Computational Geometry

Chapter 3

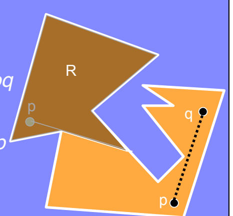
Polygons and Triangulation

On the Agenda

- The Art Gallery Problem
- Polygon Triangulation

Art Gallery Problem

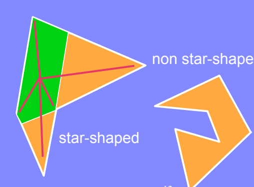
- Given a simple polygon *P*, say that two points *p* and *q* can *see* each other if the open segment *pq* lies entirely within *P*.
- A point *guards* a region $R \subseteq P$ if *p* sees all $q \in R$.
- Given a polygon *P*, what is the minimal number of guards required to guard *P*, and what are their locations ?



Observations

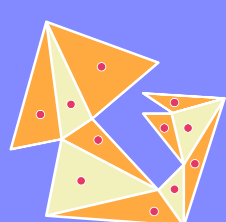
- The *entire* interior of a convex polygon is visible from *any* interior point.

- A *star-shaped* polygon requires only one guard located in its *kernel.*



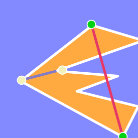
Art Gallery Problem – Easy Upper Bound

- *n*-2 guards suffice:
  - Subdivide the polygon into *n*-2 triangles (triangulation)
  - Place one guard in each triangle.
- **Theorem**: Any simple planar polygon with *n* vertices has a triangulation of size *n*-2.

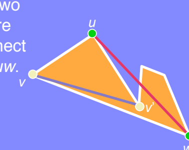


Diagonals in Polygons

- A *diagonal* of a polygon *P* is a line segment connecting two vertices which lies entirely within *P.*



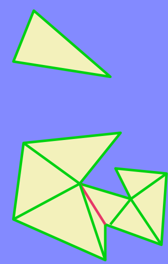
- **Theorem:** Any polygon with n>3 vertices has a diagonal, which may be found in O(*n*) time.
- **Proof:** Find the leftmost vertex *v.* Connect its two neighbors *u* and *w.* If this is not a diagonal there are other vertices inside the triangle *uvw.* Connect *v* with the vertex *v'* furthest from the segment *uw.*



- **Question:** Why not connect *v* with the second leftmost vertex?

O(n^2) Polygon Triangulation

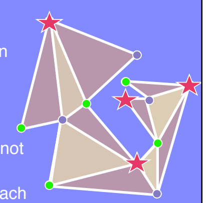
- **Theorem:** Every simple polygon with n vertices has a triangulation consisting of n-3 diagonals and n-2 triangles.
- **Proof:** By induction on n:
  - Basis: A triangle (n=3) has a triangulation (itself) with no diagonals and one triangle.
  - Induction: for a n+1 vertex polygon, construct a diagonal dividing the polygon into two polygons with n_1 and n_2 vertices such that $n_1+n_2-2=n$. Triangulate the two parts of the polygon. There are now $n_1-3+n_2-3+1=n-3$ diagonals and $n_1-2+n_2-2=n-2$ triangles.



73.

Art Gallery Problem – Upper Bound

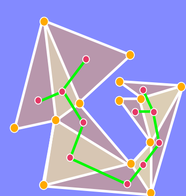
- Color the triangulated polygon vertices with three colors such that there is no edge between two vertices with the same color.
- Pick a color which is least used. This color cannot appear more than ⌊n/3⌋ times.
- Place a guard on each vertex with this color. Each triangle has only one guarded vertex, but this guard covers all triangles incident on the vertex.
- ⇒ New upper bound: ⌊n/3⌋



83.

3-Coloring

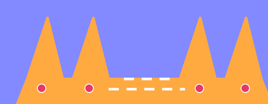
- **Theorem:** Every triangulated polygon has an *ear* (a triangle containing two boundary edges).
- **Proof:** Consider the *dual* to the triangulation.
  - Since any diagonal disconnects the polygon, the dual is a tree.
  - The vertex degrees are bounded by 3.
  - Ears correspond to leaves in the dual.
- **Theorem:** Every triangulated polygon may be 3-colored.
- **Proof:** By induction on n:
  - Basis: Trivial if n=3.
  - Induction: Cut off an ear. 3-color the n-1 vertex remainder. Color the n'th vertex with the third color different from the two on its supporting edge.



