# An Incremental Linear Time Algorithm for Digital Line and Plane Recognition Using a Linear Incremental Feasibility Problem

Lilian Buzer

LLAIC, Université Clermont 1, IUT département Informatique, B.P. 86, 63172
AUBIERE cedex, FRANCE, Buzer@llaic.u-clermont1.fr

**Abstract.** We present a new linear incremental method for digital hyperplane[1] recognition. The first linear incremental algorithm was given for 8-connected planar lines in [DR95]. Our method recognizes any subset of line in the plane or plane in the space. We present the Megiddo linear programming (LP) algorithm in linear time and describe its adaptation to our problem. Then we explain its improvement toward a linear incremental method.

**Keywords:** digital line recognition, digital plane recognition, feasibility problem, incremental, linear time.
**Conference Topic:** Models for Discrete Geometry.
**Type of Presentation:** oral presentation.

## 1 Introduction

We study digital line and plane recognition. The seminal definition of these objects was given by Reveillès in [Rev91]. A set of points $P$ of $\mathbb{Z}^d$ is a digital hyperplane if it verifies:

$$\exists (a_i)_{1\leq i\leq d} \in \mathbb{Z}^{d*}, \exists \gamma \in \mathbb{Z}, \forall x \in P, \text{ we have: } \gamma \leq \sum_{i=1}^{d} a_i.x_i < \gamma + ||a||_\infty \qquad (1)$$

where $||.||_\infty$ denotes infinite norm equal to $Sup\{|a_i|_{1\leq i\leq d}\}$. We present a new algorithm which recognizes any set of points. Our technique is based on LP algorithm, more precisely the linear Megiddo method which is described in Section 2. In Section 3, we transform our recognition problem into a feasibility problem solvable by Megiddo algorithm. The improvement to linear incremental complexity is then presented in Section 4.

[1] By extension of the notion of an euclidian hyperplane in a $d$-dimensional vector-space. We remind that this object refers to a $(d-1)$-dimensional affine subspace.

A. Braquelaire, J.-O. Lachaud, and A. Vialard (Eds.): DGCI 2002, LNCS 2301, pp. 372–381, 2002.

## 2 Megiddo Algorithm in $\mathbb{R}^2$ and $\mathbb{R}^3$

### 2.1 Preliminaries

Our digital line and plane recognition algorithm requires to solve LP problems in two and three dimensions. For this, we will use Megiddo linear-time algorithm.

**History.** Working in the field of computational geometry, Megiddo gave the first deterministic algorithm for LP whose running time is linear in the number of constraints when the dimension is fixed. The decimation technique was first introduced in [Meg83] for the two and three-dimensional cases. Later, in [Meg84] he extended his method to develop an $O(2^{2^d}.n)$ time algorithm for LP in $\mathbb{R}^d$. The factor in $d$ was improved by Clarkson to $3.d^2$ (see [Cla86]). In recent years, no progress has been made on this front, nevertheless new developments occured in randomized and parallel algorithms for linear programming. Numerous simpler and more practical randomized algorithms have been given (see [Cla98,Sei91]). An introduction can be read in [BKOS00] and a comprehensive summary of this field can be found in [AS98]. We will hereafter present Megiddo's technique. Note that randomized methods are unusable in the incremental approach.

**Summary of the prune and search technique of linear programming.** We know that a limited number of constraints are tight at an optimal solution of a LP problem. This method tries to eliminate input constraints that do not affect the optimum value. At each step, a constant fraction of $n$ constraints is eliminated from the current set in $O(n)$ time. Therefore after a logarithmic number of steps, the size of the problem becomes constant. By using any strongly polynomial LP algorithm, we solve our remaining set of constraints in constant time. Because of this decimation, the global cost remains bounded by the cost of the first pruning step.

