2021/11/21 v0.6 IPOL article class

Published in Image Processing On Line on 2022–04–08.
Submitted on 2017–07–18, accepted on 2021–03–04.
ISSN 2105–1232 © 2022 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2022.215

A Parallel, $\mathcal{O}$(n) Algorithm for an Unbiased, Thin Watershed

Théodore Chabardès[3], Petr Dokládal[1], Matthieu Faessel[2], Michel Bilodeau[1]

[1] MINES Paris - Universit PSL - Centre for Mathematical Morphology (CMM) - 77300 Fontainebleau, France (firstname.lastname@mines-paristech.fr)
[2] now with Coh@bit, IUT de Bordeaux, 33175 Gradignan Cedex, matthieu.faessel@gmail.com
[3] now with The Cross Product (TCP), 77300 Fontainebleau, theodore.chabardes@thecrossproduct.com

Communicated by Pascal Monasse *Demo edited by* Pascal Monasse

Abstract

The watershed transform is a powerful tool for morphological segmentation. Most common implementations of this method involve a strict hierarchy on gray tones in processing the pixels composing an image. This hierarchical dependency complexifies the efficient use of modern computational architectures. This paper introduces a new way of computing the watershed transform that alleviates the sequential nature of hierarchical queue propagation. It is shown that this method can directly relate to the hierarchical flooding. Simultaneous and disorderly growth can now be used to maximize performances on modern architectures. Higher speed is reached, bigger data volume can be processed. Experimental results show increased performances regarding execution speed and memory consumption.

Source Code

The reviewed source code and documentation for this algorithm are available from the web page of this article[1]. Compilation and usage instruction are included in the `README.txt` file of the archive.

Keywords: segmentation; watershed; local implementation; linear time

1 Introduction

The watershed transform is a standard tool for morphological segmentation. It is based on region growth and edge detection in most cases applied to a gradient of the original image. One definition reflects the water flooding a topographic relief. This definition creates thin watershed lines and is chosen for the most common algorithms. Considering the topographic surface as perforated in regional minima, it is immersed in water, so that the water rises from those holes and creates lakes.

[1] https://doi.org/10.5201/ipol.2022.215

Those different lakes, each originating from a different minimum, are not allowed to mix. Their intersections will form dams. The set of all dams is called the watershed lines and represents the desired contours.

Most algorithms replicate this principle by processing pixels following a global, strict hierarchical order implemented using a specific type of queue: a hierarchical queue. This queue-driven propagation is sequential and only efficient for uniprocessors. The current limitation of the clock rate by physical constraints brought the need for multiprocessor architectures and adaptation of usual methods to the new specifications. The hierarchical queue principle is challenging to optimize for modern architectures and does not allow to meet the speed required by the ever-growing amount of data to be processed. This limitation motivates the work presented in this study.

In this article, we present a new method to calculate watershed lines using a local hierarchy. Compared to traditional methods, the proposed method preserves the locality of data and allows an optimized implementation without compromising the quality of the extracted contours. The new propagation scheme alleviates the sequential nature of hierarchical queue propagation. This yields an increase in speed and an efficient use of memory. We compare ourselves with the hierarchical queue-driven propagation [5].

2 Related Work

The notion of watershed was first introduced in [3] and its computation involves an iterative process of successive thresholding at increasing heights. An algorithm based on a hierarchical queue was introduced in [17] and is considered in many aspects as optimal. The hierarchical queue is used to process pixels in increasing order. A rigorous definition of the watershed was given during the first workshop of mathematical morphology in 1993 [11, 22] and the topographic distance was introduced to unify the watershed line and the SKIZ transform independently in [18, 22]. The watershed was shown to be equivalent to the SKIZ of the topographic distance and the hierarchical queue implements this definition.

In a digital grid, complications appear and lead to different approaches. One can be interested in creating a tessellation, where each tile contains pixels that are closer to a seed than any other. Ambiguity appears when a pixel is equidistant to two different seeds. On the other hand, one can be interested in a thin contiguous line that defines a set of points equidistant from two distinct seeds. Two adjacent pixels can exist, so that the first is closer to a seed and the second to another without a third pixel in between. Both approaches require an arbitrary choice for assigning such pixels to one seed or a watershed line living on both edges and nodes, as produced by the proposed method.

A review on watershed algorithms is made in [24]. The watershed cuts on edge-weighted graphs using minimum spanning forest were defined in [8].

The processus of flooding a relief reproduced using a hierarchical queue suffers unfortunately from a bias introduced by sequential processing of queued pixels at the same level [2]. The bias is detrimental for two reasons, i) it introduces deformations that can potentially grow to arbitrary size, and ii) the resulting watershed line is not unique. Beucher [2] proposes a different approach to remove this bias. Another criterion has been proposed in [20], in which it is shown that the hierarchical queue implements an infinite lexicographic distance. This last criterion minimized the number of arbitrary choices in order to build a partition.

Many parallel implementations have been introduced (see [24], [10], [9], [23]), and a tendency to introduce altered definitions of the watershed transform appeared to simplify the parallelization process. Two classes of parallel algorithms exist: domain decomposition and functional decomposition. Splitting the image into smaller images was studied in [16] and in [6], with a preprocessing of the overlapping areas and distribution of those images to multiple processing units. Gillibert and

Jeulin worked on an iterative processing of smaller images and a merging procedure of the results until idempotence in [12]. However, Beucher et al. performed a study [4] showing the difficulties encountered in the parallelization process of the hierarchical queue algorithm.

Morphological transformations often rely on the relative heights of neighboring pixels. The arrowing operator was first introduced as an efficient representation of the local relation between pixels. This operator was for the first time introduced as early as in 1982 as a mean of processing the watershed transform [15]. Another iterative approach based on the arrowing operator has been introduced in 1990 in [1]. A more recent method on the watershed generalized on graph and implemented with arrowing operators was developed in 2012 in [19].

The algorithm proposed in this paper can be classified as a parallel algorithm, based on the arrowing operator. We propose a method to produce a hybrid result as in [2], where no arbitrary choice is necessary. The resulting watershed line is centered and thin but discontinuous. A tesselation is produced on all pixels except those equidistant to two seeds, where a watershed line is produced.

3 Notations

3.1 The Topographic Relief

We define a gray-level image by the mapping $f : \mathcal{D} \to \mathcal{V}$, where $\mathcal{D}$ is the set of pixels and $\mathcal{V}$ the valuation domain. $\mathcal{V}$ can be $\mathbb{N}$, $\mathbb{Z}$ or $\mathbb{R}$ without impact on performance. We associate this function with its corresponding topographic relief, where all gray values are seen as elevations on the relief. This topographic relief contains a various number of topographic structures such as domes, valleys, ridges, thalwegs, regional minima, plateaus, and so on. Among those, a few interest us as flooding paths, plateaus, regional minima and catchment basins. The absolute heights of the pixels are not needed for defining the topographic structures. They depend only on the relative heights of neighboring pixels.

In the following, let $\sim$ denote the neighborhood relationship on $\mathcal{D}$ defined by the considered connectivity. If a, $b \in \mathcal{D}$ are neighbors and $a \neq b$, we write $a \sim b$. Any usual connectivity can be used, i.e. 4, 6 or 8 in 2D and 6, 18 or 26 in 3D.

By using the relative height of adjacent pixels $a, b \in \mathcal{D}$, $a \sim b$, we can further distinguish

$$a \sim b, \begin{cases} f(a) < f(b) & \Leftrightarrow a \text{ is a } \textit{lower-neighbor} \text{ of } b, \text{ noted } a \prec b, \\ f(a) > f(b) & \Leftrightarrow a \text{ is an } \textit{upper-neighbor} \text{ of } b, \text{ noted } a \succ b, \\ f(a) = f(b) & \Leftrightarrow a, b \text{ are } \textit{level-neighbors}, \text{ noted } a \simeq b. \end{cases}$$

3.2 Digraphs

The connectivity relation of adjacent pixels is often encoded by using a non-oriented graph $\mathcal{G}(\mathcal{N}, E)$, where $\mathcal{N} \leftrightarrow \mathcal{D}$ and the set E is generated by the connexity $\sim$. The upper-neighbor and lower-neighbor relations allow defining an *oriented* graph by using the edges from $E_\prec$, $E_\succ$ or $E_\simeq$ generated by the relations $\prec$, $\succ$ or $\simeq$, respectively. These edge sets define the increasing graph $\mathcal{G}(\mathcal{N}, E_\succ)$, the decreasing graph $\mathcal{G}(\mathcal{N}, E_\prec)$ and the level-neighbor graph $\mathcal{G}(\mathcal{N}, E_\simeq)$.

The relations $\simeq$, $\succ$ and $\prec$ are transitive. If $a \simeq b \simeq c$ then $f(a) = f(b)$, $f(b) = f(c)$ and also $f(a) = f(c)$ and a is path-connected to c by a level constant path. Similarly, if $a \succ b \succ c$ then $f(a) > f(c)$ also and a is connected to c by a strictly decreasing path.

3.3 Plateaus and the Minima Set

Definition 1 *For a point a the transitive closure $\simeq^+$ of a defines a* plateau p *– a connected region of a constant-level – with $a \in p$.*

From now on, let $\mathcal{P} = \bigcup_i \{p_i\}$ denote the set of all plateaus and p_i its elements.

