Elementary Algorithms for Reporting Intersections of Curve Segments

Jean-Daniel Boissonnat, Antoine Vigneron

Elementary Algorithms for Reporting Intersections of Curve Segments

Jean-Daniel Boissonnat — Antoine Vigneron

N° 3825

Novembre 1999

THÈME 2

ISSN 0249-6399

Elementary Algorithms for Reporting Intersections of Curve Segments

Jean-Daniel Boissonnat, Antoine Vigneron

Thème 2 — Génie logiciel
et calcul symbolique
Projet Prisme

Rapport de recherche n° 3825 — Novembre 1999 — 18 pages

Abstract: We propose several algorithms to report the k intersecting pairs among a set of n curve segments. Apart from the intersection predicate, our algorithms only use two simple predicates : the predicate that compares the coordinates of two points and the predicate that says if a point is below, on, or above a segment. In particular, the predicates we use do not allow to count the number of intersection points nor to sort them, and the time complexity of our algorithms depends on the number of intersecting pairs, not on the number of intersection points (differently from the other non trivial algorithms). We present an algorithm for the red-blue variant of the problem where we have a set of blue segments and a set of red segments so that no two segments of the same set intersect. The time complexity is $O((n + k) \log n)$. This algorithm is then used to solve the general case in $O(n\sqrt{k} \log n)$ time. In the case of pseudo-segments (i.e. segments that intersect in at most one point) we propose a better algorithm whose time complexity is $O((k + n) \log n + n\sqrt{k})$. All our time complexity results are a log factor from optimal.

Important notice : Since the writing of this report, a flaw has been found in the proof of Theorem 3. As a consequence, the analysis of the algorithm for the general case and the analysis of the algorithm for the pseudo-segment case are not correct. A new report that only contains the red-blue case is available as INRIA Report Research RR-3999.

Key-words: Computational Geometry, Robustness, Exact arithmetic, Sweep-line algorithms

This work has been partially supported by the ESPRIT IV LTR Project No 28155 (GALIA) and the Action de Recherche Coopérative Géométrica.

Algorithmes élémentaires pour calculer les intersections d'arcs de courbes

Résumé : On propose plusieurs algorithmes pour détecter, dans un ensemble de n arcs de courbe monotones du plan, les k paires qui se coupent. Nos algorithmes n'utilisent, en plus du prédicat qui décide si deux arcs se coupent, que deux prédicats simples pour comparer les coordonnées de deux points et décider si un point est au dessus, au dessous, ou sur un arc. En particulier, nos prédicats ne permettent pas de compter le nombre de points d'intersection ni de trier les points d'intersection, et le temps de calcul est fonction du nombre de paires d'arcs qui se coupent et non du nombre de points d'intersection à la différence des autres algorithmes non triviaux connus. Le cas bleu-rouge (où les segments sont colorés en bleu ou en rouge de telle façon que les segments rouges ne se coupent pas et de même pour les bleus) est traité en temps $O((n+k)\log n)$. Cet algorithme est à la base d'un algorithme qui traite le cas général en temps $O(n\sqrt{k}\log n)$. Dans le cas de pseudo-segments (c'est-à-dire des arcs qui se coupent en un point au plus), on présente un meilleur algorithme de complexité $O((k+n)\log n + n\sqrt{k})$. Tous nos résultats de complexité sont optimaux à un facteur logarithmique près.

Note importante : Une erreur dans la preuve du théorème 3 a été trouvée après la rédaction de ce rapport. Il s'ensuit que les analyses de l'algorithme général et de l'algorithme pour les pseudo-segments sont incorrectes. Le cas rouge-bleu est repris dans le rapport de recherche INRIA RR-3999

Mots-clés : Géométrie algorithmique, Robustesse, Arithmétique exacte, Algorithmes de balayage

1 Introduction

The usual model to analyze geometric algorithms is the Real RAM which is assumed to compute exactly with real numbers [13]. This model hides the fact that the arithmetic of real computers has a limited precision and ignores numerical and robustness issues. As a consequence a direct implementation of an algorithm that is correct under the Real RAM model does not necessarily translate into a robust and/or efficient program, and catastrophic behaviours are commonly observed.

A first approach to remedy this problem is to use exact arithmetic. In the context of geometric algorithms, much progress has been done in the recent past [16, 17, 14, 12]. Another approach, to be followed here, has emerged recently. Decisions in geometric algorithms depend on geometric predicates which are usually algebraic expressions. For example, for a triple of points given by their cartesian coordinates, deciding what is the orientation of the triangle reduces to evaluating (the sign of) a multivariate polynomial of degree two. If an algorithm only uses predicates of degree 2 as a function of the input data, and if the input data are coded as simple fixed precision numbers, computations can be done exactly using the native double precision hardware of the computer. The degree of an algorithm is therefore related to the precision required to run an algorithm using exact arithmetic. This motivates the design of efficient algorithms of low degree. Reducing the degree of the algorithms will reduce the number and the complexity of the degenerate configurations, make the algorithms more elementary and more general, reduce the amount of numerical computations, which is usually quite a large fraction of the total execution time especially if multi-precision computing is invoked, and possibly also refrain the use of complicated data structures resulting in low space requirements. However, reducing the degree of an algorithm may increase its time complexity in the Real RAM model. Following Liotta et al. [11], we consider the degree of the predicates as an additional measure of the complexity of problems and algorithms, and intend to elucidate the relationship between time-complexity and degree of the predicates. Related research can be found in Knuth's seminal work [10] and in some recent papers [8, 4, 3].

