A QUASI-LINEAR ALGORITHM TO COMPUTE THE TREE OF SHAPES OF $n\mathbf{D}$ IMAGES

Thierry Géraud$^1$, Edwin Carlinet$^1$, Sébastien Crozet$^1$, and Laurent Najman$^2$

$^1$ EPITA Research and Development Laboratory (LRDE), France

$^2$ Université Paris-Est, LIGM, A3SI, ESIEE, France

ISMM, Uppsala, May 2013
Given a nD image \( u : \mathbb{Z}^n \rightarrow \mathbb{Z} \),

- lower cuts: \([u < \lambda] = \{x \in X \mid u(x) < \lambda\}\)
- upper cuts: \([u \geq \lambda] = \{x \in X \mid u(x) \geq \lambda\}\)
**DUAL TREES**

**TWO SETS OF CUTS/THRESHOLDS**

Given a $n$D image $u : \mathbb{Z}^n \rightarrow \mathbb{Z}$,

- lower cuts: $[u < \lambda] = \{ x \in X \mid u(x) < \lambda \}$
- upper cuts: $[u \geq \lambda] = \{ x \in X \mid u(x) \geq \lambda \}$

**A COUPLE OF DUAL TREES**

\[ \leadsto \text{min-tree: } T_\leq(u) = \{ \Gamma \in CC_{c_2n}([u < \lambda]) \} \lambda \]

\[ \leadsto \text{max-tree: } T_\geq(u) = \{ \Gamma \in CC_{c_3n - 1}([u \geq \lambda]) \} \lambda \]
A Schematic Example

image

max-tree

min-tree

T. Géraud et al. (LRDE) Quasi-linear Tree of Shapes Computation ISMM, Uppsala, May 2013
DEALING WITH TREES

SOME NICE PROPERTIES

- a tree is a versatile structure
- it is straightforward to deal with them
- and dual trees are easy to compute
DEALING WITH TREES

SOME NICE PROPERTIES

- a tree is a versatile structure
- it is straightforward to deal with them
- and dual trees are easy to compute

CONNECTED OPERATORS

Those operators are powerful:

- they filter (simplify) images while preserving (i.e., do not shift) contours
- some of them are based on component trees

Examples:

- algebraic openings and closings
- levelings
- ...
DUALITY V. SELF-DUALITY

Different ideas:
Duality v. self-duality

Different ideas:

- we can try to be self-dual with two trees, yet...
  - we get some information redundancy between those trees
  - we have to juggle with two structures
DUALITY V. SELF-DUALITY

Different ideas:

- we can try to be self-dual with two trees, yet...
  - we get some information redundancy between those trees
  - we have to juggle with two structures

- self-duality is nice
  - when we cannot make any assumption about contrast
  - when we do not want to make such an assumption
    
    *(the “object-background” paradigm is limited...)*
Different ideas:

- we can try to be self-dual with two trees, yet...
  - we get some information redundancy between those trees
  - we have to juggle with two structures

- self-duality is nice
  - when we cannot make any assumption about contrast
  - when we do not want to make such an assumption

  *(the “object-background” paradigm is limited...)*

- “object contours belong to the set of level lines”
  - so there are many applications...
With the cavity-fill-in operator Sat:

\[ S_< (u) = \{ \text{Sat}_{c_{3n-1}} (\Gamma); \ \Gamma \in T_< (u) \} \]

\[ S_> (u) = \{ \text{Sat}_{c_{2n}} (\Gamma); \ \Gamma \in T_> (u) \} \]
ONE SELF-DUAL TREE

SHAPES

With the cavity-fill-in operator Sat:

\[ S_<(u) = \{ \text{Sat}_{c_{3^n-1}}(\Gamma); \ \Gamma \in T_<(u) \} \]
\[ S_\geq(u) = \{ \text{Sat}_{c_{2^n}}(\Gamma); \ \Gamma \in T_\geq(u) \} \]

A SELF-DUAL TREE

\[ \rightsquigarrow \text{tree of shapes: } \mathcal{S}(u) = S_< (u) \cup S_\geq(u) \]
ONE SELF-DUAL TREE

SHAPES

With the cavity-fill-in operator Sat:

\[ S_<(u) = \{ \text{Sat}_{c_3^{n-1}}(\Gamma); \ \Gamma \in T_<(u) \} \]
\[ S_(u) = \{ \text{Sat}_{c_2^n}(\Gamma); \ \Gamma \in T_(u) \} \]

