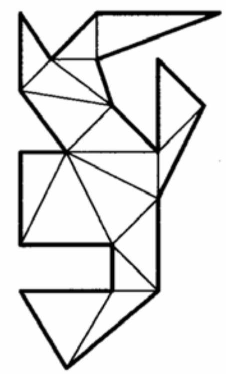
Polygon Triangulation

(slides partially by Daniel Vlasic)

Triangulation: Definition

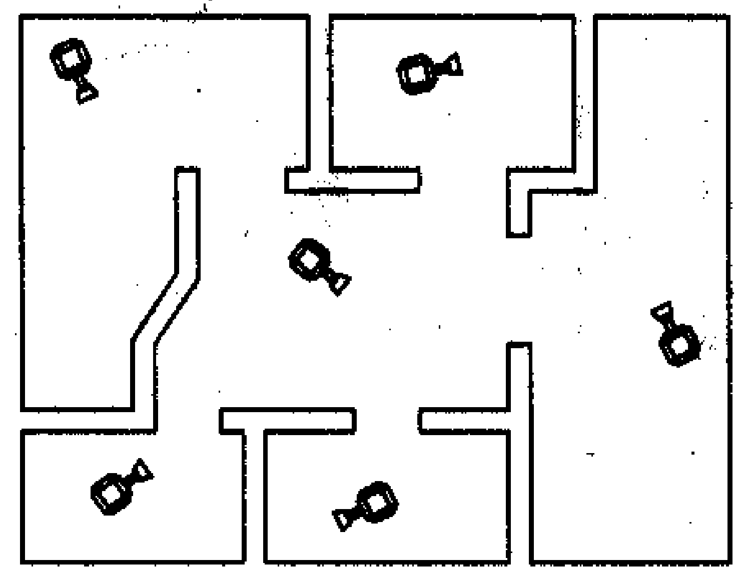
- Triangulation of a simple polygon P: decomposition of P into triangles by a maximal set of non-intersecting diagonals



- Diagonal: an open line segment that connects two vertices of P and lies in the interior of P
- Triangulations are usually not unique

Motivation: Guarding an Art Gallery

- An art gallery has several rooms
- Each room guarded by cameras that see in all directions
- Want to have few cameras that cover the whole gallery

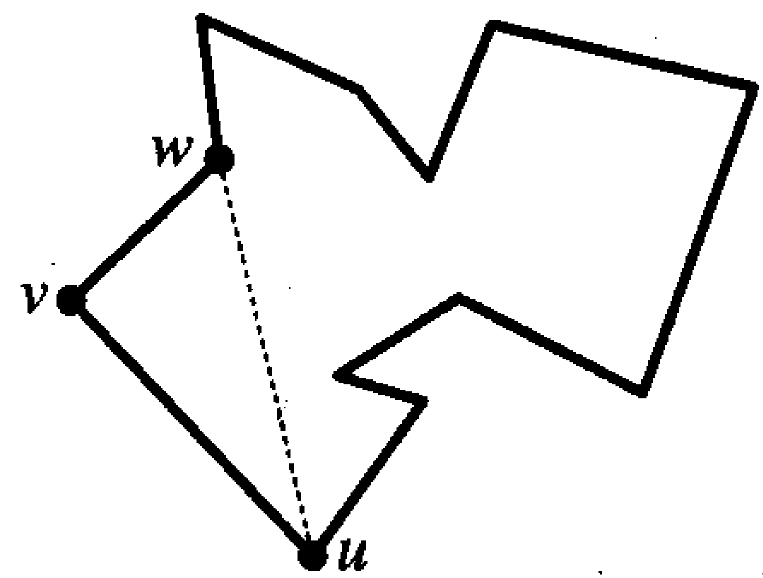


Triangulation: Existence

- Theorem:
  - *Every simple polygon admits a triangulation*
  - *Any triangulation of a simple polygon with n vertices consists of exactly $n-2$ triangles*
- Proof:
  - Base case: $n=3$
    - 1 triangle ($=n-2$)
    - trivially correct
  - Inductive step: assume theorem holds for all $m<n$

