Reprinted from INFORMATION AND CONTROL
Vol. 64, Nos. 1–3, January/February/March 1985
Printed in Belgium

Fast Triangulation of the Plane with Respect to Simple Polygons*

STEFAN HERTEL AND KURT MEHLHORN

FB 10, Universität des Saarlandes,
6600 Saarbrücken, Federal Republic of Germany

Let $P_1,\ldots, P_k$ be pairwise non-intersecting simple polygons with a total of n vertices and s start vertices. A start vertex, in general, is a vertex both of which neighbors have larger x coordinate. We present an algorithm for triangulating $P_1,\ldots, P_k$ in time $O(n + s \log s)$. s may be viewed as a measure of non-convexity. In particular, s is always bounded by the number of concave angles $+1$, and is usually much smaller. We also describe two new applications of triangulation. Given a triangulation of the plane with respect to a set of k pairwise non-intersecting simple polygons, then the intersection of this set with a convex polygon Q can be computed in time linear with respect to the combined number of vertices of the $k+1$ polygons. Such a result had only be known for two *convex polygons*. The other application improves the bound on the number of convex parts into which a polygon can be decomposed.

1. INTRODUCTION

A *triangulation* of a finite point set V in the plane is any maximal set of pairwise non-intersecting straight line segments between points in this set. A triangulation of a set $P_1,\ldots, P_k$ of pairwise non-intersecting simple polygons is a triangulation of $V = V_1 \cup \cdots \cup V_k$, where V_i is the vertex set of P_i such that all edges of the polygons are edges of the triangulation. A triangulation of a set $P_1,\ldots, P_k$ of polygons naturally decomposes into an inner and an outer part. The *inner* (*outer*) triangulation consists of exactly those edges of the traingulation which are contained in an odd (even) number of polygons. Figure 1 gives an example. Polygon edges are shown solid. Dashed lines are inner triangulation edges, dotted lines outer triangulation edges.

Triangulations have numerous applications, e.g., closest point problems (Lee and Preparata, 1977; Lipton and Tarjan, 1977), and polygon triangulations serve for area calculations as well as for solving visibility and internal path problems (Chazelle, 1982).

* This paper is a revised and expanded version of a paper presented at the International Conference on "Foundations of Computation Theory" held in Borgholm, Sweden, August 21–27, 1983.

0019-9958/85 $3.00

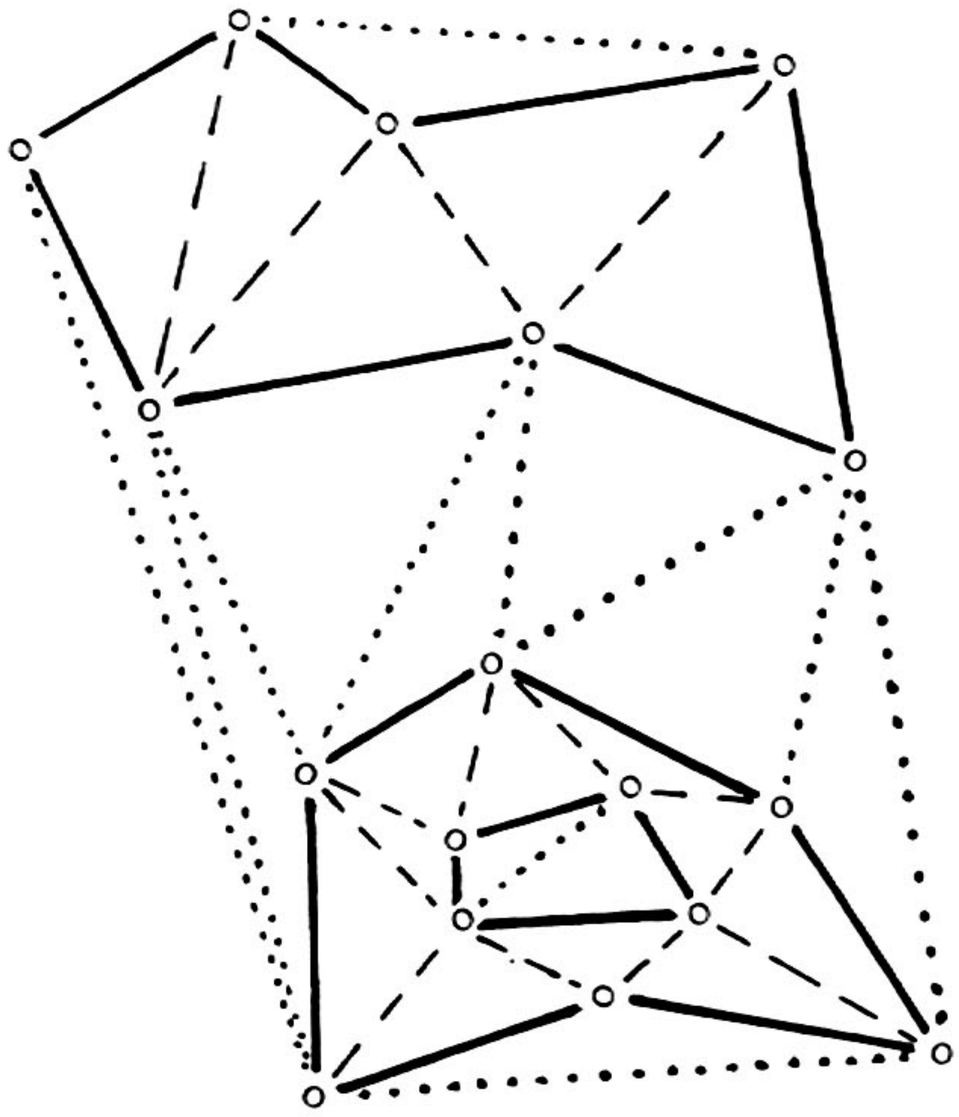


FIG. 1. Triangulation of a set of 3 polygons.

In this paper we show the following theorems. Let $P_1,\ldots, P_k$ be a set of pairwise non-intersecting simple polygons. Let n be the total number of vertices, and let s be the total number of start vertices, i.e., vertices that have smaller x coordinates than both their respective neighbors.

THEOREM. *A triangulation of $P_1,\ldots, P_k$ can be constructed in time $O(n + s \log s)$ and space $O(n)$.*

We also describe two novel applications of triangulation.

THEOREM. *Let Q be a convex polygon with m vertices. Then the intersection of Q with $P_1,\ldots, P_k$ can be computed in time $O(n + m + s \log s)$.*

THEOREM. *Let P be a simple polygon with n vertices and s start vertices. Then in time $O(n + s \log s)$ one can decompose P into at most $4 \cdot \mathrm{OPT}$ convex parts.*

Previously, a linear time bound for intersecting polygons (observe that our bound is linear if we assume a triangulation of $P_1,\ldots, P_k$) had been known for two *convex* polygons only (Shamos, 1975). The best factor known so far for the number of convex parts into which a simple polygon can be decomposed was 4.333 (Chazelle, 1982).

$O(n \log n)$ algorithms to triangulate the interior of a simple n-gon have been proposed by Garey *et al.* (1978) and by Chazelle (1982). It is an open question whether the lower bound of $\Omega(n \log n)$ on the time for computing any triangulation of n points in the plane (Shamos, 1975) can be beaten if the points are vertices of a simple polygon. Some research has been direc-

ted towards improving the time bound for special polygons, e.g., there are linear-time algorithms for star-shaped polygons (by Schoone and van Leeuwen, 1980).

Recently, Chazelle and Incerpi (1984) have described a divide-and-conquer triangulation algorithm that runs in time $O(n \log u)$, with $u \leqslant n$. The parameter u measures the so-called sinuosity of the polygon, which is the number of times the boundary alternates between spirals of opposite orientation; u is very small for most polygons arising in practice. Depending on the polygon parameters s and u, there are several cases in which their algorithm is more efficient than ours. It does not handle sets of simple polygons, however.

Our triangulation algorithm is based on plane-sweep (Nievergelt and Preparata, 1982; see Mehlhorn, 1984b, for a textbook discussion). In Section 2 we will exhibit the necessary data structures and describe an $O(n \log n)$ plane-sweep algorithm for computing an inner triangulation of a simple polygon P. This algorithm matches the time bound of the previous algorithms but has the additional advantage of striking simplicity. Correctness and timing analysis are almost self-evident. In Section 3 we will then modify the algorithm so as to achieve running time $O(n + s \log s)$ for the inner triangulation of a simple polygon. The main additional idea is to modify plane-sweep such that it stops the sweep line only at start vertices, and not at all vertices. The improved algorithm constructs the same triangulation as the basic algorithm and therefore its correctness is also easily seen. In Sections 4 and 5 we will then modify the improved algorithm so as to handle outer triangulation of a simple polygon (Sect. 4) and then many polygons (Sect. 5). Section 6 is devoted to the novel applications.

2. Data Structures and Basic Algorithm

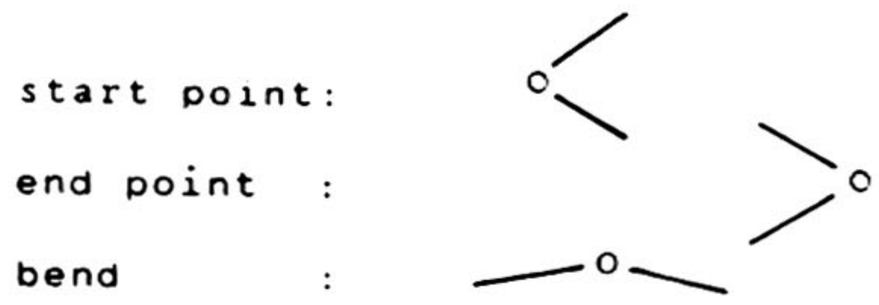
It is our goal in this section to triangulate the interior of a simple polygon P, i.e., to find a subdivision of P into triangles, without introducing new vertices. Vertices will be denoted by lower case letters p and q. Figure 2 gives an example. Our algorithm will operate upon four data structures that will be described in the first part of this section. The second part then presents the algorithm, including its straightforward timing analysis.

2.1. *The Basic Data Structures*

Our triangulation algorithm will operate upon four data structures. Their basic form will be modified in later sections as needed. In addition to

the x structure and the y structure that can be found in most plane-sweep algorithms (compare Nievergelt and Preparata, 1982; Mehlhorn, 1984b), we introduce two specific data structures. The "c-structure" represents those parts of the polygon to the left of the current sweeping line of which the triangulation is not finished, yet. The "g-structure" is the desired output structure; it represents triangles constructed so far, together with their edges.

