Computational Geometry · Lecture
Projects & Polygon Triangulation

Tamara Mchedlidze · Darren Strash
02.11.2015
Projects
Project Specifications

Groups:
- 2 or 3 students, assigned to a supervisor

Tools:
- Use any programming language, should run with little effort on Linux

Due date:
- 08.02.2016

Visualization:
- Need to visualize output; ipe, svg, video? This is also helpful for debugging output
- Do not need to use graphics APIs, unless you really want to

Group Presentations:
- Each group gives a 20-minute presentation (last 2 weeks of class)
Project—Next Steps

Project proposals: due in 2 weeks (16.11.2015):
- No more than 1 page
- Formalize your problem
- Describe geometric primitives
- State any simplifying assumptions
- Describe the algorithms / data structures you will use
- Avoid brute force algorithms

Supervisor:
- Assigned supervisor (Tamara, Darren, or Benjamin)
- Meet with supervisor in two weeks to discuss proposal.
Project—Next Steps

**Project proposals:** due in 2 weeks (16.11.2015):

- No more than 1 page
- Formalize your problem
- Describe geometric primitives
- State any simplifying assumptions
- Describe the algorithms / data structures you will use
- **Avoid brute force algorithms**

**Supervisor:**

- Assigned supervisor (Tamara, Darren, or Benjamin)
- Meet with supervisor in two weeks to discuss proposal.

Now, form groups and sit by each other!
Project—Next Steps

**Project proposals:** due in 2 weeks (16.11.2015):

- No more than 1 page
- Formalize your problem
- Describe geometric primitives
- State any simplifying assumptions
- Describe the algorithms / data structures you will use
- **Avoid brute force algorithms**

**Supervisor:**

- Assigned supervisor (Tamara, Darren, or Benjamin)
- Meet with supervisor in two weeks to discuss proposal.

Now, form groups and sit by each other!

Time to present the projects...
Scenario: Development of the game Pacman.

Story:
Pacman is strolling through a large city with many buildings, gathering items. However, incredibly many ghosts want to hinder Pacman by eating him.
Project 1: Collision Detection

moving objects ↔ moving objects
moving objects ↔ buildings
Project 1: Collision Detection

Project 2: Artificial Intelligence I

Ghosts seeing Pacman follow him until they lose eye contact.

Fast visibility check:
Is Pacman visible from point \( p \)?
Project 1: Collision Detection

Project 2: Artificial Intelligence I

Project 3: Artificial Intelligence II

Fast visibility check:
Is Pacman visible from point \( p \)?

Fast smell check:
Can Pacman be smelled from point \( p \)?

Ghosts seeing Pacman follow him until they lose eye contact.

Ghosts smelling Pacman follow him until they lose the track.
Secret Agent
Shooting Secret Agent
The ray of the laser can reflect only $d$ times!
The ray of the laser can reflect only \( d \) times!
The ray of the laser can reflect only $d$ times!

Can the agent shoot the target?
The ray of the laser can reflect only $d$ times!

Can the agent shoot the target?
The ray of the laser can reflect only $d$ times!

Can the agent shoot the target?

OOPS!
Secret Agent’s Robot
The robot can travel distance $d$ without getting charged.
Secret Agent’s Robot

- The robot can travel distance $d$ without getting charged.

- Can the robot escape the warehouse?
Secret Agent’s Robot

- The robot can travel distance $d$ without getting charged.
- Can the robot escape the warehouse?
Secret Agent’s Robot

- The robot can travel distance $d$ without getting charged.
- Can the robot escape the warehouse?
Secret Agent’s Robot

- The robot can travel distance $d$ without getting charged.
- Can the robot escape the warehouse?
Secret Agent’s Robot

- The robot can travel distance $d$ without getting charged.
- Can the robot escape the warehouse?
Secret Agent Protects Jewels
Secret Agent Protects Jewels
Secret Agent Protects Jewels
Secret Agent Protects Jewels
Secret Agent Protects Jewels
Secret Agent Protects Jewels
Recruit the small number of guards.
Recruit the small number of guards.

