A quasi-linear algorithm to compute the tree of shapes of n-D images

Thierry Géraud, Edwin Carlinet, Sébastien Crozet, Laurent Najman

A quasi-linear algorithm to compute the tree of shapes of nD images

Thierry Géraud[1,2], Edwin Carlinet[1,2], Sébastien Crozet[1], and Laurent Najman[2]

[1] EPITA Research and Development Laboratory (LRDE)
[2] Université Paris-Est, LIGM, Équipe A3SI, ESIEE
firstname.lastname@lrde.epita.fr, l.najman@esiee.fr

Abstract. To compute the morphological self-dual representation of images, namely the tree of shapes, the state-of-the-art algorithms do not have a satisfactory time complexity. Furthermore the proposed algorithms are only effective for 2D images and they are far from being simple to implement. That is really penalizing since a self-dual representation of images is a structure that gives rise to many powerful operators and applications, and that could be very useful for 3D images. In this paper we propose a simple-to-write algorithm to compute the tree of shapes; it works for nD images and has a quasi-linear complexity when data quantization is low, typically 12 bits or less. To get that result, this paper introduces a novel representation of images that has some amazing properties of continuity, while remaining discrete.

1 Introduction

The tree of shapes [16] is an important morphological structure that represents images in a self-dual way. Shortly put it can be seen as the result of merging the pair of dual component trees, min-tree and max-tree, into a single tree. Using the tree of shapes has many advantages. Since it is self-dual, it makes no assumption about the contrast of objects (either light object over dark background or the contrary). We only have one structure that represents the image contents so we do not have to juggle with the couple of dual trees. It intrinsically eliminates the redundancy of information contained in those trees. Last, it encodes the spatial inclusion of connected components in gray-level images so it is complementary to some other representations that focus on component (or region) adjacency. As a consequence the tree of shapes is not only an easy access to self-dual operators such as grain filters but it has many applications, as listed in [14] (pp. 15–17), and some very recent works illustrate several powerful perspectives offered by that tree (see [20,21,22], and their bibliography).

In the following we consider a nD digital image u as a function defined on a regular cubical grid (precisely, $u : \mathbb{Z}^n \to \mathbb{Z}$), and to properly deal with some subsets of $\mathbb{Z}^n$ and with their complementary, we consider the dual connectivities c_{2n} and c_{3^n-1}. For any $\lambda \in \mathbb{Z}$, the lower (strict) cuts[3] and upper (large) cuts of u are defined as $[u < \lambda] = \{x \in X \mid u(x) < \lambda\}$ and $[u \geq \lambda] = \{x \in$

[3] We can indifferently use the term "cut" or "threshold".

$X \mid u(x) \geq \lambda\}$. From them we deduce two sets, $\mathcal{T}_<(u)$ and $\mathcal{T}_\geq(u)$, composed of the connected components of respectively lower and upper cuts of u: $\mathcal{T}_<(u) = \{\Gamma \in \mathcal{CC}_{c_{2n}}([u < \lambda])\}_\lambda$ and $\mathcal{T}_\geq(u) = \{\Gamma \in \mathcal{CC}_{c_{3^n-1}}([u \geq \lambda])\}_\lambda$, where $\mathcal{CC}$ denotes the operator that gives the set of connected components of a set. The elements of $\mathcal{T}_<(u)$ and $\mathcal{T}_\geq(u)$ respectively give rise to two dual trees: the min-tree and the max-tree of u. We then define two other sets, $\mathcal{S}_<(u)$ (set of lower shapes) and $\mathcal{S}_\geq(u)$ (set of upper shape), as the sets of components of resp. $\mathcal{T}_<(u)$ and $\mathcal{T}_\geq(u)$ after having filled the cavities [4] of those components. With the cavity-filling (or saturation) operator denoted by Sat, we have: $\mathcal{S}_<(u) = \{\mathrm{Sat}_{c_{3^n-1}}(\Gamma);\ \Gamma \in \mathcal{T}_<(u)\}$ and $\mathcal{S}_\geq(u) = \{\mathrm{Sat}_{c_{2n}}(\Gamma);\ \Gamma \in \mathcal{T}_\geq(u)\}$.

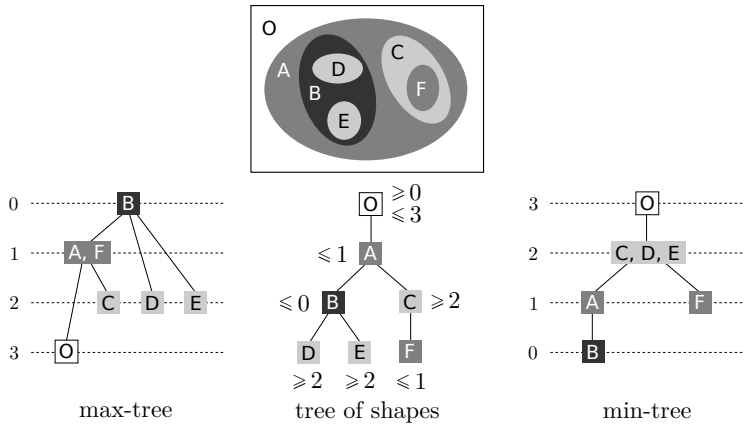

Fig. 1. Three morphological trees of the same image.

