View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Elsevier - Publisher Connector

ELSEVIER

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Discrete Applied Mathematics 151 (2005) 169–183

DISCRETE APPLIED MATHEMATICS

www.elsevier.com/locate/dam

An elementary digital plane recognition algorithm

Y. Gerard[b], I. Debled-Rennesson[a], P. Zimmermann[a]

[a]*LORIA, INRIA Lorraine, 615 rue du Jardin Botanique, BP 101, F-54600 Villers-lès-Nancy, France*
[b]*LLAIC, IUT, Ensemble Universitaire des Cézeaux, 63172 Aubière, France*

Received 1 October 2003; received in revised form 30 April 2004; accepted 10 February 2005
Available online 11 July 2005
Dedicated to Maurice Nivat

Abstract

A naive digital plane is a subset of points $(x, y, z) \in \mathbb{Z}^3$ verifying $h \leqslant ax + by + cz < h + \max\{|a|, |b|, |c|\}$, where $(a, b, c, h) \in \mathbb{Z}^4$. Given a finite unstructured subset of $\mathbb{Z}^3$, the problem of the *digital plane recognition* is to determine whether there exists a naive digital plane containing it. This question is rather classical in the field of digital geometry (also called discrete geometry). We suggest in this paper a new algorithm to solve it. Its asymptotic complexity is bounded by $O(n^7)$ but its behavior seems to be linear in practice. It uses an original strategy of optimization in a set of triangular facets (triangles). The code is short and elementary (less than 300 lines) and available on `http://www.loria.fr/~debled/plane`.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Naive digital plane; Chords set; Triangular facet; Linear programming

1. Introduction

The geometry of the $\mathbb{Z}$-module $\mathbb{Z}^n$ is often called *discrete* or *digital* in contrast to the classical *continuous* geometry of $\mathbb{R}^n$.

A digital hyperplane of $\mathbb{Z}^n$ is defined by a double inequality $h \leqslant \varphi(x) < h + \delta$, where φ is a linear form [17]. In dimension 2, it provides a definition of digital straight lines of $\mathbb{Z}^2$. With this general approach, diophantine and *Bresenham* straight lines are both digital straight lines [2]. In dimension 3, it provides a definition of digital plane.

E-mail addresses: gerard@llaic.u-clermont1.fr (Y. Gerard), Isabelle.Debled-Rennesson@loria.fr (I. Debled-Rennesson), Paul.Zimmermann@loria.fr (P. Zimmermann).

0166-218X/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2005.02.026

Definition 1. A digital plane is the set of the solutions $(x, y, z) \in \mathbb{Z}^3$ of a double inequality $h \leqslant ax + by + cz < h + \delta$, where $(a, b, c) \in \mathbb{Z}^3 - \{(0, 0, 0)\}$, $h \in \mathbb{Z}$ and $\delta \in \mathbb{Z}^+$. A digital plane is said to be naive if $\delta = \max\{|a|, |b|, |c|\}$.

The class of naive digital planes has good topological properties: a naive digital plane is 26-connected, its complementary has two 6-connected components and there is no simple point in the plane.

It is easy to determine whether there exists a digital plane containing a given finite set of points $S \in \mathbb{Z}^3$ because for any given $(a, b, c) \in \mathbb{Z}^3 - \{(0, 0, 0)\}$, the set S belongs to the digital plane of double inequality $h \leqslant ax + by + cz < h + \delta$ with $h = \min\{ax + by + cz | (x, y, z) \in S\}$ and $\delta = \max\{ax + by + cz | (x, y, z) \in S\} - h + 1$. However, it is not obvious to determine whether there exists a *naive* digital plane containing a given finite set of points.

Problem 2 (*Naive digital plane recognition*). Given a finite subset $S \subset \mathbb{Z}^3$, does there exist $(a, b, c, h) \in \mathbb{Z}^4$ verifying

$$\forall (x, y, z) \in S, \quad h \leqslant ax + by + cz < h + \max\{|a|, |b|, |c|\}?$$

In some applications (polyhedralization in $\mathbb{Z}^3$ and more generally image analysis or synthesis, ...), the problem can be asked in quite different terms. Let us assume that S is contained in a naive digital plane and given a point m, we search if $S \cup \{m\}$ is still contained in a naive digital plane. In this framework, an incremental algorithm is better.

Problem 2 has been the aim of several publications since 1990 (see Section 2). A well-known solution is to express the problem in terms of linear programming and to solve it by using a method of this field. Starting from the same point of view, we suggest in Section 3 a new algorithm as well as its incremental version and its proof. In Section 4, we give methods to check the optimality of the digital plane characteristics provided by the new algorithm. At last, in Section 5 experimental results are presented to show the efficiency of our algorithm.

2. Review

We can distinguish three kinds of methods for solving the digital plane recognition problem. They belong to the domains of combinatorics, linear programming and computational geometry.

The two-dimensional version of Problem 2, i.e., the digital line recognition, has been solved at the beginning of the 1990s by combinatorial tools [15,20]. A review of the digital line segment recognition algorithms is given in [19]. Debled-Rennesson associated with Reveillès have solved it in 1992 with an arithmetic method related to continued fractions and properties of Sturmian words [5]; this algorithm is incremental and linear but it is unfortunately difficult to use the same principles for a digital plane recognition algorithm; they have, however, proposed an incremental algorithm to recognize "rectangular" subsets of naive digital planes [6], demonstrated later in [16].

Still in the first half of the 1990s, Veelaert [21] has generalized in dimension three the evenness property of digital lines given by Hung [11] and used it to recognize rectangular subsets of digital planes [22].

Following a combinatorial approach, Vittone has provided an incremental digital plane recognition algorithm using the dual space [23].

Other works have followed another direction [3,7,10,13,14,18,20] writing the digital plane recognition problem as an integer linear programming problem.

