An Efficient and Quasi Linear Worst-Case Time Algorithm for Digital Plane Recognition

Emilie Charrier[1,2,3] and Lilian Buzer[1,2]

[1] Université Paris-Est, LABINFO-IGM, CNRS, UMR 8049
[2] ESIEE, 2, boulevard Blaise Pascal, Cité DESCARTES, BP 99
93162 Noisy le Grand CEDEX, France
[3] DGA/D4S/MRIS
charriee@esiee.fr, buzerl@esiee.fr

Abstract. This paper introduces a method for the digital naive plane recognition problem. This method is a revision of a previous one. It is the only method which guarantees an $O(n \log D)$ time complexity in the worst-case, where $(D-1)$ represents the size of a bounding box that encloses the points, and which is very efficient in practice. The presented approach consists in determining if a set of n points in $\mathbb{Z}^3$ corresponds to a piece of digital naive hyperplane in $\lfloor 4 \log_{9/5} D \rfloor + 10$ iterations in the worst case. Each iteration performs n dot products. The method determines whether a set of 10^6 voxels corresponds to a piece of a digital plane in ten iterations in the average which is five times less than the upper bound. In addition, the approach succeeds in reducing the digital naive plane recognition problem in $\mathbb{Z}^3$ to a feasibility problem on a two-dimensional convex function. This method is especially fitted when the set of points is dense in the bounding box, i.e. when $D = O(\sqrt{n})$.

Keywords: Digital naive plane recognition, convex optimization, feasibility problem, quasi linear time complexity, chord's algorithm.

1 Introduction

Digital naive plane recognition is a deeply studied problem in digital geometry (see a review in [1]). It consists in determining whether a set of n points in $\mathbb{Z}^d$ is a piece of a digital naive plane or not. The method presented in this paper is a revision of the algorithm proposed by Buzer in [3]. Both algorithms achieve a quasi linear worst-case time complexity for the three-dimensional recognition problem but the new one is more efficient in practice and achieves an $O(n \log D)$ time complexity. The result can be extended without difficulty to the recognition of digital planes of fixed thickness.

The approaches used in recognition are usually based on linear programming [2,9,14], convex hulls and geometrical methods [4,5,12,16], combinatorial optimization [4,5,6,7,13,16] or the evenness property [15]. The methods based on linear programming can be separated into two groups. The first group [2,14] relies on the optimal result obtained by Megiddo [9]. However, even if the approaches

D. Coeurjolly et al. (Eds.): DGCI 2008, LNCS 4992, pp. 346–357, 2008.

in this group achieve an optimal linear time complexity, the resulting algorithms are too complex to be used in practice. The second group is based on efficient linear programming techniques like the simplex algorithm but their worst-case time complexity is too high in practice. Methods that partially traverse the convex hull of the chords' space of a given set of points do not override this problem. For example, the chord's algorithm [7] processes 10^6 voxels in about ten traversals of the point set. Nevertheless, this technique exhibits an $O(n^7)$ time complexity. All of the previous methods do not combine both efficiency in practice and a low worst-case time complexity. Buzer proposes in [3] an $O(n \log^2 D)$ worst-case time complexity algorithm which recognizes a digital naive plane of 10^6 voxels in $\mathbb{Z}^3$ in about 360 iterations, where $(D-1)$ represents the size of a bounding box that encloses the points. This paper presents a revision of this algorithm that achieves an $O(n \log D)$ worst-case time complexity and that recognizes a digital naive plane of 10^6 voxels in $\mathbb{Z}^3$ in about 10 iterations. These two last approaches reduce the digital naive plane recognition in $\mathbb{Z}^d$ into a feasibility problem on a $(d-1)$-dimensional convex function. Thus, for the three-dimensional recognition problem, we only need to manage two parameters and so we can apply planar geometrical techniques to determine whether the two-dimensional solution space is empty. To solve the feasibility problem, the presented approach uses a property of the center of gravity whereas the previous method combines Megiddo oracle and one-dimensional binary search. This main change improves the time complexity and decreases the number of iterations of the algorithm.

In Sect. 2, we introduce some useful notations and definitions. In Sect. 3, we present how the recognition problem in $\mathbb{Z}^d$ can be transformed into a feasibility problem on a $(d-1)$-dimensional convex function. Then, we focus on the three-dimensional recognition problem. In Sect. 4, we describe the solution space of the feasibility problem. Then, we sketch the algorithm and analyze its complexity in Sect. 5. Finally, in the last section, we describe some enhancements in order to improve the efficiency of our method and we present some experimental results compared to the fastest known algorithm. Note that Sect. 3 is already existing in the previous paper but we recall it for presentation convenience. The bounds in Sect. 4 are enhanced. Finally, the algorithm design is mostly new.

