Digital Plane Recognition With Fewer Probes

T. Roussillon[1], J-O. Lachaud[2]

[1]Université de Lyon, INSA Lyon, LIRIS, France

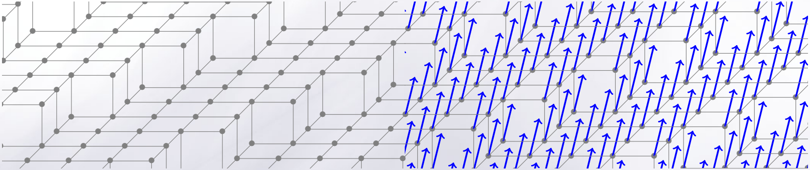
[2] Université Savoie Mont Blanc, LAMA, France

March 28, DGCI'2019

Partly funded by:

- CoMeDiC ANR-15-CE40-0006
- PARADIS ANR-18-CE23-0007-01

Context



Main objective

Parameter-free estimation of normal vectors over a digital surface

Approach

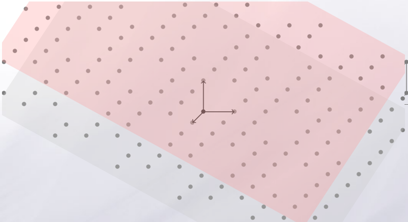
⇒ One need to average things in a small area around each estimate

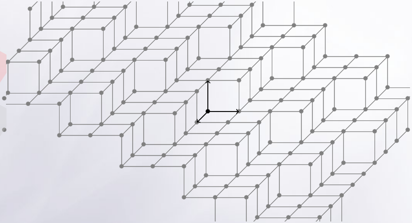
(?) *without specifying the size and shape of the area.*

(-) Existing methods have at least one size parameter (fitting, convolution, integral invariants, variational approaches, ...)

⇒ *Digital plane segments* are able to adapt to the local geometry.

Digital plane and digital plane segment (DPS)





Standard and 6-connected digital plane (segment)

Let $\mathbf{N}(a, b, c)$ be a normal vector $(a, b, c \in \mathbb{Z}, \gcd(a, b, c) = 1)$ and $\mu \in \mathbb{Z}$ be an intercept. A standard digital plane is defined as the set

$$\mathbf{P} = \{x \in \mathbb{Z}^3 | \mu \leq x \cdot \mathbf{N} < \mu + \omega\}.$$

(We assume that $0 < a \leq b \leq c$, $\mu = 0$, $\omega = \|\mathbf{N}\|_1$).
A DPS is any 6-connected subset of a digital plane.

Algorithms for DPS recognition

There exists a lot of recognition algorithms! See, for instance,

E. Charrier and L. Buzer, *An efficient and quasi linear worst-case time algorithm for digital plane recognition*, DGCI'2008, LNCS, vol. 4992, Springer, 2008, pp. 346–357.

I. Debled-Rennesson and J.-P. Reveilles, *An incremental algorithm for digital plane recognition*, DGCI'1994, 1994, pp. 207–222.

Y. Gérard, I. Debled-Rennesson, and P. Zimmermann, *An elementary digital plane recognition algorithm*, Discrete Applied Mathematics 151 (2005), no. 1, 169–183.

C. E. Kim and I. Stojmenović, *On the recognition of digital planes in three-dimensional space*, Pattern Recognition Letters 12 (1991), no. 11, 665–669.

R. Klette and H. J. Sun, *Digital planar segment based polyhedrization for surface area estimation*, Proc. Visual form 2001, LNCS, vol. 2059, Springer, 2001, pp. 356–366.

L. Provot and I. Debled-Rennesson, *3d noisy discrete objects: Segmentation and application to smoothing*, Pattern Recognition 42 (2009), no. 8, 1626–1636.

P. Veelaert, *Digital planarity of rectangular surface segments*, Pattern Analysis and Machine Intelligence, IEEE Transactions on 16 (1994), no. 6, 647–652.

Incremental recognition of DPS for normal estimation

Classical approach: *select-and-decide* algorithms

(?) Select a new point $\mathbf{x}$ and decide if $S \cup \{\mathbf{x}\}$ is still a DPS

(−) A too small DPS does not provide a relevant normal vector

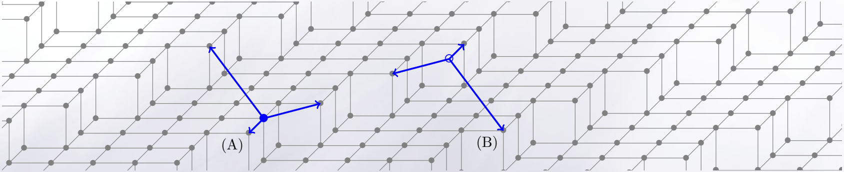
(−) An inextensible DPS may not reveal the local geometry

$\Rightarrow$ They require heuristics with hidden input parameters

Another approach: *plane-probing* algorithms

They probe $\mathbf{P}$ to select $\mathbf{x}$ for us. Parameter-free.

