# Fast MSER

Hailiang Xu[1,2]*, Siqi Xie[3], Fan Chen[4]
[1]Alibaba Group [2]Nanjing University [3]Beijing Language and Culture University [4]Columbia University
xhl_student@163.com, coderxiesiqi1996@126.com, fc2484@columbia.edu

## Abstract

*Maximally Stable Extremal Regions (MSER) algorithms are based on the component tree and are used to detect invariant regions. OpenCV MSER, the most popular MSER implementation, uses a linked list to associate pixels with ERs. The data-structure of an ER contains the attributes of a head and a tail linked node, which makes OpenCV MSER hard to be performed in parallel using existing parallel component tree strategies. Besides, pixel extraction (*i.e. *extracting the pixels in MSERs) in OpenCV MSER is very slow. In this paper, we propose two novel MSER algorithms, called Fast MSER V1 and V2. They first divide an image into several spatial partitions, then construct sub-trees and doubly linked lists (for V1) or a labelled image (for V2) on the partitions in parallel. A novel sub-tree merging algorithm is used in V1 to merge the sub-trees into the final tree, and the doubly linked lists are also merged in the process. While V2 merges the sub-trees using an existing merging algorithm. Finally, MSERs are recognized, the pixels in them are extracted through two novel pixel extraction methods taking advantage of the fact that a lot of pixels in parent and child MSERs are duplicated.*

*Both V1 and V2 outperform three open source MSER algorithms (*28 *and* 26 *times faster than OpenCV MSER), and reduce the memory of the pixels in MSERs by* 78%.

## 1. Introduction

Invariant region extraction [36, 21, 14, 17, 42, 4, 32, 31, 34, 43, 5, 18, 6, 3] has been widely used in large scale image retrieval tasks, object detection and recognition, object tracking and view matching. The Maximally Stable Extremal Regions (MSER) algorithm was invented by Matas *et al*. [20] and optimized by Nister *et al*. [28]. It constructs a component tree [33, 24], recognizes MSERs from the tree and then extracts the pixels in MSERs. We call these three steps component tree construction, MSER recognition and pixel extraction. Note that some tasks such as wide-baseline stereo do not need the pixel extraction. Each node in the tree is an extremal region (ER), which has the characteristic that the pixels inside the ER are brighter (bright ER) or darker (dark ER) than the pixels at its outer edge. An MSER is an ER that is stable across a range of gray-level thresholds. The MSER algorithm runs in two different passes: dark to bright pass (detecting dark MSERs) and bright to dark pass (detecting bright MSERs). It has been used in wide-baseline stereo [20, 10, 19], large scale image retrieval [27], object tracking [8], object recognition [29] and scene text detection [46, 13, 26, 25, 45, 44, 12]. It has been extended to color [9], volumetric images [7], 1-D images [41] and has been optimized on FPGA [16].

The MSER algorithm requires low computing resources (suitable for embedding devices and mobile phones), and works well with small training data (MSER features are high-level handicraft features). Although deep-learning techniques are very popular in academic areas, the MSER algorithm is still active in industrial tasks such as stereo matching (possibly combined with SIFT, SURF and ORB feature descriptors), document text process and traffic sign detection, *etc*. We can also use the MSER algorithm to analyse heat-map, *i.e*. find regions whose heats exceed a certain threshold. Thus, the MSER algorithm needs to run very fast as well as use less memory (considering the relatively small memory in embedding devices and mobile phones). Besides, some optimization techniques of the MSER algorithm can be extended to other component tree algorithms.

Parallel strategies [39, 23] have been proposed to accelerate the component tree based algorithms. They divide an image into several partitions. A sub-tree merging algorithm [39] is used to merge the sub-trees which are constructed on all partitions in parallel. The sub-tree merging algorithm can correctly accumulate the attributes (the attributes must be simple enough to accumulate, *e.g*. the area of a region) of each tree node. We call this partition parallel strategy. Moschini *et al*. [23] described two partition strategies: spatial partition and intensity partition, as shown in Fig. 1. Intensity partition is suitable for the algorithms which work on pixels that are ordered by gray-levels [23].

Fig. 2 shows the comparison of partition and channel

[1]This work is an amateur research. Part of this work was done when Hailiang Xu worked in Alibaba Group (using spare time).

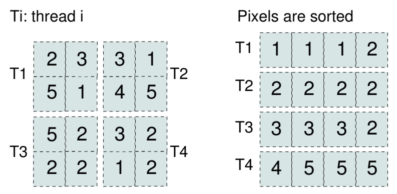

Figure 1. Left: spatial partition. Right: intensity partition.

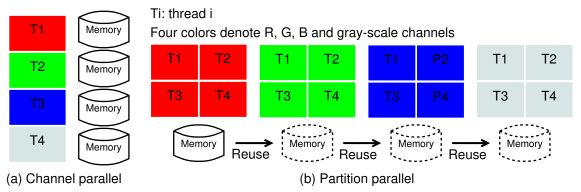

Figure 2. A channel parallel algorithm works on 4 channels at the same time and thus allocates 4 blocks of memory. However, a partition parallel algorithm only allocates 1 block of memory (the memory will be reused in all the partitions).

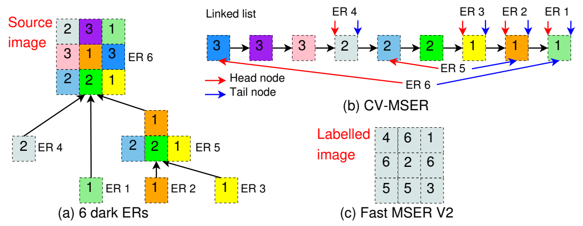

Figure 3. (a) 6 dark ERs. (b) CV-MSER generates a linked list from the source image. The data-structure of an ER contains a head and a tail linked node. In Fast MSER V1, the linked list is a doubly linked list. (c) Fast MSER V2 uses a labelled image in which each pixel records the corresponding MSER index.

parallel strategies. As the partition parallel algorithm using less running memory than the channel parallel algorithm, we focus on the partition parallel strategy.

MSER algorithms can be divided into two categories: Standard MSER [20] and Linear MSER [28]. Standard MSER takes quasi-linear time in the number of pixels, while Linear MSER takes true worst-case linear time. VLFeat MSER (*VF-MSER*) [37] is a Standard MSER. It uses the same way of the well known flooding simulation algorithm [38] in the task of watershed segmentation, and is easily performed in partition parallel using the merging algorithm in [39]. Since VL-MSER first sorts the pixels, it is suitable for intensity partition. Idiap MSER (*ID-MSER*) [2] is a Linear MSER. By using a different calculation ordering of the pixels, it uses significantly less memory and runs with better cache-locality (suitable for intensity partition), resulting in faster execution. Both of them do not implement the pixel extraction, and can only calculate moment features which are easily accumulated. More complex features such as skeleton [11] are unable to be extracted.

