IMAGE SEGMENTATION WITH THE EXACT WATERSHED TRANSFORM

Hans Meine and Ullrich Köthe
Cognitive Systems Group, University of Hamburg, Germany
{meine|koethe}@informatik.uni-hamburg.de

Abstract

Discrete algorithms for low-level boundary detection are geometrically inaccurate and topologically unreliable. Corresponding continuous methods are often more accurate and need fewer or no heuristics. Thus, we transfer discrete boundary indicators into a continuous form by means of differentiable spline interpolation and detect boundaries using the *exact* watershed transform. We demonstrate that this significantly improves the obtained segmentations.
keywords: sub-pixel segmentation, watersheds, splines

1 Introduction

Edge detection and segmentation are fundamental tasks in image analysis. But conventional algorithms often miss parts of the true boundary or hallucinate non-existing boundaries. Usually, these errors are attributed to noise and low contrast, but this cannot be the whole truth: Errors occur even under good conditions, mainly near junctions and in areas with small or narrow objects. In this paper we will investigate how a *continuous approach* helps in overcoming these problems. We consider low-level boundary detection as a four-step process, whose most prominent example is *Canny's algorithm* [1]:

1. A *boundary indicator* extracts a measure of boundary strength from an image, e.g. the gradient magnitude.
2. During *boundary detection* (*"non-maxima suppression"*), boundary points are extracted from the pixel-wise output of the boundary indicator.
3. These points are connected into a complete boundary graph during the *boundary linking* stage.
4. Suitable *relevance filtering* (e.g. hysteresis thresholding) is applied to remove insignificant boundaries and create a perceptually meaningful result.

It is important to note that these steps can be varied relatively independently of each other, provided that suitable interfaces are defined. In Canny's algorithm, step 2 usually achieves localization accuracy significantly below the pixel size, but step 3 is less satisfying, as it heavily relies on heuristics and does not always lead to topologically consistent results. In this paper we will employ the *exact watershed transform* for steps 2 and 3.

In the context of the watershed transform, we think of boundaries as ridges of the boundary indicator function. The formal definition of watershed ridges was given by Maxwell [5]: A watershed is a flowline that connects a local maximum with a saddle point, where a flowline is defined as the curve along which a drop of water runs downward along the local gradient. This definition has two drawbacks: First, one cannot decide locally whether a given flowline ends at a saddle, i.e. whether it is indeed a watershed. Second, flowlines are only uniquely defined when the function is differentiable and all critical points are isolated.

To avoid the first problem, alternative ridge definitions have been proposed that can be evaluated independently at every pixel. Canny's local maxima along the gradient direction and the zero crossings of the Laplacian are two common examples. A comparison of alternatives can be found in [2]. Unfortunately, it turns out that local definitions only work well for 1-dimensional ridges, but fail near junctions. In contrast, watersheds can represent junctions without difficulties, because arbitrary many watershed lines may meet at a maximum. This makes the watershed transform preferable for an integrated boundary detection algorithm that handles edges, junctions and regions simultaneously and consistently.

The second problem is generally solved by replacing Maxwell's definition with some discrete analog. The most common one restricts flowlines to a grid, i.e. water can only flow horizontally and vertically, possibly also diagonally, see [7] for a survey of different algorithms. Discrete watershed algorithms are fast and relatively simple, and produce consistently linked boundary graphs that include both edges and junctions. However, watersheds computed on the grid differ significantly from their continuous counterparts. Not only is the geometry restricted to pixel accuracy, but one commonly encounters missed or hallucinated edges and junctions because pixels in only a 4- or 8-neighborhood cannot reveal the geometry of the boundary indicator with sufficient certainty.

In this paper we solve both problems by applying the watershed algorithm in the *continuous domain*, i.e. on a differentiable spline interpolation of the boundary indicator. To our knowledge, this has not been done before in the context of image segmentation. We show that the new approach has two important advantages: It keeps the topological advantages of the watershed transform (closed contours, arbitrary junctions without heuristics), and combines it with the high subpixel accuracy of Canny's algorithm. We also show that subsequent relevance filtering (in a way very similar to Canny's algorithm) can remove the oversegmentation typically associated with the watershed algorithm.

