Segment-based simple-connectivity measure design and implementation

Joao Soares[1], *Andrea Baraldi*[2], *and David Jacobs*[1]

[1] Department of Computer Science, University of Maryland
[2] Department of Geography, University of Maryland, 4321 Hartwick Rd, Suite 209, College Park, Maryland, USA 20740.
E-Mail: andrea.baraldi@hermes.geog.umd.edu
Tel.: +1-301-314-1467; Fax: +1-301-405-6806

Table of Contents

1 Introduction

In developing different measures for the description of a segment's shape, we noted that it would be useful to include a measure capable of quantifying the presence of holes. This was motivated by the following scenario. The measures we use to characterize a segment's shape, such as RoundnessAndNoHole (also known as compactness), ConvexityAndNoHole and RectangularityAndNoHole are monotonically decreasing with the presence of holes, namely:

- RoundnessAndNoHole is high if Roundness is high and condition NoHole is true,
- ConvexityAndNoHole is high if Convexity is high and condition NoHole is true and, finally,
- RectangularityAndNoHole is high if Rectangularity is high and condition NoHole is true.

For example, a region with a perfectly round external boundary, but containing several holes, will present a low RoundnessAndNoHole measure. Were the holes not present in the region, it would instead feature a very high RoundnessAndNoHole measure. Besides these measures, our newly introduced version of a measure of elongatedness[1] is also affected by the presence of holes, increasing as the number of holes increases.

In our study of satellite images, it is very common to find segments that contain holes, whether due to the underlying holes in the original observed structure or whether due to segmentation errors. In order to reason about these types of situations without having to change the definitions of the shape measures already in use (which are quite natural and intuitive), we introduce a new measure to quantify the presence of holes, which we call simple-connectivity. The simple-connectivity measure quantifies the extent to which a region is simply-connected, i.e., the measure should be monotonically decreasing with holes whose cardinality increases or whose size increases (at fixed cardinality).

The rest of this paper is organized as follows. In Section 2, nomenclature is introduced. Existing definitions of simple-connectivity indexes are provided in Section 4, which provides this document with a relevant survey value. Section 5 presents an original combination of indexes of

[1] In particular, Elongatedness = Number of pixels belonging to the 8-adjacency skeleton (to replace the traditional segment's longest path, which requires a segment to be simply-connected) / Average width = [Mean(Sum(2 × min distance from the boundary of the skeleton pixel, where this distance is computed by the Euclidean distance transform generated during the skeletonization process))], such that Elongatedness ≥ 1.

simple-connectivity. As a supplement, Appendix A presents a review of algorithms for tracing the region boundaries. Appendix B presents an alternative simple-connectivity index based on boundary lengths. Appendix C describes an efficient original algorithm for computing the length of a region's boundaries, to be used in computing the alternative measure presented in Appendix B.

2 Nomenclature

Following is a list with some nomenclature used in this document.

1. External (region) boundary.
2. Outer external boundary.
3. Internal (region) boundary (facing holes).
4. Outer internal boundary.
5. Total (region) boundary = External boundary + internal boundary.
6. Total outer boundary = outer external boundary + outer internal boundary.

These terms are defined in Section 5. Throughout, in order to define our measures of simple-connectivity, we will only consider *region* boundaries, and thus we will only be using terms 1, 3, and 5 above. The *outer* boundaries will only be discussed briefly, so terms 2, 4, and 6 above are included here more for completeness.

3 Project requirements specification

These are our requirements of a geometric index.

- Dimensionless (it is a pure number).
- Normalized range of change (intuitive to deal with by a user).
- Scale-invariant.
- In compliance with human perception. For example, a simple-connectivity index should be monotonically decreasing with holes whose cardinality increases or whose size increases when the number of holes is fixed.
- Robust to the presence of noise in the input data. For example, in satellite imagery an increasing level of data noise may increase, firstly, image fragmentation, i.e., it increases the number of image-objects (segments) along with the average presence of holes within each segment and, secondly, the roughness of image-objects whose boundaries become less smooth (more irregular).

4 Existing definitions of simple-connectivity indexes

This section investigates existing definitions of simple-connectivity indexes to provide this document with a relevant survey value.

Much work in shape analysis does not provide an appropriate treatment of holes as integral characteristics of shapes. Holes are sometimes eliminated in a pre-processing stage as a nuisance. Similarly, there is a body of work that focuses on characterizing only the shape's external boundary (refer to the adopted nomenclature in Section 2), in effect ignoring eventual holes [1]. Our work in developing indexes for shape description follows the line of treating holes as important constituents of a shape. Besides the practical importance of taking such an approach, discussed in Section 1, this view is also interesting as it holds a counterpart in human perception, as discussed below.

Bertamini [2] reviews a series of works that strongly indicate that *the borders of holes are not perceived as belonging to the hole itself, but to the region that encloses it*. In this sense, from a perceptual standpoint, the boundary that surrounds a hole is actually contributing to define the shape of the enclosing region. (This does not imply that the shape of the hole itself is not perceived at some level, but that its boundary is in fact perceived as an integral part of the shape that surrounds it.) It is important to point out that, in dealing with satellite images, in many cases there is not a clear definition of what region perceptually constitutes "figure" (*foreground*) or "ground" (*background*), since many regions present the same, or similar, (3-D) depth. Therefore it is not always clear which regions should be considered perceptually as foreground versus hole. Nonetheless, in considering the problem of shape analysis in general, if we assume that we are given a foreground region, it becomes perceptually relevant to use the boundaries of the region's holes to help define its shape.

Wentz [3] defines a measure to aid in characterizing shapes in geographic applications. Wentz's measure is called *Perforation*, defined as the ratio between the area of a segment's holes and the area of the segment when all its holes are filled in. In this manner, *Perforation* $\in$ [0, 1]. In addition to being dimensionless and normalized, which make it easy to use, index *Perforation* is also scale invariant, which is a fundamental property of geometric descriptors.

Soffer and Samet [4] define a *Hole-Area Ratio* for recognition of symbols and logos. The Hole-Area Ratio is defined as the ratio between the area of a segment's holes and the segment's area (without its holes filled in). This index is in the same spirit as the Perforation index, with the difference that the denominator in the Perforation index is the area of the segment *with* its holes filled in. Though in practice the indexes may present similar behavior, the Hole-Area ratio is not contained in the [0,1] range and thus is less desirable for our purposes.

Topological properties of a region can also be used to quantify the presence of holes, such as the number of holes, or Euler number [1] [5]. The limitation of topological measures is that they do not quantify the size or shape of holes, focusing instead solely on counting their number. In our applications, we would like to quantify the size/extent of holes, and thus we opt for different types of indexes.

5 Original definitions of simple-connectivity indexes

We begin by defining three relatively simple original simple-connectivity measures. These more simple measures are defined in Sections 5.1, 5.2, and Appendix B. The first two of these indexes are of interest to us, since they satisfy the requirements specification listed in Section 3. At the same time, these two indexes are complementary in that they present different kinds of sensitivities to holes. In order to develop a final index that is sensitive to these different types of holes, our final formulation is derived as a combination of the two original indexes, which is presented in Section 5.3. The third index, although not completely scale invariant, is approximately scale invariant. It is described in Appendix B.

5.1 Simple-connectivity index based on boundary 4-adjacency cross-aura measures

We define our simple-connectivity index, based on boundary 4-adjacency cross-aura measures, starting from some basic definitions.

We consider a 2-D *region*, also called 2-D *segment*, *polygon* or (2-D) *image-object*, to be any set of 8-connected pixels on a usual 2-D discrete rectangular grid. Sonka *et al.* distinguish between two types of boundaries where regions are assumed to be simply connected, i.e., where image-objects are assumed to have no holes [6] (pp. 129-131).

- A simply-connected region's *inner* boundary is traced by searching the image from top left until a pixel of the target region is found. Next, an inner boundary tracing algorithm moves anticlockwise along the region's boundary pixels. By definition, boundary pixels belong to the region at hand, are either 4- or 8-adjacency connected to one another and face, respectively in their 8- or 4-adjacency neighborhood, at least one non-region pixel. Thus, inner boundary pixels form a subset of the region, see Figure 1. A review of an algorithm for tracing this *inner* boundary is presented in Appendix A.
- A region's *outer* boundary denotes outer border pixels not belonging to the target region, i.e., pixels facing the region from the “outer” world. If the region inner boundary is 8- (4-)adjacency connected, then outer pixels are those non-region pixels faced by inner boundary pixels in their 4-(8-)adjacency neighborhood, see Figure 1.

	o	o		o			
o	x	x	ooo	x	oo		
	oo	x	x	x	x	oo	
	o	x	x	ooo	x	x	o
		o	o		o	o	

Figure 1: Example of a region’s 8-adjacency inner boundary, in dark highlight, and the region's outer boundary (with o meaning one count in the 4-adjacency neighborhood of a region's inner boundary pixel). (This example was provided by Sonka *et al.* [6], page 131.)

