Pixel Approximation Errors in Common Watershed Algorithms

Hans Meine[1], Peer Stelldinger[1], and Ullrich Köthe[2]

[1] Cognitive Systems Laboratory, University of Hamburg, Germany
[2] Heidelberg Collaboratory for Image Processing, University of Heidelberg, Germany

Abstract. The exact, subpixel watershed algorithm delivers very accurate watershed boundaries based on a spline interpolation, but is slow and only works in 2D. On the other hand, there are very fast pixel watershed algorithms, but they produce errors not only in certain exotic cases, but also in real-world images and even in the most simple scenarios. In this work, we examine closely the source of these errors and propose a new algorithm that is fast, approximates the exact watersheds (with pixel resolution), and can be extended to 3D.

1 Introduction

Watershed algorithms are among the most important approaches to image segmentation. This is in large part due to the fact that they are based on a sound mathematical definition of segmentation boundaries: watershed segmentations are equivalent to Voronoi tessellations of a suitable boundary indicator function (e.g. the image gradient magnitude) with respect to the topographic distance [1]. That is, catchment basins (regions) are defined as the set of points which are closer (in topographic distance) to a particular minimum than to any other minimum, and watersheds form the boundaries between the basins.

The *exact watershed algorithm* [2,3] detects such boundaries with an accuracy that is only limited by adjustable numerical tolerances and the quality of the input image. The results are very precise and achieve theoretically predicted limits under realistic image acquisition models [4]. Unfortunately, the exact watershed algorithm is also rather time consuming, and its computational principle is restricted to two dimensions.

In contrast, the popular pixel-based watershed algorithms are much faster, and can be applied in any dimension, but these advantages come at the cost of much lower accuracy: pixel-based algorithms only produce inter-pixel boundaries or 8-connected pixel boundaries, i.e. they round boundary locations to certain fixed coordinates defined by the grid. Furthermore, loss of accuracy is not only observed in terms of geometry (i.e. displacement of boundaries from their true position), but also in terms of topology: it is not uncommon that some catchment basins are missing or are split into several regions. Furthermore, there are systematic errors that lead to watersheds snapping into "preferred" directions.

In this paper, we are going to compare pixel-accurate results to the corresponding exact watersheds and, for the first time, investigate systematically the

S. Brlek, C. Reutenauer, and X. Provençal (Eds.): DGCI 2009, LNCS 5810, pp. 193–202, 2009.

errors caused by common watershed discretization schemes. This investigation reveals a number of interesting findings:

- Correct detection of minima (i.e. catchment basin seeds) in discrete images is much more difficult than commonly believed.
- Grid-based flowline discretization suffers from accumulated quantization errors leading to systematic errors in the boundaries, i.e. "preferred directions".

We propose a compromise between the exact and pixel-accurate algorithms, which we call the *fast flowline algorithm*. While our new algorithm only computes the assignment of each pixel to a catchment basin (instead of a subpixel-accurate boundary), it does so by tracing the *exact, subpixel-accurate flowline* starting at each pixel center. In this way, the topological inconsistencies described above can be avoided, and the approximation effect indeed reduces to boundary rounding. This principle is applicable in higher dimensions as well, and the boundary can be refined by adaptive subdivision (i.e. use of a finer grid near the boundary) if desired. The algorithm's speed is significantly increased by early flowline termination. This optimization resembles the UNION-FIND variety of the watershed transform, but instead of tracing flowlines only to an immediate neighbor, we follow them for as long as is necessary for a correct decision about basin assignment.

2 The Exact Watershed Transform

The exact watershed transform assumes that a continuous boundary indicator image is given, i.e. an image that takes high values near boundaries and low values within regions, and which is defined over a compact subset D of $\mathbb{R}^2$. The latter assumption is not a significant restriction for image analysis, because band-limited images of standard quality can be accurately interpolated to the real domain, e.g. by spline interpolation. In addition, we assume that the boundary indicator function fulfills a Morse condition in the domain of interest, i.e. all points with zero gradient are isolated, so that the image has only extrema and saddles, but no plateaus and horizontal ridges (this is a sufficient, but not a necessary condition which we pose for convenience of exposition). In contrast to discrete watersheds, where the plateau problem is central [1], it is not very significant in interpolated images because band-limited functions can only violate the Morse condition in degenerate cases. Moreover, since the space of Morse functions is dense in the space of continuous functions, an infinitesimal change of the pixel values (e.g. by adding noise with standard deviation of a fraction of a gray level, and/or application of an infinite impulse response filter) would turn a non-Morse function into a Morse function.[1]

