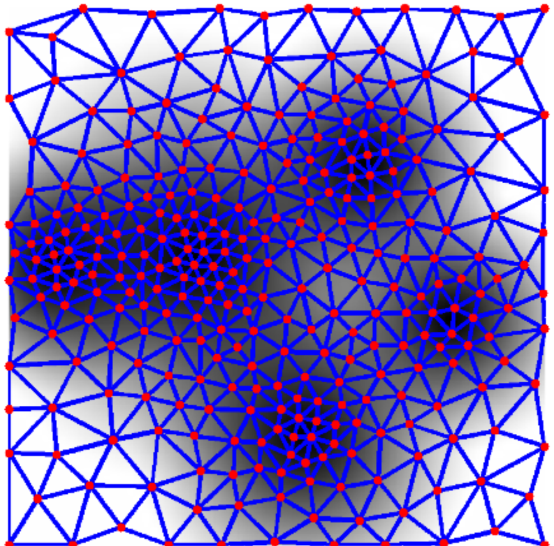
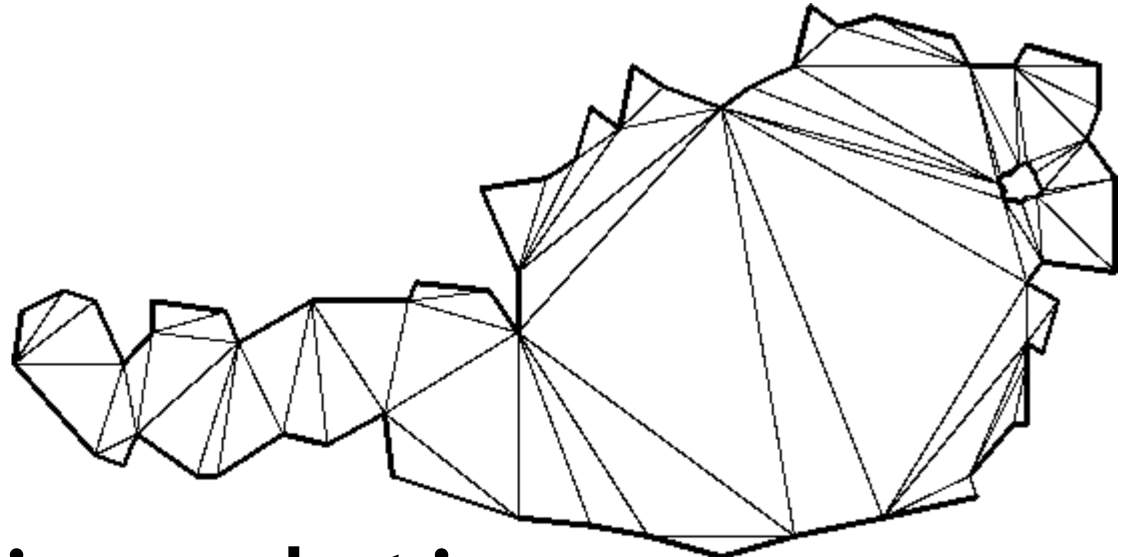
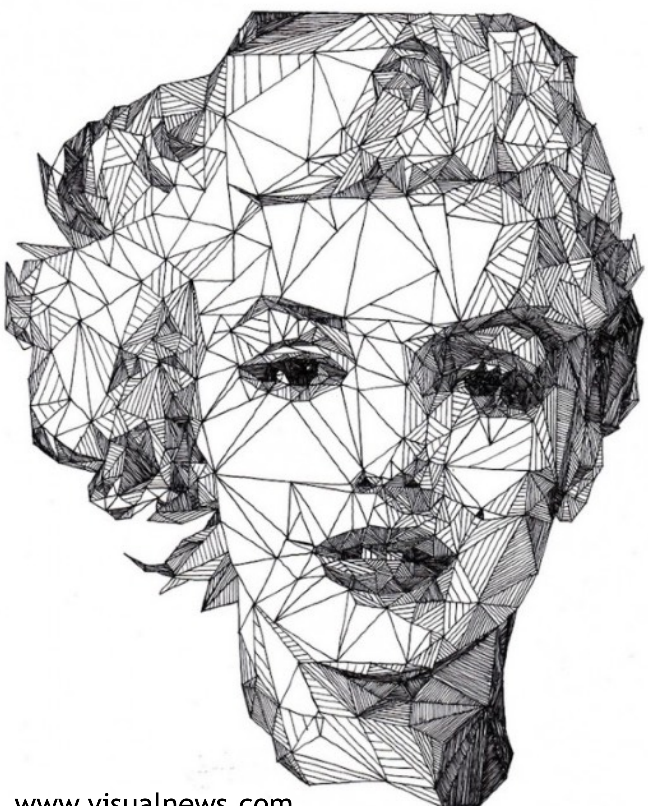
Triangulation