In this paper we consider only *8-adjacent* region boundaries, which are boundaries whose pixels are 8-adjacency connected.

Introduced in Section 2, a different terminology from that of Sonka *et al.* is adopted hereafter.

(a) External 8-adjacency region boundary. Set of pixels that: (i) belong to the region (ii) are 4-adjacent to an outer pixel which: (a) falls, by definition, outside the region and (b) does not belong to an inner hole of the region.
(b) Internal 8-adjacency region boundary (facing holes). Set of pixels that: (i) belong to the region and (ii) are 4-adjacent to an outer pixel which: (a) falls outside the region and (b)

belongs to an inner hole of the region.

(c) Total 8-adjacency region boundary = External 8-adjacency region boundary + internal 8-adjacency region boundary. Set of pixels that: (i) belong to the region and (ii) are 4-adjacent to any outer pixel that (by definition of outer pixel) falls outside the region, irrespective of whether this outer pixel belongs to an inner hole of the region at hand.

These definitions are illustrated by the example in Figure 2. Since most of the boundaries that we consider in this paper are *region* and *8-adjacent*, we will omit these terms throughout. Thus, the three previously defined boundaries will be referred to simply as external boundary, internal boundary, and total boundary.

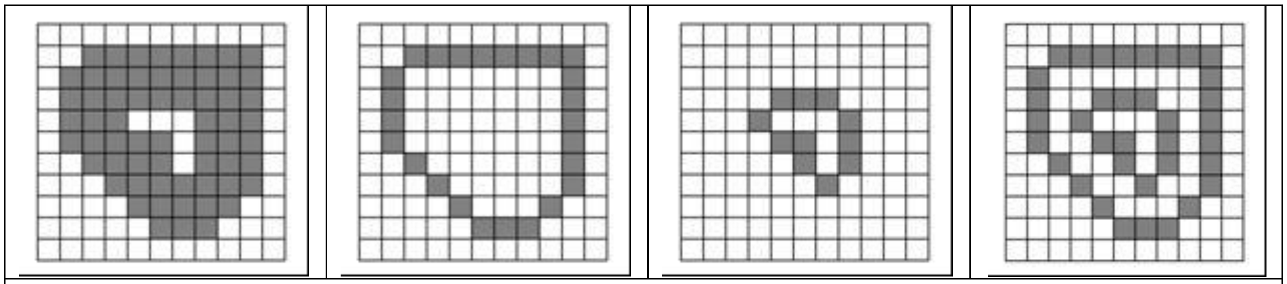
Figure 2: Example illustrating a region's 8-adjacency region boundaries (a) region (gray pixels) containing a hole; (b) the corresponding external 8-adjacency region boundary; (c) the internal 8-adjacency region boundary; (d) the total 8-adjacency region boundary, which is the same as the union of the external and internal 8-adjacency region boundaries.

We go on to define two types of cross-aura measures of importance. The first is the "4-adjacency cross-aura measure of the *external* boundary". For each pixel of the external boundary, we count its 4-neighbors that: (i) do not belong to the region, and (ii) do not belong to the region's holes. These counts are tallied up, resulting in the desired cross-aura measure. The counting procedure is such that a pixel that does not belong to the region and is a 4-neighbor to multiple external boundary pixels will be counted multiple times (once for each of its 4-neighboring external boundary pixels), as an example, refer to Figure 1.

The "4-adjacency cross-aura measure of the *total* boundary" is defined similarly, but also takes into account contributions from holes. For each pixel in the *total* boundary, we count its 4-neighbors that do not belong to the region, irrespective of whether they belong to a hole or not. These counts are added up over all pixels in the total boundary. Again, a pixel that does not belong to the region and is a 4-neighbor to multiple boundary pixels will be counted multiple times.

We are now ready to define our first measure of simple-connectivity, which we denote *SmplConnctvty4Adjacency*, as

SmplConnctvty4Adjacency = (4-adjacency cross-aura measure of the *external* boundary) / (4-adjacency cross-aura measure of the *total* boundary), (1)

where *SmplConnctvty4Adjacency* $\in$ [0, 1], such that *SmplConnctvty4Adjacency* is equal to 1 if and only if the region has no holes, i.e., if it is simply-connected.

It is easy to prove that pros and cons of *SmplConnctvty4Adjacency* are the following (refer to further Section 0).

- Pros.
    - Dimensionless (it is a pure number).
    - Normalized range of change (intuitive to deal with by a user).
    - Scale-invariant.
- Cons.
    - Somehow contrary to human perception when a single hole of increasing size is met, it does not cause *SmplConnctvty4Adjacency* to decrease significantly relative to the size of the hole.

In the following subsections, we will present numerical examples to illustrate the properties above.

The computation of the cross-aura measures is done in the following manner. We begin by explaining the procedure used to compute the 4-adjacency cross-aura measure of the *total* boundary. Our system for geographical image analysis provides us with an image indicating which pixels belong to the region to be analyzed, as well as the coordinates of a bounding box around the region. To compute the 4-adjacency cross-aura measure of the total boundary, we visit each pixel inside the bounding box and only analyze those that belong to the region. (Alternatively, we could analyze only the set of pixels that belong to the total boundary, if this set is available in some data structure.) For each pixel belonging to the region, we count the number of its 4-neighbors that do not belong to the region. This number is accumulated over all pixels, resulting in the desired cross-aura measure. Note that, in accordance with the definition of the cross-aura measures presented above, a non-region pixel will be counted multiple times if it is a 4-neighbor to multiple region pixels.

To compute the 4-adjacency cross-aura measure of the *external* boundary, we follow the same procedure as above, after having filled in the holes of the region. In this manner, we avoid counting any contribution of non-region pixels that belong to holes. The region with its holes filled in is stored as an intermediate product of our image analysis system, as it is also used to compute the *FilledAreaRatio*. The procedure used to fill in the holes of a given region is described along with the definition of *FilledAreaRatio*, in Section 5.2. To say that, when the 4-adjacency cross-aura measure of the *external* boundary is computed, this computation benefits from pre-filling of the region's holes, if any.

Perhaps a more direct approach to compute the 4-adjacency cross-aura measure of the external boundary would be to trace the external outer boundary as described by Sonka *et al*. in their “outer boundary tracing” algorithm. This algorithm is briefly reviewed in Appendix A.

5.1.1 **Test case: scale invariance**

To illustrate how *SmplConnctvty4Adjacency* behaves under variations in the scale of a region, we present a simple example. (a) shows a region containing a hole, where we have *SmplConnctvty4Adjacency* =12/16 = 0.75. We then scale the region to twice its size, as shown in (b). The result is *SmplConnctvty4Adjacency* =24/32 = 0.75. Note that the value has not changed.

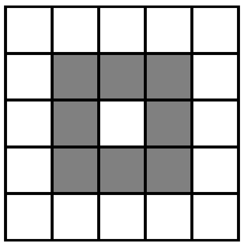
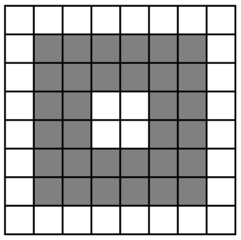

Figure 3: An example to illustrate the behavior of the simple-connectivity measures when a region is subjected to scaling. The regions are denoted by the set of gray pixels, so that each region has a single hole. In (a), a one-pixel thin region with a hole is shown. If the image in (a) is enlarged to twice its size, we obtain the result shown in (b). See the text for the simple-connectivity values corresponding to the regions.

5.1.2 **Test case: one-pixel thin region with a hole**

The definition of *SmplConnctvty4Adjacency* (see Equation 1) is justified to properly deal with a particularly difficult type of (connected) region which is not simply connected, i.e., this region has (inner) holes. shows an example of such a difficult region. The region is one-pixel thin and contains a relatively large hole. In this example, a practical difficulty arises due to the fact that the set of pixels that form the external boundary is the same as the set of pixels that form the total boundary, even though a hole is present.