93.

⌊n/3⌋ - Tight Bound

- There exists a polygon with n vertices, for which n/3 guards are necessary.

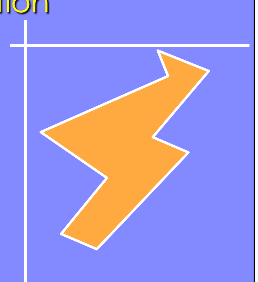


103.

O(n log n)-Time Polygon Triangulation

- A simple polygon is called *monotone* with respect to a line ℓ if for any line ℓ' perpendicular to ℓ the intersection of the polygon with ℓ' is connected.
- A polygon is called *monotone* if there exists any such line ℓ.
- A polygon that is monotone with respect to the x/y-axis is called *x/y-monotone.*

Question: How can we check in O(n) time whether a polygon is y-monotone?



y-monotone but not x-monotone polygon

113.

Triangulation Algorithm – cont.

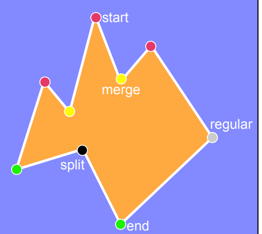
1) Partition the polygon into y-monotone pieces.

2) Triangulate each y-monotone piece separately.

123.

Y-Monotone Polygons

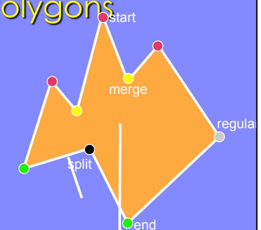
- Classify polygon vertices:
  - A *start* (resp., *end*) vertex is one whose interior angle is less than π and its two neighboring vertices both lie below (resp., above) it.
  - A *split* (resp., *merge*) vertex is one whose interior angle is greater than π and its two neighboring vertices both lie below (resp., above) it.
  - All other vertices are *regular*.



133.

Y-Monotone Polygons

Observation: A polygon without *split* nor *merge* vertices is *y* monotone.

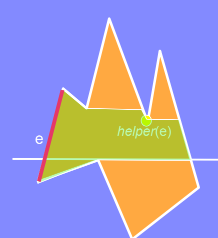


- To partition a polygon to monotone pieces, eliminate split (merge) vertices by adding diagonals upward (downward) from the vertex.
- Naturally, the diagonals must not intersect!

143.

Monotone Partitioning

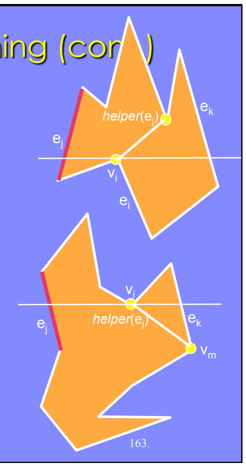
- Classify all vertices.
- Sweep the polygon from top to bottom.
- Maintain the edges intersected by the sweep line in a *sweep line status* (SLS sorted by *x* coordinate).
- Maintain vertex events in an event queue (EQ sorted by *y* coordinate).
- Eliminate split/merge vertices by connecting them to other vertices.
- For each edge *e*, define *helper*(*e*) as the lowest vertex (seen so far) above the sweep line **visible** to the right of the edge.
- *helper*(*e*) is initialized to the upper endpoint of *e*.



153.

Monotone Partitioning (cont.)

- A split vertex may be connected to the helper vertex of the edge immediately to its left (along the sweep line).
- However, a merge vertex should be connected to a vertex which has not yet been processed !
- Clever idea: Every merge vertex is the helper of some edge, and will be handled when this edge "terminates".



163.

Monotone Partitioning Algorithm

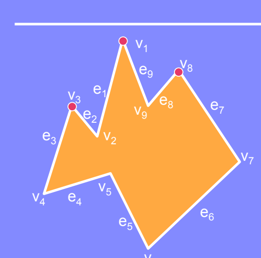
- Input: A counterclockwise ordered list of vertices. The edge e_i immediately follows the vertex v_i.
- Construct an EQ on the vertices of P using *y*-coordinates. (In case two or more vertices have the same *y*-coordinates, the vertex with the smaller *x*-coordinate has a higher priority.)
- Initialize SLS to be empty.
- While EQ is not empty:
  - Pop vertex v;
  - Handle v.

