1); S.push(u_2) \)
for \( j \leftarrow 3 \) to \( n - 1 \) do
  if \( u_j \) and \( S.top() \) from different paths then
    while not \( S.empty() \) do
      \( v \leftarrow S.pop() \)
      if not \( S.empty() \) then draw \((u_j, v)\)
      \( S.push(u_{j-1}); S.push(u_j) \)
Algorithm TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon \( P \) as doubly-connected list of edges)

Merge vertices on left and right chains into desc. seq. \( u_1, \ldots, u_n \)

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

\[\text{for } j \leftarrow 3 \text{ to } n - 1 \text{ do} \]

\[\text{if } u_j \text{ and } S.\text{top}() \text{ from different paths then}\]

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

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

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

\[S.\text{push}(u_{j-1}); S.\text{push}(u_j)\]
Algorithm TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon $P$ as doubly-connected list of edges)

Merge vertices on left and right chains into desc. seq. → $u_1, \ldots, u_n$

Stack $S \leftarrow \emptyset$; $S$.push($u_1$); $S$.push($u_2$)

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

if $u_j$ and $S$.top() from different paths then

while not $S$.empty() do

$v \leftarrow S$.pop()

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

$S$.push($u_{j-1}$); $S$.push($u_j$)
Algorithm TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon $P$ as doubly-connected list of edges)

Merge vertices on left and right chains into desc. seq. $\rightarrow u_1, \ldots, u_n$

Stack $S \leftarrow \emptyset$; $S$.push($u_1$); $S$.push($u_2$)

for $j \leftarrow 3$ to $n - 1$ do
    if $u_j$ and $S$.top() from different paths then
        while not $S$.empty() do
            $v \leftarrow S$.pop()
            if not $S$.empty() then draw $(u_j, v)$
            $S$.push($u_{j-1}$); $S$.push($u_j$)
Algorithm TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon \( P \) as doubly-connected list of edges)

Merge vertices on left and right chains into desc. seq. → \( u_1, \ldots, u_n \)
Stack \( S \leftarrow \emptyset \); \( S\).push(\( u_1 \)); \( S\).push(\( u_2 \))

for \( j \leftarrow 3 \) to \( n - 1 \) do
  if \( u_j \) and \( S\).top() from different paths then
    while not \( S\).empty() do
      \( v \leftarrow S\).pop()
      if not \( S\).empty() then draw \((u_j, v)\)
      \( S\).push(\( u_{j-1} \)); \( S\).push(\( u_j \))
Algorithm TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon $P$ as doubly-connected list of edges)

Merge vertices on left and right chains into desc. seq. $\to u_1, \ldots, u_n$

Stack $S \leftarrow \emptyset$; $S$.push($u_1$); $S$.push($u_2$)

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

if $u_j$ and $S$.top() from different paths then

while not $S$.empty() do

$v \leftarrow S$.pop()

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

$S$.push($u_{j-1}$); $S$.push($u_j$)
Algorithm TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon $P$ as doubly-connected list of edges)

Merge vertices on left and right chains into desc. seq. $\rightarrow u_1, \ldots, u_n$

Stack $S \leftarrow \emptyset$; $S$.push($u_1$); $S$.push($u_2$)

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

if $u_j$ and $S$.top() from different paths then

while not $S$.empty() do

$v \leftarrow S$.pop()

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

$S$.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

TriangulateMonotonePolygon(Polygon $P$ as doubly-connected list of edges)

Merge vertices on left and right chains into desc. seq. $\rightarrow u_1, \ldots, u_n$

Stack $S \leftarrow \emptyset$; $S$.push($u_1$); $S$.push($u_2$)

for $j \leftarrow 3$ to $n-1$ do
  if $u_j$ and $S$.top() from different paths then
    while not $S$.empty() do
      $v \leftarrow S$.pop()
      if not $S$.empty() then draw $(u_j, v)$
      $S$.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($u_j$)

Connect $u_n$ to all the vertices in $S$ (except for the first and the last)
Algorithm TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon \( P \) as doubly-connected list of edges)

