Efficient Computation of Morphological Greyscale Reconstruction

Pavel Karas

Centre for Biomedical Image Analysis
Faculty of Informatics
Masaryk University Brno

Abstract

Morphological reconstruction is an important image operator from mathematical morphology. It is very often used for filtering, segmentation, and feature extraction. However, its computation can be very time-consuming for some input data. In this paper we review several efficient algorithms to compute the reconstruction, and compare their performance on real 3D images of large sizes. Furthermore, we propose a GPU implementation which performs up to 15 × faster than the CPU methods. To our best knowledge, this is the first GPU implementation of the morphological reconstruction, described in literature.

Digital Object Identifier 10.4230/OASIcs.MEMICS.2010.54

1 Introduction

Mathematical morphology is a theory for analysis and processing spatial structures in images. Morphological methods can be used for image pre-processing and for image analysis [9, 14, 16].

Morphological reconstruction is an advanced approach to image analysis. It can be used for various applications, such as filtering, segmentation, and feature extraction [18], image and video compression [13], remote sensing [15], and biomedical image analysis [12]. In image segmentation, the reconstruction is often used for pre-processing, to avoid over-segmentation [5, 8].

To compute the morphological reconstruction, several sequential and FIFO-based algorithms were proposed [11, 18]. To our best knowledge, no GPU implementation of the morphological reconstruction has been described in literature. Eidheim et al. [6] proposed a GPU implementation of basic morphological operations, such as dilation and erosion. These algorithms are easy to implement in parallel as described in [3]. Jivet et al. [10] implemented the morphological reconstruction on a dedicated FPGA hardware using the iterative computation of the geodesic dilation. In this paper, we adopt the sequential algorithm [18] with the reduced number of iterations and propose a parallel GPU-based implementation. We compare its performance with several algorithms executed on CPU.

1.1 Notations

In the following text, an n-dimensional image f is considered a mapping from a finite subset $D_f \subset \mathbb{Z}^n$ into a set I of image values. I is usually a finite discrete set of m levels $\{0, 1, \ldots, m-1\}$. The discrete grid $G \subset \mathbb{Z}^n \times \mathbb{Z}^n$ provides the neighbourhood relationship between pixels: p is a neighbour of q if and only if $(p, q) \in G$. Depending on a particular application, various grids can be used; in 2-D case, 4-, 6-, or 8-connectivity are the most common examples.

Sixth Doctoral Workshop on Math. and Eng. Methods in Computer Science (MEMICS'10)—Selected Papers.
Editors: L. Matyska, M. Kozubek, T. Vojnar, P. Zemčík, D. Antoš; pp. 54–61

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1.2 Definition of Morphological Reconstruction by Dilation

Before we define the reconstruction by dilation, we define the geodesic dilation first. We use the same definition as in [15]. Let f, g be two images fulfiling the following properties:

- $\mathcal{D}_f = \mathcal{D}_g$, i.e., both images share the same definition domain,
- $f(p) \leq g(p), \forall p \in \mathcal{D}_f$, i.e., f is smaller or equal to g in all pixels

We call f the *marker* image and g the *mask* image and define the *geodesic dilation* of size 1 as follows:

$$\delta_g^{(1)}(f) = \delta^{(1)}(f) \wedge g, \tag{1}$$

where $\delta^{(1)}$ denotes the elementary dilation (i.e., dilation with the smallest non-trivial structure element of the used connectivity) and the $\wedge$ operator denotes point-wise minimum. The geodesic dilation of size $n > 1$ is obtained by performing n successive geodesic dilations:

