A Topologically Consistent Representation for Image Analysis: The Frontiers Topological Graph

Christophe Fiorio**

TU Berlin, Sekr MA 6-1
10623 Berlin, Germany

Abstract. In this paper a "topologically consistent" representation for images is presented. It is called the *Frontiers Topological Graph* and is derived from the combinatorial maps model. Thus it establishes a link between image analysis and image synthesis. An efficient algorithm which constructs the Frontiers Topological Graph is developed.

keywords: image representation, topology, combinatorial map, graph.

1 Introduction

In image analysis the choice of a good representation is crucial. Indeed after a first analysis within the so-called segmentation process, more "intelligent" algorithms must be run on the images in order to achieve the goal of the application, i.e. often a recognition process. This type of algorithms needs a data structure and cannot proceed with only a matrix of pixels.

The important thing here is what we mean by a "good representation". For example, the Region Adjacency Graph (RAG for short) exists since long ago [Ros74,Pav77] and is one of the most commonly used representation. But some informations are missing, such as the number of frontiers between two regions, or the inclusion of regions (we call such a configuration a *hole*).

Other well-known representations are the so-called *neighborhood graphs*: a vertex is associated to each pixel and an edge links two vertices if the corresponding pixels are adjacent. The more commonly used are the 4- and 8-neighborhood graphs. V.A. Kovalevsky explained in [Kov89] that "the attempts to develop a consistent topology of 2-dimensional images by means of neighborhood-graphs have failed". This is due to the well-known connectivity paradox [Pav77]. The problem does not arise with a 6-neighborhood graph, but contradictions in the definitions of boundaries still stay: boundaries are sets of pixels and have a (finite) area.

In fact a good representation should show the interactions between regions and then realize the topology of the image. This is what we mean by a topologically consistent representation. The Frontiers Topological Graph is conformant to the *star-topology* [AAF95]. It is derived from combinatorial maps (for a survey see [Lie91]), a representation used in CAD, and is the first topology based representation (the notion of map first appears in 1960 [Edm60]).

** Supported by a postdoctoral grant of DIMANET.

AS it is important not only to have a good representation but to be able to compute it, an extraction algorithm is developed in this paper. It is simple and very efficient as it runs in linear time.

The choice of a model derived from combinatorial maps is justified in Section 2. Moreover it is explained why we prefer not use the dual graphs as presented by W.G. Kropatsch in [Kro94] or the Cell-List data structure of V.A Kovalevsky [Kov89]. A description of our structure is given in Section 3. Section 4 gives the principles of the extraction algorithm and Section 5 details the implementation. Section 6 presents some perspectives as offered by the Frontiers Topological Graph (FTG for short).

2 Topological Representations

As already claimed, the Region Adjacency Graph is not a good topological representation since it does not respect the basic axioms of topology. Kovalevsky in [Kov89] proposes a cell-list representation. He introduces *block cells*: 2 dimensional elements of the block cells are 2-dimensional open sub-complexes that correspond to regions (sets of two by two adjacent pixels); frontiers in between regions are 1-dimensional elements and the junction points of frontiers are 0-dimensional elements. As said in [Kov89], the 1-dimensional skeleton of the dual complex of block cells is equivalent to the RAG. So, it seems that the cell-list structure does not have more properties that the RAG. Moreover we prefer a graph structure since pattern matching algorithms usually use such structures (see examples in [Vos92]), and this is not a graph structure.

W.G. Kropatsch and H. Macho present in [KM95] a data structure based on planar graphs and used in image pyramidal analysis. The proposed structure is totally homogeneous and realizes the topology of images. Moreover the planar graphs used in this structure are connected, so processing is easier. But two graphs must be maintained and a fictive edges connecting holes[1] to the surrounding face have been added. For further information about this analysis method, see e.g [MMR91]. Moreover the extension to a 3-dimensional representation seems to be difficult.

