Chapter 3
On Degeneracy in Geometric Computations*

Christoph Burnikel[†] Kurt Mehlhorn[‡] Stefan Schirra[†]

Abstract
The main goal of this paper is to argue against the belief that perturbation is a "theoretical paradise" and to put forward the claim that it is simpler (in terms of programming effort) and more efficient (in terms of running time) to avoid the perturbation technique and to deal directly with degenerate inputs. We substantiate our claim on two basic problems in computational geometry, the line segment intersection problem and the convex hull problem.

1 Introduction.

Following Emiris and Canny [EC92] we view a geometric problem P as a function from $\mathbb{R}^{nd}$ to $S \times \mathbb{R}^m$, where n, m, and d are integers and S is some discrete space, modelling the symbolic part of the output (e.g., a planar graph or a face incidence lattice). A problem instance $x \in \mathbb{R}^{nd}$, which for concreteness we view as n points in d–dimensional space, is called *degenerate* if P is discontinuous at x. For example, if $d = 2$ and $P(x)$ is the Voronoi diagram of x, i.e., a straight–line planar graph together with coordinates for its vertices, then x is degenerate iff x contains four cocircular points. A geometric algorithm can be viewed as a decision tree where the decision nodes test the sign (+, −, or 0) of some function (usually a low degree polynomial) of the input variables. An instance x is called *degenerate with respect to some algorithm* A if the computation of A on input x contains a test with outcome zero. Clearly[1], if A solves P and x is a degenerate problem instance then x is also degenerate for A. Typical cases of degeneracy are four cocircular points, three collinear points, or two points with the same x–coordinate. Degeneracy is considered to be a *curse* in geometric computations. It is common belief that the requirement to handle degenerate inputs leads to difficult and boring case analyses, complicates algorithms, and produces cluttered code.

Fortunately, there is a general technique for coping with degeneracies: the *perturbation technique* introduced by Edelsbrunner and Mücke [EM90] and later refined by Yap [Yap90] and Emiris and Canny [EC92]. In this technique, the input is perturbed symbolically, e.g., Emiris and Canny propose to replace the j–th coordinate x_{ij} of the i–th input point by $x_{ij} + \varepsilon \cdot i^j$, where ε is a positive infinitesimal, and the computation[2] is carried out on the perturbed input (all intermediate results are now polynomials in ε). It can be shown that the Emiris and Canny scheme removes many geometric degeneracies, e.g., collinearity of three points, at only a constant factor increase in running time. The same statement holds for the other perturbation schemes, although with a larger constant of proportionality. So perturbation seems to be the perfect solution for the problem of degeneracy. As Yap [Yap90] puts it: the perturbation technique is "the theoretical paradise in which degeneracies are abolished".

The main goal of this paper is to argue against this belief and to put forward the claim that it is simpler (in terms of programming effort) and more efficient (in terms of running time) to avoid the perturbation technique and to deal directly with

*This work was supported by the ESPRIT Basic Research Actions Program under contract No. 7141 (project Alcom II) and by the BMFT (Förderungskennzeichen ITS 9103).

[†]Max–Planck–Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany

[‡]Max–Planck–Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany and Fachbereich Informatik, Universität des Saarlandes

[1]this assumes all tests to be continuous functions of the inputs

[2]throughout this paper we assume that all computations are carried out exactly.

degenerate inputs. Our argument rests on the following three observations:

- On degenerate inputs the perturbation schemes may incur an arbitrary overhead in running time.
- The complexity of the postprocessing required to retrieve the answer $P(x)$ for a degenerate input x from the answer $P(x(\varepsilon))$ to the perturbed input $x(\varepsilon)$ is significant.
- For many geometric problems algorithms handling degeneracies directly are only moderately more complex than algorithms assuming non–degenerate inputs.

Before we substantiate these observations we want to remark that degenerate inputs arise frequently in practice. For example, 100 points with integer coordinates between 0 and 1023, i.e., 100 pixels on a standard screen, are very likely to contain three collinear points.

To substantiate the first claim let us consider two basic problems in computational geometry, the line segment intersection problem and the convex hull problem.

