Journal of Mathematical Imaging and Vision manuscript No.
(will be inserted by the editor)

An Optimized Framework for Plane-Probing Algorithms

Jacques-Olivier Lachaud · Jocelyn Meyron · Tristan Roussillon

Received: date / Accepted: date

Abstract A plane-probing algorithm computes the normal vector of a digital plane from a starting point and a predicate "Is a point x in the digital plane?". This predicate is used to probe the digital plane as locally as possible and decide on-the-fly the next points to consider. However, several existing plane-probing algorithms return the correct normal vector only for some specific starting points and an approximation otherwise, e.g. the H- and R-algorithm proposed in Lachaud et al. (J. Math. Imaging Vis., 59, 1, 23–39, 2017). In this paper, we present a general framework for these plane-probing algorithms that provides a way of retrieving the correct normal vector from *any* starting point, while keeping their main features. There are $O(\omega \log \omega)$ calls to the predicate in the worst-case scenario, where ω is the thickness of the underlying digital plane, but far fewer calls are experimentally observed on average. In the context of digital surface analysis, the resulting algorithm is expected to be of great interest for normal estimation and shape reconstruction.

Keywords Digital Plane Recognition · Normal Estimation · Plane-Probing Algorithm.

This work has been partly funded by CoMeDiC ANR-15-CE40-0006 and PARADIS ANR-18-CE23-0007-01 research grants.

Jacques-Olivier Lachaud
Univ. Grenoble Alpes, Univ. Savoie Mont Blanc, CNRS, LAMA, 73000 Chambéry, France
E-mail: jacques-olivier.lachaud@univ-smb.fr

Jocelyn Meyron
Univ. Lyon, INSA Lyon, LIRIS, UMR CNRS 5205, F-69622, France
E-mail: jocelyn.meyron@insa-lyon.fr

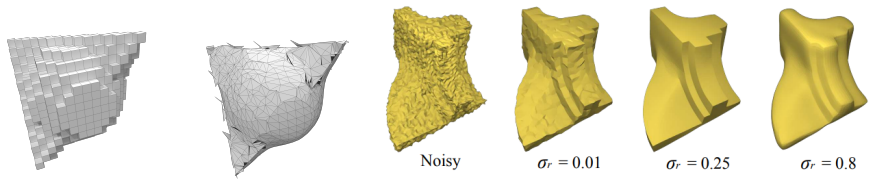
Tristan Roussillon
Univ. Lyon, INSA Lyon, LIRIS, UMR CNRS 5205, F-69622, France
E-mail: tristan.roussillon@liris.cnrs.fr

1 Introduction

In numerous fields, e.g., material sciences, medical imaging, non-invasive acquisition devices such as magnetic resonance, X-ray tomography or micro-tomography are required for observation, measurements or diagnostic aids. These acquisition devices usually generate volumetric data, i.e., 3D images, composed of regularly spaced data in a cuboidal domain. 3D volumes come from the segmentation of such 3D images. They are also generated in scientific modeling because numerous simulation schemes rely on the regularity of the data support.

Digital surfaces form the boundary of 3D volumes. Their geometry is difficult to analyze because for any resolution a digital surface is only made up of quadrangular surface elements, whose normal vector is parallel to one axis. A standard approach consists in choosing a larger computation window in order to estimate the geometry by some kind of smoothing or averaging method. It is even possible to get convergent estimates of the normal vector field along digitizations of sufficiently smooth shapes [11,21]. However, this approach has two drawbacks. First it comes at the cost of blurring sharp features. The trade-off between a sufficiently large neighborhood to get a relevant normal direction and a sufficiently small neighborhood to preserve sharp features is hard to find and may vary across the digital shape. Second, these approaches ignore the arithmetic nature of the geometry of digital surfaces. At a given digitization scale, this implies that in planar zones the computation window size depends on the arithmetic direction of the normal vector. Purely digital methods have thus emerged and try to perform digital surface analysis without any external parameters. We review in the next paragraphs some of the main methods that can be used to infer the geometry of digital surfaces, especially the ones that focus on estimating their normal vector field, since this information is the first step toward deeper geometric analysis. Indeed, many high-level tasks in computer graphics, vision and 3D image analysis rely on the quality of the normal estimation on digital surfaces: rendering, surface fairing, surface deformation for physical simulation or tracking, scene understanding, precise geometric measurements, primitive extraction, etc.

Generic methods for normal inference from point clouds. Of course, a first way to analyze a digital surface is to consider its vertices as a point cloud that samples some continuous surface. There exists a lot of methods to infer the geometry of a point cloud. The simplest approach is to compute the k-nearest neighbors and fit a plane to these points. In our case, it is hard to find a number k that is meaningful on the whole digital surface, so results would be considerably biased for any choice. More evolved methods are related to the Voronoi diagram of the point cloud, like cocone [13] and its many variants. However the point cloud must sample the continuous surface or be very close to it for these methods to be reliable. The Voronoi Covariance Measure (VCM) [26] is interesting in our case since it provides provable stability of inferred normals when the point cloud is close to the continuous surface.



(a) input (b) denoised (c) Influence of parameters

Fig. 1: Limited applicability of mesh denoising methods to digital surfaces. Here, we have chosen the recent and representative guided mesh normal filtering method [34]. (a) Input digital surface and (b) denoised digital surface, using the program provided by authors. (c) Choice of parameters is crucial even on standard meshes – image taken from [34].

This is our case since we know that the digitization of a continuous surface is close to it in the Hausdorff sense [25]. Further works have also shown that this approach can be extended to point clouds with outliers [10]. A drawback is that the VCM involves several parameters that depend on the geometry of the point sample. Typical parameter setting would oversmooth normals near features of the digital surface. On the contrary, the statistical approach of [2] and its follow-up [3] recognize the piecewise smooth aspect of normal vector fields and are efficient on point clouds with sharp features and with Gaussian random noise. It would still be difficult to find the correct set of parameters for their algorithm in the case of digital surfaces, and the possible improvements provided by deep learning is yet unclear in our case.

Geometry inference through mesh denoising. When processing triangulated or quadrangulated meshes, most works do not distinguish between noise in vertex positions or bad normals for faces. Inferring a good geometry is thus achieved in general by regularizing the mesh, often called mesh denoising. Bilateral mesh filtering is a popular approach [15], but many others are possible like L_0-minimization [19] or the more recent guided mesh normal filtering [34] or Mumford-Shah mesh processing [1]. As illustrated on Fig. 1, the tuning of parameters is crucial for these approaches, and it is difficult to find a correct set of parameters for digital surfaces. Furthermore all these approaches are variational by nature, they have no guarantee to converge toward the correct solution and require a lot of computations.

Normal estimation methods dedicated to digital surfaces. As said in the beginning, most methods for estimating the normal vector field of a digital surface perform a local averaging of the face trivial normals, e.g., [16]. More recently, several methods based on a variable radius/window of computation have been shown to be multigrid-convergent, like the digital integral invariant approach [9,21] or a digital variant of the VCM [11]. The established convergence requires digitizations of smooth surfaces, and the radius of computation depends

on the digitization step. These approaches oversmooth features and we must still rely on an external parameter to infer an accurate normal vector field. We finally note that the variational model of [8] is able to estimate piecewise smooth normal vector field over digital surfaces from previous normal estimations, at the price of further parameters (especially the user-chosen length of discontinuities) and increased computations.

Greedy decomposition into digital plane segments. The previous approaches do not really exploit the specific nature of digital surface data and their arithmetic properties. We therefore turn our attention to more specific digital geometry methods. A natural approach, which was successfully used in 2D, is to try to recover the linear geometry of the digital surface. The idea is to find digital plane segments (DPSs) that locally fit the digital surface. Such strategy has been used for surface area estimation [20], reversible polyhedrization [17,31], surface segmentation and smoothing [28], piecewise smooth reconstruction [7], or normal estimation [6]. Note that the difficulty is not to recognize if a set of digital points is a DPS, since there are numerous methods to do so, e.g., [32,12, 27,4,18,5,14,33], to quote a few. All these DPS recognition algorithms take a point set as input, possibly in an incremental way, determine whether this set can be a DPS or not, and if so, provide its geometric characteristics. The most difficult part consists in determining which input points should be taken into account during the recognition process in order to guarantee that the obtained DPSs are indeed *tangent* to the digital surface. For instance, all the approaches listed above rely on heuristics to determine a candidate set of points for DPS recognition: greedy decomposition [20,31], repetitive identification of largest DPS [7] or approximately largest [28], expansion from maximal planar disks [6].

Plane-probing algorithms. Therefore, recently, another category of recognition algorithms has been developed [22–24,30]. These algorithms, called *plane-probing* algorithms in [24], decide on-the-fly where to probe the digital surface in order to grow a tetrahedron, which is tangent by construction. The growth direction is given by both arithmetic and geometric properties. The main characteristics of these algorithms is that *no parameter* is required for the analysis of the local geometry of digital surfaces. Furthermore, they present theoretical guarantees, most notably they extract the exact normal vector of any digital plane. Henceforth, these approaches have not only a correct asymptotic behavior, but they provide a correct output at a given scale.

A new framework for plane-probing algorithms. In this paper, we present a framework that gathers the advantages of both [22] and [23,24,30]. It provides a way of locally computing the normal vector of a digital plane from any starting point. Denoting by ω the thickness of the underlying digital plane, we prove in Theorem 2 that there are $O(\omega \log \omega)$ probed points in the worst-case scenario. We prove that the returned normal vector is correct in Corollary 2.

We experimentally show that the proposed method is local and even give a formal proof for specific starting points characterized in Theorem 4.

Outline of the paper. After a thorough review of plane-probing algorithms in section 2, the motivation and outline of the proposed framework are presented in section 3.1. The rest of section 3 provides all necessary details. Theoretical results and proofs are gathered in section 4. ~~We review plane-probing algorithms in more detail in section 2. We present a general framework for some of the reviewed plane-probing algorithms in section 3. In section 4, we prove that our method is a way of locally computing the correct normal vector of a digital plane, *from any starting point*. In addition, denoting by ω the thickness of the underlying digital plane, we prove that there are $O(\omega \log \omega)$ probed points in the worst-case scenario.~~ In section 5, we conduct an experimental analysis in order to assess the number and position of the probed points on average. The proposed method is also shown to be relevant for analyzing the local planar geometry of digital surfaces.

2 Review of Plane-Probing Algorithms

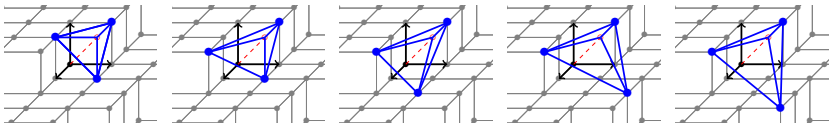
A *digital plane* is an infinite digital set that consists of several consecutive and parallel layers of coplanar points. It is defined by a (nonzero) normal $\mathbf{N} \in \mathbb{Z}^3$ and a position $\mu \in \mathbb{Z}$ as follows [29]:

$$\mathbf{P}_{\mu,\mathbf{N}} := \{\vec{x} \in \mathbb{Z}^3 \mid \mu \leq \vec{x} \cdot \mathbf{N} < \mu + \|\mathbf{N}\|_1\}. \tag{1}$$

Given a digital plane $\mathbf{P}_{\mu,\mathbf{N}} \subset \mathbb{Z}^3$ and a starting point $\vec{p} \in \mathbf{S}$, a *plane-probing algorithm* computes the parameters of a digital plane $\mathbf{P}_{\mu',\mathbf{N}'}$ containing $\vec{p}$ by sparsely probing $\mathbf{P}_{\mu,\mathbf{N}}$ with the predicate InPlane := "is $\vec{x}$ in $\mathbf{P}_{\mu,\mathbf{N}}$?". When the algorithm terminates, μ and $\mathbf{N}$ are expected to be equal to μ' and $\mathbf{N}'$ respectively.

What makes plane-probing algorithms promising is that they decide on-the-fly how to probe the digital surface and make grow a piece of digital plane, which is tangent by construction. The first algorithm of this category has been proposed in [22]. Its principle is to deform an initial unit tetrahedron, positioned at some starting point, with only unimodular transformations. Each transformation is decided by looking mostly at a few points around the tetrahedron. These points are chosen so that the transformed tetrahedron lies in $\mathbf{P}_{\mu,\mathbf{N}}$, with the same volume, but closer to the upper plane $\mathcal{P}^+_{\mu,\mathbf{N}} := \{x \in \mathbb{R}^3 \mid x \cdot \mathbf{N} = \mu + \|\mathbf{N}\|_1 - 1\}$. At the end of this iterative process, one face of the tetrahedron has an extremal position in the plane and is thus parallel to $\mathcal{P}^+_{\mu,\mathbf{N}}$.

New plane-probing algorithms were proposed in [23, 24]. They also iteratively deform an initial tetrahedron and stop when one face is parallel to $\mathcal{P}^+_{\mu,\mathbf{N}}$ (see Fig. 2 for an illustration). However, these new algorithms differ from the first work on several aspects. First, they are simpler, because they repeat one simple operation instead of several possible transformations depending on the



(a) (b) (c) (d) (e)

Fig. 2: The current tetrahedron is depicted in blue. It is initialized at a corner (a) and it is then deformed step by step by the R-algorithm proposed in [24] (b-e). When the algorithm terminates (e), the normal of the facet not incident to the fixed vertex (depicted with a red point) is equal to the normal of the underlying digital plane, i.e., $(1, 2, 5)$.

current configuration as in [22]. Second, one vertex of the evolving tetrahedron is a fixed point lying above the starting point and the opposite triangular facet (red point in Fig. 2). The position of the evolving tetrahedron is thus better controlled than in [22]. Third, a geometric criterion has been introduced in [23, Algorithm 2] and is also used for the two algorithms proposed in [24], called H- and R-algorithm, so that the evolving tetrahedron is not too much elongated during the computation. Last, these algorithms do extract the exact normal vector to a digital plane, but only when starting from specific positions, and otherwise return only approximations of the normal vector.

