ELSEVIER

Computational Geometry 17 (2000) 153–163

Computational Geometry
Theory and Applications

www. elsevier.nl/locate/comgeo

Union and split operations on dynamic trapezoidal maps ☆

Monique Teillaud [1]

INRIA, B.P.93, 06902 Sophia-Antipolis cedex, France

Communicated by K. Mehlhorn; received 25 January 2000; received in revised form 15 September 2000; accepted 12 October 2000

Abstract

We propose algorithms to perform two new operations on an arrangement of line segments in the plane, represented by a trapezoidal map: the split of the map along a given vertical line D, and the union of two trapezoidal maps computed in two vertical slabs of the plane that are adjacent through a vertical line D. The data structure we use is a modified Influence Graph, still allowing dynamic insertions and deletions of line segments in the map. The algorithms for both operations run in $O(s_D \log n + \log^2 n)$ time, where n is the number of line segments in the map, and s_D is the number of line segments intersected by D. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Dynamic algorithms; Randomized algorithms; Data structures; Parallel data structures; Trapezoidal map

1. Introduction

Let us assume that we are computing the arrangement of a set of line segments in the plane in parallel, under the following framework: the plane is partitioned into vertical slabs, and each processor is associated with a slab V, namely $V =]a, b[\times]-\infty, +\infty[$, where $a, b \in \mathbb{R} \cup \{-\infty, +\infty\}$, and $a < b$. Such a slab will be denoted as $V_{a,b}$, and its associated processor $P_{a,b}$. We call the line $x = a$ the *left boundary line* of V and the line $x = b$ its *right boundary line*. A processor computes the trapezoidal map formed by the line segments intersecting its associated slab V. In addition, the construction of the arrangement must be dynamically maintained, which means that line segments can be inserted or deleted, and the arrangement must be consequently updated.

This can lead to a bad balancing of the data between the processors, some of them being overloaded, while others are under-loaded. We must thus perform load-balancing between the processors. The vertical slab $V_{a,b}$ associated to an overloaded processor P will be split into two disjoint slabs, $V_{a,b} = V_{a,c} \cup V_{c,b}$,

☆ This work was partially supported by ESPRIT Basic Research Action r. 7141 (ALCOM II). A preliminary version appeared as [22].

E-mail address: Monique.Teillaud@sophia.inria.fr (M. Teillaud).

[1] http://www-sop.inria.fr/prisme/personnel/teillaud/.

PII: S0925-7721(00)00019-5

$c \in]a, b[$. Then P will be associated for example with the vertical slab $V_{a,c}$, and $V_{c,b}$ will be united to the adjacent slab $V_{b,d}$. While these split/union operations are performed on the slabs, the corresponding split/union operations must be performed on the arrangements by the associated processors.

As previously stated, the possibility of inserting and deleting segments dynamically and efficiently in each slab must be preserved. As the Influence Graph [2] allowed us to construct the trapezoidal map for a set of line segments in the plane in a dynamic way, we try here to modify this structure in such a way that split and union operations along vertical lines can be handled. Moreover, locating points or more general objects in trapezoidal maps appears to be useful, and the Influence Graph is naturally adapted to efficient location queries. With a similar motivation, Duboux and Ferreira study the reorganization of a dictionary and its implementation on a multicomputer [15,16].

In Section 2, we adapt the basic definitions about trapezoidal maps to our problem, then in Section 3 we show how the Influence Graph can be modified to permit the split and union operations, for which algorithms are given in Sections 4 and 5, respectively. A randomized analysis of the algorithms is proposed in Section 6.

2. The trapezoidal map

Trapezoidal maps are often used to compute the intersecting pairs among a set S of n line segments. The trapezoidal map $\boldsymbol{T}(S)$ is defined as follows: from each endpoint of a line segment in S, or each intersection point of two line segments in S, extend a vertical segment to the first segment in S above, and to the first segment below (or to infinity if there is no such segment in S). We obtain in this way a subdivision of the plane into trapezoids, some of them being degenerate (Fig. 1). A trapezoid is determined by at most 4 segments of S, it has a *floor*, a *ceiling*, and two vertical *walls*, through which it can have at most 4 adjacent trapezoids (at most 2 per wall), called *horizontal neighbors* (Fig. 2).

