Plane-probing algorithms
for the analysis of digital surfaces

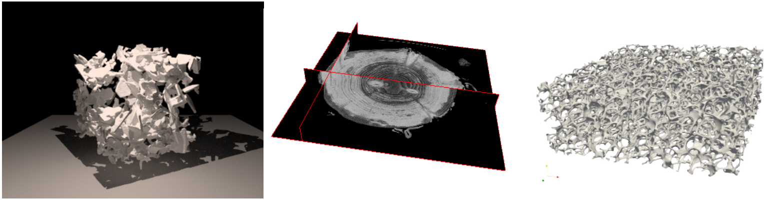
Tristan Roussillon

Université de Lyon, INSA Lyon, LIRIS, France

DGDVC, 30/03/2021

PARADIS (ANR-18-CE23-0007-01) research grant

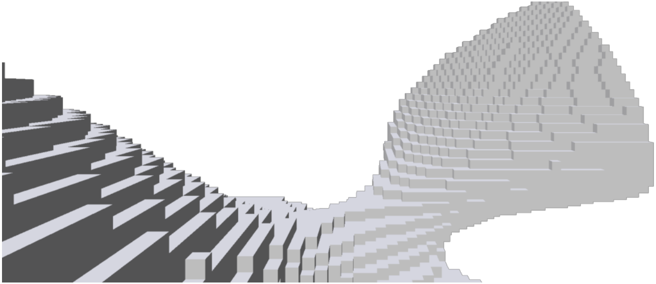
Data



(a) (b) (c)

voxel sets in 3d digital images

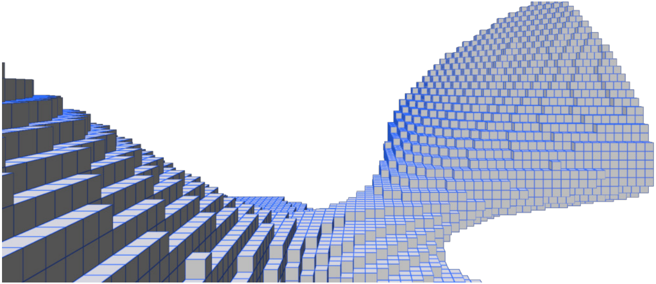
Digital surfaces



pros/cons

- + efficient spatial data structures
- + set operations (union, intersection, ...)
- + integer-only, exact computations
- + ...
- − poor geometry

Digital surfaces



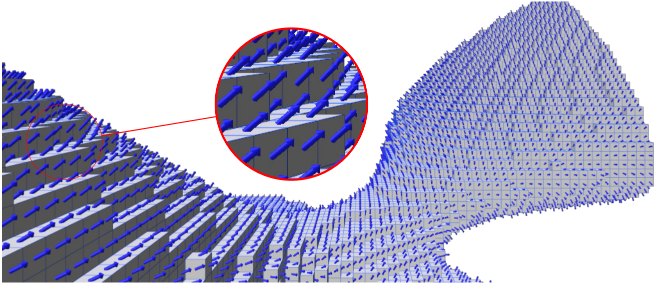
pros/cons

+ efficient spatial data structures
+ set operations (union, intersection, . . .)
+ integer-only, exact computations
+ . . .
− poor geometry

Analysis of digital surfaces

- enhance the geometry by estimating normal vectors

⇒ applications: measurements, deformation for simulation or tracking, surface fairing, rendering...



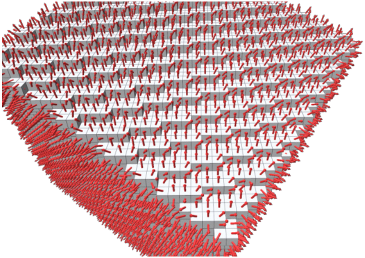
A lot of methods

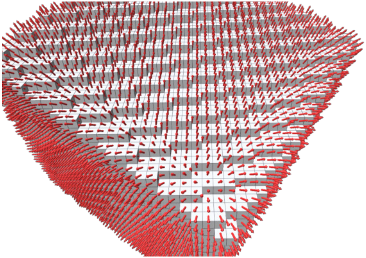
- fitting,
- Voronoi diagram,
- integral invariants,
- convolution,
- energy minimization,
- probabilistic approaches,
- ...

Flaw

Existing methods are not quite satisfactory

- parameter required ($\approx$ width of a neighborhood)
- that parameter is hard to pick
  - get decent estimates in flat/smooth parts
  - preserve sharp features





Challenge

Desiderata

- parameter-free method
- theoretical guarantees
  - exact on flat parts
  - converge on smooth parts as resolution increases

Key idea

- bound neighborhoods by their thickness instead of their width
- digitized planes have a thickness bounded by a small constant

Plane-probing algorithms

Definition

Given a digitized plane P and a starting point $p \in P$, a plane-probing algorithm computes the normal vector of P by sparsely probing it with the predicate “is $x \in P$?”.

H and R	[LPR2017] J-O. L., X. P., T. R. Two Plane-Probing Algorithms for the Computation of the Normal Vector to a Digital Plane. *J. Math. Imaging Vis.*, 59(1):23–39, 2017.
R^1	[LR2019] T. R., J-O. L., An efficient and quasi linear worst-case time algorithm for digital plane recognition, *DGCI'19*, LNCS, vol. 11414, p.380–393, 2019.
PH, PR, PR^1	[LMR2020] J-O. L., J. M., T. R. An Optimized Framework for Plane-Probing Algorithms, *J. Math. Imaging Vis.*, 62(5):718–736, 2020.

Implemented in DGtal (`dgtal.org`)

Outline

Context and motivation

Plane-probing algorithms

- Generalized Euclidean algorithm
- Delaunay triangulation
- Generalization

Application to digital surfaces

One of the oldest algorithms

Euclidean algorithm

Given a couple of integers,

- subtract the smaller from the larger one, and repeat
- until both numbers are equal.

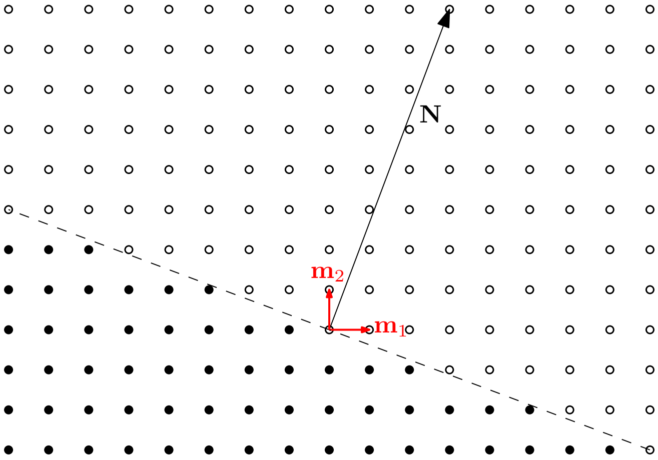
Example

step	0	1	2	3	4
a	3	3	3	1	1
b	8	5	2	2	1

we focus on the sequence of subtractions, assume $\gcd(a, b) = 1$

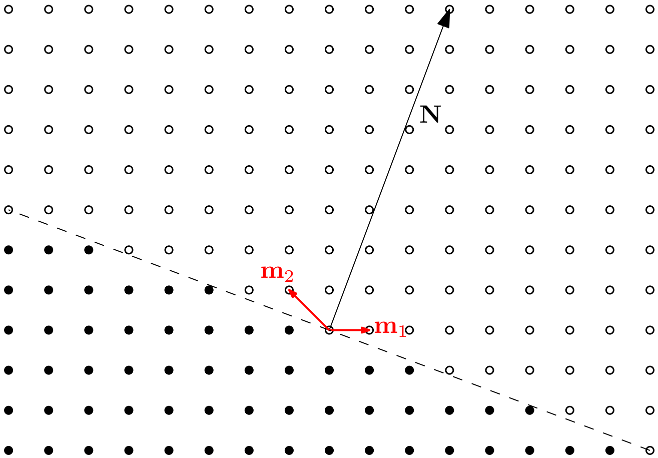
One geometrical interpretation of the Euclidean algorithm

$$\mathsf{m}_1 = (1,0), \quad \mathsf{m}_1 \cdot \mathsf{N} = a = 3$$
$$\mathsf{m}_2 = (0,1), \quad \mathsf{m}_2 \cdot \mathsf{N} = b = 8$$



