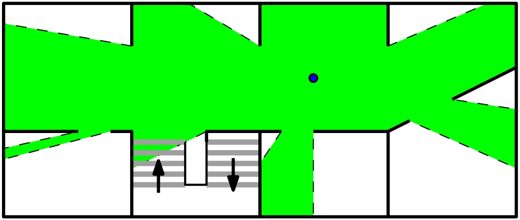
Triangulating a polygon

Computational Geometry

Lecture 4: Triangulating a polygon

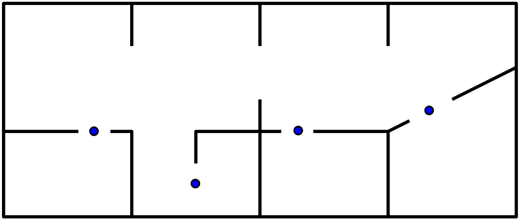
Polygons and visibility

Two points in a simple polygon can see each other if their connecting line segment is in the polygon



Art gallery problem

Art Gallery Problem: How many cameras are needed to guard a given art gallery so that every point is seen?



Art gallery problem

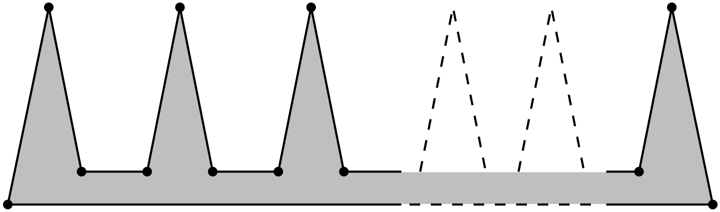
In geometry terminology: How many points are needed in a simple polygon with n vertices so that every point in the polygon is seen?

The optimization problem is computationally difficult

Art Gallery Theorem: $\lfloor n/3 \rfloor$ cameras are occasionally necessary but always sufficient

Art gallery problem

Art Gallery Theorem: $\lfloor n/3 \rfloor$ cameras are occasionally necessary but always sufficient

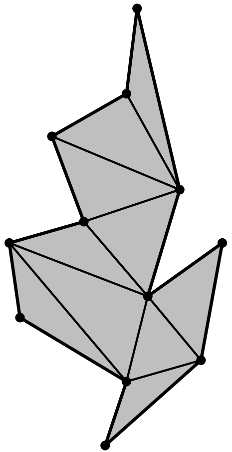


Triangulation, diagonal

Why are $\lfloor n/3 \rfloor$ cameras always enough?

Assume polygon P is triangulated: a decomposition of P into disjoint triangles by a maximal set of non-intersecting diagonals

Diagonal of P: open line segment that connects two vertices of P and fully lies in the interior of P



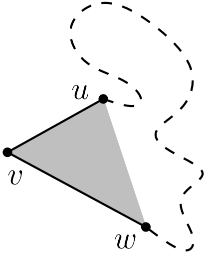
A triangulation always exists

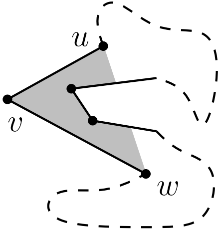
Lemma: A simple polygon with n vertices can always be triangulated, and always with $n-2$ triangles

Proof: Induction on n. If $n=3$, it is trivial

Assume $n>3$. Consider the leftmost vertex v and its two neighbors u and w. Either uw is a diagonal (case 1), or part of the boundary of P is in $\triangle uvw$ (case 2)

Case 2: choose the vertex t in $\triangle uvw$ farthest from the line through u and w, then $\overline{vt}$ must be a diagonal





A triangulation always exists

In case 1, $\overline{uw}$ cuts the polygon into a triangle and a simple polygon with $n-1$ vertices, and we apply induction

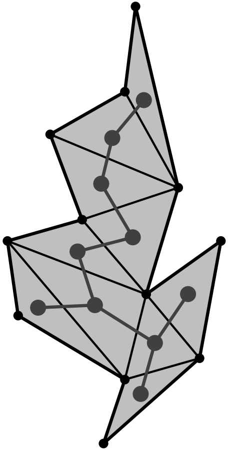
In case 2, $\overline{vt}$ cuts the polygon into two simple polygons with m and $n-m+2$ vertices, $3 \leq m \leq n-1$, and we also apply induction