In the *line segment intersection problem* the input consists of n line segments in the plane, i.e., a point in $\mathbb{R}^{4n}$, and the output is the planar graph whose vertices are the endpoints and the crossings of the segments and whose edges are the subsegments induced by the vertices. Alternatively, we may want to compute a triangulation of this planar graph or the related trapezoidal diagram. We use m to denote the number of vertices of this planar graph and s to denote the number of pairs of intersecting segments. Note that s might be as large as m^2, e.g., if all segments pass through the origin. The perturbation technique yields running time $O((n+s)\log n)$ when combined with Bentley–Ottmann plane–sweep, and time $O(s + n\log n)$ when combined with the randomized incremental algorithms of Clarkson and Shor, Mulmuley, Boissonnat et al, or Seidel [CS89,Mul89,BDS+92, Sei91], or the optimal deterministic algorithm of Chazelle and Edelsbrunner [CE92]. On the other hand there are variants of these algorithms handling degeneracies directly and running in time $O(m\log n)$ and $O(m+n\log n)$ respectively. This is folklore for the plane sweep algorithm, was shown by Seidel [Sei91] for the randomized incremental algorithm, and is a new result for the optimal deterministic algorithm, cf. Section 3.

In the *convex hull problem*, the input is a multiset S of n points in $\mathbb{R}^d$ and the output is their convex hull, e.g., represented through its set of facets and their incidence graph. Consider first an instance which consists of n copies of the same point, say $x_{ij} = 0$ for all i and j with $1 \le i \le n$ and $1 \le j \le n$. The Emiris and Canny perturbation scheme changes these coordinates into $x_{ij} = \varepsilon \cdot i^j$, i.e., the perturbed input consists of n points on the moment curve [Ede87, Section 6.2.1]. The resulting hull is a monster; it has $n^{\lfloor d/2 \rfloor}$ facets and it takes at least that long to compute it. The algorithm of this paper runs in time $O(n)$. We admit that a simple preprocessing step will catch this input. The following example escapes simple preprocessing techniques. Assume that the input consists of the corners of a triangle and $n-3$ additional points which lie on one of the sides of the triangle. Perturbation may turn the convex hull into an n-gon. Perturbation combined with any of the standard hull algorithms (Graham scan, plane sweep, divide-and-conquer, randomized incremental) leads to a running time of $\Omega(n\log n)$. The algorithm of this paper runs in time $O(n)$. More generally, we will show in Section 2 that the running time of our algorithm is determined by the structural complexity of the convex hull of S (and random subsets of S). Algorithms with *structure–sensitive* running time were previously only known for non–degenerate inputs: gift-wrapping [CK70], the algorithm of Seidel [Sei86], and the randomized incremental algorithms [CS89,BDS+92,CMS93] work in time $|F|\cdot n$, $|F|\cdot\log n$, and expected time $|F|$ respectively, where F is the number of facets of conv S. Our algorithm is a modification of [CMS93].

Let us now turn to our second claim: perturbation requires non–trivial postprocessing. Let x be a problem instance of a geometric problem P. Perturbation computes $P(x(\varepsilon))$ instead of $P(x)$ where ε is an infinitesimal. For degenerate inputs x the mapping $x \mapsto P(x)$ is *non–continuous* at x and therefore we can in general not obtain $P(x)$ as the limit of $P(x(\varepsilon))$ as ε goes to zero. Rather we should expect some non–trivial changes in the symbolic part of the output. Let us be more concrete. In the line segment intersection problem we may have the endpoint of some segment lying in the relative interior of some other segment. Perturbation may remove the point of intersection. This intersection is hard to retrieve from the perturbed output. Hard here means hard compared to the simplicity of the plane sweep intersection algorithm. In the case of the convex hull problem the limit process is also non–trivial, one needs to remove empty facets and identify some ridges. A small variation of the problem – compute all points contained in the boundary of the hull – makes the limit process very difficult. In summary, we may state that the perturbation technique entails complex postprocessing.

With respect to the third claim we observe that the algorithms presented in this paper are modifications of algorithms designed for non–degenerate inputs. We can also offer some experimental evidence. The LEDA–implementation [Näh93b, Näh93a] of the plane sweep algorithm for line segment intersection has 581 lines of code out of which about 100 deal with degeneracies (vertical segments and high–degree intersections). For our convex hull algorithm (without deletion) the LEDA-implementation by Michael Müller and Joachim Ziegler [MZ93] has 1124 lines of code. Out of this 134 lines deal with degeneracies.

2 Convex Hull Maintenance.

We show how to maintain convex hulls in arbitrary dimension without any non–degeneracy assumption. Our solution is a minor[3] modification of the randomized incremental algorithm of Clarkson, Mehlhorn, and Seidel [CMS93]. Its expected running time is structure–sensitive.