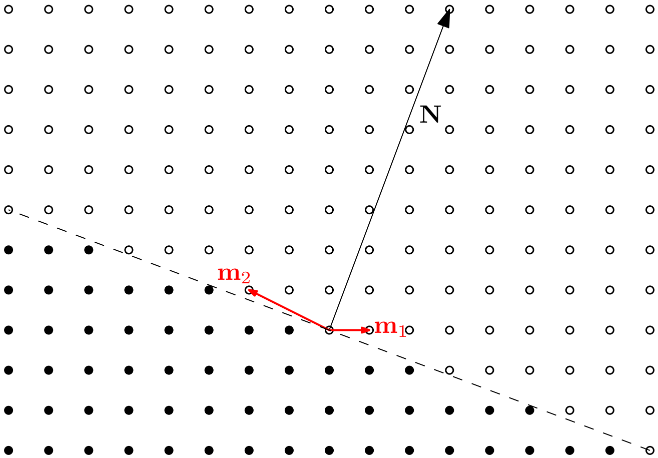
One geometrical interpretation of the Euclidean algorithm

$$\mathsf{m}_1 = (1,0), \qquad \mathsf{m}_1 \cdot \mathsf{N} = a = 3$$
$$\mathsf{m}_2 = (-1,1), \quad \mathsf{m}_2 \cdot \mathsf{N} = b = 5$$



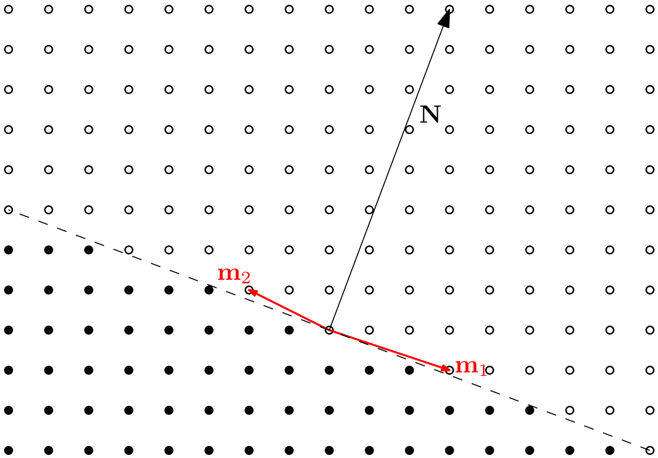
One geometrical interpretation of the Euclidean algorithm

$$\mathsf{m}_1 = (1,0), \quad \mathsf{m}_1 \cdot \mathsf{N} = a = 3$$
$$\mathsf{m}_2 = (-2,1), \quad \mathsf{m}_2 \cdot \mathsf{N} = b = 2$$



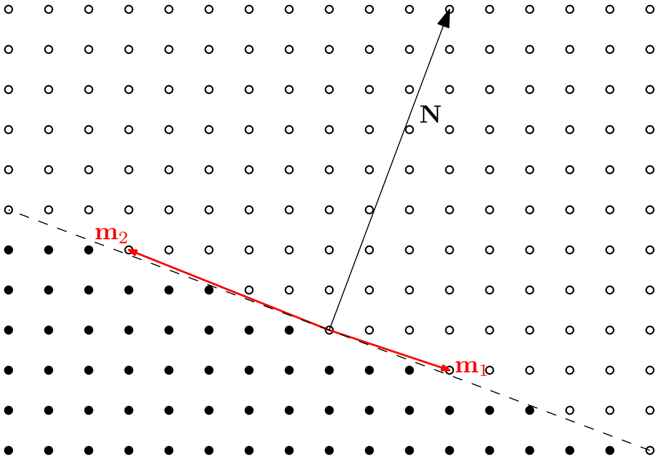
One geometrical interpretation of the Euclidean algorithm

$$\mathsf{m}_1 = (3, -1), \quad \mathsf{m}_1 \cdot \mathsf{N} = a = 1$$
$$\mathsf{m}_2 = (-2, 1), \quad \mathsf{m}_2 \cdot \mathsf{N} = b = 2$$



One geometrical interpretation of the Euclidean algorithm

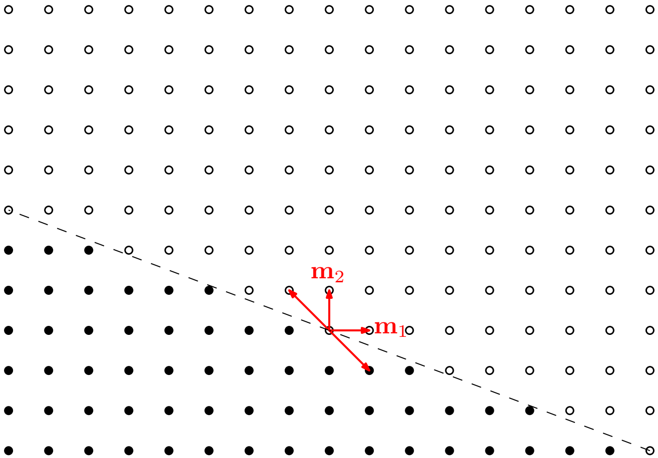
$$\mathsf{m}_1 = (3, -1), \quad \mathsf{m}_1 \cdot \mathsf{N} = a = 1$$
$$\mathsf{m}_2 = (-5, 2), \quad \mathsf{m}_2 \cdot \mathsf{N} = b = 1$$



An algorithm to compute N

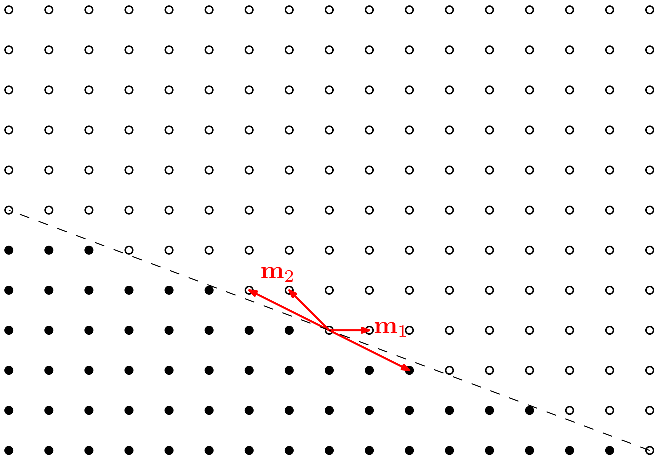
N is unknown, but a predicate IsBlack is given

IsBlack($m_1 - m_2$)? IsBlack($m_2 - m_1$)?



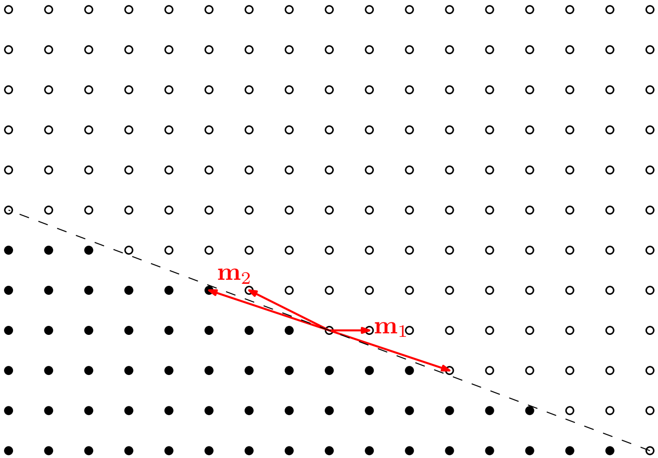
An algorithm to compute N

N is unknown, but a predicate IsBlack is given
IsBlack($m_1 - m_2$)? IsBlack($m_2 - m_1$)?



