Journal of Real-Time Image Processing (2021) 18:99–112
https://doi.org/10.1007/s11554-020-00951-6

ORIGINAL RESEARCH PAPER

Real-time maximally stable homogeneous regions

Tobias Böttger[1]

Received: 10 July 2019 / Accepted: 29 January 2020 / Published online: 20 February 2020

Abstract
In this paper, we present the concept of maximally stable homogeneous regions (MSHR). MSHR are conceptually very similar to maximally stable extremal regions but can be segmented in images with an arbitrary number of channels. The computation of the presented MSHR relies on the construction of a quasi-flat zone hierarchy. We present a fast algorithm for computing the hierarchy that overcomes the runtime restrictions of existing approaches. The proposed algorithm can construct the quasi-flat zone hierarchy efficiently in real time, scales linearly in the number of pixels and, in practice, sub-linearly in the number of channels. In the experiments, we display how MSHR can be used to improve the results of optical character recognition systems and to perform 3D object segmentation. We further demonstrate the universality and speed of the proposed algorithm for three example applications: image segmentation, object tracking, and image filtering.

Keywords Quasi-flat zone hierarchy · Component-trees · Multi-valued component-trees · Image segmentation · Object tracking

1 Introduction

The component-tree (also known as dendrone [11], confinement tree [29] or max-tree [7]) is a hierarchical data structure that models gray-scale images by considering the connected components of their binary level sets obtained from successive thresholdings [26]. It has a wide range of applications: image filtering [21, 41], motion extraction [41], feature and region extraction with maximally stable extremal regions (MSER) [28], astronomical imagery [2], object tracking [3, 14] and 3D visualization [46]. For gray-scale images, efficient algorithms exist that enable the construction of the component-tree in linear time [7].

The success of component-trees for gray-scale images and the increasing demand for image processing techniques devoted to color, motivates their extension to multi-channel images. A common approach to computing component-trees for multi-channel images are those based on a component-trees computed on an edge-weighted graph [13]. The respective approaches create morphological hierarchies of connected pixels [43], also known as quasi-flat zone hierarchies [30, 31]. The respective hierarchies are very general and algorithms devoted to the gray-value component-tree can be adapted to the more general quasi-flat zone hierarchy.

In this work, we present an efficient algorithm to compute the quasi-flat zone hierarchy based on a local flooding immersion. The algorithm is truly linear in the number of pixels and considerably faster than the existing union-find-based algorithms in the experiments. The hierarchy construction is generic and does not require the definition of a partial or total ordering relation. We further introduce the concept of maximally stable homogeneous regions (MSHR), which are conceptually very similar to MSER but can be computed for multi-channel images. They can be used for OCR applications, where MSER character extraction fails, as is displayed in Fig. 1. As for MSER, the runtime of MSHR is linear in the number of pixels and furthermore scales sub-linearly with the number of image channels. We display the universality of the proposed tree construction algorithm and of MSHR by presenting further example applications: 3D object segmentation, object tracking and image filtering.

This work extends our previous work [3], which introduces object tracking in component-trees. This paper additionally presents the flooding-based quasi-flat zone hierarchy construction for multi-channel images in a profound manner and highlights implementation details to enable the tree to be constructed in real time. Furthermore, to highlight the universality and speed of the algorithm, we evaluate the

✉ Tobias Böttger
boettger@mvtec.com

[1] MVTec Software GmbH, 80469 Munich, Germany

(a) Original Image **(b)** MSER **(c)** MSHR (our approach)

Fig. 1 Image **a** is from the ICDAR 2015 "Focused Scene Text" [23] challenge. The colored regions in **b** are extracted using MSER on the corresponding gray-scale image, those in **c** using MSHR. In contrast to MSER, MSHR can be segmented in images with an arbitrary number of channels and also work for characters that simultaneously have a lighter *and* a darker background. The regions touching the image border are not displayed

proposed approach for image segmentation, object tracking, and image filtering.

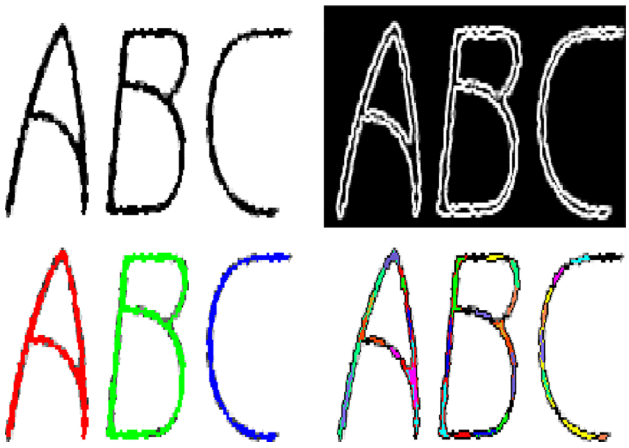

Fig. 2 The top row displays an example input image (left) and the respective gradient magnitude image (right). The second row displays the regions that are extracted from our method (left) and those by [15] (right), which uses the gradient magnitude image as input and applies an ordinary MSER algorithm

2 Related work

In general, component-trees have been used for a diverse set of applications and some efforts have been undertaken to enable their efficient computation [7]. There are essentially three different kinds of component-tree computation algorithms: immersion algorithms, flooding algorithms and merge-based algorithms. Carlinet and Géraud [7] present an extensive comparison of the main approaches and show that the flooding-based approaches of Wilkinson [47], Salembier et al. [41] and Nistér and Stewénius [35] are superior in terms of speed for 8-bit and 16-bit images. Although many applications for gray-scale component-trees have been presented, most are devoted to image segmentation and filtering [21, 27, 41]. For example, MSER can be extracted efficiently using component-trees [35]. MSER have a wide range of applications, ranging from stereo feature point extraction [28] over optical character recognition (OCR) [34] to image tracking [14].

Motivated by their success on gray-scale image processing applications, there have also been attempts to extend MSER specifically to multi-channel images. Chavez and Gustafson [10] transform the RGB image to the HSV color space and extract gray-scale MSER on the single channels separately. Forssén [17] overcomes the problem that multi-channel images cannot be totally ordered using pixel differences of neighboring RGB values as opposed to the RGB values directly. This allows the extraction of so-called maximally stable color regions (MSCR). Although no component-tree is constructed in the process, the idea of using differences is appealing, since it does not require a user-defined partial ordering and can further be trivially extended to images with an arbitrary number of channels. Unfortunately, the approach is computationally demanding and, although theoretically very closely related to MSER, MSCR have completely different parameters. This makes it difficult to compare the performance of both approaches. Similarly, Donoser et al. [15] construct the component-tree of an RGB image from the gradient magnitudes of the input image. Stable regions are then extracted by comparing the shape of regions at different levels using a shape matching method. While computing MSER from the gradient magnitude images allows the use of an ordinary MSER implementation, it has several disadvantages for applications. The central differences at the image coordinates used to compute the gradient magnitude image make the edges at least two pixels wide. As a consequence, the extracted regions are smaller than the actual regions and very slim regions cannot be extracted. As shown in Fig. 2, especially for applications like OCR, where characters are generally only a few pixels wide, this is a crucial disadvantage.

A more general approach to extending MSER to multi-channel images is to formulate them on a component-tree structure that is defined for multi-channel images. In general, the component-tree algorithms devoted to single-channel images assume that the values of the images can be equipped with a binary relation to form a totally ordered set. Unfortunately, multi-channel images are not canonically equipped with total orders, but with partial ones. Existing work tries to circumvent this restriction by either splitting the value space into several totally ordered sets, or by defining a total order relation [1]. A further variant is that of the *component-graph*, which extends the notation of component-trees to multi-channel images in a generic way [38]. Unfortunately, the multi-channel component-graph construction is computationally demanding and requires a suitable, user-defined,

piecewise ordering of the multi-channel image data that is specific to the target application [33]. A further extension of the component-tree to multi-channel images that is conceptually similar to ours is the *Multivariate tree of shapes* (MToS) [8, 49]. The MToS is a five-step process which first computes a tree of shapes (ToS) [9] for each channel individually. Hence, the runtime has a large linear factor in the number of image channels. In contrast to the above approaches, the proposed algorithm constructs the hierarchy using pixel differences and is applicable to images of different domains and numbers of channels and does not require any pre-defined partial ordering. As a consequence, fewer parameters are required and the tree construction is significantly faster. We compared our approach to the binaries of MToS and are 9 times faster for a three-channel image. The performance advantage should be even more prominent for hyper-spectral images, which contain significantly more channels.