By induction, the two polygons can be triangulated using $m-2$ and $n-m+2-2 = n-m$ triangles. So the original polygon is triangulated using $m-2\ +\ n-m = n-2$ triangles $\square$

A 3-coloring always exists

Observe: the dual graph of a triangulated simple polygon is a tree

Dual graph: each face gives a node; two nodes are connected if the faces are adjacent

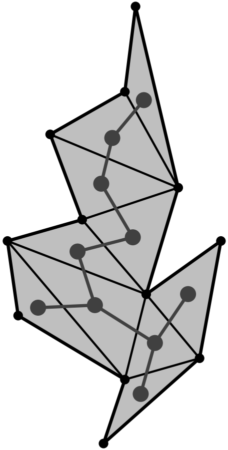


A 3-coloring always exists

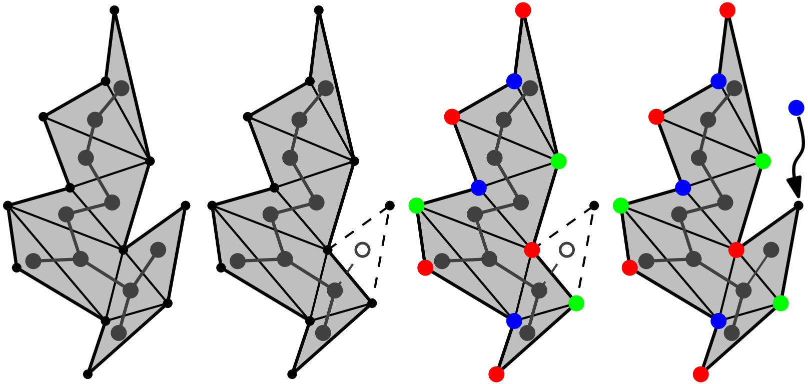
Lemma: The vertices of a triangulated simple polygon can always be 3-colored

Proof: Induction on the number of triangles in the triangulation. Base case: True for a triangle

Every tree has a leaf, in particular the one that is the dual graph. Remove the corresponding triangle from the triangulated polygon, color its vertices, add the triangle back, and let the extra vertex have the color different from its neighbors



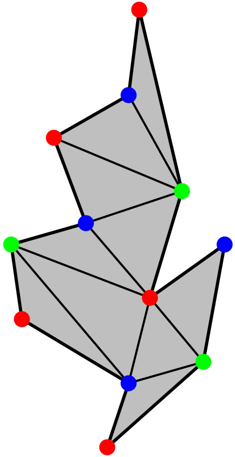
A 3-coloring always exists



$\lfloor n/3 \rfloor$ cameras are enough

For a 3-colored, triangulated simple polygon, one of the color classes is used by at most $\lfloor n/3 \rfloor$ colors. Place the cameras at these vertices

This argument is called
the pigeon-hole principle



$\lfloor n/3 \rfloor$ cameras are enough

Question: Why does the proof fail when the polygon has holes?

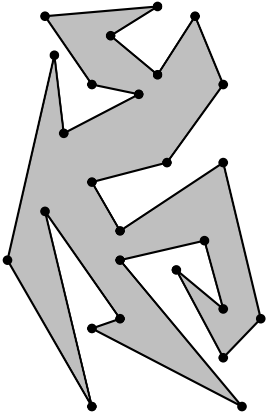
Two-ears for triangulation

Using the two-ears theorem:
(an ear consists of three consecutive vertices u, v, w where $\overline{uw}$ is a diagonal)

Find an ear, cut it off with a diagonal, triangulate the rest iteratively

Question: Why does every simple polygon have an ear?

Question: How efficient is this algorithm?

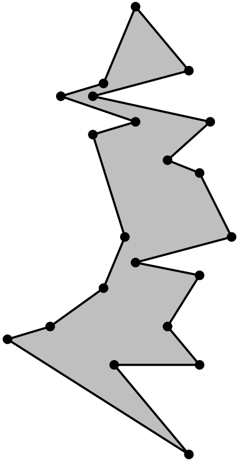


Overview

A simple polygon is y-monotone iff any horizontal line intersects it in a connected set (or not at all)

Use plane sweep to partition the polygon into y-monotone polygons

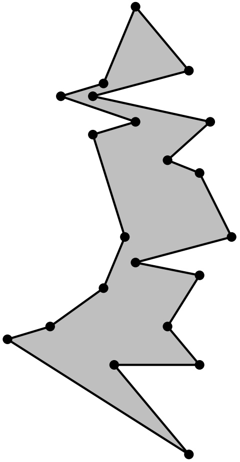
Then triangulate each y-monotone polygon