The H- and R-algorithm follow the same general procedure but use two different candidate sets to probe next the digital surface. The candidate set of the H-algorithm consists of six points that form a *H*exagon, whereas the candidate set of the R-algorithm consists of six *R*ays. Each ray starts from a vertex of the previous hexagon, which means that the candidate set of the H-algorithm is included in the one of the R-algorithm.

The H-algorithm is essentially the same but slightly different from [23, Algorithm 2] because only one point of the hexagon is selected at each iteration, instead of possibly several ones in case of cospherical points in [23, Algorithm 2]. This choice leads to a simpler characterization because two consecutive triangles must share exactly two vertices, instead of one or two, but do not significantly change the running time or the results.

To end, an optimized version of the R-algorithm has also been proposed in [30]. It returns the same triangular facet – and normal vector – as the R-algorithm, but requires fewer probes. It is called R^1-algorithm because only a few points along one ray over six are probed in the worst case.

Regarding the quality of the results, we have observed that the R-algorithm is the most *local* algorithm. Indeed, it computes a sequence of tetrahedra generally closer to the starting point than the one computed by other algorithms. This observation is somehow related to the concept of *lattice reduction*. Let us recall that the plane-probing algorithms proposed in [23, 24] iteratively deforms an initial tetrahedron. One vertex, denoted by q, is however fixed and is always projected into the opposite triangular facet, denoted by $\mathbf{T} := (v_0, v_1, v_2)$. For any permutation σ over $\{0, 1, 2\}$, $\mathbf{T}$ defines a 2D lattice embedded in

algorithm	principle	initialization	candidate set
[22]	deforms and moves 4 vertices of an upward-oriented tetrahedron inside $\mathbf{P}$	any 4 points in $\mathbf{P}$	6 points in a triangle above + rays
[23, Alg. 1]	deforms 3 vertices of a downward-oriented tetrahedron inside $\mathbf{P}$ with a fixed 4th vertex q outside $\mathbf{P}$	any reentrant corner: 3 inside $\mathbf{P}$, point q outside	6 points in a hexagon around q
[23, Alg. 2]			
[24, H-alg.]			
[24, R-alg.]			6 points in a hexagon around q + 6 rays
[30, R^1-alg.]			6 points in a hexagon around q + 1 ray
PH-alg.	deforms 8 vertices of a parallelepiped with at least 1 inside and 1 outside $\mathbf{P}$	any surfel: 4 inside $\mathbf{P}$, at least 1 point q outside	6 points in a hexagon around q
PR^1-alg.			6 points in a hexagon around q + 1 ray

Table 1: Principle and evolution of plane-probing algorithms. The last two lines sum up our contribution. $\mathbf{P}$ is the digital plane.

3D: $\Lambda := \{v_{\sigma(0)} + k(v_{\sigma(1)} - v_{\sigma(0)}) + k'(v_{\sigma(2)} - v_{\sigma(1)}) \mid k, k' \in \mathbb{Z}\}$. The basis $(v_{\sigma(1)} - v_{\sigma(0)}, v_{\sigma(2)} - v_{\sigma(1)})$ is *reduced* if and only if they are the two shortest nonzero vectors of Λ.

We ran all algorithms on all vectors ranging from (1,1,1) to (200,200,200). There are 6578833 vectors with relatively prime components in this range. We have observed that the R-algorithm, unlike all others, always returns a reduced basis. However, less than 0.01% of bases computed by the H-algorithm were non-reduced. In addition, we have found that there always exists a sequence of choices, in case of cospherical candidate points, such that all bases computed in the range (1,1,1) to (200,200,200) were reduced.

In what follows, we assume w.l.o.g. that $\mu = 0$ and we simply write $\mathbf{P}$ for $\mathbf{P}_{0,\mathbf{N}}$. We also assume w.l.o.g. that the components $(a, b, c) \in \mathbf{N}^3$ of the normal vector $\mathbf{N}$ are such that $0 \leq a \leq b \leq c$, and $\gcd(a, b, c,) = 1$. For any point $\vec{x} \in \mathbb{Z}^3$, the quantity $\vec{x} \cdot \mathbf{N}$ is called the *height* of x. The height of $(1, 1, 1)$, i.e., $(1, 1, 1) \cdot \mathbf{N} = \|\mathbf{N}\|_1$ is denoted by ω and called *thickness*. It is a key quantity because the complexity of all plane-probing algorithms depends on it. Table 1 and Table 2 sum up the differences and similarities between existing plane-probing algorithms, including our contribution.

Contribution. In the next section, we present a new variant of the H-algorithm, called PH-algorithm, because we deform a *P*arallelepiped, instead of a tetrahedron. It gathers in a unique framework the advantages of both [22] and [24, H-alg.], i.e., arbitrary starting point, exact normal extraction, good localness and almost always reduced bases. We could have presented similar variants

algorithm	complexity	observed	reduced basis	local	output
[22]	$O(\omega \log \omega)$	$c_3 \log \omega$	6%	no	$\mathbf{N}$ in all cases
[23, Alg. 1]	$O(\omega)$	$c_2 \log \omega$	27%	yes	$\mathbf{N}$ only if the height of q is ω
[23, Alg. 2]	$O(\omega)$	$c_2 \log \omega$	99.99%		
[24, H-alg.]	$O(\omega)$	$c_2 \log \omega$	99.99%		
[24, R-alg.]	$O(\omega \log \omega)$	$c_5 \log \omega$	100%		
[30, R^1-alg.]	$O(\omega)$	$c_1 \log \omega$	100%		
PH-alg.	$O(\omega \log \omega)$	$c_6 \log \omega$	99.99%	yes in practice	$\mathbf{N}$ in all cases
PR^1-alg.		$c_4 \log \omega$	100%		

Table 2: Main properties of plane-probing algorithms. If $\mathbf{P}$ is a digital plane of normal $\mathbf{N}$, let $\omega = \|\mathbf{N}\|_1$ be its thickness. The complexity is given in terms of the number of calls to predicate InPlane. Constants $(c_i)_{i=1\ldots6}$ are sorted so that $c_1 < c_2 < c_3 < c_4 < c_5 < c_6$ (see also Table 5 for more details). The last two lines sum up our contribution.

for the R and R^1-algorithms, called PR and PR^1-algorithms respectively, but the presentation would have been more cumbersome. Note also that the PH-algorithm is the easiest to implement, it almost always outputs a reduced basis and is still quite fast in practical use.

3 A Generalized Plane-Probing Algorithm

3.1 Motivation and Outline of the Algorithm

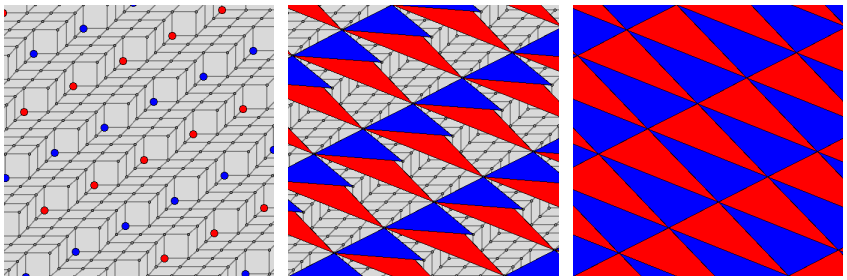
In the most recent plane-probing algorithms, i.e. [23, Alg.1 and Alg.2], [24, H- and R-alg.] and [30], the starting point must be a *reentrant corner*. This restriction avoids degenerate cases, but limits the practical usage of the algorithm for digital surface analysis. In addition, these plane-probing algorithms stop prematurely and output only an approximation of the normal vector for starting points, which are not located deeply enough in the digital plane (see Fig. 3 (a) and (b)). Even if we know how to detect and remove approximated results *a posteriori* [24], we would like to keep these algorithms running until they return the right normal vector.

Another but related problem is that all plane-probing algorithms are able to return only one kind of triangular facet with the right normal: those located above points of height $\omega - 2$ for [22], those located below points of height ω for all other algorithms (see Fig. 3 (c)).

In this section, we provide a general framework for plane-probing algorithms so that they can start from any point and that return one kind of final triangular facets or another, depending on the starting point.

The approach may be coarsely described as follows:

- we use a parallelepiped, which can be thought as a pair of a upward-oriented and downward-oriented tetrahedra along one direction. Its vertex set is in addition partitioned into two non-empty sets, one in $\mathbf{P}$, one outside $\mathbf{P}$.



(a) (b) (c)

Fig. 3: (a) Starting points of height 0 (red) and 1 (blue). (b) Last triangular facets obtained by the H-algorithm on a digital plane of normal $(2, 6, 15)$. When started from the red (resp. blue) points, it outputs the red (resp. blue) triangles of normal $(2, 6, 15)$ (resp. $(1, 3, 7)$). In (c), the red triangles are those located below points of height ω and can be returned by the H-algorithm, whereas the blue ones are those located above points of height $\omega - 2$ and can be returned only by [22].

– we iteratively deform the parallelepiped using the H, R or R^1-algorithm on the downward-oriented tetrahedron so that we can make a vertex closer to the upper plane $\{x \in \mathbb{R}^3 \mid x \cdot \mathbf{N} = \omega - 1\}$, even if it was not in $\mathbf{P}$ or does not stay in $\mathbf{P}$ after the update.
– to guarantee that the parallelepiped is *separating*, i.e. at least one of its vertex is in $\mathbf{P}$ and one is not, there may be some steps where we apply our update procedure on the upward-oriented tetrahedron. In this case, the algorithm is said to be in a *reverse state*.

3.2 A Parallelepiped-Based Representation

In the rest of the paper and due to its repetitive use, the symbol $\mathbf{3}$ stands for the set $\{0, 1, 2\}$ or $\mathbb{Z}/3\mathbb{Z}$ depending on the context. In addition, Σ is the set of permutations over $\{0, 1, 2\}$.

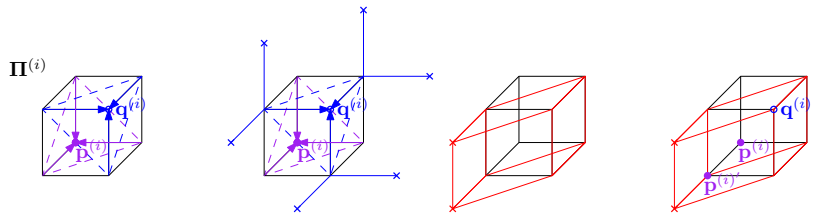
In the H, R and R^1-algorithms, at each step $i \in \{0, \ldots, n\}$, the current tetrahedron consists of a fixed point q, the apex, and three vectors $(\vec{m}_k^{(i)})_{k\in\mathbf{3}}$, which defines the base triangle $\mathbf{T}^{(i)} := (q^{(i)} - \vec{m}_k^{(i)})_{k\in\mathbf{3}}$. In our new framework, q may be modified and that is why we add the iteration step (i) as an exponent to q. We also denote by n the last iteration. For now n could be infinite, but we will show that it is finite in Corollary 1.

At each step $i \in \{0, \ldots, n\}$, the algorithm deforms a parallelepiped, implicitly described by $q^{(i)}$ and $(\vec{m}_k^{(i)})_{k \in \mathbf{3}}$:

$$p^{(i)} := q^{(i)} - \sum_{k \in \mathbf{3}} \vec{m}_k^{(i)}, \tag{2}$$

$$\mathbf{\Pi}^{(i)} := \{q^{(i)}\} \cup \{q^{(i)} - \vec{m}_k^{(i)}\}_{k \in \mathbf{3}} \cup \{p^{(i)} + \vec{m}_k^{(i)}\}_{k \in \mathbf{3}} \cup \{p^{(i)}\}. \tag{3}$$

This parallelepiped can be thought as a pair of a upward-oriented and downward-oriented tetrahedra along one direction (see Fig. 4 (a)).



(a) (b) (c) (d)

Fig. 4: In (a), the current parallelepiped is represented as the union of two tetrahedra. If the H-algorithm is used, only six points may be used to update the tetrahedron containing $q^{(i)}$ and thus $\mathbf{\Pi}^{(i)}$. They are depicted with crosses in (b). As illustrated in (c), two consecutive parallelepipeds share exactly six vertices in this case, whatever the chosen point. In (d), the relative positions of $q^{(i)}$, $p^{(i)}$ and $p^{(i)'}$ are indicated.