The set of all shapes $\mathfrak{S}(u) = \mathcal{S}_<(u) \cup \mathcal{S}_\geq(u)$ forms a tree, the so-called *tree of shapes* of u [16]. Indeed, for any pair of shapes Γ and Γ' in $\mathfrak{S}$, we have $\Gamma \subset \Gamma'$ or $\Gamma' \subset \Gamma$ or $\Gamma \cap \Gamma' = \emptyset$. Actually, the shapes are the cavities of the elements of $\mathcal{T}_<$ and $\mathcal{T}_\geq$. For instance, if we consider a lower component $\Gamma \in [u < \lambda]$ and a cavity H of Γ, this cavity is an upper shape, i.e., $H \in \mathcal{S}_\geq$. Furthermore, in a discrete setting, H is obtained after having filled the cavities of a component of $[u \geq \lambda]$. Figure 1 depicts on a sample image the three components trees ($\mathcal{T}_<$, $\mathcal{T}_\geq$, and $\mathfrak{S}$). Just note that the Equations so far rely on the pair of dual connectivities, c_{2n} and c_{3^n-1}, so discrete topological problems are avoided, and, in addition, we are forced to consider two kind of cuts: strict ones for c_{2n} and large ones for c_{3^n-1}.

The state-of-the-art of tree of shapes computation (detailed in Section 5) suffers from two major flaws: existing algorithms have a time complexity of $\mathrm{O}(n^2)$ and they cannot easily be extended to nD images. Briefly put, this is

[4] In 2D, a cavity of a set $S \in \Omega$ is called a "hole"; in nD, it is a connected component of $\Omega \backslash S$ which is not the "exterior" of S. Browsing the elements of S in nD, with $n \geq 3$, does not allow to know whether S has a cavity or not[8].

due to the fact that either they follow shape contours or they have to know if a component has a cavity[4]. This paper presents an algorithm that can compute the tree of shapes with quasi-linear time complexity when image data are low quantized; furthermore this algorithm straightforwardly applies to nD images.

This paper is organized as follows. First we explain that a well-known algorithmic scheme can be reused to compute the tree of shapes (Section 2). Then this paper introduces a new discrete representation of images (Section 3) that has some properties borrowed from the continuous world. At that point we are ready to glue together the algorithmic scheme and the novel image representation to present a quasi-linear algorithm that compute the tree of shapes (Section 4). Related works about that tree computation is presented so that the reader can compare our approach to existing ones (Section 5). Last, we give a short conclusion (Section 6) [5].

2 Algorithmic scheme and the need for continuity

This section shows that the max-tree algorithm presented in [2] is actually an algorithmic "canvas" [7], that is, a kind of meta-algorithm that can be "filled in" so that it can serve different aims. In the present paper it gives an algorithm to compute the tree of shapes.

2.1 About union-find and component trees

An extremely simple union-find structure (attributed by Aho to McIlroy and Morris) was shown by Tarjan [19] to be very efficient. This structure, also called disjoint-set data structure or merge-find set, has many advantages that are detailed in [3]; amongst them, memory compactedness, simplicity of use, and versatility. This structure and its related algorithms are of prime importance to handle connected operators [13,6].

Let us denote by $\mathcal{R}$ the ancestor relationship in trees: we have $a\,\mathcal{R}\,p$ iff a is an ancestor of p. $\mathcal{R}$ can be encoded as an array of elements (nodes) so that $a\,\mathcal{R}\,p \Leftrightarrow index_{\mathcal{R}}(a) < index_{\mathcal{R}}(p)$; browsing that array thus corresponds to a downwards browsing of the tree, i.e., from root to leaves. To construct the max-tree of a given image, we rely on a rooted tree defined by a parenthood function, named *parent*, and encoded as an nD image (so $parent(p)$ is an nD point). When a node of the max-tree contains several points, we choose its first point

[5] Due to limited place, this paper does not contain the following topics (they will be included into an extended version of this paper). *A comparison of execution times of existing algorithms.* Actually it is possible to reduce the space complexity (i.e., memory usage) of the algorithm proposed in this paper so the shorter version presented here is not our "competitive" version. *The union-by-rank procedure that guaranties quasi-linear complexity.* So that the Union-Find routine (given in [2] and recalled in Algorithm 1) remains short, its code does not feature tree balancing; yet it is explained in [3]. *A formal proof of our algorithm.* This paper focuses on how the proposed algorithm works and gives an insight into the reasons why it works; to give a formal proof requires a large amount of materials, the first part of which can be found in [17]. *About high bit-depths data.* That case is not detailled in this paper.