**Problem posing.** We want to find the optimum value of a $d$-dimensional LP problem of $n$ constraints. We can always transform the gradient function into $(0, \ldots, 0, 1)$ by rotation. For presentation convenience, we only consider constraints that have a strictly positive coefficient associated to $x_d$ (wlog). We want to solve:

$$\begin{array}{ll} \text{Minimize} & x_d \\ \text{So that} & x_d \geq \Sigma_{j=1}^{d-1} a_{ij}.x_j + b_i \qquad (i = 1, \ldots, n) \end{array} \tag{2}$$

**Deletion criterion.** The core of the technique consists in coupling constraints. Under the previous assumption, if we take two non-parallel constraints, there exists a vertical hyperplane passing through their intersection, that divides the space into two half spaces. Such a hyperplane is called a separating line (SL) or a separating plane (SP). If optimal solutions are located in a certain half space, then we may discard one of the two constraints.

### 2.2 The Two-Dimensional Case

We give an overview of Megiddo two-dimensional algorithm described in [Meg83]. A comprehensive description of this method is given in [PS85] and in [Ede87] . We hereafter describe its inner loop in four steps (see Fig. 1).

**The algorithm steps.**

1. Coupling: we create couples of constraints (except for one at most) and their associated SL. Under the assumption of 2.1, if two inequalities are parallel, one of them is redundant and can be immediately suppressed.
2. Selection of a test line : vertical SL have horizontal coordinates. We compute in linear time the median of these values. We select the test line to be the particular SL that corresponds to this median.
3. Testing a line: we want to know on which side of the test line are the optima located. For this, we first compute the optimum of the LP problem restricted to this line. We determine the right and left slopes given by the constraints passing through this point. By convex properties of the feasibility polyhedron, if a decreasing slope exists, optima will be located on its side. Else this point is a minimum and the problem is solved.
4. Pruning: As the test line is a median for the SL set, we deduce the optima location relative to one half of the SL. We then apply the deletion criteria to each couple of constraints associated to these SL. After this, we iterate as long as the number of constraints is above a fixed constant.

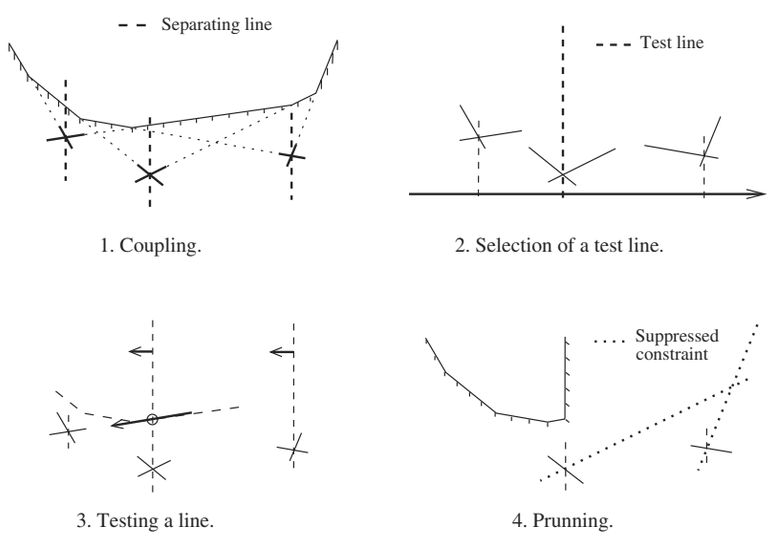



**Fig. 1.** Steps of Megiddo two-dimensional algorithm.

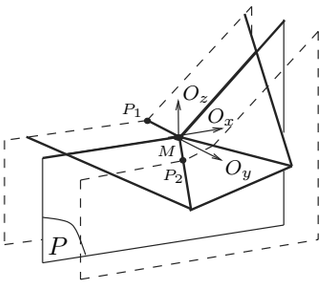



**Fig. 2.** Testing a plane.

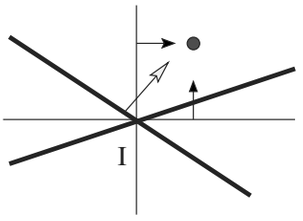



**Fig. 3.** Point location.

**Global complexity.** One quarter of the inequalities are rejected from the current set. Each of the four steps has a linear time complexity in the number of constraints. This implies that the runtime $T(n)$ of our algorithm satisfies $T(n) = O(n) + T(\frac{3}{4}.n)$. Therefore this algorithm solves a linear program in two variables and n constraints in $O(n)$ time.