Monotone polygons

A y-monotone polygon has a top vertex, a bottom vertex, and two y-monotone chains between top and bottom as its boundary

Any simple polygon with one top vertex and one bottom vertex is y-monotone

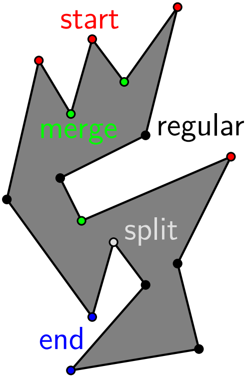


Vertex types

What types of vertices does a simple polygon have?

- start
- stop
- split
- merge
- regular

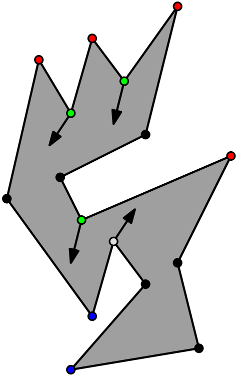
... imagining a sweep line going top to bottom



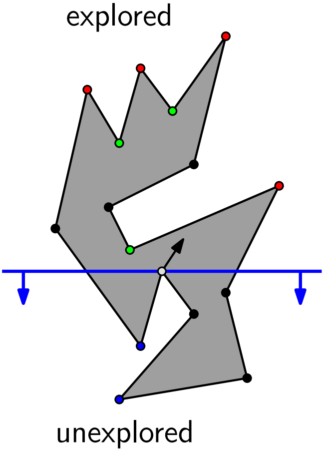
Sweep ideas

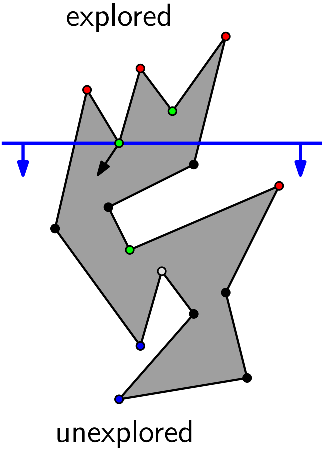
Find diagonals from each merge vertex down, and from each split vertex up

A simple polygon with no split or merge vertices can have at most one start and one end vertex, so it is y-monotone



Sweep ideas

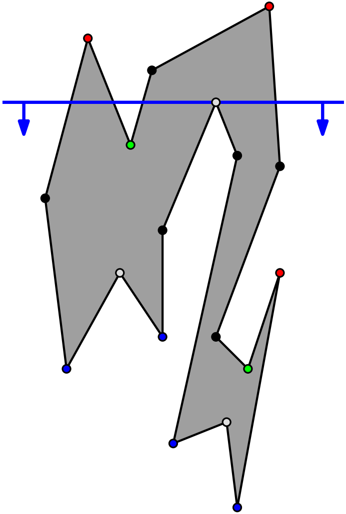




Sweep ideas

Where can a diagonal from a split vertex go?

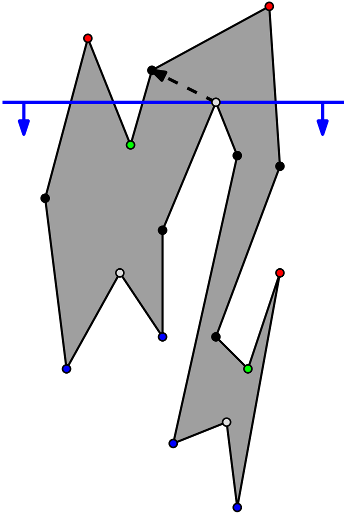
Perhaps the upper endpoint of the edge immediately left of the merge vertex?



Sweep ideas

Where can a diagonal from a split vertex go?

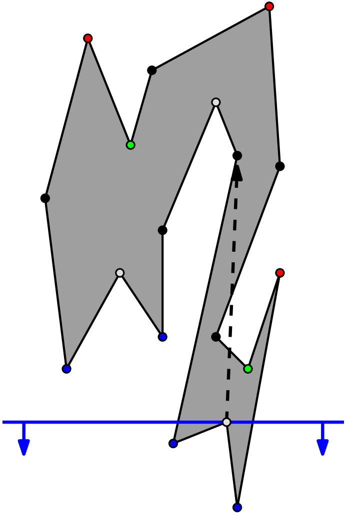
Perhaps the upper endpoint of the edge immediately left of the merge vertex?