2 The Exact Watershed Transform

In order to detect watersheds with subpixel accuracy we must start with Maxwell's original definition: watersheds are flowlines between maxima and saddles. Practical application of this definition is simplified if the function f is differentiable. Then a unique flowline exists at every point with non-zero gradient, and flowlines can be traced (upwards) by integrating the following differential equation:

$$\frac{\partial \vec{x}(t)}{\partial t} = \nabla f(\vec{x}(t)) \tag{1}$$

Numerical integration of (1) is stable near a watershed, because all flowlines in a neighborhood converge to the same maximum. Matters simplify further if f is a *Morse function* [11], i.e. it is at least twice differentiable, and all critical points (points of zero gradient) are isolated and have non-degenerate Hessian. This eliminates special cases such as monkey saddles and plateaus and allows to classify critical points according to the signs of the Hessian's eigenvalues: minima, maxima, and saddles have positive, negative and mixed eigenvalues respectively.

For a long time, watershed algorithms based on these assumptions have been considered as too expensive for image analysis. However, Steger [9] demonstrated the opposite by implementing a subpixel watershed algorithm for digital elevation maps, which we adapt and enhance for image segmentation. Key to his approach is the possibility to adaptively sample the image at any required (subpixel) location by means of an efficient spline interpolation. In the present paper, we apply this idea to boundary indicator functions. In addition to cubic splines as in [9], we also investigate splines of order 2, 5 and 7. Splines of order n possess $n-1$ continuous derivatives and can be efficiently computed at any location $\vec{x} = (x, y)$ by convolution of discrete spline coefficients c_{ij} with continuous B-spline basis functions β_n:

$$f(x,y) = \sum_{i,j} c_{ij}\, \beta_n(i-x)\, \beta_n(j-y) \tag{2}$$

The coefficients c_{ij} depend on the order n of the spline and can be computed from the sampling values f_{ij} by a cascade of $\lfloor n/2 \rfloor$ first-order recursive filters. Details on these computations can be found in [10]. In order for the spline to be a good approximation of the continuous boundary indicator, the sampling density of the f_{ij} must be sufficiently high. For small scales (up to $\sigma \approx 1$), the boundary indicators must be computed with oversampling filters [3].

Next, we must determine the saddles and maxima of the spline. There are two principle methods. First, we could compute the gradient of (2) and set it to zero. For every spline facet, this gives a polynomial system in two variables that can be solved by standard methods. Unfortunately, the number of possible solutions per facet grows as $2n(n-1)+1$ which quickly becomes impractical, given that only very few solutions (at most 2 or 3 per facet) do actually represent critical points of $f(\vec{x})$. Therefore, an iterative algorithm is more adequate. Consider the second order Taylor series expansion of $f(\vec{x})$ around a point $\vec{x}_0$:

$$\begin{aligned} f(\vec{x}_\mathrm{o} + \vec{x}) &= f(\vec{x}_0) + \nabla f(\vec{x}_0)(\vec{x} - \vec{x}_0) + \\ &\quad \frac{1}{2}(\vec{x} - \vec{x}_0)^T \mathbf{H}(\vec{x}_0)(\vec{x} - \vec{x}_0) \end{aligned} \tag{3}$$

where $\mathbf{H}(\vec{x}_0)$ is the Hessian. When $\vec{x}_0 + \vec{x}$ is a critical point (saddle or extremum), the gradient w.r.t. $\vec{x}$ of this expression must be zero:

$$\nabla f(\vec{x}_0 + \vec{x}) = \nabla f(\vec{x}_0) + \mathbf{H}(\vec{x}_0)(\vec{x} - \vec{x}_0) \stackrel{!}{=} 0 \tag{4}$$

Solving for $\delta\vec{x} = (\vec{x} - \vec{x}_0)$ gives the following iteration scheme:

$$\begin{aligned} \vec{x}^{(n+1)} &= \vec{x}^{(n)} + \delta\vec{x}^{(n)} \\ \delta\vec{x}^{(n)} &= -\mathbf{H}(\vec{x}^{(n)})^{-1}\, \nabla f(\vec{x}^{(n)}) \end{aligned} \tag{5}$$

It rapidly converges to a (usually nearby) critical point. We improved Steger's algorithm in a seemingly simple, but critical respect: Steger applied the above iteration only when a critical point was first detected at pixel accuracy, and also required the iteration to stay within the current pixel. We dropped these restrictions because many critical points were missed this way. To avoid missing points we start iterating at four points within every pixel. Since many points are now found multiple times, we added an efficient algorithm to eliminate multiple detections.

