REPORT R-699 NOVEMBER, 1975 UILU-ENG 75-2234

CSL COORDINATED SCIENCE LABORATORY

LOCATION OF A POINT IN A PLANAR SUBDIVISION AND ITS APPLICATION

D.T. LEE
F.P. PREPARATA

UNIVERSITY OF ILLINOIS – URBANA, ILLINOIS

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* LOCATION OF A POINT IN A PLANAR SUBDIVISION AND ITS APPLICATION		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER R-699; UILU-ENG 75-2234
7. AUTHOR(s) D. T. Lee and F. P. Preparata		8. CONTRACT OR GRANT NUMBER(s) DAAB-07-72-C-0259
9. PERFORMING ORGANIZATION NAME AND ADDRESS Coordinated Science Laboratory University of Illinois at Urbana-Champaign Urbana, Illinois 61801		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Joint Services Electronics Program		12. REPORT DATE November, 1975
		13. NUMBER OF PAGES 26
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Planar Region Identification
Computational Geometry
Point Location
Computational Complexity
Analysis of Algorithms
Spatial Convex Inclusion
Inclusion in Polygon

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

Given a subdivision of the plane induced by a planar graph with n vertices, in this paper we consider the problem of identifying which region of the subdivision contains a given test point. We present a search algorithm, called point-location algorithm, which operates on a suitably preprocessed data structure. The search runs in time at most $O((\log_2 n)^2)$, while the preprocessing task runs in time at most $O(n \log_2 n)$ and requires $O(n)$ storage. The methods are quite general, since an arbitrary subdivision can be transformed in time at most $O(n \log n)$ into one to which the preprocessing

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

20. ABSTRACT (continued)

procedure is applicable. This solution of the point location problem yields interesting and efficient solutions of other geometric problems, such as spatial convex inclusion and inclusion in an arbitrary polygon.

UILU-ENG 75-2234

LOCATION OF A POINT IN A PLANAR SUBDIVISION AND ITS APPLICATION

by

D. T. Lee and F. P. Preparata

This work was supported in part by the Joint Services Electronics Program (U.S. Army, U.S. Navy and U.S. Air Force) under Contract DAAB-07-72-C-0259.

LOCATION OF A POINT IN A PLANAR SUBDIVISION AND ITS APPLICATIONS[†]

D. T. Lee*and F. P. Preparata[#]
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign

November 1, 1975

Abstract

Given a subdivision of the plane induced by a planar graph with n vertices, in this paper we consider the problem of identifying which region of the subdivision contains a given test point. We present a search algorithm, called point-location algorithm, which operates on a suitably preprocessed data structure. The search runs in time at most $O((\log_2 n)^2)$, while the preprocessing task runs in time at most $O(n \log_2 n)$ and requires $O(n)$ storage. The methods are quite general, since an arbitrary subdivision can be transformed in time at most $O(n \log n)$ into one to which the preprocessing procedure is applicable. This solution of the point location problem yields interesting and efficient solutions of other geometric problems, such as spatial convex inclusion and inclusion in an arbitrary polygon.

[†]This work was supported in part by the Joint Services Electronics Program (U.S. Army, U.S. Navy, and U.S. Air Force) under Contract DAAB-07-72-C-0259.

*Department of Computer Science, University of Illinois.

[#]Department of Electrical Engineering, University of Illinois.

LOCATION OF A POINT IN A PLANAR SUBDIVISION AND ITS APPLICATIONS

D. T. Lee and F. P. Preparata

Coordinated Science Laboratory

University of Illinois at Urbana-Champaign

1. Introduction

We shall consider the following problem referred to as "point location": A given planar straight line graph on n vertices induces a subdivision of the plane; given a test point P, find which region of this subdivision contains P. We shall assume that the graph is originally given as the collection of the edge-lists of its n vertices.

To solve this problem, one must produce an algorithm and its associated data structure, obtained by preprocessing the original data structure. Thus, from a computational viewpoint, any solution to the point location problem should be evaluated with respect to the following three measures: (i) the *search time*, that is, the number of operations required to locate the test point in the subdivision; (ii) the *preprocessing time*, that is, the number of operations required to construct the data structure postulated by the search algorithm; (iii) the amount of *storage* required by the preprocessed data structure.

Some workers have recently considered this problem. For example Ketelsen [1] proposed an algorithm whose search time is O(n). The most

recent and interesting results are due to Shamos [2]. His approach is an adaptation of a nearest-neighbor algorithm, also developed by him. It consists of tracing a sheaf of parallel lines through each of the n points, thereby slicing the plane in strips, called "slabs", each of which contains no vertex of the graph and is subdivided by transversal segments in at most $O(n)$ regions. Since each of these slab-regions belongs to a unique plane region, $O(\log n)$ [1] comparisons are sufficient to locate the slab and additional $O(\log n)$ comparisons are sufficient to locate the region. Thus, Shamos's search algorithm has running time $O(\log n)$, but, as is easily realized, both preprocessing time and storage are $O(n^2)$. Shamos also outlines another possible procedure [2], based on recursively splitting the planar subdivision by means of polygonal lines, so that at each step half of the regions to be subdivided lie on either side of a polygonal line. Such an algorithm could achieve a search time $O((\log n)^2)$, but the polygonal lines must be "star-shaped" (in his terminology) and the existence of such lines is an open question even for the special case of a triangulation. In this paper we show that, if one does not insist on recursively halving sets of regions, a suitable set of polygonal lines can be found within a preprocessing time $O(n \log n)$; the resulting data structure can be stored in $O(n)$ memory locations and the search time is at most $O((\log n)^2)$.

The point location problem occurs in a number of applications in operations research, pattern recognition, job scheduling, etc. We

---

[1] Here and hereafter "log" denotes logarithm in base 2.

shall show in this paper how it can also be applied to solve a number of other problems, among which is spatial convex inclusion.

