Implementation and analysis of an optimized rainfalling watershed algorithm

Patrick De Smet and Rui Luís V. P. M. Pires

Dep. for Telecommunications and Information Processing
University of Ghent (TELIN-TW07V)
Sint–Pietersnieuwstraat 41, B–9000 Gent, Belgium

ABSTRACT

In this paper we discuss a new implementation of a floating point based rainfalling watershed algorithm. First, we analyse and compare our proposed algorithm and its implementation with two implementations based on the well-known discrete Vincent-Soille flooding watershed algorithms. Next, we show that by carefully designing and optimizing our algorithm a memory (bandwidth) efficient and high speed implementation can be realised. We report on timing and memory usage results for different compiler settings, computer systems and algorithmic parameters. Our optimized implementation turns out to be significantly faster than the two Vincent-Soille based implementations with which we compare. Finally, we include some segmentation results to illustrate that visually acceptable and almost identical segmentation results can always be obtained for all algorithms being compared. And, we also explain how, in combination with other pre- or post-processing techniques, the problem of oversegmentation (a typical problem of all raw watershed algorithms) can be (partially) overcome. All these properties make that our proposed implementation is an excellent candidate for use in various practical applications where high speed performance and/or efficient memory usage is needed.

Keywords: watershed, segmentation, optimized implementation, high-speed, memory efficient

1. INTRODUCTION

In image processing and analysis, image segmentation is still one of the crucial issues. The watershed transformation has proven to be a powerful basic segmentation tool that can be attributed properties of both edge detection and region growing techniques. Originally, watershed algorithms found their application in the field of topography (see Ref. 1), but since then they have been studied for several other important image segmentation tasks and applications ranging from image coding to object recognition and tracking. In this paper we will summarize and compare speed, memory and segmentation behaviour and results for two different (types of) watershed algorithms.

2. THEORY

2.1. Vincent and Soille Based Algorithms

In Ref. 1 and Ref. 2 fast and flexible algorithms for computing watersheds for digital greyscale images are discussed. These well-known algorithms are based on an immersion process analogy, in which the flooding of water on an "activity" image, considered to be a *discrete* topographic relief, is efficiently simulated using a queue of pixels. This activity image is supposed to take high values in the neighbourhood of edges and low values for interior pixels. So, this activity image serves as a fuzzy edge indicator. The actual activities can be calculated using e.g. the squared amplitude of the gradient, or the Teager energy (see Ref. 3). The number of discrete levels retained in the quantized activity image should be kept small so the algorithm can be calculated easily, i.e. fast, but should not be chosen too small in order to obtain acceptable segmentation results. To obtain the desired segmentation, the catchment basins for the water have to be delineated on the topographic surface.

Further author information: (Send correspondence to P.D.S.)
P.D.S.: E-mail: pds@telin.rug.ac.be,
R.P.: E-mail: rp@telin.rug.ac.be;
Tel.: +32 9 2643416, Fax: +32 9 2644295
WWW-site: `http://telin.rug.ac.be/ipi/watershed/`

