PFG 2013 / 4, 0285–0296
Stuttgart, August 2013

DijkstraFPS: Graph Partitioning in Geometry and Image Processing

Falko Schindler & Wolfgang Förstner, Bonn

Keywords: graph partitioning, surface segmentation, triangulation, clustering, image segmentation

Summary: Data partitioning is a common problem in the field of point cloud and image processing applicable to segmentation and clustering. The general principle is to have high similarity of two data points, e.g. pixels or 3D points, within one region and low similarity among regions. This pair-wise similarity between data points can be represented in an attributed graph. In this article we propose a novel graph partitioning algorithm. It integrates a sampling strategy known as farthest point sampling with Dijkstra's algorithm for deriving a distance transform on a general graph, which does not need to be embedded in some space. According to the pair-wise attributes a Voronoi diagram on the graph is generated yielding the desired segmentation. We demonstrate our approach on various applications such as surface triangulation, surface segmentation, clustering and image segmentation.

Zusammenfassung: *DijkstraFPS: Graphpartitionierung in Geometrie und Bildverarbeitung*. Datenpartitionierung ist eine elementare Aufgabe im Bereich Punktwolken- und Bildverarbeitung, vor allem zur Segmentierung und zum Clustern. Das generelle Prinzip ist es, hohe Ähnlichkeit zwischen zwei Datenpunkten derselben Region und geringe Ähnlichkeit zwischen verschiedenen Regionen zu erreichen. Diese paarweise Ähnlichkeit kann als attributierter Graph auf den gegebenen Daten repräsentiert werden. In diesem Artikel stellen wir einen neuen Graphpartitionierungsalgorithmus vor. Er integriert eine Samplingstrategie namens Farthest Point Sampling mit dem Verfahren von Dijkstra zur Ableitung einer Distanztransformation auf einem allgemeinen Graphen, der nicht in einen Raum eingebettet sein muss. Gemäß der paarweisen Attribute wird ein Voronoi-Diagramm auf dem Graphen generiert, das die gewünschte Segmentierung liefert. Wir demonstrieren unseren Ansatz für verschiedene Anwendungen, wie die Oberflächentriangulierung, die Oberflächensegmentierung, das Clustering und die Bildsegmentierung.

1 Introduction

In the fields of point cloud and image processing many applications require the partitioning of the underlying data as a pre-processing step. For both, surface and image segmentation, a 2D manifold is to be partitioned into non-overlapping regions. But also line partitioning, reconstructing, i.e. triangulating, surfaces as well as clustering data points in high dimensional feature space involve partitioning the geometric or the feature space.

The number of possible partitionings of a dataset is extremely large. Already for a binary partitioning of an image with N pixels into foreground and background one has 2^N possible partitionings. Therefore, no generally optimal technique for partitioning exists.

Methods for partitioning fall into two types. *Split and/or merge techniques* start from a dissimilarity and/or similarity measure within and/or between regions which are *iteratively* found by splitting the complete dataset and/or by merging the individual elements. A large number of partitioning techniques exists, e.g. based on quad- or octrees, normalized cuts (Shi & Malik 2000) – which are only optimal for one partitioning – or graph-cut based meth-

DOI: 10.1127/1432-8364/2013/0177 1432-8364/13/0177 $ 3.00

ods (BOYKOV & FUNKA-LEA 2006) – which are only optimal for binary partitioning and special similarity functions. Split and merge techniques are only describable by the local properties of the individual iteration steps and do not possess a global property.

Partitioning can be interpreted as *clustering* in a feature space, the features depending on the original data (FORSYTH & PONCE 2002). This immediately suggests to seek for modes of the density function induced by the features and finding the valley lines (COMANICIU & MEER 2002). The principle of *watershed algorithms* is the inverse (SZELISKI 2010): Regions are catchment areas bounded by the watershed lines of a gradient image, the gradient measuring the dissimilarity between neighbouring elements (VINCENT & SOILLE 1991, MEINE & KÖTHE 2005). The quality of the mean shift and watershed partitioning depends on the ability to define problem adequate features, why these methods often lead to oversegmentation, i.e. a too large number of regions requiring a subsequent merging step. However, both methods can be described by the global properties of their solution.

A segmentation is either used to reduce the complexity of subsequent algorithms as the number of basic elements usually is several orders of magnitudes larger than the number of regions, recently consequently named superpixels (VEKSLER et al. 2010, MESTER et al. 2011, ACHANTA et al. 2012). Alternatively, segmentation is understood as a first step towards a symbolic image description, where the regions are basic units for a subsequent image interpretation. In the last years this lead to the concept of semantic image segmentation, where the segmentation is understood and realized as supervised classification (ROTHER et al. 2004, ROSCHER 2012, ARBELAEZ et al. 2012).

Most techniques can be interpreted as graph partitioning. The graph is either given by the structure of the data as for digital images, or derived from the data by some similarity measure based on geometry alone as in point cloud processing, on feature similarity as in clustering, or on semantic closeness as in semantic segmentation.

We propose a new efficient split and merge type graph partitioning algorithm, which iteratively determines a set of Voronoi cells based on an application dependent metric. The strength of the algorithm lies in its ability to overwrite the partitioning of the previous step within the sequence of split and merge steps. Due to a careful choice of the similarity metric for the graph's edge attributes we are able to control the alignment of Voronoi edges according to our objective. We will describe how to choose the edge attributes for different applications like curve and surface reconstruction, curve and surface segmentation, clustering and image segmentation.

In section 2 we will recapitulate previous work on marching front based sampling and partitioning methods before we formulate a generalized graph partitioning algorithm in section 3. We demonstrate our graph partitioning method on various applications in section 4 and conclude in section 5.

2 Background and related Work

Our graph partitioning method is based on computing Voronoi diagrams on an edge attributed graph, the attributes containing some application dependent distance between two nodes. In discrete geometry there are two common algorithms for deriving a *distance map*: Dijktra's algorithm (DIJKSTRA 1959) and fast marching method (FMM, SETHIAN 1996).

Given one or more seed points, Dijkstra's algorithm computes the shortest path along *existing graph edges* from each vertex to its closest seed point yielding a distance map. Implicitly this yields path lengths, also called intrinsic or geodesic distances. Note that the *distance* is not necessarily the spatial distance, but the sum of all edge attributes along the path.