Problem 3. Given a finite subset $S \in \mathbb{Z}^3$, find $(a, b, c, h) \in \mathbb{Z}^4$ satisfying one of the following systems of linear constraints:

$$\text{(A)}\ \forall (x, y, z) \in S, \quad h \leqslant ax + by + cz < h + a,$$

$$\text{(B)}\ \forall (x, y, z) \in S, \quad h \leqslant ax + by + cz < h + b,$$

$$\text{(C)}\ \forall (x, y, z) \in S, \quad h \leqslant ax + by + cz < h + c.$$

Problem 3 is defined by three systems of linear constraints (A), (B) or (C). Each one corresponds to a value of $\max\{|a|, |b|, |c|\}$ (the system (C) corresponds for instance to the case $\max\{|a|, |b|, |c|\} = |c|$). If one of them is feasible (i.e., its set of solutions is not empty), then S is contained in a naive digital plane. Otherwise, S is not contained in any naive digital plane. Thus, from now on, we consider that the digital plane recognition problem consists in solving the system of linear inequalities (C). As we look for integer solutions, this problem belongs to the domain of integer linear programming. Problems from this domain can be NP-hard in arbitrary dimension, but the fixed dimension and the homogeneity of system (C)—if (a, b, c, h) is a solution, then for any $\lambda \in \mathbb{R}+$, $(\lambda a, \lambda b, \lambda c, \lambda h)$ is also a solution—make it simpler. Our problem of integer linear programming can be solved by using classical algorithms of linear programming:

The constraints of (C) being rational, the linear programming algorithms provide rational solutions, and after multiplication by the denominators, one obtains integer solutions. If there exists no rational solution, there is no integer solution either.

Thus the digital plane recognition problem can be solved by any linear programming method providing a rational solution from constraints having rational coefficients. The Fourier–Motzkin algorithm [24] has been used in [7] to obtain theoretical results on tricubes, but the more efficient methods are the simplex algorithm [4], Megiddo's algorithm (see [3] for an incremental version providing a linear time algorithm for digital plane recognition) and interior points methods [1].

The third approach belongs to the domain of computational geometry. In 1984, Kim was already interested in digital plane recognition [12]. He proposed a method using the 3D convex hull of the considered set. This approach of computational geometry has been improved in [13]. In this framework, the digital plane recognition is reduced to a collision detection problem which consists in determining whether two polytopes intersect. It can be solved by computing the convex hulls of the polytopes. A more efficient strategy is to compute the minimal distance between the two polytopes as done by the GJK algorithm [10]. This well-known method is very efficient in practice with an average linear-time behavior. This algorithm is similar to the new algorithm that we present in next section; both work

on the chords sets, the main difference being that our method uses an optimization on the axis *Oz* instead of a distance. It leads to a different choice of linear forms and it makes an important difference between both algorithms.

3. New algorithm for naive digital plane recognition

3.1. Theoretical background: a notion of geometrical thickness

According to Section 2, the digital plane recognition problem can be reduced to the system of linear inequalities (C): find $(a, b, c, h) \in \mathbb{Z}^4$ verifying $\forall (x, y, z) \in S$, $h \leqslant ax + by + cz < h + c$. The feasibility of the system is related to a notion of thickness which requires some basic geometrical material.

Definition 4. The chords set of $S \subset \mathbb{R}^n$ is the set of differences between points of S (Fig. 1). We denote it by chord(S):

$$\text{chord}(S) = S + (-S) = \{m' - m / m, m' \in S\}.$$

The chords set of S is the Minkowski sum of S with its mirror $-S$. If the cardinality of S is n, the cardinality of chord(S) is at most $n(n-1)+1$.

We denote by conv(S) the convex hull of any subset S of $\mathbb{R}^n$. We can mention that the chords set of the operator which associates with a set S commutes with the convex hull: $\forall S \subset \mathbb{R}^n$, conv(chord$(S)$) = chord(conv$(S)$) [8]. This convex set conv(chord(S)) is used for defining the geometrical thickness of S in one direction.

Definition 5. The geometrical thickness of a finite set $S \subset \mathbb{R}^n$ in the direction *Oz* is the *z*-coordinate of the point of the *Oz* half-line (with $z \geqslant 0$) belonging to the surface of the convex hull of chord(S) (Fig. 2). We denote it by $\tau_{Oz}(S)$).

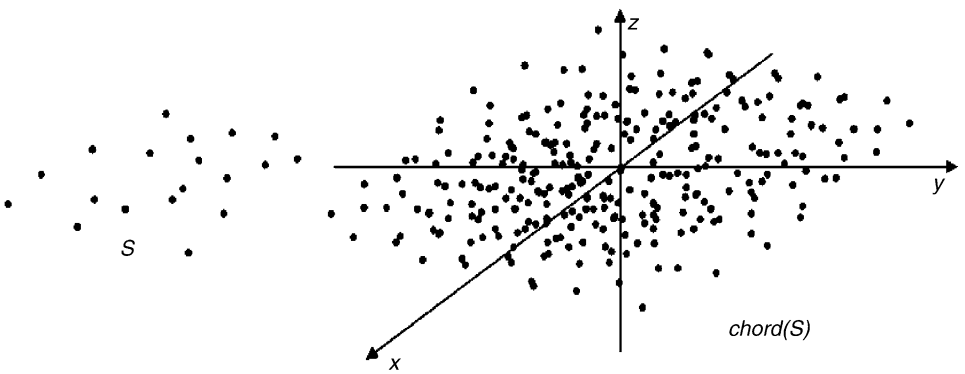

Fig. 1. A three-dimensional finite set S, card$(S) = 19$, and chord(S).