www.manaraa.com

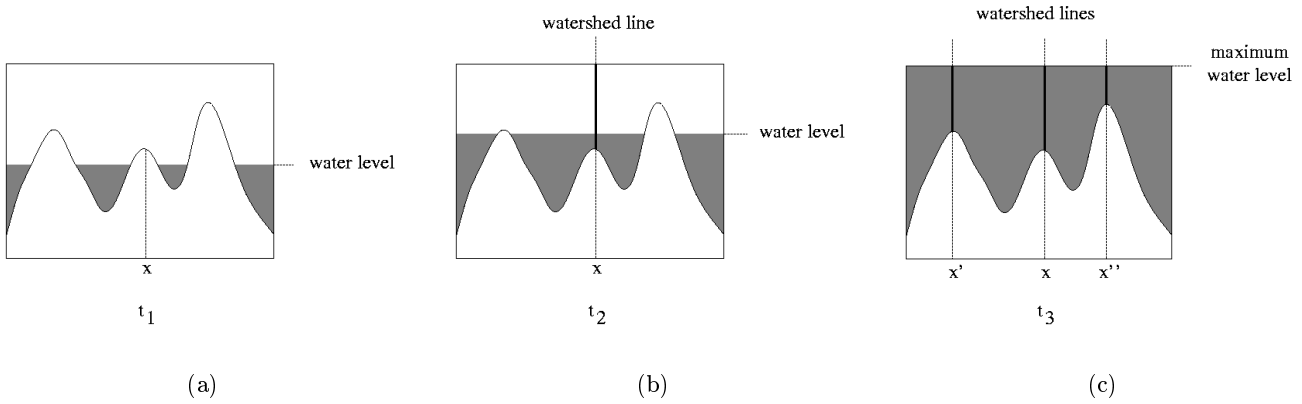


Figure 1. Flooding based watersheds; as time progresses more dams are constructed until finally all four segments have been delineated.

Figure 1 illustrates how the flooding process proceeds as time progresses; where the water coming from different catchment basins would meet, a "dam" is constructed on the topographic rims. These dams will delineate the segment borders and are called the watersheds or watershed lines.

Basically, the algorithm consists of two steps; a sorting step and a flooding step. The sorting step first computes the frequency distribution of each grey level in the activity image. The cumulative frequency is then computed, so each pixel(pointer) can be assigned to a unique cell in a sorted array; see Ref. 1 and Ref. 2. By doing this, pixels of equal activity "altitude" can then be examined sequentially. In the flooding step the (initial) catchment basins are recursively grown by using a FIFO-queue. The queue based flooding is indeed quite fast, but remains computationally complex; e.g. a *second* scanning of the pixels with a certain grey value, i.e. a topograhic level, is necessary to detect if any new catchment basins can be discovered.

In this paper we use two adapted implementations of the algorithm from Ref. 2 available for the Khoros image processing environment; see Ref. 4. The first implementation is (almost) identical to the one from Ref. 4, and uses dynamic memory allocations for managing the (data in the) FIFO-queue, i.e. the queue shrinks and grows if more elements are resp. deleted or added. The second implementation uses a large enough statically allocated array for storing the queue data. By large enough we mean that the maximum queue size needed will be determined and statically allocated each time before the real watershed routine is run.

An important remark is also that both these implementations do *not* calculate any geodesic distances which would improve their mathematical accuracy; see Ref. 1.

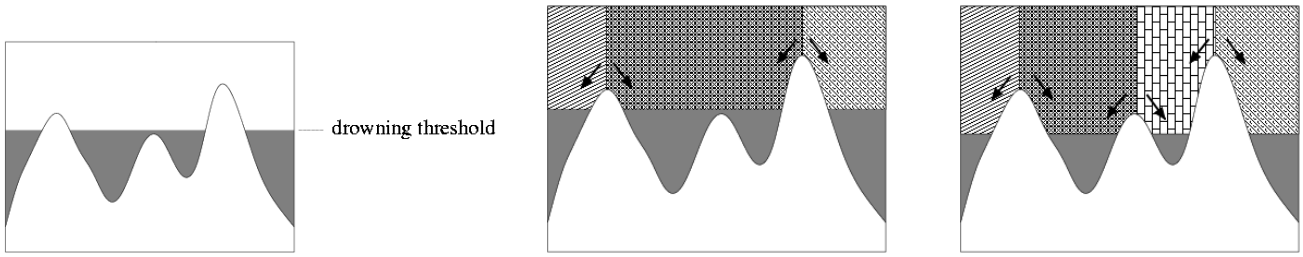
2.2. A General Rainfalling Algorithm

The rainfalling algorithm in our proposed implementation uses similar concepts as the discrete algorithm, but also has some significant differences. Most importantly, we use the concept of a *floating point* activity image as the input of the watershed algorithm.