On the other side combinatorial maps are a boundary representation of geometric objects and are in increasing use. The map notion has first appeared in 1960 [Edm60]. Then, 2-maps have been the first mathematical modelisation of topology based representations [Jac70,Cor75]. Lot of works have been done since; the reader is referred to the survey of P. Lienhardt [Lie91]. We can note that P. Lienhardt and J. Dufourd have proposed in [Lie89,Duf91] some extensions like the n-generalized maps and n-hypermaps that allow to represent n-dimensional, orientable or not, objects. Such maps present useful properties [BDFL92] for a representation used in image analysis:

- they are defined from a unique basic element: the dart;
- numerous topological properties can easily be computed;

[1] a region or a set of regions totally included in another is called a hole

- only two basic operations to manipulate it are needed,
- data structures are immediately deduced from the definition.

The FTG is a derived model of the 2-maps. Its advantages are as follows:

- 2-maps are a topological representation as wanted;
- they are close to planar graph representations;
- 2-maps have been extended to n-maps for n-dimensional objects; we can expect the same for the FTG.

There are two reasons for not using directly the 2-maps model:

1. In the 2-maps models vertices represent particular points of the plane and darts the edges of the object. For image analysis it is more convenient tp have vertices represent the regions of the image.
2. The inclusion of a face in another one (*hole*) is not directly coded. M. Gangnet and *al* propose in [GHPT89] to add an inclusion tree of the contours. We think that a unique representation is preferable in terms of image analysis.

Hence the Frontiers Topological Graph is a planar multi-graph implemented by a derived representation of 2-maps.

3 The Frontiers Topological Graph

3.1 Frontiers and Contours

In order to define FTG we have to precise some notions such as frontiers, contours and related ones.

What we call here a frontier is slightly different from the classical topological notion of frontier. In fact it is a continuous part of the topological frontier, common to two adjacent regions. Note that there can be more than one frontier between two given regions. That is why the FTG is a multi-graph.

The topological frontier is called here the (set of) contours of a region and denoted by $Co(R)$ for a given region R. Moreover a region can contain "holes" so that the difference between **the** (one) exterior contour and possible interior contour**s** of a region must be done.

definition 31 (exterior contour). The *exterior contour* of a region R is the connected subset $CoExt(R)$ containing all the elements e of $Co(R)$ such that the region R stays to the righthand side when following it in clockwise direction.

definition 32 (interior contour). An *interior contour* of a region R is the connected subset $CoInt(R)$ containing all the elements e of $Co(R)$ such that the region R stays to the righthand side when following it in counter clockwise direction.

3.2 Definition

The FTG is different from the RAG in the way that edges do not represent only adjacency relation but also the existence of a frontier between the regions. It benefits from the facilities given by the algebraic description of combinatorial maps and by the graphs algorithmic. Moreover it stays a natural representation (one vertex for one region) in image analysis.

So a FTG, $G(V, D, \alpha, \sigma)$, where V is the set of vertices of the graph, D is the set of darts (half-edges), σ an involution on D and α a permutation on D, has the following properties:

1. Each region is represented by one and only one vertex.
2. A dart (half-edge) e is always incident to a vertex R of V. The notation e^R denotes the incident vertex of e.
3. An edge of the graph is a non-ordered pair $(e, \sigma(e))$ where e belongs to D. Let E be the set of edges, $(e_1^R, e_2^{R'}) \in E$ if and only if there exists a frontier between R and R'. Thus an edge represents a frontier. A dart symbolizes the frontier "as seen from" the region to which it is incident.
4. The cycles of the permutation α on D correspond to a contour of a given region and respect the order induced by the sequence of frontiers making the contour.
5. ∞ is a particular vertex of the graph symbolizing the exterior of the image.
6. To each vertex is associated the list of the cycles of α related to the contours of the region represented by this vertex. By convention, the exterior contour will always be the first of the list.

Figure 1 shows an example of an image and the corresponding FTG. This figure uses a graphical representation of the graph: α is represented by arrows and σ by the short lines cutting the edges.

