SIAM J. COMPUT.
Vol. 21, No. 2, pp. 267–280, April 1992

005

EFFICIENT POINT LOCATION IN A CONVEX SPATIAL CELL-COMPLEX*

FRANCO P. PREPARATA† AND ROBERTO TAMASSIA‡

Abstract. In this paper a new approach is proposed to point-location in a three-dimensional cell-complex $\mathcal{P}$, which may be viewed as a nontrivial generalization of a corresponding two-dimensional technique due to Sarnak and Tarjan. Specifically, in a space-sweep of $\mathcal{P}$, the intersections of the sweep-plane with $\mathcal{P}$ occurring in a given slab, i.e., between two consecutive vertices, are topologically conformal planar subdivisions. If the sweep direction is viewed as time, the descriptions of the various slabs are distinct "versions" of a two-dimensional point-location data structure, dynamically updated each time a vertex is swept. Combining the persistence-addition technique of Driscoll, Sarnak, Sleator, and Tarjan [*J. Comput. System. Sci.*, 38 (1989), pp. 86–124] with the recently discovered dynamic structure for planar point-location in monotone subdivisions, a method with query time $O(\log^2 N)$ and space $O(N \log^2 N)$ for point-location in a convex cell-complex with N facets is obtained.

Key words. point location, convex cell complex, computational geometry, analysis of algorithms

AMS(MOS) subject classifications. 68U05, 68Q25, 68P05, 68P10

1. Introduction.

Point-location in three-dimensional space, called spatial point-location, is a natural generalization of the well-known planar point location (see, e.g., [6], [11], [12]). The space is partitioned into polyhedral regions, called *cells*, and the resulting subdivision is frequently referred to as a *cell-complex*. The problem is so stated: Given a cell-complex $\mathcal{P}$ and a query point q, determine the cell of $\mathcal{P}$ containing q.

Unlike its two-dimensional counterpart, spatial point-location has not yet received extensive attention. In all reported research, the cell-complex satisfies some restrictive condition. Cole's Similar Lists method [4] has been applied to the cell complex determined by an arrangement of n planes, and yields query time $O(\log n)$ but uses space $O(n^4/\log n)$. The space bound has been recently improved by Chazelle and Friedman [2] to $O(n^3)$, by a modification of the random sampling technique of Clarkson [3]. Chazelle's earlier Canal Tree technique [1] trades space for query time, and achieves space $O(n^3)$ and query time $O(\log^2 n)$. The same technique can be applied to a convex cell-complex with N facets, yielding query time $O(\log^2 N)$ and space $O(N)$; the cell-complex, however, is subject to the restrictive condition that the vertical dominance relation on the cells be acyclic.

Two recent results can be profitably combined to provide a new attractive approach to spatial point-location: the persistence-addition technique of Driscoll, Sarnak, Sleator, and Tarjan [5] and the dynamic planar point-location technique of

* Received by the editors December 26, 1989; accepted for publication (in revised form) March 26, 1991. An extended abstract of this paper was presented at the 1989 Workshop on Algorithms and Data Structures, Ottawa, Canada, August 1989.

† Department of Computer Science, Brown University, Providence, Rhode Island 02912–1910. The research of this author was supported in part by National Science Foundation grant CCR-89-06419. This research was initiated while the author was with the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.

‡ Department of Computer Science, Brown University, Providence, Rhode Island 02912–1910. The research of this author was supported in part by National Science Foundation grant CCR-9007851, by the United States Army Research Office grant DAAL03-91-G-0035, by Office of Naval Research and Defense Advanced Research Projects Agency contract N00014-91-J-4052 and ARPA order 8225, and by Cadre Technologies, Inc.

Preparata and Tamassia [13]. Our result is a new method with query time $O(\log^2 N)$ and space $O(N \log^2 N)$ for a convex spatial cell-complex with N facets.

The general methodology of [5] is designed to add *persistence* to a dynamic linked data structure (referred to as *ephemeral*). In an abstract setting, the ephemeral data structure supports accesses and updates, and each update produces a new version of the structure. Thus the history of the data structure is the sequence of its versions, terminating in the *current* version. Persistence is the ability to access past versions, and it is *full* or *partial* depending upon whether updates are also permitted in past versions or not. In [5] a systematic and efficient technique is presented to transform an ephemeral linked data structure into a persistent one, provided that the ephemeral structure satisfies the weak condition that its nodes have bounded in-degree. This methodology was applied in a companion paper [14] to planar point-location, by viewing one dimension (e.g., the y-direction) as "time," so that the planar subdivision is swept over in time by a horizontal line. The dictionary of the sequence of intersected edges is the ephemeral data structure, a new version of which is created each time a vertex of the subdivision is reached in the sweep. The persistent version of the dictionary becomes therefore the data structure for planar point location; in other words, *static* two-dimensional point-location is modeled by a *partially persistent* one-dimensional dictionary process.

The last observation is the clue to higher-dimensional generalizations. Until recently, however, the obstacle to a three-dimensional generalization was the lack of a suitable (ephemeral) dynamic planar point-location structure. The recent discovery [13] of an efficient such structure for monotone planar subdivisions provides the missing component that, combined with the technique for the addition of persistence, yields the method for point location in a spatial convex cell complex discussed in this paper.

This paper is organized as follows. In §2 we review the separator-tree structure that is the basis of the two-dimensional point-location primitive, as well as the essentials of the persistence-addition technique. Section 3 illustrates the modifications required to adapt the ephemeral data structure to the projected spatial point-location method, described in its most general version in §4. Finally, in §5 we describe the simplifications obtainable when exploiting the specific nature of particular cell-complexes such as Voronoi diagrams and those determined by arrangements of planes.

