Digital Plane Recognition With Fewer Probes

T. Roussillon\textsuperscript{1}, J-O. Lachaud\textsuperscript{2}

\textsuperscript{1}Université de Lyon, INSA Lyon, LIRIS, France
\textsuperscript{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
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

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

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

Incremental recognition of DPS for normal estimation

Classical approach: *select-and-decide* algorithms

(?) Select a new point $x$ and decide if $S \cup \{x\}$ is still a DPS
(−) A too small DPS does not provide a relevant normal vector
(−) An inextensible DPS may not reveal the local geometry
⇒ They require heuristics with hidden input parameters

Another approach: *plane-probing* algorithms

They probe $P$ to select $x$ for us. Parameter-free.
Previous plane-probing algorithms

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


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


- H-algorithm,
- R-algorithm.
A common procedure for both H- and R-algorithm

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

- start with a triangle $T$
  - in a reentrant corner $N(T) = (1, 1, 1)$
- update one vertex
- repeat until $N(T) = N$ (for a deep enough corner)
A common procedure for both H- and R-algorithm

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

- start with a triangle $T$
  in a reentrant corner
  $N(T) = (1, 1, 1)$
- update one vertex
- repeat until $N(T) = N$
  (for a deep enough corner)
A common procedure for both H- and R-algorithm

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

- start with a triangle $T$
  in a reentrant corner
  $\mathbf{N}(T) = (1, 1, 1)$
- update one vertex
- repeat 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 $x \in \mathbb{P}$?".

- start with a triangle $T$
in a reentrant corner
$N(T) = (1, 1, 1)$
- update one vertex
- repeat until $N(T) = N$
(for a deep enough corner)
We are given a predicate $\mathcal{P}$: “is $x \in \mathbf{P}$?”.

- start with a triangle $T$
- in a reentrant corner $\mathbf{N}(T) = (1, 1, 1)$
- update one vertex
- repeat 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 $x \in \mathcal{P}$?”.

- start with a triangle $T$
  in a reentrant corner
  $\mathbf{N}(T) = (1, 1, 1)$
- update one vertex
- repeat 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 $P$: “is $x \in P$?”.

- start with a triangle $T$
  in a reentrant corner
  $N(T) = (1, 1, 1)$
- update one vertex
- reapeat until $N(T) = N$
  (for a deep enough corner)
We are given a predicate $P$: “is $x \in P$?”.

- start with a triangle $T$
- in a reentrant corner
- $N(T) = (1, 1, 1)$
- update one vertex
- repeat until $N(T) = N$
  (for a deep enough corner)
A common procedure for both H- and R-algorithm

We are given a predicate $P$: “is $x \in P$?”.

- start with a triangle $T$
  in a reentrant corner
  $N(T) = (1, 1, 1)$
- update one vertex
- repeat until $N(T) = N$
  (for a deep enough corner)
A common procedure for both H- and R-algorithm

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

- start with a triangle $T$
  in a reentrant corner
  $N(T) = (1, 1, 1)$
- update one vertex
- repeat until $N(T) = N$
  (for a deep enough corner)
Update procedure

At a given step:
- consider a candidate set $S$
- filter $S$ through $P$
- select a closest point $s^*$: the circumsphere of $T \cup s^*$ 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 \( s^* \):
  - the circumsphere of \( T \cup s^* \)
    doesn’t contain any other
- update \( T \) with this point
Update procedure

At a given step:

1. consider a candidate set $S$
2. filter $S$ through $\mathcal{P}$
3. select a closest point $s^*$:
   * the circumsphere of $T \cup s^*$
   * doesn’t contain any other
4. 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 $s^*$: the circumsphere of $T \cup s^*$ doesn’t contain any other
- update $T$ with this point
Update procedure

At a given step:
- consider a candidate set $S$
- filter $S$ through $P$
- select a closest point $s^*$: the circumsphere of $T \cup s^*$ doesn’t contain any other
- update $T$ with this point
Difference between H- and R-algorithm

Each algorithm considers a distinct candidate set:

\[ S_H (\times) : 6 \text{ Hexagon vertices} \]
\[ S_R (\diamond) : 6 \text{ Rays (which are infinite)} \]
The R-algorithm experimentally requires a smaller area

H-algorithm

\[ N(1, 73, 100) \]

R-algorithm
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 \(\mathcal{P}: \text{is } x \in \mathcal{P}\)?
- reaches \(N\), the normal of \(\mathcal{P}\) (if the corner is deep enough)
- triangles stay around the starting corner “within a small area”
- \(O(\omega \log \omega)\) calls to \(\mathcal{P}\)
Motivation

Plane-probing algorithms

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 \( P \) (tight upper bound), instead of \( O(\omega \log \omega) \)

Outline

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

Plane-probing algorithms

Contribution

1. Local probing

Tip: and are impossible on digital planes.

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

(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($S_H \cap P$):

(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 \( \text{card}(S_H \cap P) \):

(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 \( \text{card}(S_H \cap P) \):

(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($S_H \cap P$):

(0) stop
(1) unique candidate, trivial
(2) (e) select closest, trivial
(3) 2 rays...
1. Local probing

Tip: and are impossible on digital planes.

