



Two Plane-Probing Algorithms for the Computation of the Normal Vector to a Digital Plane

Jacques-Olivier Lachaud, Xavier Provençal, Tristan Roussillon

► To cite this version:

Jacques-Olivier Lachaud, Xavier Provençal, Tristan Roussillon. Two Plane-Probing Algorithms for the Computation of the Normal Vector to a Digital Plane. *Journal of Mathematical Imaging and Vision*, 2017, 59 (1), pp.23 - 39. 10.1007/s10851-017-0704-x . hal-01621516

HAL Id: hal-01621516

<https://hal.science/hal-01621516>

Submitted on 23 Oct 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Two plane-probing algorithms for the computation of the normal vector to a digital plane *

Jacques-Olivier Lachaud Xavier Provençal
Tristan Roussillon

October 4, 2017

Abstract

Digital planes are sets of integer points located between two parallel planes. We present a new algorithm that computes the normal vector of a digital plane given only a predicate “is a point x in the digital plane or not”. In opposition to classical recognition algorithm, this algorithm decides *on-the-fly* which points to test in order to output at the end the exact surface characteristics of the plane. We present two variants: the H -algorithm, which is purely local, and the R -algorithm which probes further along rays coming out from the local neighborhood tested by the H -algorithm. Both algorithms are shown to output the correct normal to the digital planes if the starting point is a lower leaning point. The worst-case time complexity is in $O(\omega)$ for the H -algorithm and $O(\omega \log \omega)$ for the R -algorithm, where ω is the arithmetic thickness of the digital plane. In practice, the H -algorithm often outputs a reduced basis of the digital plane while the R -algorithm always returns a reduced basis. Both variants perform much better than the theoretical bound, with an average behavior close to $O(\log \omega)$. Finally we show how this algorithm can be used to analyze the geometry of arbitrary digital surfaces, by computing normals and identifying convex, concave or saddle parts of the surface. This paper is an extension of [16].

1 Introduction

The study of the linear geometry of digital sets has raised a considerable amount of work in the digital geometry community. In 2D, digital straightness has been extremely fruitful. Indeed, digital straight lines present many properties, whether geometric, arithmetic, or combinatoric (e.g. see survey [13]). Furthermore, these properties have impacted the practical analysis of 2D shapes,

*This work has been partly funded by DIGITALSNOW ANR-11-BS02-009 research grant and CoMeDiC ANR-15-CE40-0006 research grant.

especially through the notion of *maximal segments* [6, 22], which are unextensible pieces of digital straight lines along digital contours. To sum up, they were shown to be characteristic of convex and concave parts [8, 6, 20], to induce several multigrid convergence results [22, 18] and even to be able to identify noisy parts of the contour [11].

A lot was thus expected from the study of 3D digital planes (e.g. see the survey [2]). It is true that several plane recognition algorithms were proposed to determine if a given set of points could be a piece of digital plane. Among them, the most fruitful ones adopt a geometric approach [12, 21, 9, 3]. A first difficulty is the more complex combinatorial structure of digital planes [15], which appears when studying its connectedness [10] or the link with multidimensional continued fractions [7, 1]. Their multiple definitions become obvious when looking at the geometry of digital planes: there are multiple ways of approximating them depending on the chosen point of view.

These ambiguities make practical 3D digital shape analysis difficult. The segmentation of the shape boundary into linear parts is thus generally greedy [14, 19, 5], for instance by extracting first the biggest plane, and then repeat the process [23]. In 3D, the main problem is that there is no more an implication between “being a maximal plane” and “being a tangent plane” as it is in 2D. This was highlighted in [4], where maximal planes were then defined as planar extension of maximal disks. Contrary to the 2D case, the surface topology around the point of interest does not give us sufficient constraints to identify unambiguously the set of points that should define the tangent plane. To sum up, the problem is not so much to recognize a piece of plane, but more to group together the pertinent points onto the digital shape.

We need thus methods that identify locally significant points to test and extract plane parameters at the same time. From now on, we have as input a starting point in some plane \mathbf{P} and a predicate that answers to the question “is point \vec{x} in plane \mathbf{P} ?”. The objective is to find the exact plane parameters solely from this information, by testing points as locally as possible. We call *plane-probing algorithms* that class of plane recognition algorithms.

We proposed a first approach to solve this problem in [17]. Its principle is to deform an initial unit tetrahedron based at the 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 is in \mathbf{P} , with the same volume, and is pushed towards one side of the plane. At the end of this iterative process, one face of the tetrahedron has an extremal position in the plane and is thus parallel to \mathbf{P} .

In [16], we designed a new algorithm for this problem. It shares some features of the previous one, because it is also an iterative process that deforms an initial tetrahedron and stops when one face is parallel to \mathbf{P} . It differs from it on several points, as illustrated on fig. 1. One vertex of the evolving tetrahedron is the fixed point lying above the starting point and the opposite triangular facet. The position of the evolving tetrahedron is thus better controlled than in [17]. Moreover, this new algorithm is mostly a geometrical algorithm using Delaunay circumsphere property, while the former was mostly arithmetic. Its theoretical

complexity is slightly better in theory, since it drops a log factor, and in practice also.

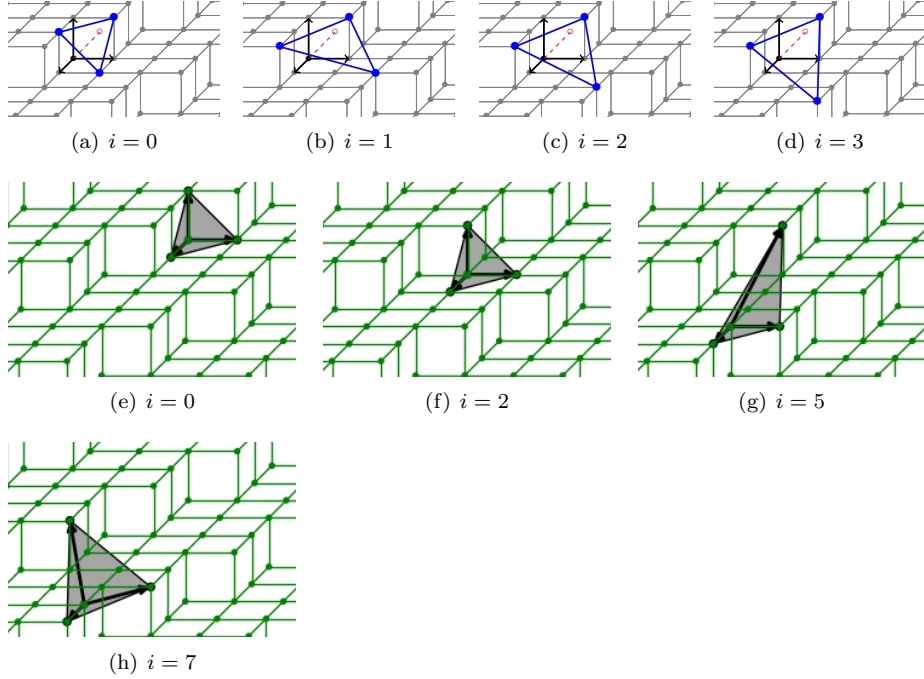


Figure 1: Illustration of the running of [16, Algorithm 2] and the algorithm from [17] on a digital plane of normal vector $(1, 2, 5)$. Images (a) to (d) show the four iterations of [16, Algorithm 1] starting from the origin. Each triangle is intersected by the small dashed red segment. In (d), the normal of the last triangle is $(1, 2, 5)$. Images (e) to (h) show iterations 0 (initial), 2, 5 and 7 (final) of the algorithm from [17]. The initial tetrahedron (a) is placed at the origin and the final one (h) has an upper triangle with normal vector $(1, 2, 5)$.

This paper extends in several ways the DGCi paper [16]:

- The H -algorithm is essentially the same algorithm as Algorithm 2 of [16]. We give a new variant, called R -algorithm, for finding the next tetrahedron, which guarantees that, at the end of the process, the output facet forms a reduced basis of the plane \mathbf{P} . For now, we observe this property but we are not able to fully prove it.
- The running time of this new variant is similar to the previous algorithms, and even better for big ω .
- We detail how this algorithm can be used for digital shape analysis. We show how to detect convex/concave/inflexion zones onto a digital surface.

- For some well-identified starting points, this algorithm stops and outputs only an approximation of the normal to \mathbf{P} . We show how to detect such *bad* starting points and how to connect them to their corresponding facet.
- We give a comprehensive experimental evaluation of its time complexity and compare it extensively with the plane-probing algorithm of [17].

The paper is organized as follows. First, we give basic definitions and present the H - and the R -algorithms. Then we show their correctness and provide worst-case time complexity. We then conduct an experimental evaluation, which shows that the new variant always output a reduced basis. The last part describes how this algorithm can be used to analyze the linear geometry of digital surfaces.

2 Two plane-probing algorithms

We introduce a few notations before presenting in a unified manner our new plane-probing algorithms: the H - and the R -algorithms.

2.1 Digital plane and initialization

We wish to extract the parameters of an arbitrary standard digital plane \mathbf{P} , defined as the set

$$\mathbf{P} = \{\vec{x} \in \mathbb{Z}^3 \mid \mu \leq \vec{x} \cdot \mathbf{N} < \mu + \mathbf{s} \cdot \mathbf{N}\},$$

where $\mathbf{N} \in \mathbb{Z}^3$ is the *normal vector* whose components (a, b, c) are relatively prime, $\mu \in \mathbb{Z}$ is the *intercept*, \mathbf{s} is the *shift vector*. In the standard case, the shift vector is equal to $(\pm 1, \pm 1, \pm 1)$, where the sign of the components are chosen so that the *thickness* $\omega := \mathbf{s} \cdot \mathbf{N}$ is equal to $|a| + |b| + |c|$. Moreover, we extract a *basis of \mathbf{P}* , that is a pair of vectors that forms a basis of the 2D lattice $\{\vec{x} \in \mathbb{Z}^3 \mid \vec{x} \cdot \mathbf{N} = 0\}$. A basis of a two dimensional lattice is *reduced* if and only if it consists of the two shortest non-zero lattice vectors.

By translation, we assume w.l.o.g. that $\mu = 0$. Moreover, by symmetry, we assume w.l.o.g. that the components of the normal vector are all positive. We also exclude cases where a component is null since then it falls back to a 2D algorithm. Thus, $a, b, c > 0$, which implies that $\mathbf{s} = (1, 1, 1)$.

We may see the space as partitioned into layers of coplanar points, orthogonal to \mathbf{N} . The *height* $\vec{x} \cdot \mathbf{N}$ sorts these layers along direction \mathbf{N} . Points of height 0 and $\omega - 1$ are extreme within \mathbf{P} , and are called *lower leaning points* and *upper leaning points* respectively.

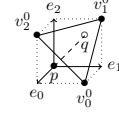
We propose an algorithm that, given a predicate “is $\vec{x} \in \mathbf{P}$?”, computes the *normal vector* of a piece of digital plane surrounding a starting point $\vec{p} \in \mathbf{P}$.

The algorithm places an initial sequence of 3 points $\mathbf{T}^{(0)} := (\vec{v}_k^{(0)})_{k \in \{0,1,2\}}$ such that

$$\forall k, \vec{v}_k^{(0)} := \vec{p} + \vec{e}_k + \vec{e}_{k+1},$$

(for sake of clarity, we write $\forall k$ instead of $\forall k \in \mathbb{Z}/3\mathbb{Z}$) and requires that $\mathbf{T}^{(0)} \subset \mathbf{P}$.

It is easy to check that $\mathbf{T}^{(0)} \subset \mathbf{P}$ for any \vec{p} such that $0 \leq \vec{p} \cdot \mathbf{N} < \min\{a, b, c\}$, which corresponds to points lying inside *reentrant corners* (see fig. on the right). The algorithm then iteratively updates this initial point set by calling the above predicate for well-chosen points. We explain in the next subsection how the *shifted point* $\vec{q} := \vec{p} + \vec{s}$, which is not in \mathbf{P} because $\vec{s} \cdot \mathbf{N}$ is the thickness of \mathbf{P} , is used to select those points.



2.2 Iteration and termination

At any step $i \in \mathbb{N}$, the tangent plane surrounding the starting point is described by a sequence of 3 points denoted by $\mathbf{T}^{(i)}$.

For any finite point set S , let us denote by $\text{conv}(S)$ its convex hull and $\text{aff}(S)$ its affine hull. Thus, $\text{conv}(\mathbf{T}^{(i)})$ is a triangle, whose three counterclockwise oriented vertices are denoted by $(\vec{v}_k^{(i)})_{k \in \{0,1,2\}}$ (see fig. 2), whereas $\text{aff}(\mathbf{T}^{(i)})$ is the plane passing by these vertices.

