Real-Time Extraction of Maximally Stable Extremal Regions on an FPGA

Fredrik Kristensen[1] and W. James MacLean[2]
[1]Dept. of Electroscience, Box 118, Lund University, Sweden
[2]Dept. of Electrical & Computer Engineering, University of Toronto, Canada
Email: Fredrik.Kristensen@es.lth.se, maclean@eecg.toronto.edu

***Abstract*— This paper describes the implementation of a real-time Maximally Stable Extremal Region (MSER) detector. In order to reach real-time performance, both algorithmic and memory issues have been addressed. The Union-find algorithm, which is the heart of the MSER detector, is extended to create linked regions that significantly decrease the time to extract MSERs. Hash indexed memory structures are used to locate stored regions fast while keeping the amount of stored data low. The design is verified by including it in a demonstrator circuit. Timing and memory requirements are presented for the demonstrator and as a function of image resolution.**

I. Introduction

The MSER algorithm is an interest region detector originally used in wide-baseline stereo matching [1]. What separates MSER from many other interest region detectors is that it operates on the input image directly without any smoothing, which results in detection of both fine and coarse structures. That MSER performs well compared to other local detectors is shown in [2]. Moreover, MSER is used in applications such as automatic 3D-reconstruction from a set of images [3] and object and scene retrieval in videos, also known as Video Google [4]. In Video Google, MSER is used together with an affine invariant detector.

The aim of this paper is to show that MSER extraction can be included in an embedded system to handle real-time image streams, i.e. 25 frames per second (fps) or more. With real-time performance MSER extraction can be used, for example, for real-time 3D-reconstruction of stereo camera inputs, or object tracking [5] in an automated surveillance system [6]. The process of creating databases for Video Google could be sped up; by the time you have seen the movie it is in the database. To verify real-time performance, a hardware demonstrator of MSER is designed and presented in this paper. The demonstrator is implemented on a Xilinx XC2VP100 Field Programmable Gate Array (FPGA), and allows a user to change various input parameters and see the effect of the changes presented in real-time on a monitor.

II. Maximally Stable Extremal Regions

An informal explanation of an MSER detector is as follows. Consider an image in which all pixels are removed, leaving an empty pixel grid. We now start to reinsert the pixels in intensity order, i.e. first we place all black (intensity=0) pixels at their correct locations, then we place all pixels with an intensity value 1 and so on until the complete image is restored. During this process there will appear regions (clusters) of connected pixels that will grow and connect to other regions as more and more pixels of higher intensity are placed. The rate of growth as a function of intensity ($q(i)$), is measured for all these regions, and a region is detected as an MSER when the growth rate has a local minimum. The sensitivity of the detection is controlled with a parameter (Δ). In Fig. 6 typical examples of detected MSERs are shown. The formal definition of MSER is found in [1].

This paper explains how MSER detection for minimum intensity regions, i.e. dark regions, can be implemented. To perform MSER detection for maximum intensity regions, bright regions, the same method can be used only the input is inverted, i.e. $I_{inverted} = I_{max} - I$.

The MSER algorithm can be divided into four major parts:

1) Preprocessing. Pixels are sorted in intensity order, and the number of pixels for each intensity is determined.
2) Clustering. A representation of all regions at each intensity level is created.
3) MSER detection. The size, $|Q|$, of all regions are tracked and the growth rate, q, are monitored for local minimums.
4) Display result. All pixels belonging to a detected MSER are identified and presented as an output.

A. Preprocessing

The minimum amount of data that is required from this part is a vector with pixel positions sorted by intensity level and the number of pixels in each intensity level, the intensity histogram. Thus, either a sorting algorithm can be used to find out the intensity histogram or a histogram can be calculated first and used as an input to a sorting algorithm. The former will result in a lot of data shuffling, since the final position of a pixel is not known until all other pixels with a lower intensity value are processed. The latter alternative, used in this paper, is preferred since bin sort, or radix sort [7], can be implemented in a very efficient way if the intensity level histogram is known before the sorting starts. With the histogram a cumulative histogram can be formed as the sum of all lower bins, e.g. the histogram $\{1, 4, 2, 6\}$ will produce the cumulative histogram $\{0, 1, 5, 7\}$. A pixel is then sorted by using the intensity level of the pixel as an index to the cumulative histogram, which will produce the correct address in the sorted vector. Finally, the indexed bin in the cumulative histogram is increased with one before the next input is read.