Algorithm 1: "Union-Find"-based computation of a morphological tree.

```
UNION_FIND(R) : T
begin
    for all p do
        zpar(p) ← undef
    for i ← N − 1 to 0 do
        p ← R[i]
        parent(p) ← p
        zpar(p) ← p
        for all n ∈
        N(p) such as zpar(n) ≠ undef
        do
            r ← FIND_ROOT(zpar, n)
            if r ≠ p then
                parent(r) ← p
                zpar(r) ← p
    return parent
```

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

```
COMPUTE_TREE (u) : Pair(Array[P], T)
begin
    R ← SORT(u)
    parent ← UNION_FIND(R)
    CANONICALIZE_TREE(u, R, parent)
    return (R, parent)
```

(with respect to $\mathcal{R}$) as the representative for this node; that point is called a component "canonical point" or a "level root". Let Γ denote a component corresponding to a node of the max-tree, p_Γ its canonical element, and p_r the root canonical element. The *parent* function that we want to construct should verify the following four properties: **1.** $parent(p_r) = p_r$; **2.** $\forall\, p \neq p_r,\ parent(p)\,\mathcal{R}\,p$; **3.** p is a canonical element iff $p = p_r \vee u(parent(p)) \neq u(p)$; **4.** $\forall\, p,\ p \in \Gamma \Leftrightarrow u(p) = u(p_\Gamma) \wedge \exists\, i, parent^i(p) = p_\Gamma$ (therefore $\forall\, p \in \Gamma,\ p = p_\Gamma \vee p_\Gamma\,\mathcal{R}\,p$).

The routine UNION_FIND, given in Algorithm 1, is the classical "union-find" algorithm [19] but *modified* so that it computes the expected morphological tree [2] while browsing pixels following $\mathcal{R}^{-1}$, i.e., from leaves to root (let us recall that we do not feature here the union-by-rank version). Its result is a *parent* function that fulfills those first four properties. Obtaining the following extra property, "5. $\forall p$, $parent(p)$ is a canonical element," is extremely interesting since it ensures that the parent function, when restricted to canonical elements only, gives a "compact" morphological tree such as the ones depicted in Figure 1. Precisely it allows to browse components while discarding their contents: a traversal is thus limited to one element (one pixel) per component, instead of passing through every image elements (pixels). Transforming the parent function so that property 5 is verified can be performed by a simple post-processing of the union-find computation. The resulting tree has now the simplest form that we can expect; furthermore we have an isomorphism between images and their canonical representations.

2.2 Computing the max-tree and the tree of shapes

The algorithm presented in [2] to compute the max-tree is surprisingly also able to compute the tree of shapes. The skeleton, or *canvas*, of this algorithm is the routine COMPUTE_TREE given in the right part of Algorithm 1; it is composed

of *three* steps: sort the image elements (pixels); then run the modified union-find algorithm to compute a tree encoded by a parent function; last modify the parent function to give that tree its canonical form.

In the case of the max-tree, the sorting step provides $\mathcal{R}$ encoded as an array of points sorted by increasing gray-levels in u, i.e., such that the array indices satisfy $i < i' \;\Rightarrow\; u(\mathcal{R}[i]) \leq u(\mathcal{R}[i'])$. When image data are low quantized, typically 12 bit data or less, then sorting points can be performed by a distribution sort algorithm. Last, the canonicalization post-processing is a trivial 5-line routine that the reader can find in [2]. In the case of the tree of shapes, it is also a tree that represents an inclusion relationship between connected components of the input image. As a consequence a first important idea to catch is that the tree of shapes can be computed with the exact same routine, UNION_FIND, as the one used by max-tree.

2.3 What if...

The major and crucial difference between the max-tree and the tree of shapes computations is obviously the sorting step. For the UNION_FIND routine to be able to compute the tree of shapes using $\mathcal{R}^{-1}$, the SORT routine has to sort the image elements so that $\mathcal{R}$ corresponds to a downward browsing of the tree of shapes. Schematically we expect that $\mathcal{R}$ contains the image pixels going from the "external" shapes to the "internal" ones (included in the former ones).