As mentioned above, approaches based on constructing an edge-weighted graph [13] do not require a piece-wise ordering and scale favorably for images with many channels. The respective approaches create morphological hierarchies of connected pixels [43]. Hereby, any two pixels are connected if there exists a path linking these pixels, such that the maximal edge weight does not exceed a given threshold value. For a varying threshold, the resulting regions form a hierarchy. The concept originates from the single linkage clustering method [18] used for data analysis. It was introduced to image processing by Nagao et al. [33] in 1979. Prominent examples include α-components and α-trees [36, 43], and (in mathematical morphology) quasi-flat zones and the quasi-flat zone hierarchy [30, 31].

In contrast to the aforementioned approaches, we propose a flooding-based immersion for the generation of the quasi-flat zone hierarchy. The presented immersion is significantly faster than a union-find-based immersion and allows an efficient computation that is linear in the number of pixels and scales favorably in the number of channels. Furthermore, we extend the concept of MSER from component-trees to the hierarchy constructed from the edge-weighted graph.

3 Flooding-based quasi-flat zone hierarchy

For gray-scale images, the component-tree is constructed by considering the binary level sets of the input image. The level sets are obtained from successive thresholds, e.g., for byte images, the thresholds are selected to include all pixels within $[0, \alpha]$ or $[\alpha, 255]$. Hereby, α is either increased incrementally from 0 to 255 or decreased incrementally from 255 to 0. The evolution of the connected components of the respective level sets are then encoded in the component-tree. Hereby, each component in the tree represents an *extremal* region. These are identified by the fact that all pixels in the region have a gray value strictly larger or strictly smaller than the pixels in the outer boundary of the region. In the context of multi-channel images, the concept of *larger* or *smaller* is not well defined and the component-tree cannot be trivially constructed in the same fashion.

To overcome this limitation, the quasi-flat zone hierarchy considers the derivatives *between* neighboring pixels in x (column) and y (row) direction. The derivatives at the image coordinates between two pixels are approximated by central differences. Hence for each image pixel at the coordinate (x, y), four differences are computed: two vertical ones (δ_{upper} and δ_{lower}) and two horizontal ones (δ_{left} and δ_{right}). They are calculated as

$$\delta_{\text{left}}\mathscr{I}(x, y) = \mathscr{I}(x-1, y) - \mathscr{I}(x, y) \tag{1}$$

$$\delta_{\text{right}}\mathscr{I}(x, y) = \mathscr{I}(x, y) - \mathscr{I}(x+1, y) \tag{2}$$

$$\delta_{\text{upper}}\mathscr{I}(x, y) = \mathscr{I}(x, y-1) - \mathscr{I}(x, y) \tag{3}$$

$$\delta_{\text{lower}}\mathscr{I}(x, y) = \mathscr{I}(x, y) - \mathscr{I}(x, y+1), \tag{4}$$

where $\mathscr{I}(x, y)$ is the pixel value of the image $\mathscr{I}$ at coordinate (x, y). Independent on the number of channels of $\mathscr{I}$, the magnitude (absolute value for single channel and the Euclidean norm for multi-channel images) of the derivatives is totally ordered and can be used for the tree construction. Conceptually, instead of computing the level sets from successive thresholds of the pixel values, the level sets are obtained from thresholds of the derivative magnitudes. As a consequence, the components are characterized by the fact that each pixel within them has a derivative with a magnitude that is smaller than the derivatives at the boundary of the region. We denote such regions as *homogeneous* regions.

The concept of the resulting hierarchy is illustrated in the toy example in Fig. 3. Homogeneous regions (e.g., regions with the same color) are identified by the fact that the pixels are connected by derivatives with a very small magnitude. Hence, the child nodes consist of disjunct and differently colored regions. With a growing derivative magnitude threshold, similar colored regions merge into single components (e.g., red and pink, light-green and green) and, eventually, the whole image is connected.

3.1 Local flooding tree construction

The tree can be efficiently constructed by a flooding-based immersion. The concept is very similar to the flooding-based immersion of the ordinary component-tree [35], with the exception, that the derivatives are flooded instead of the image pixels and need to be mapped to the image pixels in

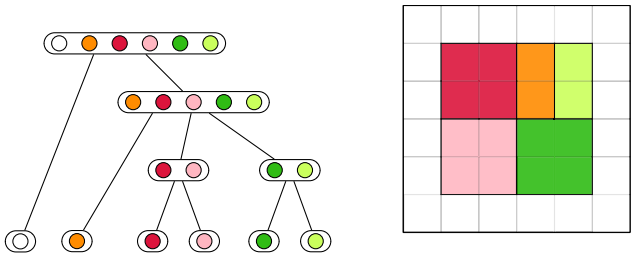

Fig. 3 (Best viewed in color) Toy example of the quasi-flat zone hierarchy for a three-channel image. Conceptually, the tree is constructed by iteratively thresholding the derivative magnitudes and connecting the resulting connected components. Hence, in an early stage, each of the uniquely colored regions is connected in a component (child node). In a next step, the most similar colors are connected in parent components. Since the orange region has a similar "distance" to the red and green regions it is connected to these components in a later step. Finally, the white region, having the largest distance to all the colors, is connected to the other components in the root node, which represents the complete image

the construction process. An overview of the flooding-based tree construction is visualized in Fig. 4.

In an initial step, starting from an arbitrary derivative, the flooding-based immersion searches for a derivative with a local minimal magnitude. Hereby, the local minimum does not need to be strict. It is sufficient to find a plateau (a derivative where the neighboring derivatives have the same or larger magnitude). In general, this step requires the notion of the neighborhood of a derivative. This is essentially determined by the two pixels it connects. Each of these pixels has 4 derivatives, two vertical ones and two horizontal ones. However, since they share the derivative that connects them, both pixels only have three unique derivatives. Furthermore, the neighbors of vertical and horizontal derivatives are different. They are both displayed in Fig. 5 for clarification. The alternating 6-connected structure is the edge graph of the Khalimsky grid [12]. To ensure that each true inner distance has 6 neighbors and that no explicit border treatment is required, the derivatives at the border of the image are artificially

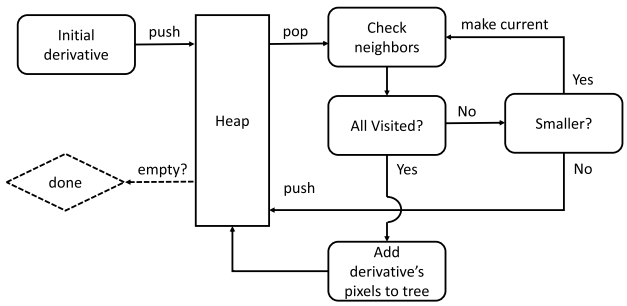

Fig. 4 Flow chart of the proposed flooding-based immersion. The single steps are explained in detail in Sect. 3.1

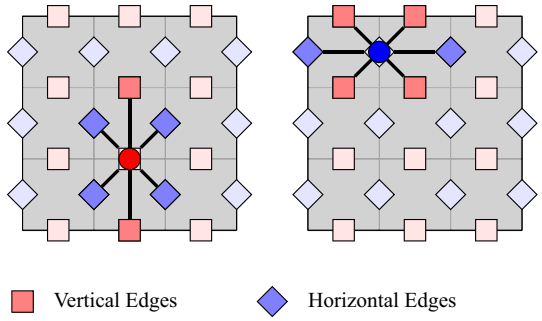

Fig. 5 The derivatives are between two pixels (gray boxes) and each have 6 neighbors. On the left, the 6 neighbors of a vertical (red) derivative are displayed and on the right, the 6 neighbors of a horizontal derivative (blue) are displayed

added with an infinite magnitude. Therefore, there are $w + 1$ horizontal derivatives within each row of the image and w vertical derivatives, where w is the width of original image.

Starting from an arbitrary derivative, its magnitude is compared to the magnitude of its six neighboring derivatives. As soon as a derivative with a lower magnitude is encountered, the process stops checking the other neighbors and *floods* into the respective derivative. This process is continued until a derivative that has a locally minimal magnitude (not strictly minimal) is encountered. Then, the two pixels belonging to the respective derivative are merged into a new component of the component-tree. During this process, all visited derivatives are stored in a heap. As a consequence, each derivative needs to be visited once during the tree construction. Hence, the flooding-based immersion is linear in the number of pixels.

