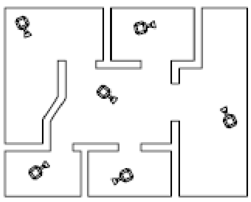
Polygon Triangulation

Guarding an Art Gallery

Min-Te Sun, Ph.D.

Guarding an Art Gallery

- How many cameras are needed?
- Where to place these cameras?
- We would like to minimize the number of cameras!



Art Gallery Model

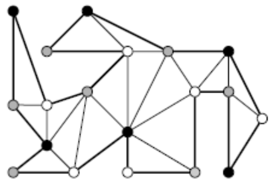
- If we model an art gallery by a simple polygon P, then
  - the minimum number of cameras to guard it is NP-hard
  - the conservative approach is to place one camera at each vertex
- Can we do better?

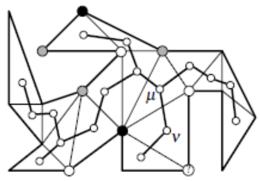
Triangulate Simple Polygon

- Every simple polygon P admits a triangulation T_P, and any triangulation of a simple polygon with n vertices consists of exactly n−2 triangles
  - n-2 cameras?
  - n/2 cameras?
  - Can we do better?

3-Coloring

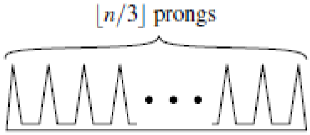
- 3-coloring of T_P
  - Each vertex of P is assigned a color white, gray, or black
  - The coloring will be such that any two vertices connected by an edge or a diagonal have different colors
- Dual graph of T_P, $G(T_P)$, has a node for every triangle in T_P
  - the triangle corresponding to a node v is denoted by t(v).
  - There is an arc between two nodes v and μ if t(v) and t(μ) share a diagonal





The Property of Dual Graph

- $G(T_P)$ is a tree, because
  - $G(T_P)$ is connected
  - The removal of any diagonal cuts P into two => The removal of any edge from $G(T_P)$ splits $G(T_P)$
- How to do 3-coloring? => DFS is started from any node of $G(T_P)$ for coloring
  - Because $G(T_P)$ is a tree, the node adjacent to the newly visit one have not been visited before
- We now only need floor(n/3) cameras
  - Can we do better?

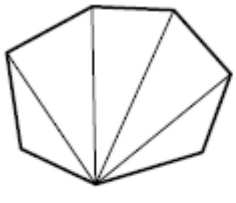


Art Gallery Theorem

- For a simple polygon with n vertices,
- floor(n/3) cameras are occasionally necessary and always sufficient to have every point in the polygon visible from at least one of the cameras.

Triangulation of Simple Polygon

- Can be easily done in $O(n^2)$
  - Can we do better?
- If the polygon can be divided into several convex polygon, then triangulation can be done in O(n)
  - Not easily doable in most cases



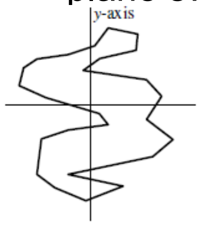
Triangulation of Simple Polygon

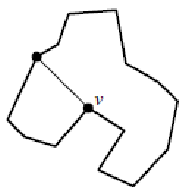
- Idea Sketch

1. Partition a polygon into y-monotone pieces
2. Triangulate each y-monotone piece separately

y-Monotone Piece

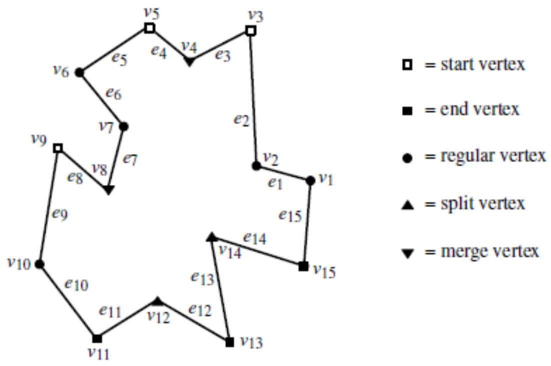
- A simple polygon is monotone with respect to a line *l* if for any line *l'* perpendicular to *l* the intersection of the polygon with *l'* is connected.
  - How to partition simple polygon into monotone pieces? => plane sweep!





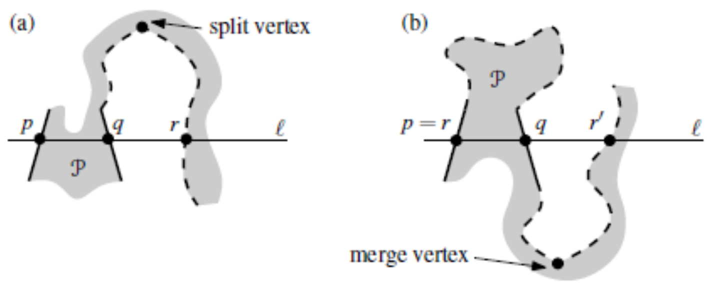
Five Types of Vertices

- Start vertex
- Split vertex
- End vertex
- Merge vertex
- Regular vertex



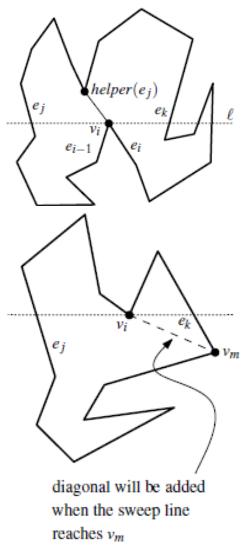
y-monotone vs Split/Merge Vertices

- A polygon is y-monotone if it has no split vertices or merge vertices



Removal of Split/Merge Vertices

- Add diagonal
  - going upward from each split vertex
  - going downward from each merge vertex
- *helper(e_j)* is defined as the lowest vertex above the sweep line such that the horizontal segment connecting the vertex to e_j lies inside P
  - *helper(e_j)* can be the upper endpoint of e_j itself

