A New Sub-Pixel Map for Image Analysis

Hans Meine and Ullrich Köthe

Cognitive Systems Laboratory, University of Hamburg ,
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
{meine,koethe}@informatik.uni-hamburg.de

Abstract. Planar maps have been proposed as a powerful and easy-to-use representation for various kinds of image analysis results, but so far they are restricted to pixel accuracy. This leads to limitations in the representation of complex structures (such as junctions, triangulations, and skeletons) and discards the sub-pixel information available in grayvalue and color images. We extend the planar map formalism to sub-pixel accuracy and introduce various algorithms to create such a map, thereby demonstrating significant gains over the existing approaches.

1 Introduction

When information is extracted from an image's raw pixel data, the results must be stored in a well-defined way. Still, many image analysis approaches use their own representations (labeled images, region adjacency graphs, regular, or irregular pyramids, edgel chains, polygons, etc.). This is not only highly confusing, but also prevents algorithms that perfectly complement each other from actually being used together – their representation are simply incompatible. During the last decade, several researchers have worked on powerful unified representations.

The most promising approach is based on the notion of *planar map*s [1,2,3]. Planar maps encode the topological entities (regions, edges, vertices) of a partitioning of the (image) plane, their relations (neighborhood, boundary, containment, etc.) and their geometry. Basic modification operations support well-defined manipulations of an existing map structure. Similar concepts have been used in computer graphics for a long time [4]. Two key problems must be solved to enable their adaptation to image analysis: first, image analysis algorithms must create valid map structures. This requires the establishment of a formal correspondence between the initial pixel data and the map's entities. Second, the map must be realized in an efficient and easy-to-use way due to the huge amount of data and the complexity of the image analysis problem in itself.

So far, these goals have only been achieved with grid-based planar maps. Here, regions, edges and vertices correspond directly to sets of pixels and/or inter-pixel boundaries, i.e. can be accessed and manipulated by fast array operations. The map entities can be derived from labeled images, watershed segmentations, and pixel-based edge detectors (see Sect. 2.2). However, the gray values or colors of real images contain a considerable amount of sub-pixel information. For example, in real images step edges are always blurred by the camera's point

spread function (before sampling) and by the edge detection filters (after sampling). It is well known that the location of the ideal step can be recovered to at least 1/10 of a pixel by careful analysis of the blurred step's shape. This information is discarded when the representation is restricted to pixel accuracy.

Another limitation of grid-based maps is the representation of junctions. In an inter-pixel boundary map, at most four edges at 90° of each other can ever meet at a vertex. A pixel-based map can in principle represent more complex junctions, but these junctions are no longer single Euclidean points [2]. In real images, the corner and junction geometry is often much more complicated. This is one of the reasons why vectorial data structures are preferred for the representation of object geometry in computer graphics. Moreover, grid-based representations are harder to refine as new information arrives, whereas vectorial representations can be refined ad infinitum.

In this paper, we extend the existing grid-based map formalism to sub-pixel accuracy. We show that the map can still be efficiently realized by means of polygonal lines. Finally, we demonstrate various algorithms to create our new representation from image data, not only covering boundary detection, but also the creation of Delaunay triangulations and skeletons. Comparisons of our new results with their pixel-accurate counterparts reveal a significant gain.

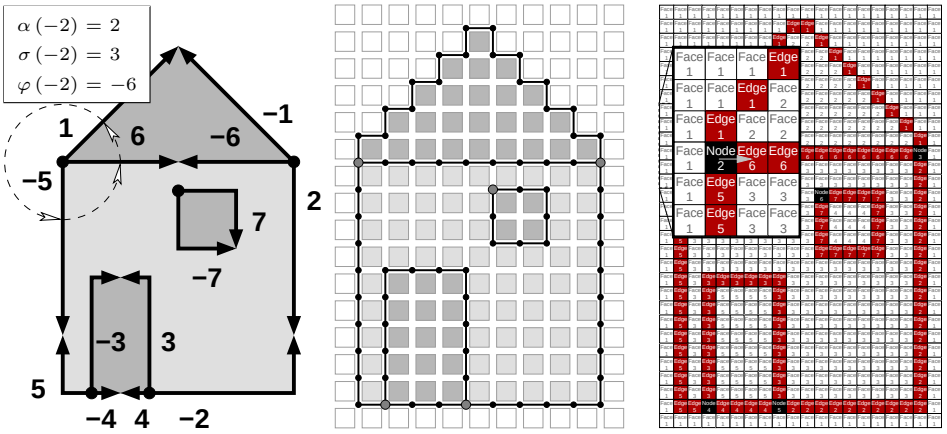
2 A Unified Representation for Topology and Geometry

Before we discuss our new sub-pixel accurate GeoMap, let us summarize previous efforts for finding a suitable representation for image segmentation purposes. Segmentation methods impose the following requirements on such a structure [5,6,2]:

1. **Topology Inspection** Algorithms need to access topological properties like the neighborhoods of regions and/or edges, the number of holes, etc. Thus, a sound topological formalism is required.
2. **Geometry Inspection** During the segmentation process, photometric / geometric properties of regions and / or boundaries are to be derived (e.g. mean color, variance, size, etc.); typical subtasks include region reconstruction in a given image, region containment queries, or inspecting image properties along boundaries (e.g. the image gradient).
3. **Modifications** If the representation is to be useful for the segmentation process itself, it must not be static. We need operations (e.g. merging two regions) modifying both the topology and the geometry in a consistent way.