2. Preliminaries.

2.1. Planar point location. Let $\mathcal{P}$ be a three-dimensional cell complex, i.e., a partition of the three-dimensional space into polyhedra. We say that $\mathcal{P}$ is *convex* if either each of its cells is a convex polyhedron, or at most one is nonconvex (in this case, the nonconvex cell is the complement of a convex polyhedron). Any planar section of $\mathcal{P}$ is a planar convex subdivision (with the analogous exception of at most one nonconvex region), which is a special case of a monotone subdivision [9]. We shall now review the essentials of the dynamic point-location method of Preparata and Tamassia [13] for planar monotone subdivisions, in order to bring into sharper focus the requirements for the addition of persistence.

An *edge* is either a segment or a half-line in the plane (for simplicity, no edge is horizontal). A (polygonal) *chain* is a sequence of edges, so that two consecutive edges share a terminus; it is *monotone* if each horizontal line intersects it in at most a single point. A polygon is *monotone* if its boundary is partitionable into two monotone chains. A subdivision $\mathcal{R}$ is *monotone* if each of its regions is a monotone polygon.

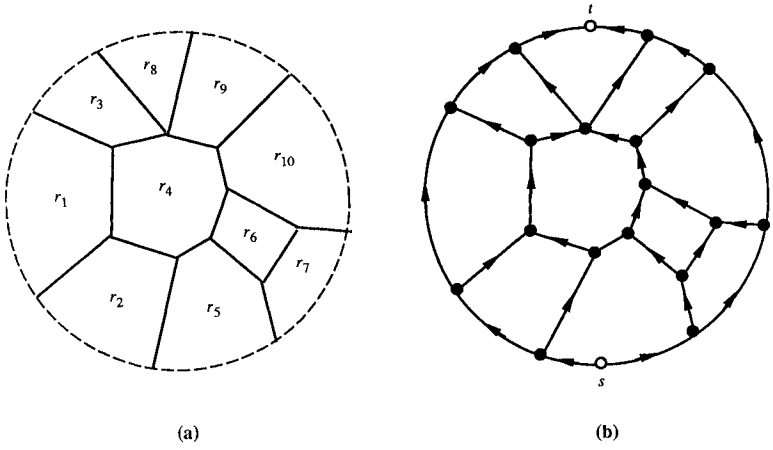

FIG. 1. *Dynamic point location technique:* (a) *monotone subdivision* $\mathcal{R}$ (*the dashed circle represents the line at infinity*); (b) *planar st-graph* G *associated with* $\mathcal{R}$.

A *separating chain* (or, more succinctly, a *separator*) σ of a monotone subdivision $\mathcal{R}$ is a monotone chain consisting of edges of $\mathcal{R}$, whose extreme points are at infinity. Given two separators σ_1 and σ_2, σ_1 *is to the left* of σ_2 if any horizontal line intersects σ_1 not to the right of σ_2. A *complete family of separators* Σ of $\mathcal{R}$ is a sequence of separators $(\sigma_1, \sigma_2, \cdots, \sigma_t)$ such that σ_i is to the left of σ_{i+1} $(i = 1, \cdots, t-1)$, and every edge of $\mathcal{R}$ belongs to at least one separator $\sigma \in \Sigma$. For a given Σ, each $\sigma \in \Sigma$ is associated with a node (briefly referred to as "node σ") of a balanced binary tree $\mathcal{T}$ (called *separator tree*). An edge e of $\mathcal{R}$ belongs, in general, to a nonempty interval $(\sigma_i, \sigma_{i+1}, \cdots, \sigma_j)$ of separators. However, it is assigned to the least common ancestor σ_k of $(\sigma_i, \cdots, \sigma_j)$ in $\mathcal{T}$, and is called a *proper edge* of σ_k. Correspondingly, *proper* (σ_k) denotes the set of proper edges of σ_k, which are stored in a secondary component (a dictionary) appended to node σ_k of $\mathcal{T}$. Since each edge of $\mathcal{R}$ is stored exactly once, the above data structure has size $O(n)$, where n is the number of vertices of $\mathcal{R}$. It is well known [9] that point location in $\mathcal{R}$ corresponds to traversing a path from the root of $\mathcal{T}$, and performing at each node a point/chain discrimination by means of the secondary component. Thus point-location can be done in $O(\log^2 n)$ time.

The underlying topological structure of a monotone subdivision is a *planar st-graph*, i.e., a planar acyclic digraph with exactly one source (vertex without incoming edges), s, and exactly one sink (vertex without outgoing edges), t, embedded in the plane with vertices s and t on the boundary of the external face. Namely, we associate a monotone subdivision $\mathcal{R}$ with the planar st-graph G defined as follows (see Fig. 1):

1. The vertices of G are the vertices of $\mathcal{R}$, including vertices at infinity determined by edges that are rays, plus vertices s and t corresponding to the points at negative and positive infinity on the vertical axis, respectively.
2. The edges of G are the edges of $\mathcal{R}$, oriented from the lower to the upper endpoint, plus two chains from s to t that connect the vertices at infinity.

There are two main obstacles to the dynamization of $\mathcal{T}$ for an arbitrary family of separators Σ:

1. It may be difficult to rebuild the secondary components of the nodes participating in a rotation of the primary component. Indeed, in the worst case a rotation takes time $\Omega(n)$.
2. The deletion of an edge may break many separators of Σ, with $\Omega(n)$ time used for restructuring the complete family Σ.

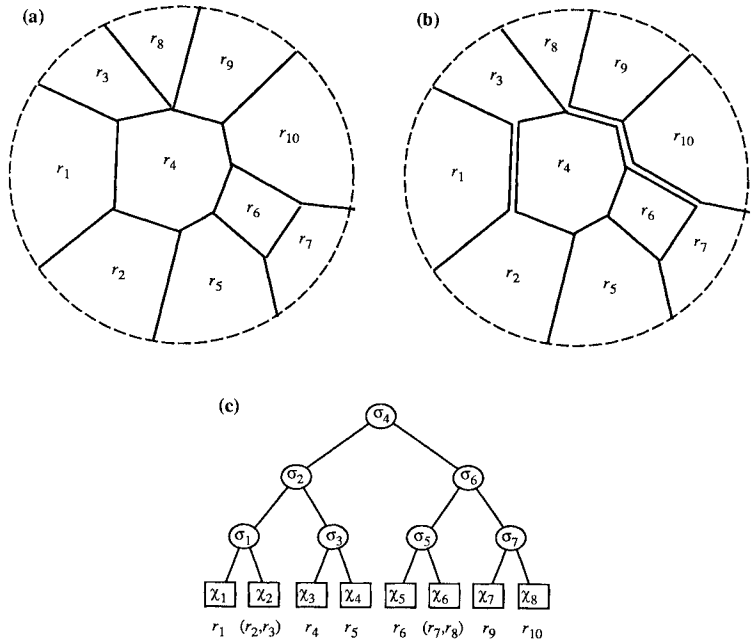

FIG. 2. *Dynamic point location technique:* (a) *monotone subdivision* $\mathcal{R}$; (b) *subdivision* $\mathcal{R}^*$ *and its chains of proper edges;* (c) *separator tree of* $\mathcal{R}^*$.