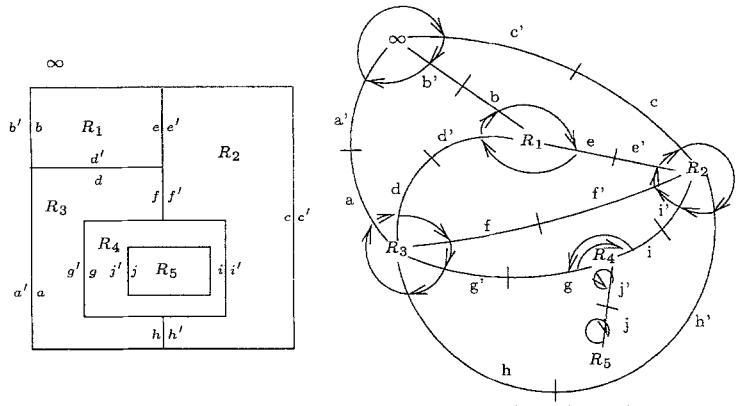
An algebraic representation can also be given (see Figure 1). The graph is then described by the applications α and σ. The relations between darts are represented by n-uplets. For example, (e_1, e_2) for the application σ means that $e_2 = \sigma(e_1)$ and $e_1 = \sigma(e_2)$; (e_1, e_2, e_3) for the application α means that $e_2 = \alpha(e_1)$, $e_3 = \alpha(e_2)$ and $e_1 = \alpha(e_3)$.

Note that the dart incident to R_4 and part of the edge between R_4 and R_5 is not in α related to the other darts incident to R_4. Indeed the region R_5 is "included" in region R_4, so that frontier belongs to a different contour than the exterior contour of R_4. Moreover we can remark that there exist two edges between R_2 and R_3 since there are two frontiers between these two regions.

4 Extraction Algorithm: Principle

The extraction algorithm is a combinatorial algorithm that takes advantage of the algebraic description of the graph. The graph is updated according to local configurations of frontiers. Section 5 details the processing for each configuration.

This algorithm runs in linear time. The proof of the complexity and validity of the algorithm is out of the scope of this paper. A complete study is presented in



σ : $(a,a')(b,b')(c,c')(d,d')(e,e')(f,f')(g,g')(h,h')(i,i')(j,j')$
$\alpha : \infty$: $()$; (a',c',b')
R_1 : (b,e,d')
R_2 : (c,h',i',f',e')
R_3 : (a,d,f,g',h)
R_4 : (i,g) ; (j')
R_5 : (j)

Fig. 1. Example of a FTG and its algebraic representation

[Fio95]. To verify the complexity, we have only to check that the operations made for each configuration are time constant. Note that it is supposed here that for each pixel we know the region it belongs to, i.e. a connected component labeling was made. This hypothesis does not contradict the linearity of the algorithm since C. Fiorio and J. Gustedt have presented in [FG96] a linear algorithm that performs both a segmentation and a connected component labeling in linear time.

4.1 General Scheme

The principle of Algorithm 1 is to scan the image line by line and to build greedily the FTG.

Algorithm 1: Extraction of the *Frontiers Topological Graph*

Data : an image
Result : the Frontiers Topological Graph

Initialize the graph with an isolated vertex ∞;
Initialize D (set of darts) to an empty set;
Initialize the L list to an empty list;
foreach *precode of the image* **do**
    Execute the code related to the current precode;
end

The scan is done by moving a 2×2 window from the left to the right and from the first line to the last. Is starts with the first pixel of the image covered by a particular window[2]. The algorithm looks at the configuration of the frontiers in this window. We call such a configuration a precode. It is easy to see that there are only 12 possible configurations[3].

A list called L (see Section 4.3) allows to retrieve the edges of the graph corresponding to the frontiers in each precode. Depending on the precode, edges are created, applications α and σ are updated, then edges and vertices are merged. At the end of the scan, the obtained graph is the Frontiers Topological Graph.

4.2 particular configurations

Two frontier configurations are particular. They can be found in different precodes.

definition 41 (Initiator). An *Initiator* is a configuration within a precode where a frontier surrounds the bottom-right pixel of the precode.