FMM, especially its formulation for meshed manifolds (KIMMEL & SETHIAN 1998), computes *surface intrinsic distances* on meshed surfaces. In contrast to Dijkstra's algorithm, however, paths can pass *through* triangles, thus are not restricted to triangular edges. Technically, both Dijkstra's algorithm and FMM solve the so-called Eikonal equation $|\nabla d(\mathbf{x})| = F(\mathbf{x})$ with the boundary condition $d(\mathbf{x}_0) = 0$. In terms of distances on meshed manifolds its interpretation is as follows: Given a seed point $\mathbf{x}_0$ and a function F defining the friction at

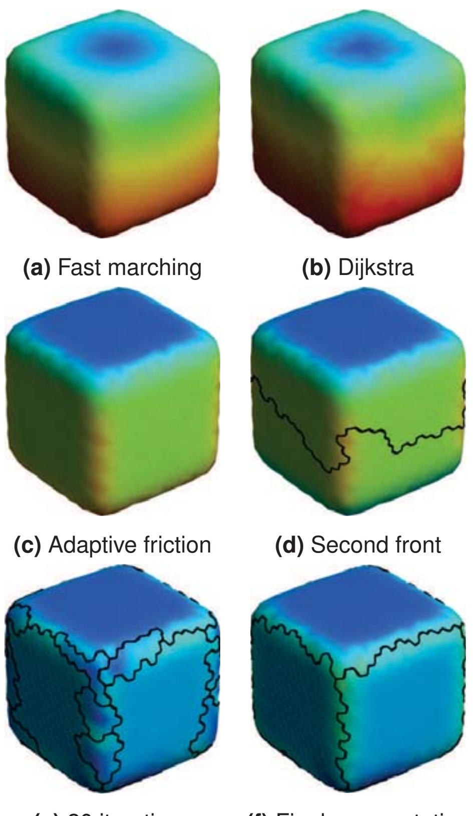


Fig. 1: (a): Demonstration of surface segmentation as proposed in SCHINDLER & FÖRSTNER (2011). Starting from a random seed point the geodesic distance to each other surface point is computed via fast marching (KIMMEL & SETHIAN 1998). (b): The distance is colour-coded from near (blue) to far (red). This distance map can be approximated with Dijkstra's algorithm. (c): Using a curvature adaptive friction function $F(\mathbf{x})$ we obtain small increases along planar regions and large gradients at sharp edges. (d): Adding the farthest point as new seed point and repeating Dijkstra's algorithm we obtain a segmentation into two segments. (e): The second wave front stops somewhere in the middle when newly computed distances exceed distances from the first iteration. After 20 iterations we clearly observe an oversegmentation. (f): To eliminate small segments along the edges we proposed a decremental segmentation step yielding the desired result with 6 segments only. The combination of incremental and decremental segmentation turned out to increase both robustness to data noise and independence of the random location of the first seed point (SCHINDLER & FÖRSTNER 2011).

each point $\mathbf{x}$ on the manifold, FMM yields the distance map $d(\mathbf{x})$ such that the gradient magnitude $|\nabla d(\mathbf{x})|$ is identical to the local friction $F(\mathbf{x})$. Then, the distance $d(\mathbf{x})$ is proportional to the arrival time of a propagating wave front starting at point $\mathbf{x}_0$ in case the local friction is inversely proportional to the local speed of the wave front. Both algorithms run in $O(N \log N)$ on sparse graphs, Dijkstra's algorithm of course only achieving an approximate solution.

Exploiting FMM for calculating exact distance maps on surfaces, MOENNING & DODGSON (2003a,b) propose a strategy for surface segmentation called fast marching farthest point sampling (FastFPS). It tries to find a set of *seed points* such that the corresponding Voronoi cells correspond to the desired partitioning of a surface. Their method starts from a random seed point $\mathbf{x}_0$. Every next seed point will be the point $\mathbf{x}^*$ with largest distance after updating the distance map d via FMM. After N iterations they obtain N seed points $\mathbf{x}_n$, $n = 1 \ldots N$, a distance map representing the distance to the closest of all N seed points and implicitly a Voronoi segmentation. PEYRÉ & COHEN (2004, 2006) apply a similar farthest-point strategy on meshed surfaces for segmentation, re-meshing and surface flattening using geodesic distances.

In SCHINDLER & FÖRSTNER (2011) we proposed FPS with Dijkstra's algorithm to robustly segment meshed surfaces (Fig. 1), taking into account the sub-optimality of Dijkstra's algorithm to generate a distance map. We also proposed an application specific stopping criterion to automatically determine a suitable number of seed points N. Such a stopping criterion can be found for other applications as well, but is out of scope of this paper. Further, we introduced a decremental segmentation strategy: We iteratively remove small segments by setting their vertex distances to infinity and re-run Dijkstra's algorithm starting from neighbouring segments.

Although FMM yields more accurate results, it is limited to manifolds. Dijkstra's algorithm approximates the geodesic distance, but for densely sampled manifolds the differences are small (Figs. 1a and 1b). More decisive, Dijkstra's algorithm does not require the underlying graph to be embedded in some space. Thus, in case the graph is embedded, i.e. its nodes possess coordinates on a line, a surface or in a volume, one may partition the line, the surface or the volume according to the metric. Otherwise one may just partition the nodes of the graph according to distances encoded in the edges. This increases the flexibility of the proposed approach.

Here we transfer the idea of surface segmentation using FPS to a graph partitioning procedure. Therefore, we replace FMM by Dijkstra's algorithm to be able to handle more general graphs which are not embedded in some space. In the following, we present the novel algorithm, called DijkstraFPS, and demonstrate it with various applications in geometry and image processing.

3 Graph Partitioning

DijkstraFPS can be seen as a general graph partitioning algorithm, detached from the underlying semantics. We assume that the semantics of the partitioning problem can be coded in the distances between the nodes of a general graph. It does not need to be embedded in some space.

The procedure is given in Algorithm 1. It assumes, all pair-wise distances are pre-computed and encoded as *edge costs* between adjacent pairs of *nodes*.

The decremental segmentation step is almost identical. Only the FPS (lines 3–5 in Algorithm 1) are modified: Instead of initializing the front $\mathcal{Q}$ with the farthest point $s \leftarrow \text{argmax}_n d_n$ with distance $d_s \leftarrow 0$ and label $l_s \leftarrow l^+$, we initialize the front $\mathcal{Q}$ with the neighbouring vertices n of the smallest segment l^- with distance $d_n \leftarrow \infty$ and label $l_n \leftarrow$ undefined. The *Dijkstra step* (lines 6–13) will propagate the front into the unlabelled region l^- and update all corresponding vertices.