2 Definitions and Notations

Definition 1. *An digital plane is defined by* $P_{N,\mu,\omega} = \{p \in \mathbb{Z}^d | \mu \leq N \cdot p < \mu + \omega\}$ *where* $N = (N_1, \ldots, N_d)$ *denotes the* normal vector *in* $\mathbb{Z}^d$ *such that* $gcd(N_1, \ldots, N_d)$ *is equal to one and where* ω *denotes the* arithmetic thickness. *When* $\omega = ||N||_\infty = \max_{1 \leq i \leq d} |N_i|$ *we obtain a* naive plane . *When* $\omega = \sum_{i=1}^{d} |N_i|$, *we obtain a* standard plane.

In this paper, we only consider the case where the digital hyperplanes in $\mathbb{Z}^d$ are a function from $(x_1, \ldots, x_{d-1})$ to $\mathbb{Z}$. Other cases can be deduced by symmetry. Consequently, we consider that $||N||_\infty$ is equal to $|N_d|$. Moreover, we can suppose w.l.o.g. that N_d is a positive value.

Remark 1. In this paper, the expression $S = \{p^1, \ldots, p^n\}$ denotes the set of n points in $\mathbb{Z}^d$ we study. Moreover, we always use the notation $(x_1, \ldots, x_d)$ to denote the coordinates of a point x in a d-dimensional space.

We study the *digital naive plane recognition problem*. This problem consists in determining whether S corresponds to a subset of a digital naive plane. For this, we want to find a vector $N = (N_1, \ldots, N_d)$ in $\mathbb{Z}^d$ such that $gcd(N_1, \ldots, N_d) = 1$ and such that $\mu \leq N \cdot p^i < \mu + ||N||_\infty, 1 \leq i \leq n$.

3 Feasibility Problem

In this section, we show that recognizing a digital naive plane in $\mathbb{Z}^d$ is equivalent to solving a feasibility problem on a $(d-1)$-dimensional convex function.

3.1 Introduction

Proposition 1. *If there exists $\gamma \in \mathbb{R}$ and $N' \in \mathbb{R}^d$ such that $||N'||_\infty = 1$ and such that $\gamma \leq N' \cdot p^i < \gamma + 1, 1 \leq i \leq n$, then S corresponds to a piece of a digital naive plane.*

As a result of this first proposition, we can allow us to consider only normal vectors $N = (N_1, \ldots, N_d)$ such that $||N||_\infty = 1$. From the assumptions made in Sec. 2, we consider that N_d is equal to one.

Definition 2. *For $u \in \mathbb{R}^{d-1}$, the symbol N_u denotes the vector $(u_1, \ldots, u_{d-1}, 1)$. We define the function $h_S(u)$ from $\mathbb{R}^{d-1}$ to $\mathbb{R}^+$ that computes the distance, relative to the d-th axis, between the two supporting hyperplanes of normal vector N_u that enclose all the points of S (see [1]). The value $h_S(u)$ is equal to:*

$$h_S(u) = \max_{1 \leq i \leq n} (N_u \cdot p^i) - \min_{1 \leq i \leq n} (N_u \cdot p^i) \tag{1}$$

From Prop. 1, if there exists a vector u such that $h_S(u)$ is strictly less than one then S is a piece of a digital naive plane. As a result, to solve the recognition problem, we try to find a vector $u \in \mathbb{R}^{d-1}$ such that $||u||_\infty \leq 1$ and such that $h_S(u) < 1$.

Property 1. $h_S(u)$ is a convex function.

Proof. Let N_u denote a normal vector associated with the value $u \in \mathbb{R}^{d-1}$. Consider the function $g^i(u) = N_u \cdot p^i$. This function is an affine function and it is also convex. Thus, the maximum of the functions $(g^i)_{1 \leq i \leq n}$ is convex too. The function $h_S(u)$ can be rewritten as $\max_{1 \leq i \leq n}(N_u \cdot p^i) + \max_{1 \leq i \leq n}(-N_u \cdot p^i)$. By the same logic, this expression is also a convex function. Since the sum of two convex functions is convex, we deduce that h_S is convex.