The definition of a trapezoidal map can be adapted to our problem: a processor P computes the trapezoidal map in its associated slab $V = V_{a,b}$. Then we must distinguish different types of trapezoids:

- *Usual trapezoids*, also called complete trapezoids (Fig. 3(a)).
- *Simply incomplete trapezoids*, that are bounded on one side by a boundary line w of V. Such a trapezoid T has a floor and a ceiling that both intersect w, T is *left-incomplete* if w is the left boundary

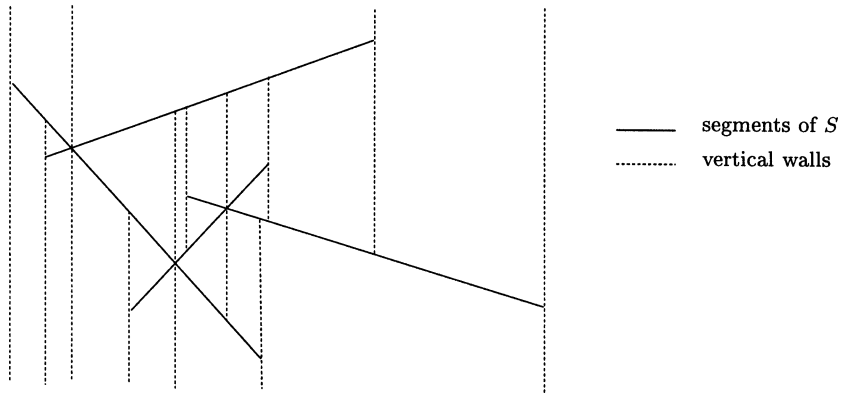

Fig. 1. A trapezoidal map.

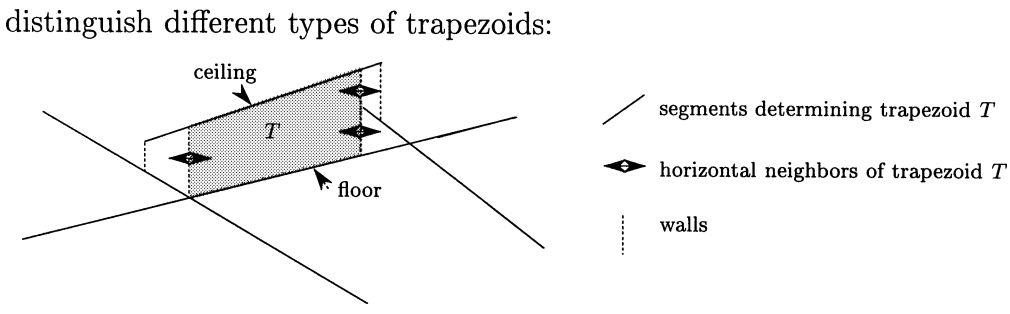

Fig. 2. A trapezoid.

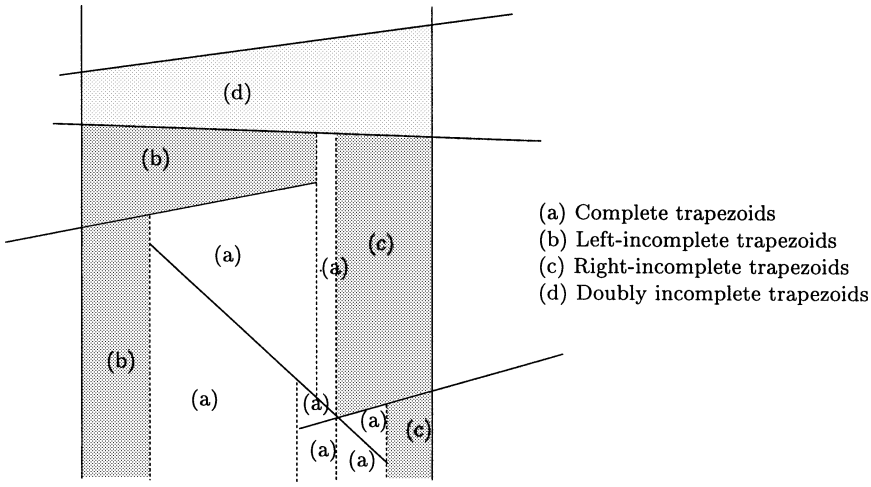

Fig. 3. Trapezoids in a slab.