B. Clustering

To keep track of regions of connected pixels the Union-find algorithm [7] is used. This algorithm can determine if two pixels belong to the same region or not, and if they do not belong to same region it can force them to become part of the same region. In addition, it keeps track of the size of each region.

The algorithm uses a memory, with as many positions as the input image, where each location corresponds to the status of the pixel at the same location in the original input image. This memory is called the Region Map (RM). Each memory location in RM holds one number, U, that is interpreted as:

- $U=0$ This pixel is not connected to any other pixel, or this pixel has not yet been placed in the RM.
- $U>0$ This pixel is part of the same region as the pixel at position U.
- $U<0$ This pixel is the reference point for this region and $1-U$ equals the number of pixels in this region.

Since the Union-find algorithm does not keep track of which pixels have already been placed, a single bit of memory is added to each location in the RM to keep track of this. This removes the uncertainty when $U=0$. A simple example of how pixels are added to the RM is shown in Fig. 1.

To find out which region a pixel belongs to, the RM is read at the pixel location (p) and U_0 is returned, i.e. $RM(p)=U_0$. If U_0 is greater than zero the memory is read again this time at the location given by U_0 which returns U_1 ($RM(U_0)=U_1$), and so on until $U_k \leq 0$. Then, U_{k-1} is the position of this region's reference point and $1-U_k$ is the size of the region. To find out if two pixels are part of the same region, the above method is used to find the reference points for both pixels, and then those are compared, if equal the pixels belong to the same region. If the pixels belonged to different regions these regions can be merged simply by writing the reference point location of one of the pixels to the reference point of the other pixel, and updating the size of the other reference point. In order to keep the depth of the RM low, i.e. number of reads before a reference point is found, and to ensure stable region reference points, the smaller region becomes part of the larger region when two regions merge.

The complete procedure for each new pixel (p_{new}) placed in the RM is; for each 4-connected neighboring pixel (p_{neigh}) of p_{new} that is already placed invoke the Union-find algorithm with p_{new} and p_{neigh} as input, finally mark p_{new} as placed in the RM. In Fig. 2 an example of how a newly placed pixel will merge two regions is shown.

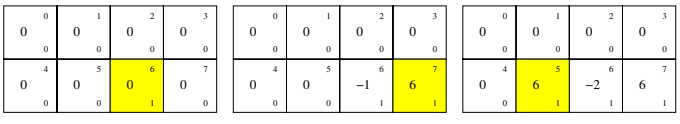

Fig. 1. Basic Union-find operation. Each square represent one pixel position in the RM, the position is marked in the upper right corner. The lower right number indicates if a position has been placed in the RM yet (1) or not (0) and the large middle number is the union-find number (U). The yellow position shows which pixel is added to the RM. The sequence is shown left to right.

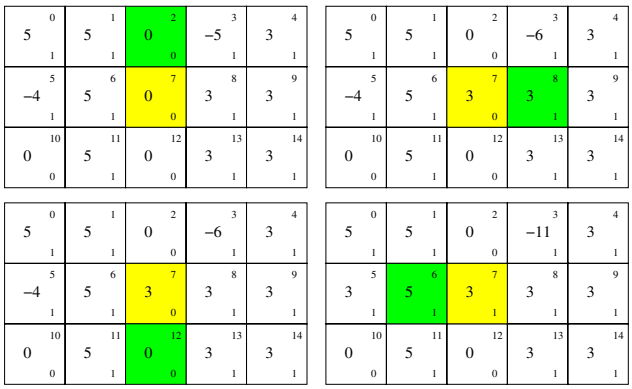

Fig. 2. How a new pixel at position 7, marked in yellow, is placed in the RM and how it causes two regions to merge. Each neighborhood position, marked in green, is checked for an already placed pixel. If there is a pixel, the corresponding region reference positions are found and updated accordingly. The sequence is shown top to bottom and left to right.

An advantage of the Union-find algorithm, beside its simplicity, is that the location of region reference points is fixed. A reference point will never move once it is created, it can however disappear if it is merged with another region. This means that to keep track of the growth of a region it is enough to monitor one location in the RM until that reference point disappears.

The disadvantage of the Union-find algorithm is that given the location of a region reference point there is no fast way to find which pixels belong to this region. The only way is to check every pixel in the RM if they have the same region reference point. This is addressed further in Section II-D.

