

Morphological Hierarchical Image Decomposition Based on Laplacian 0-Crossings

Lê Duy Huỳnh¹, Yongchao Xu^{1,2}, and Thierry Géraud^{1†}

¹ EPITA Research and Development Laboratory (LRDE)

² Institut Mines Telecom, Telecom ParisTech
theo@lrde.epita.fr

Abstract. A method of text detection in natural images, to be turned into an effective embedded software on a mobile device, shall be both efficient and lightweight. We observed that a simple method based on the morphological Laplace operator can do the trick: we can construct in quasi-linear time a hierarchical image decomposition / simplification based on its 0-crossings, and search for some text in the resulting tree. Yet, for this decomposition to be sound, we need “0-crossings” to be Jordan curves, and to that aim, we rely on some discrete topology tools. Eventually, the hierarchical representation is the morphological tree of shapes of the Laplacian sign (ToSL). Moreover, we provide an algorithm with linear time complexity to compute this representation. We expect that the proposed hierarchical representation can be useful in some applications other than text detection.

Keywords: morphological Laplace operator · well-composed images · tree of shapes · hierarchical decomposition · text detection

1 Introduction

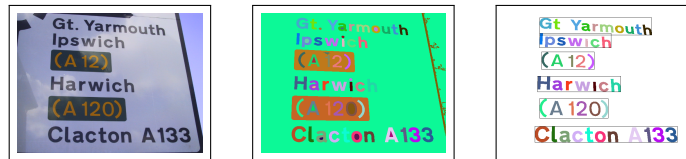


Fig. 1. A hierarchical image decomposition (center) leading to text detection (right).

In [7] we have proposed a method to perform robust text segmentation in natural images; see Fig. 1 for an illustration. This method is interesting for several reasons. **1.** To select candidate regions for characters, we rely on the morphological Laplace operator. We can observe that this operator is relatively well robust to poor contrast and uneven illumination; the experimental results show that it outperforms the “more classical” component-based methods—namely the

[†] This work has been conducted in the context of the MOBIDEM project, part of the “Systematic Paris-Region” and “Images & Network” Clusters (France). This project is partially funded by the French Gov. and its economic development agencies.

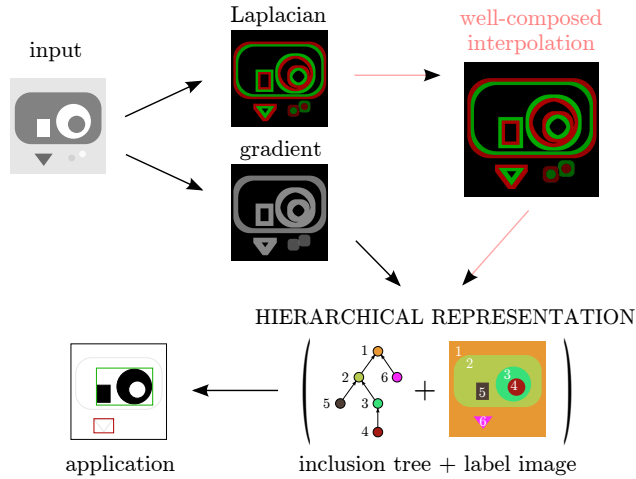


Fig. 2. Overview of the proposed method to get a hierarchical image decomposition.

Stroke Width Transform (SWT), the Toggle Mapping Morphological Segmentation (TMMS), and the Maximally Stable Extrema Region (MSER) methods—see [7] for details. **2.** Based on the 0-crossings of the morphological Laplace operator, we compute a hierarchical representation of the image contents, so that regions form an inclusion tree. Such a structure is great because we have a strong simplification of the image contents, and because dealing with a simple tree structure allows for powerful decision taking when grouping regions / characters into text boxes. **3.** This text segmentation method is suitable to real-time implementation on mobile devices. Indeed computing the hierarchical representation eventually translates to a *simple labeling algorithm* that gives both an inclusion tree and a label image; in addition, identifying and grouping text components is achieved by an easy tree-based processing.

In this present paper we focus on two points which are not detailed in [7]: how we can rely on some discrete topology tools to get a sound definition of a hierarchical decomposition of an image into regions, and how we take benefit from some properties to get an efficient algorithm to compute such a representation.

Section 2 starts by giving an overview of the hierarchical representation and of the segmentation method. Section 3 recalls some theoretical background, and Section 4 details the algorithm that computes the hierarchical representation. Last we conclude in Section 5.