line of V, *right-incomplete* otherwise (Fig. 3(b,c)). The floor and the ceiling of T also determine a trapezoid in the slab adjacent to V with the same boundary line w. This trapezoid would complete T to form a usual trapezoid if we were considering the trapezoidal map in the whole plane. A simply incomplete trapezoid has one wall, thus at most two horizontal neighbors, that are the at most two trapezoids adjacent to it in the same slab V, and it is determined by at most 3 segments. Inversely, we will also use the terms *left-complete* and *right-complete*.

- *Doubly incomplete* trapezoids are determined by two segments traversing V, they are bounded by the two boundary lines of V (Fig. 3(d)), they have no wall, and no horizontal neighbor.

3. The data structure

In this section, we first recall the definition of the Influence Graph and the way it can be used to compute the trapezoidal map of a set S of line segments in the plane, then we show how the structure must be modified to allow us to perform splits and unions of trapezoidal maps.

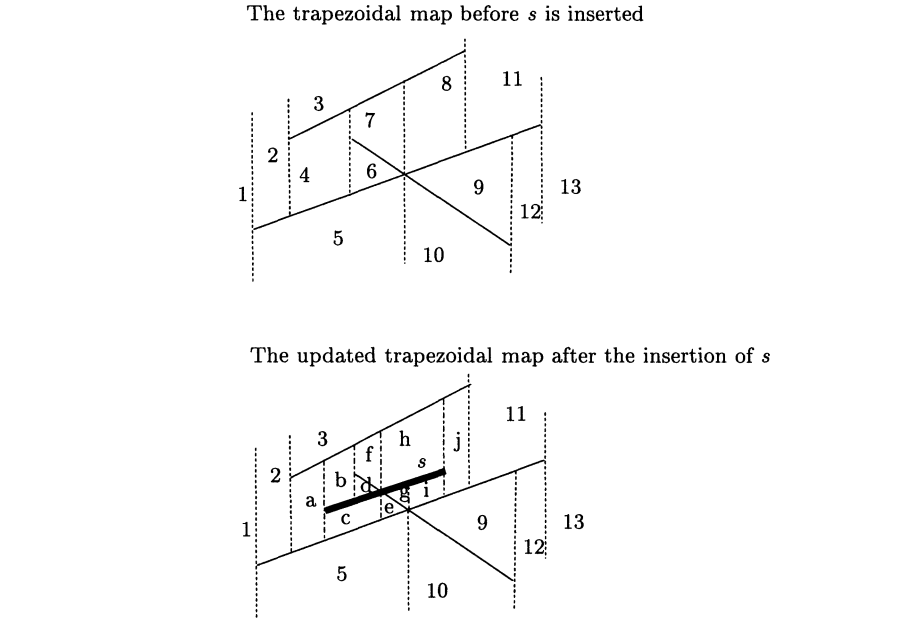

Fig. 4. Insertion of segment s.

3.1. The Influence Graph for trapezoidal maps

The Influence Graph [2,21] is based on the idea of maintaining the *history* of the construction by an incremental algorithm [3,4]. It allows semi-dynamic constructions, with only insertions, and dynamic constructions, with insertions and deletions, in particular cases [11], and in a general setting, it can be *augmented to* obtain such dynamic constructions [14]. However, for the trapezoidal map, it has been shown that a standard Influence Graph was sufficient (see [12,21], or [5,6] for a revised algorithm) to allow deletions.

We have a special interest in trapezoids determined by segments of a set of line segments S, as we defined them in Section 2. We will say that such trapezoids are *defined by* S. The *Influence Graph* is a rooted directed acyclic graph, whose nodes are associated with trapezoids defined by S. Its root corresponds to the whole plane, which is the trapezoid defined by an empty set of line segments. If we compute a trapezoidal map incrementally, each new segment s will intersect some of the trapezoids

defined by the segments inserted before it. Those trapezoids must be removed from the updated trapezoidal map, and replaced by new ones. But they will remain stored in the Influence Graph as trapezoids *killed* by s. In this way, the Influence Graph stores the whole *history* of the construction. The new trapezoids are said to be *created* by s. In the Influence Graph, a trapezoid killed by s becomes parent of a trapezoid created by s if, and only if, they overlap (Fig. 4). Thus, a trapezoid can have at most four children, but it may have an arbitrary number of parents.