This paper is organized as follows. In the next section we shall introduce some useful geometric constructs, specifically, monotone complete sets of chains of a planar graph, and justify their application to the solution of the point location problem. In Section 3 we shall give a procedure for the construction of such sets of chains for an important subclass of planar subdivisions and illustrate the resulting data structure. In Section 4 we shall illustrate in detail the point location algorithm. In Section 5 we shall extend the results to arbitrary planar subdivisions. Finally, in Section 6 we shall describe further applications of the outlined algorithms to problems in computational geometry.

2. Complete sets of chains.

We begin by recalling some standard geometric definitions and introducing some useful notions.

A connected planar straight line (PSL) graph G with a finite number of vertices subdivides the plane into nonempty regions of which exactly one is infinite. These regions form a *planar subdivision* and will be also referred to as the regions of G.

Definition 1. Two vertices u and v of a PSL graph G are said to be *antipodal* if there are two parallel straight lines through u and v, respectively, with the property that all vertices of G be on one side of each line.

Definition 2. Let u and v be two antipodal vertices of a PSL graph G. Consider a chain $C = (u = u_1, u_2, \ldots, u_p = v)$ between u and v. C is said to be *monotone* with respect to a straight line ℓ if the orthogonal projections $\ell(u_1), \ldots, \ell(u_p)$ of the vertices of C on ℓ are ordered.

Example: The chain in Figure 1a is monotone with respect to ℓ whereas there is no straight line with respect to which the chain in Figure 1b is monotone.

It is convenient to extend a chain $(u_1, \ldots, u_p)$ which is monotone with respect to a line ℓ with half-lines parallel to ℓ beyond each terminal point u_1 and u_p. Any such extended chain thus divides the plane into two regions. In the sequel, a monotone chain will always be thought of as being extended in the manner.

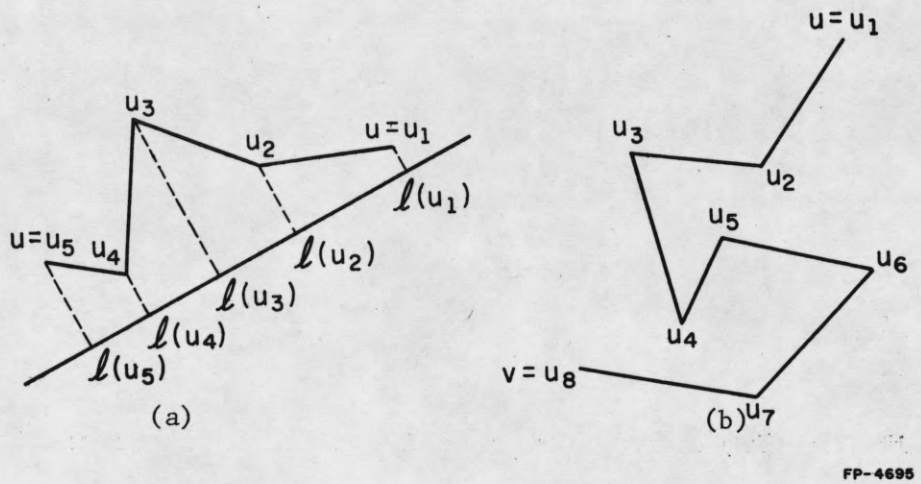
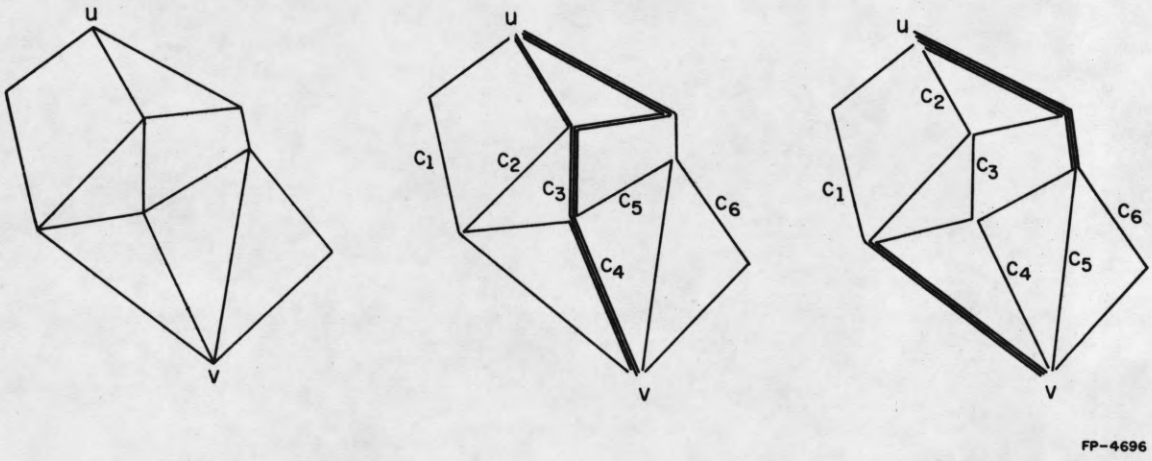


Figure 1. Examples of chains.

Definition 3. A set C of chains between two antipodal points of a PSL graph G is said to be *complete* if the following two conditions hold:

(i) each edge of G belongs to at least one chain of C;

(ii) for any two chains C_1 and C_2 of C, the vertices of C_1 which are not vertices of C_2 lie on the same side of C_2.



(a) (b) (c)

Figure 2. A PSL graph G and two complete sets of chains between u and v in G.

Given a set of chains C between two points of a PSL graph, it is natural to introduce a binary relation "$\prec$" on C as follows: for $C_1, C_2 \in C$, the notation $C_1 \prec C_2$ means that C_1 lies on a selected side of C_2 with respect to a fixed observer. It is then obvious that condition (ii) in Definition 3 implies that $\prec$ on a complete set C is a total ordering, i.e., C is a sequence

$(C_1, C_2, \ldots, C_r)$. As an example, in Figure 2b and 2c we exhibit two complete sets of chains between two antipodal points u and v of the PSL graph G given in Figure 2a.