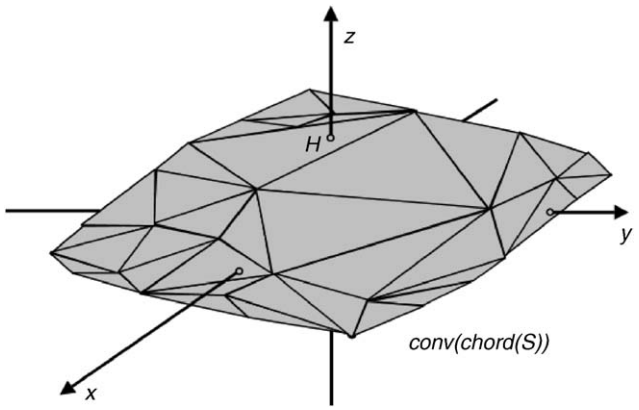

Fig. 2. A view of the convex hull of chord(S): the thickness of S in the direction Oz is the z-coordinate of the point H.

For simplicity, we simply say "the thickness of S" in the following, instead of "the geometrical thickness of S in the direction Oz".

Theorem 6. *Given a finite subset $S \subset \mathbb{Z}^3$, there exist $(a, b, c, h) \in \mathbb{Z}^4$ verifying $\forall (x, y, z) \in S$, $h \leqslant ax + by + cz < h + c$ if and only if $\tau_{Oz}(S) < 1$.*

Proof. Assume the system of linear constraints (C) is feasible: $\exists (a, b, c, h) \in \mathbb{Z}^4$, $\forall (x, y, z) \in S$, $h \leqslant ax + by + cz < h + c$. Let $m = (x, y, z)$ and $m' = (x', y', z')$ be two points of S. They both satisfy the double-inequality $h \leqslant ax + by + cz < h + c$ and then we have $-c < a(x - x') + b(y - y') + c(z - z') < c$. It proves that all the points $M = (x'', y'', z'')$ of chord(S) verify the double-inequality $-c < ax'' + by'' + cz'' < c$. Then it is also true with the points of the convex hull of the chords of S (conv(chord(S))): all the points of the convex hull of the chords of S verify the double-inequality $-c < ax'' + by'' + cz'' < c$. As the point $(0, 0, 1)$ does not satisfy $ax'' + by'' + cz'' < c$, it cannot belong to the convex hull of the chords of S; since the origin belongs to this convex set, its surface cuts the Oz half-line in a point of z-coordinate strictly less than 1. It proves that the thickness of S is less than 1 ($\tau_{Oz}(S) < 1$).

We assume now that the thickness of S in the direction Oz is strictly less than 1. The set conv(chord(S)) is a convex polytope of $\mathbb{Z}^3$ which contains the origin. Let us consider a facet of it—there can exist more than one—cutting the Oz half-line. By definition $(0, 0, \tau_{Oz}(S))$ is a point of this facet and we denote by (a, b, c) (with $c > 0$ and $(a, b, c) \in \mathbb{Z}^3$) its normal direction. For any point $M = (x, y, z)$ of conv(chord(S)) and in particular for any point M of chord(S), we have the inequality (i) $ax + by + cz \leqslant a0 + b0 + c\tau_{Oz}(S)$, namely (i) $ax + by + cz \leqslant c\tau_{Oz}(S)$.

We introduce now two points of S denoted by $m_{\min} = (x_{\min}, y_{\min}, z_{\min})$ and $m_{\max} = (x_{\max}, y_{\max}, z_{\max})$ and realizing the minimum and the maximum of $ax + by + cz$ on the finite set S. By denoting $h = ax_{\min} + by_{\min} + cz_{\min}$ and $H = ax_{\max} + by_{\max} + cz_{\max}$, we have the double-inequality (ii) $\forall (x, y, z) \in S$, $h \leqslant ax + by + cz \leqslant H$. It follows from inequality (i)

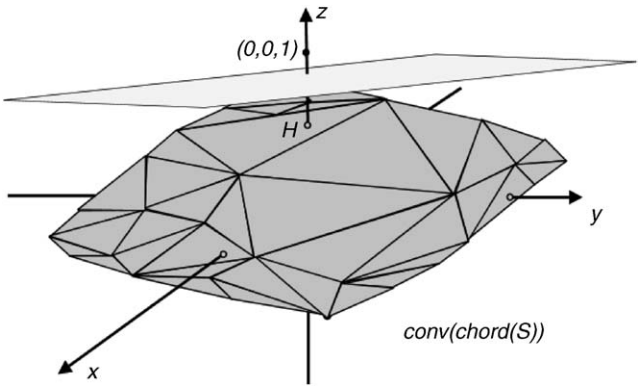

Fig. 3. Any plane separating the point $(0, 0, 1)$ from the convex polytope conv(chord(S)) provides a feasible (a, b, c) of the system of linear inequalities (C).

with $M = m_{\max} - m_{\min}$ that $a(x_{\max} - x_{\min}) + b(y_{\max} - y_{\min}) + c(z_{\max} - z_{\min}) \leqslant c\tau_{Oz}(S)$, namely $H - h \leqslant c\tau_{Oz}(S)$. The strict inequality $\tau_{Oz}(S) < 1$ leads to $H < h + c$ and (ii) implies $\forall(x, y, z) \in S$, $h \leqslant ax + by + cz < h + c$. It proves that (a, b, c, h) is a solution of the system of linear constraints (C). □

If the thickness of S in the direction Oz is greater than or equal to 1, then the system of linear constraints (C) has no solution. Otherwise, there exist solutions. The proof of Theorem 6 uses the face of the convex polytope conv(chord(S)) which cuts the Oz half-line ("under" the point $(0, 0, 1)$) because it provides an Euclidian plane strictly separating the point $(0, 0, 1)$ from the convex polytope conv(chord(S)). We could do the same with any Euclidian plane having this property. The normal directions of the Euclidian planes strictly separating conv(chord(S)) from $(0, 0, 1)$ are exactly the possible values (a, b, c) of the solutions of the system of linear inequalities (C) (Fig. 3). Anyone of these planes can be used to obtain a solution, but the affine hull of the face of the convex polytope conv(chord(S)) cutting the Oz half-line seems to be the easiest one to compute.