Next, we solve (1) starting at every saddle. Since the gradient at a saddle is zero by definition, we cannot use it for the initial step. Instead, we make use of the fact that the local shape of a non-degenerate saddle is completely determined by the Hessian $\mathbf{H}$. Thus, the proper initial directions are $\frac{\partial \vec{x}}{\partial t}\big|_{t=0} = \pm\tau\vec{e}_1$ where $\vec{e}_1$ is the unit eigenvector corresponding to the positive eigenvalue of $\mathbf{H}$, $\tau = 0.1$ is the initial step size, and the sign selects one of the two opposite watersheds starting at every saddle. Subsequent steps are computed by means of a second-order Runge-Kutta method with adaptive step size control which ensures that a tracing error of 10^{-4} is not exceeded. This translates into a typical step size of about 0.1 pixel. We could increase the step size by using higher-order Runge-Kutta methods, as in [9], but this is undesirable because we want the resulting polygonal arc to be a good approximation of the actual edge, which would no longer be the case with larger steps, especially near corners. Iteration is finished when the curve's distance from the nearest maximum drops below τ.

Besides its high geometric accuracy, this algorithm has a crucial advantage: In contrast to Canny's method, where edgel detection and edgel linking are separate steps, Runge-Kutta flowline tracing connects the detected points into a polygonal arc automatically and without any heuristics. Thus, a large portion of boundary linking (step 3) is already done. Subsequent linking of the arcs into a boundary graph is also simple as their end points are exactly known.

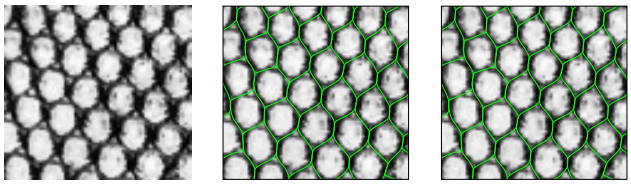

Figure 1. Left to right: Subregion of Brodatz texture D36; watersheds from quadratic and quintic splines. The boundary indicator was $-f \star g_\sigma$, with g_σ a Gaussian and $\sigma = 1.6$. This segmentation was obtained with only two parameters: the Gaussian's scale and the spline order (whose influence is unnoticeable here). Quadratic splines can be used if no critical point falls exactly on a facet border (where the Hessian is discontinuous).

Fig. 1 shows the graph thus obtained for a Brodatz texture image. However, a boundary graph is not yet the final result of boundary linking: In addition to edges and junctions, we also need the faces of the graph, i.e. the regions of its planar embedding. Hence, we transform the graph into a *map* [6]:

Definition 1 *A* map *is a triple* (D, σ, α) *where* D *is a set of* half-edges*, and* σ, α *are permutations on* D*. The orbits (cycles) of* α *must have length 2 and pair half-edges into edges, whereas the orbits of* σ *determine the vertices (and thus the map's connectivity) and the cyclic order of the half-edges around vertices.*

Regions then correspond to the orbits of the combined permutation $\phi = \sigma^{-1}\alpha$. Every flowline between a maximum and a saddle is a half-edge of the map. The α-orbits are obtained by pairing the two half-edges that meet at the same saddle. We also know which half-edges meet at each junction. To obtain the σ-orbits, we must sort these edges so that their order represents a counter-clockwise cycle around the junction. Sorting is complicated by the phenomenon that watersheds often converge tangentially towards the maximum, cf. fig. 2. This is not an artifact of our implementation, but a well-known watershed property [2, 9]. In theory, watersheds do not meet before the maximum, even if they converge tangentially; however, due to the finite accuracy of flowline tracing, the computed polygonal arcs do cross in practice when they are very close to each other. Therefore, one cannot simply use the angles at which flowlines leave the maximum to recover the orientation of the graph. Instead, we apply the following algorithm to determine the ordering at the position where the watersheds eventually diverge:

1. Initialization: To recover the σ-order at junction k located at $\vec{p}_k$: Set the reference point $\vec{p}_{\text{ref}} = \vec{p}_k$, reference angle $\varphi_{\text{ref}} = 0\,\text{rad}$. Let g be the (unordered) set of half-edges starting at $\vec{p}_k$.