In this paper, we consider the problem of reporting the k intersecting pairs among a set of n monotone curve segments. Optimal $O(n \log n + k)$-time algorithms are known for this problem [7, 1]. However these algorithms use predicates of high degree, e.g. to compare the abscissae of two intersection points or to locate an intersection point with respect to a vertical slab. These predicates have a degree and an algebraic complexity that are usually higher than the intersection predicate : this is in particular the case for line segments and circle segments [3]. Our algorithms only use the intersection predicate and two other simple predicates : the predicate that sorts two points by abscissae, and the predicate that says if a point is below, on, or above a segment. In particular, we do not compute the arrangement nor the trapezoidal map of the segments. Moreover, the predicates we use do not say anything about the number or the positions of the intersection points, and the time complexity of these algorithms depends only on the number of intersecting pairs of segments, not the number of intersection points (differently from the other non trivial algorithms [2, 7, 6, 5, 1]).

We first present an algorithm for the red-blue variant of the problem where the segments are colored blue or red so that there is no intersection among the blue segments and no intersection among the red ones. The time complexity is $O((n + k)\log n)$, which is a log factor from optimal. This result generalizes a similar result for the case of pseudo-segments, i.e. segments that intersect in at most one point [3]. This algorithm is then used to solve the general case in time $O(n\sqrt{k}\log n)$. In the case of pseudo-segments, we propose a better algorithm whose time complexity is $O((k + n)\log n + n\sqrt{k})$. These bounds almost match the $O(n\sqrt{k} + n\log n)$ lower bound [3].

2 The problem

Let E be a set of n monotone segments. A segment is *monotone* if it is the graph of a partially defined univariate function (i.e. any vertical line intersect such a segment in at most one point). The problem is to report the k pairs of segments of E that intersect.

Let s and s' be two segments of E and let p and q be two endpoints of some segments of E, not necessarily of s or s'. $x(p)$ and $y(p)$ denote the coordinates of p, and $s(x)$ denotes the point of s whose abscissa is x (if such a point exists). We consider the following predicates :

Predicate 1: $s \cap s' \neq \emptyset$
Predicate 2: $x(p) \leq x(q)$
Predicate 3: $y(p) \leq y(s(x(p)))$
Predicate 4: $y(s(x(p))) \leq y(s'(x(p)))$
Predicate 5: $\exists x \in [x(p), x(q)]$ such that $s(x) = s'(x)$

Predicate 1 is mandatory. Predicate 2 allows to sort the endpoints. Predicate 3 tells whether an endpoint lies above or below a segment. Predicate 4 provides the order of two segments along a vertical line passing through an endpoint. Predicate 5 checks whether two segments intersect within a vertical slab defined by two endpoints.

We do not specify a precise representation for the segments, which may depend on the application. For example, the function associated to a segment may be given explicitly together with the interval where it is defined, or the function may be given implicitly as the algebraic function of degree d that interpolates $d+1$ given points (including the endpoints of the segment). Other representations are also possible. The degree of the predicates clearly depends on the chosen representation. In the table below (see also [3, 9]), the degrees of Predicates 2-5 are given for line segments represented by the coordinates of their endpoints, for half-circles defined by centers, radii plus a boolean (to distinguish between the upper and the lower arc), for circle segments defined by three points (including the two endpoints of the segment), and for curves defined by a polynomial equation $y = f(x)$. Observe that the predicates are ordered by increasing degrees in all those cases. However, this order may be different for some other types of segments. For instance, for half-circles defined by three non x-extreme points, the degree of Predicate 2 is 20 while the degree of Predicate 3 is only 4.

Predicate	degree			
	line segment	half circle	circle segment 3 points	polynomial of degree d
1	2	2	12	d
2	1	1	1	1
3	2	2	4	d
4	3	4	12	d
5	3	4	12	d

The table below gives the known lower and upper bounds for reporting segment intersections depending on the predicates that are used.

