A new contour filling algorithm based on 2D topological map

Guillaume Damiand, Denis Arrivault

A New Contour Filling Algorithm Based on 2D Topological Map *

Guillaume Damiand** and Denis Arrivault

SIC - bât. SP2MI, Bvd M. et P. Curie
BP 30179, 86962 Futuroscope Chasseneuil Cedex - France
{damiand,arrivault}@sic.univ-poitiers.fr

Abstract. In this paper, we present a topological algorithm which allows to fill contours images. The filling problem has been widely treated and it recently appeared that it can always be split into two different process : a generic topological process and a dedicated geometrical post-processing which depends on the application. Our algorithm, based on a 2D topological map description of the image, addresses the first step of processing. It is fast, generic and robust. Moreover, the complete topological description allows to easily integrate geometrical constrains and makes our approach an interesting basis for every filling process.

Keywords. topological maps, filling process, character reconstruction.

1 Introduction

Filling algorithms are used in many applications but especially in character image generation [1]. The motivation of our work comes from the handwritten characters description. In order to use structural methods for describing a character (for example fuzzy hierarchical graphs as in [2]) one needs to extract a clean skeleton. The filling algorithm was developed for the characters reconstruction phase of this process. Actually, a skeleton-based graph is built from character image through a contour approximation and a character reconstruction (Fig. 1).

The problem of filling the contour of a region has been widely treated during the last three decades. Depending on the application, this can be solved by using an a priori knowledge on the contour topology (with contour approximations for example) or directly with the raster graphics. Pavlidis [3] separated also the "polygon based" techniques from the "pixel based" ones. The purpose of this article is to present a "pixel based" algorithm using a topological description without a priori knowledge.

* Paper published in Proceedings of 6th IAPR International Workshop on Graph Based Representation in Pattern Recognition, LNCS 4538, pages 319-329, June 2007. Thanks to Springer Berlin / Heidelberg. The original publication is available at http://www.springerlink.com/content/r5j430q78141wlj6/

** Partially supported by the ANR Fundation under grants ANR-06-MDCA-008-05/FOGRIMMI.

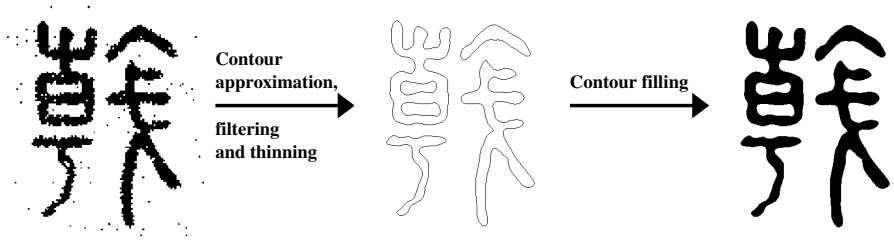

Fig. 1. Processing of a chinese handwritten characters.

There are many "pixel based" filling algorithms in the literature. They can be divided into two broad categories [4]: parity check filling algorithms (also called scan-line or edge filling) and seeds growing (also referred to as connectivity filling or region growing). Filling by parity check is fast and requires less or no additional working memory comparing with seeds growing filling. Line by line, the background pixels are associated with a depth number according to the number of black pixels previously encountered during the line scan. Then the filling is done by using the depth number parity. The major difficulty facing such a scheme is that different arcs of the contour may be mapped on the same pixel (Fig. 2 (A)). That is the reason why it often fails to correctly handle complex objects while seeds growing methods are theoretically more robust. Starting from interior starting points (the seeds), seed growing algorithms are propagation procedures that color the regions of interest. Nevertheless, the seed choice is a non trivial issue and can not be automated without an a priory knowledge of the contours relation.