2. For each half-edge i in g find the intersection $\vec{p}_i$ of the corresponding polygonal arc with an r-circle around $\vec{p}_{\text{ref}}$ (we use $r = 0.5$). If there are several intersections, select the one whose arc length distance from the half-edge's start is maximum. If there is no intersection, use the half-edge's end point.

3. For each half-edge i in g calculate the angle φ_i between the vector $\vec{p}_i - \vec{p}_{\text{ref}}$ and φ_0, measured in the interval $-\pi < \varphi_i \leq \pi$. Sort g according to φ_i.

4. Detect tangential half-edges: Compute the difference angles $\Delta\varphi_i = \varphi_{i+1} - \varphi_i$. If $\Delta\varphi_i < \varphi_{\text{tang}} = 0.5\,\text{rad}$, half-edges i and $i+1$ are considered tangential. Group the half-edges of g into groups $\hat{g}_m$ such that each group contains a maximal sequence of tangential half-edges. If all groups have only one member, there are no tangential half-edges, and the algorithm stops. Otherwise repeat the procedure for each group with several members: Compute a new reference point $\hat{\vec{p}}_{\text{ref}}$ and reference angle $\hat{\varphi}_0$ as the average of the current group's members, and goto 2. (The choice of r and φ_{tang} is not critical.)

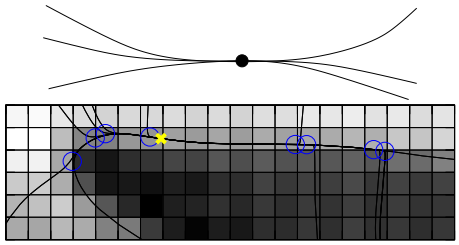

Figure 2. Tangential watershed convergence at a local maximum. Top: illustration; bottom: actual watersheds on an image – the single maximum of the boundary indicator in the ROI is marked yellow, and the blue circles mark locations where the half-edges converge (precisely, the r-circles where the σ-order is eventually found).

In order to analyze region properties (e.g. for relevance filtering), one can employ standard polygon filling procedures on the contours given by the ϕ-orbits; we propose to leave the pixels intersected by the boundary unlabeled as they are not representative for any of the adjacent regions.

3 Boundary Indicator Functions

In this section we define the boundary indicators that we are going to compare in section 4. Ideally, the ridges of a boundary indicator correspond to the edges in the original image, and its local maxima correspond to corners and junctions. The most straightforward choice is the gradient squared magnitude:

$$b_1 = f_x^2 + f_y^2 \quad \text{with} \quad (f_x, f_y) = (g_{x,\sigma}, g_{y,\sigma}) \star f \qquad (6)$$

where $g_{x,\sigma}$ and $g_{y,\sigma}$ are Gaussian derivative filters at scale σ, which also perform oversampling when $\sigma \leq 1$ [3]. However, the gradient squared gives unsatisfying responses near junctions – they don't usually correspond to maxima and can even be minima. Thus, it is customary to integrate edge information over a neighborhood by means of the *structure*

tensor. The trace of this tensor is again a boundary indicator and can be computed by Gaussian smoothing of b_1:

$$b_2 = g_{\sigma'} \star b_1 \tag{7}$$

where $\sigma' \approx 2\sigma$ is the integration scale. This improves the behavior near junctions, but it also smoothes perpendicular to edges. Parallel edges are thus smeared into each other, which reduces the effective resolution of the boundary indicator. This can be avoided by means of an anisotropic filter that only integrates along edges, but not perpendicular to them, e.g. an hour-glass shaped filter [3] defined by:

$$h_{\sigma',\rho}(\vec{x},\vec{n}) = \begin{cases} e^{-\frac{\vec{x}^T\vec{x}}{2\sigma'^2} - \frac{1}{2\rho^2}\left(\frac{\vec{n}_\perp^T\vec{x}}{\vec{n}^T\vec{x}}\right)^2} & \text{if } \vec{n}^T\vec{x} \neq 0 \\ 0 & \text{if } \vec{n}^T\vec{x} = 0,\ \vec{x} \neq 0 \\ 1 & \text{if } \vec{x} = 0 \end{cases} \tag{8}$$