Inductive step

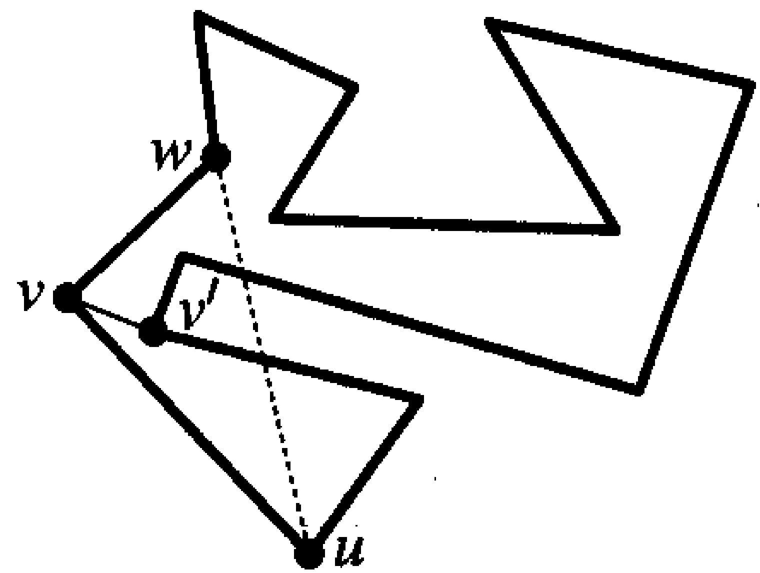
- First, prove existence of a diagonal:
  - Let *v* be the leftmost vertex of *P*
  - Let *u* and *w* be the two neighboring vertices of *v*
  - If open segment *uw* lies inside *P*, then *uw* is a diagonal



Inductive step ctd.

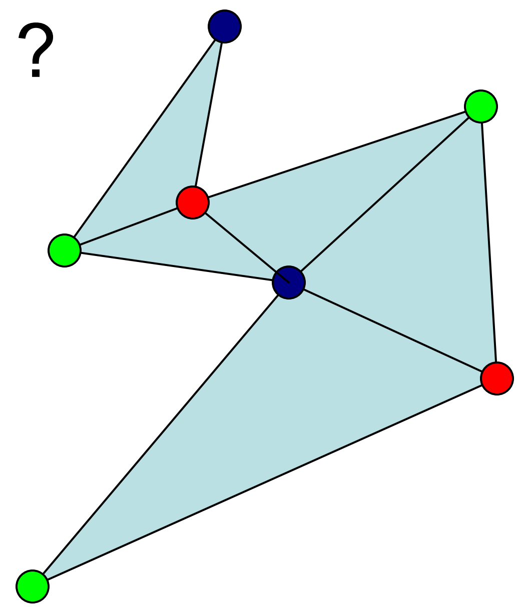
- If open segment *uw* does not lie inside *P*
  - there are one or more vertices inside triangle *uvw*
  - of those vertices, let *v'* be the farthest one from *uw*
  - segment *vv'* cannot intersect any edge of *P*, so *vv'* is a diagonal
- Thus, a diagonal exists
- Can recurse on both sides
- Math works out:

  (n1-2) + (n2-2) = (n1+n2-2)-2



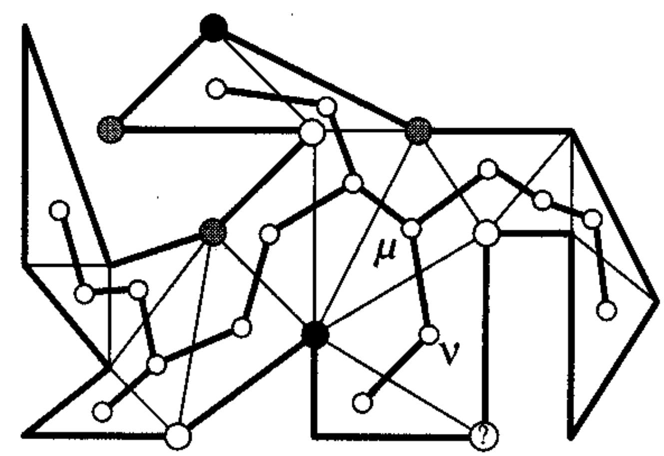
Back to cameras

- Where should we put the cameras ?
- Idea: cover every triangle
  - 3-color the nodes (for each edge, endpoints have different colors)
  - Each triangle has vertices with all 3 colors
  - Can choose the least frequent color class → $\lfloor n/3 \rfloor$ cameras suffice
  - There are polygons that require $\lfloor n/3 \rfloor$ cameras