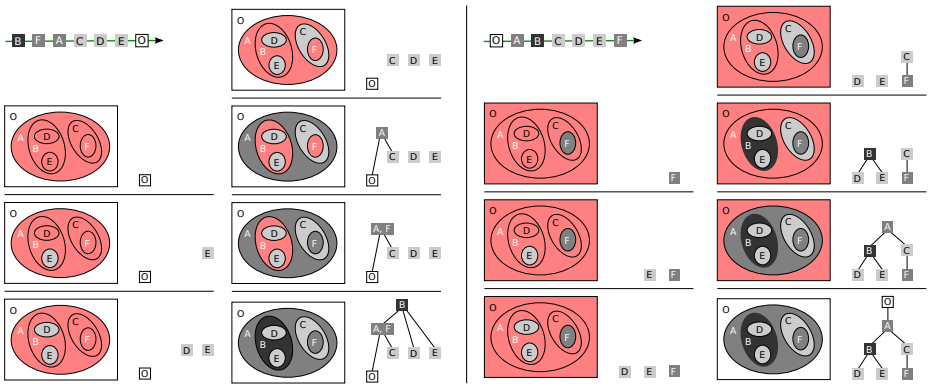

Fig. 2. Tree computation of the max-tree (left) and of the tree of shapes (right). For both cases, the result $\mathcal{R}$ of the sorting step is given over the green arrow and the tree computation, browsing $\mathcal{R}^{-1}$, is progressively depicted.

The similarity between the computations of both trees is illustrated in Figure 2. We can see that the modified union-find algorithm correctly computes both trees once $\mathcal{R}$ is properly defined. Therefore we "just" need to know how to compute $\mathcal{R}$ in the case of the tree of shapes to turn the canvas given in the previous Section 2.2 as the expected algorithm.

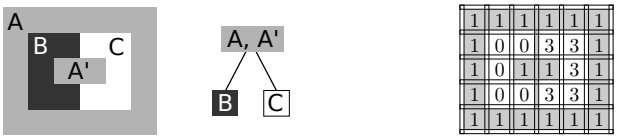

Fig. 3. A sample image and its tree of shapes (left); a step towards an *ad-hoc* image representation (right).

Let us consider the image depicted on the left of Figure 3 with its tree of shapes. We can see that we need to reach the regions A and A' before the regions B and C in order to properly sort pixels, i.e., to compute $\mathcal{R}$. It is only possible if we can pass "between" pixels. The representation depicted on the right of Figure 3 is well-suited for that since it contains some elements that materialize inter-pixel spaces. Furthermore, given a two adjacent pixels with respective values 0 and 3, the element in-between them has to bear all the "intermediate" values: not only 1 but also 2. Indeed, if we change the value of regions A and A' from 1 to 2, the tree structure is unchanged but inter-pixel elements between regions B and C have now to make A and A' connect with value 2. Eventually we need an image representation that is "continuous" in some way with respect to both the domain space and the value space.

3 Image representation

To be able to sort the image pixels so that $\mathcal{R}$ corresponds to a top-down browsing of tree of shapes elements, this paper introduces a novel representation of images[6]. It relies on a couple of theoretical tools briefly described hereafter[7].

3.1 Cellular complex and Khalimsky grid

From the sets $H_0^1 = \{\{a\}; a \in \mathbb{Z}\}$ and $H_1^1 = \{\{a, a+1\}; a \in \mathbb{Z}\}$, we can define $H^1 = H_0^1 \cup H_1^1$ and the set H^n as the n-ary Cartesian power of H^1. If an element $h \subset \mathbb{Z}^n$ is the Cartesian product of d elements of H_1^1 and $n-d$ elements of H_0^1, we say that h is a d-face of H^n and that d is the dimension of h. The set of all faces, H^n, is called the nD space of cubical complexes. Figure 4 depicts a set of faces $\{f, g, h\} \subset H^2$ where $f = \{0\}\times\{1\}$, $g = \{0,1\}\times\{0,1\}$, and $h = \{1\}\times\{0,1\}$; the dimension of those faces are respectively 0, 2, and 1. Let us write $h^\uparrow = \{h' \in H^n \,|\, h \subseteq h'\}$ and $h^\downarrow = \{h' \in H^n \,|\, h' \subseteq h\}$. The pair $(H^n, \subseteq)$ forms a poset and the set $\mathcal{U} = \{U \subseteq H^n \,|\, \forall h \in U,\, h^\uparrow \subseteq U\}$ is a T0-Alexandroff topology on H^n. With $E \subseteq H^n$, we have a star operator $st(E) = \cup_{h\in E} h^\uparrow$ and a closure operator $cl(E) = \cup_{h\in E} h^\downarrow$, that respectively gives the smallest open set and the smallest closed set of $\mathcal{P}(H^n)$ containing E.

The set of faces of H^n is arranged onto a grid, the so-called Khalimsky's grid, depicted in gray in Figure 4 (right); and inclusion between faces lead to a

[6] In [17], a formal characterization of the discrete topology underlying this novel representation is presented.

[7] The authors recommend [12] and [1] for extra readings about those tools.