We will first describe our data structure for maintaining convex hulls. Insertions and deletions are then treated in Sections 2.1 and 2.2 respectively.

Let $R = \{x_1, \ldots, x_n\}$ be the multi–set of points whose convex hull has to be maintained and let $\pi = x_1 \ldots x_n$ be the insertion order. Let $\pi_i = x_1 \ldots x_i$, $R_i = \{x_1, \ldots, x_i\}$ and let conv R_i be the convex hull of the points in R_i. Let $d = \dim R$ be the dimension of the convex hull of R and let $DJ = \{x_{j_1}, x_{j_2}, \ldots, x_{j_{d+1}}\}$ with $1 \leq j_1 \leq \ldots \leq j_d \leq n$ be the set of dimension jumps where x_k is called a *dimension jump* if $\dim R_{k-1} < \dim R_k$. Clearly, $j_1 = 1$. In the incremental construction of conv R we maintain a triangulation $\Delta(\pi_i)$ of conv R_i: a simplical complex whose union is conv R_i (a simplical complex is a collection of simplices such that the intersection of any two is a face of both). The vertices of the simplices in $\Delta(\pi_i)$ are points in R_i. The triangulation $\Delta(\pi_i)$ induces a triangulation $\mathrm{CH}(\pi_i)$ of the boundary of conv R_i: it consists of all facets of $\Delta(\pi_i)$ which are incident to only one simplex of $\Delta(\pi_i)$. If $x \in \mathrm{aff}\, R_i$ then a facet F of $\mathrm{CH}(\pi_i)$ is called *visible from* x or x*–visible* (we also say: x can see the facet) if x does not lie in the closed halfspace of aff R_i that is supported by F and contains conv R_i.

The triangulation $\Delta(\pi_1)$ consists of the single simplex $\{x_1\}$. For $i \geq 2$, the triangulation $\Delta(\pi_i)$ is obtained from $\Delta(\pi_{i-1})$ as follows. If x_i is a dimension jump, i.e., $x_i \notin \mathrm{aff}\, R_{i-1}$, then x_i is added to the vertex set of every simplex of $\Delta(\pi_{i-1})$. If x_i is not a dimension jump then a simplex $S(F \cup \{x_i\}) = \mathrm{conv}(F \cup \{x_i\})$ is added for every x_i–visible facet of $\mathrm{CH}(\pi_{i-1})$. Figure 1 gives an example. For a simplex S let vert(S) denote the set of vertices that define this simplex. It is clear that $\Delta(\pi)$ contains a simplex whose vertex set is precisely the set of dimension jumps. We call this simplex the *origin simplex* of $\Delta(\pi)$. For every simplex S (besides the origin simplex) we call the vertex in $\mathrm{vert}(S) - DJ$, that has been inserted last, the *peak* of S and the facet of S opposite to the peak the *base facet* of S.

The triangulation $\Delta(\pi)$ does not change if the dimension jumps are inserted first. More precisely, for $T \subset R$ let $\pi \setminus T$ denote the insertion order with the elements in T removed, e.g., $\pi \setminus \{x_i\} = x_1 \ldots x_{i-1} x_{i+1} \ldots x_n$.

LEMMA 2.1. *Let* $DJ = \{x_{j_1}, \ldots, x_{j_{d+1}}\}$ *be the set of dimension jumps,* $1 = j_1 < j_2 < \cdots < j_{d+1} \leq n$. *Then*

$$\Delta(\pi) = \Delta(x_{j_1} x_{j_2} \ldots x_{j_{d+1}} \pi \setminus DJ).$$

[3]Recall that one of our claims is precisely that many algorithms require only minor modifications

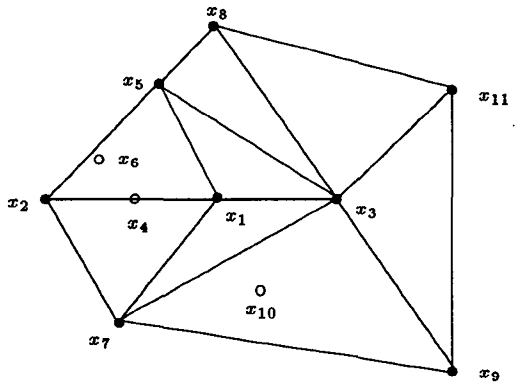

Figure 1: A triangulation. The dimension jumps are points x_1, x_2, and x_5.