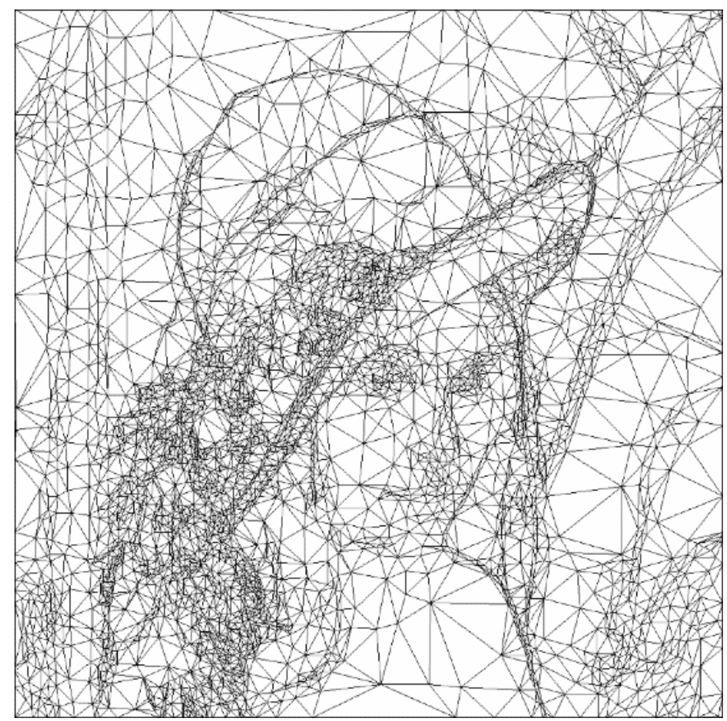
www.ceremade.dauphine.fr



www.iue.tuwien.ac.at



www.visualnews.com



www.cescg.org

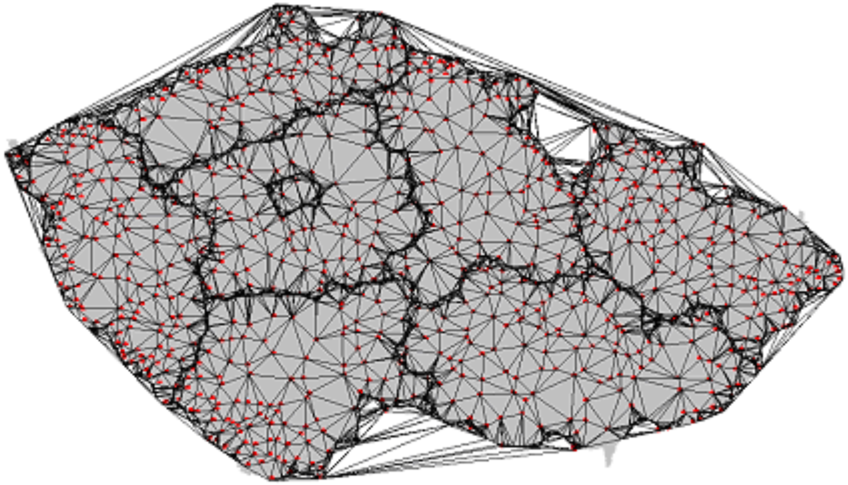
Triangulation

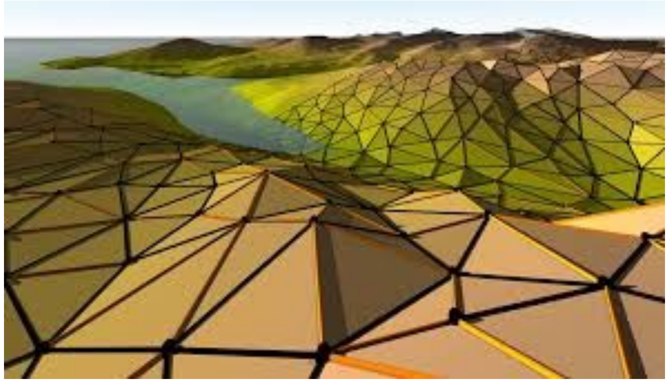
- Dividing a polygon to a set of triangles
- Often with the constraint that each triangle edge is fully shared by two triangles
- In 1925, it was proved that each surface can be triangulated

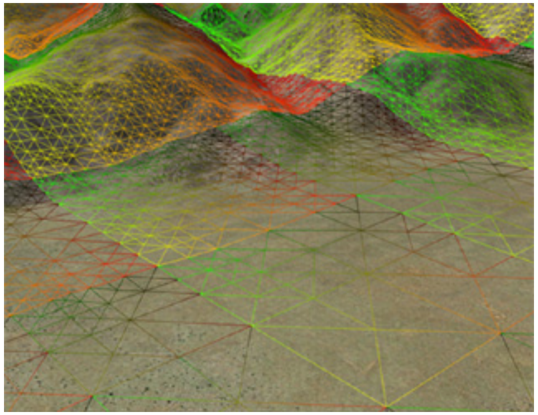