Find the regions they must patrol.
Puzzles
Puzzles

Puzzle # 1: Food fit

Given:
- A region representing an ant colony’s home
- A “picnic” containing food items

Output:
- Which food items can the ants fit in their home?
Puzzles

Puzzle # 1: Food fit
Puzzles

Puzzle # 1: Food fit

Fits!
Puzzle # 1: Food fit
Puzzle # 1: Food fit
Puzzle # 1: Food fit

Fits!
Puzzle # 1: Food fit
Puzzle # 1: Food fit
Puzzle # 1: Food fit

Does not fit!
Puzzle # 2: Protect the colony

Given:
- A region representing an ant colony’s home
- A nail of length \( l \), to be hammered into the home

Output:
- Will the nail break the home apart?
- How can we make “small” changes to protect the home?
Puzzle # 2: Protect the colony

Given:
- A region representing an ant colony’s home
- A nail of length $l$, to be hammered into the home

Output:
- Will the nail break the home apart?
- How can we make “small” changes to protect the home?
Puzzle # 2: Protect the colony

Given:
- A region representing an ant colony’s home
- A nail of length $l$, to be hammered into the home

Output:
- Will the nail break the home apart?
- How can we make “small” changes to protect the home?
Puzzles

Puzzle # 2: Protect the colony

Given:
- A region representing an ant colony’s home
- A nail of length $l$, to be hammered into the home

Output:
- Will the nail break the home apart?
- How can we make “small” changes to protect the home?
Puzzles

Puzzle # 2: Protect the colony

Given:
- A region representing an ant colony’s home
- A nail of length $l$, to be hammered into the home

Output:
- Will the nail break the home apart?
- How can we make “small” changes to protect the home?
Puzzle # 2: Protect the colony

Given:
- A region representing an ant colony’s home
- A nail of length $l$, to be hammered into the home

Output:
- Will the nail break the home apart?
- How can we make “small” changes to protect the home?
Puzzle # 3: Keep away

Given:
- A collection of ant colonies that grow over time
- A keep-away distance $k$

Output:
- The state of the colonies after $t$ time steps
Puzzle # 3: Keep away

Given:
- A collection of ant colonies that grow over time
- A keep-away distance $k$

Output:
- The state of the colonies after $t$ time steps
Puzzle # 3: Keep away

Given:
- A collection of ant colonies that grow over time
- A keep-away distance $k$

Output:
- The state of the colonies after $t$ time steps
Project Selection
Projects

1. **Pacman**: Collision detection
2. **Pacman**: Ghosts activate on sight
3. **Pacman**: Ghosts activate on smell
4. **Secret Agent**: Shoot the laser
5. **Secret Agent**: Save the robot
6. **Secret Agent**: Guard the jewels
7. **Puzzle**: Food fit
8. **Puzzle**: Protect the colony
9. **Puzzle**: Keep away
Polygon Triangulation
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!

$\rightarrow$ 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!

$\rightarrow$ 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!*
**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
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.

- $P$ can 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, $\lfloor n/3 \rfloor$ 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, $\lceil n/3 \rceil$ 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 \cite{Chvatal75}

**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
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!

![Diagram of a polygon with cameras](image)

- 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 $\approx n/3$ 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.

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.
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.

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.

- **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:

- regular vertices
Partition into $y$-monotone Polygons

**Idea:** Five different types of vertices

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

**Idea:** Five different types of vertices

- **Turn vertices:**
  - vertical change in direction
    - *start* vertices
    - if $\alpha < 180^\circ$

- **regular vertices**
Partition into $y$-monotone Polygons

**Idea:** Five different types of vertices

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

- **regular vertices**
Partition into $y$-monotone Polygons

**Idea:** Five different types of vertices

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

- **regular 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
- **regular 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*

- $\alpha < 180^\circ$ if $\alpha < 180^\circ$
- $\beta > 180^\circ$ if $\beta > 180^\circ$
- $\gamma < 180^\circ$ if $\gamma < 180^\circ$
- $\delta > 180^\circ$ if $\delta > 180^\circ$
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
Lemma 1: A polygon is \( y \)-monotone if it has no split vertices or merge vertices.

