FAST TRIANGULATION OF SIMPLE POLYGONS

Stefan Hertel
Kurt Mehlhorn

Fachbereich 10
Universität des Saarlandes
D - 6600 Saarbrücken

ABSTRACT

We present a new algorithm for triangulating simple polygons that has four advantages over previous solutions [GJPT, Ch].

a) It is faster: Whilst previous solutions worked in time O(nlogn), the new algorithm only needs time O(n+rlogr) where r is the number of concave angles of the polygon.

b) It works for a larger class of inputs: Whilst previous solutions worked for simple polygons, the new algorithm handles simple polygons with polygonal holes.

c) It does more: Whilst previous solutions only triangulated the interior of a simple polygon, the new algorithm triangulates both the interior and the exterior region.

d) It is simpler: The algorithm is based on the plane-sweep paradigm and is - at least in its O(nlogn) version - very simple.

In addition to the new triangulation algorithm, we present two new applications of triangulation.

a) We show that one can compute the intersection of a convex m-gon Q and a triangulated simple n-gon P in time O(n+m). This improves a result by Shamos [Sh] stating that the intersection of two convex polygons can be computed in time O(n).

b) Given the triangulation of a simple n-gon P, we show how to compute in time O(n) a convex decomposition of P into at most 4·OPT pieces. Here OPT denotes the minimum number of pieces in any convex decomposition. The best factor known so far was 4.333 (Chazelle[Ch]).

O. INTRODUCTION

In computational plane geometry, a powerful new type of algorithm seems to apply to many problems. It sweeps the plane from left to right, in direction of the x-axis, advancing a more or less vertical "cross section" from one point to the next. All processing is done at this moving front the state of which is represented by the "y-structure", while the "x-structure" represents a queue of tasks to be performed.

We tailor the plane-sweep technique as detailed by Nievergelt and Preparata [NP] to the problem of polygon triangulation. Triangulations of the plane are useful in e.g. closest point problems [LP, LT], and polygon triangulations serve for area calculations as well as for solving visibility and internal path problems [Ch], to name just a few applications.

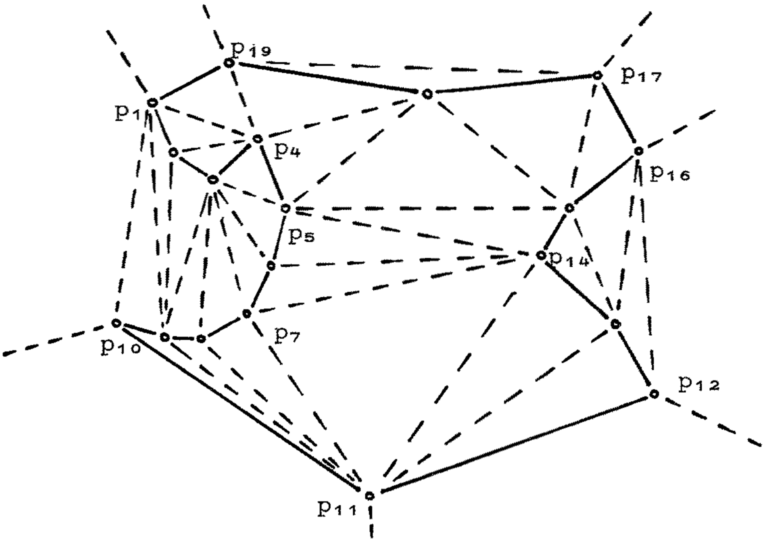


Figure 1. Simple polygon with a possible inner and outer triangulation. Dashed line segments are triangulation edges.

We shall triangulate the plane with respect to arbitrary simple n-gons with $r<n$ concave angles, and we shall give two important applications. Our time $O(n + r\log r)$, space $O(n)$ solution outperforms previous $O(n\log n)$ solutions [GJPT,Ch].

The next section exhibits the necessary basic data structures. Section 2 illustrates the ideas we use by describing a triangulation algorithm that matches the performance of [GJPT] and [Ch]. A few modifications to this algorithm including its data structures allow us to improve the upper bound to $O(n + r\log r)$ in section 3. Within the same time bound, we can also triangulate the region outside the polygon, as shown in section 4. Finally, we give two applications of polygon triangulation.

