Digital Plane Recognition With Fewer Probes *

Tristan Roussillon[1] and Jacques-Olivier Lachaud[2]

[1] Université de Lyon, INSA Lyon, LIRIS, UMR CNRS 5205, F-69622, France
tristan.roussillon@liris.cnrs.fr
[2] Université Savoie Mont Blanc, LAMA, UMR CNRS 5127, F-73376, France
jacques-olivier.lachaud@univ-smb.fr

Abstract. We present a new plane-probing algorithm, i.e., an algorithm that computes the normal vector of a digital plane from a starting point and a predicate "Is a point $\boldsymbol{x}$ 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. We show that this algorithm returns the same vector as another plane-probing algorithm proposed in Lachaud et al. (J. Math. Imaging Vis., 59, 1, 23–39, 2017), but requires fewer probes. The theoretical upper bound is indeed lowered from $O(\omega \log \omega)$ to $O(\omega)$ calls to the predicate, where ω is the arithmetical thickness of the digital plane, and far fewer calls are experimentally observed on average. This reduction is made possible by a study that shows how to avoid computations that do not contribute to the final solution. In the context of digital surface analysis, this new algorithm is expected to be of great interest for normal estimation and shape reconstruction.

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

1 Introduction

Analyzing the geometry of digital surfaces is a challenging task, since the local geometry is very poor (only six possible normal directions). A classical approach is to estimate the geometry by observing a larger neighborhood, whose size is given rather arbitrarily by the user. However, this approach 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. Purely digital methods have thus emerged and try to perform digital surface analysis without any external parameter.

A natural approach, e.g., [3], consists in computing a set of digital plane segments (DPSs) that locally fit the digital surface. This strategy has also been used for surface area estimation [7] or reversible polyhedrization [12]. However, finding how to scan the digital surface to efficiently recognize DPSs whose size

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

and shape reveal the local geometry is difficult. There are numerous algorithms for the recognition of DPSs (to quote a few: [13, 4, 11, 1, 6, 2, 5, 14]). All these 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. But 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.

Therefore, recently, another category of recognition algorithms have been developped [8–10]. These algorithms, called *plane-probing* algorithms in [10], decide on-the-fly where to probe next the digital surface while growing a triangular facet, which is tangent and separating by construction. The growth direction is given by both arithmetic and geometric properties.

Two of these algorithms proposed in [10], called H- and R-algorithm, are local in the sense that the returned triangular facet is guaranteed to stay around the starting point (but this is not the case for the one proposed in [8]). The R-algorithm is even more local in the sense that the final triangular facet has experimentally always acute angles and is less elongated.

In this paper, we present a new plane-probing algorithm that returns the same triangular facet (and normal vector) as the R-algorithm, but requires fewer probes. For comparison, the R-algorithm requires $O(\omega \log \omega)$ calls to the predicate "Is $\boldsymbol{x}$ in the digital plane?" and the exhibited worst-case example implies $\Theta(\omega)$ calls. We present here an improvement that achieves the tight bound of $O(\omega)$ calls. Furthermore, far fewer calls are observed in practice.

In sec. 2, we give an overview of our new algorithm. In sec. 3 and sec. 4, we go into details and show how to avoid computations that do not contribute to the final solution. Some experimental results are discussed in sec. 5.

2 A New Plane-Probing Algorithm

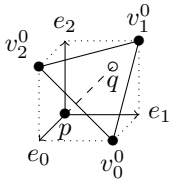
We keep definitions and notations introduced in [10]. We wish to extract the parameters of a standard digital plane $\mathbf{P}$, defined as the set

$$\mathbf{P} = \{\boldsymbol{x} \in \mathbb{Z}^3 \mid 0 \leq \boldsymbol{x} \cdot \mathbf{N} < \omega\},$$

where $\mathbf{N} \in \mathbb{N}^3$ is the *normal vector* whose components (a, b, c) are such that $0 < a \leq b \leq c$, $\gcd(a, b, c) = 1$ and $\omega := (1, 1, 1) \cdot \mathbf{N}$ is the *thickness*. Our approach can be straightforwardly extended to digital plane with arbitrary intercept and with a normal vector in any orthant (see [10]).

As in [9, 10], we propose an algorithm that, given a predicate $P(\boldsymbol{x})$:="Is $\boldsymbol{x} \in \mathbf{P}$?" and a starting point $\boldsymbol{p}$ at the base of a reentrant corner of $\mathbf{P}$, computes the *normal vector* of a piece of digital plane surrounding $\boldsymbol{p}$. Moreover, if $\boldsymbol{p}$ is a *lower leaning point*, i.e., $\boldsymbol{p} \cdot \mathbf{N} = 0$, the algorithm extracts the exact normal $\mathbf{N}$ of $\mathbf{P}$ and a *basis of* $\mathbf{P}$, that is a pair of vectors that forms a basis of the 2D lattice $\{\boldsymbol{x} \in \mathbb{Z}^3 \mid \boldsymbol{x} \cdot \mathbf{N} = 0\}$. This basis is returned as three *upper leaning points* of $\mathbf{P}$, i.e., points $\boldsymbol{x}$ such that $\boldsymbol{x} \cdot \mathbf{N} = \omega - 1$.

Initialization Given a starting point $\boldsymbol{p} \in \mathbf{P}$, the algorithm places an initial triplet of points $\mathbf{T}^{(0)} := (\boldsymbol{v}_k^{(0)})_{k \in \{0,1,2\}}$ such that $\forall k, \boldsymbol{v}_k^{(0)} := \boldsymbol{p} + \boldsymbol{e}_k + \boldsymbol{e}_{k+1}$, where $(\boldsymbol{e}_0, \boldsymbol{e}_1, \boldsymbol{e}_2)$ is the canonical basis of $\mathbb{R}^3$, and "$\forall k$" stands for "$\forall k \in \mathbb{Z}/3\mathbb{Z}$" for clarity. The algorithm requires $\mathbf{T}^{(0)} \subset \mathbf{P}$ which is the case when $\boldsymbol{p}$ is the base of a reentrant corner (see inset figure). We also denote by $\boldsymbol{q}$ the point $\boldsymbol{p} + (1,1,1)$, which is not in $\mathbf{P}$.



Evolution At each step $i \in \mathbb{Z}_{\geq 0}$, the triangle $\mathbf{T}^{(i)}$ represents the current approximation of the plane $\mathbf{P}$. The algorithm updates one vertex of $\mathbf{T}^{(i)}$ per iteration, while $\boldsymbol{q}$ does not move and stays above the triangle. The new vertex $\boldsymbol{x}^\star$ in $\mathbf{T}^{(i+1)} \setminus \mathbf{T}^{(i)}$, is a point both in $\mathbf{P}$ and in a specific neighborhood $\mathcal{N}^{(i)}$ yet to be defined, such that the circumsphere of $\mathbf{T}^{(i)} \cup \{\boldsymbol{x}^\star\}$ does not include any point $\boldsymbol{x} \in (\mathcal{N}^{(i)} \cap \mathbf{P})$ in its interior. Denoting by Σ the set of permutations over $\{0,1,2\}$, we introduce the following notations (illustrated in fig. 1) in order to define $\mathcal{N}^{(i)}$:

$$\forall k,\ \boldsymbol{m}_k^{(i)} := \boldsymbol{q} - \boldsymbol{v}_k^{(i)} \tag{1}$$

$$\forall \sigma \in \Sigma, \boldsymbol{y}_\sigma^{(i)} := \boldsymbol{v}_{\sigma(0)}^{(i)} + \boldsymbol{m}_{\sigma(1)}^{(i)} \tag{2}$$

$$\forall \sigma \in \Sigma, \forall \lambda \in \mathbb{Z}_{\geq 0}, \mathcal{R}_\sigma^{(i)}[\lambda] := \boldsymbol{y}_\sigma^{(i)} + \lambda \boldsymbol{m}_{\sigma(2)}^{(i)} \tag{3}$$

$$\forall \sigma \in \Sigma, \mathcal{R}_\sigma^{(i)} := (\mathcal{R}_\sigma^{(i)}[\lambda])_{\lambda \in \mathbb{Z}_{\geq 0}} \tag{4}$$