It has been shown in [13], however, that a modification (consisting of virtually duplicating a subset of edges) transforms $\mathcal{R}$ into a new subdivision $\mathcal{R}^*$ that admits a unique complete family of separators Σ^* such that, for any $\sigma \in \Sigma^*$, *proper* (σ) consists of a single chain. This permits an $O(\log n)$-time execution of a rotation in the corresponding separator tree $\mathcal{T}^*$ by standard operations on concatenable queues. Also, the restructuring of Σ^* after a deletion can be done with $O(\log n)$ split/splice operations on the chains of proper edges. The family Σ^* enjoying these properties is associated with the following total order "$\prec$" on the regions of $\mathcal{R}$ (see Fig. 2):

1. r_1 *is below* r_2 $(r_1 \uparrow r_2)$ if there is a monotone chain in $\mathcal{R}$ from the highest vertex of r_1 to the lowest vertex of r_2 (such chain corresponds to a directed path in digraph G);
2. r_1 *is to the left of* r_2 $(r_1 \rightarrow r_2)$ if there is sequence of regions $r_1 = r'_1, r'_2, \cdots, r'_s = r_2$ such that r'_i and r'_{i+1} share an edge and r'_i is to the left of it $(i = 1, \cdots, s-1)$. Partial orders "$\uparrow$" and "$\rightarrow$" are shown to be complementary, so that their union is the desired total order "$\prec$";
3. The members of any maximal sequence of regions $r_1 \prec r_2 \prec \cdots \prec r_s$, consecutive in "$\prec$" and such that $r_i \uparrow r_{i+1}$ $(i = 1, \cdots, s-1)$ are merged into a generalized region called *cluster* by duplicating the vertices and, when they exist, the edges of the (unique) monotone chain connecting the highest vertex of r_i to the lowest vertex of r_{i+1}, for $i = 1, \cdots, s-1$. This gives the subdivision $\mathcal{R}^*$. Σ^* is the unique complete family of separators for $\mathcal{R}^*$, each member of which separates contiguous clusters.

By using the correspondence between monotone subdivisions and planar *st*-graphs, the topological underpinning of the total order $\prec$ on the regions of $\mathcal{R}$ can be found in the theory of planar *st*-graphs and planar lattices [8], [10], [15].

We shall use, where appropriate, a string notation to illustrate the order $\prec$, so

that AB denotes that the partial subdivision described by the string A precedes in $\prec$ the one described by B.

The repertory of updates considered in [13] consists of: insertion/deletion of a vertex internal to a segment, and insertion/deletion of a chain of edges between two vertices, under the condition that monotonicity be preserved. For these specifications the following performance is achieved.

THEOREM 2.1 [13]. *Let $\mathcal{R}$ be a monotone subdivision with n vertices. There exists a dynamic point-location data structure with space $O(n)$ and query time $O(\log^2 n)$, which allows for insertion/deletion of a vertex in time $O(\log n)$ and insertion/deletion of a k-edge chain in time $O(\log^2 n + k)$.*

2.2. The technique for the addition of persistence. We now review the essentials of the technique of Driscoll, Sarnak, Sleator, and Tarjan [5] to add persistence to a dynamic linked data structure, which supports access and update operations. (A conventional dynamic data structure is called *ephemeral* if its instantiation preceding an update is not recoverable after the execution of the update.) The m updates are chronologically numbered from 1 to m, and the ith update generates version i of the ephemeral data structure. A *fully persistent* structure supports both accesses and updates to any of its versions; a *partially persistent* structure supports accesses to any of its versions, but updates only to its most recent (*current*) version. In this paper we are concerned exclusively with partial persistence.

The persistent structure embeds all versions of the ephemeral structure, so that access to any of them can be effectively simulated. Specifically, an access is the traversal of a path in an ephemeral version; the simulation of this access is possible if the persistent structure contains an image of this path. This is effectively accomplished by replacing each outgoing pointer of the ephemeral structure with a bundle of instantiations of that pointer in the persistent structure, each one time-stamped with the index of the update that established it, and ordered accordingly. Thus simulation of an access to version j is effectively accomplished by following at each step the appropriate pointer with maximum time-stamp not exceeding j. Arbitrarily large bundle size, however, negatively affects access time and storage. To avoid this shortcoming, Driscoll, Sarnak, Sleator, and Tarjan enforce a bound K on the number of pointers issuing from any given node and introduce the device of *limited node copying*, to be briefly outlined below.

A new copy v' of a node v is to be created at update j either when an information field of v is modified or a pointer field of v is modified, and, in either case, the corresponding update would cause the number of pointers issuing from v to exceed the bound K. In both cases, node v is declared *dead*, while the *live* node v' is time-stamped j; in addition, node v' must be correctly linked within the structure. This is facilitated if each pointer from live node to live node is paired with a reverse pointer (which need not be time-stamped). Specifically, we have the following actions, which implement copying of node v following an update of field f at time j:

1. All fields of v, but f (which receives the updated value), are copied into v'. Among these, for each pointer field to a live node w, the corresponding reverse pointer in w is switched from v to v'.
2. Each reverse pointer from v to some (live) w is suppressed, and (if the time-stamp of the direct pointer from w to v is less than j) a new pointer time-stamped j is established from w to v' (along with the corresponding reverse pointer).