1. THE BASIC DATA STRUCTURES

Our triangulation algorithm will operate upon four data structures.

Their basic form described here will be modified in later sections as needed. In addition to the x-structure and the y-structure already mentioned we introduce two specific data structures.

<u>The x-structure X</u>

X is a simple queue containing the corners of the polygon yet to be processed, sorted in order of increasing x-coordinate. For simplicity of exposition we assume that all x-coordinates are different. Initially, X contains all n corners. The algorithm removes one point from X at a time, and performs one transition each.

<u>The y-structure Y</u>

The vertical cross section cuts through line segments which partition it into intervals. Intervals inside alternate with intervals outside the polygon, to be referred to as in-intervals and out-intervals, resp. Y describes the cross section, and is very similar to the y-structure in [NP]. It has an entry for each interval, including those that extend to $y = +(-)\infty$; equivalently, an entry for each line segment intersected by the cross section, including two sentinels $+(-)\infty$. A line segment entry is a formula of the form $y = ax + b$ that defines this segment. This allows to find the y-value corresponding to a given x-value in constant time. Y is a dictionary (see [AHU]) that must support the operations FIND, INSERT, DELETE in time $O(\log k)$ when it contains k entries, and the operations SUCC and PRED in time $O(1)$, by means of additional pointers. The definition of these 5 operations is slightly modified such that they fit our purpose.

In a left-to-right scan of the plane, each point can be uniquely classified into one of three main categories:

start point:  bend:  end point: 

A start (end) point with its convex angle belonging to the interior of the polygon is called proper, improper otherwise.

The result of the operation FIND(P) depends on the type of point P. For brevity's sake, the exact definitions are not given here. Suffice it to say that such a dictionary can be implemented by any of several kinds of balanced trees.

<u>The p-structure P</u>

P assembles information about parts of the polygon passed already whose triangulation depends on points unseen so far. For any given cross section it contains information about exactly those regions corresponding to in-intervals of this cross section. Specifically, P associates with each line segment s above an in-interval a list L(s), doubly linked by means of NEXT and PREV pointers. L(s) is a chain of corners of the polygon that are connected by either a polygon edge or a triangulation edge, starting with the left endpoint of s. In addition, RM(s) points to the rightmost element of the polygonal chain L(s). A typical cross section with the corresponding structures Y and P is shown in figure 2.

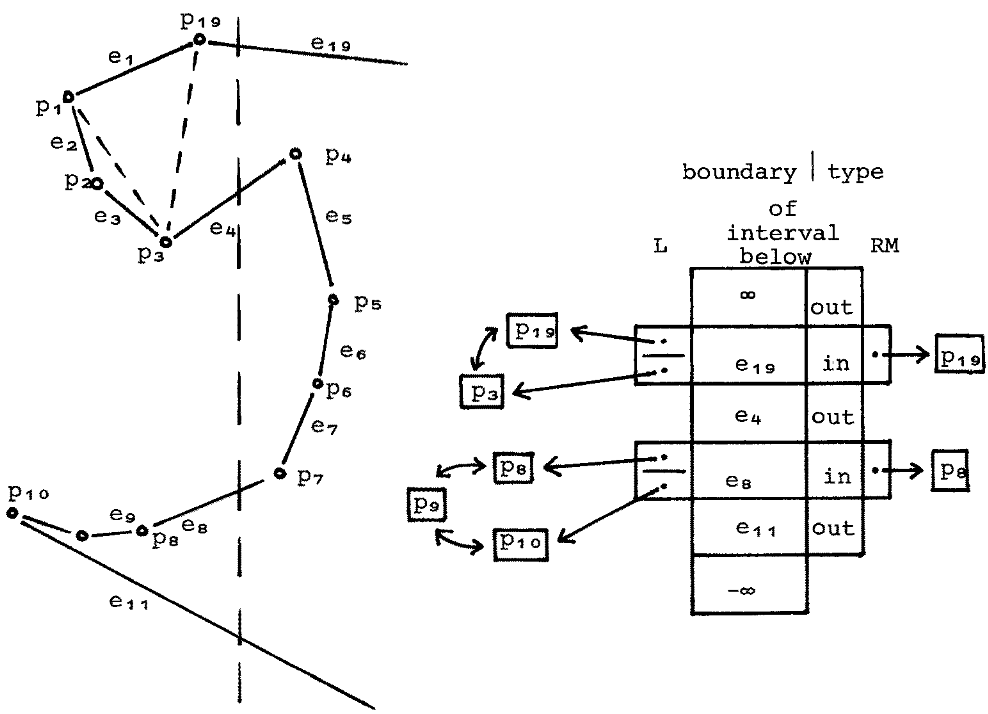


Figure 2. Structure Y-P in a cross section between points p_{19} and p_7. Two triangulation edges have been constructed.