2 Method Overview

The method that we propose to represent an image by a hierarchical decomposition and to extract text in natural images is very simple: put shortly, text components are selected among the 0-crossing lines of the morphological Laplace operator of the gray-level input image. The method is composed of several steps, depicted in Fig. 2, and described and justified just below.

We start with a gray-level input image. If the primary data is a color image, we just take the luminance value of its pixels (we lose color information but it almost never negatively affects text retrieval). We then compute its morphological dilation δ_{\square} and erosion ε_{\square} to directly deduce two images. First, we obtain the morphological thick gradient $\nabla_{\square} = \delta_{\square} - \varepsilon_{\square}$; it is used later to discard contours that are not enough contrasted. Second, we obtain the morphological Laplace operator $\Delta_{\square} = \delta_{\square} + \varepsilon_{\square} - 2 \text{id}$. The text character boundaries are expected to belong to the 0-crossing contours of this non-linear operator; actually they are, and their localization is precise.

We *virtually*³ compute a particular interpolated image, $\Delta_{\square}^{\text{wc}}$, of the Laplacian image Δ_{\square} , having 4 times more pixels than the original. This resulting image is *well-composed*, meaning that the boundaries of every components of any threshold set are Jordan curves. As a consequence, the (boundaries of the) 0-crossings are simple closed curves: they cannot have the shape of a ‘8’. In addition, they are disjoint, and this set of curves can be organized in an inclusion tree.

Due to the fact that $\Delta_{\square}^{\text{wc}}$ is well-composed, the regions delimited by the 0-crossings can be labeled very efficiently (by the classical blob labeling algorithm), and their inclusion tree is built. In addition, many 0-crossings are discarded on the fly during the labeling, because they are not contrasted enough (we use ∇_{\square}), or because they do not satisfy some geometrical criteria (when they are too small for instance). The result “inclusion tree + label image” is the hierarchical decomposition of the image contents into regions; it is depicted on the bottom-right part of Fig. 2. We end up with two structures: the inclusion tree, encoded by a parenthesis relationship between labels, and an image of labels, so that each pixel is assigned to a region. In addition, we have some additional information related to every regions, that are computed during the labeling: the region area, its bounding box, etc.

From the resulting hierarchical representation, we have derived an application described in [7]: text detection. Indeed, we can group regions, that are separated components, together to form text boxes. For that, we only consider the bottom of this tree (the leaves and sometimes their parent): for each component, we search spatially in the label image what are their left and right components to be grouped into a text box. In this step, we highly take advantage of the tree structure: it allows very easily to discard many regions as non-text, and to determine if a leaf region is a character hole or a plain character.

3 Theoretical Background

3.1 Khalimsky’s 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

³ We will see later that we actually do *not* interpolate the Laplacian image, but proceed *as if* there were an interpolation. Practically, it means that we avoid the need of multiplying by 4 the number of pixels in the process.

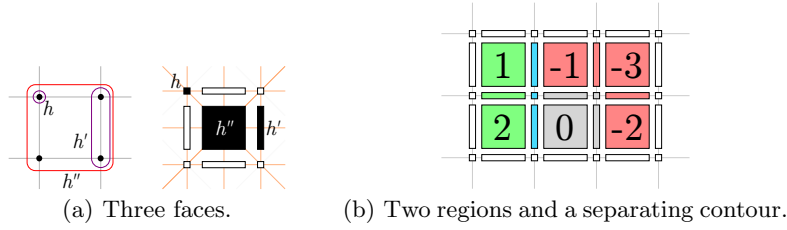


Fig. 3. Representation of images on the Khalimsky's grid.

h . The set of all faces, H^n , is called the n D space of cubical complexes. Fig. 3(a) depicts a set of faces $\{h, h', h''\} \subset H^2$ where $h = \{0\} \times \{1\}$, $h' = \{1\} \times \{0, 1\}$, and $h'' = \{0, 1\} \times \{0, 1\}$; the dimension of those faces are respectively 0, 1, and 2. The set of faces of H^n is arranged onto a grid, called the Khalimsky's grid [8], depicted in orange in Fig. 3(a) (right). Consequently, it means that we just can store values defined on such a grid in a simple matrix structure.