The x-structure X. X is a simple queue containing the vertices of the polygon yet to be processed, sorted in order of increasing x coordinate, and secondarily in order of increasing y coordinate. The special case of vertical edges is detailed below. We naturally assume that the polygon boundary does not contain three subsequent collinear vertices. Each point can then be uniquely classified into one of three main categories:



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

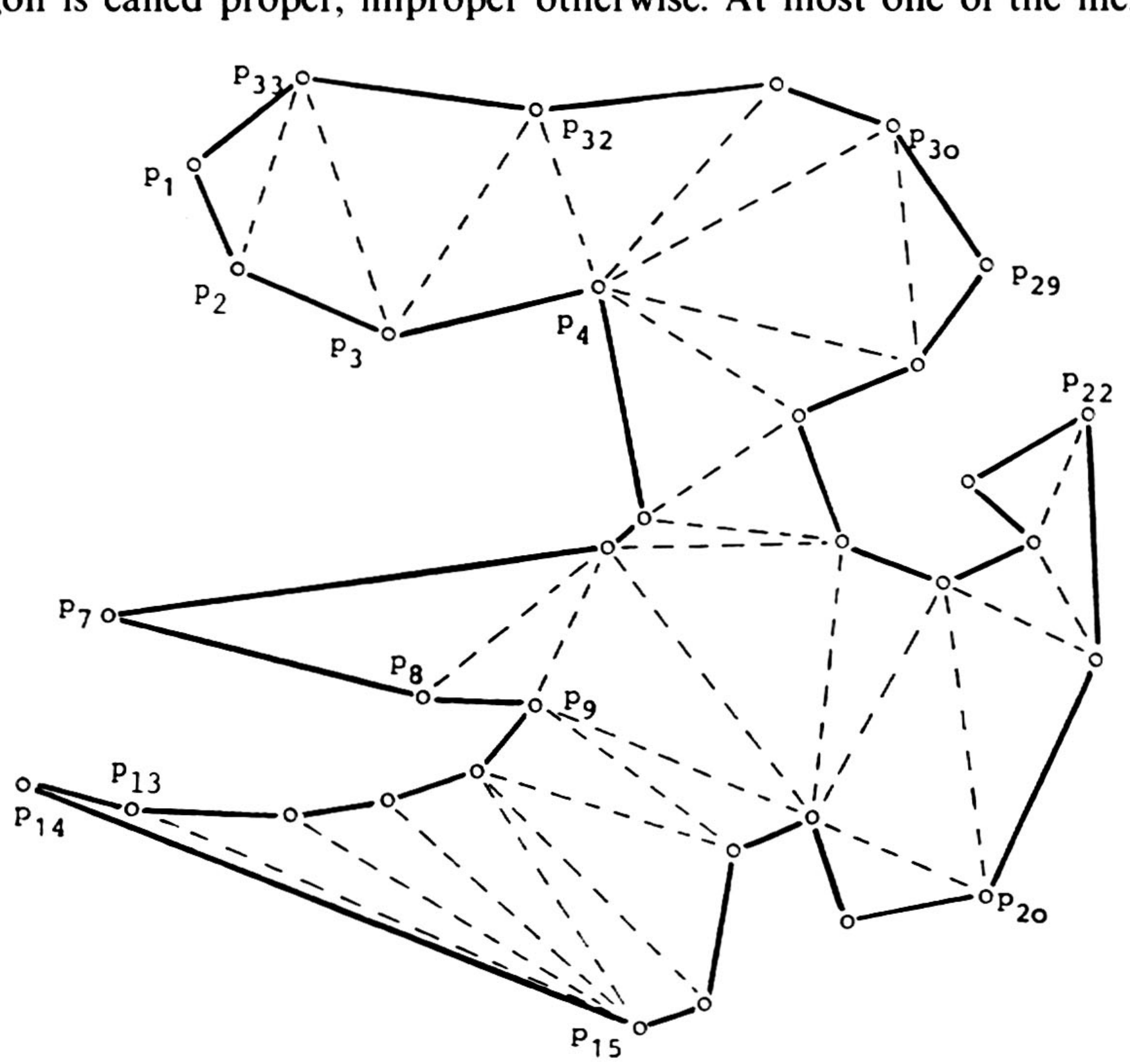


FIG. 2. Inner triangulation of a simple 33-gon. Dashed lines are triangulation edges.

edges of such a point can be vertical (special case). We associate a predicate SPEC(p) with each point. SPEC(p) is true iff there is an edge between p and a point with equal x coordinate and bigger y coordinate. This other point is then called co_p. Intuitively, the idea is to always process p and co_p together. Figure 3 gives an example of a set of four points with identical x coordinate, containing one pair of end points and one pair of start points.

This classification—the *type* of a point—is tagged to each entry in the queue X. The triangulation algorithm removes one point from X at a time. At each point it performs a transaction as described in Section 2.2. In Figure 3 p and co_p were processed but q and co_q are not processed yet.

The y-structure Y. Consider the state of a sweep between two points with different x coordinates. The vertical sweep line cuts through edges of

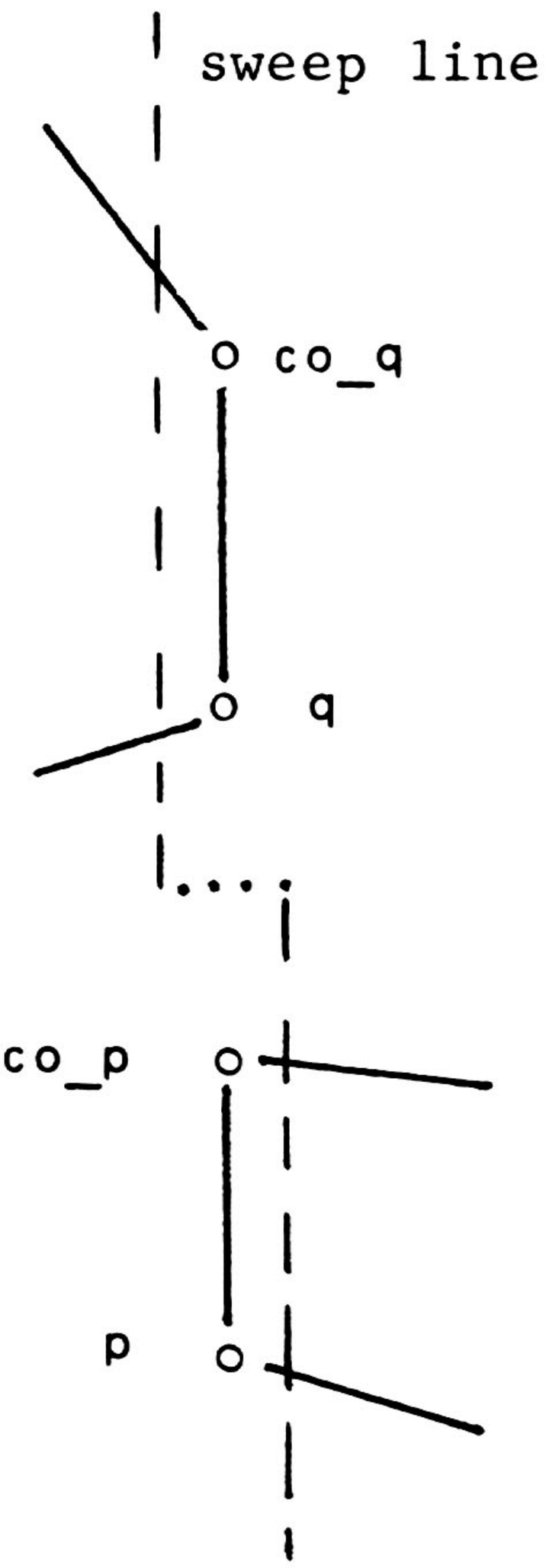


FIG. 3. Four points with the identical x coordinate.

P which partition it into intervals. If P has a set of vertices with identical x coordinates, the sweep line consists of two vertical parts as long as the set is only partly processed. Consult Fig. 3 for the sweep line (dashed) immediately before processing point q. Intervals inside P alternate with intervals outside P, and are referred to as inintervals and outintervals, respectively. Y describes these intervals in the following manner: It has an entry for each edge of P intersected by the sweep line—henceforth, we will call those edges "active edges"—including two sentinels $+\infty$ and $-\infty$. An edge entry is a formula of the form $y = ax + b$ that defines this edge. This allows to find the y value corresponding to a given x value in constant time. An interval bounded below by edge t and above by edge s will be denoted by $[t, s]$; its type will be either *in* or *out*. Figure 4 gives an example of the y structure between the points p_{13} and p_1 of Fig. 2. The entry e_i contains the formula of the edge between $p_{i-1(\mathrm{mod}\, n)}$ and p_i.

Y is a dictionary (see Aho, Hopcroft, and Ullman, 1974) for the interval boundaries that supports 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 standard operations is modified as follows to tailor them to the intended algorithm:

—FIND(p): Depending on the type of point p it delivers
  —if p is a start point: The two edges s (above) and t (below) of the boundary of P in whose interval $[t, s]$ the point p lies, as described in Section 2.2.
  —if p is an end point: The two edges s (above) and t (below) whose common endpoint is p.

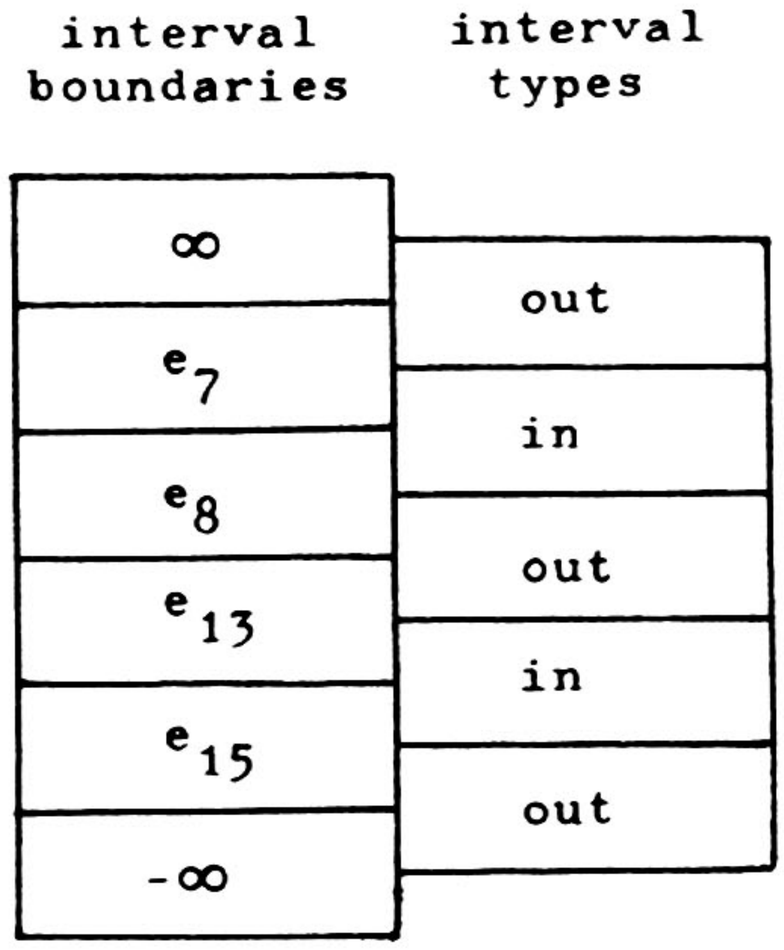


FIG. 4. Example of the y structure.

—if p is a bend: The edge s whose right endpoint is p.

—INSERT(s, $\langle t \rangle$): Given an active edge s of the boundary of P and a type $\langle t \rangle$ (which can be either *in* or *out*) of the interval below s, i.e., of the interval bounded above by s, insert the pair (s, $\langle t \rangle$) at the place determined by the y value of s at the current x value.