<u>The t-structure T</u>

The output structure T is steadily built up while the plane is swept from left to right. It consists of two lists, a list TRI of triangles and a list EDGES of polygon and triangulation edges. Pointers between the two lists represent triangle-edge adjacencies.

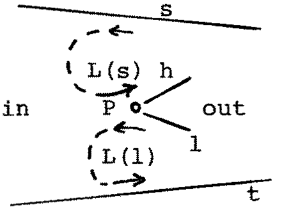
2. THE BASIC ALGORITHM FOR INTERIOR TRIANGULATION

The algorithm that sweeps the plane and constructs triangulation edges has a simple overall structure similar to that of several plane-sweep algorithms. We follow the approach of [NP].

```
procedure SWEEP:
  X  ←  n given points, sorted by increasing x-coordinate
  Y  ←  (∞,out)  ∪ (-∞)
  P  ←  ∅
  TRI  ←  ∅
  EDGES  ←  n polygon edges, given in counterclockwise order
  while  X ≠ ∅  do
                 P  ←  MIN(X)
                 TRANSITION(P)
              od
  end of SWEEP.
```

All the work involved in moving the current cross section across P is performed by procedure TRANSITION. It is invoked exactly n times. Since each invocation will use O(logn) time, this will result in an O(nlogn) algorithm.

TRANSITION handles each of the types of the "next point" P differently. Here we only describe, as an example, the case "improper start" which can be considered to be the most complicated one. "o" denotes concatenation.




```
case "improper start":
   FIND(P) yields the two adjacent
   line segments s and t in whose
   interval [t,s] P lies;
   h  ←  high segment starting at P
   l  ←  low segment starting at P
   Q  ←  RM(s)
   EDGES  ←  EDGES ∪ QP (with pointers set to NIL)
   INSERT((h,out))
   INSERT((l,in))
   RM(s)  ←  P ;  RM(l)  ←  P
   L(l)  ←  P o "remainder of L(s) starting at Q"
   L(s)  ←  "L(s) up to and including Q" o P
   TRIANGULATE(s,"c")
   TRIANGULATE(l,"cc")
   end of case "improper start".
```

TRIANGULATE(e,dir) starts at a point P at one end of a polygonal chain L(e) where e is a polygon edge in the y-structure, and it triangulates "along" L(e) as far as possible. If dir = "cc", P is the head of L(e), and the triangulation proceeds counterclockwise. If dir = "c", P is the tail of L(e), and we triangulate in clockwise direction. The exact description is omitted here but it should be clear that constructing a new triangulation edge and updating the appropriate polygonal chain (also to be referred to as p-chain) as well as T can be done in time O(1). Thus the running time of TRIANGULATE is proportional to the number of new triangulation edges, and the total time spent in TRIANGULATE is O(n). In addition, one call to TRANSITION takes time O(logn), yielding an overall running time of O(nlogn), including the initial sorting.

Extension to polygonal regions

It is easy to see that our algorithm also triangulates polygonal regions, i.e., ring-shaped regions with the circles replaced by simple polygons. A triangulation edge is drawn when the leftmost corner of the interior polygon is encountered - thus, in effect, cutting the polygonal region along this edge and transforming it into a simple polygon. A similar argument holds for regions with several polygonal holes.

3. THE IMPROVED ALGORITHM

We refine our previous algorithm to show that it suffices to consider "not many more" points than the intruding corners, i.e., the corners with in-angle $> \pi$, also to be referred to as points with concave angle. If their number is r, the refined algorithm will run in time O(n + rlogr).