Definition 2 *A pixel whose neighbors are all upper-neighbors is a* local minimum*.*

Let $\mathcal{S}$ denote the set of all local minima.

A plateau from where one cannot reach a lower altitude with a non-increasing path is called *regional minimum*. We can define it using the following property:

Definition 3

$$m \in \mathcal{P}, m \text{ is a regional minimum} \Leftrightarrow \\ \forall a \in m, \nexists b \in \mathcal{N}, a \succ b.$$

We will denote by $\mathcal{R} = \bigcup_i \{m_i\}$ the set of all regional minima. The set of all minima of an image, denoted by $\mathcal{M}$, is the union of both regional and singleton minima, i.e. $\mathcal{M} = \mathcal{R} \cup \mathcal{S}$. Note that local minima are not included in regional minima because isolated points are not plateaus (recall from above that $\sim$ is not considered reflexive).

3.4 Detection of Minima

This section discusses a detection method of minima and the labeling of these sets. The set of minima $\mathcal{M}$ is obtained by removing from $\mathcal{P}$ all plateaus p_i that are not minima.

A plateau that is not a minimum is a plateau containing at least one pixel that has a lower-neighbor. For some plateau p_i, the subset of pixels $V \subset p_i$ that have at least one lower-neighbor is called a *decreasing border* $V = \{a \in p_i, \exists b \in \mathcal{N}, b \prec a\}$.

The set of minima is then obtained by

$$\mathcal{M} = \mathcal{P} \setminus \{p_i \mid \exists a \in p_i, \exists b \in \mathcal{N}, b \prec a\}$$

Notice that, in terms of mathematical morphology, this operation is an opening by reconstruction of $\mathcal{P}$ with a marker set $V = \{a \mid \exists b, b \prec a\}$ which is the decreasing border (provided it exists) of any plateau which is consequently not a minimum. See Figure 1 for a 1-D example of this process. The circles denote the nodes $\mathcal{N}$ of a graph, denoted by letters a to i. The nodes with higher/lower altitude appear higher/lower in the drawing. The single-headed arrows represent the edges of the increasing graph and the double-headed arrows the edges of the level-neighbor graph. Nodes connected with double-headed arrows represent the plateaus, denoted by p_1, p_2 and p_3. The decreasing borders of plateaus appear in red. Plateaus containing at least one red node are not minima. The Figure 1 contains two minima: p_1 and $\{h\}$.

The minima need to be assigned a unique label, in order to reconstruct the different basins. Several methods can be used in order to label the connected components. Most common methods involve a two pass raster scan method as in [25, 26]. A review of parallel architectures and algorithms for this problem can be found in [7].

In the following section we show that various sets (defined using paths) can be associated to these minima and that these sets can be constructed using local information only.

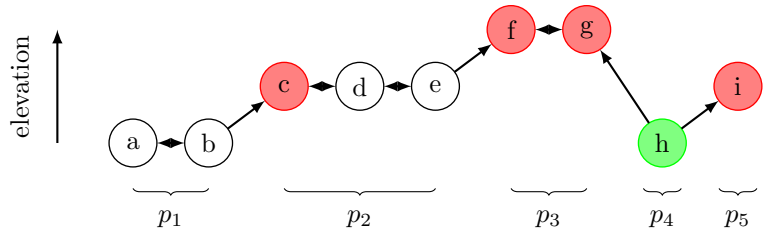


Figure 1: The extraction of minima using $\mathcal{G}(\mathcal{N}, E_{\simeq} \cup E_{\succ})$ in 1-D. Circles represent nodes $\mathcal{N}$. Double/single-headed arrows represent $E_{\simeq}/E_{\succ}$, respectively. The plateaus are $\mathcal{P} = \{p_1, p_2, p_3, p_4, p_5\}$, with the decreasing borders $V = \{\{c\}, \{f, g\}, \{i\}\}$ (in red). $\mathcal{M} = \mathcal{P} \setminus \{p_2, p_3, p_5\} = \{p_1, p_4\}$.

3.5 Extraction of Catchment Basins from Minima

Given a convenient distance $d : \mathcal{N} \times \mathcal{N} \to \mathcal{R}^+$, a watershed basin associated to a minimum m_i is usually defined as a collection of points closer to m_i than to any other minimum m_j

$$\mathcal{B}_i = \{x \mid d(m_i, x) < d(m_j, x), \forall j \neq i\}, \tag{1}$$

ensuring the empty intersection of any two basins

$$i \neq j \Rightarrow \mathcal{B}_i \cap \mathcal{B}_j = \emptyset. \tag{2}$$

In case of equality of the distance to two minima a watershed line is formed

$$\mathcal{W} = \{x \mid \exists i,j, i \neq j : d(m_i, x) = d(m_j, x) = \min_k d(m_k, x), \forall k\},$$

and we have

$$\mathcal{W} = \mathcal{N} \setminus \bigcup_i \mathcal{B}_i. \tag{3}$$

A topographic distance is usually defined through the concept of the length of path. Let $\rho(a, b) = (a = x_1, \ldots, x_n = b) \subset \mathcal{N}$ with $x_{i+1} \sim x_i$ be a path from a to b. The distance from a to b is the length of the shortest path from a to b

$$d(a, b) = \min_{\rho_i \in \mathscr{R}(a,b)} l(\rho_i), \tag{4}$$

with $\mathscr{R}(a, b) = \{\rho(a, b)\}$ the collection of all paths from a to b and $l(.)$ some measure of length (precised below).

In order for the basins $\mathcal{B}_i$ to be connected we also need $\forall x \in \mathcal{B}_i$ that the shortest path to its minimum to be included in $\mathcal{B}_i$

$$\forall x \in \mathcal{B}_i \Rightarrow \exists \rho(m_i, x) \subset \mathcal{B}_i \text{ such that } \rho(m_i, x) = \arg \min_{\rho_j \in \mathscr{R}(m_i,x)} l(\rho_j). \tag{5}$$

Let $l(\rho)$ denote the length of the path ρ defined by $l(\rho) = \max_{x_i \in \rho} f(x_i)$ as the maximum value found alongside the path. This distance reflects the minimum altitude to climb when moving on a topographic relief.

Let us further restrict $\mathscr{R}(a, b) = \{\rho^{\succ}(a, b)\}$ that is to the increasing paths $\rho^{\succ}(a, b) = (a = x_1, \ldots, x_n = b)$ with $x_{i+1} \succ x_i$. When no increasing path exists from a to b we say that $d(a, b) = \infty$.

Two sets can be assigned to each minimum m_i. The set of all points reachable from m_i

$$\mathcal{B}_i^{\cup} = \{x \mid d(m_i, x) < \infty\},$$

and the set of points reachable only from m_i is

$$\mathcal{B}_i^{\subset} = \{x \mid d(m_i, x) < \infty \text{ and } d(m_j, x) = \infty,\ j \neq i\}. \tag{6}$$

We obviously have $m_i \subseteq \mathcal{B}_i^{\subset} \subseteq \mathcal{B}_i^{\cup}$.

On increasing paths the distance in b from a equals

$$d(a, b) = \begin{cases} f(b) & \text{when } \exists \rho^{\succ}(a, b), \\ \infty & \text{otherwise.} \end{cases} \tag{7}$$

meaning that when some point b is reachable from some minimum then its distance to this minimum is the height of b and is independent of the height of the minimum. This intuitively corresponds to a rising water level that has the same height everywhere at any time. It however has several unpleasant consequences:

1. The shortest path to minima (even to that of its basin) is not unique.
2. This means that if a point is reachable from several minima then it is equidistant to all of them.
3. As a consequence of 2), the distance Equation (4) does not provide an order we need for Equation (1). Although it allows to simulate the process of flooding it does not yield a convenient tesselation. Indeed, for some point a and two minima m_i, m_j, with $i \neq j$ we have either $d(m_i, a) = d(m_j, a)$ or $d(m_i, a) < d(m_j, a)$ when $d(m_j, a) = \infty$. Consequently, the basins $\{\mathcal{B}_i^{\cup}\}$ are not disjoint, see Figure 2, and the zones equidistant to two minima are not thin.

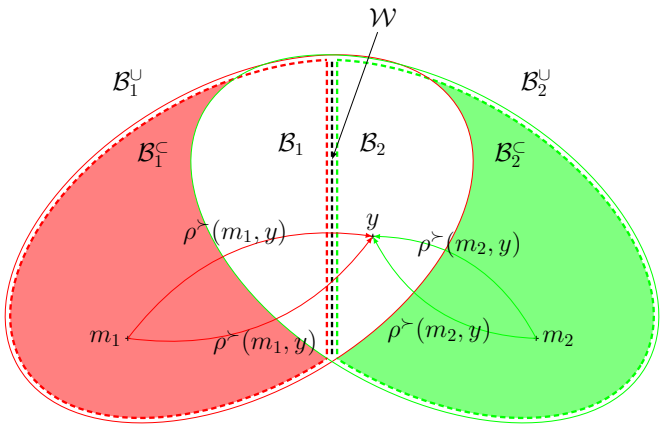


Figure 2: The sets $\mathcal{B}_i^{\subset}$, $\mathcal{B}_i$ and $\mathcal{B}_i^{\cup}$ (with the inclusion relation $\mathcal{B}_i^{\subset} \subseteq \mathcal{B}_i \subseteq \mathcal{B}_i^{\cup}$) generated by the paths $\rho^{\succ}(m_i, y)$ starting in two minima m_1 and m_2 and terminating in y. Any path from m_1 to y must intersect $\mathcal{W}$ therefore $y \notin \mathcal{B}_1$. Legend: $\mathcal{B}_i^{\subset}$ – colour-filled set; $\mathcal{B}_i$ – colour-dashed-line delimited set; $\mathcal{B}_i^{\cup}$ – solid-line delimited set; $\mathcal{W}$ – dashed black line.