A SELF-DUAL TREE

\[ \rightsquigarrow \text{tree of shapes: } \mathcal{G}(u) = S_<(u) \cup S_(u) \]

PROPERTY

we have \[ \mathcal{G}(-u) = \mathcal{G}(u) \] (whereas \[ T_(u) = T_<(u) \])
Schematic Example

Image:

Tree of shapes:

O

A

B

C

D

E

F

O

A

B

C

D

E

F
Alt. Definitions of shapes

- the cavities of upper and lower cuts
- the interior regions of level lines.
Applications 1/2

Grain filter ($\approx (\phi \gamma + \gamma \phi)/2$)

Object detection

Contour saliency $\rightarrow$ extinction $\rightarrow$ (hierarchical) segmentation
APPLICATIONS 2/2

image simplification

morphological “shaping”

local feature detection
State of the art: 3 different algorithms to compute the tree of shapes
State of the art: **3 different algorithms** to compute the tree of shapes

- **Monasse & Guichard,**
  

- **Song,**
  

- **Caselles & Monasse,**
  
Now, how to...

State of the art: 3 different algorithms to compute the tree of shapes

- Monasse & Guichard,

- Song,

- Caselles & Monasse,

Several issues:

- their time complexity is \(O(N^2)\)...
  \(\leftarrow\) that is bad

- they are unusable for \(n\)D images (limited to 2D images)
  \(\leftarrow\) that is a pity

- they are hard to implement

  \(\rightsquigarrow\) so eventually they are merely used!
A TWO-PASS ALGORITHM

To compute the max-tree or the min-tree, we have a two-pass algorithm:

- Najman & Couprie, *Quasi-Linear Algorithm for the Component Tree*, IS&T/SPIE Symp. on E.I., 2004
- Berger et al., *Effective Component Tree Computation with App. to Pat.Rec. in Astro. Imaging*, ICIP, 2007
To compute the max-tree or the min-tree, we have a two-pass algorithm:

- **Najman & Couprie**, *Quasi-Linear Algorithm for the Component Tree*, IS&T/SPIE Symp. on E.I., 2004
- **Berger et al.**, *Effective Component Tree Computation with App. to Pat.Rec. in Astro. Imaging*, ICIP, 2007
- **Carlinet & me**, *A Comparison of Many Max-Tree Computation Algorithms*, ISMM, 2013

**The Two Steps**

1. sort the pixels in the *descending tree order*
2. in the *reverse order*, rely on the Union-Find algorithm to compute the tree
A TWO-PASS ALGORITHM

To compute the max-tree or the min-tree, we have a two-pass algorithm:

- Najman & Couprie, *Quasi-Linear Algorithm for the Component Tree*, IS&T/SPIE Symp. on E.I., 2004
- Berger et al., *Effective Component Tree Computation with App. to Pat.Rec. in Astro. Imaging*, ICIP, 2007

THE TWO STEPS

1. sort the pixels in the *descending tree order*
2. in the *reverse order*, rely on the Union-Find algorithm to compute the tree

Algorithm properties:

- quasi-linear time complexity for low quantized data
- works on $n$D images
- very easy to implement
First key idea

if we succeed in sorting the pixels such as descending the tree of shapes, then we have a simple and efficient algorithm.
COMPUTING THE TREE OF SHAPES: 2ND PASS

sort : O A B C D E F

Now let us browse the pixels in the reserve sort order...
Computing the tree of shapes: 2nd pass
Computing the tree of shapes: 2nd pass

D E F

O A B C D E F

D E F

D E F

T. Géraud et al. (LRDE) | Quasi-linear tree of shapes computation | ISMM, Uppsala, May 2013
Computing the tree of shapes: 2nd pass

O A B C D E F

D E

C

D E F

C

D E F
COMPUTING THE TREE OF SHAPES: 2ND PASS
COMPUTING THE TREE OF SHAPES: 2ND PASS
Done for the tree construction (2nd pass)!
Done for the tree construction (2nd pass)!