Observe, only the distances of those vertices are updated which are closer to the current seed node (line 11), whose number diminishes with each additional seed node.

Fig. 2 illustrates the partitioning of a synthetic example graph with 7 nodes connected by 12 weighted edges. After three incremental Dijkstra steps (Figs. 2b–2d) one decremental step (Fig. 2e) is performed. Note that with

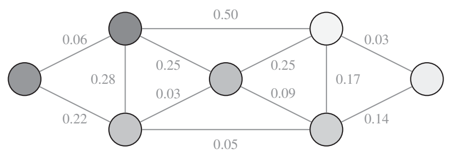
```
**In**: neighbours $\mathcal{N}$, costs $F$, #segments $L_{\text{inc}}$
**Out**: node distances $\mathbf{d}$, node labels $\mathbf{l}$
1  distances $\mathbf{d} \leftarrow \infty$, labels $\mathbf{l} \leftarrow$ undefined;
2  **for** $l^+ \leftarrow 1$ **to** $L_{\text{inc}}$ **do**
3      pick new seed $s \leftarrow \text{argmax}_n d_n$;
4      distance $d_s \leftarrow 0$ and label $l_s \leftarrow l^+$;
5      initialize new front $\mathcal{Q} \leftarrow \{s\}$;
6      **while** *front is not empty:* $\mathcal{Q} \neq \{\}$ **do**
7          select node $u \leftarrow \text{argmin}_{q \in \mathcal{Q}} d_q$;
8          remove $u$ from front $\mathcal{Q} \leftarrow \mathcal{Q} \setminus u$;
9          **foreach** *neighbour* $v \in \mathcal{N}_u$ **do**
10             new distance $d'_v \leftarrow d_u + F_u^v$;
11             **if** *new distance* $d'_v < d_v$ **then**
12                 update $d_v \leftarrow d'_v, l_v \leftarrow l_u$;
13                 add to front $\mathcal{Q} \leftarrow \mathcal{Q} \cup v$;
```

Algorithm 1: DijkstraFPS graph partitioning. New seed nodes s are added iteratively by choosing the farthest node w. r. t. node distances $\mathbf{d}$ (FPS, lines 3–5) that are constantly updated using Dijkstra's algorithm (lines 6–13) with edge costs F_u^v, yielding a labelling $\mathbf{l}$.

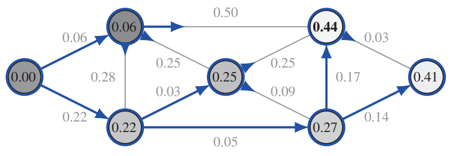
DijkstraFPS graph partitioning all boundaries are possibly subject to change in a following Dijkstra step.

Within this paper we will restrict to undirected graphs only. Directed graphs, however, work as well, i.e. the propagation is cheaper in one direction than the other. Moreover, edge costs do not have to fulfill the triangle inequality of a metric space nor has the graph to be Euclideanly embeddable. Only negative costs are disallowed to avoid infinite loops during Dijkstra's algorithm.

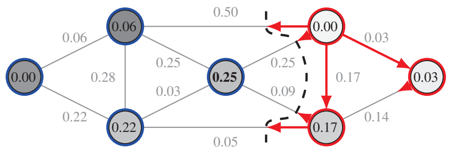
Many applications suggest an embedding of the graph into a surface or a volume, leading to a graph based on a triangulation or tetrahedralization. Fig. 3 depicts two possible graph structures for a set of given 2D points. The triangle-based graph structure introduces one node per triangle (Fig. 3a). When labeling triangles via DijkstraFPS we obtain a segmentation boundary along triangular edges. The vertex-based graph structure introduces one node per vertex, i.e. 2D point, (Fig. 3b) yielding a segmentation boundary along Voronoi edges. In the following section 4 we will demonstrate both and point out when to use which structure.



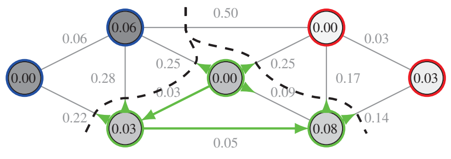
(a) Synthetic graph with edge costs



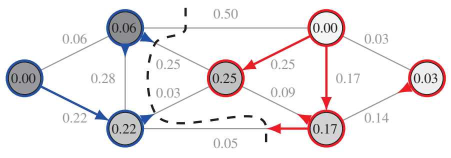
(b) 1^{st} Dijkstra step from random seed node



(c) 2^{nd} Dijkstra step from farthest node from (b)

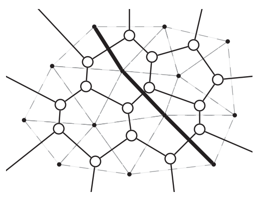


(d) 3^{rd} Dijkstra step from farthest node from (c)

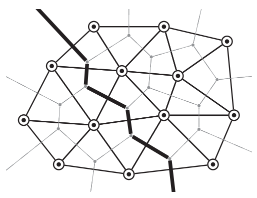


(e) Decremental step removing green segment

Fig. 2: Graph partitioning via DijkstraFPS. (a): Synthetic example graph with edge costs. The costs are computed as the absolute value of gray value differences. (b)–(d): Result of a Dijkstra step traversing from the very left node and incrementally summing up edge weights. The shortest path to each node is indicated by the blue arrows and leads to a distance printed within the circular nodes. The *farthest point* is the one with distance $d = 0.44$. A second Dijkstra step starts at this farthest node and updates node distances until an update would increase the distance value, e.g. to the left $0.00 + 0.50 > 0.06$. A third Dijkstra step starts at the centre node and again updates node distances creating a third segment shown in green. (e): Decremental step removing the green segment. All green distances are initialized to ∞ and Dijkstra starts propagating a front from neighbouring segments. The boundary slightly changes compared to (c).



(a) Triangle-based



(b) Vertex-based

Fig. 3: Two possible graph structures for embedded graphs, here for 2D points (black dots). Graph nodes (circles) are either (a) Delaunay triangles or (b) Voronoi vertices. A possible segmentation boundary is shown as bold polyline along the Delaunay triangulation (a) or the Voronoi diagram (b).