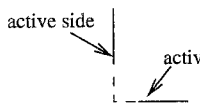
definition 42 (Unitor). An *Unitor* is a configuration within a precode where a frontier surrounds the top-left pixel of the precode.

These configurations are the only ones that imply creation or merging of edges and vertices.

4.3 L list

The L list records the *active* frontiers, i.e. the known and yet not entirely determined frontiers. The elements of L are ordered according to the order induced by a left-right scan of the image. The current element of L, `Current(`L`)`, corresponds to the expected frontier, i.e. the frontier to meet in the next precode if no particular configuration occurs before. The list is reordered when Initiator or Unitor precodes are met.

A unique frontier can be present more than once in the list L. Indeed it will be inserted in the list as many times as it is met on a line. Each exemplar of a frontier is represented in the list by one of the darts of its edge. The dart is not always the same for a given frontier and moreover can change during the process.



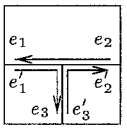
The dart inserted in the list corresponds to the *active* side of a frontier. The active side is the left side for a vertical frontier and the upper side for an horizontal one.

[2] The image is considered to be surrounded by a blank infinite region

[3] The 4 configurations which have only one "linel" (edge of a pixel) inside the window are not valid in terms of frontiers between regions

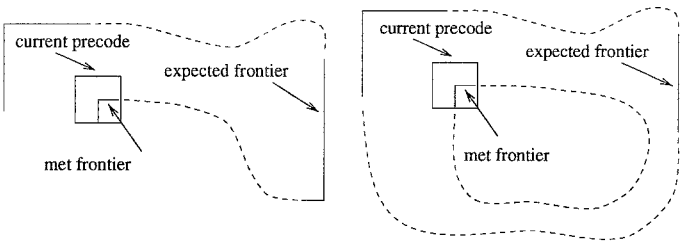
4.4 α application

The α application determines for each contour a circular permutation on the darts according the order induced by the sequence of frontiers composing each contour. The determination is possible as soon as precodes are processed. Indeed the interior and exterior contour definitions (see definitions 31 and 32) imply a particular order. Remember that a dart symbolizes the frontier as seen from the (interior) of the region. So with the help of the precode we can determine the order relation between two darts.



In this example the α relation is represented by arrows when processing this precode. We can easily verify that the induced order is the one required by the definitions of the exterior and interior contours.

A problem stays: in the case of an Initiator, we cannot know if the frontier met belongs to the same or to a different contour than the expected one – i.e. pointed by `Current(`L`)` – (see Figure 2). Nevertheless a decision must be taken,



(a) same contour (b) different contour

Fig. 2. Does the met frontier belong to the same contour as the one waited?

though it can be changed later. The List L allows this: for a Unitor precode, the current element e of L is the active frontier and the following e' in L the one that should close the contour. So it is sufficient to check if effectively these frontiers belong to the same cycle of α (i.e. the same contour). For this we have only to check if $e = \alpha(e')$. Else the two contours must be merged. This last operation will be called `Unify-Contour`.

5 Implementation

In this section the extraction algorithm is described more precisely. First, some naming conventions:

- let e be a dart; e' will denote the dart such that $e' = \sigma(e)$,
- $e' \leftarrow \sigma(e)$ says that now $e' = \sigma(e)$,
- let $\alpha(e_1) = e_3$; the notation $\alpha(e_1) \leftarrow e_2$ says that now e_2 is equal to $\alpha(e_1)$ and e_3 is equal to $\alpha(e_2)$, so we have $e_3 = \alpha(\alpha(e_1))$.

5.1 functions of the algorithm

In this part the different functions of the extraction algorithm are explained and one is detailed: `Unify-Contour`, as it is is the more complicated and sensible one.

`Create(`e`,`R`)`
add a new dart e incident to vertex R in the graph.

$(e, e') \leftarrow$ `Create-Edge(`R, R'`)`
A new frontier between two regions R and R' have just been detected, a new edge must be added. So 2 darts e and e' are added and linked by σ.