Now we have to know **how to sort pixels** (1st pass).
Towards the 2nd key idea

About the 1st pass

sorting the pixels means progress “continuously”
both in image space\(^1\) and in value space\(^2\)
(starting from the image boundary, i.e., the root node)
Towards the 2nd Key Idea

About the 1st Pass

sorting the pixels means progress “continuously” both in image space\(^1\) and in value space\(^2\)

(starting from the image boundary, i.e., the root node)

\(^1\) through a spatially consistent growing
Towards the 2nd key idea

About the 1st pass

sorting the pixels means progress “continuously”
both in image space\(^1\) and in value space\(^2\)
(starting from the image boundary, i.e., the root node)

\(^1\) through a spatially consistent growing
\(^2\) jumping from a gray level to the “next” one (either upper or lower)
Towards the 2nd key idea

About the 1st pass

sorting the pixels means progress “continuously” both in image space\(^1\) and in value space\(^2\)

(starting from the image boundary, i.e., the root node)

\(^1\) through a spatially consistent growing

\(^2\) jumping from a gray level to the “next” one (either upper or lower)

⇒ we can use a propagation front and a hierarchical queue
Towards the 2nd key idea

About the 1st pass

Sorting the pixels means progress “continuously” both in image space\(^1\) and in value space\(^2\) (starting from the image boundary, i.e., the root node).

\(^1\) through a spatially consistent growing
\(^2\) jumping from a gray level to the “next” one (either upper or lower)

⇒ we can use a propagation front and a hierarchical queue

That seems easy... (unfortunately we’ll see that is not!)
To sort pixels we start from the image boundary...
RUNNING THE 1ST PASS

sort: O A B C D E F

[Diagram of shapes and tree structure]

T. Géraud et al. (LRDE) QUASI-LINEAR TREE OF SHAPES COMPUTATION ISMM, Uppsala, May 2013
Running the 1st pass

sort: O A B C D E F

Diagram of shapes and connections:

- O
  - A
    - B
    - E
  - D
  - C
    - F

- B -> D, E -> F
- C

Running the 1st pass

sort: O A B C D E F
Running the 1st pass

sort: O A B C D E F

Diagram with shapes and labels.
Running the 1st pass

sort: [O, A, B, C, D, E, F]

Diagram with shapes and elements E and F highlighted.
Running the 1st pass

sort: O A B C D E F
Done! Indeed, sorting pixels **seems** easy...
TOWARDS THE 2ND KEY IDEA
Towards the 2nd key idea

We need to pass between pixels...
Towards the 2nd key idea

\[
\begin{array}{cccccc}
1 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 2 & 2 & 1 \\
1 & 0 & 1 & 1 & 2 & 1 \\
1 & 0 & 0 & 2 & 2 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

we need to pass between pixels...

...and with many values
The (impossible?) 2nd key idea

To progress correctly both in image space and in value space...

Second key idea

...we need a discrete image representation that has some appropriate continuous properties!
To progress correctly both in image space and in value space...

**SECOND KEY IDEA**

...we need a **discrete** image representation that has some appropriate **continuous** properties!

To that aim a couple of tools are required:

- cubical complexes / Khalimsky grid
- set-valued maps
The $nD$ space of cubical complexes:

\[
H_0^1 = \{ \{a\}; \ a \in \mathbb{Z} \} \quad \quad H_1^1 = \{ \{a, a + 1\}; \ a \in \mathbb{Z} \}
\]

\[
H^1 = H_0^1 \cup H_1^1 \quad \quad H^n = \times_n H^1
\]
The nD space of cubical complexes:

\[
\begin{align*}
H_0^1 &= \{ \{a\}; \ a \in \mathbb{Z} \} & H_1^1 &= \{ \{a, a + 1\}; \ a \in \mathbb{Z} \} \\
H^1 &= H_0^1 \cup H_1^1 & H^n &= \times_n H^1
\end{align*}
\]

Consider \( h \in H^n \) being the \( \times \) product of \( d \) elements of \( H_1^1 \) and \( n - d \) elements of \( H_0^1 \)

- we have \( h \subset \mathbb{Z}^n \)
- we say that \( h \) is a \( d \)-face and that \( d \) is the dimension of \( h \)
The $nD$ space of cubical complexes:

\[
\begin{align*}
H_0^1 &= \{ \{a\}; a \in \mathbb{Z} \} \\
H^1 &= H_0^1 \cup H_1^1 \\
H_1^1 &= \{ \{a, a + 1\}; a \in \mathbb{Z} \} \\
H^n &= \times_n H^1
\end{align*}
\]

Consider $h \in H^n$ being the $\times$ product of $d$ elements of $H_1^1$ and $n - d$ elements of $H_0^1$

- we have $h \subset \mathbb{Z}^n$
- we say that $h$ is a $d$-face and that $d$ is the dimension of $h$

Two representations of a set of faces... ...and Khalimsky’s grid.
With \( h^\uparrow = \{ h' \in H^n \mid h \subseteq h' \} \) and \( h^\downarrow = \{ h' \in H^n \mid h' \subseteq h \} \):

- the pair \((H^n, \subseteq)\) forms a poset,
- the set \( U = \{ U \subseteq H^n \mid \forall h \in U, \ h^\uparrow \subseteq U \} \) is a T0-Alexandroff topology on \( H^n \).