Previous plane-probing algorithms



(A) Upward-oriented frame. No guarantee that it stays near the starting point.

[LPR2016] J-O. Lachaud, X. Provençal, T. R. An output-sensitive algorithm to compute the normal vector of a digital plane. *Theoretical Computer Science*, 624:73–88, 2016.

(B) Downward-oriented frame. The origin is *immutable*.

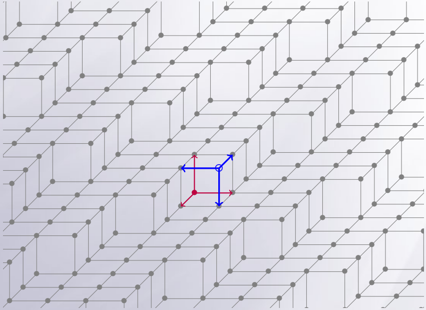
[LPR2017] J-O. Lachaud, X. Provençal, T. R. Two Plane-Probing Algorithms for the Computation of the Normal Vector to a Digital Plane. *Journal of Mathematical Imaging and Vision*, 59(1):23–39, 2017.

- H-algorithm,
- R-algorithm.

A common procedure for both H- and R-algorithm

We are given a predicate $\mathcal{P}$:
"is $\mathbf{x} \in \mathbf{P}$?".

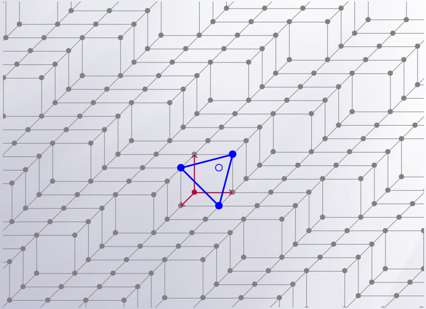
- start with a triangle T in a reentrant corner $\mathbf{N}(T) = (1, 1, 1)$
- update one vertex
- reapeat until $\mathbf{N}(T) = \mathbf{N}$ (for a deep enough corner)



A common procedure for both H- and R-algorithm

We are given a predicate $\mathcal{P}$:
"is $\mathbf{x} \in \mathbf{P}$?".

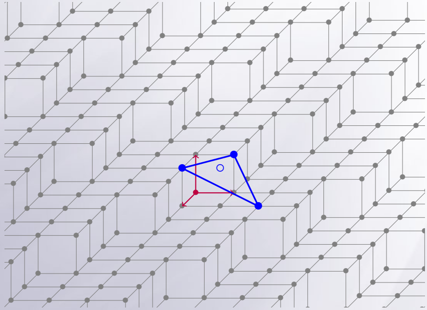
- start with a triangle T in a reentrant corner $\mathbf{N}(T) = (1, 1, 1)$
- update one vertex
- reapeat until $\mathbf{N}(T) = \mathbf{N}$ (for a deep enough corner)



A common procedure for both H- and R-algorithm

We are given a predicate $\mathcal{P}$:
"is $\mathbf{x} \in \mathbf{P}$?".

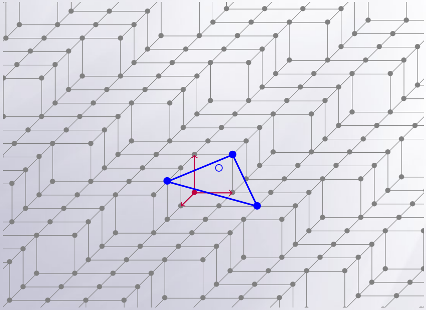
- start with a triangle T in a reentrant corner $\mathbf{N}(T) = (1,1,1)$
- update one vertex
- reapeat until $\mathbf{N}(T) = \mathbf{N}$ (for a deep enough corner)



A common procedure for both H- and R-algorithm

We are given a predicate $\mathcal{P}$:
"is $\mathbf{x} \in \mathbf{P}$?".