This is an immediate consequence of

Lemma 2.2. *Let y be a dimension jump, i.e., $y \notin \operatorname{aff} R_k$. Then*

$$\Delta(\pi_{k-1}x_k y) = \Delta(\pi_{k-1} y x_k).$$

The triangulation $\mathrm{CH}(\pi)$ induces a triangulation of every facet F of the convex hull of R. This triangulation depends only on the insertion order of the points in $R \cap \operatorname{aff} F$.

Lemma 2.3. *Let F be a facet of* $\operatorname{conv} R$ *and let $R^F = \{x \in R; x \in \operatorname{aff} F\}$. Furthermore, let $\pi^F = \pi \setminus (R - R^F)$. Then*

$$\Delta(\pi^F) = \{G \in \mathrm{CH}(\pi); G \in \operatorname{aff} F\}.$$

It is convenient to extend $\Delta(\pi)$ to a triangulation $\overline{\Delta}(\pi)$ by also making the facets of $\mathrm{CH}(\pi)$ the base facet of some simplex: $\overline{\Delta}(\pi)$ is obtained from $\Delta(\pi)$ by adding a simplex $S(F \cup \{\overline{O}\})$ with base facet F and peak $\overline{O}$ for every facet F of $\mathrm{CH}(\pi)$. Here $\overline{O}$ is a fictitious point without geometric meaning. We propose to store $\overline{\Delta}(\pi)$ as the set of its simplices together with some additional information: For each simplex $S \in \overline{\Delta}(\pi)$ we store its set of vertices, the equation of its base facet normalized such that the peak lies in the positive halfspace, and for each simplex S and vertex $x \in \mathrm{vert}(S)$ we store the other simplex sharing the facet with vertex set $\mathrm{vert}(S) \setminus \{x\}$. We also store a pointer to the origin simplex and a suitable representation of $\operatorname{aff} R$, e.g., a maximal set of affinely independent points. The simplical complex $\overline{\Delta}(\pi_1)$ consists of two simplices: the bounded[4] simplex $S(\{x_1\})$ and the unbounded simplex $S(\{\overline{O}\})$.

2.1 Insertions.

We give additional details of the insertion process. Consider the addition of the i-th point $x = x_i$, $i \geq 2$. First decide whether x is a dimension jump (an $O(d^3)$ test). If x is a dimension jump then add x_i to $|(S)$ for every simplex of $\overline{\Delta}(\pi_{i-1})$ and add the simplex $S(F \cup \{\overline{0}\})$ to $\overline{\Delta}(\pi_i)$ for every bounded simplex F of $\overline{\Delta}(\pi_{i-1})$.

Lemma 2.4. *If x_i is a dimension jump then*

$$|\overline{\Delta}(\pi_i)| = |\overline{\Delta}(\pi_{i-1})| + |\Delta(\pi_{i-1})|$$

and $\overline{\Delta}(\pi_i)$ can be constructed from $\overline{\Delta}(\pi_{i-1})$ in time $O(d^3|\overline{\Delta}(\pi_{i-1})|)$.

If x_i is not a dimension jump then we proceed as described in [CMS93]. We first compute all x_i–visible facets F of $\mathrm{CH}(\pi_{i-1})$ and then update the extended triangulation $\overline{\Delta}$ as follows: For each x_i–visible facet F of $\mathrm{CH}(\pi_{i-1})$ ($\equiv$ x_i–visible base facet of an unbounded simplex in $\overline{\Delta}(\pi_{i-1})$) we alter the simplex $S(F \cup \{\overline{O}\})$ of $\overline{\Delta}(\pi_{i-1})$ into $S(F \cup \{x_i\})$. Moreover, for each new hull facet $F \in \mathrm{CH}(\pi_i) \setminus \mathrm{CH}(\pi_{i-1})$ we add the unbounded simplex $S(F \cup \{\overline{O}\})$. In other words, for each horizon ridge f of $\mathrm{CH}(\pi_{i-1})$, i.e., ridge where exactly one of the incident facets is x_i–visible, we add the simplex $S(f \cup \{x_i, \overline{O}\})$. The set of x_i–visible facets F of $\mathrm{CH}(\pi_{i-1})$ can be found by visiting simplices according to the rule: Starting at the origin simplex visit any neighbor of a visited simplex that has an x_i–visible base facet.