For each node, we store

- the at most 4 segments determining it (its creator is one of them),
- its at most 4 horizontal neighbors,
- its killer,
- its at most 4 children.

The Influence Graph allows one to locate a segment s to be inserted: since a child is contained in the union of its parents, we will find all the trapezoids of the current trapezoidal map that are intersected by s by recursively traversing the Influence Graph from the root to the leaves (though the graph is not a tree, we call *leaves* its nodes without children), visiting all the trapezoids intersected by s.

3.2. The modified Influence Graph

A processor P computes the trapezoidal map in its associated slab $V = V_{a,b}$, by constructing incrementally an Influence Graph, with the following modifications in each node.

- We store a mark "complete, left-incomplete, right-incomplete, doubly-incomplete".
- We need to store not only the links from a node to its children, but also to its parents. The number of parents of a node is arbitrary, but it can be noticed that the parents of a node are naturally sorted from left to right along the segment that created it. This allows us to store pointers to the parents of each node in a concatenable queue [1], enabling the splitting of a set of parents, or the concatenation of two consecutive (sorted along the same segment) lists of parents of two adjacent trapezoids in time logarithmic in the total number of parents. Such a structure can be implemented as a 2–3 tree. The root of a tree corresponds to a node of the Influence Graph, its leaves correspond to the parents of this node.

In the 2–3 tree, the links between nodes will not be uni-directional as usual: we need bi-directional links to be able to reach the root from the leaves as well as to reach the leaves from the root. With these bi-directional links, when an Influence Graph is traversed, the children of a node N will be obtained by accessing to the roots of the 2–3 trees N belongs to. Since a node of the Influence Graph has up to four children, it appears as a leaf in up to four 2–3 trees. So, finding N's children can be done in time logarithmic in the number of elements of the 2–3 trees it appears in, which is itself bounded by the total number of parents of the (at most four) children of N. This will result in a $O(\log n)$ term in the complexity of locating a new segment in the modified Influence Graph, where n is the total number of segments.

Note that this structure does not change the time of creating a new node: as previously noticed, the set of parents is sorted, so the structure can be constructed in time proportional to the number of parents.

Notice that a complete trapezoid can have incomplete children, and that incomplete trapezoids usually have both complete and incomplete children.

4. Splitting a trapezoidal map

Let D be a vertical line, included in a vertical slab V. Let us assume that the trapezoidal map of the set of line segments intersecting V, together with the corresponding Influence Graph, have already been computed. We want to construct the two Influence Graphs corresponding to the slabs V_l (on the left hand side of D) and V_r (on the right hand side of D), respectively, which are obtained by splitting V along D. Both of the two new Influence Graphs must be the same as what would have been obtained by directly constructing them in the new slabs. Of course, moreover, we do not want to compute them from scratch.

The idea of the algorithm consists in recursively traversing V's Influence Graph, in order to find all trapezoids in the history that are split by D. While performing this traversal, the two new Influence Graphs will be constructed.

First a new root is created, that will, for example, be the root of the right Influence Graph, and the old root will be the root of the left Influence Graph. The following operations are then recursively performed: each node N visited must be split into two parts, called N_l, which is the part of N on the left of D, and which will be a node of the left Influence Graph, and similarly N_r, on the right hand side.