Due to the above assumptions, every point (with the exception of extrema and saddles) is crossed by a unique flowline which runs along the local gradient

[1] We always imply that pixel values are stored in a floating point representation during analysis.

direction, and all flowlines converge to and diverge from a critical point. With reflective boundary conditions, this even applies at the image border. The *catchment basin* of a minimum is now the set of points whose flowlines converge at this minimum, and watersheds are those points that do not belong to any catchment basin. This definition is the basis for the flowline (or "tobogganing" [5]) type of watershed algorithms.

Equivalently, one can base the watershed definition on the *topographic distance* [1] between two points $\boldsymbol{x}$ and $\boldsymbol{y}$

$$d_t\left(\boldsymbol{x},\boldsymbol{y}\right) = \min_C \int_C |\nabla f(\boldsymbol{z})|\, d\boldsymbol{z} \tag{1}$$

where f is the boundary indicator function, and the minimum is taken over all paths C connecting $\boldsymbol{x}$ and $\boldsymbol{y}$. When $\boldsymbol{x}$ and $\boldsymbol{y}$ are on the same flowline, that flowline is necessarily a path of smallest distance, and the topographic distance reduces to $d_t\left(\boldsymbol{x},\boldsymbol{y}\right) = |f(\boldsymbol{x}) - f(\boldsymbol{y})|$. In particular, when $\boldsymbol{x}$ is a minimum point, $d_t\left(\boldsymbol{x},\boldsymbol{y}\right) + f(\boldsymbol{x}) = f(\boldsymbol{y})$ holds for all points $\boldsymbol{y}$ whose flowline converges to that minimum.

Suppose for a moment that all minima of f have the same value $f_{\min}$ (when this is not the case, one may change f in an infinitely small neighborhood around each minimum to fulfill the condition). Let $\{\boldsymbol{x}_i\}$ be the set of minima. Then we can define the Voronoi tessellation of f with respect to its minima as

$$r_i = \{\boldsymbol{y} \in D \,|\, \forall j \neq i:\ d_t\left(\boldsymbol{x}_i,\boldsymbol{y}\right) < d_t\left(\boldsymbol{x}_j,\boldsymbol{y}\right)\} \tag{2}$$

where r_i denotes the Voronoi region ("catchment basin") around minimum i. This is the same definition as for standard Voronoi tessellation, except that the Euclidean distance has been replaced with the topographic distance. Watersheds are now precisely the points in D whose distance to at least two minima is equal. When the restriction that all minima have the same value is dropped, we get the same tessellation as before by replacing (2) with

$$r_i = \{\boldsymbol{y} \in D \,|\, \forall j \neq i:\ d_t\left(\boldsymbol{x}_i,\boldsymbol{y}\right) + f(\boldsymbol{x}_i) < d_t\left(\boldsymbol{x}_j,\boldsymbol{y}\right) + f(\boldsymbol{x}_j)\} \tag{3}$$

This definition is the basis for the flooding variety of watershed algorithms [6].

Yet another way to define a watershed segmentation is via the watersheds themselves: as Maxwell noted, watersheds are precisely the flowlines that converge to saddle points of the boundary indicator. In the continuous domain, all three definitions are equivalent. However, in practice, the last definition is the best choice when geometric and topological accuracy is of major concern. This so called the *exact watershed algorithm* [2,3] can be described as follows:

1. Use Newton-Raphson iterations to find all saddle points of f.
2. For every saddle $\boldsymbol{s}$ (optionally only for those that fulfill some application specific predicate, such as sufficient edge strength), solve the inverse flowline PDE

$$\frac{\partial \boldsymbol{x}\left(t\right)}{\partial t} = \nabla f(\boldsymbol{x}\left(t\right)) \tag{4}$$

   starting in the two directions of positive curvature until converging to a maximum (e.g. using the second-order Runge-Kutta method).