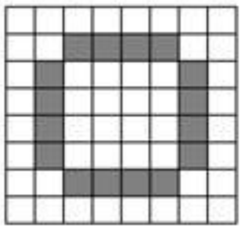

Figure 4: Example of a pixel-thin region containing a relatively large hole. The region is defined by the set of pixels marked in gray. Note that in this case, the pixels that form the region, its external boundary, and its total boundary all coincide.

We will briefly discuss an alternative approach to defining simple-connectivity, which is more direct and simple, but less effective (i.e., less sensitive to the presence of holes, refer to this text below). Instead of considering 4-adjacency cross-aura measures, one could attempt to directly detect boundary pixels and simply count them up. This alternative approach would consist of the following steps.

(a) Count the number of pixels that form the external boundary. Recalling the definition in Section 5.1, the set of pixels forming the external boundary are the pixels that: (i) belong to the region and (ii) are 4-adjacent to an outer pixel which: (a) falls outside the region and (b) does not belong to an inner hole of the region. For example, the region in Figure 2(a) presents 25 such pixels (shown in Figure 2(b)), whereas the region in presents 16 such pixels, which coincide with the region itself.

(b) Count the number of pixels that form the total boundary. To recall the definition from Section 5.1, the total boundary is the set of pixels that: (i) belong to the region and (ii) are

4-adjacent to any outer pixel that (by definition of outer pixel) falls outside the region, irrespective of whether this outer pixel belongs to an inner hole of the region at hand. The region in Figure 2(a) presents 36 such pixels (shown in Figure 2(d)), while the region in presents 16 such pixels, which again coincide with the complete region.

(c) Take the simple-connectivity to be the number of pixels that form the external boundary (from (a)) divided by the number of pixels that form the total boundary (from (b)).

Noteworthy, by using this alternative approach, the region in Figure 2 would present a simple-connectivity measure of 25 / 36 below 1, which indicates the presence of holes. Also *SmplConnctvty4Adjacency* would result in a value of 36 / 48 below 1, which appears reasonable. On the other hand, for the region type shown in , where at least part of a region is one-pixel thin and presents a hole, which shows up frequently in the segments produced by our analysis of satellite images and thus needs to be dealt with appropriately, the *SmplConnctvty4Adjacency* index value is equal to 24 / 40, well below 1, which correctly reflects the presence of the hole in the region. However, the aforementioned alternative approach would result in a simple-connectivity of 1. This is undesirable, since the region is certainly not simply-connected.

To recapitulate, the original *SmplConnctvty4Adjacency* index proposed in Equation 1, based on 4-adjacency cross-aura measures, complies with geometric index requirements proposed in Section 3 and effectively copes with the difficult test case shown in Figure 4, where at least part of a region is one-pixel thin and presents a hole, which shows up frequently in the segments produced by our analysis of satellite images and thus needs to be dealt with appropriately.

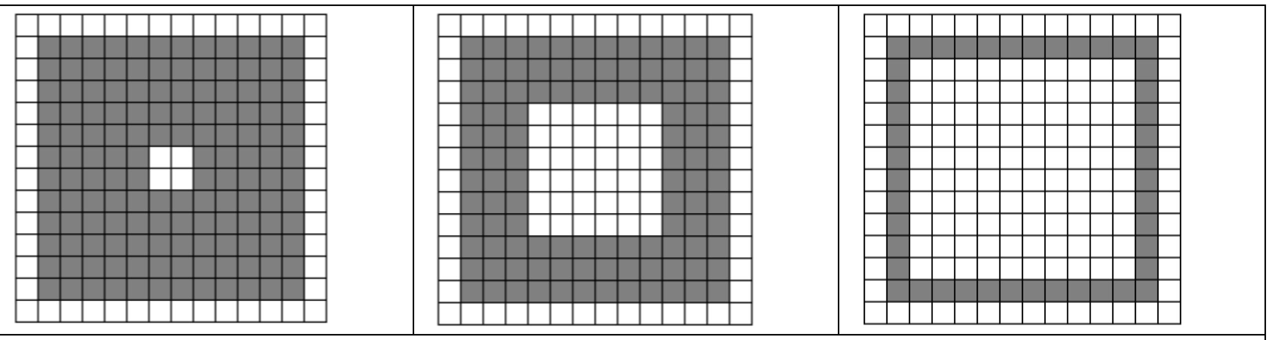

Figure 5: Example regions used to illustrate the sensitivities of the different simple-connectivity measures to a hole of increasing size. See text for the simple-connectivity values associated to each region.

5.1.3 **Test case: sensitivity to a single hole of increasing size**

The next example shows how *SmplConnctvty4Adjacency* responds to a single hole of increasing size. We use to illustrate this. In (a), the size of the hole is relatively small, resulting in *SmplConnctvty4Adjacency* ~ 0.86, while for the square with the medium-sized hole in (b), we obtain *SmplConnctvty4Adjacency* ~ 0.67. Finally, the square with the large hole in (c) has *SmplConnctvty4Adjacency* ~ 0.55. Note that, though in (c) we see a very large hole, *SmplConnctvty4Adjacency* takes on the value of 0.55, which is intermediate in its range, instead of being "small". This does not correctly align with our perception regarding the size of the hole. In practice, index *SmplConnctvty4Adjacency* does not appear to be sensitive "enough" to changes in size of holes whose cardinality is fixed. In other words, index *SmplConnctvty4Adjacency* does not appear effective in exploiting its whole normalized output domain of change [0, 1] to describe input changes in size of holes whose cardinality is fixed.

5.1.4 **Test case: sensitivity to the presence of multiple holes**

The following example illustrates how *SmplConnctvty4Adjacency* behaves in the presence of multiple holes. presents two squares with holes. In (a), the square contains 4 small holes, each of 1 pixel in size. This region results in *SmplConnctvty4Adjacency* ~ 0.86. (b) presents a square of the same size, but with a single hole of 4 pixels in size. For this case, we obtain *SmplConnctvty4Adjacency* ~ 0.56. Note that *SmplConnctvty4Adjacency* is sensitive to the variation in the number of holes, increasing with the increase in number of holes.

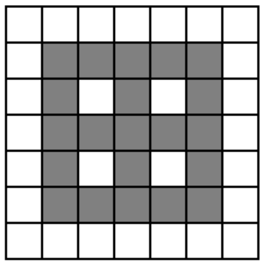
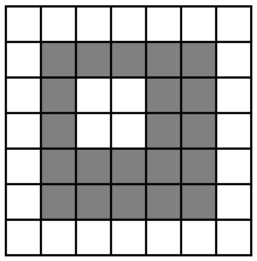

Figure 6: Two regions with holes, used to illustrate the sensitivities of the simple-connectivity measures to the presence of multiple holes. The total area of the holes is the same in example (a) and (b), though in (a) the region contains 4 holes, while in (b) there is only one.

5.1.5 **Test case: sensitivity to a noisy external boundary**

In this example, we show the effect of a noisy external boundary on *SmplConnctvty4Adjacency* where the internal region boundary facing holes is kept the same. On theory, a case where noise affects only the external region boundary and not the inner region boundary facing holes seems an unlucky circumstance. In practice, this situation seems to frequently affect segments generated in our satellite image analysis.

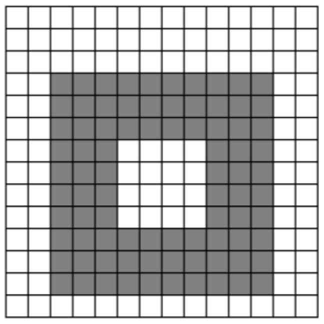
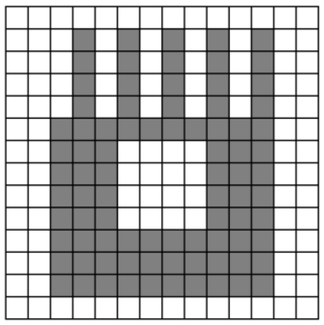

Figure 7: Regions used to illustrate the effects of a noisy external boundary on the values of our simple-connectivity measures. See text for discussion.

In , both regions have the same area and present the same hole. However, the external boundary of the region in (b) is noisier (and thus longer) than the one of the region in (b). The region in (a) presents *SmplConnctvty4Adjacency* ~ 0.71, while the region in (b) presents *SmplConnctvty4Adjacency* ~ 0.83. *SmplConnctvty4Adjacency* is thus affected to a certain extent by the noisy boundary.