OpenCV MSER (*CV-MSER*) [1] is another Linear MSER algorithm, and its procedure is in Fig. 7. To calculate

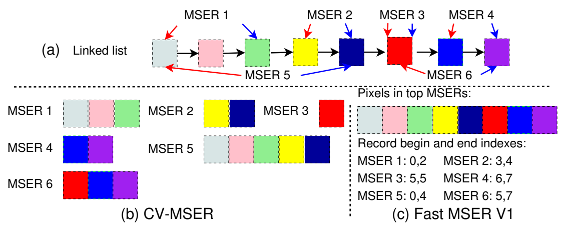

Figure 4. (a) 6 MSERs are recognized. (b) CV-MSER extracts the pixels in the 6 MSERs independently. The pixels in MSER 1, 2 and 3 are repeatedly extracted in their parent MSER 5 and 6. (c) Fast MSER V1 first stores the pixels in those top MSERs (having no parent MSERs) in a block of continuous memory. Then all the MSERs record the begin and end indexes in the continuous memory. All pixels are extracted only once.

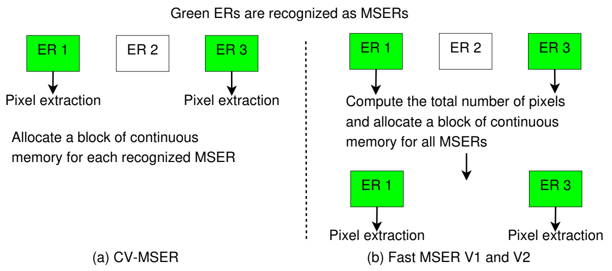
Green ERs are recognized as MSERs
ER 1
ER 2
ER 3
Pixel extraction
Allocate a block of continuous memory for each recognized MSER
Compute the total number of pixels and allocate a block of continuous memory for all MSERs
(a) CV-MSER
(b) Fast MSER V1 and V2

Figure 5. (a) The pixel extraction is performed immediately after an ER is recognized as an MSER. (b) V1 and V2 first finish the MSER recognition and then extract the pixels in MSERs. Releasing a block of continuous memory in Fast MSER is faster than releasing many blocks of continuous memory in CV-MSER.

complex features of MSERs, it extracts the pixels in each MSER, and stores the pixels in a linked list which contains the attributes of a head and a tail node (see Fig. 3 (b)). It successfully extracts the pixels, but does not take advantage of the relationship between parent and child MSER, thereby slowing down extraction speed, see Fig. 4 (b). Moreover, CV-MSER is incompatible with the partition parallel strategy because the attributes of linked nodes cannot be accumulated by the merging algorithm in [39]. Another major flaw limiting the execution speed is that CV-MSER does not allocate a block of continuous memory to store the pixels, thereby slowing memory release, see Fig. 5 (a).

**Fast MSER V1.** Since the pixels in an MSER also belong to its parent MSERs, the pixels in the MSER can be shared by its parent MSERs. By using a linked list, we can significantly accelerate the pixel extraction, as shown in Fig. 4. Thus, we would like to extract pixels through visiting the linked list. To merge sub-trees and correctly update the attributes of linked nodes, we need to disconnect the linked list into sub-lists and merge the sub-lists under certain conditions. Thus, a doubly linked list may be useful. Besides, releasing a block of continuous memory for storing the pixels in all MSERs is very fast (see Fig. 5 (b)). Inspired by these, we propose Fast MSER V1, which per-

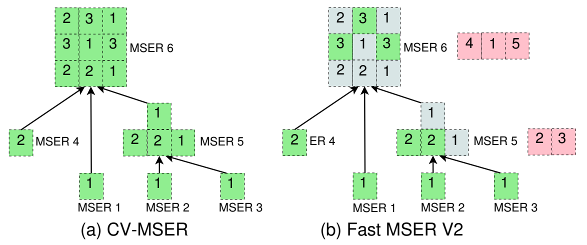

Figure 6. We assume that the memory of a pixel and an index is 1 byte. The numbers in red pixels denote MSER indexes, while the numbers in green and gray pixels denote gray-levels. Green pixels in (a) are extracted directly through visiting a linked list. In (b), pixels in an MSER $R$ are divided into child-pixels (belong to both $R$ and its child MSERs, *e.g.* gray pixels) and self-pixels (only belong to $R$, *e.g.* green pixels). Self-pixels (green pixels) are extracted first through visiting a labelled image. We store the indexes of child MSERs (red pixels) instead of extracting the pixels in child MSERs (gray pixels). Note the pixel in MSER 2 is extracted three times in (a), while it is only extracted one time in (b). V2 reduces the visiting times, which makes the pixel extraction faster. Besides, the memory size of ER 6 in (a) is 9 bytes, while in (b) is 9 bytes (three pixels and three MSER indexes). V2 compresses the memory of ER 6. However, the memory is uncompressed for MSER 5 because the area of its two children is a small value of 1 (equalling the size of an MSER index).

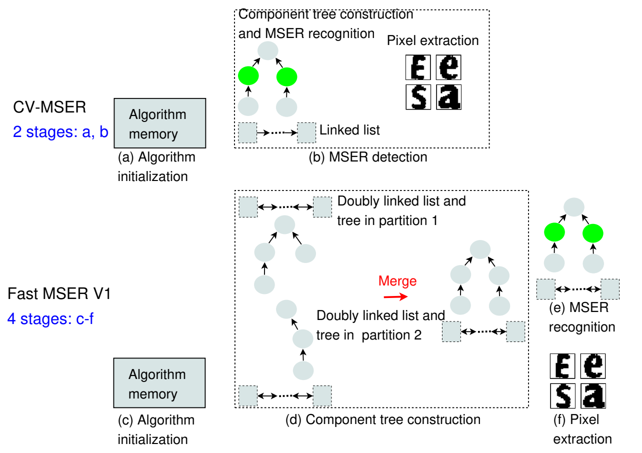
CV-MSER
2 stages: a, b
Algorithm memory
(a) Algorithm initialization
Component tree construction and MSER recognition
Pixel extraction
Linked list
(b) MSER detection
Fast MSER V1
4 stages: c-f
Algorithm memory
(c) Algorithm initialization
Doubly linked list and tree in partition 1
Merge
Doubly linked list and tree in partition 2
(d) Component tree construction
(e) MSER recognition
(f) Pixel extraction

