A Composite and Quasi Linear Time Method for Digital Plane Recognition

Lilian Buzer

Laboratory CNRS-UMLV-ESIEE, UMR 8049
ESIEE, 2, boulevard Blaise Pascal, Cité Descartes, BP 99
93162 Noisy le Grand Cedex, France
buzerl@esiee.fr

Abstract. This paper introduces a new method for the naive digital plane recognition problem. As efficient as existing alternatives, it is the only method known to the author that also guarantees a quasi linear time complexity in the worst case. The approach presented can be used to determine if a set of n points is a naive digital hyperplane in $\mathbb{Z}^d$ in $O(n \log^2 D)$ worst case time where D represents the size of a bounding box that encloses the points. In addition, the approach succeeds in reducing the naive digital plane recognition problem to a two-dimensional convex optimization program. Thus, the solution space is planar and only simple two-dimensional geometrical methods need to be applied during the recognition process. The algorithm is a composite of simple techniques based on one-dimensional optimization: Megiddo Oracle for linear programming and two-dimensional discrete geometry.

1 Introduction

1.1 History

Naive digital plane recognition is a deeply studied problem in digital geometry (see a review in [1]). It consists of determining wether or not a given set of points is a piece of a naive digital plane. This paper presents a new technique with quasi linear worst case complexity that is very efficient in practice. The result can be extended without difficulty to the recognition of digital planes of fixed thickness relative to the infinite norm.

Currently, the approaches used in recognition are based on linear programming [2,3,4], convex hull and geometrical methodologies [5,6,7,8], combinatorial optimization [5, 6, 9, 10, 11, 8] or the evenness property [12]. The methods that exploit linear programming techniques can be separated into two groups. The first group [2, 4] relies on the optimal result obtained by Megiddo [3]. However, even if the approaches in this group achieve 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 complexity is often too high in practice. Methods that partially traverse the convex hull or the chords' space of the given set of points suffer from

A. Kuba, L.G. Nyúl, and K. Palágyi (Eds.): DGCI 2006, LNCS 4245, pp. 331–342, 2006.

the same problem. For example, the chord's algorithm [10] processes 10^6 voxels in about ten traversals of the point set. Nevertheless, this technique exhibits an $O(n^7)$ time complexity. Other algorithms that partially traverse the vertices and the edges of the convex hull obtain an $O(n \log n)$ time complexity, but they are less efficient. All of the previous methods always balance between efficiency in practice and a low worst case complexity. Thus, this paper presents a simple algorithm that has a quasi linear time complexity in the worst case and that is efficient in practice. Moreover, it does not require the piece of the digital plane to be rectangular as in [12, 6, 11].

In addition, the approach is different from previous approaches because it requires only $d-1$ rational parameters to recognize a valid naive digital hyperplane in $\mathbb{Z}^d$, unlike the previous algorithms that require the determination of d rational variables. Thus, for the three-dimensional recognition problem, we need only to apply planar geometrical techniques to know wether or not the two-dimensional set of solutions is empty.

In the next section, we present how our recognition problem can be transformed into a two-dimensional convex optimization program. In the next section, we present the method core of our approach. Then, we sketch the algorithm and compute its complexity in section 4. Finally, in the last section we describe how to implement some enhancements in order to improve the efficiency of our method in practice.

1.2 Definition

Arithmetic geometry provides a uniform approach to study *digital hyperplanes* in any dimension. In this paper, we only consider the case where the digital hyperplanes in $\mathbb{Z}^d$ are a function from $(x_1, \ldots, x_{d-1})$ into $\mathbb{Z}^d$. Other cases can be simply deduced by symmetry. An arithmetic plane is defined by $P_{N,\mu,\omega} = \{x \in \mathbb{Z}^d | \mu \leq N \cdot x < \mu + \omega\}$ with N the normal vector and ω the *arithmetic thickness*. Recall that when $\omega = ||N||_\infty = \max_{1 \leq i \leq d}\{|N_i|\}$ we obtain a *naive plane* (see Fig. 3 for an example). When $\omega = \sum_{i=1}^{d} |N_i|$, we obtain a *standard plane*.