In order to update $\mathbf{T}^{(i)}$, the algorithm checks whether the points of some neighborhood around \vec{q} and above $\text{aff}(\mathbf{T}^{(i)})$ belong to \mathbf{P} or not. Before defining two such neighborhoods, let us introduce the following notation:

$$\forall k, \vec{m}_k^{(i)} := \vec{q} - \vec{v}_k^{(i)}. \quad (1)$$

The following two neighborhoods are some subsets of a cone of apex $\vec{q} - \sum_k \vec{m}_k^{(i)}$ and base $\{\vec{m}_k^{(i)}\}_{k \in \{0,1,2\}}$ (see fig. 2).

First, let us define the *H-neighborhood* at step i as follows:

$$\mathcal{N}_H^{(i)} := \{\vec{q} - \vec{m}_{\sigma(0)}^{(i)} + \vec{m}_{\sigma(1)}^{(i)}\}_{\sigma, \text{ a permutation over } \{0,1,2\}}, \quad (2)$$

In other words, $\mathcal{N}_H^{(i)} := \{\vec{q} \pm (\vec{m}_k^{(i)} - \vec{m}_{k+1}^{(i)})\}_{k \in \{0,1,2\}}$. This set consists of six points arranged in an hexagon and that is why it is called *H-neighborhood* (see red disks in fig. 2). We define below the *R-neighborhood*, using the notion of *ray*, with:

$$\mathcal{R}_\sigma^{(i)} := (\vec{q} - \vec{m}_{\sigma(0)}^{(i)} + \vec{m}_{\sigma(1)}^{(i)} + \lambda \vec{m}_{\sigma(2)}^{(i)})_{\lambda \geq 0}, \quad (3)$$

$$\mathcal{N}_R^{(i)} := \{\mathcal{R}_\sigma^{(i)}\}_{\sigma, \text{ a permutation over } \{0,1,2\}}. \quad (4)$$

Looking at the first definition of *H-neighborhood*, it is clear that the *R-neighborhood* contains it and extends it along rays that go out of the hexagon, hence the name *R-neighborhood* (see green squares in fig. 2). We thus have $\mathcal{N}_H^{(i)} \subset \mathcal{N}_R^{(i)}$.

Let $\mathcal{N}^{(i)}$ be any neighborhood in $\{\mathcal{N}_H^{(i)}, \mathcal{N}_R^{(i)}\}$. Our algorithm selects a point \vec{x}^* of $\mathcal{N}^{(i)} \cap \mathbf{P}$ such that the circumsphere of $\mathbf{T}^{(i)} \cup \{\vec{x}^*\}$ does not include any point of $\mathcal{N}^{(i)} \cap \mathbf{P}$ in its interior. The upper triangular facet of the tetrahedron $\text{conv}(\mathbf{T}^{(i)} \cup \{\vec{x}^*\})$ intersected by the straight line passing by \vec{p} and \vec{q} is the new triangle $\text{conv}(\mathbf{T}^{(i+1)})$.

Let us introduce the edge vectors of $\text{conv}(\mathbf{T}^{(i)})$:

$$\forall k, \vec{d}_k^{(i)} := \vec{v}_{k+1}^{(i)} - \vec{v}_k^{(i)} = \vec{m}_k^{(i)} - \vec{m}_{k+1}^{(i)}. \quad (5)$$

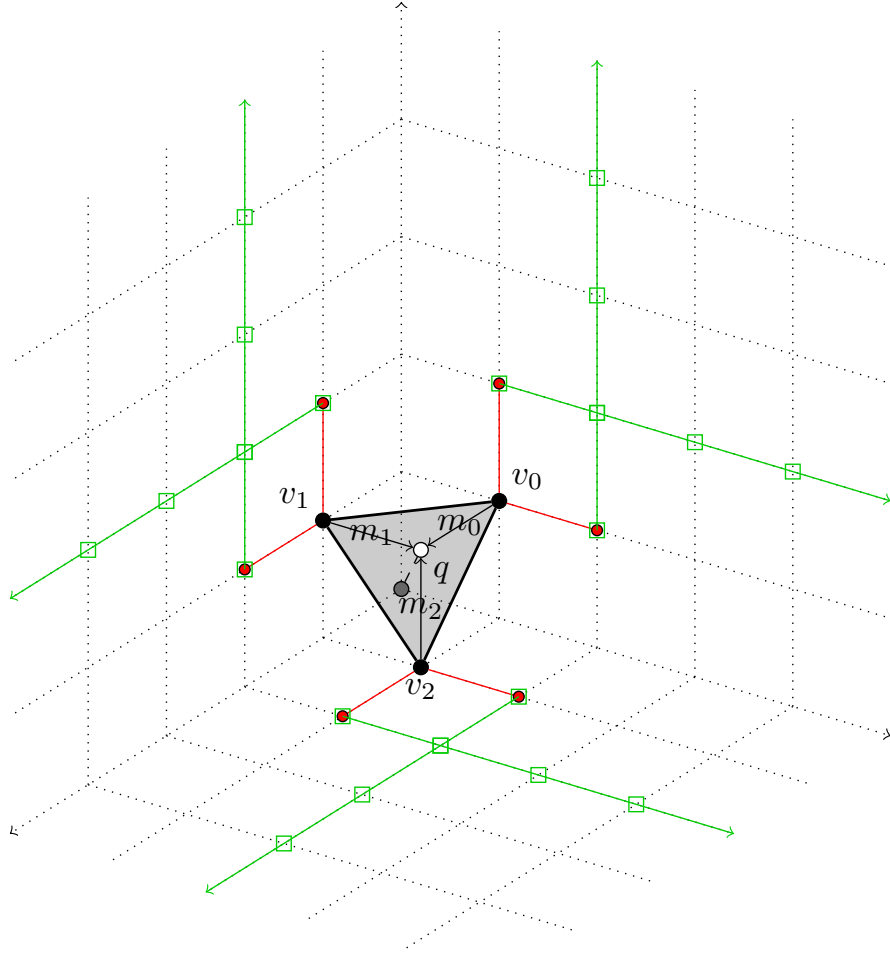


Figure 2: The triangle $\text{conv}(\mathbf{T}^{(i)})$ is depicted in grey. The set $\mathcal{N}_H^{(i)}$, called the *hexagon* at step i , is depicted with red disks, whereas the set $\mathcal{N}_R^{(i)}$ is depicted with green squares, located along rays coming out of the hexagon vertices. We can see that $\mathcal{N}_H^{(i)} \subset \mathcal{N}_R^{(i)}$. (Iteration number is dropped for sake of clarity).

The normal of $\text{aff}(\mathbf{T}^{(i)})$, denoted by $\hat{\mathbf{N}}(\mathbf{T}^{(i)})$, is merely defined by the cross product between two consecutive edge vectors of $\text{conv}(\mathbf{T}^{(i)})$, i.e. $\hat{\mathbf{N}}(\mathbf{T}^{(i)}) = \vec{d}_0^{(i)} \times \vec{d}_1^{(i)}$.

The algorithm stops at a step n , when $\mathcal{N}^{(n)}$ has an empty intersection with \mathbf{P} . The output of the algorithm is $\mathbf{T}^{(n)}$. We will prove in sec. 3.3 that if \vec{p} is a lower leaning point, then the points of $\mathbf{T}^{(n)}$ are upper leaning points of \mathbf{P} and that the normal $\hat{\mathbf{N}}(\mathbf{T}^{(n)})$ of the triangle $\text{conv}(\mathbf{T}^{(n)})$ is aligned with the normal \mathbf{N} of \mathbf{P} .

2.3 Unified presentation of plane-probing algorithms

Algorithm 1 summarizes our two variants for recognizing on-the-fly a digital plane. The predicate “is \vec{x} in \mathbf{P} ?” is used to compute the intersection between the neighborhood and \mathbf{P} (see lines 2 and 3). Clearly, Algorithm 1 remains around its starting point by construction, since every triangle has a non-empty intersection with the straight line passing by \vec{p} and \vec{q} . It also stays as local as possible with the empty circumsphere property.

Algorithm 1: Unified plane-probing algorithm: for any neighborhood definition (H or R), it extracts a 3-point sequence of upper leaning points

Input: a shift vector \vec{s} , a point \vec{q} and an initial 3-point sequence $\mathbf{T}^{(0)}$

```

1  $i \leftarrow 0$  ;
2 while  $\mathcal{N}^{(i)} \cap \mathbf{P} \neq \emptyset$  do
3   Compute a point  $\vec{x}^* \in (\mathcal{N}^{(i)} \cap \mathbf{P})$  such that the circumsphere of
    $\mathbf{T}^{(i)} \cup \{\vec{x}^*\}$  does not include any point  $\vec{x} \in (\mathcal{N}^{(i)} \cap \mathbf{P})$  in its interior ;
4   Find  $\mathbf{T}^{(i+1)}$ , defined as the vertex sequence of the upper facet of
    $\text{conv}(\mathbf{T}^{(i)} \cup \{\vec{x}^*\})$  intersected at a single point by the straight line of
   direction  $\vec{s}$  passing by  $\vec{q}$  ;
5    $i \leftarrow i + 1$  ;
6 return  $\mathbf{T}^{(i)}$ ;
```

The algorithm variant using the H -neighborhood is called the *H-algorithm*, whereas the one using the R -neighborhood is called the *R-algorithm*. We show in the next section that both variants always extract the exact parameters of plane \mathbf{P} in $O(\omega)$ iterations in worst cases. As we will see in the experimental section, the *H-algorithm* *often* extracts the reduced basis of \mathbf{P} , while the *R-algorithm* *always* extracts the reduced basis of \mathbf{P} in our experiments. In most cases, it falls back to the *H-algorithm*, but sometimes it looks for points further away along rays that go out of the hexagon.

To end, note that the *H-algorithm* is slightly different from [16, 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 [16, Algorithm 2]. This choice leads to a simpler characterization (by lemma 1-item 5, two consecutive triangles must share exactly two vertices, instead of one or two in [16, lemma 1]),

while the number of iterations is not changed ($O(\omega)$ in both case). The result is even better, since the H -algorithm more often leads to a reduced basis of upper leaning points than [16, Algorithm 2] (for all 6578833 vectors with relatively prime components ranging from $(1, 1, 1)$ to $(200, 200, 200)$, the H -algorithm returns 480 non reduced basis against 924 with [16, Algorithm 2], see sec. 4).

2.4 Implementation details

In algorithm 1, line 3, at a step i , we have to compute a point $\vec{x}^* \in (\mathcal{N}^{(i)} \cap \mathbf{P})$ such that the sphere passing by $\mathbf{T}^{(i)} \cup \{\vec{x}^*\}$ does not contain any other point $\vec{x} \in (\mathcal{N}^{(i)} \cap \mathbf{P})$ in its interior. We say below that \vec{x}^* is a *closest point* to $\mathbf{T}^{(i)}$, since the sphere passing by $\mathbf{T}^{(i)} \cup \{\vec{x}^*\}$ has minimal radius (over the set of spheres passing by $\mathbf{T}^{(i)} \cup \{\vec{x}\}$).

Searching for a closest point to $\mathbf{T}^{(i)}$ is trivial and in $O(1)$ for the H -algorithm, because the H -neighborhood is finite and its intersection with \mathbf{P} has at most 6 points. The algorithm is then similar to finding the minimum element of a sequence: we take an arbitrary point $\vec{y} \in (\mathcal{N}^{(i)} \cap \mathbf{P})$ as a current closest point and for each remaining point \vec{x} of this set, if the sphere passing by $\mathbf{T}^{(i)} \cup \{\vec{y}\}$ strictly contains \vec{x} , then \vec{x} becomes the new current closest point (see algorithm 2).

Algorithm 2: ClosestPointInSet(\mathbf{T}, S)

Input: a 3-point sequence \mathbf{T} , a point set S

- 1 Let \vec{y} be an arbitrary point of S ;
- 2 **foreach** $\vec{x} \in S$ **do**
- 3 **if** *the sphere passing by $\mathbf{T} \cup \{\vec{y}\}$ strictly contains \vec{x}* **then**
- 4 $\vec{y} \leftarrow \vec{x}$;
- 5 **return** \vec{y} ;

Searching for a closest point to $\mathbf{T}^{(i)}$ is more tricky for the R -algorithm, because the R -neighborhood is unbounded. Two questions may be raised: how to compute its intersection with \mathbf{P} and how to determine a closest point without visiting all the points of the intersection?

We omit the iteration exponent (i) below to simplify the presentation. Algorithm 3 processes each ray of the R -neighborhood separately, because over one ray, we can efficiently find a closest one to \mathbf{T} in \mathbf{P} and add it to the candidate point set. Then, we compute a closest point over the candidate point set by algorithm 2 in $O(1)$ since this set contains at most 6 points (one per ray).

Along a ray, we have two different tasks to perform: find the points that belong to \mathbf{P} and over such points, find a closest one to \mathbf{T} .