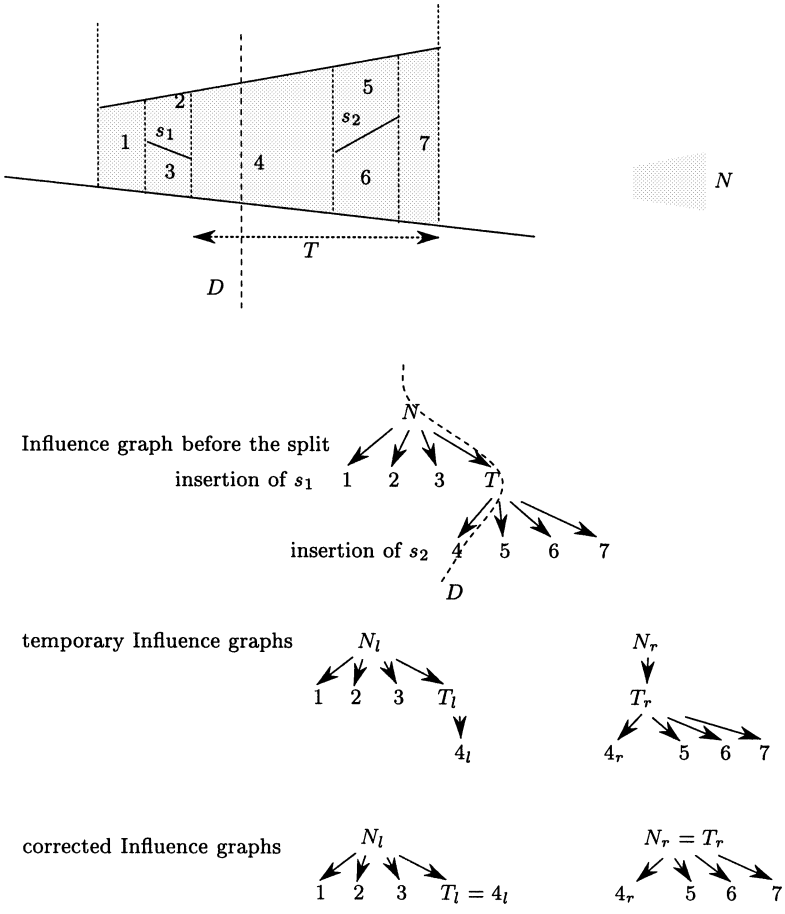

Fig. 5. Splitting a node N.

- If N was right-incomplete, then N_r is doubly-incomplete, and if N was left-incomplete, then N_l is doubly-incomplete; if N was doubly-incomplete, both N_r and N_l are doubly-incomplete; otherwise, N was complete, N_l is right-incomplete, and N_r is left-incomplete.
- A child that is not split by D becomes a child of N_l or N_r according to its position with respect to D.
- A child T split by D is recursively examined. It will be split into two trapezoids T_l, child of N_l and T_r, child of N_r.
- The parents of N that are not split by D must also be updated: the ordered set of parents of N is split by D forming the two sets of parents of N_l and N_r. This is performed by splitting in logarithmic time the concatenable queue storing the set of parents of N into two concatenable queues, storing the parents of N_l and N_r, respectively.

The preceding operations update the children of N. However, the Influence Graphs thus created are only temporary ones, and in order to create correct Influence Graphs, the following improvements must be done.

- If N_r and N_l both have 2 or 3 children, the modified part of the structure is correct.
- If N_r only has one child T_r, then it is easy to see that in fact N_r and T_r are two identical trapezoids. The link parent–child between them has no reason to exist, it would not have been created if the Influence Graph for the slab V_r had directly been constructed, because it does not correspond to any insertion of a segment in this slab. The two nodes must then be merged. (In fact, this could have been detected and solved while performing the operations described just above.)
- The same holds for N_l and T_l.

See Fig. 5 for the illustration of the whole process.

5. Union of two trapezoidal maps

In fact, this operation is nothing but the exact inverse of a split. We are given the Influence Graphs for two slabs V_r and V_l, adjacent along a vertical line D. The Influence Graph in slab $V = V_r \cup V_l$ must be deduced from these two given Influence Graphs.

To this aim, both Influence Graphs will be simultaneously traversed starting from their root, by visiting all left-incomplete nodes in V_r and all right-incomplete nodes in V_l, which will be appropriately merged. The other nodes are not modified and need not to be traversed. The new Influence Graph is constructed during the recursive traversal.

More precisely, the roots of the two Influence Graphs must be first merged. Then, at each step of the recursion, let N_r and N_l be nodes of the Influence Graphs of V_r and V_l, respectively, to be merged to form a trapezoid $N = N_r \cup N_l$ in the new Influence Graph. They have the same ceiling and the same floor. A new node N is thus created. The ceiling and the floor of this new node are the same as N_r's and N_l's. Its horizontal neighbors on the right (respectively left) are those of N_r (respectively N_l), if they exist, otherwise N is right-incomplete if N_r was, and left-incomplete if N_l was, doubly-incomplete if N_r was right-incomplete and N_l was left-incomplete.