4 Applications

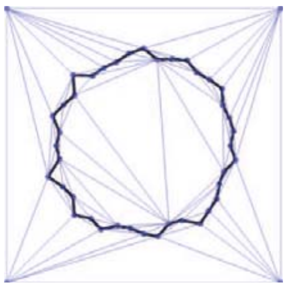
The DijkstraFPS graph partitioning algorithm is applicable to many problems that can be expressed as attributed graph. In this section we will demonstrate four examples: *surface triangulation* by partitioning the 3D space and *curve reconstruction* by partitioning the 2D space, *surface segmentation* by partitioning a triangulated 2D manifold and *curve segmentation* by partitioning a 1D polyline, *clustering* by partitioning points in an nD – possibly non-metric – space, and *image segmentation* by partitioning pixels or superpixels of the 2D image plane.

In this article we ignore the possibility to formulate an application-specific stopping criterion (SCHINDLER & FÖRSTNER 2011). Instead, we will define the number of segments of the incremental (L_{inc}) and decremental (L_{dec}) segmentation manually.

4.1 *Surface Triangulation*

The problem of triangulating a surface given an unordered set of 3D points is also known as meshing or surface reconstruction. The points are to be connected by triangles such that the triangulation approximates the original surface well. The problem can be easily transferred to 2D space where the boundary is a curve.

The underlying assumption is that the points sample the curve densely enough. A common measure for characterizing the sampling density is the ϵ-sampling: It depends on the *local feature size* of a point $\mathbf{x}$ on a curve γ. Given the *medial axis* as the set of points with more than


(a) 40 points on a circle with 2.5 % noise and 4 auxiliary corner points ($L_{\text{inc}} = 20$, $L_{\text{dec}} = 2$)

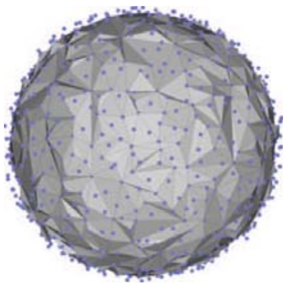
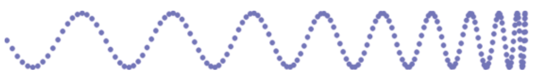

(b) 1000 points on a sphere with 1 % noise and 8 auxiliary points ($L_{\text{inc}} = 30$, $L_{\text{dec}} = 2$)

Fig. 4: Reconstruction results with the triangle-based graph structure (Fig. 3a). The original point cloud is shown with blue dots, the Delaunay triangulation with thin lines (omitted in 3D) and the boundary as bold polyline in 2D and surface in 3D.


(a) Sampled curve with increasing ϵ-sampling

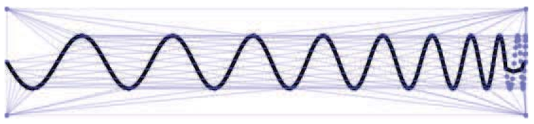

(b) Delaunay triangulation and curve reconstruction

Fig. 5: (a): Dependence on sampling density. The original 2D cloud of 200 points samples a curve with increasing ϵ-sampling. (b): The curve reconstruction using the triangle-based partitioning approach (Fig. 3a, $L_{\text{inc}} = 100$, $L_{\text{dec}} = 2$) yields correct results until it breaks at a point with $\epsilon = 1.02$, as the sampling density on the right-hand side of the curve is not high enough anymore referring to its increasing curvature.

one closest point on the curve γ, the local feature size $\text{lfs}(\mathbf{x})$ is the Euclidean distance of $\mathbf{x}$ to the medial axis. Then a set of points $\mathcal{X}$ is an ϵ-sample of a curve γ if every point $\mathbf{x} \in \gamma$ on the curve γ is within distance $\epsilon \cdot \text{lfs}(\mathbf{x})$ of some point in $\mathcal{X}$ (EDELSBRUNNER 1998). We will use this concept to empirically characterize the success of a segmentation procedure by the maximum value ϵ may have, as larger values of ϵ correspond to lower sampling density.

The idea of determining a 2D polyline or 3D surface triangulation via partitioning is to divide the space into simplices and to partition these simplices into "inside" and "outside". In 3D space we reconstruct a surface by partitioning a Delaunay tetrahedralization or the corresponding 3D Voronoi complex. In 2D this corresponds to the reconstruction of a curve by partitioning a Delaunay triangulation or the corresponding 2D Voronoi complex (Fig. 3).

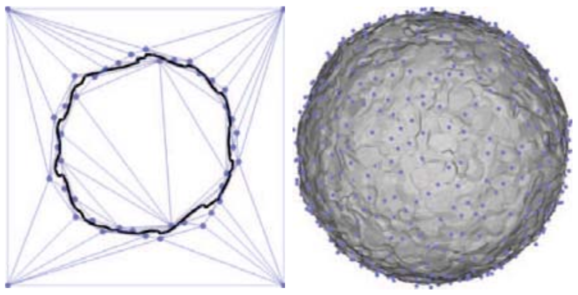
We choose to work with the Delaunay and Voronoi diagrams, since they are commonly used for this application. They were first exploited for surface reconstruction by BOISSONNAT (1984) and led to the concept of α-*shapes* (EDELSBRUNNER & MÜCKE 1994), *r-regular shapes* (ATTALI 1997) and the *crust* (AMENTA et al. 1998) with various derivatives like *conservative crust* (DEY et al. 1999), *power crust* (AMENTA et al. 2001) and *eigen-crust* (KOLLURI et al. 2004).

Since the partitioning divides all data points into two segments, the resulting boundary surface will be a closed surface within the Delaunay triangulated sampling volume.

We want to support the wave front to travel from an *inside* triangle to another *inside* or from an *outside* triangle to another *outside*, but hinder crossing the boundary. Under the above-mentioned sampling assumption a triangle edge that belongs to the boundary will be relatively short (Fig. 4a). Thus we construct our graph with edge costs simply being the inverse length of the common triangular edge. The inverse edge length can be raised to an exponent ≥ 2 in order to put more weight on very short edges, i.e. approximating an L_∞-norm.

The Delaunay triangulation is always bounded by the convex hull. If the desired segmentation boundary is part of the convex hull, it is not surrounded by triangles and thus can not be the boundary between differently labelled triangles. Therefore, we add auxiliary points to the original point cloud to extend the convex hull and avoid the boundary problem (Fig. 4a).