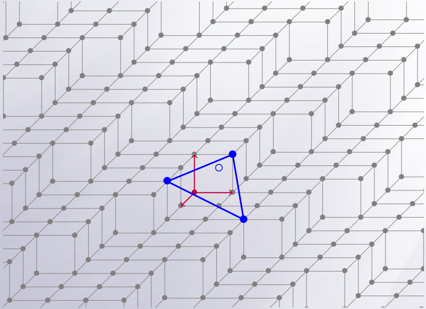
- start with a triangle T in a reentrant corner $\mathbf{N}(T) = (1, 1, 1)$
- update one vertex
- reapeat until $\mathbf{N}(T) = \mathbf{N}$ (for a deep enough corner)



A common procedure for both H- and R-algorithm

We are given a predicate $\mathcal{P}$:
"is $\mathbf{x} \in \mathbf{P}$?".

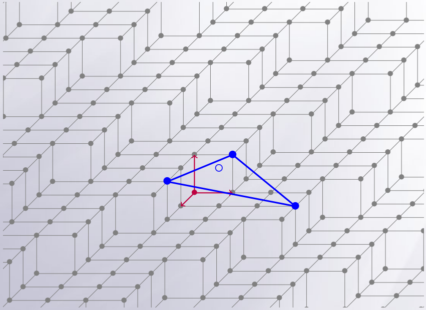
- start with a triangle T in a reentrant corner $\mathbf{N}(T) = (1, 1, 1)$
- update one vertex
- reapeat until $\mathbf{N}(T) = \mathbf{N}$ (for a deep enough corner)



A common procedure for both H- and R-algorithm

We are given a predicate $\mathcal{P}$:
"is $\mathbf{x} \in \mathbf{P}$?".

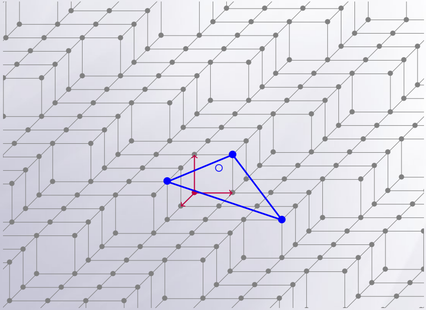
- start with a triangle T in a reentrant corner $\mathbf{N}(T) = (1,1,1)$
- update one vertex
- reapeat until $\mathbf{N}(T) = \mathbf{N}$ (for a deep enough corner)



A common procedure for both H- and R-algorithm

We are given a predicate $\mathcal{P}$:
"is $\mathbf{x} \in \mathbf{P}$?".

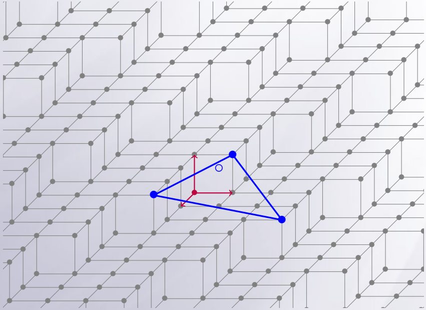
- start with a triangle T in a reentrant corner $\mathbf{N}(T) = (1,1,1)$
- update one vertex
- reapeat until $\mathbf{N}(T) = \mathbf{N}$ (for a deep enough corner)



A common procedure for both H- and R-algorithm

We are given a predicate $\mathcal{P}$:
"is $\mathbf{x} \in \mathbf{P}$?".

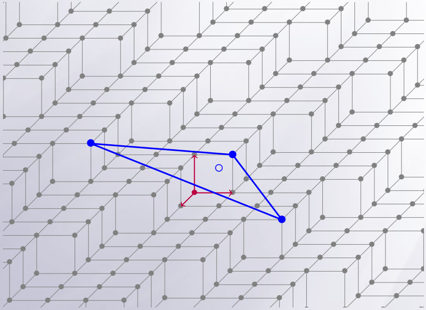
- start with a triangle T in a reentrant corner $\mathbf{N}(T) = (1,1,1)$
- update one vertex
- reapeat until $\mathbf{N}(T) = \mathbf{N}$ (for a deep enough corner)



A common procedure for both H- and R-algorithm

We are given a predicate $\mathcal{P}$:
"is $\mathbf{x} \in \mathbf{P}$?".