The actions of step 2 deserve further discussion. First, to ensure that copying of

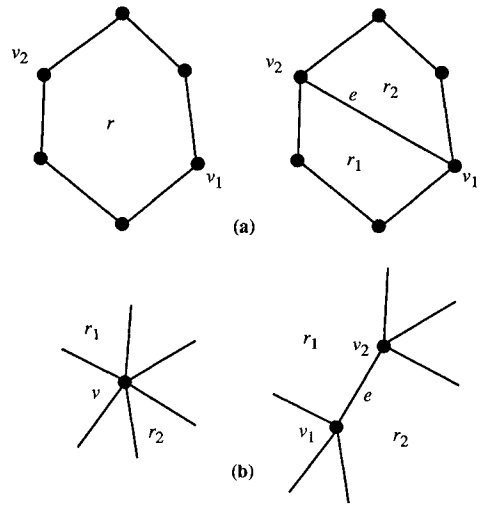

FIG. 3. (a) *Operations Insert and Delete.* (b) *Operations Expand and Contract.*

a node be accomplished in $O(1)$ time, it is sufficient to require that the in-degree of a node in the ephemeral structure be bounded by a constant. (Under this condition, only a bounded number of reverse pointers must ever be suppressed.) Second, step 2 introduces additional pointers outgoing from an existing node w and may cause the total number of outgoing pointers of w to exceed bound K. Consequently, node copying may "bubble up" the linked data structure. However, amortized over a sequence of updates, Driscoll, Sarnak, Sleator, and Tarjan [5] show that there are only $O(1)$ nodes copied per update.

This is summarized as follows.

THEOREM 2.2 [5]. *If an ephemeral data structure has nodes of constant bounded in-degree, then the structure can be made partially persistent at an amortized space cost of $O(1)$ per update step and a constant factor in the amortized time per operation.*

Considerable simplifications arise when the ephemeral data structure is a red-black tree [7]. In this case the node in-degree is at most 1, and rebalancing after an insertion/deletion requires $O(1)$ rotations.

3. The ephemeral planar point location structure. In this section we describe algorithms and data structures designed to support the following repertory of update operations on a monotone subdivision $\mathcal{R}$ (see Fig. 3).

Insert$(e, r, v_1, v_2; r_1, r_2)$: Insert edge e between vertices v_1 and v_2 inside region r, which is decomposed into regions r_1 and r_2 to the left and right of e, respectively.

Delete$(e, v_1, v_2, r_1, r_2; r)$: Remove edge e between vertices v_1 and v_2 and merge into region r the two regions r_1 and r_2 formerly to the left and right of e, respectively.

Expand$(e, v, r_1, r_2; v_1, v_2)$: Expand vertex v into vertices v_1 and v_2 connected by edge e, which has regions r_1 and r_2 to its left and right, respectively.

Contract$(e, r_1, r_2, v_1, v_2; v)$: Contract edge e between vertices v_1 and v_2 into vertex v. Regions r_1 and r_2 are those formerly to the left and right of e, respectively.

The present techniques represent modifications of those described in [13], where only operations *Insert* and *Delete* are supported, and nodes of the data structures do

not have the now required bounded in-degree. Note that, in terms of the underlying topological structure of a monotone subdivision discussed in §2.1, operations (*Insert*, *Expand*) and (*Delete*, *Contract*) form dual pairs, the first term acting on the primal graph (coincident with the subdivision), the second term acting on its dual graph. The data structure, however, will not explicitly reflect this duality; indeed, the vertices of $\mathcal{R}$ play a central role, since the geometry of $\mathcal{R}$ is determined by the coordinates of its vertices.

In the original data structure [13], a vertex was pointed to by all of its incident edges and by all regions having it as the lowest/highest point. Since a vertex can receive $\Omega(n)$ such pointers, the original representation both prevents an efficient implementation of operations *Expand*/*Contract* and violates the bounded in-degree requirement. Therefore, we propose the following modified structure, which comprises a main constituent, called the "augmented separator tree," and an auxiliary constituent, called the "dictionary."

The *dictionary* has a record for each vertex, edge, and region of $\mathcal{R}$, as follows:

1. The record of vertex v stores the coordinates of v and pointers to two balanced binary search trees, denoted $in(v)$ and $out(v)$, and called the *incidence trees* of vertex v. The leaves of tree $in(v)$ represent the incoming edges of v (i.e., the edges (u, v) with $y(u) < y(v)$), and the internal nodes of $in(v)$ represent the regions of $\mathcal{R}$ whose highest vertex is v, where the in-order sequence of the nodes of $in(v)$ corresponds to the counterclockwise order of the corresponding edges and regions around v. Each node of $in(v)$ has a pointer to the record of the corresponding edge or region. Tree $out(v)$ is similarly defined with respect to the outgoing edges of v. Also, the record of vertex v contains a pointer to the representative node of v in the (secondary component of the) augmented separator tree, to be described below.
2. The record of edge $e = (u, v)$ stores pointers to the representative nodes of e in the trees $in(v)$ and $out(u)$. Also, the record of e stores pointers to the two representative nodes of e in the (secondary component of the) augmented separator tree, to be described below.
3. The record of region r stores pointers to the representative nodes of r in the trees $in(v)$ and $out(u)$, where u and v are the lowest and highest vertex of r, respectively. Also, the record of region r stores a pointer to the representative node of r in the (secondary component of the) augmented separator tree, to be described below.

The dictionary allows us to find the endpoints of an edge and the lowest and highest vertices of a region in $O(\log n)$ time. The dynamic maintenance of the dictionary can be performed in $O(\log n)$ time per update, and will not be explicitly discussed. Note that an *Expand* or a *Contract* operation corresponds to performing $O(1)$ split/splice operations in the incidence trees.

