The GeoMap: A Unified Representation for Topology and Geometry

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. We propose the GeoMap abstract data type as a unified representation for image segmentation purposes. It manages both topology (based on XPMaps) and pixel-based information, and its interface is carefully designed to support a variety of automatic and interactive segmentation methods. We have successfully used the abstract concept of a GeoMap as a foundation for the implementation of well-known segmentation methods.

1 Introduction

The goal of image segmentation is to identify regions that are conceptually coherent and serve as a basis for further analysis steps. Segmentation methods rely on local information on both direct properties of pixels and regions and the neighborhood. Today, computer vision researchers agree that correct handling of topology is needed when dealing with regions and boundaries, in order to avoid problems like the connectivity paradox.

Information on neighborhood-relations is conveniently stored in graph structures like the well-known *region adjacency graphs* (RAG [1]). These structures differ in expressiveness; some have problems with representing certain configurations occurring in image analysis (separate contours / holes, see e.g. [2]). Thus, a number of advanced formalisms for finite topology have been proposed for solving these problems [3,2,4,5]. Another problem related to these graph structures is that usually, the geometry of the regions is stored separately (in so-called label images, edgel lists, or the like), and an algorithm has to modify both the graph and the external data when for example regions are merged. This puts the burden of preventing inconsistencies between the graph representation and the pixel geometry on the user (i.e. developer of the algorithm).

Furthermore, there are several possible definitions of regions and boundaries in discrete images - like crack edges, 8-connected boundaries between 4-connected regions or vice versa, or working with a hexagonal grid (some examples follow in Sect. 2, see Fig. 1 on page 4) - but they cannot be used interchangeably, since algorithms usually work directly on the pixel layer. We can generalize algorithms by formulating them on a higher abstraction level, and managing all relations between the topology and geometry on the pixel level in one abstract data type.

The GeoMap we introduce here will i) allow to work on a *natural abstraction level* with faces, edges, and vertices as basic entities (resulting in more concise, readable and reusable code), while ii) offering access to *both their neighborhoods and their associated pixels* at any time. This leads to *considerable advantages*: Having a common, unified representation for different automatic and interactive segmentation algorithms makes it possible to use them not only alternatively, but also together on one image. Furthermore, it facilitates the separation of the basic segmentation approach i) from the definition of topology on the pixel layer, but also from e.g. ii) cost definitions driving an optimization process, and thus allows to recombine parts from different publications.

2 The GeoMap Concept

As mentioned above, the GeoMap builds upon the XPMap formalism [5], and extends it by integrating the required geometrical information. We will now formally introduce the concept of a GeoMap, then carefully design an application interface suitable to exploit the advantages of our unified representation in Sect. 2.1, and finally propose a possible internal representation for our abstract data type (ADT) in Sect. 2.2. First, we need to define 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 orbits of σ*,* α*, and the composed permutation* $\varphi = \sigma^{-1} \circ \alpha$ *are called* vertices*,* edges*, and* faces *respectively.*

A combinatorial map is *planar*, if and only if its number of vertices, edges, and faces fulfills Euler's equation ($|\alpha|$ denotes the number of orbits in α):

$$|\sigma| - |\alpha| + |\varphi| = 2 \tag{1}$$

An obstacle when trying to use planar combinatorial maps for image segmentation is that they cannot represent multiple boundary sets, which occur if we have regions with holes.

A common solution is to introduce auxiliary bridges which connect the contours (cf. [2]), but this complicates further handling, since i) algorithms working with edges have to explicitly check for these, and ii) there is no naturally defined place where these bridges should be attached to the contours. The latter becomes even more bothersome when we add geometrical information to the combinatorial structure. Then the auxilliary bridges also need geometric representations, which is unnecessary and may even be impossible if the geometry is defined with finite resolution as in our pixel-based approaches below. Furthermore, such

bridges are undesirable if they have to be distinguished from "real" bridges that represent incomplete boundaries information.

We avoid auxilliary bridges by means of the XPMap formalism [5]:

Definition 2. *We call a tuple* $(C, c_0, \mathsf{exterior}, \mathsf{contains})$ extended planar map (XPMap) *where* C *is a set of non-trivial planar combinatorial maps (the components of the XPMap),* c_0 *is a trivial map that represents the infinite face of the XPMap,* exterior *is a relation that labels one* φ*-orbit of each component in* C *as the exterior orbit, and* contains *is a relation that assigns each exterior orbit to exactly one non-exterior* φ*-orbit or to the infinite face of* c_0*.*

Note that an XPMap naturally defines permutations σ, α, and φ, which are simply the compositions of all permutations of the combinatorial maps in C.