—DELETE(s, $\langle t \rangle$); Given an active edge s and a type $\langle t \rangle$ of the interval below s, delete the pair (s, $\langle t \rangle$).

—SUCC(s) (Pred(s)): Given an active edge s and the current x value, deliver the neighboring edge above (below) s.

The required dictionary can be implemented by any one of several kinds of dynamic balanced trees (Aho *et al.*, 1974; Mehlhorn, 1984). Binary search for a given y value at a given position of the sweep line is performed by evaluating the edge formulae $y = ax + b$ along a path from the root to the leaves.

The c-structure C. C assembles information about parts of the polygon passed already whose triangulation depends on points unseen so far. It records for every in-interval the status of the triangulation. More precisely, C stores for every in-interval with upper boundary s a sequence $L(s)$ of vertices of P connected by polygon or triangulation edges. $L(s)$ is doubly linked by means of NEXT and PREV pointers, and starts with the left endpoint of s (cf. Fig. 5). Intuitively, the triangulation is completed to the left and incomplete to the right of this chain. In addition, $RM(s)$ points to a rightmost element of the polygonal chain $L(s)$, i.e., to an element of $L(s)$ with maximal x coordinate. In Fig. 5a, $RM(s)$ points to q_2, in Fig. 5b to q_0.

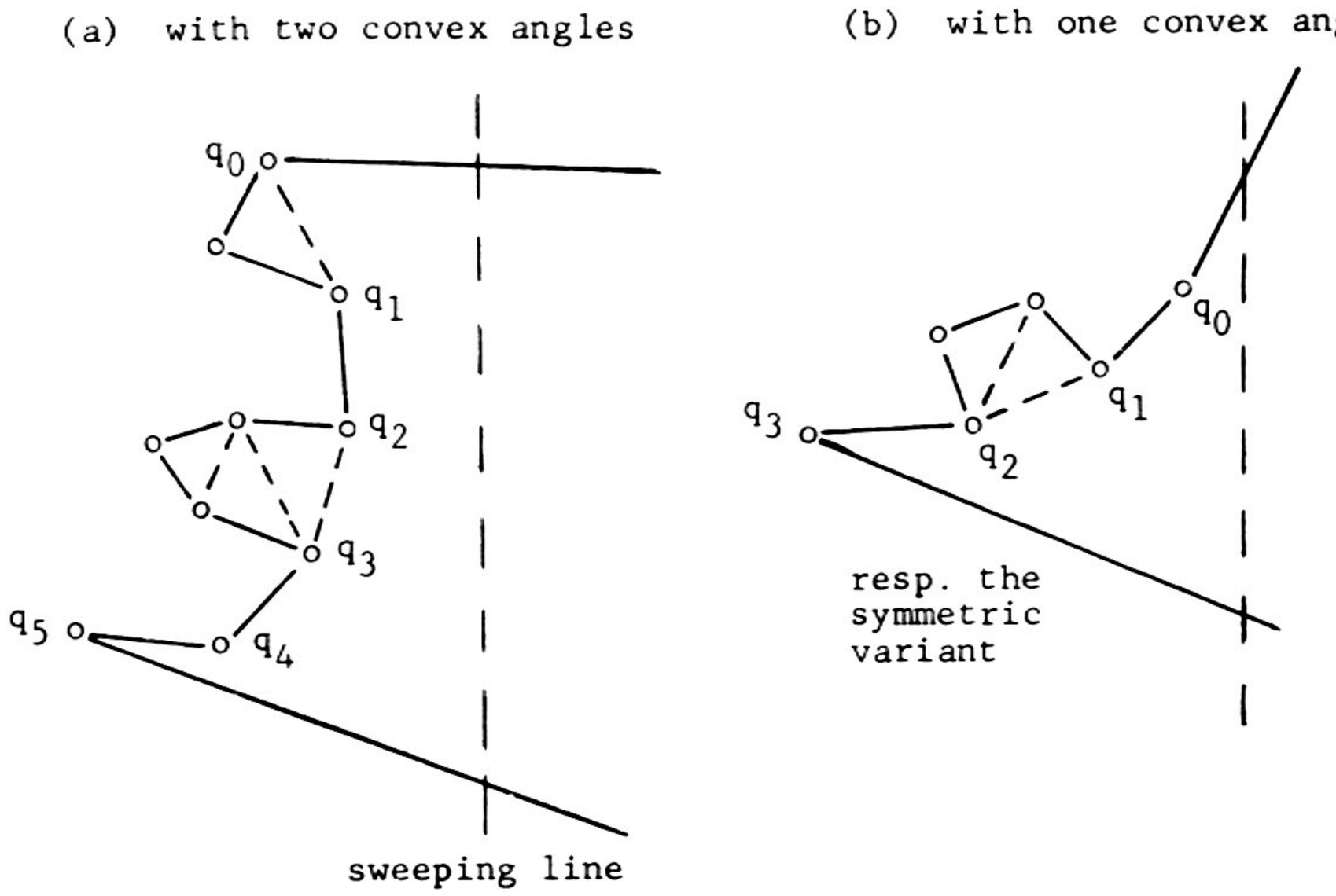


FIG. 5. Forms of polygonal chains $L(s)$. Solid lines are polygon edges, dashed ones are triangulation edges.

We will maintain the following invariant for every in-interval I. Let I be bounded above by edge s:

$L(s)$ is a sequence of points $q_0,\ldots, q_{m(I)}$ with

(i) q_0 is left endpoint of s

(ii) $q_{m(I)}$ is right endpoint of PRED(s)

(iii) $\overline{q_i q_{i+1}}$ is either a polygon or a triangulation edge for $i = 1,\ldots, m(I)-2$.

(iv) The in-angle at q_j is $\geqslant \pi$ (i.e., concave) for $1 \leqslant j \leqslant m(I)-1$.

(v) The part of the polygon bounded to the right by the triangulation edges in $L(s)$ is completely triangulated, already.

C is appended to the y structure, as shown in Fig. 6 for a typical situation, between points p_{11} and p_8 of Fig. 2. The lists $L(s)$ are circular lists with a specified head/tail. Two triangulation edges have been constructed.

The g-structure G. The output structure G 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. TRI is empty initially, EDGES contains the edges of the polygon in one direction of traversal.

2.2. *The Basic Triangulation Algorithm*

The algorithm for constructing triangulation edges has a simple overall structure similar to that of several plane-sweep algorithms. We follow the approach of Nievergelt and Preparata (1982):

```
procedure TRIANGULATE:
begin
X ← n given points, sorted by increasing x coordinate;
Y ← {−∞, ∞}; type ([−∞, ∞]) ← out;
C ← ∅;
TRI ← ∅;
EDGES ← n polygon edges, given in counterclockwise order;
while x ≠ ∅ do
                p ← MIN(X);
                TRANSITION(p)
            od
end {of TRIANGULATE}.
```

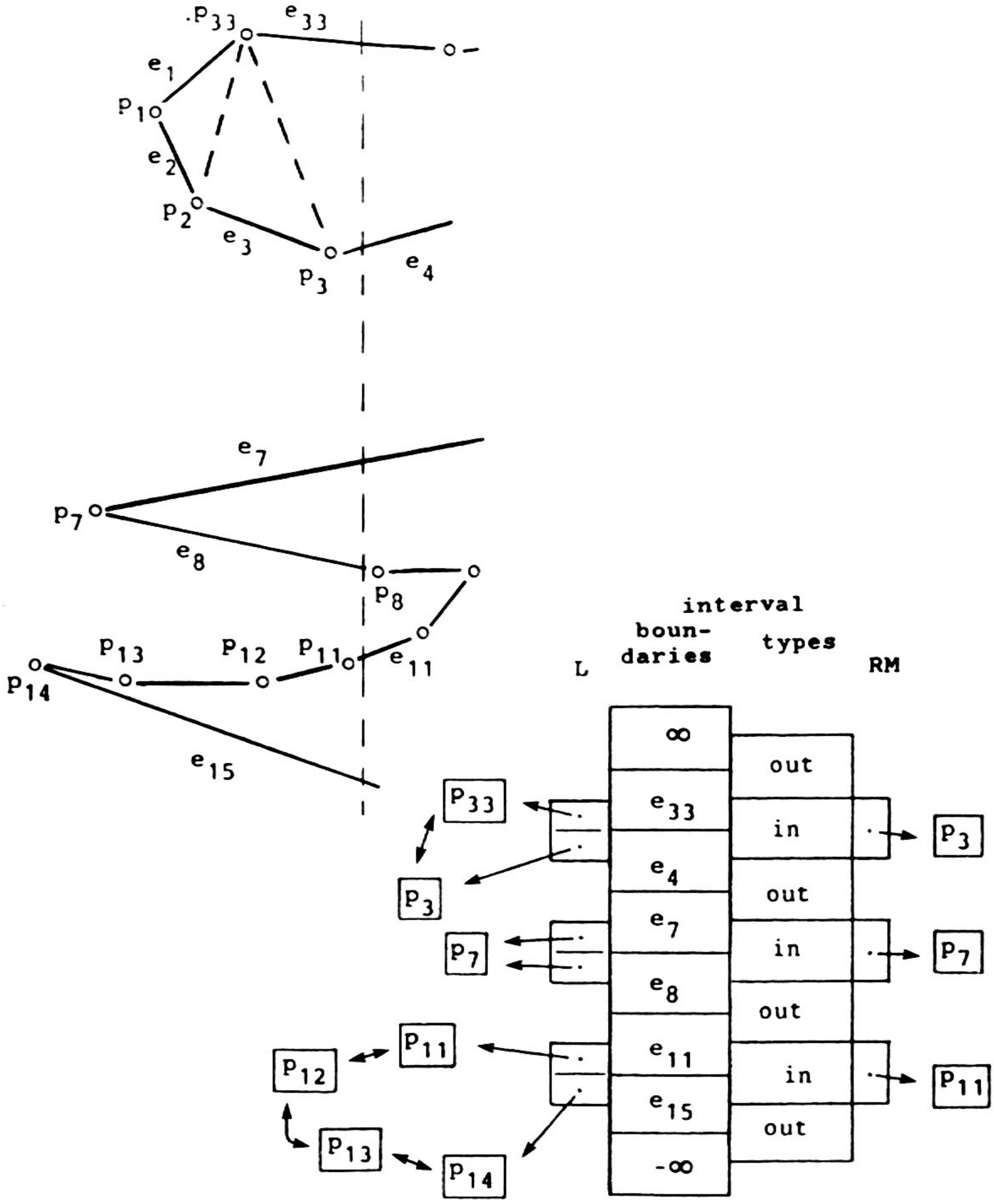


FIG. 6. Structure *Y-C* in a typical situation.

All the work involved in moving the current sweep line across P is performed by procedure TRANSITION. It is invoked exactly n times (less frequently in case of vertical edges). Since each invocation will use $O(\log n)$ time, this will result in an $O(n \log n)$ algorithm.

TRANSITION handles each of the five possible types of the "next point" p differently. We describe each case by a figure, by a verbal exposition, and by a high-level algorithm. The algorithms we give can serve as a guideline for the implementer without restricting him too much. Detailed situations for the cases of start or end points show both the normal and the special (vertical edge) cases; the respective figures include the sweep linr immediately after processing the point(s). In the algorithms, "∘" denotes

the concatenation of two polygonal chains or the appending of a point p to a polygonal chain as new head or tail, respectively (i.e., a point is considered to be naturally embedded in a singleton list).