In the next step, the derivative with the lowest magnitude in the heap is removed and compared to its neighbors. Either the process floods towards a new local minimum, or if all neighbors have been visited, merges the respective pixels to existing components. More precisely, every emerging derivative has four possibilities:

1. It connects two pixels that have not yet been visited ⇒ a new child node is generated.
2. It connects a pixel that has not been visited yet to an existing component ⇒ if the derivative magnitude is larger than those generating the respective component, a new parent node is generated. Otherwise, it is merely added to the component.
3. It connects two existing components ⇒ a new parent node connecting both components is generated.
4. It connects two pixels already within the same component ⇒ nothing needs to be done for this derivative. Continue with the next element in the heap

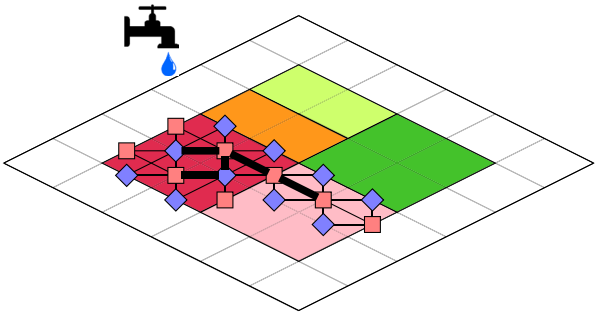

Fig. 6 (Best viewed in color) In the flooding-based immersion, starting from an arbitrary derivative, the immersion floods along the smallest neighboring derivative. When it finds a local minimum, the pixels belonging to the derivative are merged. In this example, the path first follows the zero derivatives within the red region, creating a red component within the component-tree. The next smallest derivative is at the border to the pink region; hence, the path floods into the pink area

It is important to note that the resulting hierarchy is independent of the choice of the starting point and of the order in which the neighboring derivatives are visited for the local flooding-based immersion. As soon as all derivatives have been visited (hence, the heap is empty), the process terminates. A toy example of the tree construction that displays the horizontal and vertical derivatives and the respective local flooding-based approach is displayed in Fig. 6.

Although the flooding immersion walks through all of the image derivatives, the pixels belonging to the derivatives are added to the component-tree. The derivatives can be efficiently mapped to the pixel values by their linearized image index δ^l. The mapping is different for vertical and horizontal derivatives and can be computed as:

$$\begin{aligned}\mathscr{P}_{\text{horz.}}(\delta^l) &= \{\delta^l - (w+1)(\lfloor \delta^l/(2w+1)\rfloor + 1),\\ &\qquad \delta^l - (w+1)(\lfloor \delta^l/(2w+1)\rfloor + 1) + 1\}\\ \mathscr{P}_{\text{vert.}}(\delta^l) &= \{\delta^l - (w+1)(\lfloor \delta^l/(2w+1)\rfloor) - w,\\ &\qquad \delta^l - (w+1)(\lfloor \delta^l/(2w+1)\rfloor)\},\end{aligned} \tag{5}$$

where w is the image width, $\mathscr{P}_{\text{horz.}}$ are the index of the two image pixel for a horizontal derivative, $\mathscr{P}_{\text{vert.}}$ the index of the two image pixel for a vertical derivative, and $\lfloor\cdot\rfloor$ represents the floor function, respectively.

3.2 Characteristics of the gradient-based component-tree

The granularity of the component-tree can be configured by discretizing the derivative magnitudes. We achieve this by quantizing the derivative magnitudes into a certain number of *bins*. However, since the distribution of the derivative magnitude is far from uniform in natural images, it is reasonable to not bin the derivatives equidistantly. This was also observed by Forssén [17] when extracting the so-called maximally stable color regions (MSCR). For the quasi-flat zone hierarchy, a very coarse binning leads to very compact trees; while, a more fine binning leads to more complex and descriptive trees. As shown in Fig. 7, although less descriptive, the coarse trees have the advantage that they can be computed significantly faster and that they reduce the computational complexity of the image processing techniques applied to them.

To ensure that the regions in the single components of the component-tree are pixel precise and include each pixel within a homogeneous region, it is essential to consider the derivatives at their true position *between* two pixels. Other approaches flood the gradient magnitude image directly for simplicity [15]. Although this enable to assume a 4-connected neighborhood of the derivatives and use an ordinary component-tree construction algorithm, the resulting components do not contain the true pixels of each homogeneous region. The differences between two pixels are added to both pixels. As a consequence, every edge is it least 2 pixels wide in the resulting images and only regions that are significantly large can be extracted. Furthermore, the resulting homogeneous regions are smaller than the actual homogeneous regions in the input image. For example, when flooding the gradient magnitude image of the toy example in Fig. 3 directly (as in [15]), it is not possible to extract the small regions. The gradient magnitude image has a high value at each pixel and has no valleys to flood. This is highlighted in Fig. 8.

3.3 Implementation details

The described quasi-flat zone hierarchy has the same structure as the gray-value component-tree. Therefore, the same algorithms may be applied. Nevertheless, the following modifications help to improve the algorithm's robustness:

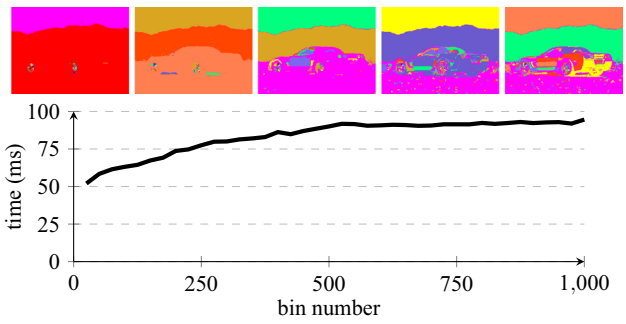

Fig. 7 The granularity of the component-tree can be configured by quantizing the derivative magnitudes into bins. The number of bins influences the runtime and the granularity of the possible segmentations. The measurements were obtained for 50 random images from PASCAL VOC 2007 [16]. Since the variation of the runtime was very small (≈ 0.3), error bars have not been added

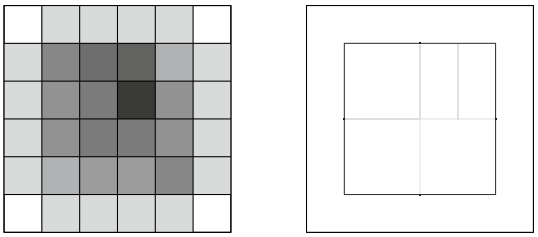

Fig. 8 The gradient magnitude image (left) and the derivatives *between* the image pixels (right) of the toy image in Fig. 3 are indicated. The gradient magnitude image has the problem that it spreads derivatives into all adjacent pixel values. It is not suited for extracting small homogeneous regions and generates eroded versions of the actual homogeneous regions in an image

1. Since the algorithm works on pixel differences, it is susceptible to image noise. This was also observed by Forssén [17] who proposed to perform Gaussian smoothing as a preprocessing step. Unfortunately, this may add artificial components at strict vertical or horizontal image derivatives. We found that edge-preserving smoothing, such as bilateral or guided image filtering [20] helps to remove these artifacts. An example is shown in Fig. 9.
2. Furthermore, since very small image regions are rarely of interest, we found it very useful to restrict the minimal area a component must have to create a node within the tree. This leads to more compact trees and can significantly reduce the runtime, while it has virtually no impact on later queries of the component-tree. As opposed to [45], we do not delete the regions from the tree explicitly and then apply a region growing algorithm to extend the remaining components. Instead, the flooding-based immersion allows to merge these regions into their parent component during the tree construction implicitly. For this, each component is only added to the tree once it is big enough. Until then, the connecting pixels are added to the component without creating parent components. This allows to filter the regions without adding any computational overhead. Each derivative still only needs to be visited once by the algorithm.

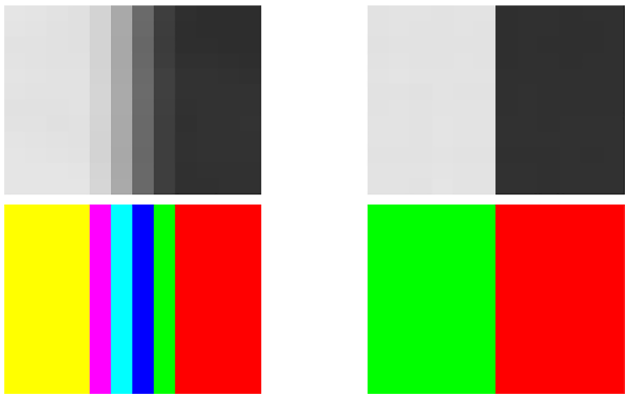

Fig. 9 In the top row, Gaussian smoothing (left) and edge-preserving bilateral filtering (right) are applied to a vertical image edge. The resulting homogeneous regions are shown in the bottom row. Gaussian smoothing adds artifacts that create artificial homogeneous regions

In our experiments, we also use both concepts for the gray-scale component-tree, since they are equally applicable.