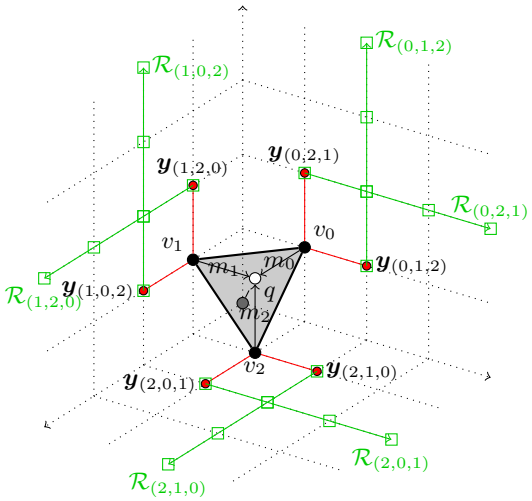
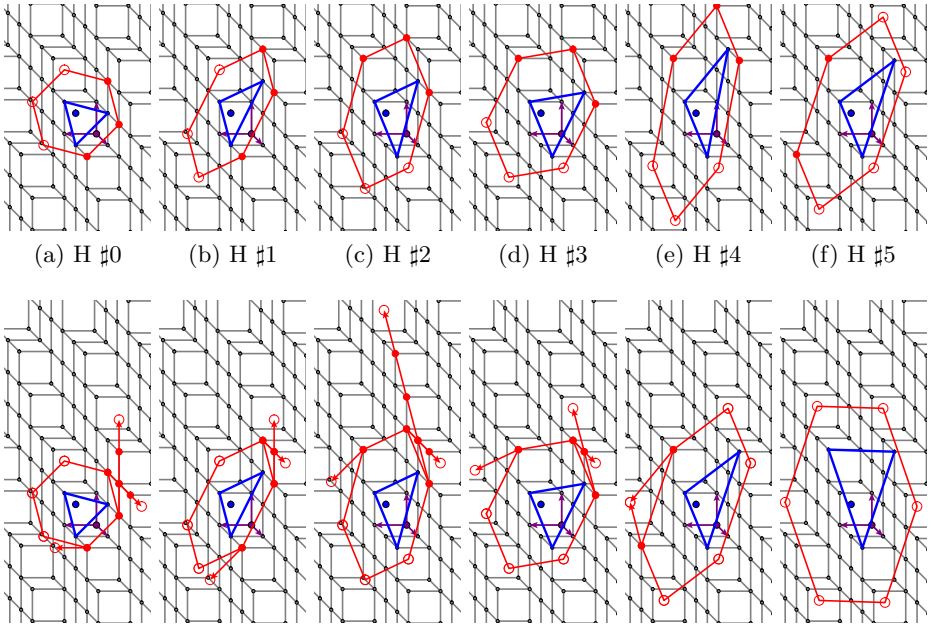


Fig. 1: The triangle $conv(\mathbf{T}^{(i)})$ is depicted in grey. The H-neighborhood $\mathcal{N}_H^{(i)} := (\boldsymbol{y}_\sigma^{(i)})_{\sigma \in \Sigma}$ is depicted with red disks, whereas rays $(\mathcal{R}_\sigma^{(i)})_{\sigma \in \Sigma}$ are depicted with green squares and include the H-neighborhood (iteration number dropped).

At step i, the six points $\mathcal{N}_H^{(i)} := (\boldsymbol{y}_\sigma^{(i)})_{\sigma\in\Sigma}$ forms an hexagon, called *H-neighborhood*, while the six rays $\mathcal{N}_R^{(i)} := (\mathcal{R}_\sigma^{(i)})_{\sigma\in\Sigma}$ forms the *R-neighborhood*. In [10], the H-(resp. R-)algorithm uses the H-(resp. R-)neighborhood (see fig. 2 for an illustration of the running of the two algorithms). While the update procedure of the H-algorithm is trivial and constant-time, since $\boldsymbol{x}^\star$ is one of the six points of the H-neighborhood, the update procedure of the R-algorithm computes a candidate point *for each ray* having a non-empty intersection with $\mathbf{P}$, before choosing one of them as $\boldsymbol{x}^\star$. This strategy is not optimal since a candidate point belonging to a ray may not be chosen finally.



(a) H ♯0 (b) H ♯1 (c) H ♯2 (d) H ♯3 (e) H ♯4 (f) H ♯5

(g) R ♯0 (h) R ♯1 (i) R ♯2 (j) R ♯3 (k) R ♯4 (l) R ♯5, H ♯6

Fig. 2: The H and R algorithms are applied on a digital plane of normal vector $\mathbf{N}(9,2,3)$. Images (a) to (f) and (l) show the six iterations of the H-algorithm, whereas images (g) to (l) show the five iterations of the R-algorithm. In each image, the current triangle is depicted in blue, whereas the neighborhood is depicted in red – disks (resp. circles) for points lying inside (resp. outside) the digital plane. Note that, in this example, the output of the two algorithms only differ at step ♯4.

On the contrary, in this paper, we propose to only probe the H-neighborhood and *one* ray to determine triangle $\mathbf{T}^{(i+1)}$. In addition, we may probe fewer points onto the ray. We call this new algorithm R^1 since it tests at most one ray

per iteration. It can be coarsely described as repetitive calls to the function UpdateTriangle (see algorithm 1).

Algorithm 1: R^1-algorithm: it extracts a triplet of upper leaning points by probing the H-neighborhood and one ray.

Input: The predicate $P(\boldsymbol{x})$ "Is $\boldsymbol{x} \in \mathbf{P}$?", the exterior point $\boldsymbol{q}$ and triangle $\mathbf{T}^{(0)}$
Output: A triangle $\mathbf{T}^{(n)}$
$i \leftarrow 0$;
while $\mathcal{N}_H^{(i)} \cap \mathbf{P} \neq \emptyset$ **do**
  $\mathbf{T}^{(i+1)} \leftarrow$ UpdateTriangle$(P, \mathbf{T}^{(i)}, \boldsymbol{q})$;
  $i \leftarrow i + 1$;
return $\mathbf{T}^{(i)}$

Table 1: Function UpdateTriangle$(P, \mathbf{T}, \boldsymbol{q})$, with $\mathbf{T} = (\boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2)$. After all cases, the triangle $\mathbf{T}$ is returned.