Fig. 4 shows results obtained with the described graph structure and edge costs on a synthetic 2D as well as on a 3D dataset. The point clouds are a circle with 40 points and a sphere with 1000 points disturbed with Gaussian noise. The circle is correctly reconstructed even when further reducing the number of points. The


(a) 40 points on a circle with 2.5 % noise and 4 auxiliary corner points ($L_{\text{inc}} = 20$, $L_{\text{dec}} = 2$)

(b) 1000 points on a sphere with 1 % noise and 8 auxiliary points ($L_{\text{inc}} = 30$, $L_{\text{dec}} = 2$)

Fig. 6: Reconstruction results with the vertex-based graph structure (Fig. 3b). The original point cloud is shown with blue dots, the Delaunay triangulation with thin lines (omitted in 3D) and the boundary as bold polyline in 2D and surface in 3D. The boundary polyline/surface is drawn by linking edge centres of differently labelled vertices with the triangular/tetrahedral centroid.

topology of points on the surface of a sphere is a bit more complicated leading to some points not included in the triangulation. Their large deviation from the underlying surface contradicts the above-mentioned sampling assumption. The overall shape, however, is preserved.

In Fig. 5 we investigate the dependence on a dense sampling. The point cloud is generated as $(x, y) = (-t^2/2, -\sin(10t))$ with 200 uniformly sampled values $t \in [0, 2\pi]$. Since the x-intervals decrease from left to right, the local ϵ-sampling increases. The reconstruction via partitioning the Delaunay complex is shown as a bold line. It yields visually correct results until it breaks at a point with $\epsilon = 1.02$. In comparison: The popular *Crust* (AMENTA et al. 1998) and *Power Crust* (AMENTA et al. 2001) algorithms guarantee correct results only for $\epsilon < 0.252$ – TCHERNIAVSKI & STELLDINGER (2008) even claim $\epsilon < 0.1$ – while *Cocone* (AMENTA et al. 2000) and *Tight Cocone* (DEY & GOSWAMI 2003) require $\epsilon < 0.06$. Of course our test is much weaker than a theoretical proof, but still indicates robustness to rather low sampling density.

The alternative vertex-based structure would be the dual graph: Instead of representing triangles by nodes we use the vertices and link them by graph edges equivalent to the Delaunay edges. This representation might be better suited if the boundary sample points are disturbed by random noise such that an approximation is desired. By partitioning the vertices into *inside* and *outside* we obtain a boundary curve lying *in between* data points.

Graph edges crossing the boundary curve are usually rather short (Fig. 6a). Thus we define the edge costs as the inverse length of the crossed triangle edge.

Fig. 6 shows results obtained with this alternative graph structure on the very same synthetic datasets as in Fig. 4. The boundary is correctly reconstructed in between the given data points.

4.2 *Surface Segmentation*

The problem of segmenting a surface requires an algorithm to partition a surface into usually compact segments with similar elements – w. r. t. pre-defined properties. Here, we assume the surface to be represented by a triangular mesh, at each vertex being isomorphic to a disk.

To our knowledge there is only one other work that applies a front propagation method to surface segmentation: PAGE et al. (2003) use FMM as a final region growing step called "marching watersheds", referring to the popular watershed segmentation (BEUCHER & LANTUEJOUL 1979) that has been implemented for meshed surfaces by MANGAN & WHITAKER (1999).

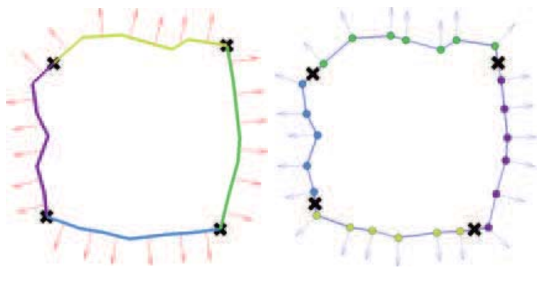
For segmenting a surface in 3D space we will work with the very same graph structure as for reconstructing a boundary curve in 2D since we again want to partition a 2D manifold. Now, however, the manifold is non-planar and we exploit other geometric properties for defining edge costs.

For many applications in surface reconstruction and object modelling one wants to partition a triangular mesh into piece-wise planar regions. Thus we introduce high costs for graph edges linking non-planar nodes. For the triangle-based graph structure it is natural to compute triangle normals and derive costs from the angle between two of them. For the vertex-based graph structure vertex normals are needed that can be computed via principal component analysis from neighbouring points (HOPPE et al. 1992), possibly robustified by computing the coordinate-wise median (ROUSSEEUW & LEROY 1987, section 3.2.1) of multiple neighbouring normals (SCHINDLER & FÖRSTNER 2011).

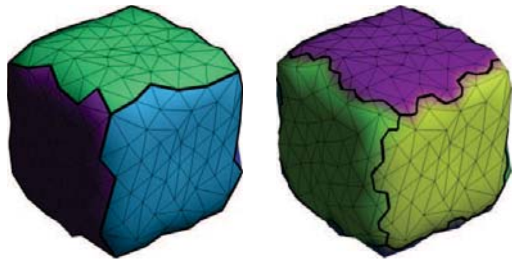
Analogous to partitioning adjacent triangles in 3D space we can partition straight line segments in 2D space as it is commonly done using the Ramer-Douglas-Peucker algorithm (RAMER 1972, DOUGLAS & PEUCKER 1973). Again the edge costs are derived from the angle between two line segments or – in the vertex-based graph structure – between two points.

Fig. 7 shows resulting segmentations in both 2D and 3D as well as for both graph structures. In the 2D examples the line and vertex normals are shown as red and blue arrows, respectively. The final segmentation is indicated with coloured triangles, lines and vertices. Even though both the cube and the square have rounded corners and the points are disturbed by 3 % (square) and 1 % (cube) Gaussian noise, the DijkstraFPS segmentation yields visually pleasing results.

Fig. 8 shows the segmentation result on a real dataset: It was reconstructed from 159 images using Bundler (SNAVELY et al. 2006), PMVS2 (FURUKAWA & PONCE 2010) and Poisson surface reconstruction (KAZHDAN et al. 2006).



(a) $L_{\text{inc}} = 10$, $L_{\text{dec}} = 4$ **(b)** $L_{\text{inc}} = 10$, $L_{\text{dec}} = 4$



(c) $L_{\text{inc}} = 50$, $L_{\text{dec}} = 6$ **(d)** $L_{\text{inc}} = 50$, $L_{\text{dec}} = 6$

Fig. 7: Segmentation of a polyline in 2D space (top row: 24 points on a square, 3 % noise) and a surface in 3D space (bottom row: 240 points on a cube, 1 % noise) using the triangle-based (left) and vertex-based (right) graph structure. Edge costs are defined as the angle of the two normals. The normals are either computed for triangles, for lines or for vertices, depending on the dimensionality and the graph structure. In the 2D case they are shown as red and blue arrows. The resulting segmentation is indicated as coloured triangles, lines and points, respectively, as well as a bold, black boundary.

