Un algorithme de complexité linéaire pour le calcul de l’arbre des formes

E. Carlinet, S. Crozet, T. Géraud
firstname.lastname@lrde.epita.fr

EPITA Research & Development Laboratory (LRDE)

2018/06/27 — RFIAP — Marne-la-Vallée, France
Forewords
What is the Tree of Shapes (def 1)?

- Definition 1: The tree of inclusion of the image level lines
What is the Tree of Shapes (def 1)?

- Definition 1: The tree of inclusion of the image level lines

The organization of the level lines is:

- invariant by a *global* contrast change (increasing function)
- invariant by gray-level inversion
- robust to *local* changes of illumination
Applications

From general simple filters... 

- Grain filter (= tree pruning)

This is a connected operator: no contours are shifted (some contours are removed, based on the component size).
Applications

From general simple filters, or general advanced filters...

► Shapings (= filtering in the space of shapes)

Xu, Géraud, and Najman. Connected filtering on tree-based shape-spaces. PAMI 2016
Applications

From *general simple* filters, or *general advanced* filters, to *app-specific* methods...

- Interactive segmentation

Applications

From *general simple* filters, or *general advanced* filters, to *app-specific* methods...

- Interactive segmentation

Applications

From *general* simple filters, or *general* advanced filters, to *app-specific* methods, passing by *general-purpose* CV methods

- Object detection / Image simplification

Xu, Géraud, and Najman. *Context-based energy estimator: Application to object segmentation on the ToS.* ICIP 2012
Applications

From *general* simple filters, or *general* advanced filters, to *app-specific* methods, passing by *general-purpose* CV methods

- Hierarchy of Segmentations

Xu, Carlinet, Géraud, and Najman. *Hierarchical segmentation using tree-based shape spaces*. PAMI 2017
What is the Tree of Shapes (def 2)?

- **Definition 2:**
  The tree of inclusion $\mathcal{T}$ of the hole-filled connected components
What is the Tree of Shapes (def 2)?

- **Definition 2:**
  The tree of inclusion $\mathcal{T}$ of the hole-filled connected components

- **Lower level sets and min-tree**

\[
[u < \lambda] = \{x \in X \mid u(x) < \lambda\}
\]
\[
\mathcal{T}_{<}(u) = \{\Gamma \in CC([u < \lambda])\}
\]

Upper level sets and max-tree defined dually.
What is the Tree of Shapes (def 2 cont.)?

Definition 2:
The tree of inclusion $\mathcal{T}$ of the hole-filled connected components from min and max-trees:

$$\mathcal{T} = \text{Sat}(\mathcal{T}_<(u)) \cup \text{Sat}(\mathcal{T}_>(u))$$
Computing the ToS
How to

Actually, few algorithms... 


How to

Actually, few algorithms...


... but many issues:

- (1,2,3) *worst* time complexity is $O(N^2)$
- (1,2,3) are hard to implement
- (2,3) are limited to 2D images and (1) might be untractable in 3D...
- (4) is quasi-linear but has a high constant multiplier
The idea

There is much more literature about min/max trees.

Thus, more research about efficient algorithms:

The idea

- There is much more literature about min/max-trees
- Thus, more research about efficient algorithms:

Idea

- Turn the ToS computation into a max-tree computation
A new ToS algorithm
ToS algorithm

A two steps algorithm:

1. Turn the image into a depth map
2. Compute the max-tree of the depth map

Algorithm properties

- $O(n)$ for low-quantized data and non-degenerated cases
- Worst cases:
  - quasi-linear for low quantized data
  - $O(n \log n)$ for high-quantized data
Step #2: max-tree computation

\[ \text{ToS}(\text{original image}) = \text{maxtree}(\text{processed image}) \]
Step #2: max-tree computation

\[ \text{ToS}(\cdot) = \text{maxtree}(\cdot) \]

Explanations

Q: how to get \( \tilde{u} \) without \( \mathcal{T} \)?
Step #1: turning the image into a depth map

Slight modification of a step from Géraud et al., ISMM 2013:

- Sort the pixels in the *descending tree order*

  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)
Step #1: turning the image into a *depth* map
Step #1: turning the image into a depth map
Step #1: turning the image into a depth map
Step #1: turning the image into a depth map
Step #1: turning the image into a *depth* map
Step #1: turning the image into a depth map
Problem(s) & solution(s)

Problem #1

We need to pass between pixels and with intermediate values
Problem(s) & solution(s)

Problem #1

We need to pass between pixels and with intermediate values

Solution #1 (not presented in this talk)

Interval-valued set on the Khalimsky grid (see Géraud et al., ISMM 2013)

⇒ Problem #2

It requires to *double twice* the size of the image: x16 in 2D, x64 in 3D
Problem(s) & solution(s)

Problem #1

We need to pass between pixels and with intermediate values

Solution #1 (not presented in this talk)

Interval-valued set on the Khalimsky grid (see Géraud et al., ISMM 2013)

⇒ Problem #2

It requires to *double twice* the size of the image: ×16 in 2D, ×64 in 3D

Solution to the problem #2 of the pb #1’s soluce (in the paper)

A way to reduce memory space usage: *only* ×4 in 2D, ×8 in 3D
Results
Performances

Protocol

- Competitors:
  - Song, 2007
  - Géraud et al., 2013
  - *FLST*: Caselles & Monasse, 2009

- 20-MPix natural images (cropped from 1M to 16M)

- Intel Core i7 7500U, 2.7Ghz, 8Gb of RAM
Most algorithms are (quite) linear in practice ($O(n^2)$ worst case not reached)
Performances

- Our algorithm is:
  - 4X faster than Géraud et al.
  - 2.5X faster than the FLST
  - more stable than Song’s
Conclusion

What we have proposed:

- an idea: turning the ToS into a MaxTree computation
- an optimization (not presented here)

Why?

To use any blasting maxtree algorithm you need, e.g.

- for distributed systems
- for parallel shared-memory systems
- for embedded systems (i.e. low memory constraints)
Conclusion

What we have proposed:

- an idea: turning the ToS into a MaxTree computation
- an optimization (not presented here)

Why?

To use any blasting maxtree algorithm you need, e.g.

- for distributed systems
- for parallel shared-memory systems
- for embedded systems (i.e. low memory constraints)

What’s next?

- Improving the step 1 (in terms of efficiency/concurrency...)
- Optimization generalized to n-D (described only in 2D for now)
Reproducible Research

- Implemented with our library
  https://gitlab.lrde.epita.fr/olena/pylene

- Source code available
  https://gitlab.lrde.epita.fr/olena/pylene-apps
Thanks for your attention. Any questions?

Toute? Non! Un village d'irréductibles résiste toujours à l'envahisseur!