In general, our approach has a larger computational overhead than that of the gray-scale component-tree. First of all, there are around twice as many image derivatives as there are pixels. Furthermore, each derivative has to consider 6 derivative neighbors compared to 4 pixel neighbors. Hence, the construction process is expected to be approximately 3 times slower than that of the gray-scale component-tree for a single polarity (plus the overhead of calculating the image derivatives). However, since the quasi-flat zone hierarchy implicitly captures all regions which are lighter *or* darker than their background (for gray-scale images), the runtime is essentially only 1.5 times slower when extracting regions of both polarities. This factor is confirmed empirically in Fig. 10.

Furthermore, the union-find data structure (with path compression) that is also shown in Fig. 10 supports quasi-linear time in the number of pixels [17]. More specifically, the time is bounded by $\mathcal{O}(n\alpha(n))$, where n is the number of pixels and $\alpha(n)$ is the inverse of the Ackermann function, whose value is smaller than 5 if n is of the order 10^{80}. The flooding-based approach has a complexity of $\mathcal{O}(n\beta)$, where β is independent of n, since each pixel is only considered

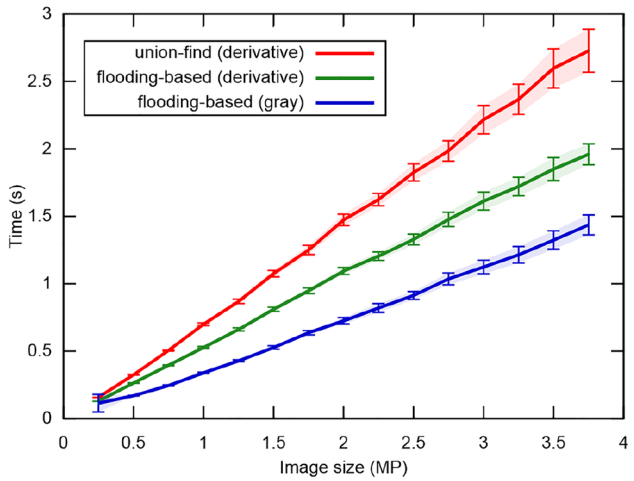

Fig. 10 The runtime and variance of constructing the tree with the proposed flooding-based immersion and the standard union-find-based algorithm (Kruskal). The average runtimes are computed using 30 randomly selected pictures from the Pascal VOC 2007 dataset [16]. The computation time of the ordinary gray-scale component-tree is added as orientation

once. Furthermore, the representation is less memory intensive, since the data structure does not require each pixel to be represented by an element of a data structure before it is visited by the flooding process.

Please note, the complexity of the tree traversal is the same for both component-trees. All of the routines presented in this paper are implemented in C and the code is optimized and parallelized where possible. In general, the tree construction is very efficient and requires less than 250ms for a 800×1000 image on an IntelCore i7-4810 CPU @2.8GHz with 16GB of RAM with Windows 7 (x64).

3.4 Multiple disconnected component-trees

To speed up the construction process even further, it is possible to restrict the maximal derivative magnitude to be considered during the tree construction. A large derivative magnitude indicates that the difference between the two pixels is very large. However, many image processing applications focus on finding regions that are very homogeneous and are not concerned with representing the complete image in the component-tree. Furthermore, since pixels may be connected over different paths of derivatives, the complete image is often represented within the component-tree without needing to consider all derivatives. However, restricting the maximal derivative magnitude may yield multiple, separated, trees. In this case, the image processing algorithms described later are applied to the single trees independently. Again, the concept is also applicable to gray-value component-trees. In this case, the minimal and/or maximal gray-value threshold to be considered is set. This may reduce the number of pixels that need to be considered considerably. Please note, the parametrization of both concepts is highly application specific.

4 Maximally stable homogeneous regions

The constructed hierarchy can essentially be used for the same image processing tasks as the gray-scale component-tree. For example, the tree can be used to efficiently extract stable regions similar to MSER. The only difference is that the tree nodes do not consist of extremal regions but of homogeneous regions. As mentioned above, homogeneous regions are characterized by the fact that each pixel within the region has a vertical or horizontal derivative with a smaller magnitude than all outer derivatives of the region. Otherwise, the concept of stable regions is the same. Hence, both approaches share the same parameters and scale equally with growing image sizes. Let $R_1, \ldots, R_{i-1}, R_i, R_{i+1} \ldots$ be a set of nested homogeneous or extremal regions, respectively (hence $R_i \subset R_{i+1}$). In the context of component-trees, the index i encodes the gray-value threshold or the derivative magnitude threshold that generated the region. A maximally stable region R_{i*} in the context of MSER and MSHR is a region that has a local minimum of

$$s(i) = \frac{|R_{i+\Delta} \setminus R_{i-\Delta}|}{|R_i|}, \tag{6}$$

at $i*$. Here, $|\cdot|$ denotes the cardinality and Δ is a parameter of the method. The parameter Δ encodes how stable a region is over $\pm\Delta$ thresholds. The larger the value, the more stable the regions need to be.

In a component-tree, the sequence of ancestor and descendant nodes for a node is a set of nested regions. To simplify the computation, each node in the tree stores its area and the smallest derivative magnitude that connects its inner points (these can be adapted on the fly during the tree construction). Hence, $s(i)$ can be computed for each node by checking the area of the ancestor and descendant nodes at a distance of Δ, respectively. The resulting maximally stable homogeneous regions (MSHR) are possibly overlapping regions that do not change their area significantly over a given derivative magnitude range.

An example of MSHR for an image from PASCAL VOC 2007 [16] is shown in Fig. 11. For all MSHR examples, we use a guided image filter with a radius of 3 and

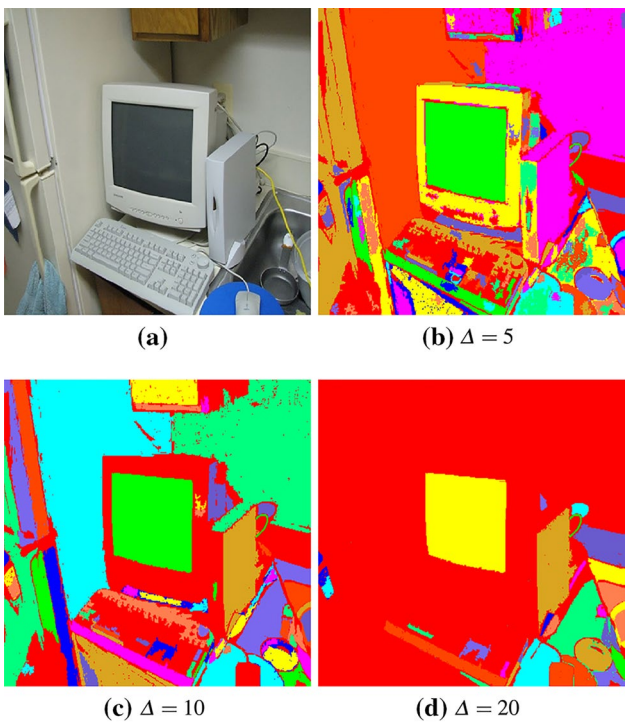

Fig. 11 The quasi-flat zone hierarchy can be used to extract stable regions from color images. The images **b**–**d** display the extracted MSHR from image **a** for different settings of Δ. Larger values of Δ lead to a coarser segmentation