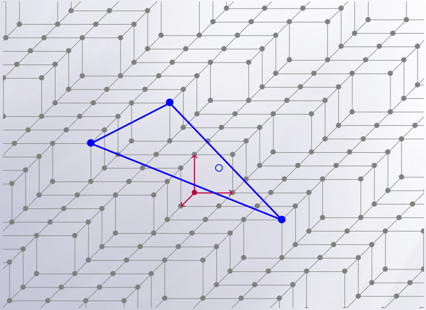
- start with a triangle T in a reentrant corner $\mathbf{N}(T) = (1, 1, 1)$
- update one vertex
- reapeat until $\mathbf{N}(T) = \mathbf{N}$ (for a deep enough corner)



A common procedure for both H- and R-algorithm

We are given a predicate $\mathcal{P}$: "is $\mathbf{x} \in \mathbf{P}$?".

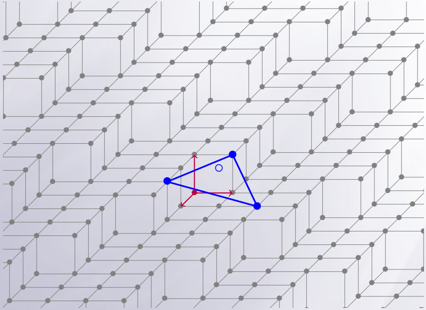
- start with a triangle T in a reentrant corner $\mathbf{N}(T) = (1, 1, 1)$
- update one vertex
- reapeat until $\mathbf{N}(T) = \mathbf{N}$ (for a deep enough corner)



Update procedure

At a given step:

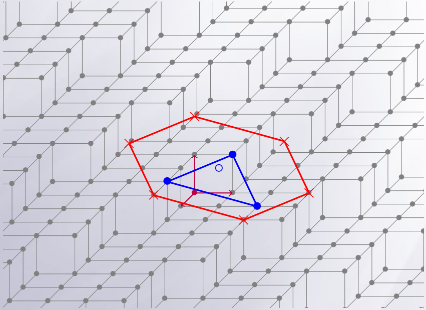
- consider a candidate set S
- filter S through $\mathcal{P}$
- select a *closest* point $s^{\star}$: the circumsphere of $T \cup s^{\star}$ doesn't contain any other
- update T with this point



Update procedure

At a given step:

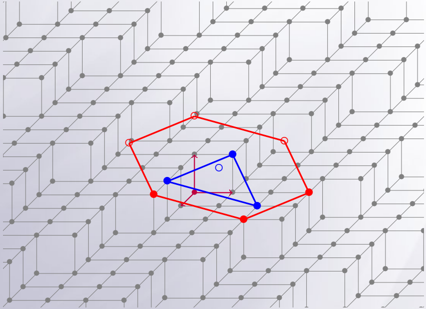
- consider a candidate set S
- filter S through $\mathcal{P}$
- select a *closest* point $s^\star$: the circumsphere of $T \cup s^\star$ doesn't contain any other
- update T with this point



Update procedure

At a given step:

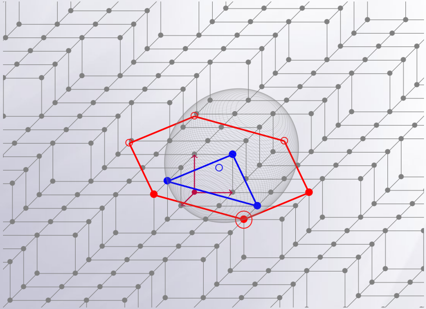
- consider a candidate set S
- filter S through $\mathcal{P}$
- select a *closest* point $s^\star$: the circumsphere of $T \cup s^\star$ doesn't contain any other
- update T with this point



Update procedure

At a given step:

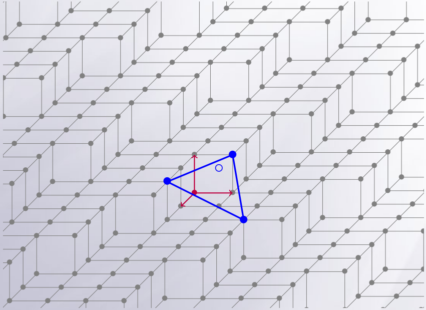
- consider a candidate set S
- filter S through $\mathcal{P}$
- select a *closest* point $s^\star$: the circumsphere of $T \cup s^\star$ doesn't contain any other
- update T with this point



Update procedure

At a given step:

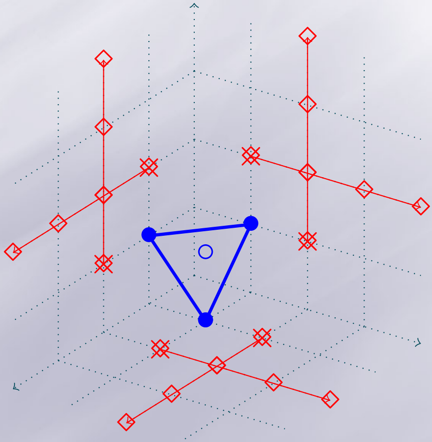
- consider a candidate set S
- filter S through $\mathcal{P}$
- select a *closest* point $s^\star$: the circumsphere of $T \cup s^\star$ doesn't contain any other
- update T with this point



Difference between H- and R-algorithm

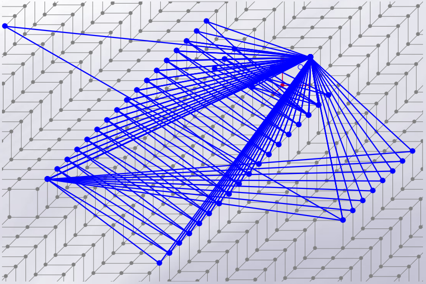
Each algorithm considers a distinct candidate set:

- S_H ($\times$): 6 Hexagon vertices
- S_R ($\diamond$): 6 Rays (which are infinite)

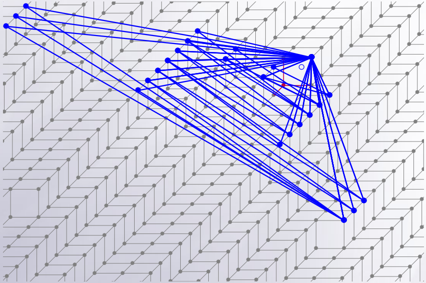


The R-algorithm experimentally requires a smaller area

H-algorithm



R-algorithm



$\mathbf{N}(1, 73, 100)$

Main features of the R-algorithm

R-algorithm

- starts with a triangle of normal $(1, 1, 1)$ in a corner
- updates the current triangle by one geometrical operation
- using only the predicate $\mathcal{P}$: “is $\mathbf{x} \in \mathbf{P}$?”
- reaches $\mathbf{N}$, the normal of $\mathbf{P}$ (if the corner is deep enough)
- triangles stay around the starting corner “within a small area”
- $O(\omega \log \omega)$ calls to $\mathcal{P}$

Contribution and outline

R^1-algorithm

- has the same output as the R-algorithm
- but keeps only 1 ray and 1 point over 6 rays at each step
- $O(\omega)$ calls to $\mathcal{P}$ (tight upper bound), instead of $O(\omega \log \omega)$

Outline

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

1. Local probing

Tip:  and  are impossible on digital planes.

Switch on card($S_H \cap \mathbf{P}$):

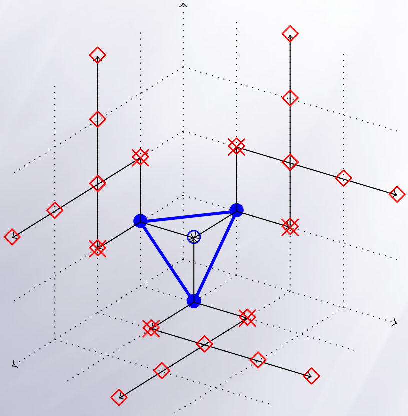
(0) stop

(1) unique candidate, trivial

(2) (e) select closest, trivial

(v) 2 rays...

(3) 2 rays and a point...



1. Local probing

Tip:  and  are impossible on digital planes.

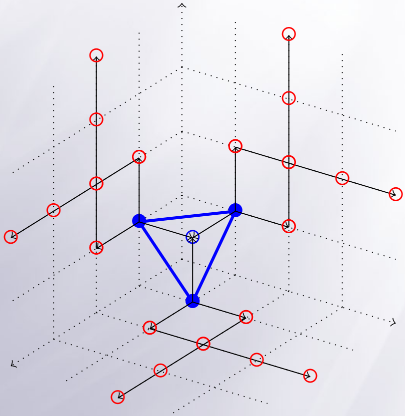
Switch on card$(S_H \cap \mathbf{P})$:

(0) stop

(1) unique candidate, trivial

(2) (e) select closest, trivial
    (v) 2 rays...

(3) 2 rays and a point...



1. Local probing

Tip:  and  are impossible on digital planes.