Predicates	time complexity		reference	
	lower bound	upper bound	lower bound	upper bound
Monotone segments				
1	n^2	n^2	this paper	trivial
1 to 3	$n\sqrt{k} + n\log n$	$n\sqrt{k}\log n$	Boiss.-Snoey. [3]	this paper
1 to 4	$n\sqrt{k} + n\log n$	$n\sqrt{k}\log n$	this paper	this paper
1 to 5	$n\log n + k$	$n\log n + k$	Chaz.-Edelsb. [6]	Balaban [1]
Monotone red-blue segments				
1 to 3	$n\log n + k$	$(n+k)\log n$		this paper
1 to 5	$n\log n + k$	$n\log n + k$		e.g. Chan [5]
Monotone pseudo-segments				
1 to 3	$n\sqrt{k} + n\log n$	$n\sqrt{k} + (n+k)\log n$		this paper
1 to 4	$n\log n + k$	$n\log n + k$		Balaban [1]
Monotone red-blue pseudo-segments				
1 to 3	$n\log n + k$	$n\log n + k$		Boiss.-Snoey. [3]

3 Lower bounds

3.1 Lower bound with Predicate 1

If we only use the predicate that checks whether or not two segments intersect, the lower bound on the time complexity of any algorithm that reports the intersecting pairs is $\Omega(n^2)$. Indeed, consider figure 1 where we have two sets E_1 and E_2 of $\frac{n}{2}$ segments each; the number of intersecting pairs among the segments of E_1 is k_1, and the number of intersecting pairs among the segments of E_2 is k_2. We can give to k_1 and k_2 any value lower than $\frac{n^2}{8}$. Let $k = k_1 + k_2$.

Assume that there is at most one intersection among the pairs of $E_1 \times E_2$. In order to decide whether there are k or $k+1$ intersections, we need to test $|E_1 \times E_2| = \frac{n^2}{4}$ pairs of segments in the worst-case, since it is always possible to arrange the segments in such a way that a non checked pair intersects.

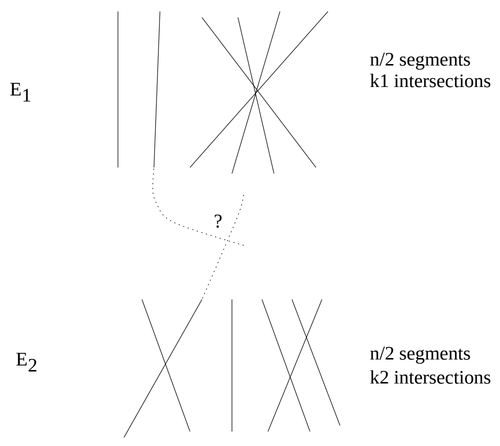

Figure 1: Lower bound with Predicate 1 only.

3.2 Lower bound with Predicates 1 to 4

Boissonnat and Snoeyink [3] have shown that $\Omega(n\sqrt{k} + n \log n)$ is a lower bound if we only use Predicates 1, 2 and 3. Predicate 4 provides the order of the segments along a vertical line passing through an endpoint of some segment. It seems natural to see what is the power of this predicate and if the lower bound on the time complexity can be improved if it is added to the three first predicates. Unfortunately, a slight modification of the proof of Boissonnat and Snoeyink shows that Predicate 4 does not allow to improve the lower bound. Consider figure 2 where we have a first group of m segments on top of the figure, and a second group of n segments on a same line below the segments of the first group.

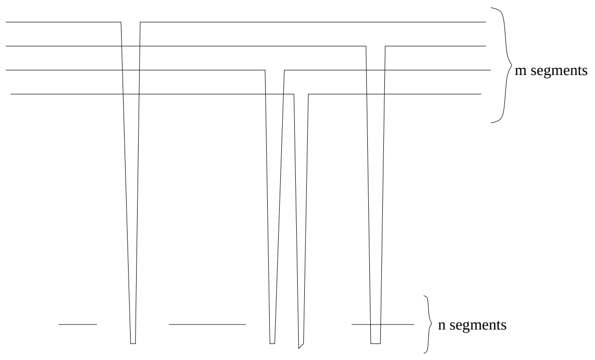

Figure 2: Lower bound with predicates $1 - 4$

Each of the m segments on top of the figure has a very narrow portion that does not intersect any vertical line passing through an endpoint. In order to decide whether or not there are only $\frac{m(m-1)}{2}$ pairs of segments that intersect (i.e. only the segments of the first group), we have to check for intersection, in the worst-case, the $mn = \Omega(n\sqrt{k})$ pairs consisting of a segment of the first group and a segment of the second group. Indeed, it is possible to deform the segments of the first group without modifying their combinatorial arrangement so that a given segment s_1 of this group has its minimum y-coordinate immediately above any segment s_2 of the second group. Whether or not s_1 and s_2 intersect now depends only on those two segments, which makes necessary to check the pair. The time complexity of the problem is therefore $\Omega(n\sqrt{k} + n\log n)$

Theorem 1 *Reporting the intersecting pairs using only predicates 1 to 4 requires $\Omega(n\sqrt{k})$ intersection tests.*

4 Red/blue segments

In this section, we address the special case where we have two sets of segments E_r and E_b (the red and the blue segments) such that no two segments of the same set intersect. In this section, e_b (resp. e_r) denotes a blue (resp. red) segment. We will prove the following result:

Theorem 2 *The red-blue curve segment intersection problem can be solved in $O((n + k)\log n)$ time and $O(n)$ space using predicates* 1, 2 *and* 3.