Merge vertices on left and right chains into desc. seq. \( \rightarrow u_1, \ldots, u_n \)
Stack \( S \leftarrow \emptyset; S\text{.push}(u_1); S\text{.push}(u_2) \)

\[ \text{for } j \leftarrow 3 \text{ to } n-1 \text{ do} \]
\[ \quad \text{if } u_j \text{ and } S\text{.top()} \text{ from different paths then} \]
\[ \quad \quad \quad \text{while not } S\text{.empty()} \text{ do} \]
\[ \quad \quad \quad \quad v \leftarrow S\text{.pop()} \]
\[ \quad \quad \quad \quad \text{if not } S\text{.empty()} \text{ then draw } (u_j, v) \]
\[ \quad \quad \quad \quad S\text{.push}(u_{j-1}); S\text{.push}(u_j) \]
\[ \quad \text{else} \]
\[ \quad \quad v \leftarrow S\text{.pop()} \]
\[ \quad \quad \text{while not } S\text{.empty()} \text{ and } u_j \text{ sees } S\text{.top()} \text{ do} \]
\[ \quad \quad \quad v \leftarrow S\text{.pop()} \]
\[ \quad \quad \quad \text{draw diagonal } (u_j, v) \]
\[ \quad \quad \quad S\text{.push}(v); S\text{.push}(u_j) \]
Algorithm TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon P as doubly-connected list of edges)

Merge vertices on left and right chains into desc. seq. → \( u_1, \ldots, u_n \)

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

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

if \( u_j \) and \( S\).top() from different paths then

while not \( S\).empty() do

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

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

\( S\).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

TriangulateMonotonePolygon(Polygon $P$ as doubly-connected list of edges)

Merge vertices on left and right chains into desc. seq. → $u_1, \ldots, u_n$

Stack $S \leftarrow \emptyset$; $S$.push($u_1$); $S$.push($u_2$)

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

  if $u_j$ and $S$.top() from different paths then
    while not $S$.empty() do
      $v \leftarrow S$.pop()
      if not $S$.empty() then draw $(u_j, v)$
      $S$.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($u_j$)

Connect $u_n$ to all the vertices in $S$ (except for the first and the last)
Algorithm TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon $P$ as doubly-connected list of edges)

1. Merge vertices on left and right chains into desc. seq. → $u_1, \ldots, u_n$
2. Stack $S ← \emptyset$; $S$.push($u_1$); $S$.push($u_2$)
3. for $j ← 3$ to $n − 1$ do
   - if $u_j$ and $S$.top() from different paths then
     - while not $S$.empty() do
       - $v ← S$.pop()
       - if not $S$.empty() then draw $(u_j, v)$
       - $S$.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($u_j$)
Algorithm TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon $P$ as doubly-connected list of edges)

Merge vertices on left and right chains into desc. seq. → $u_1, \ldots, u_n$

Stack $S ← \emptyset$; $S$.push($u_1$); $S$.push($u_2$)

for $j ← 3$ to $n - 1$ do
  if $u_j$ and $S$.top() from different paths then
    while not $S$.empty() do
      $v ← S$.pop()
      if not $S$.empty() then draw $(u_j, v)$
      $S$.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($u_j$)

Connect $u_n$ to all the vertices in $S$ (except for the first and the last)
Algorithm TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon $P$ as doubly-connected list of edges)

Merge vertices on left and right chains into desc. seq. → $u_1, \ldots, u_n$
Stack $S \leftarrow \emptyset$; $S$.push($u_1$); $S$.push($u_2$)

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

  if $u_j$ and $S$.top() from different paths then
    while not $S$.empty() do
      $v \leftarrow S$.pop()
      if not $S$.empty() then draw $(u_j, v)$
      $S$.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($u_j$)

Connect $u_n$ to all the vertices in $S$ (except for the first and the last)
Algorithm TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon $P$ as doubly-connected list of edges)

Merge vertices on left and right chains into desc. seq. $\rightarrow u_1, \ldots, u_n$

Stack $S \leftarrow \emptyset$; $S$.push($u_1$); $S$.push($u_2$)

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

  if $u_j$ and $S$.top() from different paths then

    while not $S$.empty() do

      $v \leftarrow S$.pop()

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

      $S$.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

TriangulateMonotonePolygon(Polygon $P$ as doubly-connected list of edges)