Switch on card($S_H \cap \mathbf{P}$):

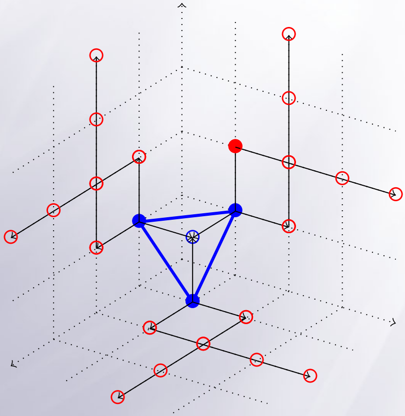
(0) stop

(1) unique candidate, trivial

(2) (e) select closest, trivial

(v) 2 rays...

(3) 2 rays and a point...



1. Local probing

Tip:  and  are impossible on digital planes.

Switch on card$(S_H \cap \mathbf{P})$:

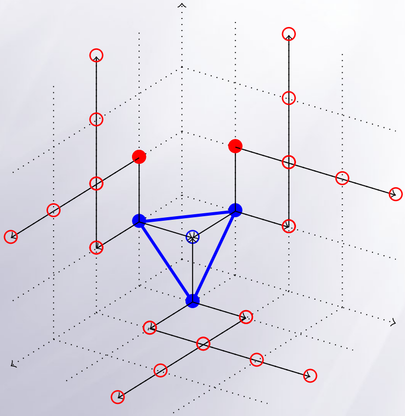
(0) stop

(1) unique candidate, trivial

(2) (e) select closest, trivial

(v) 2 rays...

(3) 2 rays and a point...



1. Local probing

Tip:  and  are impossible on digital planes.

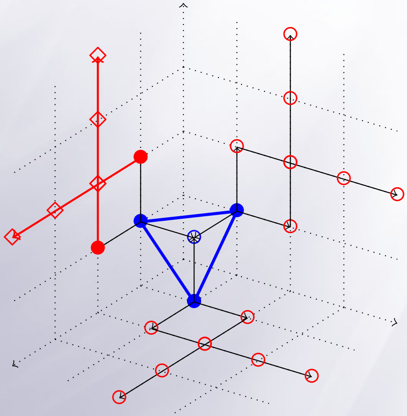
Switch on card($S_H \cap \mathbf{P}$):

(0) stop

(1) unique candidate, trivial

(2) (e) select closest, trivial
    (v) 2 rays...

(3) 2 rays and a point...



1. Local probing

Tip:  and  are impossible on digital planes.

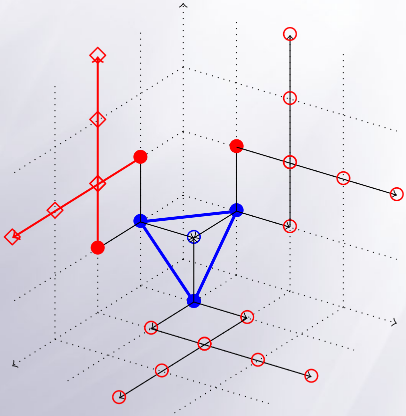
Switch on card$(S_H \cap \mathbf{P})$:

(0) stop

(1) unique candidate, trivial

(2) (e) select closest, trivial
    (v) 2 rays...

(3) 2 rays and a point...



2. Geometrical study (**acute case**)

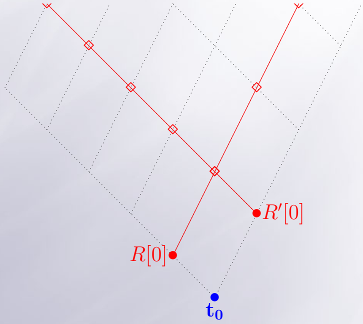
$R[i]$ is the i-th point on ray R.

Lemma

Either $R[0]$ or $R'[0]$ is closest.

Proof (sketch)

The sphere passing by T(and so t_0) and $R'[i+1]$ contains either $R'[i]$ or $R[0]$ (or both), i.e. another candidate point.



2. Geometrical study (**acute case**)

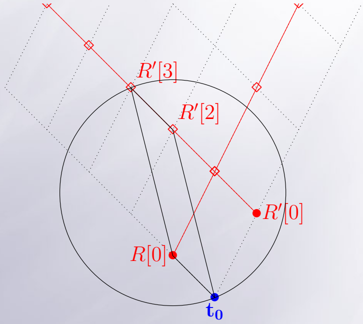
$R[i]$ is the i-th point on ray R.

Lemma

Either $R[0]$ or $R'[0]$ is closest.