4.3 *Clustering*

Within this section we focus on clustering data points via graph partitioning. We choose to create undirected links between each graph node and its $k = 25$ nearest neighbours (k-NN). In contrast to, e.g. Delaunay triangulation the complexity for k-NN is independent of the dimensionality. Edge costs are the squared Euclidean distance, possibly raised to an exponent ≥ 2.

We compare DijkstraFPS to common clustering algorithms on two synthetic datasets (Tab. 1):

The *k-means* algorithm (MACQUEEN 1965 iteratively computes the mean coordinates per class and updates class labels according to the closest mean. In case of different scatter the second step yields incorrect labels (dataset A).

The *Expectation–maximization* (EM) algorithm (DEMPSTER et al. 1977) not only estimates the means but also the class variances, thus yields correct results (up to one falsely classified point) for dataset A. If it, however, assumes Gaussian distributions, it does not succeed with dataset B.

The *mean shift* clustering (COMANICIU & MEER 2002) iteratively replaces points by the centre of neighbouring points within a certain radius. It yields correct results for dataset A, but possibly too many clusters in the second example depending on the chosen radius. Here, a larger radius yields a similar result like *k-means*.

DijkstraFPS returns perfect point labels on both datasets shown in Tab. 1, certainly benefiting from the long edges between points of different clusters. On datasets with touching or overlapping distributions DijkstraFPS might fail.

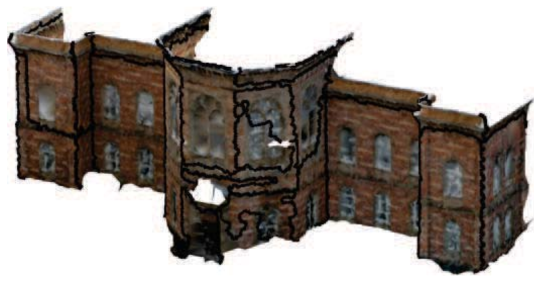
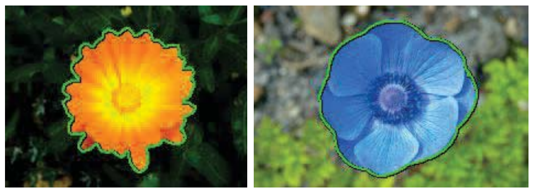
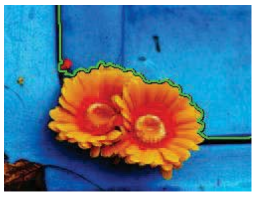


Fig. 8: Surface segmentation on a visually reconstructed, meshed point cloud.



(a) FLOWER 1 ($L_{\mathrm{inc}} = 100$, $L_{\mathrm{dec}} = 2$) **(b)** FLOWER 2 ($L_{\mathrm{inc}} = 100$, $L_{\mathrm{dec}} = 2$)



(c) FLOWER 3 ($L_{\mathrm{inc}} = 200$, $L_{\mathrm{dec}} = 2$)

Fig. 9: DijkstraFPS image segmentation. Three test images are segmented using our proposed graph partitioning scheme. Edge costs are computed as squared Euclidean distances in hue-saturation-value (HSV) colour space. While FLOWER 1 (a) and FLOWER 2 (b) are perfectly segmented, in FLOWER 3 (c) the dark background edges attract the segmentation boundary more.

4.4 *Image Segmentation*

FPS has been used for progressive image sampling by ELDAR et al. (1997). As "farthest point" they choose a vertex of the Euclidean Voronoi diagram that maximizes some weighted distance function. Combining FPS with a marching scheme like FMM or Dijkstra's algorithm we can approximate the non-Euclidean Voronoi diagram and thus derive not only a suitable image sampling but an image segmentation as well.

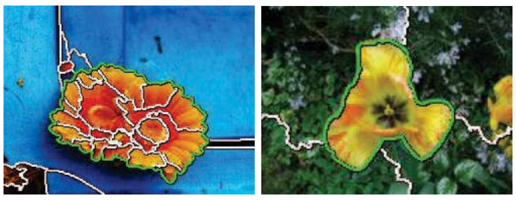
The graph structure is inherently given by the pixel grid. One can, however, choose between a 4-neighbourhood (pixels horizontally and vertically connected), an 8-neighbourhood (pixels additionally diagonally connected) or a triangulation (diagonal connections in one direction only).

In contrast to other image segmentation algorithms we can use arbitrary colour similarity measures as edge costs and thus are not restricted to gray images or Euclidean colour spaces. We compute the Euclidean distance of two pixels in the hue-saturation-value (HSV) colour cone.

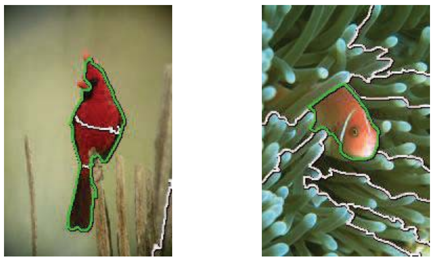
Fig. 9 shows three segmentation results. While FLOWER 1 and FLOWER 2 are successfully segmented, the approach fails for FLOWER 3. Instead of segmenting "flower" and "not flower" the *unsupervised* segmentation draws the boundary along the dark bold edges in the background.

Since the DijkstraFPS graph partitioning is not restricted to a regular pixel grid, we can work with irregular image regions such as superpixels. Further we can generate superpixels with DijkstraFPS by partitioning the image into more than two segments, e.g. 30 segments as indicated by the white boundaries in Fig. 10a. Then we build a second graph with only 30 nodes and edges according to the superpixel adjacencies in the image. Edge costs are derived, e.g. from the average colours of the superpixels. As can be seen from the green boundary in Fig. 10a the final segmentation is very accurate due to the flexibility of 30 initial segments but also robust due to the larger number of pixels that contribute to the superpixel colour. Results for two images from the Berkeley Segmentation Dataset and Benchmarks 500 (BSDS500 ARBELAEZ et al. 2011) are given in Figs. 10c and 10d.

For large images DijkstraFPS is rather slow, since it is very general and not optimized for the regular image grid. Of course other superpixel generating algorithms (LEVINSHTEIN et al. 2009, ACHANTA et al. 2012) can be used and might save significant computing time. The second segmentation step would be performed using DijkstraFPS. An evaluation of different superpixel schemes combined with our DijkstraFPS graph partitioning might be reasonable but is out of scope of this article.