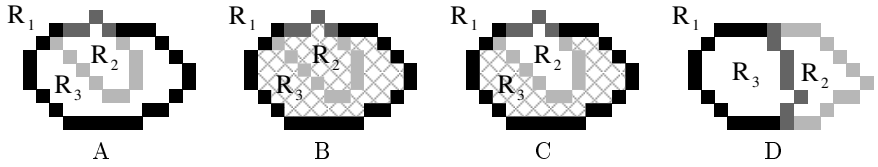

Fig. 2. The problem which occurred when two boundaries are shared. (A) Regions R_2 and R_3 share a part of a boundary (drawn in dark grey in the figure). (B) What we will obtain by our algorithm: both regions are filled. (C) What we want to obtain intuitively. (D) Another configuration, topologically equivant to (A), but in this case, intuitivelly, we want to fill both regions.

The contours relation are defined by Codrea & Al. [5] who explain that a filling operation need a formal or explicit description of what is an interior or exterior region. In this perspective, Martin & Al. [6] propose a topology-based filling algorithm which uses not only inclusion relation for the filling decision but

also the ideas of *dominant subobject* and *exteriority* for addressing the images with ambiguities. The dominant subobject is defined as the region which comprises most of the external perimeter of an object and exteriority allows to fill subobjects that are outside enough of a dominant subobject even if the sharing boundary is small. This algorithm is powerful but requires complex definitions and thresholds. Moreover it does not allow to deal with non closed contours. Nevertheless, Martin & Al. outline a fundamental aspect of the filling problem. The contour images has to be processed at two levels : a topological process first that is quite generic for all applications and a geometrical process that is dedicated to the application.

The topological process given by Martin & Al. is based on the inclusion relation. This approach is quite poor and does not allow to propose efficient geometrical constraints. The 2D topological maps used in our approach provide a complete topological description that can be efficiently adapt to every application. The filling algorithm proposed in this article is a topological one, generic, complete and fast. We are not addressing the geometrical constraints at the moment but we will explain why this approach is an interesting basis for every filling problems.

In the next section, recalls on 2D-topological maps will be given. Then we will present our algorithm which allows to fill contours image by using this structure. Finally we will provide some examples and try to highlight the advantages and the geometrical extensions of such an approach. The article will end with a conclusion and some perspectives.

In this article, the background pixels of the images are the white ones and contours are drawn in black. Furthermore contours are composed of digital curves (closed or not) which connectivity is 8 with no redundancy. These properties are guaranteed by our processing chain which applies a thinning algorithm.

2 Recalls

Topological maps are an extension of combinatorial maps [7–10] in order to represent in a unique and minimal way a labeled image. We present here briefly the main notions of combinatorial maps and of topological maps (see [11, 12] for more details).

2.1 Combinatorial Maps

Intuitively, a 2D combinatorial map (called also a *2-map*) is an extension of a planar graph that keeps the orientation of edges around each vertex. Each edge of the graph is divided in two parts. Basic elements obtained are called *darts* and are the unique basics of the combinatorial map definition. A 2D combinatorial map can represent the topology of a 2D subdivision of orientable spaces without boundary. This model has been extended to represent any type of subdivision, orientable or not, and with or without boundaries (see [11] and Fig. 3).

More precisely, a subdivision of a 2D topological space is a partition of the space into 3 subsets whose elements are 0D, 1D and 2D *cells* (respectively called vertices, edges and faces, and noted i-cell, $i = 0 \dots 2$). Border relations are defined between these cells, where the border of an i-cell is a set of *(j<i)*-cells. Two cells are *incident* when one belongs to the border of the second, and two i-cells are *adjacent* if they are both incident to a common *(j<i)*-cell.

A combinatorial map is an algebra composed by a set of darts that represents the elements of the subdivision, and 2 mappings (called β_1 and β_2) defined on these darts that represent adjacency relations (this can be easily extended in nD, with n mappings). β_1 puts in relation a dart and the next dart of the same face, and β_2 puts in relation both darts incident to a same edge. These β_i have to verify some particular properties in order to ensure the validity of the represented subdivision (β_1 is a permutation and β_2 is an involution, see for example [11] for the formal definition).