The desired segmentation can again be obtained using two steps. First, some of the weakest edges (e.g. due to noise) can be removed by "drowning" the image. This drowning step will create a number of "lakes" grouping all the pixels that lie below a certain threshold. The threshold can also be interpreted as an "(under)ground water level". This is useful to reduce the influence of noise, and reduces the oversegmentation; see also section 3.1. Second, for each pixel we determine in which direction a raindrop would flow if it would fall on the topographic activity surface. This steepest descent neighbour and the pixel under consideration are then merged, finally enabling the localization of the remaining edges and segments, i.e. the areas ("slopes" and "lakes") surrounded by the topographic surface rims.

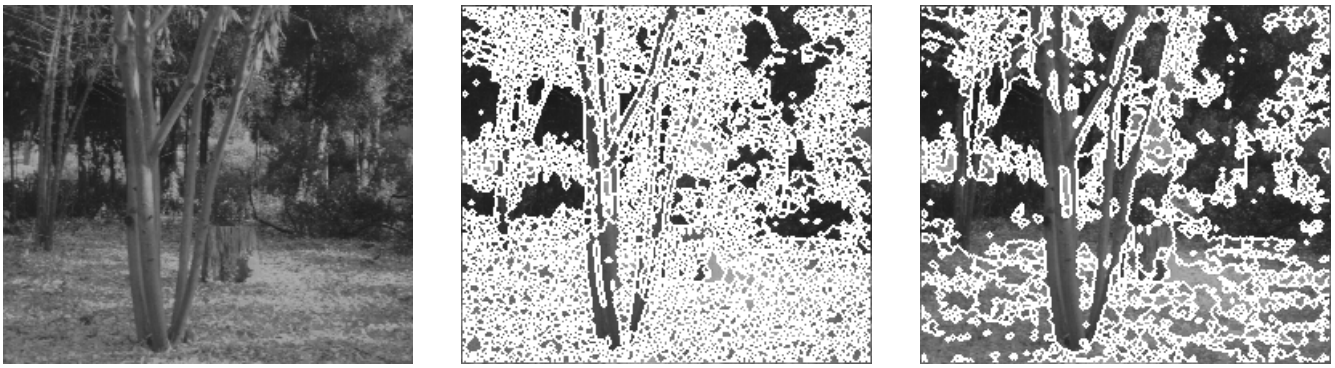
Figure 2 illustrates how the rainfalling (on the rims that have not been drowned) segments an image.

www.manaraa.com



(a) (b) (c)

Figure 2. Rainfalling based watersheds; (a) illustrates the drowning threshold (DT); (b) resp. (c) illustrate the steepest descent rainfalling principle and the segmentation results for different choices of the drowning threshold.



(a) (b) (c)

Figure 3. Segmentation results for the image TREES (a) original image; (b) rainfalling segmentation results for DT=1/255, 4106 segments; (c) rainfalling segmentation results for DT=0.015, 1696 segments. Similar results can be obtained with the flooding based approach (if an appropriate quantization of the activity image is performed first).

In this paper we do not formulate all the algorithmic details and optimizations, nor the full description of all the datastructures which we used to obtain our optimized implementation; these issues are discussed in Ref. 5 (first implementation without memory optimizations) and in Ref. 6 (revised implementation examined in this paper).

Another issue when discussing both the discrete and the floating point based implementations, is the treatment of the so-called "plateau" pixels; this is also discussed in Ref. 6.

3. PROPERTIES AND RESULTS

3.1. Segmentation Properties and Results

A typical problem of the watershed algorithms as discussed above, is the resulting oversegmentation of the image. As is illustrated in figure 3, this is especially the case for images with a lot of texture.