In conclusion, to solve the three-dimensional recognition problem, we study the two-dimensional convex function $h_S(u) : [-1, 1]^2 \to \mathbb{R}^+$. If there exists a point u in $[-1, 1]^2$ such that the value of $h_S(u)$ is strictly less than one, we can conclude that S is a piece of a digital plane. As a result, our recognition problem is equivalent to a feasibility problem on h_S.

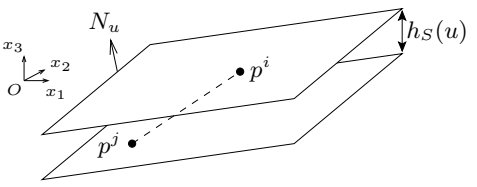


Fig. 1. Definition of h_S

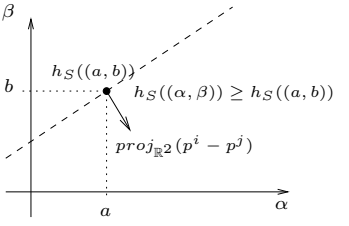


Fig. 2. A subgradient of h_S

3.2 Subgradient Computation

We first recall that a *subgradient* $g \in \mathbb{R}^d$ of a convex function $f : \mathbb{R}^d \to \mathbb{R}$ at the point x satisfies: for any $x' \in \mathbb{R}^d$, $f(x') - f(x) \geq g \cdot (x' - x)$. The subgradient indicates the steepest descent of the function f at x.

In order to solve the feasibility problem in $\mathbb{R}^{d-1}$, we have to determine how to compute a subgradient of the convex function h_S. For a given value $u \in \mathbb{R}^{d-1}$, we only have to traverse the set of points S in order to compute the value $h_S(u)$. This implies that the computation of $h_S(u)$ has a linear time complexity. Relative to the definition of h_S, we know that $h_S(u)$ is equal to $\max_{1 \leq i \leq n}(N_u \cdot p^i) - \min_{1 \leq i \leq n}(N_u \cdot p^i)$. As a result, there exist two points p^i and p^j associated with the max and the min expressions (see Fig. 1). Therefore, $h_S(u) = N_u \cdot (p^i - p^j)$. Let T denote the set of points $\{p^i, p^j\}$. As T is included in S we have: $\forall v \in \mathbb{R}^{d-1}, h_T(v) \leq h_S(v)$.

The value of $h_T(v)$ is equal to:

$$h_T(v) = |N_v \cdot p^i - N_v \cdot p^j| = |N_v \cdot (p^i - p^j)| \tag{2}$$

and it follows:

$$\begin{aligned} h_T(v) &= |N_{v-u+u} \cdot (p^i - p^j)| \\ &= |N_u \cdot (p^i - p^j) + (v-u) \cdot proj_{\mathbb{R}^{d-1}}(p^i - p^j)| \\ &= |h_T(u) + (v-u) \cdot proj_{\mathbb{R}^{d-1}}(p^i - p^j)| \end{aligned}$$

By definition of the absolute value, we obtain:

$$\forall v \in \mathbb{R}^{d-1}, h_S(v) \geq h_T(v) \geq h_T(u) + (v-u) \cdot proj_{\mathbb{R}^{d-1}}(p^i - p^j) \tag{3}$$

We recall that $h_S(u) = h_T(u)$ and so it follows:

$$\forall v \in \mathbb{R}^{d-1}, h_S(v) \geq h_S(u) + (v-u) \cdot proj_{\mathbb{R}^{d-1}}(p^i - p^j) \tag{4}$$

We can now conclude that the expression $proj_{\mathbb{R}^{d-1}}(p^i - p^j)$ is a subgradient of h_S at the point u (see Fig. 2). This value corresponds to the projection of the vector $p^i p^j$ in the $\mathbb{R}^{d-1}$ space. This means that the first $d-1$ components of the vector $p^i p^j$ are sufficient to locally determine the variation of the function h_S. When we evaluate $h_S(u)$, we indirectly deduce the two points p^i and p^j. Thus, in constant time, we determine one of its subgradients.

In the next section, we study the solution space of the feasibility problem.

4 Studying the Solution Space

From now on, we focus on the three-dimensional recognition problem.