$$\delta_g^{(n)}(f) = \delta_g^{(1)}\left[\delta_g^{(n-1)}(f)\right]. \tag{2}$$

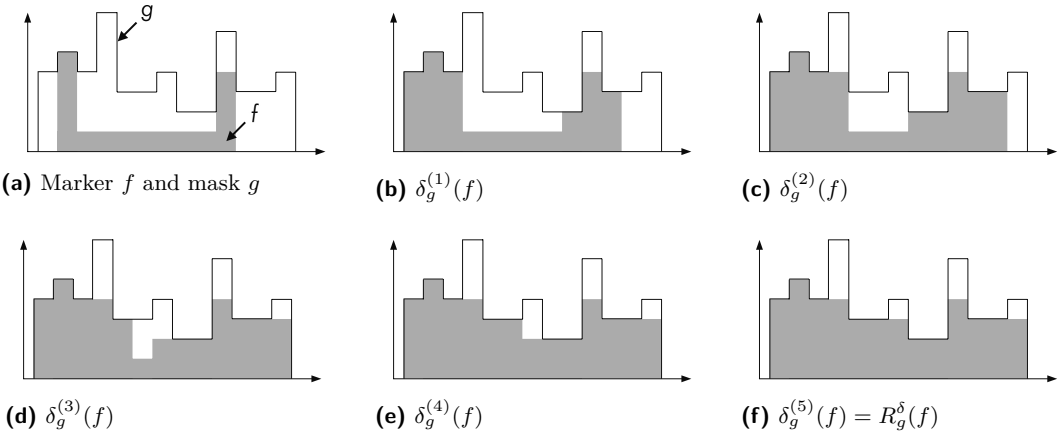
The *reconstruction by dilation* is defined as the geodesic dilation iterated until stability:

$$R_g^{\delta}(f) = \delta_g^{(i)}(f), \tag{3}$$

where i is such that $\delta_g^{(i+1)}(f) = \delta_g^{(i)}(f)$. The reconstruction by dilation of a 1-D signal is illustrated in Fig. 1.

In our application of biomedical image analysis, we process 3-D grayscale images. Therefore, we describe our implementations for the 3-D images with the 6-connectivity.

The paper is organized as follows: First, we review three algorithms for computing the morphological reconstruction, described in literature [18]. Second, we describe our GPU implementation. Finally, we analyze and compare the performance of all implementations on several 3D images from our field.



(a) Marker f and mask g **(b)** $\delta_g^{(1)}(f)$ **(c)** $\delta_g^{(2)}(f)$

(d) $\delta_g^{(3)}(f)$ **(e)** $\delta_g^{(4)}(f)$ **(f)** $\delta_g^{(5)}(f) = R_g^{\delta}(f)$

Figure 1 Reconstruction by dilation R^{δ} of a 1-D signal g from a marker signal f.

2 Methods

2.1 Existing Algorithms

Standard technique

The reconstruction by dilation can be computed directly from its definition (3). Even though the iterations can be performed efficiently using van Herk/Gil-Werman algorithm [7, 17], an enormous number of iterations is required to converge. Therefore, we do not consider this algorithm in our paper in the performance evaluation.

Sequential reconstruction (SR)

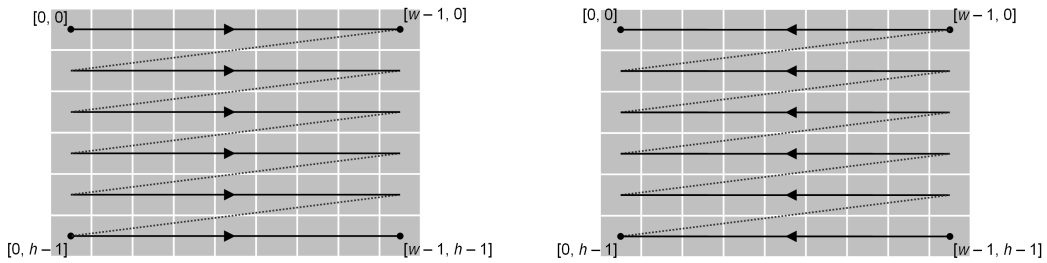
This algorithm was proposed to reduce the number of iterations [18]. The image is scanned in a predefined order and the information is propagated throughout the image. First, the image is scanned in the raster order—see Fig. 2a. Subsequently, it is scanned in the anti-raster order—Fig. 2b. The scans are repeated until convergence. The computation is performed "in-place" in the marker image.

Hybrid reconstruction algorithm (HRA)

HRA does not yield multiple iterations, thus, the computation time is further reduced [18]. It has two phases: a sequential and a FIFO phase. The former uses the previous sequential algorithm to execute a single iteration. During the anti-raster scan, pixels of regional maxima are put into a queue. In the latter phase, the pixels are read from the queue and their neighbourhood pixels are examined. If the information is propagated to the neighbourhood pixels, they are put into the queue. Once the queue is empty, the computation is complete.