Definition 4. A complete set of chains of a PSL graph G is said to be *monotone* if all of its members are monotone with respect to the same straight line.

Referring to the preceding example, the set in Figure 2c is monotone, whereas the set in Figure 2b is not.

We shall now motivate our interest in monotone complete sets of chains. Let C be a chain with vertices $u_1, \ldots, u_s$ which is monotone with respect to a straight line ℓ and let P be a test point. We shall call a *discrimination* (of P against C) the operation of deciding on which side of C the point P lies. As Shamos pointed out [2], such discrimination can be performed with O(log s) comparisons of coordinates. In fact (see Figure 3), the projections of the vertices of C on ℓ form a sequence of points $(\ell(u_1), \ldots, \ell(u_s))$. With a binary search, i.e., with $\lceil \log(s+1) \rceil$ comparisons of coordinates, we can determine a unique interval $(\ell(u_i), \ell(u_{i+1}))$ of ℓ which contains the projection $\ell(P)$ of P on ℓ. Next, with a fixed number of arithmetic operations and a single comparison we can determine on which side of the straight line containing $\overline{u_i u_{i+1}}$ the point P lies. This also determines on which side of C the point P lies. Suppose now that an ordered monotone set of chains $C = (C_1, C_2, \ldots, C_r)$ is given and let s be the maximum

number of vertices in any chain of this set. Applying binary search to the set of chains C, with $\lceil \log(r + 1) \rceil$ discriminations we can determine a unique pair of consecutive chains C_j and C_{j+1} of C which comprise the given point P. Since the set C is also complete, by property (i) in Definition 3 the portion of plane comprised between two consecutive members of C is a concatenation of regions of G and (possibly) of chain segments (see Figure 4). Recall now that the discrimination of P against a chain C_j entails the identification of a unique edge $e^{(j)}$ of C_j. Thus, if P has been located

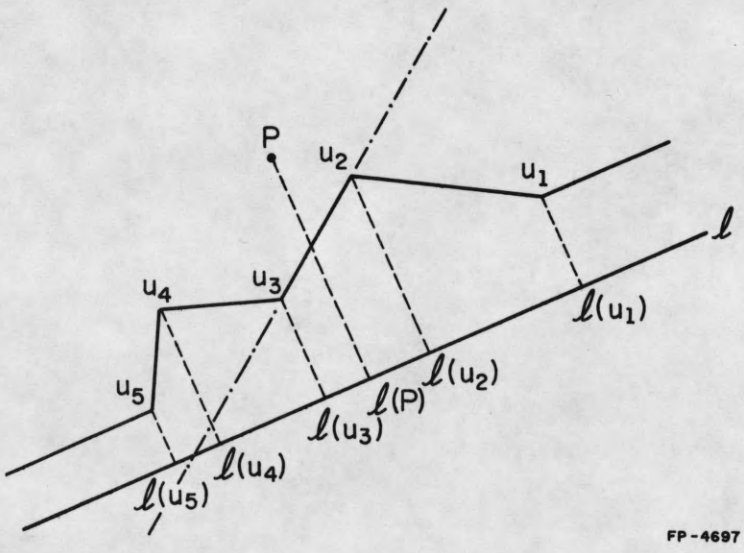


Figure 3. Illustration of a "discrimination" of a point P against a chain C.

between two consecutive chains C_j and C_{j+1}, the two edges $e^{(j)}$ and $e^{(j+1)}$ uniquely identify the region of the graph G to which P belongs, and the point location problem is solved.

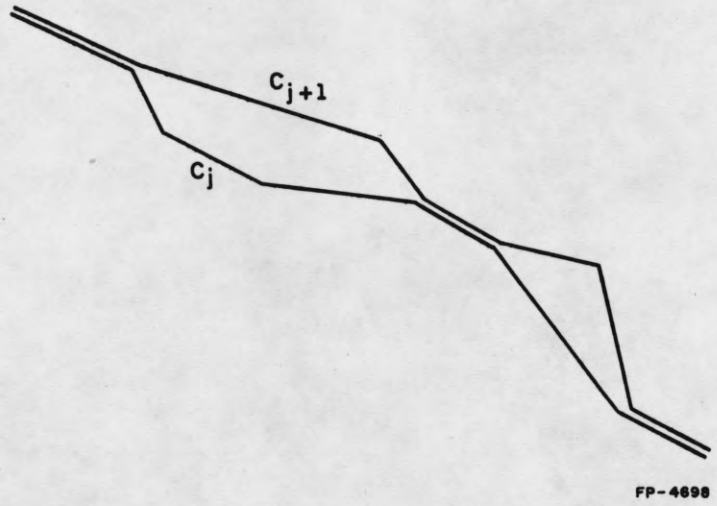


Figure 4. Portion of plane comprised between two consecutive chains C_j and C_{j+1}.

This justifies the usefulness of monotone complete sets of chains of a PSL graph G. The preceding discussion also indicates that at most $O(\log s) \cdot O(\log r)$ comparisons are needed for point location. Notice now that, if G has n vertices, each chain of a complete monotone set $\mathcal{C}$ on G has at most n vertices; moreover, by property (i) in Definition 3, there are at most (n-1) chains in $\mathcal{C}$. It follows that point location requires at most $O((\log n)^2)$ comparisons.

The remaining crucial question is whether an arbitrary PSL graph always admits of a monotone complete set of chains. The answer is in general negative. However, we shall constructively show that a PSL graph G admits of a monotone complete set of chains, provided it satisfies a rather weak requirement. In addition, we shall show that an arbitrary PSL graph can be

easily embedded into one to which the chain construction procedure is applicable. This embedding creates some new "artificial" regions, with no harm, however, to the effective solution of the point location problem.