Let F denote the solution space of the feasibility problem. The set F is convex and it corresponds to the points u in $[-1,1]^2$ that satisfy $h_S(u) < 1$. We show afterwards that if the set F is not empty, then it contains a square of side length $1/(2D^3)$ where D denotes the size of a bounding box including S. This property allows us to restrict the search space to a regular grid of step $1/(2D^3)$ on $[-1,1]^2$.

We suppose that the set S lies in a bounding box of size $D-1$ and that the origin is located at the center of the bounding box. Thus, any point p^i of S satisfies $||p^i||_\infty \leq D/2$ for $1 \leq i \leq n$ and any vector k whose endpoints correspond to two points of S satisfies:

$$||k||_\infty \leq D-1 \tag{5}$$

Considering the convex hull of the set S, we know that there exist two supporting planes of normal vector $N = (N_1, N_2, N_3)$ such that $h_S((N_1/N_3, N_2/N_3))$ is minimal for S and such that these two planes are supported by four vertices of the convex hull (see [4]). From [12], N corresponds to the cross product of two vectors supported by the given vertices. Therefore, there exist two vectors k and k' in S such that N is equal to $k \wedge k'$. From (5), we have $||N||_\infty < 2D^2$ and in particular:

$$N_3 < 2D^2 \tag{6}$$

From Def. 1, if S corresponds to a digital naive plane then for all (x_1, x_2, x_3) in S, N satisfies $\mu \leq N_1x_1 + N_2x_2 + N_3x_3 \leq \mu + N_3 - 1$. Let (N'_1, N'_2) denote the vector $(N_1/N_3, N_2/N_3)$, (N'_1, N'_2) belongs to the solution space and we have:

$$\forall (x_1,x_2,x_3) \in S,\ \mu' \leq N'_1x_1 + N'_2x_2 + x_3 \leq \mu' + 1 - 1/N_3 \tag{7}$$

We want to find an upper bound for the positive real value Δ such that the vector $(N_{\Delta 1}, N_{\Delta 2}) = (N'_1 \pm \Delta, N'_2 \pm \Delta)$ belongs to the solution space too. From (5) and (7), the following inequalities hold:

$$\forall (x_1,x_2,x_3) \in S,\ \mu' - D\Delta \leq N_{\Delta 1}x_1 + N_{\Delta 2}x_2 + x_3 \leq \mu' + 1 - 1/N_3 + D\Delta \tag{8}$$

This is equivalent to:

$$\forall (x_1,x_2,x_3) \in S,\ \mu' - D\Delta \leq N_{\Delta 1}x_1 + N_{\Delta 2}x_2 + x_3 \leq \mu' - D\Delta + 1 + 2D\Delta - 1/N_3 \tag{9}$$

From Prop. 1 and (9), we deduce that the expression $2D\Delta - 1/N_3$ is strictly negative and so:

$$\Delta < 1/(2DN_3) \tag{10}$$

It follows from (6) and (10) that:

$$\Delta \leq 1/(4D^3) \tag{11}$$

As a result, we can restrict the search space to a regular grid G of step $2\Delta = 1/(2D^3)$ on $[-1,1]^2$. Indeed, for any normal vector $N = (N_1, N_2, N_3)$

such that $h_S((N_1/N_3, N_2/N_3)) < 1$ and such that $N_3 < 2D^2$, there exists a square centered on $(N_1/N_3, N_2/N_3)$ in the search space corresponding to the solutions u such that $h_S(u) < 1$. If F is not empty there exists such a vector $N = (N_1, N_2, N_3)$ and its corresponding square contains a grid point. If none of the sampled values corresponds to a solution of the feasibility problem then the solution space is empty. This step size is improved relative to the one proposed in [3].

5 Algorithm Design

To solve convex optimization problems, we compute the minimum value of a convex function f. For this, we could use gradient descent methods. Using this technique, we approach the minimum of the function by moving iteratively in the direction of the steepest descent. At the i-th iteration, the gradient $\nabla(x^i)$ of f at the point x^i is computed. By definition of the gradient, there exists a value τ_i such that $f(x^i)$ is less than $f(x^i + \tau_i \nabla(x^i))$. Thus, the point x^{i+1} is equal to:

$$x^{i+1} = x^i + \tau_i \nabla(x^i) \tag{12}$$

However, we cannot easily apply such algorithms. In fact, if we want to use exact numerical computation, the value of τ_i and consequently the value of x^i must be represented by rational numbers. From (12), the size of the numerator and denominator of x^i would increase at each iteration and so this would significantly slow down the calculation. Our method evades this problem.