Usage of triangulation

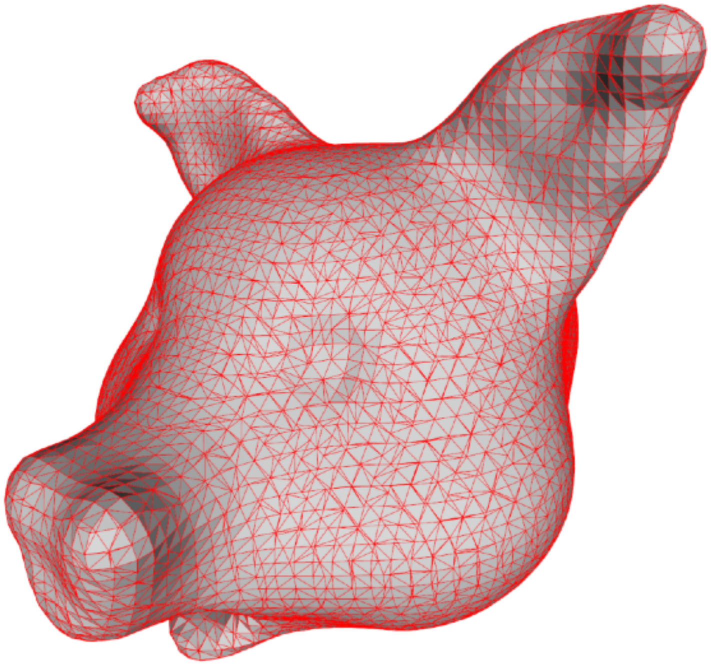
- Cartography, GIS
- Image processing – segmentation, pattern recognition
- Creating spatial models from laser scanning
- Spatial data visualization
- Finite element level set method – analysis of material structure and properties, simulation
- Robot motion planning
- Simulation of natural phenomena – erosion
- Interpolation – transfer of point clouds to surfaces
- Biometry – fingerprints detection