---

**Cubical Complexes / Khalimsky’s Grid**
With $h^\uparrow = \{ h' \in H^n \mid h \subseteq h' \}$ and $h^\downarrow = \{ h' \in H^n \mid h' \subseteq h \}$:

- the pair $(H^n, \subseteq)$ forms a poset,
- the set $\mathcal{U} = \{ U \subseteq H^n \mid \forall h \in U, h^\uparrow \subseteq U \}$ is a T0-Alexandroff topology on $H^n$.

so we have some topological operators:

\[ E = \{ f, g, h \} \]

\[
\text{star: } E^\uparrow
\]

\[
\text{closure: } E^\downarrow
\]
With $h^\uparrow = \{ h' \in H^n \mid h \subseteq h' \}$ and $h^\downarrow = \{ h' \in H^n \mid h' \subseteq h \}$:

- the pair $(H^n, \subseteq)$ forms a poset,
- the set $\mathcal{U} = \{ U \subseteq H^n \mid \forall h \in U, h^\uparrow \subseteq U \}$ is a T0-Alexandroff topology on $H^n$.

so we have some topological operators:

$E = \{ f, g, h \}$

star: $E^\uparrow$

closure: $E^\downarrow$

and an easy and effective structure to work on...
A set-valued map $U$ is characterized by its graph:

$$\text{Gra}(U) = \{ (x, y) \in X \times Y \mid y \in U(x) \}$$

Actually we have $U : X \rightarrow \mathcal{P}(Y)$.
A set-valued map $U$

$U : X \leadsto Y$ is characterized by its graph:

$$\text{Gra}(U) = \{ (x, y) \in X \times Y \mid y \in U(x) \}$$

Actually we have $U : X \rightarrow \mathcal{P}(Y)$
Set-Valued Maps

Continuity:

- When $U(x)$ is compact, $U$ is u.s.c. at $x$ if
  \[ \forall \varepsilon > 0, \exists \eta > 0 \text{ such that } \forall x' \in B_X(x, \eta), \ U(x') \subset B_Y(U(x), \varepsilon). \]

- $U$ is u.s.c. iif $\forall x \in X$, $U$ is u.s.c. at $x$

- This is the “natural” extension of the continuity of a single-valued function
SET-VALUED MAPS

Continuity:

- when \( U(x) \) is compact, \( U \) is U.S.C. at \( x \) if
  \[
  \forall \varepsilon > 0, \ \exists \eta > 0 \text{ such that } \forall x' \in B_X(x, \eta), \ U(x') \subset B_Y(U(x), \varepsilon).
  \]
- \( U \) is U.S.C. iif \( \forall x \in X, \ U \) is U.S.C. at \( x \)
- this is the “natural” extension of the continuity of a single-valued function

Inverse:

the core of \( M \subset Y \) by \( U \) is \( \{ x \in X \mid U(x) \subset M \} \)
**Set-Valued Maps**

Continuity:
- when $U(x)$ is compact, $U$ is U.S.C. at $x$ if
  \[ \forall \varepsilon > 0, \exists \eta > 0 \text{ such that } \forall x' \in B_X(x, \eta), U(x') \subset B_Y(U(x), \varepsilon). \]
- $U$ is U.S.C. iff $\forall x \in X$, $U$ is U.S.C. at $x$
- this is the “natural” extension of the continuity of a single-valued function

Inverse:
- the core of $M \subset Y$ by $U$ is \( \{ x \in X \mid U(x) \subset M \} \)

A continuity characterization:
- $U$ is U.S.C. if and only if the core of any open subset is open
Recap:

- we want a continuous propagation in order to sort pixels
- we can rely
  - on cellular complexes
    for a propagation front to be able pass between pixels
  - and on set-valued maps
to model that we can have several level lines passing between pixels
Recap:

- we want a continuous propagation in order to sort pixels
- we can rely
  - on cellular complexes
  \[\text{for a propagation front to be able pass between pixels}\]
  - and on set-valued maps
  \[\text{to model that we can have several level lines passing between pixels}\]

At that point:

- we have to change \(u\) into \(U = \mathcal{I}(u)\)
  \[\text{so that } \mathcal{I}(u) \text{ is suitable to compute the tree of shapes}\]

- we need to define cuts of set-valued maps
  \[\text{so that we can have the notion of shapes of } \mathcal{I}(u)\]
definition of large cuts:

\[ U \trianglelefteq \lambda \] = \{ x \in X | \exists \mu \in U(x), \mu \leq \lambda \}

\[ U \triangleright \lambda \] = \{ x \in X | \exists \mu \in U(x), \mu \geq \lambda \}
definition of large cuts:

\[
[U \sqsubseteq \lambda] = \{ x \in X \mid \exists \mu \in U(x), \ \mu \leq \lambda \}
\]
\[
[U \sqsupset \lambda] = \{ x \in X \mid \exists \mu \in U(x), \ \mu \geq \lambda \}
\]

by extension we define:

\[
[U \triangledown \lambda] = X \setminus [U \sqsupset \lambda]
\]
\[
[U \triangledown \lambda] = X \setminus [U \sqsubseteq \lambda]
\]
\[
[U \square \lambda] = [U \sqsubseteq \lambda] \cap [U \sqsupset \lambda]
\]
\[
[U \lozenge \lambda] = X \setminus [U \square \lambda]
\]
definition of large cuts:

\[ [U \triangleleft \lambda] = \{ x \in X | \exists \mu \in U(x), \mu \leq \lambda \} \]
\[ [U \triangleright \lambda] = \{ x \in X | \exists \mu \in U(x), \mu \geq \lambda \} \]

by extension we define:

\[ [U \triangleleft \lambda] = X \setminus [U \triangleright \lambda] \]
\[ [U \triangleright \lambda] = X \setminus [U \triangleleft \lambda] \]
\[ [U \Box \lambda] = [U \triangleleft \lambda] \cap [U \triangleright \lambda] \]
\[ [U \nabla \lambda] = X \setminus [U \Box \lambda] \]

so we have:

\[ [U \triangleleft \lambda] = \{ x \in X | \forall \mu \in U(x), \mu < \lambda \} \]
\[ [U \triangleright \lambda] = \{ x \in X | \forall \mu \in U(x), \mu > \lambda \} \]
\[ [U \Box \lambda] = \{ x \in X | \lambda \in U(x) \} \]
\[ [U \nabla \lambda] = \{ x \in X | \lambda \notin U(x) \} \]
From \( u \) to \( I(u) \):

\[
\begin{array}{cc}
4 & 2 \\
1 & 8 \\
\end{array}
\]
From $u$ to $\mathcal{I}(u)$:

We subdivide $\mathbb{Z}^n$ into $\frac{1}{2}\mathbb{Z}^n$. 
From $u$ to $\mathcal{I}(u)$:

\[
\begin{array}{ccc}
4 & 4 & 2 \\
1 & 8 & 8 \\
4 & 8 & 8 \\
1 & 8 & 8 \\
\end{array}
\]

the $\max$ operator allows to “simulate” the $c_{2n} / c_{3n-1}$ connectivities.
From $u$ to $\mathcal{I}(u)$:

from $\frac{1}{2}\mathbb{Z}^n$ to $\frac{1}{2}H^n$ (introducing $d$-faces, $d < n$)
From $u$ to $I(u)$:

the \textit{span} operator makes this set-valued map \textit{continuous}.
Get a pen, than test continuity of this image:
Testing cuts

\[ \mathcal{I}(u) < 5 \]

\[ \mathcal{I}(u) \geq 3 \]
ANALOGIES

with $u$

dual trees:
\[
\mathcal{T}_<(u) = \{ \Gamma \in CC_{c_{2n}}([u < \lambda]) \} \lambda \\
\mathcal{T}_\geq(u) = \{ \Gamma \in CC_{c_{3n-1}}([u \geq \lambda]) \} \lambda
\]

shapes:
\[
\mathcal{S}_<(u) = \{ \text{Sat}_{c_{3n-1}}(\Gamma); \ \Gamma \in \mathcal{T}_<(u) \} \\
\mathcal{S}_\geq(u) = \{ \text{Sat}_{c_{2n}}(\Gamma); \ \Gamma \in \mathcal{T}_\geq(u) \}
\]

tree of shapes:
\[
\mathcal{S}(u) = \mathcal{S}_<(u) \cup \mathcal{S}_\geq(u)
\]
ANALOGIES

with $u$

dual trees:

$T_<(u) = \{ \Gamma \in CC_{c_{2n}}([u < \lambda]) \}_\lambda$

$T_>(u) = \{ \Gamma \in CC_{c_{3n-1}}([u \geq \lambda]) \}_\lambda$

shapes:

$S_<(u) = \{ \text{Sat}_{c_{3n-1}}(\Gamma); \ \Gamma \in T_<(u) \}$

$S_>(u) = \{ \text{Sat}_{c_{2n}}(\Gamma); \ \Gamma \in T_>(u) \}$

tree of shapes:

$\mathcal{S}(u) = S_<(u) \cup S_>(u)$

with $U = \mathcal{I}(u)$

$T_<(U) = \{ \Gamma \in CC([U < \lambda]) \}_\lambda$

$T_>(U) = \{ \Gamma \in CC([U > \lambda - \frac{1}{2}]) \}_\lambda$

shapes:

$S_<(U) = \{ \text{Sat}(\Gamma); \ \Gamma \in T_<(U) \}$

$S_>(U) = \{ \text{Sat}(\Gamma); \ \Gamma \in T_>(U) \}$

tree of shapes:

$\mathcal{S}(U) = S_<(U) \cup S_>(U)$
**ANALOGIES**

with $u$

**dual trees:**

$$T_<(u) = \{ \Gamma \in CC_{c2n}(\lfloor u < \lambda \rfloor) \}_\lambda$$

$$T_{\geq}(u) = \{ \Gamma \in CC_{c3n-1}(\lfloor u \geq \lambda \rfloor) \}_\lambda$$

**shapes:**

$$S_<(u) = \{ \text{Sat}_{c3n-1}(\Gamma); \Gamma \in T_(u) \}$$

$$S_{\geq}(u) = \{ \text{Sat}_{c2n}(\Gamma); \Gamma \in T_\geq(u) \}$$

**tree of shapes:**

$$S(u) = S_<(u) \cup S_{\geq}(u)$$

with $U = \mathcal{I}(u)$

**dual trees:**

$$T_<(u) = \{ \Gamma \in CC([u < \lambda]) \}_\lambda$$

$$T_{\geq}(u) = \{ \Gamma \in CC([u \geq \lambda]) \}_\lambda$$

**shapes:**

$$S_<(u) = \{ \text{Sat}(\Gamma); \Gamma \in T_(u) \}$$

$$S_{\geq}(u) = \{ \text{Sat}(\Gamma); \Gamma \in T_\geq(u) \}$$

**tree of shapes:**

$$S(U) = S_<(U) \cup S_{\geq}(U)$$

We have

- components and shapes of $U$ are open sets
- for any tree, couples of components coincide: $\Gamma_U \cap \mathbb{Z}^n = \Gamma_u$
- $S(u) = S(U) \mid \mathbb{Z}^n$
Eventually we have:

$$\mathcal{S}(u) = \{ \Gamma \cap \mathbb{Z}^n ; \Gamma \in \mathcal{S}(\mathcal{I}(u)) \}$$
Eventually we have:

\[ \mathcal{G}(u) = \{ \Gamma \cap \mathbb{Z}^n ; \Gamma \in \mathcal{G}(I(u)) \} \]

So the algorithm to obtain \( \mathcal{G}(u) \) is:

- represent \( u \) by \( I(u) \)
- run a “continuous propagation” on \( I(u) \) to sort faces (1st step)
- run a Union-Find-based computation of the tree (2nd step)
- clean-up the tree, i.e., keep \( \mathbb{Z}^n \) elements only
**SOME DIRECT CONSEQUENCES**

Eventually we have:

\[ \mathcal{G}(u) = \{ \Gamma \cap \mathbb{Z}^n ; \Gamma \in \mathcal{G}(J(u)) \} \]

So the algorithm to obtain \( \mathcal{G}(u) \) is:

- represent \( u \) by \( J(u) \)
- run a “continuous propagation” on \( J(u) \) to sort faces (1st step)
- run a Union-Find-based computation of the tree (2nd step)
- clean-up the tree, i.e., keep \( \mathbb{Z}^n \) elements only