Figure 7. The comparison of CV-MSER and Fast MSER V1. CV-MSER (a) allocates the memory needed, (b) constructs a component tree and recognizes MSERs (left part), and extracts the pixels in MSERs (right part). V1 divides the input image into multiple partitions, (c) allocates the memory needed, (d) constructs component tree in partition parallel with our novel sub-tree merging algorithm, (e) recognizes MSERs and (f) extracts pixels using our novel pixel extraction method. Note that V2 is similar as V1 but eliminates the linked list.

forms on multiple image partitions in parallel, and reduces pixel extraction time, memory release time and the memory of the pixels in MSERs. As can be seen in Fig. 7, V1 first divides an image into spatial partitions. Then it constructs sub-trees and doubly linked lists on the partitions in parallel. A novel sub-tree merging algorithm is used to merge the sub-trees into the final tree, and the doubly linked lists are also merged in the process. The attributes of linked nodes can be updated correctly. Finally, MSERs are recognized, and the pixels in them are extracted through a novel pixel extraction method (see Fig. 4 (c)) in partition parallel.

**Fast MSER V2.** Connected component algorithm uses an image to label component indexes and then extracts the pixels in each component. Moreover, since the pixels in an MSER also belong to its parent MSER, the pixels in $R$ can be shared by the parent ER, see Fig. 6. Inspired by these, we propose Fast MSER V2, which follows the same process flow as V1. However, V2 uses a labelled image (see Fig. 3 (c)) instead of a linked list to construct sub-trees, thereby merging the sub-trees easily with an exist sub-tree merging algorithm. Besides, V2 uses a novel pixel extraction method (see Fig. 6 (b)) to accelerate the pixel extraction.

Both V1 and V2 can significantly accelerate MSER extraction and reduce the memory of the pixels in MSERs. Our contributions are summarized as three folds:

1 The component tree construction, MSER recognition and pixel extraction are all partition parallel in V1 and V2. As partition parallel algorithms, V1 and V2 use less memory than channel parallel algorithms. Note that they work without synchronization mechanisms. Our code has been made publicly available at [40].

2 In V1, a novel sub-tree merging algorithm is proposed to merge sub-trees produced on all partitions. The attributes of linked nodes in the data-structure of an ER can be correctly updated.

3 Two novel pixel extraction methods to accelerate the pixel extraction and memory release, and reduce the memory size of the pixels in MSERs, are proposed in V1 and V2 respectively.

## 2. Related Works

Standard MSER [20] first uses BINSORT to sort the pixels in an image by gray-level. The computational complexity is $O(N)$, where $N$ is the number of pixels. Then the efficient union-find algorithm [35] is applied to obtain a component tree. The complexity of the union-find is quasi-linear. Finally, ERs which are maximally stable across a range of gray-levels are considered as MSERs. Some performance optimization techniques for Standard MSER can be found in [30]. VF-MSER [37] is a Standard MSER. It can be easily performed in partition parallel using the merging algorithm [39]. However, VF-MSER is much slower than Linear MSER. Besides, it does not implement the pixel extraction. Thus, such complex features (skeleton [11]) are unable to be extracted. While the component tree construction in our algorithm is similar to that in Linear MSER and our algorithm can extract the pixels in MSERs.

Linear MSER [28] behaves like a true flood-fill, provides exactly identical results as Standard MSER, and takes linear time in number of pixels. By working with a single connected component of pixels, Linear MSER uses less running memory and runs with better cache-locality. Thus Linear MSER is much faster than Standard MSER. ID-MSER and CV-MSER are both Linear MSER. For ID-MSER, it does not implement the pixel extraction. Moreover, the data-structure of an ER contains a double array with the size of $5$ to store moment features. Only MSERs use the moment features in the following process. Thus, the moment features waste a lot of running memory because only a small number of ERs ($1\%$) can be recognized as MSERs.

CV-MSER contains algorithm initialization and MSER detecting steps (see Fig. 7). The data-structure of an ER contains the attributes of a head and a tail node in a linked list. CV-MSER can extract the pixels in MSERs through visiting the linked list. However, it can not be performed in parallel because the attributes of linked nodes can not be accumulated by the merging algorithm [39]. Moreover, the pixel extraction and the memory release in CV-MSER are very slow, as explained in Fig. 5 (a) and Fig. 4 (b).

Unlike CV-MSER, to merge sub-trees and correctly update the attributes of linked nodes, Fast MSER V1 uses a doubly linked list instead of using a linked list, and uses a novel sub-tree merging algorithm. While V2 eliminates the attributes of linked nodes in the data-structure of an ER and uses a labelled image (each pixel records its corresponding ER index). V2 can be easily performed in partition parallel using the sub-tree merging algorithm [39]. Both V1 and V2 divide the MSER detecting in CV-MSER into three sequential states: component tree construction, MSER recognition and pixel extraction (see Fig. 7). The total memory size of MSERs is calculated in MSER recognition, and thus the continuous memory can be allocated before the pixel extraction (see Fig. 5 (b)). Besides, V1 only extracts the pixels in *top* MSERs (see Fig. 4 (c)), while V2 only directly extracts the self-pixels in MSERs (see Fig. 6 (b)).

## 3. Fast MSER V1

For simplicity, we take the dark to bright pass as an example to introduce V1. Fig. 7 shows the four stages of V1. We introduce them in the following sections.

### 3.1. Algorithm Initialization

We define $5$ data-structures: **1) an array of ERs**. The data-structure of an ER contains the variables of gray-level, area, the pointer of the parent ER, variation, partition index, head and tail linked nodes; **2) an array of pixels.** Each entry contains the variables of gray-level and the binary mask of an accessible pixel; **3) a doubly linked list.** Each node contains $4$ variables: the coordinate of corresponding pixel, the index referring to previous entry, the index referring to the next entry and reference index (used in our sub-tree merging algorithm); **4) a priority queue of boundary pixels [28]. 5) a component stack [28].**

Data-structure $1$ is dynamically allocated in the component tree construction. The number of entries in data-structure $5$ equals the number ($256$) of gray-levels. The size of other data-structures is $N$ (the number of pixels). The memory of each data-structure is split into $P$ partitions, where $P$ equals the number of threads.