where σ' is the integration scale as before, ρ controls the opening angle of the hour-glass and thus the degree of spatial anisotropy, and $\vec{n}$ is a unit vector determining the filter orientation. In our experiments, we use $\rho = 0.4$ corresponding to an opening angle of $\approx 22°$. At every point, the filter is oriented along the local edge direction. This gives the following filtering equation:

$$b_3(\vec{x}) = \sum_{\vec{x}'} h_{\sigma',\rho}\left(\vec{x} - \vec{x}', \vec{n}(\vec{x}')\right)\ b_1(\vec{x}') \tag{9}$$

where $\vec{n}(\vec{x}')$ is the unit vector perpendicular to the gradient at point $\vec{x}'$. An alternative to edge integration according to (7) or (9) is the extension of the boundary model: The gradient is insufficient to characterize junctions as it is only sensitive to step edges. A possible extension is the *boundary energy* [4]. It can be computed by a family of filters defined in the Fourier domain as:

$$\mathcal{F}[t_{k,\sigma}] = e^{\mathrm{i}k\phi}\, r^2 e^{-r^2\sigma^2/2}$$

where $\mathcal{F}[.]$ denotes the Fourier transform, (r,ϕ) are polar coordinates in the Fourier domain and σ is the scale. For $k = 0$, this filter is the Laplacian of Gaussian, and for $k > 0$ we get the (complex valued) k^{th}-order *Riesz transforms* of the Laplacian. The filter pair for $k = 1$ is very similar to the gradient (it has the same angular behavior) and acts as a step edge detector. Combining this with the filters for $k = 0, 2$, we obtain the *boundary energy* which correctly indicates edges, lines and many junctions types [4]:

$$b_4 = |t_{0,\sigma} \star f|^2 + 2|t_{1,\sigma} \star f|^2 + |t_{2,\sigma} \star f|^2 \tag{10}$$

where $|.|$ is the pointwise magnitude of the complex filter result. Yet another principle is applied by the *SUSAN operator* [8]. Its underlying assumption is that pixels within a homogeneous region have similar intensities. Therefore, when one measures the average similarity of a pixel with its neighbors, one expects minima at junctions and edges. The average similarity is inverted and a threshold γ added to get the SUSAN boundary indicator:

$$b_5(\vec{x}) = \max\left(0,\ \gamma - \sum_{\vec{x}' \in \mathcal{W}(\vec{x})} e^{-\left(\frac{f(\vec{x}) - f(\vec{x}')}{\delta}\right)^6}\right) \tag{11}$$

where $\mathcal{W}(\vec{x})$ is a window around $\vec{x}$, and δ scales the similarity of intensities.

4 Experiments

In our experiments, we computed the boundary indicators b_1 to b_5, interpolated them with splines of order 2 to 5 and computed the exact watershed transform. For comparison, we also determined pixel-based watersheds and Canny edgels. To isolate the properties of these algorithms w.r.t. boundary detection and linking, we first performed experiments *without* any relevance filtering. Fig. 3 illustrates important results on the well-known "blox" image (3.1). It does not contain very small objects, so we can work at scales where no oversampling is necessary. As expected, pixel-based watersheds (3.2) result in low geometric accuracy and significant oversegmentation, and inter-pixel (crack-edge) ones fare only slightly better (3.3). Canny's algorithm (3.4) produces relatively accurate edges, but it leaves gaps of about twice the gradient scale near junctions, and this significantly complicates boundary linking. The *exact watershed algorithm* achieves both high geometric accuracy and completely linked boundaries (3.5 to 3.7). The quintic spline slightly reduces oversegmentation compared with quadratic and cubic splines. No further improvement is achieved by going to a seventh order spline (not shown). These results were all obtained from identical gradient magnitude data, only the watershed algorithm was changed. In the remaining experiments, we will compare alternative boundary indicators using the quintic spline-based watershed transform. The hour-glass operator (3.8) gives the best results: oversegmentation is further reduced, and geometric accuracy of the junction response improves (especially visible at the T-junctions in the lower part of the ROI). The boundary energy (3.9) also improves the junction response, but it is slightly more susceptible to noise. Finally, the SUSAN operator (3.10) seems to be a less useful boundary indicator for the exact watershed transform.