In most cases, TRANSITION makes use of a procedure CHAIN_TRI that will be specified later. CHAIN_TRI(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 traiangulate in clockwise direction.

Case "proper start" (Fig. 7a). p lies in an out-interval $[t, s]$ (Fig. 7a). We simply split the out-interval into three intervals of types *out*, *in*, *out* and associate a chain only consisting of node p (in special case, of nodes p and co_p) with the in-interval. Also, p is the rightmost node of that chain (Figs. 7b, c).

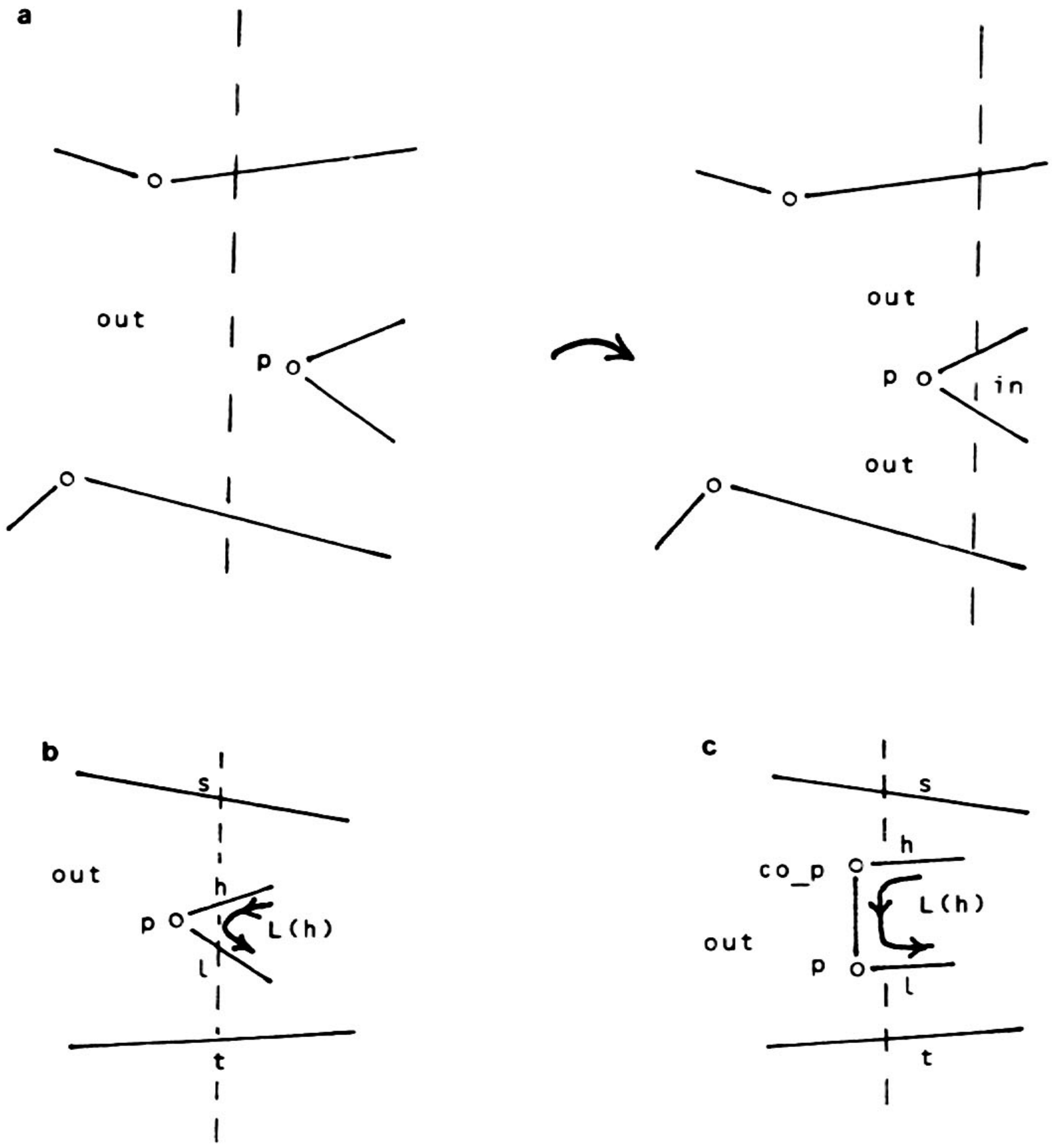


FIG. 7. (a) Transition for a "proper start"; (b), (c) detailed situation for a "proper start."

ALGORITHM.

begin
FIND(p); {delivers the two adjacent active edges t and s in whose interval $[t, s]$ p lies (cf. Figs. 7b.c)}
$l \leftarrow$ low edge starting at p;
$h \leftarrow$ **if** SPEC(p) **then** high edge starting at co_p
else high edge starting at p;
INSERT((1,**out**));
INSERT((h,**in**));
if SPEC(p) **then** $L(h) \leftarrow$ co_$p \circ p$
else $L(h) \leftarrow p$;
$RM(h) \leftarrow p$
end {of the case "proper start"}

Case "bend" (Fig. 8a). Let s be the edge ending in p and let t be the edge starting in p. Then s is on the boundary of an in-interval; let $v_1, \ldots, v_l$ be the associated polygonal chain, where v_1 is the other endpoint of edge s. We add triangulation edges $\overline{pv_2}, \ldots, \overline{pv_i}$ until $\measuredangle(p, v_i, v_{i+1}) \geq \pi$ and change the associated polygonal chain into $p, v_i, \ldots, v_1$. p becomes the new rightmost node of this chain (Figs. 8b).

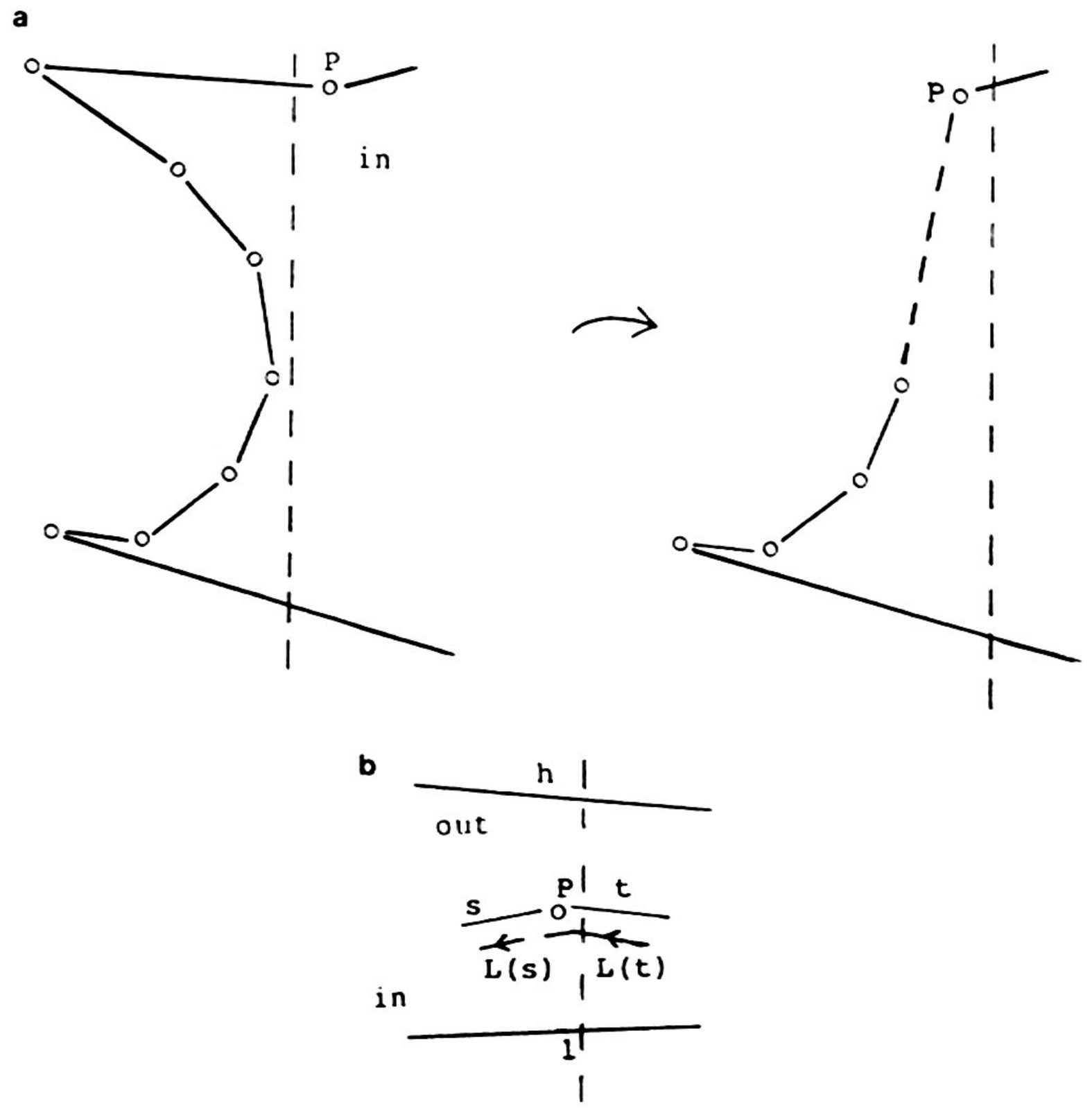


FIG. 8. (a) Transition for a "bend"; (b) detailed situation for a "bend."