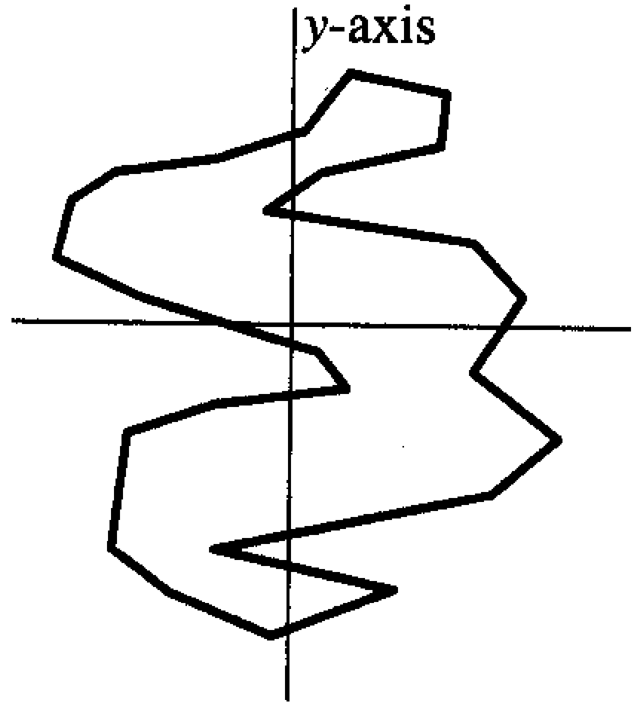
3-coloring Always Possible

- Simple inductive argument (Ilya)
- More complex, but linear-time algorithm:
  - Take the dual graph G
  - This graph has no cycles
  - Find 3-coloring by DFS traversal of G:
    - Start from any triangle and 3-color its vertices
    - When reaching new triangle we cross an already colored diagonal
    - Choose the third color to finish the triangle



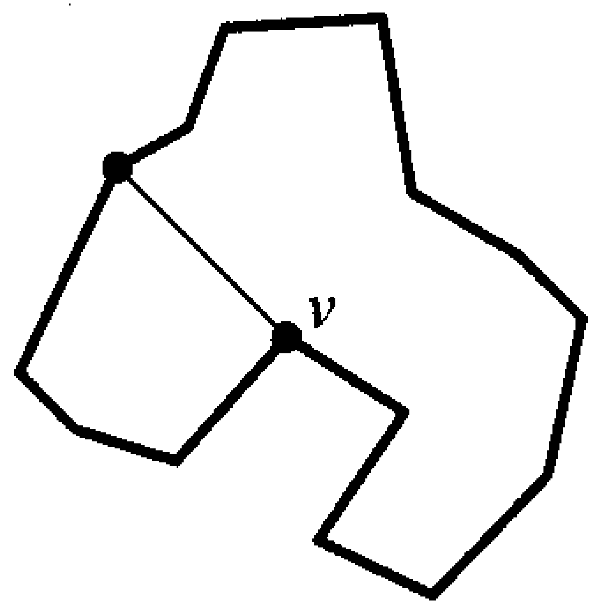
How to triangulate fast

- Partition the polygon into y-monotone parts, i.e., into polygons P such that an intersection of any horizontal line L with P is connected
- Triangulate the monotone parts