### 3.2. Component Tree Construction

We divide an image into $P$ partitions, and then construct $P$ component trees (sub-trees) on the partitions in parallel. The tree construction on each partition is similar to CV-MSER. Note that we use a doubly linked list, while CV-MSER uses a single linked list. Finally, all the sub-trees are merged into the final tree. For example, in Fig. 8, tree $1$ and $2$ are merged (each time two trees are merged), and tree $3$ and $4$ are also merged. Then we merge the two new trees generated by the previous merging processes. Specifically, for tree $1$ and $2$, we get the each ER pair corresponding to the green adjacent pixels, then use Alg. 1 to connect them. Some important symbols in our merging algorithm are:
$split\_par$: determines to which partition an ER belongs;
$chg$: $0$ indicates there is no change in this connection;
$merged\_set$: records which nodes have been changed;
$discon\_ers$: records the ERs that have been removed from the raw linked list;
$parent\,[s]$, $gray\,[s]$, $area\,[s]$, $t\_area\,[s]$: the parent, gray-level, area and temp area (used in procedure ChangeNode) of ER $s$. $parent\,[s] = \perp$ denotes that $s$ has no parent;
$head\,[a]$, $tail\,[a]$: the head and the tail node of ER $a$;
$next\,[n]$, $prev\,[n]$, $ref\,[n]$: the next index, previous index and reference index of node $n$.

In Fig. 9, during the connection of ER $1$ and $4$ (they are adjacent), we change the parent of ER $1$ to ER $4$. The key problem here is how to update the attributes of linked nodes in the data-structure of ER $1$ and $4$. As the attributes of linked nodes can not be accumulated, the existing merging algorithm can not handle the problem. In our merging algorithm, we disconnect (see Alg. 2) the parts of the doubly linked lists corresponding to ER $1$ and $4$ (see Fig. 9 (b)), resulting in a new linked list. In the following merging process in Alg. 1, other linked nodes will be connected to the new linked list. Note that after the disconnecting process, the real tail node of ER $2$ and $3$ is the gray node (*real_tail*) with the gray-level of $3$. However, the attributes of the tail nodes in the data-structures of ER $2$ and $3$ are still the yellow node (*tail_1*) with the gray-level of $1$. Thus, we set the reference index of *tail_1* to *real_tail*, see line 12 in Alg. 2.

To avoid duplicated merging (see Fig. 10), before the merging process, we sort the ER pairs from small to large by their gray-levels (the gray-level of an ER pair is the min-

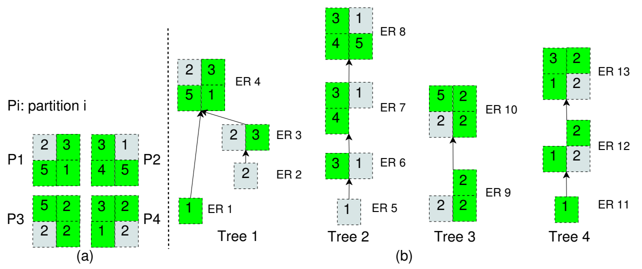
Figure 8. (a) A $4 \times 4$ image divided into 4 partitions. (b) 4 subtrees constructed on the 4 partitions. Green pixels are adjacent to other partitions. The numbers denote gray-levels.

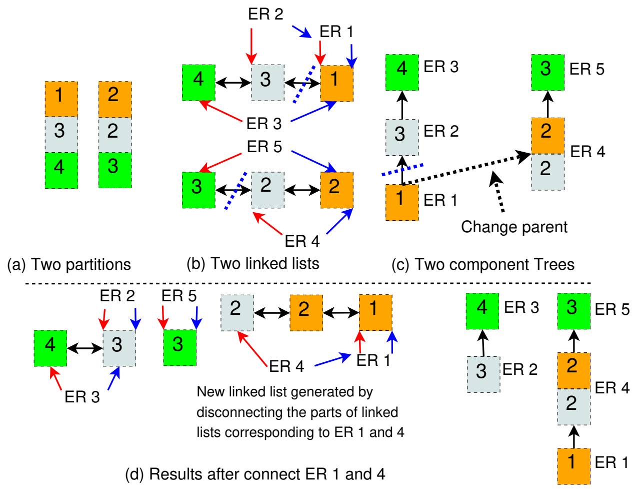
Figure 9. To connect ER 1 and 4, we disconnect the part of linked list corresponding to them, and change the parent of ER 1 to ER 4.

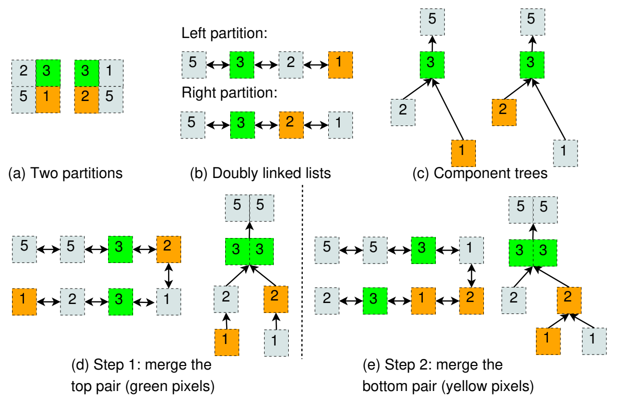
Figure 10. If we first merge the top pair, then merge the bottom pair, the linked list will be modified twice. However, if we first merge the bottom, the other merging is not needed (the component tree and the linked list is already the final result).

imum gray-level in the pair). In other words, the ER pairs with low gray-levels will be merged first.

**Algorithm 1** Connect two ERs.

```
1:  procedure ConnectER($s, b, split\_par$)
2:    $discon\_ers \leftarrow [\perp, \perp]$; $chg \leftarrow 0$; $merged\_set \leftarrow \{\}$
3:    while $s \neq \perp$ do
4:      if $b \neq \perp$ and $gray[s] > gray[b]$ then
5:        Swap s and b  end if
6:      $sp \leftarrow$ LevelRoot($parent[s]$)
7:      if $sp = b$ and $chg = 1$ then Break loop  end if
8:      if $b = \perp$ then
9:        ProcessSmall($s, sp$); $s \leftarrow sp$; $chg \leftarrow 1$
10:     else
11:       $bp \leftarrow$ LevelRoot($parent[b]$)
12:       if gray[s] = gray[b] or $sp = \perp$ or $gray[sp] > gray[b]$ then
13:         for $r \in \{s, b\}$ do
14:           if $discon\_ers[Index(r)] \neq r$ then
15:             Disconnect($r, merged\_set$) // See Alg. 2
16:             $discon\_ers[Index(r)] \leftarrow s$
17:           end if
18:         end for
19:         ChangeNode($b, s$) // See Alg. 2
20:         $parent[s] \leftarrow b$; $s \leftarrow b$; $b \leftarrow sp$; $chg \leftarrow 1$
21:       else
22:         ProcessSmall($s, sp$); $s \leftarrow sp$; $chg \leftarrow 1$
23:       end if
24:     end if
25:   end while
26:   $ref[node] \leftarrow \perp$, where $node \in merged\_set$
```

## 3.3. MSER Recognition

We use 4 rules to recognize MSERs: 1) the area is in the range of $[min\_p, max\_p]$; 2) it is stable that its variation is smaller than $var$; 3) it is maximally stable. 4) it is obviously larger than its child MSERs by $1 + dvar$ times.