5.1 Sketching the Method

We briefly introduce our method. Let us call a *valid cut* on the search space a cut by an half-plane which preserves all the feasible solutions. Our algorithm consists in recursively applying valid cuts on the search space in order to reduce it. We ensure that each cut eliminates at least a constant fraction of the current search space. Thus, our algorithm solves the problem in a logarithmic number of iterations. After each cut, we transform the remaining search space into the equivalent convex polygon whose vertices are supported by the grid. When the search space is reduced to one point, our problem is solved. In the following, we give details on the main steps of our algorithm.

Valid Cuts. At each iteration, we compute the value of the function h_S at a given point u. If $h_S(u)$ is strictly less than one, then u is a feasible solution and the problem is solved. Otherwise, we cut the search space. The cut passes through u and it is orthogonal to the subgradient at this point. By definition of the subgradient, it follows that any grid point u' eliminated by the cut satisfies $h_S(u') \geq 1$. As this point does not correspond to a feasible solution, we can conclude that each cut is a valid cut.

Computing the Current Point. At each iteration, we compute the value of the function h_S at the *center of gravity* C of the search space R. We recall that the center of gravity of a polygon is equivalent to its *centroid* (also called *barycenter*). We compute its coordinates in linear time relative to the number of vertices of the polygon.

The following proposition ensures the efficiency of our algorithm (see [10]).

Proposition 2. *Let K denote a convex body in the plane, let C denote its center of gravity, then each half-plane supported by C contains between $\frac{4}{9}$ and $\frac{5}{9}$ of the area of K.*

This means that each cut passing through the center of gravity of the search space eliminates at least $\frac{4}{9}$ of its area. Thus, at most $\frac{5}{9}$ of the area of the search space is kept. From [10], we know that this ratio is optimal in the two-dimensional case. We can notice that it is very close to the ratio $\frac{1}{2}$ of the binary search.

Search Space after a Cut. We want to avoid that the numerator and the denominator of the rational numbers we use increase at each iteration. Thus, we describe the search space as a convex polygon whose vertices are supported by the grid. Let R_i and R'_i denote the search space respectively before and after the cut at the i-th iteration. Let $\overline{R_i}$ denote the largest polygon included in R_i whose vertices are included in $R_i \cap G$. By definition of $\overline{R_i}$, no grid point can lie in $R_i \backslash \overline{R_i}$. As we are looking for a solution on the grid, we set R_{i+1} at $\overline{R_i}$. For this, we use Harvey's algorithm (see [8]).

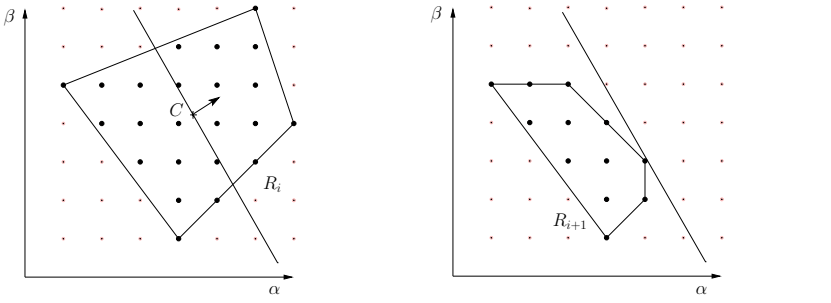
Complexity Analysis. At each iteration, as long as no solution is found, we produce a valid cut passing through the center of gravity of the current search space. Let A_i denote the area of the search space R_i. From Prop. 2 the area of R'_i is less or equal to $5/9A_i$. We set R_{i+1} at the largest convex polygon included in R'_i whose vertices are supported by the grid. Clearly, the area of R_{i+1} is less or equal to the area of R'_i. As a result, we have $A_{i+1} \leq 5/9A_i$ and consequently we have: $A_i \leq (5/9)^i A_0$.

Figure 3 shows an example of a cut on a search space R_i passing through C. We notice that the area of the search space R_{i+1} is less or equal to the 5/9 of the area of the search space R_i.

When the area of the search space is equal to zero, it means that it is reduced to a single point or to a straight line segment. If the search space is reduced to one point, the problem is solved in linear time. Otherwise, the center of gravity is replaced by the middle of the straight line segment. Thus, the ratio of each cut becomes 1/2 and the problem is solved in $\log_2 D$ iterations. As a result, we can conclude that our algorithm makes $O(\log A_0)$ iterations in the worst case, where A_0 denotes the area of the initial search space.