This grid is a way to **1.** get a discrete topology, and **2.** represent what lies in-between pixels, for instance contours. In Fig. 3(b), an image is depicted with integer values. The green and red regions are open sets corresponding to components having respectively positive and negative values. A 1D contour (thus with 0-faces and 1-faces only) is depicted in blue: it is precisely the discrete boundary of the non-positive region on the right.

3.2 Morphological Laplace Operator

In order to detect text in images, many methods first look for candidate regions for characters. A seminal method, based on contour detection, is to consider the 0-crossings of a discrete Laplace operator: given a gray-level image u , the contours of interest are given by $\Delta u = u_{xx} + u_{yy} = 0$. This method is interesting for several reasons: **1.** it is a very simple approach; **2.** it provides closed contours; **3.** labeling the components of the image having the same sign, resp. positive and negative; gives a segmentation; **4)** it is self-dual, i.e., it processes dark objects and bright ones the same way.

The morphological Laplace operator has been defined in [11] by $\Delta_{\square} = (\delta_{\square} - \text{id}) - (\text{id} - \varepsilon_{\square})$, relying on the elementary dilation (δ) and erosion (ε) morphological operators. Actually, the morphological non-linear version features a much higher fidelity to actual object contours than the linear version; in addition, salient contours hardly depends on the size of the structuring element.

The image depicted in Fig. 4(a) has been created by the authors of [1] to make classical binarization methods fail, due to the presence of uneven illumination. On Fig. 4(c), one can observe that the object boundaries belong to the 0-crossings of the morphological Laplace operator.

3.3 Tree of Shapes

The tree of shapes is a morphological self-dual representation of an image; see [6] for history, implementation, and references. This tree encodes the inclusion of

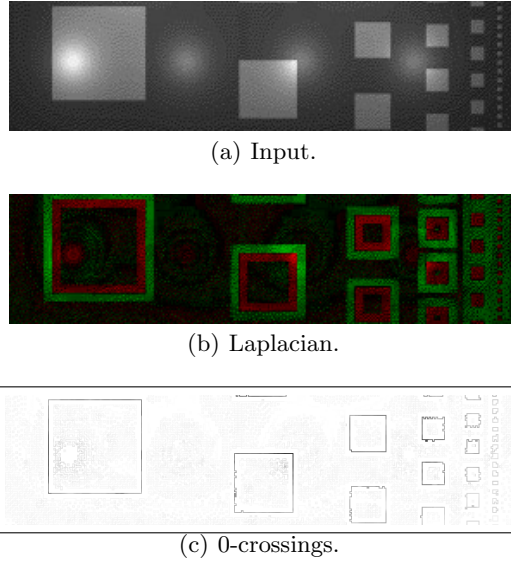




Fig. 4. Contour characterization by morphological operators; 4(b) is the morphological Laplacian $\Delta_B = \delta_B + \varepsilon_B - 2\text{id}$; 4(c) depicts on $\Delta_B = 0$ the gradient values $\nabla_B = \delta_B - \varepsilon_B$ (inverted) showing that the contours of actual objects are effectively salient.

the level sets, i.e., the connected components obtained by thresholding. An illustration is given on a very simple image by Fig. 5(a). The node \mathbf{B} lists the pixels of the region \mathbf{B} . The contour of this node is a level line, and the component, called *shape*, associated with this node is actually the sub-tree rooted at this node. The node \mathbf{B} thus represents the shape given by $\mathbf{B} \cup \mathbf{D} \cup \mathbf{E}$. In [4] the reader can find more details about how to store and process efficiently such a tree structure.

With X representing a 2D discrete space, given an image $u : X \rightarrow \mathbb{Z}$ and any scalar $\lambda \in \mathbb{Z}$, the lower and upper level sets are respectively defined by $[u < \lambda] = \{x \in X \mid u(x) < \lambda\}$ and $[u \geq \lambda] = \{x \in X \mid u(x) \geq \lambda\}$. Using \mathcal{CC} , the operator that takes a set and gives its set of connected components, and Sat , the cavity-fill-in operator, we can define: $\mathfrak{S}(u) = \{\text{Sat}(\Gamma); \Gamma \in \mathcal{CC}([u < \lambda]) \cup \mathcal{CC}([u \geq \lambda])\}_\lambda$, called the tree of shapes of u .