$R \leftarrow$ `Create-Vertex()`
This function is called only if the precode is an Initiator; it adds a new vertex R to the graph if the region in the precode is a new one, else it returns the vertex.

α`-Insert(`e_1, e_2`)`
Here the notation is particular in order to simplify the description of the algorithm. It is supposed that one of the two darts (the arguments) is already in a cycle of α. The other one has just been created and is inserted in the same cycle of α according to the order of the arguments, i.e. before, if it is the first argument or else, after.

α`-Link(`e_1, e_2`)`
This operation creates a new cycle with e_1 and e_2, so the operations $\alpha(e_1) \leftarrow e_2$ and $\alpha(e_2) \leftarrow e_1$ are made.

`L-Insert(`$e_1, ..., e_n$`)`
The given darts are inserted in this order into the list L before `Current(`L`)`.

`L-Remove(`$e_1, ..., e_n$`)`
The given darts are removed one by one from the list L. If one is `Current(`L`)`, then the following in the list becomes `Current(`L`)`.

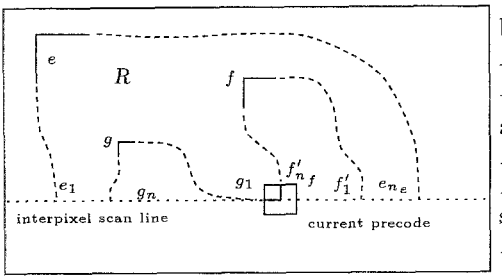
`L-Replace(`e`)`
`Current(`L`)` is replaced in L by the element e which becomes the `Current(`L`)`.

`Unify-Contour(`e_1, e_2`)`
In the case of an Unitor, two frontiers must be merged into one. But the frontiers represented by edges (e_1, e'_1) and (e_2, e'_2) are not necessarily on the same contour[4]. So the two contours must be merged. `Unify-Contour(`e_1, e_2`)` does that by inserting the cycle containing e_2 before e_1 in its cycle (see examples in Figure 3):

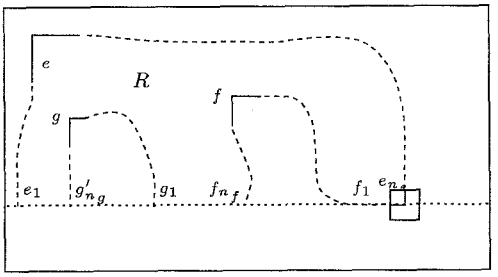
`Unify-Contour(`e_1, e_2`)`
$\alpha(\alpha^{-1}(e_1)) \leftarrow \alpha(e_2)$
$\alpha(e_2) \leftarrow e_1$
$\alpha(\alpha^{-1}(e'_2)) \leftarrow \alpha(e'_1)$
$\alpha(e'_1) \leftarrow e'_2$

$\alpha^{-1}(e_i)$ stands for the dart preceding e_i in the α permutation. On the contrary, the notation $\alpha(\alpha^{-1}(e_i)) \leftarrow \alpha(e_j)$ does not mean "the following of the previous", i.e. itself, but: "the previous has now $\alpha(e_j)$ as follower".

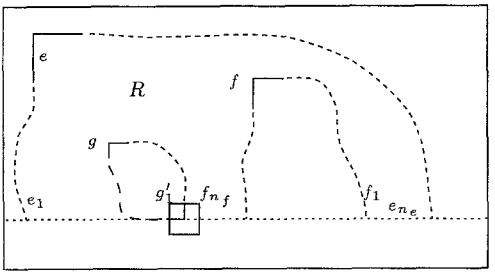
[4] this can be easily checked by the condition $\alpha(e_2) \neq e_1$



before :
$R : (e_1 \dots e_{n_e})\ (g_1 \dots g_{n_g})(f_1 \dots f_{n_f})$
$L : e'_1 \leftarrow g_{n_g} \leftarrow g'_1 \leftarrow f_{n_f} \leftarrow f'_1 \leftarrow e_{n_e}$
after :
$R : (e_1 \dots e_{n_e})\ (g_1 \dots g_{n_g}\ f_1 \dots f_{n_f})$
$L : e'_1 \leftarrow g_n \leftarrow f'_1 \leftarrow e_{n_e}$
so
$R : (e_1 \dots e_{n_e})\ (f_1 \dots f_{n_f}\ g_1 \dots g_{n_g})$