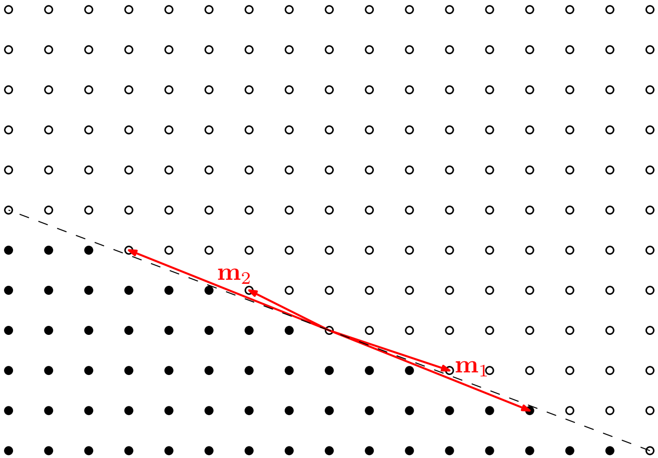
An algorithm to compute N

N is unknown, but a predicate IsBlack is given
IsBlack($m_1 - m_2$)? IsBlack($m_2 - m_1$)?



An algorithm to compute N

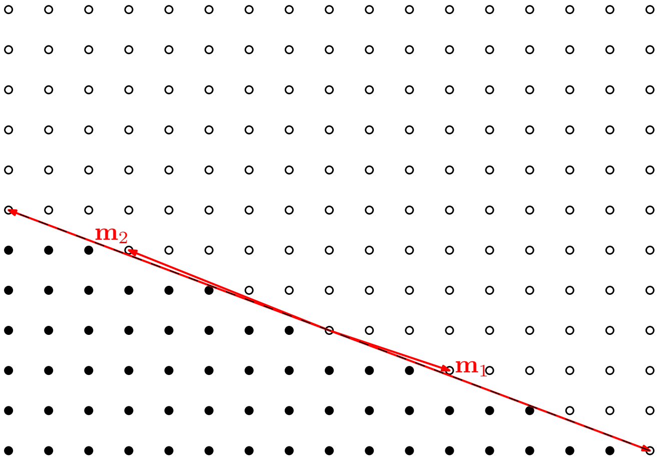
N is unknown, but a predicate IsBlack is given
IsBlack($m_1 - m_2$)? IsBlack($m_2 - m_1$)?



An algorithm to compute N

N is unknown, but a predicate IsBlack is given

IsBlack($m_1 - m_2$)? IsBlack($m_2 - m_1$)?



Extension to 3d

No unique extension to the Euclidean algorithm!

Assuming $0 \leq a \leq b \leq c$:

- *Brun*: $(a, b, c) \to (a, b, c - b)$;
- *Selmer*: $(a, b, c) \to (a, b, c - a)$;
- *Farey*: $(a, b, c) \to (a, b - a, c)$;
- *Fully-Subtractive*: $(a, b, c) \to (a, b - a, c - a)$;
- *Poincaré*: $(a, b, c) \to (a, b - a, c - b)$.
- ...

Note: the same operation is done at each step

A class of generalized Euclidean algorithms

Given three positive numbers (a, b, c), with $\gcd(a, b, c) = 1$,

- while they are not all equal to 1,
- subtract from a number $x \in \{a, b, c\}$ a strictly smaller number $y \in \{a, b, c\}$, $y < x$.

Example

$$\begin{aligned} \mathrm{m}_1 &= (1, 0, 0), & \mathrm{m}_1 \cdot \mathrm{N} &= a = 1 \\ \mathrm{m}_2 &= (0, 1, 0), & \mathrm{m}_2 \cdot \mathrm{N} &= b = 2 \\ \mathrm{m}_3 &= (0, 0, 1), & \mathrm{m}_3 \cdot \mathrm{N} &= c = 3 \end{aligned}$$

A class of generalized Euclidean algorithms

Given three positive numbers (a, b, c), with $\gcd(a, b, c) = 1$,

- while they are not all equal to 1,
- subtract from a number $x \in \{a, b, c\}$ a strictly smaller number $y \in \{a, b, c\}$, $y < x$.

Example

$$\begin{aligned} \mathrm{m}_1 &= (1, 0, 0), & \mathrm{m}_1 \cdot \mathrm{N} &= a = 1 \\ \mathrm{m}_2 &= (0, 1, 0), & \mathrm{m}_2 \cdot \mathrm{N} &= b = 2 \\ \mathrm{m}_3 &= (0, -1, 1), & \mathrm{m}_3 \cdot \mathrm{N} &= c = 1 \end{aligned}$$

A class of generalized Euclidean algorithms

Given three positive numbers (a, b, c), with $\gcd(a, b, c) = 1$,

- while they are not all equal to 1,
- subtract from a number $x \in \{a, b, c\}$ a strictly smaller number $y \in \{a, b, c\}$, $y < x$.

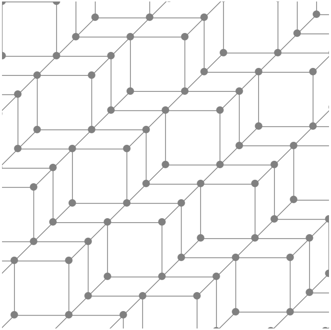
Example

$$\begin{aligned} \mathrm{m}_1 &= (1, 0, 0), & \mathrm{m}_1 \cdot \mathrm{N} &= a = 1 \\ \mathrm{m}_2 &= (-1, 1, 0), & \mathrm{m}_2 \cdot \mathrm{N} &= b = 1 \\ \mathrm{m}_3 &= (0, -1, 1), & \mathrm{m}_3 \cdot \mathrm{N} &= c = 1 \end{aligned}$$

Digital plane

Let $N \in \mathbb{Z}^3$ whose components (a, b, c) are coprime integers s.t. $0 < a \leq b \leq c$,

$$P_N := \{x \in \mathbb{Z}^3 \mid 0 \leq x \cdot N < \|N\|_1\}$$



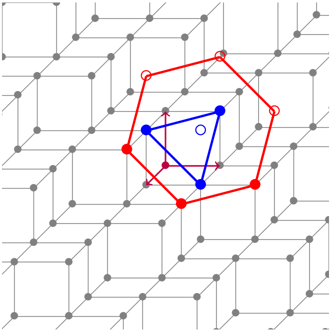
Interpretation of a generalized Euclidean algorithm

Internals

- $(\mathsf{m}_1, \mathsf{m}_2, \mathsf{m}_3) := (\mathsf{e}_1, \mathsf{e}_2, \mathsf{e}_3)$, $\mathsf{q} := (1, 1, 1) \notin \mathsf{P}_N$

$\Rightarrow$ triangle $(\mathsf{q} - \mathsf{m}_1, \mathsf{q} - \mathsf{m}_2, \mathsf{q} - \mathsf{m}_3)$

$\Rightarrow$ hexagon $\{\mathsf{q} + \mathsf{m}_i - \mathsf{m}_j \mid i, j \in \{1, 2, 3\}, i \neq j\}$



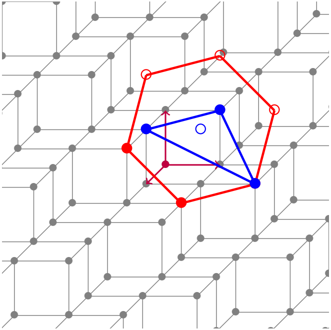
Interpretation of a generalized Euclidean algorithm