- If N_r and N_l had been killed by the same segment, then N can receive their children. And the recursion goes on, for the two children that must be merged among all children of N: two respective children T_l and T_r of N_l and N_r must be merged if they have the same ceiling and the same floor. The set of parents of the new node T thus created is obtained by concatenating, from left to right, the parents of T_l, the new trapezoid N, and the parents of T_r.

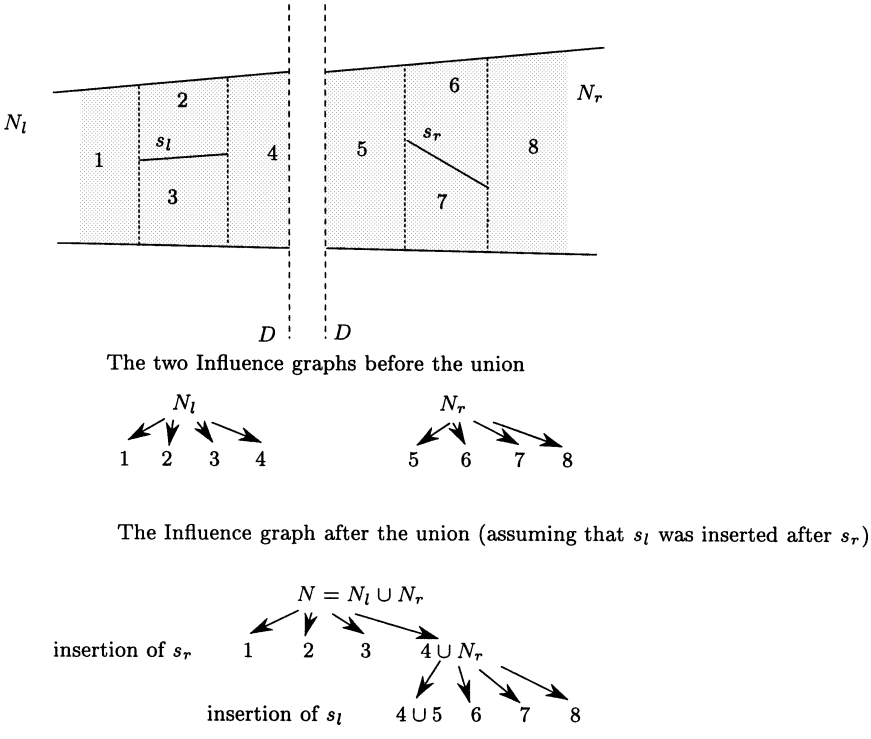

Fig. 6. Merging two nodes N_l and N_r.

- Otherwise, if N_r was killed by a segment s_r before N_l was killed by s_l. In this case, N is killed by s_r in the new Influence Graph, its children are the left-complete children of N_r, and a new trapezoid T formed by the union of N_l (which is not dead yet when s_r is inserted) with the child T_r of N_r that is left-incomplete. The total set of parents of T is obtained by adding N to the set of parents of T_r. The recursion then goes on with the merge of N_l and T_r.
- The symmetric case when N_l was killed before N_r is handled in the same way.

6. Analysis

Let S denote a set of n line segments. We perform a randomized analysis of the algorithm. We refer to Clarkson's introduction to this method of analysis in computational geometry [7,8]. Other articles or books present the analysis differently [9,10,18,20]. Dynamic cases were also studied in the literature [11, 17,19]. We assume that the n line segments are inserted in random order, i.e., that the $n!$ possible orders of insertion are equally likely. For a deletion, we also assume that any segment is chosen with the same probability. Then a randomized analysis shows the following.