C. Detection

The MSER is detected as a local minimum of $q(i) = |Q_{i+\Delta} \setminus Q_{i-\Delta}|/|Q_i|$, i.e. if the number of new pixels in a region at intensity level $i+\Delta$ compared to the region at level $i-\Delta$, normalized with the region size at level i is a local minimum. This means that all regions at intensity level $[i-\Delta, i, i+\Delta]$ must be known. To keep one RM for each of the three intensity levels is unnecessary, since it is only the region sizes, $|Q|$, that are used to calculate $q(i)$. Instead all regions sizes from intensity levels $i-\Delta$ to $i+\Delta$ are stored, the regions sizes at the intermediate intensities are needed since they will be used to calculate q at the succeeding intensity levels.

A simple and efficient memory structure and memory storage scheme that can access all three $|Q|$-values of a region at the same time is to use 2^n hash indexed memories, where $2^n > 2\Delta+1$. The $|Q|$-values for all regions at one intensity level are stored in one memory and the complete memory structure will thus hold all $|Q|$-values from intensity levels $i+\Delta$ down to $i+\Delta-2^n-1$. In addition, to use 2^n memories removes the need for a mod-operator to calculate which memory to store $|Q(i)|$ in, i.e. $mod(i, 2^n)$ is known without computation. Since the location of a region reference point is the same for all intensity levels, due to the Union-find algorithm, a region can be easily located in the $|Q|$-memory if

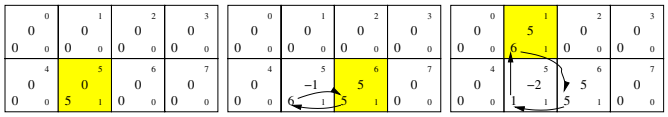

Fig. 3. Linked regions. Each pixel position in the RM is now extended with a pointer, shown in the lower left corner. For each new pixel added to a region, the new pixel and the region reference point swap pointer and thereby creating a linked region.

it is stored at a position based on the location of the reference point. In order to compromise between memory space and time to locate a region in $|Q|$-memory, hash indexing [7] is used. The most significant bits of the reference point location are used as hash function.

After all pixels from an intensity level have been placed, the size of all regions that grew is updated in the $|Q(i)|$-memory. The size of all non growing regions is copied from the $|Q(i-1)|$-memory, to ensure that a region can be found at the same location in all $|Q|$-memories. In order to limit the number of regions that are monitored a minimum and maximum size limit is imposed. These limits are used since very small or very large regions serve a limited use as distinguished regions [1]. In the hardware demonstrator these limits are set via signals at runtime.

To use as little memory as possible and still be able to decide if $q(j)$ is a local minimum, only $q(j)$ and dq/dj are stored in a hash indexed memory with the same number of addresses and hash functions as the $|Q|$-memory. dq/dj is a one-bit flag that keeps track of $sign(q(j)-q(j-1))$. To find a local minimum, $q(j+1)$ is calculated and compared to the stored $q(j)$ and dq/dj. If $q(j+1)$ is larger than $q(j)$ and dq/dj is negative a local minimum is found at $q(j)$. Finally, $q(j+1)$ and the new dq/dj is written to the q-memory.

The implementation of the MSER algorithm can now be summarized as follows. For each intensity level, i, do:

1) Place all pixels with intensity i in the RM with help of the Union-find algorithm. The size of all regions that grew are written in $|Q(i)|$-memory and all non growing regions are copied from $|Q(i-1)|$-memory.
2) $j = i - \Delta$. If $j \geq \Delta$, go through $|Q|$-memory. If $|Q(j-\Delta)| \neq 0$, calculate $q(j)$ and check for MSER.

The fact that the $|Q|$-memory is hash indexed will in many cases reduce the time it takes to go through the memory, since there are fewer hash entries to the memory than there are memory addresses. If a hash entry is empty, then all consecutive memory addresses until the next hash entry are also empty and can be ignored. When a hash entry is occupied, consecutive memory addresses have to be read until an empty address is found, in which case the search can continue at the next hash entry.