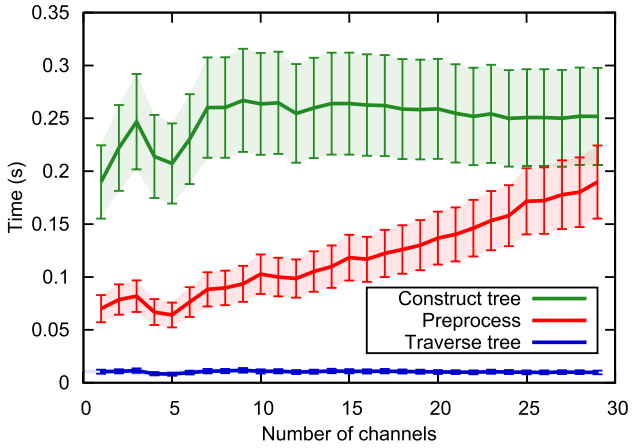

Fig. 12 The runtime of constructing and traversing the tree is independent of the number of channels. Only the preprocessing step scales linearly if additional channels are added to an image. The average runtimes and variances displayed use channels with 801 x 1000 pixels

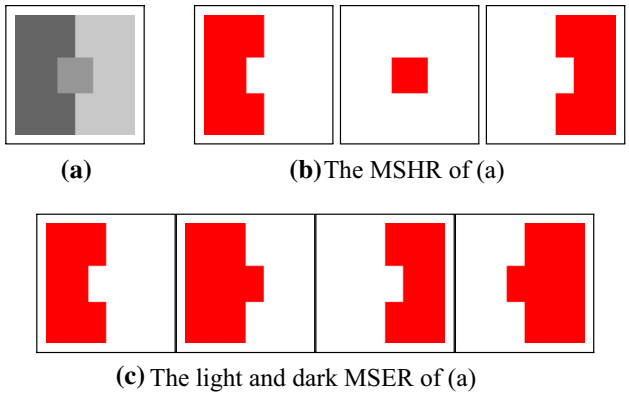

Fig. 13 The center region of **a** is no extremal region. Hence, regardless of the parameter settings, it will never be an MSER (**c**). For MSHR, on the other hand, the inner derivatives of the center region are smaller than its outer class derivatives, and hence it is a homogeneous region (**b**). The whole image is an MSER as well as an MSHR but is omitted in **b** and **c** for the sake of clarity

an amplitude threshold of 20 (denoted as ϵ in [20]) for preprocessing. The parameter Δ in (6) determines the stability of the segmented regions. An advantage of using the derivative magnitude-based component-tree for the MSHR extraction process is that various different parameter settings of Δ can be used to extract a large collection of different regions without significantly increasing the runtime. As shown in Fig. 12, the runtime of the tree traversal is negligible to the time required for the tree construction. Hence, regions for multiple values of Δ can be extracted from an image at little cost. The displayed preprocessing time includes the guided filtering step and the derivative calculation. For the evaluation, we used hyper-spectral image data obtained from the Stanford Center for Image Systems Engineering (SCIEN) [42]. We randomly chose a subset of channels and evaluated the runtime for 50 runs.

4.1 MSHR vs MSER

The building blocks of MSER are extremal regions that either have gray-scales strictly larger or strictly smaller than their outer border. This means that some regions of interest can never be segmented with the help of MSER. In contrast, the building blocks of MSHR are homogeneous regions, which require that each inner pixel has a smaller derivative than all outer derivatives of the region. Hence, as shown in Fig. 13, our approach is able to extract regions that simultaneously have a lighter *and* darker background. In the applications section, we show how this attribute can be very helpful in applications such as OCR, where MSER-based approaches fail.

4.2 MSHR for stereo feature-point matching

As their gray-value counterpart, MSHR can also be used for stereo feature matching. Forssén [17] showed that color-based MSER are superior to the original MSER [28] in terms of repeatability in wide-baseline stereo feature-point matching on a set of 8 image sets [37]. We conducted extensive experiments and were able to repeat the results. Nevertheless, we observed that the repeatability of the stereo features of MSER, MSCR and MSHR depend more on the parameter settings (mostly on Δ) than on the chosen method. Hence, we omit our experiments on stereo feature point mapping.

5 Applications

5.1 Optical character recognition

One of most common applications of the gray-scale component-tree is the extraction of maximally stable extremal regions (MSER) [29]. Although initially proposed as stereo features [28], they are used extensively as a preprocessing step for optical character recognition (OCR) systems [24, 34]. Please note, although the performance of MSER-based OCR systems can be outperformed by techniques building on convolutional and recurrent neural networks [6, 40], they are still used in many running OCR systems. This is due to the fact that they have a much lower computational complexity and can be computed in real time on embedded devices and machines without a GPU.

Since MSER assume the regions to be extremal, they cannot extract characters that have a lighter *and* darker background (see Figs. 1 and 13). This can be a problem in OCR

systems, since most approaches fail if the characters cannot be segmented in an early stage. We evaluate the text segmentation capabilities of MSER [29], MSCR [17] and MSHR regions on the ICDAR 2015 "Focused Scene Text challenge" [23] dataset. As in common in the ICDAR challenges [22], we consider a character to be found if it overlaps the ground truth bounding box according to the PASCAL overlap criterion [16] by more than 50%. As shown in Table 1, MSHR clearly outperforms MSCR. However, when applied alone, both methods are weaker than MSER.

The different approaches of either flooding the derivatives or the image pixels essential create regions with complementary attributes. This can be leveraged to improve the segmentation results. For this, the union of the proposed character regions of MSHR and MSER (MSER + MSHR), and of MSHR and MSER (MSER + MSHR) is computed. As expected, the combination of MSHR and MSER is able to significantly improve the segmentation obtained by only MSER. Please note, the initial recall of the segmentation is an important indicator of how well an OCR system can perform. Later steps are usually concerned with grouping and filtering out undesired regions, hence what is not found in an initial step will not be found. A handful of example images where MSHR are superior to MSER is presented in Figs. 1 and 14.

The runtimes of the approaches are displayed in Table 2. As expected, MSER slightly outperforms MSHR. However, since both approaches construct a component-tree, characters for different settings of Δ can be extracted extremely efficiently. As a consequence, both approaches are significantly faster than MSCR.

5.2 Tracking

The efficient tracking of regions in component-trees was first proposed for gray-scale trees in [14] and then extended to multi-channel images [3]. The tracking algorithm consists of three steps: model initialization, model tracking, and the model update.

Table 1 The segmentation results of MSER [29], MSCR [17] and MSER augmented with MSHR on the ICDAR 2015 "Focused Scene Text challenge" [23] dataset are displayed

Method	$\Delta = 1$	$\Delta = 5$	$\Delta = 10$
MSER [29]	89.69	85.44	79.88
MSCR [17]	80.75	71.41	57.28
MSHR	88.73	83.69	76.21
MSER + MSCR	90.84	87.68	80.76
MSER + MSHR	**93.64**	**89.12**	**84.46**

The proposed approach clearly outperforms MSCR and is able to improve the segmentation obtained by only MSER

The best performing method is highlighted in bold

Table 2 The average computation time of MSER [29], MSCR [17] and MSER augmented with MSHR on the ICDAR 2015 "Focused Scene Text challenge" [23] dataset are displayed

Method	∅ runtime in ms.
MSER [29]	93.34
MSCR [17]	841.71
MSHR	133.13
MSER + MSCR	951.0
MSER + MSHR	227.28

The runtime is averaged over all images (which have varying size and complexity)

Model initialization In the first step, we extract MSHR from the given target location. MSHR divide the image into multiple, possibly overlapping, connected components. The tracking can either be restricted to the largest, the most stable or all MSHR regions within the template.

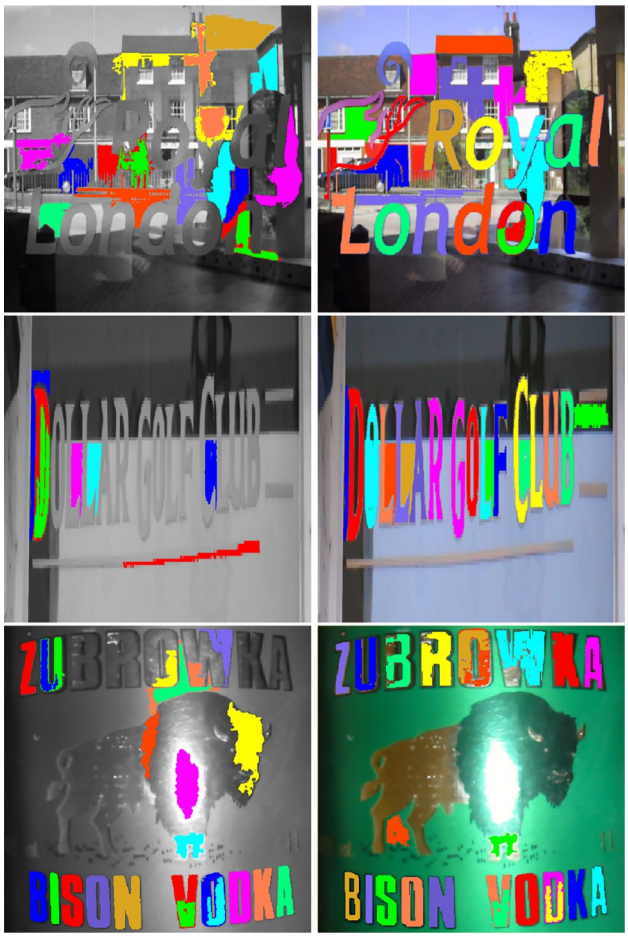

Fig. 14 The MSER segmentation in the first row has difficulties with characters that are simultaneously lighter *and* darker than their background. The MSHR segmentation is able to extract all relevant character regions and displayed in the second row

Model tracking In the successive frames, the component-tree is constructed and the MSHR regions are matched against all regions in the respective tree. The tree is constructed for a rectangular search domain that is twice the size of the objects bounding box in the prior frame.