We have a proof of this algorithm.
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII ART: PROPAGATION / SORTING (1ST PASS)
ASCII art: propagation / sorting (1st pass)
ASCII ART: PROPAGATION / SORTING (1ST PASS)
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII ART: PROPAGATION / SORTING (1ST PASS)
ASCII art: propagation / sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII art: Propagation / Sorting (1st pass)
ASCII art: propagation / sorting (1st pass)
ASCII art: TREE COMPUTATION (2ND PASS)
ASCII art: TREE COMPUTATION (2nd pass)
ASCII art: TREE COMPUTATION (2ND PASS)
ASCII art: TREE COMPUTATION (2ND PASS)
ASCII art: tree computation (2nd pass)
QUASI-LINEAR TREE OF SHAPES COMPUTATION

T. GÉRAUD ET AL. (LRDE)

ISMM. UPPSALA, MAY 2013

74/1
ASCII art: TREE COMPUTATION (2ND PASS)

---

T. Géraud et al. (LRDE)  Quasi-linear Tree of Shapes computation  ISMM, Uppsala, May 2013
ASCII art: TREE COMPUTATION (2ND PASS)
ASCII art: TREE COMPUTATION (2ND PASS)
ASCII art: tree computation (2nd pass)
ASCII art: TREE COMPUTATION (2ND PASS)
ASCII art: tree computation (2nd pass)
ASCII art: TREE COMPUTATION (2ND PASS)
ASCII art: TREE COMPUTATION (2ND PASS)
ASCII art: TREE COMPUTATION (2ND PASS)
ASCII art: TREE COMPUTATION (2ND PASS)
ASCII art: TREE COMPUTATION (2ND PASS)
ASCII art: TREE COMPUTATION (2ND PASS)
ASCII art: TREE COMPUTATION (2ND PASS)
ASCII art: Tree computation (2nd pass)
ASCII art: TREE COMPUTATION (2ND PASS)
ASCII art: Tree computation (2nd pass)
ASCII art: tree computation (2nd pass)
ASCII art: TREE COMPUTATION (2ND PASS)
ASCII art: tree computation (2nd pass)
ASCII art: tree computation (2nd pass)
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ASCII ART: TREE COMPUTATION (2ND PASS)**
ASCII art: tree computation (2nd pass)
ASCII art: TREE COMPUTATION (2ND PASS)
ASCII art: Tree computation (2nd pass)
ASCII art: tree computation (2nd pass)
ASCII art: tree computation (2nd pass)
ASCII ART: TREE COMPUTATION (2ND PASS)
ASCII art: tree computation (2nd pass)
ASCII art: tree computation (2nd pass)
ASCII art: tree computation (2nd pass)
Source code available in the next release of Milena:

- generic and efficient image processing C++ library
- free software under the GPL v2 licence
- including many pieces of mathematical morphology in it...

Try Milena!

- “Since I use Milena, I feel different,” Steve Jobs
- “Google is named after Milena, yet we change some letters,” L. Page and S. Brin
- “Milena just leaves me speechless,” M. Wilkinson

http://olena.lrde.epita.fr
CONCLUSION

Contributions:

- a new algorithm to compute the tree of shapes
  - working on $n$D images
  - quasi-linear for low quantized data
  - very easy to implement

- a new image representation
  - based on cellular complexes and set-valued maps
  - discrete yet "continuous"
  - with many potential developments
Contributions:

- a new algorithm to compute the tree of shapes
  - working on $n$D images
  - quasi-linear for low quantized data
  - very easy to implement

- a new image representation
  - based on cellular complexes and set-valued maps
  - discrete yet "continuous"
  - with many potential developments
CONCLUSION

Contributions:

- A new algorithm to compute the tree of shapes
  - Working on nD images
  - Quasi-linear for low quantized data
  - Very easy to implement

- A new image representation
  - Based on cellular complexes and set-valued maps
  - Discrete yet “continuous”
  - With many potential developments
CONCLUSION

Contributions:

- a new algorithm to compute the tree of shapes
  - working on nD images
  - quasi-linear for low quantized data
  - very easy to implement

- a new image representation
  - based on cellular complexes and set-valued maps
  - discrete yet “continuous”
  - with many potential developments
CONCLUSION

Contributions:

- a new algorithm to compute the tree of shapes
  - working on $nD$ images
  - quasi-linear for low quantized data
  - very easy to implement

- a new image representation
  - based on cellular complexes and set-valued maps
  - discrete yet “continuous”
  - with many potential developments
CONCLUSION

Contributions:

- a new algorithm to compute the tree of shapes
  - working on $n$D images
  - quasi-linear for low quantized data
  - very easy to implement

- a new image representation
  - based on cellular complexes and set-valued maps
  - discrete yet “continuous”
  - with many potential developments
CONCLUSION

Contributions:

- a new algorithm to compute the tree of shapes
  - working on $nD$ images
  - quasi-linear for low quantized data
  - very easy to implement

- a new image representation
  - based on cellular complexes and set-valued maps
  - discrete yet “continuous”
  - with many potential developments
CONCLUSION

Contributions:

- a new algorithm to compute the tree of shapes
  - working on nD images
  - quasi-linear for low quantized data
  - very easy to implement

- a new image representation
  - based on cellular complexes and set-valued maps
  - discrete yet “continuous”
  - with many potential developments
WHAT IS *not* PRESENTED HERE

In a journal paper (in preparation):

- a formal proof of our algorithm
- a discussion about propagation initialization (root node)
- how to reduce the memory bloat ($\times 4^n$ from $u$ to $\mathcal{I}(u)$)
- a comparison of execution times of existing algorithms
- parallization strategy
Contributions

Thierry Géraud: funny ideas and lead
Edwin Carlinet: algorithm design, seminal implementation
Sébastien Crozet: tests, algorithm proof, parallelization
Laurent Najman: good ideas and references
THE END

Yet another killing app: food detection

Thanks for your attention. Any question?
**Step 1: “Union-Find”-Based Tree Computation**

**FIND_ROOT(zpar, x) : P**

```plaintext
begin
if zpar(x) = x then
  return x
else
  zpar(x) ← FIND_ROOT(zpar, zpar(x))
return zpar(x)
```

**UNION_FIND(\mathcal{R}) : T**

```plaintext
begin
for all \( p \) do
  zpar(p) ← undef
for i ← N - 1 to 0 do
  p ← \mathcal{R}[i]
  parent(p) ← p
  zpar(p) ← p
for all \( n \in \mathcal{N}(p) \) such as \( zpar(n) \neq \text{undef} \) do
  r ← FIND_ROOT(zpar, n)
  if r \neq p then
    parent(r) ← p
    zpar(r) ← p
return parent
```
**Step 1: Sorting (“Continuous” Propagation)**

**Priority Push**

```plaintext
/* modify q */
begin
[lower, upper] ← U(h)
if lower > ℓ then
    ℓ' ← lower
else if upper < ℓ then
    ℓ' ← upper
else
    ℓ' ← ℓ
PUSH(q[ℓ'], h)
end
```

**Priority Pop**

```plaintext
/* modify q, and sometimes ℓ */
begin
if q[ℓ] is empty then
    ℓ' ← level next to ℓ such as q[ℓ'] is not empty
    ℓ ← ℓ'
return POP(q[ℓ])
end
```

**Sort**

```plaintext
SORT(U) : Pair(Array[H], Image)
begin
for all h do
    deja_vu(h) ← false
    i ← 0
    PUSH(q[ℓ∞], p∞)
    deja_vu(p∞) ← true
    ℓ ← ℓ∞ /* start from root level */
while q is not empty do
    h ← PRIORITY_POP(q, ℓ)
    u♭(h) ← ℓ
    R[i] ← h
    for all n ∈ N(h) such as deja_vu(n) = false do
        PRIORITY_PUSH(q, n, U, ℓ)
        deja_vu(n) ← true
    i ← i + 1
/* if q[ℓ] is empty we are done with level ℓ */
return (R, u♭)
end
```
TREE OF SHAPE COMPUTATION

\[
\text{COMPUTE_TREE_OF_SHAPES}\ (u) : \text{Pair(Array}[P], T)
\]

begin

\[
U \leftarrow \text{INTERPOLATE}(u)
\]

\[
(\mathcal{R}, u^b) \leftarrow \text{SORT}(U)
\]

\[
\text{parent} \leftarrow \text{UNION_FIND}(\mathcal{R})
\]

\[
\text{CANONICALIZE_TREE}(u^b, \mathcal{R}, \text{parent})
\]

return \text{UN-INTERPOLATE}(\mathcal{R}, \text{parent})
END OF THE SLIDE SET...
**Answer to Michael’s question**

**Low quantized data:**
- appropriate data structure = hierarchical queue
- time complexity $O(N)$
- $K$ (number of quanta) is a multiplicative constant

**Oddity:**
- having high-bit-depth data...
- ...and using a hierarchical queue (**unappropriate** data structure!)
- time complexity is quasi-polynomial: $O(KN)$
- nobody wants that!

**High-bit-depth data:**
- appropriate data structure = Y-fast trie
- time complexity $O(N \log K)$
- space complexity $O(N)$