Internals

- $(m_1, m_2, m_3) := (e_1, e_2, e_3)$, $q := (1, 1, 1) \notin P_N$

$\Rightarrow$ triangle $(q - m_1, q - m_2, q - m_3)$

$\Rightarrow$ hexagon $\{q + m_i - m_j \mid i, j \in \{1, 2, 3\}, i \neq j\}$



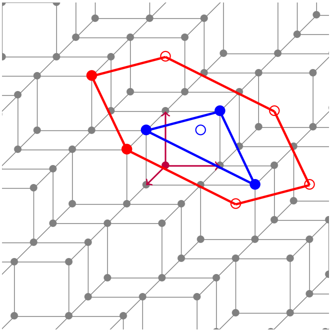
Interpretation of a generalized Euclidean algorithm

Internals

- $(m_1, m_2, m_3) := (e_1, e_2, e_3)$, $q := (1, 1, 1) \notin P_N$

$\Rightarrow$ triangle $(q - m_1, q - m_2, q - m_3)$

$\Rightarrow$ hexagon $\{q + m_i - m_j \mid i, j \in \{1, 2, 3\}, i \neq j\}$



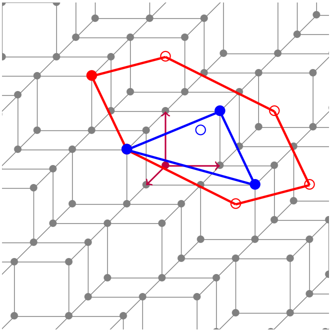
Interpretation of a generalized Euclidean algorithm

Internals

- $(\mathsf{m}_1, \mathsf{m}_2, \mathsf{m}_3) := (\mathsf{e}_1, \mathsf{e}_2, \mathsf{e}_3)$, $\mathsf{q} := (1, 1, 1) \notin \mathsf{P}_N$

$\Rightarrow$ triangle $(\mathsf{q} - \mathsf{m}_1, \mathsf{q} - \mathsf{m}_2, \mathsf{q} - \mathsf{m}_3)$

$\Rightarrow$ hexagon $\{\mathsf{q} + \mathsf{m}_i - \mathsf{m}_j \mid i, j \in \{1, 2, 3\}, i \neq j\}$



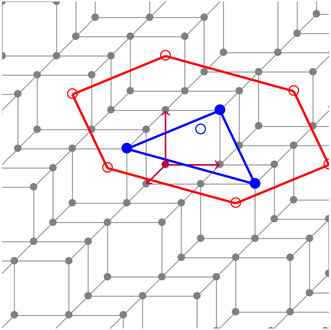
Interpretation of a generalized Euclidean algorithm

Internals

- $(m_1, m_2, m_3) := (e_1, e_2, e_3)$, $q := (1, 1, 1) \notin P_N$

$\Rightarrow$ triangle $(q - m_1, q - m_2, q - m_3)$

$\Rightarrow$ hexagon $\{q + m_i - m_j \mid i, j \in \{1, 2, 3\}, i \neq j\}$



$\Rightarrow$ a plane-probing algorithm

$$\Pi := \{P_N \mid N \in \mathbb{Z}^3 \setminus 0\}$$

Input

- $P \in \Pi$ described by the predicate InPlane: "is $x \in P$?"
- a starting point p s.t. InPlane(p), $q := p + (1, 1, 1)$

Main trick

- Assume $p \cdot N = 0$ ($\Rightarrow q \cdot N = \|N\|_1$), where N, the normal of P
- InPlane(x) $\Leftrightarrow (x - q) \cdot N < 0$.

Properties of generalized Euclidean algorithms

At each step

P1 p and q both project into triangle $(q - m_1, q - m_2, q - m_3)$ along $(1, 1, 1)$

P2 matrix $M := [m_1, m_2, m_3]$ is unimodular, i.e. $\det(M) = 1$

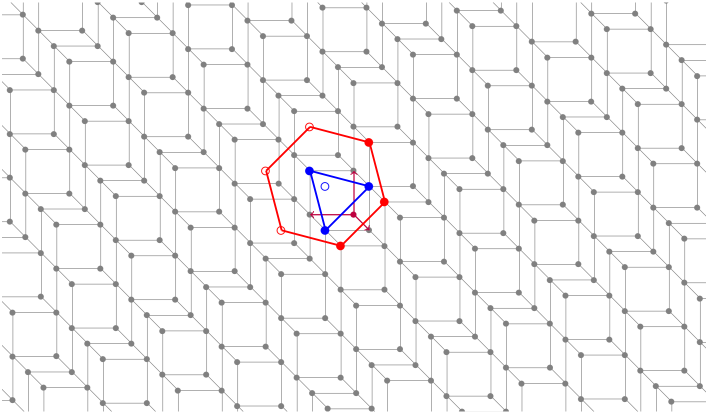
Termination

- number of steps $\leq \|N\|_1 - 3$ (6 calls to InPlane per step)
- at the end, if $p \cdot N = 0$ ($\Rightarrow q \cdot N = \|N\|_1$)
  $\forall k \in \{1, 2, 3\},\ m_k \cdot N = 1$
  $\Rightarrow$ the normal of triangle $(q - m_1, q - m_2, q - m_3)$ is N