Proof (sketch)

The sphere passing by T(and so $\mathbf{t_0}$) and $R'[i+1]$ contains either $R'[i]$ or $R[0]$ (or both), i.e. another candidate point.



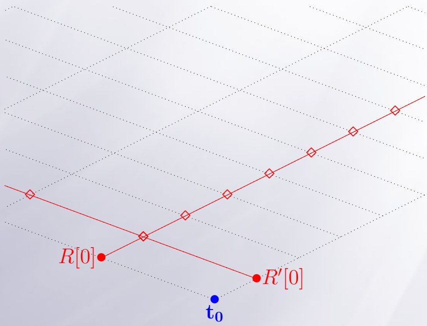
2. Geometrical study (**obtuse case**)

Theorem

A closest point is either in $R \cup \{R'[0]\}$ or in $R' \cup \{R[0]\}$.

Proof (sketch)

- we cut rays through their common point
- on one side, we are in the *acute* case and use the previous result



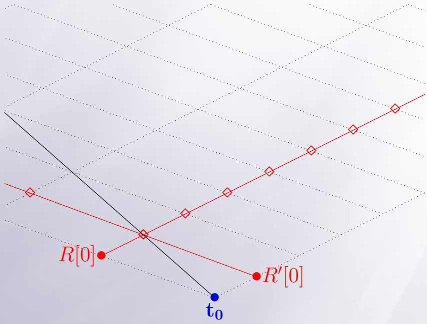
2. Geometrical study (**obtuse case**)

Theorem

A closest point is either in $R \cup \{R'[0]\}$ or in $R' \cup \{R[0]\}$.

Proof (sketch)

- we cut rays through their common point
- on one side, we are in the *acute* case and use the previous result



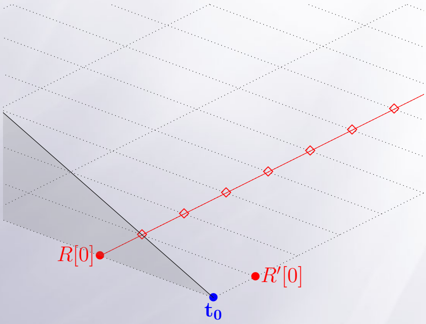
2. Geometrical study (**obtuse case**)

Theorem

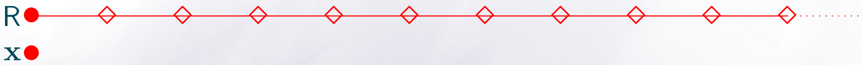
A closest point is either in $R \cup \{R'[0]\}$ or in $R' \cup \{R[0]\}$.

Proof (sketch)

- we cut rays through their common point
- on one side, we are in the *acute* case and use the previous result



3. Efficient algorithm for 1 ray and 1 point



$\Rightarrow$ $\mathcal{S} \leftarrow$ sphere circumscribing $\mathrm{T} \cup \{\mathbf{x}\}$;

$(i, j) \leftarrow$ intersection($\mathcal{S}$, R) ;

`// R[`k`] closer than x iff `$k \in [i, j]$

if $\neg\mathcal{P}(\mathrm{R}[i])$ **then** **return** $\mathbf{x}$;

else

  $k \leftarrow$ closestOnRay(T, R) ;

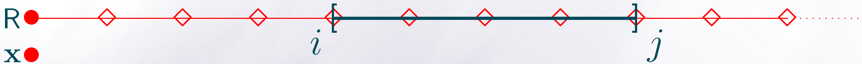
  if $k \notin [i, j]$ **then** **return** $\mathbf{x}$;

  else $k \in [i, j]$

    if $\mathcal{P}(\mathrm{R}[k])$ **then** **return** R[k] ;

    else **return** findLast($\mathcal{P}$, R, i, k) ;

3. Efficient algorithm for 1 ray and 1 point



$\mathcal{S} \leftarrow$ sphere circumscribing T$\cup\{\mathbf{x}\}$;

$\Rightarrow$ $(i, j) \leftarrow$ intersection($\mathcal{S}$, R) ;

// R[k] closer than x iff $k \in [i, j]$

if $\neg\mathcal{P}$(R[i]) **then** **return** $\mathbf{x}$;

else

  $k \leftarrow$ closestOnRay(T, R) ;

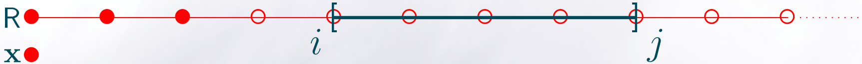
  if $k \notin [i, j]$ **then** **return** $\mathbf{x}$;

  else $k \in [i, j]$

    if $\mathcal{P}$(R[k]) **then** **return** R[k] ;