To enable efficient matching, we compute a handful of region and gray-scale features for each MSHR. We solely use features that can efficiently be calculated by region and gray-scale moments. The advantage of using moments, is that they can be computed incrementally during the tree construction. The moment of order (p, q) of a region $\mathscr{R}$ is defined as

$$m_{p,q} = \sum_{(r,c)\in\mathscr{R}} r^p c^q, \tag{7}$$

where r and c represent the row and the column coordinates, respectively, and $p, q \geq 0$. Since the flooding-based immersion considers each image pixel in the tree construction anyway, our choice of features can be calculated without adding significant computational complexity. We use the area of the region ($m_{0,0}$), the center of gravity ($m_{1,1}/m_{0,0}$) and the ellipse parameters r_1, r_2 and θ as tracking features. The ellipse parameters can be calculated with the normalized moments, please see [19] for details. Analogously, for each channel, we use gray-scale moments to calculate the average gray-scale and the gray-scale deviation as further features. Please note, our selection of features makes the approach invariant to rotations of the MSHR.

To further improve the robustness, the single features in the matching step can be weighted for specific applications. For example, if the object undergoes heavy deformations, but has a relatively constant color, the weight of the region moments is reduced and the gray-scale features weights are increased. The weights are estimated automatically from the variation of the color and the variation of the region moments within the first five frames.

In the tracking step, we do not extract the most stable MSHR and match their features to those from the initial frame. Instead, we compare the features of the initial MSHR to the features of all the homogeneous regions in the component-tree nodes. This helps to improve the robustness and ensures we do not restrict the search to only *maximally* stable homogeneous regions.

Model update As opposed to [14], we update the region features incrementally in each frame. This enables to handle short occlusions and detection failures in single frames. Furthermore, it allows the object to change its appearance throughout the sequence and allows more robust tracking. Hence, after successfully locating the node that best fits the to-be-tracked MSHR at time step t, the feature vector (denoted as $\mathscr{F}$) is updated as

$$\mathscr{F}_{t+1} = (1-\lambda)\mathscr{F}_t + \lambda\mathscr{F}_{t+1}. \tag{8}$$

In our experiments, we used $\lambda = 0.5$.

Results The proposed MSHR tracking approach is not restricted to bounding boxes. Hence, to evaluate the quality of the tracking results, we manually annotated dense by-pixel segmentations of scenes from the OTB [48] and VOT2016 [25] datasets. Otherwise, the given bounding box ground truth would introduce an undesired bias when measuring the overlap scores of by-pixel segmentations. As accuracy measure, we use the commonly used Intersection over Union (IoU) criterion.

To bring the results into perspective, we compute the best possible overlap an axis-aligned tracker could obtain for the segmentation of a given scene. By this means, the performance gain of using segmentations can be highlighted without introducing a bias by choosing a specific set of state-of-the-art axis-aligned trackers to compete against. We refer to this tracker as the `Best box`. Please note, the `Best box` is an upper performance bound for all box-based trackers. Hence, it outperforms the current state-of-the-art of deep-learning-based trackers, since these are all restricted to axis-aligned boxes. Please see [4, 5] for details on how the `Best box` can be efficiently computed.

For the gray-scale sequence `dress` from OTB [48], the MSER tracker, `Best box` and the MSHR tracker all perform similarly, as is displayed in Fig. 15. In the respective sequence, the MSER tracker is able to track the head and the dress of the dancer, while the MSHR tracker only tracks the dress. Hence, the overlap scores of MSER are

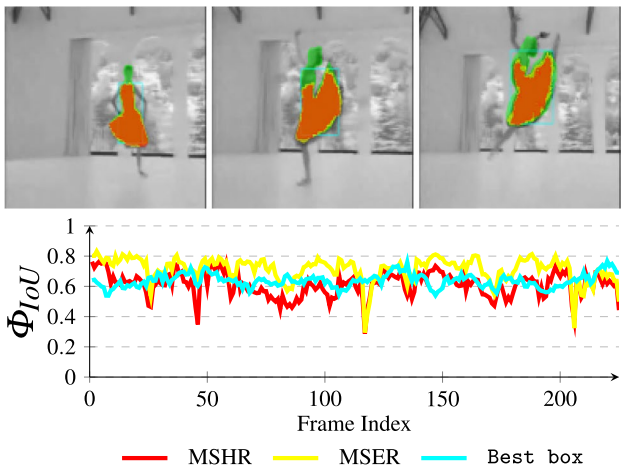

Fig. 15 `dress` from OTB [48]. For this gray-scale scene, the MSER tracker (average $\phi_{\text{IoU}} = 0.71$) is able to outperform the MSHR tracker (average $\phi_{\text{IoU}} = 0.69$). The ground truth segmentation of the torso is visualized for reference (green). The MSER tracker also outperforms the best possible overlap an axis-aligned tracker (`Best box`, average $\phi_{\text{IoU}} = 0.64$) can achieve for the segmentations within the scene

slightly superior. Nevertheless, it is important to note that both approaches are compared against the *best possible* axis-aligned tracker and, accordingly, the overlap scores are impressive.

For color images, the difference of MSER and MSHR becomes more evident. In the `book` sequence from VOT2016 [25], the MSER tracker fails, as is shown in Fig. 16. The book is, per definition, not an extremal region in the gray-scale image. Hence, the initialization is not successful and the MSER tracker fails. Nevertheless, the book is a homogeneous region in both the gray-scale and the color image, and accordingly, the MSHR tracker is successful. In most frames, the MSHR tracker is even able to outperform the `Best box` and obtains an average IoU of 0.7.

For the `book` sequence, the MSHR tracking requires at most of 24ms per frame and for the `dress` sequence at most 19ms per frame on an Intel Core i7-4810 CPU @2.8GHz with 16GB of RAM with Windows 7 (x64).

5.3 3D segmentation

In the third application, we reconstruct organs in 3D by tracking a slice of an Computed Tomography (CT) scan along the axis orthogonal to the image data. We use the CT data provided in the 3DIRCADb dataset [44].

To initialize the tracking process, the organ is marked in an arbitrary slice of the CT data by a bounding box. The most stable MSHR is then automatically segmented in the initialization process for tracking. An example of the tracked regions is visualized for two examples in Fig. 17. Given the segmentations of the single slices, the organ can be reconstructed in 3D. We compare the reconstruction for MSER and MSHR tracking in Fig. 18. To enhance the visualization, the datapoints are triangulated and the surface normals are calculated. Since the contrast of the organs can be very low in CT images, the MSER tracking has difficulties catching the organ boundaries. Furthermore, the organ is sometimes partly lighter *and* darker than the background, which may lead to MSER tracking failure. The proposed MSHR tracking copes well with these difficulties, and the reconstructions are significantly better.

Please note, the tracking of the regions in the CT slices is extremely efficient and only requires an average of 5ms per slice. Hence, for the 45 slices in Fig. 18 the complete 3D reconstruction process, which includes the triangulation (≈ 1 s), the calculation of the surface normals (≈ 130 ms), and the segmentation (≈ 220 ms), requires only around 1.5 s.