5.1.6 **Test case: range of SmplConnctvty4Adjacency values**

When designing an expert system for classification of image-objects (e.g., detected in a satellite image), namely, a prior knowledge-based vision system, which is our case, it is desirable to have shape indexes with an intuitive physical meaning and values in a finite range (bounded above and below) of pure numbers, like percentage values in the normalized range of change [0, 1]. This allows a knowledge engineer or domain expert to more easily design decision thresholds to quantize shape index values of target objects. According to Section 3, this is one of the project requirements which drives our search for a simple-connectivity index.

In this subsection, the effectiveness (sensitivity) of the original index *SmplConnctvty4Adjacency* = (1) is tested in a variety of input cases, shown in . *SmplConnctvty4Adjacency* presents a value of 1 when the region is simply-connected, since there are no holes contributing towards the 4-adjacency cross-aura measure of the total boundary. The measure will present a value close to 0.5 when a region contains a large hole, such as in the example shown in (c) and (b). A region may also present values around 0.5 when the region contains a series of smaller holes, as shown in (c). Finally, in the extreme case of a region that contains several holes, each with long boundaries, *SmplConnctvty4Adjacency* will be very low, as in (d).

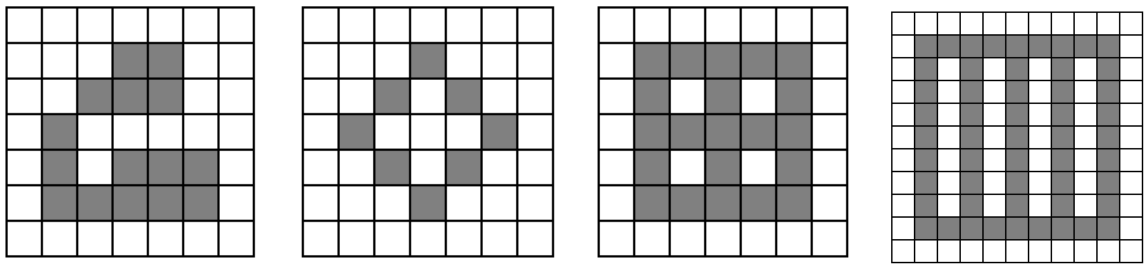

Figure 8: Example of regions illustrating the range of SmplConnctvty4Adjacency values: (a) an 8-connected region with no holes, presenting a SmplConnctvty4Adjacency value of 1; (b) a region with a large hole, presenting SmplConnctvty4Adjacency equal to 20 / (20 + 12) ~ 0.52; (c) a region with several small holes, presenting SmplConnctvty4Adjacency equal to 20 / (20 + 4 + 4 + 4 + 4) ~ 0.56; (d) a region with several holes, each presenting long boundaries. The SmplConnctvty4Adjacency value of this last region is 36 / (36 + 16 + 16 + 16 + 16) = 36 / 100 = 0.36.

5.1.7 **Conclusions about the first proposed shape index**

Based on Sections 5.1.1 to 5.1.6, our experimental conclusions about the original index *SmplConnctvty4Adjacency* = (1) are that:

- Overall, its numerical behavior complies with the project requirements specification of Section 3.
- Possible improvements to counterbalance its operational limitations.
    - *SmplConnctvty4Adjacency* appears to exploit only one half of its output normalized domain of change to describe input holes that change in size, rather than in number (cardinality).
    - In addition to the presence of holes, *SmplConnctvty4Adjacency* is somehow sensitive to noisy (irregular) external boundaries when the inner region boundary facing holes remains unaffected by noise (which appears as an unlucky circumstance, but seems to occur in practice).

5.2 Simple-connectivity index based on areas

To improve the original index *SmplConnctvty4Adjacency* = (1) in compliance with Section 5.1.7, we keep searching for a better simple-connectivity index driven by the project requirements specification of Section 3.

The next simple-connectivity index we experiment with is based on the simple ratio between the total area of the holes divided by the area of the region with its holes filled in. Similar area-based shape indexes proved to be scale invariant, refer to Section 4. Intuitively, such an area-based simple-connectivity index is expected to behave very differently from the contour-based *SmplConnctvty4Adjacency* = (1) formulation. This intuition is verified in a series of examples in this section.

We define an "original" *FilledAreaRatio* index as

FilledAreaRatio = (Area of the region) / (Area of the region with its holes filled in), (2)

such that *FilledAreaRatio* $\in$ [0, 1], where *FilledAreaRatio* equals 1 if there is no hole in the polygon.

Noteworthy, this index is inversely related to shape indexes presented by Wentz [3] and Soffer and Samet [4], in fact, *FilledAreaRatio* = (1 – *Perforation*) $\in$ [0, 1]. In other words, index *FilledAreaRatio* is not totally new, but it is somehow new based on the degree of novelty of the framework (refer to Section 3) where it is formulated and adopted.

It is easy to prove that pros and cons of *FilledAreaRatio* are the following (refer to further Section 0).

- Pros.
    - Dimensionless (it is a pure number).
    - Normalized range of change (intuitive to deal with by a user).
    - Scale-invariant.
- Cons.
    Little sensitivity to the presence of multiple small holes. This behavior is somehow complementary to that of index *SmplConnctvty4Adjacency*.

Later on, we present numerical examples that illustrate the properties above.

We implement the computation of the shape index *FilledAreaRatio* in the following manner. Our system for spaceborne/airborne image analysis provides us with an image indicating which pixels belong to the region being analyzed, as well as the coordinates of a bounding box around the region. In order to compute the area of the region, we visit each pixel inside the bounding box and count the pixel towards the area each time that it belongs to the region.

In order to compute the area of the region with its holes filled in, we follow the same procedure as above, but after having first filled in the holes of the region. To fill in the holes, we follow a two-step approach.

1. The external boundary of the region is traced (see Appendix A).
2. Given the tracing along the external boundary, the region enclosed by the boundary is

drawn as a new image, which will not present any of the holes. The algorithm we use to draw the region after tracing its external boundary is the scan-line polygon filling algorithm [8] (pages 169-173), whose implementation is provided in the OpenCV library [7].

5.2.1 **Test case: scale invariance**

To illustrate how *FilledAreaRatio* behaves under variations in the scale of a region, we recall the example from Figure 3. (a) shows a region containing a hole, which presents *FilledAreaRatio* = 8/9 ~ 0.89. When the region is scaled to twice its size, as shown in (b), the resulting measure is again *FilledAreaRatio* = 32/36 ~ 0.89. Note that the value has not changed, as was also the case with *SmplConnctvty4Adjacency* (see Section 5.1.1).

5.2.2 **Test case: sensitivity to a single hole of increasing size**

The next example shows how FilledAreaRatio responds to a single hole of increasing size, illustrated in . In (a), the size of the hole is relatively small, resulting in *FilledAreaRatio* ~ 0.97. For the square with the medium-sized hole in (b), we obtain *FilledAreaRatio* = 0.75. Finally, for the square with the large hole in (c), we have *FilledAreaRatio* ~ 0.31. Note that the measure *FilledAreaRatio* had a large rate of change across the different hole sizes, producing a low value for the region in (c). This was not the case for the previously presented measure *SmplConnctvty4Adjacency*, whose value for (c) was approximately 0.55. Thus, in this example, *FilledAreaRatio* is more in accordance to our intuitive perception of the size of the hole.

5.2.3 **Test case: sensitivity to the presence of multiple holes**

The following example illustrates how *FilledAreaRatio* behaves in the presence of multiple holes. presents two squares with holes. In (a), the square contains 4 small holes, each of 1 pixel in size. This region results in *FilledAreaRatio* = 0.84. (b) presents a square of the same size, but with a single hole of 4 pixels in size. Since *FilledAreaRatio* measures only the relative area attributed to holes, again we have *FilledAreaRatio* = 0.84. While the *FilledAreaRatio* does not change between the two regions, our previously presented index *SmplConnctvty4Adjacency* is sensitive to the variation in the number of holes, as presented in Section 0, which is more in accordance to our intuitive perception of the polygon's "hole-ness" (internal emptiness).