Our x-structure X will contain these r intruding corners, and O(r) more points - the proper start and the proper end points. Sorting them according to their x-coordinate requires an effort of O(rlogr).

The y-structure Y now is different from that in the straightforward algorithm in that the current cross section is not any longer simply the "sweeping line", a vertical line right after the point just processed. Instead, a cross section consists of vertical parts that may lag behind the sweeping line, each one of them cutting two polygon edges with an in-interval in-between. A cross section part lagging behind implies points with a convex angle on the extension of the corresponding polygonal chain to the right up to the sweeping line. Note that points after which the number of in-intervals changes - start and end points - are still contained in X. Thus we can justifiably require that the ordering of in-intervals is the same as it would have been with the first algorithm.

Example: After having processed p_7 in figure 1, we could have the situation shown in figure 3.

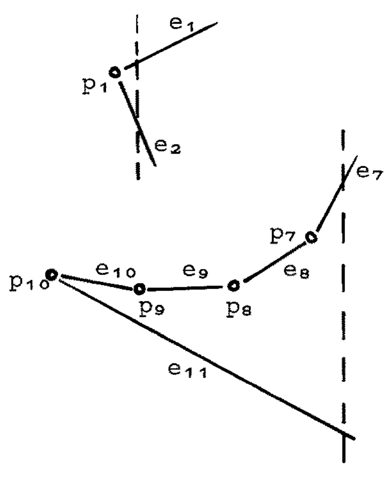


Figure 3. Possible situation after processing p_7 in figure 1. Shown are the cross section parts through the two in-intervals.

To find the location of a new point with respect to the y-structure, we extend some polygonal chains locally, while searching for P in the balanced tree Y. We start at the root and search down the tree. Whenever we encounter an edge e_s, we walk along the main polygon chain to the right, adding new edges to the triangulation, as long as the x-coordinate is smaller than that of P, and proceed, "in parallel", in the same manner with the other end of the polygonal chain of the in-interval adjacent to e_s. Then we can safely add new triangulation edges while searching down the tree Y, as is stated in the following theorem.

Theorem: No triangulation edge drawn from a convex bend intersects an edge we have not seen so far.

All the points processed "on the go" as described above are convex bends. We find each one of them in time O(1) walking along the main polygon chain, and then they are handled like bends in section 2. For an edge starting at a bend, we have to find its successor and its predecessor. INSERTs/DELETEs are not necessary; thus, processing a convex bend takes time O(1) apart from the time spent in TRIANGULATE.

Since the number of in-intervals is bounded by the number of proper start points, Y has at most O(r) entries, and one operation on Y can be implemented to work in O(logr) time. Thus, processing one of the r points in X takes time O(logr) apart from the time for processing convex bends and for triangulating. The latter amounts to a total of O(n), yielding an overall time bound for our algorithm of O(n + rlogr). The space requirement clearly is O(n).

As in section 2, polygonal regions present no difficulties.

4. EXTENSION TO EXTERIOR TRIANGULATION

To triangulate the exterior region of a polygon as well, we expand our data structures. The "p-structure" now represents parts of the plane left of the current cross section and bounded by the current convex hull whose triangulation with respect to the polygon is not finished, yet. The structure H which will be implemented as part of the p-structure is a second output structure; it represents the current state of the convex hull of the polygon.

The main change is in the current cross section. It now consists of vertical parts, each one of them touching two polygon edges (and cutting none) with an in-interval or an out-interval between the two edges. Polygonal chains are also associated with interior out-intervals. Each entry (but for the sentinels $\pm \infty$) in Y now is a double entry for the adjacent in- and out-intervals, respectively. If, during our search for P in Y, we encounter an in-node, we update the in-chain below and the out-chain above; vice versa for an out-node.

The new structure H comprises information about the convex hull of the polygon as seen so far. An h-chain each - similar to the p-chains - is associated with the two fringe intervals. Both h-chains have the very first start point in common; upon processing the very last end point, they are combined to form the convex hull.

<u>Illustration</u>: After processing p_7 in figure 1, we have the situation shown in figure 4.