In rule 2, we use $\delta$ to calculate the variation $var$ of an ER. $var$ is defined as $var = \frac{|R(+\delta)|-|R|}{|R|}$ [37, 2], where $|R|$ denotes the area of ER $R$, $R(+\delta)$ is the ER $+\delta$ levels up which contains $R$, and $|R+\delta| - |R|$ is the area difference of the two regions. We first calculate the variations of all ERs and simultaneously apply rule 1 and 2 to filter ERs in parallel. Rule 3 and 4 can not be applied in parallel because parent and child information is used at the same time.

## 3.4. Pixel Extraction

Once MSERs are recognized, we first extract the pixels in top MSERs (see Fig. 4 (c)) through visiting the doubly linked list in parallel. The extracted pixels are stored in a block of continuous memory. Then each MSER records the begin and end indexes in the continuous memory.

**Algorithm 2** Help functions.

```
procedure Disconnect($a, merged\_set$)
$real\_tail \leftarrow$ RealTail($tail[a]$)
if $prev[head[a]] \neq \perp$ then
    $next[prev[head[a]]] \leftarrow next[real\_tail]$ end if
if $next[real\_tail] \neq \perp$ then
    $prev[next[real\_tail]] \leftarrow prev[head[a]]$ end if
$p \leftarrow$ LevelRoot($parent[a]$)
if $p \neq \perp$ then
    $p\_real\_tail \leftarrow$ RealTail($tail[p]$)
    $area[p] \leftarrow area[p] - t\_area[a]$
    if $tail[a] = tail[p]$ or $p\_real\_tail = real\_tail$ then
        $ref[real\_tail] \leftarrow prev[head[a]]$
        Add $real\_tail$ to $merged\_set$ end if
end if
$tail[r] \leftarrow real\_tail; prev[head[r]], next[tail[r]] \leftarrow \perp$

procedure LevelRoot($r$)
while $gray[parent[r]] = gray[r]$ do $r \leftarrow parent[r]$
end while    return $r$

procedure ChangeNode($a, b$)
$next[tail[a]] \leftarrow head[b]; prev[head[b]] \leftarrow tail[a]$
$tail[a] \leftarrow tail[b]$
$area[a] \leftarrow area[a] + area[b]; t\_area[a] \leftarrow area[a]$

procedure ProcessSmall($s, sp$)
if $sp \neq \perp$ and $discon\_ers[Index(s)] = s$ then
    Disconnect($sp$); $discon\_ers[Index(s)] \leftarrow s$
    ChangeNode($sp, s$) end if

procedure RealTail($a$)
while $ref[a] \neq \perp$ do $a \leftarrow ref[a]$ end while return $a$

procedure Index($a, split\_par$)
return $partition[a] < split\_par$ ? 0 : 1
```

## 4. Fast MSER V2

Fast MSER V2 is different from Fast MSER V1 in two aspects (more details of V2 can be found in our code). First, V2 uses a labelled image instead of a linked list (the data-structure of an ER does not contain the linked nodes), thereby merging sub-trees easily with the merging algorithm [39]. Second, V2 first adds the coordinates of self-pixels (see Fig. 6) to their corresponding MSERs through visiting a labelled image (this process can be easily performed in partition parallel). If a pixel corresponds to a non-MSER $R_n$ (recognized as a non-MSER), the MSER with the smallest gray-level in the parent ERs of $R_n$ is its corresponding MSER. In Fig. 6, self-pixels are sequentially extracted (row-major order) to MSER 4, 6, 1, 6, 2, 6, 5, 5 and 3. Note that the memory addresses of MSER 6 are relocated 3 times, resulting in slower pixel extraction. Then V2 iterates over the MSERs in the MSER array (sorted by gray-level from small to large), and adds their MSER indexes (instead of replicating their pixels) to their parent MSERs. Note that this process can not be performed in parallel because it visits the MSER array sequentially.

Figure 11. Illustration of 3 images (left) from ICDAR dataset and 2 images (right) from DetectorEval dataset.

## 5. Experimental Validation

The proposed algorithm was implemented in C++. Timings were performed on a Lenovo laptop with a 4-core Intel i7-6700HQ processor and 8GB of RAM memory. We use two measures: execution time and memory usage. Note that if not mentioned otherwise, execution time and memory usage are the averages on each image.

### 5.1. Datasets

In order to evaluate Fast MSER V1 and V2 on images with various sizes, several images with a fixed size are selected and then resized to different sizes. We use two real-world image datasets: the test set of focused scene text in ICDAR robust reading competition [15] (*ICDAR* dataset), and the feature detector evaluation sequences of [22] (*DetectorEval* dataset). ICDAR dataset consists of 233 images with various sizes from 350 $\times$ 200 to 3888 $\times$ 2592, while DetectorEval dataset contains 49 images with various sizes from 800 $\times$ 640 to 1000 $\times$ 700. We resize all the images in the two datasets to a fixed size of 3888 $\times$ 2592 (the image size is about 10M, where M denotes mega-pixel). Then we generate the dataset with 5 scales of images from the size of 2M (corresponding to the size of 1732 $\times$ 1154) to 10M. Each image contains 3 channels: R, G and B. We would like to compare channel parallel MSER algorithms on our computer with 4 cores (executing a thread on each core), and thus add a gray-scale channel for each image. In this way, an image (4 channels) with the size of 10M actually has 40 mega-pixel. Some image examples are shown in Fig. 11.

To avoid process scheduling, execution times were produced by running five independent times on the images of each size and taking the fastest run [28].