2.1 Topology: Combinatorial Maps

For the representation of topology in image processing, a number of graph-like structures have been used (dating back to the RAG [7]). Nowadays, the more powerful formalism of *combinatorial maps* is commonly used, since it allows to efficiently encode most information on the embedding of a planar graph:



(a) combinatorial map (two components) (b) representation with inter-pixel contours [8] (c) representation with pixel boundaries [6,2]

Fig. 1. Examples of discrete embeddings of combinatorial maps.

Definition 1. *A* combinatorial map *is a triple* (D, σ, α) *where* D *is a set of* darts *(half-edges), and* σ, α *are permutations defined on* D *such that all* α *orbits have length* 2 *and the map is* connected*, i.e. there exists a* σ-α*-path between any two darts:*

$$\forall d_1, d_2 \in D\colon\ \exists \pi \in \left\{ \prod_{0 \le i \le k} \tau_i \,\middle|\, \tau_i \in \{\sigma, \alpha\},\ k \in \mathbb{N} \right\}\colon\ \pi(d_1) = d_2$$

The dual permutation of σ *is defined as* $\varphi(d) = \sigma^{-1}(\alpha(d))$*, where* σ^{-1} *denotes the* σ*-predecessor of* d*.*

The orbits of σ, α, and φ are called *vertices*, *edges*, and *faces* respectively, and we use the notation $\sigma^\star(d)$, $\alpha^\star(d)$ and $\varphi^\star(d)$ for the σ-, α-, and φ-orbits which contain d. The orbit $\sigma^\star(d)$ is the start vertex of d, and $\varphi^\star(d)$ is the contour of the face to the left of d. A combinatorial map is *planar*, iff the number of vertices, edges, and faces conforms to Euler's equation:

$$|\sigma| - |\alpha| + |\varphi| = 2 \quad \text{(where } |\alpha| \text{ denotes the number of orbits in } \alpha \text{ etc.)} \qquad (1)$$

When one face is designated as the (infinite) *exterior face*, all possible embeddings of a planar combinatorial map become topologically equivalent. By convention, one uses positive and negative integer labels for the darts so that $\alpha(d) = -d$ for each dart labeled d. Since φ is determined through α and σ, a single lookup table for the permutation σ is sufficient to represent a combinatorial map.

Definition 1 does not yet allow to represent multiple boundaries which commonly arise in image segmentation (inner contours like the window in Fig. 1(a)). This is usually solved by using one planar combinatorial map with a marked

exterior face per connected component, plus an additional *inclusion relation* between the maps which associates the exterior faces with their parent faces [9,10].

An alternative is to introduce *auxiliary edges* [11] to make the map connected, which we decided against because it spoils the one-to-one correspondence between topological edges and their geometrical counterparts (we do not want to make up geometrical information for the auxiliary edges).

Note that it is perfectly legal that $-d \in \varphi^\star(d)$, which means that edge $\alpha^\star(d)$ has the same region on both its left and right side. Such edges are called *bridges*, since every path between their two end-vertices must contain d or $-d$. In many publications, bridges have been considered illegal [11,1], but in fact they are required to (a) represent incomplete boundaries (e.g. arising from edge detectors like Canny's [12], which in general does not deliver complete boundaries, or during sketching) or for (b) representing skeletons (see Sect. 4.4 and Fig. 7).

Given a set of combinatorial maps $(D_i, \sigma_i, \alpha_i)$ with $i \neq j \Rightarrow D_i \cap D_j = \emptyset$, it is possible to define $D = \bigcup D_i$ and compose the permutations into a single tuple (D, σ, α) representing all components, such that e.g. $d \in D_i \Rightarrow \alpha(d) = \alpha_i(d)$. In the following, the orbits of σ, α, and φ are meant to represent *all* vertices, edges, and faces respectively. Furthermore, we will occasionally use the general term "cells" for vertices, edges, or faces, which correspond to 0-, 1-, and 2-cells in the related context of cell complexes [13].

2.2 Pixel-accurate Approaches

Combinatorial maps can be used to represent the topology of planar subdivisions, but they do not define the geometry of a tessellation, which is crucial for image segmentation. Thus, algorithms often employ a *label image* (aka. "region image") to store the geometry of regions. It is straight-forward to extract a consistent topology from the inter-pixel boundaries of such an image, in which each pixel carries the label of the region it belongs to. It has even been shown that the same is possible for *thin 8-connected pixel boundaries* [14], which for example result from watershed algorithms which leave the watersheds unlabelled.