3.4 Well-Composed Sets and Images

A sub-class of sets defined on the cubical grid, called *well-composed*, has been proposed in [10], where all connectivities are equivalent, thus avoiding many topological problems. A very easy characterization of well-composed sets is based on the notion of “critical configurations”; in the 2D case, a set is well-composed if the configurations  and  do not appear. The notion of well-composedness has been extended in [9] from sets to functions, i.e., gray-level images, and has been generalized in the n D case in [3]. A gray-level image u is well-composed if any set $[u \geq \lambda]$ is well-composed. A straightforward character-

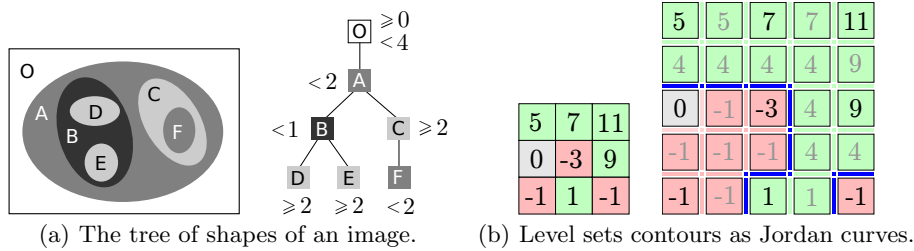


Fig. 5. Topological representations and some topological considerations.

ization of well-composed 2D gray-level images is that every block $\begin{bmatrix} a & d \\ c & b \end{bmatrix}$ should verify: $\text{intvl}(a, b) \cap \text{intvl}(c, d) \neq \emptyset$, where $\text{intvl}(v, w) = [\min(v, w), \max(v, w)]$. An image is not *a priori* well-composed. To get a well-composed image from a primary image, one can compute an interpolation of the primary image that is well-composed. Fig. 5(b) gives an example of an image which is not well-composed (left), but whose interpolation is well-composed (right). In [2] the authors have proposed a simple method to get a self-dual well-composed interpolation, that makes sense when considering the tree of shapes of an image [5].

3.5 Putting Things Together

As explained in Section 2, from a gray-level input image u , we consider the Laplacian image once well-composed; both images are depicted respectively in Figs. 6(a) and 6(b). The tree of shapes of the Laplacian *sign* image (ToSL), $\mathfrak{S}(\text{sign}(\Delta_{\square}^{\text{wc}}(u)))$, contains nodes corresponding to positive, negative, and null regions; it is depicted in Fig. 6(d).

Yet the 0-crossings, corresponding to nodes at level 0 (depicted in white), can be “thick”, that is, they can contain pixels (2-faces) instead of being only defined by 1D objects (set of 0-faces and 1-faces). In Fig. 6(c) for instance, we can see that some pixels of the ‘s’ contour belong to the 0-crossing region. Since we want to obtain a partition of the image into “positive” and “negative” regions, we merge null regions with their parent regions; that way, instead of the tree of shapes depicted in Fig. 6(d), we consider the simplified one, depicted in Fig. 6(e). Eventually, the actual “0-crossings” we are looking at are the boundaries of the shapes of this final tree, so they are effectively 1D objects. It is illustrated in Fig. 5(b) (and also in Fig. 3(b)): the null region is merged with the negative one, so we consider the blue contour to be the 1D “0-crossing” separating regions having different signs.

In the next section, to compute the hierarchical representation, we do not consider that we have a cubical complex as the space structure: we just ignore that 0-faces and 1-faces exist. Though, from a theoretical point of view, the contours / 0-crossings expressed in terms of 0-faces and 1-faces really are Jordan curves.

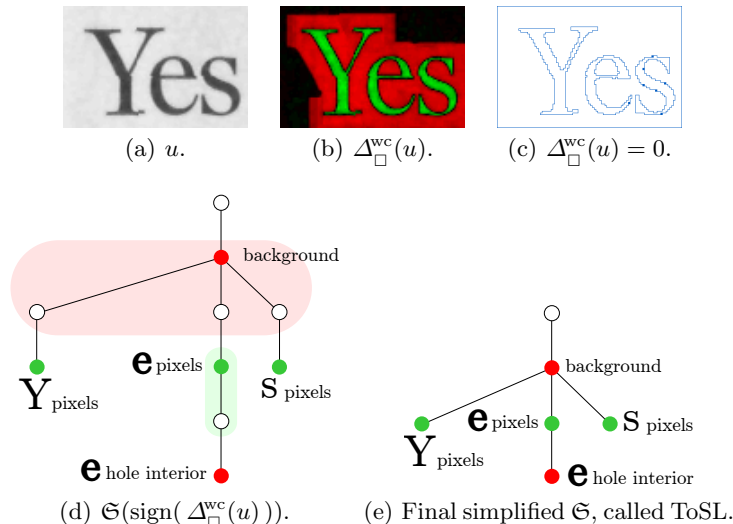


Fig. 6. Tree of shapes of Laplacian sign (ToSL): positive and negative regions are respectively green and red nodes of the ToS, and null regions are white nodes.