whichever the subtraction we choose

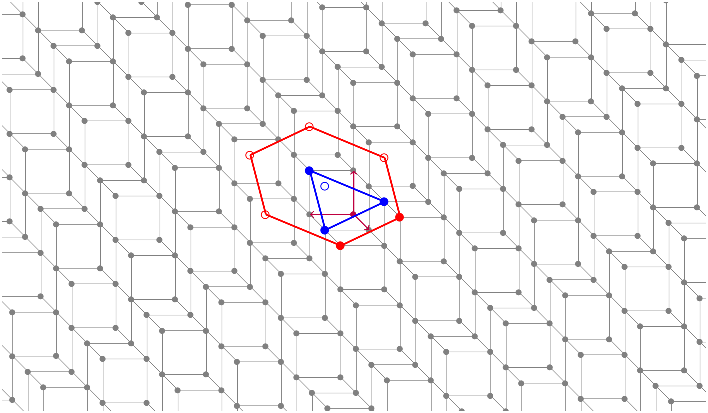
Example

Digital plane of normal $(5, 2, 3)$



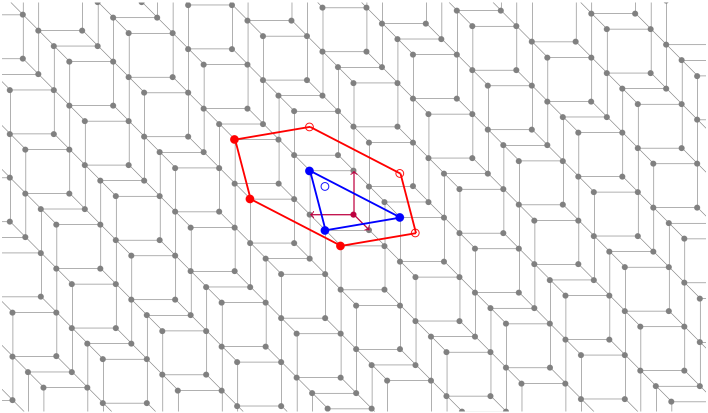
Example

Digital plane of normal $(5, 2, 3)$



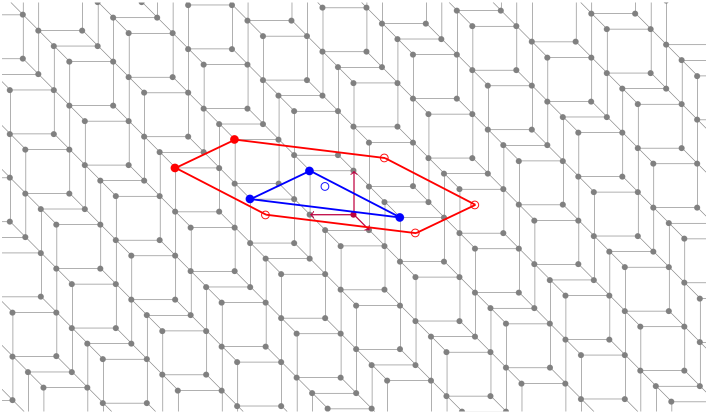
Example

Digital plane of normal $(5, 2, 3)$



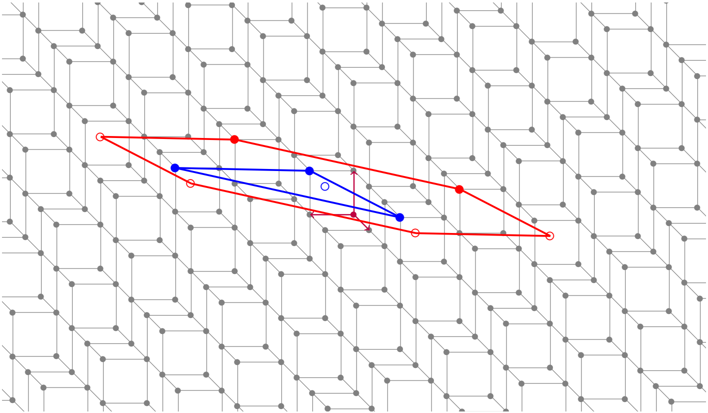
Example

Digital plane of normal $(5, 2, 3)$



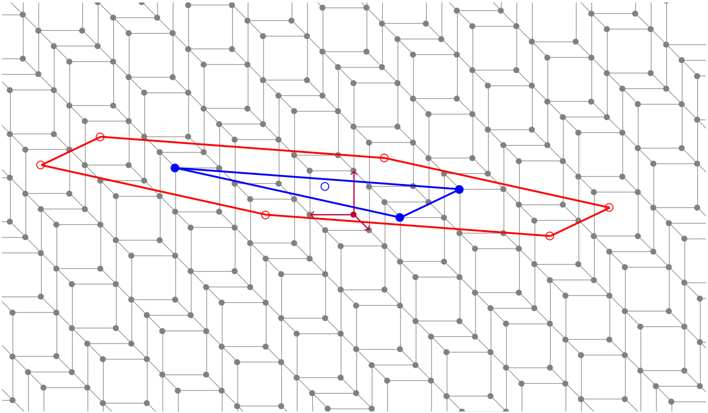
Example

Digital plane of normal $(5, 2, 3)$



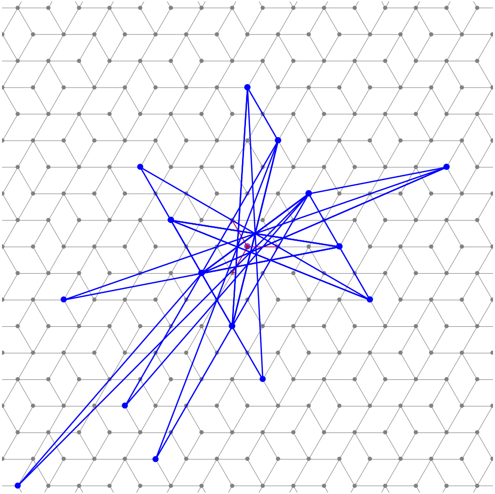
Example

Digital plane of normal $(5, 2, 3)$



All possible final triangles

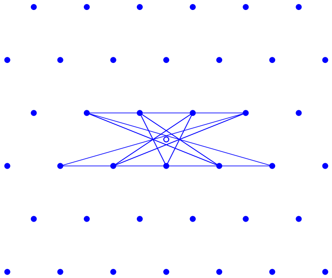
Digital plane of normal $(2,3,5)$



About final triangles

- vertices $\in \Lambda := \{x \in \mathbb{Z}^3 \mid x \cdot N = \|N\|_1 - 1\}$
- do not contain any other point of Λ (P2)
- projection of p along $(1,1,1)$ (P1)

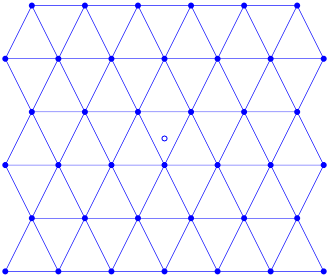
Digital plane of normal $(2,2,5)$



Towards a selection criterion

- The Delaunay triangulation of Λ gives acute triangles
- p projects into one of them (if no co-circularity)

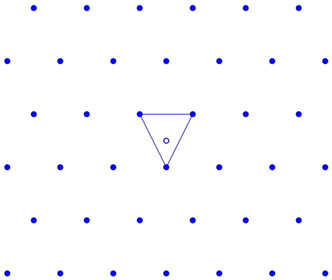
Digital plane of normal $(2, 2, 5)$



Towards a selection criterion

- The Delaunay triangulation of Λ gives acute triangles
- p projects into one of them (if no co-circularity)

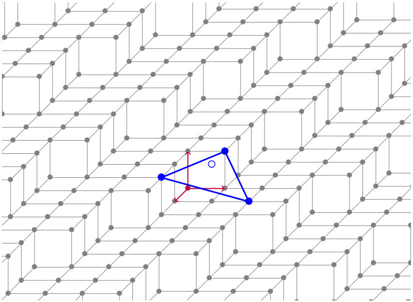
Digital plane of normal $(2, 2, 5)$



Algorithm H (candidates in an hexagon)

At each step:

- consider a candidate set S
- filter S through InPlane
- pick a *closest* point $s^\star$: the circumsphere of $T \cup s^\star$ doesn't contain any other
- update T with this point

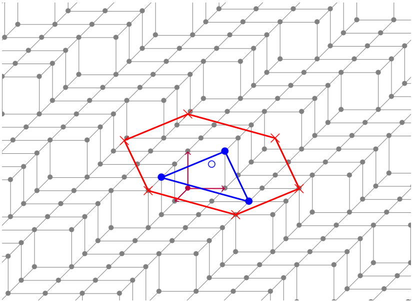


The last triangle is very often acute, but not always

Algorithm H (candidates in an hexagon)

At each step:

- consider a candidate set S
- filter S through InPlane
- pick a *closest* point $s^\star$:
  the circumsphere of $T \cup s^\star$
  doesn't contain any other
- update T with this point

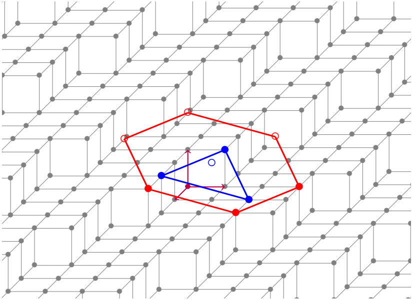


The last triangle is very often acute, but not always

Algorithm H (candidates in an hexagon)

At each step:

- consider a candidate set S
- filter S through InPlane
- pick a *closest* point $s^\star$: the circumsphere of $T \cup s^\star$ doesn't contain any other
- update T with this point

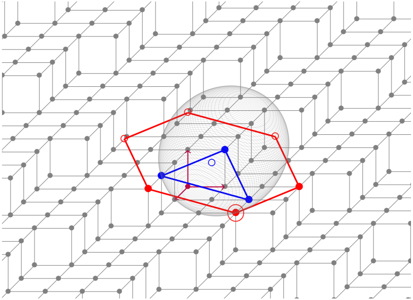


The last triangle is very often acute, but not always

Algorithm H (candidates in an hexagon)

At each step:

- consider a candidate set S
- filter S through InPlane
- pick a *closest* point $s^\star$:
  the circumsphere of $T \cup s^\star$
  doesn't contain any other
- update T with this point

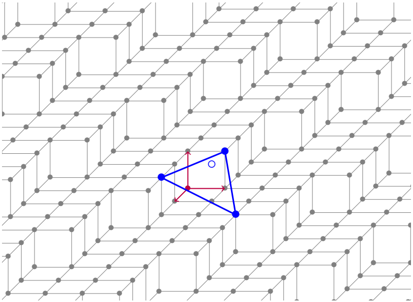


The last triangle is very often acute, but not always

Algorithm H (candidates in an hexagon)

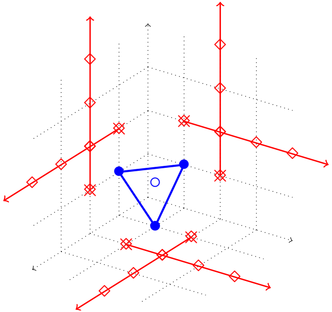
At each step:

- consider a candidate set S
- filter S through InPlane
- pick a *closest* point $s^\star$: the circumsphere of $T \cup s^\star$ doesn't contain any other
- update T with this point



The last triangle is very often acute, but not always

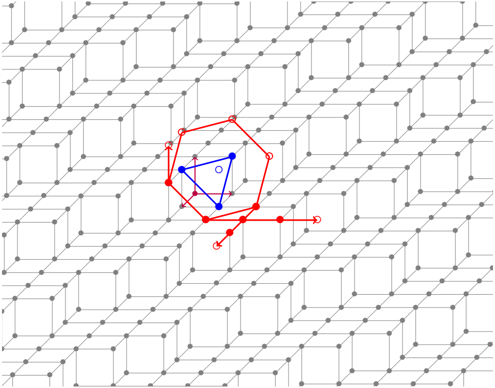
Algorithm R (candidates along rays)



- same algorithm as before, only S differs
- S is infinite but the filtering by InPlane gives a finite point set
- $O(\|N\|_1)$ steps, $O(\log(\|N\|_1))$ calls to InPlane per step
- the last triangle is always acute

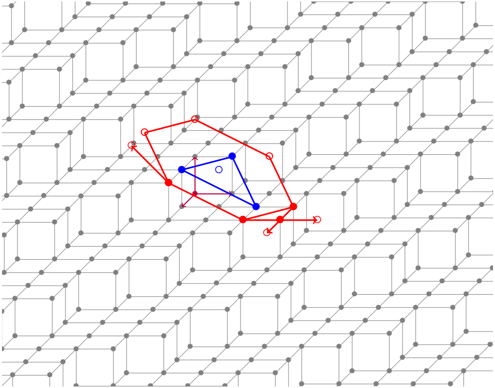
Example

Digital plane of normal $(2, 3, 9)$



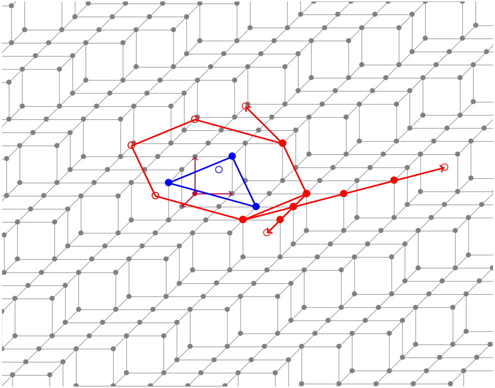
Example

Digital plane of normal $(2, 3, 9)$



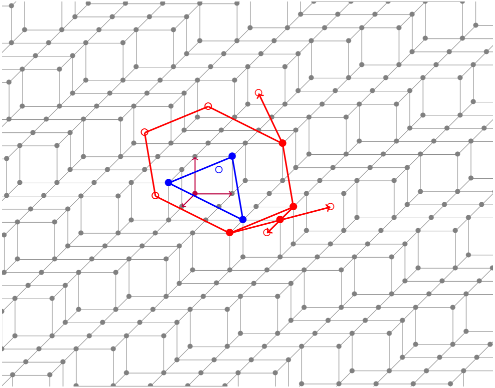
Example

Digital plane of normal $(2, 3, 9)$



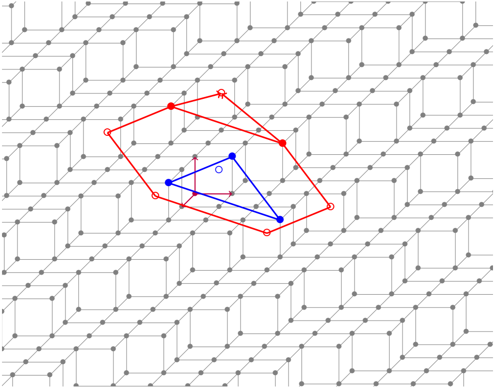
Example

Digital plane of normal $(2, 3, 9)$



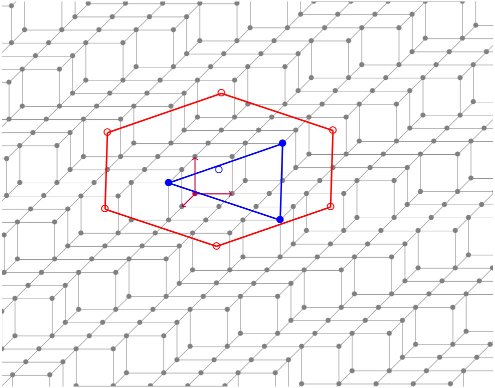
Example

Digital plane of normal $(2, 3, 9)$



Example

Digital plane of normal $(2, 3, 9)$



Algorithm R^1

Features

- has the same output as R
- but $O(\|N\|_1)$ calls to InPlane instead of $O(\|N\|_1 \log \|N\|_1)$

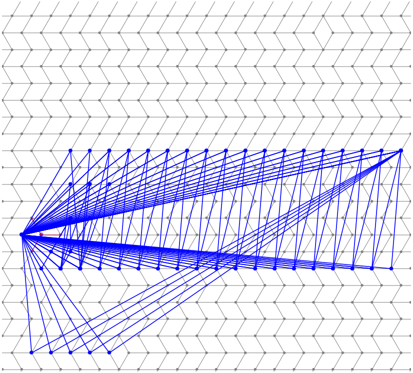
How?

1. local probing: 6 rays $\rightarrow$ at most 2 rays and 1 point
2. geometrical study: 2 rays $\rightarrow$ 1 ray and 1 point
3. efficient algorithm: 1 ray and 1 point $\rightarrow$ a *closest* point

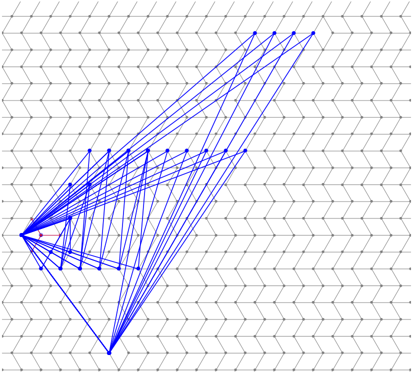
Example

Digital plane of normal $(67, 1, 91)$

H



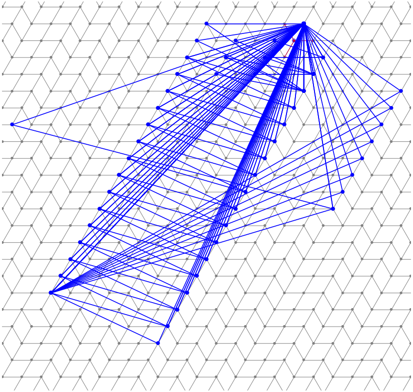
R/R^1



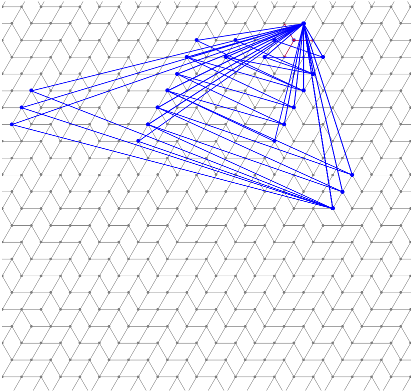
Example

Digital plane of normal $(1, 73, 100)$

H



R/R^1



Recap

Main features

- N from a point p s.t. $p \cdot N = 0$
- by sparse and local computations:
  - p projects into all triangles
  - with R and R^1, the current triangle is acute every two steps, always acute at the end
- $O(\|N\|_1)$ calls to InPlane with H and R^1, $O(\|N\|_1 \log(\|N\|_1))$ with R

Drawbacks

1. do not retrieve N from any point
2. do not retrieve all triangles of the lattice Λ

Problem #1: starting from any point

Input

- P of normal N
- InPlane: "is $x \in P$?"