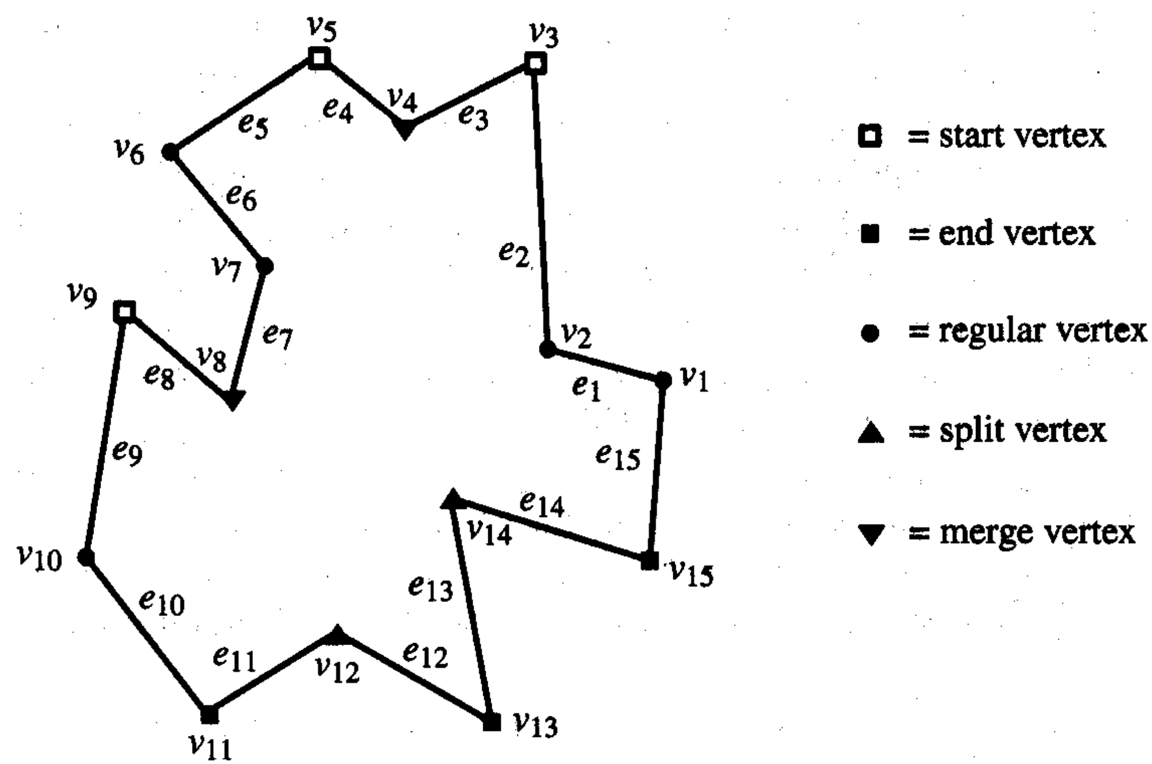


Monotone partitioning

- Line sweep (top down)
- Vertices where the direction changes downward<>upward are called *turn vertices*
- To have y-monotone pieces, we need to get rid of turn vertices:
  - when we encounter a turn vertex, it might be necessary to introduce a diagonal and split the polygon into pieces

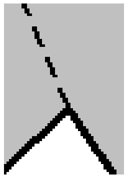


Vertex Ontology

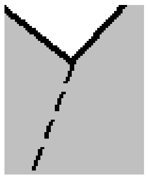


Adding diagonals

- To partition P into y-monotone pieces, get rid of split and merge vertices
  - add a diagonal going upward from each split vertex



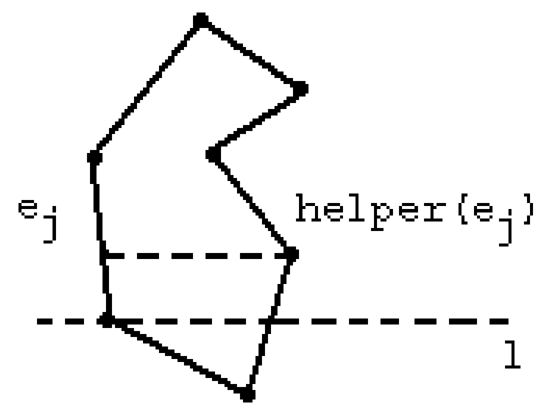
  - add a diagonal going downward from each merge vertex



- Where do the edges go ?

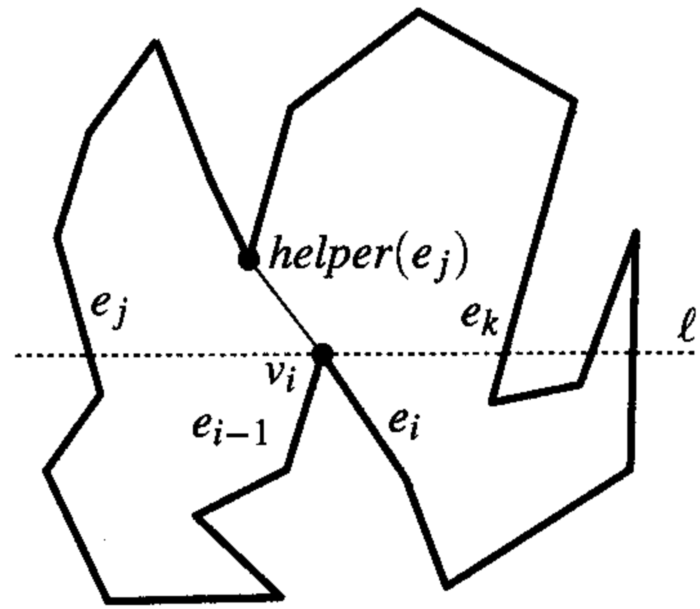
Helpers

- Let *helper*(e_j) be the lowest vertex above the sweep-line such that the horizontal segment connecting the vertex to e_j lies inside *P*