First, the starting point of a given ray is guaranteed to be in \mathbf{P} (see algorithm 3, line 3). Second, knowing that $\vec{q} \notin \mathbf{P}$ and $T \subset \mathbf{P}$ (see lemma 1-item 1) and by equations (1) and (3), there always exist some points of a ray that are not in \mathbf{P} . Third, due to the convexity of \mathbf{P} , all the points not in \mathbf{P} must follow the points in \mathbf{P} on a given ray (see lemma 7). Locating the furthest point in \mathbf{P}

Algorithm 3: Search for a closest point to a 3-point sequence \mathbf{T} over the R -neighborhood.

Input: a 3-point sequence \mathbf{T} and a shifted point \vec{q}

- 1 $CandidatePointSet \leftarrow \emptyset$;
- 2 Compute the set of rays $\{\mathcal{R}_\sigma\}_\sigma$, a permutation over $\{0,1,2\}$ from \mathbf{T} and \vec{q} ;
- 3 **foreach** ray \mathcal{R}_σ **do**
- 4 **if** the starting point of \mathcal{R}_σ is in \mathbf{P} **then**
- 5 $\vec{x} \leftarrow \text{ClosestPointInRay}(\mathbf{T}, \mathcal{R}_\sigma)$;
- 6 $CandidatePointSet \leftarrow CandidatePointSet \cup \{\vec{x}\}$;
- 7 **return** $\text{ClosestPointInSet}(\mathbf{T}, CandidatePointSet)$;

on a ray can be done in two stages. In the first stage, one advance in the ray direction by doubling the step at each iteration until a point not in \mathbf{P} is found. The last two points determine a range whose lower bound is in \mathbf{P} , but upper bound is not in \mathbf{P} . In the second stage, a binary search is performed on this range. The overall time complexity is logarithmic in the number of points in \mathbf{P} on a given ray.

Finding a closest point to \mathbf{T} may be done in a similar way. Indeed, the function that maps a point \vec{x} on the ray to the radius of the sphere passing by $\mathbf{T} \cup \{\vec{x}\}$ is convex and has a global minimum, because the ray does not intersect $\text{aff}(\mathbf{T})$ (see lemma 1-item 3 and corollary 2).

In fig. 3, we provide a two-dimensional illustration of this function. For each point \vec{x} on the ray, depicted with green boxes, the spheres passing by $\mathbf{T} \cup \{\vec{x}\}$ are depicted with blue circles. We can see that in the ray direction, the function is first decreasing, reaches its minimum at the middle point before increasing.

Thus, in algorithm 4, we use two predicates in the exponential march and the binary search to find a closest point to \mathbf{T} in \mathbf{P} : the predicate “is \vec{x} in \mathbf{P} ” because we are only interested in points in \mathbf{P} , and the in-sphere test: “does the sphere passing by $\mathbf{T} \cup \{\vec{x}\}$ contain \vec{y} ?” in order to find a closest point. This in-sphere test is made with a determinant computation.

3 Correctness of plane exploration algorithms

We start by giving some properties that are valid for both algorithms. We denote by n the last iteration. The proof that the algorithm always terminates is postponed to section 3.2, theorem 1.

3.1 Algorithm invariants and characterization of update operations

Lemma 1 *The following properties are true:*

1. $\forall i \in \{0, \dots, n\}, \forall k, \vec{v}_k^{(i)} \in \mathbf{P}$.

Algorithm 4: ClosestPointInRay($\mathbf{T}, \mathcal{R}_\sigma, \vec{q}$)

Input: a 3-point sequence \mathbf{T} , a ray \mathcal{R}_σ , a shifted point \vec{q}

```

1  $\vec{x} \leftarrow \vec{q} - \vec{m}_{\sigma(0)} + \vec{m}_{\sigma(1)}$  ; // starting point of the ray  $\mathcal{R}_\sigma$  (must be
   in  $\mathbf{P}$ )
2  $\vec{y} \leftarrow \vec{m}_{\sigma(2)}$  ; // direction vector of the ray  $\mathcal{R}_\sigma$ 
   // exponential march
3  $\kappa \leftarrow 0, \lambda \leftarrow 1$  ; // lower and upper bound respectively
4 while  $\vec{x} + \kappa\vec{y} \in \mathbf{P}$  and the sphere passing by  $\mathbf{T} \cup \{\vec{x} + \kappa\vec{y}\}$  contains
    $\vec{x} + \lambda\vec{y}$  do
5    $\lfloor \kappa \leftarrow \lambda, \lambda \leftarrow 2\lambda$  ;
6    $\kappa \leftarrow \lfloor \frac{\kappa}{2} \rfloor$  ;
   // binary search
7 while  $(\lambda - \kappa) > 4$  do
   // invariant:  $\vec{x} + \kappa\vec{y} \in \mathbf{P}$ 
8    $\alpha = \lfloor \frac{(3\kappa + \lambda)}{4} \rfloor, \beta = \lfloor \frac{(\kappa + \lambda)}{2} \rfloor, \gamma = \lfloor \frac{(\kappa + 3\lambda)}{4} \rfloor$  ; //  $\kappa < \alpha < \beta < \gamma < \lambda$ 
9   if  $\vec{x} + \beta\vec{y} \in \mathbf{P}$  and the sphere passing by  $\mathbf{T} \cup \{\vec{x} + \beta\vec{y}\}$  contains  $\vec{x} + \gamma\vec{y}$ 
   then
10     $\lfloor \kappa \leftarrow \beta$  ;
11   else if  $\vec{x} + \alpha\vec{y} \notin \mathbf{P}$  or the sphere passing by  $\mathbf{T} \cup \{\vec{x} + \beta\vec{y}\}$  contains
    $\vec{x} + \alpha\vec{y}$  then
12     $\lfloor \lambda \leftarrow \beta$  ;
13   else
14     $\lfloor \kappa \leftarrow \alpha, \lambda \leftarrow \gamma$  ;
15 return ClosestPointInSet( $\mathbf{T}, \{\vec{x} + \delta\vec{y}\}_{\delta \in [\kappa, \lambda]}$ ) ;

```

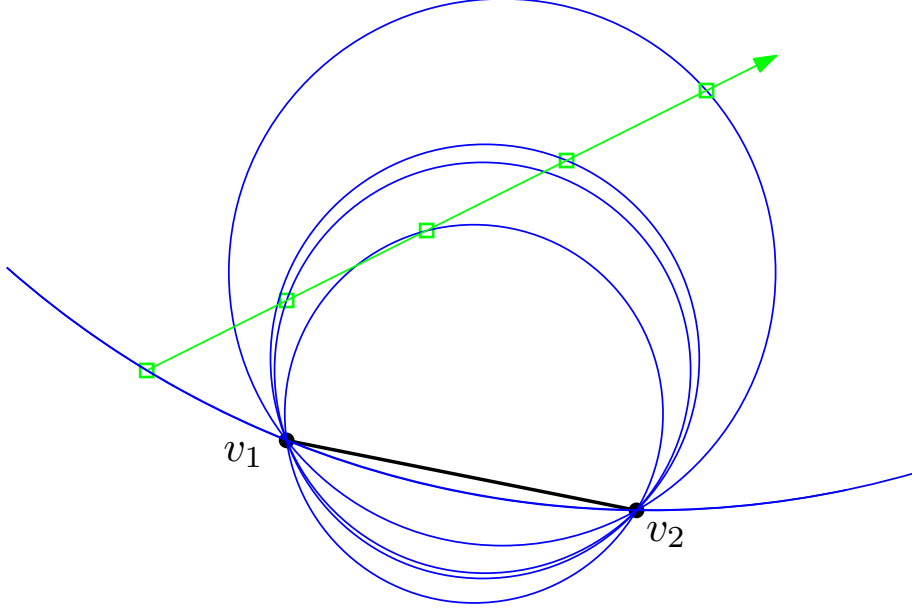


Figure 3: Two-dimensional illustration of the function that maps a point \vec{x} on a ray (in green) to the radius of the sphere passing by $\mathbf{T} \cup \{\vec{x}\}$ (blue circles).

2. $\forall i \in \{0, \dots, n\}$, $\{\vec{m}_k^{(i)}\}_{k \in \{0,1,2\}}$ are linearly independent.
3. $\forall i \in \{0, \dots, n\}$, \vec{q} is strictly above $\text{aff}(\mathbf{T}^{(i)})$ in direction \vec{s} .
4. $\forall i \in \{0, \dots, n-1\}$, for a point $\vec{x}^* \in (\mathcal{N}^{(i)} \cap \mathbf{P})$, the convex hull of $\mathbf{T}^{(i)} \cup \{\vec{x}^*\}$ has non-zero volume.
5. $\forall i \in \{0, \dots, n-1\}$, $\mathbf{T}^{(i)}$ and $\mathbf{T}^{(i+1)}$ have two vertices in common.

Proof. We prove the first item by induction. Others follow. The first property is obviously true for $i = 0$. Let us assume now that it is true for $i < n$. The set $(\mathcal{N}^{(i)} \cap \mathbf{P})$ contains at least one point because $i < n$ (when the set is empty, the algorithm stops and $i = n$). For a point $\vec{x}^* \in (\mathcal{N}^{(i)} \cap \mathbf{P})$, let us consider the tetrahedron defined as the convex hull of $\mathbf{T}^{(i)} \cup \{\vec{x}^*\}$. Since $\mathbf{T}^{(i)} \subset \mathbf{P}$ by the induction hypothesis, any three-point subsets of $\mathbf{T}^{(i)} \cup \{\vec{x}^*\}$ (and therefore $\mathbf{T}^{(i+1)}$) belong to \mathbf{P} , which proves item 1.

We can now prove items 2 and 3. Indeed, let us assume that $\{\vec{m}_k^{(i)}\}_{k \in \{0,1,2\}}$ are coplanar. Then, by construction (algorithm 1, line 4), \vec{q} belongs to $\text{conv}(\mathbf{T}^{(i)})$. However, since $\vec{q} \notin \mathbf{P}$ and $\mathbf{T}^{(i)} \subset \mathbf{P}$, this contradicts the convexity of \mathbf{P} , which proves item 2 by contradiction.

The same argument of convexity may be used to prove that \vec{q} is strictly above $\text{aff}(\mathbf{T}^{(i)})$ in direction \vec{s} for all $i \in \{0, \dots, n\}$ (item 3).

To end, we prove items 4 and 5. Since \vec{q} is strictly above $\text{aff}(\mathbf{T}^{(i)})$ and since the neighborhoods are defined in (2) and (4) by vectors $\{\vec{m}_k^{(i)}\}_{k \in \{0,1,2\}}$ going from points of $\mathbf{T}^{(i)}$ to \vec{q} (by (1)), all the points of $\mathcal{N}^{(i)}$ are strictly above $\text{aff}(\mathbf{T}^{(i)})$ in direction \vec{s} . As a consequence, the convex hull of $\mathbf{T}^{(i)} \cup \{\vec{x}^*\}$ has non-zero volume (item 4) and $\mathbf{T}^{(i)} \neq \mathbf{T}^{(i+1)}$. Since the convex hull of $\mathbf{T}^{(i)} \cup \{\vec{x}^*\}$ is a non degenerate tetrahedron, there are three triangular facets different from $\text{conv}(\mathbf{T}^{(i)})$. All of them (including $\text{conv}(\mathbf{T}^{(i+1)})$) share exactly two vertices with $\mathbf{T}^{(i)}$ and have point \vec{x}^* as third vertex (item 5). \square

The following lemma fully characterizes the main operation of algorithm 1 (lines 3-5):

Lemma 2

$\forall i \in \{0, \dots, n-1\}, \exists k^* \text{ s.t.}$

$$\begin{cases} \vec{v}_{k^*}^{(i+1)} = \vec{v}_{k^*}^{(i)} + \alpha \vec{m}_{k^*+1}^{(i)} + \beta \vec{m}_{k^*+2}^{(i)}, \\ \text{with } \alpha, \beta \in \mathbb{N}, \alpha = 1 \text{ or } \beta = 1, \alpha + \beta \geq 1, \\ \vec{v}_{k^*+1}^{(i+1)} = \vec{v}_{k^*+1}^{(i)}, \\ \vec{v}_{k^*+2}^{(i+1)} = \vec{v}_{k^*+2}^{(i)} \end{cases}$$

Proof. Let us assume w.l.o.g. that $k^* = 0$. By algorithm 1, (2) and (4), for a permutation σ over $\{0, 1, 2\}$, $\forall i \in \{0, \dots, n-1\}$, $\vec{v}_0^{(i+1)} = \vec{q} - \vec{m}_{\sigma(0)}^{(i)} + \alpha \vec{m}_{\sigma(1)}^{(i)} + \beta \vec{m}_{\sigma(2)}^{(i)}$, and by (1), $-\vec{m}_0^{(i+1)} = -\vec{m}_{\sigma(0)}^{(i)} + \alpha \vec{m}_{\sigma(1)}^{(i)} + \beta \vec{m}_{\sigma(2)}^{(i)}$.