Lemma 2.5. *If x_i is not a dimension jump then the time to add x_i is $O(d^2)$ times the number of simplices in $\overline{\Delta}(\pi_{i-1})$ with x_i–visible base facet plus $O(d^3)$ times the number of altered and new simplices.*

We next turn to the analysis of the insertion algorithm. Let $\overline{T}(R)$ ($T(R)$) denote the expected[5] number of (bounded) simplices in the extended triangulation of R and let $U(R)$ denote the expected

[4] A simplex is called *bounded* if $\overline{O}$ does not belong to its vertex set and it is called *unbounded* otherwise.

[5] all expectations are computed with respect to the $n!$ insertion orders

number of simplices incident to the point inserted first. For $R' \subseteq R$ and $x \in R'$ let $H(R', x)$ be the set of horizon ridges of conv $R' \setminus x$ with respect to x, i.e., the set of ridges of conv $R' \setminus x$ with exactly one incident x-visible facet. Finally, say that a set R is in *quasi-general position* if any d points of R have dimension $d-1$ where $d = \dim R$ (however, a hyperplane may contain an arbitrary number of points of R).

LEMMA 2.6. *Let $n = |R|$ and $d = \dim R$.*

1. *If* $\dim R = 0$ *then* $\overline{T}(R) = 2$ *and* $T(R) = U(R) = 1$.

2. *If* $\dim R > 1$ *then* $T(R) \le \overline{T}(R) \le 2 * 2^d + \sum_{2 \le k \le n} h_k/k$ *, where*

$$h_k = \frac{1}{\binom{n}{k}} \sum_{\substack{R' \subseteq R \\ |R'|=k}} 2^{d-\dim R'} \sum_{x \in R'} \sum_{r \in H(R',x)} T(R' \cap \text{aff } r)$$

3. *The expected time for the n-th insertion is* $O(d^2 h_n/n + d^3 U(R))$.

4. $U(R) =$

$$2^d + \sum_{2 \le k \le n} \frac{1}{k(k-1)\binom{n}{k}} \sum_{\substack{R' \subseteq R \\ |R'|=k}} 2^{d-\dim R'} \cdot$$

$$\cdot \sum_{x \in R'} \sum_{y \in R' \setminus x} \sum_{\substack{r \in H(R',x) \\ \text{with } x \in \text{aff} R'}} U(R' \cap \text{aff } r)$$

5. *If R is in quasi-general position then*

$$h_k/k = \frac{1}{\binom{n}{k}} \sum_{\substack{R' \subseteq R \\ |R'|=k}} 2^{d-\dim R'} \sum_{x \in R'} |H(R', x)|/|R'|$$

and

$$U(R) = 2^d + \sum_{2 \le k \le n} (d-1) h_k/(k(k-1))$$

Lemma 2.6 quantifies the dependence of the running time of our algorithm on the structural properties of the constructed hull. Since the formulae in the lemma are quite daunting we explain it a bit. The expected time for the n-th insertion is $O(d^2 h_n/n + d^3 U(R))$ according to item 4. The first term accounts for the time to construct new hull facets and the second term accounts for the time to search visible hull facets. If R is in quasi-general position then h_k/k is simply the expected size of the horizon for a random subset $R' \subseteq R$ of size k and a random point $x \in R'$ times a factor that accounts for later dimension jumps. Moreover, $U(R)$ is also determined by the h_k's. As a concrete example, assume $h_k/k \le k^\varepsilon$ for some positive ε and all k. Then $\overline{T}(R) = n^{1+\varepsilon}$, $U(R) = n^\varepsilon$ and the expected cost of the k-th insertion is $O(k^\varepsilon)$. If R is not in quasi-general position then the running time depends also on the structural properties of lower dimensional faces.

We close with a comparison with perturbation methods.

LEMMA 2.7. *1. For every n and d there is an example where the algorithm above runs in linear time but the Emiris and Canny scheme produces a hull with* $\Omega(n^{\lfloor d/2 \rfloor})$ *facets.*

2. *For inputs in quasi-general position our algorithm never does more work than our algorithm combined with any perturbation scheme.*

3. *Our algorithm can be viewed as choosing a perturbation on-line. Whenever a point is inserted it is moved by an infinitesimal amount towards the current hull.*

2.2 Deletions.

Deletion of a point x_i means to change $\overline{\Delta}$ such that in effect x_i was never added. In comparison to [CMS93] the main additional difficulty is the correct treatment of dimension jumps.