2 Convex Optimization

2.1 Introduction

Property 1. Let $S = (p_j)_{1 \leq j \leq n} = (x_1^j, \ldots, x_d^j)$ denote a set of points in $\mathbb{Z}^d$. If all the points $(p_j)_{1 \leq j \leq n}$ satisfy $\gamma \leq N \cdot p_j < \gamma + 1$ with $\gamma \in \mathbb{R}$ and with N in $\mathbb{Q}^d$ then S is a subset of a naive digital plane.

Proof. Let $\frac{D_i}{N_i}$ denote the i-th component of the normal vector N. By multiplying by $\Pi_{i=1}^{d} N_i$, we obtain for any point p_j: $\beta \leq \sum_{i=1}^{d} (D_i \Pi_{k \neq i} N_k) x_i^j < \beta + \Pi_{i=1}^{d} N_i$ with $\beta \in \mathbb{Z}$. We can simplify this expression by g, the *gcd* of $(D_i \Pi_{k \neq i} N_k)_{1 \leq i \leq d}$. Let δ denote the ceiling of $\frac{\beta}{g}$. As $[\sum_{i=1}^{d} (D_i \Pi_{k \neq i} N_k) x_i^j]/g$ is an integer value, we finally obtain : $\delta \leq \sum_{i=1}^{d} N_i' x_i^j < \delta + ||N'||_\infty$. As δ and the components of N' are integer values, this double inequality corresponds to a naive digital hyperplane.

Definition 1. *Consider a set $S = (p_i)_{1 \leq i \leq n}$ of n points in an Euclidean space $\mathbb{R}^d$. We define a function $h_S(x_1, \ldots, x_{d-1}) : \mathbb{R}^{d-1} \rightarrow \mathbb{R}^+$ as the distance relative to the d-th axis between the two supporting hyperplanes of normal vector $(x_1, \ldots, x_{d-1}, 1)$ that enclose all the points (see [1]).*

Property 2. $h_S(x)$ is a convex function.

Proof. Let $N_x = (x_1, \ldots, x_{d-1}, 1)$ denote the current normal vector associated with the current value $x \in R^{d-1}$ that is being processed. By definition of the function $h_S(x)$, we have: $h_S(x) = Max_{p \in S}(N_x \cdot p) - Min_{p \in S}(N_x \cdot p)$. Consider the function $g_i(x) = N_x \cdot p_i$. This function is an affine function and it is also convex. Thus, the maximum defined by all the functions $(g_i)_{1 \leq i \leq n}$ is convex too. The function $h_S(x)$ can be rewritten as $Max_{p \in S}(N_x \cdot p) + Max_{p \in S}(-N_x \cdot p)$. By the same logic, the right side of this expression is also a convex function. Since the sum of two convex functions is convex, we conclude that $h_S(x)$ is convex.

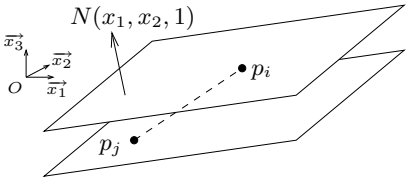

Fig. 1. Definition of $h_S(x_1, x_2)$

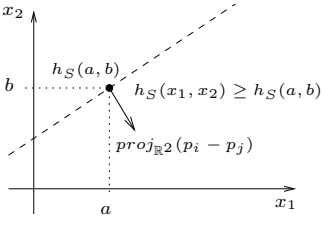

Fig. 2. A subgradient of h_S

2.2 Subgradient Computation

For a given value $x \in \mathbb{R}^{d-1}$, we only have to traverse the list of the points S in order to compute the value of $h_S(x)$. This implies that the computation of $h_S(x)$ has linear time complexity. Relative to the definition of $h_S(x)$, we know that:

$$h_S(x) = Max_{p \in S}(N_x \cdot p) - Min_{p \in S}(N_x \cdot p)$$

For a given value x, there exists two points p_i and p_j associated with the max and the min expressions (see Fig. 1). Therefore, $h_S(x) = N_x \cdot p_i - N_x \cdot p_j = N_x \cdot (p_i - p_j)$. As $T = \{p_i, p_j\}$ is included in S and we have:

$$\forall y \in \mathbb{R}^{d-1}, h_T(y) \leq h_S(y)$$

By definition, $h_T(y)$ is equal to:

$$h_T(y) = |N_y \cdot p_i - N_y \cdot p_j| = |N_y \cdot (p_i - p_j)|$$

and it follows:

$$\begin{aligned} h_T(y) &= |N_{y-x+x} \cdot (p_i - p_j)| \\ &= |N_x \cdot (p_i - p_j) + (y-x) \cdot proj_{\mathbb{R}^{d-1}}(p_i - p_j)| \\ &= |h_T(x) + (y-x) \cdot proj_{\mathbb{R}^{d-1}}(p_i - p_j)| \end{aligned}$$

Since $h_T(x)$ is positive, we have:

$$\begin{aligned} \forall y \in \mathbb{R}^{d-1}, h_S(y) &\geq h_T(y) \\ &\geq h_T(x) + (y-x) \cdot proj_{\mathbb{R}^{d-1}}(p_i - p_j) \end{aligned}$$

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

$$\forall y \in \mathbb{R}^{d-1}, h_S(y) \geq h_S(x) + (y-x) \cdot proj_{\mathbb{R}^{d-1}}(p_i - p_j) \quad (1)$$

We can now conclude that the expression $proj_{\mathbb{R}^{d-1}}(p_i - p_j)$ is a subgradient of $h_S(x)$ (see Fig. 2). This value corresponds to the projection of the vector p_ip_j in the $\mathbb{R}^{d-1}$ space. This means that the first $d-1$ components of the vector p_ip_j are sufficient to locally determine the variation of the function h_S. When we evaluate this function in linear time, we indirectly derive the two points p_i and p_j. Thus, in constant time, we determine one of its subgradients. For example in the three-dimensional case, for a set S of grid points represented as voxels (Fig. 3), h_S is a continuous piecewise affine function. We need to determine whether or not the domain defined by $h_S(x_1, x_2) < 1$ is empty or not (see Fig. 4).

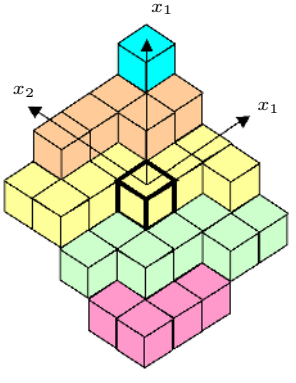

Fig. 3. A set of voxels

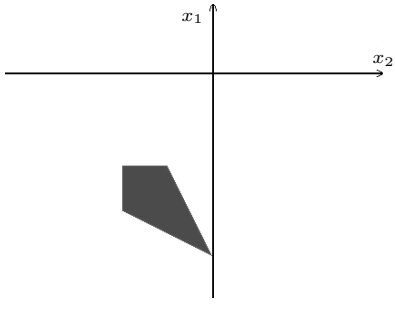

Fig. 4. Domain where $h_S(x_1, x_2) < 1$

3 The Basic Principles of Our Method

3.1 Studying the Solution Space

In the following, we focus our attention on the three-dimensional recognition problem. The core of our approach is based on the existence of the strict inequality in the double diophantine inequality. Analyzing $\mu \leq ax + by + cz <$

$\mu + ||(a,b,c)||_\infty$, which always produces integer values, the following inequality holds:

$$\mu \leq ax + by + cz \leq \mu + ||(a,b,c)||_\infty - 1$$

Most of the time, the voxels of an image are connected and so the diameter of the set of voxels is not very large. Suppose that the voxels we study all lie in a bounding box of size D. Let $||.||_\infty$ denote $||(x,y,z)||_\infty = max\{|w|,|y|,|z|\}$. Thus, any point p in this image satisfies $||p||_\infty \leq D$. In the same way, any vector k whose endpoints are supported by some voxels satisfies:

$$||k||_\infty \leq 2D$$

We can restrict our attention to the case where the naive digital planes are a mapping from (x,y) into z. This means that c is nonzero and that $||(a,b,c)||_\infty = |c|$. Wlog, we can force c to be positive. This produces $\mu' \leq a'x_i + b'y_i + z_i \leq \mu' + 1 - \frac{1}{c}$, where $a' = \frac{a}{c}$ and $b' = \frac{b}{c}$ represent the rational slopes of the digital plane. Relative to our previous assumption, we know that $|a'| \leq 1$ and that $|b'| \leq 1$. We can now determine the influence of a small variation on the normal vector $N(a',b',1)$. Let $N_\Delta(\alpha = \frac{r}{t}, \beta = \frac{s}{t}, 1)$ denote another normal vector in the neighborhood of N such that $|\alpha - a'| \leq \Delta$ and $|\beta - b'| \leq \Delta$. Thus, we have:

$$\mu' - 2D\Delta \leq \alpha x_i + \beta y_i + z_i \leq \mu' + 1 - \frac{1}{c} + 2D\Delta \tag{2}$$

Considering the convex hull of the set of points, we know that there exists two parallel planes that enclose all of the points and that have a minimal distance relative to the z-axis. These two planes are supported by four vertices of the convex hull [5]. The minimum of h_S is reached at a value associated with a normal vector whose coefficients are the vector product of two segments supported by the given voxels [7]. Therefore, there exists k and k' in $\mathbb{Z}^3$ such that the solution $(a,b,c) = k \wedge k'$. As $||k||_\infty \leq 2D$ and $||k'||_\infty \leq 2D$, we obtain:

$$1 \leq c \leq 8D^2 \tag{3}$$

Combining property 1 and inequality (2), we know that when S is a subset of a naive digital plane, any normal vector N_Δ in the neighborhood of N is valid iff: $4D\Delta < \frac{1}{c}$. It follows that when the set of solutions $h_S() < 1$ is not empty, it contains a *critical square* whose side length Δ is less than:

$$\Delta < \frac{1}{4Dc} \tag{4}$$

For example, from (3), we know that $\Delta < 1/(32 \cdot D^3)$. In this paper, we denote by Γ the value $\frac{1}{64D^3}$. In conclusion, we only have to study the function h_S relative to a uniform grid in the domain $[-1,1] \times [-1,1]$ whose resolution is given by Γ. If none of the sampled values represent a valid normal vector then the space of solutions is empty. Thus the solution space of our optimization problem reduces to a two-dimensional grid.

4 Algorithm Design

To attack our optimization problem, we could use standard algorithms from mathematical programming, including subgradient descent methods. However, we can not easily apply such algorithms. For instance, if we want to retain our philosophy of exact numerical computation with rationals, these methods will increase the size of the numerator and denominator of the normal vector at each iteration and slow down the calculation significantly. That is why the approach we present mixes methods developed in one-dimensional binary minimization, Megiddo Oracle technique and subgradient optimization. As we only pick simple techniques, we obtain a very simple approach.

4.1 Megiddo Cut

When Megiddo in [3] describes his optimization method, he cuts the current search domain by a hyperplane. If the optimum solution belongs to the cut, the problem is solved. If not, we create an Oracle (presented in section 4.3) that determines on which side of the cut the optimal solution lies. The search domain is then reduced to the solution space on the side of the cut in which the optimal solution lies. For example, suppose that the current search domain is $\{(a,b)|\{a,b\} \in [-1,1]\times[-1,1]\}$. We can choose to cut by the line $a = 0$. Next, we compute the minimum of h_S relative to this one-dimensional domain. After that step, if the optimum is less than 1, the problem is solved. Elsewhere, we call the Oracle and reduce the search domain to $[-1,0]\times[-1,1]$ or $[0,1]\times[-1,1]$. The next cut will be the line $b = 0$ and thus the resulting domains will be a square again. We perform the same operations at each iteration. When the search domain is smaller than the size of the critical square, we know that the set of solutions is empty. Using one iteration, we divide by two the side length of our search domain. Thus, in at most $log_2(2/\Gamma)$ iterations the program terminates.

