4. Polygon triangulation

Introduction. In this chapter we show how to divide a given polygon into triangles with vertices at the vertices of this polygon. Recall that in a simple polygon any closed curve can be withdrawn continuously to a single point. This means that there are no "holes" in it, and that it only has an outer boundary formed by a single closed polygonal chain.

FIGURE 4.1 Simple and nonsimple polygon

The input of our algorithm will be a double-connected edge list for a planar subdivision defined by a polygon and the output will be a double-connected edge list for the plane subdivision given by the triangulation of the polygon.

FIGURE 4.2 A sample of triangulation.

If you look at the previous picture and count the number of triangles that we have divided the seventeen-edged polygon, then the following statement will not surprise you.

Theorem 4.1. *Every simple polygon can be triangulated. Any triangulation of a simple polygon with n vertices consists of $n-2$ triangles.*

Proof. We will proceed by induction with respect to n. For $n=3$, both statements are obvious. Assume that they are true for all polygons with k edges for $3 \leq k < n$ and conedger a polygon P with n edges. First we will show that it can be triangulated.

Let's take its vertex v which is the smallest in the lexicographic arrangement first by the coordinate x and then by y. (It lies the most to the left of all vertices, and if there are more such vertices, then the most down.) From this vertex the edges lead to the vertices u and w. If the whole line segment uw lies in P, we can create a triangle uvw and so P splits into this triangle and $(n-1)$-edged polygon Q. This can be triangulated according to the induction assumption.

FIGURE 4.3 To the proof of Theorem 4.1.

If the segment uw is not entirely in P, there must be some more vertices of P in the interior of the triangle uvw. From them, we select the vertex t lying furthest from the line uw. Then the whole line segment vt lies in the polygon P. (If it did not, an edge had to cross it, and one of its vertices would lie from uw further than the vertex t.) Thus, the segment vt divides the polygon P into two simple polygons Q_1 a Q_2 with the common edge vt. Since both have less than n edges, they can be triangulated according to the induction assumption. Their triangulations give triangulation of the polygon P.

Now conedger an arbitrary triangulation of the polygon P with n edges. If $n > 3$, then some triangle edge in this triangulation is not a polygon edge and thus divides P into two simple triangulated polygons Q_1 and Q_2 with n_1 and n_2 vertices, respectively. Since they have one common edge, $n_1+n_2 = n+2$. If we use the induction assumption on Q_1 and Q_2, the total number of triangles in the triangulation of the polygon P is

$$(n_1-2)+(n_2-2) = (n_1+n_2)-4 = (n+2)-4 = n-2.$$

□

Monotone polygons. To triangle a convex polygon is very simple. One of the many options is that we take a vertex and connect it with all non-adjacent vertices.

FIGURE 4.4 Triangulation of a convex polygon.

We can weaken the convexity requirement. We can require that every straight line parallel to the axis x (given by the equation $y = k$) intersects the polygon in the convex set, that is, in a line segment, a point, or an empty set. Such a polygon will be called *monotone with respect to the axis y*.

FIGURE 4.5 Monotone and non-monotone polygons.

This definition is geometrically illustrative, but needs to be refined for our needs. Conedger the lexicographic arrangement used for the sweep line method:

$$p > q \text{ iff } (p_y > q_y) \text{ or } (p_y = q_y \text{ and } p_x < q_x).$$

Denote the vertex maximal in this arrangement as u and minimal as d. These points divide the polygon boundary into two parts - left and right. We will talk about the left path and the right path from u to d. The refinement of the definition of the monotone polygon with respect to the axis y consists in the requirement that the both paths from u to d decrease in the described lexicographical arrangement. The polygon monotone according to this definition is definitely monotone according to the previous geometric definition. The opposite is not true, as shown in the following figure.

FIGURE 4.6 According to the geometric definition, both polygons are monotone. According to the refined definition, only P_1 is monotone. The right path of the polygon P_2 is not decreasing.