At each step $i \in \{0, \ldots, n-1\}$ of the H, R and R^1-algorithms, the triangles $\mathbf{T}^{(i)}$ and $\mathbf{T}^{(i+1)}$ share two vertices; the new point $\mathbf{T}^{(i+1)} \setminus \mathbf{T}^{(i)}$ is drawn from a candidate set included in the set of rays $\left\{\mathcal{R}_\sigma^{(i)}\right\}_{\sigma \in \Sigma}$, where a ray is defined as follows:

$$\mathcal{R}_\sigma^{(i)} := \left\{q^{(i)} - \vec{m}_{\sigma(0)}^{(i)} + \vec{m}_{\sigma(1)}^{(i)} + \lambda \vec{m}_{\sigma(2)}^{(i)}\right\}_{\lambda \in \mathbb{Z}_{\geq 0}}.$$

In our framework, we can use the update procedure of either the H, R or R^1-algorithm, which means that $\forall i \in \{0, \ldots, n-1\}$, there exist a permutation $\sigma^\star \in \Sigma$ and a positive integer λ such that:

$$\begin{cases} \vec{m}_{\sigma^\star(0)}^{(i)'} := \vec{m}_{\sigma^\star(0)}^{(i)} - \vec{m}_{\sigma^\star(1)}^{(i)} - \lambda \vec{m}_{\sigma^\star(2)}^{(i)}, \\ \vec{m}_{\sigma^\star(1)}^{(i)'} := \vec{m}_{\sigma^\star(1)}^{(i)}, \\ \vec{m}_{\sigma^\star(2)}^{(i)'} := \vec{m}_{\sigma^\star(2)}^{(i)}. \end{cases} \tag{4}$$

Note that λ is free in the R and R^1-algorithms, whereas it is set to 0 in the H-algorithm. For sake of clarity, we assume in what follows that the H-algorithm is used and that $\lambda = 0$. In this case, it is easy to check from (3) and (4) that $\mathbf{\Pi}^{(i)}$ and $\mathbf{\Pi}^{(i)'}$ share exactly two consecutive quads, i.e. six vertices (see Fig. 4 (b) and Fig. 4 (c)).

Let us imagine that each vertex $x \in \mathbf{\Pi}^{(i)}$ is colored as white if $\text{InPlane}(x) = \text{InPlane}(q^{(i)})$ and black otherwise. We now assume that $\mathbf{\Pi}^{(i)}$ contains at least 4 black vertices. After the update $\mathbf{\Pi}^{(i)'}$ contains at least one black and one white vertex. However, if $\mathbf{\Pi}^{(i)'}$ is separating, it may contain less than four black vertices, and therefore may lead to a non-separating parallelepiped after the next iteration. In order to guarantee that the parallelepiped is always separating, we check whether at least four vertices of $\mathbf{\Pi}^{(i)'}$ are black or not after the deformation. If there are strictly less than four black vertices, the representation of the parallelepiped is turned upside-down (see also lines 7-9 of Algorithm 2, which is described in subsection 3.4):

$$\forall i \in \{0,\dots,n\},\ \begin{cases} q^{(i+1)} := q^{(i)'},\ \forall k \in \mathbf{3},\ \vec{m}_k^{(i+1)} := \vec{m}_k^{(i)'} \\ \ \text{if}\ \text{Card}(\{x \in \mathbf{\Pi}^{(i)'} \mid \text{InPlane}(x) \neq \text{InPlane}(q^{(i)'})\}) \geq 4, \\ q^{(i+1)} := q^{(i)'} - \sum_{k\in\mathbf{3}} \vec{m}_k^{(i)'}, \\ \vec{m}_0^{(i+1)} := -\vec{m}_1^{(i)'}, \vec{m}_1^{(i+1)} := -\vec{m}_0^{(i)'}, \vec{m}_2^{(i+1)} := -\vec{m}_2^{(i)'} \\ \ \text{otherwise.} \end{cases} \tag{5}$$

Note that in the second case, we take the opposite of the basis vectors and we swap two of them to keep a consistent orientation. This operation is the only one that makes $q^{(i+1)} \neq q^{(i)}$. In addition, after this step, there are at least five black vertices in $\mathbf{\Pi}^{(i+1)}$, because the colors then depend on $\text{InPlane}(q^{(i+1)})$, which is equal to $\text{InPlane}(p^{(i)'})$, which is itself not equal to $\text{InPlane}(q^{(i)})$. We prove that $\text{InPlane}(p^{(i)}) \neq \text{InPlane}(q^{(i)})$ for all $i \in \{0,\dots,n\}$ in Lemma 2 (see section 4.2).

We say that when $\neg\,\text{InPlane}(q^{(i)})$ the algorithm and $\mathbf{\Pi}^{(i)}$ are in *common state*, else they are in *reverse state*.

3.3 A New Predicate

Plane-probing algorithms are based on the predicate InPlane := "is x in $\mathbf{P}$?". For all steps $i \in \{0,\dots,n\}$, the triangle $\mathbf{T}^{(i)}$ must be included in $\mathbf{P}$, while q must lie above $\mathbf{P}$, i.e. $q \cdot \mathbf{N} \geq \omega$.

The predicate InPlane is actually a way of comparing the height of a given point x to the height of q, without knowing the normal direction $\mathbf{N}$ that is sought for. Indeed, since $q \cdot \mathbf{N} \geq \omega$ by definition, we have $x \cdot \mathbf{N} < q \cdot \mathbf{N}$ for all $x \in \mathbf{P}$. However, for $x \notin \mathbf{P}$, $x \cdot \mathbf{N}$ may be less than, equal to or greater than $q \cdot \mathbf{N}$. Therefore, requiring that $\forall i \in \{0,\dots,n\}$, $\mathbf{T}^{(i)} \subset \mathbf{P}$, is a way of requiring that $\forall i \in \{0,\dots,n\}$, $\forall k \in \mathbf{3}$, $\vec{m}_k^{(i)} \cdot \mathbf{N} > 0$.

The goal of this subsection is to use a more general predicate that provides a way of comparing the height of q to the height of points that belong or not to $\mathbf{P}$. More precisely, the predicate NotAbove is defined so as to return true if and only if $x \cdot \mathbf{N} < q \cdot \mathbf{N}$, for all $x \in \mathbb{Z}^3$ such that $x \cdot \mathbf{N} \geq q \cdot \mathbf{N} - \omega$. This condition is not restrictive because at each step, the candidate set consists of points such that $x \cdot \mathbf{N} \geq q \cdot \mathbf{N} - \omega$. Using the predicate NotAbove, instead of

InPlane, is then a way of having $\forall i \in \{0,\dots,n\}$, $\forall k \in \mathbf{3}$, $\vec{m}_k^{(i)} \cdot \mathbf{N} > 0$, without requiring that $\forall i \in \{0,\dots,n\}$, $\mathbf{T}^{(i)} \subset \mathbf{P}$. It may happen that no vertex of $\mathbf{T}^{(i)}$ at all belong to $\mathbf{P}$. However, we would like to stay close to the digital plane, while having points in $\mathbf{P}$ and not in $\mathbf{P}$. That is why we use this predicate into the framework presented above.

In what follows, we use the bar notation $\bar{\cdot}$ above any point $x \in \mathbb{Z}^3$ to denote its height relative to the current state: in common state, i.e. if $\neg\,\mathrm{InPlane}(q^{(i)})$, $\overline{x} := x \cdot \mathbf{N}$, while in reverse state, i.e. if $\mathrm{InPlane}(q^{(i)})$, $\overline{x} := (-x + p^{(i)} + q^{(i)}) \cdot \mathbf{N}$. For instance, $\overline{x} < \overline{q}^{(i)}$ must be read $x \cdot \mathbf{N} < q^{(i)} \cdot \mathbf{N}$ if $\neg\,\mathrm{InPlane}(q^{(i)})$, but $x \cdot \mathbf{N} > q^{(i)} \cdot \mathbf{N}$ otherwise. Similarly, for an iteration (i)-dependent vector, $\overline{\vec{w}}^{(i)} < 0$ must be read $\vec{w}^{(i)} \cdot \mathbf{N} < 0$ if $\neg\,\mathrm{InPlane}(q^{(i)})$, but $\vec{w}^{(i)} \cdot \mathbf{N} > 0$ otherwise.

For all $i \in \{0,\dots,n\}$, the predicate NotAbove is defined so as to return true if and only if $\overline{x} < \overline{q}^{(i)}$, for all $x \in \mathbb{Z}^3$ such that $\overline{x} \geq \overline{q}^{(i)} - \omega$. Such a predicate is implemented in Algorithm 1. The idea is to consider a ray in direction $\vec{u} := x - q^{(i)}$ starting from a point s lying above $\mathbf{P}$ and to search for a non-negative integer l such that $\mathrm{InPlane}(s + l\vec{u})$ in order to be able to determine the sign of $\vec{u} \cdot \mathbf{N}$ and compare $\overline{x}$ with respect to $\overline{q}^{(i)}$. We also consider the ray of opposite direction $-\vec{u}$ to minimize the number of calls to InPlane (see Fig. 5 for an illustration).

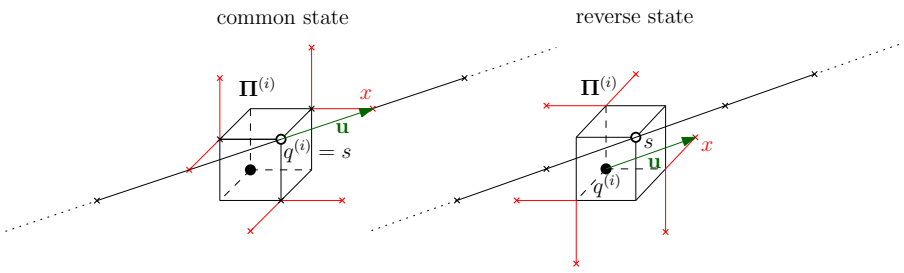


Fig. 5: Illustration of the way predicate NotAbove is implemented in common state on the left and reverse state on the right. We search for points in $\mathbf{P}$ along two rays of direction $\pm\vec{u}$ from s. Direction $\vec{u}$ is defined as the difference between the point x to test and $q^{(i)}$. Point s is defined as the highest point of the current parallelepiped and is always above $\mathbf{P}$ (see Lemma 2 in section 4.2 for the proof).

Algorithm 1 is proven to be correct in Proposition 1 provided some prerequisites involving the height of $q^{(i)}$ and x (P1) and a constant L big enough (P2). Note that this constant is used to stop the search when direction $\vec{u}$ is collinear with $\mathbf{P}$. The proof of Proposition 1 is postponed until section 4.2.

Proposition 1 *For all $i \in \{0,\dots,n\}$, $\mathrm{NotAbove}(x)$, as it is implemented in Algorithm 1, returns true if and only if $\overline{x} < \overline{q}^{(i)}$ (P1) for all $x \in \mathbb{Z}^3$ such that $\overline{q}^{(i)} - \omega \leq \overline{x}$, (P2) provided that $L \geq \omega$.*

Algorithm 1: Implementation of predicate NotAbove such that NotAbove(x) iff $\overline{x} < \overline{q}^{(i)}$ for some $x \in \mathbb{Z}^3$ such that $\overline{q}^{(i)} - \omega \leq \overline{x}$.

Data: InPlane, $q^{(i)}$, $(\vec{m}_k^{(i)})_{k\in\mathbf{3}}$ and an integer $L \geq \omega$
Input: A point $x \in \mathbb{Z}^3$ such that $\overline{q}^{(i)} - \omega \leq \overline{x}$
Output: true iff $\overline{x} < \overline{q}^{(i)}$

```
1  isReversed ← InPlane(q^(i)) ; // common or reverse state
2  u⃗ ← x − q^(i) ; // direction
3  s ← q^(i) ; // starting point
4  if isReversed then s ← s − Σ_{k∈3} m⃗_k^(i) ; // ω ≤ s · N < 2ω
5  l ← 1;
6  while l < L do
7  |  if InPlane(s + lu⃗) then return ¬isReversed ;
8  |  if InPlane(s − lu⃗) then return isReversed ;
9  |  l ← 2l;
10 return False;
```

3.4 PH-algorithm

The whole plane-probing procedure is given by Algorithm 2. The parallelepiped is initialized from an oriented surfel included in $\mathbf{P}$ (lines 1 and 2). This surfel is described by a point p and three unit vectors $(\vec{u}_k)_{k\in\mathbf{3}}$ such that the set $\{p + \mu\vec{u}_0 + \nu\vec{u}_1 \mid (\mu,\nu) \in [0,1]^2\}$ is the geometrical realization of the surfel and $\vec{u}_2$ is its normal direction. Since we assume that the components of $\mathbf{N}$ are non-negative integers, $(\vec{u}_k)_{k\in\mathbf{3}} = (\vec{e}_k)_{k\in\mathbf{3}}$, the canonical basis of $\mathbb{R}^3$ and it is easily checked that $\sum_k \vec{u}_k = (1,1,1)$ and $\sum_k \vec{u}_k \cdot \mathbf{N} = \omega$.

The parallelepiped is then iteratively deformed by calls to function FIND and possibly turned upside-down (from line 7 to line 11). The algorithm stops at step n, when triangle $\mathbf{T}^{(n)}$ is aligned with the plane. It then returns the estimated normal to $\mathbf{P}$ and a basis of the lattice of points of same height (line 14).

If the update procedure of the H-algorithm is used, function FIND returns a permutation $\sigma^\star \in \Sigma$ such that the circumsphere of $\mathbf{T}^{(i)} \cup \{q^{(i)} - \vec{m}^{(i)}_{\sigma^\star(0)} + \vec{m}^{(i)}_{\sigma^\star(1)}\}$ does not include in its interior any other point of the candidate set

$$\mathbf{S}^{(i)} := \{x \in \{q^{(i)} - \vec{m}^{(i)}_{\sigma(0)} + \vec{m}^{(i)}_{\sigma(1)}\}_{\sigma\in\Sigma} \mid \text{NotAbove}(x)\}. \quad (6)$$

Implementation details for FIND may be found in [24], where InPlane is used instead of NotAbove. We however include an implementation in this paper for completeness, see Algorithm 3.