$\lvert\mathcal{N}_H \cap \mathbf{P}\rvert$	$\mathcal{N}_H \cap \mathbf{P}$	Output
0	()	algorithm termination
1	$(\boldsymbol{y}_\sigma)$	$\boldsymbol{v}_{\sigma(0)} \leftarrow \boldsymbol{v}_{\sigma(0)} + \boldsymbol{m}_{\sigma(1)}$
2	$(\boldsymbol{y}_\sigma, \boldsymbol{y}_{\sigma'})$ with $\sigma(0) \neq \sigma'(0), \sigma(1) = \sigma'(1)$	**if** $\boldsymbol{y}_{\sigma'} \leq_{\mathbf{T}} \boldsymbol{y}_\sigma$ **then** $\sigma \leftarrow \sigma'$; $\boldsymbol{v}_{\sigma(0)} \leftarrow \boldsymbol{v}_{\sigma(0)} + \boldsymbol{m}_{\sigma(1)}$
2	$(\boldsymbol{y}_\sigma, \boldsymbol{y}_{\sigma'})$ with $\sigma(0) = \sigma'(0), \sigma(1) \neq \sigma'(1)$	**if** $\boldsymbol{m}_{\sigma(1)} \geq \boldsymbol{m}_{\sigma'(1)}$ **then** $(\tau, \tau') \leftarrow (\sigma, \sigma')$; **else** $(\tau, \tau') \leftarrow (\sigma', \sigma)$; $(\pi, \alpha) \leftarrow$ ClosestAmongPointAndRay$(P, \mathbf{T}, \boldsymbol{q}, \tau', \tau)$ $\boldsymbol{v}_{\pi(0)} \leftarrow \boldsymbol{v}_{\pi(0)} + \boldsymbol{m}_{\pi(1)} + \alpha \boldsymbol{m}_{\pi(2)}$
3	$(\boldsymbol{y}_\sigma, \boldsymbol{y}_{\sigma'}, \boldsymbol{y}_{\sigma''})$ with $\sigma(0) = \sigma'(0)$, $\sigma(0) \neq \sigma''(0)$, $\sigma(1) \neq \sigma'(1)$	**if** $\boldsymbol{m}_{\sigma(1)} \geq \boldsymbol{m}_{\sigma'(1)}$ **then** $(\tau, \tau') \leftarrow (\sigma, \sigma')$; **else** $(\tau, \tau') \leftarrow (\sigma', \sigma)$; **if** $\boldsymbol{y}_{\sigma''} \leq_{\mathbf{T}} \boldsymbol{y}_{\tau'}$ **then** $\tau' \leftarrow \sigma''$; $(\pi, \alpha) \leftarrow$ ClosestAmongPointAndRay$(P, \mathbf{T}, \boldsymbol{q}, \tau', \tau)$ $\boldsymbol{v}_{\pi(0)} \leftarrow \boldsymbol{v}_{\pi(0)} + \boldsymbol{m}_{\pi(1)} + \alpha \boldsymbol{m}_{\pi(2)}$
else		Error, $\mathbf{P}$ is not a plane.

Function UpdateTriangle is detailed in tab. 1 and performs a case analysis on the cardinal and the composition of the H-neighborhood.

Order induced by circumspheres to $\mathbf{T}$*:* Let I^+ be the half-plane delimited by $\mathbf{T}$ and containing $\boldsymbol{q}$. We claim that the ball $C(\mathbf{T}, \boldsymbol{z})$ circumscribing $\mathbf{T}$ and some $\boldsymbol{z} \in I^+$, induces a total pre-order on I^+ through the inclusion relation. Indeed, if $\boldsymbol{z}' \in I^+$, whenever $\boldsymbol{z}' \in C(\mathbf{T}, \boldsymbol{z})$, then $(C(T, \boldsymbol{z}') \cap I^+) \subset (C(T, \boldsymbol{z}) \cap I^+)$. Clearly, this relation is reflexive, transitive and connex. We shall say that $\boldsymbol{z}'$ is *closer* to $\mathbf{T}$ than $\boldsymbol{z}$ and we write $\boldsymbol{z}' \leq_{\mathbf{T}} \boldsymbol{z}$. We can use this relation because the R-neighborhood is included in I^+.

Correctness: To prove that R^1-algorithm extracts the normal vector of $\mathbf{P}$ when starting from a lower leaning point $\boldsymbol{p}$, it is enough to show that UpdateTriangle outputs in all cases the same triangle $\mathbf{T}$ as in the R-algorithm, i.e., it updates a vertex of $\mathbf{T}$ by a point $\boldsymbol{x}^\star$ in $\mathcal{N}_R \cap \mathbf{P}$ such that $\forall \boldsymbol{x} \in (\mathcal{N}_R \cap \mathbf{P}), \boldsymbol{x}^\star \leq_{\mathbf{T}} \boldsymbol{x}$. In the next sections, we show that it does so, by appropriate calls to function ClosestAmongPointAndRay, which is detailed in algorithm 4.

3 Local Configuration

Informally, function UpdateTriangle updates $\mathbf{T}$ as follows:

$|\mathcal{N}_H \cap \mathbf{P}| = 0$**:** the algorithm stops,
$|\mathcal{N}_H \cap \mathbf{P}| = 1$**:** the algorithm updates the associated vertex of $\mathbf{T}$ with this point,
$|\mathcal{N}_H \cap \mathbf{P}| = 2$**, the two points are linked to distinct vertices of T:** the algorithm picks the closest according to $\leq_{\mathbf{T}}$ and updates the associated vertex,
$|\mathcal{N}_H \cap \mathbf{P}| = 2$**, the two points are linked to the same vertex of T:** the algorithm determines which ray originating from these points may possibly contain the target point, then picks the closest according to $\leq_{\mathbf{T}}$ and updates the associated vertex,
$|\mathcal{N}_H \cap \mathbf{P}| = 3$**, two of the points are linked to the same vertex of T:** the algorithm determines which ray originating from these points may possibly contain the target point, then picks the closest according to $\leq_{\mathbf{T}}$ and updates the associated vertex.

Even if there are exactly six points in $\mathcal{N}_H$ by definition, there are fewer points in $\mathcal{N}_H \cap \mathbf{P}$:

Lemma 1. *There are no more than three points in $\mathcal{N}_H \cap \mathbf{P}$. In addition, they are consecutive when counter-clockwise ordered around $\boldsymbol{q}$.*

For the proof, we introduce the edge vectors defined as $\boldsymbol{d}_k := \boldsymbol{v}_{k+1} - \boldsymbol{v}_k$ for all $k \in \{0, 1, 2\}$.

Proof. Since $\mathcal{N}_H = \{\boldsymbol{q} \pm \boldsymbol{d}_k\}_{k \in \{0,1,2\}}$ and $\boldsymbol{q} \notin \mathbf{P}$, it is clear that $\boldsymbol{q} - \boldsymbol{d}_k$ and $\boldsymbol{q} + \boldsymbol{d}_k$ cannot be both in $\mathbf{P}$ by linearity, which means that there are no more than three points in $\mathcal{N}_H \cap \mathbf{P}$.