### 2.3 The Three-Dimensional Case

We keep the technique described in the two-dimensional method. We test particular planes which enable us to drop a constant fraction of inequalities. After this, we iterate. We precisely describe steps 2 and 3 in the next sections.

**Step 2: Selection of testing planes.** As the separating planes (SP) are all vertical, we represent them by lines in the $O_{xy}$ plane. Our problem also becomes a two-dimensional search problem. Megiddo describes a technique to solve it effectively in [Meg83] and in [Meg84]. Readers can refer to [Ede87]. Suppose we have two planar lines of opposite slopes, let $I$ denote their intersection. If we know the position of a point relative to the vertical and horizontal lines passing trough $I$, we can determine the location of this point relative to at least one of the two given lines (see Fig. 3). The trick consists in finding in linear time the median of all present slopes. To shorten explanations, we consider the median direction to be vertical. We can also couple lines (except for one of them at most) and create couples of opposite slopes. So we use the previous remark and we obtain a set of vertical lines. We compute their median line and test it. We know the optimum position relative to one half of the vertical lines. We select the horizontal lines coupled with these vertical lines, take their median and test it. We also determine the optima position relative to $\frac{1}{8}$ of SP and we can also delete $\frac{1}{16}$ of constraints. All these steps are shown in Fig. 4.

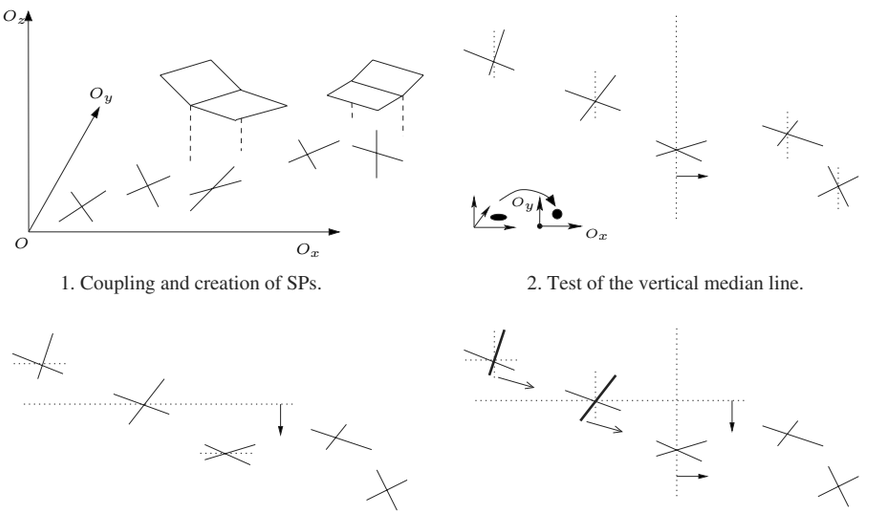
**Step 3: Testing a plane.** We want to know on which side of a plane $P$ lies the optima (see Fig. 2). We first solve our LP problem restricted to this

plane and obtain a minimum called $M$. If this problem is unbounded the three-dimensional problem is unbounded too. Now, we have to test both sides of $P$. Let $C$ denote the set of constraints passing through $M$. Because of the finite number of inequalities, there is a neighbouring ball of radius $r$ around $M$ where no other constraint can interfere, so the local decrease around $M$ only depends on constraints in $C$. Without loss of generality, we can assume that $P$ is $O_{xz}$ and $M$ is the origin. Let $P_1$ and $P_2$ denote the solutions of the two following systems:

$$\begin{cases} \text{Minimize } z, \text{ under } C \\ \text{with } y_i = r \end{cases} \quad \begin{cases} \text{Minimize } z, \text{ under } C \\ \text{with } y_i = -r \end{cases} \tag{3}$$