Merge vertices on left and right chains into desc. seq. → $u_1, \ldots, u_n$

Stack $S \leftarrow \emptyset$; $S$.push($u_1$); $S$.push($u_2$)

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

if $u_j$ and $S$.top() from different paths then

while not $S$.empty() do

$v \leftarrow S$.pop()

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

$S$.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($u_j$)

Connect $u_n$ to all the vertices in $S$ (except for the first and the last)
Algorithm TriangulateMonotonePolygon

TriangulateMonotonePolygon(Polygon $P$ as doubly-connected list of edges)

Merge vertices on left and right chains into desc. seq. $\rightarrow u_1, \ldots, u_n$
Stack $S \leftarrow \emptyset$; $S$.push($u_1$); $S$.push($u_2$)
for $j \leftarrow 3$ to $n - 1$ do
  if $u_j$ and $S$.top() from different paths then
    while not $S$.empty() do
      $v \leftarrow S$.pop()
      if not $S$.empty() then draw ($u_j$, $v$)
      $S$.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($u_j$)

Connect $u_n$ to all the vertices in $S$ (except for the first and the last)

Task: What is the running time?
Summary

**Theorem 4:** A $y$-monotone polygon with $n$ vertices can be triangulated in $O(n)$ time.
Summary

Theorem 4: A \( y \)-monotone polygon with \( n \) vertices can be triangulated in \( O(n) \) time.

Theorem 3: A simple polygon with \( n \) vertices can be partitioned into \( y \)-monotone polygons in \( O(n \log n) \) time and \( O(n) \) space.
Summary

**Theorem 4:** A \( y \)-monotone polygon with \( n \) vertices can be triangulated in \( O(n) \) time.

**Theorem 3:** A simple polygon with \( n \) vertices can be partitioned into \( y \)-monotone polygons in \( O(n \log n) \) time and \( O(n) \) space.

\[
\downarrow
\]

**Theorem 5:** A simple polygon with \( n \) vertices can be triangulated in \( O(n \log n) \) time and \( O(n) \) space.
Proof of Art-Gallery-Theorem: Overview

Three-step procedure:

• Step 1: Decompose $P$ in $y$-monotone polygons

  **Definition:** A polygon $P$ is $y$-monotone, if for each horizontal line $\ell$ the intersection $\ell \cap P$ is connected.

  ![Diagram showing definition of y-monotone polygons]

• Step 2: Triangulate $y$-monotone polygons

• Step 3: use DFS to color the triangulated polygon

![Diagram showing triangulation process]
Discussion

Can the triangulation algorithm be expanded to work with polygons with holes?
Discussion

Can the triangulation algorithm be expanded to work with polygons with holes?

• Triangulation: yes

• But are $\lceil n/3 \rceil$ cameras still sufficient to guard it?
  No, a generalization of Art-Gallery-Theorems says that $\lfloor (n + h)/3 \rfloor$ cameras are sometimes necessary, and always sufficient, where $h$ is the number of holes. [Hoffmann et al., 1991]
Discussion

Can the triangulation algorithm be expanded to work with polygons with holes?

• Triangulation: yes

• But are \(\lfloor n/3 \rfloor\) cameras still sufficient to guard it?
  No, a generalization of Art-Gallery-Theorems says that \(\lfloor (n + h)/3 \rfloor\) cameras are sometimes necessary, and always sufficient, where \(h\) is the number of holes. [Hoffmann et al., 1991]

Can we solve the triangulation problem faster for simple polygons?
Discussion

Can the triangulation algorithm be expanded to work with polygons with holes?

• Triangulation: yes

• But are $\left\lfloor n/3 \right\rfloor$ cameras still sufficient to guard it?
  No, a generalization of Art-Gallery-Theorems says that $\left\lfloor (n + h)/3 \right\rfloor$ cameras are sometimes necessary, and always sufficient, where $h$ is the number of holes. [Hoffmann et al., 1991]

Can we solve the triangulation problem faster for simple polygons?
Yes. The question whether it is possible was open for more than a decade. In the end of 80’s a faster randomized algorithm was given, and in 1990 Chazelle presented a deterministic linear-time algorithm (complicated).