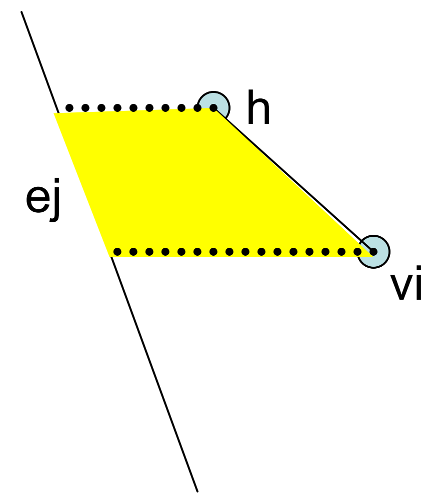
Removing Split Vertices

- For a split vertex v_i, let e_j be the edge immediately to the left of it
- Add a diagonal from v_i to *helper*(e_j)
- Question: does the new edge intersect any existing one ?



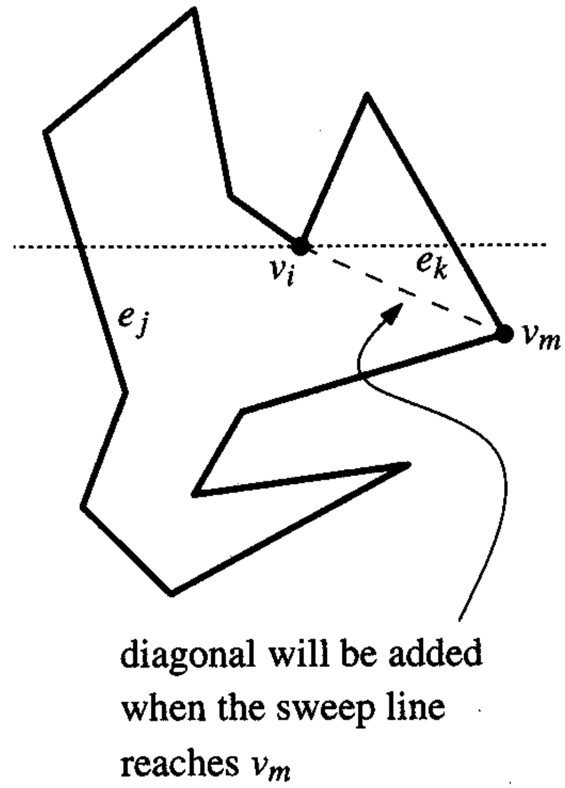
Proof

- Recall:
  - vi is the split vertex
  - ej is the edge to the left of vi
  - h = helper(ej)
- The segment h-vi does not intersect any other boundary segment because:
  - The dotted segments do not intersect any polygon boundaries, by def. Same holds for ej.
  - If there was any polygon segment intersecting h-vi, one of its endpoints would have to be inside the yellow region
  - Take the lowest point p in the region which is an endpoint of some segment. By assumptions, this point has a unique y-coordinate.
  - Thus, a left ray starting from p hits ej without touching any other edge. Thus, p is a candidate for a helper of ej.
  - Any helper of ej must lie inside the yellow region, and p is the lowest such point.
  - Thus, p is a helper – a contradiction.
- Note: the edges added earlier are, for the above purposes, considered to be the boundary edges