However, from an applications' perspective it is preferable to have just one structure to deal with, not separate ones for the geometry and the topology. Thus, data structures have been developed [8,5,6,2] which encapsulate both the geometrical and topological aspects and offer means to inspect or modify the tessellation in a consistent way. Fig. 1 illustrates two pixel-based representations:

1. *Inter-pixel boundaries*: In the TOGER framework [15,1], a boundary plane is used to represent the connections between inter-pixel boundaries (at pixel corners, cf. black dots in 1(b)). This is very memory efficient (only three bits / pixel), but requires traversals and hash lookups to find the edges / regions at arbitrary positions. Darts are represented by the vertex position (cf. gray dots) and a direction.
2. *Pixel-based boundaries*: In [2,6], the internal representation of a GeoMap is based on a *cell image*, where each pixel carries a label and a type (Region / Line / Vertex). All three topological cell types are represented as connected

components of pixels carrying the corresponding type and label. All topological information is extracted via a DartTraverser, which is represented with a position / direction pair (cf. arrow in Fig. 1(c)). For details see [2,6].

The limited resolution of these approaches is not only a cosmetic problem but also affects the topology: the vertices of inter-pixel boundaries cannot have a degree > 4, while pixel-based vertices as defined in [2] can have higher degrees if they consist of more than one pixel, which reduces the geometrical quality and needs complicated thinning operations after modifications. The new representation which is presented in the following does not have that problem.

3 Representing Sub-pixel Geometry

The representations discussed in the last section serve as powerful frameworks which ease the implementation of automatic and (semi-)interactive segmentation algorithms. However, they are limited to the pixel grid, while many edge detectors deliver *edgels* (edge elements) with sub-pixel accuracy (e.g. [12,16]) which cannot be represented within these frameworks. We will now present a new approach which overcomes this limitation.

Let us assume we have sub-pixel accurate edgel positions linked into *edgel chains* (Sect. 4 will discuss some algorithms which produce these). These chains are commonly visualized with their *approximating polyline* (by connecting the points in order), and these ordered point lists serve as the main representation of edges in our new sub-pixel GeoMap. This is illustrated in Fig. 2 (left). It should be stressed that the polylines are only an approximation of the edges, and that the actual run of an edge between two support points is not represented (but could be determined on demand). This matters for algorithms analyzing the geometry, like for instance skeletonization or curvature calculation.

3.1 Meeting Algorithm Requirements

This section explains how the requirements listed in Sect. 2 are fulfilled in our implementation of the GeoMap framework.

Topology Inspection In our object-oriented design, each cell is represented with a CellInfo object which carries its properties. The framework supports the enumeration of all vertices, edges, or faces of a map, and lookups by label. CellInfo objects can be queried for canonical darts (*anchors*) whose σ, α, or φ-orbits represent the cell (a face contains one anchor per contour, the first always belonging to the outer contour). The central tool to inspect the map topology is the DartTraverser [6]. Similar to an iterator, it represents a current position – a dart within the map. It offers methods to move to the successor / predecessor in any of the three permutations, and to get the start-/end-vertices, the edge it belongs to, or the face to the left/right. Many of the methods are only for your convenience, but this interface has proven to make the GeoMap framework very powerful in practice.

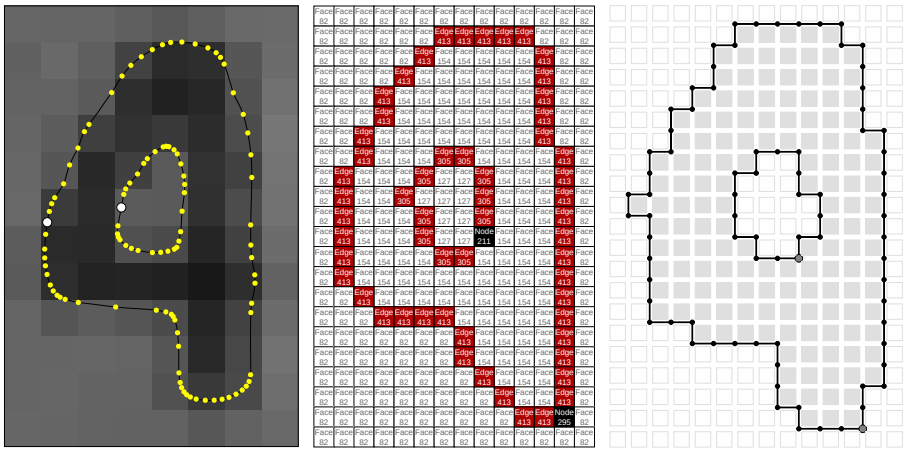


Fig. 2. Comparison of the new sub-pixel representation with approaches restricted by the pixel grid (to integer or half-integer coordinates, respectively).

Geometry Inspection The CellInfo objects mentioned above also carry the cells' geometrical properties (as well as application-specific information, cf. Sect. 3.3): a vertex simply contains its sub-pixel position, and an edge is represented as a polyline. The geometry of faces is represented implicitly; its anchors can be used to get closed polygons for each contour, and standard polygon techniques can be applied to these for reconstruction of the region, point inclusion tests, or finding the region containing a point. Since these operations are common, but rather slow, we speed them up internally with an additional label image, which Sect. 3.2 describes in more detail.

Note that it is very convenient to have the edge geometry include the vertex positions - in spite of the slight redundancy, this simplifies many algorithms, since all polyline segments can be derived from the edges, without looking at the vertices.