Usage of triangulation









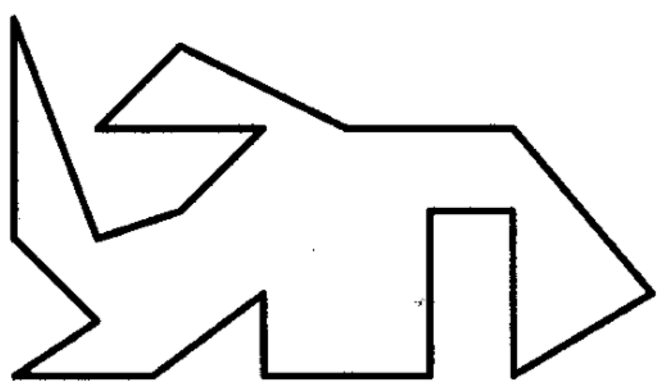
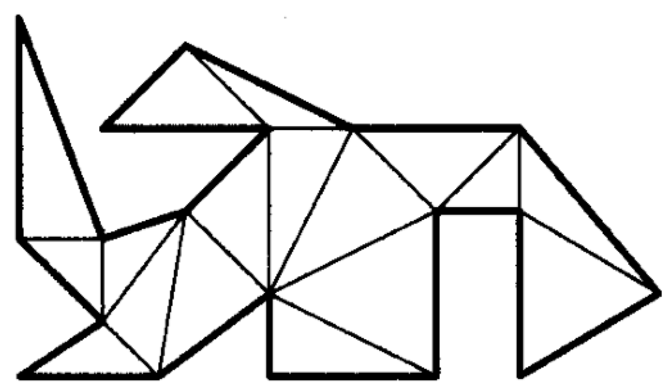
Triangulation

- Set of triangles $T = T_i$, $i = 1, \ldots, n$ is considered to be a triangulation when:
  - an arbitrary pair of triangles from *T* mutually intersects in one common vertex or along a common edge
  - union of triangles from *T* is a continuous set

- Generally, the input is a continuous polygon which does not have to be necessarily convex and can contain holes

Triangulation

- Triangulation of a simple polygon P = dividing P to triangles by a set of non-intersecting lines, connecting two vertices from P and fully lying inside P
- Triangulation is mostly non-unique

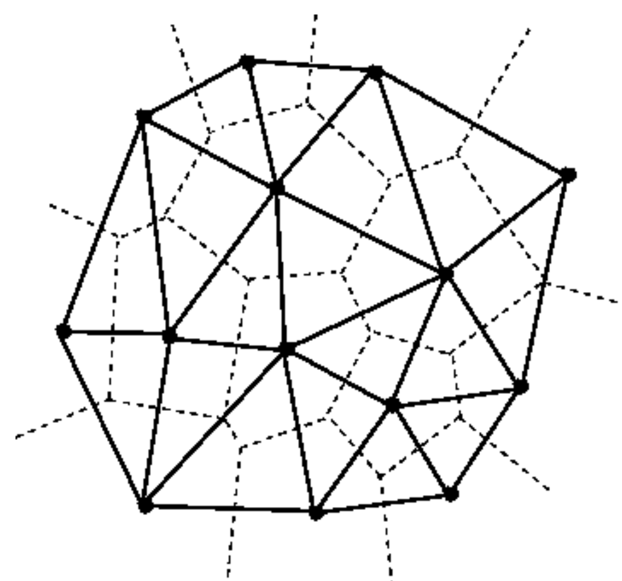



Triangulation

- Triangulation is the basic problem of computational geometry – dividing complex objects to simple ones
- The most simple objects are triangles in 2D (tetrahedra in 3D)

Triangulation

- There are several types of triangulation, e.g.:
  - Delaunay triangulation – from all existing triangulations, it has the smallest sum of the lenghts of all its edges, it is dual to the Voronoi diagram
    - we will implement it later
      in the semester