Results at low resolution are shown in Fig. 4: The tiling of the building's wall is only coarsely sampled – the smallest tiles (small white triangles) have a diameter of only 4 pixels. The gradient magnitude result (4.2) exhibits the usual oversegmentation, but there is also a severe systematic error: The operator hallucinates non-existing tiles at the corners of the actual ones, because such configurations (saddle junctions) are not covered by the gradient edge model. In principle, this kind of error can be avoided by using the structure tensor (4.3), but in this image it does not work well: Due to the low resolution, small regions (e.g. the white triangles) are completely lost, and due to uneven

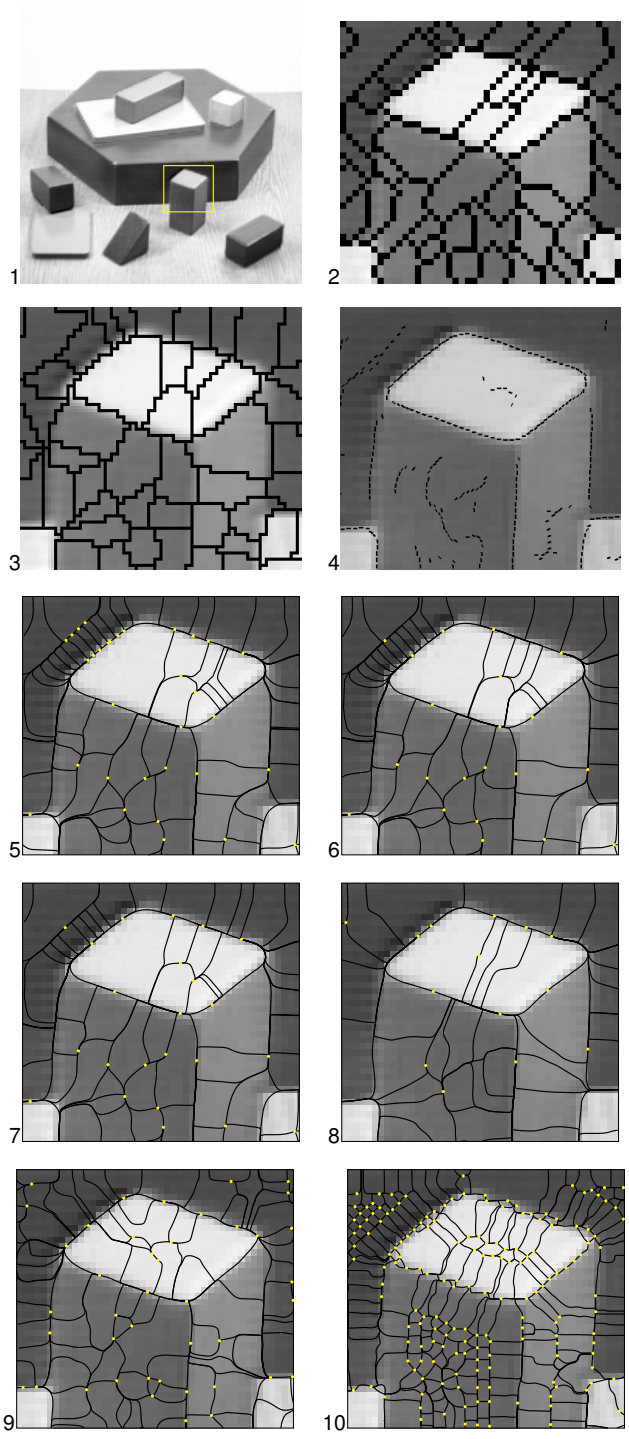

Figure 3. Evaluation of the exact watershed algorithm. **1)** original image, ROI marked in yellow; **2)** pixel-based watersheds of b_1 ($\sigma = 1.6$); **3)** inter-pixel (crack edge) watersheds of b_1; **4)** Canny edgels from Gaussian gradient with $\sigma = 1.6$; **5 to 9:** exact watersheds: **5)** order 2 interpolation of b_1; **6)** order 3 interpolation of b_1; **7)** order 5 interpolation of b_1; **8)** dto. for b_3 ($\sigma = 1.1$, $\sigma' = 2.2$); **9)** dto. for b_4 ($\sigma = 1.1$); **10)** dto. for b_5 (window radius $= 3.4$, $\gamma = ||\mathcal{W}||$, $\delta = 15$). Maxima of the boundary indicators are marked in yellow.

edge contrast, the edge geometry is significantly distorted. In contrast, an almost perfect segmentation is achieved by the nonlinear hour-glass filter (4.4).