However, for a range of different parameter settings, both (types of) algorithms demonstrate very similar behaviour. Also, in this paper our aim is not to obtain the best segmentation results possible, but rather to illustrate the performance of the basic algorithms using activity input images which result in visually acceptable segmentation results. Hence, to solve the oversegmentation problem for the discrete Vincent-Soille based (DVSB) implementations

www.manaraa.com

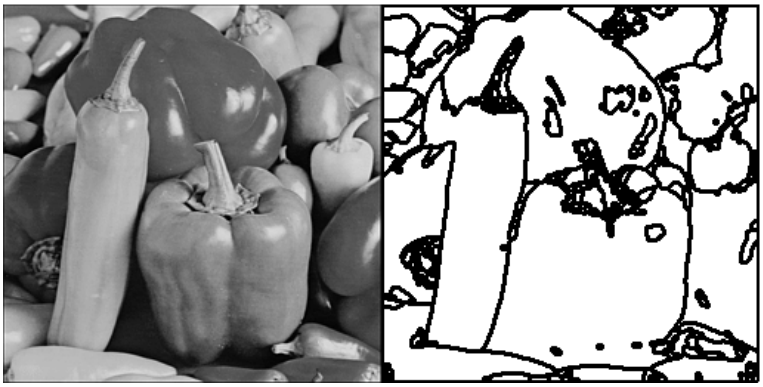


Figure 4. The image PEPPERS and the segmentation results (256 × 256; parameters see text) obtained using our proposed OFPBR implementation (583 remaining segments). The results obtained with the DVSB implementations (518 remaining segments) are visually identical.

we introduce the following simple pre-processing procedure. First, the squared amplitude of the gradient of the normalized image (floating point range 0 to 1; by scaling with 1/255) is calculated. This gradient based activity image is then clipped with a clipping threshold CT, i.e. all pixels with a value bigger than CT are set to the value CT, and the remaining range is uniformly quantized into 256 levels. The segmentation results for both the dynamically (DVSB1) and the statically (DVSB2) allocated FIFO-queue implementations are obviously identical. Their differences in terms of memory and computational efficiency are discussed below.

For the optimized floating point based rainfalling implementation (OFPBR), we have already discussed the use of a drowning threshold DT. The input image is always the non-clipped floating point activity image. In Ref. 7, 8 and 9 we discussed and illustrated an additional pre-processing strategy for reducing the oversegmentation and extending the OFPBR segmentation scheme to colour images. The use of floating point information is quite important in this improvement method, and it also avoids the problem of having to quantize the activity image which would obviously lead to a loss of information and accuracy. Quantization of the activity image would also require additional computation time.

Figure 4 illustrates the raw watershed segmentation results for the image PEPPERS. The OFPBR-routine results were obtained by setting DT=1/255. No clipping was used in the DVSB case (CT=maximum squared amplitude of the gradient image).

Some more segmentation results (e.g. for some of the 512 × 512 images mentioned in tables 1 and 2 below) and the OFPBR-source code are available on the WWW; see Ref. 10.

One of the interesting additional properties of the datastructures we used to implement our optimized watershed, is that all the pixels of a segment are available as a singly linked list, and each pixel is assigned a unique label. Hence, further post-watershed processing is straightforward and efficient (e.g. determining the size of each segment).

The DVSB implementations only return a segment label image, and do not label each pixel uniquely, i.e. the edge pixels are given the label "WSHED". Hence, additional processing would be needed to determine to which segment these pixels would belong (see also Ref. 1).

Some other problems related to the DVSB algorithms are discussed in Ref. 6 and Ref. 11.