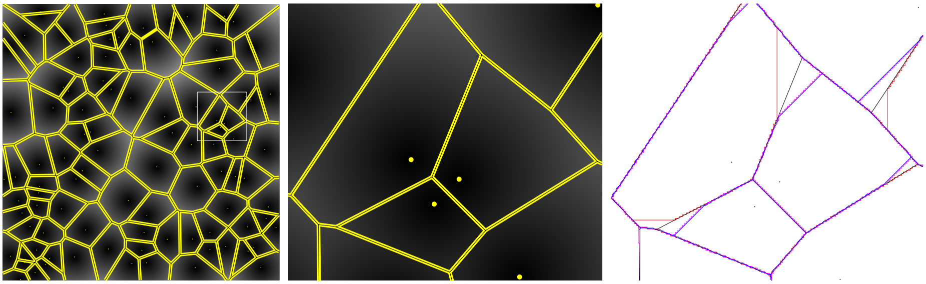


Fig. 1. Comparison of the results of the exact watershed algorithm (*black, thin*) with the known ground truth (*yellow, thick*) of a Euclidean distance transform – *middle:* close-up of ROI – *right:* pixel-based results with same colors as in Fig. 3

3. Connect all such flowline pairs into edges of a graph whose nodes consist of maxima of f.
4. Determine the cyclic order of edges around nodes (taking their tangential convergence into account) in order to reconstruct the graph's faces (i.e. the catchment basins) by contour traversal [3].

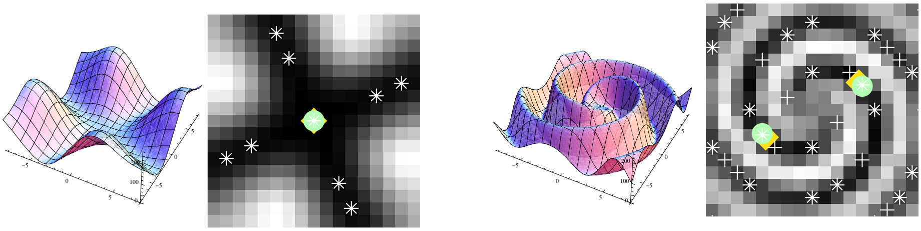
Experiments show that this algorithm produces very accurate watershed segmentations that are close to the ground truth (if available), cf. Fig. 1.

3 Error Analysis of Common Watershed Algorithms

Discrete watershed algorithms only determine an assignment of pixels (i.e. points at integer coordinates) to catchment basins. Since there are only a finite number of these points, algorithms of the flowline or flooding variety are most common. Often, these algorithms are defined without reference to continuous watershed definitions. In contrast, we interpret them as discretizations of the corresponding continuous definitions in order to better understand their properties and differences. Discretization then boils down to two questions: i) which discrete approximations of flowlines (or inverse flowlines for flooding) are used, and ii) how are minima detected?

The latter is usually done by detecting minima in a 4- or 8-neighborhood; however, this misses some real minima, and many false minima are detected. For instance, Fig. 2 shows the minima detected on digital images generated by sampling two test functions with known ground truth minima. Every false minimum gives rise to a false catchment basin.

The first function exhibits a cross-like arrangement of four valleys and is given by $10(\mathrm{s}(\frac{u}{8}) + \mathrm{s}(\frac{v}{8})) + 164\mathrm{s}(\frac{u}{4})\mathrm{s}(\frac{v}{4})$ with $u = \sin\frac{\pi}{8}x + \cos\frac{\pi}{8}y$, $v = \cos\frac{\pi}{8}x - \sin\frac{\pi}{8}y$ and $s(t) = 1 - \frac{\sin \pi t}{\pi t}$. The plot shows the window $-8 \leq x, y \leq 8$. Sampling at integer coordinates results in an oversampling factor of 2.8 with regard to the bandlimit of the function. There is only one minimum in the origin, which has



◆: ground-truth minima; ●: minima of spline-interpolated image (order 5)
+/✳: minima in 4/8-neighborhood, respectively (ranges shown: $x, y \in [-8 \ldots 8]$)

Fig. 2. Prototypical spurious results of discrete minima detection. The ground truth is only reproduced by subpixel-accurate minima detection in a spline-interpolated image.

been correctly found by all methods. Due to the discretization, the pixel-accurate methods detect 8 wrong minima, inside the slowly decreasing valleys.

The second function shows a spiral having two minima at $(2.1904, -3.7276)$ and $(-2.1904, 3.7276)$, two maxima at $(-2.1904, -3.7276)$ and $(2.1904, 3.7276)$ and a saddle point at the origin. There are two spiral-shaped catchment basins. Here, the sampling preserves 99.98% of the energy of the function and is thus effectively band-limited.