2.2 GPU Implementation (*SR_GPU*)

Our GPU implementation is a modified version of the *SR* algorithm described in Section 2.1 and Fig. 3. It was written in the CUDA parallel programming model [2]. For the flowchart of the *SR_GPU* algorithm refer to Fig. 3. The raster and anti-raster scans are performed in each dimension, separately. Thus, each iteration requires 4 or 6 passes for a 2-D or 3-D image, respectively. The raster scans are executed by CUDA kernels called *x_Forward*, *y_Forward*, and *z_Forward*; the anti-raster scans are performed by kernels called *x_Backward*, *y_Backward*, and *z_Backward*. Since the Forward and Backward kernels are analogous, only the Forward variants will be described.



(a) Raster scan **(b)** Anti-raster scan

Figure 2 Scanning patterns in a 2-D image of size $w \times h$ pixels.

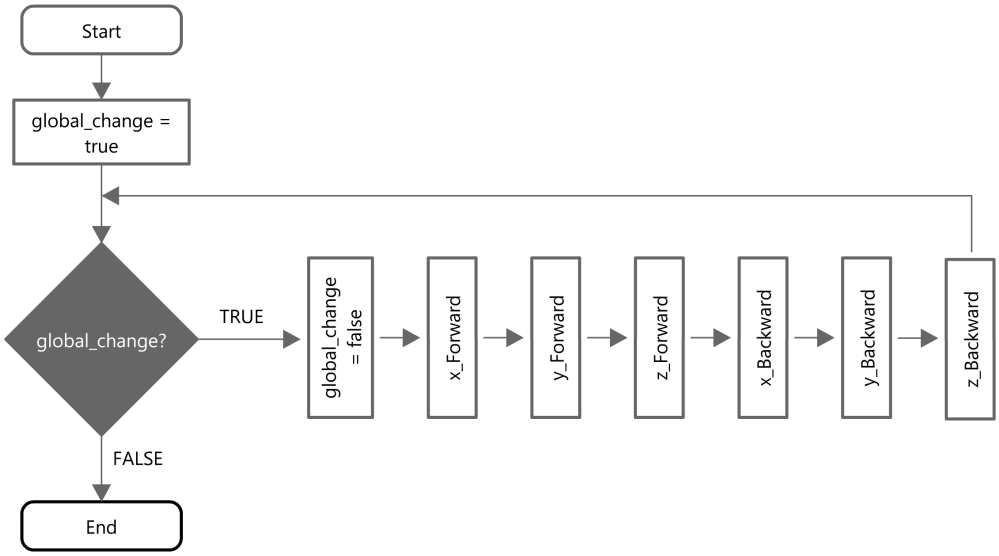


Figure 3 Flowchart of the *SR_GPU* algorithm for the morphological reconstruction. The functions, called *x_Forward*, *y_Forward*, *z_Forward*, *x_Backward*, *y_Backward*, and *z_Backward*, provide image scans. The *global_change* variable indicates changes in the marker image.

y_Forward

The y_Forward kernel is executed by $N_x N_z$ threads, where N_x, N_z are the sizes of the input images in x and z dimension, respectively. Except the input images *marker* and *mask* and the variable called *global_change* stored in the global memory, all the variables can be stored in registers. Since the threads are regularly distributed across the x and z dimensions of the images, the accesses to the global memory are naturally coalesced [2], achieving the maximum bandwidth.