3. Preprocessing algorithm (construction of complete sets of chains)

Let G be a PSL graph in the plane (x,y) with vertex set $\{v_1,v_2,\ldots,v_n\}$. Each vertex v_i is given by a pair of coordinates (x_i,y_i) and the graph G is described by its edge-lists, i.e., by a collection $\{L_1,L_2,\ldots,L_n\}$, where L_j, the edge-list for vertex v_j, is the set of indices of vertices to which v_j is connected. The set of chains we shall construct will be monotone with respect to the y-axis.

We initially arrange the original data structure by sorting the y-coordinates of the vertices and renaming the indices so that $y_1 \geq y_2 \geq \ldots \geq y_n$; this processing requires O(n log n) operations. Next we construct a two-dimensional list, called the *standard representation of G*, representing a modified connection matrix A of G, where A[i,j] = 1 if and only if there is an edge (v_i,v_j) in G and either $y_i > y_j$ or, if $y_i = y_j$, then $i < j$. Notice that the row-lists thus obtained describe a directed graph G', obtained from G by assigning to each edge e of G a direction so that the projection of e on the y-axis runs opposite to this axis. For ease of reference, we shall say that an edge of G is "incoming" or "outgoing" depending upon its direction in G'. The construction of the standard representation of G is a simple operation. We scan the original data structure

and process each edge (v_i,v_j) as follows: If either $y_i > y_j$ or $y_i = y_j$ and $i < j$, we assign the edge to the row-list of v_i and to the column-list of v_j; otherwise, we ignore the edge. It is clear that the running time of this construction is proportional to the number of edges, i.e., since G is planar, it is O(n). Finally, the edges of each row- and column-list are sorted according to counterclockwise ascending slope: this sorting operation requires time at most O(n log n). The two-dimensional list data structure which gives the standard representation of the graph of Figure 5a is illustrated in Figure 5b, along with the formats of the vertex and edge nodes. The denominations of the fields of the two formats are now explained. For a vertex v_i, PRED[v_i] and SUCC[v_i] point to the locations

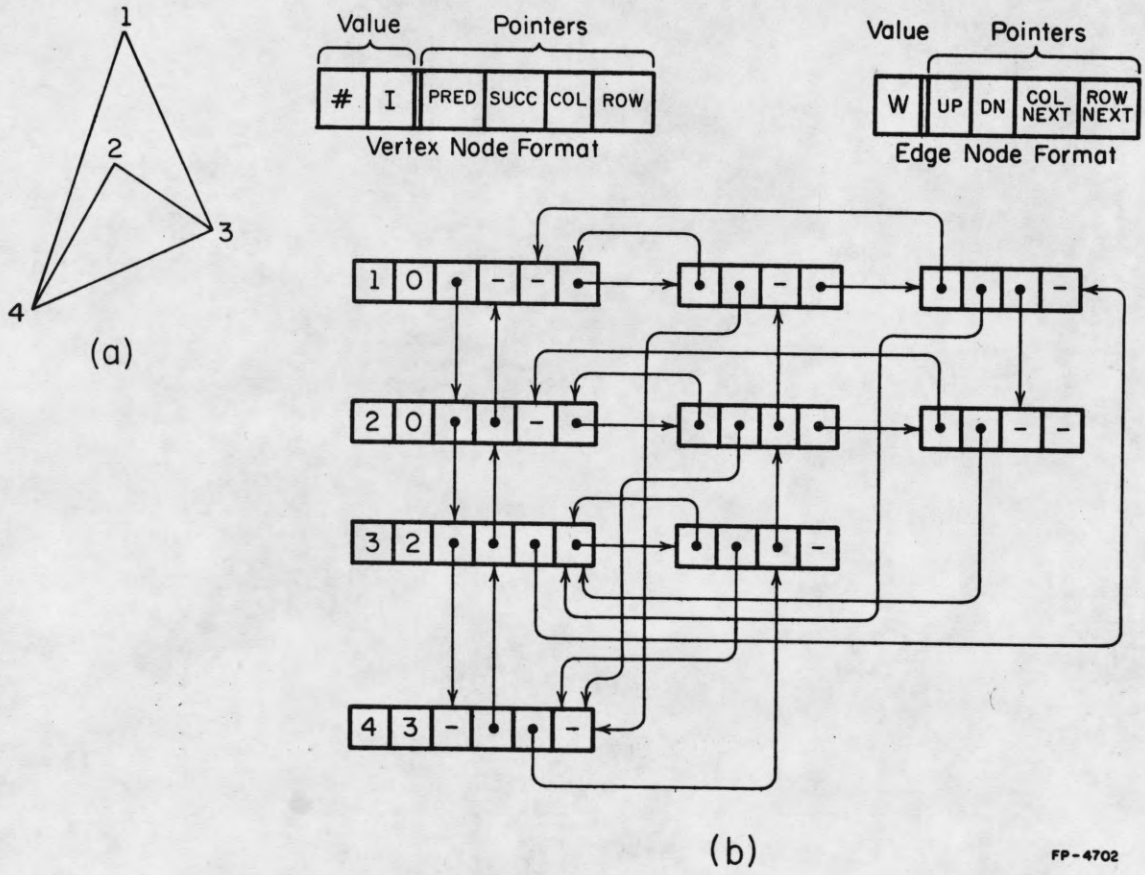


Figure 5. A PSL Graph and the corresponding data structure.

of the nodes v_{i-1} and v_{i+1}, respectively, while COL[v_i] and ROW[v_i] point to the first members of the column- and row-lists, respectively; as to the value fields, #[v_i] = i and I[v_i], the column degree of v_i, is the number of incoming edges of v_i, i.e., the cardinality of its column-list. For an edge $e = (v_i, v_j)$, with $y_i > y_j$, UP[e] and DN[e] point to v_i and v_j, respectively, while COLNEXT[e] and ROWNEXT[e] point to the next members of the column- and row-lists, respectively; W[e], the weight of e, will be used to denote the number of chains containing the edge e.