Furthermore, algorithms also differ in how they approximate flowlines with respect to their location and direction:

1. True flowlines may be rounded to grid-based paths, usually connecting pixel centers according to 4- or 8-connectivity or (at best) using adaptive support points on grid lines. Pixel paths have the obvious speed advantage that each path can be terminated after only one step (i.e. in a neighboring pixel) when basins are constructed incrementally.
2. The flowline direction may locally be derived from different gradient approximations (e.g. using a Gaussian or Sobel kernel) or even from simple forward differences (i.e. looking for the smallest neighbor [5]).

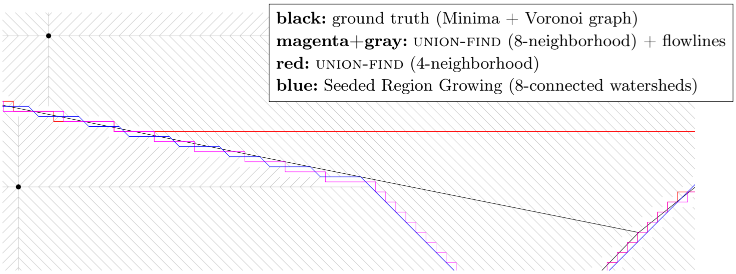

black: ground truth (Minima + Voronoi graph)
magenta+gray: UNION-FIND (8-neighborhood) + flowlines
red: UNION-FIND (4-neighborhood)
blue: Seeded Region Growing (8-connected watersheds)

Fig. 3. Systematic errors due to discrete flowline directions (cf. Fig. 1)

However, these discrete flowline approximations suffer from systematic errors, as can be seen in Fig. 3. These errors become most apparent when watersheds suddenly turn into a preferred direction (parallel to the principal axes or diagonals, depending on the neighborhood). This phenomenon often affects large regions because once a pixel is assigned to the wrong basin, all approximate flowlines converging to this pixel are wrongly assigned as well.

4 Subpixel Flowline Algorithms

As shown in the previous section, common grid-based flowline discretization schemes lead to discretization errors. These can only be fixed by lifting the restriction that flowline sampling points coincide with grid points. Interestingly, an algorithm realizing this idea already exists, although its close relationship to the watershed algorithm is not immediately obvious: the *mean-shift algorithm.*

4.1 Mean-Shift as a Watershed Algorithm

The mean-shift algorithm [7] is a popular segmentation algorithm whose results are similar to watersheds, but with markedly less oversegmentation. We show in the sequel that the latter is not a consequence of a different algorithmic approach, but stems solely from the differences in the boundary indicator definition. Indeed, one trivial difference is that mean-shift does not use a boundary indicator, but a region indicator (i.e. a measure of homogeneity rather than edgeness). Accordingly, when searching for region centers (catchment basins), one looks for local maxima of the region indicator instead of local minima of the boundary indicator, as in the watershed algorithm. That is, one traces flowlines upwards instead of downwards. But this difference is easily eliminated by inversion of the indicator function.

A more interesting difference is found in the way the indicator function itself is defined. In the watershed context, we simply assume that the values of the indicator function are given at the grid points. The subpixel watershed algorithm then interpolates these values into a smooth function over $\mathbb{R}^2$ (in order for the flowline differential equation (4) to be well-defined), whereas the pixel-based watershed variants use nearest-neighbor interpolation or, equivalently, interpret the grid as a 4- or 8-connected graph. In contrast, the indicator function of the mean-shift algorithm is implicitly defined by a *kernel density estimation* in the combined domain of the spatial coordinates and corresponding feature values. In particular, the region probability of the point at $\boldsymbol{x}$ is defined as

$$\mathbb{P}\left(\boldsymbol{x} \text{ is region}\right) = \mathbb{P}\left(\boldsymbol{x} \text{ is region } |\mathcal{N}\left(\boldsymbol{x}\right)\right) = \frac{1}{C} \sum_{\boldsymbol{y} \in \mathcal{N}(\boldsymbol{x})} \mathbb{P}\left(\boldsymbol{x} \text{ is region } |\boldsymbol{y}, f\left(\boldsymbol{y}\right)\right)$$

where $\mathcal{N}\left(\boldsymbol{x}\right)$ denotes some neighborhood of $\boldsymbol{x}$, C is a normalization constant, and $f\left(\boldsymbol{y}\right)$ is the feature vector at point $\boldsymbol{y}$ (e.g. its RGB color). The probability