XPMaps are a powerful representation for finite topology and suitable for image segmentation; however, segmentation algorithms are normally not entirely topology-based, but in general need to access the geometry and other (pixel-) properties of the boundaries and regions, such as brightness and gradient. Due to this important observation, we will now introduce the GeoMap.

Consider a complete partitioning of the plane into a set $\mathcal{P}$ of open regions that we call *basis cells* (which normally correspond to pixels or Khalimsky cells [6]). Furthermore, consider a relation $\mathsf{dim}\colon \mathcal{P} \to \{0,1,2\}$ that assigns a *dimension* to each basis cell. We then group connected basis cells of the same dimension into *block cells* according to the following rules (where $\mathcal{P}_d := \{p \in \mathcal{P} | \mathsf{dim}\,(p) = d\}$):

$$V := CC\left[\bigcup_{\mathcal{P}_0} p^c\right],\ E := CC\left[\left(\bigcup_{\mathcal{P}_1} p^c\right) \setminus \bigcup V\right],\ F := CC\left[\left(\bigcup_{\mathcal{P}_2} p^c\right) \setminus \left(\bigcup V \cup \bigcup E\right)\right]$$

where p^c denotes the closure of p and $CC\,[\ldots]$ is the set of connected components. These three types of block cells are called *vertices*, *edges*, and *faces* respectively. Fig. 1 shows some example $(\mathcal{P}, \mathsf{dim})$ pairs; these variants will be discussed in Sect. 2.2.

The *neighborhood* of a block cell c is defined as $N(c) := \{c_i \,|\, c \cup c_i \text{ is connected}\}$ where $c, c_i \in V \cup E \cup F$. Note that $N(c)$ will never contain cells $c_i \neq c$ of the same type as c, since the basis cells would have identical types and thus be combined into one connected component.

If all vertices and edges are simply connected (i.e. have no holes), and $\forall e \in E : ((|N\,(e)| \geq 3) \wedge (N\,(e) \cap V \leq 2))$ holds, we can represent the discrete topology of the block cells with an XPMap [7], and use this to build a GeoMap.

Definition 3. *A* GeoMap *is a tuple* $(\mathcal{P}, X, g)$ *where* $\mathcal{P}$ *is a set of basis cells,* X *is an XPMap that represents their induced topology, and* $g : V \cup E \cup F \to 2^{\mathcal{P}}$ *is a relation that assigns the set of contained basis cells to each block cell.*

The handling of both basis- and block cells is simplified by introducing labels: We require each basis cell p to have a unique label $b = \mathsf{label}\,(p)$ (usually, this will be the pixel coordinate) and assign unique labels l to the block cells. Note that we do not require the block cell labels to be continuous, since this leads to difficulties later with modifications which remove cells.

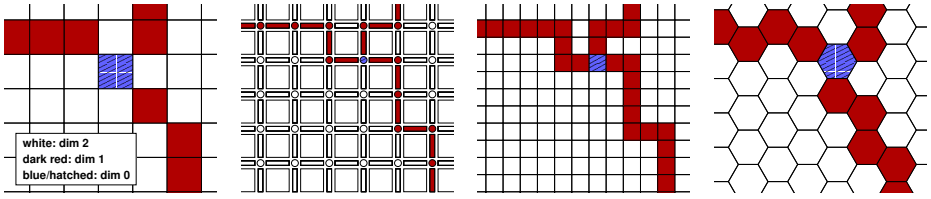

Fig. 1. Example GeoMap cells (left to right: 8-connected pixel boundaries, inter-pixel interpretation, explicit crack edges, example boundary on hexagonal pixel grid)

2.1 GeoMap Interface Design

In order to make the GeoMap a useful representation in practice, it is important that we define an abstract interface that reflects all requirements of segmentation algorithms. In [8] we systematically examined these needs of several algorithms; the results will be summarized in the following.

Topology Queries There are several topology-related tasks that must be supported: Testing whether two regions are adjacent, querying all boundary components of a face, and listing all adjacent faces.

Cell Geometry Queries Segmentation algorithms frequently need to know the shape of block cells, for example for collecting statistics on their basis cells' properties (i.e. boundary strength, mean region color).

Inverse Geometry Queries Interactive segmentation requires a mapping from a specific basis cell (e.g. position obtained with a pointing device) to the block cell at that position.