Proof: On the blackboard

\[\Rightarrow \text{We need to eliminate all split and merge vertices by using diagonals}\]
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

**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 $\text{helper}(e) := w$
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 \( \text{helper}(\text{left}(v')) \) is \( v \) 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)

\textbf{MakeMonotone}(Polygon \ P)

\begin{algorithmic}
\State $D \gets$ doubly-connected edge list for $(V(P), E(P))$
\State $Q \gets$ priority queue for $V(P)$ sorted lexicographically; $T \gets \emptyset$ (binary search tree for sweep-line status)
\While{$Q \neq \emptyset$}
\State $v \gets Q$.nextVertex()
\State $Q$.deleteVertex($v$)
\State handleVertex($v$)
\EndWhile
\State \textbf{return} $D$
\end{algorithmic}
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 \( v \))

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

\[ v = \text{helper}(e) \]
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)

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

MakeMonotone(Polygon \( P \))

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

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

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

**return** \( \mathcal{D} \)

handleSplitVertex(vertex \( v \))

\[ e \leftarrow \text{Edge to the left of } v \text{ in } \mathcal{T} \]
\[ \mathcal{D} \leftarrow \text{add edge } (\text{helper}(e), v) \]
\[ \text{helper}(e) \leftarrow v \]
\[ \mathcal{T} \leftarrow \text{add the right edge } e' \text{ of } v \]
\[ \text{helper}(e') \leftarrow v \]
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 $P$)

1. $\mathcal{D} \leftarrow$ doubly-connected edge list for $(V(P), E(P))$
2. $\mathcal{Q} \leftarrow$ priority queue for $V(P)$ sorted lexicographically; $\mathcal{T} \leftarrow \emptyset$ (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** $\mathcal{D}$

**handleRegularVertex**(vertex $v$)

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

- **else**
  - $e' \leftarrow$ edge to the left of $v$
  - add $e'$ to $\mathcal{T}$
  - **if** $\text{isMergeVertex}(\text{helper}(e))$ **then**
    - $\mathcal{D} \leftarrow$ add $(\text{helper}(e), v)$
    - $\text{helper}(e) \leftarrow v$
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  
  
  **ToDo!**

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

![Diagram of triangulation process](image-url)
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
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?**

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

![Diagram of triangulation process]

**Invariant?**

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

- chains of concave vertices
Triangulate $\gamma$-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 $P$

$> 180^\circ$

concave

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 $P > 180^\circ$**

**concave**

**convex**

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°

**In our case:**
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).

In our case: only 1 chain!

**Angle in $P$**

$> 180^\circ$

Convex

Concave

**Invariant?**

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).

In our case:

- Only 1 chain!
- Simpler case

**Angle in $P$**

- $> 180^\circ$

**Convex**

**Concave**

chains of concave vertices
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

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($u_j$)
  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 \( j \leftarrow 3 \text{ to } n - 1 \) 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() \n
if not \( S \).empty() then draw \((u_j, v)\) \n
\( S \).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

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

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

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

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

\[ S.\text{push}(u_{j-1}); S.\text{push}(u_j) \]
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 $S ← \emptyset$; $S$.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($u_j$)
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($u_j$)

  Connect $u_{j}$ 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 $S ← \emptyset; S$.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($u_j$)
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 \( S \leftarrow \emptyset; S\text{.push}(u_1); S\text{.push}(u_2) \)

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

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

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

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

if not \( S\text{.empty}() \) 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. $\rightarrow u_1, \ldots, u_n$

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

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

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

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

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

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

\( \text{else} \)

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

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

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

\[ \text{draw diagonal } (u_j, v) \]

\[ S.push(v); S.push(u_j) \]
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($u_j$)
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($u_j$)
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 $S ← \emptyset$; $S$.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($u_j$)
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() \n
if not \( S\).empty() then draw \((u_j, v)\) \n
\( S\).push\((u_{j - 1})\); \( S\).push\((u_j)\)

else

\( v \leftarrow S\).pop() \n
while not \( S\).empty() and \( u_j \) sees \( S\).top() do

\( v \leftarrow S\).pop() \n
draw diagonal \((u_j, v)\) \n
\( S\).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($u_j$)
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. $\rightarrow u_1, \ldots, u_n$

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.

- Step 2: Triangulate $y$-monotone polygons ✓
- Step 3: use DFS to color the triangulated polygon ✓
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 \(\lceil (n + h)/3 \rceil\)
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 \( \lfloor n/3 \rfloor \) 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?

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).