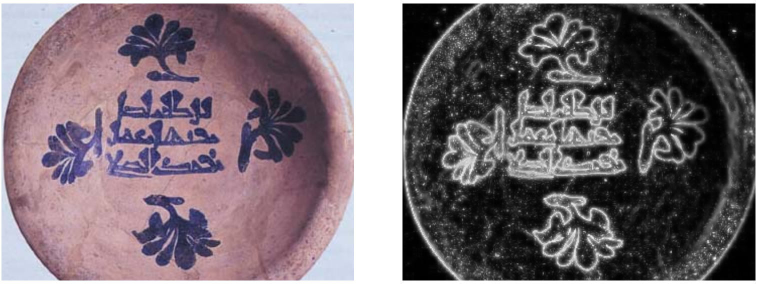


Fig. 4. Example image and illustration of $\mathbb{P}(\boldsymbol{x} \text{ is boundary}) = 1 - \mathbb{P}(\boldsymbol{x} \text{ is region})$, i.e. the boundary indicator implicitly used by the mean-shift algorithm (parameters used: $\sigma_{\text{spatial}} = 2$, $\sigma_{\text{rgb}} = 30$)

$\mathbb{P}(\boldsymbol{x} \text{ is region } | \boldsymbol{y}, f(\boldsymbol{y}))$ is expanded as a product of spatial similarity ("nearness") between $\boldsymbol{x}$ and $\boldsymbol{y}$, and feature similarity between the values at $\boldsymbol{x}$ and $\boldsymbol{y}$:

$$\mathbb{P}(\boldsymbol{x} \text{ is region } | \boldsymbol{y}, f(\boldsymbol{y})) = \mathbb{P}(|\boldsymbol{x} - \boldsymbol{y}|)\, \mathbb{P}(|f(\boldsymbol{x}) - f(\boldsymbol{y})|) \tag{5}$$

In other words, $\boldsymbol{x}$ is considered part of a region when its neighboring points have similar feature vectors, which is clearly a measure of feature homogeneity. In the framework of kernel density estimation, the probabilities in (5) are expressed by means of kernel functions which give high values when their argument (i.e. the distance in the spatial or feature domain) is small. A popular kernel choice is the *Gaussian kernel* $\mathbb{P}_\sigma(d) = \frac{1}{C(\sigma)} \exp\left(-\frac{d^2}{2\sigma^2}\right)$ where $C(\sigma)$ is a normalization constant depending on σ and the dimension of the space where d is defined. In effect, the spatial probability defines a Gaussian window around $\boldsymbol{x}$, and the feature probability measures the degree of similarity between $\boldsymbol{x}$ and the points in this window[2]. The width of the spatial kernel is usually set to a few pixel distances, whereas the width of the feature kernel equals the noise standard deviation in the data. It should be noted that the coordinate $\boldsymbol{x}$ is *not* restricted to lie on the grid, even if all points in $\mathcal{N}(\boldsymbol{x})$ must be on the grid. Therefore, $\mathbb{P}(\boldsymbol{x} \text{ is region})$ is defined over all of $\mathbb{R}^2$, similarly to the spline-interpolated boundary indicator we are using in the exact watershed algorithm. In fact, spline interpolation is a special case of kernel-based interpolation, where the kernel is a cardinal spline function instead of a Gaussian.

The similarity between watersheds and mean-shift is not readily visible because the kernel density estimates $\mathbb{P}(\boldsymbol{x} \text{ is region})$ are never explicitly computed in the algorithm, for reasons of efficiency. However, there is no theoretical obstacle for doing so, and Fig. 4 (right) shows an example for an inverted region indicator. This image very much resembles the typical boundary indicators that are used in watershed algorithms.

[2] In practice, one truncates the spatial-domain Gaussian $\mathbb{P}(|\boldsymbol{x} - \boldsymbol{y}|)$ at some maximal distance d_0 in order to avoid infinite window sizes.