Equivalence used so far

- assume $q \cdot N = \|N\|_1$
- $\text{InPlane}(x) \Leftrightarrow (x - q) \cdot N < 0$

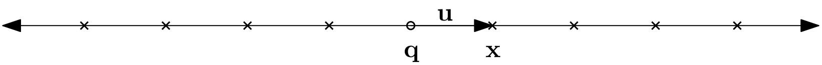
Generalized equivalence

- assume $q \cdot N \geq \|N\|_1$
- $\exists l \in \mathbb{N}$ s.t. $\text{InPlane}(q + l(x - q)) \Leftrightarrow (x - q) \cdot N < 0$.

Predicate NotAbove

Data: InPlane, q and an integer $L \geq 2\|N\|_1$
Input: A point $x \in \mathbb{Z}^3$ s.t. $q \cdot N - \|N\|_1 \leq x \cdot N$
Output: True iff $(x - q) \cdot N < 0$ in $O(\log(L))$ calls to InPlane

```
u ← x − q ; // direction
l ← 1;
while l < L do
    if InPlane(q + lu) then return True ;
    if InPlane(q − lu) then return False ;
    l ← 2l;
return False;
```

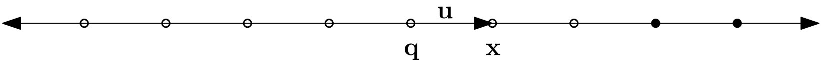


it is enough to use NotAbove instead of InPlane

Predicate NotAbove

Data: InPlane, q and an integer $L \geq 2\|N\|_1$
Input: A point $x \in \mathbb{Z}^3$ s.t. $q \cdot N - \|N\|_1 \leq x \cdot N$
Output: True iff $(x - q) \cdot N < 0$ in $O(\log(L))$ calls to InPlane

```
u ← x − q ; // direction
l ← 1;
while l < L do
    if InPlane(q + lu) then return True ;
    if InPlane(q − lu) then return False ;
    l ← 2l;
return False;
```

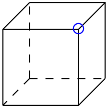


it is enough to use NotAbove instead of InPlane

Problem #2: retrieving all triangles

Triangle $\rightarrow$ Parallelepiped

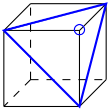
- top point q
- upper triangle $(q - m_1, q - m_2, q - m_3)$
- lower triangle $(q - m_2 - m_3, q - m_3 - m_1, q - m_1 - m_2)$
- bottom point $q - \sum_k m_k$



Problem #2: retrieving all triangles

Triangle → Parallelepiped

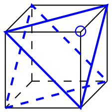
- top point q
- upper triangle $(q - m_1, q - m_2, q - m_3)$
- lower triangle $(q - m_2 - m_3, q - m_3 - m_1, q - m_1 - m_2)$
- bottom point $q - \sum_k m_k$



Problem #2: retrieving all triangles

Triangle → Parallelepiped

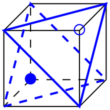
- top point q
- upper triangle $(q - m_1, q - m_2, q - m_3)$
- lower triangle $(q - m_2 - m_3, q - m_3 - m_1, q - m_1 - m_2)$
- bottom point $q - \sum_k m_k$



Problem #2: retrieving all triangles

Triangle → Parallelepiped

- top point q
- upper triangle $(q - m_1, q - m_2, q - m_3)$
- lower triangle $(q - m_2 - m_3, q - m_3 - m_1, q - m_1 - m_2)$
- bottom point $q - \sum_k m_k$



Staying close to the digital plane

Update rule

- when the parallelepiped has less than 4 vertices in P,
  - $\Leftrightarrow$ the lower triangle is updated (bottom moves, not top)
- otherwise
  - $\Leftrightarrow$ the upper triangle is updated (top moves, not bottom)
- invariant: at least one point in P (bottom), one not (top)

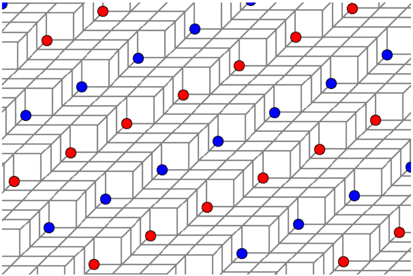
Generalized versions of H, R and R^1

For each $X \in \{H, R, R^1\}$, PX uses a parallelepiped and the above update rule with NotAbove instead of InPlane.

Recap

Main features

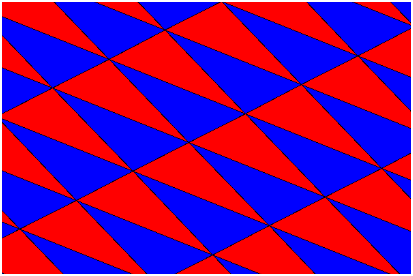
- N from any point p such that InPlane(p),
- all triangles of the lattice $\Lambda = \{x \in \mathbb{Z}^3 \mid x \cdot N = \|N\|_1 - 1\}$
- PH and PR[1] require $O(\|N\|_1)$ calls to NotAbove
  $\Rightarrow O(\|N\|_1 \log(\|N\|_1))$ calls to InPlane.



Recap

Main features

- N from any point p such that InPlane(p),
- all triangles of the lattice $\Lambda = \{x \in \mathbb{Z}^3 \mid x \cdot N = \|N\|_1 - 1\}$
- PH and PR[1] require $O(\|N\|_1)$ calls to NotAbove
  $\Rightarrow O(\|N\|_1 \log(\|N\|_1))$ calls to InPlane.



Outline

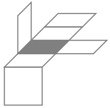
A similar algorithm for a digital surface S

Input

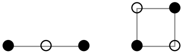
- a predicate InSurface : x $\in S$?
- a starting square face s in S

Additional constraints

- find an origin and a basis from s



- stop if non-planar configurations (parallelepiped/hexagon/rays)



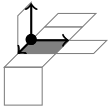
A similar algorithm for a digital surface S

Input

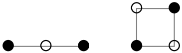
- a predicate InSurface : $x \in S$?
- a starting square face s in S

Additional constraints

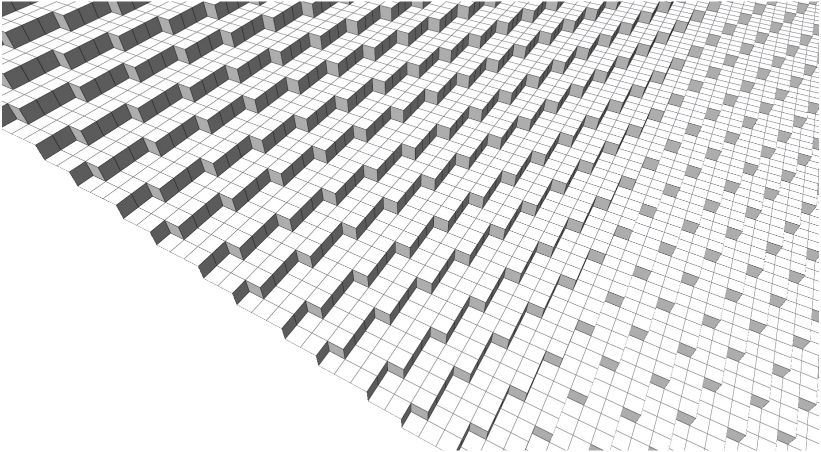
- find an origin and a basis from s



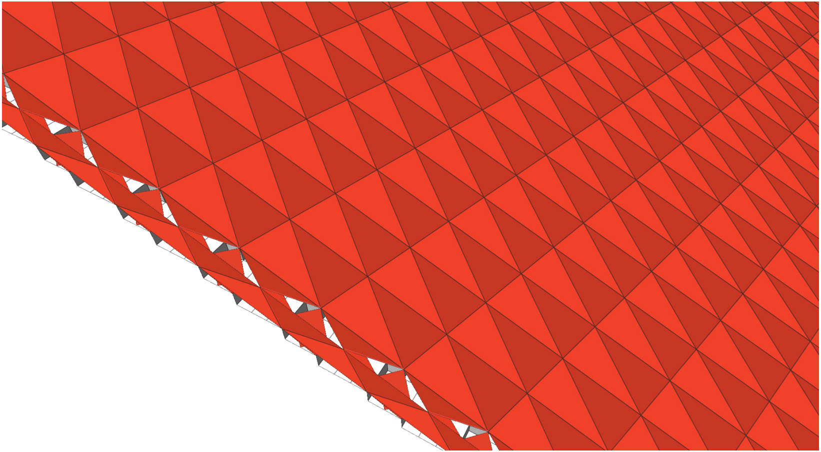
- stop if non-planar configurations (parallelepiped/hexagon/rays)