5.2.4 **Test case: sensitivity to a noisy external boundary**

In this example, we show that *FilledAreaRatio* can be relatively insensitive to noisy external boundaries. This situation is especially relevant, since it tends to happen in practice in segments from our satellite image analysis. In , the two regions have the same area and present the same hole. However, the external boundary of the region in (b) is noisier (and thus longer) than the one of the region in (b). As shown in Section 5.1.5, this noisy boundary affects the measure *SmplConnctvty4Adjacency* to some extent. However, the value of FilledAreaRatio = 0.84 is the same for both regions, in effect ignoring the noise in the external boundary.

5.2.5 **Conclusions about the second proposed shape index**

Based on Sections 5.2.1 to 5.2.4, our experimental conclusions about the original index *FilledAreaRatio* = (2) are that:

- Overall, its numerical behavior complies with the project requirements specification of Section 3.

- It is suitable to counterbalance operational limitations of the *SmplConnctvty4Adjacency* = (1) formulation and vice versa. In fact:
    - *SmplConnctvty4Adjacency* appears less sensitive to changes of input holes in terms of size than to changes of input holes in terms of their overall number (cardinality). Index *FilledAreaRatio* = (2) behaves the opposite, it is more sensitive to changes of input holes in terms of size than to changes of input holes in terms of their overall cardinality.
    - Whereas *SmplConnctvty4Adjacency* is somehow sensitive to noisy (irregular) external boundaries when the inner region boundary facing holes remains unaffected by noise (which appears as an unlucky circumstance, but seems to occur in practice), *FilledAreaRatio* is insensitive to this noise effect.

5.3 Combined simple-connectivity index

Based on conclusions proposed in Section 5.2.4, we define our final simple-connectivity index, *CombinedSmplConnctvty*, as

$$CombinedSmplConnctvty = \mathrm{Min}\{SmplConnctvty4Adjacency, FilledAreaRatio\}. \tag{3}$$

such that *CombinedSmplConnctvty* $\in$ [0, 1], where *CombinedSmplConnctvty* equals 1 if there is no hole in the polygon.

Hence, *CombinedSmplConnctvty* = (3) is severe as (as low as) the most severe (lowest) between the two simple-connectivity indexes *SmplConnctvty4Adjacency* = (1) and *FilledAreaRatio* = (2).

For example, shows that *FilledAreaRatio* is more sensitive to the size of large holes. For the region in (c), which presents a single very large hole, we obtain *FilledAreaRatio* = 0.31 and *SmplConnctvty4Adjacency* ~ 0.55, therefore *CombinedSmplConnctvty = FilledAreaRatio = 0.31*. On the other hand, the example in shows that *SmplConnctvty4Adjacency* is more sensitive to the presence of multiple holes when the total area of the holes in the region is the same. For example, in the two regions shown in Figure 6(a) and Figure 6(b), *FilledAreaRatio* = 0.83 is the same for both regions (since the total area of the holes does not change), while *SmplConnctvty4Adjacency* ~ 0.86 in Figure 6(a) and *SmplConnctvty4Adjacency* ~ 0.56 in Figure 6(a). Therefore, *CombinedSmplConnctvty = FilledAreaRatio = 0.83* in Figure 6(a) and *CombinedSmplConnctvty = SmplConnctvty4Adjacency* = 0.56 in Figure 6(b).

Appendix A. Review of algorithms for tracing the external region boundary and the external outer boundary

In this Appendix, we review the algorithm for tracing the external region boundary, along with its variant that is used to trace the external outer boundary. The definitions of these different types of boundaries can be found in Section 5.1. We follow the presentation from Sonka *et al*. [6] (p. 129). Similar descriptions of the standard boundary tracing algorithm are presented by Pavlidis [8] (p. 143) and by Gonzalez and Woods [5] (p. 796-797). The latter authors note that the algorithm is sometimes referred to as the Moore boundary tracking algorithm, after Moore [9].

In order to avoid any confusion, in the following discussion we explicitly differentiate between the two types of boundaries we are interested in by always qualifying them as either *external* or *internal* (the latter surround holes). Sonka *et al.*'s initial description of the method is not concerned with holes, so that their algorithm only traces the external boundary. Additionally, we make use of the term *walk* (borrowed from graph theory), in order to refer to the result of the tracing algorithms. For our purposes, we define a walk as an ordered sequence of pixels, in which pixels are allowed to repeat. A *closed* walk is one whose first and last pixels coincide, as is the case for the walks we are interested in, which are traced along boundaries.

The algorithm for external region boundary tracing begins by raster scanning the image, from top left, until it finds a pixel that belongs to the region. This pixel is the first in the external region boundary walk. The algorithm proceeds to follow the boundary in counter-clockwise direction. While the boundary tracing is on-going, the latest pixel added to the walk has its neighborhood scanned, also in counter-clockwise direction. In this step, either the 4-adjaceny or 8-adjaceny neighborhood of the pixel can be scanned, each resulting in a different kind of detected boundary walk (whose pixels are respectively 4-connected and 8-connected). The neighborhood scan begins at the previous boundary pixel and proceeds until a new region pixel is found. This new pixel, which will also belong to the boundary, is then added to the walk and the process is re-iterated. The algorithm terminates after having re-encountered the initial boundary pixel and checked that the next pixel that would be visited is the second pixel already found in the walk, in which case the external boundary has been completely traced. The steps of the algorithm are such that it is robust to the possible presence of holes in the region, simply ignoring them.

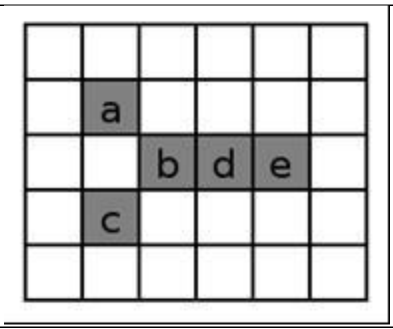

Figure 9: Assessment of the length of the external region boundary (traced in 8-connectivity) for a region whose parts are one-pixel thin. By searching the region from top left, we find that pixel **a** has the minimum column value of all pixels of the region having the minimum row value. Following the boundary in counter-clockwise direction starting from pixel **a**, the region's external walk is found to be **abcbdedba**. The length of the external boundary is thus 8. Note that pixel **b** appears three times in the walk, while pixel **d** appears twice. Pixels **c**, and **e** each appear only once. Pixel **a** also appears twice, but since it is the first and last pixel, it is only counted once towards the length of the boundary.

The external region boundary walk is defined as the sequence of pixels visited by the tracing algorithm when using 8-connectivity. As a consequence of this operational definition, the set of pixels that belong to the external region boundary walk will: (a) belong to the region, and (b) be 4-connected with outer pixels belonging to the region's "outer" world (not to be confused with the region's (inner) holes, if any).

In cases in which parts of a region are one-pixel thin, the same pixel may appear more than once in the external region walk. An example of a region presenting this behavior is shown in Figure

9.

When defining an *internal* (hole-surrounding) walk, we can also resort to the boundary tracing algorithm described by Sonka *et al*. [6] (p. 129). For each hole, we consider the closed walk along the boundary that surrounds that hole, as would be traced by the algorithm described by Sonka *et al*. In this case, the initialization of the algorithm is modified so that it starts at a region pixel that is 4-adjacent to the hole. As usual, the algorithm will then successively trace the walk along the boundary, searching around each boundary pixel's neighborhood in counter-clockwise direction. It is important to note that, differently from when the external part of the boundary is traced, applying the algorithm around a hole will result in a walk that is ordered in clockwise fashion around the hole, as noted by Pavlidis [8] (p. 143).

Thus, we define an internal region boundary walk as the sequence of pixels resulting from the tracing algorithm applied around a given hole, using 8-connectivity. As a consequence of this operational definition, the set of pixels that belong to the internal walks will: (a) belong to the region (region pixels), and (b) be 4-connected with outer pixels belonging to the region's holes.

In Section 5.1, we briefly mentioned that it is possible to compute the 4-adjacency cross-aura measure of the external boundary by using an external outer boundary tracing method. This method is referred to by Sonka *et al.* [6] (p. 129) as "outer boundary tracing" and consists of the following. The external region boundary is traced in 4-connectivity as described previously. During the tracing process, when we are analyzing a boundary pixel's neighborhood, the set of non-region pixels that are tested form the external *outer* boundary. Each time that an outer boundary pixel is tested during the algorithm, a count should be incremented towards the 4-adjacency cross-aura measure of the external boundary. Following this procedure, after having completed the tracing, the count will result in the desired cross-aura measure. As an example, consider the region in Figure 1. Each time an outer pixel is visited by the boundary tracing algorithm, it is marked with an 'o'. Therefore, the 4-adjacency cross-aura measure of the external boundary is equal to the number of 'o's in the image. Note that this algorithm does not require that the holes be filled in, and in effect holes are simply ignored.