Sweep ideas

Where can a diagonal from a split vertex go?

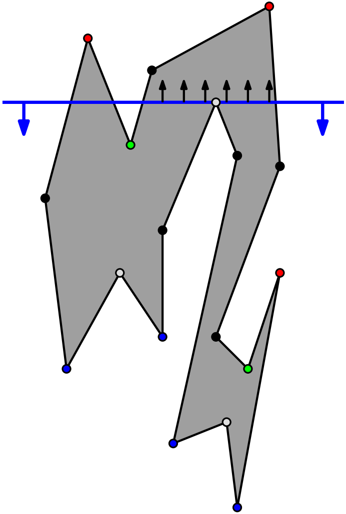
Perhaps the upper endpoint of the edge immediately left of the merge vertex?



Sweep ideas

Where can a diagonal from a split vertex go?

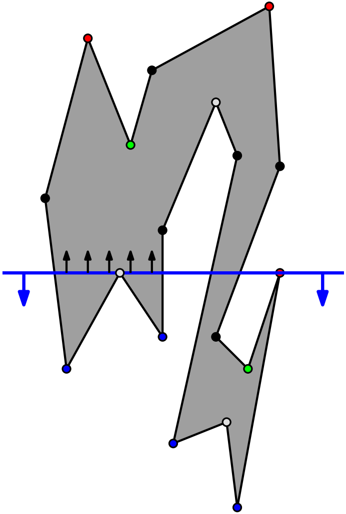
Perhaps the last vertex passed in the same “component”?



Sweep ideas

Where can a diagonal from a split vertex go?

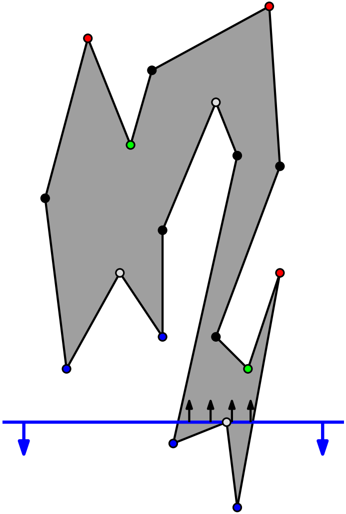
Perhaps the last vertex passed in the same "component"?



Sweep ideas

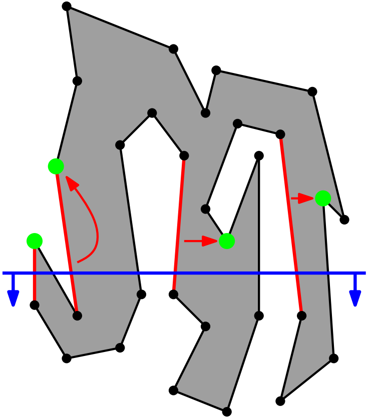
Where can a diagonal from a split vertex go?

Perhaps the last vertex passed in the same "component"?



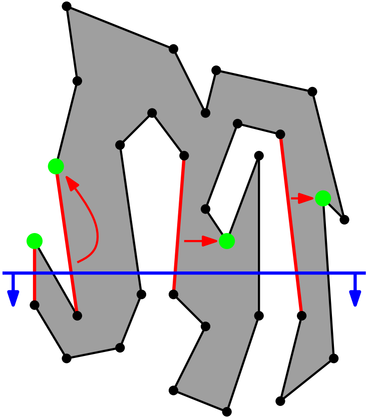
Helpers of edges

The helper for an edge e that has the polygon right of it, and a position of the sweep line, is the lowest vertex v above the sweep line such that the horizontal line segment connecting e and v is inside the polygon



Status of sweep

The status is the set of edges intersecting the sweep line that have the polygon to their right, sorted from left to right, and each with their *helper*: the last vertex passed in that component



Status structure, event list

The status structure stores all edges that have the polygon to the right, with their helper, sorted from left to right in the leaves of a balanced binary search tree

The events happen only at the vertices: sort them by y-coordinate and put them in a list (or array, or tree)

Main algorithm

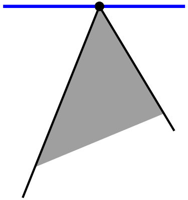
Initialize the event list (all vertices sorted by decreasing y-coordinate) and the status structure (empty)