Instead of computing $\mathbb{P}(\boldsymbol{x} \text{ is region})$, the mean-shift algorithm directly computes an incremental flowline. Let $\boldsymbol{z} = (\boldsymbol{x} \quad f(\boldsymbol{x}))^\top$ denote a point in the combined domain, i.e. the coordinate vector and feature vector stacked on top of each other, and $\boldsymbol{z}^{(t)}$ such a vector at time step t. Then the flowline is computed via iterations[3]

$$\boldsymbol{z}^{(t+1)} = \boldsymbol{z}^{(t)} + \frac{\sum_{\boldsymbol{z}' \in \mathcal{N}\left(\boldsymbol{z}^{(t)}\right)} \boldsymbol{z}' \, \nabla\mathbb{P}\left(\boldsymbol{z}^{(t)} \text{ is region } |\boldsymbol{z}'\right)}{\sum_{\boldsymbol{z}' \in \mathcal{N}\left(\boldsymbol{z}^{(t)}\right)} \mathbb{P}\left(\boldsymbol{z}^{(t)} \text{ is region } |\boldsymbol{z}'\right)} \tag{6}$$

These iterations converge toward a local maximum of the probability, and all points $\boldsymbol{x}$ whose flowlines converge at the same maximum are considered as one region. This procedure immediately turns into a flowline-type watershed algorithm when the probability is inverted and flowlines are traced downwards to minima. Equation (6) is simply a first order approximation of the flowline equation (4) with $f(\boldsymbol{x})$ replaced by $\mathbb{P}(\boldsymbol{z} \text{ is region } |\mathcal{N}(\boldsymbol{z}))$. Since $\boldsymbol{z}^{(t)}$ is not restricted to the grid for $t > 0$, the mean-shift procedure provides more accurate flowline approximations than purely grid-based algorithms. Still, it is not quite as accurate as the Runge-Kutta flowline tracing employed in the exact watershed algorithm, because it lacks adaptive step length control. This means that mean-shift iterations occasionally converge to a maximum far from the original point.

4.2 The Fast Flowline Algorithm

Our new watershed algorithm is based on three considerations:

1. Since the computation of exact watersheds is expensive, we only determine an assignment of pixels to catchment basins, i.e. an inter-pixel boundary like in pixel-based watershed algorithms. (Boundary accuracy can be improved by refining the grid, possibly only near boundaries.)
2. As in the mean-shift algorithm, we compute flowlines and their points of convergence with subpixel accuracy using the flowline equation (4).
3. Since the computation of complete flowlines for all pixels would be even more expensive than the computation of exact watersheds, we introduce an early stopping criterion that provides correct region assignments after only a few tracing steps.

The only missing ingredient of the new algorithm is the early stopping criterion. The underlying idea is very simple: stop flowline tracing as soon as the flowline crosses a grid line whose end-points are already assigned to the same catchment basin. In order for this criterion to lead to *early* stopping, we have to process the pixels in a clever order: pixels close to minima must be investigated first, so that areas of already assigned pixels spread around each minimum. This is exactly the same idea as in the usual region-growing algorithm, with one important exception: in our approach, the initial seeds need *not* be true minima of the

[3] Notice that $\mathbb{P}(\boldsymbol{x} \text{ is region } |\mathcal{N}(\boldsymbol{x})) = \mathbb{P}(\boldsymbol{z} \text{ is region } |\mathcal{N}(\boldsymbol{z}))$ since $f(\boldsymbol{x})$ is a deterministic function of $\boldsymbol{x}$. The introduction of $\boldsymbol{z}$ just serves notational convenience.

boundary indicator, because seeds only serve to speed up flowline computations. Early flowline termination will simply succeed more often when the initial seeds are close to true minima. But the choice of seeds (be it good or poor) will have no influence on the actual region assignments, because those assignments only depend on the point of convergence of the flowlines.

Given a boundary indicator function f, we define the *fast flowline algorithm* as follows:

1. Find minima of f (e.g. in a 4-neighborhood) and put them into a priority queue which assigns highest priority to the point with minimal $f(\boldsymbol{x})$.
2. While the queue is not empty:
   (a) Pop the next point from the queue and trace its flowline by Runge-Kutta iterations with adaptive step size control according to the flowline equation
   $$\frac{\partial \boldsymbol{x}\,(t)}{\partial t} = -\nabla f(\boldsymbol{x}\,(t))$$
   Stop the iteration if either
      i. $\boldsymbol{x}\,(t \to \infty)$ converges at a minimum. When this minimum was already labeled, assign this label to the starting point $\boldsymbol{x}\,(0)$. Otherwise, create a new label and use that for both $\boldsymbol{x}\,(0)$ and the minimum.
      ii. If the path defined by $\boldsymbol{x}\,(t)$ crosses a grid line (i.e. the connection between two neighboring pixels) whose end points have already been assigned to the same label. Assign this label to $\boldsymbol{x}\,(0)$.