z_Forward

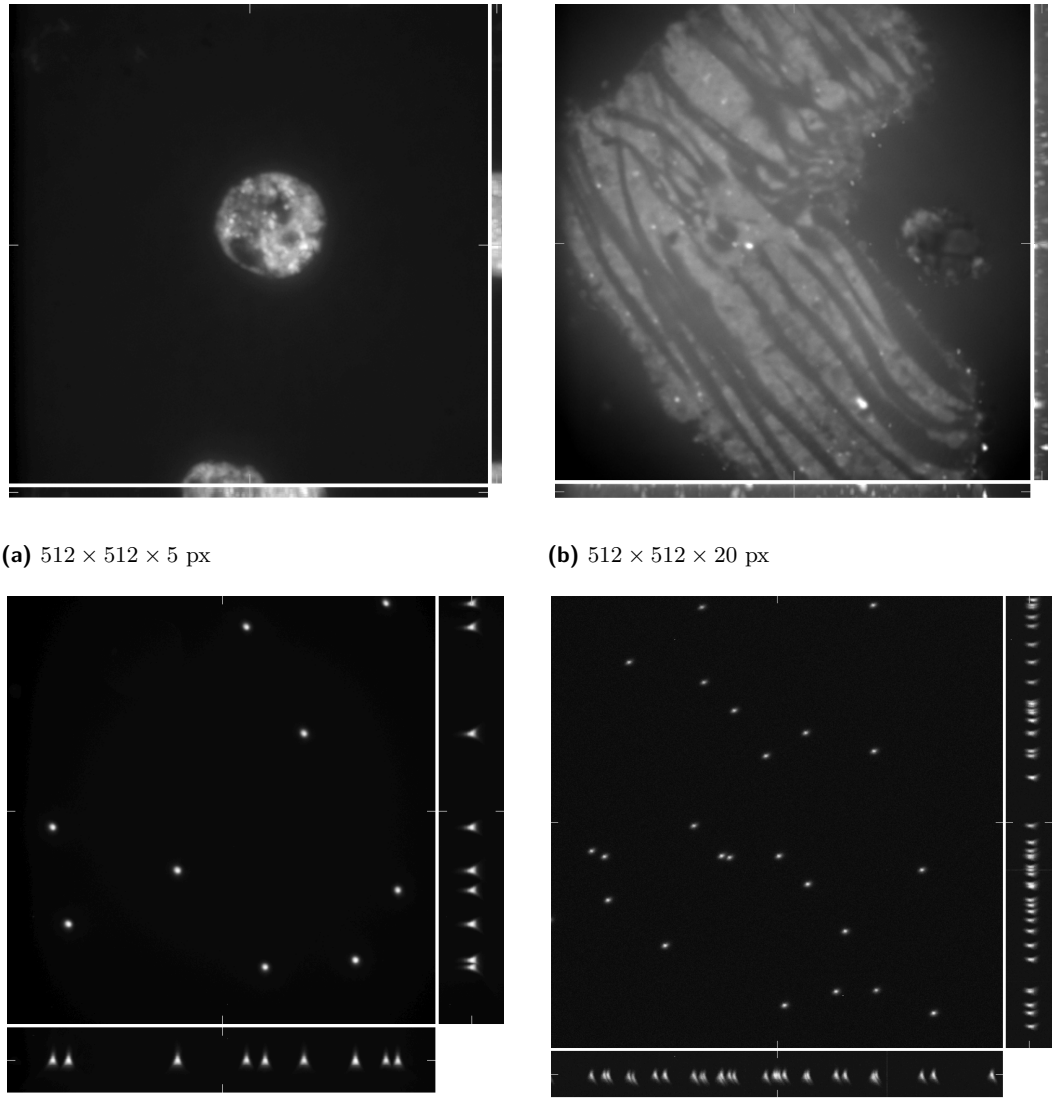
The z_Forward kernel is analogous to the y_Forward kernel.

x_Forward

The x_Forward kernel is executed by $N_y N_z$ threads. Since the threads are now distributed across the y and z dimensions of the images, we cannot use the same approach as above to achieve optimum access to the global memory. Therefore, we pre-load data and compute results in the shared memory [2]. Afterwards, the data is written back to the global memory. Only few threads of a block perform the computation itself, so this approach may seem to be less efficient. However, the GPU thread scheduler can switch between warps, thus overlapping the arithmetic operations and memory accesses and hiding the global memory latency.

Stopping criterion

The scans are repeated until convergence, much like in the classic *SR* algorithm. The test of convergence is performed by inspecting the *global_change* variable after each scan. Gath-



(a) 512 × 512 × 5 px

(b) 512 × 512 × 20 px

(c) 512 × 512 × 100 px

(d) 1300 × 1030 × 80 px

Figure 4 Input images.

ering information from all threads to one output variable generally requires the *reduction* kernel [1]. However, in this case, only a boolean-type information is needed, thus, the reduction can be avoided. Before each scan, *global_change* is set to *false*. Iff a thread performs the first change in the marker image, it assigns *global_change* to *true*. This approach also avoids write-before-read conflicts.

3 Results

The performance of the three algorithms, namely *SR*, *HRA*, and *SR_GPU*, was compared on four real 3-D images from confocal microscopy (Fig. 4). These images were taken as the input mask image g.