It turns out that monotone polygons can be relatively well triangulated. Therefore, we divide the triangulation algorithm of a simple polygon into two parts:

- dividing the polygon into monotone polygons,
- triangulation of monotone polygons.

Types of vertices of the polygon and its monotony. Let v be a vertex of the polygon and let p and q be its adjacent vertices. The vertex v is one of the following types:
Start – if the paths from v to p and to q are decreasing in the conedgered lexicographic arrangement and the polygon P is adjacent to the vertex v from below.
End – if the paths from v to p and to q are increasing in our lexicographic arrangement and the polygon P is adjacent to the vertex v from above.
Split – if the paths from v to p and to q are decreasing and the polygon P is adjacent to the vertex v from above.
Merge – if the paths from v to p and to q are increasing and the polygon P is adjacent to the vertex v from below.
Regular – if the path from p to q through v is either increasing or decreasing.

FIGURE 4.7 Types of vertices.

The algorithm for dividing the polygon into monotone parts is based on the statement:

Theorem 4.2. *A simple polygon is monotone with respect to the axis y if and only if it does not have split and merge vertices.*

Proof. It is clear from the definitions that if the polygon has a split or merge vertex, it can not be monotone.

If the polygon P is not monotone, then a part of its right or left path between the vertices u and d is not decreasing. Thus there are vertices of $p \neq u$, and $s \neq d$ such that

(1) the path from u to p decreases and p is the minimal (in our lexicographic arrangement) vertex with this property,
(2) the path from s to d decreases and s is the maximal vertex with this property,
(3) $p < s$.

If p has the polygon from below, it is a merge vertex. If p has the polygon from above, so does the vertex s, and that it why s is a split vertex.

FIGURE 4.8 To the proof of Theorem 4.2.

□

Algorithm for dividing a polygon into a monotone parts. The goal of the algorithm is to add line segments connecting a split or merge vertex with another vertex inedge the polygon P so that we divide the polygon into polygons that will no longer contain any vertices of these two types. To remove a split vertex a new segment must be routed upward while to remove a merge vertex we add a segment going downwards. We will do this by a sweep line method. The events will be the vertices of the polygon arranged lexicographically from the largest (i.e. u) to the smallest (i.e. d) into a queue $\mathcal{Q}$. The sweep line will go from top to bottom, and when the event (= vertex) is passed, we will execute a procedure that will depend on the the type of the vertex and will only concern the "closest neighbourhood" of that vertex above the sweep line.

This "closest neighbourhood" can be found by using a balanced binary tree $\mathcal{T}$ whose leaves are polygon edges that

- intersect the sweep line and
- have the polygon P from the right.

In the description of the algorithm, the notion of the *helper of an edge* plays an important role. Let the sweep line stop on an event v. Consider an edge e intersecting the sweep line. A vertex p is now said to be a helper of the edge e (notation helper(e)) iff

(1) p is above the sweep line l,
(2) p lies to the right of the segment e,
(3) the whole horizontal line segment joining the point p with the segment e lies in the polygon P,
(4) of all vertices meeting the previous conditions p has minimal y coordinate (it is the closest to the sweep line).

In the balanced binary tree $\mathcal{T}$ we will hold a helper with each edge e.

FIGURE 4.9 The point p is the helper of the edge e.

The strategy of the algorithm is as follows. Split vertices are removed at the moment when the sweep line passes through them. In this case, we join the split vertex with a helper of the nearest edge to the left. We can find this edge using the binary tree $\mathcal{T}$.

FIGURE 4.10 Removing the split vertex v.

Removing merge vertices is more complex. We do not remove these vertices when the sweep line passes them, because at this point we do not know the ”situation” under the sweep line and we cannot link the vertex to a vertex below. The merge vertices are removed retroactively. In each event which is passed by the sweep line we test whether the helper of the closest edge or edges is merge. If so, we will connect the vertex in the event with this helper.