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

(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 $t_0$) 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

\[ S \leftarrow \text{sphere circumscribing } T \cup \{x\} ; \]
\[ (i, j) \leftarrow \text{intersection}(S, R) ; \]
\[ \quad \text{// } R[k] \text{ closer than } x \text{ iff } k \in [i, j] \]
\[ \text{if } \neg \mathcal{P}(R[i]) \text{ then return } x; \]
\[ \text{else} \]
\[ k \leftarrow \text{closestOnRay}(T, R) ; \]
\[ \text{if } k \notin [i, j] \text{ then return } x; \]
\[ \text{else } k \in [i, j] \]
\[ \text{if } \mathcal{P}(R[k]) \text{ then return } R[k] ; \]
\[ \text{else return findLast}(\mathcal{P}, R, i, k) ; \]
3. Efficient algorithm for 1 ray and 1 point

\[ S \leftarrow \text{sphere circumscribing } T \cup \{x\} ; \]
\[ (i, j) \leftarrow \text{intersection}(S, R) ; \]
\[ // R[k] \text{ closer than } x \text{ iff } k \in [i, j] \]
\[ \text{if } \neg \mathcal{P}(R[i]) \text{ then return } x; \]
\[ \text{else} \]
\[ k \leftarrow \text{closestOnRay}(T, R) ; \]
\[ \text{if } k \notin [i, j] \text{ then return } x; \]
\[ \text{else } k \in [i, j] \]
\[ \text{if } \mathcal{P}(R[k]) \text{ then return } R[k] ; \]
\[ \text{else return findLast(} \mathcal{P}, R, i, k) ; \]
3. Efficient algorithm for 1 ray and 1 point

1. $S \leftarrow$ sphere circumscribing $T \cup \{x\}$ ;
2. $(i, j) \leftarrow$ intersection($S, R$) ;
   
   \[// R[k] closer than x iff k \in [i, j]\]
   
   $\Rightarrow$ 3 if $\neg P(R[i])$ then return $x$;
4. else
5.     $k \leftarrow$ closestOnRay($T, R$) ;
6.     if $k \notin [i, j]$ then return $x$;
7.     else $k \in [i, j]$
8.     if $P(R[k])$ then return $R[k]$ ;
9.     else return findLast($P, R, i, k$) ;
3. Efficient algorithm for 1 ray and 1 point

1. $S \leftarrow$ sphere circumscribing $T \cup \{x\}$;
2. $(i, j) \leftarrow$ intersection($S$, $R$);
   $\quad$// $R[k]$ closer than $x$ iff $k \in [i, j]$;
3. if $\neg \mathcal{P}(R[i])$ then return $x$;
4. else
   5. $k \leftarrow$ closestOnRay($T$, $R$);
   6. if $k \notin [i, j]$ then return $x$;
   7. else $k \in [i, j]$
   8. if $\mathcal{P}(R[k])$ then return $R[k]$;
   9. else return findLast($\mathcal{P}$, $R$, $i$, $k$);
3. Efficient algorithm for 1 ray and 1 point

$S \leftarrow$ sphere circumscribing $T \cup \{x\}$;
$(i, j) \leftarrow$ intersection($S$, $R$);
// $R[k]$ closer than $x$ iff $k \in [i, j]$
if $\neg P(R[i])$ then return $x$;
else
$k \leftarrow$ closestOnRay($T$, $R$);
if $k \notin [i, j]$ then return $x$;
else $k \in [i, j]$
if $P(R[k])$ then return $R[k]$;
else return findLast($P$, $R$, $i$, $k$);
3. Efficient algorithm for 1 ray and 1 point

1. \[ S \leftarrow \text{sphere circumscribing } T \cup \{x\} \ ; \]
2. \[ (i, j) \leftarrow \text{intersection}(S, R) ; \]
   \[ // \ R[k] \text{ closer than } x \iff k \in [i, j] \]
3. \[ \text{if } \neg \mathcal{P}(R[i]) \text{ then return } x; \]
4. \[ \text{else } \]
5. \[ k \leftarrow \text{closestOnRay}(T, R) ; \]
6. \[ \text{if } k \notin [i, j] \text{ then return } x; \]
7. \[ \text{else } k \in [i, j] \]
8. \[ \text{if } \mathcal{P}(R[k]) \text{ then return } R[k] ; \]
9. \[ \text{else return } \text{findLast}(\mathcal{P}, R, i, k) ; \]
### Update

<table>
<thead>
<tr>
<th>step</th>
<th>calls to $P$</th>
<th>arithmetical operations</th>
<th>$\sqrt{\cdot}, \lfloor \cdot \rfloor$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. local probing</td>
<td>6</td>
<td>$O(1)$</td>
<td>0</td>
</tr>
<tr>
<td>2. geometrical study</td>
<td>0</td>
<td>$O(1)$</td>
<td>0</td>
</tr>
<tr>
<td>3. final algorithm</td>
<td>1 or 2 most often, exceptionnally more</td>
<td>$O(1)$</td>
<td>1 or 2</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th></th>
<th>calls to $\mathcal{P}$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(per update)</td>
<td>(total)</td>
</tr>
<tr>
<td>alg.</td>
<td>avg.</td>
<td>max.</td>
</tr>
<tr>
<td>$R$</td>
<td>14.49</td>
<td>25</td>
</tr>
<tr>
<td>$R^1$</td>
<td>7.06</td>
<td>14</td>
</tr>
</tbody>
</table>

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