Finally, we report an experiment with relevance filtering similar to Canny's hysteresis thresholding: We compute the minimal boundary strength for each edge and the contrast between the means of the adjacent regions, weighted by their average area. An edge is removed if both measures are below a threshold, fig. 5. The second measure replaces Canny's upper threshold which would make no sense in the context of the watershed transform because all edgels form a single connected component. The exact watershed result contains fewer irrelevant edges, because high geometric accuracy indirectly improves the expressiveness of the relevance measures. The difference in accuracy can be directly measured by fitting lines to the straigth edges of the scene. For example, the RMS residuals for the edge marked by an arrow are 0.57 pixels for the discrete vs. 0.10 pixels for the exact algorithm, given the same raw data.

5 Conclusions

In this paper we analyzed boundary detection with an exact watershed transform. This algorithm combines the advantages of the discrete watershed transform and Canny's method: It produces connected boundary graphs, and it finds edges with sub-pixel accuracy. We thus demonstrated that both topology and geometry are determined more accurately than was proviously possible. Results can be further improved by selecting appropriate boundary indicator functions (e.g. the hour-glass filtered gradient), and the method could be adapted to new applications (e.g. texture analysis) by changing the boundary indicator.

The high geometric accuracy of the exact watershed transform makes it a prime candidate for tasks involving geometric measurements, such as 3D reconstruction. Furthermore, we are currently investigating new classes of geometric edge relevance criteria which would be too inaccurate and therefore useless on pixel-based edges. On the other hand, exact watershed computation may be too slow for certain tasks (on a 256×256 image, it takes about 10 seconds on a Pentium M, 1.7 GHz), but it can still be used as a reference algorithm during solution development in order to check how much information is in the data.

A problem with our method is tangential convergence of watersheds: When watersheds come very close to each other, we humans perceive this as a junction, but the data tell otherwise. There are two possible solutions: First, we can explicitly place additional vertices at such points. This is topologically trivial, but finding the correct modified geometry is much harder. Alternatively, we can seek boundary indicators that have maxima at all perceptually significant junctions, but this is also a hard problem. This issue will be discussed in a future publication.

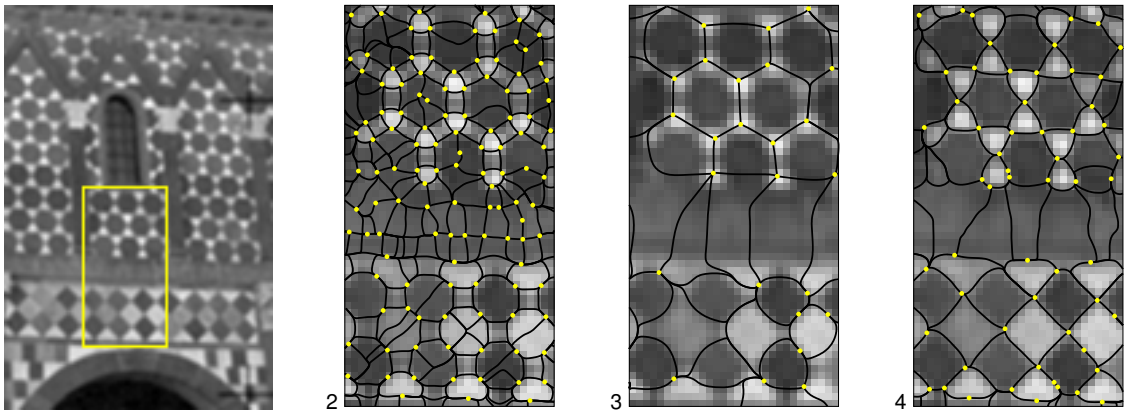

Figure 4. **1)** original image with ROI (30×58 pixels, the small triangular tiles are ≈ 4 pixels in diameter); **2)** order 5 interpolation of b_1 ($\sigma = 0.9$, oversampling); **3)** dto. for b_2 ($\sigma = 0.7, \sigma' = 1.4$, oversampling); **4)** dto. for b_3 ($\sigma = 0.9, \sigma' = 1.8$, oversampling)