3.2. Naive plane recognition algorithm

We still consider a finite subset S of $\mathbb{Z}^3$ and we want to determine whether there exists a naive digital plane containing it. If the projection of S on one of the planes of coordinates belongs to a line then the answer is yes. From now on, we assume that it is not the case.

3.2.1. First level of the algorithm

We have seen in Section 2 that the problem can be reduced to find $(a, b, c, h) \in \mathbb{Z}^4$ verifying $\forall(x, y, z) \in S$, $h \leqslant ax + by + cz < h + c$ (system (C)). If this linear system of inequalities is not feasible, then we permute circularly the coordinates of S and look for a solution. If there is still no solution, we permute the coordinates a last time and restart the search. At the end, if we have found a solution, then there exists a naive digital plane containing S and otherwise, there exists no one.

3.2.2. A dynamic strategy to solve system (C)

How do we solve the system of linear inequalities (C)? Theorem 6 and its proof show that an answer is given by the face (or one of the faces) of the convex polytope conv(chord(S)) cutting the Oz half-line. This face gives the thickness of the set S in the direction Oz. If this number is greater than or equal to 1, then there is no solution; if it is strictly below, then there exist solutions and one of them is given by the normal direction of the face cutting Oz.

A first algorithm consists in computing the convex hull of chord(S)—complexity $O(\text{card}(S)^2 \log \text{card}(S))$—but we do not need this complete structure. Computing the whole convex hull would be too time consuming. Our algorithm is focused on a more precise purpose: computing a triangular facet on the surface of the convex hull of chord(S) which cuts the Oz axis. We proceed on the set of the triangular facets whose vertices belong to the chords set chord(S) and which cut the Oz axis. We call them triangles:

Definition 7. The 2-simplexes with vertices in chord(S) which cut the Oz axis are called triangles.

The *height* of a triangle T (Fig. 4) is the z-coordinate of the intersection point between T and Oz (if the intersection is not reduced to a point then the height of T is the maximal z-coordinate in the intersection but this case will not appear in the computation).

The face of the convex hull conv(chord(S)) which cuts the Oz half-line contains the highest triangle (in degenerated cases, one of the highest triangles). This triangle provides all the information that we need: its direction and its height are the ones of the face cutting Oz.

A naive algorithm would be to compute the height of all the triangles but it leads to a complexity of $O(\text{card}(S)^6)$. Instead of enumerating all triangles, we choose a strategy which improves at each step the height of a current triangle. This strategy is the following: given a current triangle T, we compute its normal vector v (with positive z-coordinate). We compute two points $m_{\min}$ and $m_{\max}$ belonging to S such that the scalar products $v \cdot m_{\min}$ and $v \cdot m_{\max}$ are, respectively, minimal and maximal. The point $M = m_{\max} - m_{\min}$ belongs

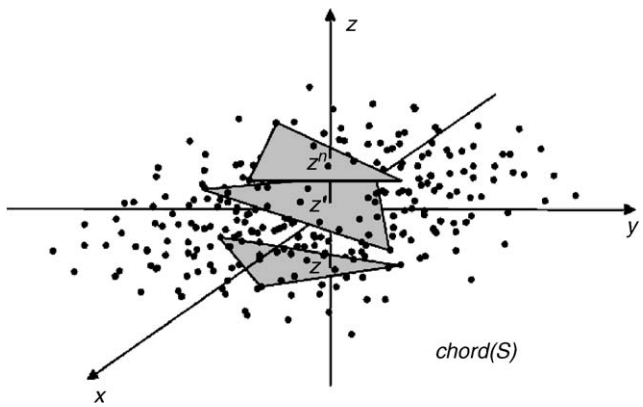

Fig. 4. Three triangles of the chords set as in Fig. 1 and of heights z, z', z''.

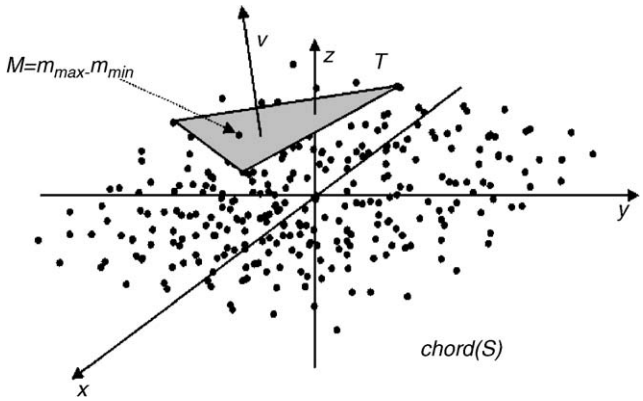

Fig. 5. The case where the point $M = m_{\max} - m_{\min}$ belongs to the plane of the current triangle T. In this case, the current triangle T belongs to the face of the convex hull of chord(S) cutting Oz and it is the (or one of the) highest triangle(s).