If one solution is better than $M$, it indicates the optima location. If both are worse, $M$ is a solution of our problem. Because of the convexity of the feasibility polyhedron, we cannot have two better solutions. These two LP problems are linear in the number of constraints as we saw in the previous section.

*Remark 1.* By definition of $C$, we do not have to determine a value for $r$. We can also choose $r$ equal to 1.




1. Coupling and creation of SPs.
2. Test of the vertical median line.
3. Test of the horizontal median line.
4. Optima location relative to two SPs.

**Fig. 4.** Solving the search problem.

## 3 Digital Line and Plane Recognition

We know we have to solve diophantine equations of the form: $\gamma \leq N.P(x_1, \ldots, x_d) < \gamma + \|N\|_\infty$. Each point P is linked to two inequalities. We

cannot solve integer programming with LP method. I hereafter describe my rewriting of the three-dimensional problem that allows to use LP. As the normal vector is nonzero, the following transform is always possible:

$$\frac{\gamma}{\|N\|_\infty} \leq \sum_{i=1}^{d} \frac{N_i.x_i}{\|N\|_\infty} < \frac{\gamma}{\|N\|_\infty} + 1 \tag{4}$$

We now have a set of linear inequalities with $d$ variables at most. For instance, in the three-dimensional case, we can solve this system using three-dimensional Megiddo algorithm. We will hereafter only consider the case when $\|N(u,v,w)\|_\infty = w$. We are not concerned with negative values of $w$ because $N$ is valid iff $-N$ is. We have:

$$\begin{cases} \gamma \leq N.P(x,y,z) < \gamma + \|N\|_\infty \\ \|N(u,v,w)\|_\infty = w \end{cases} \Leftrightarrow \begin{cases} \frac{\gamma}{w} \leq \frac{u}{w}.x + \frac{v}{w}.y + z < \frac{\gamma}{w} + 1 \\ |\frac{u}{w}| \leq 1, |\frac{v}{w}| \leq 1 \end{cases} \tag{5}$$

Linear programming consists in optimizing linear function. Nevertheless, our recognition problem only requires to solve a feasibility problem:

$$\begin{array}{ll} \text{Find} & (a,b,h) \\ \text{So that} & \begin{cases} a.x_i + b.y_i - h \geq -z_i \\ a.x_i + b.y_i - h < -z_i + 1 \quad (i = 1,\ldots,n) \\ |a| \leq 1, |b| \leq 1 \end{cases} \end{array} \tag{6}$$

*Remark 2.* This transform is equivalent to a linear separability problem. Let $S$ be a set of points, we want to find a plane passing between $S$ and a virtual set of forbidden points $S + \boldsymbol{e_z}$. If such a plane exists, the convex hull of $S$ and the convex hull of $S+\boldsymbol{e_z}$ do not intersect, see [PS85] for details. The vertical distance between these two convex hulls is in $]0,1[$. This implies that $S$ is lying in a band of vertical thickness less than 1. By definition, $S$ is a subset of a digital plane.