The average $\boldsymbol{\Phi}_{\text{IoU}}$ for MSER and MSHR is displayed for a selection of organs in the 3DIRCADb dataset [44] in Table 3. As expected, the proposed MSHR significantly outperforms MSER on CT images. MSER struggles with the fact that the organs are sometimes partly lighter and

Table 3 The average $\boldsymbol{\Phi}_{\text{IoU}}$ scores of MSER [29] our proposed MSHR on the 3DIRCADb dataset [44] are displayed

Method	∅ $\boldsymbol{\Phi}_{\text{IoU}}$	∅ Runtime per frame in ms.
MSER [29]	0.51	3.62
MSHR	**0.87**	4.89

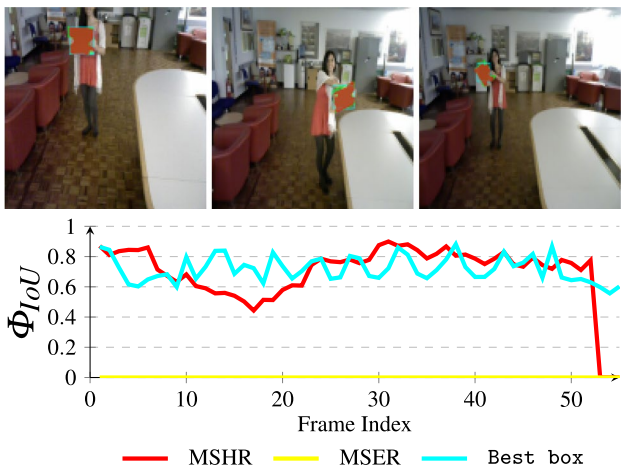

Fig. 16 `book` from VOT2016 [25]. Since the gray-scale region is not an MSER, it cannot be tracked with MSER tracking (average $\phi_{\text{IoU}} = 0.0$) . The overlap scores of the MSHR tracking (average $\phi_{\text{IoU}} = 0.71$) are comparable and sometimes even better than the overlap the best possible axis-aligned tracker (`Best box`, average $\phi_{\text{IoU}} = 0.72$) could theoretically achieve

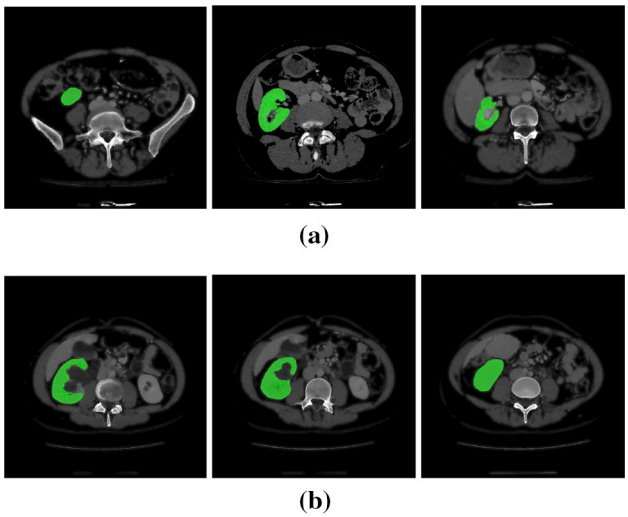

Fig. 17 Two example sequences from the 3DIRCADb dataset [44]. Given an initial selection of a single slice (the middle image in **a** and **b**) of the right kidney, the proposed MSHR tracking tracks the region forward and backwards in space. The segmented slices can be used to reconstruct the organ, see Fig. 18 for an example reconstruction

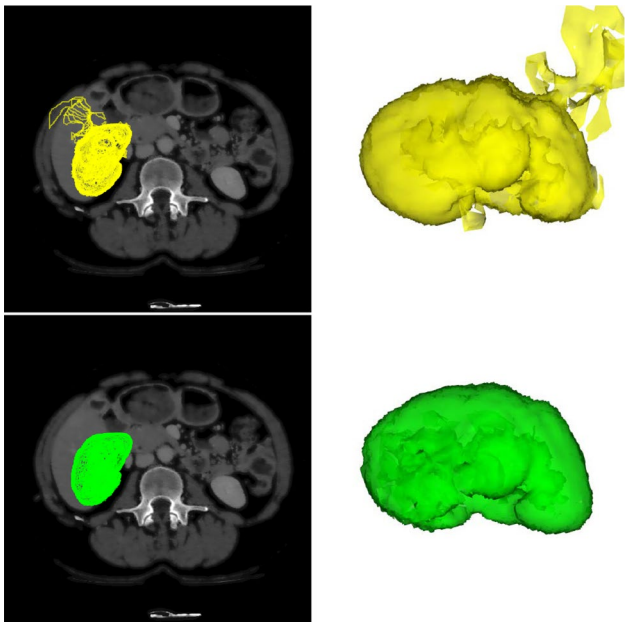

Fig. 18 In the first row, the reconstruction of the right kidney is displayed for MSER tracking. The low contrast and the fact that the background is partly darker and lighter than the objects makes the reconstruction noisy. The proposed MSHR tracking can cope with these situations and the reconstruction is significantly better

darker than the background. Both methods are extremely fast, since the component-tree only needs to be constructed for a small part of the image.

5.4 Image filtering

A popular application for gray-value component-trees is image filtering or simplification [21, 41]. Since the component-tree inherently incorporates the connectedness of the different image components, smaller regions can be easily removed or single components filtered without being influenced by other components. As opposed to approaches that build on component graphs [27, 32], the gray-value based concepts can be applied to the quasi-flat zones hierarchy without adaptation. For example, by restricting the maximum derivative magnitude, we can construct a collection of component-trees from the input image. Each component within the trees consists of pixels that are connected by an derivative smaller than the defined maximum size. We can filter these components without influencing the filter results by the large derivatives. An example which sets each component smaller than a certain area to its mean pixel values is shown in Fig. 19. The schemes within [21] or [41] can be applied equivalently.

Fig. 19 The quasi-flat zones hierarchy can be used to filter images. The images (top row) are filtered by setting the pixel value to the mean pixel value of the respective component (bottom row)

6 Conclusion

In this paper we proposed an efficient algorithm to construct the quasi-flat zone hierarchy of an input image. As opposed to the ordinary component-tree, the presented approach can be applied to images with an arbitrary number of channels. The presented algorithm is extremely efficient and opens the door for a number of gray-scale image processing techniques to multi-channel images. To highlight the universality of the proposed algorithm, we present a number of example applications: object tracking, 3D segmentation, and image filtering.

Furthermore, we introduce the concept of maximally stable homogeneous regions for image segmentation. The extension of an existing gray-scale component-tree algorithm is straightforward and runs efficiently in linear time

(in the number of pixels). We show how maximally stable homogeneous regions can be used to improve OCR results on the ICDAR 2015 "Focused Scene Text challenge" and display a number of examples.

In future work, we intend to investigate tightening the connectivity of homogeneous regions to require multiple vertical or horizontal derivatives with a smaller magnitude than the current threshold. Although this adds algorithmic complexity, it should help to tackle segmentation tasks without the need of prior image pre-processing. Secondly, we intend to proceed towards an automatic estimation of the Δ parameter in the MSHR computation. This should allow to improve the usability of the approach for generic OCR applications.