Indeed, the basins $\mathcal{B}_i$ from Equation (1) we are looking for verify $\mathcal{B}_i^{\subset} \subseteq \mathcal{B}_i \subseteq \mathcal{B}_i^{\cup}$. In the sequel we show that this distance allows obtaining the collection $\{\mathcal{B}_i\}$ of disjoint basins and a thin watershed line by construction.

For every point, we know all its lower-neighbors. A point can have:

1. no lower-neighbor. This configuration is not interesting since it is either a minimum, or an interior point of a plateau (refer to Section 3.6).
2. all lower-neighbors that belong to the same basin. Then this point also belongs to the same basin.
3. lower-neighbors that belong to different basins (there are increasing paths arriving from different minima). Then because of Equation (7) the above distance does not allow to solve Equation (1).

The cases 2 and 3 are decidable locally as soon as all the lower-neighbors of a point know their labels.

All $\mathcal{B}_i^{\subset}$ can be obtained recursively by using the following lemma.

Lemma 1 *If all lower-neighbors x of y belong to $\mathcal{B}_i^{\subset}$ then y also belongs to $\mathcal{B}_i^{\subset}$*

$$y : \forall x,\ x \prec y, x \in \mathcal{B}_i^{\subset} \ \Rightarrow\ y \in \mathcal{B}_i^{\subset}. \tag{8}$$

Proof 1 *If all lower-neighbors x of y belong to $\mathcal{B}_i^{\subset}$ then there is no increasing path from another $\mathcal{B}_j^{\subset}$ to y. Consequently y belongs to $\mathcal{B}_i^{\subset}$.*

From which it immediately follows that

Proposition 1 *$\mathcal{B}_i^{\subset}$ can be extracted by region growing from its seeding minimum m_i and independently of any other $\mathcal{B}_j^{\subset}$, $j \neq i$.*

There will remain some points y that cannot be associated to any $\mathcal{B}_i^{\subset}$ according to Equation (8)

$$y : \exists\, a,b \prec y, \text{ with } a \in \mathcal{B}_i^{\subset}, b \in \mathcal{B}_j^{\subset}, j \neq i \Rightarrow y \in \mathcal{W}. \tag{9}$$

These points will make part of the complementary set, denoted by $\mathcal{W}$ in Equation (3) above. However, $\mathcal{W}$ obtained in this way is not thin.

The condition on the lower-neighbors x of y in Equation (8) can be further relaxed to obtain larger $\mathcal{B}_i$ instead of much too restrictive $\mathcal{B}_i^{\subset}$. The $\mathcal{B}_i$ sets will also be obtained by construction using region growing. Proposition 2 below gives the definition by construction of $\{\mathcal{B}_i\}$ and the property that they are disjoint.

Definition 4 (Watershed basins) *Given two collections of sets $\{\mathcal{B}_i^{\subset}\}$ and $\{\mathcal{B}_i^{\cup}\}$ there is a collection of watershed basins $\{\mathcal{B}_i\}$ such that $\mathcal{B}_i^{\subset} \subseteq \mathcal{B}_i \subseteq \mathcal{B}_i^{\cup}$, for $\forall i$. The basins $\mathcal{B}_i$ are disjoint (Equation (2)) and each point of a basin $\mathcal{B}_i$ has a shortest path to its minimum m_i that is fully contained in $\mathcal{B}_i$, Equation (5).*

The collection $\{\mathcal{B}_i\}$ defined by this definition is not necessarily unique. However, a unique solution can be obtained by construction using the following proposition.

Proposition 2 *The collection* $\{\mathcal{B}_i\}$ *can be obtained by region growing, each* $\mathcal{B}_i$ *from the minimum* m_i*, by using*

$$\{x \prec y\} \subset \mathcal{B}_i \cup \mathcal{W} \text{ and } \{x \prec y\} \cap \mathcal{B}_i \neq \emptyset \Rightarrow y \in \mathcal{B}_i. \tag{10}$$

Proof 2 *We show two properties.*

1. *We show that* $y \in \mathcal{B}_i^{\cup}$ *such that* $\mathcal{B}_i \subset \mathcal{B}_i^{\cup}$ *and* $\mathcal{B}_i$ *satisfies Equation (5).*

   If $\exists x \in \mathcal{B}_i$ *then there is a path* $\rho^{\succ}(m_i, x) \subset \mathcal{B}_i$ *and* $l(\rho^{\succ}(m_i, x)) < \infty$ *and therefore also* $l(\rho^{\succ}(m_i, y)) < \infty$ *and* $\rho^{\succ}(m_i, y) \subseteq \mathcal{B}_i$*.*

2. *We show that* $y \notin \mathcal{B}_j$ *when* $i \neq j$ *(the disjointness of* $\{\mathcal{B}_i\}$*). Remark: We use the induction on the increasing graph* $\mathcal{G}(\mathcal{N}, E_{\succ})$ *since* $x \in \mathcal{B}_i$ *implies that* $x \notin \mathcal{B}_j$*. See the illustration in Figure 2.*

   Suppose some $y \in \mathcal{B}_i$ *such that* y *is also reachable from some other minimum* m_j*,* $j \neq i$*. That is, there is* $\rho^{\succ}(m_j, y)$ *such that* $l(\rho^{\succ}(m_j, y)) < \infty$*.*

   Consider $x \prec y$ *and* $x \in \rho^{\succ}(m_j, y)$*, the predecessor of* y *that links* y *to* m_j*. From Equation (10) and* $y \in \mathcal{B}_i$*, we have for all* $x \prec y$ *either* $x \in \mathcal{B}_i$ *or* $x \in \mathcal{W}$*. Consequently, we have that* $\rho^{\succ}(m_j, y) \not\subseteq \mathcal{B}_j$ *then* $y \notin \mathcal{B}_j$*.*

From Proposition 2 immediately follows the Equation (12b) of the algorithm. The condition Equation (9) gives Equation (12c).

3.6 The Flooding Graph

Water flooding a relief rises from bottom up. A natural choice to simulate this phenomenon is to use the increasing graph $\mathcal{G}(\mathcal{N}, E_{\succ})$, since at every pixel there are arrows to all its upper-neighbors. However, this graph contains no arcs on plateaus to indicate how to flood the plateaus.

If a natural relief plateau is flooded, the water floods from its decreasing borders onwards. To simulate this behavior analogously, we shall equip all plateaus with $E_{\succ}$ arrows oriented from the decreasing borders onwards. This behaviour is well known and has been referred to as the lower completion of the graph [18]. We equip the graph on plateaus with arrows in the way that interior points in plateaus can now be reached from minima. The image values and element ordering in the rest of the image remain the same.

Consider some plateau p and its decreasing border $V \subset p$. Let $d_p(a, V)$, $a \in p$ denote a geodesic distance in p from a to V. We redefine the set of increasing edges in the following way. Let $a, b \in \mathcal{N}$, such that $a \sim b$

$$e_{b,a} \in E_{\succ} \text{ if } \begin{cases} f(a) > f(b) \text{ or} \\ d_p(a, V) > d_p(b, V) \text{ when } a, b \in p. \end{cases} \tag{11}$$

Figure 3 shows in 1-D the completion of the increasing graph of the initial image with an increasing graph of the geodesic distance on plateaus. The plateau p_2 is equipped with two arrows (in red). The plateaus p_1 and p_3 are left unchanged (a regional minimum, and a plateau only containing a decreasing border). Now, when all minima are labeled and the increasing graph is completed on the plateaus, we will propagate the labels to the catchment basins. Figure 4 shows in 2-D the completion of the increasing graph and the corresponding flooding.

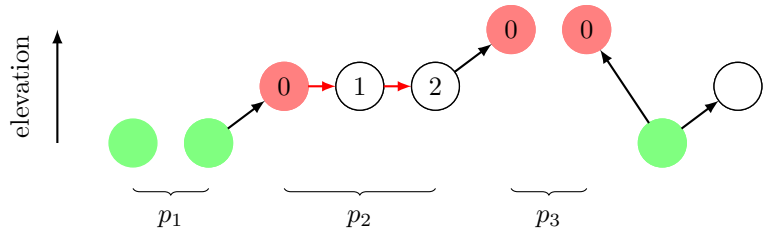


Figure 3: Completion of the graph $\mathcal{G}(\mathcal{N}, E_{\succ})$ from Figure 1. Nodes are valued with the distance from the decreasing border. The added arrows are given in red. (Edges from $E_{\simeq}$ are not shown.)