We store some additional information in the insertion process. Each point $x_k \in R$ is stored in exactly one simplex in $\Delta(\pi_k)$ that contains x_k in its closure. The insertion algorithm gives us such a simplex. Furthermore we store the set of dimension jumps in a list. For a vertex x_l of $\overline{\Delta}(\pi)$ let $S(x_l)$ denote the set of simplices with vertex x_l and let $S_k(x_l)$ be the set of simplices in $S(x_l)$ whose peak index is at most k. Furthermore let $R(x_l)$ denote the set of points stored in the simplices in

$S(x_l)$, and let $P(x_l)$ be the set of vertices that are opposite[6] to x_l.

The global plan is as follows: Let x_i be the point to be deleted. First of all, we check whether $x_i \in DJ$. Then we delete all simplices in $\overline{\Delta}(\pi) \setminus \overline{\Delta}(\pi \setminus \{x_i\})$. Next we basically reinsert the points x_k, $k > i$, thereby creating the simplices in $\overline{\Delta}(\pi \setminus \{x_i\}) \setminus \overline{\Delta}(\pi)$ according to the insertion order. During the reinsertion process different strategies apply to dimension jumps and non-dimension-jumps.

The simplices to be deleted are exactly the simplices in $S(x_i)$.

Lemma 2.8. *Let S be a simplex in $\overline{\Delta}(\pi)$. $S \notin \overline{\Delta}(\pi \setminus \{x_i\})$ if and only if $S \in S(x_i)$.*

If x_i was a dimension jump we have either $\dim(R \setminus \{x_i\}) = \dim R - 1$ or we get a new dimension jump, say x_j. Deletion of x_i reduces dimension if all other points lie in $\operatorname{aff}(DJ \setminus \{x_i\})$. If $P(x_i) = \emptyset$ and $\operatorname{aff}(R(x_i) \setminus \{x_i\}) = \operatorname{aff}(DJ \setminus \{x_i\})$ then x_i reduces dimension. We simply remove all unbounded simplices that do not have x_i in their vertex set and remove x_i from the vertex set of the remaining simplices. Otherwise we get a new dimension jump x_j where

$$j = \min\{k; x_k \in R(x_i) \cup P(x_i), x_k \notin \operatorname{aff}(DJ \setminus \{x_i\})\}.$$

By Lemma 2.1 dimension jumps can be moved to the front of the insertion order. Hence we have

$$\Delta(\pi \setminus \{x_i\}) = \Delta(\sigma\pi \setminus (\{x_i\} \cup DJ))$$

if $x_i \notin DJ$ and

$$\Delta(\pi \setminus \{x_i\}) = \Delta(\sigma'\pi \setminus DJ)$$

if $x_i \in DJ$, where σ and σ' are arbitrary permutations of DJ and $DJ \setminus \{x_i\}$ resp. Hence we can assume $DJ \subset R_i$ wlog. If $x_i \in DJ$ and x_j is a new dimension jump we have by Lemma 2.2

$$\Delta(\pi \setminus \{x_i\}) = \Delta(\sigma'\pi_{i-1}x_j\pi \setminus (R_i \cup DJ \cup \{x_j\}))$$

So we can reinsert x_j first, thereby making sure that all dimension jumps are already inserted, when we reinsert the remaining points. The last paragraph describes reinsertion of a dimension jump.

Let us first consider reinsertion of x_k, where x_k is not a new dimension jump. A facet $F \in \mathrm{CH}(\pi_{k-1} \setminus x_i)$ is called *new* if it is not constructed in $\Delta(\pi_{k-1})$. We have

Lemma 2.9. *Let F be a facet of $\mathrm{CH}(\pi_{k-1} \setminus x_i)$. If $\dim(R_k \setminus \{x_i\}) = \dim(R_{k-1} \setminus \{x_i\})$ we have $S(F, x_k) \in \Delta(\pi_k \setminus \{x_i\}) - \Delta(\pi_k)$ if and only if F is x_k-visible and either F is x_i-visible or F is new and $x_i \in \operatorname{aff} F$.*

Hence it is not necessary to reinsert points that are not in $R(x_i)$. We assume inductively that we have $\Delta(\pi_{k-1} \setminus \{x_i\})$ and $\Delta(\pi \setminus \{x_i\}) \cap \Delta(\pi)$ and the set B_{k-1} of facets of $\mathrm{CH}(\pi_{k-1} \setminus x_i)$, that are x_i-visible or contain x_i in their affine hull and are new. If $x_i \notin DJ$, B_i is the collection of facets opposite to x_i in the simplices in $S_i(x_i)$. Reinsertion of x_k means to add a simplex $S(F, x_k)$ for every x_k-visible facet F of B_{k-1}. The procedure for finding x_k-visible facets of B_{k-1} is analogous to [CMS93].