Modifications We define *Euler operators* to allow the modification of our GeoMap. These are atomic operations which make sure that Euler's equation (here in its form for more than one boundary component) is an invariant:

$$|\sigma| - |\alpha| + |\varphi| - C = 1 \text{ where } C \text{ is the number of connected components} \qquad (2)$$

In contrast to the relatively complex operations used in other approaches (e.g. contraction kernels [11]), we define the following minimal set of simple operations:

merge_edges merge the two edges $\alpha^*(d)$ and $\alpha^*(\sigma(d))$ and the vertex $\sigma^*(d)$ (must have degree 2) into one single edge $(|\sigma'| = |\sigma| - 1,\ |\alpha'| = |\alpha| - 1)$

remove_bridge merge the edge $\alpha^*(d)$ (which must be a bridge) into the surrounding face $\varphi^*(d)$ $(|\alpha'| = |\alpha| - 1,\ C' = C + 1)$

merge_faces merge the two faces $\varphi^*(d)$ and $\varphi^*(\sigma(d))$ (must not be identical) and their common edge $\alpha^*(d)$ into one face $(|\alpha'| = |\alpha| - 1,\ |\varphi'| = |\varphi| - 1)$

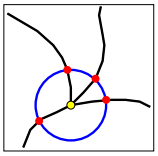
These operations can be composed into more complex ones. For instance, the removal of all edges between two regions[1] is done with the composed operation merge_faces_completely which uses merge_faces to remove the first common edge, after which the rest of the common boundary will consist of bridges which are handled one-by-one with remove_bridge.

Note that after the removal of edges, which reduces the degree of their end-vertices, these vertices may become dispensable. Vertices of degree 2 can be merged into their surrounding contour with merge_edges. However, it may be worthwhile to purposely leave vertices of degree 2 in the structure, if their geometrical counterpart marks a point of interest (e.g. a corner). Singular vertices (degree 0) are discarded in our structure.

In theory, all the mentioned operations have their natural inverses (split_edge, create bridge, split_face respectively). However, we currently restrict ourselves to operations reducing the number of cells. The reasons are manifold: (a) Our Euler Operations can all be parametrized with a single dart, and it is straightforward to prove their correctness. Their inverses need additional parameters for the geometry of the new cells to be created, which poses a problem when adding edges, since it has to be ensured that the given geometry does not violate the topology. (b) Conventional split and merge algorithms do not split faces into two, but use an implicit description of the split regions which is intrinsically limited to the pixel grid [9]. (c) The bottom-up approach of transforming an initial oversegmentation into the desired result fits well the basic idea of first looking for *any* evidence for boundaries and then applying *relevance filtering* to it.

3.2 Initializing a GeoMap

Assuming that we have already extracted boundaries from an image (examples follow in Sect. 4), this section discusses the remaining task for initializing a complete GeoMap: the determination of the boundary topology from its geometry.



The first problem is the initialization of the permutation σ, which means that we must determine the local cyclic order of edges around vertices. This may be as trivial as calculating the angles of the first segments of the approximating polylines attached to the vertex (see illustration). However, when trying to do this with sub-pixel watersheds (Sect. 4.2), this leads to numerical problems, since watersheds converge tangentially near a maximum, so subgroups of tangential darts have to be followed until they eventually diverge (see [16] for details).

Given the σ-orbits, we still have to determine the exterior faces of each connected boundary component and their parent faces. The exterior faces can be found by calculating the signed area of each contour given by the φ-orbits:

$$A = \frac{1}{2}\sum_i (x_i y_{i+1} - y_i x_{i+1}) \leq 0 \ \Rightarrow\ \text{exterior contour} \qquad (3)$$

[1] Note that merge_faces removes just one edge, whereas the common boundary might consist of several edges (cf. Fig. 1, edges 5 and 2 between wall and background).

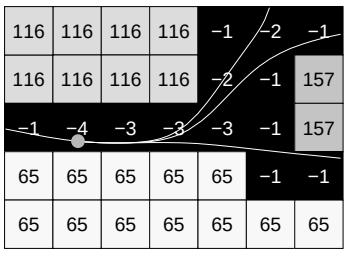


Fig. 3. Label image as used internally to speed up common geometric queries (degree four vertex in the second column, negative labels indicate number of lines intersecting a pixel facet).

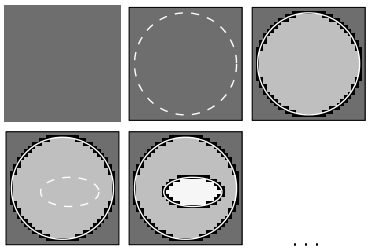


Fig. 4. incremental label image initialization and face embedding (from top left to bottom right)

If a contour contains only bridges, it is an exterior contour and A should be zero, but may be a small positive number due to numerical problems. Thus, this case must be checked explicitly.

As mentioned in Sect. 3.1, we make use of an internal label image to speed up geometry queries. For point-in-region tests, we mark pixels whose unit square is not intersected by contours with the corresponding region label. Thus, we can immediately determine which region contains a given point if it's not near the contour, see Fig. 3. Otherwise, the pixel is marked with a negative label, and we must apply a (more expensive) standard point-in-polygon test on all regions whose (cached) axis-parallel bounding box contains the point.