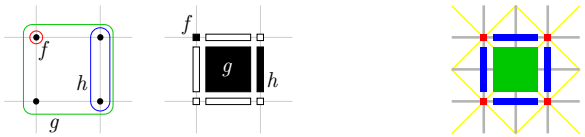

Fig. 4. Three faces depicted as subsets of $\mathbb{Z}^2$ (left) and as geometrical objects (middle); Khalimsky grid (right) with 0- to 2-faces respectively painted in red, blue, and green.

neighborhood relationship, depicted in gray and yellow. The set of 2-faces, the minimal open sets of H^n, is the n-Cartesian product of H_1 and is denoted by H_1^n.

3.2 Set-valued maps

A set-valued map $\mathrm{U} : X \rightsquigarrow Y$ is characterized by its graph, $\mathrm{Gra}(\mathrm{U}) = \{ (x, y) \in X \times Y \mid y \in \mathrm{U}(x) \}$. There are two different ways to define the "inverse" of a subset by a set-valued map: $\mathrm{U}^{\oplus}(M) = \{ x \in X \mid \mathrm{U}(x) \cap M \neq \emptyset \}$ is the *inverse image* of M by U, whereas $\mathrm{U}^{\ominus}(M) = \{ x \in X \mid \mathrm{U}(x) \subset M \}$ is the *core* of M by U. Two distinct continuities are defined on set-valued maps. The one we are interested in is the "natural" extension of the continuity of a single-valued function. When X and Y are metric spaces and when $\mathrm{U}(x)$ is compact, U is said to be Upper Semi-Continuous (U.S.C.) at x if $\forall \varepsilon > 0$, $\exists \eta > 0$ such that $\forall\, x' \in B_X(x, \eta)$, $\mathrm{U}(x') \subset B_Y(\mathrm{U}(x), \varepsilon)$, where $B_X(x, \eta)$ denotes the ball of X of radius η centered at x. One characterization of U.S.C. maps is the following: U is U.S.C. if and only if the core of any open subset is open.

3.3 Interpolation

Following the conclusions of Section 2.3, we are going to immerse a discrete nD function defined on a cubical grid $u : \mathbb{Z}^n \to \mathbb{Z}$ into some larger spaces in order to get some continuity properties. For the *domain space*, we use the subdivision $X = \frac{1}{2}H^n$ of H^n. Every element $z \in \mathbb{Z}^n$ is mapped to an element $m(z) \in \frac{1}{2}H_1^n$ with $z = (z_1, \ldots, z_n) \longmapsto m(z) = \{z_1, z_1 + \frac{1}{2}\} \times \ldots \times \{z_n, z_n + \frac{1}{2}\}$. The definition domain of u, $\mathcal{D} \subseteq \mathbb{Z}^n$, has thus a counterpart in X, that will also be denoted $\mathcal{D}$, and that is depicted in bold in Figure 5. For the *value space*, we immerse $\mathbb{Z}$ (the set of pixel values) into the larger space $Y = \frac{1}{2}H^1$, where every integer becomes a closed singleton of H_0^1. Thanks to an "interpolation" function, we can now define from u a set-valued map $\mathrm{U} = \mathfrak{I}(u)$. We have $\mathrm{U} : X \rightsquigarrow Y$ and we set:

$$\forall\, h \in X,\ \ \mathrm{U}(h) = \begin{cases} \{\, u(m^{-1}(h)) \,\} & \text{if } \ h \in \mathcal{D} \\ \max(\, \mathrm{U}(h') : h' \in st(cl(h)) \cap \mathcal{D} \,) & \text{if } \ h \in \frac{1}{2}H_1^n \backslash \mathcal{D} \\ \mathrm{span}(\, \mathrm{U}(h') : h' \in st(h) \cap \mathcal{D} \,) & \text{if } \ h \in X \backslash \frac{1}{2}H_1^n. \end{cases} \qquad (1)$$

An example of interpolation is given in Figure 5. Actually, whatever u, such a discrete interpolation $\mathfrak{I}(u)$ can also be interpreted as a non-discrete set-valued map $\mathfrak{I}_{\mathbb{R}}(u) : \mathbb{R}^n \rightsquigarrow \mathbb{R}$ (schematically $\mathfrak{I}_{\mathbb{R}}(u)(x) = \mathfrak{I}(u)(h)$ with h such as $x \in \mathbb{R}^n$ falls in $h \in \frac{1}{2}H^n$), and we can show that $\mathfrak{I}_{\mathbb{R}}(u)$ is an U.S.C. map.