Strict and large inequalities in a same system may cause problems. Let $P'$ denote the polyhedron defined with inequalities of (6) transformed into large inequalities and $P$ the polyhedron associated to (6). The difference between $P$ and $P'$ is equal to the superior border of $P$ denoted $\Gamma^{sup}$. From this observation, (6) is equivalent to this problem:

$$\begin{array}{ll} \text{Find} & (a,b,h) \notin \Gamma^{sup} \ ^{(*)} \\ \text{So that} & \begin{cases} a.x_i + b.y_i - h \geq -z_i \\ a.x_i + b.y_i - h \leq -z_i + 1 \quad (i = 1,\ldots,n) \\ |a| \leq 1, |b| \leq 1 \end{cases} \end{array} \tag{7}$$

*Remark 3.* $^{(*)}$ requires just a modification in the algorithm. In the two-dimensional problem, when we cut by a vertical line, we only have to keep the highest and lowest feasible points on the cut and check if they are different. Otherwise, no solution lies on this cut. The three-dimensional problem calls the two-dimensional algorithm and uses its improvement.

## 4 The Incremental Problem

We are able to recognize digital lines and planes in linear time using Megiddo's technique. Nevertheless the incremental algorithm is quadratic. I adapt our feasibility problem to create a linear time incremental method. The core of this method uses the following lemma:

**Lemma 1.** *Let $H$ be a separating hyperplane associated to a couple of constraints. If the set of feasible solutions lying on $H$ is included in a hyperplane of $H$ then one of the two constraints is inactive.*

*Remark 4.* An inactive constraint supports no face of the polyhedron and its suppression does not affect the polyhedron.

### 4.1 The Starting Point

To apply Megiddo's technique in $\mathbb{Z}^d$ ($d = 2$ or 3 in our study), we must suppose that the $i^{th}$ coefficient of each possible normal is equal to its infinite norm. In other words, we want to know if all the possible hyperplanes can be written as a function of the same $d-1$ coordinates. The two-dimensional case is quite easy. Suppose the first inserted point M is the origin. If the next point $P(x, y)$ verifies $|y| < |x|$, (resp. $>$), all the possible normals $N(u, v)$ will verify $|u| > |v|$ (resp. $<$). In the three-dimensional case, we do not have the same property. Therefore we have to run a recognition for each direction.

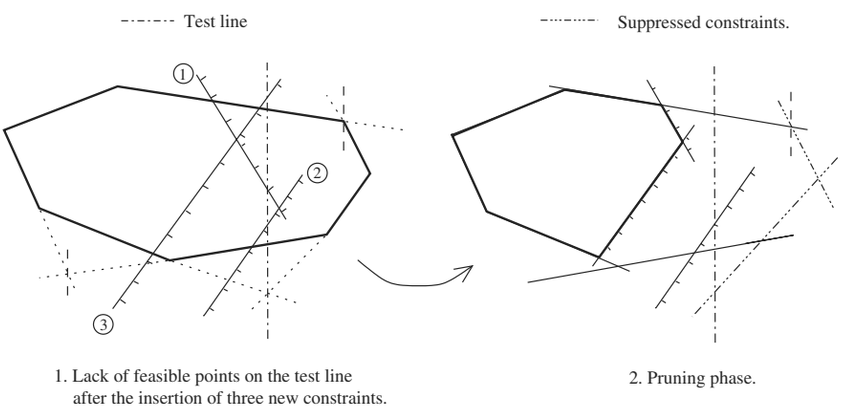



**Fig. 5.** Three added constraints before the pruning step.

### 4.2 The Two-Dimensional Case

We now work with constraints oriented in any direction. We couple inequalities of the same form $z \geq a_i.x + b_i.x + c_i$ or $z \leq a_i.x + b_i.x + c_i$ for instance. At

most two inequalities are not treated. We begin as in Megiddo algorithm and stop before the pruning step. We are cutting by a vertical line and we can have three different cases:

1. We have no feasible point on the test line, the condition of Lemma 1 is fulfilled. We can thus go on with Megiddo's method.
2. There is only one point on the test line. As the line is vertical, this point lies on the superior border of the polyhedron. According to Remark 3, it cannot be a solution for the global problem (7). This case is equivalent to case 1.
3. There is a segment of feasible points lying on the test line. The point of lowest ordinate is a solution. We do not enter the pruning phase and store new constraints in a waiting set. All the core of the incremental method is here. At once, Megiddo algorithm is frozen. For each new constraint we only update the segment of feasible points. While the result is a segment, its point of lowest ordinate is a solution, otherwise we are in case 1 or 2. We wait for new constraints to reject all the feasible points on the test line (see Fig. 5). After this, we can unfreeze Megiddo algorithm. As Lemma 1 is verified, the pruning step suppresses inactive constraints among the previously coupled constraints. Before the next iteration, we add all the inequalities from the waiting set to the current set of constraints.