3.2. Memory Usage and Efficiency

As described in Ref. 1 and Ref. 2 the minimal memory requirements of the discrete algorithm are: an image of pointers (to grey value sorted pixels), the FIFO-queue, and a segment label image (output). The maximum size of the FIFO-queue is dependent on the image under consideration. Hence, the amount of memory that will be needed for a certain image needs to be either pre-determined and allocated before the actual watershed algorithm is initiated, or the maximal amount of memory needs to be calculated based upon the image dimensions, or the queue has to be given dynamic (re-)allocation possibilities.

www.manaraa.com

Table 1. Averaged DVSB-timing results on the Linux-PC-portable

PEPPERS DVSB1	NoS	timing (s)		CT
		g++	g++ -O3	
128x128	299	0.15	0.12	none
256x256	518	0.61	0.50	none
512x512	1401	2.59	2.08	0.20
512x512	2070	2.62	2.12	0.15
512x512	3430	2.71	2.18	0.10
PEPPERS DVSB2	NoS	timing (s)		CT
		g++	g++ -O3	
128x128	299	0.11	0.09	none
256x256	518	0.43	0.31	none
512x512	1401	1.91	1.32	0.20
512x512	2070	1.94	1.33	0.15
512x512	3430	1.97	1.35	0.10

The OFPBR implementation only needs a segment label image and an image for storing pointers (linking pixels into segments). Our implementation also uses a small amount of additional memory for storing temporary, i.e. cached, data (see Ref. 5 and Ref. 6), but this is never influenced by the content of the image being considered.

The actual amount of memory can obviously also be dependent on the architecture of the system on which the code is compiled or on a set of implementation related choices (e.g. are C code "longs" or "shorts" used for the segment labeling of the pixels).

A disadvantage of the OFPBR algorithm is that it requires a floating point activity input image; the DVSB watershed routines use a 256 byte valued input image. However, if no further, or only limited other pre-processing is required (e.g. filtering with spatially small kernels), the activity image data could be calculated on the fly (this is not the case in the current OFPBR implementation). This would be possible since the OFPBR algorithm only runs through the image once in a single video scanning order; see Ref. 5 and Ref. 6. This also offers implicit (e.g. caches) or explicit possibilities for (hardware related) memory bandwidth optimizations as e.g. caching and/or data-pre-fetching strategies, etc. Also, the segment labels could be copied into the activity image memory as soon as a certain image line has been processed.

The DVSB algorithms have a spatially more random access like behaviour; they process pixels of the same grey value sequentially. In earlier papers (see Ref. 5 and Ref. 12) we investigated the DVSB implementations with the source code as available on the internet (see Ref. 4). In this paper we report results for optimized and improved implementations; first, we optimized the histogram based pixel sorting code. This allowed e.g. a reduction of the histogram computation time from approximately 120 ms to 30 ms (for a 512×512 image, on the Linux architecture mentioned below). In a second step we removed a serious inefficiency from the original implementation which needlessly added the same pixel several times to the queue. By correcting this latter inefficiency in the code, a very significant speed-up and reduction in the amount of used FIFO-memory could be obtained. E.g. for the image PEPPERS and with CT set to 0.15, the maximum amount of elements in the queue decreased from 188181 to 39667.

As will be shown below, choosing between the two DVSB implementations based on their memory allocation and usage strategies can also have a significant impact on the speed of the segmentation process.

For 3D-segmentation applications the DVSB random pixel access behaviour is even more undesirable; see Ref. 13. Hence, the progressive single scan nature of the OFPBR watershed offers interesting possibilities for further research.

www.manaraa.com

Table 2. Averaged OFPBR-timing results on the Linux-PC-portable

PEPPERS OFPBR	NoS	timing (s)		DT
		g++	g++ -O3	
128x128	335	0.08	0.06	1/255
256x256	581	0.30	0.22	1/255
512x512	1408	1.22	0.93	0.001
512x512	2319	1.25	0.96	0.0007
512x512	3885	1.27	0.99	0.0005

Table 3. Averaged DVSB-timing results on the Sun Ultra Sparc II workstation

PEPPERS DVSB1	NoS	timing (s)		CT
		g++	g++ -O3	
512x512	1401	1.000	0.600	0.20
512x512	2070	1.020	0.620	0.15
512x512	3430	1.040	0.640	0.10
PEPPERS DVSB2	NoS	timing (s)		CT
		g++	g++ -O3	
512x512	1401	0.670	0.270	0.20
512x512	2070	0.680	0.280	0.15
512x512	3430	0.680	0.290	0.10

3.3. Timing Results

Tables 1 and 2 indicate the speed (in seconds) of the different algorithms.