The *augmented separator tree*, denoted $\mathcal{T}^*$, has a primary and secondary component. The *primary component* is a balanced separator tree for $\mathcal{R}^*$, i.e., each of its leaves is associated with a region of $\mathcal{R}^*$ (a cluster of $\mathcal{R}$), and each of its internal nodes is associated with a separator of $\mathcal{R}^*$. The left-to-right order of the leaves of the primary component of $\mathcal{T}^*$ corresponds to the order $\prec$ on the regions of $\mathcal{R}^*$. The *secondary component* is a collection of balanced search trees, each associated with a node of $\mathcal{T}^*$:

1. Each internal node σ of $\mathcal{T}^*$ points to a balanced search tree $proper(\sigma)$ representing the chain $proper(\sigma)$, and to a balanced search tree $double(\sigma)$ asso-

ciated with the proper edges of σ that do not form a channel (called *double edges*). Whereas in [13] the detailed organization of the concatenable queues describing a separator was left unspecified, here the bounded in-degree requirement can be achieved as follows, without affecting the efficiency of point-location queries. Tree *proper*(σ) is organized so that each leaf represents an edge, and each internal node represents a vertex, where the in-order sequence of the nodes corresponds to traversing the chain of proper edges of σ from bottom to top. Each node of *proper*(σ) points to the record of the corresponding vertex or edge in the dictionary. Tree *double*(σ) is organized so that each node represents an edge, and the in-order sequence of nodes corresponds to the bottom-to-top subsequence of the double edges of *proper*(σ). Note that each edge e of $\mathcal{R}$ has exactly two representative nodes in the secondary components of the augmented separator tree $\mathcal{T}^*$.

2. Each leaf χ of $\mathcal{T}^*$ (i.e., the leaf representing cluster χ) points to a balanced search tree *regions*(χ) associated with the sequence of regions that form cluster χ, in bottom-to-top order.

The following theorem summarizes the properties of the ephemeral structure.

THEOREM 3.1. *Let $\mathcal{R}$ be a monotone planar subdivision with n vertices. There exists a dynamic point-location data structure for $\mathcal{R}$ that* (i) *has bounded record in-degree,* (ii) *uses $O(n)$ space, and* (iii) *supports queries and update operations Insert, Expand, Delete, and Contract, each in $O(\log^2 n)$ time.*

Proof. (i) It is straightforward to verify that the above data structure has records with bounded in-degree. (ii) The space used is $O(1)$ per vertex, edge, and region. Hence, by Euler's formula, the total space requirement is $O(n)$. (iii) The algorithms for queries and operations *Insert* and *Delete* are essentially as described in [13], with the following minor modifications to take into account the variations of the data structure. In the query algorithm, tree *proper*(σ) is used to discriminate the query point with respect to separator σ. Once the cluster χ containing the query point has been determined, finding the region of χ containing the query point takes time $O(\log^2 n)$, since the lowest and highest vertices of each region in *regions*(χ) are retrieved in $O(\log n)$ time through the dictionary. As shown in [13], the *Insert* and *Delete* algorithms determine a partition of the subdivision $\mathcal{R}$ into $O(1)$ partial subdivisions, called *canonical components*, each represented by an augmented separator-tree. The augmented separator-trees of the canonical components are obtained by splitting the augmented separator tree of $\mathcal{R}$. The canonical components are subsequently reassembled, to yield the updated subdivision, and their augmented separator-trees are appropriately spliced to yield the updated augmented separator-tree of $\mathcal{R}$. Each split/splice operation of the augmented separator-tree is achieved with $O(\log n)$ rotations of the primary separator-tree, for a total time complexity $O(\log^2 n)$.

Now, we discuss operation *Expand*$(e, v, r_1, r_2; v_1, v_2)$. (Operation *Contract* is symmetric.) We consider two cases, depending on the relative order of r_1 and r_2 in $\prec$ prior to the execution of *Expand*:

Case 1. $r_2 \prec r_1$ (see Fig. 4(a)). Note that in this case we must also have $r_2 \uparrow r_1$. The sequence of regions before the expansion, sorted according to the order $\prec$, can be written as a string of the form:

$$A\alpha - - r_2 B\beta \text{ — } \gamma C r_1 - - \delta D.$$

Here, Greek letters denote clusters or portions of clusters, capital letters denote subsequences of regions that contain complete clusters, "—" denotes a channel, and "– –" denotes a potential channel. (Some of the letters may denote empty subsequences.)

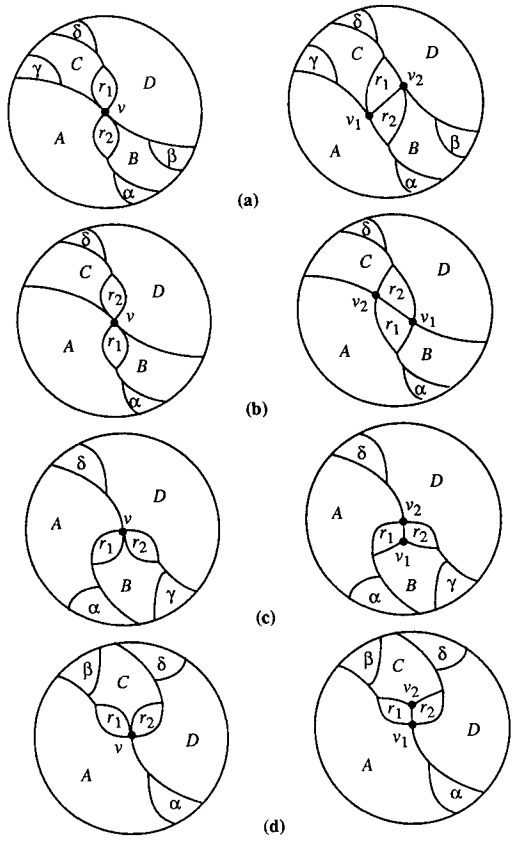

FIG. 4. *Four cases for operation Expand*$(e, v, r_1, r_2; v_1, v_2)$: (a) $r_2 \uparrow r_1$; (b) $r_1 \uparrow r_2$; (c) $r_1 \rightarrow r_2$ *and same lowest vertex for* r_1 *and* r_2; (d) $r_1 \rightarrow r_2$ *and same highest vertex for* r_1 *and* r_2.

