

Topology Based Method of Segmentation of Gray Scale Images

Peter Saveliev
Marshall University
One John Marshall Drive, Huntington, WV 25755, USA
saveliev@marshall.edu

Abstract

The paper provides a method of image segmentation of binary and gray scale images. For binary images, the method captures not only connected components but also the holes. For gray scale images, there are two kinds of “connected components” – dark regions surrounded by lighter areas or light regions surrounded by darker areas.

1. Introduction

We address the issue of image segmentation of binary and gray scale images. The scientific foundation of the proposed method is topology, the science of continuity and connectedness [7].

The topological analysis is intended to reveal the most basic things about the image: how many connected components are present, which ones have holes and how many. It is also a part of the analysis to capture these topological features. The next stage is geometric: measuring them and finding their locations. With this data, it may be possible to understand the content of the image.

We will use the tools that are standard in the discipline. The first tool is cell decomposition: the image is represented as a combination of pixels as well as edges and vertices. The second tool is cycles: both the connected components and the holes are captured by circular sequences of edges.

The challenge is that these topological tools have not been commonly applied in the analysis of gray scale images. For example, while in binary images the meaning of connected components is clear, for gray scale images, a “connected component” should be a dark region surrounded by a lighter area. In other words, it is a sublevel set of the gray level function. To deal with the multitude of sublevel sets we record them in a graph. This graph captures the inclusion hierarchy among the sublevel sets. A similar data structure is created for the holes. Combined these two graphs represent the topology of the image.

The algorithm has been tested and shown successful in segmenting a variety of images.

2. Related work

Image segmentation is the extraction of objects from an image [3].

A common technique finds the boundaries of the objects, called edge search. In the case of a gray scale image, the edges are found by locating the areas where the rate of change from black to white is high. Examples are contour trees [8], active contours [3], the digital Morse theory [11], and many others. This technique, however, fails when the boundaries of objects are not well defined.

Another approach to capturing objects in gray scale images is to produce a tree representation of the image [6], [17], [20]. In [6], the tree is built by cutting (simultaneously) minima and maxima of the gray scale function - slice by slice. The result is a hierarchy of objects that is recorded as a tree called “scale tree”. Their example is below.

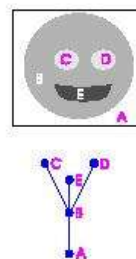


Figure 1. An image and its scale tree.

Homology theory provides the algebraic description of the topology. There have been a few attempts to address homology computation in imaging in the academic literature and the treatment tends to be cursory [16], [18]. More recent publications [24], [15] provide a number of useful algorithms for homology but their applicability in image analysis is not addressed. In [12] homology is used for image matching. In the context of digital topology this subject is addressed in [14], [19].

There are several software packages designed to compute homology. CHomP, the Computational

Homology Project, [10] is written in C++ and works with a command prompt. PLEX [21] is a set of routines for MATLAB that computes the Betti numbers. Alpha Shapes [1], [5] computes Betti numbers of simplicial complexes representing point clouds. CGAL, the Computational Geometry Algorithms Library, [9] and Simplicial Homology for GAP [23] are collections of C++ code for computation of simplicial homology. From this list, only CHomP can compute the homology of cubical complexes and, therefore, is applicable to digital images. None of these methods and programs is applicable to gray scale images.

The present research was inspired primarily by the notion of “persistent homology” [13] and grew from the desire to apply this concept to digital image analysis.

3. The topology of a binary image



Figure 2. A binary image and its negative.

Consider the first image in Figure 2. We would like the computer to see what we see – black “objects” on white background. And we want these objects captured – pixel-wise – so that we can deal with them as separate entities. Then we will have these objects measured and their locations found.

This problem is commonly known as “connected component labeling” and has many different solutions. The approach we suggest below and the algorithm we develop are different from those. The output is of course the same.

Some of these components have topological features of their own, the “holes”. The holes will also be counted, captured, measured etc.

We think of black objects as connected components and white objects as holes in dark objects. However, in the second image in Figure 2 we see white objects on black background. The approach is also feasible.

To stay consistent, we have to choose one of these two options and we choose the former. *Binary images are analyzed as if they have black objects on white background.* As a result, the white objects that touch the border are not counted.

4. Cell decomposition of binary images

A binary image is a combination of black and white pixels. Each pixel is given by its location in the image, so it is just a pair of numbers. It is then natural to represent an object simply as a list of pairs of numbers. We will take a different approach.

We will think of a pixel as a square, or a tile. Next, the boundary of a pixel is a combination of its four edges. Since an edge is shared by two adjacent pixels, keeping the list of these edges is a way to record how pixels are attached to each other. Meanwhile, keeping the list of vertices is a way to record how edges are attached to each other. This is called *cell decomposition*. It is a standard in algebraic topology.

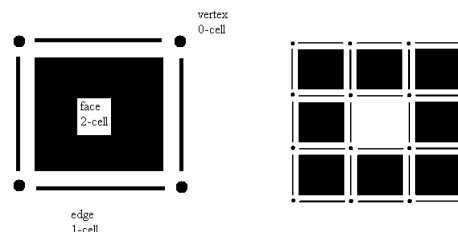


Figure 3. Left: Cell decomposition of the pixel. The edges and vertices may be shared with adjacent pixels. Right: Cell decomposition of 8 pixels arranged in a square.