For the second part, it is enough to see that for any k, $\boldsymbol{q} - \boldsymbol{d}_{k+1}$ is necessarily in $\mathbf{P}$ by linearity if both $\boldsymbol{q} + \boldsymbol{d}_k$ and $\boldsymbol{q} + \boldsymbol{d}_{k+2}$ are in $\mathbf{P}$. □

Lemma 1 shows that all possible cardinalities are taken into account in function UpdateTriangle. Now, we explain why we can only probe the H-neighborhood and one ray at each step. To do so, we use the following lemma:

Lemma 2 ([10], Lemma 7). *For any permutation $\sigma \in \Sigma$, if there is a point $\boldsymbol{x}$ of ray $\mathcal{R}_\sigma$ that is not in $\mathbf{P}$, then no point further than $\boldsymbol{x}$ on the ray is in $\mathbf{P}$.*

Due to Lemma 2, if the starting point of a ray is not in $\mathbf{P}$, then no other ray point has to be considered, which explains the first two lines of tab. 1. The following result explains the third line:

Lemma 3. *For any k, let $\mathcal{R}_\sigma$ and $\mathcal{R}_{\sigma'}$ be two distinct rays such that $\sigma(0) = \sigma'(0)$. If the starting point of one ray is not in $\mathbf{P}$, then only the starting point of the other ray may be in $\mathbf{P}$ among all the points of $\mathcal{R}_\sigma \cup \mathcal{R}_{\sigma'}$.*

Proof. The key point is to notice that the two rays cross at a point $\boldsymbol{v}_{\sigma(0)} + \boldsymbol{m}_{\sigma(1)} + \boldsymbol{m}_{\sigma(2)}$ (because $\sigma'(1) = \sigma(2)$ and $\sigma'(2) = \sigma(1)$, see fig. 1). Due to lemma 2, we conclude that if the starting point of a ray, let us say $\mathcal{R}_\sigma$, is not in $\mathbf{P}$, then neither the crossing point, nor any further point in $\mathcal{R}_{\sigma'}$ (and obviously in $\mathcal{R}_\sigma$) is in $\mathbf{P}$. □

In other words, only one point, instead of two rays, has to be considered in this case. If only two such points belong to $\mathcal{N}_H \cap \mathbf{P}$, it is enough to determine which one is the closest according to $\leq_{\mathbf{T}}$, hence the third line of tab. 1. The following result provides the rationale for the fourth and fifth lines:

Lemma 4. *Let $\mathcal{R}_\sigma$ and $\mathcal{R}_{\sigma'}$ be two distinct rays such that $\sigma(0) = \sigma'(0)$ and such that $\boldsymbol{y}_\sigma, \boldsymbol{y}_{\sigma'} \in \mathbf{P}$. If $\boldsymbol{m}_{\sigma(1)} \geq \boldsymbol{m}_{\sigma'(1)}$ (resp. $\boldsymbol{m}_{\sigma(1)} \leq \boldsymbol{m}_{\sigma'(1)}$), a closest point $\boldsymbol{x}^\star$ according to $\leq_{\mathbf{T}}$, among all the points of $\mathcal{R}_\sigma \cup \mathcal{R}_{\sigma'}$, belongs to $\mathcal{R}_\sigma \cup \boldsymbol{y}_{\sigma'}$ (resp. $\mathcal{R}_{\sigma'} \cup \boldsymbol{y}_\sigma$).*

The proof of lemma 4 requires this result in the case of an acute angle:

Lemma 5. *Let $\mathcal{R}_\sigma$ and $\mathcal{R}_{\sigma'}$ be two distinct rays such that $\sigma(0) = \sigma'(0)$ and such that $\boldsymbol{y}_\sigma, \boldsymbol{y}_{\sigma'} \in \mathbf{P}$. If $\boldsymbol{m}_{\sigma(1)} \cdot \boldsymbol{m}_{\sigma'(1)} \geq 0$, a closest point $\boldsymbol{x}^\star$ according to $\leq_{\mathbf{T}}$, among all the points of $\mathcal{R}_\sigma \cup \mathcal{R}_{\sigma'}$, is either $\boldsymbol{y}_\sigma$ or $\boldsymbol{y}_{\sigma'}$.*

Proof. Let us focus on $\mathcal{R}_\sigma$ because the same is true for $\mathcal{R}_{\sigma'}$. To show that either $\boldsymbol{y}_\sigma$ or $\boldsymbol{y}_{\sigma'}$ is closer than any point of $\mathcal{R}_\sigma$, let us consider the parallelogram whose first diagonal links $\boldsymbol{v}_{\sigma(0)}$ to a ray point $\mathcal{R}_\sigma[\lambda]$, with $\lambda \in \mathbb{Z}_{>0}$, and the second one links $\boldsymbol{v}_{\sigma(0)} + \boldsymbol{m}_{\sigma(1)} = \boldsymbol{y}_\sigma$ to $\mathcal{R}_\sigma[\lambda] - \boldsymbol{m}_{\sigma(1)} = \boldsymbol{v}_{\sigma(0)} + \lambda \boldsymbol{m}_{\sigma(2)}$. We prove below that the first diagonal is always strictly longer than the second one, which means that a sphere passing by $\boldsymbol{v}_{\sigma(0)}$ and $\mathcal{R}_\sigma[\lambda]$ contains either $\boldsymbol{y}_\sigma$ or $\boldsymbol{v}_{\sigma(0)} + \lambda \boldsymbol{m}_{\sigma(2)}$, i.e., $\boldsymbol{y}_\sigma \leq_{\mathbf{T}} \mathcal{R}_\sigma[\lambda]$ or $\boldsymbol{y}_{\sigma'} = \boldsymbol{v}_{\sigma(0)} + \boldsymbol{m}_{\sigma(2)} \leq_{\mathbf{T}} \boldsymbol{v}_{\sigma(0)} + \lambda \boldsymbol{m}_{\sigma(2)} \leq_{\mathbf{T}} \mathcal{R}_\sigma[\lambda]$.

Indeed, we have:

$$(\boldsymbol{m}_{\sigma(1)} + \lambda \boldsymbol{m}_{\sigma(2)})^2 - (-\boldsymbol{m}_{\sigma(1)} + \lambda \boldsymbol{m}_{\sigma(2)})^2 = 4\lambda(\boldsymbol{m}_{\sigma(1)} \cdot \boldsymbol{m}_{\sigma(2)}) > 0,$$

because $\lambda > 0$ and $\boldsymbol{m}_{\sigma(1)} \cdot \boldsymbol{m}_{\sigma(2)} = \boldsymbol{m}_{\sigma(1)} \cdot \boldsymbol{m}_{\sigma'(1)} \geq 0$ by hypothesis.

Proof (lemma 4). The proof is divided into two cases. If $\boldsymbol{m}_{\sigma(1)} \cdot \boldsymbol{m}_{\sigma'(1)} \geq 0$, $\boldsymbol{x}^\star$ is the starting point of $\mathcal{R}_\sigma$ or $\mathcal{R}_{\sigma'}$ by lemma 5.

Otherwise, $\boldsymbol{m}_{\sigma(1)} \cdot \boldsymbol{m}_{\sigma'(1)} < 0$ and we have:

$$-\max\{\boldsymbol{m}_{\sigma(1)}^2, \boldsymbol{m}_{\sigma'(1)}^2\} < \boldsymbol{m}_{\sigma(1)} \cdot \boldsymbol{m}_{\sigma'(1)} < 0,$$

which is equivalent to

$$0 < \boldsymbol{m}_{\sigma(1)} \cdot \boldsymbol{m}_{\sigma'(1)} + \max\{\boldsymbol{m}_{\sigma(1)}^2, \boldsymbol{m}_{\sigma'(1)}^2\} < \max\{\boldsymbol{m}_{\sigma(1)}^2, \boldsymbol{m}_{\sigma'(1)}^2\}.$$