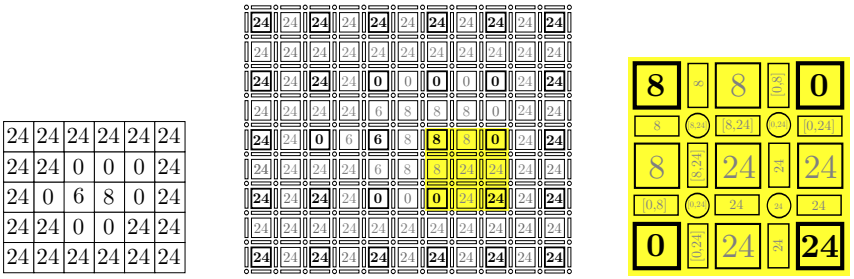

Fig. 5. The function $u : \mathbb{Z}^2 \to \mathbb{Z}$ (left) is transformed into the set-valued map $\mathrm{U} : \frac{1}{2}H^2 \rightsquigarrow \frac{1}{2}H^1$ (middle); d-faces with $d \in \{0, 1\}$ are interval-valued in U with the span of their respective $(d+1)$-face neighbors (right).

To the authors knowledge the notion of cuts (or thresholds) have not been defined for set-valued maps. Since they are of prime importance for mathematical morphology, and for the tree of shapes in particular, we propose in this paper the following definitions. Given $\lambda \in Y$, let us state that $[\,\mathrm{U} \lhd \lambda\,] = \{\, x \in X \mid \forall\, \mu \in \mathrm{U}(x),\ \mu < \lambda \,\}$ and $[\,\mathrm{U} \rhd \lambda\,] = \{\, x \in X \mid \forall\, \mu \in \mathrm{U}(x),\ \mu > \lambda \,\}$. We can show [17] that, with those definitions, $\forall u$, $\forall \lambda$, $[\,\mathfrak{I}(u) \lhd \lambda\,]$ and $[\,\mathfrak{I}(u) \rhd \lambda\,]$ are well-composed [9]. That is, strict cut components and their complementary sets can be handled both with the same unique connectivity, c_{2n}. As a consequence, the operators *star* and Sat commute on those sets, and we can prove [17] that:

$$\mathfrak{S}_{\mathfrak{I}}(u) = \{\, \mathrm{Sat}_{c_{2n}}(\Gamma);\ \Gamma \in \{\mathcal{CC}_{c_{2n}}([\,\mathfrak{I}(u) \lhd \lambda\,])\}_\lambda \cup \{\mathcal{CC}_{c_{2n}}([\,\mathfrak{I}(u) \rhd \lambda\,])\}_\lambda \,\}$$

is a set of components that forms a tree. Moreover, we can also prove that $\mathcal{T}_<(u) = \{\, \Gamma \cap \mathcal{D};\ \Gamma \in \{\, \mathcal{CC}_{c_{2n}}([\,\mathfrak{I}(u) \lhd \lambda + 1/2\,])\,\}_{\lambda \in H_0} \,\}$ and $\mathcal{T}_\geq(u) = \{\, \Gamma \cap \mathcal{D};\ \Gamma \in \{\, \mathcal{CC}_{c_{2n}}([\,\mathfrak{I}(u) \rhd \lambda + 1/2\,])\,\}_{\lambda \in H_0} \,\}$. So eventually we have: $\mathfrak{S}(u) = \{\, \Gamma \cap \mathcal{D};\ \Gamma \in \mathfrak{S}_{\mathfrak{I}}(u)\}$. That final property means that strict cuts of the interpolation of u, considering only c_{2n} for the different operators, allows for retrieving the shapes of u, as defined with the pair of dual connectivities c_{2n} and c_{3^n-1}.

4 Putting things altogether

4.1 About saturation and initialization

Classically the root node of the tree of shapes represents the whole image and, formally, the saturation operator is defined w.r.t. a point at infinity, p_∞, located outside the image domain $\mathcal{D}$. A rather natural idea is that the root level, ℓ_∞, should only depend on the internal border of $\mathcal{D}$ (which is unfortunately not the case for the algorithms proposed in the literature). To that aim, before interpolating u, we add to this image an external border with a unique value, ℓ_∞, set to the median value of the internal border. p_∞ is then one point from the added border.

4.2 Handling the hierarchical queue

To sort the faces of the domain X of U, we use a classical front propagation based on a hierarchical queue [15], denoted by q, the current level being denoted

Algorithm 2: Sorting for tree of shapes computation.

```
PRIORITY_PUSH(q, h, U, ℓ)
/* 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)

PRIORITY_POP(q, ℓ) : H
/* 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])
```

```
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
    return (R, u♭)
```

by ℓ. The sorting algorithm is given in Algorithm 2. There are two notable differences with the well-known hierarchical-queue-based propagation. First the d-faces, with $d < n$, are interval-valued so we have to decide at which (single-valued) level to enqueue those elements. The solution is straightforward: a face h is enqueued at the value of the interval $U(h)$ that is the closest to ℓ (see the procedure PRIORITY_PUSH). Just also note that we memorize the enqueuing level of faces thanks to the image $u^\flat$ (see the procedure SORT). Second, when the queue at current level, $q[\ell]$, is empty (and when the hierarchical queue q is not yet empty), we shall decide what the next level to be processed is. We have the choice of taking the next level, either less or greater than ℓ, such that the queue at that level is not empty (see the procedure PRIORITY_POP). Practically choosing going up or down the levels does not change the resulting tree since it just means exploring some sub-tree before some other disjoint sub-tree.