We will prove below by contradiction that $\sigma(0) = 0$. Let us assume that $\sigma(0) = 1$ (the case where $\sigma(0) = 2$ is similar).

Let us consider $\det(-\vec{m}_1^{(i)}, -\vec{m}_2^{(i)}, -\vec{m}_0^{(i+1)})$. Replacing $-\vec{m}_0^{(i+1)}$ by $(-\vec{m}_1^{(i)} + \alpha \vec{m}_0^{(i)} + \beta \vec{m}_2^{(i)})$ in the previous determinant, we obtain the following identity:

$$\begin{aligned} \det(-\vec{m}_1^{(i)}, -\vec{m}_2^{(i)}, -\vec{m}_0^{(i+1)}) \\ = \alpha \det(-\vec{m}_1^{(i)}, -\vec{m}_2^{(i)}, \vec{m}_0^{(i)}). \end{aligned}$$

If $\alpha = 0$, then $\det(-\vec{m}_1^{(i)}, -\vec{m}_2^{(i)}, -\vec{m}_0^{(i+1)}) = 0$, which is in contradiction with lemma 1-item 2.

Otherwise, since $\alpha > 0$, $\det(-\vec{m}_1^{(i)}, -\vec{m}_2^{(i)}, -\vec{m}_0^{(i+1)})$ and $\det(-\vec{m}_1^{(i)}, -\vec{m}_2^{(i)}, -\vec{m}_0^{(i)})$ must have opposite signs. By (1), it follows that the plane passing by $\vec{v}_1^{(i)}, \vec{q}, \vec{v}_2^{(i)}$ separates $\vec{v}_0^{(i)}$ from $\vec{v}_0^{(i+1)}$ (see fig. 4.a). In this case, the straight line passing by \vec{p} and \vec{q} can intersect both $\mathbf{T}^{(i)}$ and $\mathbf{T}^{(i+1)}$ only if it intersects segment $[\vec{v}_1^{(i)} \vec{v}_2^{(i)}]$. However, a ray that goes inside the convex hull of $\mathbf{T}^{(i)} \cup \mathbf{T}^{(i+1)}$ through edge $[\vec{v}_1^{(i)} \vec{v}_2^{(i)}]$ must exit the convex tetrahedron through a single point of a facet that is not $\mathbf{T}^{(i+1)}$ (see fig. 4.b), which raises a contradiction ($\mathbf{T}^{(i+1)}$ is not the upper facet intersected by such a ray in this case).

We conclude that $\sigma(0) \neq 1$ and similarly that $\sigma(0) \neq 2$, which implies that $\sigma(0) = 0$. \square

This characterization leads to extra properties: lemma 3 and corollary 2 are stronger versions of lemma 1, items 2 and 3, respectively.

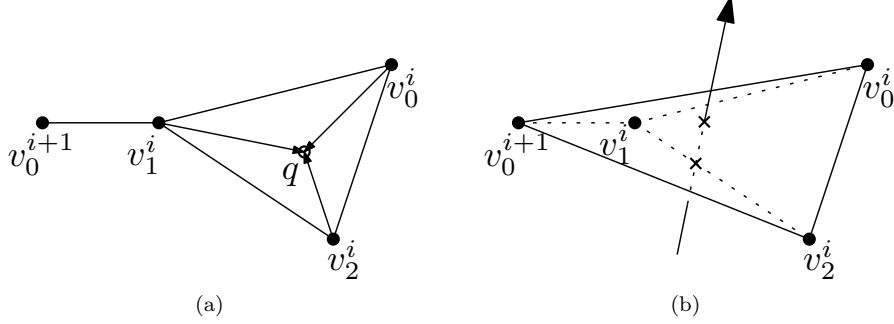


Figure 4: Illustration of lemma 2. In (a), the plane passing by $\vec{v}_1^{(i)}, \vec{q}, \vec{v}_2^{(i)}$ separates $\vec{v}_0^{(i)}$ from $\vec{v}_0^{(i+1)}$. In (b), a ray that goes inside the tetrahedron through edge $[\vec{v}_1^{(i)} \vec{v}_2^{(i)}]$ must exit through a point of a facet that is not $\mathbf{T}^{(i+1)}$.

Let $\mathbf{M}^{(i)}$ be the 3×3 matrix formed by the three vectors joining the vertices of the current triangle to \vec{q} . Otherwise said, it consists of the three row vectors $(\vec{m}_k^{(i)})_{k \in \{0,1,2\}}$. We prove below that $\mathbf{M}^{(i)}$ is unimodular, which is an important property to show that the algorithm returns a basis of upper leaning points at termination (see theorem 2 and corollary 5).

Lemma 3 $\forall i \in \{0, \dots, n\}, \det(\mathbf{M}^{(i)}) = 1$.

Proof. It is easily checked that $\det(\mathbf{M}^{(0)}) = 1$. We now prove that if $\det(\mathbf{M}^{(i)}) = 1$ for $\forall i \in \{0, \dots, n-1\}$, then $\det(\mathbf{M}^{(i+1)}) = 1$. As proven in lemma 1-item 5, only one vertex changes at each step, for instance $\vec{v}_k^{(i)}$. If we look at lemma 2, we have $\vec{v}_k^{(i+1)} = \vec{v}_k^{(i)} + \alpha \vec{m}_{k+1}^{(i)} + \beta \vec{m}_{k+2}^{(i)}$, which is equivalent to $\vec{m}_k^{(i+1)} = \vec{m}_k^{(i)} - \alpha \vec{m}_{k+1}^{(i)} - \beta \vec{m}_{k+2}^{(i)}$, for some non negative integers α, β such that α or β equals to 1, $\alpha + \beta \geq 1$. The other vertices are not modified so the remaining two rows of $\mathbf{M}^{(i+1)}$ are not modified. We get

$$\begin{aligned} \det(\mathbf{M}^{(i+1)}) &= \det(\vec{m}_k^{(i)} - \alpha \vec{m}_{k+1}^{(i)} - \beta \vec{m}_{k+2}^{(i)}, \vec{m}_{k+1}^{(i)}, \vec{m}_{k+2}^{(i)}) \\ &= \det(\vec{m}_k^{(i)}, \vec{m}_{k+1}^{(i)}, \vec{m}_{k+2}^{(i)}) \quad (\text{by linearity}) \\ &= \det(\mathbf{M}^{(i)}) = 1. \quad (\text{by induction hypothesis}) \end{aligned}$$

□

The height of each vector \vec{m}_k in the direction given by the estimated normal is equal to 1.

Corollary 1 $\forall i = 0, \dots, n, \mathbf{M}^{(i)} \cdot \hat{\mathbf{N}}(\mathbf{T}^{(i)}) = \mathbf{1}$.

Proof. $\mathbf{M}^{(i)} \cdot \hat{\mathbf{N}}(\mathbf{T}^{(i)}) = \mathbf{1}$ because $\forall k, (\vec{d}_0^{(i)} \times \vec{d}_1^{(i)}) \cdot \vec{m}_k^{(i)} = ((\vec{m}_0^{(i)} - \vec{m}_1^{(i)}) \times (\vec{m}_1^{(i)} - \vec{m}_2^{(i)})) \cdot \vec{m}_k^{(i)} = (\vec{m}_{k+1}^{(i)} \times \vec{m}_{k+2}^{(i)}) \cdot \vec{m}_k^{(i)} = \det(\mathbf{M}^{(i)})$, which is equal to 1 by lemma 3. □

The height of \vec{q} in the direction given by the estimated normal is equal to 1.

Corollary 2 $\forall i = 0, \dots, n, \vec{q} \cdot \hat{\mathbf{N}}(\mathbf{T}^{(i)}) = 1$.

Proof. Since $\forall i = 0, \dots, n, \vec{q} = \vec{v}_0^{(i)} + \vec{m}_0^{(i)}$, we compute on one hand $\vec{v}_0^{(i)} \cdot \hat{\mathbf{N}}(\mathbf{T}^{(i)})$, which is equal to 0 by definition, and on the other hand $\vec{m}_0^{(i)} \cdot \hat{\mathbf{N}}(\mathbf{T}^{(i)})$, which is equal to 1 by corollary 1. \square

To end, the following lemma leads to a strong geometrical property : the straight line passing by \vec{p} and \vec{q} intersects the *interior* of every triangle.

Lemma 4 $\forall i \in \{0, \dots, n\}, \forall k, (\vec{m}_k^{(i)} \times \vec{m}_{k+1}^{(i)}) \cdot \mathbf{s} > 0$.

Proof.

By definition, $\forall k, (\vec{m}_k^{(0)} \times \vec{m}_{k+1}^{(0)}) \cdot \mathbf{s} = 1$. Let us now assume that $\forall k, (\vec{m}_k^{(i)} \times \vec{m}_{k+1}^{(i)}) \cdot \mathbf{s} > 0$ and let us prove that $\forall k, (\vec{m}_k^{(i+1)} \times \vec{m}_{k+1}^{(i+1)}) \cdot \mathbf{s} > 0$. Let k^* be the index of the vertex of $\mathbf{T}^{(i+1)}$, that is not a vertex of $\mathbf{T}^{(i)}$. By lemma 2, we have $\vec{m}_{k^*}^{(i+1)} = \vec{m}_{k^*}^{(i)} - \alpha \vec{m}_{k^*+1}^{(i)} - \beta \vec{m}_{k^*+2}^{(i)}$, for some non negative integers α, β such that α or β equals to 1, $\alpha + \beta \geq 1$, while $\vec{m}_l^{(i+1)} = \vec{m}_l^{(i)}$ for $l \in \{0, 1, 2\} \setminus k^*$.

We must check two expressions involving $\vec{m}_{k^*}^{(i+1)}$ (the remaining one does not change).

The first one is:

$$\begin{aligned} (\vec{m}_{k^*}^{(i+1)} \times \vec{m}_{k^*+1}^{(i+1)}) \cdot \vec{s} &= (\vec{m}_{k^*}^{(i)} \times \vec{m}_{k^*+1}^{(i)}) \cdot \vec{s} \\ &\quad - \alpha (\vec{m}_{k^*+1}^{(i)} \times \vec{m}_{k^*+1}^{(i)}) \cdot \vec{s} - \beta (\vec{m}_{k^*+2}^{(i)} \times \vec{m}_{k^*+1}^{(i)}) \cdot \vec{s}. \end{aligned}$$

We conclude that $(\vec{m}_{k^*}^{(i+1)} \times \vec{m}_{k^*+1}^{(i+1)}) \cdot \vec{s} > 0$ because

- $(\vec{m}_{k^*}^{(i)} \times \vec{m}_{k^*+1}^{(i)}) \cdot \vec{s} > 0$ by induction hypothesis
- $-\alpha (\vec{m}_{k^*+1}^{(i)} \times \vec{m}_{k^*+1}^{(i)}) \cdot \vec{s} = 0$
- $-\beta (\vec{m}_{k^*+2}^{(i)} \times \vec{m}_{k^*+1}^{(i)}) \cdot \vec{s} = \beta (\vec{m}_{k^*+1}^{(i)} \times \vec{m}_{k^*+2}^{(i)}) \cdot \vec{s} \geq 0$ by induction hypothesis (and $\beta \geq 0$).

The second expression is similar. \square

By (1), lemma 4 is equivalent to:

$$\forall i \in \{0, \dots, n\}, \forall k, ((\vec{q} - \vec{m}_k^{(i)}) \times (\vec{q} - \vec{m}_{k+1}^{(i)})) \cdot (\vec{q} - \vec{p}) > 0,$$

which means that \vec{p} is strictly in the first octant of the frame $(\vec{q}, -\vec{m}_0^{(i)}, -\vec{m}_1^{(i)}, -\vec{m}_2^{(i)})$. This implies that the straight line passing by \vec{p} and \vec{q} intersects the interior of the triangle whose vertices are $(\vec{q} - \vec{m}_k^{(i)})_{k \in \{0, 1, 2\}}$, i.e. $(\vec{v}_k^{(i)})_{k \in \{0, 1, 2\}}$.

This result guarantees that there is no ambiguity in the computation of $\mathbf{T}^{(i+1)}$ in line 4 of algorithm 1, since the straight line passing by \vec{p} and \vec{q} never crosses an edge of $\text{conv}(\mathbf{T}^{(i)} \cup \mathbf{T}^{(i+1)})$, but only the interior of $\mathbf{T}^{(i)}$ and $\mathbf{T}^{(i+1)}$.

3.2 Termination

In the following proofs, we compare the position of the points along direction \mathbf{N} . For the sake of simplicity, we use the bar notation $\bar{\cdot}$ above any vector \vec{x} to denote its height relative to \mathbf{N} . Otherwise said, $\bar{\vec{x}} := \vec{x} \cdot \mathbf{N}$. Even if \mathbf{N} is not known, $\bar{\vec{q}} \geq \omega$ by definition and for all $\vec{x} \in \mathbf{P}$, $0 \leq \bar{\vec{x}} < \omega$. By (1) and lemma 1-item 1 we straightforwardly get the following lemma:

Lemma 5 $\forall i \in \{0, \dots, n-1\}, \forall k, \bar{\vec{m}}_k^{(i)} > 0$.