Let us assume w.l.o.g. that $\boldsymbol{m}_{\sigma(1)} \geq \boldsymbol{m}_{\sigma'(1)}$ so that $\boldsymbol{m}_{\sigma(1)} \cdot (\boldsymbol{m}_{\sigma(1)} + \boldsymbol{m}_{\sigma'(1)}) \geq 0$. Let us consider the following linear transform: $\widetilde{\boldsymbol{m}}_{\sigma(1)} := \boldsymbol{m}_{\sigma(1)}$ and $\widetilde{\boldsymbol{m}}_{\sigma'(1)} := \boldsymbol{m}_{\sigma(1)} + \boldsymbol{m}_{\sigma'(1)}$. Since $\widetilde{\boldsymbol{m}}_{\sigma(1)} \cdot \widetilde{\boldsymbol{m}}_{\sigma'(1)} \geq 0$, due to lemma 5, $\boldsymbol{v}_{\sigma(0)} + \widetilde{\boldsymbol{m}}_{\sigma(1)}$ or $\boldsymbol{v}_{\sigma(0)} + \widetilde{\boldsymbol{m}}_{\sigma'(1)}$ is closer according to $\leq_{\mathbf{T}}$, than any point $\boldsymbol{v}_{\sigma(0)} + \widetilde{\boldsymbol{m}}_{\sigma'(1)} + \lambda \widetilde{\boldsymbol{m}}_{\sigma(1)}$, with $\lambda \in \mathbb{Z}_{>0}$. In other words, a closest point $\boldsymbol{x}^\star$ cannot belong to $\{\mathcal{R}_{\sigma'}[\lambda+1], \lambda \in \mathbb{Z}_{>0}\} \subset \mathcal{R}_{\sigma'} \backslash \boldsymbol{y}_{\sigma'}$, because either $\boldsymbol{v}_{\sigma(0)} + \widetilde{\boldsymbol{m}}_{\sigma(1)} = \mathcal{R}_\sigma[0]$ or $\boldsymbol{v}_{\sigma(0)} + \widetilde{\boldsymbol{m}}_{\sigma'(1)} = \mathcal{R}_\sigma[1]$, are closer according to $\leq_{\mathbf{T}}$. □

According to lemma 4, it is enough to call the function ClosestAmongPointAndRay on an appropriate point and on an appropriate ray (lines 4 and 5 of tab. 1). The body of the function is given in the next section.

4 One point and one ray $\mathcal{R}_\sigma$

It is well known that the implicit equation of the sphere can be written as a determinant (e.g., see MathWorld). More precisely, the algebraic distance of $\boldsymbol{x}'$ to the circumsphere of $\mathbf{T} \cup \{\boldsymbol{x}\}$ is given by the following 5×5 matrix determinant:

$$\delta_{\mathbf{T}}(\boldsymbol{x}, \boldsymbol{x}') := \begin{vmatrix} \boldsymbol{v}_0 & \boldsymbol{v}_1 & \boldsymbol{v}_2 & \boldsymbol{x} & \boldsymbol{x}' \\ {\boldsymbol{v}_0}^2 & {\boldsymbol{v}_1}^2 & {\boldsymbol{v}_2}^2 & \boldsymbol{x}^2 & {\boldsymbol{x}'}^2 \\ 1 & 1 & 1 & 1 & 1 \end{vmatrix}.$$

Note that $\delta_{\mathbf{T}}(\boldsymbol{x}, \boldsymbol{x}') \leq 0 \Leftrightarrow \boldsymbol{x}' \leq_{\mathbf{T}} \boldsymbol{x}$ means that $\boldsymbol{x}'$ is inside or on the circumsphere of $\mathbf{T} \cup \{\boldsymbol{x}\}$.

From now on, we assume w.l.o.g. that $\sigma(0) = 0$, $\sigma(1) = 1$, $\sigma(2) = 2$ so that we can take $\boldsymbol{v}_0$ as the origin. In order to shorten notations, we set $\boldsymbol{y} := \boldsymbol{x} - \boldsymbol{v}_0$, $\boldsymbol{y}' := \boldsymbol{x}' - \boldsymbol{v}_0$ and using $\boldsymbol{d}_k = \boldsymbol{v}_{k+1} - \boldsymbol{v}_k = \boldsymbol{m}_k - \boldsymbol{m}_{k+1}$ for all k, we have:

$$\delta^0_{\mathbf{T}}(\boldsymbol{y}, \boldsymbol{y}') := \delta_{\mathbf{T}}(\boldsymbol{v}_0 + \boldsymbol{y}, \boldsymbol{v}_0 + \boldsymbol{y}') = \begin{vmatrix} \boldsymbol{d}_0 & -\boldsymbol{d}_2 & \boldsymbol{y} & \boldsymbol{y}' \\ {\boldsymbol{d}_0}^2 & {\boldsymbol{d}_2}^2 & \boldsymbol{y}^2 & {\boldsymbol{y}'}^2 \end{vmatrix}. \tag{5}$$

Let us denote by $[\boldsymbol{z}, \boldsymbol{z}', \boldsymbol{z}'']$ the 3×3 matrix composed of columns $\boldsymbol{z}, \boldsymbol{z}', \boldsymbol{z}''$. We give below a formula for $\delta^0_{\mathbf{T}}(\boldsymbol{z}, \boldsymbol{z}' + \alpha \boldsymbol{z}'')$ for any $\boldsymbol{z}, \boldsymbol{z}', \boldsymbol{z}'' \in \mathbb{R}^3$ using the cofactor expansion of the determinant (5):

$$\begin{aligned} \delta^0_{\mathbf{T}}(\boldsymbol{z}, \boldsymbol{z}' + \alpha \boldsymbol{z}'') = & -{\boldsymbol{d}_0}^2 \det[-\boldsymbol{d}_2, \boldsymbol{z}, \boldsymbol{z}' + \alpha \boldsymbol{z}''] + {\boldsymbol{d}_2}^2 \det[\boldsymbol{d}_0, \boldsymbol{z}, \boldsymbol{z}' + \alpha \boldsymbol{z}''] \\ & - \boldsymbol{z}^2 \det[\boldsymbol{d}_0, -\boldsymbol{d}_2, \boldsymbol{z}' + \alpha \boldsymbol{z}''] + (\boldsymbol{z}' + \alpha \boldsymbol{z}'')^2 \det[\boldsymbol{d}_0, -\boldsymbol{d}_2, \boldsymbol{z}]. \end{aligned}$$

Since the determinant is multilinear, the following identity can be obtained:

$$\begin{aligned} \delta^0_{\mathbf{T}}(\boldsymbol{z}, \boldsymbol{z}' + \alpha \boldsymbol{z}'') = \; & \delta^0_{\mathbf{T}}(\boldsymbol{z}, \boldsymbol{z}') + \alpha \delta^0_{\mathbf{T}}(\boldsymbol{z}, \boldsymbol{z}'') \\ & + \Big(\alpha^2 ({\boldsymbol{z}''}^2) + \alpha \big(-{\boldsymbol{z}''}^2 + 2 \boldsymbol{z}' \cdot \boldsymbol{z}'' \big) \Big) \det[\boldsymbol{d}_0, -\boldsymbol{d}_2, \boldsymbol{z}]. \end{aligned} \tag{6}$$

We now use (6) in order to find an implementation of algorithm 2 and algorithm 3 in constant time, which are used in algorithm 4.

SphereRayIntersection In order to implement algorithm 2, we consider the circumsphere of $\mathbf{T} \cup \{\boldsymbol{v}_0 + \boldsymbol{z}\}$, where $\boldsymbol{z} \in \{\boldsymbol{m}_2, \boldsymbol{d}_0 + \boldsymbol{m}_2, -\boldsymbol{d}_2 + \boldsymbol{m}_1\}$, and its intersection with the ray points $\boldsymbol{v}_0 + \boldsymbol{m}_1 + \lambda \boldsymbol{m}_2$, $\lambda \in \mathbb{Z}_{\geq 0}$.