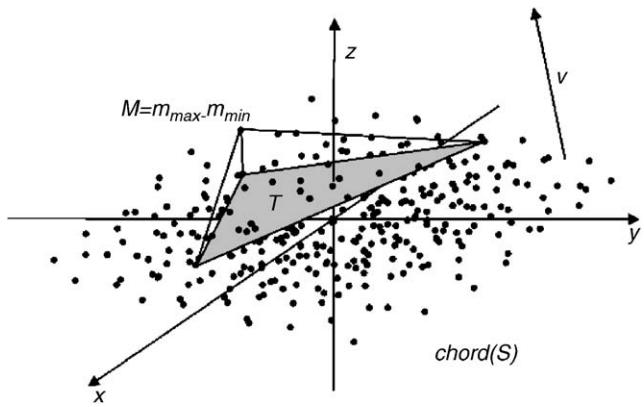

Fig. 6. The case where the current triangle T and $M = m_{\max} - m_{\min}$ define a non-degenerated tetrahedron. In this case, one of the faces of the tetrahedron is a higher triangle than T and we take it as new current triangle (if its height remains < 1).

to chord(S) and maximizes the linear form $(v\cdot)$ on chord(S). Two cases are possible:

- The point M is below the plane of T. In this case, M belongs in fact to the plane of T as illustrated in Fig. 5. This case happens if and only if T is on a face of the convex hull of chord(S). If the height of T is less than 1 then we have a solution (a, b, c are the coordinates of the normal vector v and h is given by $v \cdot m_{\min}$).
- Otherwise, the vertices of T define with M a tetrahedron (Fig. 6). This tetrahedron has four faces and by construction, one of them is a higher triangle than T (there is a special case considered below). If its height is $\geqslant 1$ then the problem has no solution. Otherwise, we take the highest triangle as new current triangle T.

This strategy provides a first algorithm (not incremental) for computing the highest triangle and we explain in the next remark how it can be transformed to solve system (C). We call this algorithm `FindHighestTriangle`. Its code is short and elementary. It uses four auxiliary routines: `FindStartingTriangle` finds a good starting triangle, `Normal`(T) finds the normal to the triangle T that is oriented toward the positive z-coordinates, `FindMinMax`(S, v) finds the point from chord(S) with maximal scalar product with v, and routine `FindCuttingTriangle`(T, M) returns the highest triangle of the tetrahedron formed by T and the point M.

Algorithm `FindHighestTriangle`.
Input: a finite subset S of $\mathbb{Z}^3$
Output: the highest triangle from chord(S) in the Oz direction
1. $T \leftarrow$ `FindStartingTriangle`(S)
2. **do**
3. $v \leftarrow$ `Normal`(T)
4. $M \leftarrow$ `FindMinMax`(S, v)
5. **if** $M \cdot v \leqslant T \cdot v$ **then** break
6. $T \leftarrow$ `FindCuttingTriangle`(T, M)
7. Return T.

Theorem 8. *Algorithm* `FindHighestTriangle` *is correct and terminates.*

Proof. At line 1, the routine `FindStartingTriangle` searches a triangle $T := (M_1, M_2, M_3) \in$ chord(S) whose projection on Oxy strictly contains the origin. A possible solution is to take $(M_1, M_2, M_3) = (m_1 - m_2, m_2 - m_3, m_3 - m_1)$, where m_1, m_2, m_3 are three points from S whose projections on Oxy are not aligned (we have assumed that the projection of S on Oxy does not belong to a line); this can be done in O(n) operations.

Line 3 computes the normal v to the triangle T, oriented toward the positive z-coordinates. Then `FindMinMax` at line 4 searches for a point $M \in$ chord(S) giving the maximal value of $M \cdot v$; this can be done in O(n) again by taking $M := m_{\max} - m_{\min}$, where $m_{\max}, m_{\min} \in S$ give, respectively, the maximal and minimal value of $m \cdot v$ for $m \in S$. If $M \cdot v \leqslant T \cdot v$—where $T \cdot v$ denotes the common value of $M_1 \cdot v$, $M_2 \cdot v$, $M_3 \cdot v$—at line 5, then all points M' of chord(S) satisfy $M' \cdot v \leqslant T \cdot v$, thus all points of chord(S) are "under" the triangle T, whose intersection with Oz is thus maximal. If not, then M is "above" the triangle $T := (M_1, M_2, M_3)$, and (M_1, M_2, M_3, M) is a tetrahedron which has exactly one triangle—apart from (M_1, M_2, M_3)—traversed by the Oz axis; `FindCuttingTriangle` finds that new triangle.

Since S is finite, there exists a finite number of triangles. The end of the algorithm follows because no triangle can be chosen more than once. The reason is that the height of the current triangle increases strictly at each step. There is, however, a particular case where this property does not hold. This case should be investigated precisely:

- Firstly, the starting triangle strictly contains the origin. Thus the second triangle—if any—intersects the Oz axis strictly higher than 0. It means that the origin can never be a vertex of the current triangle. The same holds for the other integer point of the Oz axis—at least for our particular problem—because if one of them is chosen as

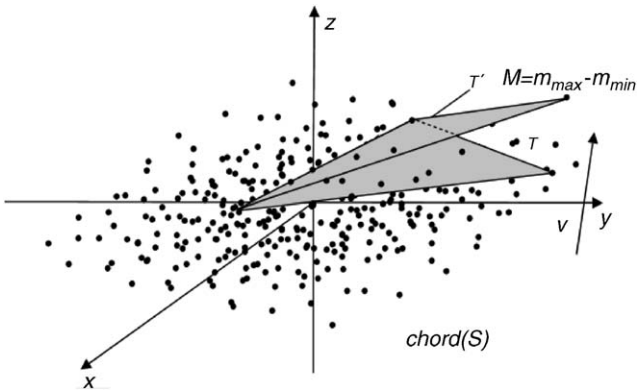

Fig. 7. The special case where there exists no higher triangle than the current triangle T in the tetrahedron defined by T and $M = m_{\max} - m_{\min}$. In this case, the triangle T' becomes the new current triangle.

new vertex, the triangle is higher than 1 and it ends the computation with a negative answer.

- Secondly, if one edge of the current triangle T cuts the Oz axis, it is possible that there exists no triangle higher than T in the considered tetrahedron but only a triangle with the same height (Fig. 7). In that case we choose the triangle with the largest angle between its normal and the Oz axis.