D. Display Result

Once an MSER is detected, all pixels that are part of that region have to be identified and presented as an output. In order to perform this part efficiently, two difficulties have to be solved; first, given the location of a region reference point, there is no efficient way, using the RM, to find all pixels that belong to this region. The only way is to check every pixel in the RM if they have the same reference point. Secondly, $q(i)$ is not identified as a local minimum until all pixels belonging to intensity level $i+\Delta+1$ are placed in the RM and $q(i+1)$ has been calculated. This means that, even if the complete RM is scanned to find all pixels that belong to the MSER the result would be wrong. It would include pixels from intensity levels $i+1$ to $i+\Delta+1$ and, even worse, other regions that have merged with this region due to the newly placed pixels.

We propose to solve both difficulties with an extension to the Union-find algorithm that we call linked regions. The idea with linked regions is that every placed pixel in the RM should point to the next pixel in the same region, and in this way form a chain that passes through all pixels in a region once. The link is circular with no beginning and no end.

To create the link an additional field is added for all positions in the RM. This field will hold a pointer to the next pixel that is part of this region, the initial value is the pixel position. When two regions merge with the Union-find algorithm the link fields of the two reference points are swapped. This will create a linked list, where the link field of a region reference point always points to the last added pixel of this region which in turn points to the next last pixel to be added and so on. An example of how a linked region is created is shown in Fig. 3.

With linked regions the problem of efficiently presenting the correct output when a MSER is found becomes trivial. The location of the region reference point is known along with the size of the region at both intensity i and $i+\Delta+1$, since this is stored in the $|Q|$-memory. Starting at the reference point the $|Q(i+\Delta+1)| - |Q(i)|$ first pixels in the linked list are not part of the output, since they were added to the region after intensity level i. After those pixels are bypassed the rest of the linked list is part of the output. An example is shown in Fig. 4.

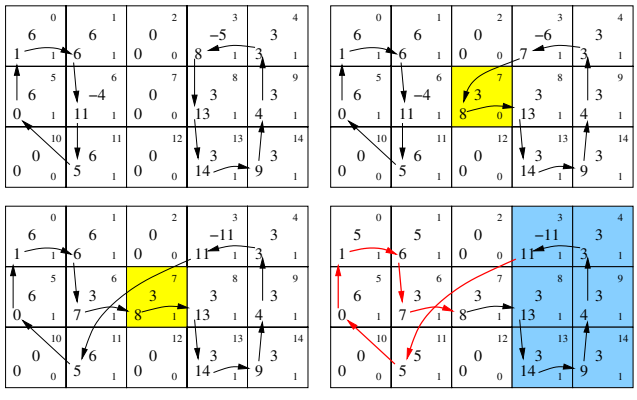

Fig. 4. Upper left image shows an RM at intensity i. Upper right and lower left image shows an new pixel being added to the RM at intensity $i+\Delta+1$ which causes $q(i)$ for the region at position 3 to be a local minimum. Lower right image shows how the RM at $i+\Delta+1$ is used to extract the original region, shown in blue, from intensity i. The first links are bypassed, shown in red, and only the last $|Q(i)|$ pixels are part of the region.

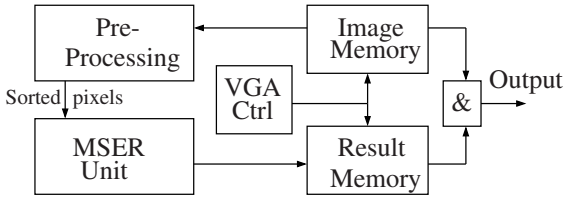

Fig. 5. Architectural overview of FPGA demonstrator.

III. Hardware Demonstrator

To demonstrate and verify the MSER algorithm, it was implemented on an FPGA board. The architecture, shown in Fig. 5, consists of the preprocessing block, the main MSER unit, an image memory, a result memory, and a VGA controller. Minimum size, maximum size, and Δ can be set to a number of different values, with switches on the FPGA board. The result memory is a binary memory that stores if a pixel is part of a MSER or not. Typical output images are shown in Fig. 6, where the MSERs are shown as red regions in the original image.

Some simplifications were used to reduce the implementation time of the demonstrator. No means of image capturing is implemented, the image that is used is downloaded in a bit-file to the FPGA. In a real application a video interface, such as the one in [8] could be added to capture images in real-time. Furthermore, to avoid boundary problems, e.g. regions that wrap around the image, all boundary pixels of the input image are set to 255 and the MSER algorithm is only executed for $i \in [0..254]$.