before :
$R : (e_1 \dots e_{n_e})\ (g_1 \dots g_{n_g})\ (f_1 \dots f_{n_f})$
$L : e'_1 \leftarrow g_{n_g} \leftarrow g'_1 \leftarrow f_{n_f} \leftarrow f'_1 \leftarrow e_{n_e}$
after :
$R : (e_1 \dots e_{n_e} f_1 \dots f_{n_f})\ (g_1 \dots g_{n_g})$
$L : e'_1 \leftarrow g_{n_g} \leftarrow g'_1 \leftarrow f_{n_f}$



before :
$R : (e_1 \dots e_{n_e})\ (g_1 \dots g_{n_g})\ (f_1 \dots f_{n_f})$
$L : e'_1 \leftarrow g_{n_g} \leftarrow g'_1 \leftarrow f_{n_f} \leftarrow f'_1 \leftarrow e_{n_e}$
after :
$R : (e_1 \dots e_{n_e})\ (f_1 \dots f_{n_f})\ (g_1 \dots g_{n_g})$
$L : e'_1 \leftarrow f_{n_f} \leftarrow f'_1 \leftarrow e_{n_e}$

Fig. 3. Effect of `Unify-Contour` for the contours of R.

5.2 Processing of the precodes

The processing to make for each precode is now presented. We can note that there are two precodes which do not require any processing, and two couples of precodes which require almost the same processing.

In the following, the $\equiv$ notation indicates a fact and not some processing. More precisely it indicates how edges present in a precode are recovered.

There is nothing to do for the precodes (0) and (2).

The precode (1) is particular: it is the basic *Initiator*. The frontier met is supposed to be new and so a new edge is inserted in the graph. Precodes (6) and (7) are more complex *Initiators*; their processings are identical, except for dart e which denotes different frontiers in the two precodes.

The precode (10) is the basic *Unitor* precode. Precodes (8) and (9) are also *Unitors*; their two processings are merely identical; only the last operation is not the same.

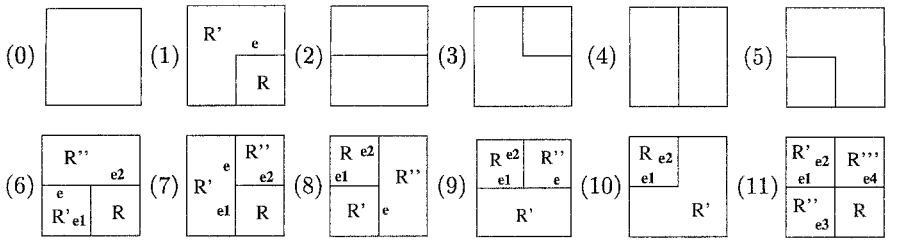


Fig. 4. The precodes list.

Precodes (3), (4) and (5) are the simplest ones (except for the empty precode); they just update the L list. The last precode (precode (11)) is the most complex, since it is at the same time an *Initiator* and an *Unitor*.

Figure 5 shows implementation of the precodes.

6 Conclusion and Perspectives

In this paper we have presented a new representation for images in image analysis. It is "topologically consistent" and allows more sophisticated processing. It is based on the star-topology described in [AAF95] and uses the combinatorial maps model. It establishes a link between image analysis and synthesis. Moreover an extraction algorithm running in linear time was detailed. The representation can efficiently be built from a segmented image.

To be fully exploitable this representation must be completed by a coding of the frontier. This can easily be done by an interpixel coding as described in [Cha95]. It is a chain-link like coding ([Fre61,Ced79]), located in-between the pixels. Furthermore it is trivial to see that at each precode the coding of each frontier can be updated.