Fig. 6 shows the behaviour of Algorithm 2 when run from two distinct surfels of the digital plane of normal vector $\mathbf{N}(1,2,4)$. Fig. 7 shows the final parallelepiped, computed from three distinct surfels of the digital plane of normal vector $\mathbf{N}(3,7,15)$. The position of the final parallelepiped with respect to the starting surfel is discussed in section 5.1.

Algorithm 2: PH-ALGORITHM: computes a normal vector and a basis from the predicate InPlane a starting point and a frame.

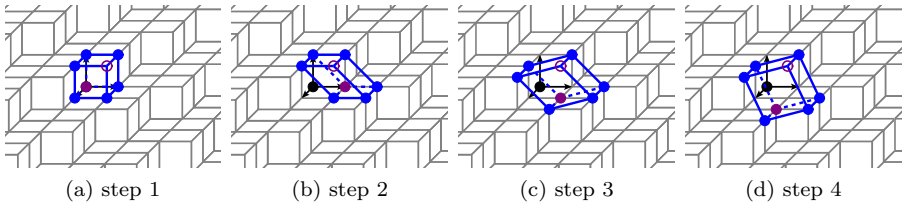
Input: The predicate InPlane, a starting point $p \in \mathbf{P}$ and a frame $(\vec{u}_k)_{k\in\mathbf{3}}$ such that $\forall k\in \mathbf{3},\ \|\vec{u}_k\|_2 = 1$, $\vec{u}_k \cdot \mathbf{N} > 0$ and $\{p + \vec{u}_0, p + \vec{u}_1, p + \vec{u}_0 + \vec{u}_1\} \subset \mathbf{P}$.
Output: A normal vector and a basis of a 2D lattice.

$(\vec{m}_k^{(0)})_{k\in\mathbf{3}} \leftarrow (\vec{u}_k)_{k\in\mathbf{3}}$; `// initialization`
$q^{(0)} \leftarrow p + \sum_k \vec{m}_k^{(0)}$;
$i \leftarrow 0$;
while $\{x \in \{q^{(i)} \pm (\vec{m}_k^{(i)} - \vec{m}_{k+1}^{(i)})\}_{k\in\mathbf{3}} \mid \text{NotAbove}(x)\} \neq \emptyset$ **do**
  $\sigma^\star \leftarrow$ FIND(NotAbove, $q^{(i)}, (\vec{m}_k^{(i)})_{k\in\mathbf{3}}$) ; `// update`
  $\vec{m}_{\sigma^\star(0)}^{(i)\prime} \leftarrow \vec{m}_{\sigma^\star(0)}^{(i)} - \vec{m}_{\sigma^\star(1)}^{(i)}, \vec{m}_{\sigma^\star(1)}^{(i)\prime} \leftarrow \vec{m}_{\sigma^\star(1)}^{(i)}, \vec{m}_{\sigma^\star(2)}^{(i)\prime} \leftarrow \vec{m}_{\sigma^\star(2)}^{(i)}$;
  if $\text{Card}(\{x \in \mathbf{\Pi}^{(i)} \mid \text{InPlane}(x) \neq \text{InPlane}(q^{(i)})\}) < 4$ **then**
    `// Change state: reverse inside/out.`
    $\vec{m}_0^{(i+1)} \leftarrow -\vec{m}_1^{(i)\prime}, \vec{m}_1^{(i+1)} \leftarrow -\vec{m}_0^{(i)\prime}, \vec{m}_2^{(i+1)\prime} \leftarrow -\vec{m}_2^{(i)\prime}$;
    $q^{(i+1)} \leftarrow q^{(i)\prime} - \sum_k \vec{m}_k^{(i)\prime}$;
  else
    `// Keep the same state.`
    $(\vec{m}_k^{(i+1)})_{k\in\mathbf{3}} \leftarrow (\vec{m}_k^{(i)\prime})_{k\in\mathbf{3}}$, $q^{(i+1)} \leftarrow q^{(i)\prime}$;
  $i \leftarrow i + 1$;
$B \leftarrow \{\vec{m}_0^{(i)} - \vec{m}_1^{(i)}, \vec{m}_1^{(i)} - \vec{m}_2^{(i)}, \vec{m}_2^{(i)} - \vec{m}_0^{(i)}\}$;
return $\sum_k \vec{m}_k^{(i)} \times \vec{m}_{k+1}^{(i)}, B \setminus \arg\max_{\vec{b}\in B} \|\vec{b}\|_2$;

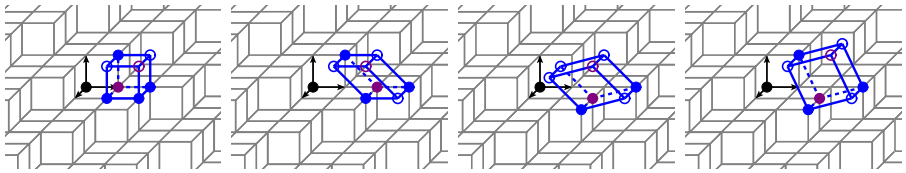
Algorithm 3: FIND: candidate selection used in the H-algorithm to update the current parallelepiped.

Input: A predicate P, the point q and three vectors $(\vec{m}_k)_{k\in\mathbf{3}}$.
Output: A permutation $\sigma^\star \in \Sigma$ such that the circumsphere of $\mathbf{T}^{(i)} \cup \{q^{(i)} - \vec{m}_{\sigma^\star(0)}^{(i)} + \vec{m}_{\sigma^\star(1)}^{(i)}\}$ does not include in its interior any other point of the candidate set $\mathbf{S}^{(i)}$.

Candidates $\leftarrow \emptyset$;
foreach $\sigma \in \Sigma$ **do**
  if $\text{P}(q - \vec{m}_{\sigma(0)} + \vec{m}_{\sigma(1)})$ **then**
    Candidates $\leftarrow$ *Candidates* $\cup \{\sigma\}$;
Let $\sigma^\star$ be an arbitrary permutation in *Candidates* ;
foreach $\sigma \in$ *Candidates* **do**
  if the circumsphere of $\mathbf{T}^{(i)} \cup \{q - \vec{m}_{\sigma^\star(0)} + \vec{m}_{\sigma^\star(1)}\}$ includes $q - \vec{m}_{\sigma(0)} + \vec{m}_{\sigma(1)}$ in its interior **then**
    $\sigma^\star \leftarrow \sigma$;
return $\sigma^\star$;

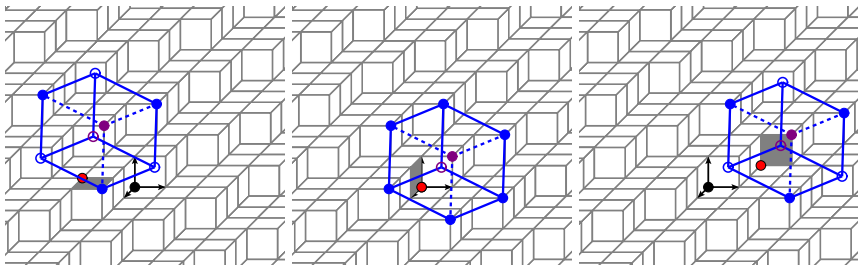


(a) step 1 (b) step 2 (c) step 3 (d) step 4



(e) step 1 (f) step 2 (g) step 3 (h) step 4

Fig. 6: Algorithm 2 has been applied on a digital plane of normal vector $\mathbf{N}(1, 2, 4)$ from two distinct surfels in (a) to (d) and (e) to (h). In each image, the current parallelepiped is depicted in blue and purple for $p^{(i)}$ and $q^{(i)}$. Disks (resp. circles) are used for points lying inside (resp. outside) the digital plane. Note that in the second row, since there are only three vertices in the digital plane at step (g), the parallelepiped is in reverse state at the end (h) (see (5) in section 3.2).



(a) (b) (c)

Fig. 7: Algorithm 2 has been applied on a digital plane of normal vector $\mathbf{N}(3, 7, 15)$ from three distinct surfels in (a), (b) and (c). Each image shows the last parallelepiped. The starting surfel (resp. point) is drawn in gray (resp. red) and $p^{(0)} \cdot \mathbf{N}$ equals to 14 in (a), 0 in (b) and 1 in (c).

3.5 Variants

The use of the R and R^1-algorithm is possible by adapting the FIND function, see [24, Section 2.4] for the implementation details of the R-algorithm and [30, Sections 3 and 4] for the R^1-algorithm. But replacing the FIND function is not enough as directly updating the parallelepiped by a point located far away on a ray may make it non-separating, because only four vertices instead of six are

kept unchanged. If neither the four unchanged vertices are in $\mathbf{P}$, nor the four new vertices, then no vertex of the parallelepiped at all may belong to $\mathbf{P}$.

To overcome this difficulty, we simply *decompose* the update operation into smaller steps similar to updates done in the H-algorithm. We now briefly explain this procedure. Assume that we want to assign to $\vec{m}_{\sigma(0)}$ the following quantity: $\vec{m}_{\sigma(0)} - \vec{m}_{\sigma(1)} - \lambda\vec{m}_{\sigma(2)}$, with a permutation $\sigma \in \Sigma$ and a non-negative integer λ (see (4)). We remove $\vec{m}_{\sigma(1)}$ from $\vec{m}_{\sigma(0)}$ once and we remove λ times $\vec{m}_{\sigma(2)}$ from $\vec{m}_{\sigma(0)}$. After each operation, if needed, we reverse the state of the parallelepiped and update σ accordingly.

At the end, we have assigned to $\vec{m}_{\sigma(0)}$ the quantity $\vec{m}_{\sigma(0)} - \vec{m}_{\sigma(1)} - \lambda\vec{m}_{\sigma(2)}$ while preserving the separability of the parallelepiped. Note that this decomposition adds some calls to the predicate InPlane as we need to check whether points are in $\mathbf{P}$ or not to reverse the state. However, the parallelepiped is guaranteed to contain at least one point in $\mathbf{P}$ and one point not in $\mathbf{P}$.

4 Algorithm Analysis

In this section, we analyze the algorithm described in section 3. We first prove that it always terminates in section 4.1. Then, in section 4.2, we prove that Algorithm 1 correctly implements the predicate NotAbove, which is a key point of our method. Finally, we prove that the proposed algorithm returns the correct normal vector and a basis of the 2D lattice of the points of same height in section 4.3. The section ends with two extra results about the number of state changes during the execution.

4.1 Termination

In this section, we simply write $\forall k$ instead of $\forall k \in \mathbf{3}$ when no confusion may arise.

Thanks to the definition of the candidate set (6) that involves an appropriate predicate (see section 3.4), we get the following result:

Lemma 1 $\forall i \in \{0, \ldots, n\},\ \forall k,\ \overline{\vec{m}}_k^{(i)} > 0.$

Proof It is easily checked that $\forall k,\ \overline{\vec{m}}_k^{(0)} = \vec{e}_k \cdot \mathbf{N} > 0$ (see section 3.4 and lines 1, 2 of Algorithm 2). We now prove that if for some $i \in \{0, ..., n-1\},\ \forall k,\ \overline{\vec{m}}_k^{(i)} > 0$, then $\forall k,\ \overline{\vec{m}}_k^{(i+1)} > 0$.

Recall that $\sigma^\star$ is the permutation returned in line 5 of Algorithm 2. We have $\overline{\vec{m}}_{\sigma^\star(1)}^{(i)'} > 0$ and $\overline{\vec{m}}_{\sigma^\star(2)}^{(i)'} > 0$ due to the induction hypothesis. It remains to show that $\overline{\vec{m}}_{\sigma^\star(0)}^{(i)'} > 0$. Since the point $q^{(i)} - \vec{m}_{\sigma^\star(0)}^{(i)} + \vec{m}_{\sigma^\star(1)}^{(i)}$ is by definition in $\mathbf{S}^{(i)}$ and therefore *not above* $q^{(i)}$, we have $\overline{q}^{(i)} - \overline{\vec{m}}_{\sigma^\star(0)}^{(i)} + \overline{\vec{m}}_{\sigma^\star(1)}^{(i)} < \overline{q}^{(i)}$, which is equivalent to $\overline{\vec{m}}_{\sigma^\star(0)}^{(i)'} > 0$ (line 6 of Algorithm 2).