### 5.2. Parameter Configurations

We use two configurations: *TextDetection* ($min\_p = 20$, $max\_p = 0.25 \times N$, $var = 0.5$, $dvar = 0.1$, $\delta = 1$) and *DetectorEval* ($min\_p = 20$, $max\_p = 0.25 \times N$, $var = 0.25$, $dvar = 0.2$, $\delta = 5$). Configurations only

| Category | 2M | 4M | 6M | 8M | 10M |
|---|---|---|---|---|---|
| ICDAR | 22.2 | 29.1 | 32.5 | 36.5 | 39.8 |
| ICDAR Sort | 20 | 26.8 | 29.3 | 33.6 | 34.6 |
| DE | 22 | 30.8 | 32.5 | 40.78 | 43.8 |
| DE Sort | 20 | 25.5 | 30.2 | 33 | 34.7 |

Table 1. Execution times (millisecond) of our novel sub-tree merging algorithm on images with different sizes. ICDAR and DE denote that the merging algorithm runs on ICDAR and DetectorEval dataset respectively. The suffix "Sort" indicates that the merging algorithm runs with the sorted ER pairs (see Sec.3.2).

affect the execution times on the MSER recognition and pixel extraction. *TextDetection* [45] is used in scene text detection tasks. $\delta$ is set to 1 that makes it possible to detect most challenging cases [13]. Less MSERs are recognized under *DetectorEval* because the criteria for an ER to be recognized as an MSER is more strict ($var$, $dvar$ and $\delta$ are different in the two configurations).

In our experiments, we only show the results on ICDAR dataset under *TextDetection* and the results on DetectorEval dataset under *DetectorEval* (actually, the results under the *same configuration* on the two datasets are similar).

## 5.3. Tests of Merging Times

The sub-tree merging process in V1 is very fast ((taking about 1 ms on processing an image with 10 mega-pixels), here we only evaluate the merging process in V2. Tab. 1 shows that the merging process with the sorted ER pairs is faster than the merging process without the sorted ER pairs, which demonstrates that it is necessary to sort the ER pairs before merging two sub-trees.

## 5.4. Comparison of Memory Usage

The memory usage of MSER algorithms are mainly defined by the input image size. Fig. 13 shows the memory usage of different algorithms. CV-MSER+ is our implementation that fully optimizes CV-MSER by replacing the vector data-structures (allocating memory is slow) with pointer arrays, and exploiting continuous memory to store output MSERs instead of storing MSERs independently (releasing memory is time consuming). The algorithms with "CP" are channel parallel versions.

For non-parallel algorithms, ID-MSER dynamically allocates the running memory in the component tree construction, resulting in the least memory usage and slow component tree construction. The data-structure of an ER in VF-MSER is very simple because it does not extract the pixels in MSERs. Compared to CV-MSER, CV-MSER+ uses less memory because the region only stores its parent region instead of storing its parent and child regions.

For parallel algorithms, V1 and V2 use significantly less memory than other parallel algorithms. Both V1 and V2 dynamically allocate the array of ERs. Compared to V1, V2 uses less memory because it uses a labelled image instead of a doubly linked list. V2 uses the minimal memory of all parallel algorithms. Note that CPCV-MSER, CPCV-MSER+, CPVF-MSER and CPID-MSER use $4$ times as much memory as their non-parallel versions.

## 5.5. Tests of Execution Times

In Fig. 12(a), the speeds (mega-pixel per second) of those on 10M images are $0.36$ (CV-MSER), $7.94$ (CV-MSER+), $0.43$ (VF-MSER), $4.41$ (ID-MSER), $0.98$ (CPCV-MSER), $18.78$ (CPCV-MSER+), $1.05$ (CPVF-MSER), $15.51$ (CPID-MSER), $27.15$ (Fast MSER V1) and $25.23$ (Fast MSER V2). V1 and V2 are $28$ and $26$ times faster than CPCV-MSER, and only use $\frac{1}{9}$ and $\frac{1}{18}$ running memory of CPCV-MSER. Compared to CV-MSER+, V1 and V2 reach the speed-ups of $3.42$ and $3.18$.

As a standard MSER, VF-MSER takes too much time. ID-MSER is not fully optimized. Although both of them does not extract the pixels in MSERs, they and their parallel versions are all slower than Fast MSER. CV-MSER+ is much faster than CV-MSER because it is fully optimized.

In Fig. 12(b), the conclusions are similar to the conclusions in Fig. 12(a). All algorithms in Fig. 12(b) are faster because less MSERs are recognized under *DetectorEval*.

We also investigate the performance of V1 and V2 with respect to different $\delta$ (a key parameter in MSER algorithms). As can be seen in Fig. 14, lower $\delta$ implies significantly more detected MSERs (the pixel extraction may takes more running time), while higher $\delta$ implies less detected MSERs (the pixel extraction may takes less time). Compared to CV-MSER+, the speed-ups of V1 are $3.42$ ($\delta = 1$), $3.28$ ($\delta = 2$), $3.25$ ($\delta = 3$), $3.09$ ($\delta = 4$) and $3.09$ ($\delta = 5$). While the speed-ups of V2 are $3.18$, $3.17$, $3.21$, $3.07$ and $3.06$. Thus, the larger $\delta$, the lower speed-ups.

## 5.6. Tests of Execution Times on Different Steps

In Tab. 2, compared to CV-MSER+, V1 and V2 reach the speed-ups of $2.8$ and $2.95$ in the component tree construction, and reach the speed-ups of $6.7$ and $3.6$ in the pixel extraction, which demonstrates the efficiency of our algorithms. However, the speed-ups in MSER recognition are $1.1$ and $1.2$ because minimal suppression of variation is not parallel. Since MSER recognition takes less time, the speed-up of whole V1 and V2 are still the high values of $3.42$ and $3.18$. V1 is faster than V2 in the pixel extraction (see the reason in Sec. 4), but is slightly slower than V2 in other stages. Thus, when less MSERs are recognized (in smooth images or under a configuration with a high $\delta$ and a small $var$) or the pixel extraction is not needed, we prefer to use V2. Note that V2 also uses less memory than V1.