In order to derive the inclusion relation from the geometry, we need to check for polygon inclusion, which corresponds to inclusion of a single point, since the boundaries do not overlap. For efficiency, the following algorithm will do the face embedding in parallel to the initialization of the label image (see Fig. 4):

1. The label image is initialized with the label of the infinite outer face.
2. We sort all contours by decreasing absolute area.
3. For each contour, beginning with the largest:
   (a) If it is an exterior contour, we find the existing face including this hole contour and embed it.
   (b) Else, we add a new face to the map and apply polygon scan conversion techniques to update the label image with the new region and its contour.

In order to facilitate updates of the label image, we store the number of edges intersecting a pixel facet as negative integer (see Fig. 3). Whenever an edge is removed (by merge_faces or remove_bridge), the labels of these pixels are incremented and eventually assigned to the surrounding region if they become zero.

3.3 Maintaining Consistency of Application-Specific Data

A bottom-up image segmentation process can be described as reducing an initial set of candidate boundaries into the final tessellation. We call this reduction process *relevance filtering*. In the context of irregular pyramids, this corresponds to the pyramid bottom containing an initial oversegmentation and a "tapering" stack of levels on top with decreasing numbers of cells. In order to create such a pyramid, automatic segmentation algorithms need to consider (in)homogeneity properties of regions (boundaries) to decide upon insignificant boundaries.

Typical region properties used for relevance filtering are statistics on the regions' colors (mean, variance, ...), area, or circumference. Boundaries are often assessed based on the local image gradient, their length, or curvature. The GeoMap makes it very simple to calculate such information and attach it to the CellInfo objects. During the segmentation process, this information has to be kept up-to-date when removing (parts of) boundaries. It would be possible to re-calculate the information after each change, but for common statistics it is possible (and much more efficient) to incrementally compute it from the cell information before the change.

Our GeoMap representation thus supports to register separate pre- and post-operation callback functions for each Euler operation in order to enable application-specific statistics to be maintained in a consistent way [6,1]. This ensures that each Euler operation is accompanied by the appropriate updating procedures. The dart which parameterizes the operation is passed to the pre-operation callbacks, to inform them which cells will be merged. The update functions will collect the necessary information from the old cells and wait for the post-operation call, which attaches the updated information to the CellInfo object of the surviving cell, which it gets passed as parameter.

This approach makes it very easy for an application to manage e.g. photometric information on the regions, specific flags needed to perform the segmentation algorithm, or information on the boundary (like the mean gradient or a watersheds' pass value), and it is always guaranteed that this information is up-to-date. The GeoMap itself maintains some meta information on the cells' geometry (lengths, areas, bounding boxes), which is also made available and does not have to be recalculated.

Note that we internally store the partial sum of the signed area (3) for each edge, which allows us to quickly determine the signed area of any contour. (The removal of a bridge leads to a new contour whose area is unknown, and the partial sums efficiently solve the problem that the area is needed to determine the new exterior contour if the bridge belonged to the old exterior contour.)

4 Applications

Now that we have introduced our new sub-pixel precise representation formalism, we will show how it can be used with some image analysis algorithms.

4.1 Preliminaries: Continuous View on Input Images

A key tool to all our sub-pixel resolution experiments is that we can adaptively sample images at any desired (sub-pixel) position. This can be done efficiently by means of spline interpolation.

Splines of order n possess $n-1$ continuous derivatives and can be efficiently computed at any location $\boldsymbol{x} = (x, y)$ by convolution of discrete spline coefficients c_{ij} with continuous B-spline basis functions β_n:

$$f(x,y) = \sum_{i,j} c_{ij}\, \beta_n(i-x)\, \beta_n(j-y) \tag{4}$$

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 [17,16]. We use spline interpolation throughout this work for retrieving image values at sub-pixel locations, because of their global continuity across facet borders.

A side effect of the spline reconstruction is that interpolated real images (containing noise) will not have any plateaus in practice (when represented with floating-point accuracy). This is important for methods relying on the gradient vanishing only at isolated points (like the contour following methods described below). Note that it is not necessary to use convolution filters for derivatives, because they can be derived analytically from the spline approximation.

4.2 Sub-Pixel Watersheds

When comparing the classical watersheds-by-flooding algorithm [18] with e.g. Canny's edge detector [12], watersheds have the disadvantage of being limited to the pixel grid. On the other hand, they provide closed contours, so that a complete topology can be derived [8,14]. The advantages of both worlds can be combined by applying a sub-pixel watershed algorithm to the interpolated boundary indicator function [16,19]. This algorithm is based on a mathematical definition of watersheds given by Maxwell [20]: watersheds are flowlines between maxima and saddles. If the function f is differentiable, a unique flowline exists at every point with non-zero gradient, and flowlines can be traced (upwards, starting at saddle points) by numerically solving their differential equation

$$\frac{\partial \boldsymbol{x}(t)}{\partial t} = \nabla f(\boldsymbol{x}(t)) \tag{5}$$

(e.g. with the Runge-Kutta method). This is stable near a watershed, because all flowlines in a neighborhood converge to the same maximum (for details, see [16]).