Transformations Considering segmentation as the transformation of an initial partitioning of the image plane into the desired result, we need operations like removing single edges or completely merging faces.

Application-Specific Data Algorithms will rely on application-specific properties of the cells, for example to decide about which regions to merge. Thus, it must be possible to store and update this information in such a way that it is kept consistent with the current segmentation.

The last requirement is satisfied through the association of labels with each block cell, which can be used to index arrays with application-specific data; it is important however that these labels do not change in undefined ways. We will now explain solutions to the other tasks in detail, and then illustrate our implementation in Sect. 2.2.

Topology Queries: The DartTraverser Concept Since the GeoMap is based on XPMaps, the basic entities used to encode its topology are *darts*. In order to make inspection of the topology as easy as possible, we introduce the DartTraverser concept, which uses a dart d to represent the current position during navigation within the GeoMap.

A GeoMap defines the permutations σ, α, and φ on its darts. The current position of a DartTraverser can be changed by moving to the successor or predecessor of the current dart in σ (which corresponds to turning around the vertex), α (jumping to the opposite side of the edge), or the composed permutation $\varphi = \sigma^{-1} \circ \alpha$ (following the contour of the face to the left):

nextSigma: $d := \sigma(d)$ nextAlpha: $d := \alpha(d)$ nextPhi: $d := \phi(d)$

prevSigma: $d := \sigma^{-1}(d)$ prevAlpha: $d := \alpha^{-1}(d)$ prevPhi: $d := \phi^{-1}(d)$

Now we do not only want to navigate on the darts, but we also want to access any information associated with the vertices, edges, or faces, so the DartTraverser interface also allows to query the identifying labels of the vertex which the current dart is attached to (represented by the orbit $\sigma^*(d)$), the edge it belongs to ($\alpha^*(d)$) and the face to the left ($\varphi^*(d)$).

startNodeLabel: $d \mapsto \mathsf{label}(\sigma^*(d))$ endNodeLabel: $d \mapsto \mathsf{label}(\sigma^*(\alpha(d)))$

edgeLabel: $d \mapsto \mathsf{label}(\alpha^*(d))$

leftFaceLabel: $d \mapsto \mathsf{label}(\phi^*(d))$ rightFaceLabel: $d \mapsto \mathsf{label}(\phi^*(\alpha(d)))$

Geometry Queries As stated above, there need to be means to i) get the block cell associated with a given basis cell or to ii) query all basis cells belonging to one block cell. In order to answer the first question, the GeoMap offers

$$\text{cellAt: } b \mapsto l$$

which returns the label l of the block cell for which $g(l)$ contains the basis cell labelled b. The second task - finding all basis cells belonging to a cell - is usually closely related to collecting properties of these basis cells (e.g. finding the mean color, inspecting the gradient, calculating the center of mass). This can be accomplished by querying the GeoMap for a CellScanIterator:

$$\text{cellScanIterator: } l \mapsto \textsc{CellScanIterator}(\{\mathsf{label}(p) \,|\, p \in g(l)\})$$

This CellScanIterator is then used to iterate over the basis cell labels, which are needed in order to look up the properties for the corresponding cells.

Transformations Since image segmentation is a dynamic process, the GeoMap would be useless without means for modification. An important design decision for this part of the interface is that the GeoMap should offer a small set of simple transformations, which makes formal correctness proofs possible and ensures that the representation stays in a consistent state. Non-admissible transformations can be rejected by checking the preconditions of each operation.

Köthe [5] proposes a set of Euler Operators [9] for image segmentation; these are operators that leave Euler's equation on the number of cells and connected boundary components $|C|$ in a planar XPMap valid: $|V| - |E| + |F| - |C| = 1$. We define the following operations on the GeoMap G, which all take the form $G, d \mapsto G'$ and return a $G' = (\mathcal{P}, X', g')$ with X' and g' created from X and g as follows:

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 $(|V'| = |V| - 1,\ |E'| = |E| - 1)$

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

remove isolated vertex merge an isolated vertex represented by the empty orbit $\alpha^*(d)$ into the surrounding face $\varphi^*(d)$ $(|V'| = |V| - 1,\ |C'| = |C| - 1)$

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 $(|E'| = |E| - 1,\ |F'| = |F| - 1)$

The relation g' is derived from g by assigning the basis cells of all cells being merged to the resulting cell.