Compared to CV-MSER+, V1 and V2 reduce $85\%$ and $72\%$ execution time in the pixel extraction. The average

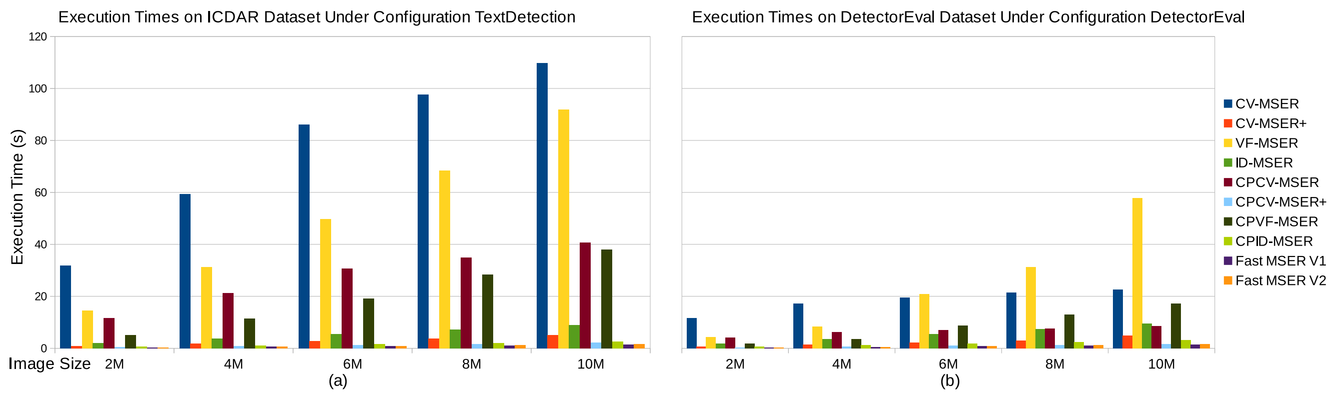

Figure 12. Execution times on the two datasets under the two configurations.

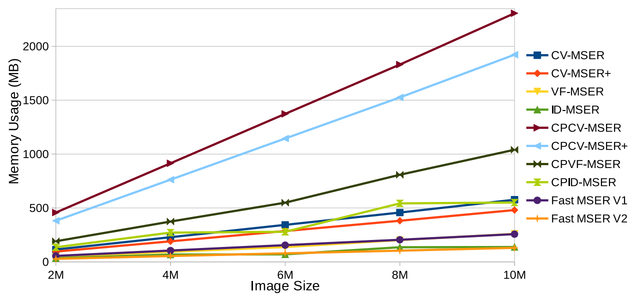

Figure 13. Comparison of memory usage on all algorithms. Notice that the memory usage of ID-MSER and Fast MSER V2 are so close that their lines overlapped.

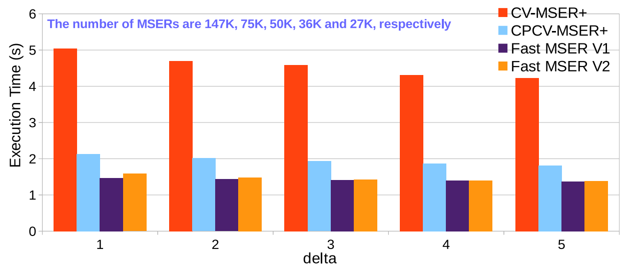

Figure 14. Execution times (second) of Fast MSER V1 and V2 with respect to different $\delta$ on the 10M images in ICDAR dataset. The other parameters are the same as configuration *TextDetection*.

| Algorithms | Init | Tree | Reco | Extr | Rele |
|---|---|---|---|---|---|
| CV-MSER | 0.339 | 3.4 | 0.133 | 83.1 | 22.86 |
| CV-MSER+ | 0.002 | 3.116 | 0.083 | 1.399 | 0.441 |
| VF-MSER | 0.001 | 91.75 | 0.058 | \ | 0.025 |
| ID-MSER | 0.046 | 8.807 | 0.156 | \ | 0.024 |
| Fast MSER V1 | 0.002 | 1.107 | 0.074 | 0.209 | 0.081 |
| Fast MSER V2 | 0.001 | 1.054 | 0.068 | 0.385 | 0.077 |

Table 2. Execution times (second) in different steps on the 10M images in ICDAR dataset under *TextDetection*. Init, Tree, Reco, Extr and Rela indicate algorithm initialization, component tree construction, MSER recognition, pixel extraction and memory release. About 147 thousand MSERs are recognized in an image.

| Algorithms | Init | Tree | Reco | Extr | Rele |
|---|---|---|---|---|---|
| CV-MSER | 0.306 | 4.015 | 0.101 | 14.27 | 3.97 |
| CV-MSER+ | 0.002 | 4.184 | 0.07 | 0.518 | 0.173 |
| VF-MSER | 0.001 | 57.68 | 0.024 | \ | 0.025 |
| ID-MSER | 0.061 | 9.45 | 0.571 | \ | 0.02 |
| Fast MSER V1 | 0.002 | 1.19 | 0.051 | 0.132 | 0.058 |
| Fast MSER V2 | 0.001 | 1.15 | 0.05 | 0.312 | 0.048 |

Table 3. Execution times (second) in different steps on the 10M images in DetectorEval dataset under *DetectorEval*. About 33 thousand MSERs are recognized in an image.

memory sizes of the pixels in MSERs in an image produced by CV-MSER+ and Fast MSER (V1 and V2) are 1.3GB and 296MB. Our two novel pixel extraction methods both compress the memory of the pixels in MSERs by 78%, thereby reducing memory release time by 82%. As a Standard MSER, VF-MSER takes much time in component tree construction. ID-MSER is not fully optimized and is slower than other Linear MSER algorithms.

In Tab. 3, the conclusions are similar to Tab. 2. Note that, compared to CV-MSER+, V1 and V2 only reduce pixel extraction time by 75% and 40% because less MSERs are recognized under *DetectorEval*.

## 6. Conclusion

In this paper, we have proposed Fast MSER V1 and V2. V1 integrates a novel sub-tree merging algorithm and a novel pixel extraction method, while V2 uses an existing sub-tree merging algorithm and another novel pixel extraction method. V1 and V2 are 28 and 26 times faster than OpenCV MSER and use less memory. In future work, we will improve the speed-up in MSER recognition, and further reduce the running memory of V1 and V2.

## References

[1] https://opencv.org/opencv-3-4-1.html.

[2] Idiap mser. https://github.com/idiap/mser.

[3] Daniel Barath. Five-Point fundamental matrix estimation for uncalibrated cameras. In *CVPR2018*, pages 235–243, 2018.

[4] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: speeded up robust features. In *ECCV*, pages 404–417, 2006.

[5] Jiawang Bian, Wenyan Lin, Yasuyuki Matsushita, Saikit Yeung, Tandat Nguyen, and Mingming Cheng. Gms: Grid-based motion statistics for fast, ultra-robust feature correspondence. In *CVPR*, pages 2828–2837, 2017.