Notice that the lower bound of section 3 does not hold in the red-blue case : the best known lower bound in this case is still $\Omega(n\log n + k)$. An upper bound that is better than the one above is known for red-blue pseudo-segments [3].

Here we present a new plane sweep algorithm for the red-blue problem. A vertical sweep line moves across the plane from left to right. When it reaches an enpoint, some data structures are updated and intersections can be reported. At any time, we only consider *active* segments which are the segments that cross the sweep line.

Definition 1 *A good intersection (see figure 4) is an intersection between a blue segment e_b and a red segment e_r such that the left endpoint of e_b is above e_r and on the right of the left endpoint of e_r.*

In the following subsections, we describe an algorithm that reports the *good* intersections. The other intersections are reported by the same algorithm after exchanging the orientation of the y-axis or the colors of the segments. The algorithm to be described is therefore applied four times.

Definition 2 *a segment is said to be active when it crosses the sweep-line.*

4.1 A special case

For the sake of clarity, let us first consider a simpler problem where we have one blue segment only. The intersections are reported by calculating the *floor* segment of e_b. This segment, denoted $f(e_b)$, is obtained by pushing e_b downwards as much as possible, provided that $f(e_b)$ keeps the same left endpoint as e_b and does not intersect segments that e_b does not intersect (see figure 3).

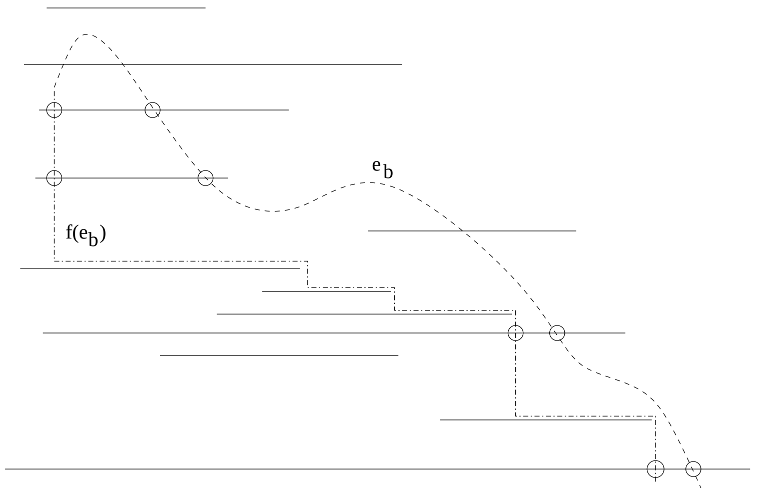

Figure 3: The floor segment (the circles correspond to good intersections)

The floor segment can be calculated using only the first three predicates. First we determine the red segment that is immediately below the left endpoint of e_b. Then we move downwards until we reach a red segment that does not intersect e_b. We then move along this segment until its right endpoint. This procedure is repeated until we encounter the right most red endpoint. Note that $f(e_b)$ has the same set of good intersections as e_b. Hence, we just need to report these intersections while constructing the floor segment.

In the general case with several blue segments, it seems difficult to efficiently compute the floor segments as they are defined above. We therefore relax our definition and allow a floor segment $l(e_b)$ to intersect e_b, provided that the intersection has a larger abscissa than the next endpoint to be encountered.

4.2 Data structures

The idea is to report the good intersections while pushing the blue segments downwards. Each blue segment is stored in a set $U(e_r)$ for some active red segment e_r. The set of segments in $U(e_r)$ will be maintained during the sweep. It consists of the blue segments that cannot be pushed down because they are blocked either by e_r or by the lowest segment of $U(e_r)$.

The above/below relationship within E_b (resp. E_r) is a partial order, which is a total order when restricted to active segments. We will maintain the ordered list of the active red segments in a balanced search tree just like in Chan's algorithm [5] for red-blue segments.

For each blue segment e_b, we need to consider a red segment $l(e_b)$ that plays the same role as the floor segment in the previous section. However, e_b may intersect $l(e_b)$ and is only required to satisfy the following weaker conditions:

Property 1 *The good intersections of e_b with a red segment that lies above $l(e_b)$ have been reported previously (see figure 6).*

Property 2 *As the plane is swept, $l(e_b)$ decreases with respect to the vertical order of red segments.*

For each active red segment e_r, we denote by $U(e_r)$ the set of all active blue segments e_b such that $l(e_b) = e_r$. It is stored in a data structure that allows union, minimum extraction and suppression in $O(\log n)$ time (e.g. a binomial heap). The underlying order is the vertical order of active red segments along the sweep line. Note that we shall not maintain $l(e_b)$ explicitely, as it is not clear whether we can do it within the same time bounds. Its value can be retrieved in logarithmic time from the heaps $U(e_r)$. For any red segment e_r, the set $U(e_r)$ satisfies the following property:

Property 3 *The minimum of $U(e_r)$ does not intersect e_r. The other segments of $U(e_r)$ can only intersect e_r or any red segment below e_r on the right of the right endpoint of this minimum (see figure 4).*

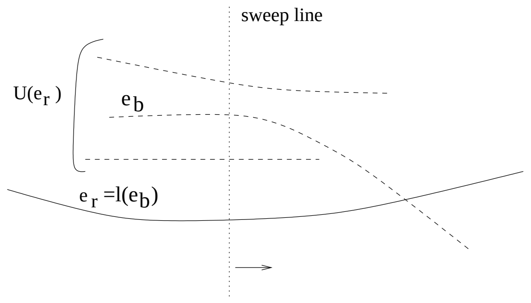

Figure 4: $U(e_r)$ and a good intersection

However, the sets $U(e_r)$ do not have any simple monotonicity property. For example, if e_r is below e'_r, then $U(e_r)$ may be entirely above $U(e'_r)$ or they may be interlaced (see figure 5).

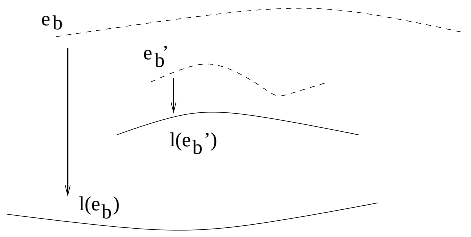

Figure 5:

4.3 Handling the events

We have to consider four kinds of events that occur when the sweep line reaches an endpoint of a segment. Each event corresponds to the insertion, the deletion of a blue or a red segment. We will now explain in detail how these events are handled, and prove that our invariants (properties 1, 2, 3) are maintained.

4.3.1 Inserting a red segment

The new red segment e_r is just inserted in the list of active red segments and the heap $U(e_r)$ is initialized as an empty set. Our invariants are obviously maintained.

4.3.2 Inserting a blue segment

When the sweep line reaches the endpoint of a blue segment e_b, we first determine the red segment e_r that lies just below the endpoint. If e_b does not intersect e_r, then it is inserted in $U(e_r)$. Otherwise, we report this intersecting pair and repeat the same process with the active red segment that is below e_r until we reach a red segment that does intersect e_r .

This procedure can be written in the following way, where e_{r-1} denotes the active red segment that lies immediately below e_r.

```
**Insert**($e_b, e_r$)
While $e_b$ intersects $e_r$
        report the intersection
        $e_r \leftarrow e_r - 1$
Insert $e_b$ in $U(e_r)$
```

The intersections between e_b and the red segments that are below the left endpoint of e_b and above $l(e_b)$ have been reported, therefore property 1 is true for e_b. Besides, it is still valid for the other blue segments.

$l(e)$ remained the same for each active blue segment e then property 2 has been maintained.

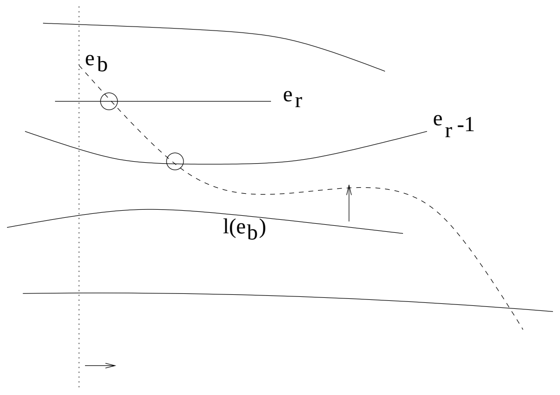

Figure 6: Insertion of e_b

Finally, let us check property 3. If e_b is the new minimum of $U(e_r)$ where $e_r = l(e_b)$, then of course it does not cross $l(e_b)$. Moreover, the other segments of $U(e_r)$ may only cross $U(e_r)$ on the right of the sweep line, therefore they can not cross it before the right endpoint of e_b is reached since e_b lies between these segments and e_r. The case where e_b is not the new minimum of $U(e_r)$ is trivial.

4.3.3 Removing a red segment

Now suppose the sweep line reaches the right endpoint of a red segment e_r. m denotes the minimum of $U(e_r)$. If m intersects $e_r - 1$, then report the intersection, try again with $e_r - 2$ ... until a segment $e_r - j$ is found that does not intersect m. Then m is moved from $U(e_r)$ to $U(e_r - j)$, and we start again with the new minimum of $U(e_r)$. Otherwise, if m does not intersect e_r, we simply merge $U(e_r)$ and $U(e_r - 1)$ thus maintaining property 3.