4 Fast Computation of the Hierarchical Representation

4.1 A Particular Well-Composed Non-Local Interpolation

In the following, we do not need to consider the cubical complex, so we only deal with pixels; they are, for instance, the valued 2-faces in Fig. 5(b) (right).

The hierarchical representation is computed from an interpolated image, $\Delta_{\square}^{\text{wc}}(u)$, which is a very particular well-composed version of the Laplacian image $\Delta_{\square}(u)$. This particular interpolation takes its origin from the work in [6], and is detailed in [2]. Briefly put, it is a *non-local* interpolation driven by a propagation from the border of the image, which browses the nodes of its tree of shapes from the root to the leaves. The interpolated pixels are assigned with the current gray-level value, which evolves “continuously” during the process. This interpolation has two important features [5]: it is related to the tree of shapes of the input image, so it actually follows the same scheme as a blob labeling algorithm where a blob would be a level set, and its topological behavior is deterministic.

There are two main consequences: this particular interpolation makes sense in the present context, since we want to label regions that are in an inclusion relationship, and we can optimize the computation of the hierarchical representation (the label image and the inclusion tree) by actually *emulating* the interpolation.

An example on the image of Laplacian sign given in Fig. 7(a) is depicted in Fig. 7(b)⁴. We can observe in Fig. 7(b) that we obtain the desired properties:

⁴ Note that the two identical local configurations enclosed by red rectangles in Fig. 7(a) do not lead to the same interpolation; this is due to the non-local interpolation process that depends of the outer region, which is different in the two cases: respectively negative for the top configuration, and positive for the bottom one.

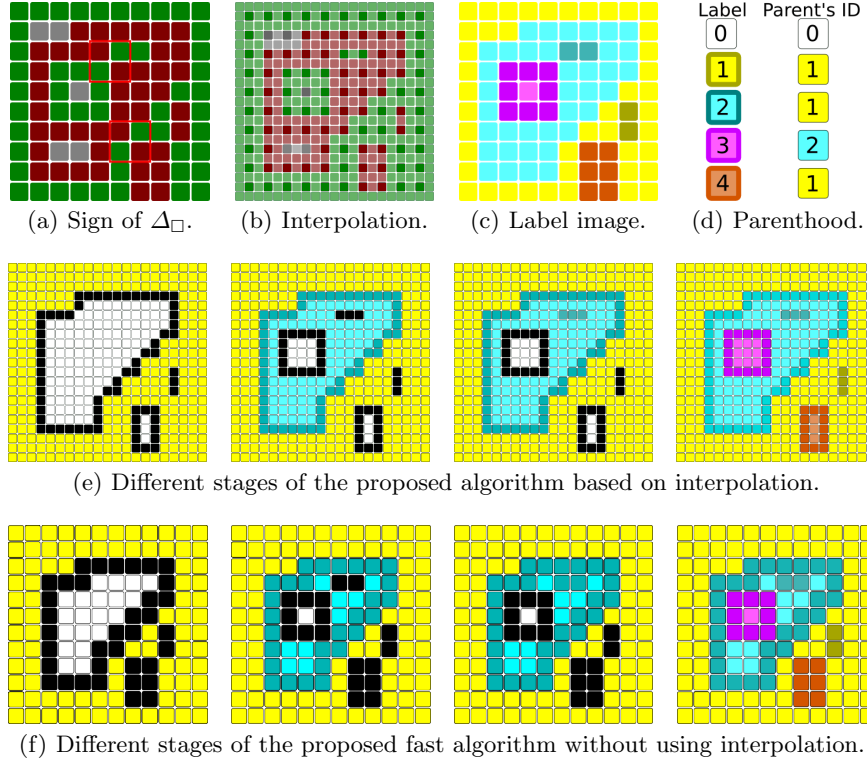


Fig. 7. An example of the proposed labeling algorithm. Black pixels: contours of components inside labeled ones. White pixels: pixels that are not yet labeled and are also not marked as inside contours at the current stage of the labeling process.

the boundaries of the regions are Jordan curves, and the regions are in an unambiguous spatial inclusion relationship. The inclusion tree of the Laplacian sign image, called ToSL, is thus a hierarchical image decomposition.