As a consequence, any operation strictly increases the height of the updated vertex.

Lemma 6 $\forall i \in \{0, \dots, n-1\}$, let k^* be the index of the updated vertex such that $\vec{v}_{k^*}^{(i+1)} \neq \vec{v}_{k^*}^{(i)}$. Then, $\bar{\vec{v}}_{k^*}^{(i+1)} > \bar{\vec{v}}_{k^*}^{(i)}$. A corollary is $\bar{\vec{m}}_{k^*}^{(i+1)} < \bar{\vec{m}}_{k^*}^{(i)}$.

Proof.

By lemma 2, we have $\forall i \in \{0, \dots, n-1\}$, $\vec{v}_{k^*}^{(i+1)} = \vec{v}_{k^*}^{(i)} + \alpha \vec{m}_{k^*+1}^{(i)} + \beta \vec{m}_{k^*+2}^{(i)}$, with two non negative integers α, β such that $\alpha = 1$ or $\beta = 1$, $\alpha + \beta \geq 1$.

Since $\forall k, \bar{\vec{m}}_k^{(i)} > 0$ by lemma 5 and $\alpha, \beta \geq 0$, but α and β are not both equal to 0, we clearly have $\bar{\vec{v}}_{k^*}^{(i+1)} > \bar{\vec{v}}_{k^*}^{(i)}$ and, by (1), $\bar{\vec{m}}_{k^*}^{(i)} < \bar{\vec{m}}_{k^*}^{(i+1)}$. \square

The termination theorem follows:

Theorem 1 The number of steps in algorithm 1 is bounded from above by $\omega - 3$.

Proof. The result comes from the fact that the sequence $(\sum_k \bar{\vec{m}}_k^{(i)})_{i=0, \dots, n}$ is a strictly decreasing sequence of integers between ω and 3 because:

- $\forall k, \vec{m}_k^{(0)} = \vec{e}_{k+2}$ and $\sum_k \bar{\vec{m}}_k^{(0)} = \omega$.
- by lemma 5,

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

- by lemma 6,

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

\square

Remark that this bound is tight since it is reached when running the algorithm on a plane with normal $\mathbf{N}(1, 1, r)$.

3.3 Correctness

We show that our two plane-probing algorithms extract the correct normal of the input digital plane. Let us begin with a small technical lemma:

Lemma 7 *For any permutation σ over $\{0, 1, 2\}$, $\forall i \in \{0, \dots, n\}$, if there is a point \vec{x} of ray $\mathcal{R}_\sigma^{(i)}$ that is not in \mathbf{P} , then no point further than \vec{x} on the ray is in \mathbf{P} .*

Proof. For two non negative integers λ and λ' such that $\lambda < \lambda'$, let $\vec{x} := \vec{q} - \vec{m}_{\sigma(0)}^{(i)} + \vec{m}_{\sigma(1)}^{(i)} + \lambda \vec{m}_{\sigma(2)}^{(i)}$ and $\vec{y} := \vec{q} - \vec{m}_{\sigma(0)}^{(i)} + \vec{m}_{\sigma(1)}^{(i)} + \lambda' \vec{m}_{\sigma(2)}^{(i)}$ be two points of $\mathcal{R}_\sigma^{(i)}$ (see equation 3). If $\vec{x} \notin \mathbf{P}$, then either $\bar{x} < 0$ or $\bar{x} \geq \omega$. However, it cannot be the former because $\vec{x} = \vec{v}_{\sigma(0)}^{(i)} + \vec{m}_{\sigma(1)}^{(i)} + \lambda \vec{m}_{\sigma(2)}^{(i)}$ and

- $\bar{v}_{\sigma(0)}^{(i)} > 0$ by lemma 1-item 1,
- $\bar{m}_{\sigma(1)}^{(i)}, \bar{m}_{\sigma(2)}^{(i)} > 0$ by lemma 5,
- λ is assumed to be non negative.

We can now bound from below the height of $\vec{y} = \vec{x} + (\lambda' - \lambda) \vec{m}_{\sigma(2)}^{(i)}$. Since $\bar{x} \geq \omega$, $\bar{m}_{\sigma(2)}^{(i)} > 0$ (by lemma 5) and $(\lambda' - \lambda) > 0$, we have $\bar{y} \geq \omega$, which implies that $\vec{y} \notin \mathbf{P}$. \square

Corollary 3 $\forall i \in \{0, \dots, n\}$, $(\mathcal{N}_H^{(i)} \cap \mathbf{P}) = \emptyset \Rightarrow (\mathcal{N}_R^{(i)} \cap \mathbf{P}) = \emptyset$.

Proof. Due to the neighborhood definitions (2) and (4), $\forall i \in \{0, \dots, n\}$, any point \vec{y} in $\mathcal{N}_R^{(i)}$ but not in $\mathcal{N}_H^{(i)}$ is located in a ray whose starting point \vec{x} is in $\mathcal{N}_H^{(i)}$ and $\vec{y} = \vec{x} + \lambda \vec{m}_k^{(i)}$ for some index $k \in \{0, 1, 2\}$ and non negative integer λ .

If $(\mathcal{N}_H^{(i)} \cap \mathbf{P}) = \emptyset$, then $\vec{x} \notin \mathbf{P}$ and the result follows by lemma 7. \square

This corollary implies in particular that at the last step we can focus on the H -neighborhood. Since we focus below on the last step n , we omit the exponent (n) in the proofs to improve their readability.

We now give the correctness result when the starting point \vec{p} is a lower leaning point, i.e. $\vec{p} = 0$.

Theorem 2 *If \vec{p} is a lower leaning point (i.e. $\vec{p} = 0$ and thus $\vec{q} = \omega$), the vertices of the last triangle are upper leaning points, i.e. $\forall k, \bar{v}_k^{(n)} = \omega - 1$.*

Proof. The first step of the proof is to show that the vertices of the last triangle are all at the same height, i.e. $\bar{m}_0 = \bar{m}_1 = \bar{m}_2$. If not, then there exists $k \in \{0, 1, 2\}$ such that $\bar{d}_k \neq 0$. In this case, either (i) $\bar{d}_k < 0$ or (ii) $\bar{d}_k > 0$. Since $\vec{q} = \omega$ and $|\bar{d}_k| < \omega$, either (i) $\vec{q} + \vec{d}_k \in \mathbf{P}$ or (ii) $\vec{q} - \vec{d}_k \in \mathbf{P}$. This implies that $\mathcal{N} \cap \mathbf{P} \neq \emptyset$, which is a contradiction because $\mathcal{N} \cap \mathbf{P} = \emptyset$ at the last step

(see algorithm 1, l. 2). As a consequence, $\forall k, \vec{d}_k = 0$ and $\forall k, \vec{m}_k = \gamma$, a strictly positive integer.

The second step of the proof is to show that $\gamma = 1$. Let us denote by $\mathbf{1}$ the vector $(1, 1, 1)^T$. We can write the last system as $\mathbf{M}\mathbf{N} = \gamma\mathbf{1}$. Since \mathbf{M} is invertible (because $\det(\mathbf{M}) = 1$ by lemma 3), $\mathbf{N} = \mathbf{M}^{-1}\gamma\mathbf{1}$ and as a consequence $\gamma = 1$ (because the components of \mathbf{N} are relatively prime and \mathbf{M}^{-1} is unimodular).

We conclude that $\forall k, \vec{m}_k = 1$ and, straightforwardly, $\vec{v}_k = \omega - 1$. \square

The following two corollaries are derived from lemma 3 and theorem 2.

Corollary 4 *If \vec{p} is a lower leaning point, the normal of the last triangle is equal to \mathbf{N} , i.e. $\hat{\mathbf{N}}(\mathbf{T}^{(n)}) = \mathbf{N}$.*

Proof. On one hand, $\mathbf{M}\hat{\mathbf{N}}(\mathbf{T}) = \mathbf{1}$ because $\forall k, (\vec{d}_0 \times \vec{d}_1) \cdot \vec{m}_k = ((\vec{m}_0 - \vec{m}_1) \times (\vec{m}_1 - \vec{m}_2)) \cdot \vec{m}_k = (\vec{m}_{k+1} \times \vec{m}_{k+2}) \cdot \vec{m}_k = \det(\mathbf{M})$, which is equal to 1 by lemma 3.

On the other hand, $\mathbf{M}\mathbf{N} = \mathbf{1}$ by theorem 2. Since \mathbf{M} is invertible, we have $\hat{\mathbf{N}}(\mathbf{T}) = \mathbf{N}$. \square

Corollary 5 *If \vec{p} is a lower leaning point, $(\vec{d}_0^{(n)}, \vec{d}_1^{(n)})$ is a basis of the lattice of upper leaning points $\{\vec{x} \in \mathbf{P} \mid \vec{x} = \omega - 1\}$.*

Proof. The unit parallelepiped in the lattice

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

does not contain any integer point because it is equivalent to \mathbb{Z}^3 ($\det(\mathbf{M}) = 1$ by lemma 3). It follows that the facet $\text{conv}(\mathbf{T})$ does not contain any integer point. Since the points of \mathbf{T} are upper leaning points by theorem 2, (\vec{d}_0, \vec{d}_1) is a basis of the lattice of upper leaning points. \square

We end the section by providing the worst-case time complexity of both algorithms in a computation model where the evaluation of the predicate “is \vec{x} in \mathbf{P} ” only requires a constant time:

Theorem 3 *If \vec{p} is a lower leaning point, the H -algorithm (resp. R -algorithm) returns three upper leaning points of \mathbf{P} in $O(\omega)$ (resp. $O(\omega \log \omega)$), where ω is the arithmetical thickness of the digital plane.*

Proof. The time complexity and correctness of the H -algorithm straightforwardly comes from theorem 1 and theorem 2 respectively, because each iteration runs in $O(1)$ ($\mathcal{N}_H^{(i)} \cap \mathbf{P}$ contains at most 6 points and algorithm 2 runs in linear time, see sec. 2.4).

However, the time complexity and correctness of the R -algorithm depend also on the time complexity and correctness of algorithm 4, run at each iteration to find the closest point in \mathbf{P} to the current triangle. Algorithm 4 uses an exponential march (lines 4-5) followed by a binary search (lines 7-14). The relevance of such an approach comes from lemma 5 and lemma 7 (along a ray,

points in \mathbf{P} are followed by points lying above \mathbf{P}) and corollary 2 (which implies, together with (1) and (3), that the function that maps a point \vec{x} in the ray sequence to the radius of the sphere passing by $\mathbf{T} \cup \{\vec{x}\}$ is convex and has a global minimum).

Let us focus now on the complexity of algorithm 4. After line 3, $\kappa = 0$ and $\vec{x} + \kappa\vec{y} \in \mathbf{P}$ (precondition, see algorithm 3, line 3). Therefore, $\vec{x} + \kappa\vec{y} \leq \omega$. Moreover, we know by lemma 5 and lemma 7, that there must be a greater value for κ , such that $\vec{x} + \kappa\vec{y} > \omega$. It is clear that this value is reached after at most $\log \omega + 2$ iterations in the exponential march (lines 4-5). After line 6, the size of the range $[\kappa, \lambda]$ is at most $\lceil \frac{3\omega}{2} \rceil$. In the binary search (lines 7-14), there are $O(\log \omega)$ iterations because each iteration shrinks the range to half its size until a size less than 4. The last line takes a constant time since algorithm 2 runs in linear time and the cardinality of the input point set is at most 4. The overall complexity of algorithm 4 is thus $O(\log \omega)$.

Since there are $O(\omega)$ iterations in algorithm 1 (theorem 1) and since algorithm 4 is used at each iteration at most 6 times in algorithm 3, we conclude that the overall complexity of the R -algorithm is $O(\omega \log \omega)$. \square

It is worth noting that in both cases, the time complexity corresponds to the number of calls to the predicate “is \vec{x} in \mathbf{P} ?”. This means that the time taken by a call to the predicate impacts directly the multiplicative constant in $O(\omega)$ (resp. $O(\omega \log \omega)$).

4 Experimental evaluation

In this section, we conduct an experimental evaluation of both H- and R-algorithms. Furthermore, we also compare these new algorithms to the plane-probing algorithm presented in [17] called *FindNormal*. We evaluate the number of steps, the number of calls to the predicate “is \vec{x} in \mathbf{P} ?” as a function of the norm of the normal vector of \mathbf{P} . We also check the ability of the algorithms to produce reduced lattice basis, and, in the case where the basis is not reduced, the number of lattice reduction operations necessary to transform it into a reduced basis.

