CMSC 754
Computational Geometry

David M. Mount
Department of Computer Science
University of Maryland
Fall 2005

Copyright, David M. Mount, 2005, Dept. of Computer Science, University of Maryland, College Park, MD, 20742. These lecture notes were prepared by David Mount for the course CMSC 754, Computational Geometry, at the University of Maryland. Permission to use, copy, modify, and distribute these notes for educational purposes and without fee is hereby granted, provided that this copyright notice appear in all copies.
**Triangulation of Monotone Polygons:** We can triangulate a monotone polygon by a simple variation of the plane-sweep method. We begin with the assumption that the vertices of the polygon have been sorted in increasing order of their $x$-coordinates. (For simplicity we assume no duplicate $x$-coordinates. Otherwise, break ties between the upper and lower chains arbitrarily, and within a chain break ties so that the chain order is preserved.) Observe that this does not require sorting. We can simply extract the upper and lower chain, and merge them (as done in mergesort) in $O(n)$ time.

The idea behind the triangulation algorithm is quite simple: Try to triangulate everything you can to the left of the current vertex by adding diagonals, and then remove the triangulated region from further consideration.

![Figure 17: Triangulating a monotone polygon.](image)

In the example, there is obviously nothing to do until we have at least 3 vertices. With vertex 3, it is possible to add the diagonal to vertex 2, and so we do this. In adding vertex 4, we can add the diagonal to vertex 2.
However, vertices 5 and 6 are not visible to any other nonadjacent vertices so no new diagonals can be added. When we get to vertex 7, it can be connected to 4, 5, and 6. The process continues until reaching the final vertex. The important thing that makes the algorithm efficient is the fact that when we arrive at a vertex the untriangulated region that lies to the left of this vertex always has a very simple structure. This structure allows us to determine in constant time whether it is possible to add another diagonal. And in general we can add each additional diagonal in constant time. Since any triangulation consists of \( n - 3 \) diagonals, the process runs in \( O(n) \) total time. This structure is described in the lemma below.

**Lemma: (Main Invariant)** For \( i \geq 2 \), let \( v_i \) be the vertex just processed by the triangulation algorithm. The untriangulated region lying to the left of \( v_i \) consists of two \( x \)-monotone chains, a lower chain and an upper chain each containing at least one edge. If the chain from \( v_i \) to \( u \) has two or more edges, then these edges form a reflex chain (that is, a sequence of vertices with interior angles all at least 180 degrees). The other chain consists of a single edge whose left endpoint is \( u \) and whose right endpoint lies to the right of \( v_i \).

We will prove the invariant by induction. As the basis case, consider the case of \( v_2 \). Here \( u = v_1 \), and one chain consists of the single edge \( v_2v_1 \) and the other chain consists of the other edge adjacent to \( v_1 \).

To prove the main invariant, we will give a case analysis of how to handle the next event, involving \( v_i \), assuming that the invariant holds at \( v_{i-1} \) and see that the invariant is satisfied after each event has been processed. There are the following cases that the algorithm needs to deal with.

**Case 1:** \( v_i \) lies on the opposite chain from \( v_{i-1} \):

In this case we add diagonals joining \( v_i \) to all the vertices on the reflex chain, from \( v_{i-1} \) back to (but not including) \( u \). Note that all of these vertices are visible from \( v_i \). Certainly \( u \) is visible to \( v_i \). Because the chain is reflex, \( x \)-monotone, and lies to the left of \( v_i \) it follows that the chain itself cannot block the visibility from \( v_i \) to some other vertex on the chain. Finally, the fact that the polygon is \( x \)-monotone implies that the unprocessed portion of the polygon (lying to the right of \( v_i \)) cannot “sneak back” and block visibility to the chain.

After doing this, we set \( u = v_{i-1} \). The invariant holds, and the reflex chain is trivial, consisting of the single edge \( v_iv_{i-1} \).

![Figure 18: Triangulation cases.](image)

**Case 2:** \( v \) is on the same chain as \( v_{i-1} \):

We walk back along the reflex chain adding diagonals joining \( v_i \) to prior vertices until we find the first that is not visible to \( v_i \). As can be seen in the figure, this may involve connecting \( v_i \) to one or more vertices (2a) or it may involve connecting \( v_i \) to no additional vertices (2b), depending on whether the first angle is less or greater than 180 degrees. In either case the vertices that were cut off by diagonals are no longer
in the chain, and \( v_i \) becomes the new endpoint to the chain. Again, by \( x \)-monotonicity it follows that the unprocessed portion of the polygon cannot block visibility of \( v_i \) to the chain.

Note that when we are done the remaining chain from \( v_j \) to \( u \) is a reflex chain. (Note the similarity between this step and the main iteration in Graham’s scan.)

How is this implemented? The vertices on the reflex chain can be stored in a stack. We keep a flag indicating whether the stack is on the upper chain or lower chain, and assume that with each new vertex we know which chain of the polygon it is on. Note that decisions about visibility can be based simply on orientation tests involving \( v_i \) and the top two entries on the stack. When we connect \( v_i \) by a diagonal, we just pop the stack.

**Analysis:** We claim that this algorithm runs in \( O(n) \) time. As we mentioned earlier, the sorted list of vertices can be constructed in \( O(n) \) time through merging. The reflex chain is stored on a stack. In \( O(1) \) time per diagonal, we can perform an orientation test to determine whether to add the diagonal and (assuming a DCEL) the diagonal can be added in constant time. Since the number of diagonals is \( n - 3 \), the total time is \( O(n) \).