1 Previous lecture

We described triangulations of simple polygons:

And described quadratic time algorithm to compute a triangulation.

2 Monotone Polygons - the easy case

Definition 2.1 A curve $\gamma$ is $x$-monotone if any vertical line either does not intersect $\gamma$, or it intersect $\gamma$ in a single point.

Definition 2.2 A polygon is $x$-monotone, if its boundary can be partitioned into two chains that are $x$ monotone.
Let us think that we are given a monotone polygon as two sorted linked lists of vertices on the upper chain and lower chain.

**Definition 2.3** A vertex $v$ of a polygon is reflex, if the interior angle at this vertex exceeds $\pi$. It is convex if the interior angle at this vertex is smaller than $\pi$.

A vertex $v$ is an interior cusp of a polygon $P$ if the two adjacent vertices $v_-$ and $v_+$ are both to the left of $v$ or to the right of $v$.

**Lemma 2.4** If a polygon $P$ is $x$-monotone if and only if it has no interior cups.

*Proof:* Clearly, if it has an interior cusp, then it is not monotone.

As for the other direction, let us split $P$ by adding the two $x$-extreme vertices to $P$. If both the upper and lower chain are monotone, then $P$ is monotone. Since $P$ has no interior cusp we claim that both chains are monotone.

Consider the lower chain, and start walking on it from left to right. If the lower chain is not monotone, then there must be a vertex $u$ where we arrive to it from the left, and leave also through the left. However, since the polygon interior is on our left, it must be that $u$ is an interior cusp. A contradiction. Thus, it must be that the lower chain is monotone.

Similar argument applies to the upper chain.

**2.1 Triangulating a Monotone Polygon**

**Definition 2.5** We want to triangulate a monotone polygon.

Consider the first segments on the upper and lower chains:

One segment must be longer in the $x$-span, and assume it is the first segment $s$ on the top chain ($v_0v_32$ in the figure) - $s = v_0v_{n-1}$. Let $v_1, v_2, \ldots, v_{m-1}$ be the vertices on the bottom chain that lie in the $x$-range of $s$, such that $v_m$ is to the right of $v_{n-1}$ in the ordering of the points according to their $x$-coordinate. The claim is that there is always a diagonal which is valid. This is...
We can compute this diagonal in $O(m)$ time. What we did, we broke the monotone polygon into two parts. The part of the left, have a single segment as one of its chains. This is known as *mountain polygon*:

If we can now triangulate a mountain in linear time, then we can triangulate a monotone polygon in linear time: We “steal” a mountain of size $m$ in $O(m)$ time, and triangulate it in $O(m)$ time. Namely, triangulating a monotone polygon takes:

$$T(n) = O(m) + T(n - m)$$

which is $T(n) = O(n)$.

### 2.2 Triangulating a mountain

**Definition 2.6** The two extreme vertices of a mountain polygon, are base vertices.

**Lemma 2.7** Given a mountain polygon, one of the non-base vertices must be convex. Namely, the internal angle at this vertex is smaller than $\pi$.

*Proof:* There are $n - 2$ non base vertices in a polygon $P$ with $n$ vertices. If the angle in all those vertices exceeds $\pi$, then the total sum of angle in the polygon $P$ exceeds $\pi(n - 2)$. However, we provide in the previous lecture that the sum of angles of a polygon is exactly $\pi(n - 2)$. A contradiction. One of the non base vertices of a mountain polygon is convex. ■

**Lemma 2.8** Every convex vertex in a mountain polygon, which is not a base vertex, is an ear.

*Proof:* If $v$ is a convex vertex than its two neighbors must see each other inside the polygon. As such the diagonal they define is legal, and $v$ is an ear, as the other chain is a single segment that can not intersect this diagonal.

This suggests a natural algorithm for triangulating a mountain polygon $P$:  

\[ v_{m-1}v_{n-1}: \]
1. Compute a list $L$ of all convex vertices of $P$ which are not base vertices.

2. As long as $L$ is not empty

   (a) Pick a vertex $v \in L$, and remove $v$ from $L$
   (b) Since $v$ is an ear. Remove the triangle $\triangle v_-vv_+$ it defines.
   (c) Update the angles of $v_-, v_+$ (the two vertices adjacent to $v$), and add them to $L$ if needed.