Proposition 3. *The algorithm admits an upper bounds for the number of iterations relative to A_0:*

$$\lfloor \log_{\frac{9}{5}} A_0 + \log_{\frac{9}{5}} 2 \rfloor + 3$$



(a) The search space R_i of area 19. (b) The reduced search space R_{i+1} of area 7.5 ($7.5 \leq \frac{5}{9} * 19$).

Fig. 3. Cut of R_i passing through the center of gravity C

Proof. While the search space R_i contains at least three grid points, we apply a cut and then the area of R_{i+1} is less or equal to $5/9A_i$. Let I_i denote the number of interior grid points of R_i. Let B_i denote the number of boundary grid points of R_i. Thanks to Pick's theorem (see [11]) and as R_i corresponds to a polygon whose vertices are grid points, we can claim that $A_i = I_i + B_i/2 - 1$. As a result, the following inequality always holds: $I_i + B_i \leq 2A_i + 2$. Then, to determine the maximum number of iterations k used to reduce the search space to two points, we just have to solve: $2A_k + 2 < 3$ which is equivalent to $2(5/9)^k A_0 < 3$. It follows that the maximum value for k is $\log_{9/5} A_0 + \log_{9/5} 2$. As k must be an integer value, we fix it at: $\lfloor \log_{9/5} A_0 + \log_{9/5} 2 \rfloor + 1$. Finally, when only two points remain, we have to evaluate them in two iterations. Note that when the search space is reduced to one straight line segment each cut has a ratio of $1/2$ which do not increase the upper bound.

At each iteration, we generate a cut and we compute the largest convex polygon with integer vertices included in the new search space. The vertices of the convex we determine are located in an area delineated by the straight line of the cut and two straight lines supported by edges of the search space. To determine this new search space, we use Harvey's algorithm (see [8]) which runs in logarithmic time relative to the coordinates of the normal vector of these three straight lines. As the normal vector of the straight line of the cut corresponds to a subgradient, from (5) we know that its coordinates do not exceed $O(D)$. Moreover, according to the step size of the grid (see Sect. 4), the coordinates of the two other normal vectors do not exceed $O(D^3)$. As a result, this computation runs in $O(\log D)$ time. As the number of cuts does not exceed $O(\log D)$ and as each cuts adds $O(\log D)$ new vertices, the computation of the center of gravity of the search space runs in $O(\log^2 D)$ time. Moreover, from Def. 2, we know that the evaluation of the function h_S requires n dot products. In conclusion, as $O(\log^2 D)$ and $O(\log D)$ can be neglected relative to $O(n)$ (considering that $D \approx \sqrt{n}$), each cut runs in $O(n)$ time in the worst case which implies the $O(n \log D)$ time complexity of our algorithm.

We can sum up the algorithm as follows:

```
ALGORITHM FOR THE FEASIBILITY PROBLEM:
0  R_i ← [−1, 1]^2 ∩ G    i ← 1
1  WHILE NumberOfIntegerPoints(R_i) > 1
2     C_i ← CenterOfGravity(R_i)
3     IF h_S(C_i) < 1
4        return TRUE
5     ELSE
6        sg_i ←SubGradient(C_i)
7        R'_i ←Cut(R_i, C_i, sg_i)
8        R_{i+1} ← IntegerConvexHull(R'_i)
9        i ← i + 1
10 IF NumberOfIntegerPoints(R_i) = 0
11    return FALSE
12 ELSE
13    return h_S(R_i) < 1
```

6 Improving Performance and Experimental Results

In this section, we describe some modifications already introduced in [3] that improve the performance of our method in practice. These improvements are used in the program of our algorithm.

6.1 Initial Step

Starting with a search space equal to $[-1,1] \times [-1,1]$ is often awkward. As we assume that the set of points is contained in a bounding box of size D, then we can claim that some points lie on the borders. Let $P_1(-D/2, y, z_1)$ and $P_2(D/2, y, z_2)$ denote two of these. They provide one constraint on the solution space. As they belong to a digital naive plane of normal $N(\alpha, \beta, 1)$, we have: $\gamma' \leq N \cdot (-D/2, y, z_1) < \gamma' + 1$ and $\gamma' \leq N \cdot (D/2, y, z_2) < \gamma' + 1$. We infer that: $|N \cdot (D, 0, z_2 - z_1)| < 1$. Thus, we can deduce the following upper and lower bound for α:

$$\frac{z_1 - z_2}{D} - \frac{1}{D} < \alpha < \frac{z_1 - z_2}{D} + \frac{1}{D} \tag{13}$$

In the same way, we can determine an upper and a lower bound for β. So, the side length of the search domain is divided by D. This stage only requires a simple traversal of the set of points in order to find the extreme ones.