ALGORITHM.

```
begin
  FIND(p); {delivers the uniquely determined edge s of which the right
            endpoint is p (cf. Fig. 8b)}
  t ← edge starting at p;
  l ← PRED(s);
  h ← SUCC(s);
  if type([1, s]) = in
    then begin L(t) ← p ∘ L(s);
               replace s by t in Y;
               CHAIN_TRI(t, "cc");
               RM(t) ← p
         end
    else begin {type([s,h]) = in}
               L(h) ← L(h) ∘ p;
               replace s by t in Y;
               CHAIN_TRI(h, "c");
               RM(h) ← p
         end
end {of the case "bend"}
```

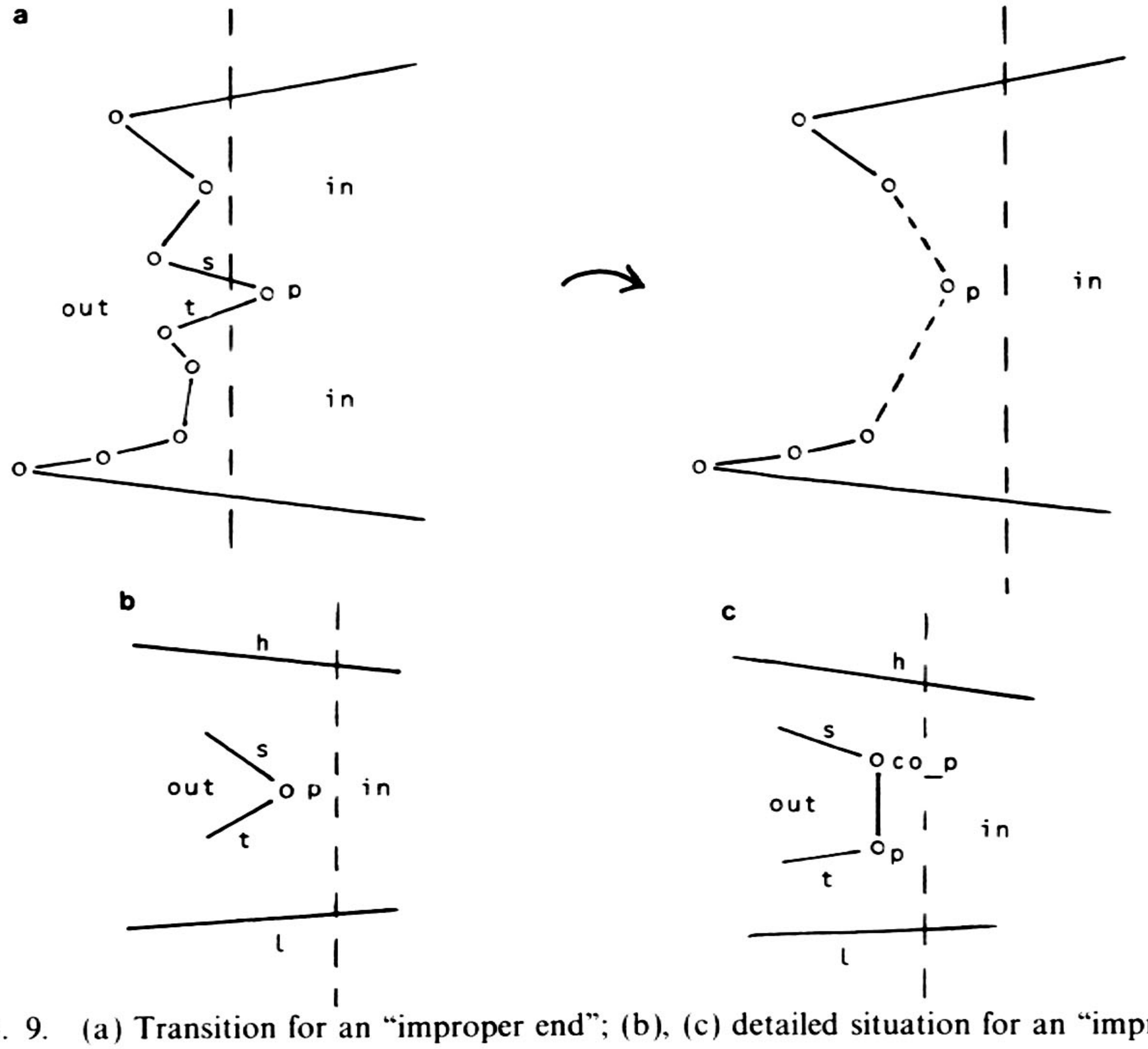


FIG. 9. (a) Transition for an "improper end"; (b), (c) detailed situation for an "improper end."

Case "improper end" (Fig. 9a). s, the upper edge ending at p (resp. co_p,), bounds an in-interval from below; t, the lower edge ending at p, bounds an in-interval from above. We can interpret p as *a* bend for both associated polygonal chains, and can triangulate along both chains analoguously to the case above. Then we merge both in-intervals by deleting edges s and t from the y structure. Also, we concatenate both associated chains, using p as a connecting element. p (resp. co_p,) becomes the new rightmost element of this chain (Figs. 9b,c).

ALGORITHM.

```
begin
  FIND(p); {delivers the two active edges t and s of which the common
            endpoint is p; if SPEC(p) the two edges with right endpoints
            p and co_p (cf. Figs. 9b,c)}
  l ← PRED(t);
  h ← SUCC(s);
  if SPEC(p) then begin L(h) ← L(h) ∘ co_p;
                        CHAIN_TRI(h, "c");
                        L(t) ← p ∘ L(t)
                  end
             else begin L(h) ← L(h) ∘ p;
                        CHAIN_TRI(h, "c");
                        remove tail p from L(h);
                        L(t) ← p ∘ L(t)
                  end;
  CHAIN_TRI(t, "cc");
  L(h) ← L(h) ∘ L(t);
  if SPEC(p) then RM(h) ← co_p;
             else RM(h) ← p;
  DELETE((t, in));
  DELETE((s, out))
end {of the case "improper end"}
```

Case "improper start" (Fig. 10a). p lies in the in-interval $[t, s]$ with associated polygonal chain $v_1, \ldots, v_l$ that has v_z as its rightmost node. We can certainly add edge $\overline{v_z p}$ to the triangulation. Analoguously to the case "bend," we then triangulate along the chain in both directions, starting from p via v_z. Now we split the in-interval into three intervals of types *in*, *out*, *in* and split the polygonal chain appropriately between the two new in-intervals. p (resp. co_p,) becomes the rightmost node of both in-intervals (Figs. 10b, c).

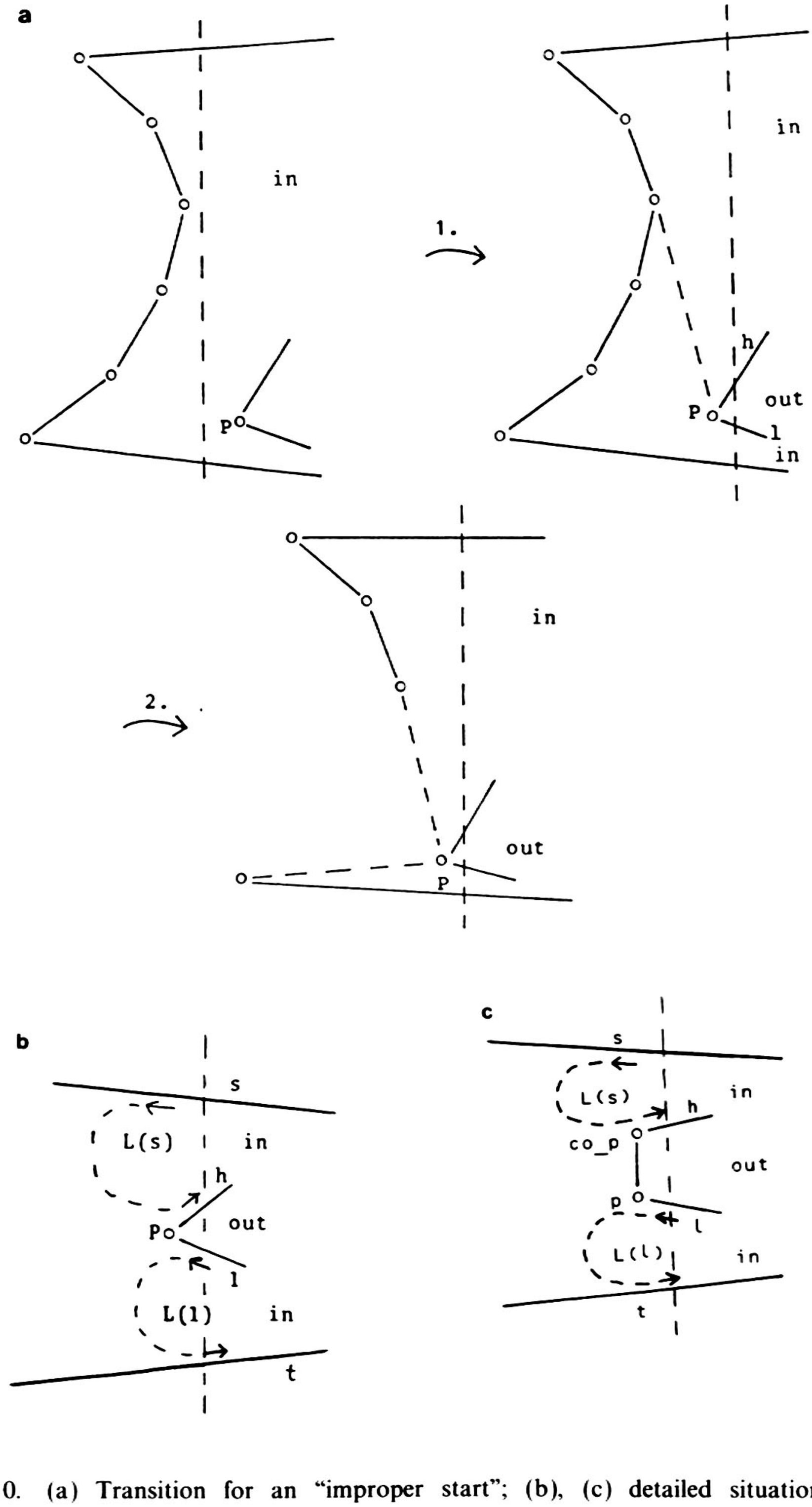


FIG. 10. (a) Transition for an "improper start"; (b), (c) detailed situation for an "improper start."

ALGORITHM.

```
begin
  FIND(p); {delivers the two neighboring active edges t and s in whose
            interval [t, s] p lies (cf. Figs. 10b,c)}
  l ← low edge starting at p;
  h ← if SPEC(p) then high edge starting at co_p
                 else high edge starting at p;
  q ← RM(s);
  "add qp̄ to EDGES";
  INSERT((l, in));
  INSERT((h, out));
  RM(s) ← if SPEC(p) then co_p
                     else p;
  RM(l) ← p;
  L(l) ← p ∘ "remainder of L(s) starting at q";
  L(s) ← "L(s) up to and including q" ∘ p;
  if SPEC(p) then L(s) ← L(s) ∘ co_p;
  CHAIN_TRI(s, "c");
  CHAIN_TRI(l, "cc")
end {of the case "improper start"}
```

Case "proper end" (Fig. 11a). p lies in an in-interval $[t, s]$ with associated polygonal chain $L(s)$. The invariant of Section 2.1 guarantees that we can finish off the triangulation of $L(s)$ since we can "see" all its nodes from p (resp. from one of p and co_p,). Then we delete the edges t and s from Y and merge the two adjacent out-intervals (Figs. 11b, c).