Now the state may stay the same and we get $\forall k,\ \vec{m}_k^{(i+1)} = \vec{m}_k^{(i)'}$ which implies that $\forall k,\ \overline{\vec{m}}_k^{(i+1)} = \overline{\vec{m}}_k^{(i)'} > 0$. If the state change, there is a sign change (see (5)), which also implies that $\forall k,\ \overline{\vec{m}}_k^{(i+1)} = \overline{\vec{m}}_k^{(i)'} > 0$, since the $\bar{\cdot}$ notation makes the sign change. □

We can now derive the following theorem:

Theorem 1 *The sequence* $(\sum_k \overline{\vec{m}}_k^{(i)})_{i=0,\dots,n}$ *is a strictly decreasing sequence of integers between* ω *and* 3*, where* ω *is the thickness of the digital plane* $\mathbf{P}$*.*

Proof It is easily checked that $\forall k,\ \vec{m}_k^{(0)} = \vec{e}_k$ and $\sum_k \overline{\vec{m}}_k^{(0)} = \omega$ (see section 3.4). In addition, we already know by Lemma 1 that

$$\forall i \in \{0,\dots,n\},\ \forall k,\ \overline{\vec{m}}_k^{(i)} > 0 \text{ and } \sum_k \overline{\vec{m}}_k^{(i)} \geq 3.$$

Therefore, it remains to show that $(\sum_k \overline{\vec{m}}_k^{(i)})_{i=0,\dots,n}$ is a strictly decreasing sequence of integers, i.e.

$$\forall i \in \{0,\dots,n-1\},\ \sum_k \overline{\vec{m}}_k^{(i)} > \sum_k \overline{\vec{m}}_k^{(i+1)}.$$

Recall that $\sigma^\star$ is the permutation returned in line 5 of Algorithm 2. After line 6 of Algorithm 2, we have $\forall i \in \{0,\dots,n-1\}$, $\vec{m}_{\sigma^\star(0)}^{(i)'} = \vec{m}_{\sigma^\star(0)}^{(i)} - \vec{m}_{\sigma^\star(1)}^{(i)}$. Since $\forall k,\ \overline{\vec{m}}_k^{(i)} > 0$ by Lemma 1, we clearly have $\overline{\vec{m}}_{\sigma^\star(0)}^{(i)'} = \overline{\vec{m}}_{\sigma^\star(0)}^{(i)} - \overline{\vec{m}}_{\sigma^\star(1)}^{(i)} < \overline{\vec{m}}_{\sigma^\star(0)}^{(i)}$. We can conclude that $\sum_k \overline{\vec{m}}_k^{(i+1)} = \sum_k \overline{\vec{m}}_k^{(i)'} < \sum_k \overline{\vec{m}}_k^{(i)}$ whether there is a change of state or not. □

A straightforward corollary is that the algorithm terminates:

Corollary 1 *The number of steps* n *in Algorithm 2 is bounded from above by* $\omega - 3$*, where* ω *is the thickness of the digital plane* $\mathbf{P}$*.*

We end this subsection by the worst-case complexity in terms of predicate calls:

Theorem 2 *The number of calls to predicate* InPlane *in Algorithm 2 is upper bounded by* $O(\omega \log \omega)$*, where* ω *is the thickness of the digital plane* $\mathbf{P}$*.*

Proof First, according to Corollary 1, there are at most $\omega - 3$ steps in Algorithm 2. Second, at each step, there are 6 calls to NotAbove at line 4 to check if the algorithm must stop or not, 6 other calls to NotAbove to choose a candidate point (line 3 of Algorithm 3, where P is the predicate NotAbove) and 8 calls to InPlane at line 7 to determine if we need to reverse or not the state of the current parallelepiped.

To sum up, there are at most $8(\omega - 3)$ calls to InPlane and $12(\omega - 3)$ calls to NotAbove. However, each call to NotAbove requires some calls to InPlane.

In Algorithm 1, there is one call to InPlane at the very beginning and at most two calls to InPlane in the while loop. The worst case is when the algorithm returns after the while loop. In this case, due to the exponential march, the number of iterations is less than $\log_2(L)$ and therefore, the number of calls to InPlane is less than $1 + 2\log_2(L)$. According to Proposition 1, L must be chosen such that $L \geq \omega$. With $L = \omega$, each call to NotAbove requires $O(\log \omega)$ calls to InPlane.

In conclusion, the total number of calls to InPlane is upper bounded by $O(\omega \log \omega)$. □

Note, as it was shown in [30, Theorem 1], that the number of calls to InPlane in the R^1-algorithm is asymptotically equal to the number of steps, i.e., is in $O(\omega)$. It can be similarly shown that the number of calls to NotAbove in PR^1-algorithm is also in $O(\omega)$ and that the overall complexity is $O(\omega \log \omega)$.

4.2 Correctness of Algorithm 1, Which Implements Predicate NotAbove

We first show that the current parallelepiped always lies across the upper plane $\{x \in \mathbb{R}^3 \mid x \cdot \mathbf{N} = \omega - 1\}$ because its lowest vertex belongs to $\mathbf{P}$, while its highest vertex lies above $\mathbf{P}$. One of the key point to prove Proposition 1 is that the highest vertex is never too far above $\mathbf{P}$, because its height is always between ω and $2\omega - 1$.

Lemma 2 *Let us recall that ω is the thickness of the digital plane $\mathbf{P}$. For all $i \in \{0, \ldots, n\}$, $p^{(i)} \cdot \mathbf{N}$ (resp. $q^{(i)} \cdot \mathbf{N}$) lies in the range $[0, \ldots, \omega - 1]$, while $q^{(i)} \cdot \mathbf{N}$ (resp. $p^{(i)} \cdot \mathbf{N}$) lies in the range $[\omega, \ldots, 2\omega - 1]$ in common (resp. reverse) state.*

Proof The result is trivial for $i = 0$, because $p^{(0)}$ must belong to $\mathbf{P}$ and $q^{(0)} = p^{(0)} + \sum_k \vec{u}_k$ (line 2 of Algorithm 2), which implies that $0 \leq p^{(0)} \cdot \mathbf{N} < \omega$ and $q^{(0)} \cdot \mathbf{N} = p^{(0)} \cdot \mathbf{N} + \omega$.

We assume that, for some $i \in \{0, \ldots, n-1\}$, in common state, $0 \leq p^{(i)} \cdot \mathbf{N} < \omega \leq q^{(i)} \cdot \mathbf{N} < 2\omega$. The case where we are in reverse state is symmetric and left to the reader. We now want to show that Lemma 2 is true at step $i + 1$.

There is a permutation $\sigma^\star \in \Sigma$ such that $\vec{m}^{(i)'}_{\sigma^\star(0)} = \vec{m}^{(i)}_{\sigma^\star(0)} - \vec{m}^{(i)}_{\sigma^\star(1)}$ (line 6 of Algorithm 2). On one hand,

$$\begin{aligned} p^{(i)'} &= q^{(i)'} - \sum_k \vec{m}^{(i)'}_k \\ &= q^{(i)} - (\vec{m}^{(i)}_{\sigma^\star(0)} - \vec{m}^{(i)}_{\sigma^\star(1)}) - \vec{m}^{(i)}_{\sigma^\star(1)} - \vec{m}^{(i)}_{\sigma^\star(2)} \\ &= p^{(i)} + \vec{m}^{(i)}_{\sigma^\star(1)}. \end{aligned}$$

Note that, due to the induction hypothesis, $p^{(i)} \cdot \mathbf{N} \geq 0$, which means that, by Lemma 1, $(p^{(i)} + \vec{m}^{(i)}_{\sigma^\star(1)}) \cdot \mathbf{N} \geq 0$.

On the other hand, by Lemma 1, $\forall k,\ \overline{\vec{m}}_k^{(i)} > 0$ and $\overline{\vec{m}}_{\sigma^\star(0)}^{(i)'} = \overline{\vec{m}}_{\sigma^\star(0)}^{(i)} - \overline{\vec{m}}_{\sigma^\star(1)}^{(i)} > 0$. Consequently,

$$\begin{aligned}
0 < \overline{\vec{m}}_{\sigma^\star(1)}^{(i)} &< \overline{\vec{m}}_{\sigma^\star(0)}^{(i)},\\
\overline{\vec{m}}_{\sigma^\star(1)}^{(i)} &< (\overline{\vec{m}}_{\sigma^\star(0)}^{(i)} + \overline{\vec{m}}_{\sigma^\star(1)}^{(i)}),\\
\overline{\vec{m}}_{\sigma^\star(1)}^{(i)} &< (\overline{\vec{m}}_{\sigma^\star(1)}^{(i)} + \overline{\vec{m}}_{\sigma^\star(2)}^{(i)}),\\
\overline{\vec{m}}_{\sigma^\star(1)}^{(i)} &< (\overline{\vec{m}}_{\sigma^\star(0)}^{(i)} + \overline{\vec{m}}_{\sigma^\star(2)}^{(i)}),\\
\overline{\vec{m}}_{\sigma^\star(1)}^{(i)} &< (\overline{\vec{m}}_{\sigma^\star(0)}^{(i)} + \overline{\vec{m}}_{\sigma^\star(1)}^{(i)} + \overline{\vec{m}}_{\sigma^\star(2)}^{(i)}).
\end{aligned}$$

Due to the above result, if $(p^{(i)} + \vec{m}_{\sigma^\star(1)}^{(i)}) \cdot \mathbf{N} \geq \omega$, at least six vertices of $\mathbf{\Pi}^{(i)}$, i.e., $\{(p^{(i)} + \vec{m}_{\sigma^\star(0)}^{(i)}), (p^{(i)} + \vec{m}_{\sigma^\star(1)}^{(i)}), (p^{(i)} + \vec{m}_{\sigma^\star(0)}^{(i)} + \vec{m}_{\sigma^\star(1)}^{(i)}), (p^{(i)} + \vec{m}_{\sigma^\star(1)}^{(i)} + \vec{m}_{\sigma^\star(2)}^{(i)}), (p^{(i)} + \vec{m}_{\sigma^\star(0)}^{(i)} + \vec{m}_{\sigma^\star(2)}^{(i)}), (p^{(i)} + \vec{m}_{\sigma^\star(0)}^{(i)} + \vec{m}_{\sigma^\star(1)}^{(i)} + \vec{m}_{\sigma^\star(2)}^{())}\}$, do not belong to $\mathbf{P}$, which raises a contradiction because in this case, the state should have been reversed. We conclude that

$$0 \leq (p^{(i)} + \vec{m}_{\sigma^\star(1)}^{(i)}) \cdot \mathbf{N} = p^{(i)'} \cdot \mathbf{N} < \omega.$$

In addition, due the induction hypothesis, we have:

$$\omega \leq q^{(i)} \cdot \mathbf{N} = q^{(i)'} \cdot \mathbf{N} < 2\omega.$$

If the state stays the same, $p^{(i+1)} \cdot \mathbf{N}$ lies in the range $[0, \ldots, \omega - 1]$, while $q^{(i+1)} \cdot \mathbf{N}$ lies in the range $[\omega, \ldots, 2\omega - 1]$. Conversely, if the state change, $p^{(i+1)} = q^{(i)'}$ and $q^{(i+1)} = p^{(i)'}$ (lines 7 to 9 of algorithm 2) and therefore $q^{(i+1)} \cdot \mathbf{N}$ lies in the range $[0, \ldots, \omega - 1]$, while $p^{(i+1)} \cdot \mathbf{N}$ lies in the range $[\omega, \ldots, 2\omega - 1]$. □

Before proving Proposition 1, we show its first prerequisite. There is nothing to prove for the second one, which simply states the values for the input parameter L in Algorithm 1 for which Proposition 1 is true.

Lemma 3 *For all $i \in \{0, \ldots, n\}$, Algorithm 2 guarantees that the prerequisite (P1) of Proposition 1 is true, i.e., $\overline{q}^{(i)} - \omega \leq \overline{x}$ for all points x passed as argument to the predicate* NotAbove*.*

Proof At each step $i \in \{0, \ldots, n\}$, any point x passed as argument to NotAbove belong to the candidate set $\mathbf{S}^{(i)}$ (6), i.e., there is a permutation $\sigma \in \Sigma$ such that $x = q^{(i)} - \vec{m}_{\sigma(0)}^{(i)} + \vec{m}_{\sigma(1)}^{(i)}$ (see line 4 of Algorithm 2 and line 3 of Algorithm 3).

By Lemma 1 and Corollary 1, we have

$$\overline{\vec{m}}_{\sigma(0)}^{(i)} < \sum_k \overline{\vec{m}}_k^{(i)} \leq \omega < \omega + \overline{\vec{m}}_{\sigma(1)}^{(i)},$$

which gives

$$-\omega < -\overline{\vec{m}}_{\sigma(0)}^{(i)} + \overline{\vec{m}}_{\sigma(1)}^{(i)}.$$

We conclude that, in both states,

$$\overline{q}^{(i)} - \omega < \overline{q}^{(i)} - \overline{\vec{m}}^{(i)}_{\sigma(0)} + \overline{\vec{m}}^{(i)}_{\sigma(1)} = \overline{x}.$$

□

We can now prove that Algorithm 1 is correct:

Proof (of Proposition 1) The proof is decomposed into three parts:

1. At line 7, Algorithm 1 returns true if and only if $\overline{x} < \overline{q}^{(i)}$,
2. At line 8, Algorithm 1 also returns true if and only if $\overline{x} < \overline{q}^{(i)}$,
3. If Algorithm 1 returns false at the end (line 10), then $\overline{x} \geq \overline{q}^{(i)}$.

The three above results obviously prove Proposition 1.

One point, s, and one vector, $\vec{u}$, are involved in Algorithm 1. We have $\vec{u} = x - q^{(i)}$ in both state, $s = q^{(i)}$ in common state, but $s = q^{(i)} - \sum_k \vec{m}^{(i)}_k = p^{(i)}$ in reverse state. Thus, in both states, $s \cdot \mathbf{N} \geq \omega$ by Lemma 2 and we focus now on the sign of $\vec{u} \cdot \mathbf{N}$.

1. Assume that Algorithm 1 returns $\neg isReversed$, i.e. true (resp. false) in common state (resp. reverse state) (line 7 of Algorithm 1). There exists $l > 0$ such that $\mathrm{InPlane}(s + l\vec{u})$ with $\neg\,\mathrm{InPlane}(s + l'\vec{u})$ for all $l' = 0, \ldots, l-1$. As a consequence,

$$(s + l\vec{u}) \cdot \mathbf{N} < \omega \leq s \cdot \mathbf{N} \Leftrightarrow \vec{u} \cdot \mathbf{N} < 0 \Leftrightarrow x \cdot \mathbf{N} < q^{(i)} \cdot \mathbf{N},$$

which means that $\overline{x} < \overline{q}^{(i)}$ (resp. $\overline{x} > \overline{q}^{(i)}$) in common state (resp. reverse state).