6.2 Improving Valid Cuts

We previously show that the knowledge of the subgradient allows us to reject one part of the search domain relative to the point x. Nevertheless, we do not use all of the available information. In fact, when we compute the value of $h_S(x)$,

we determine two points $p_i = (x_i, y_i, z_i)$ and $p_j = (x_j, y_j, z_j)$ of S such that $h_S(x) = |N_x \cdot (p_i - p_j)|$. Let T denote the set $\{p_i, p_j\}$. We have:

$$\forall x' \in \mathbb{R}^{d-1}, h_S(x') \geq h_T(x') \tag{14}$$

As a result, we can restrict the search space to the vectors $x' = (\alpha, \beta)$ which satisfy $|N_{x'} \cdot (p_i - p_j)| < 1$. By definition of the absolute value, the inequality $|N_{x'} \cdot (p_i - p_j)| < 1$ is equivalent to $N_{x'} \cdot (p_i - p_j) < 1$ and $-N_{x'} \cdot (p_i - p_j) < 1$. Thus, if the value of $h_S(x)$ is strictly less than one, the problem is solved. Otherwise, we reduce the search space by applying two valid cuts. Let $\nabla h_S(x)$ denote the subgradient of h_S at the point x, the two cuts are defined by the two following inequalities (see Fig. 4 for an example):

$$\nabla h_S(x) \cdot (\alpha, \beta) < 1 - (z_i - z_j) \tag{15}$$

$$-\nabla h_S(x) \cdot (\alpha, \beta) < 1 - (z_j - z_i) \tag{16}$$

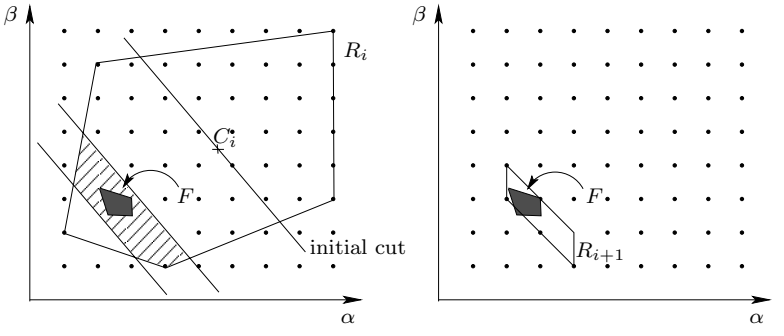


Fig. 4. The search space is reduced by a strip that contains the solution space

6.3 Experimental Results

As the fastest algorithm we know for the recognition problem is the chords' algorithm (see [7]), we compare our optimized algorithm to it. We test the two algorithms with several sets of points which correspond to a piece of a digital plane or not. As the chords' algorithm computes at each iteration n scalar products as our algorithm, we just compare their average number of iterations. Table 1 shows the experimental results. We recall that D denotes the size of a bounding box including the set of points S. To simplify the table, we call our method COBA algorithm (Convex Optimization Based Algorithm). Thanks to Prop. 3, we compute the upper bound of the number of iterations for each D whether we consider the full grid or the reduced grid introduced in Sect. 6.1. The experimental results show that our algorithm makes less iterations than the chords' algorithm in the average when the set of points is dense in the bounding box. Moreover, our algorithm makes about five times less iterations in the mean than the worst case upper bound. We have to develop our algorithm using optimal big integer operations to provide computation times in the future.

	$D = \sqrt{n}$	5	10	100	300	1000
	Number of tries	100	100	100	100	20
Upper Bound	full grid $A_0 = (4D^3 + 1)^2$	25	32	55	67	79
	reduced grid $A_0 = (4D^2 + 1)^2$	19	24	40	47	55
DP	Chords' algorithm	4.80	6.03	9.85	10.75	11.85
	COBA algorithm	2.75	4.15	7.67	9.06	9.75
NDP	Chords' algorithm	3.17	2.72	2.51	2.56	2.55
	COBA algorithm	2.01	2.01	2.00	2.00	2.00

Table 1. Experimental results for digital planes (DP) and non digital planes (NDP), the experimental values correspond to the average number of iterations