The graphics on fig. 5 and fig. 6 shows that the three algorithms have quite a similar behavior. For the three algorithms, the number of steps is clearly sublinear on average. That being said, there are still some cases that reach the linear bound of theorem 1. Unlike the *FindNormal* algorithm, at each step the H- and R-algorithms select a point based on geometrical criteria. These criteria are stronger in the case of the R-algorithm which explains that in general, it terminates with less steps than the others.

Regarding the number of points tested or, equivalently, the number of calls to the predicate “is \vec{x} in \mathbf{P} ?”, the H-algorithm shows a better behavior on average. Of course, the systematic exploration of 6 rays using algorithm 2 generates extra calls to the predicates.

Corollary 5 states that the edge vectors of the last triangle form a basis of the lattice of upper leaning points to \mathbf{P} .

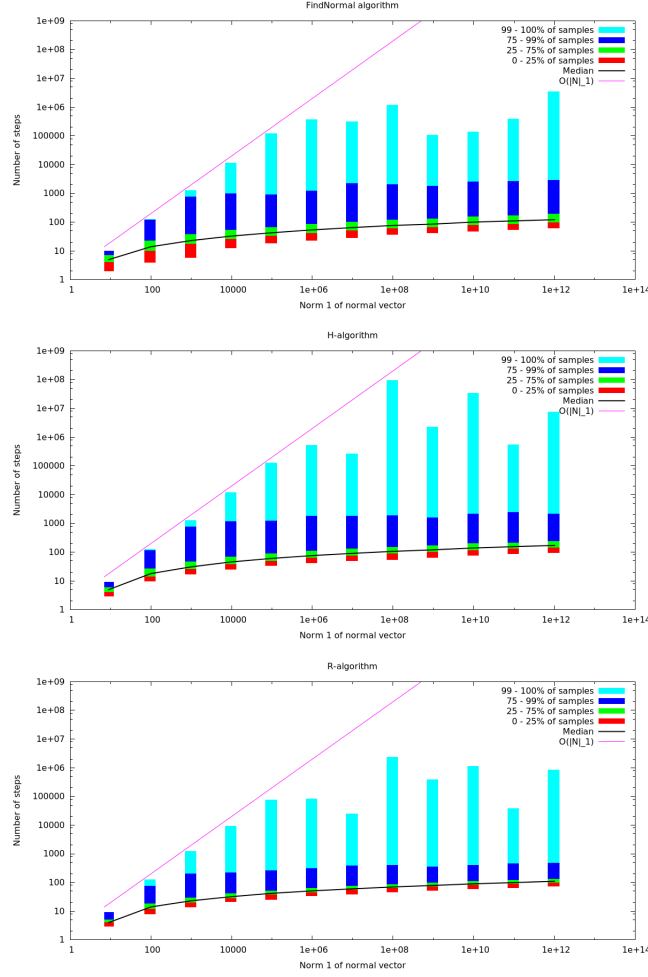


Figure 5: Number of iterative steps performed by the plane-probing algorithms as a function of the 1-norm of the normal vector \mathbf{N} . Top: the algorithm is *FindNormal* from [17], middle: H-algorithm and bottom: R-algorithm. All graphics are made using the same vectors picked randomly in such way that their 1-norm is located in the interval displayed by the width of each column.

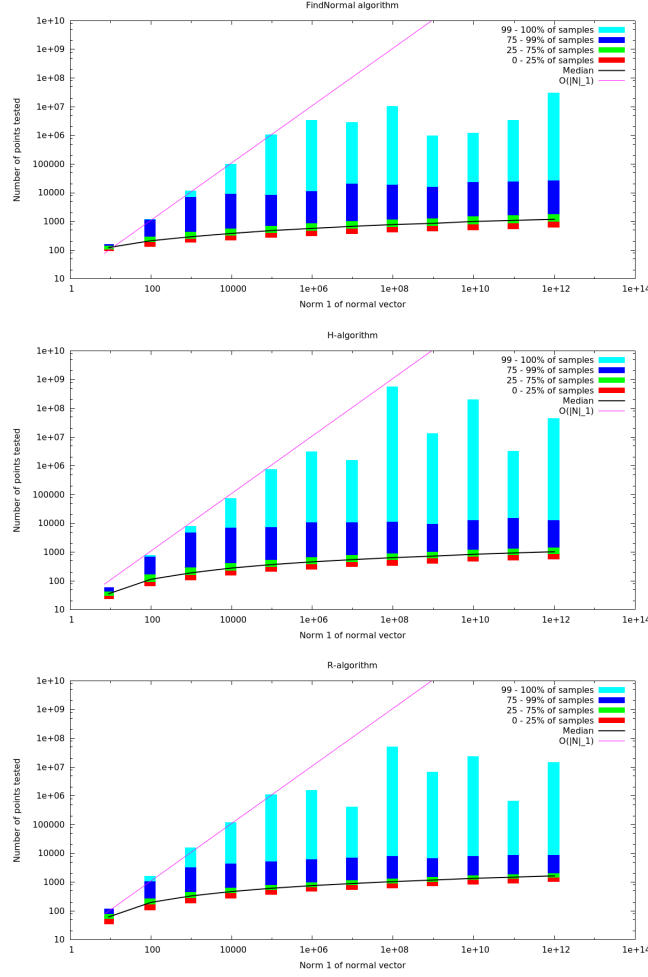


Figure 6: Number of calls to the predicates “is $\vec{x} \in \mathbf{P}$ ” versus the 1-norm of the normal vector \mathbf{N} . Top: the algorithm is *FindNormal* from [17], middle: H-algorithm and bottom: R-algorithm. All graphics are made using the same vectors picked randomly in such a way that their 1-norm is located in the interval displayed by the width of each column.

Let us recall that a basis (\vec{x}, \vec{y}) is reduced if and only if $\|\vec{x}\|, \|\vec{y}\| \leq \|\vec{x} - \vec{y}\| \leq \|\vec{x} + \vec{y}\|$. Given (\vec{x}, \vec{y}) a basis of a two dimensional lattice, there is an iterative algorithm to compute a reduced basis from (\vec{x}, \vec{y}) . This algorithm consists in replacing the longest vector among (\vec{x}, \vec{y}) by the shortest one among $\vec{x} + \vec{y}$ and $\vec{x} - \vec{y}$, if it is smaller. This operation is called a *reduction*.

In [17], a solution is proposed in order to generate reduced basis. This method can be summarized as *if at one step, a non-reduced basis is formed and that it may be corrected using a reduction, then do it*. In H- and R-algorithm, we return the two shortest vectors in $\{\vec{d}_k^{(n)}\}_{k \in \{0,1,2\}}$ as a basis. We do not perform any reduction because such a basis is almost always reduced. We ran all three algorithms (FindNormal, H-algorithm, R-algorithm) on all vectors ranging from (1,1,1) to (200,200,200). There are 6578833 vectors with relatively prime components in this range. Tab. 1 shows that less than 0.01% of basis computed by the H-algorithm were non-reduced. Regarding the R-algorithm, not only all bases computed in the range (1,1,1) to (200,200,200) were reduced but we have performed billions of tests on different normal vectors and we have never found a basis that was not reduced.

Algorithm	FindNormal	H-alg.	R-alg.
nb. non-reduced	6197115 (94.2%)	480 (0.007%)	0
avg. nb. reductions	6	1	0
max. nb. reductions	117	1	0

Table 1: Algorithms FindNormal, H-algorithm and R-algorithm were used on all 6578833 vectors ranging from (1, 1, 1) to (200, 200, 200) with relatively prime coordinates. The average number of reduction is computed only among non-reduced basis.

5 Digital surface analysis

In this section, we consider a set of voxels, Z , where voxels are seen as axis-aligned unit cubes whose vertices belong to \mathbb{Z}^3 . The *digital boundary*, $\text{Bd}Z$, is defined as the topological boundary of the union of the voxels of Z . Since a digital boundary looks locally like a digital plane, it is natural to run our plane probe algorithms at each reentrant corner of the digital boundary with predicate “is \vec{x} in $\text{Bd}Z$?” in order to estimate a local tangential facet to the volume Z (see fig. 7). This facet also defines naturally a local normal vector to the volume Z .

Since the predicate “is \vec{x} in $\text{Bd}Z$?” defines only locally a plane, our plane-probing algorithms must be slightly adapted to this new context. We discuss first the case where Z is the digitization of a convex set, and we address the problem of initializing the algorithms at bad reentrant corners, that is a corner that does not correspond to a lower leaning point. Then we present how

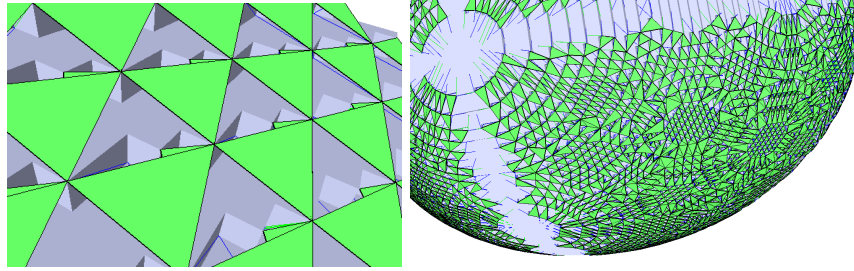


Figure 7: Our H -algorithm has been run at each reentrant corner of a digital plane (left) and an ellipse (right). The last triangle of each run is printed and each triangle is associated one-one to a pattern. On the left, the large triangles that regularly tile the digital plane are Bezout patterns; the small triangles hidden underneath are reduced patterns.

we can use the H -neighborhood to detect planarity defects. Last we explain how to make facets follow closely the local planar geometry along the digital surface. Presented experiments use H -algorithm, but would be identical with R -algorithm.

From now on, we call *pattern* any tuple $(\vec{q}, \vec{s}, \vec{m}_0, \vec{m}_1, \vec{m}_2)$, such that \vec{q} and \vec{s} define a reentrant corner of $\text{Bd}Z$ and $\vec{q} - \vec{m}_0, \vec{q} - \vec{m}_1, \vec{q} - \vec{m}_2$ is the output facet of the H -algorithm run at this reentrant corner. In proofs, since vector \vec{s} will be $(1, 1, 1)$ for all considered patterns, we will omit vector \vec{s} in the tuples.

5.1 Pattern on convex shapes

Let us assume that Z is a digitally convex shape, i.e. the digitization of its *convex hull* is Z itself. Let \mathcal{F} be a facet of the convex hull $\text{Conv}(\text{Bd}Z)$ of $\text{Bd}Z$, oriented so that its normal points outside. The *shift vector* $\vec{s}_{\mathcal{F}}$ of \mathcal{F} is the vector $(\pm 1, \pm 1, \pm 1)$ whose component signs match the sign of the normal vector to \mathcal{F} . We define the *set of boundary points of $\text{Bd}Z$ below \mathcal{F}* as

$$\text{Bd}Z(\mathcal{F}) := \{\vec{x} \in \text{Bd}Z \cap \mathbb{Z}^3 \text{ s.t. } [\vec{x}, \vec{x} + \vec{s}_{\mathcal{F}}] \cap \mathcal{F} \neq \emptyset\}.$$

By convexity, for any point $\vec{y} \in \text{Bd}Z(\mathcal{F})$, the point $\vec{y} + \vec{s}_{\mathcal{F}}$ does not belong to $\text{Bd}Z$.

It is clear that $\text{Bd}Z(\mathcal{F})$ is a piece of some digital plane \mathbf{P} : it suffices to define \mathbf{P} as the digital plane with normal vector identical to the normal vector of \mathcal{F} and with intercept such that the vertices of \mathcal{F} are all upper leaning points.

If a pattern $(\vec{q}, \vec{s}, \vec{m}_0, \vec{m}_1, \vec{m}_2)$ rooted at point $\vec{q} - \vec{s} \in \text{Bd}Z(\mathcal{F})$ and with $\vec{s} = \vec{s}_{\mathcal{F}}$ has its triangle aligned with \mathcal{F} , then $\vec{q} - \vec{s}$ is a lower leaning point of the digital plane carrying $\text{Bd}Z(\mathcal{F})$. In this case, \vec{q} is called a *Bezout point* and $(\vec{q}, \vec{s}, \vec{m}_0, \vec{m}_1, \vec{m}_2)$ is called a *Bezout pattern* of \mathcal{F} . Unfortunately not every facet of $\text{Conv}(\text{Bd}Z)$ has a Bezout pattern. This is illustrated on fig. 7. We can see that approximately half of the facets of the convex hull of $\text{Bd}Z$ are extracted.

The other half of the facets are not extracted because these facets have no Bezout point above them ! Their Bezout point is above the other half of the fundamental domain.

An interesting consequence is that we can use the first plane-probing algorithm of [17] to extract the complementary facets. This is illustrated on fig. 8. Almost all the geometry of the convex shape is captured. Missing parts are related to facets with normal vectors with a null component.