Triangulation

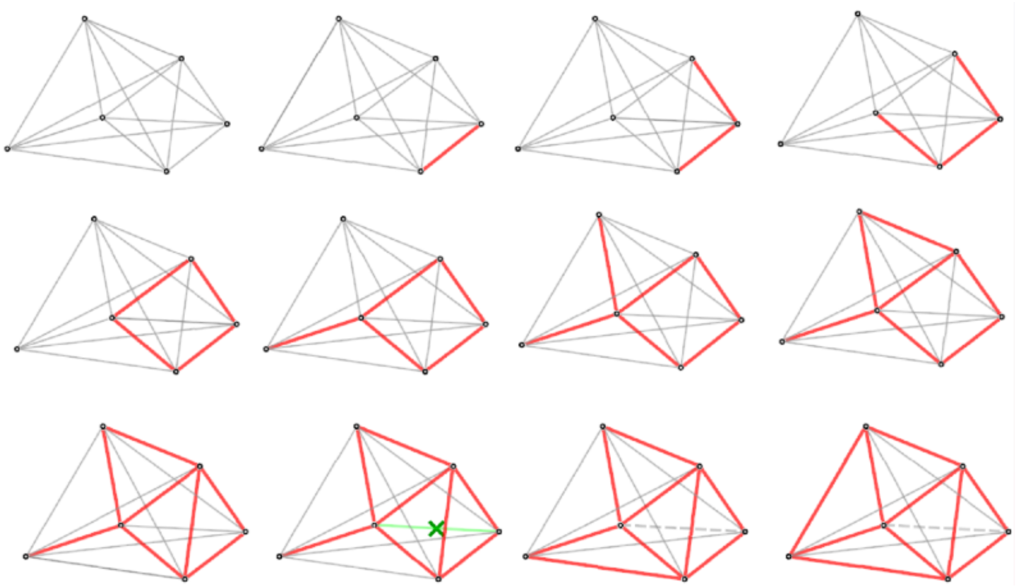
- For a given set of points (or a polygon), there are several possible triangulations. But all of them have the same number of triangles – triangulated polygon with *n* edges has *n* – *2* triangles.
- Some polygons can be triangulated easily – e.g., **convex** ones
- Non-convex polygons have to be first divided to so-called **monotone** polygons. These can be then easily triangulated.

Greedy triangulation

- Naïve approach
- Creates all potential edges, sorts them according their length in an ascending order (the number of these edges is $n(n-1)/2$)
- The edges are one by one added to the resulting triangulation, we start with the shortest one
- The algorithm ends when the list of edges is empty or when the number of edges in the triangulation is $3n - 6$

Greedy triangulation

- Criterion for adding the edge:
  - Edge is added when it does not intersect with any other edge already present in the triangulation



Greedy triangulation

repeat for all p_i, $i \in [1, n]$:

    repeat for $j \in [i + 1, n]$:

        create edge $e = (p_i, p_j)$

        for e compute $d = dist(p_i, p_j)$ and store to Q

sort Q according to d

remove $Q[0]$ and add it to T

until Q not empty

    $e = pop(Q)$

    repeat for all $e_i \in T$:

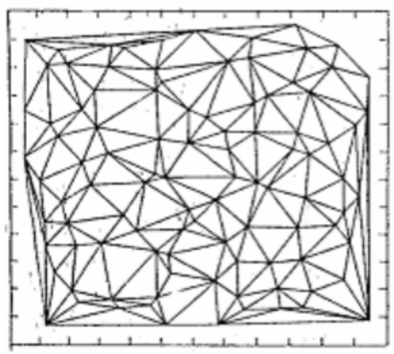
        test if e intersects with $e_i \in T$

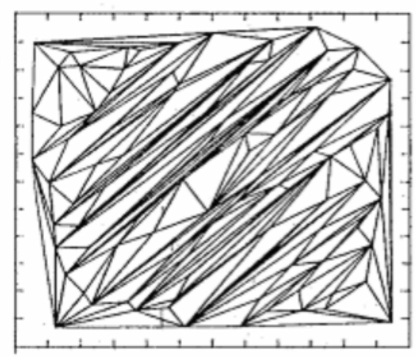
    if e does not intersect with any $e_i \in T$:

        add e to T

Greedy triangulation

- Triangles do not have to fulfill any special condition – the triangulation can contain "ugly" triangles


1a)


1b)

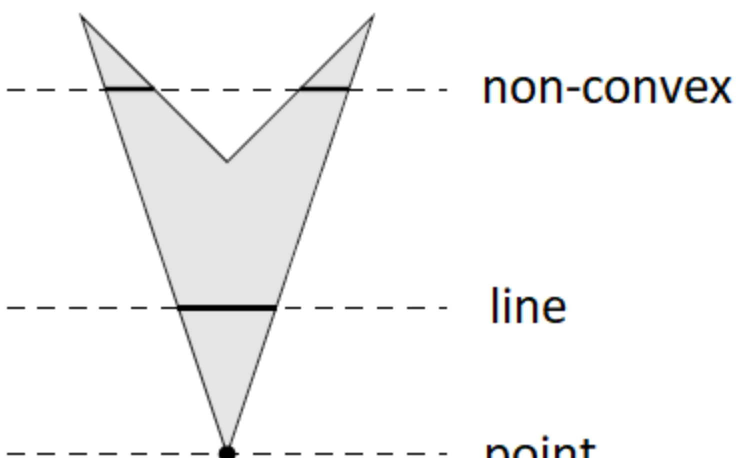
- Complexity $O(n^3)$, can be optimized to $O(n^2 \log n)$