7 Conclusion

We describe a different approach for the digital naive plane recognition problem. We show how to transform the recognition process in $\mathbb{Z}^3$ into a feasibility problem on a two-dimensional convex function. This convex function corresponds to the vertical distance between two parallel planes that enclose the points and whose slopes relative to the axis are the parameters of the function. We show how to evaluate this function and how to obtain one of its subgradients in linear time. As the search domain is planar and digital, we apply simple geometrical techniques in order to reduce its size until we find a feasible solution. Moreover, we choose to cut the search space at its center of gravity which ensures a logarithmic number of iterations. This version of our algorithm achieves an $O(n \log D)$ time complexity in the worst-case. We present different modifications to speed up the optimization stage. The experimental result we obtain shows that our algorithm recognizes a digital plane of 10^6 voxels in about ten iterations. These results imply that our algorithm makes less iterations in the average than the chords' algorithm when the set of points is dense. Our algorithm has been designed to be efficient in practice and especially when the set of points is dense in the bounding box. This is the only method known to the authors that achieves a $O(n \log D)$ worst-case time complexity and that is efficient in practice. We show that the recognition problem in $\mathbb{Z}^d$ can be transformed into a feasibility problem in $\mathbb{Z}^{d-1}$ for all dimensions. Nevertheless, some steps of our algorithm uses planar geometrical methods whose extension in higher dimension is not obvious. We are not able yet to extend our method in any dimension.

The authors thank the reviewers for their helpful comments.

References

1. Brimkov, V., Coeurjolly, D., Klette, R.: Digital Planarity - A review. Discrete Applied Mathematics 15(4), 468–495 (2007)
2. Buzer, L.: A linear incremental algorithm for naive and standard digital lines and planes recognition. Graphical Models 65(1-3), 61–76 (2003)

3. Buzer, L.: A composite and quasi linear time method for the digital plane recognition. In: Kuba, A., Nyúl, L.G., Palágyi, K. (eds.) DGCI 2006. LNCS, vol. 4245, pp. 331–342. Springer, Heidelberg (2006)
4. Debled-Rennesson, I.: Etude et reconnaissance des droites et plans discrets. PhD Thesis, Université Louis Pasteur, Strasbourg (1995)
5. Debled-Rennesson, I., Reveillès, J.-P.: A new approach to digital planes. In: Proc. Vision Geometry III, SPIE, vol. 2356, pp. 12–21 (1994)
6. Françon, J., Schramm, J.M., Tajine, M.: Recognizing arithmetic straight lines and planes. In: Miguet, S., Ubéda, S., Montanvert, A. (eds.) DGCI 1996. LNCS, vol. 1176, pp. 141–150. Springer, Heidelberg (1996)
7. Gerard, Y., Debled-Rennesson, I., Zimmermann, P.: An elementary digital plane recognition algorithm, 151(1-3), pp. 169–183 (2005)
8. Harvey, W.: Computing two-dimensional integer hulls. SIAM J. Compute. 28(6), 2285–2299 (1999)
9. Megiddo, N.: Linear programming in linear time when the dimension is fixed. J. ACM 31, 114–127 (1984)
10. Neumann, B.H.: On an invariant of plane regions and mass distributions. Journ. London Math. Soc. 20, 226–237 (1945)
11. Pick, G.: Geometrisches zur Zahlentheorie. Sitzenber. Lotos 19, 311–319 (1899)
12. Preparata, F.P., Shamos, M.I.: Computational Geometry: An Introduction. Springer, New York (1985)
13. Reveillès, J.-P.: Combinatorial pieces in digital lines and planes. In: Proc. Vision Geometry IV, SPIE, vol. 2573, pp. 23–34 (1995)
14. Stojmenovic, I., Tosic, R.: Digitization schemes and the recognition of digital straight lines, hyperplanes and flats in arbitrary dimensions. Vision Geometry, Contemporary Mathematics Series 119, 197–212 (1991)
15. Veelaert, P.: Digital planarity of rectangular surface segments. IEEE Trans. Pattern Analysis Machine Intelligence 16, 647–652 (1994)
16. Vittone, J., Chassery, J.-M.: Recognition of digital naive planes and polyhedrization. In: Nyström, I., Sanniti di Baja, G., Borgefors, G. (eds.) DGCI 2000. LNCS, vol. 1953, pp. 296–307. Springer, Heidelberg (2000)