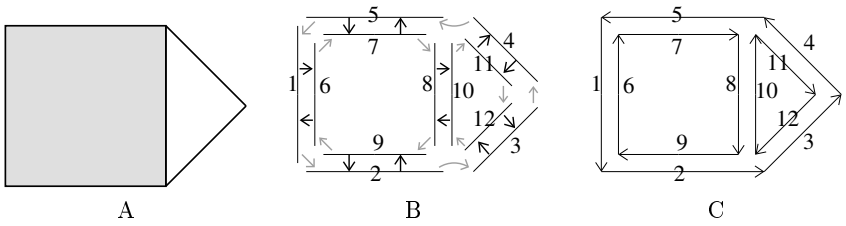

Fig. 3. Usual representation of a 2D combinatorial map. (A) A 2D object. (B) Explicit representation where each dart and each one to one mapping are drawn. Darts are represented by black segments, β_1 by grey arrows and β_2 by black arrows. (C) Implicit representation, where β_i applications are not explicitly drawn but can be deduced from the shape of the objects. Two darts in relation by β_1 are drawn consecutively, and the arrow on darts shows the orientation of β_1. Two darts in relation by β_2 are drawn near, parallel and in reverse orientation.

We can see in Fig. 3B the combinatorial map representing the object shown in Fig. 3A. In this example, each dart and each one to one mapping are drawn. In general, we do not use this representation but we prefer the one shown in Fig. 3C where β_i are not explicitly drawn but can be (generally) deduced from the shape of objects.

Within the combinatorial map framework, all cells are implicitly represented through the notion of *orbit*. Intuitively, an orbit $< \beta_{i_1}, \dots, \beta_{i_j} > (d)$ is the set of darts that can be reached with a breadth-first search algorithm, starting with d, and using all combinations of all β_{i_k} or $\beta_{i_k}^{-1}$ permutations $\forall k, 1 \leq k \leq j$. With this notion, each cell is defined as a particular orbit. Based on the cells definition, we can retrieve the classical *cell degree* notion. The degree of an i-cell c is the number of distinct *(i+1)*-cells incident to c. Note that in a n-dimensional

space, the degree is not defined for n-cells, since *(n+1)*-cells do not exist in such a space.

2.2 Topological Maps

Topological maps are an extension of combinatorial maps in order to represent in a unique and minimal way a labeled image.

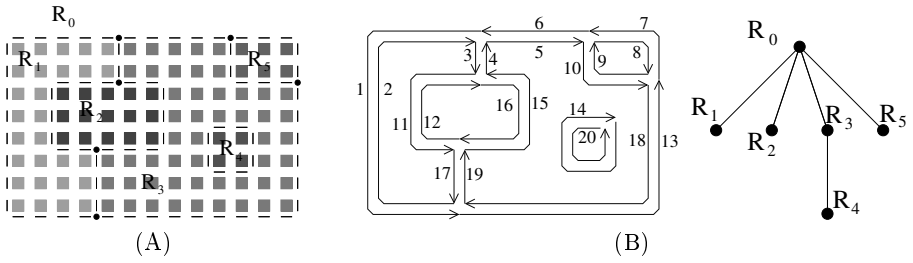

Fig. 4. (A) A 2D labeled image drawn with its interpixel boundaries. (B) The corresponding topological map with its inclusion tree.

We can see in Fig. 4 a 2D labeled image and the corresponding topological map. A topological map is a combinatorial map that represents a labeled image and that verifies particular properties. Indeed, this map is minimal, complete and unique. These properties lead to another characteristic of the topological map: each edge represents exactly an interpixel boundary between two regions of the image (this can be seen in Fig. 4). An interpixel boundary between two regions R_i and R_j, is the set of maximal interpixel curves such that each linel of these curves is incident to exactly one pixel of R_i and one pixel of R_j (see [12] for proofs concerning topological map properties).