The FTG points out the difference between two adjacent regions (regions that share a frontier from their exterior contour) and a (set of) region included in another. The difference is implicit in the representation. May be that an explicit expression of the inclusion would be interesting: for example by adding an involution between edges of the exterior contour and edges of interior contours. Then the overload of the representation and the algorithm should be studied. Furthermore a 3-dimensional extension of this graph can be forecasted since it is based on combinatorial maps that are able to represent n-dimensional objects.

In terms of image analysis, more sophisticated merging algorithms can be implemented: they can take into account the presence of more than one frontier between regions or the inclusion of regions (holes). Moreover the algebraic description allows easier implementation of operations on the graph. As we have a description of each frontier, we expect edge linking algorithms to be implemented and the use of the detection of sub-structures (e.g minimal cycles or maximal chains) in order to parameterize the linking.

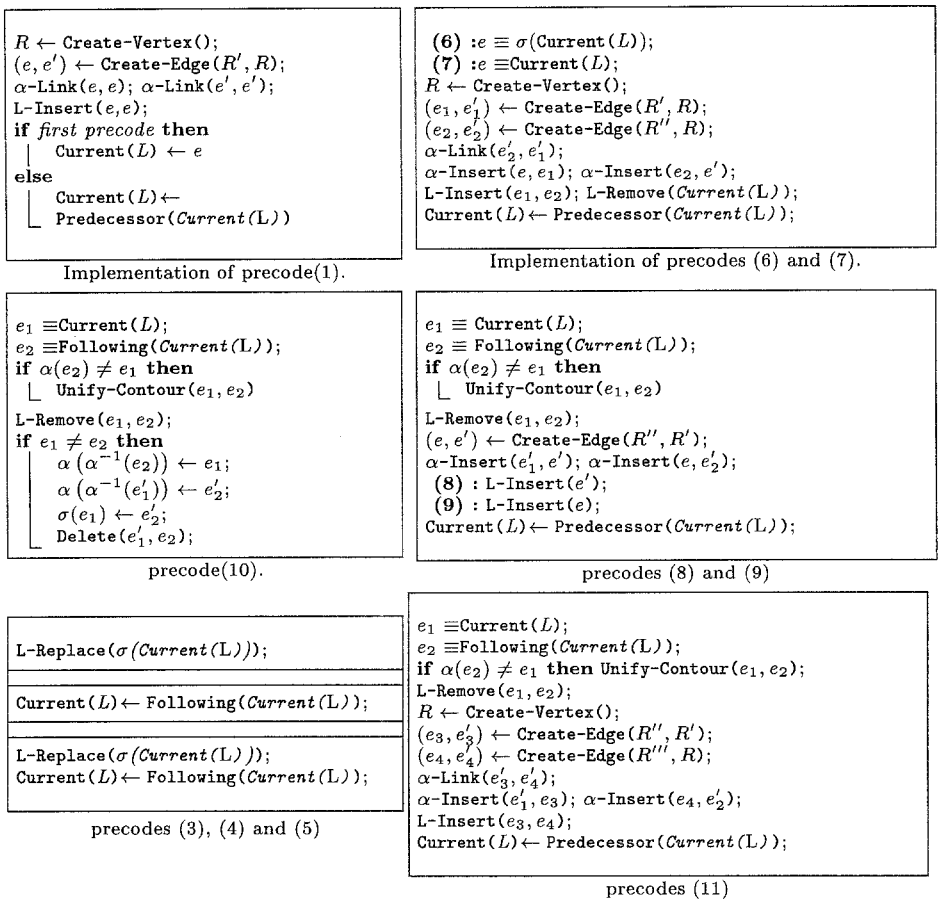


Fig. 5. Implementation of the precodes

References

[AAF95] Ehoud Ahronovitz, Jean-Pierre Aubert, and Christophe Fiorio. The star-topology: a topology for image analysis. In 5^{th} *Discrete Geometry for Computer Imagery, Proceedings*, pages 107–116. Groupe GDR PRC/AMI du CNRS, september 1995.