In this algorithm, **Insert** is the procedure we described in the previous section.

```
While $m = \min(U(e_r))$ intersects $e_r - 1$
    report the intersection
    extract $m$ from $U(e_r)$
    **Insert**($m$,$e_r - 2$)
$U(e_r - 1) = U(e_r - 1) \cup U(e_r)$
```

Bad intersections may be reported during this operation, we may just discard them by comparing the coordinates of the left endpoints.

Note that $l(e_b)$ is not updated after merging $U(e_r - 1)$ and $U(e_r)$ in order to keep our time complexity bound.

The invariants are maintained as is easily checked as in the previous case.

4.3.4 Removing a blue segment

Let e_b be the segment to be removed and $e_r = l(e_b)$. As we said before, $l(e_b)$ can be obtained in $O(\log n)$ time from the mergeable heap data structure that stores the sets $U(.)$.

If e_b is not the minimum of $U(e_r)$, we just remove it. Otherwise, we still remove e_b from $U(e_r)$, but then we need to check whether the new minimum m of $U(e_r)$ intersects e_r. If it does, we push it downwards by running the procedure **Insert**$(m,e_r - 1)$, then repeat the whole process with the new minimum until it does not intersect e_r, so that property 3 remains true.

```
extract $e_b$ from $U(e_r)$
while $m = \min(U(e_r))$ instersects $e_r$
    report this intersection
    extract $m$ from $U(e_r)$
    **Insert**$(m,e_r - 1)$
```

Once again, this procedure may report bad intersection which can be fixed by a simple test.

It can be easily checked that the invariants are maintained.

4.4 Proof

The correctness of this algorithm directly follows from properties 1 and 3.

4.5 Analysis

Maintaining the ordered red segments list takes $O(n \log n)$ time. Localizing an endpoint or removing it takes $O(\log n)$. The other parts of the algorithm take at most $O(\log n)$ for each event or reported intersection, then we just need to prove that an intersection cannot be reported twice. Let e_r and e_b be two intersecting segments. After reporting their intersection once, $l(e_b) < e_r$. Moreover, property 2 means that $l(e_b)$ can only go down the list of red active segments, and we never test the intersection between e_b and a segment that lies above $l(e_b)$.

4.6 Degenerate cases

Since this algorithm is elementary we only need to consider a few degenerate cases which turn out to be very easy to handle. Two kinds of degeneracy may occur: either two endpoints have the same abcissa or an endpoint lies on a curve.

The first case can be solved by extending the partial order on endpoints abcissae to any total order. The order should be the same for each one of the four plane sweeps we perform for otherwise some intersections would never be good.

If a point lies on a segment, we just consider that it is above the segment during two sweeps and below the segment during the two sweeps with the y-axis reversed.

5 The case of general monotone segments

5.1 Grouping the segments

We present a grouping technique that will be useful in this section and in the next one.

Theorem 3 *Let E be a set of n monotone segments and let k be the number of intersecting pairs. It is possible to subdivide E into p subsets $E_1, E_2, ...E_p$ such that $p = O(\sqrt{k})$ and such that, for all i, the segments in E_i are disjoint. This partition can be computed in $O((n+k)\log n)$ time using only Predicates 1, 2 and 3.*

In a first step, we apply a variant of Bentley-Ottman's algorithm [2] to set E. Each time an intersection is detected, one of the intersecting segments is removed. More precisely, when the sweeping line encounters the j-th endpoint, two cases may occur. If the endpoint is a left endpoint e_l of some segment s, we check whether s intersects the segments immediately above or below e_l. In such a case, s is removed and we proceed to the next endpoint. When the j-th endpoint is a right endpoint e_r, we check whether the segments immediately above and immediately below e_r intersect. In such a case, one of the two intersecting segments is removed and we process the j-th endpoint. Otherwise, we directly proceed to the $(j+1)$-th endpoint. At the end of the sweep, the remaining segments are stored in E_1. We then repeat the procedure for $E \setminus E_1$ and the subsequent subsets $E \setminus \cup_{i=1}^{j} E_j$ for $j = 2, \ldots, p-1$. Clearly we have :

Property 4 *$\forall i < j$ and $\forall e_j \in E_j$, there exists $e_i \in E_i$ such that e_i and e_j intersect.*

As a consequence, $k \geq \frac{(p-1)(p-2)}{2}$ and therefore $p = O(\sqrt{k})$.

Constructing E_i $(i = 1, \ldots, n)$ takes $O(k_{i-1} \log k_{i-1} + k_i)$ time where k_i $(i > 0)$ is the number of reported intersections during the construction and $k_0 = n$. Indeed, inserting or removing an endpoint in a sorted list can be done in $O(\log n)$ time, and the cost of processing an intersection can be charged to the number of intersections that are detected.

5.2 The general case

The general case can be solved by running the red-blue algorithm on every pair of sets (E_i, E_j). A straightforward analysis yields:

Corollary 1 *Given n monotone segments, one can report the k pairs that intersect in $O(n\sqrt{k}\log n)$ time and $O(n)$ space. It uses only Predicates $1, 2$ and 3.*

6 Intersections of pseudo-segments

The result of the previous section can be slightly improved when the segments are pseudo-segments.