While there are still events in the event list, remove the first (topmost) one and handle it

Event handling

Start vertex v:

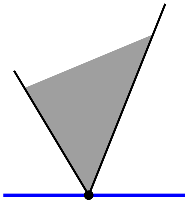
- Insert the counterclockwise incident edge in T with v as the helper



Event handling

End vertex v:

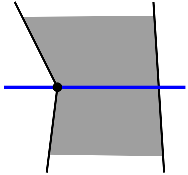
- Delete the clockwise incident edge and its helper from T

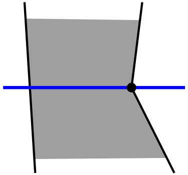


Event handling

Regular vertex v:

- If the polygon is right of the two incident edges, then replace the upper edge by the lower edge in T, and make v the helper
- If the polygon is left of the two incident edges, then find the edge e directly left of v, and replace its helper by v

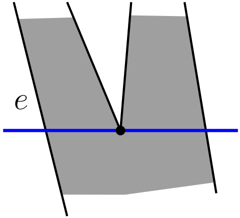




Event handling

Merge vertex v:

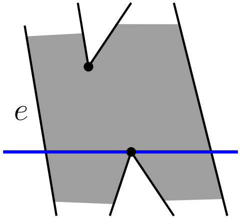
- Remove the edge clockwise from v from T
- Find the edge e directly left of v, and replace its helper by v



Event handling

Split vertex v:

- Find the edge e directly left of v, and choose as a diagonal the edge between its helper and v
- Replace the helper of e by v
- Insert the edge counterclockwise from v in T, with v as its helper



Efficiency

Sorting all events by y-coordinate takes $O(n \log n)$ time

Every event takes $O(\log n)$ time, because it only involves querying, inserting and deleting in T

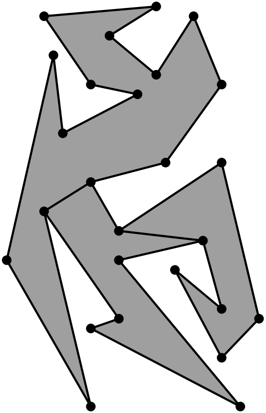
Degenerate cases

Question: Which degenerate cases arise in this algorithm?

Representation

A simple polygon with some diagonals is a subdivision $\Rightarrow$ use a DCEL

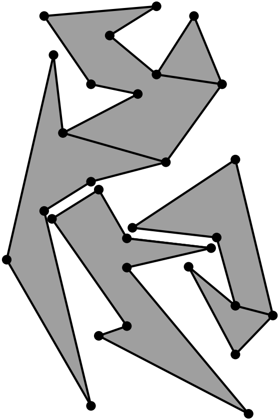
Question: How many diagonals may be chosen to the same vertex?



More sweeping

With an upward sweep in each subpolygon, we can find a diagonal down from every merge vertex (which is a split vertex for an upward sweep!)

This makes all subpolygons y-monotone

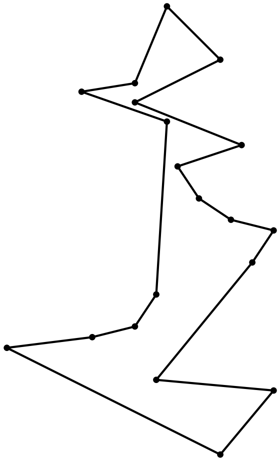


Result

Theorem: A simple polygon with n vertices can be partitioned into y-monotone pieces in $O(n \log n)$ time

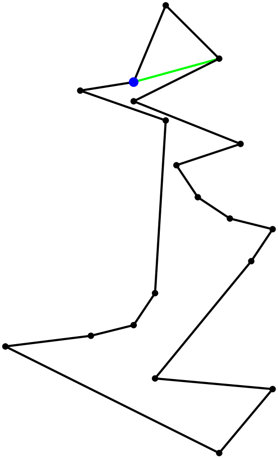
Triangulating a monotone polygon

How to triangulate a y-monotone polygon?



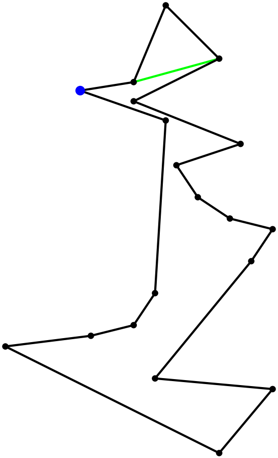
Triangulating a monotone polygon

How to triangulate a y-monotone polygon?