The demonstrator was implemented on an AMIRIX board with a Xilinx Virtex-II Pro FPGA, XC2VP100, and the most important values of the implementation are presented in Table I. To present more general results on the execution time for the algorithm is hard since it is highly dependent on the number of regions in the input image. However, the preprocessing parts of the algorithm can be measured exactly and an estimate on the MSER execution time is

$$T_{exe} = T_{pp} + T_{MSER} \approx 3N + 7N \approx 10N \ clockcycles \quad (1)$$

where N is the resolution, i.e. total image pixel count. If the resolution is fixed to 320×240 pixels, a maximum frame rate of 54 fps is achieved in the demonstrator.

The memory requirement, including both image and result memories, can be expressed as

$$\begin{aligned} M &= (im + res + im_{sort} + region_{map} + region_{link}) \\ &= (8 + 1 + \log_2(N) + (\log_2(N) + 2) + \log_2(N))N \\ &= (11 + 3\log_2(N))N \qquad bits. \end{aligned} \quad (2)$$

TABLE I
Characteristics of the MSER demonstrator

Resolution	320×240	**LUTs**	$9800(11\%)$
Delta	$1 - 4$	**Memory**	$4.6Mbit(63\%)$
Max size	$128 - 16384$	**Exe. time**	$\sim 19ms$
Min size	$64 - 8192$	**Speed**	$42MHz$

Fig. 6. Typical output images from demonstrator. To the left is the original image. The result of different max- and min-size settings are shown in the other images, red indicates MSERs. Detected MSER sizes are (middle) 64-200 and (right) 800-1500 pixels. (The MSERs are only visible in color.)

Using Equation 1 and 2 the upper limit on the image resolution, for this particularly FPGA, can be calculated as

$$\begin{aligned} N_{max} &= \min[N_{mem}, N_{speed}] = \min[350^2, 390^2] = \\ &= 350 \times 350 \ pixels, \end{aligned}$$

where N_{mem} is the upper limit due to the limited amount of on-chip memory on the FPGA and N_{speed} is the upper limit if real-time performance is expected. N_{mem} can be increased if off-chip memories are used for non-timing critical memories such as the image, result, and sort memories.

IV. Conclusion

In this paper it is shown how the MSER detector can be implemented for high speed performance with memory efficient hardware. In order to reach high performance an extension to the Union-find algorithm has been proposed. With the extension, all pixels part of a region are linked and therefore fast to find and extract. To verify the results an FPGA demonstrator has been implemented that can perform 25 fps MSER detection on images up to 350×350 pixels or 54 fps detection on image streams with 320×240 pixels resolution without using any off-chip memory.

References

[1] J. Matas, O. Chum, M. Urban, and T. Pajdla, "Robust Wide Baseline Stereo from Maximally Stable Extremal Regions." in *In Proceedings of the British Machine Vision Conference (BMVC)*, London, UK, sep 2002, pp. 384–393.

[2] F. Fraundorfer and H. Bischof, "A Novel Performance Evaluation Method of local detectors on non-planar scenes. ," in *In Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, San Diego, CA, USA, June 20-25 2005, pp. 553–560.

[3] D. Martinec and T. Pajdla, "Consistent Multi-View Reconstruction from Epipolar Geometries with Outliers." in *In Proc. of SCIA 2003*, Göteborg, Sweden, June 2003, pp. 493–500.

[4] J. Sivic and A. Zisserman, "Video Google: A Text Retrieval Approach to Object Matching in Videos," in *In Proc. of 9th IEEE International Conference on Computer Vision (ICCV 2003)*, Nice,France, Oct. 13-16 2003.

[5] M. Donoser and H. Bischof, "Efficient maximally stable extremal region (mser) tracking," in *In Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, New York, USA, June 17-22 2006, pp. 553–560.

[6] F. Kristensen, H. Hedberg, H. Jiang, P. Nilsson, and V. Öwall, "Hardware Aspects of a Real-Time Surveillance System," in *IEEE International Workshop on Visual Surveillance at ECCV 2006*, Graz, Austria, May 7-13 2006.

[7] R. Sedgewick, *Algorithms*, 2nd ed. Addison-Wesley, , 1988.

[8] D. K. Masrani and W. J. MacLean, "A Real-Time Large Disparity Range Stereo-System using FPGAs," in *IEEE International Conference on Computer Vision Systems, ICVS '06*, New York, USA, Jan. 4-7 2006.