Thus it is not possible to use a triangle more than once. □

Remark 9. It is easy to improve the algorithm for our particular problem, where we want to determine if the highest triangle intersects Oz at $z < 1$: just return `FAIL` after line 3 when $T \cdot v \geqslant v_z$ (where v_z is the z-coordinate of v).

3.3. Complexity

The strategy chosen for improving the height of the triangles does not provide a better theoretical complexity than the naive algorithm. The complexity of the function `Find-HighestTriangle` is in $O(loop \cdot \text{card}(S))$, where the variable *loop* denotes the number of current triangles T used during the process. According to the previous section, this number is bounded by $\text{card}(S)^6$.

It provides a complexity in $O(\text{card}(S)^7)$ (as far as we know) which is worse than the one of the naive algorithm. Fortunately, this very high complexity bound does not correspond with the experimental behavior of the algorithm and it seems very difficult to build any example for which this upper bound is reached. The bound of complexity $O(\text{card}(S)^7)$ corresponds to a variable *loop* of the order of $\text{card}(S)^6$, but we do not know any set S where *loop* is larger than $\text{card}(S) + constant$ (let us say $constant = 10$). We know only some examples with a variable *loop* of the order of $\text{card}(S)$ but even these examples do not correspond to real data because the coordinates of the points of S grow exponentially with the cardinality of S.

3.4. Incremental version

We present here an incremental version of Algorithm `FindHighestTriangle`.

Algorithm `FindHighestTriangleIncremental`(S).
Input: a finite subset S of $\mathbb{Z}^3$
Output: the highest triangle from chord(S) in the Oz direction
1. $T := (m_1 - m_2, m_2 - m_3, m_3 - m_1) \leftarrow$ `FindStartingTriangle`(S)
2. $v \leftarrow$ `Normal`(T)
3. $S_0 \leftarrow \{m_1, m_2, m_3\}$, $m_{min} \leftarrow m_1$, $m_{max} \leftarrow m_1$
4. **for** $m \in (S - S_0)$ **do**
5. $\quad T, v, m_{min}, m_{max} \leftarrow$ `AddPoint`($T, v, m_{min}, m_{max}, S_0, m$)
6. $\quad S_0 \leftarrow S_0 \cup \{m\}$
7. Return T.

Function `AddPoint`($T, v, m_{min}, m_{max}, S_0, m$)
1. **if** $m \cdot v > m_{max} \cdot v$ **then** $m_{max} \leftarrow m$ **elif** $m \cdot v < m_{min} \cdot v$ **then** $m_{min} \leftarrow m$
2. **while** $(m_{max} - m_{min}) \cdot v > T \cdot v$ **do**
3. $\quad T \leftarrow$ `FindCuttingTriangle`($T, m_{max} - m_{min}$)
4. $\quad v \leftarrow$ `Normal`(T)
5. $\quad (m_{max}, m_{min}) \leftarrow$ `FindMinMax'`($S_0 \cup \{m\}, v$)
6. Return T, v, m_{min}, m_{max}.

The function `FindMinMax'` does not return a point of chord(S) like `FindMinMax` but the two points of S whose dot product with v are maximal and minimal.

Theorem 10. *Algorithm* `FindHighestTriangleIncremental` *is correct and terminates.*

Sketch of the proof. It follows from the following invariants: in algorithm `FindHighestTriangleIncremental`, T is a highest triangle of the current set of points S_0, and $m_{\min}$ (resp. $m_{\max}$) is a point of S which reaches the minimal (resp. maximal) value of $m \cdot v$ for $m \in S$. The function `AddPoint` checks if the new point $m \in S$ modifies the highest triangle, updates $m_{\min}$ and $m_{\max}$ in that case, and loops until the current triangle is the highest for $S_0 \cup \{m\}$.

Remark 11. The purpose of this incremental algorithm is still to find the highest triangle, but it is again easy to improve it to determine, given a set S, a maximal subset S_0 which is contained in a naive digital plane: just return FAIL after line 4 of function `AddPoint` when $T \cdot v \geqslant v_z$ (where v_z is the z-coordinate of v). In this case, $T, v, m_{\min}, m_{\max}$ should not be modified by line 5 of `FindHighestTriangleIncremental` and line 6 should not be executed (point m is rejected).

4. Optimality

In this section, we are interested in determining whether the digital plane containing a finite subset $S \subset \mathbb{Z}^3$ provided by `FindHighestTriangle` is the naive digital plane

containing S with the smallest possible coefficients (i.e. $|c|$ is minimal). It is not always true. It depends on the existence of integer points between the affine hull of the highest triangle and the plane going through (0, 0, 1) and having the same direction as the highest triangle. We prove in this section that if there are no integer points in this stripe, then the digital plane provided by `FindHighestTriangle` is optimal. It provides some tools to determine whether the digital plane computed with our method is optimal or not for the instance S.

We start with a few lemmas. We assume that $(a, b, c) \in \mathbb{Z}^3$, $c > 0$, $a \wedge b \wedge c = 1$ and we note $P(a, b, c, h)$ a naive digital plane such that

$$\forall (x, y, z) \in P(a, b, c, h), \quad h \leqslant ax + by + cz < h + c.$$

Lemma 12. *We have* $\tau_{Oz}(P(a, b, c, h)) = 1 - \frac{1}{c}$.