Theorem 4 *Among a set of n pseudo-segments, the k intersecting pairs can be reported in $O((k+n)\log n + n\sqrt{k})$ using only Predicates 1, 2 and 3.*

The first part of the algorithm is the construction of the subdivision of Section 5.1 that constructs p subsets of non intersecting segments. The second part is an extension of an algorithm by Chan [5] that solves the red-blue case. We adapt this algorithm so that segments with p colors can be handled.

The algorithm is a sweep line algorithm. It uses witnesses as introduced in [3]. If two segments s_1 and s_2 intersect, and if the left endpoint of s_1 is below s_2, their intersection is asserted by a point t satisfying $s_2 \leq t$ and $t \leq s_1$; t is called a *witness* for the intersection (see figure 7). Notice that two intersecting pseudo-segments always have a witness.

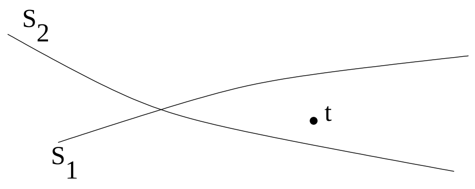

Figure 7: t is a witness of the intersection.

6.1 Searching a non sorted list

During the sweep, the active segments are inserted in a list L. L is sorted according to an order relation, noted $<$, that is compatible with the witnesses that have already been encountered by the sweeping line. In general, this order is different from the true order along the sweeping line. More precisely, we will require that the following property holds through the course of the algorithm. $\prec$ represents the actual order of the segments along the sweeping line.

Property 5 *Let $L = [O_1, O_2, O_3, ..., O_{n-1}, O_n]$ be stored in a balanced binary tree. Assume that O_1 and O_n are respectively minimum and maximum for $\prec$. If $O \notin L$, when searching O in L according to $\prec$, we get a pair (O_i, O_{i+1}) such that $O_i \prec O \prec O_{i+1}$ in $O(\log n)$ time.*

6.2 Data structure

During the sweep, we maintain p sorted lists of active segments, one for each set E_i produced in the first step. These lists are represented as binary search trees. We also maintain a sorted list L. L contains all the active segments plus two elements, $+\infty$ and $-\infty$, which are respectively maximal and minimal for the two orders and do not intersect any segment. Lastly, for any segment $e_i \in E_i$ and for all $j \neq i$, we maintain the segments $U(e_i, j)$ and $L(e_i, j)$ of E_j that are immediately above and below e_i in the $<$ order.

6.3 Insertion of a pseudo-segment

The only non-trivial part of the algorithm is the updating of L. Let x be a left endpoint of the segment to be inserted. First, thanks to Property 5, we locate x in L. If k_x denotes the number of pairs of intersecting segments that have x as their first witness, we can report these k_x intersections and update L in $O(k_x \log n + p)$ time as follows.

Let s_d and s_u be the two segments immediately below and above x that are returned when searching L. Let L_{ud} be the set of segments s such that $s_u < s$ and $s \prec x$, and let L_{du} be the set of segments s such that $s < s_d$ and $x \prec s$ (see Figure 8). For each i, the segments of $L_{ud} \cap E_i$ can be found in $O(1 + |L_{ud} \cap E_i|)$ time by walking along E_i starting at $U(s_u, i)$. Hence, we can compute L_{ud} in $O(p + |L_{ud}|)$ time.

Let $m = \min L_{du}$ and $M = \max L_{ud}$, and let n_x be the number of segments between m and M in L. L must be reordered on the interval $[m, M]$ according to the new order induced by x. We have $s_1 < s_2$ if and only if $s_1 \prec x \prec s_2$ or if we had $s_1 < s_2$ just before processing x. Sorting (according to this comparison rule) the n_x segments in the range $[m, M]$ can be done in $O(n_x \log n_x)$ time.

Let us bound n_x. We observe that any segment of L_{ud} intersects s_u and that x is the first witness of all these intersections. Moreover, any segment of $[m, M]$ that is above s_u and not in L_{ud} intersects M and x is the first witness of all these intersections. Hence, $n_x = O(k_x)$. It follows that all the sortings take $O(k \log n)$ time in total and that the computation of all the L_{ud} and L_{du} takes $O(np + k) = O(n\sqrt{k})$ time.

Once L_{ud} and L_{du} have been computed, each new intersection can be detected in constant time. This holds for the pairs of $L_{ud} \times L_{du}$ and the pairs with a segment s in L_{ud} and a segment that was initially between s and x and remains above x. Each time an intersection is reported, we update the pointers $U(s, j)$ in constant time. Then we can insert the left endpoint of x in the data structure in $O(p) = O(\sqrt{k})$ time.

6.4 Removal of a pseudo-segment

This kind of event occurs when the sweeping line reaches a right endpoint. It is handled in a way very similar to the previous one and does not offer any additional difficulty.