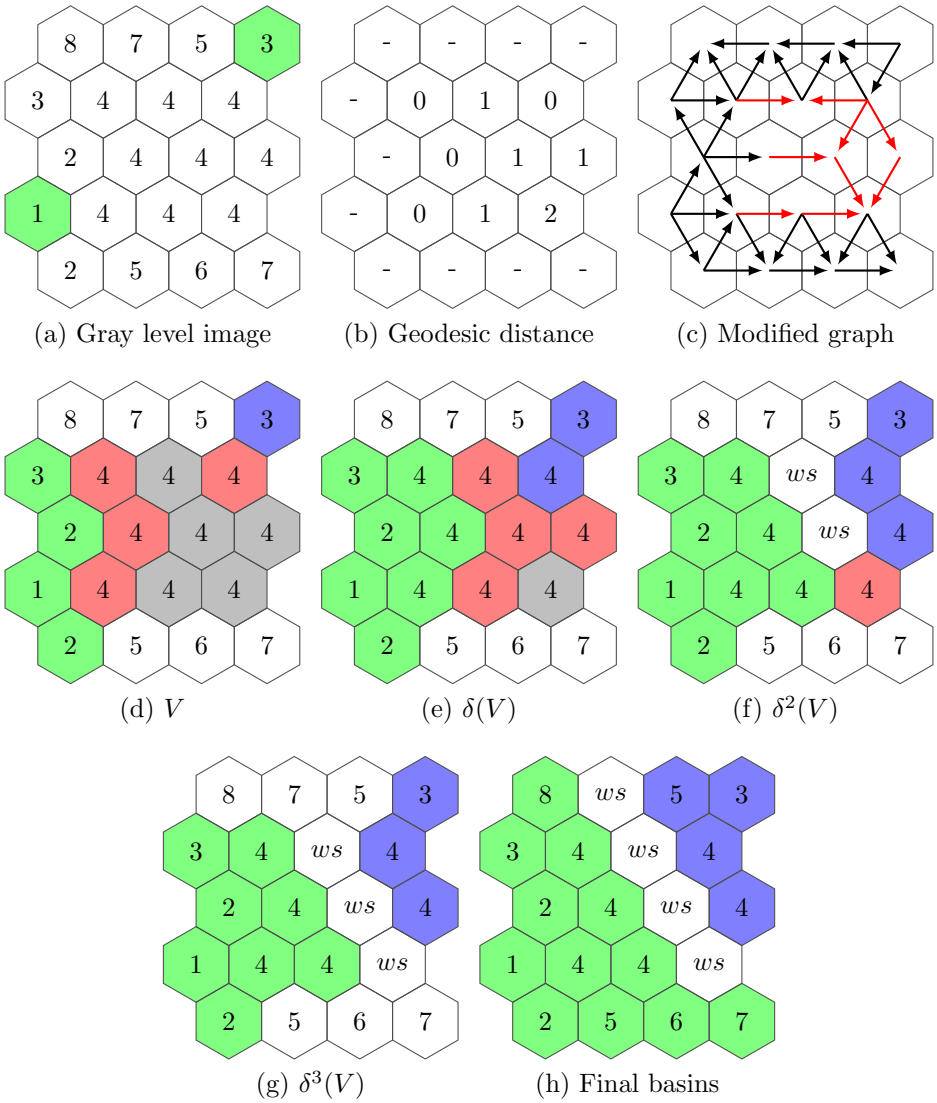


Figure 4: Completion of the graph $G(\mathcal{N}, E_{\succ})$ in 2-D. (a) gray-level image, with two local minima (green) and one plateau (pixels valued 4) (b) geodesic distance on a plateau to the decreasing border. (c) flooding graph, increasing graph on the relief (black), increasing graph on the geodesic distance (red). (d, e, f, g) the hierarchical flood of a plateau starting from the descending border from two basins (blue and green). (h) The final basins when the propagation ends. In red are shown the nodes affected by the next dilatation. Nodes from the watershed set are labeled ws. It is shown that each step of the hierarchical flood implements a geodesic dilatation on plateaus from the descending border.

4 A Cellular Automaton for Catchment Basins Extraction

Algorithms to compute the watershed transform using cellular automata have been proposed in [14, 23, 13]. We can define the proposed procedure as a cellular automaton executed independently on every node of $\mathcal{N}$. This cellular automaton will have two separate behaviors for label propagation and for homotopy modification. In this section, we will write a rule that will propagate the labels from the minima upwards. In the next section, we will introduce a new rule to modify the topographic relief in places where the first rule was not able to propagate a label due to the discrete configuration known as a button hole. In a given step of the execution of the cellular automaton, those two rules apply to two separate subsets of $\mathcal{N}$. One subset corresponds to the set of nodes where a label can propagate. The other subset corresponds to the set of nodes which are part of a button hole.

Let a cellular automaton be the tuple $(\mathcal{S}, T, f)$. $\mathcal{S}$ is the set of possible states of a node, noted as a pair of label and elevation such as $\mathcal{S} \subset \mathcal{L} \times \mathcal{V}$. Let $\mathcal{L} = U \cup F$, where $U = \{\emptyset, \beta\}$ is the set of intermediate labels and $F = \mathbb{N} \cup \{ws\}$ is the set of final labels.

We note $T \subset \mathcal{N}$, the set of neighbors, so that $\forall a \in \mathcal{N}, b \in T \rightarrow a \sim b$. The state of a node will depend on its current state and on the states of its $|T|$ neighbors. Therefore, $s : \mathcal{S}^{|T|+1} \rightarrow \mathcal{S}$ is the transition rule.

Let $l^i(x)$ be the label of x at step i and $h^i(x)$ be the elevation of x at step i. After the initial step of preprocessing minina, we obtain sets of labeled nodes corresponding to the minima and a set of unlabeled nodes. The initial state of this cellular automaton is therefore the following

$$l^0(a) \leftarrow \begin{cases} j, & \text{if } a \in m_j, \\ \emptyset, & \text{otherwise.} \end{cases}$$

$$h^0(a) \leftarrow f(a).$$

We redefine the relations $\prec$, $\succ$ and $\simeq$ taking into account the modification of homotopy

$$a \sim b, \begin{cases} h^i(a) < h^i(b) & \Leftrightarrow a \text{ is a } \textit{lower-neighbor} \text{ at step i of } b, \text{ noted } a \prec b, \\ h^i(a) > h^i(b) & \Leftrightarrow a \text{ is an } \textit{upper-neighbor} \text{ at step i of } b, \text{ noted } a \succ b, \\ h^i(a) = h^i(b) & \Leftrightarrow a, b \text{ are } \textit{level-neighbors} \text{ at step i, noted } a \simeq b. \end{cases}$$

To expand the initial catchment basins, we will expand them with nodes that verify Equation (9) or Equation (10). Notice that, in both equations, the label assigned to a node depends only on the labels of its lower-neighbors. For some node $a \in \mathcal{N}$, let $Q^i(a) = \{l^i(b) \mid b \prec a\} \setminus \{ws\}$ denote at step i the set of different labels on lower-neighbors of a except the special watershed label ws. The set Q^i is without repetition, that is, if some node a has three lower-neighbors $a \succ b_i$, $i = 1, 2, 3$, bearing labels $l(b_1) = 1$, $l(b_2) = 2$ and $l(b_3) = 1$, then $Q^i = \{1, 2\}$. Let us further denote $|Q^i|$ the cardinal of Q^i, here $|Q^i(a)| = 2$. Let $\lambda \in \mathcal{L}$ denote a label, and $a \in \mathcal{N}$ a node. The following transition rule constructs catchment basins from minima

$$\begin{aligned} &a \in \mathcal{N},\ l^i(a) = \emptyset,\ \emptyset \notin Q^i(a) : \\ &\quad \beta \in Q^i(a) \text{ or } |Q^i(a)| = 0, && l^{i+1}(a) \leftarrow \beta,\ h^{i+1}(a) \leftarrow \infty && \text{(12a)} \\ &\quad |Q^i(a)| = 1, && l^{i+1}(a) \leftarrow \lambda, Q^i(a) = \{\lambda\} \subset \mathbb{N} && \text{(12b)} \\ &\quad |Q^i(a)| > 1, && l^{i+1}(a) \leftarrow ws, && \text{(12c)} \end{aligned}$$

Notice that the rule (12) is only applied to nodes which fulfill the condition $\emptyset \notin Q^i(a)$. At step 0, this subset of $\mathcal{N}$ corresponds to some nodes adjacent to the set of minima. At step i, it corresponds to the nodes whose lower-neighbors have all been processed.

Three rules appear in Equation (12): The basins $\mathcal{B}_i^{\subset}$ are built independently from other $\mathcal{B}_j^{\subset}$ (12b). Intersecting influence zones $\mathcal{B}_i$ and $\mathcal{B}_j$ are built using (12c) by separating them by label ws. Finally, rule (12a) corresponds to a case not defined in Section 3.5, where no basin labels are available at the lower-neighbors. These pixels are assigned a special label β denoting pixels where no basin label can be assigned yet. The following section proposes a method to handle the button hole situation and assign all β nodes their final labels.

5 Button Holes

We have seen that the catchment basins grow from the minima using Propositions 1 and 2. If the lower neighbors of a point carry the same label, this label propagates further (Proposition 1, rule (12b)). If the lower-neighbors carry the watershed label and only one basin label, this label propagates too (Proposition 2, rule (12b)). Section 3.5 however does not handle the case where no lower-neighbor at all is labeled with a basin label. A button hole is entered by a point whose lower-neighbors carry all the watershed label.

Definition 5 *A pixel a belongs to a button hole if at the moment when all its lower-neighbors b, $b \prec a$, are assigned a label, i.e. $\emptyset \notin Q^i(a)$, all b carry either the watershed label ws or the button hole label β.*

The rule Equation (12a) assigns temporarily the button hole points the special (button hole) label β.