Theorem 6.1. *Using the modified Influence Graph, an arrangement of n line segments can he computed on-line with* $\mathrm{O}(n + a)$ *expected space and* $\mathrm{O}(\log^2 n + (a/n)\log n)$ *expected update time, where a*

is the complexity of the arrangement. The deletion of a line segment can be performed in expected $O(\log^2 n + (a/n)\log n \log\log n)$ *time.*

Proof. The expected complexity given for the usual Influence Graph in [2,21] revised in [5,6] are the following: $O(\log n + a/n)$ update time for an insertion, and $O(\log n + (a/n)\log\log n)$ update time for a deletion. The modification of the structure, introduced in Section 3.2, consisting in using concatenable queues to link a node to its children, results in an additional $\log n$ factor in the time complexity of the traversal of the Influence Graph (but they do not change the space complexity), which yields to the result. □

Remark. The algorithm for deleting a segment uses an efficient priority queue [23] and dynamic perfect hashing [13], in order to achieve this running time. If we use only standard balanced trees, the time complexity of a deletion is $O((1 + (a/n))\log^2 n)$.

We can now analyze the complexity of the algorithms proposed for the split and union in the previous sections. Let n_V denote the number of segments intersecting a given slab V, and s_D is the number of segments intersected by the vertical line D. We first show the following lemma.

Lemma 6.1. *The expected number of nodes of the Influence Graph for slab* V*, that are split by* D *is* $O(s_D + \log n_V)$.

Proof. After the insertion of the kth segment in slab V, the segments already inserted in the Influence Graph form a random sample of size k of the n_V segments. The expected number of present segments that are intersected by D is $(k/n_V)\cdot s_D$. Thus there are $(k/n_V)\cdot s_D + 1$ trapezoids of the current trapezoidal map that are split by D. A trapezoid in this collection has been created by the kth insertion if, and only if, the kth segment inserted is one of the at most four segments determining this trapezoid. The expected number of trapezoids of the Influence Graph, split by D is thus

$$\sum_{k=1}^{n_V}\left(\frac{k}{n_V}\cdot s_D + 1\right)\cdot\frac{4}{k} = O(s_D + \log n_V). \qquad \square$$

Theorem 6.2. *A split of a trapezoidal map along a line* D *can be performed in* $O(s_D \log n_V + \log^2 n_V)$ *expected time complexity.*

Proof. The algorithm locates in the modified Influence Graph the expected $O(s_D + \log n_V)$ nodes to be split. When a node is split, we find its at most four children in $O(\log n_V)$ worst-case time by traversing the at most four corresponding concatenable queues. We also find its neighbors in constant time. Each node to be split is updated. Note that all the required operations are done in constant time, except the update of the parents: the sorted set of parents of the split node is divided into two sorted sets corresponding to the two new nodes. Since the links joining a node to its parents are organized in a concatenable queue, this division can be achieved in $O(\log n_V)$ worst-case time, which implies the result. □

Since we noticed that computing the union of two Influence Graphs is the inverse of splitting an Influence Graph, we have the following theorem.

Theorem 6.3. *Computing the union of two trapezoidal maps along a vertical line D is done in* $O(s_D \log n_V + \log^2 n_V)$ *expected time complexity.*

The logarithmic factor added in all the complexity results because of the links from each node to its parents is probably overestimated for the practical implementation: since the number of children of a node is constant, the average number of parents of a node will also be constant, though a given node can have a linear number of parents. We can thus expect this factor to appear as a constant, in the observed complexity of the implemented algorithms.

Note that our results are output-sensitive, since the complexity depends on the number of segments intersected by the line D. No assumption on the vertical line D is made.

7. Concluding remarks

It must be noticed that this algorithm is very simple, as all algorithms using an Influence Graph are.

Though we have only been considering a vertical line D throughout this paper, we can notice that the algorithm would apply to a line with any direction. The description of the trapezoids resulting from a split by a non-vertical line would of course be more complex, but the algorithm would run similarly. However, an analysis similar to what was done in Section 6 would unfortunately be impossible: the number of trapezoids intersected by D would no longer depend on the number of line segments intersected by D, since D could traverse a trapezoid by intersecting only its two vertical walls and none of the segments determining it.

The extension of this algorithm to more general curves could also be imagined, with the same restriction on the analysis. It would provide a simple algorithm allowing a new kind of range-searching: we could not only count or report the segments intersecting a given domain, but also return their trapezoidal map, together with the corresponding Influence Graph, allowing further insertions and deletions in the domain.

Acknowledgements

I would like to thank Afonso Ferreira and Thibault Duboux for bringing the problem to my attention and Olivier Devillers for helpful discussions. The figures were realized using the interactive drawing preparation system JPdraw written by Jean-Pierre Merlet.