4.2 One-Dimensional Binary Optimization

We know that h_S is a convex function. So, when we reduce its domain to a one-dimensional space $x_1 = e$, the resulting function h_S^e is always convex. Thus, we can apply usual optimization methods from one-dimensional convex optimization. We choose binary optimization, one of the simplest and most practical methods. For example, suppose that our two-dimensional search domain is cut by a segment $[(e,f),(e,g)]$. We choose the middle point: (e,m) with $m = \frac{f+g}{2}$ and compute a two-dimensional subgradient of h_S in $O(n)$ time. By projecting this subgradient into the one-dimensional domain, we obtain a subgradient for h_S^e at the point $\frac{f+g}{2}$ (see Fig. 5). So, we can determine which interval between $[(e,f),(e,m)]$ and $[(e,m),(e,g)]$ will be kept for the next iteration. As we work on a uniform grid, the optimization stops when the interval is bounded by two adjacent points of the grid. Thus, in $\log_2(|f-g|/\Gamma)$ iterations, we solve this subproblem.

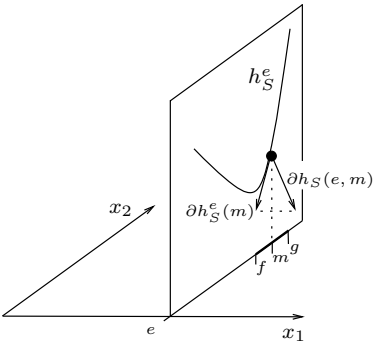

Fig. 5. Subgradient of $h_S^e : \partial h_S^e$

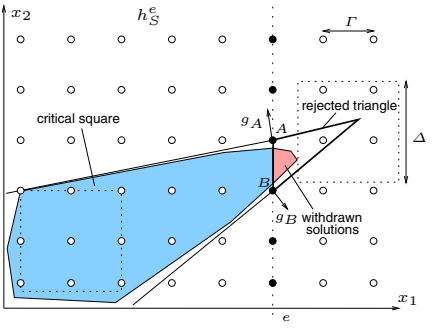

Fig. 6. Generalization of the Oracle

4.3 Subgradient and the Oracle Function

We want to determine on which side of the cut, the space of solutions lies. Megiddo Oracle computes the steepest descent from the minimum reached by h_S^e. This direction indicates on which side of the cut some solutions can lie. Nevertheless, as we work on a uniform grid, we can not determine the minimum of h_S^e. After the one-dimensional optimization step, we only know the two minimum values among the samples of the grid. Suppose that we cut by a vertical segment and that these two values are associated to two points A and B such that A lies over B (see Fig. 6). Let g_A and g_b denote the two subgradients of h_S computed at these points. When the angle (g_A, g_B) is equal to π the two subgradients are parallel and the search domain is restricted to a strip of vertical thickness equal to Γ. Thus, it can not contain a critical square and the problem has no solution. In the case where $g_A \wedge g_B < 0$, we keep the left side. However, even if the values of h_S^e at the points A and B are greater than 1, there may exist smaller values and, thus, valid solutions in the interval $]A, B[$. Selecting one of the two sides, we may withdraw some solutions that lie on the cut and on the rejected side. Nevertheless, if they exist, they lie in a triangle not large enough to contain a square of side length Δ. Thus, only the solutions lying in a critical square are of interest. Others can be withdrawn without perturbing the final result of the algorithm.

4.4 Program and Complexity

The code for the main function, **Naive Digital Plane Recognition()**, and for the one-dimensional support function, **1D-Optimization()**, used to solve the one-dimensional optimization problem follows. Each comparison processes all of the different cases.

The search domain is initialized to a square of side length 2. The program stops when the side length of the square currently being processed is less than Γ. At the i^{th} iteration, the side length is equal to $2/2^{i-1}$. Let n_i denote the number of loops performed by the main function. Thus, n_i is equal to $2 - log_2\Gamma$.