Namely, we repeatedly remove a ear, till we remain with a triangle. Clearly, handling every ear takes $O(1)$ time.

Lemma 2.9 One can triangulate a mountain polygon in linear time.

Theorem 2.10 One can triangulate a monotone polygon in linear time.

Thus, to triangulate a general polygon, we need to break it into monotone polygons. How do we do it?

3 Vertical Decomposition

Given a general polygon $P$, let us erect a vertical wall through each vertex, till we hit either a floor or a ceiling. If we do this from every vertex, we decompose a polygon into a bunch of vertical trapezoids:

Why is this interesting? Well, to decompose a polygon into monotone polygons, we need to “kill” all interior cups. An interior cups in a vertical decomposition, looks like:

So, how can we remove the interior cusp?
Note, that if a vertical trapezoid have two vertices on its boundary, we can connect them with a diagonal. Clearly, if we connect all such diagonal, what remains is a decomposition of $P$ into monotone polygons.

**Lemma 3.1** Given a simple polygon $P$, and its vertical decomposition, one can decompose $P$ into monotone polygons in linear time.

### 3.1 Computing Vertical Decomposition Using Sweeping

Let us imagine, that we drag a vertical line from $x = -\infty$ to $x = +\infty$. We want to maintain the ordering of segments that intersect the line at every point in time. Let $S(t)$ be the set of segments of $P$ that intersect the sweeping line $l(t) \equiv x = t$.

$$S(1) = \{\}, \ S(2) = \{e_1, e_2, e_3, e_4, e_{17}\},$$

$$S(3) = \{e_3, e_4, e_5, e_6, e_8, e_{16}\}$$

Assume that $S(t)$ is sorted.
When does $S(t)$ change?

A: When the sweeping line passes over a vertex. $S(t)$ changes only in vertices. So... Sort the vertices from left to right, and do an update as we go from left to right, stooping only at the vertices.

There are several types of events we need to handle:

1. Replace:

   
   ![Diagram](image)

   Remove $e_{16}$ from $S(t)$ and insert $e_{17}$

2. Insert:

   
   ![Diagram](image)

   Insert $e_{16}$ and $e_{17}$ to $S(t)$

3. Delete:

   
   ![Diagram](image)

   Remove $e_{16}$ and $e_{17}$ from $S(t)$. 
Observe, that the changes to $S(t)$ are local. Namely, either we delete an element, or insert a new element.

How to perform the sweeping?

1. $t$ - the $x$-coordinate of the sweeping like is a global variable.

2. A data-structure to maintain $S(t)$ (Sorted!). We need a data-structure that enable us to perform insertion and deletions quickly. So... What DS to use? Well, any balanced binary tree data structure would work (red-black tree or treaps).

3. Since all the events happened in segment endpoints, sort all the segment endpoints in queue.

Thus, sweeping algorithms have a very simple and generic structure:

\[
\text{SweepAlg}(P) \\
Q \leftarrow \text{store endpoints of segments of } P \text{ in } x\text{-ord heap.} \\
\text{While } Q \text{ not empty} \\
e \leftarrow \text{minHeap}(Q) \\
\text{handle the event } e.
\]

So, how much time does sweep takes?

A:

\[
O(n \log n)
\]

1. $O(n \log n)$ - price for heap.

2. $O(n)$ events, each event takes $O(\log n)$ time: Constant number of insertion/deletions from a balanced tree.

**Theorem 3.2** Sweeping a simple polygon can be done in $O(n \log n)$ time.

So, what is the connection to vertical decomposition?

A: Every time we stop to handle event, we check whether we need to insert a vertical wall, and if so, we insert it.

Q: How do we know whether to erect a vertical wall?

A: We can assume that we know for every edge, on which side of it the polygon lies.
Q: How do we know how long to make the vertical walls? A: ????
A: $S(t)$ is sorted. As such, we can find the segment just below our event vertex, and just above it in $O(\log n)$ time.

In fact, when we stop and update $S(t)$ we can also erect the vertical walls of the vertical decomposition.

Thus, by just modifying the sweeping algorithm, we can get an algorithm that constructs the vertical decomposition of $P$ in $O(n \log n)$ time.

**Theorem 3.3** Given a simple polygon $P$, one can compute the vertical decomposition of $P$ in $O(n \log n)$ time.