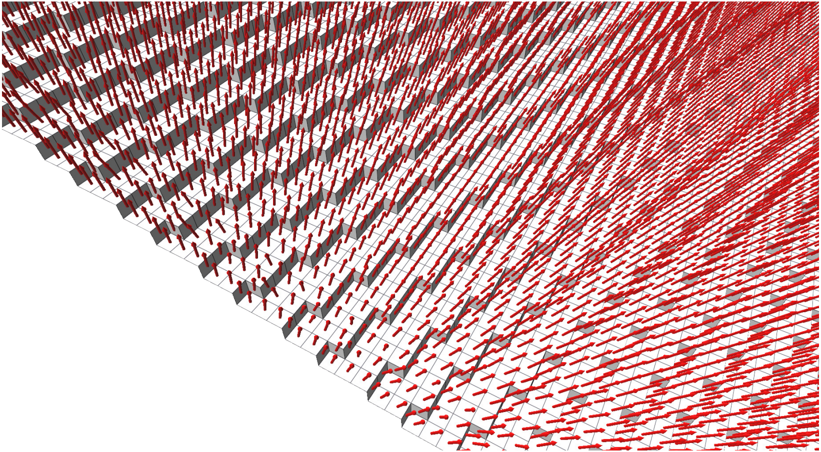
Example: flat parts and sharp edges



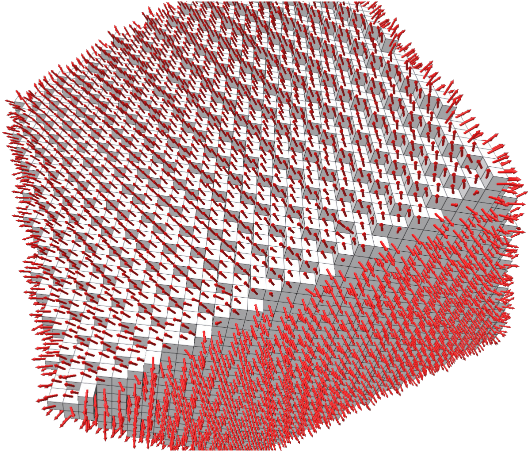
Example: flat parts and sharp edges



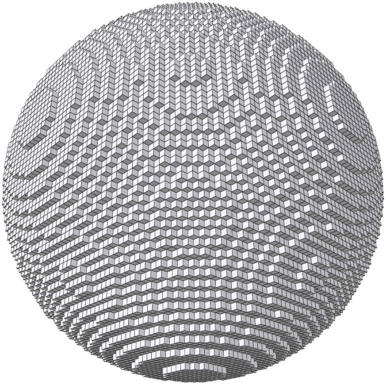
Example: flat parts and sharp edges



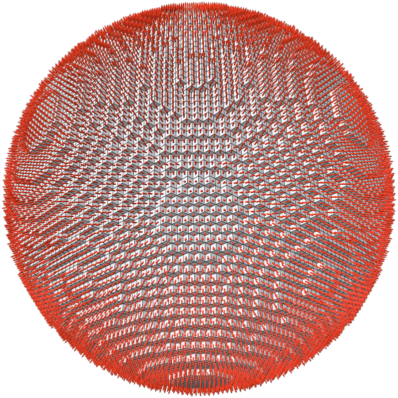
Example: flat parts and sharp edges



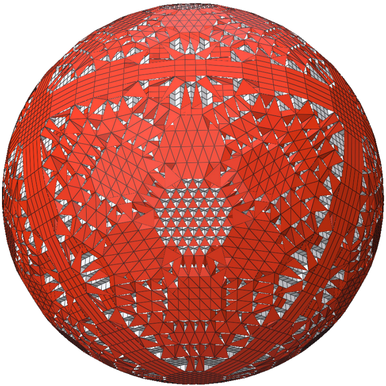
Example: convex shapes



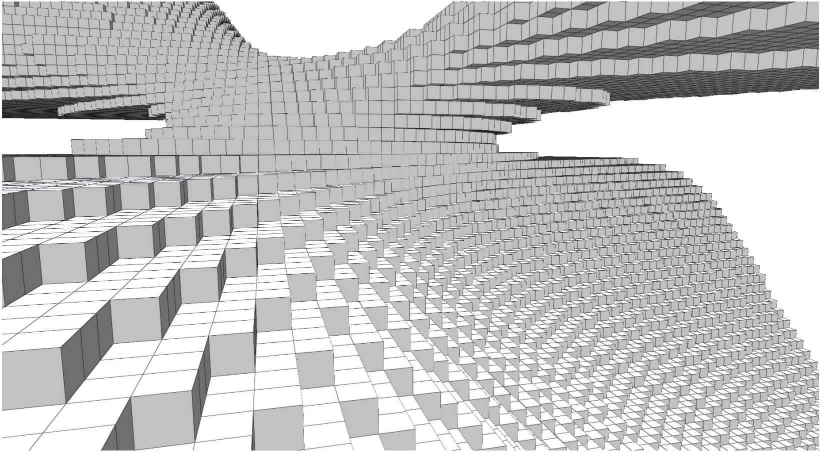
Example: convex shapes



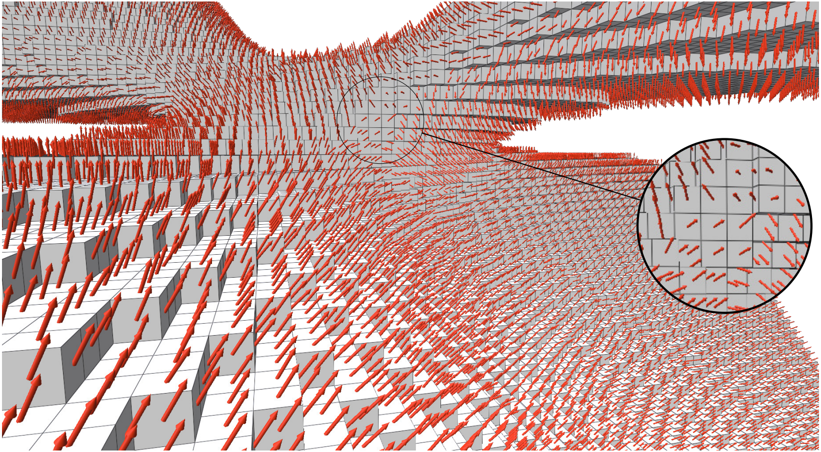
Example: convex shapes



Example: not convex shapes



Example: not convex shapes



Perspectives

Digital planes

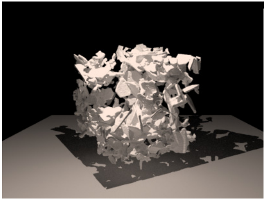
- What piece of digital plane is enough to find N?

Digital surfaces

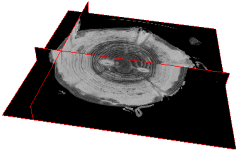
- try all candidates, obtuse triangles may be interesting
- perform a dense probing to process non-convex parts
- estimator: multigrid convergence, experimental comparison
- reconstruction: find of way of gluing triangles together

The end

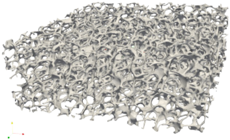
My first answer:



(a) snow



(b) wood



(c) foam