4.2 Labeling Interpolated Sign of Δ_{\square} to Construct ToSL

Thanks to the fact that $\Delta_{\square}^{\text{wc}}$ is well-composed, the regions delimited by the 0-crossings can be labeled very efficiently (by the classical blob labeling algorithm), and their inclusion tree is built. This resulting inclusion tree is the tree of shapes of the sign of the Laplacian, a ternary-valued image—with pixels set to -1 (red), 0 (gray), or 1 (green) as depicted in Fig. 7. The whole labeling process is depicted by the black part in Algorithm 1. It computes a label image \mathcal{L} that labels all components delimited by the 0-crossings of Δ_{\square} and a tree structure encoded in an array of parenthood *parent* (e.g., having $\text{parent}(l_1) = l_2$ means that the region with label l_1 is included in the region with label l_2). *Q* is a queue of pixels, *border* is an auxiliary image that marks the inner component contours as active

Algorithm 1 Computation of ToSL by labeling interpolated image (black part) and its fast version (adding red part) without using interpolation.

<pre> LABELING ($\Delta_{\square}, \nabla_{\square}$) 1: for all p do 2: $\mathcal{L}(p) \leftarrow 0$ 3: $border(p) \leftarrow \text{undef}$ 4: $nlabels \leftarrow 1$ 5: for all p do 6: if $\mathcal{L}(p) \neq 0$ then 7: continue 8: if $p = p_0$ then 9: $\ell \leftarrow 1$ 10: $parent[\ell] = 1$ 11: else 12: $\mathcal{P} \leftarrow \text{FOLLOW_CONTOUR}(p)$ 13: if $\text{evaluate}(\mathcal{P})$ then 14: $nlabels \leftarrow nlabels + 1$ 15: $\ell \leftarrow nlabels$ 16: $parent[\ell] \leftarrow \mathcal{L}(p_{-1})$ 17: else 18: $\ell \leftarrow \mathcal{L}(p_{-1})$ 19: BLOB_LABELING (p, ℓ) 20: return $parent, \mathcal{L}$ </pre>	<pre> BLOB_LABELING (p, ℓ) 21: $\mathcal{L}(p) \leftarrow \ell, Q.push(p)$ 22: while Q is not empty do 23: $q \leftarrow Q.pop(), \mathbb{N} \leftarrow \mathbb{N}_4$ 24: if $border(q) = \text{undef}$ then 25: $\mathbb{N} \leftarrow \mathbb{N}_8$ /*optimized version*/ 26: for all $n \in \mathbb{N}(q)$ do 27: if $\mathcal{L}(n) = 0$ and $\Delta_{\square}(p) \times$ $\Delta_{\square}(n) \geq 0$ then 28: $\mathcal{L}(n) \leftarrow \ell, Q.push(n)$ 29: else 30: $border(n) \leftarrow a$ </pre> <pre> FOLLOW_CONTOUR (p) 31: $\mathcal{P}.init(), border(p) \leftarrow \bar{a}, Q.push(p)$ 32: while Q is not empty do 33: $q \leftarrow Q.pop(), \mathcal{P}.update(q)$ 34: for all $n \in \mathbb{N}(q)$ do /*\mathbb{N}_8 or \mathbb{N}_4*/ 35: if $border(n) = a$ then 36: $Q.push(n), border(n) \leftarrow \bar{a}$ 37: return \mathcal{P} </pre>
---	---

=0

a or inactive \bar{a} state, $nlabels$ is the temporary number of labels, ℓ is the current label value, and \mathbb{N} represents either 4-connectivity (\mathbb{N}_4) or 8-connectivity (\mathbb{N}_8).

The core of the algorithm is an alternate application of the following two routines (depicted on the right side of Algorithm 1). BLOB_LABELING is a classical queue-based “blob labeling” algorithm that labels the underlying connected component with current label value ℓ , and also marks the inner component contours as active state a . Note that actually we do not want regions representing null values in the final tree, so we merge nodes corresponding to 0-crossings with their parent (see line 27 and Fig. 6). FOLLOW_CONTOUR is also a queue-based process which is similar with previous “blob labeling” algorithm, but applied on the underlying active contour of an unlabeled connected component. Instead of labeling the active contour, this routine browses it and marks it as inactive \bar{a} (see line 31 and 36), and collects a measure \mathcal{P} (e.g., length) characterizing the contour. Note that both these two routines are very efficient thanks to the queue-based “blob labeling”.