Proof. Let A, B and C be three upper leaning points of P (i.e. their coordinates verify $ax + by + cz = h + c - 1$), with not-aligned projected points on the plane $0xy$, and D a lower leaning point of P ($ax_D + by_D + cz_D = h$). In the chords set of P, the points $\alpha = D - A$, $\beta = D - B$ and $\gamma = D - C$ are on the superior face of the convex hull of chord(P), crossed by the Oz axis. Moreover, the equation of this face is $ax + by + cz = c - 1$. As the point $(0, 0, \tau_{Oz}(P(a, b, c, h)))$ belongs to this face, we obtain $c.\tau_{Oz}(P(a, b, c, h)) = c - 1$, therefore $\tau_{Oz}(P(a, b, c, h)) = 1 - \frac{1}{c}$. □

Lemma 13. *Considering a finite subset* $S \subset \mathbb{Z}^3$, *if* S *is included in* $P(a, b, c, h)$, *then*

$$\tau_{Oz}(S) \leqslant 1 - \frac{1}{c}.$$

Proof. We can notice that if $S \subset S'$ then $\tau_{Oz}(S) \leqslant \tau_{Oz}(S')$. With Lemma 11, $S \subset P(a, b, c, h)$ implies $\tau_{Oz}(S) \leqslant 1 - \frac{1}{c}$. □

We can of course reverse the inequality:

Corollary 14. *If a finite set* $S \subset \mathbb{Z}^3$ *is included in* $P(a, b, c, h)$ *then*

$$\frac{1}{1 - \tau_{Oz}(S)} \leqslant c,$$

Moreover, if

$$\left\lceil \frac{1}{1 - \tau_{Oz}(S)} \right\rceil = c,$$

then value c is minimal ($S \subset P(a', b', c', h') \Rightarrow c' \geqslant c$).

For a finite subset $S \subset \mathbb{Z}^3$ contained by a naive digital plane, the function `FindHighest-Triangle` provides a value c of a naive digital plane containing S (we denote it by $c_{\mathrm{FHT}}(S)$) and the thickness $\tau_{Oz}(S)$. Then we can test at the end of `FindHighestTriangle` the equality $\lceil 1/(1 - \tau_{Oz}(S)) \rceil = c_{\mathrm{FHT}}(S)$. If it is true, $c_{\mathrm{FHT}}(S)$ is minimal. This case appears quite often in practice. Otherwise, both cases can occur: $c_{\mathrm{FHT}}(S)$ can be minimal or not.

5. Experimental results

Fig. 8 gives experimental results with an implementation of algorithm `FindHighestTriangle` in GNU MP (or GMP for short). All computations are exact since we used the `mpz` class of GMP, which provides arbitrary–precision integers. We used a range of coefficients that increases with the cardinality n of the set S: a, b, c, h, x, y, z are taken uniformly at random in the range $[-100n, 100n]$.

From these results, it appears that the experimental complexity of algorithm `FindHighestTriangle` is almost linear (Fig. 9), which is confirmed by the fact that the number of loops necessary to find the highest triangle remains small.

We also implemented algorithm `FindHighestTriangle` using other types of coefficients: 24-bit single-precision or 53-bit double-precision floating-point numbers, or 32-bit integers. If the coefficients of the points of S are bounded by m, then the coefficients of chord(S) are bounded by $2m$, those of the normal v are bounded by $8m^2$, and those of the scalar products $M \cdot v$ are bounded by $48m^3$. So the coefficient type should be able to represent exactly integers in the range $[-48m^3, 48m^3]$. For single-precision floating-point numbers, it gives $m \leqslant 70$, for 32-bit integers we have $m \leqslant 355$, and for double-precision floating-point numbers, it gives $m \leqslant 57250$. Our experiments with double-precision numbers (Fig. 8) show that when this bound on coefficients holds, we get a speedup of about 10 with respect to GMP integers.

$n = \text{card}(S)$	10^2	10^3	10^4	10^5	10^6
max. coeff	10^4	10^5	10^6	10^7	10^8
tries	1000	1000	100	100	10
max. time	1ms	10ms	60ms	610ms	5.2s
aver. time	0.36ms	3.8ms	42ms	480ms	4.9s
max. loops	14	18	17	17	15
aver. loops	8.9	10.7	12.0	13.6	13.9

(a)

$n = \text{card}(S)$	10^2	10^3	10^4	10^5	10^6
max. coeff	57250				
tries	10^4	10^4	1000	100	10
aver. time	0.015ms	0.15ms	1.9ms	74ms	0.59s

(b)

Fig. 8. Experimental results on an Athlon XP 1700+ (a) with GMP 4.1.2 arbitrary precision integers and (b) with double-precision floating-point numbers.

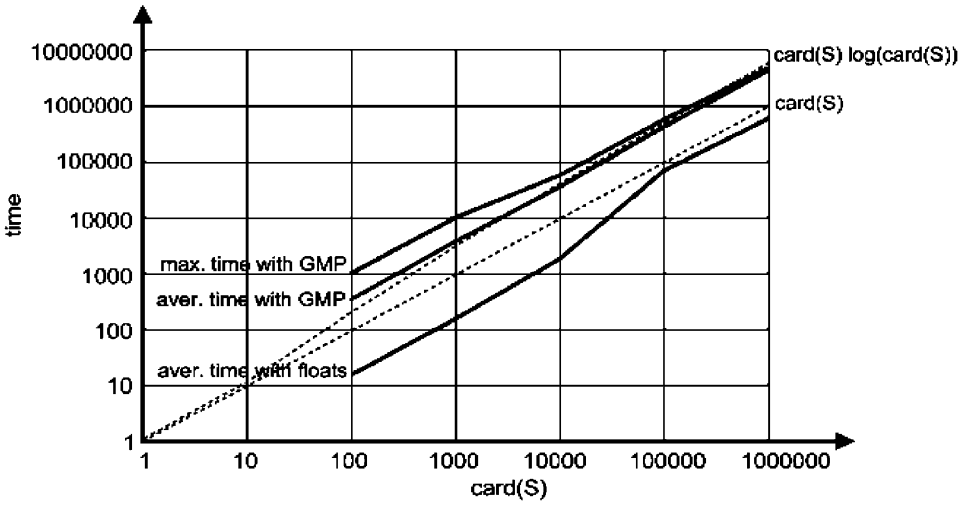

Fig. 9. The graphs of the times of computations (in microseconds) given in Fig. 8 as a function of the cardinalities of the sets, together with the curves $\mathrm{card}(S)\log(\mathrm{card}(S))$ and $\mathrm{card}(S)$.