Definition 5. A vertex v_i of a PSL graph G is said to be *regular* if both its row- and column-lists are nonempty. Moreover, G is said to be *regular* if each v_i is regular for $1 < i < n$ (i.e., with the exception of the two terminal vertices v_1 and v_n).

We shall now show that a regular PSL graph admits of a complete set of chains. The chain construction procedure is based on the observation that, assigning to chains the conventional direction from v_1 to v_n, for each vertex v_i there are as many incoming as there are outgoing chains. Thus we must assign weight to the edges so that, for every nonterminal vertex v_i, the sum $W_{in}(v_i)$ of the weights of incoming edges equals the sum $W_{out}(v_i)$ of the weights of outgoing edges.

The outlined balancing of the weights can be done in two passes over the previously described data structure. In the first pass we proceed from v_n to v_1 and assign the edge weights so that, for each nonterminal v_i, $W_{in}(v_i) \geq W_{out}(v_i)$. The second pass, in turn, proceeds from v_1 to v_n and modifies the weights so that, $W_{in}(v_i) \leq W_{out}(v_i)$, for every nonterminal v_i, thereby achieving the desired balancing.

During the latter pass we can also explicitly construct the chain data structure. It may appear at first glance that more than O(n) locations are needed to store the chains, since there are O(n) chains and there is no obvious way to bound the number of edges each of them may contain. However, we can obtain a very compact representation of this set using the fact that the chain set is to be used in a binary search. Indeed, it is well-known that a binary search algorithm on a totally ordered set S induces a natural hierarchy on S as follows: each member of S is assigned to a vertex of a binary tree T(S) through a mapping $\tau: S \to T(S)$, and an actual search operation corresponds to traversing a path from the root to a leaf of T(S). Given two chains C_i and C_j in C we shall say that C_i is *higher than* C_j, and denote it by $C_i > C_j$, if the path from the root of T(C) to $\tau(C_j)$ contains $\tau(C_i)$. Assume now that $C_i > C_j$ and that C_i and C_j share an edge e. Since, when performing binary search, the discrimination of a point P against C_j is preceded by the discrimination of P against C_i, then clearly edge e may be assigned to C_i only. Thus, as a general rule, an edge of the graph G will be assigned to the highest chain of the hierarchy which contains it.

In the interest of simplicity and with a negligible loss of search efficiency we shall adopt a *standard hierarchy* for the set $C = \{C_1, C_2, \ldots, C_r\}$ which is independent of the number of chains, as follows: for $i \neq j$, $C_i > C_j \Leftrightarrow j \in [i - 2^{p_i} + 1, i + 2^{p_i} - 1]$, where p_i is the largest power of 2 which is a factor of the integer i. For example for n = 20 we have the hierarchy illustrated in Figure 6.

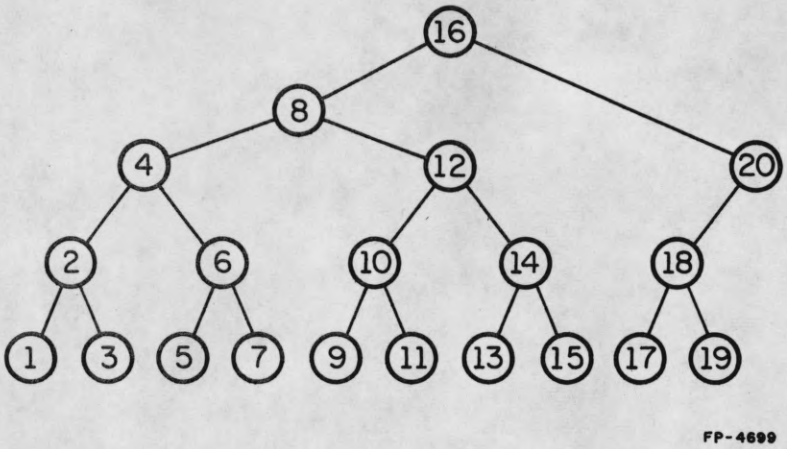


Figure 6. Hierarchy for n = 20.

The data structure describing the set C of chains will be a list representation of the tree $T(C)$, linking the chain nodes; in turn, the node for chain C_i is the header of a list $L(C_i)$, which gives the edge sequence of C_i. For reasons to become apparent later (Section 4) each edge $e = (v_i, v_j)$ is labeled with two pairs of integers, $(I_{min}[e], I_{max}[e])$ and $(L[e], R[e])$. The integers $I_{min}[e]$ and $I_{max}[e]$ are respectively the minimum and the maximum value of j such that $e \in C_j$. The integers L[e] and R[e] are the labels of the plane regions respectively to the left and to the right of the edge e directed so that its y-projection is concurrent with the y-axis. The formats of the chain and edge nodes are illustrated in Figure 7.

We can now give a formal description of the algorithms which accomplish the chain construction.

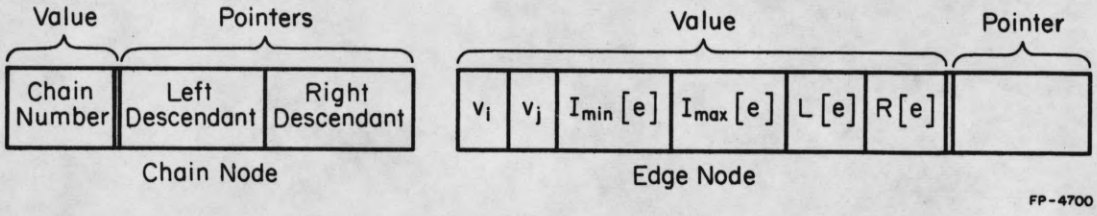


Figure 7. Formats of chain and edge nodes in the chain data structure.