Removing Merge Vertices

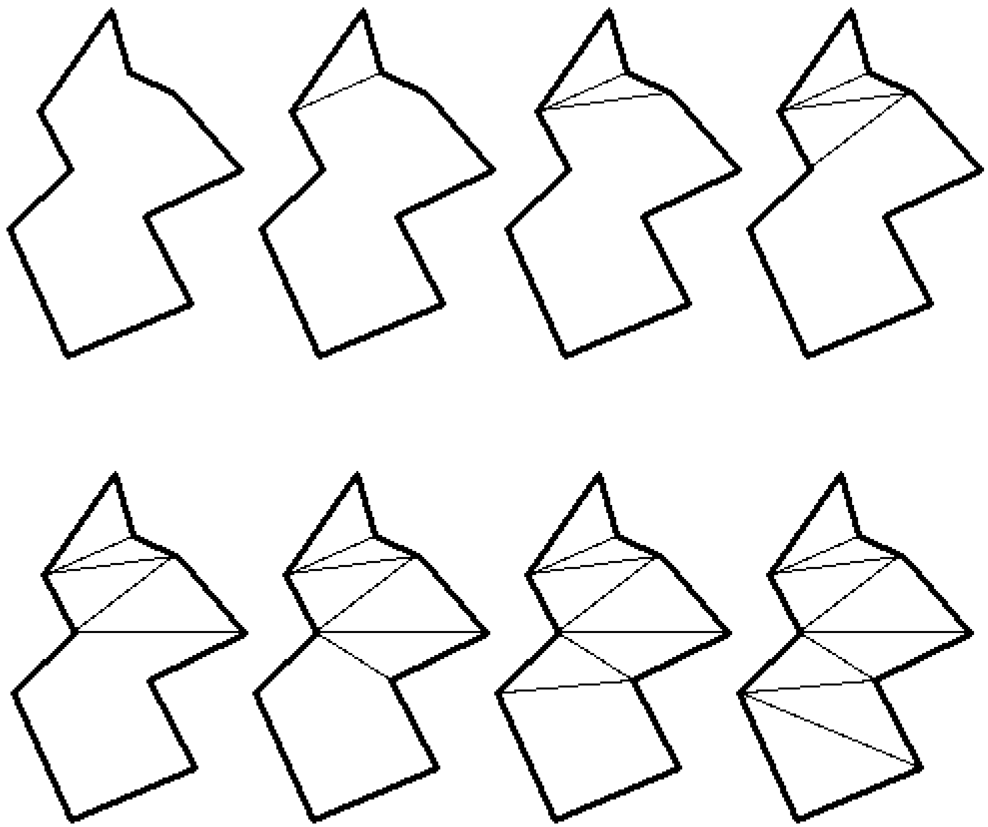
- For a merge vertex v_i, let e_j be the edge immediately to the left of it
- v_i becomes *helper*(e_j) once we reach it
- Whenever the *helper*(e_j) is replaced by some vertex v_m, add a diagonal from v_m to v_i
- If v_i is never replaced as *helper*(e_j) , we can connect it to the lower endpoint of e_j



The algorithm

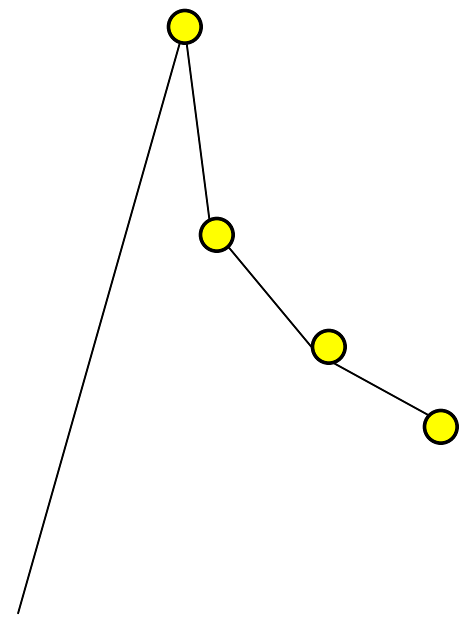
- Use plane sweep method
  - move sweep line downward over the plane (need to sort first)
  - halt the line on every vertex
  - handle the event depending on the vertex type
  - events:
    - edge starts (insert into a BST)
    - edge ends (add a diagonal if the helper is a merge vertex, remove from BST)
    - edge changes a helper (add a diagonal if old helper was a merge vertex)
    - new vertex is a split vertex (must add a diagonal)
- Time: O(n log n)

Triangulating monotone polygon



Triangulating monotone polygons

- Single pass from top to bottom
- Keeps removing triangles
- Invariants:
  - Top vertex convex
  - Other vertices form a concave chain



Altogether

- Can triangulate a polygon in $O(n \log n)$ time
- Fairly simple $O(n \log^* n)$ time algorithms
- Very complex $O(n)$ time algorithm