These results were obtained when the source code was compiled and run on a Linux-portable running RedHat Linux 6.0, kernel 2.2.5-15 with a 200 Mhz MMX processor, 96 Mbyte RAM, 256 Kbyte cache and the g++/egcs-2.91.66 compiler.

Tables 3 and 4 summarize similar results, but after compilation on a Sun Ultra 2 UPA/SBus (2 × (UltraSPARC-II 296MHz, 2Mb cache), 256 Mb RAM) workstation with SunOS version 5.7 and g++ version 2.8.1.

All the tables also report on the timing results for each implementation with both standard (g++) and optimized (g++ -O3) compilation for various images sizes.

To make a meaningful comparison of the timing results of the algorithms, we need to compare segmentation results which are both visually acceptable and similar. By experimenting with different thresholds CT and DT, ranges of visually acceptable results can be determined. Visual similarity of the segmentation results (for DVSB vs. OFPBR comparison) corresponds to setting the parameters such that the number of retained segments (NoS) is roughly the same.

Also note that the DVSB results reported in tables 1 and 3 do not include the processing time needed to quantize and clip the input activity image, nor the time needed to relabel the "WSHED" pixels (see section 3.1). A separate and fairly optimized clipping and quantization routine typically takes 150 ms (-O3 compilation) to 270 ms for a 512 × 512 image on the Linux-system mentioned earlier.

What can be seen in the tables is that, as could be expected, the statically allocated discrete algorithm (DVSB2) is significantly faster than the dynamic one (DVSB1). Additionally, tables 1 and 3 illustrate that the choice of good optimizing compiler(setting)s is quite important for the DVSB flooding algorithms.

The OFPBR implementation turns out to be "optimized by design" (little compiler dependency/influence), and is significantly faster than the DVSB implementations. Possibly the DVSB implementations could be optimized

www.manaraa.com

Table 4. Averaged OFPBR-timing results on the Sun Ultra Sparc II workstation

PEPPERS OFPBR	NoS	timing (s)		DT
		g++	g++ -O3	
512x512	1408	0.30	0.18	0.0010
512x512	2319	0.31	0.19	0.0007
512x512	3885	0.32	0.20	0.0005

further, but for region growing algorithms, this task is far from trivial compared to the straightforward OFPBR optimizations (see Ref. 5 and Ref. 6).

In Ref. 1, 14 and 15 it was demonstrated that Vincent-Soille based algorithms are still (among) the fastest algorithms currently available. Hence, the speed-up we have obtained here can be considered to be quite significant.

4. CONCLUSION

In this paper we have shown that the very intuitive rainfalling watershed principle can be implemented very efficiently. Our proposed OFPBR implementation also offers other (post-watershed processing related) advantages and does not require an intermediate input image quantization step which can result in loss of information and accuracy. In combination with other pre- or post-processing techniques the problem of oversegmentation —a typical problem of all raw watershed algorithms— can be (partially) overcome. This makes our proposed implementation an excellent candidate for use in practical applications where high speed performance and/or efficient memory usage is needed. Further optimization of the current OFPBR implementation and extensions for 3D-applications are being investigated, and parallellization and further comparison with other algorithms are being considered.

ACKNOWLEDGMENTS

This work was financially suppported by the Flemish Institute for the Advancement of Scientific and Technological Research in Industry (IWT) through the project IWT-ITA-II 980302; "Beeldverwerking voor geïntegreerde en immersieve visualisatie" (Image processing for integrated and immersive visualisation).