Appendix B. Simple-connectivity index based on boundary lengths

A simple-connectivity index alternative to the ones presented in Section 5 index can be defined using the lengths of closed walks along the external and the internal boundary. At the end of this Appendix we show examples illustrating the index's properties and explain why we opted not to use it in our final simple-connectivity index, which was presented in Section 5.3.

Two types of closed walks are considered: *external* (boundary) walks, which are along the external boundary; and *internal* (boundary) walks, which are along the internal boundary that the region forms with its (internal) holes, refer to nomenclature in Section 2. The term *walk* is borrowed from graph theory. For our purposes, we define a walk as an ordered sequence of 4-connected or 8-connected pixels, in which pixels are allowed to repeat. A *closed* walk is one whose first and last pixels coincide. Throughout this appendix, all walks considered are defined along some part of a boundary, are closed, and are 8-connected. Therefore, when referring to a walk, the terms *boundary, closed,* and *8-connected,* will be considered implicit and omitted. To recall, throughout most of this document, we are considering 8-adjacency region boundaries, i.e.

boundaries whose pixels are 8-connected and belong to the region being analyzed (see definitions in Section 5.1). All walks are defined over this type of boundary.

The simple-connectivity measure based on boundary lengths is defined using the relation between the length of a region's *external* boundary and the length of its *internal* (hole-surrounding) boundaries. The length of an external or internal boundary is defined simply as the number of pixels in its boundary walk (such that the first and last pixel, which coincide, are only counted once). This new segment-specific simple-connectivity index, denoted as SmplConnctvty, is defined according to the following equation.

SmplConnctvty = (length of the (sole, unique) external boundary) /
(total length of the (one or more) boundaries), (4)

where

Total length of the (one or more) boundaries =
Length of the (sole, unique) *external* boundary +
Sum_{**h** $\in$ set of the region holes} length of the (*internal)* boundary around hole **h**. (5)

Note that this simple-connectivity index is also in the [0, 1] range. The measure *SmplConnctvty* will be equal to 1 if and only if the region has no holes, i.e., it is simply connected.

The walk along the *external* boundary, used to compute its length, is traced along only the part of the boundary that the region forms with its "external (outer) world," i.e., the external boundary walk does not consider the part of the boundary that the region forms with its (inner) holes. In practice, this walk is defined operationally as the closed walk traced along the external boundary of a segment by the 8-adjacency version of the "inner boundary tracing" algorithm described by Sonka *et al.* [6] (p. 129). Similar descriptions of the same algorithm are presented by Pavlidis [8] (p. 143) and by Gonzalez and Woods [5] (p. 796-797). The latter authors note that the algorithm is sometimes referred to as the Moore boundary tracking algorithm, after Moore [9]. It is noteworthy the so-called external boundary walk, traced by the "inner boundary tracing" algorithm described by Sonka *et al.* [6] (p. 129) is alternative to an external *outer* boundary walk, which can be traced by the "outer boundary tracing" algorithm, also described by Sonka *et al.* [6] (p. 131). The pixels from the former all belong to the region, while the pixels from the latter face the region from the "outer" world, i.e., they do not belong to the region at hand. The algorithms for tracing these boundaries are briefly reviewed in Appendix A.

Finally, we define the total length of the (one or more) boundaries as the length of the *external* boundary plus the length of each of the region's *internal* boundary walks that surround holes, if any exists. Each eventual hole of the region faces a subset of the region's boundary. A closed walk surrounding the hole can be traced along this boundary subset, whose length is then included in the total length estimate (refer to Figure 2). In Appendix C we present an alternative original algorithm for computing the total length, which does not require the walks to be explicitly traced.

We now illustrate two important properties of the *SmplConncivty* measure. In general, the measure behaves similarly to *SmplConnctvty4Adjacency*, as it is also based on the boundaries of the region at hand. However, the two properties below illustrate important differences between the two indexes.

B.1 Test case: scale invariance

A problem with the *SmplConncivty* index is that it is not completely scale invariant (a) shows a region containing a hole, with *SmplConncivty* = 8/12 ~ 0.67. We then scale the region to twice its size, as shown in (b). The resulting region has *SmplConncivty* = 20/28 ~ 0.71. While the value of *SmplConncivty* has changed, the corresponding values of *SmplConnctvty4Adjacency* and *FilledAreaRatio* remain constant across the two regions. For this reason, we decide not to use *SmplConncivty* in our final measure, instead preferring *SmplConnctvty4Adjacency* and *FilledAreaRatio*.

SmplConnctvty, though not completely scale invariant, is approximately scale invariant. It is not difficult to see that for images with higher resolution, the error introduced by *SmplConnctvty* due to scale changes will be very small. In such situations, it might be advantageous to use *SmplConnctvty* in the place of *SmplConnctvty4Adjacency*, as it is more precise in measuring the lengths of diagonal boundaries, as illustrated in the test case below.

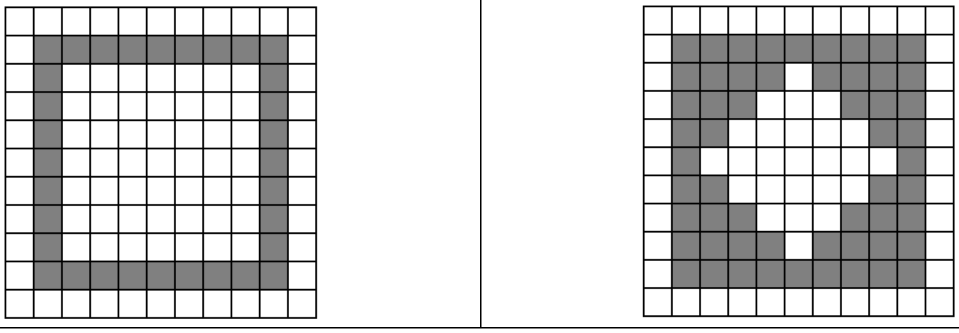

Figure 10: Example comparing regions with holes: (a) a region with a square hole whose sides are aligned with the image grid; (b) a region with a square hole whose sides are diagonal relative to the image grid.

B.2 Test case: diagonal boundaries

The following example illustrates how *SmplConnctvty* and *SmplConnctvty4Adjacency* behave differently with respect to boundaries that are diagonal relative to the image axes. Figure 10 presents two regions, each with a different hole. In Figure 10(a) the size of the hole is larger than in Figure 10(b), at the same time the two holes are oriented differently relative to the image axes. The 4-adjacency cross-aura measures of the external and total boundaries are the same for both regions, so that for both we obtain *SmplConnctvty4Adjacency* = 36 / 64 ~ 0.56. However, the difference in the lengths of the boundaries of the holes is captured by *SmplConnctvty*. For Figure 10(a) we obtain *SmplConnctvty* = 32 / 60 ~ 0.53, while for Figure 10(b) we have *SmplConnctvty* = 32 / 48 ~ 0.67.

Appendix C. Original algorithm for computing the total length of the (region) boundaries = length of the external boundary + length of the internal boundaries.

The total length of the boundaries considers, besides the external boundary, the internal boundaries that the region forms with its holes, refer to nomenclature in Section and see Figure 2. In order to compute the total length of the external plus internal boundaries, one could explicitly trace all of the walks along the region boundaries: one external, and the others internal, surrounding each of the region's holes. The lengths of all of the resulting walks would then be added in order to obtain the desired total length. An algorithm for tracing all of these walks is presented by Pavlidis [8] (p.146). The algorithm involves maintaining a queue of walk pixels (or, equivalently, the chain codes describing the on-going walks) as well as an auxiliary image (or array) to indicate the number of times each pixel has been added to a walk. After tracing the external walk, the algorithm uses pixels from the queue as starting points to scan the interior of the region, and a careful set of verifications is performed in order to detect unvisited holes. Once a new unvisited hole is found, it will have its internal hole-surrounding walk traced, which is then added to the queue, so that the scanning process may continue. Here we will present an alternative method to compute the total length of the boundaries, which does not require them to be explicitly traced, nor any additional data structure to be allocated. For our particular purposes, finding the walks themselves is not required, since we only need their lengths in order to compute the simple-connectivity measure. Besides being used for computing the *SmplConnctvty* index, this algorithm can be used in other applications in which the total length of the boundaries is desired, e.g. when computing the compactness of a region with holes.