The result $\mathcal{R}$ of the sorting step is the one expected since the image U, in addition with the browsing of level in the hierarchical queue, allows for a propagation that is "continuous" both in domain space and in level space. An interesting property due to the interpolation and the well-composedness of cuts is that the neighborhood $\mathcal{N}$, used for faces in the propagation, corresponds to the c_{2n} connectivity on the Khalimsky's grid.

4.3 Max-tree versus tree of shapes computation

The main body of the tree of shapes computation algorithm is given in Algorithm 3. The major differences between this algorithm and the one dedicated

to the max-tree (see the procedure COMPUTE_TREE in Algorithm 1) are the following ones.

Algorithm 3: Tree of shapes computation in five steps.

```
COMPUTE_TREE_OF_SHAPES (u) : Pair(Array[P], T)
begin
    U ← INTERPOLATE(u)
    (R, u♭) ← SORT(U)
    parent ← UNION_FIND(R)
    CANONICALIZE_TREE(u♭, R, parent)
    return UN-INTERPOLATE(R, parent)
```

First the three basic steps (sort, union-find, and canonicalization) are now surrounded by an interpolation and un-interpolation process. Note that the un-interpolation just cleans up both $\mathcal{R}$ and *parent* to keep only elements of $\mathcal{D}$. Second, as emphasized in Section 2.2, the sorting step is of course dedicated to the tree of shapes computation. Last, a temporary image, $u^\flat$, is introduced. It is defined on the same domain as U, namely X, and contains only single-valued elements. This image is the equivalent of the original image u when dealing with the max-tree: it is used to know when an element h is canonical, that is, when $u^\flat(\mathit{parent}(h)) \neq u^\flat(h)$ (so that image is thus required by the canonicalization step that runs on X).

Complexity analysis of the algorithm presented here is trivial. The interpolation, canonicalization, and un-interpolation are linear. The modified union-find (once augmented with tree balancing, i.e., union-by-rank) is quasi-linear when values of the input image u have a low quantization (typically 12 bits or less). Last, the time complexity of the sorting step is governed by the use hierarchical queue: it is linear with low quantized data[8]. Eventually we obtain a quasi-linear algorithm. The representation of the tree with the pair ($\mathcal{R}$, *parent*) allows for any manipulation and processing that one expects from a morphological tree [3].

5 Related works

The first known algorithm, the "Fast Level Line Transform (FLLT)" [16], computes the max-tree and the min-tree of an image and obtains the tree of shapes by merging both trees. The main drawback of the FLLT is the need to know that a component has an hole (in order to match it with a component of the other tree). To that aim the Euler characteristic is computed, which can be done *locally* (while following the border of components) but in 2D only. In [4,14] the authors show that this fusion approach is sound in nD with $n > 2$; yet it cannot be effective in practice due to unacceptable complexity.

[8] Formally the sorting step has the pseudo-polynomial $O(k\,n)$ complexity, k being the number of different gray values. Though, since we consider low bit-depths data, k shall only be considered as a complexity multiplicative factor.

In [5] the "Fast Level Set Transform" (FLST) relies on a region-growing approach to decompose the image into shapes. It extracts each branch of the tree starting from the leaves and growing them up to the root until at least one saddle point is encountered. Each time a saddle point is encountered, the branch extraction procedure has to stop until every parallel branch meeting at this point is extracted. So each saddle point invalidates the shape currently being extracted, forcing the algorithm to visit its pixels again once a parallel branch is extracted. Since an image like a checkerboard contains $\mathrm{O}(n)$ saddle points meeting on $\mathrm{O}(n)$ pixels, the FLST has a $\mathrm{O}(n^2)$ worst case time complexity.

Song [18] takes a top-down approach to build the *tree of level lines* in $\mathrm{O}(n+t)$ time, where t is the total length of all level lines (note that filling the interior of each level line allows for retrieving the tree of shapes). The algorithm is restricted to 2D images with hexagonal pixels. Its key idea is to perform a recursion (starting from the image boundary): for a given component, follow every contours of its holes, and repeat this procedure for each hole component. Since the total length of level lines of an image can be of order $\mathrm{O}(n^2)$, the worst case has a quadratic-time complexity.

6 Conclusion

In this paper, we have presented a new algorithm to compute the tree of shapes of an image which features a quasi-linear time complexity, runs on nD images, and benefits from a much simpler implementation than existing algorithms. We have also proposed a novel representation of images as set-valued maps which has some continuity properties while remaining discrete.