FIGURE 4.11 Removal of the merge vertex p.

The following pseudocode forms the basic framework of the algorithm for dividing the polygon into monotone parts.

ALGORITHM MakeMonotone from pseudo.pdf, page 7 Please, write P instead of $\mathcal{P}$.

Procedures for passing through different types of vertices. When the sweep line passes through events, we do the following:

- we connect the vertex with a helper of some edge by a segment,
- we add edges of the polygon into the binary tree $\mathcal{T}$ together with their helpers,
- we change helpers of some edges in the tree $\mathcal{T}$,
- we remove some edges of the polygon from the tree $\mathcal{T}$.

Procedures for specific types of vertices are described and illustrated in the following text and pictures. We use v_1, v_2, …, v_n to describe the polygon vertices arranged counterclockwise. The edge e_i links the vertices v_i and v_{i+1}. We take $v_{n+1} = v_1$.

FIGURE 4.12 The sweep line passes a start vertex.

ALGORITHM HandleStartVertex from pseudo.pdf, page 8

FIGURE 4.13 The sweep line passes an end vertex.

ALGORITMUS HandleEndVertex from pseudo.pdf, page 9

FIGURE 4.14 The sweep line passes a split vertex.

ALGORITHM HandleSplitVertex from pseudo.pdf, page 10

FIGURE 4.15 The sweep line passes a merge vertex.

ALGORITHM HandleMergeVertex from pseudo.pdf, page 11

FIGURE 4.16 The sweep line passes a regular vertex.

ALGORITHM HandleRegularVertex from pseudo.pdf, page 12

Correctness of the algorithm and its time complexity. To prove the correctness of the algorithm, we should show that during the algorithm

- by adding segments, we remove all split and merge vertices,
- the added segments do not intersect each other in inner points.

As for split vertices we remove them when the sweep line passes through them. How is it with merge vertices? Every merge vertex v becomes a helper of some edge e after the sweep line passes through it. We will remove it when the sweep line passes through a vertex p, which lies below v and replaces it in the role of the helper of e.

FIGURE 4.17 Removing the merge vertex v.

The proof that the added edges do not cross is inductive. We assume that the added segments above the sweep line do not intersect, and when the sweep line reaches the vertex v, then we have to show (depending on its type) that even after the procedure described above has been completed, the added segments will not intersect. For example, if v is a split vertex, then in the band between it and the helper p of the nearest left edge, none of the previously added segments can intersect the newly added segment vp.

FIGURE 4.18 To the previous proof.

Theorem 4.3. *The running time of the algorithm for dividing a simple polygon with n edges into monotone parts is $O(n \log n)$.*

Proof. Arranging the vertices of the polygon into the queue $\mathcal{Q}$ takes time $O(n \log n)$. By going through one event, we spend a constant time. We have n events. Thus, the total running time is $O(n \log n) + O(n) = O(n \log n)$.

□

Triangulation of monotone polygons. Let us conedger a monotone polygon P with n edges. Its left and right paths are decreasing in the given lexicographic order that is why we can arrange the vertices of the polygon from the biggest to the smallest into a sequence v_1, v_2, …, v_n in time $O(n)$. The basic idea of our algorithm for triangulation of a monotone polygon is as follows. We step through the vertices in the given arrangement and create triangles with linking them with previous vertices whenever it is possible.

We will use the stack instead of the queue in the algorithm. The stack is characterized by the fact that the vertices are arranged in a reverse order from how they were inserted into it.

At the beginning of the algorithm, we put the vertices v_1 and v_2 into the stack, so the stack will be $\mathcal{S} = (v_2, v_1)$. These two vertices lie both on the left or right path of the polygon P. If the following vertex v_3 lies on the opposite path, we link it with the vertex v_2 by a segment and create the triangle $v_1v_2v_3$. Now, we remove v_2 a v_1 from the stack and then insert v_2 and v_3. Thus, the stack will be $\mathcal{S} = (v_3, v_2)$.