2. Assume now that Algorithm 1 returns $isReversed$, i.e. false (resp. true) in common state (resp. reverse state) (line 8 of Algorithm 1). Then there exists $l > 0$ such that $\mathrm{InPlane}(s - l\vec{u})$ and $\neg\,\mathrm{InPlane}(s - l'\vec{u})$ for all $l' = 0, \ldots, l-1$. This case is exactly symmetric to the case **1.** and we can similarly show that $\overline{x} > \overline{q}^{(i)}$ (resp. $\overline{x} < \overline{q}^{(i)}$) in common state (resp. reverse state).

3. If Algorithm 1 returns false at the end (line 10 of Algorithm 1), then $\neg\,\mathrm{InPlane}(s + l\vec{u})$ and $\neg\,\mathrm{InPlane}(s - l\vec{u})$, for all $l = 1, \ldots, L$.

We first show by contradiction that the ray of direction $\pm\vec{u}$ cannot go through the digital plane without intersecting it due to prerequisite (P1) and Lemma 2. Let l' be the smallest non-negative integer such that $(s + l'\vec{u}) \cdot \mathbf{N} < 0$. The case where $(s - l'\vec{u}) \cdot \mathbf{N} < 0$ is exactly symmetric and left to the reader.

If $l' = 1$, we have $s \cdot \mathbf{N} + \vec{u} \cdot \mathbf{N} < 0$, but $-\omega \leq \vec{u} \cdot \mathbf{N}$ (P1) and $\omega \leq s \cdot \mathbf{N}$ (Lemma 2) lead to $s \cdot \mathbf{N} + \vec{u} \cdot \mathbf{N} \geq 0$, which raises a contradiction. If $l' \geq 1$, then $(s + l'\vec{u}) \cdot \mathbf{N} < 0$ and $\omega \leq (s + (l'/2)\vec{u}) \cdot \mathbf{N}$. Summing up these two inequalities, we get $(l'/2)\vec{u} \cdot \mathbf{N} < -\omega$. However, summing up $(s + (l'/2)\vec{u}) \cdot \mathbf{N} \geq \omega$ and $2\omega > s \cdot \mathbf{N}$ (Lemma 2), we get $(l'/2)\vec{u} \cdot \mathbf{N} > -\omega$, which raises a contradiction too.

As a consequence, since $s \cdot \mathbf{N} \geq \omega$ by Lemma 2, $(s + l\vec{u}) \cdot \mathbf{N} \geq \omega$ for all $l = 1, \ldots, L$ and especially

$$(s + L\vec{u}) \cdot \mathbf{N} \geq \omega \Leftrightarrow \frac{\omega - s \cdot \mathbf{N}}{L} \leq \vec{u} \cdot \mathbf{N}.$$

Since $s \cdot \mathbf{N} < 2\omega \Leftrightarrow -\omega < \omega - s \cdot \mathbf{N}$ by Lemma 2 and $L \geq \omega$ (P2), we get

$$-1 \leq \frac{-\omega}{L} < \frac{\omega - s \cdot \mathbf{N}}{L} \leq \vec{u} \cdot \mathbf{N},$$

which implies that $\vec{u} \cdot \mathbf{N} \geq 0 \Leftrightarrow x \cdot \mathbf{N} \geq q \cdot \mathbf{N} \Rightarrow \overline{x} \geq \overline{q}$. □

4.3 Correctness of Algorithm 2

Let $\mathbf{M}^{(i)}$ be the 3×3 matrix that consists of the three row vectors $(\vec{m}_k^{(i)})_{k \in \mathbf{3}}$. We prove below that $\mathbf{M}^{(i)}$ is unimodular, which is an important property to show that the algorithm returns $\mathbf{N}$ at termination.

Lemma 4 $\forall i \in \{0, \ldots, n\},\ \det(\mathbf{M}^{(i)}) = 1.$

Proof It is easily checked that $\det(\mathbf{M}^{(0)}) = 1$. We now prove that if $\det(\mathbf{M}^{(i)}) = 1$ for $i \in \{0, \ldots, n-1\}$, then $\det(\mathbf{M}^{(i+1)}) = 1$.

There is a permutation $\sigma^\star \in \Sigma$ such that $\vec{m}_{\sigma^\star(0)}^{(i)'} = \vec{m}_{\sigma^\star(0)}^{(i)} - \vec{m}_{\sigma^\star(1)}^{(i)}$ (line 6 of Algorithm 2). The other vectors are not modified so the remaining two rows of $\mathbf{M}^{(i)}$ are not modified in $\mathbf{M}^{(i)'}$. We get

$$\begin{aligned}
&\det(\mathbf{M}^{(i)'}) \\
&\quad = \det(\vec{m}_{\sigma^\star(0)}^{(i)} - \vec{m}_{\sigma^\star(1)}^{(i)}, \vec{m}_{\sigma^\star(1)}^{(i)}, \vec{m}_{\sigma^\star(2)}^{(i)}) \\
&\quad = \det(\vec{m}_{\sigma^\star(0)}^{(i)}, \vec{m}_{\sigma^\star(1)}^{(i)}, \vec{m}_{\sigma^\star(2)}^{(i)}) && \text{(by linearity)} \\
&\quad = \det(\mathbf{M}^{(i)}) = 1. && \text{(by induction hypothesis)}
\end{aligned}$$

But now, if the algorithm does not change state, we have $\forall k,\ \vec{m}_k^{(i+1)} = \vec{m}_k^{(i)'}$, so $\det(\mathbf{M}^{(i+1)}) = \det(\mathbf{M}^{(i)'}) = \det(\mathbf{M}^{(i)}) = 1$. And if the algorithm reverses its state, we flip the sign of the three matrix lines, but we also swap two of them (lines 7 to 9 of Algorithm 2) so that $\det(\mathbf{M}^{(i+1)}) = \det(\mathbf{M}^{(i)'}) = \det(\mathbf{M}^{(i)}) = 1$. □

Since we now focus on the last step n, we omit the exponent (n) in the proofs to improve their readability.

Theorem 3 *The height of all basis vectors is equal to one at the end, i.e.* $\forall k,\ \overline{\vec{m}}_k^{(n)} = 1.$

Proof The first step of the proof is to show that $\overline{\vec{m}}_0 = \overline{\vec{m}}_1 = \overline{\vec{m}}_2$. If not, then there exists $k \in \mathbf{3}$ such that $\overline{\vec{m}}_k - \overline{\vec{m}}_{k+1} \neq 0$. In this case, either (i) $\overline{\vec{m}}_k - \overline{\vec{m}}_{k+1} < 0$ or (ii) $-\overline{\vec{m}}_k + \overline{\vec{m}}_{k+1} < 0$. Let $x_1 := q + \vec{m}_k - \vec{m}_{k+1}$ and $x_2 := q - \vec{m}_k + \vec{m}_{k+1}$. In common state (resp. reverse state), (i) (resp. (ii)) implies that $\bar{x}_1 < \bar{q}$. Conversely, in reverse state (resp. common state), (i) (resp. (ii)) implies that $\bar{x}_2 < \bar{q}$. It follows that either $\text{NotAbove}(x_1)$ or $\text{NotAbove}(x_2)$ is true and $\{x \in \{q \pm (\vec{m}_k - \vec{m}_{k+1})\}_{k \in \mathbf{3}} \mid \text{NotAbove}(x)\} \neq \emptyset$. This is a contradiction because this set must be empty at the last step (line 4 of Algorithm 2). As

a consequence, $\forall k,\ \overline{\vec{m}}_k = \overline{\vec{m}}_{k+1}$ and $\forall k,\ \overline{\vec{m}}_k = \gamma$, which is a strictly positive integer by Lemma 1.

The second step of the proof is to show that $\gamma = 1$. Let us denote by $\mathbf{1}$ the vector $(1,1,1)^T$. Developing the $\bar{\cdot}$ notation, we can write the last system as $\mathbf{MN} = \pm\gamma\mathbf{1}$, where we have a "+" in common state and a "−" in reverse state. Since $\mathbf{M}$ is invertible (because $\det(\mathbf{M}) = 1$ by Lemma 4), $\mathbf{N} = \pm\mathbf{M}^{-1}\gamma\mathbf{1}$ and as a consequence $\gamma = 1$ (because the components of $\mathbf{N}$ are relatively prime and $\det(\mathbf{M}^{-1}) = 1$). We conclude that, with the $\bar{\cdot}$ notation, $\forall k,\ \overline{\vec{m}}_k = 1$. □

The following two corollaries are derived from Lemma 4 and Theorem 3.

Corollary 2 *The last normal estimate is equal to* $\mathbf{N}$ *in common state and* $-\mathbf{N}$ *in reverse state, i.e.* $\hat{\mathbf{N}}^{(\mathbf{n})} := \sum_{k\in\mathbf{3}}(\vec{m}_k^{(n)} \times \vec{m}_{k+1}^{(n)}) = \pm\mathbf{N}$.

Proof On one hand, $\mathbf{M}\hat{\mathbf{N}} = \mathbf{1}$ because $\forall k,\ \vec{m}_k \cdot (\sum_l \vec{m}_l \times \vec{m}_{l+1}) = \vec{m}_k \cdot (\vec{m}_{k+1} \times \vec{m}_{k+2}) = \det(\mathbf{M})$, which is equal to 1 by Lemma 4.

On the other hand, $\mathbf{MN} = \pm\mathbf{1}$ by Theorem 3. Since $\mathbf{M}$ is invertible, we have $\hat{\mathbf{N}} = \pm\mathbf{N}$. The sign is clear from the last state. □

Corollary 3 $\forall k,\ (\vec{m}_k^{(n)} - \vec{m}_{k+1}^{(n)}, \vec{m}_{k+1}^{(n)} - \vec{m}_{k+2}^{(n)})$ *is a basis of the 2D lattice* $\{x \in \mathbb{Z}^3 \mid \overline{x} = 0\}$.

Proof The unit parallelepiped in the 3D lattice

$$\{(\alpha\vec{m}_0, \beta\vec{m}_1, \gamma\vec{m}_2) | (\alpha,\beta,\gamma) \in \mathbb{Z}^3\}$$

does not contain any integer point because it is equivalent to $\mathbb{Z}^3$ ($\det(\mathbf{M}) = 1$ by Lemma 4). It follows that the facet $conv((\vec{m}_k - \vec{m}_{k+1})_{k\in\mathbf{3}})$ does not contain any integer point. Since $\forall k,\ \overline{\vec{m}}_k - \overline{\vec{m}}_{k+1} = 0$ by Theorem 3, we get the result. □

4.4 State Change

In this subsection, the behaviour of the algorithm is discussed with respect to the height of the starting point, i.e., $p^{(0)} \cdot \mathbf{N}$ (and thus $q^{(0)} \cdot \mathbf{N} = p^{(0)} \cdot \mathbf{N} + \omega$). See Fig. 7, which shows the final parallelepiped, computed from three distinct surfels of the same digital plane.

The two following theorems give sufficient conditions for the parallelepiped to pass through a reverse state during the execution of the algorithm. On one hand, Theorem 4 explains that if we start from a reentrant corner of height 0 or 1, then the algorithm terminates without passing through a reverse state, see Fig. 7 (b) and (c). This means that the highest vertex of the last parallelepiped is very close to the starting point since we have in this case: $q^{(n)} = q^{(0)} = p^{(0)} + (1,1,1)$.

On the other hand, Theorem 5 states that when starting from a point of height greater than 2, the algorithm passes through at least one reverse state, see for instance Fig. 7 (a). In this case, we do not know how to relate the

position of the last parallelepiped to the starting point, but we experimentally show in section 5.1 that even in such cases, the last parallelepiped is never too far from the starting point.

Theorem 4 *If* $q^{(0)} \cdot \mathbf{N} = h$*, for* $h \in \{\omega, \omega+1\}$*, then* $\forall i \in \{0,\dots,n\}$, $q^{(i)} = q^{(0)}$.

Proof First, note that $\text{Card}(\{x \in \mathbf{\Pi}^{(0)} \mid \text{InPlane}(x)\}) \geq 4$.

For $j \in \{0,\dots,n-1\}$, we assume that $\forall i \in \{0,\dots,j\}$, $\text{Card}(\{x \in \mathbf{\Pi}^{(i)} \mid \text{InPlane}(x)\}) \geq 4$ and $q^{(i)} = q^{(0)}$ and we show below that $\forall i \in \{0,\dots,j+1\}$, $\text{Card}(\{x \in \mathbf{\Pi}^{(i)} \mid \text{InPlane}(x)\}) \geq 4$ and $q^{(i)} = q^{(0)}$. Indeed, at step j, after the update (lines 5 and 6 of Algorithm 2), we have $\forall k$, $\vec{m}_k^{(j)'} \geq 1$ (Lemma 1). Since the height of $q^{(j)'} = q^{(j)} = q^{(0)}$ is equal to ω or $\omega+1$ by hypothesis, we get that $\text{Card}(\{x \in \mathbf{\Pi}^{(j+1)} \mid \text{InPlane}(x)\}) \geq 4$ due to the definition of the parallelepiped. Hence, lines 7 and 9 in Algorithm 2 are not executed and $q^{(j+1)} = q^{(j)'} = q^{(j)}$, which concludes. □

Theorem 5 *If* $q^{(0)} \cdot \mathbf{N} \geq \omega + 2$*, there is a step* $i \in \{0,\dots,n\}$ *such that* $\mathbf{\Pi}^{(i)}$ *is in a reverse state.*

Proof Let us assume that there is no step $i \in \{0,\dots,n\}$ such that $\mathbf{\Pi}^{(i)}$ is in a reverse state. Then, $\forall i \in \{0,\dots,n\}$, $q^{(i)} = q^{(0)}$. However, since $q^{(n)} \cdot \mathbf{N} \geq \omega+2$ by hypothesis, $\forall k$, $\overline{\vec{m}}_k^{(n)} = 1$ (Theorem 3) and the definition of the parallelepiped together imply that $\text{Card}(\{x \in \mathbf{\Pi}^{(n)} \mid \text{InPlane}(x)\}) \leq 1$, which raises a contradiction, since Algorithm 2 guarantees by construction that $\forall i \in \{0,\dots,n\}$, $\text{Card}(\{x \in \mathbf{\Pi}^{(i)} \mid \text{InPlane}(x)\}) \geq 4$. □

5 Experimental Analysis

We now illustrate on multiple numerical examples the properties of our method and compare it against the previous plane-probing algorithms. We also show that our method is useful for the piecewise linear reconstruction of digital surfaces.