When a region is included into another region (as region R_4 in Fig. 4 which is included into region R_3), the corresponding topological map is composed of several connected components. There is no information in the map that allows to place relatively the different connected components, and thus we have lost the topological information concerning the inclusion. To solve this problem, we add an inclusion tree to the topological map. This tree contains each region of the image, rooted by R_0[1], and a region R_i is son of a region R_j when R_i is included into R_j. With this tree, we are now able to retrieve each inclusion relation. Moreover, each region of this tree R is linked with a dart of the topological map that belongs to its external boundary (called *representative dart* of R), and each dart of the map is linked with its belonging region. With these two links, we can efficiently run through all the boundaries of a given region.

[1] R_0 is the region which surrounds all the image, called the infinite region.

Combinatorial map represents the topological part of our model: all the cells of the space subdivision and all the adjacency and incidence relations. But it is also necessary to represent the geometry of the image. We speak about *embedding* to design this geometrical model. There are many different possibilities to represent the geometry and the choice of one of them depends on the application. In this work, we have chosen to use an interpixel matrix.

This matrix contains all the interpixel elements that belong to interpixel boundaries of the corresponding image. We can see in Fig. 4(A) the corresponding embedding of the topological map shown in Fig. 4(B). A linel is present in the matrix if it is between both pixels that belong to two different regions. A pointel is present in the matrix if it is incident to more than two linels.

Each dart of the topological map is linked with a doublet (p, l) that allows to retrieve, given a dart d, the corresponding cells in the interpixel matrix. With p we can retrieve the pointel associated with d, and so the coordinates of the corresponding vertex. With (p, l) we can retrieve the first linel associated with d. This linel is oriented and gives the initial direction of the edge associated with d. To retrieve the embedding of the edge incident to dart d, we start from this linel, and follow the path of linels until we find a pointel, or we go back to the initial linel.

3 Using Topological Map for Filling Contours

Given a contour image obtained from a character image after contour approximation, filtering and thinning (Fig. 1), we want to fill each region which corresponds to the interior of a character. The main idea of our solution is to use topological map, and more precisely the inclusion tree associated with topological map in order to retrieve one pixel for each region to fill, and then to use a classical flood-fill algorithm starting from these germs.

Due to the type of our images, each region to fill R can be characterized by two specific properties:

1. the color of R is white in the image;
2. the depth of R in the inclusion tree is even.

The first property can easily be deduced since black pixels in the images belong to a boundary of a character. The second property is deduced from the type of boundaries present in the image as we can see in Fig. 5. Indeed, in our images, two types of boundaries are alternated: external boundaries and internal boundaries. Indeed, each region is always composed by one external boundary. When it has some holes, each one is represented by one internal boundary. This process is repeated if there is another region included in one hole, this region is represented by an external boundary and so on.

With these considerations, we can characterized each boundary by its depth in the inclusion tree. By considering that a character is never incident to the border of the image[2], we are sure that the region R_e associated to the external

[2] Even if this property is not always true, we can easily modify the image by adding white pixels all around it in order to verify the property.

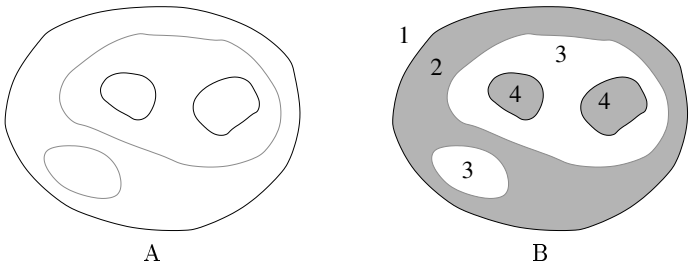

Fig. 5. An example of images we have to process. (A) Initial image. We have drawn external borders in black and internal borders in grey, but this is only for the explanations and all the borders are represented by black pixels in the real image. (B) Image we want to obtain after interiors of characters are filled. The numbers correspond to the depth of each region in the inclusion tree.

boundary is included in the region that corresponds to the background of the image R_b. The depth of R_b region is 1 since it is directly included in the infinite region, and thus the depth or R_e is 2 since R_e is directly included in R_b.