Let us finally consider reinsertion of a new dimension jump x_j. Since a new dimension jump is reinserted first we can assume $j = i + 1$ wlog. If x_i and x_{i+1} are on the same side of $\operatorname{aff}(DJ \setminus \{x_i\})$, then we take the simplices in $S_i(x_i)$, replace x_i by x_{i+1} and add them to Δ. B_{i+1} is the collection of facets on the boundary of the union of these simplices that are not opposite to x_{i+1}. If x_i and x_{i+1} are on different sides of $\operatorname{aff}(DJ \setminus \{x_i\})$, no simplices are added. Here B_{i+1} is the collection of facets opposite to x_i in the simplices in $S_{i+1}(x_i)$.

3 Line Segment Intersection.

We show how to solve the line segment intersection problem in time $O(m + n \log n)$ extending work of Chazelle and Edelsbrunner [CE92]; n denotes the number of line segments and m denotes the number of vertices of the induced planar graph. Their algorithm runs in time $O(s + n \log n)$ where s is the number of pairs of intersecting segments.

Our algorithm is a modification of the algorithm in [CE92]; so we review their algorithm and indicate the required changes. It computes the so-called web graph $web(N)$ induced by the set N of segments. This is the planar graph induced by N plus additional vertical edges incident to the end-

[6] x and y are *opposite* if there is a facet F of $\overline{\Delta}$ such that $S(F \cup x)$ and $S(F \cup y)$ are simplices of $\overline{\Delta}$.

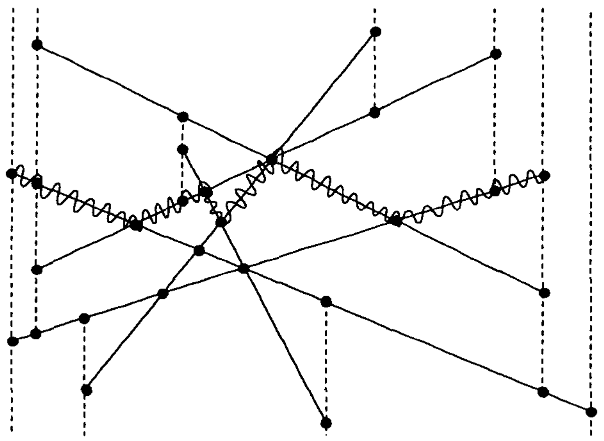

Figure 2: The web graph $web(N)$. The wiggled line indicates a layer

points and extending upwards and downwards to the neighboring segment (or to infinity); cf. Figure 2. The webgraph $web(N)$ is constructed incrementally by inserting so-called *fragments* in a specific order. A fragment is part of an input segment. The fragments are obtained by storing the x-projections of the segments in a segment tree. This will break each segment into $O(\log n)$ fragments. The x-projections of any two fragments are either disjoint or one is contained in the other. The order in which fragments are inserted is determined as follows: if the x-projections of two fragments are disjoint then the one to the left is inserted first, and if the x-projection of some fragment f is properly contained in the projection of f' then f' is inserted before f (fragments with the same projection are inserted in an arbitrary order).

Suppose now that we want to insert fragments starting at the vertical line $x = x_0$. We call this vertical line the *sweep-line*. The insertion order of fragments ensures that $web(N)$ is completely known to the left of the sweep-line. In order to facilitate the insertion of fragments starting at the sweep-line a so-called *sweep-tree* is maintained. The sweep-tree contains the so-called *active layers* of $web(N)$ in sorted order. The decomposition of $web(N)$ into layers is defined as follows: A layer is an x-monotone path through $web(N)$ starting at the left end of some segment and ending at the right end of some segment. When a layer enters a node v of $web(N)$ on the i-th steepest (not counting segments ending at v) fragment incident to v, then it leaves v on the i-th shallowest fragment (not counting segments starting at v); the wiggled line in Figure 2 indicates a layer. A layer is active if it intersects the sweep-line. In the sweep-tree one stores an edge of each active layer; this edge is either completely to the left of the sweep-line or intersects the sweep-line.

The following actions are required when the sweep-line is at x_0.

1. Remove all layers from the sweep-tree which become inactive at x_0. This takes $O(\log n)$ per layer for a total of $O(n \log n)$.