Two different approaches to create the marker image f were chosen. First, all but one of the pixels of the marker image are set to zero. The non-zero pixel was selected to be the one with the maximum value in the mask image and its value was set to the same value. Second, the values of pixels in the marker image were set to the values of those in the mask image, decreased by a constant h:

$$f(p) = \max\{g(p) - h, 0\}. \tag{4}$$

The morphological reconstruction with the marker image defined as above is often called the *HMAX* transform in literature [16].

The implementations were tested on a workstation with an Intel Core2 Quad Q6600 2.4 GHz CPU, 8 GB DDR2 RAM, and a GeForce GTX 470 GPU with 448 SPs and 1280 MB of GDDR5 memory.

3.1 Morphological Reconstruction With a Simple Marker Image

The results of the first experiment are summarized in Table 1. The computation times are in seconds, the data-transfer overhead for the GPU implementation is included.

It is obvious that results of both the *SR* and *SR_GPU* algorithm strongly depend on the number of iterations needed to complete the computation. The number of iterations depends strongly on both the image dimensions and the image content. The queue-based *HRA* algorithm does not yield such iterations and converges significantly faster. However, the GPU implementation is faster almost in all cases, due to higher performance and memory bandwidth of the graphics hardware. The image (d) is the only case where the *HRA* algorithm on CPU performs better, since the number of iterations is extremely high (661). In other cases, the GPU implementation achieves up to 15 × speedup over the *HRA* algorithm.

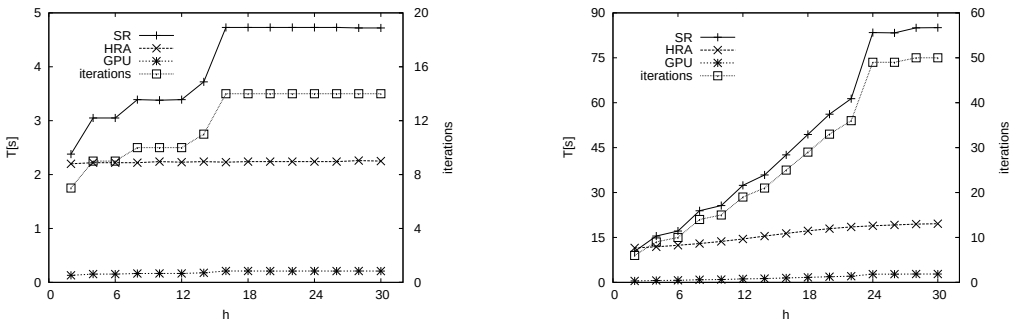
3.2 HMAX Transform

The results of the second experiment are summarized in Table 2. They are very similar to those in the previous experiment, however, the number of iterations is generally lower.

We also analysed the dependancy of both the computation time and the number of iterations on the h parameter. The results for two selected images are shown in Fig 5. As expected, the dependancy is strong for the *SR* and the *SR_GPU* algorithms, while there is almost no dependancy for the *HRA* algorithm. However, the computation time for the *SR_GPU* algorithm does not grow so fast with increasing the number of iterations, because with the higher computation time, the effect of the data-transfer overhead is reduced.

Table 1 Morphological reconstruction with a simple marker image. In the columns 2, 3, and 4, computation times in seconds are presented. The column 5 shows the number of iterations needed to complete the computation in *SR* and *SR_GPU*. In the last column, the speedup achieved by the *SR_GPU* algorithm is presented.

Image	*SR* [s]	*HRA* [s]	*SR_GPU* [s]	iterations	speedup
(a)	8.02	2.26	0.22	30	10.3
(b)	18.74	6.17	0.51	17	12.1
(c)	280.23	64.57	4.16	51	15.5
(d)	> 2 hours	98.97	135.13	661	0.7



(a) Image (b) **(b)** Image (c)

Figure 5 Computation time and number of iterations for the HMAX transform on two selected images.

4 Conclusion

In this paper we proposed a GPU implementation for the morphological reconstruction and compared its performance with two CPU algorithms. The results showed that graphics hardware offers good speedup and is able to perform significantly faster than the optimized CPU algorithm in most cases.

By optimizing our GPU implementation, further speedup could be achieved. The main issue is the high number of iterations for some input data. By implementing the optimized *HRA* algorithm, this could be avoided, but FIFO-based algorithms are not generally good candidates for GPU acceleration. In our future work, we will study possibilities of adopting the *HRA* algorithm for GPU.