Note that each of the above operations is a reduction (reducing the number of cells). Conceptually, they all have inverse operations that could be used to e.g. split block cells, effectively creating new ones from the same basis cells. Additionally, split operations could be applied on basis cells, which would change $\mathcal{P}$. However, adding geometric information introduces an asymmetry between split and merge operations (the former are no longer parametrizable with a single dart), which is why split operations are beyond the scope of this paper.

Note that the GeoMap handles both updating the geometry and the topology, but the application-specific information on the cells has to be updated by the application. Usually, it is possible to combine the statistics of the cells being merged in order to get the statistics of the resulting cell, and a GeoMap implementation can provide hooks for callbacks to ensure that this happens.

Building GeoMap Pyramids We consider segmentation as the transformation of an initial partitioning of the image plane into the desired result. Usually, the initial tessellation is an oversegmentation as resulting from a watershed transform or optimal cut [10], or the trivial one where every pixel is a separate region (i.e. the first segmentation step is to look for *any* boundary evidence). In this setting, further segmentation stages can be computed by using the above reduction operators, and one can arrange the results over time in a pyramid, where each level contains less cells than the one below. This corresponds to the approach of irregular pyramids [3,2,4], which can be used to create more coarse, abstracting segmentations without losing the ability to represent important detail.

2.2 CellImage Realization

So far, we have concentrated on the abstract properties of a GeoMap; now we focus on a possible implementation.

We propose a straight-forward extension of the common label images as internal representation for a GeoMap: The geometry information is stored in a CellImage, the pixels of which are the basis cells and each carry a dimension (specifying their type - vertex, edge, or face pixel) and the label of the block cell they belong to. The complete topological information is derived from this internal representation (see Fig. 2) according to Definition 3, which offers consistent views on the same segmentation from both perspectives.

The relation dim (page 3) is crucial for the correct derivation of topology from the basis cells (i.e. pixels). In the past, researchers have concentrated on crack edge-based interpretation of region images (inter-pixel boundaries, [11,4,1]), but it has also been shown that a topological representation can be derived from "thin", 8-connected boundaries (resulting from a watershed segmentation for example) [7]. Note that inter-pixel contours are commonly made explicit by doubling the image size and inserting boundary pixels, but the resulting 4-connected contours are visually much less appealing than 8-connected contours due to strong staircase effects. On the plus side, inter-pixel nodes always consist of one basis cell and have limited degree, which makes crack edge contours much easier to use. Definition 3 allows for both inter-pixel contours and explicitly represented ones on square or hexagonal pixel grids. We will concentrate on the interesting case of 8-connected boundaries in the following, since it has not yet received as much attention as the inter-pixel approaches.

Moving in the Orbits Lets have a look at how navigation through the topology works in some examples. Our internal representation of a DartTraverser is a pixel position and a direction, pointing to one of the pixel's neighbors (indicated by the arrows in Fig. 2).

α-orbit: Finding the opposite half-edge is accomplished by simply following the edge pixel-wise to the next vertex pixel and turning around. Both crack-edges and the boundary pixel classification by Köthe [7] guarantee by definition that each edge pixel has a unique successor and predecessor.

σ-orbit: Finding the next dart in the σ-orbit of a Khalimsky vertex is straightforward, since its degree is limited to the number of four direct neighbors. In the case of 8-connected boundaries it involves a more complex procedure: Here, vertices can consist of more than one pixel, which means that their contour has to be followed in order to find the σ successor (cf. Fig. 2 right).

Giving Access to the Geometry Section 2.1 introduced the geometry-related part of the GeoMap ADT, notably the CellScanIterator, which allows to

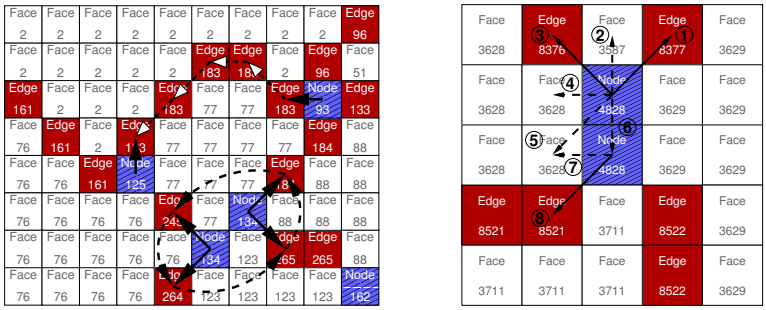

Fig. 2. *Left*: Two DartTraversers cycling through their α- and σ-orbits, respectively. *Right*: Detailed series of intermediate states for finding two σ successors.