6.5 Correctness of the algorithm

By construction, the order of L that is computed when inserting x is compatible with x, i.e. all the active segments such that $s_1 \prec x \prec s_2$ satisfy $s_1 < s_2$. Moreover, the order of these segments will change in the sequel if and only if we insert an endpoint x' that satisfies $s_2 \prec x' \prec s_1$. The invariant of the algorithm is therefore maintained and, since each pair

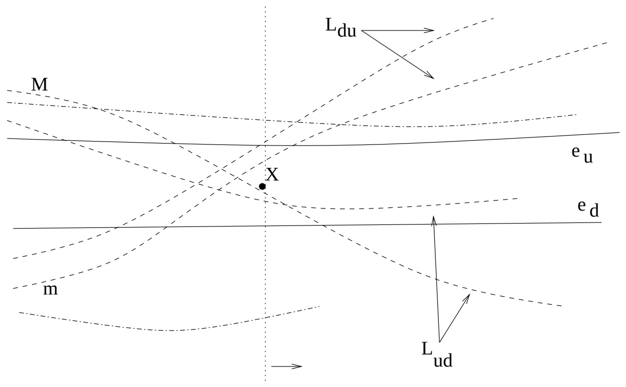

Figure 8: Updating L

of intersecting pseudo-segments has a witness, the algorithm will report all the intersecting pairs of segments.

7 Conclusion

Given a set of general monotone segments, we have presented algorithms to report the pairs of segments that intersect. Our algorithms use only three simple predicates that allow to decide if two segments intersect, if a point is left or right to another point, and if a point is above, below or on a segment. These three predicates seem to be the simplest predicates that lead to subquadratic algorithms. Our algorithms are almost optimal in this restricted model of computation.

We conclude with some open problems. First, can we remove a $\log n$ factor in our time complexity results ?

The $\Omega(n\sqrt{k}+n\log n)$ lower bound holds for general curve segments. It has been possible to do better for line segments [3]. Can we also do better for other special curve segments such as circle segments. Some preliminary results for thick objects have been obtained by Vigneron [15].

In this paper, we have restricted our attention to monotone segments. This may be a restriction when the points with a vertical tangent are difficult to compute. This is in particular the case for circles defined by three points where the predicate that compares x-extreme points has degree 20 while the intersection predicate has degree 12 only.

References

[1] Ivan J. Balaban. An optimal algorithm for finding segment intersections. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 211–219, 1995.

[2] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28(9):643–647, September 1979.

[3] J.-D. Boissonnat and J. Snoeyink. Efficient algorithms for line and curve segment intersection using restricted predicates. In *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 370–379, 1999.

[4] Jean-Daniel Boissonnat and Franco P. Preparata. Robust plane sweep for intersecting segments. *SIAM J. Comput.* to appear.

[5] T. M. Chan. A simple trapezoid sweep algorithm for reporting red/blue segment intersections. In *Proc. 6th Canad. Conf. Comput. Geom.*, pages 263–268, 1994.

[6] Bernard Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39(1):1–54, 1992.

[7] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[8] F. d'Amore, P. G. Franciosa, and G. Liotta. A robust region approach to the computation of geometric graphs. In G. Bilardi, G. F. Italiano, A. Pietracaprina, and G. Pucci, editors, *Algorithms – ESA '98*, volume 1461 of *Lecture Notes Comput. Sci.*, pages 175–186. Springer-Verlag, 1998.

[9] Olivier Devillers, Alexandra Fronville, Bernard Mourrain, and Monique Teillaud. Exact predicates for circle arcs arrangements. Rapport de recherche 3826, INRIA, 1999.

[10] Donald E. Knuth. *Axioms and Hulls*, volume 606 of *Lecture Notes Comput. Sci.* Springer-Verlag, Heidelberg, Germany, 1992.

[11] Giuseppe Liotta, Franco P. Preparata, and Roberto Tamassia. Robust proximity queries: An illustration of degree-driven algorithm design. *SIAM J. Comput.*, 28(3):864–889, 1998.

[12] Sylvain Pion. Interval arithmetic: an efficient implementation and an application to computational geometry. In *Workshop on Applications of Interval Analysis to systems and Control*, pages 99–110, 1999.

[13] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 3rd edition, October 1990.

[14] S. Schirra. Robustness issues. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, Elsevier Science Publishers B.V. North-Holland, Amsterdam, 1999.

[15] Antoine Vigneron. Algorithmes élémentaires pour reporter les intersections d'objets courbes. Rapport de DEA algorithmique, Paris, France, 1999.

[16] C. Yap. Towards exact geometric computation. *Comput. Geom. Theory Appl.*, 7(1):3–23, 1997.

[17] C. K. Yap. Robust geometric computation. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 35, pages 653–668. CRC Press LLC, Boca Raton, FL, 1997.

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, B.P. 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399