More precisely, for the complete algorithm depicted on the left side of Algorithm 1, we browse the pixels in raster scan order (main loop, line 5). when we reach an unlabeled pixel p , if this is the first pixel p_0 in the scan order (i.e., the top left pixel), we know that its label value is 1, and the label value of its parent (i.e., itself) is also 1 (line 8 to 10), because p_0 is in the root node thanks to the added external boundary described in previous section. For all other unlabeled pixel $p \neq p_0$, we follow the contour of the unlabeled region, which is a hole in

the label image, thanks to the *border* image. The FOLLOW_CONTOUR routine computes on the fly a contour-based measure \mathcal{P} characterizing the hole, such as the average of gradient’s magnitude along the contour and the bounding box of the hole. If this contour-based measure \mathcal{P} does not satisfy some given criterion, for instance, it is not contrasted enough, or it does not satisfy some geometrical criteria (when the hole is too small for instance), we discard this region by not creating a new label value for this unlabeled region. Let p_{-1} be the pixel just before p in the raster scan order (p_{-1} is guaranteed to be labeled), we assign the label value of p_{-1} to the underlying hole region, which means we merge it with its parent region. If the contour measure \mathcal{P} satisfies the corresponding criterion, we create a new label value to label the hole region, and update the parenthood relationship of this new label value to $\mathfrak{L}(p_{-1})$ (see line 13 to line 18). Then we proceed the routine BLOB_LABELING to label the connected set of pixels having the same Laplacian sign as p or being null and update the auxiliary *border* image. Note that since we use the 4-connectivity neighborhood (\mathbb{N}_4) in the routine BLOB_LABELING to label regions and mark neighboring pixels having different sign of Δ_{\square} , we need to use the 8-connectivity neighborhood (\mathbb{N}_8) to follow completely the active contour of an unlabeled region (see also the longest contour in the left image in Fig. 7(e)).

An example of the proposed algorithm on the interpolated image in Fig. 7(b) is depicted in Fig. 7(c-e). Different stages of the algorithm are depicted in Fig. 7(e). Note that the pixels having null Laplacian sign are grouped with the parent region. The two small regions inside cyan and respectively yellow region are also discarded, they are grouped with the parent region. The resulting “label image + tree” are depicted in Fig. 7(c) and 7(d).

4.3 Optimization of ToSL Construction

The well-composed interpolation described in Section 4.1 resolves all the topological issues at critical configurations with the cost of quadrupling the number of pixels. Yet, in practice, we do not need to apply this interpolation, which can be emulated efficiently without subdivising the image domain thanks to the particular non-local way of interpolation described in Section 4.1.

The optimized version of the proposed algorithm by emulating the interpolation is depicted in Algorithm 1 by the black and red parts. More precisely, the used particular non-local interpolation is based on the inclusion relationship of components. The interpolated values at critical configurations are given by the values of outer region. Besides, the proposed algorithm for such interpolated image labels regions from outside to inside. Consequently, it is equivalent to use 8-connectivity (\mathbb{N}_8) when we use blob labeling algorithm to label pixels that are not yet activated thanks to the *border* image (see line 24 and line 25 in the scope of BLOB_LABELING on the right side of Algorithm 1). This makes the other two pixels of critical configurations having a different sign of Laplacian disjoint and marked as active contours. Consequently, when we proceed the FOLLOW_CONTOUR routine in the following, we need to use 4-connectivity (\mathbb{N}_4) (see line 34 in the scope of FOLLOW_CONTOUR in Algorithm 1) to avoid connecting two disjoint unlabeled regions.



Fig. 8. Qualitative results using “ICDAR 2015 Robust Reading” database: input (top), label image (middle), text boxes (bottom).

An example is given in Fig. 7(f) which shows different stages of the proposed optimized algorithm without using interpolation. Note that the green pixels (resp. red pixels) of the critical configuration in the top (resp. bottom) red rectangle in Fig. 7(a) are considered as connected using 8-connectivity (N_8) when we proceed the routine BLOB_LABELING. The active contours (black pixels) in Fig. 7(f) are processed using 4-connectivity (N_4). Both the algorithm depicted in Fig. 7(e) and Fig. 7(f) result in the same “label image + tree” depicted in Fig. 7(c) and Fig. 7(d).