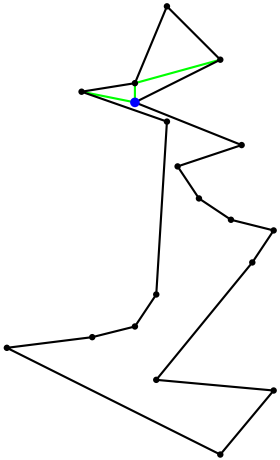
Triangulating a monotone polygon

How to triangulate a y-monotone polygon?



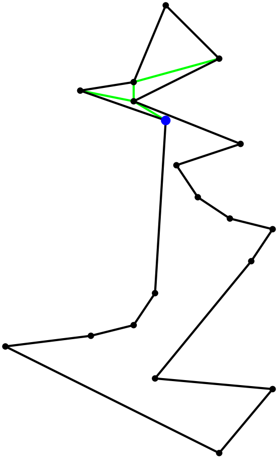
Triangulating a monotone polygon

How to triangulate a y-monotone polygon?



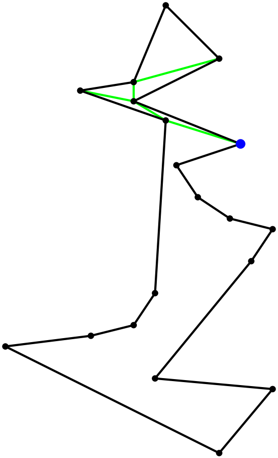
Triangulating a monotone polygon

How to triangulate a y-monotone polygon?



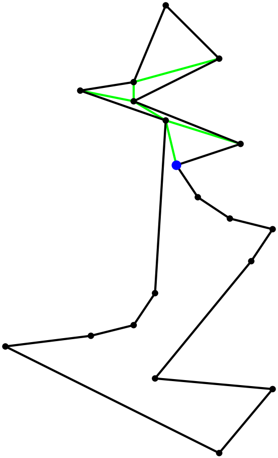
Triangulating a monotone polygon

How to triangulate a y-monotone polygon?



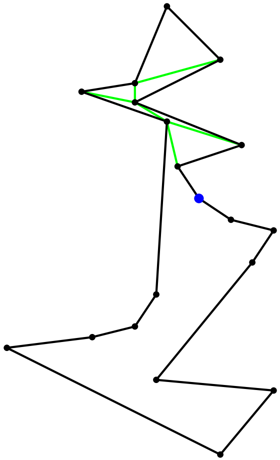
Triangulating a monotone polygon

How to triangulate a y-monotone polygon?



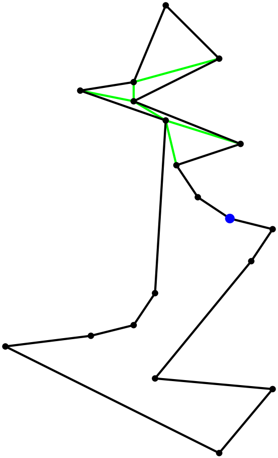
Triangulating a monotone polygon

How to triangulate a y-monotone polygon?



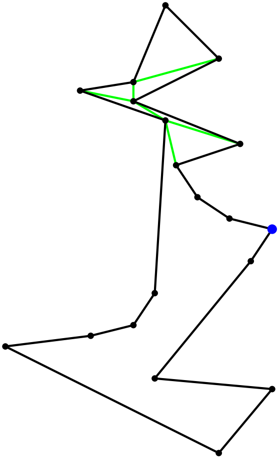
Triangulating a monotone polygon

How to triangulate a y-monotone polygon?



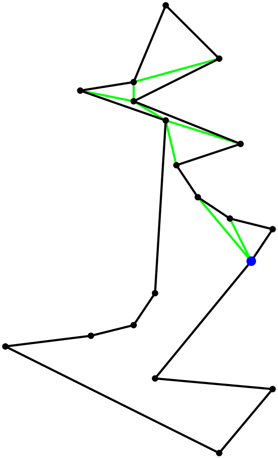
Triangulating a monotone polygon

How to triangulate a y-monotone polygon?