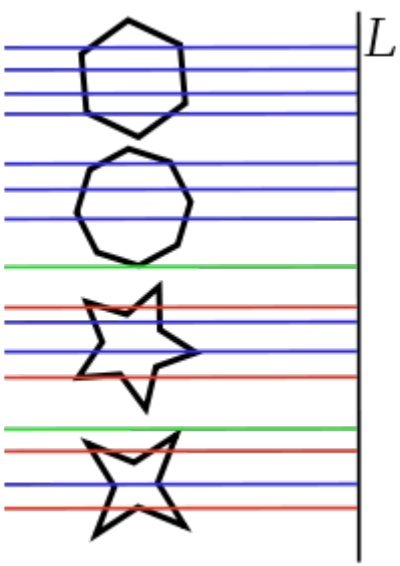
Triangulation using sweep line

- For simplicity, let's assume that we are triangulating a monotone polygon…

Monotone polygon

- Polygon is monotone when its intersection with each horizontal line is convex (it is empty set, point, or line) – the orientation of the polygon matters!





Triangulation using sweep line

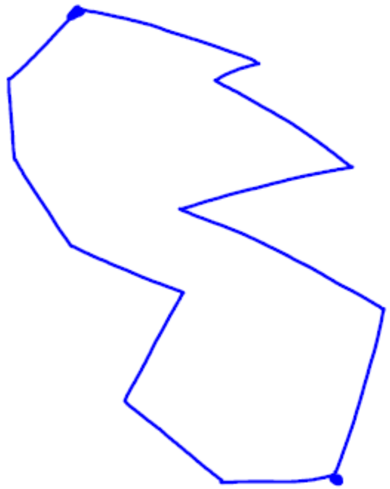
- 1st step: Lexicographically sort the vertices of the convex hull

$$p > q \Leftrightarrow p_y > q_y \text{ or } p_y = q_y \text{ and } p_x < q_x$$

Triangulation using sweep line

- We determine the left and right path (split at minimal and maximal point according to lexicographical sorting) – they are stored in two **queues**

left path



right path

Triangulation using sweep line

- Algorithm is trying to create new triangle always when the sweep line intersects with a vertex of the polygon
- We use another data structure – **stack**. It will contain vertices above the sweep line (already traversed ones), which were not yet triangulated

Triangulation using sweep line

sort vertices $v_1, v_2, \ldots, v_n$ lexicographically

put v_1, v_2 to stack

for $i = 3$ *to* n:

if v_i and the top of the stack lie on the same path (left or right)

add edges $v_iv_j, \ldots, v_iv_k$, where v_k is the last vertex forming the "correct" line

pop $v_j, \ldots, v_k$ and push v_i

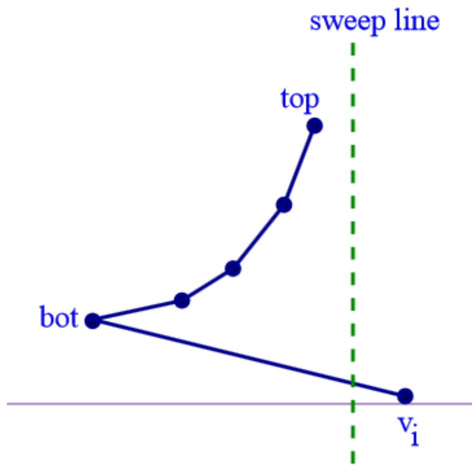
else

add edges from v_i to all vertices stored in stack and remove (pop) them from stack

store v_{top}

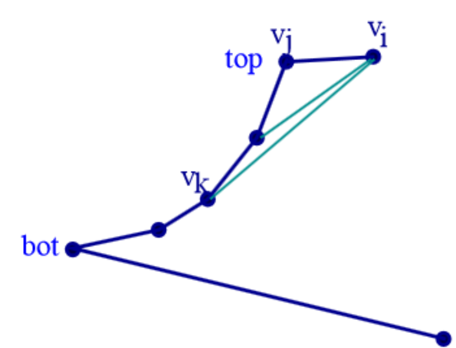
push v_{top} and v_i

Triangulation using sweep line



First branch of the ***if*** condition:

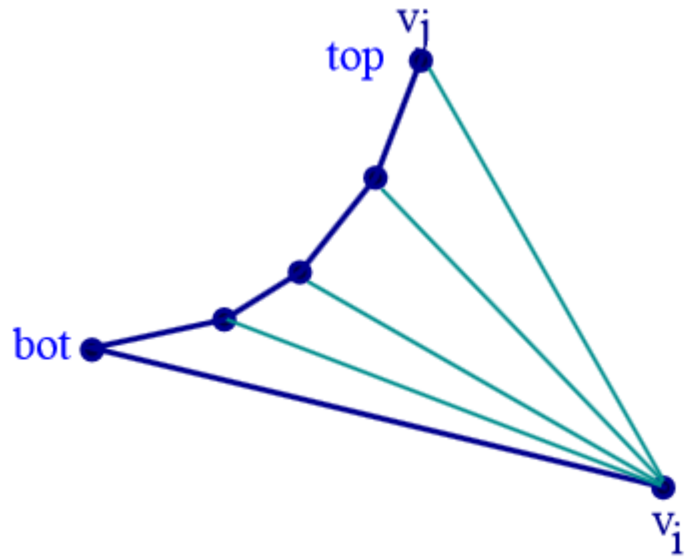
Stack will contain (bot, …, v_k, v_i)