References

1. Angulo, J.: Geometric algebra colour image representations and derived total orderings for morphological operators-part i: colour quaternions. J. Vis. Commun. Image Represent. **21**(1), 33–48 (2010)
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: IEEE International Conference on Image Processing (ICIP), pp. 40–44 (2007)
3. Böttger, T., Eisenhofer, C.: Efficiently tracking homogeneous regions in multichannel images. In: International Conference on Pattern Recognition Systems (ICPRS), pp. 14–22 (2017)
4. Böttger, T., Follmann, P.: The benefits of evaluating tracker performance using pixel-wise segmentations. In: IEEE International Conference on Computer Vision (ICCV), pp. 1983–1991 (2017)
5. Böttger, T., Follmann, P., Fauser, M.: Measuring the accuracy of object detectors and trackers. In: German Conference on Pattern Recognition (GCPR), pp. 415–426. Springer (2017)
6. Bušta, M., Neumann, L., Matas, J.: Deep textspotter: an end-to-end trainable scene text localization and recognition framework. In: IEEE International Conference on Computer Vision (ICCV), pp. 22–29 (2017)
7. Carlinet, E., Géraud, T.: A comparison of many max-tree computation algorithms. In: Mathematical Morphology and Its Applications to Signal and Image Processing (ISMM), pp. 73–85 (2013)
8. Carlinet, E., Géraud, T.: Mtos: a tree of shapes for multivariate images. IEEE Trans. Image Process. **24**(12), 5330–5342 (2015)
9. Carlinet, E., Crozet, S., Géraud, T.: Mtos: the tree of shapes turned into a max-tree: a simple and efficient linear algorithm. IEEE International Conference on Image Processing, pp. 1488–1492 (2018)
10. Chavez, A., Gustafson, D.: Color-based extensions to MSERs. In: Advances in Visual Computing—7th International Symposium (ISVC), pp. 358–366 (2011)
11. Chen, L., Berry, M.W., Hargrove, W.W.: Using dendronal signatures for feature extraction and retrieval. Int. J. Imaging Syst. Technol. **11**(4), 243–253 (2000)
12. Cousty, J., Bertrand, G., Couprie, M., Najman, L.: Fusion graphs: merging properties and watersheds. J. Math. Imaging Vis. **30**(1), 87–104 (2008)
13. Cousty, J., Najman, L., Perret, B.: Constructive links between some morphological hierarchies on edge-weighted graphs. In: Mathematical Morphology and Its Applications to Signal and Image Processing, pp. 86–97 (2013). https://doi.org/10.1007/978-3-642-38294-9_8
14. Donoser, M., Bischof, H.: Efficient maximally stable extremal region (MSER) tracking. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 553–560 (2006)
15. Donoser, M., Riemenschneider, H., Bischof, H.: Linked edges as stable region boundaries. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1665–1672 (2010)
16. Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results (2007). http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html
17. Forssén, P.: Maximally stable colour regions for recognition and matching. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2007)
18. Gower, J.C., Ross, G.J.: Minimum spanning trees and single linkage cluster analysis. Appl. Stat. **18**(1), 54–64 (1969)
19. Haralick, R.M., Shapiro, L.G.: Computer and Robot Vision, vol. 1. Addison-Wesley Longman Publishing Co., Inc, Boston (1991)
20. He, K., Sun, J., Tang, X.: Guided image filtering. In: European Conference on Computer Vision (ECCV), pp. 1–14 (2010)
21. Jones, R.: Connected filtering and segmentation using component trees. Comput. Vis. Image Underst. **75**(3), 215–228 (1999)
22. Karatzas, D., Gomez-Bigorda, L., Nicolaou, A., Ghosh, S., Bagdanov, A., Iwamura, M., Matas, J., Neumann, L., Chandrasekhar, V.R., Lu, S., et al.: Icdar 2015 competition on robust reading. In: International Conference on Document Analysis and Recognition (ICDAR), pp. 1156–1160. IEEE (2015)
23. Karatzas, D., Gomez-Bigorda, L., Nicolaou, A., Ghosh, S.K., Bagdanov, A.D., Iwamura, M., Matas, J., Neumann, L., Chandrasekhar, V.R., Lu, S., Shafait, F., Uchida, S., Valveny, E.: ICDAR 2015 competition on robust reading. In: 13th International Conference on Document Analysis and Recognition (ICDAR), pp. 1156–1160 (2015)
24. Koo, H.I., Kim, D.H.: Scene text detection via connected component clustering and nontext filtering. IEEE Trans. Image Process. **22**(6), 2296–2305 (2013)
25. Kristan, M., Leonardis, A., Matas, J., Felsberg, M., Pflugfelder, R.P., Cehovin, L., Vojír, T., Häger, G.: The visual object tracking VOT2016 challenge results. In: European Conference on Computer Vision Workshops (ECCVW), pp. 777–823 (2016). https://doi.org/10.1007/978-3-319-48881-3_54
26. Kurtz, C., Naegel, B., Passat, N.: Connected filtering based on multivalued component-trees. IEEE Trans. Image Process. **23**(12), 5152–5164 (2014)
27. Kurtz, C., Naegel, B., Passat, N.: Multivalued component-tree filtering. In: International Conference on Pattern Recognition (ICPR), pp. 1008–1013 (2014)
28. Matas, J., Chum, O., Urban, M., Pajdla, T.: Robust wide baseline stereo from maximally stable extremal regions. In: Proceedings of the British Machine Vision Conference (BMVC), pp. 1–10 (2002)
29. Mattes, J., Demongeot, J.: Efficient algorithms to implement the confinement tree. In: Discrete Geometry for Computer Imagery (DGCI), pp. 392–405 (2000)
30. Meyer, F., Maragos, P.: Morphological scale-space representation with levelings. In: Scale-Space Theories in Computer Vision, Second International Conference, Scale-Space'99, Corfu, Greece, September 26-27, 1999, Proceedings, pp. 187–198 (1999). https://doi.org/10.1007/3-540-48236-9_17
31. Meyer, F., Maragos, P.: Nonlinear scale-space representation with morphological levelings. J. Vis. Commun. Image Represent. **11**(2), 245–265 (2000). https://doi.org/10.1006/jvci.1999.0447
32. Naegel, B., Passat, N.: Towards connected filtering based on component-graphs. In: Mathematical Morphology and Its Applications to Signal and Image Processing (ISMM), pp. 353–364 (2013)

33. Nagao, M., Matsuyama, T., Ikeda, Y.: Region extraction and shape analysis in aerial photographs. Comput. Graph. Image Process. **10**(3), 195–223 (1979)
34. Neumann, L., Matas, J.: Real-time scene text localization and recognition. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3538–3545 (2012)
35. Nistér, D., Stewénius, H.: Linear time maximally stable extremal regions. In: European Conference on Computer Vision (ECCV), pp. 183–196 (2008)
36. Ouzounis, G.K., Soille, P.: The alpha-tree algorithm. JRC Scientific and Policy Report (2012)
37. http://www.robots.ox.ac.uk/~vgg/research/affine/
38. Passat, N., Naegel, B.: Component-trees and multivalued images: structural properties. J. Math. Imaging Vis. **49**(1), 37–50 (2014)
39. Passat, N., Naegel, B., Benoît, K.: Component-graph construction. J. Math. Imaging Vis. **61**, 1–26 (2019)
40. Qin, S., Manduchi, R.: A fast and robust text spotter. In: IEEE Winter Conference on Applications of Computer Vision (WACV), pp. 1–8 (2016)
41. Salembier, P., Oliveras-Vergés, A., Garrido, L.: Antiextensive connected operators for image and sequence processing. IEEE Trans. Image Process. **7**(4), 555–570 (1998)
42. Skauli, T., Farrell, J.: A collection of hyperspectral images for imaging systems research. In: Digital Photography IX, vol. 8660, p. 86600C. International Society for Optics and Photonics (2013)
43. Soille, P.: Constrained connectivity for hierarchical image decomposition and simplification. IEEE Trans. Pattern Anal. Mach. Intell. **30**(7), 1132–1145 (2008). https://doi.org/10.1109/TPAMI.2007.70817
44. Soler, L., Hostettler, A., Agnus, V., Charnoz, A., Fasquel, J., Moreau, J., Osswald, A., Bouhadjar, M., Marescaux, J.: 3d image reconstruction for comparison of algorithm database: a patient-specific anatomical and medical image database (2012). http://ircad.fr/research/3D-ircadb-01
45. Weber, J., Lefèvre, S.: Fast quasi-flat zones filtering using area threshold and region merging. J. Vis. Commun. Image Represent. **24**(3), 397–409 (2013). https://doi.org/10.1016/j.jvcir.2013.01.011
46. Westenberg, M.A., Roerdink, J.B.T.M., Wilkinson, M.H.F.: Volumetric attribute filtering and interactive visualization using the max-tree representation. IEEE Trans. Image Process. **16**(12), 2943–2952 (2007)
47. Wilkinson, M.H.F.: A fast component-tree algorithm for high dynamic-range images and second generation connectivity. In: IEEE International Conference on Image Processing (ICIP), pp. 1021–1024 (2011)
48. Wu, Y., Lim, J., Yang, M.: Object tracking benchmark. IEEE Trans. Pattern Anal. Mach. Intell. **37**(9), 1834–1848 (2015). https://doi.org/10.1109/TPAMI.2014.2388226
49. Xu, Y., Carlinet, E., Géraud, T., Najman, L.: Hierarchical segmentation using tree-based shape spaces. IEEE Trans. Pattern Anal. Mach. Intell. **39**(3), 457–469 (2017)

Tobias Böttger studied Mathematics in Science and Engineering at the Technische Universität München (TUM) and received his MSc degree in 2013. In 2013, he joined the Research Department at MVTec Software GmbH. He received the PhD degree from the Department of Informatics at the TUM in 2019. His research interests spread between the areas of machine learning and computer vision, with special focus on visual object tracking, image segmentation, and scene text detection.