First, $\det[\boldsymbol{d}_0, -\boldsymbol{d}_2, \boldsymbol{z}] = 1$ for all $\boldsymbol{z} \in \{\boldsymbol{m}_2, \boldsymbol{d}_0 + \boldsymbol{m}_2, -\boldsymbol{d}_2 + \boldsymbol{m}_1\}$, because the determinant is multilinear and $\det[\boldsymbol{m}_0, \boldsymbol{m}_1, \boldsymbol{m}_2] = 1$ [10, Lemma 3]. Consequently, replacing $\boldsymbol{z}'$ with $\boldsymbol{m}_1$, $\boldsymbol{z}''$ with $\boldsymbol{m}_2$ and α with λ in (6), we get:

$$\delta^0_{\mathbf{T}}(\boldsymbol{z}, \boldsymbol{m}_1 + \lambda \boldsymbol{m}_2) = \lambda^2(\boldsymbol{m}_2^2) + \lambda\big(2\boldsymbol{z} \cdot \boldsymbol{m}_1 - \boldsymbol{m}_2^2 + \delta^0_{\mathbf{T}}(\boldsymbol{z}, \boldsymbol{m}_2)\big) + \delta^0_{\mathbf{T}}(\boldsymbol{z}, \boldsymbol{m}_1). \quad (7)$$

The ray points $\boldsymbol{v}_0 + \boldsymbol{m}_1 + \lambda \boldsymbol{m}_2$ are in the circumsphere of $\mathbf{T} \cup \{\boldsymbol{v}_0 + \boldsymbol{z}\}$ if and only if $\delta^0_{\mathbf{T}}(\boldsymbol{z}, \boldsymbol{m}_1 + \lambda \boldsymbol{m}_2) \leq 0$. Since $\delta^0_{\mathbf{T}}(\boldsymbol{z}, \boldsymbol{m}_1 + \lambda \boldsymbol{m}_2)$ is a quadratic function in λ, there are either zero or two (possibly equal) real roots $\lambda_1 \leq \lambda_2$. In the first case, there is no intersection between the circumsphere of $\mathbf{T} \cup \{\boldsymbol{v}_0 + \boldsymbol{z}\}$ and the ray, whereas in the second case, the ray points such that $\lambda \in [\lambda_1; \lambda_2] \cap \mathbb{Z}_{\geq 0}$, lie in the circumsphere of $\mathbf{T} \cup \{\boldsymbol{v}_0 + \boldsymbol{z}\}$ and are closer than $\boldsymbol{z}$ according to $\leq_{\mathbf{T}}$.

In algorithm 2, we check the sign of the discriminant and either return an empty list if it is strictly negative or return the (possibly empty) range of ray points as the list of the (possibly equal) lower and upper bounds.

Algorithm 2: SphereRayIntersection$(\mathbf{T}, \boldsymbol{q}, \sigma', \sigma)$

Input: the base triangle $\mathbf{T}$, the exterior point $\boldsymbol{q}$, a point $\boldsymbol{y}_{\sigma'}$, a ray $\mathcal{R}_\sigma$
Output: either empty () or the bounds $(\lambda_1, \lambda_2) \in \mathbb{Z}^2_{\geq 0}$ of the greatest interval of points $\{\mathcal{R}_\sigma[\lambda], \lambda_1 \leq \lambda \leq \lambda_2, \lambda \in \mathbb{Z}_{\geq 0}\}$, such that $\boldsymbol{y}_{\sigma'} \leq_{\mathbf{T}} \mathcal{R}_\sigma[\lambda]$

$(\boldsymbol{z}, \boldsymbol{m}_2, \boldsymbol{m}_1) \leftarrow \big(\boldsymbol{y}_{\sigma'} - \boldsymbol{v}_{\sigma(0)}, \boldsymbol{q} - \boldsymbol{v}_{\sigma(2)}, \boldsymbol{q} - \boldsymbol{v}_{\sigma(1)}\big)$;
$(a, b, c) \leftarrow \big({\boldsymbol{m}_2}^2, -{\boldsymbol{m}_2}^2 + 2\boldsymbol{z} \cdot \boldsymbol{m}_1 + \delta^0_{\mathbf{T}}(\boldsymbol{z}, \boldsymbol{m}_2), \delta^0_{\mathbf{T}}(\boldsymbol{z}, \boldsymbol{m}_1)\big)$;
$d \leftarrow b^2 - 4ac$;
if $d \geq 0$ **then**
  $(\lambda_1, \lambda_2) \leftarrow \Big(\lceil(-b - \sqrt{d})/(2a)\rceil, \lfloor(-b + \sqrt{d})/(2a)\rfloor\Big)$;
  if $\lambda_1 \leq \lambda_2$ *and* $0 \leq \lambda_2$ **then return** $(\max(0, \lambda_1), \lambda_2)$;
return ()

ClosestOnRay In order to implement algorithm 3, we consider the family of spheres passing by the vertices of $\mathbf{T}$ and a ray point $\boldsymbol{v}_0 + \boldsymbol{m}_1 + \lambda \boldsymbol{m}_2$, for any $\lambda \in \mathbb{Z}_{\geq 0}$. Given a sphere, we want to check whether the next ray point, i.e., $\boldsymbol{v}_0 + \boldsymbol{m}_1 + (\lambda + 1)\boldsymbol{m}_2$, is located inside it or not.

Replacing both $\boldsymbol{z}, \boldsymbol{z}'$ with $\boldsymbol{m}_1 + \lambda \boldsymbol{m}_2$, $\boldsymbol{z}''$ with $\boldsymbol{m}_2$, α with 1 in (6), we get:

$$\begin{aligned}\delta^0_{\mathbf{T}}(\boldsymbol{m}_1 + \lambda \boldsymbol{m}_2, \boldsymbol{m}_1 + (\lambda + 1)\boldsymbol{m}_2) &= \delta^0_{\mathbf{T}}(\boldsymbol{m}_1 + \lambda \boldsymbol{m}_2, \boldsymbol{m}_2) \\ &\quad + 2(\boldsymbol{m}_1 + \lambda \boldsymbol{m}_2) \cdot \boldsymbol{m}_2 \det[\boldsymbol{d}_0, -\boldsymbol{d}_2, \boldsymbol{m}_1 + \lambda \boldsymbol{m}_2].\end{aligned}$$

Using (7) (with $\boldsymbol{z} = \boldsymbol{m}_2$) and $\det[\boldsymbol{d}_0, -\boldsymbol{d}_2, \boldsymbol{m}_1 + \lambda \boldsymbol{m}_2] = \lambda + 1$ (again from [10, Lemma 3]), this expression can be simplified into:

$$\begin{aligned}&\delta^0_{\mathbf{T}}(\boldsymbol{m}_1 + \lambda \boldsymbol{m}_2, \boldsymbol{m}_1 + (\lambda + 1)\boldsymbol{m}_2) = \\ &\qquad \lambda^2({\boldsymbol{m}_2}^2) + \lambda(3{\boldsymbol{m}_2}^2) + 2(\boldsymbol{m}_1 \cdot \boldsymbol{m}_2) + \delta^0_{\mathbf{T}}(\boldsymbol{m}_1, \boldsymbol{m}_2). \quad (8)\end{aligned}$$