[BDFL92] Y. Bertrand, J.-F. Dufourd, J. Françon, and P. Lienhardt. Modélisation volumique à base topologique. Rapport de recherche 92/16, Université Louis Pasteur, Centre de Recherche en Informatique, 7, rue René Descartes, 67094 Strasbourg, France, 1992.

[Ced79] Roger L. T. Cederberg. Chain-link coding and segmentation for raster scan devices. *Computer Graphics and Image Process.*, 10:224–234, 1979.

[Cha95] Philippe Charnier. *Outils algorithmiques pour le codage interpixel et ses applications.* Thèse de doctorat, Université Montpellier II, Janvier 1995.

[Cor75] Robert Cori. *Un code pour les graphes planaires et ses applications*, volume 27. Astérisque, SMF, Paris, France, 1975.

[Duf91] Jean-François Dufourd. Formal specification of topological subdivisions using hypermaps. *Computer-Aided Design*, 23(2):99–116, 3 1991.

[Edm60] J. Edmonds. A combinatorial representation for polyhedral surfaces. *Notices of the American Mathematical Society*, 7, 1960.

[FG96] Christophe Fiorio and Jens Gustedt. Two linear time Union-Find strategies for image processing. *Theoretical Computer Science*, 154:165–181, 1996.

[Fio95] Christophe Fiorio. *Approche interpixel en analyse d'images : une topologie et des algorithmes de segmentation.* Thèse de doctorat, Université Montpellier II, 24 novembre 1995.

[Fre61] H. Freeman. On the encoding of arbitrary geometric configurations. *IEEE trans. Elec. Computers*, 10, 1961.

[GHPT89] Michel Gangnet, Jean-Claude Hervé, Thierry Pudet, and Jean-Manuel Van Thong. Incremental computation of planar maps. report 1, Digital PRL, 5 1989.

[Jac70] A. Jacques. Constellations et graphes topologiques. In *Combinatorial Theory and Applications*, pages 657–673, Budapest, 1970.

[KKM90a] Efim Khalimsky, Ralph Kopperman, and Paul R. Meyer. Boundaries in digital planes. *J. of Applied Mathematics and Stochastic Analysis*, 3(1):27–55, 1990.

[KKM90b] Efim Khalimsky, Ralph Kopperman, and Paul R. Meyer. Computer graphics and connected topologies on finite ordered sets. *Topology and its Applications*, 36:1–17, 1990.

[KM95] Walter G. Kropatsch and Herwig Macho. Finding the structure of connected components using dual irregular pyramids. In 5^{th} *Discrete Geometry for Computer Imagery, Proceedings*, pages 147–158, *invited lecture.* Groupe GDR PRC/AMI du CNRS, September 1995.

[Kov89] V.A. Kovalevsky. Finite topology as applied to image analysis. *Computer Vision, Graphics, and Image Processing.*, 46:141–161, 1989.

[Kro94] Walter G. Kropatsch. Building irregular pyramids by dual graph contraction. Technical Report PRIP-TR-35, Dept. for Pattern Recognition and Image Processing, Institute for Automation, Technical Univerity of Vienna, July 1994.

[Lie89] P. Lienhardt. Subdivision of n-dimensional spaces and n-dimensional generalized maps. In 5^{th} *ACM Conf. Comput. Geometry*, pages 228–236, Saarbrücken, Germany, 1989.

[Lie91] P. Lienhardt. Topological models for boundary representation: A survey. *Comput. Aided Design*, 23(1):59–81, 1991.

[MMR91] Annick Montanvert, Peter Meer, and Azriel Rosenfeld. Hierarchichal image analysis using irregular tessalations. *IEEE trans. Pattern Analysis and Machine Intelligence*, 13(4):307–316, April 1991.

[Pav77] Theo Pavlidis. *Structural Pattern Recognition.* Springer-Verlag, New York, 1977.

[Ros74] Azriel Rosenfeld. Adjacency in digital pictures. *Inform. and Control*, 26:24–33, 1974.

[Vos92] George Vosselman. *Relational Matching.* Lecture Notes in Computer Science. Springer-Verlag, Berlin Heidelberg Germany, 1992.