ALGORITHM.

```
begin
  FIND(p); {delivers the two active edges t and s whose common
            endpoint is p (p and co_p, respectively;
            cf. Figs. 11b,c)}
  l ← PRED(t);
  h ← SUCC(s);
  L(s) ← if SPEC(p) then co_p ∘ L(s) ∘ p
                    else p ∘ L(s);
  CHAIN_TRI(s, "cc");
  if SPEC(p) then CHAIN_TRI(s, "c");
  DELETE((t, out));
  DELETE((s, in))
end {of the case "proper end"}
```

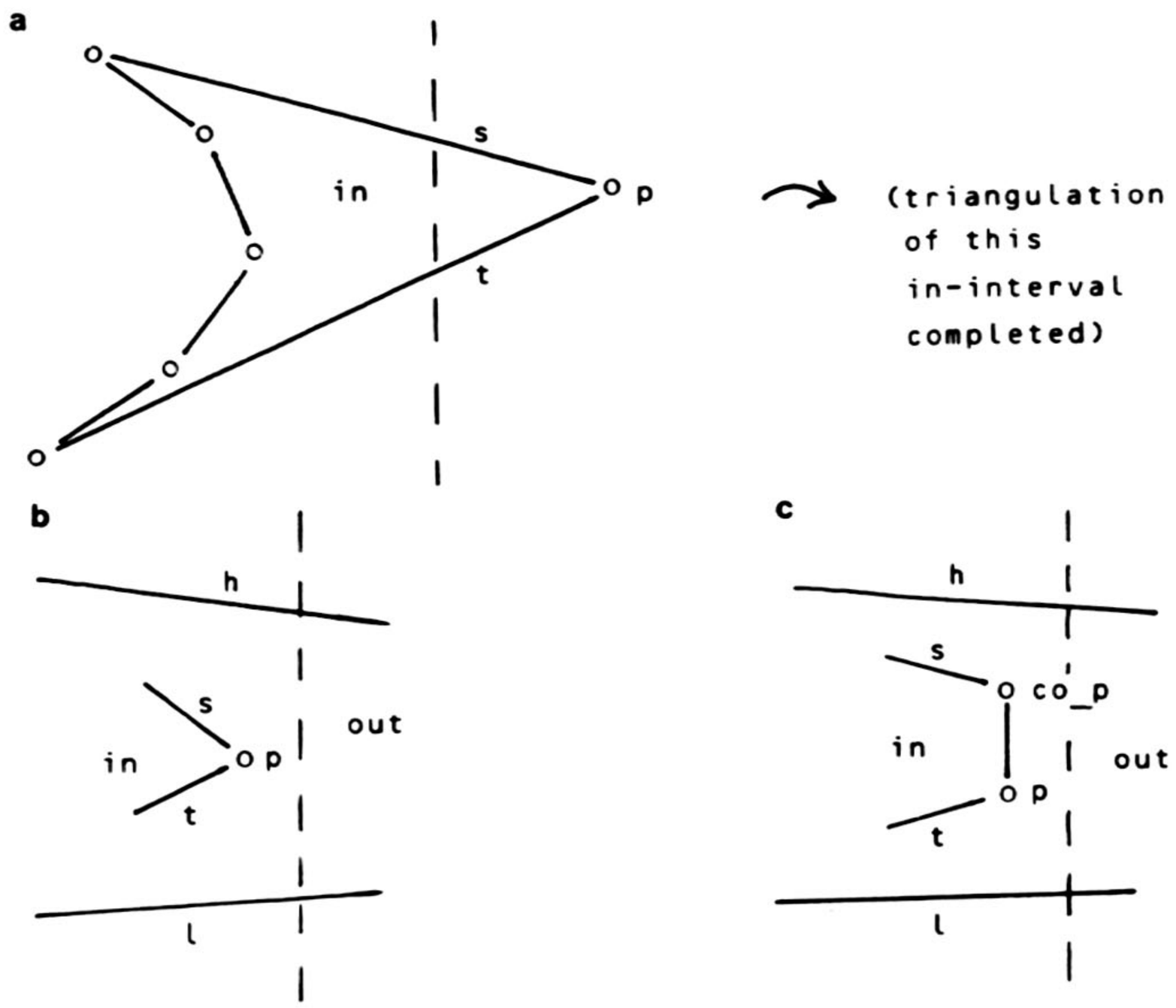


FIG. 11. (a) Transition for a "proper end"; (b), (c) detailed situation for a "proper end."

The procedure CHAIN_TRI(e, dir) starts at a point p at one end of a polygonal chain $L(e)$, e being an active edge, and it triangulates "along" $L(e)$ as far as possible. Triangulation proceeds counterclockwise from the head of $L(e)$ (dir = "cc"), or clockwise from the tail of $L(e)$ (dir = "c"). As long as the in-angle at the next point on $L(e)$ is convex, a new triangulation edge is drawn.

```
procedure CHAIN_TRI(e, dir):
begin
  if dir = "cc" then begin p ← head of L(e);
                           q ← NEXT(p)
                     end
                else begin p ← tail of L(e);
                           q ← PREV(p)
                     end;
  while (|L(e)| > 2) and (the in-angle at q is convex) do
    begin w ← if dir = "cc" then NEXT(q)
                            else PREV(q);
          "draw a triangulation edge from p to w, add it and the new
           triangle to the output structure G";
          "delete q from L(e)";

    end
end {of CHAIN_TRI}.
```

The procedure CHAIN_TRI is correct if edges of $L(e)$ and new triangulation edges are pairwise non-intersecting. This, however, follows from proposition (iv) of the invariant for $L(e)$ that was given in Section 2.1.

THEOREM 1. *The running time of the algorithm* TRIANGULATE *for triangulating a simple n-gon is* $O(n \log n)$.

Proof. The initial sorting takes time $O(n \log n)$. It should be clear that the construction of a new triangulation edge and the updating of the appropriate polygonal chain as well as of G can be done in time $O(1)$. Thus the running time of CHAIN_TRI is proportional to the number of new triangulation edges, and the total time spent in CHAIN_TRI is $O(n)$. A dictionary operation on Y takes time $O(\log n)$ at most; thus one call to TRANSITION takes time $O(\log n)$, apart from the time spent in CHAIN_TRI. This yields an overall running time of $O(n \log n)$. ∎

3. IMPROVEMENT OF THE ALGORITHM

The algorithm TRIANGULATE needed time $O(n \log n)$ even for "trivial" non-convex polygons like the one in Fig. 12, which could be easily triangulated in linear time, starting from the sole non-convex angle. This is due to the required sorting and to the consideration of n transitions. Our goal is to drastically reduce the number of points where we spend "much" time. Therefore we will drop the "bends" from explicit consideration and handle them "on the go." Only "start" and "end" points remain transitions. It is easy to see that any simple polygon has exactly as many start points as end points (counting a vertical edge as one start or end point). Let s be the number of start points. The improved algorithm TRIANGULATION will then work in time $O(n + s \log s)$.

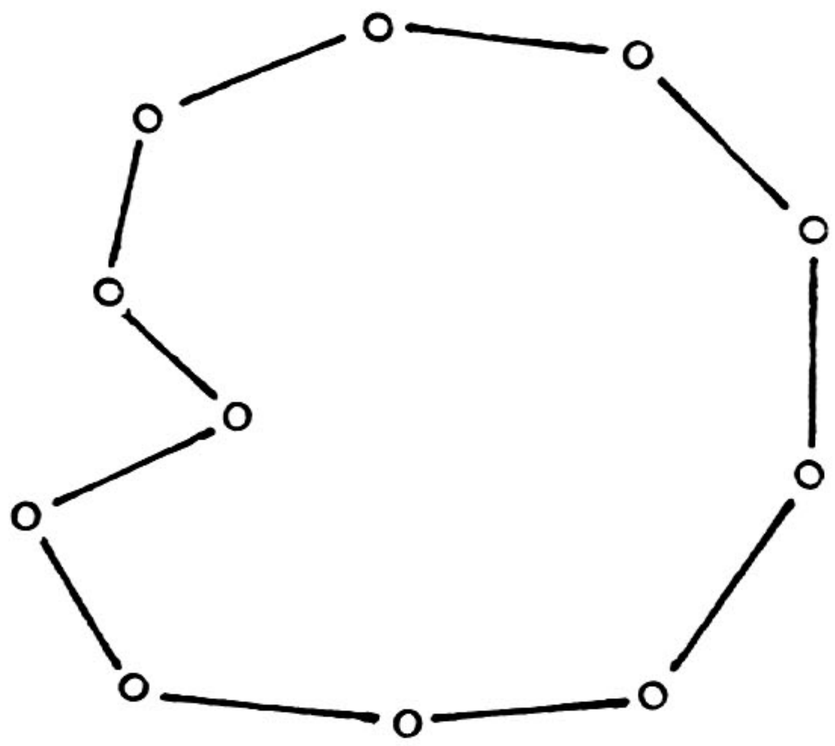


FIG. 12. "Trivial" non-convex polygon.

Refinement of the Data Structures

As mentioned above, the x structure now contains only the $2s$ start and end points. The y structure now is different from that in algorithm TRIANGULATE in that it does not simply reflect the status of the sweep line any longer. Instead, the y structure consists of vertical parts, henceforth to be called "local sweep lines," some of which may lag behind the global sweep line. The global sweep line refers to the sweep line of our original algorithm TRIANGULATE. There is one local sweep line for each in-interval. As before, we associate a polygonal chain and a pointer to the rightmost node on the chain with each in-interval. Also, the invariant given in Section 2.1 stays valid for the polygonal chains.

We introduce a second invariant that refers to the ordering of active edges in the y structure. Note that the number of in-intervals changes only after start and end points. Thus, if we conceptually follow the boundary of polygon P from each of the active edges stored in the y structure to the right until the global sweep line is reached, we can associate a point on the sweep line with every active edge in the y structure. We maintain the invariant that the ordering of active edges in the y structure coincides with the ordering of the associated points on the global sweep line. Briefly, the ordering of in-intervals is the same as it would have been in the basic algorithm TRIANGULATE.

Parts (a) and (c) of Fig. 13 illustrate the new concept. Figure 13a shows the situation after processing point p_1 of Fig. 2; p_{13} is still unprocessed since the local sweep line for the low in-interval lags behind. Figure 13c shows a possible situation after processing point p_9. Here, the four bends in the upper in-interval are unprocessed.

Relating a New Point p to Y

If the relative position of a new point p with respect to the current intervals is to be found, we are confronted with the problem that not all intervals necessarily extend to the global sweep line. Thus, comparing p with active edges does not always help us.

The solution we propose is to 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 that does not extend to the global sweep line, we walk from this edge along the polygon boundary 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. This way, we close the gap between a local sweep line and the global sweep line. We only encounter bends in this process, and handle them exactly in the same way as we did in the previous algorithm