After the expansion, the new sequence of regions is:

$$A\alpha - - \gamma C r_1 r_2 B \beta - - \delta D.$$

Case 2. $r_1 \prec r_2$ (see Figs. 4(b)–(d)). The sequence of regions before the expansion is now of the form:

$$A\alpha - - r_1 - - \beta B - C\gamma - - r_2 - - \delta D,$$

where some of the letters may denote empty subsequences. The expansion leaves the sequence of regions unaltered.

Hence, operation *Expand*$(e, v, r_1, r_2; v_1, v_2)$ can also be performed by decomposing the subdivision $\mathcal{R}$ into its canonical components A, B, C, D, α, β, γ, δ, r_1, and r_2, which are subsequently reassembled in the new order to yield the updated $\mathcal{R}$. We conclude that operation *Expand* and its symmetric *Contract* take $O(\log^2 n)$ time. □

It is important to observe that the dynamic point-location data structure of Theorem 3.1 depends on the topology, and not on the specific geometry, of the subdivision. This is formally expressed by the following straightforward lemma.

LEMMA 3.2. *Let* $\mathcal{R}_1$ *and* $\mathcal{R}_2$ *be monotone subdivisions whose associated planar st-graphs* G_1 *and* G_2 *are isomorphic. A dynamic point-location data structure for* $\mathcal{R}_1$ *(as discussed in Theorem* 3.1*) can be used for dynamic point-location in* $\mathcal{R}_2$ *after implementing the appropriate modifications of values of the vertex coordinates.*

Note, however, that for a given monotone subdivision $\mathcal{R}$, there are several versions of the dynamic point-location data structure, corresponding to equivalent versions of the primary separator-tree and of the secondary trees.

4. Spatial point-location. Let $\mathcal{P}$ be a convex cell-complex with n vertices and N facets. Note that both n and the number of edges of $\mathcal{P}$ are $O(N)$. The z-coordinates of the vertices are denoted $z_1, \cdots, z_n$, from bottom to top. Let $\mathcal{P}(z)$ be the intersection of $\mathcal{P}$ with the plane $\pi(z)$ parallel to the x and y axes and at height z. It is easy to verify that $\mathcal{P}(z)$ is a convex subdivision. Also, we define $G(z)$ as the planar st-graph associated with $\mathcal{P}(z)$.

By viewing z as a measure of "time" we consider the process of making plane $\pi(z)$ sweep the cell complex $\mathcal{P}$. While the geometry of $\mathcal{P}(z)$ continuously evolves in time, its topology changes only when plane $\pi(z)$ goes through a vertex v of $\mathcal{P}$. This is formalized as follows.

LEMMA 4.1. *For z', z'' such that $z_i < z', z'' < z_{i+1}$ the digraphs $G(z')$ and $G(z'')$ are isomorphic.*

Proof. The vertices and edges of $\mathcal{P}(z)$ are the intersections of the edges and facets of $\mathcal{P}$ with $\pi(z)$, respectively. Let $f(z)$ be the edge of $\mathcal{P}(z)$ generated by the intersection of facet f with $\pi(z)$. The slope of $f(z)$ in the plane $\pi(z)$ is the same for all z such that $\pi(z)$ intersects f. Since there are no vertices of $\mathcal{P}$ in the region of space $z' < z < z''$, we conclude that the digraphs $G(z')$ and $G(z'')$ are isomorphic. □

By Lemmas 3.2 and 4.1 the same point-location data structure can be used for all query points whose z-coordinate is in the open range (z_i, z_{i+1}), provided the x and y coordinates of the vertices are expressed as (linear) functions of z.

An *event* occurs when plane $\pi(z)$ goes through a vertex v of $\mathcal{P}$, and results in updating the subdivision $\mathcal{P}(z)$ (see Fig. 5). Let z_- and z_+ be "instants" immediately preceding and following z. The transformation from $\mathcal{P}(z_-)$ to $\mathcal{P}(z)$ consists of contracting a subgraph G_- into vertex v. The vertices (respectively, the edges) of G_- are those associated with the edges (respectively, the facets) of $\mathcal{P}$ whose highest vertex is v. Conversely, the transformation from $\mathcal{P}(z)$ to $\mathcal{P}(z_+)$ consists of expanding vertex v into a subgraph G_+. The vertices (respectively, the edges) of G_+ are those associated with the edges (respectively, the facets) of $\mathcal{P}$ whose lowest vertex is v.

The following lemma shows that the above contractions/expansions of subgraphs into/from vertices can be performed by a sequence of elementary contractions/expansions of edges into/from vertices.

A subgraph S of a planar st-graph G is said to be *contractible* if S is vertex-induced and the graph G_S obtained from G by contracting S into a single vertex is itself a planar st-graph.

LEMMA 4.2. *Let G be a planar st-graph, and S a contractible subgraph of G with m edges. There exists a sequence of m elementary updates, each a deletion or a contraction, that transforms G into G_S.*

Proof. Consider in turn each edge e of S, and recall that any edge of a planar st-graph is either removable or contractible [15]. If e is contractible, then contract it, otherwise remove it. We claim that this process will correctly produce graph G_S. Indeed, the process would fail if the current edge $e = (v', v'')$ is a bridge (separation edge) of the graph to which S has currently transformed, but is not contractible (see Fig. 6). This implies that the graph to which G has currently transformed has a directed path from v' to v'' that contains at least one vertex w not in S. Hence, G_S has a directed cycle containing vertex w. This contradiction establishes the