It is important to note that the regions represented in topological map are 4-connected regions and that each external boundary is a 8-connected path. Consequently, if an external boundary is not a single straight line, it is always associated with more than one region. Those regions are all at a depth 2 in the tree since each one is directly included in R_b. Furthermore, an external boundary can not include any region. Then, the white region R_f delimited by an external boundary is also to a depth 2 in the tree. Indeed, R_f is adjacent to regions representing the external boundary, and by definition of inclusion tree, two adjacent regions have the same depth. Thus we can conclude that each white region with a depth 2 in the tree is a region delimited by an external boundary and thus need to be filled.

Now if we consider an internal boundary, and the region R_i associated with this boundary (we can do the same remark than for external boundary, it is possible to have several regions associated with an internal boundary, but each one has the same depth in the tree). The depth of region R_i is 3 since this region is included into R_f. Indeed, otherwise, the boundary is not an internal boundary since it is adjacent to an external boundary. The white region delimited by this boundary has the same depth in the tree, but this region must not be filled since this is a hole in the surrounding region.

Now, we can do exactly the same remarks for the next external boundary, associated to a region with depth 4, and so on, to conclude that we need to fill each white region with an even depth in the inclusion tree.

Thanks to these properties, we can deduce Algorithm 1 which, given a topological map, computes the list of each germ that belongs to a region to fill.

Algorithm 1: Computation of the list of germs that belong to all regions to fill.

Input: A topological map M
Output: The list of pixels that are all the germs belonging to regions to fill.

$res \leftarrow \emptyset$
foreach *region* r *of the inclusion tree* **do**
  if *the depth of* r *is even* ***and*** *the color of* r *is white* **then**
    $d \leftarrow$ representative dart of r
    $(p, l) \leftarrow$ doublet associated with d
    add the pixel associated with (p, l) in res
return res

Given a dart that belongs to a region to fill, we need to find a pixel inside the region. For that, we first recover the doublet (p, l) associated to the dart. Then, depending on the linel, we can compute a pixel inside the region. Indeed, edges of the map are counter-clockwise oriented, and thus we know that given an edge of an external border, the interior of the region is always on the right of the oriented edge. As we can see in Fig. 6, there are only four possible configurations, and depending on the configuration we can directly retrieve the coordinates of the pixel, given the coordinates of the pointel.

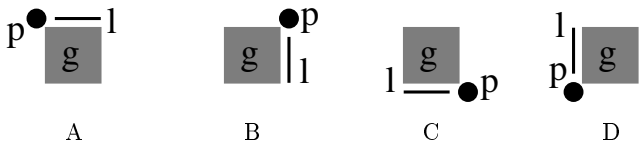

Fig. 6. The four possible configurations of a doublet (p, l) and the associated pixel g for each case. We note (p_x, p_y) the coordinates of the pointel and (g_x, g_y) the coordinates of the pixel. (A) (g_x, g_y)=(p_x, p_y). (B) (g_x, g_y)=$(p_x - 1, p_y)$. (C) (g_x, g_y)=$(p_x - 1, p_y - 1)$. (D) (g_x, g_y)=$(p_x, p_y - 1)$.

The complexity of Algorithm 1 is linear in number of regions of the image. Indeed, this algorithm runs through all regions of the image and for each region to fill, just computes the coordinates of one pixel inside the region by atomic operations. This algorithm is thus very efficient since we do not need to run through all the pixels of the image. Note that this algorithm needs a topological map, but the computation of topological map can be considered as a pre-processing operation. Moreover this computation can be achieved very quickly by using optimal extraction algorithm [12] with a complexity linear in number of pixels of the image, but also with a single image scan and with only the minimal number of operations to applied for each pixel.