Clearly the determinant $\delta^0_{\mathbf{T}}(\boldsymbol{m}_1+\lambda\boldsymbol{m}_2, \boldsymbol{m}_1+(\lambda+1)\boldsymbol{m}_2)$ is a quadratic function in λ, whose minimum is reached at $\lambda = -3/2$. It is therefore monotonically increasing over $[0;\infty)$. Let $\lambda^\star$ be the smallest integer $\lambda \in \mathbb{Z}_{\geq 0}$ such that $\delta^0_{\mathbf{T}}(\boldsymbol{m}_1 + \lambda\boldsymbol{m}_2, \boldsymbol{m}_1 + (\lambda+1)\boldsymbol{m}_2) > 0$. By definition, $\delta^0_{\mathbf{T}}(\boldsymbol{m}_1 + \lambda^\star\boldsymbol{m}_2, \boldsymbol{m}_1 + (\lambda^\star+1)\boldsymbol{m}_2) > 0$, which means that the sphere passing by the vertices of $\mathbf{T}$ and $\boldsymbol{v}_0 + \boldsymbol{m}_1 + \lambda^\star\boldsymbol{m}_2$ contains neither $\boldsymbol{v}_0 + \boldsymbol{m}_1 + (\lambda^\star+1)\boldsymbol{m}_2$ nor the following ray points by transitivity, because $\delta^0_{\mathbf{T}}(\boldsymbol{m}_1 + \lambda\boldsymbol{m}_2, \boldsymbol{m}_1 + (\lambda+1)\boldsymbol{m}_2) > 0$ also for all $\lambda \geq \lambda^\star$. In other words, $\forall \lambda \geq \lambda^\star, \boldsymbol{v}_0 + \boldsymbol{m}_1 + \lambda^\star\boldsymbol{m}_2 \leq_{\mathbf{T}} \boldsymbol{v}_0 + \boldsymbol{m}_1 + \lambda\boldsymbol{m}_2$. In addition, if $\lambda^\star \geq 1$, $\delta^0_{\mathbf{T}}(\boldsymbol{m}_1 + (\lambda^\star-1)\boldsymbol{m}_2, \boldsymbol{m}_1 + \lambda^\star\boldsymbol{m}_2) \leq 0$, which is equivalent to $\delta^0_{\mathbf{T}}(\boldsymbol{m}_1 + \lambda^\star\boldsymbol{m}_2, \boldsymbol{m}_1 + (\lambda^\star-1)\boldsymbol{m}_2) > 0$. This means that $\forall \lambda \in 0, \ldots, \lambda^\star, \boldsymbol{v}_0 + \boldsymbol{m}_1 + \lambda^\star\boldsymbol{m}_2 \leq_{\mathbf{T}} \boldsymbol{v}_0 + \boldsymbol{m}_1 + \lambda\boldsymbol{m}_2$. As a consequence, $\lambda^\star$ provides the closest ray point. Algorithm 3 just computes $\lambda^\star$ and returns the corresponding ray point.

Algorithm 3: ClosestOnRay($\mathbf{T}, \boldsymbol{q}, \sigma$). Searches for the closest point according to $\leq_{\mathbf{T}}$ on ray $\mathcal{R}_\sigma$.

Input: the base triangle $\mathbf{T}$, the exterior point $\boldsymbol{q}$, a ray $\mathcal{R}_\sigma$
Output: the smallest $\lambda \in \mathbb{Z}_{\geq 0}$ such that $\mathcal{R}_\sigma[\lambda] \leq_{\mathbf{T}} \mathcal{R}_\sigma[\lambda+1]$
$(\boldsymbol{m}_1, \boldsymbol{m}_2) \leftarrow \big(\boldsymbol{q} - \boldsymbol{v}_{\sigma(1)}, \boldsymbol{q} - \boldsymbol{v}_{\sigma(2)}\big)$;
$(a, b, c) \leftarrow \big({\boldsymbol{m}_2}^2, 3{\boldsymbol{m}_2}^2, 2\boldsymbol{m}_1 \cdot \boldsymbol{m}_2 + \delta^0_{\mathbf{T}}(\boldsymbol{m}_1, \boldsymbol{m}_2)\big)$;
$d \leftarrow b^2 - 4ac$;
if $d < 0$ **then return** 0;
$\lambda^\star \leftarrow \lceil(-b + \sqrt{d})/(2a)\rceil$;
return $\max(0, \lambda^\star)$

ClosestAmongPointAndRay Once the ray that can hold a candidate point on $\mathbf{P}$ has been identified, it remains to determine whether the closest point to the current triangle $\mathbf{T}$ (according to $\leq_{\mathbf{T}}$) lies on the ray or is another point $\boldsymbol{y}$ of the H-neighborhood. Algorithm 4 performs this operation with the following steps. First it calls algorithm 2 to find if there may be an interval of points on the ray that are closer than $\boldsymbol{y}$. If it is the case, it checks if at least one belongs to $\mathbf{P}$. If this is the case it calls algorithm 3 to determine the one that is closest. If it belongs to $\mathbf{P}$, we are done. Otherwise, we have to find the closest point on the ray that belongs to $\mathbf{P}$ by a call to algorithm 5. This routine simply performs an exponential march followed by a binary search.

5 Complexity Analysis and Experimental Results

Upper bound on the number of calls to predicate Given the previous functions that are used to update the triangle at each step, we can prove:

Theorem 1. *The number of calls to predicate $P(\boldsymbol{x})$:= "Is $\boldsymbol{x}$ in $\mathbf{P}$" in R^1-algorithm (algorithm 1) is upper bounded by $O(\omega)$.*

Proof. Let $A^{(i)} := \boldsymbol{v}_0^{(i)} \cdot \mathbf{N} + \boldsymbol{v}_1^{(i)} \cdot \mathbf{N} + \boldsymbol{v}_2^{(i)} \cdot \mathbf{N}$ be the height of the current triangle. Let us denote $\lambda^{(i)}$ the integer related to the update of a vertex at iteration i, i.e., $\exists \pi \in \Sigma, \boldsymbol{v}_{\pi(0)}^{(i+1)} \leftarrow \boldsymbol{v}_{\pi(0)}^{(i)} + \boldsymbol{m}_{\pi(1)}^{(i)} + \lambda^{(i)}\boldsymbol{m}_{\pi(2)}^{(i)}$.

Algorithm 4: ClosestAmongPointAndRay$(P, \mathbf{T}, \boldsymbol{q}, \sigma', \sigma)$

Input: the predicate P, the base triangle $\mathbf{T}$, the exterior point $\boldsymbol{q}$, a candidate point $\boldsymbol{y}_{\sigma'}$ with $P(\boldsymbol{y}_{\sigma'})$, a ray $\mathcal{R}_\sigma$ with $P(\boldsymbol{y}_\sigma)$
Output: A couple (τ, α), such that $\tau \in \{\sigma, \sigma'\}$, $\alpha \in \mathbb{Z}_{\geq 0}$ and $P(\mathcal{R}_\tau[\alpha])$

```
L ← SphereRayIntersection(T, q, y_σ', σ);
if L ≠ ∅ then
    α1, α2 ← L;
    if P(R_σ[α1]) then
        α ← ClosestOnRay(T, q, σ);
        if α1 ≤ α ≤ α2 then
            if P(R_σ[α]) then return (σ, α);
            else return (σ, FindLast(P, T, q, σ, α1));
return (σ', 0)
```

Algorithm 5: FindLast$(P, \mathbf{T}, \boldsymbol{q}, \sigma, \alpha_1)$ Use exponential march then binary search to find the last point in $\mathcal{R}_\sigma$ that is in $\mathbf{P}$.