The algorithm is significantly slower than pixel-based watershed algorithms, but gives very high resolution (as can be seen in Fig. 5). Since the flowlines connect saddles and maxima, the output of the algorithm naturally forms a graph, which can be turned into a map after determining the σ-order of edges around each vertex (maximum). As mentioned in Sect. 3.2, the cyclic order of edges

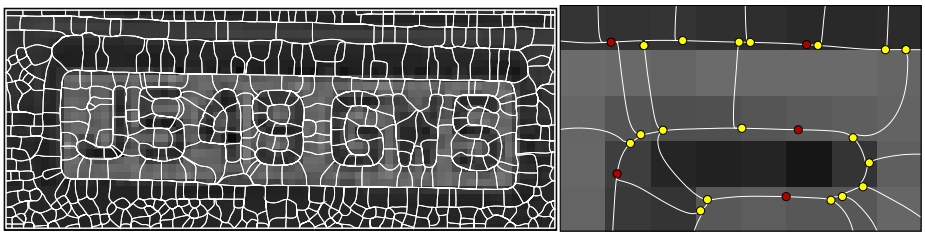


Fig. 5. Sub-pixel watersheds. *left*: initial oversegmentation, *right*: problem of tangential convergence; additional vertices added (yellow) vs. original vertices / maxima (dark red)

cannot be determined locally due to numerical problems because watersheds converge tangentially near maxima, see the detailed close-up in Fig. 5. The yellow circles mark the locations where the watersheds diverge (as found by our σ-sorting algorithm [16]). It is advisable to add additional vertices at these positions, since otherwise the statistics of a topological edge may contain mixed-up information from several geometrically unrelated segments. (The exact vertex positions may be refined later.)

4.3 Sub-Pixel Level-Set Contour Tracing

An alternative method for finding an initial boundary set is not to look for ridges, but for zero-crossings of an appropriate edge detector (e.g. the Laplacian-of-Gaussian [21]) or of a distance function resulting from variational segmentation in level-set approaches. More generally, this can be used to find any level lines implicitly defined by

$$\phi\,(x,y) = c \;\Leftrightarrow\; \tilde{\phi}\,(x,y) := \phi\,(x,y) - c = 0$$

The tangent unit vector $\boldsymbol{t}$ of a level-line is always perpendicular to the gradient direction: $\boldsymbol{t} = \nabla\phi^{\perp} / \left|\nabla\phi\right|$. Thus, the points of a level-line fulfill the PDE

$$\frac{\partial \boldsymbol{x}(\tau)}{\partial \tau} = \pm \boldsymbol{t}(\tau) = \pm \frac{\nabla\phi(\tau)^{\perp}}{|\nabla\phi(\tau)|} \tag{6}$$

with initial condition $\phi(\boldsymbol{x}_0) = 0$ and $\nabla\phi(\boldsymbol{x}_0) \neq 0$. In principle, this PDE could be solved with standard methods (like Runge-Kutta's), but this does not take advantage of the fact that the level-line must remain at this particular level. This constraint is used by *predictor-corrector methods* which significantly simplify level-line tracing. They use the tangent to extrapolate the curve towards a new candidate point (predictor step), but these predictions need not be extremely accurate because the level constraint is subsequently used to move the new point's position onto the contour (corrector step). Compared to other methods, this allows simpler predictors or larger steps. The basic algorithm is as follows [22]:

1. Given: a differentiable function $\phi(\boldsymbol{x})$ and a starting point $\boldsymbol{x}_0$ such that $\phi(\boldsymbol{x}_0) = 0$. Select an initial step size h and a bound ϵ_0 that specifies how much $\phi(\boldsymbol{x})$ may deviate from the exact zero level along the line.

Fig. 6. Level-set contours of an ancient Chinese transcript (*right*: close-up).

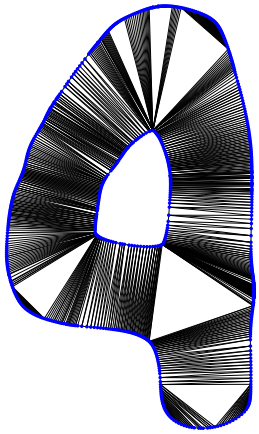
2. While stopping criterion not fulfilled:
   (a) Predict candidate point $\hat{\boldsymbol{x}}_{i+1}^{(0)} = h\,\boldsymbol{t}(\boldsymbol{x}_i)$ where $\boldsymbol{t}(\boldsymbol{x}_i) = \frac{\nabla\phi^{\perp}(\boldsymbol{x}_i)}{|\nabla\phi(\boldsymbol{x}_i)|}$ if $\boldsymbol{x}_i$ is not a saddle point of ϕ, and $\boldsymbol{t}(\boldsymbol{x}_i) = \frac{\boldsymbol{x}_i - \boldsymbol{x}_{i-1}}{|\boldsymbol{x}_i - \boldsymbol{x}_{i-1}|}$ otherwise.
   (b) While $\left|\phi\left(\hat{\boldsymbol{x}}_{i+1}^{(k)}\right)\right| > \epsilon_0$:
      i. Correct the candidate point by Newton iterations

$$\hat{\boldsymbol{x}}_{i+1}^{(k+1)} = \hat{\boldsymbol{x}}_{i+1}^{(k)} - \frac{\phi\left(\hat{\boldsymbol{x}}_{i+1}^{(k)}\right)}{\left|\nabla\phi\left(\hat{\boldsymbol{x}}_{i+1}^{(k)}\right)\right|^2}\nabla\phi\left(\hat{\boldsymbol{x}}_{i+1}^{(k)}\right)$$