### 4.3 The Three-Dimensional Case

We use the same retarded Megiddo approach (see Fig. 6). We freeze Megiddo algorithm when we want to test SPs. To cut with the first SP, we apply the two-dimensional linear incremental technique. If we have a feasible point, we enter a three-dimensional frozen phase (new constraints are not coupled and they are collected in a waiting set). Adding new constraints in the two-dimensional problem, we obtain new feasible points until there is no more solution. We then test the plane and determine on which side a feasible region may exist. We also cut with a second SP and launch a two-dimensional linear incremental problem with inequalities from coupled constraints and from the constraints of the waiting set. We are in the frozen phase and so next constraints will be added to the waiting set. New constraints bring new solutions. When there are no more feasible points on the cut, we test it. We now know that the final polyhedron lies at most by one edge on each cut. It implies that all the constraints that have to be suppressed verify Lemma 1. Therefore the pruning phase will only delete inactive constraints. We can also leave the frozen phase and enter the following step. Before the next iteration, the remaining constraints are united with the constraints of the waiting set.

### 4.4 Complexity of the Incremental Algorithm

We summarize all the different steps of a three-dimensional sequence in Fig. 7. As $O(m+k+k') \sim O(m)+O(k+k')$, the complexity is equivalent to the linear time Megiddo algorithm complexity. More precisely, when we have less than a fixed

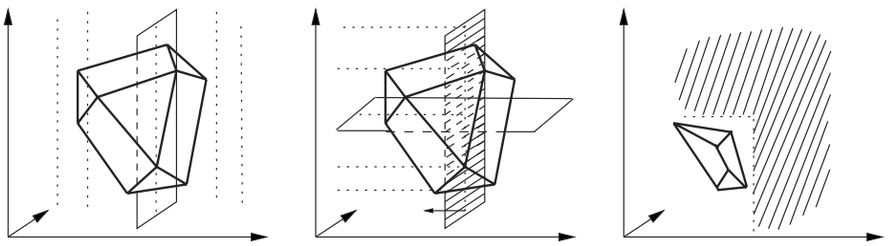


**Fig. 6.** Frozen phase of the three-dimensional incremental algorithm.

number of constraints, we use a strongly polynomial LP method (SPM) to obtain a feasible solution in constant time. For instance, we suppose that $k$ frozen phases (FP) are executed. We denote by $(a_i)_{1\leq i\leq k}$ the number of constraints added in each FP (it can be zero) and by $a_0$ the number of constraints at the beginning. Let $n_i = \sum_{j=0}^{i-1} a_j$ and $m_i$ denote the number of previously entered constraints and the number of present (coupled) constraints when we enter the $i^{th}$ FP. We denote by $\alpha.n + \beta$ the decimation function. It allows to bound the number of kept constraints when we apply the pruning technique to $n$ constraints. We then deduce according to the previous definitions:

$$T(n_j) \leq T(n_{j-1}) + O(m_{j-1} + a_j) + a_j.\Gamma_{SPM} \tag{8}$$

As we define $m_{j+1} \leq (\alpha.m_j + \beta) + a_j$, we establish:

$$m_{j+1} \leq \sum_{i=0}^{j} a_i.\alpha^{j-i} + \frac{\beta}{1-\alpha} \tag{9}$$