REFERENCES

1. L. Vincent, P. Soille, "Watersheds in Digital Spaces: An Efficient Algorithm based on Immersion Simulations", *IEEE Transactions on Pattern Analysis and Machine Intelligence* **13**(6), pp. 583—589, June 1991.
2. P. Soille, *Morphological Image Analysis: Principles and Applications*, Springer, 1999.
3. D. De Vleeschauwer, F. A. Cheikh, R. Hamila and M. Gabbouj, "Watershed Segmentation of an Image Enhanced by Teager Energy Driven Diffusion", in *6'th Int. Conf. on Image Proc. and its Applications; IPA '97*; pp. 254–258, Trinity College, Dublin, Ireland, July 1997.
4. Khoros mathematical morphology toolbox (mmach1.4); see: `http://www.khoral.com/` or `http://www.dca.fee.unicamp.br/projects/khoros/`
5. P. De Smet and D. De Vleeschauwer, "Performance and scalability of a highly optimized rainfalling watershed algorithm", in *Proc. of the 1998 International Conference on Imaging Science, Systems and Technology; CISST '98*, pp. 266–273, Las Vegas, NV, USA, July 1998.
6. P. De Smet, "Implementing and optimizing a floating point based rainfalling watershed algorithm", internal report TELIN/IPI (TW07V), University of Ghent; available on the WWW-address mentioned below.
7. P. De Smet, R. Pires, D. De Vleeschauwer, "Activity Driven Non-linear Diffusion for Color Image Segmentation" in *Noblesse Workshop on Non-linear Model Based Image Analysis; Proceedings of NMBIA*, July 1998, Glasgow, S. Marshall, N. Harvey and D. Shah (Eds.), pp. 183–187, page 2 in colour annex, 1998, Springer.
8. P. De Smet, R. Pires, D. De Vleeschauwer, "The Activity Image in Image Enhancement and Segmentation", in *Proc. of the IEEE Benelux Signal Processing Symposium*, pp. 79–82, March 1998, Leuven, Belgium.

www.manaraa.com

9. P. De Smet, R. Pires, D. De Vleeschauwer and I. Bruyland, "Activity Driven Non-linear Diffusion for Color Image Watershed Segmentation", in *SPIE Journal of Electronic Imaging* **8**(3), special section on Non-linear and Model-based Image Analysis, pp. 270–278, July 1999.
10. Segmentation results obtained using (non-linear diffusion based activity image cleaning and) the OFPBR watershed, see: `http://telin.rug.ac.be/ipi/watershed/`
11. B. Dobrin, T. Viero, M. Gabbouj, "Fast watershed algorithms: analysis and extensions", in *Proc. IS&T/SPIE Symposium on Electronic Imaging Science & Technology, Nonlinear Image Processing V*, pp. 209–220, San Jose Convention Center, San Jose, CA, USA, February 6–10, 1994.
12. P. De Smet, P. Pires, "On the implementation of a highly optimized rainfalling watershed algorithm", in *Proceedings of the VLBV'99 (workshop on Very Low Bitrate Video Coding)*, pp. 101-104, Kyoto, Japan, October 29-30, 1999.
13. C. Cotsaces and I. Pitas, "Computing the Watersheds of Large Three-dimensional Images using Limited Random Access Memory", in *Mathematical Morphology and its Applications to Image and Signal Processing*, H. J. A. M. Heijmans, J. B. T. M. Roerdink (Eds.), pp. 239–246, Kluwer Academic Publishers, 1998.
14. D. Hagyard, M. Razaz and P. Atkin, "Analysis of Watershed Algorithms for Greyscale Images", in *Proc. IEEE Int. Conf. on Image Processing; ICIP '96*, pp. 41–44, 1996.
15. M. C. D'Ornellas and R. Van Den Boomgaard, "Generic Algorithms for Morphological Image Operations - A Case Study Using Watersheds", in *Proc. International Symposium on Mathematical Morphology*, pp. 323-380, Amsterdam, The Netherlands, 1998.

www.manaraa.com