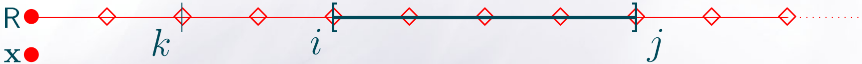
    else **return** findLast($\mathcal{P}$, R, i, k) ;

3. Efficient algorithm for 1 ray and 1 point



$\mathcal{S} \leftarrow$ sphere circumscribing $\mathrm{T} \cup \{\mathbf{x}\}$;
$(i, j) \leftarrow$ intersection($\mathcal{S}$, R) ;
`// R[k] closer than x iff` $k \in [i, j]$
$\Rightarrow$ **if** $\neg\mathcal{P}(\mathrm{R}[i])$ **then** **return** $\mathbf{x}$;
else
  $k \leftarrow$ closestOnRay(T, R) ;
  if $k \notin [i, j]$ **then** **return** $\mathbf{x}$;
  else $k \in [i, j]$
    if $\mathcal{P}(\mathrm{R}[k])$ **then** **return** R[k] ;
    else **return** findLast($\mathcal{P}$, R, i, k) ;

3. Efficient algorithm for 1 ray and 1 point




```
S ← sphere circumscribing T∪{x} ;
(i, j) ← intersection(S, R) ;
// R[k] closer than x iff k ∈ [i, j]
if ¬P(R[i]) then return x;
else
    k ← closestOnRay(T, R) ;
⇒   if k ∉ [i, j] then return x;
    else k ∈ [i, j]
        if P(R[k]) then return R[k] ;
        else return findLast(P, R, i, k) ;
```

3. Efficient algorithm for 1 ray and 1 point

$\mathcal{S} \leftarrow$ sphere circumscribing T$\cup\{\mathbf{x}\}$;
$(i, j) \leftarrow$ intersection($\mathcal{S}$, R) ;
// R[k] closer than x iff $k \in [i, j]$
if $\neg\mathcal{P}(\mathrm{R}[i])$ **then return** x;
else
  $k \leftarrow$ closestOnRay(T, R) ;
  if $k \notin [i, j]$ **then return** x;
  else $k \in [i, j]$
⇒   **if** $\mathcal{P}(\mathrm{R}[k])$ **then return** R[k] ;
    else return findLast($\mathcal{P}$, R, i, k) ;

3. Efficient algorithm for 1 ray and 1 point


```
S ← sphere circumscribing T∪{x} ;
(i, j) ← intersection(S, R) ;
// R[k] closer than x iff k ∈ [i, j]
if ¬P(R[i]) then return x;
else
    k ← closestOnRay(T, R) ;
    if k ∉ [i, j] then return x;
    else k ∈ [i, j]
        if P(R[k]) then return R[k] ;
⇒       else return findLast(P, R, i, k) ;
```

Summary

Update

step	calls to $\mathcal{P}$	arithmetical operations	$\sqrt{\cdot}, \lfloor\cdot\rfloor$
1. local probing	6	$O(1)$	0
2. geometrical study	0	$O(1)$	0
3. final algorithm	1 or 2 most often, exceptionnally more	$O(1)$	1 or 2

Complexity and experimental results

Overall complexity

- $O(\omega)$ calls to $\mathcal{P}$
- tight upper bound (see, for instance, $\mathbf{N}(1,1,r), \forall r \in \{1,2,\ldots\}$)
- lower on average: $O(\log(\omega))$ updates and 6-8 calls per update

Experimental comparison

6.578.833 digital planes whose normal vector is ranging from (1,1,1) to (200, 200, 200) (with relatively prime components).

	calls to $\mathcal{P}$		
	(per update)		(total)
alg.	avg.	max.	avg.
R	14.49	25	254.95
R^1	7.06	14	122.36

Conclusion and perspectives

R^1-algorithm

- has the same output as the R-algorithm
- but keeps only 1 ray and 1 point at each step
- $O(\omega)$ calls to $\mathcal{P}$ (instead of $O(\omega \log \omega)$ for the R-algorithm)
- far fewer calls in practice

Perspectives in the context of PARADIS research project

- short-term: bound the area required by the algorithm
- mid-term: plane-probing algorithms for digital surface analysis
- 1 Ph.D. position ($\geq$ September), applications are welcome !

Thank you for your attention