Naive Digital Plane Recognition(Set,n,D)
$\{P, Q, R, S\} = \{(-1,-1),(-1,1),(1,1),(1,-1)\}$ // the search domain
$\delta = 2 \quad \Gamma = \frac{1}{64D^3}$
do
{ // vertical cut
$MQR = (Q+R)/2 \quad MPS = (P+S)/2$
$(status, answer) =$ 1D-V-$Optimization(MQR, MPS, Set, n, \Gamma)$
if $(status = finished)$ **return** answer
if $(anwser = OnTheLeft) \quad \{R = MQR; S = MPS;\}$
else $\{Q = MQR; P = MPS;\}$
// horizontal cut
$MPQ = (P+Q)/2 \quad MRS = (R+S)/2$
$(status, answer) =$ 1D-H-$Optimization(MPQ, MRS, Set, n, \Gamma)$
if $(status = finished)$ **return** answer
if $(anwser = Above) \quad \{Q = MPQ; R = MRS;\}$
else $\{P = MPQ; S = MPS;\}$
$\delta = \delta/2$ //length of the search domain at the next iteration
} **while** $\delta \geq \Gamma$
return NotAPlane

1D-V-Optimization (A, B, Set, n, Γ)
$(value_A, g_A) = Evaluation(A, Set, n)$
if $(value_A < 1)$ **return** $(finished, A)$
$(value_B, g_B) = Evaluation(B, Set, n)$
if $(value_B < 1)$ **return** $(finished, B)$
do
{
M = (A+B)/2
$(value_M, g_M) = Evaluation(M, Set, n)$
if $(value_M < 1)$ **return** $(finished, M)$
if $(g_M.Y = 0)$
when $(g_M.X < 0)$ **return** $(continue, OnTheRight)$
$(g_M.X > 0)$ **return** $(continue, OnTheLeft)$
$(g_M.X = 0)$ **return** $(finished, NotAPlane)$
else
if $(g_M.Y < 0) \quad (B, g_B) \leftarrow (M, g_M)$
else $(A, g_A) \leftarrow (M, g_M)$
} **while** ($A.Y - B.Y \geq \Gamma$)
when $(g_A \wedge g_B < 0)$ **return** $(continue, OnTheLeft)$
$(g_A \wedge g_B > 0)$ **return** $(continue, OnTheRight)$
$(g_A \wedge g_B = 0)$ **return** $(finished, NotAPlane)$

Each iteration processes two instances of the one-dimensional subproblem. We can show that this subfunction performs $3 - log_2\Gamma - i$ evaluations of h_S during the vertical cut and $2 - log_2\Gamma - i$ evaluations during the horizontal cut of the i^{th} iteration. Thus, the overall number of evaluations is equal to:

$$\sum_{i=1}^{n_i}(5 - 2\log_2 \Gamma - 2i) = (2 - \log_2 \Gamma)^2 = (8 + 3\log_2 D)^2 \sim 9\log_2^2 D \qquad (5)$$

4.5 Calculation

The main idea is based on the following observation. Assume that we have just completed iteration i where we processed the two normal vectors $N_1(a, b, 1)$ and $N_2(a, b, 1)$. In the successive iteration, $i + 1$, of our program, we need to process the vector defined by $N = (N_1 + N_2)/2$. Instead of computing all scalar products of N with all of the points, we can build on the calculations of the previous step. Let P denote a point and P_N, P_{N_1} and P_{N_2} denote the scalar products between this point and the different normal vectors. Then we have $2P_N = P_{N_1} + P_{N_2}$. This allows us to write a complete version of our program without using a single multiplication. This property may be interesting in some models of computation.

5 Improving Performance

For $D = 512$ and $\Delta = \frac{1}{64D^3}$, the algorithm presented in section 4, processes about one thousand evaluations of h_S. One of the fastest known algorithm in practice [10] performs about ten traversals of the point set. The method presented in section 4.4 was a concise presentation of our main algorithm. In this section, we explore some modifications that will improve its performance in practice.