Definition 6 *A button hole (BH) is a connected component of pixels labeled β for $i = \infty$ when the growing process (Equations (12)) ends.*

Until now, the label of one node depended only on its direct lower-neighbors. If rules (12) are applied as such, it results in (potentially thick) zones of β label, that do not belong to the final state F. Figure 5 shows a case of button hole using the hexagonal connexity.

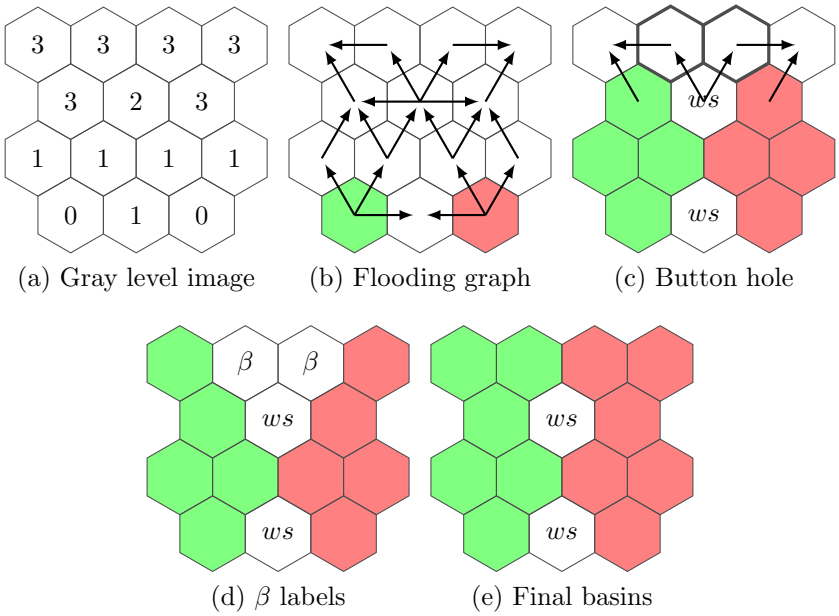
A button hole is bordered by a crest line.

Definition 7 *Button hole crest line (BCL)*

$$a \text{ is a button hole crest line} \Leftrightarrow l(a) = \beta, \exists b \sim a, l(b) \in \mathbb{N}.$$

A BCL is therefore a point whose one neighbor holds a label. However, this label was not propagated upward due to the presence of a β label in the neighborhood. One can notice that an upper-neighbor necessarily inherits the β label from the central point. Thus, a label is either carried by a lower-neighbor or a level-neighbor.

The flood of N button holes is considered as N constrained watersheds on subsets of points with their corresponding BCL points as markers, labeled by the procedure described by Equation (12). We propose in this paper to label each button hole individually with a hierarchical flooding process. In the classical algorithm, the priority of each queue is defined as the height in the relief image. In our work, BCL can occur within plateaus. Thus, the priority is defined as the height completed by the geodesic distance, as proposed in Equation (11).



(a) Gray level image (b) Flooding graph (c) Button hole

(d) β labels (e) Final basins

Figure 5: The appearance of button holes. (a) gray level image with two minima. (b) increasing graph with the two minima labeled red and green. (c) The only lower-neighbor (arrow arriving from) of the two nodes given in heavy line carries ws label. (d) the rule (12a) assigns the button hole the label β. (e) Final basin labels after propagation inside the button hole.

6 Implementation Details

In this section, details on the implementation are given as to how the parallel computing of catchment basins is achieved without data race. A memory-shared parallel architecture, that ensures cache coherency, is assumed. Nodes that are ready to be processed are queued in a task pool, from which each thread can independently extract a node to process. Firstly, the geodesic distance, an essential transformation to the proposed method, is discussed. Secondly, the implementation of rules (12a), (12b) and (12c) is discussed. Finally, details are given on the flooding of button holes. A synchronization process is presented, that relies on the use of counters, to ensure that a final label is only assigned once to a given pixel. An overview of the whole flooding process will be given at the end of this implementation analysis.

6.1 Geodesic Distance on Plateau

Section 3.6 discusses the completion of the increasing graph on f with the increasing graph of the geodesic distance on plateaus. In practice, an additional image is used, corresponding to the array *homotopy* in the code, in which the geodesic distance is stored. This image gives to each point x the value $d_p(x, V)$, where p is a plateau, and V its decreasing border. The geodesic distance allows to verify the relations of *lower-neighbor*, *upper-neighbor* and *level-neighbor* between adjacent points.

The computation of the geodesic distance on P plateaus can be considered as P independent computations of the geodesic distance. Given a plateau p, and its decreasing border V, the computation of its geodesic distance is an iterative process, in which a collection of points is maintained, acting as a `FIFO` (*First In First Out*).

At step 0, a collection is initialized with the points of V. Each point x is successively removed from this collection, and the neighbors of x are processed. If an unassigned neighbor y of x is contained within the plateau P, y is associated with $d(x, V) + 1$ and is added to the collection. This algorithm ends when the collection is empty. Algorithm 1 synthetizes this process.

Due to its nature, this algorithm can easily be transposed to a parallel architecture as each

Algorithm 1: Computation of the geodesic distance on a plateau. C is a FIFO (a queue) and is initially empty.

input : P a plateau, V its decreasing border
output: A map $d_P(x, V)$ which associates the geodesic distance to each x of P.

```
1  foreach element x of P do
2      d_P(x, V) = ∞;
3  foreach element x of V do
4      d_P(x, V) = 0;
5      ENQUEUE(x, C);
6  while C is not empty do
7      x ← DEQUEUE(C);
8      foreach y, y ∈ P, x ∼ y, d_P(y, V) = ∞ do
9          d_P(y, V) = d_P(x, V) + 1;
10         QUEUE(y, C);
```

plateau can be processed separately without requiring any synchronization. All plateaus are distributed evenly among threads. However, the labeling of plateaus would require an efficient connected component labeling algorithm, which is not provided within the source code distributed along this document. Many efficient and parallel implementations of the labeling transformation exist, and we indicate how the geodesic distance can be computed linearly without data race. In our implementation, we have chosen a different approach. Condition $d_P(y, V) = \infty$ of line 9 of Pseudo-code 1 is replaced by $d_P(y, V) > d_P(x, V)+1$. However, this method does not guarantee a linear complexity as some pixels might be queued multiple times. The number of times a pixel can be queued is however bounded by the number of active threads. The pixels from decreasing borders of all plateaus are spatially distributed amongst threads, in order to reduce the probability that a point is processed multiple times by different threads.

6.2 Upward Propagation

The upward propagation is described by Equations (12a), (12b) and (12c). Those rules activate on a pixel a when $\emptyset \notin Q^i(a)$. This last relation can be interpreted as: "All decreasing neighbors of a have been processed through either (12a), (12b), (12c)". Their labels are therefore different from $\emptyset$.

In a discrete image with a static neighborhood relationship, there is a finite number of lower-neighbors. This observation is at the core of the synchronization mechanism. The lower-neighbors are enumerated for each pixel, and we obtain the following mapping $\texttt{count} : \mathcal{N} \to \mathbb{N}$, which is given by the following equation

$$\texttt{count}(x) = |\{y, y \prec x\}|. \tag{13}$$

One can observe that, at initialization, the corresponding `count` value is 0 for each minimum, while every other point has a counter of value greater than 0. A point is considered ready to be labeled through rules (12), once its counter has reached 0.

A collection C is used to keep track of the flooding. This collection holds at any given time of the algorithm the points, whose `count` value is 0. Any point of this collection can activate rule (12a), (12b) or (12c). When those rules are activated on a given point x, the `count` value for every successor of x is decremented. If the `count` of a successor reaches 0, this point can be added to C.

At the initialization of the algorithm, the collection C holds only the labeled minima. Starting from the minima, every increasing path is progressively explored, in the breadth first search manner. Algorithm 2 describes the upward propagation of labels.

Algorithm 2: Upwards propagation of labels. C is a FIFO (a queue) and is initially empty.

input : $\mathcal{M}$, the set of minima
output: A map $l(a)$, which associates a label, ws or β to each point.

```
1  foreach element m of M do
2      foreach element x of m do
3          QUEUE(x, C);
4  while C is not empty do
5      x ← DEQUEUE(C);
6      activate on x rule (12a), (12b) or (12c);
7      foreach y, x ≺ y do
8          count(y) ← count(y) − 1;
9          if count(y) is 0 then
10             ENQUEUE(y, C);
```

Progressively, each point has its counter decreased until it reaches 0. Once a value of 0 is reached, the point is queued in C.

Multiple approaches can be explored for the parallel implementation of Pseudo-code 2. The first approach involves maintaining distinct collections $C_1, \ldots, C_T$, where T is the number of threads. If T threads are used, then the minima are distributed evenly amongst T collections. Each collection is bound to a thread and they do not overlap each other. The thread-safe implementation requires that each point is added in and removed from exactly one collection. For a given thread, nodes are extracted from its collection, processed, and successors, whose count reaches 0, are stored in the same collection. Thus, this approach could induce an imbalance in the amount of work each thread produces and a thread with an empty collection remains idle.

In the source code, another approach is preferred, which involves a more complex structure that allows for the concurrent access (read and write), and the concurrent addition of nodes from multiple consumers. Recently, the implementation of such structures has become an active topic, see [21].