6. Conclusion

We have presented a new algorithm for digital plane recognition, using an optimization strategy on the triangles from the chords set. The best asymptotic bound we could get for the complexity of this algorithm is $O(n^7)$; however, we could not find any family of inputs for which the number of operations is not $O(n^2)$, so we believe this $O(n^7)$ bound is very pessimistic.

The digital plane recognition problem can be solved in linear time, using Megiddo's algorithm for linear programming [3]; however, the code of this linear time method is really complicated and requires several thousands of lines. The purpose of this paper is to provide an efficient algorithm with a short and elementary code (cf. `http://www.loria.fr/~debled/plane`). We believe that this algorithm is faster *in practice* than the linear time one based on Megiddo's algorithm, whose constant is large. This claim is supported by the experiments we have done with several types of coefficients on many instances: the practical behavior of the algorithm seems to be linear, both using the arbitrary precision GMP library or fixed-precision machine numbers. It thus provides a fast and elementary algorithm for digital plane recognition.

References

[1] A. Ben-Tal, A. Nemirovski, Lectures on modern convex optimization: analysis, algorithms, engineering applications, MPS-SIAM Series on Optimization, SIAM, Philadelphia, 2001.
[2] J.E. Bresenham, Algorithm for computer control of digital plotter, IBM Systems J. 4 (1) (1965) 25–30.
[3] L. Buzer, An incremental linear time algorithm for digital line and plane recognition using a linear incremental feasibility problem, 10th International Conference on Discrete Geometry for Computer Imagery, Lecture Notes in Computer Science, vol. 2301, Bordeaux, France, 2002, pp. 372–381.

[4] V. Chvatal, Linear Programming, Freeman, New York, 1983.
[5] I. Debled-Rennesson, J.-P. Reveillès, A linear algorithm for segmentation of digital curves, Internat. J. Pattern Recognition Artif. Intell. 9 (1995) 635–662.
[6] I. Debled-Rennesson, J.-P. Reveillès, Incremental algorithm for recognizing pieces of digital planes, SPIE, Vision Geometry 5, vol. 2826, Denver, USA, 1996, pp. 140–151.
[7] J. Françon, J.M. Schramm, M. Tajine, Recognizing arithmetic straight lines and planes, Sixth International Conference on Discrete Geometry for Computer Imagery, Lecture Notes in Computer Science, vol. 1176, Lyon, France, 1996, pp. 141–150.
[8] Y. Gerard, Contribution à la géométrie discrète, Ph.D. Thesis, University of Auvergne, Clermont-Ferrand, 1999.
[10] E. Gilbert, D. Johnson, S. Keerth, A fast procedure for computing the distance between complex objects in three-dimensional space, IEEE J. Robotics Automation 4 (2) (1988) 193–203.
[11] S.-H.-Y. Hung, T. Kasvand, On the chord property and its equivalences, Seventh International Conference on Pattern Recognition, IEEE Computer Science Press, Montreal, Canada, 1984, pp. 116–119.
[12] C.E. Kim, Three-dimensional digital planes, IEEE Trans. Pattern Anal. Mach. Intell. PAMI 6 (1984) 639–645.
[13] C.E. Kim, I. Stojmenovic, On the recognition of digital planes in three dimensional space, Pattern Recognition Lett. 12 (1991) 665–669.
[14] R. Klette, H.J. Sun, Digital planar segment based polyhedrization for surface area estimation, Fourth International Workshop on Visual Form, Lecture Notes in Computer Science, vol. 2059, Capri, Italy, 2001, pp. 356–366.
[15] V.A. Kovalevsky, New definition and fast recognition of digital straight segments and arcs, 10th International Conference on Pattern Recognition, IEEE Computer Science Press, Atlantic City, USA, 1990, pp. 31–34.
[16] M. Mesmoudi, A simplified recognition algorithm for digital planes pieces, 10th International Conference on Discrete Geometry for Computer Imagery, Lecture Notes in Computer Science, vol. 2301, Bordeaux, France, 2002, pp. 404–416.
[17] J.-P. Reveillès, Géométrie discrète, Calcul en Nombres Entiers et Algorithmique, Thesis, Louis Pasteur University, Strasbourg, 1991.
[18] J.-P. Reveillès, Combinatorial pieces in digital lines and planes, SPIE, Vision Geometry IV, vol. 2573, San Diego, USA, 1995, pp. 23–24.
[19] A. Rosenfeld, R. Klette, Digital straightness, Eighth International Workshop on Combinatorial Image Analysis, Electronic Notes in Theoretical Computer Science, vol. 46, Philadelphia, Pennsylvania, USA, 2001.
[20] D. Sarkar, I. Stojmenovic, Parallel algorithms for separation of two sets of points and recognition of digital convex polygons, Internat. J. Parallel Programming 21 (2) (1992) 109–121.
[21] P. Veelaert, On the flatness of digital hyperplanes, J. Math. Imaging Vis. 3 (1993) 205–221.
[22] P. Veelaert, Digital planarity of rectangular surface segments, IEEE Pattern Anal. Mach. Intell. 16 (6) (1994) 647–652.
[23] J. Vittone, J.M. Chassery, Recognition of digital naive planes and polyhedrization, Ninth International Conference on Discrete Geometry for Computer Imager, Lecture Notes in Computer Science, vol. 1953, Uppsala, Sweden, 2000, pp. 296–307.
[24] G. Ziegler, Lectures on polytopes, Graduate Texts in Mathematics, vol. 152, Springer, Berlin, first rev. ed., 1995, Corr. 2nd printing, 1998.