5.1 Initial Step

Starting with a search space equal to $[-1, 1] \times [-1, 1]$ is often awkward. If we assume that the set of points is contained in a bounding box of size D, then there exists some points that lie on the borders. Suppose we determine two points $P_1(-D, y, z_1)$ and $P_2(D, y, z_2)$. They provide one constraint on the space of solutions. As they belong to a naive digital plane of normal $N(\alpha, \beta, 1)$, we have: $\gamma' \leq N(-D, y_1, z_1) < \gamma' + 1$ and $\gamma' \leq N(D, y_2, z_2) < \gamma' + 1$. From this, it follows: $|N \cdot (2D, 0, z_2 - z_1)| < 1$. Thus, we can conclude:

$$\frac{z_1 - z_2}{2D} - \frac{1}{2D} < \alpha < \frac{z_1 - z_2}{2D} + \frac{1}{2D}$$

So, the side length of the search domain reduces by a factor $\frac{1}{D}$. This stage only requires a simple traversal of the points in order to find the extreme coordinates. The number of evaluations required for h_S reduces to $4\log_2^2 D$.

5.2 Subgradient Extension

We previously demonstrated that the knowledge of the gradient allows us to reject one part of the search domain relative to the point P. Nevertheless, we were not using all of the available information. In fact, when we know the value

of $h_S(P)$, either it is less than one and the problem is finished or it reaches a value greater than one. Recall that $\partial h_S(P)$ denotes the subgradient of h_S at the point P. From (1), we keep the points P' in the search domain that verify:

$$(P' - P) \cdot \partial h_S(P) < 0 \tag{6}$$

Any point P' is associated with a value of h_S less than $h_S(P)$. As we are only interested in the values less than 1, we can reformulate (1) as follows:

$$(P' - P) \cdot \partial h_S(P) < 1 - h_S(P)$$

In the search domain, this shifts the line defined by 6 towards the solution space. The new delineations are closer to it (see Fig. 7).

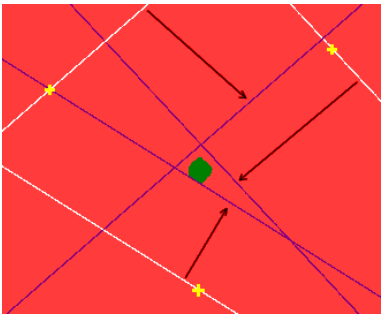

Fig. 7. Enhanced subgradient

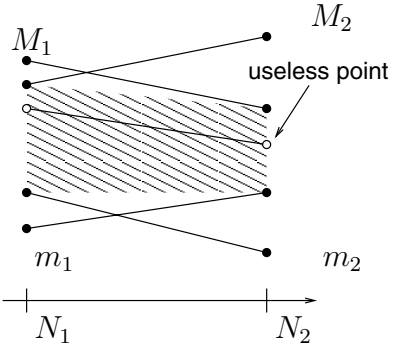

Fig. 8. The reduction criterion

5.3 History Memory

In lieu of using the information from the last subgradient computation and throwing it away after its first use, we prefer to retain the results of all of the previous subgradient computations in memory. Each time a subgradient is computed, we store the result in a list. When we want to test a point of the search domain, we do not immediately call an evaluation of h_S. Instead, we ask a more specific function if the tested point is compatible with all the results from the previous tests. This query is easy to solve because we only have to check the location of a planar point relative to a set of lines. When a previous subgradient g allows for the determination that the value of h_S is greater than 1, we do not need to perform the evaluation. The returned subgradient is equal to g. Suppose we create this function named **IsCompatible**. It could easily manage this processing and the conditions originating from the initial step. Moreover, the quantity of information we store is insignificant relative the number of voxels we process. Thus, this slight improvement will greatly increase the program efficiency in practice.

5.4 Enhancing Γ

In the previous section, we used an overestimated value for Γ in order to simplify our presentation. Nevertheless, we are not always forced to choose this extremely small value of $\frac{1}{64D^3}$. For example, we know that if there exists a valid normal vector (a, b, c) then the size Δ of the critical square is equal to $\frac{1}{4Dc}$, see (4). When we cut the search domain by a vertical line $x = \frac{r}{s}$, the one-dimensional problem would reach its exact minimum at a vector $N(\frac{r}{s}, \frac{u}{v}, 1)$. We determine the two points p_i and p_j that define the two supporting parallel planes. As the point N is a local minimum, there must exist a third point p_k that supports the two planes. Suppose that p_k is lying on the same plane as p_i. Then N is equal to $(p_i - p_k) \wedge (s, 0, r)$ and we have $||N||_\infty \leq s \cdot 2D$. So, when we perform a cut at the position $\frac{r}{s}$, it is sufficient to choose Γ equal to:

$$\Gamma_{\frac{r}{s}} = \frac{1}{16Ds}$$

In the worst case, where $s = 4D^2$, we find the previous definition of Γ. This improvement reduces the number of iterations. We must only modify the expression of Γ in the program, which is a simple operation.