Using (8) and (9), we finally obtain:

$$T(n) = \sum_{j=1}^{k} O(\sum_{i=0}^{j-1}(a_i.\alpha^{j-1-i} + \frac{\beta}{1-\alpha})) + \sum_{i=1}^{k} O(a_i) \tag{10}$$

$$T(n) = O(\sum_{i=0}^{k-1} \frac{a_i}{1-\alpha}) + O(n) = O(n) \tag{11}$$

## 5 Conclusion

We have presented a new linear incremental algorithm for line and plane recognition. It may suffer from an important linearity coefficient and from a quite difficult implementation. Nevertheless it provides not only linear incremental complexity, but also a robust and exact method for recognition of any set of points.

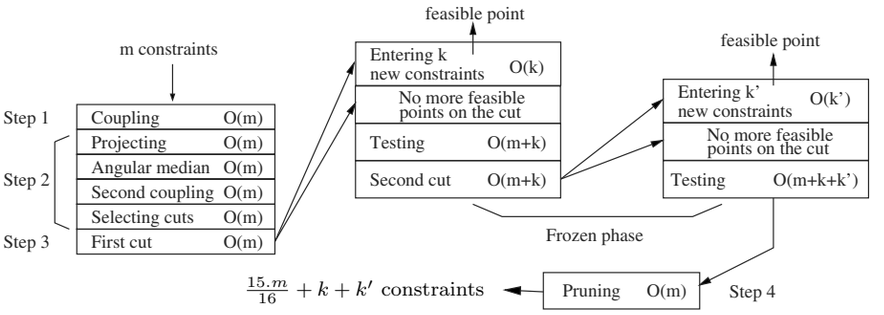



**Fig. 7.** Modified Megiddo sequence in three dimensions.

Two-dimensional implementation was done and tested. Finding new feasible points is a low cost operation (cost of computing the intersection of two lines). The time-consuming part is the Megiddo pruning sequence (the four steps), but we cannot forget that it allows to limit the number of constraints and so to achieve linear incremental complexity. We are now studying and optimizing a specific three-dimensional implementation using rational arithmetics.

## References


[AS98] Pankaj K. Agarwal and Micha Sharir (1998). Efficient algorithms for geometric optimization. *ACM Comput. Surv.,* vol. **30**, pp. 412-458.

[BKOS00] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf (2000). *Computational Geometry: Algorithms and Applications (*$2^{nd}$ *ed.).* Springer-Verlag.

[Cla86] K.L. Clarkson (1986). Linear Programming in O($n.2^{3.d^2}$) time. *Inform. Process. Lett.,* vol. **22**, pp. 21-24.

[Cla98] K.L. Clarkson (1998). A Las Vegas algorithm for linear programming when the dimension is small. *In Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci.,* pp. 452-456.

[DR95] I. Debled-Rennesson, J.-P. Reveillès (1995). A linear algorithm for segmentation of digital curves. *International Journal of Pattern Recognition and Artificial Intelligence*, vol. **9**, no. 6, pp. 635-662.

[Ede87] H. Edelsbrunner (1987). *Algorithms in Combinatorial Geometry.* Springer-Verlag, New York.

[Meg83] N. Megiddo (1983). Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM J. Comput.,* vol. **12** , pp. 759-776.

[Meg84] N. Megiddo (1984). Linear programming in linear time when the dimension is fixed. *J. ACM,* vol. **31** , pp. 114-127.

[PS85] F.P. Preparata and M.I. Shamos (1985). *Computational Geometry: an Introduction.* Springer-Verlag, New York.

[Rev91] J.P. Reveillès (1991). *Géometrie discrète, calculs en nombre entiers et algorithmique.* Thèse d'état, Université Louis Pasteur, Strasbourg.

[Sei91] R. Seidel (1991). Small-Dimensional Linear Programming and Convex Hulls Made Easy. *Discrete and Computational Geometry*, vol. **6**, pp. 423-434.