5.1 Localness

We start by testing the localness of our approach. We wish to measure how the final parallelepiped is close to the starting surfel. Indeed, even if $\forall i \in \{0,\dots,n\}$, $q^{(i)}$ is projected onto $\mathbf{T}^{(i)}$ along direction $(1,1,1)$, we have $\forall i \in \{0,\dots,n\}$, $q^{(i)} = q^{(0)}$ if and only if $q^{(0)} \cdot \mathbf{N} \in \{\omega, \omega+1\}$ (Theorem 4), i.e., for all surfels of $\mathbf{P}$ incident to a point $p = q^{(0)} - (1,1,1)$ whose height is equal to 0 or 1. In the other cases, it is not clear whether the parallelepiped stays close to the starting surfel. In order to experimentally test this condition, we launched the algorithm starting from every surfel of a digital plane of normal $(3,7,15)$. The results are displayed in Fig. 8 (see caption for a description of the figure).

We can make the following observations:

- for surfels incident to a point p of height 0 or 1 (white surfels in the right image), localness is guaranteed, due to the projection invariant involving all $q^{(i)} = p+(1,1,1)$, for $i \in \{0,\ldots,n\}$ (Theorem 4). Note that the PH, PR, PR^1-algorithms behave exactly like the H, R, R^1-algorithms respectively for a point p of height 0;
- for the other surfels (corresponding to the green arrows starting from gray surfels in the right image), the algorithm seems to stay local as arrows do not jump from more than two black triangles. To be more precise, we computed the L^2 distance between the center of the starting surfel and the opposite endpoint: 1.87 in average and 3.67 at the maximum, while the length of the two vectors forming a reduced basis is $(3.74, 4.69)$. These statistics show that the algorithm stays local even in this case;
- the algorithm is also able to find the triangles containing the projection of the points of height $\omega - 2$ along direction $(1,1,1)$ (corresponding to blue points). This was not possible with the previous H,R,R^1-algorithms. The algorithm developed in [22] is able to find them, but not the triangles containing the projection of the points of height ω;
- for every starting point, the final basis is always reduced for PR and PR^1 as it was the case experimentally for R and R^1;
- we can also see, on the right image, that the algorithm always passes through a reverse state except when starting from corners incident to a point of height 0 or 1. Note that there exist corners where we pass through a reverse state, see for instance the gray corners.

5.2 Number of steps and number of calls to predicate InPlane

We now look at the number of steps as well as the number of calls to the predicates InPlane and NotAbove. We start by comparing the H,R,R^1 and the new PH and PR^1-algorithms when starting from the reentrant corner incident to the origin for all normals with relatively prime components comprised between 1 and 200. There are 6.5 million configurations. Results are reported in Table 3. Let us note that we do not pass through a reverse state for the PH (resp. PR^1)-algorithm as it behaves like the H (resp. R^1)-algorithm. As expected, we observe that the number of steps is the same and that the number of calls to InPlane in the H (resp. R^1)-algorithm is the same as the number of calls to NotAbove in the PH (resp. PR^1)-algorithm. However, in the latter case, the total number of calls to InPlane is between two and three times higher, since we do not assume that the starting surfel is incident to a point known to be of height 0.

We also launched the PH and PR^1-algorithm on all the normals N with relatively prime components comprised between 1 and 200 and, for each one, we generated a starting surfel for each possible height (between 0 and $||N||_1 - 1$), this amounts to testing more than 985 million surfels. Results are reported in Table 4. First of all, the final basis is almost always (resp. always) reduced for PH (resp. PR^1) as it was the case for H (resp. R). Furthermore, we see

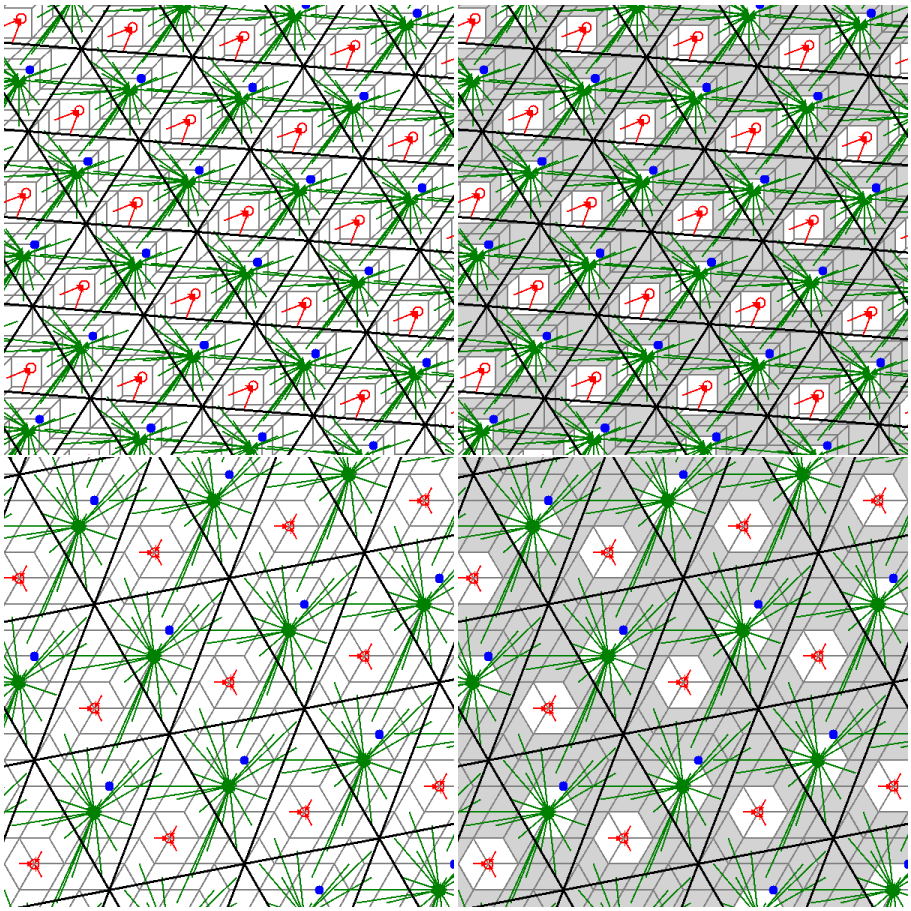


Fig. 8: Results of the PR^1-algorithm when launched on 7618 surfels lying in a digital plane of normal $(3, 7, 15)$. The lattice of points of height $\omega - 1$ is displayed in black, while the points of height $\omega - 2$, ω and $\omega + 1$ are respectively displayed in blue, red and green. For each starting surfel, we draw a line between its center and the highest vertex of the final parallelepiped, i.e., $\arg\max_{x \in \{q^{(n)}, p^{(n)}\}} x \cdot \mathbf{N}$. The color of the arrow is the same as the one of its endpoint. On the right, we color in gray each surfel for which the algorithm passes through a reverse state during its execution.

that in average more steps are needed than in the previous experiments. This is due to the fact that we start not only from reentrant corners but also from many other surfels, making the computations harder. The same explanation holds for the number of calls to the predicate InPlane. Worst cases are the same as for the H and R^1-algorithms that is for plane normals of the form $(1, r, r)$. We also computed the number of times where we need to reverse

Algorithm	Avg. steps	Max. steps	Avg. InPlane	Avg. NotAbove
H	25.38	397	152.25	Not Applicable
R	19.25	397	271.13	Not Applicable
R^1	19.25	397	131.23	Not Applicable
PH	25.38	397	359.6	152.25
PR^1	19.25	397	321.7	131.23

Table 3: Number of steps, average and maximum number of calls to the predicate InPlane and NotAbove for processing normals with relatively prime components between $(1, 1, 1)$ and $(200, 200, 200)$ when starting from a reentrant corner of height 0.

PH	Steps	InPlane	NotAbove	Inversions
Average	26.01	892.92	156.05	4.31
Maximum	397	20367	2382	197
PR^1	Steps	InPlane	NotAbove	Inversions
Average	19.41	771.07	130.8	4.38
Maximum	397	20367	2580	197

Table 4: Average and maximum of number of steps, calls to the predicates InPlane, NotAbove and number of inversions (when we pass from a common to a reverse state) for processing normals with relatively prime components between $(1, 1, 1)$ and $(200, 200, 200)$ for all surfels with valid heights.

Algorithm	[22]	H	R	R^1	PH	PR^1
Constant	40	38	66	26	80	65

Table 5: Observed complexity constants for different plane-probing algorithms. For all the algorithms, the observed complexity is $O(\log \omega)$. The normal vectors are chosen in the following way: for each $i \in [1, 8]$, we pick 10000 random vectors with a L^1-norm of 10^i with a certain fixed deviation. This amounts to testing 80000 normal vectors with a L^1-norm ranging from 10 to 10^8. In order to be able to compare all algorithms, we start from a corner of height 0. Constants are obtained by doing a linear regression in log-space.

the parallelepiped: 0 for reentrant corners, 3.78 in average and 197 at the maximum, for $(1, 199, 199)$.

Finally, we studied the complexity of different plane-probing algorithms, see Table 5. We launched each algorithm on a set of normals with increasing 1-norm (from 10 to 10^8). We first confirmed that the observed complexity is indeed logarithmic for every method. We then did a linear regression in log-space to compute the complexity constant. We observe that the PH and PR^1-algorithms are twice more costly than the H and R^1 which is quite tractable considering that they can handle much more configurations.

5.3 Digital Surfaces

In a digital image partitionned into two voxel sets, one representing the interior and one the exterior, a digital surface is defined as a set of surfels, incident to an interior voxel and an exterior one and whose normal vector points to the second one. In this final section, we briefly describe how one can run our algorithm on digital surfaces. We start by replacing the predicate InPlane by a predicate P such that $P(x)$ returns true if and only if x is a 0-dimensional vertex of the digital surface. Note that the predicate NotAbove is built on top of P.

There are several difficulties in adapting the algorithm to digital surfaces:

- Contrary to the ideal case of an infinite digital plane, any digital surface that is not a piece of digital plane is a frontier between two voxel sets that are not necessarily convex. As in [24, Section 5.3], at each step $i \in \{0, \ldots, n\}$, we detect planarity defects when two points x, y aligned with $q^{(i)}$ are such that $q^{(i)} \in [xy]$ and $\text{NotAbove}(x) = \text{NotAbove}(y)$. In this case, the algorithm stops.
- The vertex set of the current parallelepiped cannot be always partitioned into two separable sets, one in the digital surface, one outside. The algorithm also stops as soon as this case is detected.
- The digital surface may be locally flat in one or two dimensions, i.e., one or two components of the local normal direction can be null. In the first case, we use a slightly modified version of the algorithm, similar to a 2D version, while in the second case, the starting surfel trivially provides the local normal direction and the final parallelepiped.
- The orthant of the normal is not known *a priori*. In this case, one can either launch the algorithm for each of the four possible orthants, or try to estimate it. We chose the second alternative in this preliminary work, following an idea similar to the one used in the implementation of the predicate NotAbove. From a surfel whose normal $\vec{u}$ points to the exterior, we first search for an exterior point s such that $s - \vec{u}$ is one vertex of the surfel. Then, from s, we consider rays of direction parallel to the two orthogonal sides of the surfel. If there are points that belong to the digital surface along these rays, we can estimate the normal orthant. Otherwise, the digital surface is assumed to be locally flat and we use a modified version of the algorithm as described above.