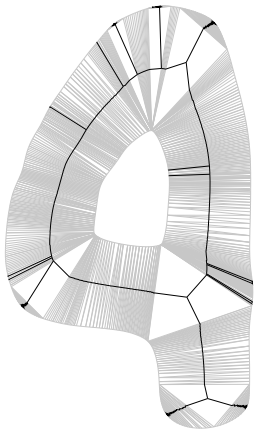
   (c) If the total correction was small, accept $\hat{\boldsymbol{x}}_{i+1}^{(k+1)}$ as new point $\boldsymbol{x}_{i+1}$, set $i := i+1$, possibly increase h, and go to 2. Else, reduce h and go to (a).

Since level-lines form closed contours, one wants to stop the algorithm when it returns to the starting point. Detecting this is not trivial, but since we define $\phi(\boldsymbol{x})$ as a spline, there is a simple solution which also solves the problem of detecting starting points: consider the explicit polynomial representation (4) of a spline and the locus of points where $x = i \vee y = j$. We get a set of horizontal and vertical lines through the sampling points, enclosing small unit squares. Along these lines, (4) simplifies to two 1-dimensional polynomials of order n, and the roots of these polynomials can easily be computed by a standard root finder. Each root that lies on the side of the corresponding unit square marks a point where the zero level-line crosses. By iteratively choosing one of the crossings as the starting point and applying the above algorithm to trace the level-line until it leaves this square at another of the known crossings, we get connected level contours.

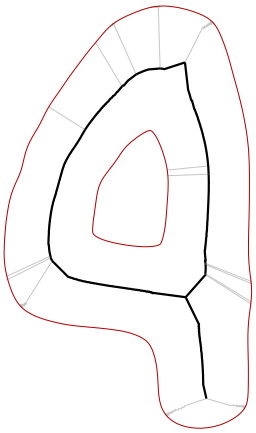
Finally, we must identify the vertices in order to initialize a GeoMap with the contours (according to Sect. 3.2). Since edges derived from zero-crossings always form closed contours, there are two kinds of vertices: if the curve self-intersects,



(a) Constrained Delaunay Triangulation



(b) CAT (Chordal Axis Transform) [23]



(c) Pruned CAT Skeleton

(d) *light red:* contours (after relevance filtering), *white:* pruned CAT skeletons

Fig. 7. GeoMaps representing contours, triangulations, and skeletons

all intersection points are vertices. Otherwise, an arbitrary point on the curve must be selected as a vertex.

Fig. 6 shows an example of such level-set contours: again, we can combine the advantage of high sub-pixel resolution with the advantage of common thresholding, which does not need any convolution filters and can thus be applied without implicit smoothing if the signal-to-noise ratio is high enough. In Fig. 6, this helps us in analyzing the cusps, which are important stroke characteristics.

4.4 Triangulation / Skeletonization

Our map is not only suitable for representing segmentation results, but it is also an adequate representation for triangulations or for skeletons (the latter requires the representation of bridges, see Sect. 3). Topological data structures have a long history in the computation of Delaunay triangulations and Voronoi diagrams (e.g. the quad-edge structure used in [24]).

The versatility of our GeoMap is illustrated in Fig. 7, which displays the result of the following example process:

1. First, we calculate sub-pixel watersheds of the original image from the spline-interpolated gradient magnitude.
2. (Simple relevance filtering) We iteratively merge regions until the difference between the average color of all adjacent regions is larger than a threshold (dark red contours in Fig. 7(d)).
3. Detect letters as hole regions which are darker than their parent face.
4. Apply a constrained Delaunay triangulation (CDT) to all letters, cf. Fig. 7(a).
5. (Chordal Axis Transform) Connect the mid-points of the inner chords to new edges, create a vertex for each inner ("join") triangle and connect vertices and edges to a CAT skeleton map, Fig. 7(b) (for details, see [23]).
6. (Simple pruning) Remove small branches: apply `remove_bridge` to edges shorter than two pixels with an end-vertex of degree 1 (Fig. 7(c), pruned parts in light gray).

Fig. 7(d) shows the contours from step 2 in red (note that the width of the original letter parts is less than two pixels, the whole region of interest is 64×19) and the pruned skeleton in white (the slight difference between the "5" and the "S" remains visible in the skeletons). The sampling of the "W" obviously violated Shannon's theorem and is hardly recognizable for a human, too.