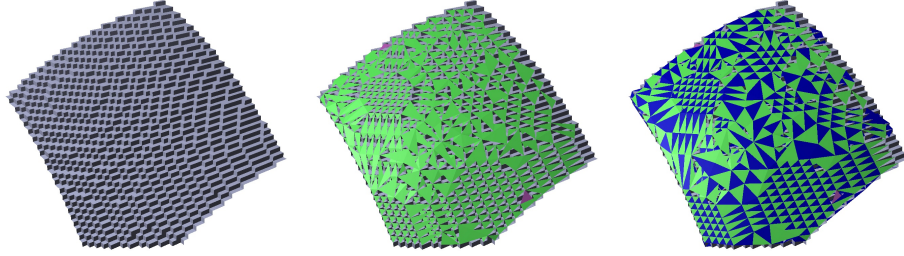


Figure 8: Left: a digital surface. Middle: triangular faces (in green) produced by the H-algorithm starting from each reentrant corners. Right: triangular faces (in blue) produced by the FindNormal algorithm are showed together with the previous ones.

5.2 Reduced patterns

If the pattern with corner point $\vec{p} - \vec{s} \in \text{Bd}Z(\mathcal{F})$ is not aligned with the facet \mathcal{F} then it is called a *reduced pattern* of \mathcal{F} . In this case the pattern is only approximately aligned with the plane. You can see some examples of reduced patterns on the digital plane of fig. 7, left. There are indeed several reentrant corners on such planes that are not located beneath a Bezout point. Starting from such corners make *H*-algorithm stops prematurely and outputs a smaller triangle that is only approximately aligned with the local tangent plane.

We propose to detect *a posteriori* the reduced patterns with algorithm 5.

Its principle is simple. Let us define $\text{Plane}(\vec{q}, \vec{m}_0, \vec{m}_1, \vec{m}_2)$ as the digital plane of upper leaning points $\vec{m}_0, \vec{m}_1, \vec{m}_2$ and Bezout point \vec{q} . Since there are Bezout patterns among all the input patterns, their vectors \vec{m}_k climb along the normal to the facet one layer at a time (their height is 1). Reduced patterns correspond to starting shifted point \vec{q}' that are not Bezout point. Their height (along the normal to some facet) is thus greater than the height of Bezout points, i.e. ω . So, in the first iterations, the algorithm finds the reduced patterns whose shifted point \vec{q}' has height $\omega + 1$. Those shifted points are put back in the queue, but with the Bezout vectors that climb one by one. So, in the following iterations, the reduced patterns whose shifted point \vec{q}' has height $\omega + 2$ are reached, and so on. This is illustrated on fig. 9.

Algorithm 5: Compute reduced patterns from the list of all patterns.

Input: L : the list of tuples $(\vec{q}, \vec{s}, \vec{m}_0, \vec{m}_1, \vec{m}_2)$, one for each output triangle, where \vec{q} is the shifted point, \vec{s} the shift and $\vec{q} - \vec{m}_k$ are the three output vertices.

Var: Q : a queue.

Output: M : the set of reduced patterns.

```

1 begin
2    $M \leftarrow \emptyset$ ;
3   Put all  $(\vec{q}, \vec{s}, \vec{m}_0, \vec{m}_1, \vec{m}_2)$  of  $L$  in queue  $Q$ ;
4   while not  $Q.empty()$  do
5      $(\vec{q}, \vec{s}, \vec{m}_0, \vec{m}_1, \vec{m}_2) \leftarrow Q.pop()$  ;
6     foreach  $k \in \{0, 1, 2\}$  do
7       if  $\exists$  a tuple  $(\vec{q}', \vec{s}', \vec{m}'_0, \vec{m}'_1, \vec{m}'_2) \in L$  such that  $\vec{s} = \vec{s}'$  and
8          $\vec{q} + \vec{m}_k = \vec{q}'$  then
9           if  $\vec{q}' - \vec{s}', \vec{q}' - \vec{m}'_0, \vec{q}' - \vec{m}'_1, \vec{q}' - \vec{m}'_2 \in \text{Plane}(\vec{q}, \vec{m}_0, \vec{m}_1, \vec{m}_2)$ 
10             then
11               // Mark  $\vec{q}'$  as a reduced pattern
12                $M \leftarrow M \cup \vec{q}'$  ;
13               // Push back  $\vec{q}'$  but with updated vectors. If
14                 it is already inside, update vectors only.
15                $Q.push( (\vec{q}', \vec{s}', \vec{m}_0, \vec{m}_1, \vec{m}_2) )$ ;
16     return  $M$ ;
17 end
```

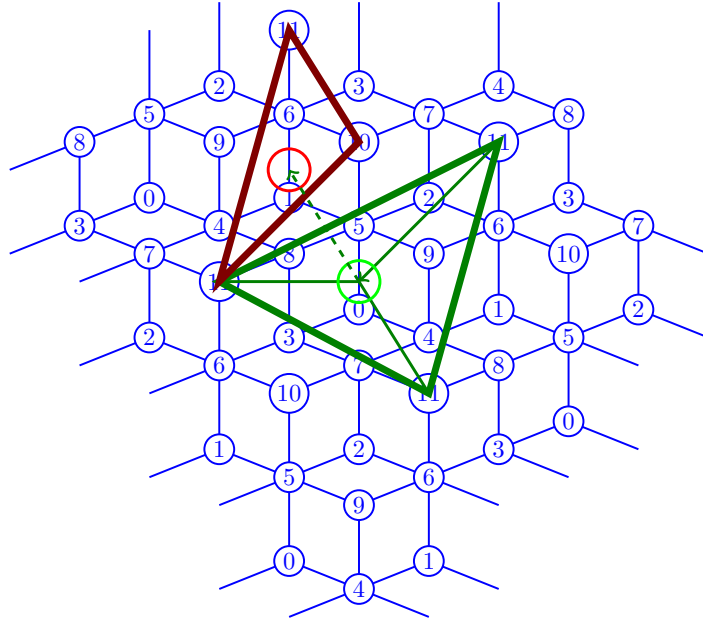


Figure 9: Locating reduced patterns with algorithm 5. A Bezout pattern of this digital plane is displayed in dark green, its shifted point \vec{q} is in green. The plane-probing algorithm was also run at the reentrant corner of height 1, its output is the reduced pattern displayed in dark red, with shifted point \vec{q}' in red. It is easily seen that $\vec{q}' = \vec{q} + \vec{m}_2$ and is thus located by algorithm 5.

We also remark that reduced patterns can be reached sooner (e.g. a shifted point of height $\omega + 3$ can be reached from a shifted point of height $\omega + 1$ with a vector \vec{m}_k climbing two by two). This is fine, we just want to mark reduced patterns. On a digital surface, it may also happen that one pattern can be reached from another pattern, but they do not correspond to the same digital plane. This is also tested in the algorithm: in order to be marked, the reduced pattern must be included in the digital plane carried by the Bezout pattern (line 10).

We now prove that algorithm 5 terminates on arbitrary BdZ. To make the explanation simpler, we suppose that the shift vector is equal to $\mathbf{1} := (1, 1, 1)$ for each pattern (see also line 8 of the algorithm, where equality of shift vectors is tested). The two following lemmas show how to build an order on patterns.

Lemma 8 *Given a pattern $\mathcal{P} = (\vec{q}, \vec{m}_0, \vec{m}_1, \vec{m}_2)$, denoting by \mathbf{M} the matrix composed of rows of \vec{m}_k , then the digital plane $\text{Plane}(\mathcal{P})$ has the following parameters (resp. normal \mathbf{N} , intercept μ and thickness ω):*

$$\mathbf{N} = \mathbf{M}^{-1}\mathbf{1}, \quad \mu = \vec{q} \cdot \mathbf{N} - \mathbf{1} \cdot \mathbf{N}, \quad \omega = \mathbf{1} \cdot \mathbf{N}.$$

To lighten the exposition, we will speak of the *normal*, *intercept* and *thickness* of the pattern \mathcal{P} , to refer to these characteristics defined on the digital plane $\text{Plane}(\mathcal{P})$. We say that a pattern $\mathcal{P}' = (\vec{q}', \vec{m}'_0, \vec{m}'_1, \vec{m}'_2)$ is *included* in the pattern $\mathcal{P} = (\vec{q}, \vec{m}_0, \vec{m}_1, \vec{m}_2)$, and we write $\mathcal{P}' \subset \mathcal{P}$, whenever every $\vec{q}' - \mathbf{1}, \vec{q}' - \vec{m}'_0, \vec{q}' - \vec{m}'_1, \vec{q}' - \vec{m}'_2$ belongs to $\text{Plane}(\vec{q}, \vec{m}_0, \vec{m}_1, \vec{m}_2)$. We have:

Lemma 9 *If $\mathcal{P}' \subset \mathcal{P}$ then it holds that $\mathbf{N}' \leq \mathbf{N}$, where \leq means less or equal for all components. A corollary is that $\omega' \leq \omega$.*

Proof. Since we have $(\vec{q}' - \mathbf{1}) \in \text{Plane}(\mathcal{P})$, we have $(\vec{q}' - \mathbf{1}) \cdot \mathbf{N} \geq \mu$ and it follows from lemma 8 that $\vec{q}' \cdot \mathbf{N} \geq \vec{q} \cdot \mathbf{N}$. Since we have $(\vec{q}' - \vec{m}'_k) \in \text{Plane}(\mathcal{P})$, we have $(\vec{q}' - \vec{m}'_k) \cdot \mathbf{N} \leq \mu + \omega - 1$. Using lemma 8, this is equivalent to $\vec{m}'_k \cdot \mathbf{N} \geq 1 + \vec{q}' \cdot \mathbf{N} - \vec{q} \cdot \mathbf{N}$. The first relation entails that $\mathbf{M}'\mathbf{N} \geq \mathbf{1}$ when writing it in matrix form. Since it holds that $\mathbf{M}'\mathbf{N}' = \mathbf{1}$ and \mathbf{M}' is invertible, the result follows.

Proposition 1 *Algorithm 5 terminates in less than $n\hat{\omega}$ iterations, where n is the number of patterns in L and $\hat{\omega}$ is the maximal thickness of the patterns in L .*

Proof.

First this algorithm manages patterns in the queue. Then it is clear that the number of patterns cannot increase, since patterns inserted back into the queue are associated to existing shifted points of L . Since only patterns with the same shift vectors are compared, we limit our reasoning to patterns with the same shift vector, here the positive orthant $\vec{s} = (1, 1, 1)$.

The key argument is that whenever some pattern is pushed back into the queue, its thickness strictly grows as well as its intercept. At some point, there won't be any pattern that can include another one, so the algorithm stops.

More precisely, looking at lines 8 and 10, we denote by $\mathcal{P} = (\vec{q}, \vec{m}_0, \vec{m}_1, \vec{m}_2)$ the pattern in the list L and by $\mathcal{P}' = (\vec{q}', \vec{m}'_0, \vec{m}'_1, \vec{m}'_2)$ the pattern popped from the queue. The pattern that is pushed back into the queue is denoted by \mathcal{P}'' . Using the above notations for thickness and intercept of digital planes associated to patterns (Lemma 8), we obtain straightforwardly that:

$$\mathbf{N}'' = \mathbf{N}, \quad \omega'' = \omega, \quad \mu'' = \mu + 1.$$


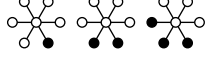
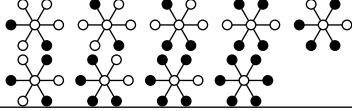
Furthermore, since a success of the test at line 10 means $\mathcal{P}' \subset \mathcal{P}$, we have $\omega' < \omega = \omega''$ (the “ \leq ” comes from Lemma 9, the strictness “ $<$ ” comes from $\vec{q}' = \vec{q} + \vec{m}_k$). We may also state that $\mathcal{P}'' \not\subset \mathcal{P}$ and $\mathcal{P} \not\subset \mathcal{P}''$ since their intercepts are different while their thickness are equal.

To sum up, the queue contains patterns with strictly increasing thicknesses. However, since the new inserted pattern \mathcal{P}'' has the same thickness as some already existing pattern, it is clear also that the thickness of all patterns is upperbounded by the thickness of the thickest pattern of L , i.e. $\hat{\omega}$. If we follow one pattern of L , it may thus be reassigned in Q at most $\hat{\omega} - 1$ times before having maximal thickness. At this moment, it cannot be included in any pattern of L , since the tests at lines 8 and 10 may be true only if the thickness of \mathcal{P}' is strictly lower than the thickness of a pattern of L . So this pattern is definitively popped out of the queue. It follows that the algorithm terminates in at most $n\hat{\omega}$ iterations. \square

In practice, as shown on tab. 2, this algorithm runs extremely fast and much fewer iterations than $n\hat{\omega}$ are needed. As expected, the algorithm is slower when analyzing shapes made of wide planar sides with important arithmetic thickness (e.g. fandisk 512³).