With the previous choices, we ran the algorithm on every surfel on the boundary of multiple digital surfaces, see Fig. 9. On the right, we display in red what we call the *separating polygons* that are either triangles or quads separating points inside the surface from points outside in the final parallelepiped (see Fig. 10 for a zoom on such polygons computed on a digitization of an ellipse). We first note that contrary to [24], we obtain an estimated normal for *every* surfel instead of only the ones touching a reentrant corner. We also see that the separating polygons form a rough linear approximation of the digital surface. One can notice that some of those polygons are flat, see for instance

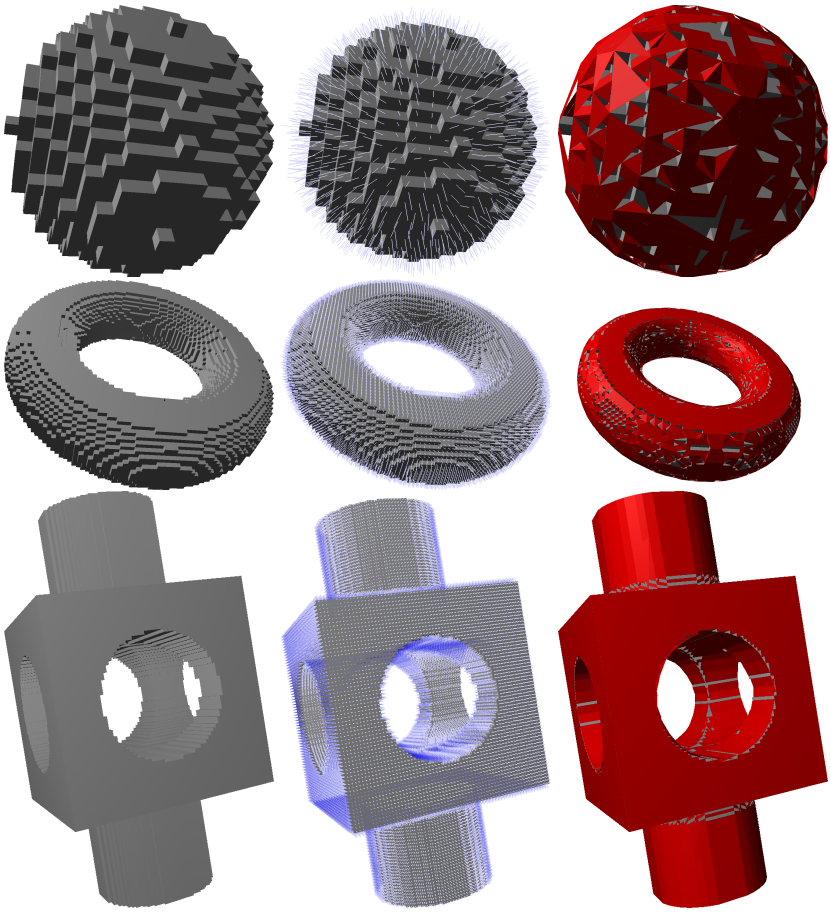


Fig. 9: Results of the PR^1-algorithm on different digital surfaces. From top to bottom: Sphere, Torus, Hollow. **Left:** digital surface in gray. **Middle:** estimated normals in blue. **Right:** separating polygons in red.

the right image of Fig. 10. This is due to the fact that we use a naive procedure to estimate the orthant of the normal before applying our method. We have also computed the running times and the number of calls to the predicates P and NotAbove of this algorithm on several digital surfaces, see Table 6. We can see that the number of calls to NotAbove (ant thus to P) can be quite large. This is due to the fact that the algorithm was slightly adapted when applied to digital surfaces as mentioned above. To conclude this section, we think that a good estimation of the orthant of the normal is not an easy task but is crucial for the geometric analysis. There is quite a lot of room for improvement in this direction.

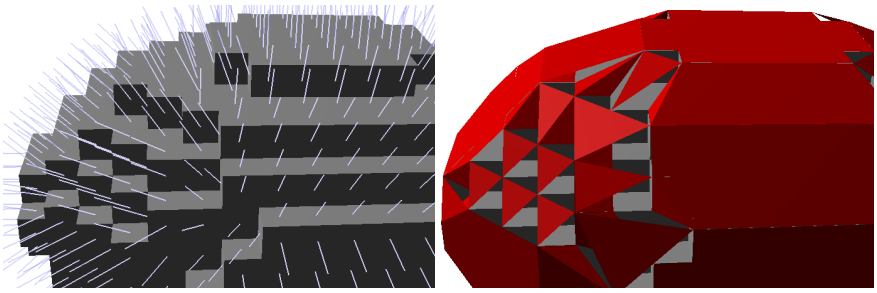


Fig. 10: Example of separating polygons (in red on the right) when using the PR^1-algorithm on the surfels of a digitization of an ellipse. One can clearly sees the triangles and quadrilaterals separating points inside and outside the shape on the right. Some polygons are flat due to the way the orthant of the normal is estimated before running our method.

Shape	Number of surfels	Time (ms)	Calls P	Calls NotAbove
Sphere	1902	42	481941	21578
Hollow	8392	863	9695460	158345
Torus	16232	453	4420893	259757
Fandisk-128	40424	1342	16361621	540195
Fandisk-256	165514	6694	77214440	2725493
Fandisk-512	666498	32823	358194686	12785576
Octaflower-512	671270	47600	359336259	28075048
CubeSphere-128	65280	1460	12589909	549477

Table 6: Running times of the PR^1-algorithm and number of calls to the predicates P and NotAbove when launched on all the boundary surfels of different digital surfaces.

6 Conclusion and Perspectives

We have presented a new framework for plane-probing algorithms. In particular, it is generic enough to extract the exact normal of the shape of interest if it is a digital plane, no matter the position we start from. It also outputs a tangent basis that is always reduced in practice if used together with the R or R^1-algorithm. Its complexity is comparable to the previous algorithms and experiments show that it can be useful for digital surface analysis.

There are still a lot of future works that we intend to explore. First, we wish to prove that the output basis is indeed reduced. This problem is difficult since the basis may not be reduced *during the parallelepiped evolution* but experimentally, we get that it is always reduced every two steps and at the end. Second, we wish to further study the localness of our algorithms: we would like to be able to bound the distance between the starting point and each vertex of the last parallelepiped and even bound the distance between the starting point and each probed point. Our new formulation of plane-probing as the deformation of a parallelepiped might help here, since distance can now

be measured in terms of state inversion. Third, much remains to be done for digital surface analysis. Our objective is to be able to decompose the surface into convex, concave and saddle parts. This could be carried out by analyzing nearby parallelepipeds. Last, we intend to improve the practical computational complexity on digital surface with clever initializations, thanks to a global analysis via Delaunay tessellation. This could solve the local orthant discovery for each surfel, and save a lot of computational effort.

References

1. Bonneel, N., Coeurjolly, D., Gueth, P., Lachaud, J.O.: Mumford-shah mesh processing using the ambrosio-tortorelli functional. In: Computer Graphics Forum, vol. 37, pp. 75–85. Wiley Online Library (2018)
2. Boulch, A., Marlet, R.: Fast and robust normal estimation for point clouds with sharp features. In: Computer graphics forum, vol. 31, pp. 1765–1774. Wiley Online Library (2012)
3. Boulch, A., Marlet, R.: Deep learning for robust normal estimation in unstructured point clouds. In: Computer Graphics Forum, vol. 35, pp. 281–290. Wiley Online Library (2016)
4. Buzer, L.: A linear incremental algorithm for naive and standard digital lines and planes recognition. Graphical Models **65**(1-3), 61–76 (2003). DOI 10.1016/S1524-0703(03)00008-0. URL http://linkinghub.elsevier.com/retrieve/pii/S1524070303000080
5. Charrier, E., Buzer, L.: An efficient and quasi linear worst-case time algorithm for digital plane recognition. In: Discrete Geometry for Computer Imagery (DGCI'2008), *LNCS*, vol. 4992, pp. 346–357. Springer (2008)
6. Charrier, E., Lachaud, J.O.: Maximal planes and multiscale tangential cover of 3d digital objects. In: Proc. Int. Workshop Combinatorial Image Analysis (IWCIA'2011), *Lecture Notes in Computer Science*, vol. 6636, pp. 132–143. Springer Berlin / Heidelberg (2011)
7. Chica, A., Williams, J., Andújar, C., Brunet, P., Navazo, I., Rossignac, J., Vinacua, À.: Pressing: Smooth isosurfaces with flats from binary grids. In: Computer Graphics Forum, vol. 27, pp. 36–46. Wiley Online Library (2008)
8. Coeurjolly, D., Foare, M., Gueth, P., Lachaud, J.O.: Piecewise smooth reconstruction of normal vector field on digital data. In: Computer Graphics Forum, vol. 35, pp. 157–167. Wiley Online Library (2016)
9. Coeurjolly, D., Lachaud, J.O., Levallois, J.: Multigrid convergent principal curvature estimators in digital geometry. Computer Vision and Image Understanding **129**, 27–41 (2014)
10. Cuel, L., Lachaud, J.O., Mérigot, Q., Thibert, B.: Robust geometry estimation using the generalized voronoi covariance measure. SIAM Journal on Imaging Sciences **8**(2), 1293–1314 (2015)
11. Cuel, L., Lachaud, J.O., Thibert, B.: Voronoi-based geometry estimator for 3d digital surfaces. In: E. Barcucci, A. Frosini, S. Rinaldi (eds.) Proc. Int. Conf. on Discrete Geometry for Computer Imagery (DGCI'2014), Sienna, Italy, *Lecture Notes in Computer Science*, vol. 8668, pp. 134–149. Springer International Publishing (2014). DOI 10.1007/978-3-319-09955-2_12. URL http://dx.doi.org/10.1007/978-3-319-09955-2_12
12. Debled-Rennesson, I., Reveillès, J.: An incremental algorithm for digital plane recognition. In: Proc. Discrete Geometry for Computer Imagery, pp. 194–205 (1994)
13. Dey, T.K., Goswami, S.: Tight cocone: a water-tight surface reconstructor. In: Proceedings of the eighth ACM symposium on Solid modeling and applications, pp. 127–134. ACM (2003)
14. Fernique, T.: Generation and recognition of digital planes using multi-dimensional continued fractions. Pattern Recognition **42**(10), 2229–2238 (2009)
15. Fleishman, S., Drori, I., Cohen-Or, D.: Bilateral mesh denoising. In: ACM transactions on graphics (TOG), vol. 22, pp. 950–953. ACM (2003)

16. Fourey, S., Malgouyres, R.: Normals estimation for digital surfaces based on convolutions. Computers & Graphics **33**(1), 2–10 (2009)
17. Françon, J., Papier, L.: Polyhedrization of the boundary of a voxel object. In: International Conference on Discrete Geometry for Computer Imagery, pp. 425–434. Springer (1999)
18. Gérard, Y., Debled-Rennesson, I., Zimmermann, P.: An elementary digital plane recognition algorithm. Discrete Applied Mathematics **151**(1), 169–183 (2005)
19. He, L., Schaefer, S.: Mesh denoising via l 0 minimization. ACM Transactions on Graphics (TOG) **32**(4), 64 (2013)
20. Klette, R., Sun, H.J.: Digital planar segment based polyhedrization for surface area estimation. In: Proc. Visual form 2001, *LNCS*, vol. 2059, pp. 356–366. Springer (2001)
21. Lachaud, J.O., Coeurjolly, D., Levallois, J.: Robust and convergent curvature and normal estimators with digital integral invariants. In: L. Najman, P. Romon (eds.) Modern Approaches to Discrete Curvature, *Lecture Notes in Mathematics*, vol. 2184, pp. 293–348. Springer International Publishing, Cham (2017). DOI 10.1007/978-3-319-58002-9_9. URL https://doi.org/10.1007/978-3-319-58002-9_9
22. Lachaud, J.O., Provençal, X., Roussillon, T.: An output-sensitive algorithm to compute the normal vector of a digital plane. Journal of Theoretical Computer Science (TCS) **624**, 73–88 (2016). DOI 10.1016/j.tcs.2015.11.021. URL https://hal.archives-ouvertes.fr/hal-01294966
23. Lachaud, J.O., Provençal, X., Roussillon, T.: Computation of the normal vector to a digital plane by sampling signicant points. In: 19th IAPR International Conference on Discrete Geometry for Computer Imagery. Nantes, France (2016). URL https://hal.archives-ouvertes.fr/hal-01621492
24. Lachaud, J.O., Provençal, X., Roussillon, T.: Two Plane-Probing Algorithms for the Computation of the Normal Vector to a Digital Plane. Journal of Mathematical Imaging and Vision **59**(1), 23 – 39 (2017). DOI 10.1007/s10851-017-0704-x. URL https://hal.archives-ouvertes.fr/hal-01621516
25. Lachaud, J.O., Thibert, B.: Properties of gauss digitized shapes and digital surface integration. Journal of Mathematical Imaging and Vision **54**(2), 162–180 (2016)
26. Mérigot, Q., Ovsjanikov, M., Guibas, L.J.: Voronoi-based curvature and feature estimation from point clouds. IEEE Transactions on Visualization and Computer Graphics **17**(6), 743–756 (2010)
27. Mesmoudi, M.M.: A Simplified Recognition Algorithm of Digital Planes Pieces. In: Proc. Discrete Geometry for Computer Imagery, pp. 404–416 (2002)
28. Provot, L., Debled-Rennesson, I.: 3d noisy discrete objects: Segmentation and application to smoothing. Pattern Recognition **42**(8), 1626–1636 (2009)
29. Reveillès, J.P.: Géométrie discrète, calculs en nombres entiers et algorithmique. Thèse d'etat, Université Louis Pasteur (1991)
30. Roussillon, T., Lachaud, J.O.: Digital Plane Recognition with Fewer Probes. In: 21st IAPR International Conference on Discrete Geometry for Computer Imagery, *Lecture Notes in Computer Science*, vol. 11414, pp. 380–393. Couprie M. and Cousty J. and Kenmochi Y. and Mustafa N., Springer, Cham, Marne-la-Vallée, France (2019). DOI 10.1007/978-3-030-14085-4_30. URL https://hal.archives-ouvertes.fr/hal-02087529
31. Sivignon, I., Dupont, F., Chassery, J.M.: Decomposition of a three-dimensional discrete object surface into discrete plane pieces. Algorithmica **38**(1), 25–43 (2004)
32. Veelaert, P.: Digital planarity of rectangular surface segments. IEEE Transactions on Pattern Analysis and Machine Intelligence **16**(6), 647–652 (1994)
33. Veelaert, P.: Fast Combinatorial Algorithm for Tightly Separating Hyperplanes. In: Proc. Int. Workshop Combinatorial Image Analysis (IWCIA'2012), pp. 31–44 (2012)
34. Zhang, W., Deng, B., Zhang, J., Bouaziz, S., Liu, L.: Guided mesh normal filtering. In: Computer Graphics Forum, vol. 34, pp. 23–34. Wiley Online Library (2015)