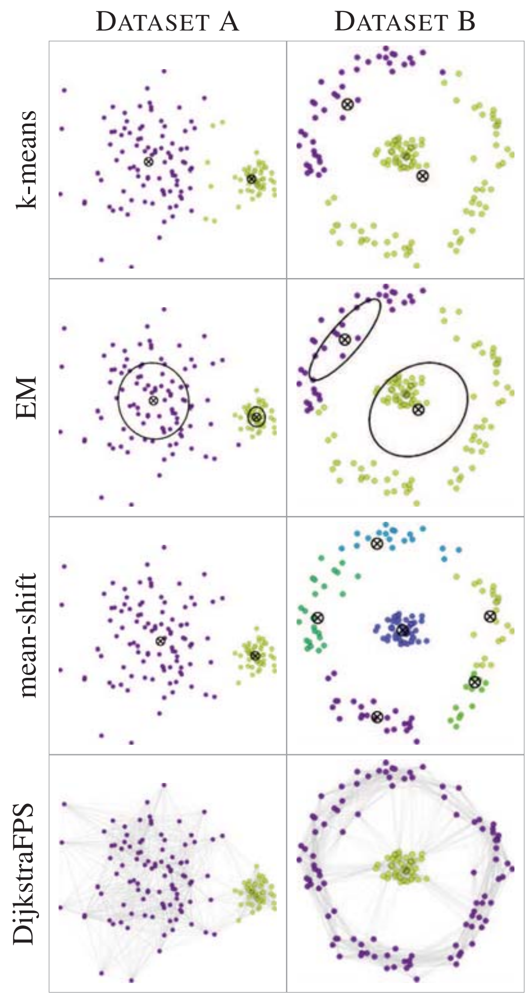
(a) FLOWER 3 ($L_{\text{inc}} = 200, L_{\text{dec}} = 30$) **(b)** FLOWER 4 ($L_{\text{inc}} = 100, L_{\text{dec}} = 5$)



(c) 196027 (ARBELAEZ et al., 2011) ($L_{\text{inc}} = 200, L_{\text{dec}} = 5$) **(d)** 210088 (ARBELAEZ et al., 2011) ($L_{\text{inc}} = 100, L_{\text{dec}} = 10$)

Fig. 10: Progressive image segmentation. (a): FLOWER 3 image from Fig. 9c segmented into 30 segments, shown with white boundaries. Then a graph with only 30 nodes is built from the obtained segmentation and again partitioned using DijkstraFPS, yielding the green boundary. (b), (c) and (d): More examples.

Tab. 1: Clustering results of three common clustering algorithms and our DijkstraFPS graph partitioning. DijkstraFPS ($L_{\text{inc}} = 10$, $L_{\text{dec}} = 2$) yields 100 % correct results in both examples. The ellipses are 1-σ confidence intervals.



5 Conclusion

We proposed a new graph partitioning scheme based on Dijkstra's distance transform and farthest point sampling (DijkstraFPS) and showed how to construct graphs for solving various problems in geometry and image processing. The partitioning is guided by pre-defined edge costs that do not have to follow a Euclidean metric. The method is not restricted to a binary partition but can yield multiple segments, possibly in terms of an oversegmentation or – in terms of image segmentation – superpixels. The latter themselves can be nodes of a subsequent graph partitioning.

As shown in SCHINDLER & FÖRSTNER (2011) the algorithm can be augmented with an automatic, application-specific stopping criterion. Establishing such criteria for each abovementioned application remains future work.

There is much space for improving the proposed edge costs. We focused on demonstrating the diversity of DijkstraFPS graph partitioning rather than on fine-tuning for specific applications.

Other possible applications of the method include triangulation and segmentation of full wave-form lidar point clouds, segmentation of radar images, hierarchical image segmentation as indicated in section 4.4, segmentation of image sequences and image segmentation based on texture similarity, which of course have to be empirically compared to standard algorithms used in that application.

References

ACHANTA, R., SHAJI, A., SMITH, K., LUCCHI, A., FUA, P. & SÜSSTRUNK, S., 2012: Slic superpixels compared to state-of-the-art superpixel methods. – IEEE Transactions on Pattern Analysis and Machine Intelligence **34** (11): 2274–2282.

AMENTA, N., BERN, M. & EPPSTEIN, D., 1998: The crust and the β-skeleton: Combinatorial curve reconstruction. – Graphical Models and Image Processing **60**: 125–135.

AMENTA, N., CHOI, S., DEY, T.K. & LEEKHA, N., 2000: A simple algorithm for homeomorphic surface reconstruction. – Sixteenth Annual Symposium on Computational Geometry.

AMENTA, N., CHOI, S. & KOLLURI, R., 2001: The power crust. – Sixth ACM Symposium on Solid Modeling and Applications.

ARBELAEZ, P., MAIRE, M., FOWLKES, C. & MALIK, J., 2011: Contour detection and hierarchical image segmentation. – IEEE Transactions on Pattern Analysis and Machine Intelligence **33** (5): 898–916.

ARBELAEZ, P., HARIHARAN, B., GU, C., GUPTA, S., BOURDEV, L.D. & MALIK, J., 2012: Semantic segmentation using regions and parts. – Computer Vision and Pattern Recognition.

ATTALI, D., 1997: r-regular shape reconstruction from unorganized points. – Thirteenth Annual Symposium on Computational Geometry: 248–253.

BEUCHER, S. & LANTUEJOUL, C., 1979: Use of watersheds in contour detection. – International Workshop on Image Processing, Real-time Edge and Motion Detection.

BOISSONNAT, J.-D., 1984: Geometric structures for three-dimensional shape representation. – ACM Transactions on Graphics **3**: 266–286.

BOYKOV, Y. & FUNKA-LEA, G., 2006: Graph cuts and efficient n-d image segmentation. – International Journal of Computer Vision **70** (2): 109–131.

COMANICIU, D. & MEER, P., 2002: Mean shift: a robust approach toward feature space analysis. – IEEE Transactions on Pattern Analysis and Machine Intelligence **24** (5): 603 –619.

DEMPSTER, A.P., LAIRD, N.M. & RUBIN, D.B., 1977: Maximum likelihood from incomplete data via the em algorithm. – Journal of the Royal Statistical Society **39** (1): 1–38.

DEY, T.K., MEHLHORN, K. & RAMOS, E.A., 1999: Curve reconstruction: Connecting dots with good reason. – Fifteenth Annual Symposium on Computational Geometry.