Input: the predicate P, the base triangle $\mathbf{T}$, the exterior point $\boldsymbol{q}$, a ray $\mathcal{R}_\sigma$, an integer α_1 with $P(\mathcal{R}_\sigma[\alpha_1])$
Output: the integer λ such that $P(\mathcal{R}_\sigma[\lambda])$ and $\neg P(\mathcal{R}_\sigma[\lambda+1])$

```
J ← 1;
while P(R_σ[α1 + J]) do J ← 2J;
λ1 ← α1 + ⌊J/2⌋; λ2 ← α1 + J;
while λ2 ≠ λ1 + 1 do
    λ ← ⌊(λ1+λ2)/2⌋ ;
    if P(R_σ[λ]) then λ1 ← λ;
    else λ2 ← λ;
return λ1
```

Since $\forall k, \boldsymbol{m}_k^{(i)} \cdot \mathbf{N} \geq 1$ [10, Lemma 5] then $A^{(i+1)} - A^{(i)} \geq 1 + \lambda^{(i)}$. Noticing that $A^{(0)} \geq 2\omega$, $A^{(n)} \leq 3\omega - 3$ [10, Theorem 2], suming over all iterations gives

$$\omega - 3 \geq A^{(n)} - A^{(0)} \geq \sum_{i=0}^{n-1} (1 + \lambda^{(i)}). \tag{9}$$

If we look now at the number $B^{(i)}$ of calls to P at iteration i, we must count the 6 calls for determining the H-neighborhood, 2 more possible calls in ClosestAmongPointAndRay, and possibly $2\log_2 J$ calls in FindLast. But $\alpha_1 + J/2 \leq \lambda^{(i)}$, thus $2\log_2 J \leq 2\log_2(1 + \lambda^{(i)})$. We get straightforwardly that $B^{(i)} \leq 8 + 2\log_2(1 + \lambda^{(i)})$. Recalling that $x \geq \log_2(1 + x)$ and suming the $B^{(i)}$, we derive from (9) that

$$\omega - 3 \geq \sum_{i=0}^{n-1} (1 + \lambda^{(i)}) \geq \frac{1}{8} \sum_{i=0}^{n-1} (8 + 8\log_2(1 + \lambda^{(i)})) \geq \frac{1}{8} \sum_{i=0}^{n-1} B^{(i)}. \tag{10}$$

Noticing that $\sum_{i=0}^{n-1} B^{(i)}$ is the total number of calls to P concludes. □

It is worthy to note that the R^1-algorithm performs also $O(\omega)$ arithmetic, square root and rounding operations.

Furthermore, plane-probing algorithms are run in this work on a digital plane, i.e., an infinite point set. However, since they are local and stop after a finite number of steps (theorem 1), there is some finite subsets for which processing the whole digital plane or only one of such subsets would be equivalent. Characterizing these subsets and bounding their size or diameter is not easy and may involve geometrical arguments based on the empty-circumsphere criterion. Then it will be possible to express the time complexity of plane-probing algorithms relatively to this bound, but it is still an open question.

Experimental evaluation We ran all three algorithms (H, R and R^1) on planes whose normal vector is ranging from (1,1,1) to (200, 200, 200). There are 6578833 vectors with relatively prime components in this range. Results are reported in the inset table. For the number of steps, i.e., n, and the total number of calls to predicate P, i.e., $\sum_i B^i$, we computed the average over the 6578833 runs. However, for the number of calls to P at step i, i.e., B^i, we computed the average and maximum over all steps and runs.

	n	B^i		$\sum_{i=0}^{n-1} B^i$
alg.	avg.	avg.	max.	avg.
H	24.84	6.00	6	149.01
R	17.59	14.49	25	254.95
R^1	17.32	7.06	14	122.36

First, the number of steps is 70% lower on average for the R- and R^1-algorithms than for the H-algorithm. Note that the number of steps is not exactly the same for R and R^1, because the arbitrary choice of a closest point in case of several closest co-spheric points is not the same. All three algorithms have however the same worst case: there are indeed always $n = 2r - 1$ steps for planes with normal $\mathbf{N}(1, r, r)$.

Second, the number of calls to P at each step is exactly 6 for the H-algorithm, close to 14 on average for the R-algorithm, but between 6 and 8 in most cases for the R^1-algorithm – a greater number of calls happens only occasionally, when function FindLast is called in algorithm 4.

The total number of calls to P is the lowest on average for the R^1-algorithm, which is a good trade-off between the number of steps and the number of calls at each step. On small-size vectors, i.e., ranging from (1,1,1) to (200, 200, 200), the R^1-algorithm is approximately twice faster than the R-algorithm, because the total number of calls to P is close to 122 on average for the R^1-algorithm while it is close to 255 for the R-algorithm.

For larger vectors, we can observe that the number of calls to P per step is still constant on average for the R^1-algorithm, but may be arbitrary large for the R-algorithm. Indeed, on digital planes of normal $\mathbf{N}(1, F_k, F_{k+1})$, where F_k is the k-th term of the Fibonacci sequence, the average number of predicate calls at each step increases as k increases for R, while it remains close to 6 for R^1 (see inset table).

alg.\k	10	20	30
R	18.50	25.39	36.33
R^1	6.80	6.20	6.11

6 Conlusion and Perspectives

We have presented a new plane-probing algorithm that outperforms both in theory and practice the state-of-the-art R-algorithm. It avoids computations that do not contribute to the final solution and performs fewer calls to the predicate "Is $\boldsymbol{x} \in \mathbf{P}$?": between 6 and 8 calls most of the time at each step and $O(\omega)$ calls in total. This algorithm is expected to be an efficient tool for digital surface analysis.

It remains to understand why these algorithms perform so much better on average than the worst-case bound $O(\omega)$. Therefore, we will investigate in the future their link with multi-dimensional continued fractions and dynamic number theory.

References

1. Buzer, L.: A linear incremental algorithm for naive and standard digital lines and planes recognition. Graphical Models **65**(1-3), 61–76 (2003)
2. Charrier, E., Buzer, L.: An efficient and quasi linear worst-case time algorithm for digital plane recognition. In: Discrete Geometry for Computer Imagery, LNCS, vol. 4992, pp. 346–357. Springer (2008)
3. Charrier, E., Lachaud, J.O.: Maximal planes and multiscale tangential cover of 3d digital objects. In: Int. Workshop Combinatorial Image Analysis. LNCS, vol. 6636, pp. 132–143. Springer Berlin / Heidelberg (2011)
4. Debled-Rennesson, I., Reveillès, J.: An incremental algorithm for digital plane recognition. In: Discrete Geometry for Computer Imagery. pp. 194–205 (1994)
5. Fernique, T.: Generation and recognition of digital planes using multi-dimensional continued fractions. Pattern Recognition **42**(10), 2229–2238 (2009)
6. Gérard, Y., Debled-Rennesson, I., Zimmermann, P.: An elementary digital plane recognition algorithm. Discrete Applied Mathematics **151**(1), 169–183 (2005)
7. Klette, R., Sun, H.J.: Digital planar segment based polyhedrization for surface area estimation. In: Visual Form, LNCS, vol. 2059, pp. 356–366. Springer (2001)
8. 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)
9. Lachaud, J.O., Provençal, X., Roussillon, T.: Computation of the normal vector to a digital plane by sampling signicant points. In: Proc. Discrete Geometry for Computer Imagery. pp. 194–205 (2016)
10. 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)
11. Mesmoudi, M.M.: A Simplified Recognition Algorithm of Digital Planes Pieces. In: Discrete Geometry for Computer Imagery. pp. 404–416 (2002)
12. 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)
13. Veelaert, P.: Digital planarity of rectangular surface segments. IEEE Transactions on Pattern Analysis and Machine Intelligence **16**(6), 647–652 (1994)
14. Veelaert, P.: Fast Combinatorial Algorithm for Tightly Separating Hyperplanes. In: Int. Workshop Combinatorial Image Analysis. pp. 31–44 (2012)