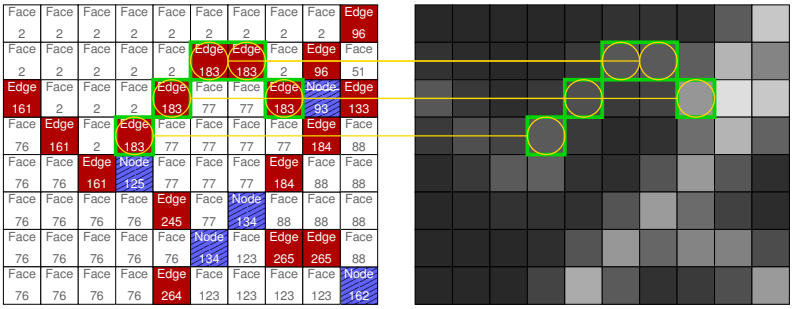

Fig. 3. CellScanIterator scanning edge 183 in a gradient magnitude image

iterate over the labels of all basis cells (pixels) of a given block cell (cf. g in Def. Definition 3). It can be efficiently realized by scanning the internal CellImage (restricted to the cached bounding box of the block cell) and stopping at pixels belonging to the cell being queried. At each step, the iterator returns the basis cell's label (i.e. its position), which is then used to look up properties in any application-specific image, for example to find the mean color of a region in the original image, or the gradient estimates on a given edge (cf. Fig. 3).[1]

3 Application

The GeoMap is designed to serve as a versatile representation for many segmentation algorithms. We have employed it to implement a variety of approaches, some of which we will describe here.

Canny Hysteresis Canny's segmentation approach is undoubtedly the most well-known one, and its steps are still representative for its state-of-the-art descendants: After collecting initial evidence for edges (the initial oversegmentation in our case), the candidate set is filtered to get the final result. We implemented this hysteresis thresholding on the basis of the edges in our GeoMap. However, we are not limited to assess the edges based on gradient information, but also implemented measures based on the adjacent faces (e.g. difference of their mean colors, T-test, ...).

Contraction Kernels The GeoMap allowed us to implement irregular pyramids as in [2] by grouping a set of Euler Operations into complex contractions. Furthermore, the CellScanIterator made it easy to implement (e.g. color-based) salience measures to define the contraction kernels.

Active Paintbrush In contrast to the above, this tool relies entirely on human interaction; is allows to "paint over" region boundaries to initiate region merge operations [12]. It is very useful to interactively mark fine structure in

[1] A more efficient implementation directly scans the target image in parallel, making the indirection via the position unnecessary.

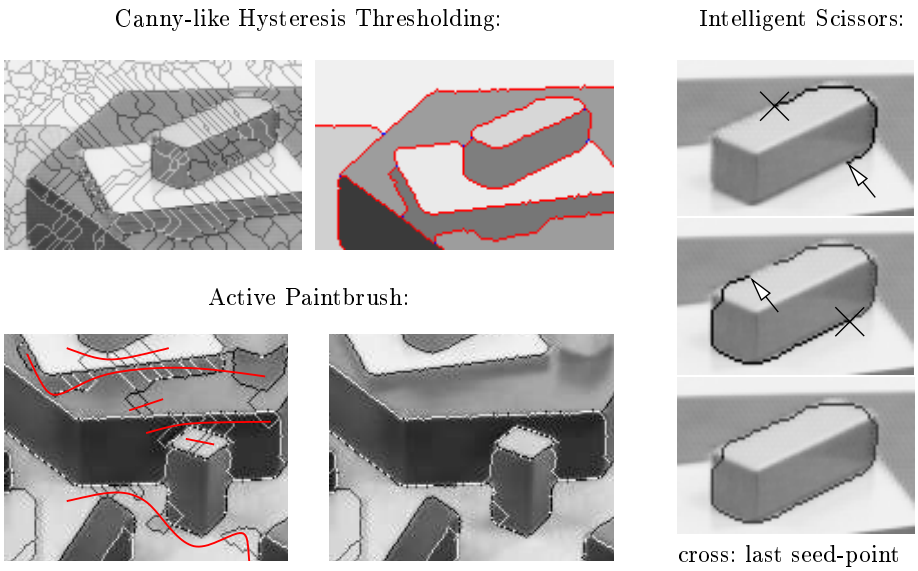

Fig. 4. Example screenshots showing the tools in action

low-contrast images (e.g. angiography). Since our framework is based on one common representation, it is also possible to use the paintbrush to correct errors made by the other, automatic tools. To facilitate this, we augmented it with a means to protect the boundary of single regions from being changed.