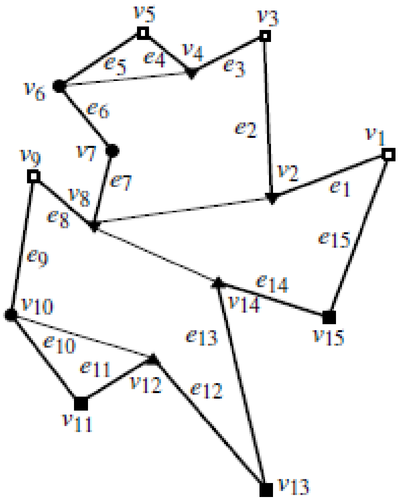


MAKEMONOTONE Algorithm

Algorithm MAKEMONOTONE($\mathcal{P}$)
Input. A simple polygon $\mathcal{P}$ stored in a doubly-connected edge list $\mathcal{D}$.
Output. A partitioning of $\mathcal{P}$ into monotone subpolygons, stored in $\mathcal{D}$.

1. Construct a priority queue $\mathcal{Q}$ on the vertices of $\mathcal{P}$, using their y-coordinates as priority. If two points have the same y-coordinate, the one with smaller x-coordinate has higher priority.
2. Initialize an empty binary search tree $\mathcal{T}$.
3. **while** $\mathcal{Q}$ is not empty
4.   **do** Remove the vertex v_i with the highest priority from $\mathcal{Q}$.
5.   Call the appropriate procedure to handle the vertex, depending on its type.

Example

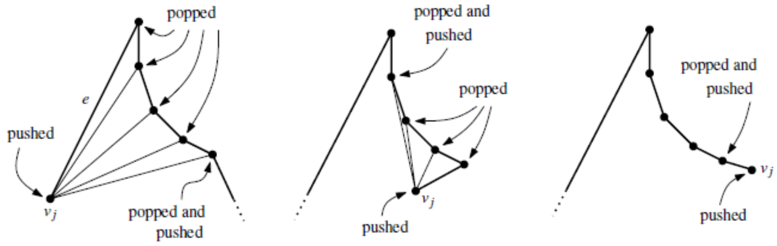


Handle of Different Vertices

- One subroutine for each type of vertices
- The doubly connected edge list is used to store the information of the polygon before and after adding diagonals
- If multiple points have the same y value, the one with smaller x value take precedence
- A simple polygon with n vertices can be partitioned into y-monotone polygons in O(nlogn) time using O(n) storage

Triangulate y-Monotone Polygon

- Idea sketch
  - Break vertices into left chain and right chain
  - Maintain a stack keeping vertices have not handled
1. If next vertex is on different chain, pop all vertices in stack and create diagonals
2. If next vertex is on the same chain, then
   - pop out vertices if diagonals can be created
   - push the current vertex otherwise



Triangulation Algorithm

Algorithm TRIANGULATEMONOTONEPOLYGON($\mathcal{P}$)
Input. A strictly y-monotone polygon $\mathcal{P}$ stored in a doubly-connected edge list $\mathcal{D}$.
Output. A triangulation of $\mathcal{P}$ stored in the doubly-connected edge list $\mathcal{D}$.

1. Merge the vertices on the left chain and the vertices on the right chain of $\mathcal{P}$ into one sequence, sorted on decreasing y-coordinate. If two vertices have the same y-coordinate, then the leftmost one comes first. Let $u_1, \dots, u_n$ denote the sorted sequence.
2. Initialize an empty stack $\mathcal{S}$, and push u_1 and u_2 onto it.
3. **for** $j \leftarrow 3$ **to** $n-1$
4.   **do if** u_j and the vertex on top of $\mathcal{S}$ are on different chains
5.   **then** Pop all vertices from $\mathcal{S}$.
6.   Insert into $\mathcal{D}$ a diagonal from u_j to each popped vertex, except the last one.
7.   Push u_{j-1} and u_j onto $\mathcal{S}$.
8.   **else** Pop one vertex from $\mathcal{S}$.
9.   Pop the other vertices from $\mathcal{S}$ as long as the diagonals from u_j to them are inside $\mathcal{P}$. Insert these diagonals into $\mathcal{D}$. Push the last vertex that has been popped back onto $\mathcal{S}$.
10.   Push u_j onto $\mathcal{S}$.
11. Add diagonals from u_n to all stack vertices except the first and the last one.

Additional Notes

- In the first case, the shape ready to be processed looks like a overturned funnel
  - The consecutive vertices stored in the stack always have angle greater than π
- A y-monotone polygon with n vertices can be triangulated in O(n)
- The algorithm is actually similar to the convex hull algorithm in Chap 1

End of Story

- A simple polygon with n vertices can be triangulated in O(nlogn) time with an algorithm that uses O(n) storage
- A planar subdivision with n vertices in total can be triangulated in O(nlogn) time with an algorithm that uses O(n) storage