The critical aspect of the parallelization of our algorithm is that each node must be queued only once. Whenever a point is added to the collection, that is its count reaches 0, the thread-safety is ensured by imposing that the decrementation from line 8 and the test from line 9 are processed in a thread-safe manner. In modern architectures, this can be achieved by using an atomic capture that involves both decrementing and reading the value of $\texttt{count}(y)$. It guarantees that only one thread will effectively see the value 0 for a given pixel, and that only this thread will add this point to its collection. Thus, no matter what the preferred approach is (a shared collection or disjoint collections), the point in guaranteed to exist only once.

Once the thread-safety of $\texttt{ENQUEUE}(y, C)$ is ensured by this atomic capture, one can easily see that no other portion of Pseudo-code 2 holds potential data races. The activation of (12a), (12b) and (12c) is based on the value of lower-neighbors, the values of which can not be changed anymore as their associated `count` value is already 0. If cache coherency is ensured, as assumed previously, and as there is no further writing once `count` has reached 0, the values of processed points are guaranteed to be coherent to every thread.

At the end of the upward propagation as described in Pseudo-code 2, each point is associated

to a value different from $\emptyset$. Some will be associated to a final state such as a label λ or the ws value. Others will hold the β value, which is a temporary state, and need to be further processed to obtain their final state. During the upward propagation, all β connected components are uniquely labeled. BCL points are simultaneously detected. After the upward propagation, each button hole is processed concurrently.

6.3 Button Holes as Connected Components

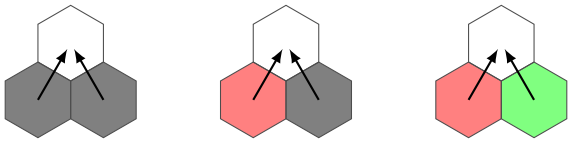
From the previous section, the nodes, which have no lower-neighbors labeled by a natural integer, obtain the β label. In Definition 6, a button hole is defined as a connected component of nodes that are labeled β. In our implementation, we keep track of those connected components with a Union-Find structure [27]. Our process is similar to a connected component labeling algorithm. The upward propagation assigns temporary labels to button holes. Redundancies are solved when they are first encountered during the upward propagation.

A mapping $B : \mathcal{N} \to \mathbb{N}$ is proposed, that keeps track of connected component of β labels. At initialization, no button hole exists and for all n, $B(n) = 0$.

When rule (12a) is applied on a given node, three possibilities arise, see Figure 6. The first possibility is that a new button hole is created. This case corresponds to $\mid Q^i(a) \mid = 0$: "No lower-neighbor holds a label". A new β connected component label is attributed to this node.

The second possibility is that, amongst the lower-neighbors of a node n, one holds β as a label. Then n is attributed to the same connected component through the operation of *Find*.

The third possibility is that two (or more) connected components of β are found within the lower-neighbors of a node n. The two connected components of β labels are fused through the operation of *Union*.



(a) New connected component (b) Find (c) Union

Figure 6: The three different situations when a node is attributed a β label. The white node is a node being attributed a β label. Black nodes are ws nodes. Green and red nodes are two disjoint button holes. In (a), the lower-neighbors are not attributed to any button hole, and a new button hole is created. In (b), a button hole is found, and its label, retrieved using the Find operation, is attributed to the white node. In (c), two button holes are conflicting, a Union is required.

At the end of the upward propagation, each β connected component is labeled uniquely, similarly to Union-Find connected component labeling algorithms.

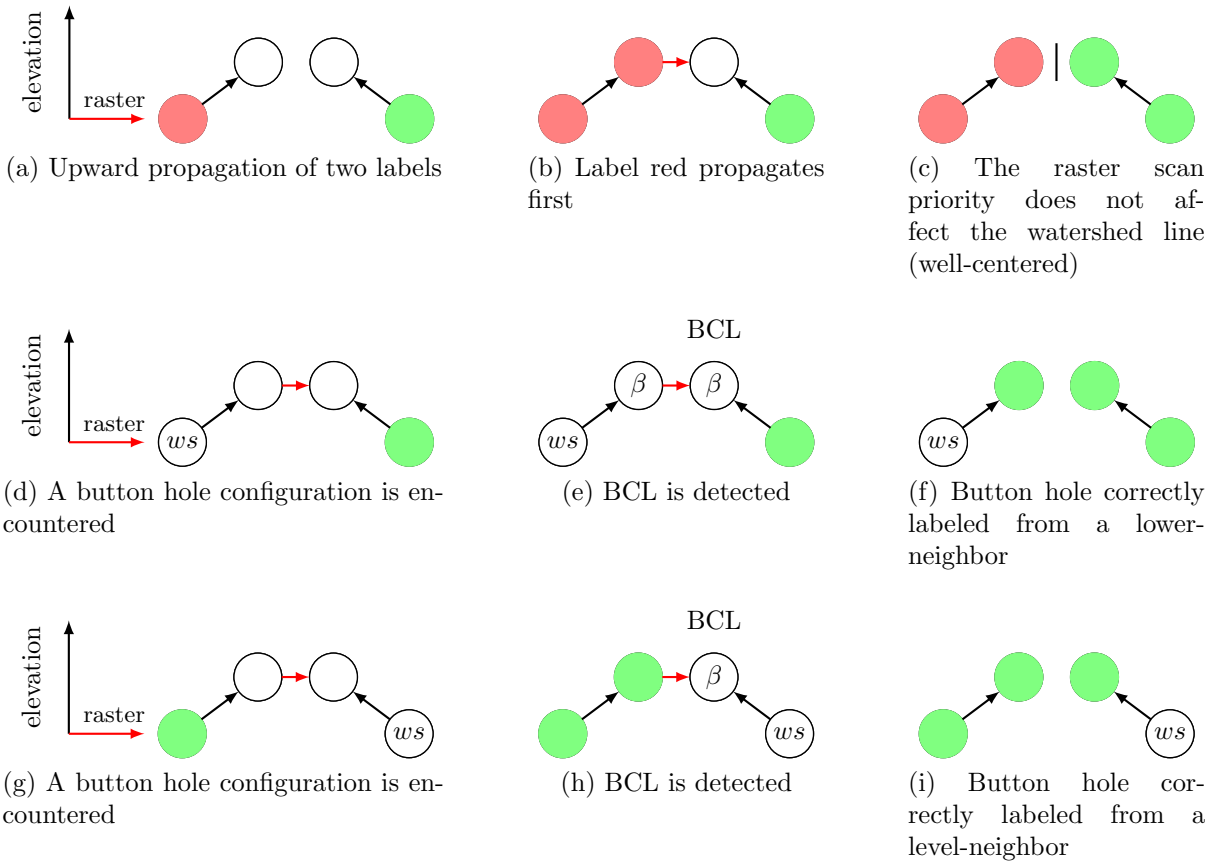
6.4 BCL Detection

The flooding of a button hole involves the hierarchical flooding of the corresponding connected component from its BCLs. In this section, we discuss the detection of BCL points during the upward propagation. We described a BCL point as a point whose one neighbor holds a label and stated that this neighbor is necessarily a lower-neighbor or a level-neighbor, as all upper neighbors inherit the β label. In Pseudo-code 2, a point that is stacked in the collection C is ensured to have all its lower-neighbors processed and labeled. However, the state of the level-neighbors is unknown. In

Figure 7a, we can not predict which of the two points of the plateau will be processed first. Thus, BCLs, as defined in Definition 7, can not be detected when we dequeue a point from the collection.

Button hole crest lines (BCLs) can be detected during the upward propagation using Definition 7, by imposing a strict hierarchy. It involves modifying Equation (13) and line 7 of Pseudo-code 2. In the included implementation, the strict hierarchy is obtained by adding the raster scan hierarchy to points with equal heights and equal geodesic distances. Thus, when a point is dequeued, its lower and level-neighbors are in a known state and Definition 7 can be applied to detect BCLs.

Figure 7 shows this process on three different configurations. The first one (7a,7b,7c) shows that the additional constraint does not affect the final labels of the upward propagation. The configurations (7d,7e,7f) and (7g,7h,7i) shows two configurations with correctly detected BCLs.



(a) Upward propagation of two labels

(b) Label red propagates first

(c) The raster scan priority does not affect the watershed line (well-centered)

(d) A button hole configuration is encountered

(e) BCL is detected

(f) Button hole correctly labeled from a lower-neighbor

(g) A button hole configuration is encountered

(h) BCL is detected

(i) Button hole correctly labeled from a level-neighbor

Figure 7: No information on level-neighbors is needed to propagate labels upwards with rules (12b,c). However, a synchronization is needed in order to correctly detect BCL points. In our implementation, β labels are propagated to level-neighbors w.r.t. the raster scan order. BCLs can therefore be detected during the upwards propagation, as described in Definition 7.

Our goal is to keep track of all BCLs of each button hole connected component for the hierarchical flooding of button holes. In our implementation, a concurrent single-linked list is associated to each button hole connected component. As BCLs are detected, they are queued in the list of the button hole. The concurrency of adding nodes to the list is required but no concurrent read or write on a node of this list is needed. The flooding of button holes can be achieved individually by the use of a hierarchical queue, each button hole being processed in parallel.