Chain Construction Algorithm

First Pass

This pass accepts a PSL graph G given by its standard representation and assigns the weight of each edge so that for each nonterminal vertex v_i we have $W_{in}(v_i) \geq W_{out}(v_i)$.

1. For each edge e in column-list of v_j, for j = 1,...,n, set W[e] ← 1.
2. Set i ← n - 1.
3. While 1 < i < n do:
   4. Set W_{out} ← sum of weights in row-list of v_i.
   5. Set d ← first edge in column-list of v_i.
   6. If $W_{out} > I[v_i]$, set $W[d] \leftarrow W_{out} - I[v_i] + 1$.
   7. Set i ← i - 1.
8. Halt.

Comment: All edges are assigned weight 1 except for the first member of each column-list, which is assigned the value $W_{out}(v_i) - W_{in}(v_i) + 1$. In this manner, the condition $W_{in}(v_i) \geq W_{out}(v_i)$ is achieved for each nonterminal vertex. Computationally, the running time is O(n), since each edge of the graph is scanned a fixed number of times (Steps 1, 4, and 5) and there are O(n) edges.

Second Pass

This pass accepts the output of the first pass and performs the following functions: (1) it balances the weight of each nonterminal vertex of G; (2) it constructs a complete set of chains for G; (3) it labels the regions of the subdivision induced by G. The assignment of an edge to a chain makes use of a special arithmetic function, called PREDECESSOR(k,ℓ), for two integers k and ℓ with $k \leq \ell$, which determined the common predecessor of both k and ℓ which is the lowest in the standard hierarchy; this function, which will not be described in detail, can operate directly on the binary representations of the integers k and ℓ and is easily seen to require a number of operations proportional to log n.

(Comment: Steps 1-3 initialize for the special vertex v_1.)

1. Set $A \leftarrow 1$, $r \leftarrow 2$, $L \leftarrow 1$, $R \leftarrow 1$.
2. Set $W_{in} \leftarrow$ sum of weights in row-list of v_1.
3. Set $i \leftarrow 1$.
4. While $1 \leq i < n$ do:

5. Set $W_{out} \leftarrow$ sum of weights in row-list of v_i.
6. Set $a \leftarrow W_{in} - W_{out}$.
7. Set $e \leftarrow$ first edge in row-list of v_i.
8. While there are edges in row-list of v_i do: (Comment: Steps 9-12 assign an edge to a chain; Steps 13-14 label the regions separated by an edge.)
    9. Set $I_{min}[e] \leftarrow A$
    10. Set $I_{max}[e] \leftarrow A + a + W[e] - 1$.
    11. Compute $c \leftarrow$ PREDECESSOR($I_{min}[e]$,$I_{max}[e]$).
    12. Assign e to list of chain c.
    13. If e is first edge of row-list, set $L[e] \leftarrow L$; else set $L[e] \leftarrow r$ and $r \leftarrow r + 1$.
    14. If e is last edge of row-list, set $R[e] \leftarrow R$; else set $R[e] \leftarrow r$.
    15. Set $a \leftarrow 0$, $A \leftarrow I_{max}[e] + 1$.
    16. Set $e \leftarrow$ next edge in row-list of v_i.
17. Set $i \leftarrow i + 1$ and $r \leftarrow r - 1$.
18. Set $W_{in} \leftarrow$ sum of weights in column-list of v_i.
19. Set $d_1 \leftarrow$ first edge in column-list of v_i.
20. Set $R \leftarrow R[d_1]$.
21. Set $d_2 \leftarrow$ last edge in column-list of v_i.
22. Let $L \leftarrow L[d_2]$, $A \leftarrow I_{min}[d_2]$.
23. Halt.

Comment: The second pass contains two nested loops. The primary loop (Steps 5-22) scans the vertices, the secondary loop (Steps 9-16) scans the edges in the row-list of a vertex. For each vertex v_i, the weight balance is achieved by modifying, if necessary, the weight of the first member of the row-list (Steps 5, 6, 7, 9, and 10). Each edge is processed only once, but the function PREDECESSOR requires time O(log n); thus the number of operations required is O(n log n).

In summary, since the initial sorting, the first balancing pass and the second balancing pass run in times O(n log n), O(n), and O(n log n), respectively, the entire preprocessing tasks can be accomplished in time O(n log n).

4. Point location algorithm (search).

We shall now describe in detail the point location algorithm which we have sketched in Section 2. The algorithm accepts the chain data structure described in Section 3 and a test point $P \equiv (x,y)$ and determines the planar region R to which P belongs in at most $O((\log n)^2)$ steps. The integer m denotes the number of chains. The search is characterized by a pair of integers (ℓ,r), with $\ell < r$, to denote that the point P is comprised between chains C_ℓ and C_r; the algorithm terminates when $r - \ell = 1$ and the region is determined from the data associated with the edges.