Intelligent Scissors After the selection of an initial seed point on a contour, this semi-interactive tool highlights the optimal path to the current pointer position in real-time with a *live-wire* [13]. A complete contour can be delineated with only a few additional selections. In order to define the optimal path, we measure and combine the significance of single edges - another example where the abstraction level of the GeoMap formalism led to directly reusable components, namely the cost measures from the hysteresis tool.

Implementing these algorithms based on the GeoMap formalism means to abstract from the boundary definition. We have used all these algorithms with both a crack edge representation and 8-connected thin boundaries [7] in an irregular pyramid, whose level 0 contained a watershed oversegmentation. This conforms to the recent approach of starting with *superpixels* [10], not pixels directly.

Note that our experimental results prove that it is possible to achieve this level of abstraction not only without losing flexibility, but that generic programming techniques allow for very efficient realizations of the formalism. For example, the GeoMap of a 640×480 image initially segmented into 11.161 vertices, 17.930 edges, and 6.775 faces can automatically be reduced (based on face statistics) into a final result with about 30 regions in 3.6 seconds on a Pentium III notebook with 800 MHz.

4 Conclusion

The GeoMap formalism demonstrates that the integration of topology and geometry in one unified representation leads to a versatile basis for image segmentation. Adapting algorithms to this framework leads to concise, comprehensible code and does not sacrifice speed. At the same time, the GeoMap introduces a level of abstraction that facilitates the decomposition of published segmentation methods and (re-)combination of approaches, i.e. interchange edge salience definitions or apply several algorithms on the same image.

In the future, we want to extend the GeoMap formalism to work with other (subpixel- or 3D) boundary definitions and add split operations and means to refine the contours retroactively. On the application side, we are currently working on the integration of learning methods and more sophisticated edge salience measures based on boundary continuity.

References

1. Pavlidis, T.: Structural Pattern Recognition. Springer (1977)
2. Kropatsch, W.G.: Building irregulars pyramids by dual graph contraction. IEEE-Proc. Vision, Image and Signal Processing **142** (1995) 366–374
3. Montanvert, A., Meer, P., Rosenfeld, A.: Hierarchical image analysis using irregular tessellations. T-PAMI **13** (1991) 307–316
4. Brun, L., Kropatsch, W.: Introduction to combinatorial pyramids. In Bertrand, G., Imiya, A., Klette, R., eds.: Digital and Image Geometry. Volume 2243 of LNCS. Springer (2001) 108–127
5. Köthe, U.: XPMaps and topological segmentation - a unified approach to finite topologies in the plane. In Braquelaire, A.J.P., Lachaud, J.O., Vialard, A., eds.: Proc. 10^{th} Intl. Conf. Discrete Geometry for Computer Imagery(DGCI 2002). Volume 2301 of LNCS., Bordeaux, France, Springer (2002) 22–33
6. Khalimsky, E., Kopperman, R., Meyer, P.R.: Computer graphics and connected topologies on finite ordered sets. Topology and its Applications **36** (1990) 1–17
7. Köthe, U.: Deriving topological representations from edge images. In Asano, T., Klette, R., Ronse, C., eds.: Geometry, Morphology, and Computational Imaging. Volume 2616 of LNCS., Berlin, Springer (2003) 320–334
8. Meine, H.: XPMap-based irregular pyramids for image segmentation. Diploma thesis, Dept. of Informatics, Univ. of Hamburg (2003)
9. Mäntylä, M.: An Introduction to Solid Modeling. Computer Science Press (1988)
10. Ren, X., Malik, J.: Learning a classification model for segmentation. In: Proceedings of the Tenth Intl. Conference On Computer Vision (ICCV-03). Volume 1., Vancouver, British Columbia, Canada, IEEE (2003) 10–16
11. Braquelaire, J.P., Domenger, J.P.: Representation of region segmented images with discrete maps. Technical Report 1127-96, Université Bordeaux, Laboratoire Bordelais de Recherche en Informatique (1996)
12. Maes, F.: Segmentation and Registration of Multimodal Images: From Theory, Implementation and Validation to a Useful Tool in Clinical Practice. PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium (1998)
13. Mortensen, E.N., Barrett, W.A.: Toboggan-based intelligent scissors with a four parameter edge model. In: Computer Vision and Pattern Recognition. Volume 2., IEEE (1999) 452–458