5.5 Reduction

Consider a set of normal vectors that are linked to a segment in the search domain. Let N_1 and N_2 denote its two endpoints. When we determine $h_S(N_1)$, we obtain two points of S: P_1 and M_1 that define this value: $h_S(N_1) = N_1 \cdot (P_1 - M_1)$. Symmetrically, we use the same notation for N_2. Studying the other possible vectors lying on the segment, we notice that some points of S are useless (see Fig. 8). In fact when a point F of S satisfies this condition:

$$N_1 m_2 \leq N_1 F \leq N_1 M_2 \text{ and } N_2 m_1 \leq N_2 F \leq N_2 M_1$$

It can be rejected by the following precessing because it can not define the value of h_S. This condition generalizes to our search domain defined by four vertices. Although this criterion for reduction is not optimal, it is very simple to estimate, so we prefer to use it over other options. This approach allows for the suppression of a fraction of the input points and for a reduction in the cost of the subsequent evaluation of h_S. However, it must be used carefully because we do not know the ratio of the points we reject.

6 Conclusion

We describe a different approach for the naive digital plane recognition problem. We show how to transform the recognition process into a two-dimensional convex optimization program. The convex function we use corresponds to the minimal 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, we apply simple and well known geometric techniques in order to reduce its size until we find a solution. We exhibit a stopping criterion that only depends on the size D of a bounding box that encloses all the points. This version of our algorithm achieves an $O(n \log^2 n)$ time complexity in the worst case. We present different modifications used to speed up the optimization stage. Development of this enhanced method is in progress. The more difficult part is to use the same integer size (like 32 bits integer for example) for this algorithm and the other methods in order to equitably compare them. Moreover, it remains to estimate the yield of the reduction criterion in order to properly use it. This algorithm has been designed to be efficient in practice and especially when the set of points is dense in the bounding box.

The author thanks the reviewers for their helpful comments.

References

1. Brimkov, V., Coeurjolly, D., Klette, R.: Digital Planarity - A Review. Discrete Applied Mathematics, accepted for publication (2006)
2. Buzer., L.: A linear incremental algorithm for naive and standard digital lines and planes recognition. Graphical Models, **65**(1-3) (2003) 61-76
3. Megiddo, N.: Linear programming in linear time when the dimension is fixed. J. ACM, **31** (1984) 114-127
4. 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** (1991) 197-212
5. Debled-Rennesson, I.: Etude et reconnaissance des droites et plans discrets. PhD Thesis, Université Louis Pasteur, Strasbourg (1995)
6. Debled-Rennesson, I., Reveillès, J.-P.: A new approach to digital planes. Proc. Vision Geometry III, SPIE 2356 (1994) 12-21
7. Preparata, F. P., Shamos., M. I.: Computational Geometry: An Introduction. Springer, New York (1985)
8. Vittone, J., Chassery, J.-M.: Recognition of digital naive planes and polyhedrization. In Proc. Discrete Geometry for Computer Imagery, Springer, Berlin, LNCS 1953 (2000) 296-307
9. Francon, J., Schramm, J. M., Tajine, M.: Recognizing arithmetic straight lines and planes. In Proc. Discrete Geometry for Computer Imagery, LNCS 1176, Springer, Berlin (1996) 141-150
10. Gerard, Y., Debled-Rennesson, I., Zimmermann, P.: An elementary digital plane recognition algorithm. **151**(1-3) (2005) 169-183
11. Reveillès, J.-P.: Combinatorial pieces in digital lines and planes. In Proc. Vision Geometry IV, SPIE 2573 (1995) 23-34
12. Veelaert., P.: Digital planarity of rectangular surface segments. IEEE Trans. Pattern Analysis Machine Intelligence, **16** (1994) 647-652