DEY, T.K. & GOSWAMI, S., 2003: Tight cocone: A water-tight surface reconstructor. – Eighth ACM Symposium on Solid Modeling and Applications.

DIJKSTRA, E.W., 1959: A note on two problems in connexion with graphs. – Numerische Mathematik **1** (1): 269–271.

DOUGLAS, D.H. & PEUCKER, T.K., 1973: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. – Cartographica: The International Journal for Geographic Information and Geovisualization **10** (2): 112–122.

EDELSBRUNNER, H. & MÜCKE, E.P., 1994: Three-dimensional alpha shapes. – ACM Transactions on Graphics **13**: 43–72.

EDELSBRUNNER, H., 1998: Shape reconstruction with delaunay complex. – Lecture Notes in Computer Science **1380**: 119–132.

ELDAR, Y., LINDENBAUM, M., PORAT, M. & ZEEVI, Y.Y., 1997: The farthest point strategy for progressive image sampling. – IEEE Transactions on Image Processing **6** (9): 1305–1315.

FORSYTH, D.A. & PONCE, J., 2002: Computer vision: a modern approach. – Prentice Hall Professional Technical Reference.

FURUKAWA, Y. & PONCE, J., 2010: Accurate, dense, and robust multi-view stereopsis. – IEEE Transactions on Pattern Analysis and Machine Intelligence **32** (8): 1362–1376.

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J. & STUETZLE, W., 1992: Surface reconstruction from unorganized points. – Nineteenth Annual Conference on Computer Graphics and Interactive Techniques: 71–78.

KAZHDAN, M., BOLITHO, M. & HOPPE, H., 2006: Poisson surface reconstruction. – Fourth Eurographics symposium on Geometry processing, SGP'06: 61–70, Eurographics Association, Aire-la-Ville, Switzerland.

KIMMEL, R. & SETHIAN, J.A., 1998: Computing geodesic paths on manifolds. – National Academy of Sciences **95**: 8431–8435.

KOLLURI, R., SHEWCHUK, J.R. & O'BRIEN, J.F., 2004: Spectral surface reconstruction from noisy point clouds. – Eurographics Symposium on Geometry Processing.

LEVINSHTEIN, A., STERE, A., KUTULAKOS, K.N., FLEET, D.J. & DICKINSON, S.J., 2009: Turbopixels: Fast superpixels using geometric flows. – IEEE Transactions on Pattern Analysis and Machine Intelligence **31** (12): 2290–2297.

MACQUEEN, J., 1965: Some methods for classification and analysis of multivariate observations. – Fifth Berkeley Symposium on Mathematical Statistics and Probability.

MANGAN, A.P. & WHITAKER, R.T., 1999: Partitioning 3d surface meshes using watershed segmentation. – IEEE Transactions on Visualization and Computer Graphics **5** (4): 308–321.

MEINE, M. & KÖTHE, U., 2005: Image segmentation with the exact watershed transform. – Visualization, Imaging and Image Processing.

MESTER, R., CONRAD, C. & GUEVARA, A., 2011: Multichannel segmentation using contour relaxation: Fast super-pixels and temporal propagation. – Scandinavian Conference on Image Analysis.

Moenning, C. & Dodgson, N.A., 2003a: Fast marching farthest point sampling. – Eurographics 2003, Granada, Spain.

Moenning, C. & Dodgson, N.A., 2003b: Fast marching farthest point sampling for point clouds and implicit surfaces. – Technical Report 565. University of Cambridge, Computer Laboratory.

Page, D.L., Koschan, A.F. & Abidi, M.A., 2003: Perception-based 3d triangle mesh segmentation using fast marching watersheds. – Computer Vision and Pattern Recognition: 27–32.

Peyré, G. & Cohen, L., 2004: Surface segmentation using geodesic centroidal tesselation. – Second International Symposium on 3D Data Processing, Visualization and Transmission: 995–1002.

Peyré, G. & Cohen, L., 2006: Geodesic remeshing using front propagation. – International Journal of Computer Vision **69**: 145–156.

Ramer, U., 1972: An iterative procedure for the polygonal approximation of plane curves. – Computer Graphics and Image Processing **1** (3): 244–256.

Roscher, R., 2012: Sequential Learning using Incremental Import Vector Machines for Semantic Segmentation, PhD thesis. Univerity of Bonn.

Rother, C., Kolmogorov, V. & Blake, A., 2004: Grabcut: Interactive foreground extraction using iterated graph cuts. – ACM Transactions on Graphics **23**: 309–314.

Rousseeuw, P.J. & Leroy, A.M., 1987: Robust regression and outlier detection. – John Wiley & Sons, Inc.

Schindler, F. & Förstner, W., 2011: Fast marching for robust surface segmentation. – Photogrammetric Image Analysis, Lecture Notes in Computer Science **6952**: 147–158.

Sethian, J.A., 1996: A fast marching level set method for monotonically advancing fronts. – National Academy of Sciences of the USA **93**: 1591–1595.

Shi, J. & Malik, J., 2000: Normalized cuts and image segmentation. – IEEE Transactions on Pattern Analysis and Machine Intelligence **22** (8): 888–905.

Snavely, N., Seitz, S.M. & Szeliski, R., 2006: Photo tourism: Exploring image collections in 3d. – ACM **25**: 835–846.

Szeliski, R., 2010: Computer Vision: Algorithms and Applications. – Springer.

Tcherniavski, L. & Stelldinger, P., 2008: A thinning algorithm for topologically correct 3d surface reconstruction. – International Conference on Visualization, Imaging and Image Processing: 119–124.

Veksler, O., Boykov, Y. & Mehrani, P., 2010: Superpixels and supervoxels in an energy optimization framework. – European Conference on Computer Vision ECCV **10**: 211–224.

Vincent, L. & Soille, P., 1991: Watersheds in digital spaces: An efficient algorithm based on immersion simulations. – IEEE Transactions on Pattern Analysis and Machine Intelligence **13**: 583–598.

Address of the Authors:

Dipl.-Ing. Falko Schindler, Rheinische-Friedrich-Wilhelms-Universität Bonn, Institut für Geodäsie und Geoinformation, Photogrammetrie, Nußallee 15, D-53115 Bonn, Tel.: +49-228-73-2908, e-mail: falko.schindler@uni-bonn.de.

Prof. Dr.-Ing. Dr. h. c. mult. Wolfgang Förstner, Josef-Schell-Str. 34, D-53121 Bonn, e-mail: wf@ipb.uni-bonn.de

Manuskript eingereicht: Februar 2013
Angenommen: April 2013