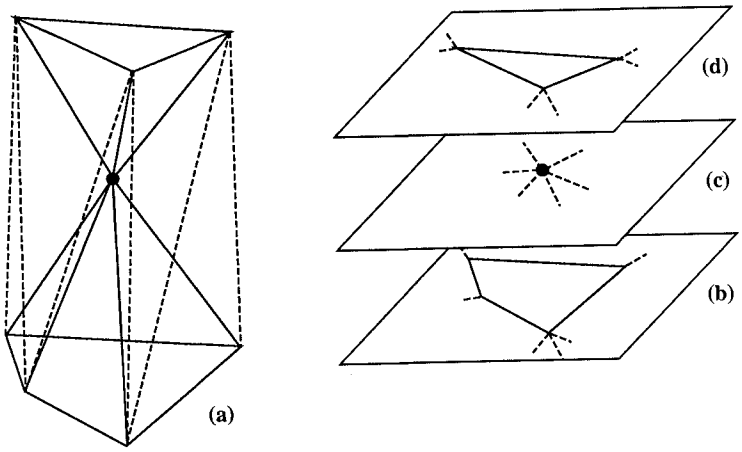

FIG. 5. (a) *Cell complex.* (b) *Graph* $G(z_-)$. (c) *Graph* $G(z)$. (d) *Graph* $G(z_+)$.

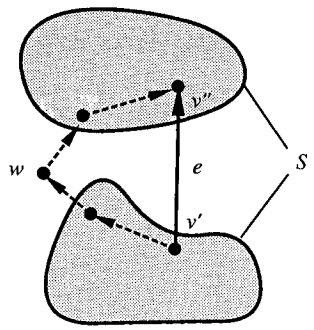

FIG. 6. *Example of contraction of a subgraph S of a planar st-graph by means of elementary deletions and contractions of edges. A contradiction is achieved if edge e is a bridge of the current S but is not contractible.*

claim. □

The proof of the above lemma shows that the contraction is executable in total time $m \times$ (update time). The symmetric pair (*Insertion, Deletion*) and their dual pair (*Expansion, Contraction*) readily establish the following complementary result.

LEMMA 4.3. *Let G be a planar st-graph, and S a contractible subgraph of G with m edges. There exists a sequence of m elementary updates, each an insertion or an expansion, that transforms* G_S *into G.*

THEOREM 4.4. *Space-sweep of a convex cell complex* $\mathcal{P}$ *with N facets can be performed in* $O(N \log^2 N)$ *time so that at any time point-location queries can be answered in* $O(\log^2 N)$ *time and the space requirement is* $O(N)$.

Proof. By Lemmas 3.2 and 4.1 the space-sweep process goes through the $2n + 1$ topologically different subdivisions $\{\mathcal{P}(z_j) : j = 1, \cdots, n\}$ and $\{\mathcal{P}(z'_j) : j = 0, \cdots, n;\ z_j < z'_j < z_{j+1};\ z_0 = -\infty;\ z_{n+1} = +\infty\}$. By Lemmas 4.2–4.3 the updates of the ephemeral point-location data structure are no more than $2N$. By Theorem 3.1 each such update can be processed in time $O(\log^2 N)$. □

The (ephemeral) dynamic point-location data structure of Theorem 3.1 verifies the conditions for the applicability of the technique of [5], and can therefore be converted

into a partially persistent one that supports queries in past versions. Therefore, we obtain the central result of this paper in the following theorem.

THEOREM 4.5. *Let $\mathcal{P}$ be a convex spatial cell-complex with N facets. Point-location in $\mathcal{P}$ can be performed in time $O(\log^2 N)$ using an $O(N \log^2 N)$-space data structure that can be constructed in time $O(N \log^2 N)$.*

The result of Theorem 4.5 can be easily extended to a nonconvex spatial cell-complex $\mathcal{P}$ such that each subdivision $\mathcal{P}(z)$ $(-\infty < z < +\infty)$ is connected and monotone.

5. Adding persistence to special cell-complexes. In this section we illustrate the simplifications of the general technique which occur when exploiting special properties of the cell-complex and the prior knowledge of the problem instance before executing the space-sweep. (Note that the latter feature is, in general, a potential source of simplification in persistence-addition techniques.) Specifically, we shall consider Voronoi diagrams and the cell-complexes determined by arrangements of planes. We shall show that the update of the order on the set of regions of subdivision $\mathcal{P}(z)$ (which defines the ephemeral point-location data structure) requires only insertions and deletions, but no substantial restructurings (corresponding to the swaps of substrings occurring in the general case). We begin with Voronoi diagrams.

THEOREM 5.1. *Let $\mathcal{P}$ be the cell-complex induced by the Voronoi diagram of n sites in three-dimensional space. Processing an event in the space-sweep of $\mathcal{P}$ takes $O(\log n)$ time.*

Proof. Excluding degeneracies, every vertex of $\mathcal{P}$ has degree four. Hence processing an event consists of expanding a vertex v into a triangle r, or of contracting r into v. Instead of simulating such transformation by means of elementary updates, we consider directly its effect of the ordering of the regions of $\mathcal{P}(z)$. We use the extension of the order $\prec$ to the set of vertices, edges, and regions of a monotone subdivision introduced in [15], and the local characterization of such ordering given in [16]. First, consider the expansion of vertex v into a triangle r. The update of the extended order is performed by simply replacing v with r and its vertices and edges. Hence, the sequence of the regions is modified by the insertion of r. Symmetrically, the contraction of r into v causes the deletion of r from the sequence of regions.

Since the primary separator-tree is modified only by insertions and deletions, we implement it as a red-black tree [7], so that rebalancing after an insertion or deletion is done with $O(1)$ rotations. The time bound follows from the fact that a rotation of the primary separator-tree takes $O(\log n)$ time, due to the split/splice operations on the trees of proper edges attached to the nodes involved in the rotation. □

The technique for arrangements of planes uses a simpler ephemeral planar point-location structure, especially designed for space-sweep of arrangements.