   (b) Put the neighbors of the just labelled point into the queue.

Notice that early termination will fail (i.e. will assign the starting point to the wrong region) if the flowline makes a sharp turn *after* the termination point and crosses the grid line again in opposite direction. Fortunately, the smoothness of real boundary indicators ensures that such errors occur very rarely.

4.3 Experimental Results

In order to aid the comparison of the performance of different watershed algorithms, we decided to use two sets of test images: first, we computed Gaussian gradient magnitude images (with $\sigma \in \{1, 1.5, 2, 3\}$) from a set of images with various real-world image analysis tasks. Here, the ground truth is unknown, and we use the exact watershed transform (with a spline interpolation of order 5) as the gold standard to compare against. Second, we generated images containing the Euclidean distance transform of randomly placed, isolated minimum points (cf. Fig. 1 on page 196) with varying density, on integer and sub-pixel positions. For the latter images, we could compute the correct catchment basins (i.e. the Voronoi tessellation w.r.t. the minima) for error analysis.

For every label image generated with one of the watershed algorithm variants, we performed a pixel-wise comparison of the labels with the reference image (after an optimal mapping of the labels using the stable marriage algorithm). The following table summarizes the results of more than 2000 single experiments:

	real-world images		distance transforms	
	mean err.	std.dev.	mean err.	std.dev.
subpixel watersheds	(gold standard)		0.23%	0.24%
Seeded Region Growing	12.8%	2.2%	1.25%	0.93%
UNION-FIND (4-neighborhood)	15.7%	3.9%	1.6%	1.0%
UNION-FIND (8-neighborhood)	10.3%	3.0%	1.8%	0.9%
flowline watersheds	0.25%	0.5%	0.18%	0.22%
fast flowline watersheds	0.28%	0.23%	0.19%	0.23%

5 Conclusions

Our experimental results show that subpixel-accurate algorithms do not suffer from the systematic errors of common grid-based watershed implementations: their errors are roughly an order of magnitude lower. However, they are also an order of magnitude slower (the “flowline watersheds” algorithm *without* early flowline termination even two orders of magnitude). Our current implementation of the fast flowline algorithm does not perform much faster than the subpixel watershed algorithm, but there is substantial potential for optimization, mainly because the time consuming detection of critical points is avoided completely.

A major advantage of the fast flowline algorithm is that it can be applied to 3D and higher dimensions, whereas the exact watershed algorithm is restricted to 2D. Our algorithm is therefore the first 3D watershed algorithm that avoids the systematic errors of grid-based approaches. Furthermore, our algorithm allows to further increase the segmentation accuracy by adaptive grid subdivision near boundaries. Our future work will explore these directions. At the same time, we will search for a faster (but equally accurate) alternative to the Runge-Kutta method for flowlines tracing.

References

1. Roerdink, J.B.T.M., Meijster, A.: The watershed transform: Definitions, algorithms, and parallellization strategies. In: Goutsias, J., Heijmans, H. (eds.) Mathematical Morphology, vol. 41, pp. 187–228. IOS Press, Amsterdam (2000)
2. Meine, H., Köthe, U.: Image segmentation with the exact watershed transform. In: Villanueva, J. (ed.) Proc. Vis., Imaging, and Image Processing, pp. 400–405 (2005)
3. Meine, H.: The GeoMap Representation: On Topologically Correct Sub-pixel Image Analysis. PhD thesis, Dept. of Informatics, Univ. of Hamburg (2009)
4. Meine, H., Köthe, U., Stelldinger, P.: A topological sampling theorem for robust boundary reconstruction and image segmentation. Discrete Applied Mathematics 157, 524–541 (2008) DGCI special issue
5. Yao, X., Hung, Y.P.: Fast image segmentation by sliding in the derivative terrain. In: Casasent, D.P. (ed.) Intelligent Robots and Computer Vision X: Algorithms and Techniques. SPIE Conference Series, vol. 1607, pp. 369–379 (1991)
6. Vincent, L., Soille, P.: Watersheds in digital spaces: an efficient algorithm based on immersion simulations. IEEE T-PAMI 13, 583–598 (1991)
7. Comaniciu, D., Meer, P.: Mean shift: A robust approach toward feature space analysis. IEEE T-PAMI 24, 603–619 (2002)