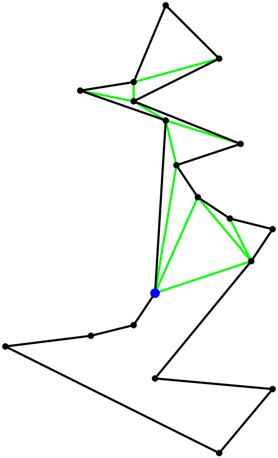
Triangulating a monotone polygon

How to triangulate a y-monotone polygon?



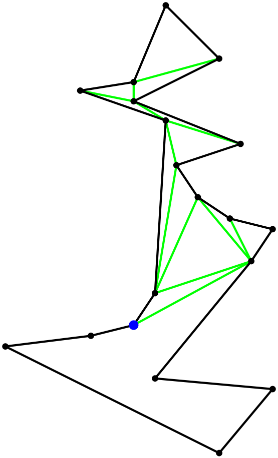
Triangulating a monotone polygon

How to triangulate a y-monotone polygon?



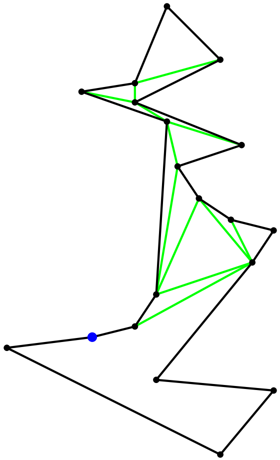
Triangulating a monotone polygon

How to triangulate a y-monotone polygon?



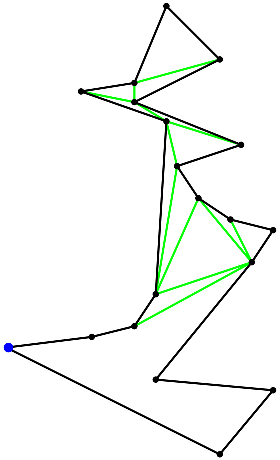
Triangulating a monotone polygon

How to triangulate a y-monotone polygon?



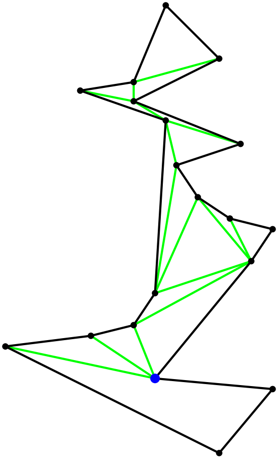
Triangulating a monotone polygon

How to triangulate a y-monotone polygon?



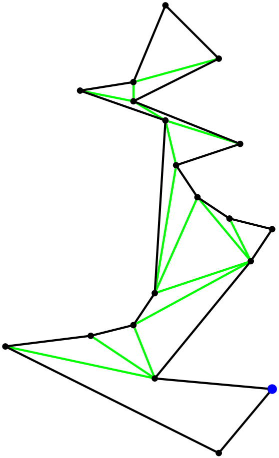
Triangulating a monotone polygon

How to triangulate a y-monotone polygon?



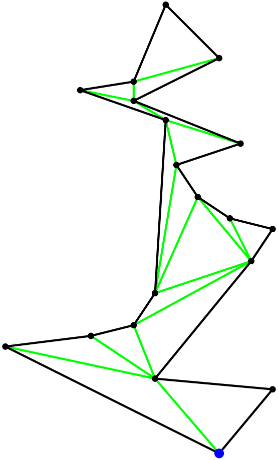
Triangulating a monotone polygon

How to triangulate a y-monotone polygon?



Triangulating a monotone polygon

How to triangulate a y-monotone polygon?



The algorithm

- Sort the vertices top-to-bottom by a merge of the two chains
- Initialize a stack. Push the first two vertices
- Take the next vertex v, and triangulate as much as possible, top-down, while popping the stack
- Push v onto the stack

Result

Theorem: A simple polygon with n vertices can be partitioned into y-monotone pieces in $O(n \log n)$ time

Theorem: A monotone polygon with n vertices can be triangulated $O(n)$ time

Can we immediately conclude:

A simple polygon with n vertices can be triangulated $O(n \log n)$ time ???

Result

We need to argue that all y-monotone polygons together that we will triangulate have $O(n)$ vertices

Initially we had n edges. We add at most $n-3$ diagonals in the sweeps. These diagonals are used on both sides as edges. So all monotone polygons together have at most $3n-6$ edges, and therefore at most $3n-6$ vertices

Hence we can conclude that triangulating all monotone polygons together takes only $O(n)$ time

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