THEOREM 5.2. *Let $\mathcal{P}$ be the cell-complex induced by an arrangement of n planes in three-dimensional space. Processing an event in the space-sweep of $\mathcal{P}$ takes $O(\log n)$ time and involves $O(1)$ changes of pointers.*

Proof. For an arbitrary z, let $l_1, l_2, \cdots, l_n$ be the intersections of the planes of the arrangement with the sweep-plane $\pi(z)$. On this plane we establish Cartesian axes x and y, which are projections according to z of the homologous axes of the space. Let x_0 be an abscissa such that, for $j = 1, \cdots, n, x_0 \leq x_j$, where x_j is the abscissa of the intersection of l_j with the x-axis. We denote by π_j the half-plane determined by l_j and not containing point $(x_0, 0)$. With each region r of the planar arrangement $\mathcal{P}(z)$ we associate an integer *weight* $w(r) \in \{0, 1, \cdots, n\}$, denoting the number of half-planes in $\{\pi_1, \cdots, \pi_n\}$ containing r. A complete family of n edge-disjoint separators for $\mathcal{P}(z)$ is

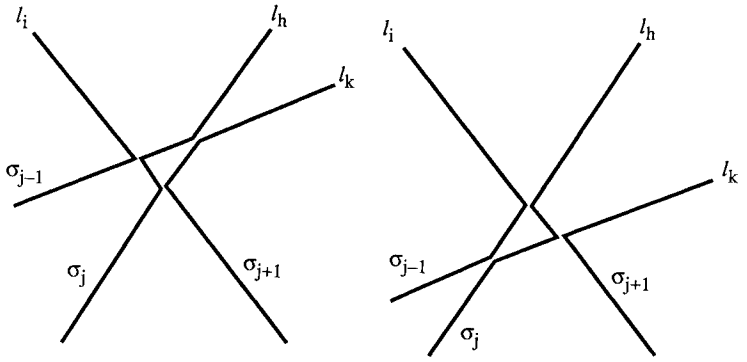

FIG. 7. *Update of the secondary components for the cell-complex induced by an arrangement of planes.*

obtained by defining separator σ_i $(i = 1, \cdots, n)$ as the monotone chain leaving to its left the regions of weight less than or equal to $(i-1)$ and to its right the regions of weight greater than or equal to i. (Note that this family of separators is, in general, not associated with the order $\prec$ defined in §2.1.)

When processing an event, the primary component of the separator tree undergoes no modification. Updates occur only in the secondary components. Specifically, an event point z of the space-sweep corresponds to a point in space where three planes meet. Let l_i, l_h, and l_k be their intersections with $\mathcal{P}(z)$ and let σ_{j-1}, σ_j, and σ_{j+1} be the three separators sharing their common intersection. Suppose that, prior to the update, σ_{j-1} does not contain any portion of l_h; then, the update consists of deleting a segment of l_h from σ_{j+1}, inserting one such segment into σ_{j-1}, and exchanging the order of segments of l_i and l_k in σ_j (see Fig. 7). (A symmetric action takes place when σ_{j+1} contains no portion of l_h.) Assuming that the triplet (l_i, l_h, l_k) corresponding to z has been precomputed, the update of the secondary components, globally involving a bounded number of pointer changes, is executed in time $O(\log n)$. □

Theorems 5.1 and 5.2 lead to the following corollary.

COROLLARY 5.3. *Let $\mathcal{P}$ be the cell-complex in three-dimensional space induced either by the Voronoi diagram of n sites or by an arrangement of n planes. Point-location in $\mathcal{P}$ can be performed in time $O(\log^2 N)$ using an $O(N)$-space data structure that can be constructed in time $O(N \log N)$, where N is the number of facets of $\mathcal{P}$ (N is $O(n^2)$ for the Voronoi diagram and $O(n^3)$ for the arrangement of planes.)*

Acknowledgment. We would like to thank Steven Fortune for useful discussions.

REFERENCES

[1] B. CHAZELLE, *How to search in history*, Inform. and Control, 64 (1985), pp. 77–99.
[2] B. CHAZELLE AND J. FRIEDMAN, *A deterministic view of random sampling and its use in geometry*, in Proc. 29th IEEE Symposium on Foundations of Computer Science, 1988, pp. 539–548.
[3] K. L. CLARKSON, *New applications of random sampling in computational geometry*, Discrete & Comput. Geom., 2 (1987), pp. 195–222.
[4] R. COLE, *Searching and storing similar lists*, J. Algorithms, 7 (1986), pp. 202–220.
[5] J. R. DRISCOLL, N. SARNAK, D. D. SLEATOR, AND R. E. TARJAN, *Making data structures persistent*, J. Comput. System Sci., 38 (1989), pp. 86–124.

[6] H. EDELSBRUNNER, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Berlin, New York, 1987.

[7] L. J. GUIBAS AND R. SEDGEWICK, *A dichromatic framework for balanced trees*, in Proc. 19th IEEE Symposium on Foundations of Computer Science, 1978, pp. 8–21.

[8] D. KELLY AND I. RIVAL, *Planar lattices*, Canad. J. Math. 27 (1975), pp. 636–665.

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

[10] A. LEMPEL, S. EVEN, AND I. CEDERBAUM, *An algorithm for planarity testing of graphs*, Internat. Symposium on Theory of Graphs, Gordon and Breach, New York, 1967, pp. 215–232.

[11] F. P. PREPARATA, *Planar point location revisited: a guided tour of a decade of research*, Internat. J. Found. Comput. Sci. 1 (1990), pp. 71–86.

[12] F. P. PREPARATA AND M. I. SHAMOS, *Computational geometry*, Springer-Verlag, New York, 1985.

[13] F. P. PREPARATA AND R. TAMASSIA, *Fully dynamic point location in a monotone subdivision*, SIAM J. Comput. 18 (1989), pp. 811–830.

[14] N. SARNAK AND R. E. TARJAN, *Planar point location using persistent search trees*, Comm. ACM, 29 (1986), pp. 669–679.

[15] R. TAMASSIA AND F. P. PREPARATA, *Dynamic maintenance of planar digraphs, with applications*, Algorithmica, 5 (1990), pp. 509–527.

[16] R. TAMASSIA AND J. S. VITTER, *Parallel transitive closure and point location in planar structures*, SIAM J. Comput., 20 (1991), pp. 708–725.