[6] Donald G Dansereau, Bernd Girod, and Gordon Wetzstein. LiFF: Light field features in scale and depth. In *CVPR*, 2019.

[7] Michael Donoser and Horst Bischof. 3d segmentation by maximally stable volumes (MSVs). In *ICPR*, pages 63–66, 2006.

[8] Michael Donoser and Horst Bischof. Efficient maximally stable extremal region (MSER) tracking. In *CVPR*, pages 553–560, 2006.

[9] Pererik Forssen. Maximally stable colour regions for recognition and matching. In *CVPR*, pages 1–8, 2007.

[10] Pererik Forssen and David G Lowe. Shape descriptors for maximally stable extremal regions. In *ICCV*, pages 1–8, 2007.

[11] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Prentice-Hall, 3 edition, 2006.

[12] Tong He, Weilin Huang, Yu Qiao, and Jian Yao. Text-attentional convolutional neural network for scene text detection. In *TIP*, pages 2529–2541, 2016.

[13] Weilin Huang, Yu Qiao, and Xiaoou Tang. Robust scene text detection with convolution neural network induced mser trees. In *ECCV*, pages 497–511, 2014.

[14] Timor Kadir, Andrew Zisserman, and Michael Brady. An affine invariant salient region detector. In *ECCV*, pages 228–241, 2004.

[15] D. Karatzas, F. Shafait, S. Uchida, M. Iwamura, L. G. Bigorda, S. R. Mestre, J. Mas, D. F. Mota, J. A. Almazan, and L. P. Heras. ICDAR 2013 robust reading competition. In *ICDAR*, pages 1484–1493, 2013.

[16] Fredrik Kristensen and W J Maclean. Real-time extraction of maximally stable extremal regions on an fpga. In *International Symposium on Circuits and Systems*, pages 165–168, 2007.

[17] David G Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[18] Zixin Luo, Tianwei Shen, Lei Zhou, Jiahui Zhang, Yao Yao, Shiwei Li, Tian Fang, and Long Quan. ContextDesc: Local descriptor augmentation with cross-modality context. In *CVPR*, 2019.

[19] Andrzej luzek. Improving performances of mser features in matching and retrieval tasks. In *ECCV*, pages 759–770, 2016.

[20] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, 22(10):761–767, Sep 2004.

[21] Krystian Mikolajczyk and Cordelia Schmid. Scale and affine invariant interest point detectors. *International Journal of Computer Vision*, 60(1):63–86, 2004.

[22] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *PAMI*, 27(10):1615–1630, 2005.

[23] Ugo Moschini, Arnold Meijster, and Michael H F Wilkinson. A hybrid shared-memory parallel max-tree algorithm for extreme dynamic-range images. *PAMI*, 40(3):513–526, 2018.

[24] Laurent Najman and Michel Couprie. Building the component tree in quasi-linear time. *TIP*, 15(11):3531–3539, 2006.

[25] Lukas Neumann and Jiri Matas. A method for text localization and recognition in real-world images. In *ACCV*, pages 770–783, 2010.

[26] Lukas Neumann and Jiri Matas. Real-time scene text localization and recognition. In *CVPR*, pages 3538–3545, 2012.

[27] David Nister and Henrik Stewenius. Scalable recognition with a vocabulary tree. In *CVPR*, pages 2161–2168, 2006.

[28] David Nister and Henrik Stewenius. Linear time maximally stable extremal regions. In *ECCV*, pages 183–196, 2008.

[29] Stepan Obdrzalek and Jiri Matas. Object recognition using local affine frames on distinguished regions. In *BMVC*, pages 1–10, 2002.

[30] Michal Perdoch. *Maximally Stable Extremal Regions and Local Geometry for Visual Correspondences*. PhD thesis, Czech Technical University in Prague, 2011.

[31] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *ECCV*, pages 430–443, 2006.

[32] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary R Bradski. ORB: An efficient alternative to SIFT or SURF. In *ICCV*, pages 2564–2571, 2011.

[33] Philippe Salembier, Albert Oliveras, and Luis Garrido. Antiextensive connected operators for image and sequence processin. *TIP*, 7(4):555–570, 1998.

[34] Johannes L Schonberger, Hans Hardmeier, Torsten Sattler, and Marc Pollefeys. Comparative evaluation of hand-crafted and learned local features. In *CVPR*, pages 6959–6968, 2017.

[35] Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975.

[36] Tinne Tuytelaars and Luc Van Gool. Matching widely separated views based on affine invariant regions. *International Journal of Computer Vision*, 59(1):61–85, 2004.

[37] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. http://www.vlfeat.org/, 2008.

[38] Luc Vincent and Pierre Soille. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *PAMI*, 13:583–598, Jun. 1991.

[39] Michael H F Wilkinson, Hui Gao, Wim H Hesselink, Janeppo Jonker, and Arnold Meijster. Concurrent computation of attribute filters on shared memory parallel machines. *PAMI*, 30(10):1800–1813, 2008.

[40] Hailiang Xu, Siqi Xie, and Fan Chen. Fast MSER. https://github.com/mmmn143/fast-mser, 2020.

[41] Ismet Zeki Yalniz, Douglas Gray, and R Manmatha. Efficient exploration of text regions in natural scene images using adaptive image sampling. In *ECCV*, pages 427–439, 2016.

[42] Kwang Moo Yi, Eduard Trulls, Vincent Lepetit, and Pascal Fua. Lift: Learned invariant feature transform. In *ECCV*, pages 467–483, 2016.

[43] Kwang Moo Yi, Eduard Trulls, Yuki Ono, Vincent Lepetit, Mathieu Salzmann, and Pascal Fua. Learning to find good correspondences. In *CVPR*, pages 2666–2674, 2018.

[44] Xucheng Yin, Weiyi Pei, Jun Zhang, and Hongwei Hao. Multi-orientation scene text detection with adaptive clustering. *PAMI*, 37:1930–1937, Sep. 2015.

[45] Xucheng Yin, Xuwang Yin, Kaizhu Huang, , and Hongwei Hao. Robust text detection in natural scene images. *PAMI*, 36(5):970–983, May 2014.

[46] Zheng Zhang, Chengquan Zhang, Wei Shen, Cong Yao, Wenyu Liu, and Xiang Bai. Multi-oriented text detection with fully convolutional networks. In *CVPR*, pages 4159–4167, 2016.