The CPU implementations can be improved, too, for example, by utilizing multiple CPU cores. However, in the case of the faster queue approach this would require a challenging effort. The performance of the *SR* algorithm strongly depends on the number of cache misses and could be also improved by using cache-efficient matrix transpositions [4]. This is the subject of our future work.

Table 2 HMAX transform with the parameter $h = 10$. In the columns 2, 3, and 4, computation times in seconds are presented. The column 5 shows the number of iterations needed to complete the computation in *SR* and *SR_GPU*. In the last column, the speedup achieved by the *SR_GPU* algorithm is presented.

Image	*SR* [s]	*HRA* [s]	*SR_GPU* [s]	iterations	speedup
(a)	0.57	0.56	0.06	7	9.3
(b)	3.39	2.23	0.17	10	13.1
(c)	25.63	13.73	0.94	15	14.6
(d)	1414.25	71.73	40.37	204	1.8

Acknowledgments

This work has been supported by the Ministry of Education of the Czech Republic (Projects No. MSM-0021622419, No. LC535 and No. 2B06052).

References

1 CUDA™ SDK Code Samples 3.1. `http://developer.nvidia.com/object/cuda_sdk_samples.html`, Jun 2010.

2 NVIDIA GPU Computing Developer Home Page. `http://developer.nvidia.com/object/gpucomputing.html`, Jun 2010.

3 Thomas Bräunl, Stefan Feyrer, Wolfgang Rapf, and Michael Reinhardt. *Parallel Image Processing*. Springer, 2001.

4 S. Chatterjee and S. Sen. Cache-efficient matrix transposition. pages 195–205, 2000.

5 Xiaowei Chen, Xiaobo Zhou, and S.T.C. Wong. Automated segmentation, classification, and tracking of cancer cell nuclei in time-lapse microscopy. *IEEE Transactions on Biomedical Engineering*, 53(4):762–766, 2006.

6 O.C. Eidheim, J. Skjermo, and L. Aurdal. Real-time analysis of ultrasound images using GPU. *International Congress Series*, 1281:284–289, 2005. CARS 2005: Computer Assisted Radiology and Surgery.

7 J. Gil and M. Werman. Computing 2-D min, median, and max filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):504–507, may. 1993.

8 K. Haris, S. N. Efstratiadis, N. Maglaveras, and A. K. Katsaggelos. Hybrid image segmentation using watersheds and fast region merging. *IEEE Transactions on Image Processing*, 7(12):1684–1699, 1998.

9 Bernd Jähne. *Digital Image Processing*. Springer, 6th edition, 2005.

10 Ioan Jivet, Alin Brindusescu, and Ivan Bogdanov. Image contrast enhancement using morphological decomposition by reconstruction. *WSEAS Trans. Cir. and Sys.*, 7(8):822–831, 2008.

11 Kevin Robinson and Paul F. Whelan. Efficient morphological reconstruction: a downhill filter. *Pattern Recognition Letters*, 25(15):1759–1767, 2004.

12 Pekka Ruusuvuori, Tarmo Aijo, Sharif Chowdhury, Cecilia Garmendia-Torres, Jyrki Selinummi, Mirko Birbaumer, Aimee Dudley, Lucas Pelkmans, and Olli Yli-Harja. Evaluation of methods for detection of fluorescence labeled subcellular objects in microscope images. *BMC Bioinformatics*, 11(1):248, 2010.

13 P. Salembier, P. Brigger, J.R. Casas, and M. Pardas. Morphological operators for image and video compression. *IEEE Transactions on Image Processing*, 5(6):881–898, 1996.

14 Jean Serra. *Image Analysis and Mathematical Morphology*. Academic Press, Inc., Orlando, FL, USA, 1983.

15 P. Soille and M. Pesaresi. Advances in mathematical morphology applied to geoscience and remote sensing. *IEEE Transactions on Geoscience and Remote Sensing*, 40(9):2042–2055, 2002.

16 Pierre Soille. *Morphological Image Analysis: Principles and Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

17 Marcel van Herk. A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels. *Pattern Recognition Letters*, 13(7):517–521, 1992.

18 Luc Vincent. Morphological grayscale reconstruction in image analysis: Applications and efficient algorithms. *IEEE Transactions on Image Processing*, 2:176–201, 1993.