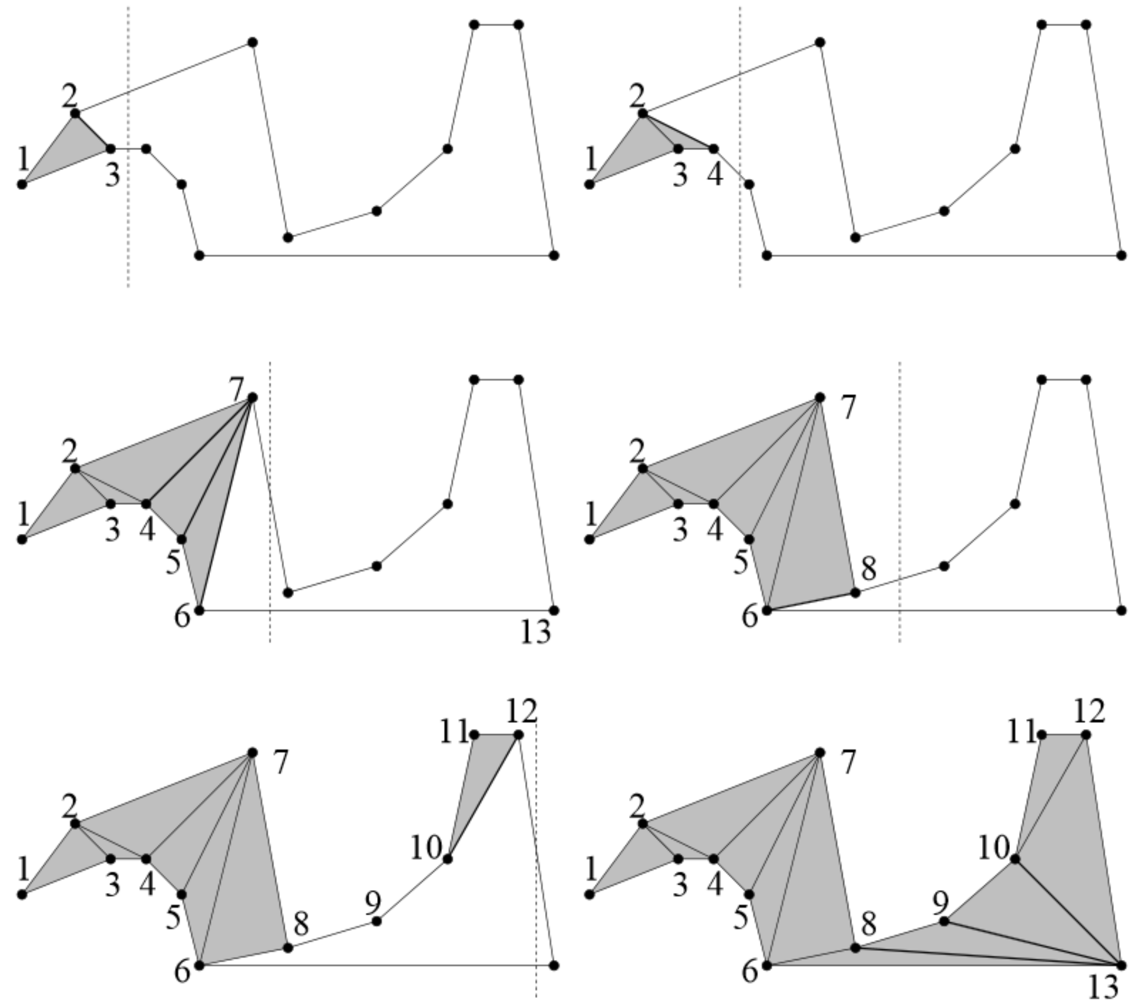
Triangulation using sweep line

else branch of the ***if*** condition:

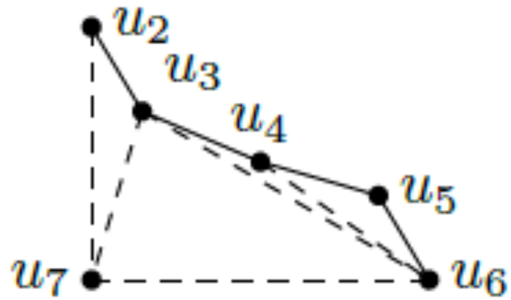
Stack will contain (v_j, v_i)

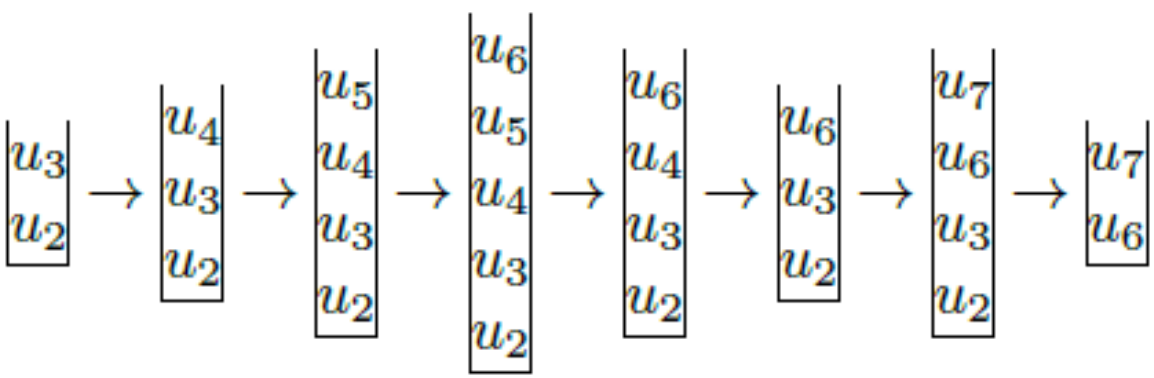


Triangulation using sweep line



Yet another example





Time complexity

- Each vertex is added to the stack only once – when "visited", it is removed from stack
- In each step we add at least one edge
- Total triangulation time: $O(n \log n)$

Your assignment

- Implement the sweep line algorithm for polygon triangulation
- Our input data:
  - Convex hull (created in previous assignments)
  - Arbitrary polygon (has to be added to the basic framework – simple connection of points added by the user to the scene. We connect them in the same order as they were inserted to the scene + connecting the first and last point to close the polygon. We skip the test for monotonity (we assume that the user creates a monotone polygon, if not, we are fine with wrong result ☺️)

Your assignment

- Your algorithm should be able to work with both mentioned inputs