2. Insert all layers starting at x_0 into the sweep-tree. For each layer L starting at x_0 we determine the position of L in the sweep-tree by binary search. Suppose that we need to test whether layer L is above layer L' at $x = x_0 + \epsilon$ for some positive infinitesimal ϵ. When the edge e representing L' in the sweep-tree intersects the sweep-line the comparison is simple. If it does not then we first traverse L' until we reach the sweep-line. In order to assist this traversal we maintain for each vertex v of the web an appropriate bijection between the fragments incident to v from the left (and not ending at v) and the fragments incident to v from the right (and not starting at v). This bijection is constructed when the first layer wants to traverse v. Note that all segments passing through v are known at this time and hence the bijection needs to be computed only once. The total time for computing all bijections is $O(n + m)$ and the total time for inserting layers is $O(n \log n)$.

3. Next we compute for each fragment p starting at the sweep line its so-called companion: *companion*(p) is the fragment p' that is immediately below p at the vertical line $x = x_0 + \varepsilon$ and when p is inserted. Note that the fragments starting at $x = x_0$ are inserted in some order. When a fragment p is inserted there will be a fragment p' immediately below it and that fragment is companion of p. The computation of companions is essentially as described

in [CE92, capsule 12]; high degree vertices require only minor adaptions. The total time for computing companions is $O(n+m)$.

4. We finally insert the fragments starting at the sweep line into the web. Consider any fragment p starting at the sweep line. When we follow p through the web we split regions as we go along. The first region to be split is given by *companion*(p). Whenever a region is left, say through vertex v, the region to be entered is readily determined by inspecting the edges adjacent to p at v. Note that the insertion order guarantees that only segments passing through v have been encountered so far and so the proper association between incoming and outgoing edges is easily maintained. The edges of each region are stored in a finger search tree [HMRT86]. This data structure allows a region to be split in amortized time $O(1)$. Our amortization argument is more complex than the one used in [CE92, capsules 4 and 5] as we cannot use their argument for terminating regions (capsule 5); it is however in the same spirit. The total time for traversing regions is $O(n+m)$.

In summary we have shown how to solve the line segment intersection problem in time $O(m + n\log n)$.

References

[BDS$^+$92] J.D. Boissonnat, O. Devillers, R. Schott, M. Teillaud, and M. Yvinec. Applications of random sampling to on-line algorithms in computational geometry. *Discrete and Computational Geometry*, 8:51–71, 1992.

[CE92] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *Journal of the ACM*, 39:1–54, 1992.

[CK70] D.R. Chand and S.S. Kapur. An algorithm for convex polytopes. *Journal of the ACM*, 17:78–86, 1970.

[CMS93] Kenneth L. Clarkson, Kurt Mehlhorn, and Raimund Seidel. Four results on randomized incremental constructions. *Computational Geometry: Theory and Applications*, 3(4):185–212, 1993.

[CS89] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete and Computational Geometry*, 4:387–421, 1989.

[EC92] Ionnais Emiris and John Canny. An efficient approach to removing geometric degeneracies. In *Proc. of the 8th Symp. on Computational Geometry*, pages 74–82, 1992.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer Berlin-Heidelberg, 1987.

[EM90] H. Edelsbrunner and E.P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graphics*, 9(1):67–104, 1990.

[HMRT86] K. Hoffmann, K. Mehlhorn, P. Rosenstiehl, and R. Tarjan. Sorting Jordan sequences in linear time using level-linked search trees. *Information and Control*, 68:170–184, 1986.

[Mul89] K. Mulmuley. A fast planar partition algorithm II. *Proc. 5th Symp. on Computational Geometry*, pages 33–43, 1989.

[MZ93] M. Müller and J. Ziegler. An implementation of a convex hull algorithm. manuscript, 1993.

[Näh93a] St. Näher. personal communication, 1993.

[Näh93b] St. Näher. *LEDA Manual.* Max-Planck-Institut für Informatik, 1993.

[Sei86] R. Seidel. Constructing higher dimensional convex hulls at logarithmic cost per face. *Proc. of the 18th Symp. on Theory of Computing*, 1986.

[Sei91] R. Seidel. Backwards analysis of randomized geometric algorithms. ALCOM Summerschool on efficient algorithms design, Århus, Denmark, 1991.

[Yap90] C. K. Yap. Symbolic treatment of geometric degeneracies. *J. Symbolic Comput.*, 10:349–370, 1990.