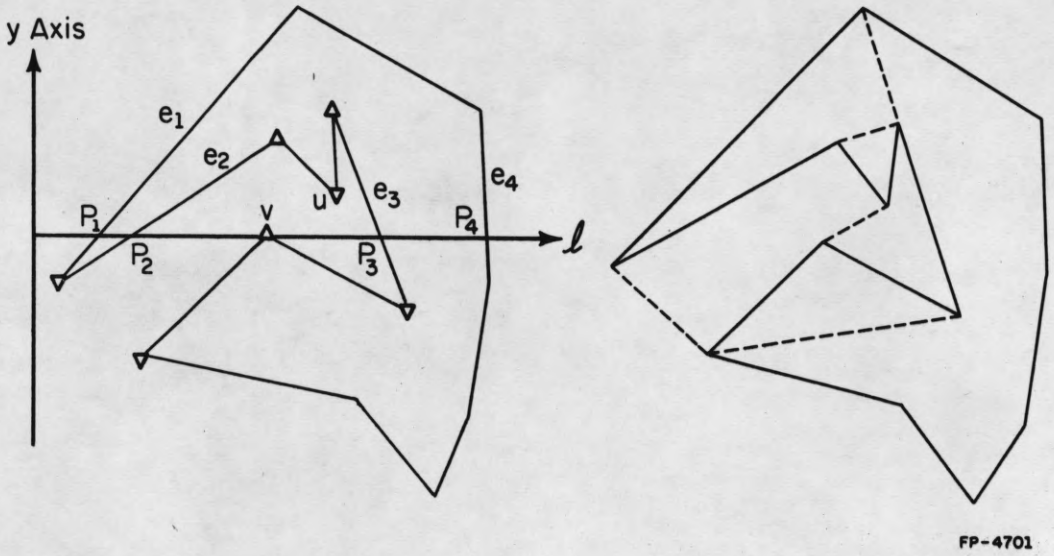
1. If $y \geq y_1$ or $y \leq y_n$, then set $R \leftarrow 1$ (Comment: P belongs to the infinite region of the plane) and halt.
2. Set $g \leftarrow$ root of $T(C)$, $\ell \leftarrow 0$, $r \leftarrow m + 1$.
3. Set $j \leftarrow$ index of the chain associated with g.
4. If $\ell \geq j$, set $g \leftarrow$ right descendant of g and goto Step 3.
5. If $r \leq j$, set $g \leftarrow$ left descendant of g and goto Step 3.
6. In the edge-list headed by g, find an edge (v_i, v_k) such that $y_k \leq y \leq y_i$ and set $e \leftarrow (v_i, v_k)$ (Comment: This step is the binary search of the edge against which P is to be discriminated.)
7. If P lies to the right of e, set $\ell \leftarrow I_{max}[e]$ and $R \leftarrow R[e]$; else set $r \leftarrow I_{min}[e]$ and $R \leftarrow L[e]$. (Comment: this step discriminates P against the edge e and makes a tentative region assignment; this assignment is final if $r - \ell = 1$, as indicated by Step 8.)
8. If $r - \ell = 1$, halt; else goto Step 3.

5. Regularization of an arbitrary planar polygon.

The preprocessing procedure described in Section 3 is applicable to regular PSL graphs. However, a PSL graph is not regular in general, since it may contain regions delimited by arbitrary polygons. In this section, we shall illustrate a procedure which transforms an arbitrary polygon with n vertices into a regular PSL graph and runs in time $O(n \log n)$. This procedure will be referred to as *regularization* of a planar polygon. With the aid of this additional procedure, our point-location algorithm and its

associated preprocessing are applicable to arbitrary planar subdivisions.

We shall briefly illustrate the idea of the regularization procedure. In a simple planar polygon not all vertices are regular with respect to a selected line (the y-axis): for example, in Figure 8a, vertices marked with "∇" require an outgoing edge and vertices marked with "Δ" require an incoming edge. Consider now in Figure 8a the horizontal line ℓ (orthogonal to the y-axis) through a vertex v requiring, for example, an incoming edge. This line ℓ intersects a set of edges $\{e_1,e_2,e_3,e_4\}$ in points P_1,P_2,P_3, and P_4, respectively. Therefore vertex v falls in a unique part of the partition of ℓ determined by these points (in our example, in segment P_2P_3). Associated with the pair of edges (e_2,e_3) there is a vertex of minimum ordinate (in our case, vertex u): thus we can introduce an auxiliary edge (v,u) which is guaranteed not to cross any edge of the polygon and which satisfies the requirement of v for an incoming edge. Thus the regularization procedure will consist of two passes: in the first pass (descending pass) we scan the vertices in order of decreasing y-coordinate and satisfy the requirements of vertices in need of incoming edges; in the second pass (ascending pass) the reverse process occurs. Each stage in the execution of the algorithm is characterized by an ordered set of edges (a sequence of edges). For every newly reached vertex v, the current edge sequence is updated as follows: if v is regular, a new edge replaces a previous edge, if v requires an incoming edge (an outgoing edge), two new edges are inserted (are deleted). Therefore, the natural data



(a) (b)

Figure 8. A simple polygon and its regularized version.

structure for the edge sequence is a list, realized by a balanced tree (an AVL tree, see [3], Sec. 6.2.3.).

We shall now give a detailed description of the regularization algorithm and, for obvious reasons, we shall confine ourselves to the descending pass. The polygon G is originally given as a sequence of vertices $u_1,u_2,\ldots,u_n$, corresponding to the edge sequence $(u_1,u_2),(u_2,u_3),\ldots,(u_n,u_1)$. Preliminarily, we construct the standard representation of G (see Section 3), that is, we sort the vertices in order of decreasing y-coordinate, we relabel them as $v_1,\ldots,v_n$ so that $y(v_i) \geq \ldots \geq y(v_n)$ and associate with each vertex the row- and column-lists of its outgoing and incoming edges, respectively. In our case, the total number of edges in

the row- and column-lists is equal to 2, since G is a closed polygon.

The data structure describing the current edge sequence is an AVL tree E. Specifically, E is a sequence $[e_0,w_0],[e_1,w_1],\ldots,[e_k,w_k]$, for some integer K, where e_j is an edge and w_j is a minimum ordinate vertex between e_j and e_{j+1}. The algorithm described below refers to a simple polygon G surrounded by the infinite region of the plane; there are some obvious modifications when G borders other simple polygons.

Descending Pass

This algorithm accepts the standard representation of G and creates an auxiliary list of edges satisfying the requirements of vertices in need of an *incoming* edge. We assume that standard operations: INSERT, DELETE, REPLACE are available for the AVL tree E. Use is also made of an artificial edge e_0 corresponding to the line $x = -\infty$. If a vertex v has two outgoing edges, the latter are denoted as e'(v) and e"(v), with e'(v) forming a counterclockwise convex angle with e"(v). The integer I(v) denotes the number of incoming edges of a vertex v.