  (No new events are generated during execution.)
- Idea: No split/merge vertex remains unhandled!

173.

Monotone Partitioning

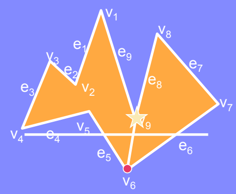
- Handling a *start* vertex (v_i):
  - Add e_i to SLS
  - *helper*(e_i) := v_i



183.

Monotone Partitioning

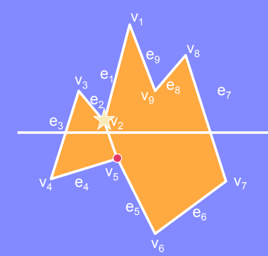
- Handling an *end* vertex (v_i):
  - If *helper*(e_{i-1}) is a merge vertex, then connect v_i to *helper*(e_{i-1})
  - Remove e_{i-1} from SLS



193.

Monotone Partitioning

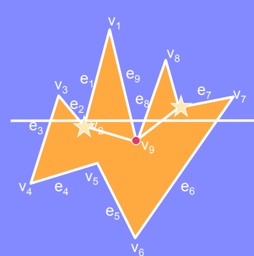
- Handling a *split* vertex (v_i):
  - Find in SLS the edge e_j directly to the left of v_i
  - Connect v_i to *helper*(e_j)
  - *helper*(e_j) := v_i
  - Insert e_i into SLS
  - *helper*(e_i) := v_i



203.

Monotone Partitioning

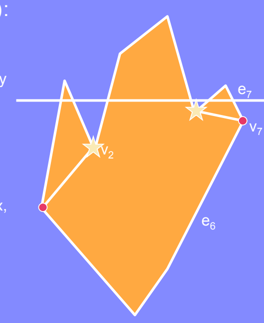
- Handling a *merge* vertex (v_i):
  - If *helper*(e_{i-1}) is a merge vertex, then connect v_i to *helper*(e_{i-1})
  - Remove e_{i-1} from SLS
  - Find in SLS the edge e_j directly to the left of v_i
  - If *helper*(e_j) is a merge vertex, then connect v_i to *helper*(e_j)
  - *helper*(e_j) := v_i



213.

Monotone Partitioning

- Handling a *regular* vertex (v_i):
  - If the polygon's interior lies to the left of v_i then:
    - Find in SLS the edge e_j directly to the left of v_i
    - If *helper*(e_j) is a merge vertex, then connect v_i to *helper*(e_j)
    - *helper*(e_j) := v_i
  - Else:
    - If *helper*(e_{i-1}) is a merge vertex, then connect v_i to *helper*(e_{i-1})
    - Remove e_{i-1} from SLS
    - Insert e_i into SLS
    - *helper*(e_i) := v_i

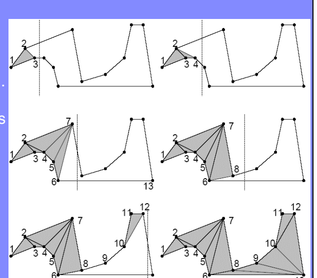


223.

Triangulating a Y-monotone Polygon

In Theory

- Sweep the polygon from top to bottom.
- Greedily triangulate anything possible above the sweep line, and then forget about this region.
  - When we process a vertex *v*, the unhandled region above it always has a simple structure: Two *y*-monotone (left and right) chains, each containing at least one edge. If a chain consists of two or more edges, it is *reflex*, and the other chain consists of a single edge whose bottom endpoint has not been handled yet.
  - Each diagonal is added in O(1) time.

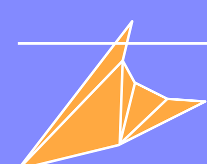


233.

Triangulating a Y-monotone Polygon

In Practice

- Continue sweeping while one chain contains only one edge, while the other edge is *concave.*
- When a "convex edge" appears in the concave chain (or a second edge appears in the other one), triangulate as much as possible using a "fan".
- Time complexity: O(*n*), where *n* is the complexity of the polygon.



Question: Why?!

243.