FIGURE 4.19 The vertex v_3 lies on the opposite path to the path on which the previous two vertices lie.

Let the vertex v_3 lie on the same path as stack vertices. If the segment v_3v_1 lies inedge P, we will join points v_3 and v_1 by a segment to create the triangle $v_1v_2v_3$. We remove v_2 and v_3 from the stack and insert v_1 and v_3, so the stack is now $\mathcal{S} = (v_3, v_1)$. If the line v_3v_1 is not contained inedge P, we can not create a triangle. In this case, insert v_3 into the stack. So the stack will be $\mathcal{S} = (v_3, v_2, v_1)$.

FIGURE 4.20 Vertex v_3 lies on the same path as the previous vertices.

Let P_3 be a polygon P without the triangle $v_1v_2v_3$ if this triangle lies in P, and let $P_3 = P$ in the opposite case. The P_3 polygon will again be monotone. In both cases, the vertices in the stack lie all on the left or the right path of the polygon P_3. This is how we describe the first step of the algorithm when passing through the vertex v_3.

Now, we will describe the algorithm step when passing through the vertex v_j for $4 \leq j \leq n-1$. At the beginning of this step we have monotone polygon P_{j-1}, which is the rest of the polygon P after cutting off parts of it in previous steps. Assume, that all the vertices of the stack

$$\mathcal{S} = (v_{i_1}, v_{i_2}, \dots, v_{i_k}), \quad j-1 = i_1 > i_2 > \cdots > i_k \geq 1$$

lie on the the same (right or left) path of the polygon P_{j-1}.

If the vertex v_j is on the opposite edge than the stack vertices, the vertex v_j will be joined with every vertex in the stack by a segment. Then we empty the stack, and put the vertices v_{j-1} and v_j into it. Thus, the stack will be $\mathcal{S} = (v_j, v_{j-1})$. We also create a polygon P_j so that from the polygon P_{j-1} we cut off new triangles.

FIGURE 4.21 v_j lies on the opposite edge than the vertices v_{j-1}, v_{j-2}, v_{j-4}, v_{j-5} of the stack.

If the vertex v_j lies on the same edge of the polygon P_{j-1} as stack vertices do, we try to join the vertex v_j with the vertices v_{i_2}, v_{i_3}, etc., by a segment if possible. This means that if the segment $v_jv_{i_2}$ lies inedge P_{j-1}, we will join the vertices. If, in addition, the line $v_jv_{i_3}$ lies within P_{j-1}, we will also connect the vertices. In the same way we proceed as long as we reach the vertex v_{i_s} of the stack so that the segment $v_jv_{i_s}$ does not lie inedge P_{j-1}. From the stack, we pop the vertices $v_{i_1}, v_{i_2}, \dots, v_{i_{s-1}}$ and put vertices $v_{i_{s-1}}$ and v_j into it. The polygon P_j is formed again from the polygon P_{j-1} by cutting off the newly created triangles.

FIGURE 4.22 v_j lies on the same path as the vertices v_{j-1}, v_{j-2}, v_{j-3}, v_{j-4} that are on the stack.

The algorithm is described by the following pseudocode:

ALGORITHM 13 TriangulateMonotonePolygon from pseudo.pdf

The triangulation of a monotone 11-edged polygon carried out according to our algorithm is illustrated in the following picture.

FIGURE 4.23 The triangles are numbered in the order in which they were created by the algorithm.

Theorem 4.4. *The running time of the algorithm for the triangulation of a monotone polygon with n edges is $O(n)$.*

Proof. Since the polygon is monotone, to arrange the vertices of the polygon lexicographically takes time $O(n)$. During the algorithm, each of the n vertices is inserted and taken out of the stack at most twice. Both these operations require a constant time. $\square$