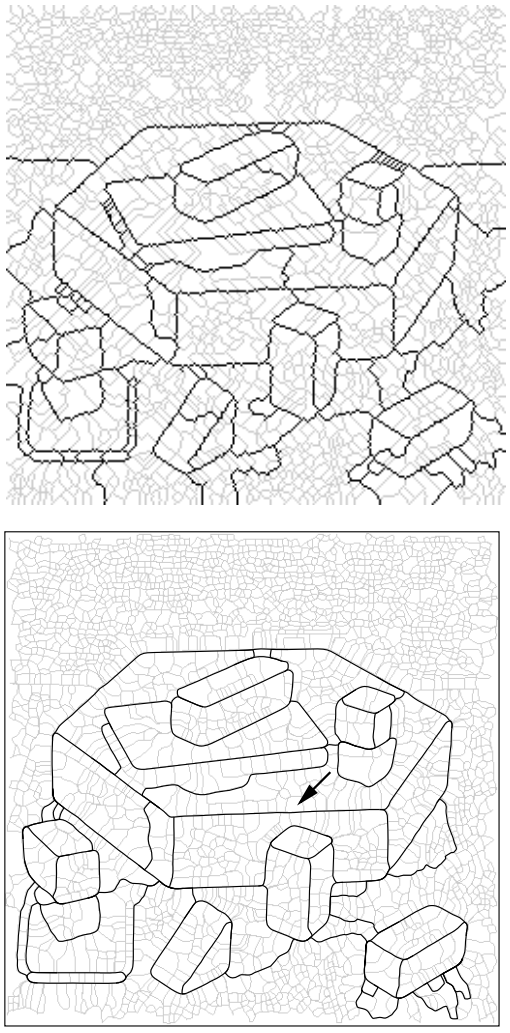

Figure 5. Pixel-based (top) and exact (bottom) watersheds after relevance filtering with the same criteria. The original oversegmentation is overlaid in light gray. When a straight line is fitted to the edge indicated by an arrow, the RMS errors are 0.57 (top image) and 0.10 (bottom) respectively.

References

[1] J. Canny: *A Computational Approach to Edge Detection*, IEEE Trans. Patt. Anal. Mach. Intell. 8(6):679-698, 1986

[2] J. Koenderink, A. v. Doorn: *Local Features of Smooth Shapes: Ridges and Courses*, SPIE vol. 2031: Proc. Geometric Meth. in Comp. Vision, pp. 2-13, 1993

[3] U. Köthe: *Edge and Junction Detection with an Improved Structure Tensor*, in: B. Michaelis, G. Krell (Eds.): Pattern Recognition, Proc. of 25th DAGM Symposium, Springer LNCS 2781, pp. 25-32, 2003

[4] U. Köthe: *Integrated Edge and Junction Detection with the Boundary Tensor*, in: ICCV '03, Proc. 9th Intl. Conf. Computer Vision, vol. I, pp. 424-431, 2003

[5] J.C. Maxwell: *On Hills and Dales,* reprinted in: W. Niven (Ed.): The Scientific Papers of James Clark Maxwell, vol. II, Dover, 1965

[6] H. Meine, U. Köthe: *The GeoMap: A Unified Representation for Topology and Geometry*, in: L. Brun, M. Vento (Eds.): Graph-Based Repr. in Pattern Recogn., Springer LNCS 3434, pp. 132-141, 2005

[7] J. Roerdink, A. Meijster: *The Watershed Transform: Definitions, Algorithms, and Parallelization Strategies,* Fundamenta Informaticae 41(1-2):187-228, 2000

[8] S. Smith, M. Brady: *SUSAN – A New Approach to Low-level Image Processing,* Intl. J. Computer Vision, 23(1):45-78, 1997

[9] C. Steger: *Subpixel-Precise Extraction of Watersheds,* in: ICCV '99, Proc. 7th Intl. Conf. Computer Vision, vol. II, 884-890, 1999

[10] M. Unser, A. Aldroubi, M. Eden: *B-Spline Signal Processing, Parts I and II,* IEEE Trans. Signal Proc. 41(2):821-848, 1993

[11] A. Zomorodian: *Topology for Computing,* Cambridge University Press, 2005