By a single scan of the image pixels, the total length of the boundaries of a target region can be directly computed. This observation leads to our proposed algorithm. For each pixel, its contribution towards the total length of the boundaries is counted by analyzing only the pixels in its 8-adjacency neighborhood. Consider the 3×3 local window that contains a boundary pixel at its center, surrounded by its 8-adjacency neighborhood. In addition, consider the set of walks that would be traced by a boundary tracing method along the external boundary as well as along the internal parts of the boundary, which surrounds holes. Ignoring the case in which the region consists of a single isolated pixel, let us consider the sets of contiguous non-region pixels within the aforementioned 3×3 window. Each set of contiguous non-region pixels will be delimited by pixels belonging to the region, which also belong to a walk (see Figure 1(a)). If a non-region set contains a 4-adjacency neighbor of the central pixel, then the walk will necessarily pass through the central pixel, in which case we should increment the count for that pixel towards the total length. If, instead, the non-region segment consists of a single pixel which is not a 4-adjacency neighbor of the central pixel, then it should not be added towards the count, since the non-region's surrounding walk will not contain the central pixel (see Figure 1(b) and Figure 1(c)).

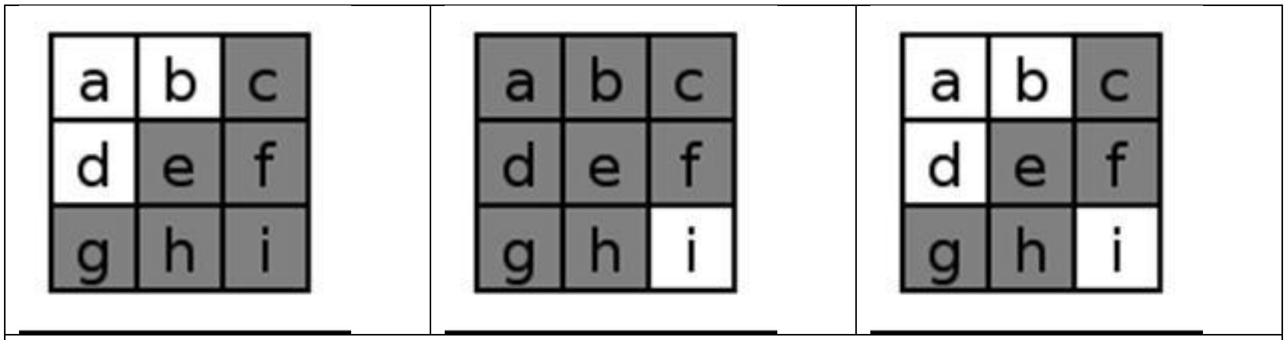

Figure 11: Examples of boundary pixels, along with their respective 3 by 3 neighborhood windows. In (a), the walk that passes through the boundary pixel **e** is **ceg**, while the set of contiguous non-region pixels of interest, which is delimited by the walk pixels, is **bad**. In (b), the central pixel is not a boundary pixel, since it has no non-region 4-neighbors. In this case, pixels **f** and **h** will form part of the walk that surrounds **i**. In (c), the central pixel is a boundary pixel due to its non-region neighboring pixels **bad**. However, as in (b), the walk that surrounds pixel **i** does not pass through **e.**

C.1 Original pseudo-code

Observations proposed in the introduction to Appendix C lead to a simple algorithm for determining the total length of the boundaries, which we present below in pseudo-code. Note that, since each pixel's neighborhood is analyzed independently, the algorithm can be easily implemented in parallel.

```
function computeTotalLengthOfBoundaries(image i, target region r)
totalLength ← 0
for each pixel p in image i
    if isRegionPixel(i, r, p) then
        unaccounted4AdjcncyNeighbor ← FALSE
        for each direction d from 0 to 7
            currentNeighbor ← neighbor(i, p, d)
            nextNeighbor ← neighbor(i, p, d + 1 (mod 8))
            if NOT isRegionPixel(i r, currentNeighbor) then
                if d ∈ {0, 2, 4, 6} then
                    unaccounted4AdjcncyNeighbor ← TRUE
                end if
                if isRegionPixel(i, r, nextNeighbor) AND unnacounted4AdjcncyNeighbor then
                    totalLength ← totalLength + 1
                    unaccounted4AdjcncyNeighbor ← FALSE
                end if
            end if
        end for
    end if
end for
return totalLength
```

In the pseudo-code above, the neighboring directions are encoded as integers from 0 through 7, using the scheme shown in Figure 112. According to this scheme a 4-adjacency neighborhood of the central pixel consists of direction numbers that are even, i.e., direction **d** ∈ {0,2,4,6}. The function **neighbor(image i, pixel p, direction d)** returns the position of the neighbor of pixel **p** in the direction **d**, while the function **isRegionPixel(image i, region r, pixel p)** returns TRUE if pixel **p** belongs to region **r**, and FALSE otherwise.

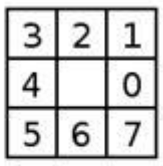

Figure 112: Neighborhood encoding scheme used in the pseudo-code presented here. The integer code denoting a neighbor grows in the counter-clockwise direction.

It is important to point out that, in order to simplify the explanation, the pseudo-code above does not handle the case in which the region is a single isolated pixel. This could be handled either by a pre-processing stage, or by modifying the above algorithm accordingly. A flag could be added, indicating whether there was ever a region pixel found that did not present any transition from non-region pixels to region pixels. This would mean that the region pixel has no neighbors that belong to the region, so that the region necessarily consists of that single isolated pixel (to recall, we assume all regions are 8-connected).

C.2 Correctness of the proposed algorithm

Here, we explain more carefully that the function **computeTotalLengthOfBoundaries()** above does in fact compute the total length of the boundaries. At the end of the Section, we provide results of the algorithm on a couple of example regions for illustration.

We would like to show that the value of variable **totalLength** provided as output at the end of the function will be equal to the sum of the length of the external walk with the lengths of internal (hole-surrounding) walks, if any. In order to do this, the main idea is to consider pairs of consecutive pixels along the boundary walks. Each pixel in a walk, in combination with the pixel that immediately succeeds it, defines a pair. For the purposes of this analysis, we will consider that the last pixel in a walk (which is equal to the first) does not form a pair with its successor (we will, however, consider the pair that it forms with its predecessor). If we were to consider the pair the last pixel forms with its successor, that pair would have to be a repetition of the pair already formed by the first and second pixels of the walk, and we do not wish to count that pair twice. With this, the number of pairs from a given walk will be equal to the number of pixels in the walk, minus one (to account for the last pixels), which is exactly its length. Additionally, it is important to note that each pair appears in only a single walk, and only once in that walk. Therefore, the total number of distinct pairs in the boundary walks is equal to the total lengths of the walks, which is what we wish to compute. At the same time, function **computeTotalLengthOfBoundaries()** also defines a set of pairs. For each pixel belonging to the region being analyzed, the algorithm considers its 3×3 neighboring window. **totalLength** is incremented each time a set of consecutive non-region pixels in the window contains a 4-adjacency neighbor of the central pixel. This set of consecutive non-region pixels indirectly determines a pair, formed respectively by:

1. the central pixel;
2. the neighbor of the central pixel that belongs to the region and immediately *follows* the set of non-region pixels, when following the central pixel's neighborhood in counter-clockwise direction.

The second pixel in the pair above is exactly the pixel that is found to belong to the region when the verification performed on line 13 of function **computeTotalLengthOfBoundaries()** is successful.

We show that the set of pairs that forms the boundary walks is the same as the set of pairs found by function **computeTotalLengthOfBoundaries()**. For this purpose, we show the following two more simple properties. The first is that every pair in a walk will be found and counted by function **computeTotalLengthOfBoundaries()**. The second is that each pair found by function **computeTotalLengthOfBoundaries()** will also belong to a walk. These are explained below.