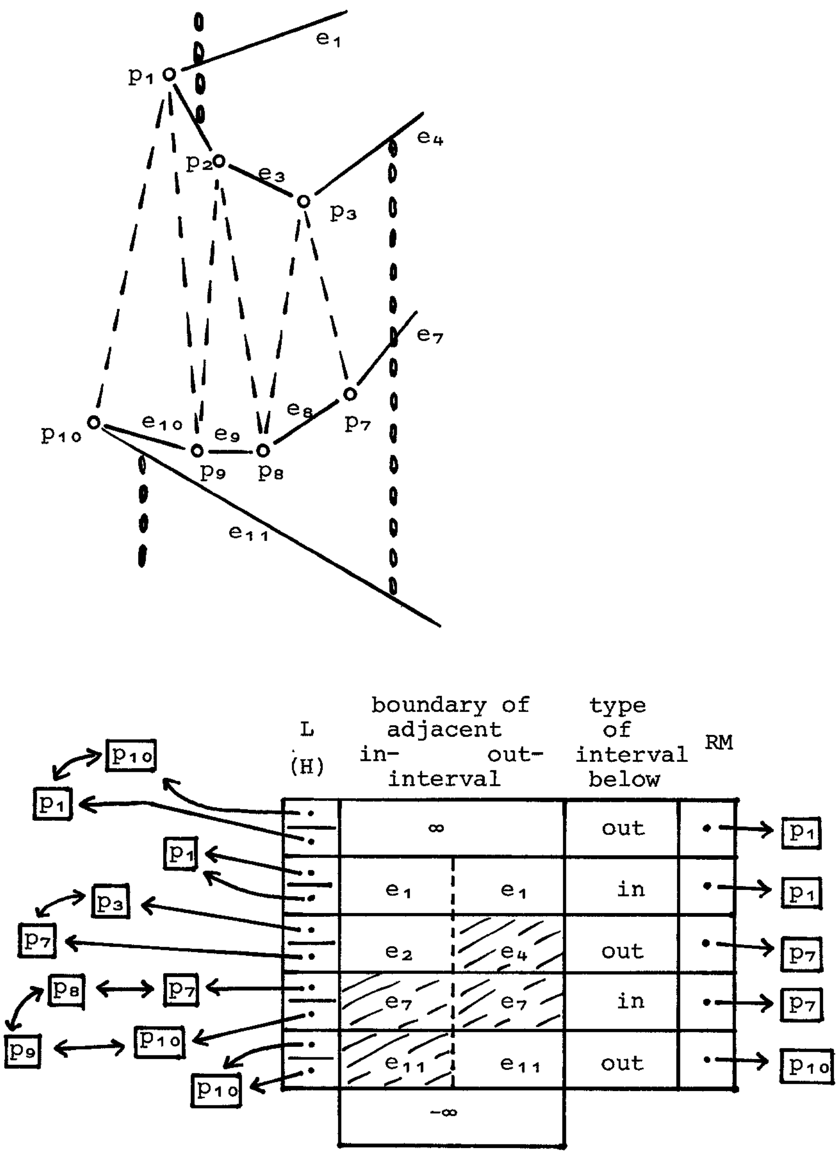


Figure 4. The structure Y-P after processing p_7 in figure 1. h-chains are associated with the top- and the bottom-most out-interval, respectively. Hatched fields are the fields which were updated while searching the lower in-node for p_7.

The triangulation algorithm maintains all these data structures. Points are processed both for interior and for exterior triangulation according to their type. Observe that points that are "proper" for interior triangulation are "improper" for exterior triangulation, and vice versa. As an example, we only give an outline of the processing of a proper start point (as defined in section 1) as far as the outer triangulation is concerned.

Case i) The very first start point
Split the initial out-interval, initialize the two h-chains.

Case ii) A top- (bottom-)most start point
Draw a new outer triangulation edge by appending this start point as new tail (head) to the high (low) h-chain, thus initializing a polygonal chain for the new interior out-interval; triangulate along the high (low) h-chain.

Case iii) An interior start point
Split out-interval by drawing outer triangulation edge to the rightmost point of the corresponding p-chain; then proceed analoguously to the case of an improper start point in section 2.

The procedure TRIANGULATE needs a third parameter indicating inner or outer triangulation.

After the plane-sweep is finished, the convex hull is used to construct the outermost triangles that extend to infinity, in time $O(n)$.

All the running time arguments of the previous section are still valid for this extended algorithm.