6.5 Button Hole Flooding

Once the upward propagation is finished and all connected button holes and their corresponding BCLs are detected, each button hole can be processed separately as described in Algorithm 3. The processing of button holes is closely related to the classical approach of watershed using a hierarchical

queue. BCL points act as markers, and the flooding starts from those points. It progressively floods the button hole from the bottom to the top. In the traditional approach, the relation between neighbors is fully enclosed in the height scalar map. Here, the additional scalar coding the geodesic distance within a plateau is to be considered. One must consider that BCLs can be at different geodesic distance from the border of a plateau. Thus, the use of two hierarchical queues is necessary. The first one acts on the height of pixel and the second one on the geodesic distance. As BCL points are not restricted to the lower part of a button hole, the flooding of button holes from BCLs will flood first lower-neighbors of BCL points. Those lower-neighbors are raised to the flood level by lines 14-15 of Algorithm 3. This raising mimics the way a hierarchical queue processes a button hole. Notice that in case of uncontrolled watershed line 14 has no effect (because of strict equality $f(y) = f(x)$). Its usage is described below, Section 6.7. Line 15 builds the auxiliary geodesic distance to BCL controlling the propagation on plateaus.

All connected components can be processed separately and distributed evenly to different threads.

Algorithm 3: Propagation of the labels within a button hole, level by level (noted BH).

input : BH, a set of connected points labeled β.
output: A map $l(a)$, which associates a label, ws or β to each point.
C_1 and C_2 are two hierarchical queues and are initially empty. QUEUE(x, p, C) is the procedure to add a point x with priority p to the hierarchical queue C. DEQUEUE(C) is the procedure to return and remove the element of C of highest priority. $h(.)$ is assumed to be initialized prior to this routine as described by Algorithm 1.

```
1  foreach element p of BH, p is a BCL do
2      QUEUE(p, f(p), C1);
3  while C1 is not empty do
4      foreach element p = arg min_{x∈C1} f(x) do
5          QUEUE(p, h(p), C2);
6      while C2 is not empty do
7          x ← DEQUEUE(C2);
8          activate on x rule (12b) or (12c) ;
9          if l(x) is a label then
10             foreach y, x ∼ y, y ∈ BH do
11                 if f(y) > f(x) then
12                     QUEUE(y, f(y), C1);
13                 else
14                     f(y) ← f(x);
15                     h(y) ← h(x) + 1;
16                     QUEUE(y, h(y), C2);
```

6.6 Overview

Pseudo-code 4 presents the overview of the proposed algorithm. One can notice that the remaining unassigned points are labeled ws at the end of the algorithm at line 12. Notice that unassigned points can remain in case of fully-enclosed button holes which do not have a BCL as defined in Definition 7.

Algorithm 4: Overview of the watershed algorithm proposed in this paper.

input : f an image defined over $\mathcal{D}$, $\mathcal{M}$ a set of markers (minima in the no-constraint case).
output: The label image l, points of which are either a natural number or ws

```
1  P ← DETECT PLATEAUS(f);
2  V ← DETECT DECREASING BORDER OF PLATEAUS(f, P);
3  foreach element p of P do
4      d_p ← GEODESIC DISTANCE(V, p);
5  foreach element x of D do
6      counts(x) ←| {y, y ≺ x} |;
7      B(y) ← 0;
8  l ← ALGORITHM2(M);
9  foreach connected component b of pixels labeled β do
10     ALGORITHM3(b);
11 foreach element x of D, l(x) = ∅ do
12     l(x) ← ws;
```

6.7 Marker-Controlled Watershed

Marker-controlled watershed consists in flooding the topographic surface from a previously defined set of markers. This principle is a major enhancement of the watershed transformation. It is used to prevent over-segmentation by choosing an adequate set of markers corresponding to the objects of interest.

Marker-controlled watershed can also be obtained using our algorithm. The topographic relief is pierced at the position of markers and, thus, we assign the level 0 to the relief at each marker. This simple modification, prior to executing Algorithm 4, allows the flooding of the relief from a controlled set of markers without requiring a preliminary, costly swamping of the relief by morphological reconstruction.

Recall that manual markers create artificial button holes containing natural unmarked minima inside which need to be handled correctly. Such minima are encountered when $f(y) < f(x)$ (Algorithm 3, line 11), and their relief is swamped by the line 14.

7 Complexity Analysis and Experimental Results

This method has been tested using a shared memory architecture implementing the multiple instructions on multiple data scheme. The synchronization between different computing units is ensured by using atomic operations to protect data I/O.

The method has been implemented on top of OpenMP for randomly generated tridimensional images which sizes vary from 1 million to 1 billion voxels. The benchmarking was done using gcc 5.4.0 on a Linux server with two processors Intel(R) Xeon(R) CPU E5-2640 v3 clocked at 2.6 GHz, each with 8 physical cores that can be hyperthreaded up to 16 virtual cores.

Figure 8 shows the execution times for 60 different images using 32 threads. It has been compared to an up-to-date hierarchical implementation of the watershed as described in [5]. The resulting curve shows that the proposed method performs linearly. The implementations [13, 14] of the watershed on GPU using cellular automaton could be a competitive candidate for this benchmark; the time measurement was estimated to be ten times faster than our method, but the limitation of memory of the GPU does not allow to execute the implementation on images of comparable size.

The execution time $T(P)$, when P processors are used, have been measured. Speedup $SP(P) = T(1)/T(P)$ and efficiency $E(P) = P/SP(P)$ are computed, and Figure 9 shows the behavior of the method when the number of threads varies. An example of execution of the watershed transform is shown in Figure 10.

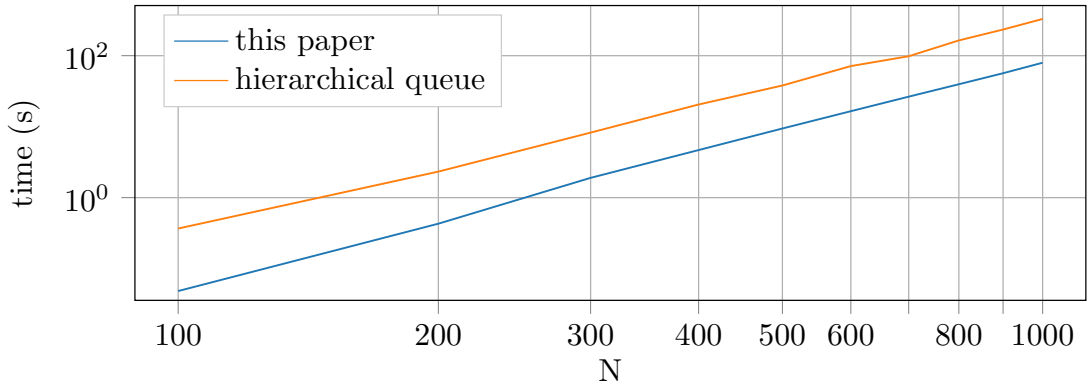


Figure 8: Timing comparison on a wide range of 3D images of size $N \times N \times N$ for N from 100 to 1000.

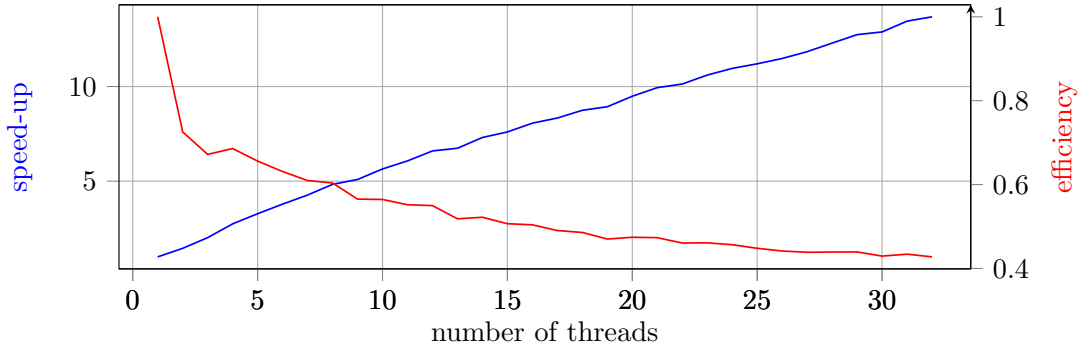


Figure 9: Multi-threaded timings on a 1000^3 image using from 1 to 32 threads.

8 Conclusion

We propose a method to calculate the watershed transform in linear time. This approach starts by detecting the seeds of the flooding, the regional minima of the input image. The image is represented in an acyclic directed graph or forest, using the increasing relation and the geodesic distance on plateaus. A bottom-up propagation is then performed on each seed asynchronously until every minimum has been processed.

It produces identical results to the hierarchical method [2]. It is observed that this algorithm is linear w.r.t. the volume of the input image and that it is scalable. An increasing speed-up is achieved up to 32 processors. Removing the sequential nature of the transform allows to reach higher performances by exploiting parallel computation. This approach shows that transformations of mathematical morphology that were considered difficult to parallelize efficiently can be optimized for modern architectures using optimized synchronizations.