References

[1] A.V. Aho, J.E. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA, 1974.

[2] J.-D. Boissonnat, O. Devillers, R. Schott, M. Teillaud, M. Yvinec, Applications of random sampling to on-line algorithms in computational geometry, Discrete Comput. Geom. 8 (1992) 51–71.

[3] J.-D. Boissonnat, M. Teillaud, A hierarchical representation of objects: The Delaunay tree, in: Proc. 2nd Annu. ACM Sympos. Comput. Geom., 1986, pp. 260–268.

[4] J.-D. Boissonnat, M. Teillaud, On the randomized construction of the Delaunay tree, Theoret. Comput. Sci. 112 (1993) 339–354.
[5] J.-D. Boissonnat, M. Yvinec, Géométrie Algorithmique, Ediscience international, Paris, 1995.
[6] J.-D. Boissonnat, M. Yvinec, Algorithmic Geometry, Cambridge University Press, UK, 1998, translated by Hervé Brönnimann.
[7] K.L. Clarkson, New applications of random sampling in computational geometry, Discrete Comput. Geom. 2 (1987) 195–222.
[8] K.L. Clarkson, Applications of random sampling in computational geometry, II, in: Proc. 4th Annu. ACM Sympos. Comput. Geom., 1988, pp. 1–11.
[9] K.L. Clarkson, Randomized geometric algorithms, in: D.-Z. Du, F.K. Hwang (Eds.), Computing in Euclidean Geometry, Lecture Notes Series on Computing, Vol. 1, World Scientific, Singapore, 1992, pp. 117–162.
[10] K.L. Clarkson, P.W. Shor, Applications of random sampling in computational geometry, II, Discrete Comput. Geom. 4 (1989) 387–421.
[11] O. Devillers, S. Meiser, M. Teillaud, Fully dynamic Delaunay triangulation in logarithmic expected time per operation, Computational Geometry 2 (2) (1992) 55–80.
[12] O. Devillers, M. Teillaud, M. Yvinec, Dynamic location in an arrangement of line segments in the plane, Algorithms Rev. 2 (3) (1992) 89–103.
[13] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, R.E. Tarjan, Dynamic perfect hashing: upper and lower bounds, SIAM J. Comput. 23 (1994) 738–761.
[14] K. Dobrindt, M. Yvinec, Remembering conflicts in history yields dynamic algorithms, in: Proc. 4th Annu. Internat. Sympos. Algorithms Comput., Lecture Notes in Computer Science, Vol. 762, Springer, Berlin, 1993, pp. 21–30.
[15] T. Duboux, A. Ferreira, Towards a dynamic parallel database machine: data balancing techniques and pipeline, Technical Report RR 94-47, LIP, Lyon, France, December 1994.
[16] T. Duboux, A. Ferreira, M. Gastaldo, MIMD dictionary machines: from theory to practice, in: L. Bougé et al. (Eds.), Parallel Processing: CONPAR 92 – VAPP V, Lecture Notes in Computer Science, Vol. 634, Springer, Berlin, 1992, pp. 545–550.
[17] K. Mulmuley, Randomized multidimensional search trees: Further results in dynamic sampling, in: Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci., 1991, pp. 216–227.
[18] K. Mulmuley, Computational Geometry: An Introduction Through Randomized Algorithms, Prentice-Hall, Englewood Cliffs, NJ, 1993.
[19] O. Schwarzkopf, Dynamic maintenance of geometric structures made easy, in: Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci., 1991, pp. 197–206.
[20] R. Seidel, Backwards analysis of randomized geometric algorithms, in: J. Pach (Ed.), New Trends in Discrete and Computational Geometry, Algorithms and Combinatorics, Vol. 10, Springer, Berlin, 1993, pp. 37–68.
[21] M. Teillaud, Towards Dynamic Randomized Algorithms in Computational Geometry, Lecture Notes in Computer Science, Vol. 758, Springer, Berlin, 1993.
[22] M. Teillaud, Union and split operations on dynamic trapezoidal maps, in: Proc. 7th Canad. Conf. Comput. Geom., 1995, pp. 181–186.
[23] P. van Emde Boas, R. Kaas, E. Zijistra, Design and implementation of an efficient priority queue, Math. Syst. Theory 10 (1977) 99–127.