5. APPLICATIONS

a) Intersection of a simple polygon P and a convex polygon Q

Shamos [Sh] showed how to compute the intersection of two convex polygons in linear time. We extend his result to

Theorem: Let P be a simple n-gon, and let Q be a convex m-gon. Assume that a triangulation of the plane with respect to P is available. Then $P \cap Q$ can be computed in linear time, i.e., in $O(m+n)$.

Proof(sketch): Let T be a triangulation of P, i.e., a division of the interior and the exterior of P into $2n-2$ triangles. Let T be given as described in section 1.

We start with the observation that the intersection has "size" $O(n)$. Note that the triangulation consists of $O(n)$ line segments. Each such line segment can intersect the convex polygon Q in at most 2 points. Hence the total number of intersections between edges of T and edges of Q is $O(n)$.

Let $v_1, \ldots, v_m$ be the vertices of Q. We can certainly find the triangle containing v_1 in time $O(n)$. Also, knowing the triangle containing v_i, we can find all intersections between T and line segment $\overline{v_i v_{i+1}}$ in time $O(s_i+1)$ where s_i is the number of such intersections. Hence the total time needed to find all points of intersection is

$O(m + \Sigma s_i) = O(m+n)$, by the argument above. □

Corollary: Let P be a simple polygon with n vertices and $r<n$ concave angles. Let Q be a convex polygon with m vertices. Then $P \cap Q$ can be computed in time $O(n + m + r\log r)$.

The best solution hitherto known required time $O((n+m)\log(n+m))$, and can be concluded from [BO].

b) Decomposing a simple polygon into few convex parts

Chazelle [Ch] showed how to decompose, in time $O(n\log n)$ and space $O(n)$, a simple n-gon P into fewer than $4.333 \cdot OPT$ convex pieces, without introducing new vertices, where OPT is the minimum number of convex pieces necessary to partition P. His algorithm has two phases. In phase 1 he triangulates P in time $O(n\log n)$, and in phase 2 he constructs a convex decomposition from the triangulation. We already showed how to improve upon phase 1. We can also improve upon phase 2.

Theorem: Let P be a simple n-gon, and let T be an interior triangulation of P. Then a convex decomposition of P with at most $4 \cdot OPT$ pieces can be constructed in time $O(n)$.

Proof(sketch): Observe that $OPT \geq r/2 + 1$ since one partitioning edge is necessary for each concave angle. We shall partition P into at most $2r + 1$ convex subpolygons.

To do this, scan the n-3 triangulation edges one by one. Drop an edge if it divides two convex angles. Call edge e essential for point p if it cannot be dropped because it divides a concave angle at point p. It is easy to show that not more than two triangulation edges are essential for each point with concave angle. □

REFERENCES

[AHU] A.V.Aho/J.E.Hopcroft/J.D.Ullman: The Design and Analysis of Computer Algorithms, Addison-Wesley Publ. Comp., Reading, Mass., 1974.

[BO] J.L.Bentley/T.A.Ottmann: Algorithms for Reporting and Counting Geometric Intersections, IEEE Trans. on Comp., Vol. C-28, No. 9(1975), pp. 643-647.

[Ch] B.Chazelle: A Theorem on Polygon Cutting with Applications, Proc. 23rd IEEE FOCS Symp. (1982), pp. 339-349.

[GJPT] M.R.Garey/D.S.Johnson/F.P.Preparata/R.E.Tarjan: Triangulating a Simple Polygon, Info. Proc. Letters, Vol. 7(4), June 1978, pp. 175-179.

[LP] D.T.Lee/F.P.Preparata: Location of a Point in a Planar Subdivision and its Applications, SIAM J. Comp., Vol. 6(1977), pp. 594-606.

[LT] R.J.Lipton/R.E.Tarjan: Applications of a Planar Separator Theorem, Proc. 18th IEEE FOCS Symp. (1977), pp. 162-170.

[NP] J.Nievergelt/F.P.Preparata: Plane-Sweep Algorithms for Intersecting Geometric Figures, CACM 25, 10(Oct. 1982), pp. 739-747.

[Sh] M.I.Shamos: Geometric Complexity, Proc. 7th ACM STOC (1975), pp. 224-233.