4.4 Application to Text Segmentation and Detection

As an application, the ToSL has been used for text detection and segmentation; the results have been further detailed in [7]. Briefly put, we compute the ToSL, and then group nodes together to form candidates for text boxes.

During the FOLLOW_CONTOUR process, we discard some regions w.r.t. their contour average gradient magnitude $\bar{\nabla}$ (to keep only contrasted regions), the contour length (to remove noise, i.e., too small regions), and the bounding box size \mathbb{B} (to keep components that reasonably look like characters).

The ToSL is such that sibling nodes are objects sharing the same background in the original image. Consequently, the grouping process is performed efficiently because the tree structure helps to limit the sets of candidates that can be grouped together. We found that most of text components are leaves, and sometimes leaf parents (in the case of characters with holes). To group text components together into text boxes, starting from each leaf, we look in the image space, that is in the label image, for a sibling, and we control the grouping thanks to some geometric information.

Some results are depicted in Fig. 8; more details and a quantitative evaluation are given in [7].

5 Conclusion

In this paper, we have presented a hierarchical decomposition of the image contents into regions. Although we use the very “classical” idea of relying on the 0-crossings of the Laplace operator image to obtain the objects of interest, our version is innovative for several reasons. First, we rely on the morphological Laplace operator, which performs well in the case of uneven illumination, which is a common defect in natural images. Second, we ensure that the “0-crossings” are really 1D objects. For that, we use a well-composed Laplacian image, we compute the tree of shapes of its sign, and we consider the 1D topological boundary of shapes. As a consequence, the positive and negative regions can be organized into a tree without topological ambiguity. Last we present a linear time complexity algorithm, which is a hardly more sophisticated blob labeling algorithm, to compute the hierarchical structure. We also provide a way, directly during the computation process, to ignore some regions if they are not relevant, and an optimization that mimics well-composedness and then avoids to duplicate pixels.

References

1. Blayvas, I., Bruckstein, A., Kimmel, R.: Efficient computation of adaptive threshold surfaces for image binarization. *Pattern Recognition* 39(1), 89–101 (2006)
2. Boutry, N., Géraud, T., Najman, L.: How to make n D functions well-composed in a self-dual way. In: *Proc. of the Intl. Symp. on Mathematical Morphology (ISMM)*. LNCS, vol. 9082, pp. 561–572. Springer (2015)
3. Boutry, N., Géraud, T., Najman, L.: On making n D images well-composed by a self-dual local interpolation. In: *Proc. of the Intl. Symp. on Discrete Geometry for Computer Imagery (DGCI)*. LNCS, vol. 8668, pp. 320–331. Springer (2014)
4. Carlinet, E., Géraud, T.: A comparative review of component tree computation algorithms. *IEEE Transactions on Image Processing* 23(9), 3885–3895 (2014)
5. Géraud, T., Carlinet, E., Crozet, S.: Self-duality and discrete topology. In: *Proc. of the Intl. Symp. on Mathematical Morphology (ISMM)*. LNCS, vol. 9082, pp. 573–584. Springer (2015)
6. Géraud, T., Carlinet, E., Crozet, S., Najman, L.: A quasi-linear algorithm to compute the tree of shapes of n -D images. In: *Proc. of the Intl. Symp. on Mathematical Morphology (ISMM)*. LNCS, vol. 7883, pp. 98–110. Springer (2013)
7. Huynh, L.D., Xu, Y., Géraud, T.: Morphology-based hierarchical representation with application to text segmentation in natural images. In: *Proc. of the Intl. Conf. on Pattern Recognition (ICPR)* (2016), <http://www.lrde.epita.fr/~theo/papers/huynh.2016.icpr.pdf>, to appear.
8. Khalimsky, E., Kopperman, R., Meyer, R.: Computer graphics and connected topologies on finite ordered sets. *Topology and its Applications* 36, 1–17 (1990)
9. Latecki, L.: 3D well-composed pictures. *Graphical Models and Image Processing* 59(3), 164–172 (1997)
10. Latecki, L., Eckhardt, U., Rosenfeld, A.: Well-composed sets. *Comp. Vis. and Image Understanding* 61(1), 70–83 (1995)
11. van Vliet, L., Young, I., Beckers, G.: An edge detection model based on non-linear laplace filtering. In: *Proc. of Intl. Workshop on Pattern Recognition and Artificial Intelligence*. pp. 63–73 (1988)