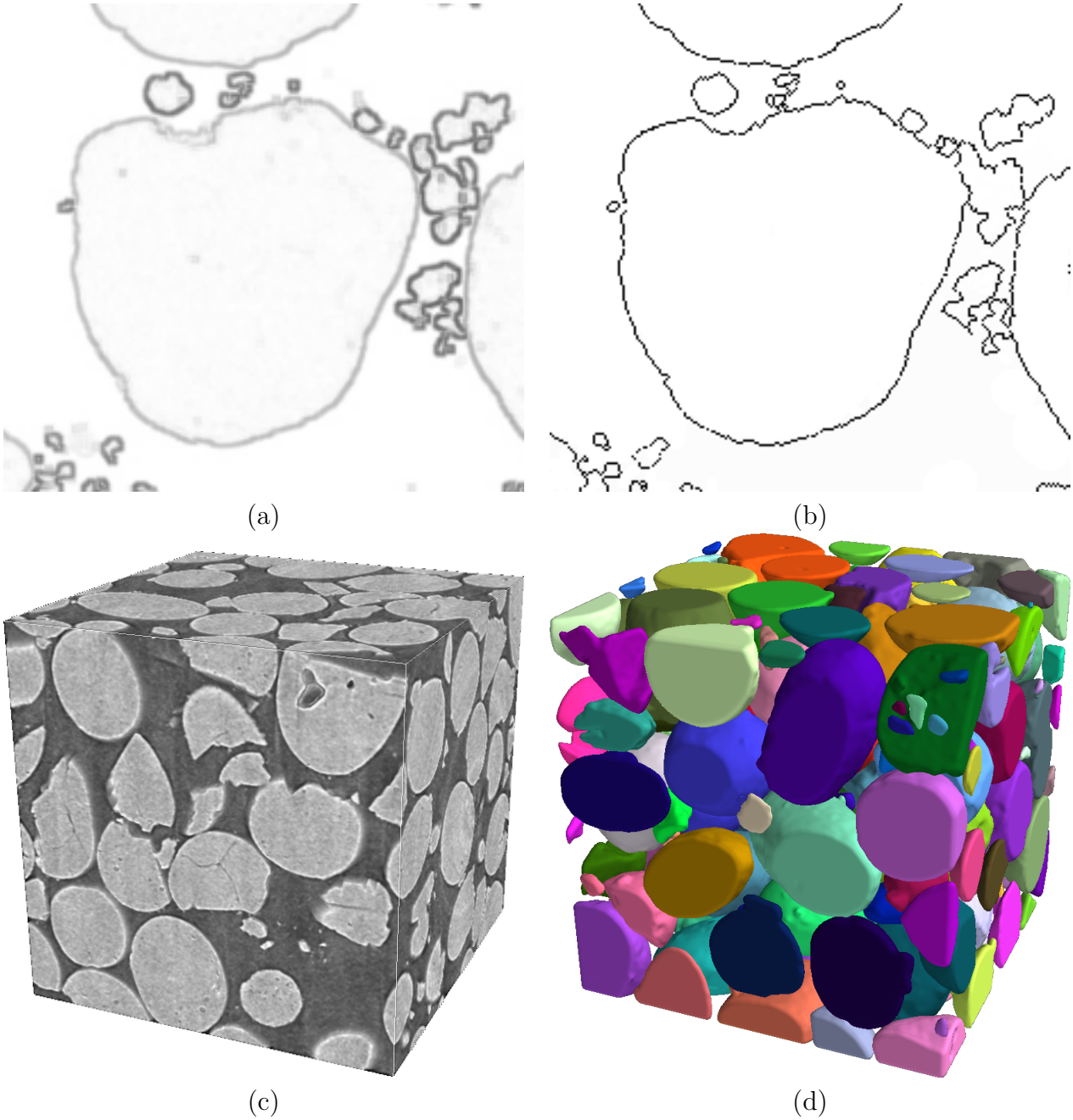


Figure 10: Granular materials. (a) Input gradient image (inversed for better rendering). (b) Watershed lines corresponding to the grain contours. (c) Input image in 3D. (d) Segmented grains in 3D. The colors correspond to labels of minima (one per grain). Notice that there is also one label for the binder (not shown here).

Acknowledgment

The authors wish to thank the reviewers for their thorough analysis of the text and of the (quite a complex) code; and the managing editor Pascal Monasse for his support and help that went well beyond the usual.

References

[1] S. Beucher, *Segmentation d'Images et Morphologie Mathématique*, PhD thesis, Ecole Nationale Supérieure des Mines de Paris, 1990.

[2] ——, *Algorithmes sans biais de ligne de partage des eaux*, tech. report, Centre de Morphologie Mathématique, 2002. http://cmm.ensmp.fr/~beucher/publi/LPE_sans_biais_V2.pdf.

[3] S. Beucher and C. Lantuejoul, *Use of watersheds in contour detection*, in International Workshop on Image Processing: Real-time Edge and Motion Detection/Estimation, 1979.

[4] S. Beucher, F. Lemonnier, and R. Sasportas, *Réalisation de la ligne de partage des eaux par file d'attente hierarchique parallèle*, tech. report, Centre de Morphologie Mathématique, 1997.

[5] S. Beucher and F. Meyer, *The morphological approach to segmentation: the watershed transformation*, in Mathematical Morphology in Image Processing, E.R. Dougherty, ed., CRC Press, 1993, ch. 12, pp. 433–433.

[6] A. Bieniek, H. Burkhardt, H. Marshner, M. Nelle, and G. Schreiber, *A parallel watershed algorithm*, tech. report, Technische Universitat Hamburg-Harburg, 1996.

[7] C.Grana, F. Bolelli, L. Baraldi, and R. Vezzani, *YACCLAB - Yet Another Connected Components Labeling Benchmark*, in International Conference on Pattern Recognition, 2016. https://doi.org/10.1109/icpr.2016.7900112.

[8] J. Cousty, G. Bertrand, L. Najman, and M. Couprie, *Watershed cuts: Minimum spanning forests and the drop of water principle*, IEEE Transactions of Pattern Analysis and Machine Intelligence, 31 (2009), pp. 1362–1374. http://dx.doi.org/10.1109/TPAMI.2008.173.

[9] E. Dejnozkova, *Architecture dédiée au traitement d'image basé sur les équations aux dérivées partielles*, PhD thesis, École Nationale Supérieure des Mines de Paris, 2004. https://pastel.archives-ouvertes.fr/pastel-00001180.

[10] E. Dejnozkova and P. Dokladal, *A parallel algorithm for solving the eikonal equation*, in International Conference on Acoustics, Speech and Signal Processing, 2003. https://doi.org/10.1109/icassp.2003.1199473.

[11] F.Meyer, *Integrals, gradients and watershed lines*, in Proceedings of Mathematical Morphology and its Applications to Signal Processing, 1993, pp. 70–75.

[12] L. Gillibert and D. Jeulin, *Stochastic multiscale segmentation constrained by image content*, in International Conference on Mathematical Morphology and its Application to Image and Signal Processing, 2011, pp. 132–142. https://doi.org/10.1007/978-3-642-21569-8_12.

[13] C. Kauffman and N. Piche, *Cellular automaton for ultra-fast watershed transform on GPU*, in International Conference on Pattern Recognition, 2008. https://doi.org/10.1109/icpr.2008.4761628.

[14] J. Kolomazník, *Interactive Processing of Volumetric Data*, PhD thesis, Faculty of mathematics and physics, Charles University, Prague, 2017.

[15] F. Maisonneuve, *Sur le partage des eaux*, tech. report, Centre for Mathematical Morphology, 1982.

[16] A. Meijster and J.B. Roerdink, *A proposal for the implementation of a parallel watershed algorithm*, in International Conference on Computer Analysis of Images and Patterns, 1995, pp. 790–795. https://doi.org/10.1007/3-540-60268-2_382.

[17] F. Meyer, *Un algorithme optimal de ligne de partage des eaux*, in Proceedings of Huitième Congrès AFCET, 1991, pp. 847–859.

[18] ——, *Topographic distance and watershed lines*, Signal Processing, 38 (1994), pp. 113–125. https://doi.org/10.1016/0165-1684(94)90060-4.

[19] ——, *The steepest watershed: from graphs to images*, tech. report, arXiv preprint, 2012. https://arxiv.org/abs/1204.2134.

[20] ——, *A watershed algorithm progressively unveiling its optimality*, 2015, pp. 717–728. https://doi.org/10.1007/978-3-319-18720-4_60.

[21] G. Milman, A. Kogan, Y. Lev, V. Luchangco, and E. Petrank, *BQ: A lock-free queue with batching*, in ACM Symposium on Parallelism in Algorithms and Architectures, 2018. https://doi.org/10.1145/3210377.3210388.

[22] L. Najman and M. Schmitt, *Definition and some properties of the watershed of a continuous function*, in Proceedings of Mathematical Morphology and its Applications to Signal Processing, 1993, pp. 75–81.

[23] D. Noguet, *A massively parallel implementation of the watershed based on cellular automata*, in International Conference on Application-Specific Systems, Architectures and Processors, 1997. https://doi.org/10.1109/asap.1997.606811.

[24] J.B.T.M. Roerdink and A. Meijster, *The watershed transform: Definitions, algorithms and parallelization strategies*, Fundamenta informaticae, 41 (2000), pp. 187–228. https://doi.org/10.3233/FI-2000-411207.

[25] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, New York: Academic, 1982. https://doi.org/10.1016/c2009-0-21955-6.

[26] A. Rosenfeld and J. Pfaltz, *Sequential operations in digital picture processing*, Journal of the ACM, 13 (1966), pp. 471–494. https://doi.org/10.1145/321356.321357.

[27] R.E. Tarjan, *Efficiency of a good but not linear set union algorithm*, Journal of the ACM, 22 (1975), pp. 215–225. https://doi.org/10.1145/321879.321884.