5.3 Detecting planarity defects

If the shape is not convex, the algorithm can be adapted to detect planarity defects. The idea is that, at a step i , $\mathcal{N}_H^{(i)} \cap \text{Bd}Z$ must contain at most three elements if it is locally a piece of plane. Moreover these elements must always be consecutive neighbors around \vec{q} in $\mathcal{N}_H^{(i)}$. We thus stop the algorithm whenever at least one of the two previous conditions fails, meaning that the surface is locally non convex. We sum up the possible cases in the following table, where the predicate values of the points of the H -neighborhood are symbolized with white circles for $\notin \text{Bd}Z$ and black circles for $\in \text{Bd}Z$. When our plane probe algorithm stops, it returns in addition to the output triangle the stopping criterion among “convex or planar” and “non convex”. These various stopping criteria are illustrated in fig. 10. Convex and planar zones contain green triangles, which are patterns which stops on a “convex and planar” H -neighborhood. Inflexion zones, saddle points, or places with at least one negative principal curvature contain mostly magenta triangles, meaning “non-convex” H -neighborhood, or yellow triangles, which refer to non-separating patterns.

H -neighborhood configurations	Stop	Local planarity
	yes	convex or planar
	no	(still probing)
	yes	non-convex

5.4 Following closely digital surface geometry

Our algorithm probes for points in a plane in a sparse way. This works well when identifying a true digital plane, but it may badly identify pieces of plane on digital surfaces. For instance the algorithm may jump over holes or cracks in the surface. Therefore, we have to check at each iteration that the current triangle fits closely the digital surface. We proceed as follows:

- the current triangle is denoted by $\mathbf{T} = (\vec{v}_0, \vec{v}_1, \vec{v}_2)$, the shifted point \vec{q} and the shift vector \vec{s} ;
- the corresponding digital plane is $D := \text{Plane}(\vec{q}, \vec{q} - \vec{v}_0, \vec{q} - \vec{v}_1, \vec{q} - \vec{v}_2)$;
- we compute the digital triangle under \mathbf{T} on D as

$$A := \{\vec{x} \in D, [\vec{x}, \vec{x} + \vec{s}] \cap \text{conv}(\mathbf{T}) \neq \emptyset\},$$

by a breadth-first traversal from \vec{q} on D ;

- we compute also the three digital segments B_k , $k \in \{0, 1, 2\}$, along the border of \mathbf{T} as the 6-connected path in D from \vec{v}_k to \vec{v}_{k+1} whose points are closest to the straight line segment $[\vec{v}_k, \vec{v}_{k+1}]$;
- then the current triangle is said *separating* iff $A \cup B_0 \cup B_1 \cup B_2 \subset \text{BdZ}$.

When running our plane-probing algorithm, we check at each step if the current triangle is separating. If not, the algorithm exits with stopping criterion “non-separating”. An illustration of such cases is given in fig. 10.

The sets B_k are important to connect points of A . Should we not consider them, the current triangle might have a needle shape that is separating while being far away from the surface (A is reduced to the three vertices of \mathbf{T}).

Some timings and statistics of our algorithms to analyze digital surface geometry are given in tab. 2.

6 Conclusion and perspectives

In this paper, we proposed two new algorithms that compute the parameters of a digital plane. In opposition to usual plane recognition algorithms, these

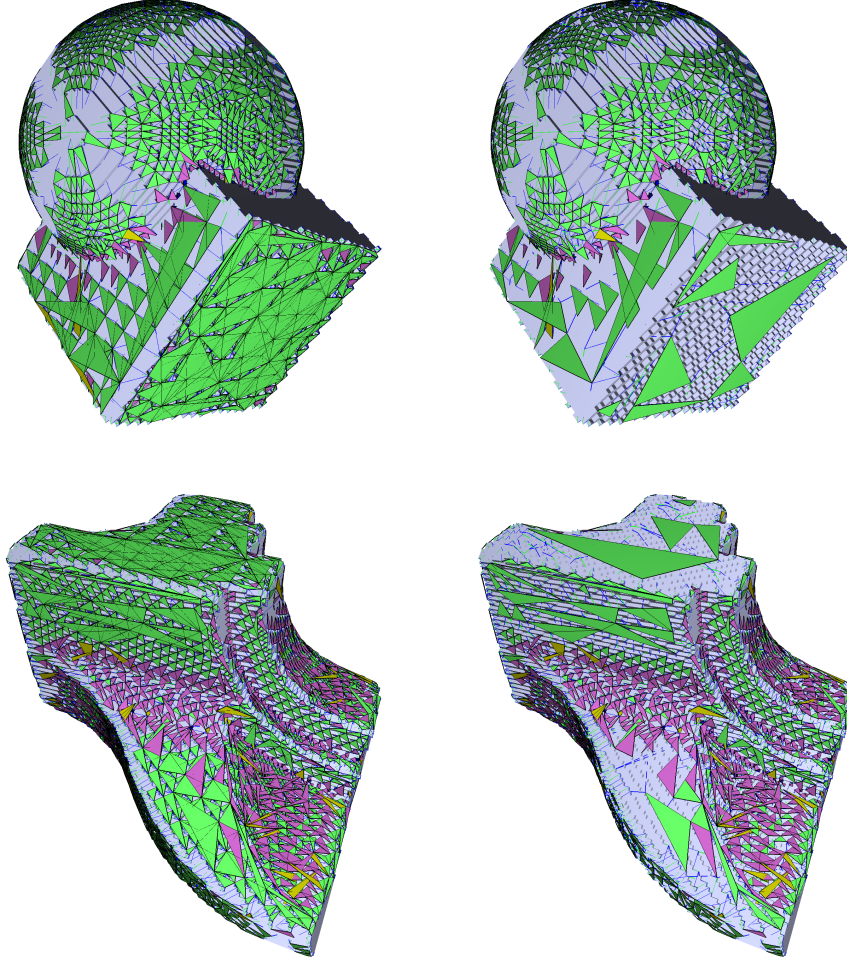


Figure 10: Illustration of H -algorithm run on every reentrant corner of digital surfaces “cube and sphere 128^3 ” (top row) and “fandisk 128^3 ” (bottom row). Extracted triangles are colored according to the stopping criterion: “convex or planar” is green, “non-convex” is magenta, “non-separating” is yellow. All patterns are displayed on left column. On right column, reduced patterns have been removed with algorithm 5.

Shape Z	#BdZ	Time (ms) H - algorithm	Time (ms) reduced	#patterns	#reduced
cube and sphere 128^3	34034	1513	30	7041	3266
fandisk 128^3	48220	2732	69	12106	6984
fandisk 256^3	186760	13954	791	45999	31948
fandisk 512^3	734658	71660	18201	178751	142293
octaflower 512^3	692738	47299	944	179905	98816

Table 2: Total timings of H -algorithm (algorithm 1) runned at each point of BdZ for several shapes. Note that this time includes the time for checking the separation of triangles. The time taken by algorithm 5 to remove reduced patterns is listed. The total number of extracted patterns and the number of patterns marked as reduced are listed.

algorithm greedily decide on-the-fly which points have to be considered like in [17]. We have called *plane-probing algorithms* that kind of plane recognition algorithms. Compared to [17], these algorithms are however simpler because they consist in iterating one geometrical operation. Furthermore the returned solution, which is described by a triangle parallel to the sought plane, always lies above the starting corner. Besides, the two shortest edge vectors of the triangle almost always form a reduced basis for the purely local H -algorithm. For the sometimes less local R -algorithm, it has always returned a reduced basis.

For the future, we would like to prove that the R -algorithm always returns a reduced basis. Moreover, we would like to find a variant of our algorithm in order to retrieve complement triangles whose Bezout point is not above the triangle. For sake of completeness, we are also interested in degenerate cases, where at least one component is null. We would also like to recognize a *piece* of digital planes, i.e. to extract the digital plane with minimal characteristics containing this piece, but with theoretical guarantees. The difficulty is that our plane-probing algorithms need few but specific points of the plane to fully recognize it. We feel that we still need a correct definition of what is a valid *piece* of digital plane in 3D. Although connectedness was a sufficient condition for digital segment in 2D, it is not enough in the 3D case and we are currently working on it.

After having achieved these goals, we would have a complete working tool for the analysis of digitally convex surfaces. General digital surfaces are even more complex to analyze. We have shown that our plane-probing algorithms are able to detect non-convex parts. To go further, it appears necessary to precisely associate to a given output triangle a subset of points of the digital surface. Then we will be able to analyze how these subsets of points overlap each other in order to delineate convex, concave and saddle zones. The segmentation of a digital surface into such zones is certainly an essential step in digital shape

analysis, which would greatly facilitate higher-level processing, like digital shape matching, indexing or recognition.

References

- [1] Berthé, V., Fernique, T.: Brun expansions of stepped surfaces. *Discrete Mathematics* **311**(7), 521–543 (2011)
- [2] Brimkov, V., Coeurjolly, D., Klette, R.: Digital planarity – a review. *Discrete Applied Mathematics* **155**(4), 468–495 (2007)
- [3] 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)
- [4] 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)
- [5] Chica, A., Williams, J., Andújar, C., Brunet, P., Navazo, I., Rossignac, J., Vinacua, A.: Pressing: Smooth isosurfaces with flats from binary grids. *Comput. Graph. Forum* **27**(1), 36–46 (2008).
- [6] Doerksen-Reiter, H., Debled-Rennesson, I.: Convex and concave parts of digital curves. In: R. Klette, R. Kozera, L. Noakes, J. Weickert (eds.) *Geometric Properties for Incomplete Data*, *Computational Imaging and Vision*, vol. 31, pp. 145–160. Springer (2006)
- [7] Fernique, T.: Generation and recognition of digital planes using multi-dimensional continued fractions. *Pattern Recognition* **42**(10), 2229–2238 (2009)
- [8] Feschet, F.: Canonical representations of discrete curves. *Pattern Analysis & Applications* **8**(1), 84–94 (2005)
- [9] Gérard, Y., Debled-Rennesson, I., Zimmermann, P.: An elementary digital plane recognition algorithm. *Discrete Applied Mathematics* **151**(1), 169–183 (2005)
- [10] Jamet, D., Toutant, J.L.: Minimal arithmetic thickness connecting discrete planes. *Discrete Appl. Math.* **157**(3), 500–509 (2009).
- [11] Kerautret, B., Lachaud, J.O.: Meaningful scales detection along digital contours for unsupervised local noise estimation. *IEEE Transaction on Pattern Analysis and Machine Intelligence* **43**, 2379–2392 (2012).
- [12] Kim, C.E., Stojmenović, I.: On the recognition of digital planes in three-dimensional space. *Pattern Recognition Letters* **12**(11), 665–669 (1991)

- [13] Klette, R., Rosenfeld, A.: Digital straightness – a review. *Discrete Applied Mathematics* **139**(1-3), 197–230 (2004)
- [14] 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)
- [15] Labbé, S., Reutenauer, C.: A d-dimensional extension of christoffel words. *Discrete and Computational Geometry* p. 26 p. (to appear). ArXiv:1404.4021
- [16] Lachaud, J.O., Provençal, X., Roussillon, T.: Computation of the normal vector to a digital plane by sampling significant points. In: N. Normand, J. Guédon, F. Aulic (eds.) *Proc. 19th IAPR Int. Conf. Discrete Geometry for Computer Imagery (DGCI'2016)*, Nantes, France, April 18-20, 2016., pp. 194–205. Springer International Publishing, Cham (2016).
- [17] Lachaud, J.O., Provençal, X., Roussillon, T.: An output-sensitive algorithm to compute the normal vector of a digital plane. *Theoretical Computer Science* **624**, 73–88 (2016)
- [18] Lachaud, J.O., Vialard, A., de Vieilleville, F.: Fast, accurate and convergent tangent estimation on digital contours. *Image and Vision Computing* **25**(10), 1572–1587 (2007)
- [19] Provot, L., Debled-Rennesson, I.: 3D noisy discrete objects: Segmentation and application to smoothing. *Pattern Recognition* **42**(8), 1626–1636 (2009)
- [20] Roussillon, T., Sivignon, I.: Faithful polygonal representation of the convex and concave parts of a digital curve. *Pattern Recognition* **44**(10-11), 2693–2700 (2011).
- [21] Veelaert, P.: Digital planarity of rectangular surface segments. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **16**(6), 647–652 (1994)
- [22] de Vieilleville, F., Lachaud, J.O., Feschet, F.: Maximal digital straight segments and convergence of discrete geometric estimators. *Journal of Mathematical Image and Vision* **27**(2), 471–502 (2007)
- [23] Zrour, R., Kenmochi, Y., Talbot, H., Buzer, L., Hamam, Y., Shimizu, I., Sugimoto, A.: Optimal consensus set for digital line and plane fitting. *International Journal of Imaging Systems and Technology* **21**(1), 45–57 (2011).