This example is not meant to be a sophisticated, general feature extraction method, but it nicely illustrates the power of the GeoMap as a representation for planar graphs which offers convenient means to

- merge regions (step 2)
- manage statistical information on regions or edges, e.g. a regions' mean color (steps 2 and 3) or area (step 3), or the length of edges (step 6)
- inspect the geometry and decide upon inner / outer of regions (CDT, step 4)

5 Conclusion

Unified representations offering both topological and geometrical perspectives on a segmentation have been shown to be powerful as well as easy-to-use. In this paper, we extended the GeoMap formalism to achieve sub-pixel accuracy. We have shown that besides advanced sub-pixel segmentation techniques, triangulation and skeletonization can be performed equally well with our representation. Our experiments have shown that the advantages of the general planar map formalism still apply: our GeoMap framework allows for a significantly faster development of algorithms than without such a representation, and their formulations tend to become more concise due to the high level of abstraction. Algorithms with previously separate data structures can easily be compared and combined.

We are planning to release our implementation in the context of the VIGRA library. On the application side, we are currently working on the integration of learning methods and more sophisticated edge salience measures (e.g. based on boundary continuity or curvature) for relevance filtering.

References

1. Braquelaire, A.: Representing and segmenting 2d images by means of planar maps with discrete embeddings: From model to applications. In Brun, L., Vento, M., eds.: Graph-based Representations in Pattern Recognition, Springer (2005) 92–121
2. Meine, H., Köthe, U.: The GeoMap: A unified representation for topology and geometry. In Brun, L., Vento, M., eds.: Proc. Graph-Based Representations in Pattern Recognition, Springer (2005) 132–141
3. Brun, L., Kropatsch, W.: Construction of combinatorial pyramids. In Hancock, E., Vento, M., eds.: Graph-Based Repr. in Pattern Recognition, Springer (2003) 1–12
4. Mäntylä, M.: An Introduction to Solid Modeling. Computer Science Press (1988)
5. Brun, L., Domenger, J.P., Braquelaire, J.P.: Discrete maps: a framework for region segmentation algorithms. In: Graph-based Representations in Pattern Recognition, Springer (1998) 83–92
6. Meine, H.: XPMap-based irregular pyramids for image segmentation. Diploma thesis, Dept. of CS, University of Hamburg (2003)
7. Pavlidis, T.: Structural Pattern Recognition. Springer (1977)
8. Brun, L., Domenger, J.P.: Incremental modifications of segmented images. Technical Report RR112696, Université Bordeaux, LABRI (1996)
9. Brun, L., Domenger, J.P.: A new split and merge algorithm with topological maps and inter-pixel boundaries. In: Proc. WSCG'97. (1997)
10. Köthe, U.: XPMaps and topological segmentation - a unified approach to finite topologies in the plane. In Braquelaire, A., Lachaud, J.O., Vialard, A., eds.: Conf. on Discrete Geometry for Computer Imagery. Springer (2002) 22–33
11. Kropatsch, W.G.: Building irregulars pyramids by dual graph contraction. IEEE-Proc. Vision, Image and Signal Processing **142** (1995) 366–374
12. Canny, J.: A computational approach to edge detection. T-PAMI **8** (1986) 679–698
13. Kovalevsky, V.A.: Finite topology as applied to image analysis. Computer Vision, Graphics, and Image Processing **42** (1989) 141–161
14. Köthe, U.: Deriving topological representations from edge images. In Asano, T., Klette, R., Ronse, C., eds.: Geometry, Morphology, and Computational Imaging, WS on Theoretical Foundations of Computer Vision. Springer (2003) 320–334
15. Braquelaire, J.P., Brun, L.: Image segmentation with topological maps and inter-pixel representation. J. Visual Comm. and Image Representation **9** (1998) 62–79
16. Meine, H., Köthe, U.: Image segmentation with the exact watershed transform. In Villanueva, J., ed.: Proc. VIIP'05, ACTA Press (2005) 400–405
17. Unser, M., Aldroubi, A., Eden, M.: B-Spline signal processing: Part I and II. IEEE Trans. on Signal Processing **41** (1993) 821–848
18. Vincent, L., Soille, P.: Watersheds in digital spaces: an efficient algorithm based on immersion simulations. In: T-PAMI **13** (1991) 583–598
19. Steger, C.: Subpixel-precise extraction of watersheds. In: Proc. of 7^{th} ICCV. Volume 2., IEEE Computer Society (1999) 884–890
20. Maxwell, J.C.: On hills and dales. Reprinted in W. D. Nivin (Ed.): The Scientific Papers of James Clerk Maxwell **2** (1952) 233–240. Dover Publications.
21. Marr, D., Hildreth, E.C.: Theory of edge detection. Proceedings of the Royal Society of London **B207** (1980) 187–217
22. Allgower, E.L., Georg, K.: Numerical path following. In: P.G. Ciarlet, J.L. Lions (Eds.), Handbook of Numerical Analysis **5** (1997) 3–207. North-Holland.
23. Prasad, L.: Morphological analysis of shapes. CNLS Newsletter **139** (1997)
24. Guibas, L.J., Stolfi, J.: Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. ACM Trans. on Graphics **4** (1985) 74–123