1. *Every pair in a walk will be counted by function* **computeTotalLengthOfBoundaries()**. Consider any region that is not a single isolated pixel. (The situation in which the region is a single pixel is not handled by function **computeTotalLengthOfBoundaries()**, though it is trivial.) Now, consider any pixel in any walk of the region, namely, either the external walk or an internal hole-surrounding walk. The current and next pixels along the walk form the pair of interest. It is important to note that this specific pair will be found within only a single walk and will be present only once in that walk: since the pair determines the state of the boundary tracing algorithm, if the same pair were to be traced twice, the algorithm would be repeating itself, which, assuming the algorithm is correct, does not happen. The next pixel along the walk is an 8-adjacency neighbor of the current pixel, and therefore is also contained in the 3×3 window centered on that pixel. The boundary tracing algorithm is such that if we follow the 8-adjacency neighborhood of the current walk pixel in counter-clockwise direction, starting from the previous walk pixel, we will reach the next walk pixel after meeting only non-region pixels in between. These non-region pixels contain one or more pixels that are 4-adjacent to the central walk pixel. (If the non-region pixel set consisted of a single non-region pixel that was *not* 4-adjacent to the central pixel, then the central pixel would not have been included in the walk in the first place, as indicated in Figure 1). Thus, this set of non-region pixels will be detected by our proposed function **computeTotalLengthOfBoundaries()**, along with its corresponding pair.
2. *Every pair counted by function* **computeTotalLengthOfBoundaries()** *will also belong to a walk.* Each **totalLength** count increment has an associated pair, as defined previously in this Section. To recall, the pair is defined by the current pixel being analyzed, along with the pixel that immediately follows its associated contiguous set of non-region pixels, when performing the analysis in the counter-clockwise direction. The non-region pixels are either external to the region or belong to a hole, both cases for which a walk will be traced. This set of non-region pixels also necessarily contains at least one pixel that is 4-adjacent to the central pixel, due to the explicit check in line 10 of function **computeTotalLengthOfBoundaries()**. Thus, the central pixel belongs to a boundary, so that it will be part of a walk that surrounds the same set of non-region pixels that were

detected by the function (based on the assumption that the tracing algorithm is correct). The boundary tracing algorithm searches for new pixels in the counter-clockwise direction, so that the ordering of pixels in the walk will be the same as the ordering in the pair defined by the scanning algorithm, and thus both pairs coincide.

Figure 13 and Figure 134 illustrate the result of function **computeTotalLengthOfBoundaries()** on a couple of example regions. For each region, we indicate the walks of interest, as well as the counts associated to each pixel, as computed by function **computeTotalLengthOfBoundaries()**.

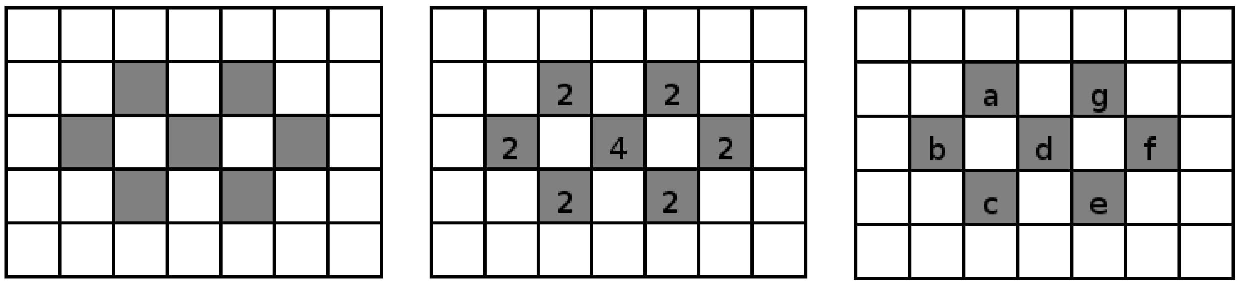

Figure 123: First illustrative example of the **computeTotalLengthOfBoundaries()** algorithm. The original region is shown in (a), indicated by the set of gray pixels. Note that the region contains two holes. The counts associated to each pixel by function **computeTotalLengthOfBoundaries()** are shown in (b). By adding these counts together we arrive at a value of 16 for **totalLength**. The pixels of the region are labeled in (c). According to this labeling, we have the following closed walks along the region’s boundaries. The external boundary walk is **abcdefgda**, whose length is 8. The internal hole-surrounding walk around the left hole is **dcbad**, while the walk around the right hole is **fedgf**. Both these walks have length 4, resulting in a total length of boundary walks equal to 16. The boundary tracing algorithm is such that the external walk is oriented in counter-clockwise fashion, while the internal walks are oriented clockwise. Note that, in this manner, no pairs along the walks are ever repeated, even though individual pixels may appear more than once, either in the same or in different walks.

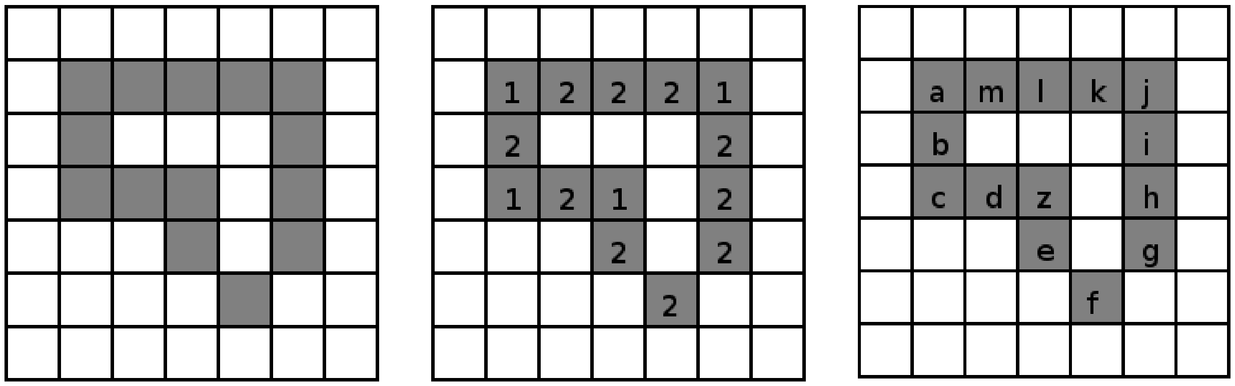

Figure 134: Second illustrative example of the **computeTotalLengthOfBoundaries()** algorithm. The original region is shown in (a), indicated by the set of gray pixels. The counts associated to each pixel by function **computeTotalLengthOfBoundaries()** are shown in (b). By adding these counts together we arrive at a value of 24 for **totalLength**. The pixels of the region are labeled in (c). According to this labeling, we have the following closed walks along the region’s boundaries. The external boundary walk is **abcdefghijklma**, whose length is 13. In turn, the single internal hole-surrounding walk is **ihgfezdbmlki**, whose length is 11, resulting in a total length of boundary walks equal to 24. The boundary tracing algorithm is such that the external walk is oriented in counter-clockwise fashion, while the internal walk is oriented clockwise. Note that, in this manner, no pairs along the walks are ever repeated, even though individual pixels may appear more than once, either in the same or in different walks.

References

[1] B. M. Mehtre, M. S. Kankanhalli and W. F. Lee, "Shape measures for content based image retrieval: a comparison," Information Processing & Management, vol. 33, no. 3, pp. 319-337, 1997.

[2] M. Bertamini, "Who owns the contour of a visual hole?," Perception, vol. 35, pp. 883-894, 2006.

[3] E. A. Wentz, "S Shape Definition for Geographic Applications Based on Edge, Elongation, and Perforation," Geographical Analysis, vol. 32, no. 2, pp. 95-112, April 2000.

[4] A. Soffer and H. Samet, "Negative shape features for image databases consisting of geographic symbols," in Proc. 3rd International Workshop on Visual Form, 1997.

[5] R. C. Gonzalez and R. E. Woods, Digital Image Processomg, 3rd ed., Upper Saddle River, NJ: Pearson Prentice Hall, 2008.

[6] M. Sonka, V. Hlavac and R. Boyle, Image Processing, Analysis, and Machine Vision, Chapman and Hall Computing, 1993.

[7] Open Source Computer Vision Library (OpenCV). http://opencv.org/. Retrieved on March, 2013.

[8] T. Pavlidis, Algorithms for Graphics and Image Processing, Rockville, MD: Computer Science Press, 1982.

[9] G. A. Moore, "Automatic Scanning and Computer Processes for the Quantitative Analysis of Micrographs and Equivalent Subjects," in Pictorial Pattern Recognition, Washington, D.C., 1968.