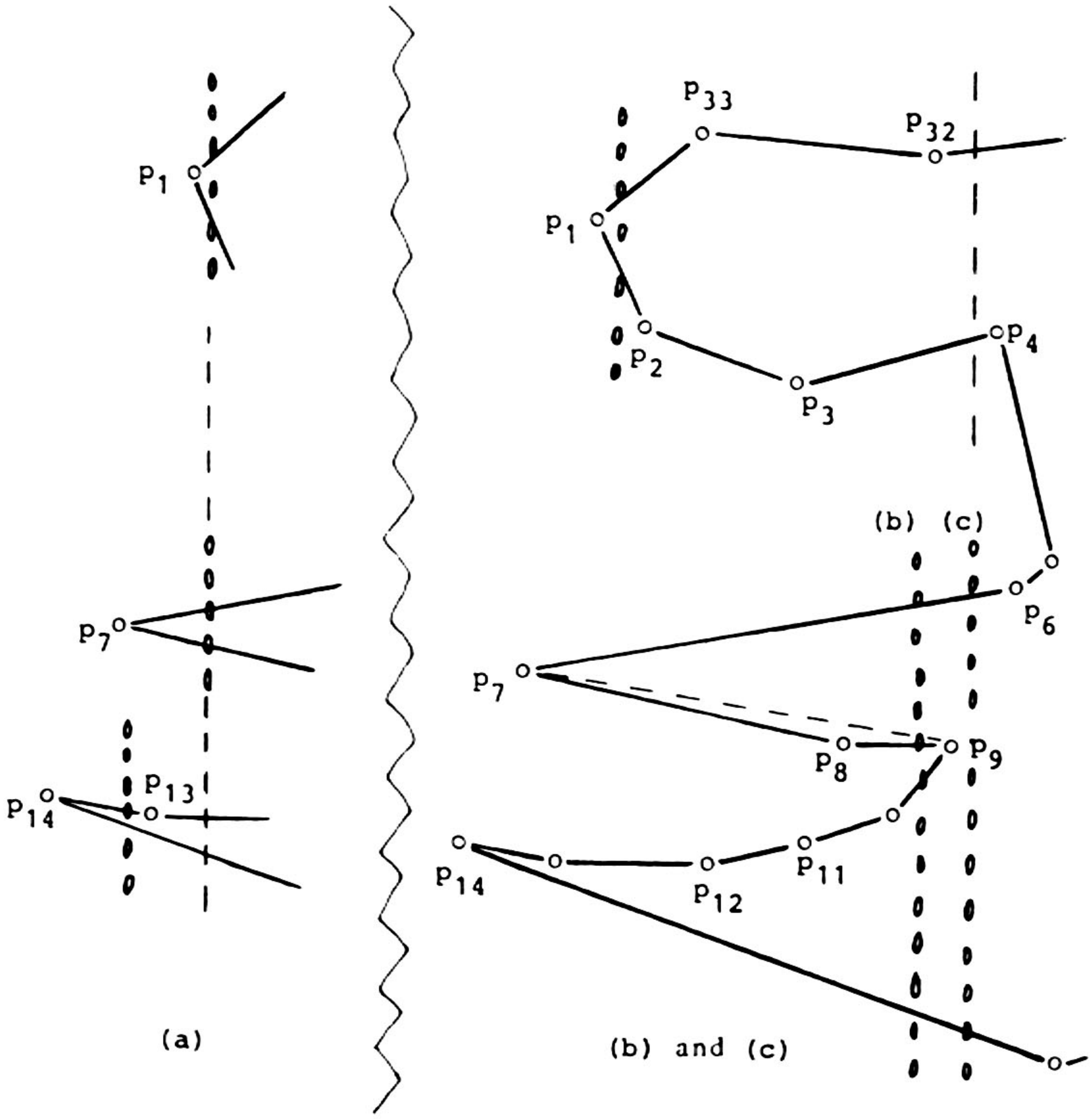


FIG. 13. Possible situations in sample polygon: the dashed line is the global sweep line; heavily dashed lines represent local sweep lines.

TRIANGULATE, except that we need not search for them. Figure 13b shows a situation immediately before finding point p_9. We have extended the local sweep lines for the two low in-intervals to the current global sweep line, and can now process p_9. This creates the triangulation edge $\overline{p_7 p_9}$.

The correctness of this method follows from the fact that the transition at bends was completely local, and that the ordering of in-intervals is the same as at the corresponding state of TRIANGULATE. We only process some points at a different time.

THEOREM 2. *The algorithm* TRIANGULATION *runs in time* $O(n + s \log s)$ *and needs space* $O(n)$ *for simple n-gons with s start points.*

Proof. All the points processed "on the go" as described above are bends. We find each one of them in time $O(1)$ by walking along the polygon boundary, and then they are handled like bends in Section 2.2. For an edge starting at a bend, we have to find its successor and its predecessor.

INSERT/DELETES are not necessary; thus, processing a bend takes time $O(1)$ apart from the time spent in CHAIN_TRI.

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

4. Constructing an Outer Triangulation

It is our goal in this section to find an outer triangulation for a simple n-gon P, as defined in Section 1. Figure 14 gives an example, including "triangulating edges" that extend from the vertices of the convex hull of P to infinity.

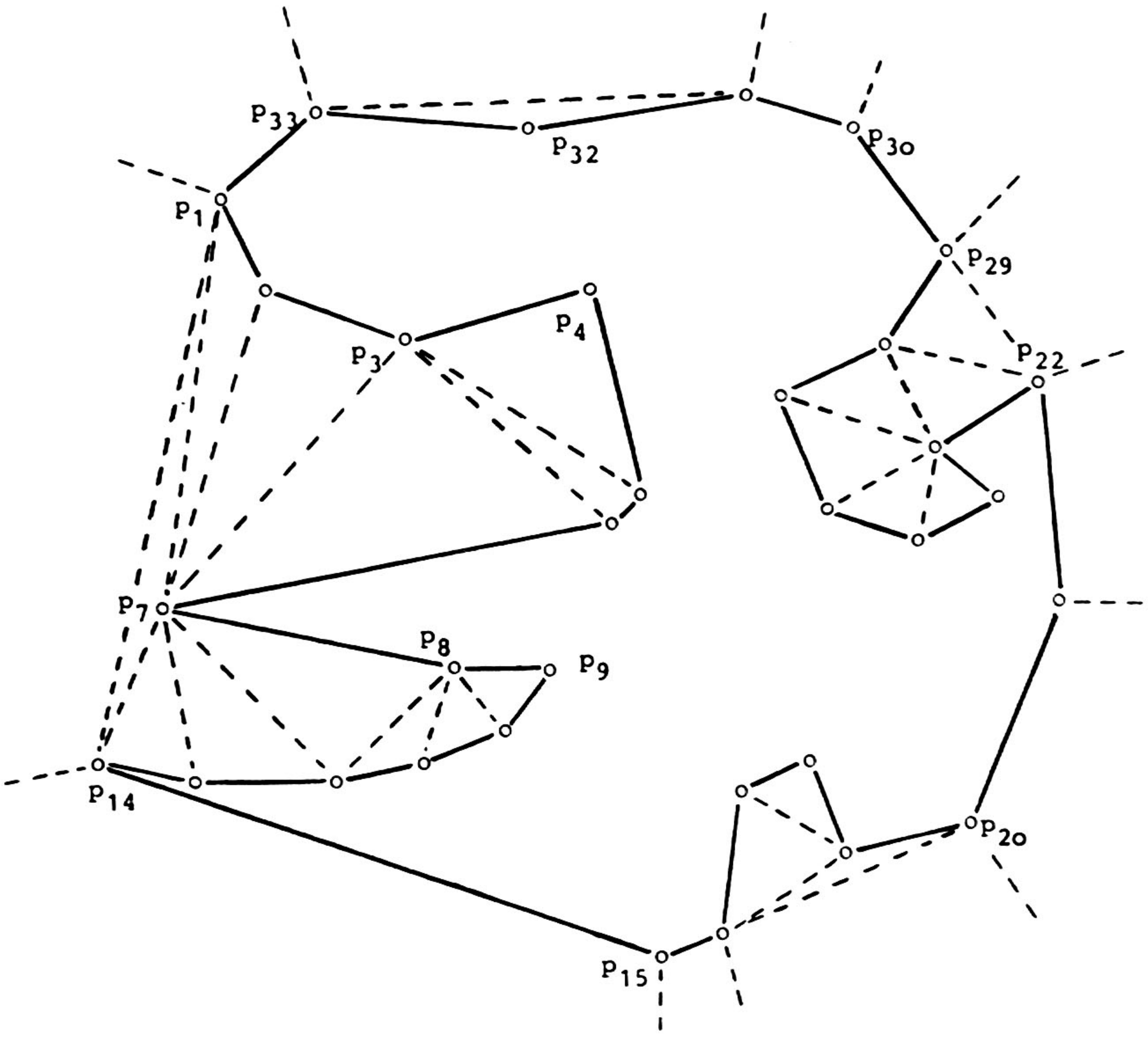


FIG. 14. Outer triangulation of a simple 33-gon.

We will reduce outer to inner triangulation. To achieve this result, we conceptually add two edges to P that extend from $LM(P)$, the leftmost vertex of P, i.e., the point of P with lowest x coordinate, to $y = -\infty$ and $y = +\infty$, respectively, and that do not intersect any edge of P. (If there are several leftmost points, we conceptually connect them by vertical edges.) Compare Fig. 15 that shows the "conceptual leftmost edges" and the types of the intervals in the y structure after processing the leftmost point.

After having processed $LM(P)$, both the interval above the high edge starting at $LM(P)$ and the interval below the low edge starting at $LM(P)$ must now be considered as in-intervals. Our algorithm proceeds as in Section 3, but it constructs an outer triangulation of P. After all points of P have been processed, one in-interval is left; its associated polygonal chain is the convex hull of P.

To find "infinite triangulation edges" as mentioned above, we simply choose an interior point of the convex hull, construct rays from there through the corners of the hull and drop the ray segments inside P.

We have achieved the following result.

THEOREM 3. *An outer triangulation of a simple n-gon with s start vertices can be constructed in time* $O(n + s \log s)$ *and space* $O(n)$.

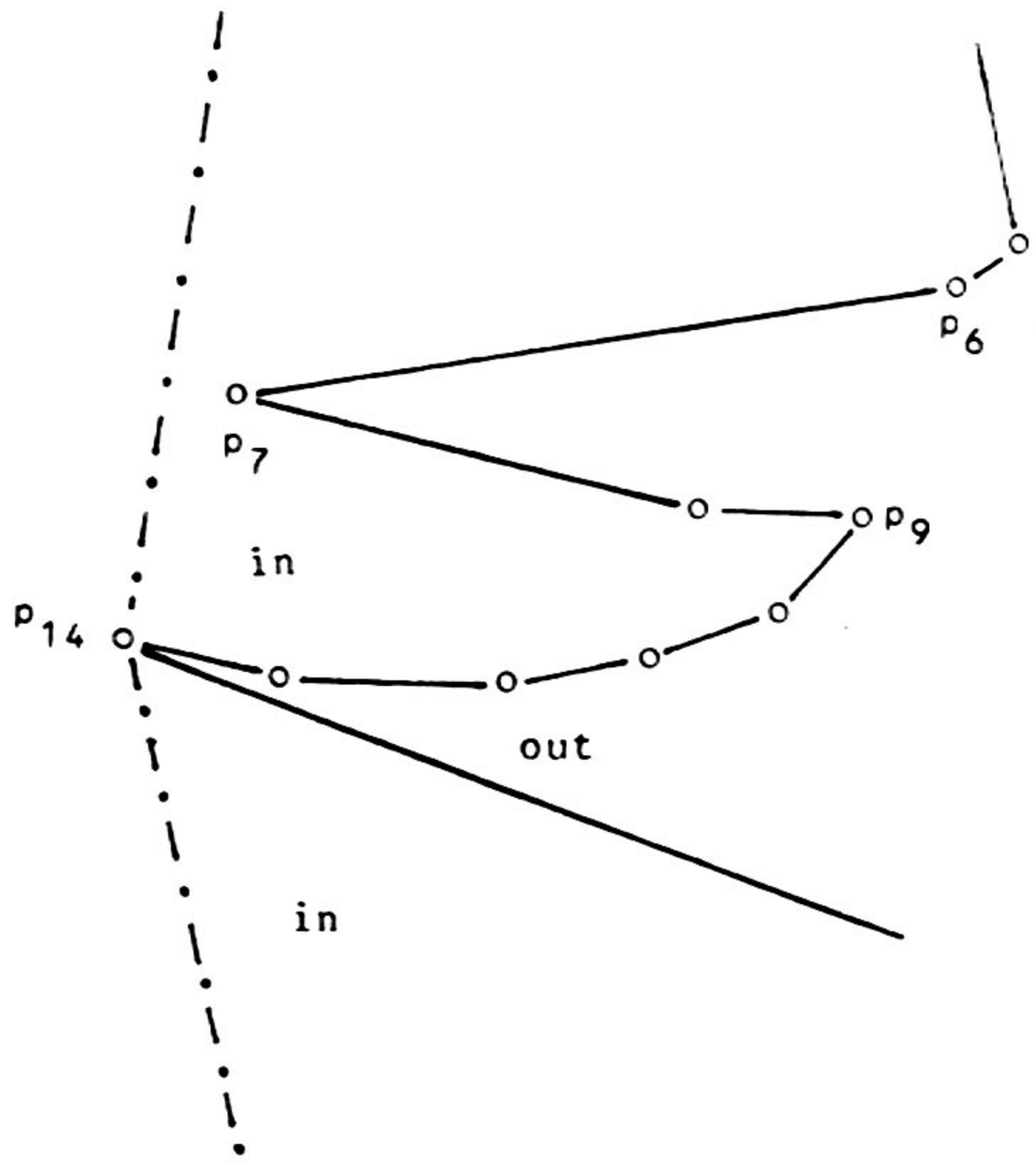


FIG. 15. "Conceptual edges" for outer triangulation.