Actually we believe that this representation is a good start to get a *"pure" self-duality* for images and operators, that is, a way to get rid of the pair of dual connectivities c_{2n} and c_{3^n-1}, and of the dissymmetry of cuts (strict and large cuts for respectively lower and upper cuts). In particular, replacing the maximum operator by the median operator in Equation 1 leads to a pure self-dual definition of the tree of shapes of 2D images [17]. Furthermore the perspectives offered by that new representation might be far from being limited to the tree of shapes computation.

For our experiments we use our free software library [10]; in particular, the fact that our tool makes it easy to write generic software in the case of mathematical morphology and discrete topology is discussed in [11]. The work presented here will be available in the next release of our software for we advocate reproducible research.

Acknowledgements. The authors would like to thanks Michel Couprie and Jean Cousty for fruitful discussions. This work received funding from the Agence Nationale de la Recherche, contract ANR-2010-BLAN-0205-03 and through "Programme d'Investissements d'Avenir" (LabEx BEZOUT n°ANR-10-LABX-58).

References

1. Aubin, J.P., Frankowska, H.: Set-Valued Analysis. Modern Birkhäuser Classics, Birkhäuser (2008)

2. Berger, C., Géraud, T., Levillain, R., Widynski, N., Baillard, A., Bertin, E.: Effective component tree computation with application to pattern recognition in astronomical imaging. In: Proceedings of ICIP. vol. 4, pp. 41–44 (2007)
3. Carlinet, E., Géraud, T.: A (fair?) comparison of many max-tree computation algorithms. In: Proceedings of ISMM (2013), This volume
4. Caselles, V., Meinhardt, E., Monasse, P.: Constructing the tree of shapes of an image by fusion of the trees of connected components of upper and lower level sets. Positivity 12(1), 55–73 (2008)
5. Caselles, V., Monasse, P.: Geometric Description of Images as Topographic Maps, Lecture Notes in Mathematics Series, vol. 1984. Springer (2009)
6. Géraud, T.: Ruminations on tarjan's union-find algorithm and connected operators. In: Proceedings of ISMM. CIVS, vol. 30, pp. 105–116. Springer (2005)
7. Géraud, T., Talbot, H., Van Droogenbroeck, M.: Mathematical Morphology—From Theory to Applications, chap. 12, pp. 323–353. ISTE & Wiley (2010)
8. Henle, M.: A Combinatorial Introduction to Topology. Dover Publications Inc. (1994)
9. Latecki, L., Eckhardt, U., Rosenfeld, A.: Well-composed sets. Computer Vision and Image Understanding 61, 70–83 (1995)
10. Levillain, R., Géraud, T., Najman, L.: Why and how to design a generic and efficient image processing framework: The case of the Milena library. In: Proceedings of ICIP. pp. 1941–1944, `http://olena.lrde.epita.fr` (2010)
11. Levillain, R., Géraud, T., Najman, L.: Writing reusable digital geometry algorithms in a generic image processing framework. In: Applications of Discrete Geometry and Mathematical Morphology. LNCS, vol. 7346, pp. 96–100. Springer-Verlag (2010)
12. Mazo, L., Passat, N., Couprie, M., Ronse, C.: Digital imaging: A unified topological framework. Journal of Mathematical Imaging and Vision 44(1), 19–37 (2012)
13. Meijster, A., Wilkinson, M.H.F.: A comparison of algorithms for connected set openings and closings. IEEE Transactions on Pattern Analysis and Machine Intelligence 24(4), 484–494 (2002)
14. Meinhardt-Llopis, E.: Morphological and Statistical Techniques for the Analysis of 3D Images. Ph.D. thesis, Universitat Pompeu Fabra, Spain (March 2011)
15. Meyer, F.: Un algorithme optimal de ligne de partage des eaux. In: Actes du 8e congrès AFCET. pp. 847–859 (1991)
16. Monasse, P., Guichard, F.: Fast computation of a contrast invariant image representation. IEEE Transactions on Image Processing 9(5), 860–872 (May 2000)
17. Najman, L., Géraud, T.: Discrete set-valued continuity and interpolation. In: Proceedings of ISMM (2013), This volume
18. Song, Y.: A topdown algorithm for computation of level line trees. IEEE Transactions on Image Processing 16(8), 2107–2116 (August 2007)
19. Tarjan, R.E.: Efficiency of a good but not linear set union algorithm. Journal of the ACM 22(2), 215–225 (1975)
20. Xu, Y., Géraud, T., Najman, L.: Context-based energy estimator: Application to object segmentation on the tree of shapes. In: Proceedings of ICIP (2012)
21. Xu, Y., Géraud, T., Najman, L.: Morphological filtering in shape spaces: Applications using tree-based image representations. In: Proceedings of ICPR (2012)
22. Xu, Y., Géraud, T., Najman, L.: Two applications of shape-based morphology: Blood vessel segmentation and generalisation of constrained connectivity. In: Proceedings of ISMM (2013), This volume