1. Set $w_0 \leftarrow v_1, e_1 \leftarrow e'(v_1), e_2 \leftarrow e''(v_1), w_1 \leftarrow v_1, w_2 \leftarrow v_1$ and INSERT $[e_0,w_0],[e_1,w_1]$, and $[e_2,w_2]$ into E.
2. Set $i \leftarrow 2$.
3. While $i < n$ do:

4. Set $j \leftarrow$ smallest integer so that v_i is not to the left of e_j.

5. If $I(v_i) = 2$, set $w_{j-1} \leftarrow v_i, w_{j+2} \leftarrow v_i$; then DELETE $[e_j, w_j]$ and $[e_{j+1}, w_{j+1}]$ from E. (Comment: This case occurs when v_i requires an outgoing edge).

6. If $I(v_i) = 1$, set $w_j \leftarrow v_i, w_{j-1} \leftarrow v_i$, $e_j \leftarrow$ outgoing edge of v_i. (Comment: v_i is a regular vertex).

7. If $I(v_i) = 0$, INSERT$[e'(v_i), v_i]$ and $[e''(v_i), v_i]$ between $[e_j, w_j]$ and $[e_{j+1}, w_{j+1}]$ in E. Add the edge (v_i, w_j) to the auxiliary list and set $w_j \leftarrow v_i$. (Comment: In this case v_i requires an incoming edge.)

8. Set $i \leftarrow i + 1$.

9. Halt.

The ascending pass is the same algorithm, once the signs of the ordinates $y(v_1), \ldots, y(v_n)$ have been changed. At the completion of the two passes the edges in the auxiliary list must be added to the standard representation and the resulting PSL graph is regular with respect to the y-axis.

We now analyze the regularization algorithm. The initial sorting pass of the ordinates requires O(n log n) operations. Step 4 involves traversing a path in the AVL tree E and requires at most O(log n) operations: thus the total amount of work involved in Step 4 is O(n log n). Each update, deletion or insertion (see Steps 5, 6, and 7) requires a computational work at most O(log n); however, there is a fixed maximum number of such

operations per vertex, whence the total number of operations required by these steps is $O(n \log n)$. We conclude that the algorithm for regularizing a simple polygon has running time at most $O(n \log n)$.

In Figure 8b, we illustrate the result of the regularization procedure applied to the polygon in Figure 8a. New regions have been created, with no difficulty however for the effective solution of the point location problem.

As a final observation, we consider the case in which we must regularize a given general PSL graph G with n vertices and N edges. For every region of G with s vertices, the regularization algorithm runs in time $O(s \log s)$. Assuming that the subdivision induced by G consists of regions $R_1, R_2, \ldots, R_J$ with respective number of vertices $s_1, s_2, \ldots, s_J$, since $s_1 + \ldots + s_J \leq 2N$, we conclude $s_1 \log s_1 + \ldots + s_J \log s_J \leq 2N \log 2N$; but N is $O(n)$, whence the regularization of a general PSL graph with n vertices can also be accomplished in time $O(n \log n)$.

6. Applications

As we indicated in Section 1, the point location problem occurs as a subproblem in a number of important applications. Thus our fast point location algorithm provides interesting solutions for other problems. In particular, we shall explicitly consider:

(i) spatial convex inclusion;

(ii) inclusion in an arbitrary planar polygon;

(iii) location in the planar subdivision determined by n straight lines.

6.1. Spatial convex inclusion--Let S be a convex polyhedron with n vertices in 3-space (x,y,z) and let P be a test point; we must determine whether P is inside or outside S. We shall proceed as follows. Let P_1 be a vertex of S with largest z coordinate. Clearly, the projection of S from P_1 onto the plane (x,y) is a planar graph S'. Since the regions induced by S' are projections of faces of S, and the latter are convex, each region of the planar subdivision is convex. It follows then that S is regular, since, the nonterminal vertices on the perimeter are obviously regular and each internal vertex has at least three edges forming convex angles. Thus, with the previous point location algorithm, we can identify a unique region R' of S' which contains a point P', which is the projection onto (x,y) from P_1 of a given test point P in 3-space. Since R' of S' corresponds to a face R of S, the testing for convex inclusion of P in S is accomplished by determining on which side of R the point P lies. Since the operation of constructing the data structure for S' from the data structure of S runs in time O(n), we conclude that, with a preprocessing time at most O(n log n), spatial convex inclusion can be tested in at most $O((\log n)^2)$ steps.

6.2. Inclusion in an arbitrary planar polygon--Let G be a planar polygon with n vertices and let P be a test point; we must determine whether P is inside or outside G. Shamos [2] provides an algorithm for solving this problem based on the "slab" notion, which has search time

$O(\log n)$, with a preprocessing time and storage both $O(n^2)$. By contrast, our algorithm has search time $O(\log^2 n)$, preprocessing time $O(n \log n)$ and storage $O(n)$. We initially regularize the polygon G and transform it into a regular PSL graph G': the regions of the resulting subdivision are partitioned depending on whether they are inside or outside G. Thus the point location algorithm is applied to G' and the problem is solved.

6.3. Location in straight line planar subdivision--A planar subdivision is induced by n straight lines in general position. In this case the regions (finite or infinite) are all convex, and therefore the planar graph on $O(n^2)$ vertices is regular. Some minor modification of the algorithms are required and it is straightforward to show that $O(\log^2 n)$ search time and $O(n^2)$ storage can be attained, as conjectured by Shamos [2].

7. References

[1] M. Ketelsen, Triangular tile identification, CS 389 course project, Department of Computer Sci., University of Illinois, Urbana, Illinois Dec. 1973.

[2] M. J. Shamos, "Problems in Computational Geometry," Department of Computer Sci., Yale University, New Haven, Connecticut, May 1975.

[3] D. E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison-Wesley, Reading, Mass., 1973.