24: Polygon Partition - Faster Triangulations

CS 473u - Algorithms - Spring 2005

April 21, 2005
- triangulations of simple polygons.

- Described a quadratic time algorithm for triangulation.
Monotone Polygons - the easy case

Definition

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

A polygon is $x$-monotone, if its boundary can be partitioned into two chains that are $x$-monotone.

A monotone polygon is given as two sorted linked lists of vertices on the upper chain and lower chain.
Definition

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

![Diagram showing a polygon with interior cusps and reflex vertices]
Lemma

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

proof

⇒ If $P$ has an interior cusp, then $P$ not monotone.
⇐ Split $P$ by adding the two $x$-extreme vertices to $P$. If both the upper and lower chain are monotone, then $P$ is monotone.

Claim: both chains are monotone.
Walk on lower chain from left to right.
If the lower chain is not monotone, then there must be a vertex $u$ where we arrive from left, and leave from left.
Polygon interior is on our left $\Rightarrow$ $u$ is an interior cusp. A contradiction.
Thus, lower chain is monotone.
Similar argument applies to the upper chain.
Consider the first segments on the upper and lower chains:

- One segment must be longer in the $x$-span,
- Assume - first segment $s$ on the top chain: $s = v_0 v_{n-1}$.
- Let $v_1, v_2, \ldots, v_{m-1}$ be the vertices on the bottom chain that lie in the $x$-range of $s$.
- $v_m$ is to the right of $v_{n-1}$
- There is always a diagonal which is valid. This is...
Compute diagonal in \( O(m) \) time.
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:
Using mountain polygons.

Triangulate a mountain in linear time
⇒
Triangulate a monotone polygon in linear time.

- We “steal” a mountain of size $m$ in $O(m)$ time
- Triangulate it in $O(m)$ time. Namely, triangulating a monotone polygon takes:

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

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

Triangulating a mountain

**Definition**

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

**Lemma**

*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.
Definition

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

Lemma

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

Lemma
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

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.
A natural algorithm for triangulating a mountain polygon $P$:

1. Compute a list $L$ of all convex vertices of $P$ which are not base vertices.
2. As long as $L$ is not empty
   1. Pick a vertex $v \in L$, and remove $v$ from $L$
   2. Since $v$ is an ear. Remove the triangle $\triangle v_-vv_+$ it defines.
   3. Update the angles of $v_-$, $v_+$ (the two vertices adjacent to $v$), and add them to $L$ if needed.

Repeatedly remove a ear, till we remain with a triangle. Handling a ear takes $O(1)$ time.
Lemma

One can triangulate a mountain polygon in linear time.

Theorem

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?
Given a general polygon $P$, erect a vertical wall through each vertex, till we hit either a floor or a ceiling. Do for every vertex → decompose a polygon into a bunch of vertical trapezoids:

This is a **vertical decomposition** of $P$. 
Observation

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?
So, how can we remove the interior cusp?
Vertical trapezoids have two vertices on its boundary, we can connect them with a diagonal.

Connect all such diagonals, what remains is a decomposition of $P$ into monotone polygons.

**Lemma**

Given a simple polygon $P$, and its vertical decomposition, one can decompose $P$ into monotone polygons in linear time.
Vertical trapezoid have two vertices on its boundary, we can connect them with a diagonal.

Connect all such diagonals, what remains is a decomposition of $P$ into monotone polygons.

**Lemma**

*Given a simple polygon $P$, and its vertical decomposition, one can decompose $P$ into monotone polygons in linear time.*
Vertical trapezoids have two vertices on its boundary, we can connect them with a diagonal.

Connect all such diagonals, what remains is a decomposition of $P$ into monotone polygons.

**Lemma**

*Given a simple polygon $P$, and its vertical decomposition, one can decompose $P$ into monotone polygons in linear time.*
Computing Vertical Decomposition Using Sweeping

- Drug a vertical line from $x = -\infty$ to $x = +\infty$.
- 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 $l(t) \equiv x = t$.

\[
S(1) = \{\}, \quad S(2) = \{e_1, e_2, e_3, e_4, e_{17}\}, \\
S(3) = \{e_3, e_4, e_5, e_6, e_8, e_{16}\}
\]

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:
Replace:

Remove $e_{16}$ from $S(t)$ and insert $e_{17}$
Insert $e_{16}$ and $e_{17}$ to $S(t)$. 
Delete:

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 data-structure 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.
A sweeping algorithm

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

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

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: \( S(t) \) 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.
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: $S(t)$ 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.
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: $S(t)$ 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.
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: $S(t)$ 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

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

Theorem

*Given a simple polygon $P$ with $n$ vertices, one can triangulate it in $O(n \log n)$ time.*