5. Triangulating a Set of Polygons

We return to our original problem of triangulating a set of k pairwise non-intersecting simple polygons with a total of n vertices and s start vertices.

With the results of Sections 3 and 4, such a triangulation does not pose any additional difficulties. We solve the problem in two passes. First we construct an inner triangulation, using the methods of Section 3. Since the y structure stores only edges and knows about in-intervals, it constructs, for a set of intervals being part of different polygons, the same triangulation as if the intervals were connected with each other somewhere "further to the right." A proper right endpoint of an embedded polygon, for example, is handled in exactly the same manner as is an endpoint of an in-interval of just *one* polygon.

In a second pass, we construct an outer triangulation, applying the methods of Section 4. The same arguments as for the inner triangulation are valid and show the correctness of this procedure.

This leads to our final result.

THEOREM 4. *Let $P_1,\ldots,P_k$ be a set of pairwise non-intersecting simple polygons with a total of n vertices and s start vertices. A triangulation of this set can be constructed in time $O(n+s \log s)$ and space $O(n)$.*

6. Applications

6.1. *Intersection of a Set of k Polygons and a Convex Polygon Q*

Shamos (1975) showed how to compute the intersection of two convex polygons in linear time. We extend his result as follows.

THEOREM 5. *Let $P_1,\ldots,P_k$ be a set of simple polygons with a total of n vertices, and let Q be a convex m-gon: Assume that a triangulation of the plane with respect to $P_1,\ldots,P_k$ is available. Then $(P_1 \cup \cdots \cup P_k) \cap Q$ can be computed in linear time, i.e., in time $O(m+n)$.*

Proof. Let T be a triangulation of the plane with respect to $P_1,\ldots,P_k$, given as in Section 2. In time $O(n)$ we can certainly add the "infinite triangles" outside the convex hull (cf. Sect. 4, Fig. 14). This yields a division of the plane into a total of $2n$-2 triangles.

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

$$O(m + \sum s_i) = O(m + n), \text{ by the argument above.} \quad \blacksquare$$

COROLLARY. *Let $P_1, \ldots, P_k$ be a set of simple polygons with a total of n vertices and s start vertices. Let Q be a convex polygon with m vertices. Then $(P_1 \cup \cdots \cup P_k) \cap Q$ can be computed in time $O(n + m + s \log s)$ and space $O(n)$.*

The best solution hitherto known required time $O((n+m)\log(n+m))$ (Bentley and Ottmann, 1979; Brown, 1981).

6.2. *Decomposing a Simple Polygon into Convex Parts*

In general, convex geometric objects are easier to handle than non-convex ones. As for polygons, Chazelle (1982) showed how to decompose, in time $O(n \log n)$ and space $O(n)$, a simple n-gon P into fewer than 4.333 OPT convex pieces, without introducing new vertices, where OPT is the minimum number of convex pieces necessary to partition P. Chazelle obtains his results by applying a separator theorem recursively. Given the convex parts, it is, of course, easy to obtain a triangulation. We proceed the other way round and start with a triangulation. This yields a solution that improves upon Chazelle's result.

THEOREM 6. *Let P be a simple n-gon, and let U 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. Let r be the number of convave in-angles of P. Observe that $\text{OPT} \geqslant r/2 + 1$ since one partitioning edge is necessary for each convave angle. We will 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 a convex angle. Call edge e essential for point p if it cannot be dropped because it divides a concave angle at point p. The following lemma completes the proof.

LEMMA. *Not more than two triangulation edges are essential for each point with concave in-angle.*

Proof. Let p be common endpoint of polygon edges e_1 and e_2. Let p have a concave in-angle, and let t_1, t_2, t_3 be three different triangulation edges that are essential for p.

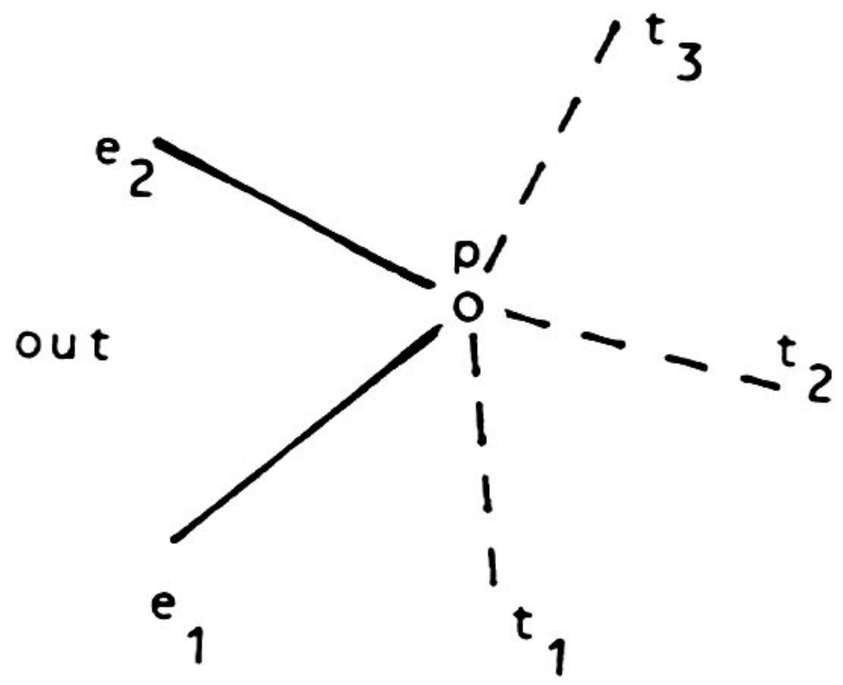


FIGURE 16

Given two edges a and b with common endpoint p, let $\measuredangle(a, b)$ the angle resulting from turning edge b counterclockwise around p towards a. The following then holds (compare Fig. 16):

$$\measuredangle(e_1, e_2) > 0,$$
$$\measuredangle(e_2, t_2) \geqslant \pi \qquad \text{(since } t_3 \text{ is essential)},$$
$$\measuredangle(t_3, t_1) \geqslant \pi \qquad \text{(since } t_2 \text{ is essential)},$$
$$\measuredangle(t_2, e_1) \geqslant \pi \qquad \text{(since } t_1 \text{ is essential)}.$$

Hence $\measuredangle(e_2, e_1) \geqslant 2\pi$, a contradiction. Thus not more than $2r$ triangulation edges are left in the decomposition. ∎

COROLLARY. *Let P be a simple n-gon with s start vertices. Then P can be decomposed into fewer than $4 \cdot OPT$ convex subpolygons in time $O(n + s \log s)$ and space $O(n)$.*

Note added in proof: Theorem 2 has been obtained independently by Dan Gordon. His result is described in D. Gordon: "The Critical Points Method in Computational Geometry," Tech. Report, Dept. of Computer Science, Univ. of Cincinnati, Cincinnati, Ohio, 45221.

REFERENCES

AHO, A. V., HOPCROFT, J. E., ULLMAN, J. D. (1974), "The Design and Analysis of Computer Algorithms," Addison–Wesley, Reading, Mass.

BENTLEY, J. L. AND OTTMANN, T. A. (1979), Algorithms for reporting and counting geometric intersections, *IEEE Trans. Comput.* **C-28**, No. 9, 643–647.

BROWN, K. Q. (1981), Comments on algorithms for reporting and counting geometric intersections, *IEEE Trans. Comput.* **C-30**, 147–148.

CHAZELLE, B. (1982), A theorem on polygon cutting with applications *in* "Proc. 23rd IEEE Sympos. Found. of Comput. Sci," pp. 339–349.

CHAZELLE, B. AND INCERPI, J. (1984), "Triangulation and Shape-Complexity," Technical report, Dept. of Comput. Sci., Brown Univ., Providence, R.I. 02912.

GAREY, M. R., JOHNSON, D. S., PREPARATA, F. P., AND TARJAN, R. E. (1978), Triangulating a simple polygon, *Inform. Process. Lett.* **7** No. 4, 175–179.

HERTEL, S. (1984), "Sweep-Algorithmen für Polygone und Polyeder," Ph. D. Dissertation FB 10, Univ. des Saarlandes, Saarbrücken.

HERTEL, S., AND MEHLHORN, K. (1983), Fast triangulation of simple polygons, *in* "Proc. 1983 Int. Conf. Found. of Comput. Theory." KARPINSKI, M. Ed.), Lecture Notes in Computer Sci. Vol. Conf. Found. of Comput. Theory," Lecture Notes in Computer Sci. Vol. 158, pp. 207–218, Springer Verlag, Berlin/New York.

LEE, D. T. AND PREPARATA, F. P. (1977), Location of a point in a planar subdividion and its applications, *SIAM J. Comput.* **6** 594–606.

LIPTON, R. J., TARJAN, R. E. (1977), Applications of a planar separator theorem, *in* "Proc. 18th IEEE Sympos. *Found. of Comput. Sci.*," pp. 162–170.

MEHLHORN, K. (1984a), Data structures and algorithms 1: Sorting and searching, EATCS Monographs on Theoret. Comput. Sci. Vol. 1, Springer Verlag, Berlin/New York.

MEHLHORN, K. (1984b), Data structures and algorithms 3: Multi-dimensional searching and computational geometry, EATCS Monographs on Theoret. Comput. Sci. Vol. 3. Springer Verlag, Berlin/New York.

NIEVERGELT, J., AND PREPARATA, F. P. (1982), Plane-sweep algorithms for intersecting geometric figures, *Comm. ACM* **25**, No. 10, 739–747.

SHAMOS, M. I. (1975), Geometric complexity, *in* "Proc. 7th ACM Sympos. Theory of Comput." pp. 224–233.

SCHOONE, A. A., AND LEEUWEN, J. V. (1980), "Triangulating a Star-shaped polygon," Techn. Report RUU-CS-80-3, Dept. of Comput. Sci., Univ. of Utrecht, April.

Printed by the St. Catherine Press Ltd., Tempelhof 41, Bruges, Belgium