The first advantage of this approach is its generality. The pixels are attached to each other along the edges they share. This it is easily applicable to all dimensions:

- a vertex is a 0-cell,
- two adjacent edges are 1-cells and they share a vertex, a 0-cell,
- two adjacent pixels are 2-cells and they share an edge, a 1-cell,
- two adjacent voxels are 3-cells and they share a face, a 2-cell,
- etc.

The second advantage is that it allows us to treat objects and holes in a uniform fashion. The result is an algorithm that simultaneously captures both.

Another advantage is related to the fact that we want our algorithm to be incremental – adding one pixel at a time. Cell decomposition gives us an incremental algorithm that is both simple and flexible. As a result it can be extended to gray scale images, and later to color images, video, etc.

With cell decomposition, when we need to add a pixel to the image, we add its vertices first, then the edges, and finally the pixel itself. This makes the algorithm simpler than one that adds a whole pixel at once. Consider the fact that the new pixel is adjacent to 8 other pixels (the 8-connectedness) and these 8 pixels may belong to up to 4 different components. The result is that the number of cases to consider is quite high. Meanwhile, if an edge is added instead, the vertices are already present. The result is that there are only 4 cases to consider: the new edge may connect two components to each other from the inside or outside, complete a hole in a component or another a hole (Figures 8-11).

The cell decomposition also makes certain concepts more straightforward. First, an object and its background don't share pixels, only edges. As a result, the area of a component plus the area of the complement is exactly the total area of the image. Second, the perimeter isn't the number of pixels in its boundary but the number of edges.

5. Using cycles to partition binary images

Both components and holes are captured by *cycles*. By cycles, we will understand circular sequences of edges. There are 0- and 1-cycles:

- A 0-cycle follows the outer boundary of a connected components (object),
- A 1-cycle follows the outer boundary of a hole.

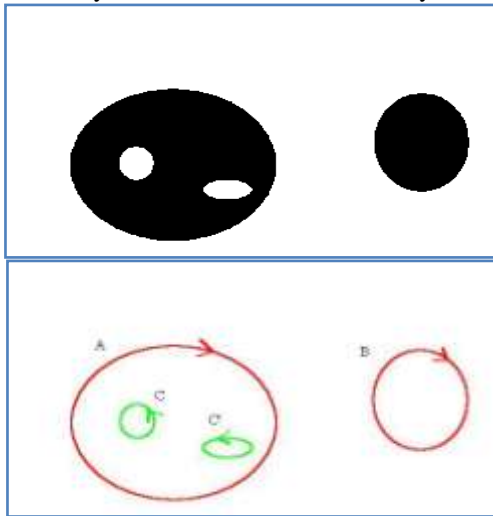


Figure 4. The objects and the holes are represented as cycles. Here A and B are 0-cycles (red), C and C' are 1-cycles (green).

This results in a natural and unambiguous representation of the regions by the curves that enclose them. A 0-cycle is traversed clockwise (on the outside of the object like a rubber band) and a 1-cycle is traversed counterclockwise. Observe that in the either case black is on the right.

Note: The name "1-cycle" is justified by the fact that this is a sequence of 1-cells. But so is a 0-cycle. This representation however is only a matter of convenience. It can just as easily be represented as a sequence of vertices or 0-cells.

Of course, 1-cycles of the picture are 0-cycles of its negative and vice versa, except for ones that touch the border of the image.

Given an image, any cycle can be traversed by taking left turns from the initial edge in such a way that black pixels are always on the right - until this edge is reached again.

The result of this topological analysis is a partition of the binary image. The partition is a collection of non-overlapping regions, connected sets of black pixels and

connected sets of white pixels, that covers the whole image. The partition is achieved by finding boundaries of these regions as 0- and 1-cycles. These cycles are closed curves made of vertical and horizontal edges of pixels.

6. The graph representation of the topology of a binary image

The algorithm is incremental. The cycles are constructed as pixels, one by one, are added to the image. Since every pixel also contains edges and vertices, the process of adding a pixel (cell) starts with adding its vertices and then its edges, unless those are already present as parts of other pixels. These vertices, edges, and pixels are marked as "current" (see the next section).

As the new pixels are being added, components merge, holes split, etc. The topology of the image changes. This process is captured by cycles as they appear and disappear, merge and split. The information about these changes is recorded in a graph. Each node in the graph represents a cycle. The directed arcs (arrows) that connect the nodes represent the merging and the splitting of the cycles.

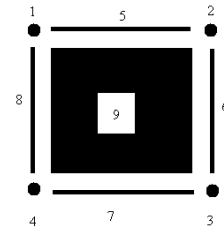


Figure 5. Adding a stand-alone pixel requires 9 steps: adding 4 vertices, 4 edges, and the cell itself.

Adding a new vertex creates a new component and a new node in the graph. Adding a new edge either connects two components or creates a hole in an existing component. Adding the square itself eliminates a 1-cycle.

Let's consider an extremely simple example - adding a single pixel to a blank image. It takes 9 steps corresponding to the 9 items to be added.

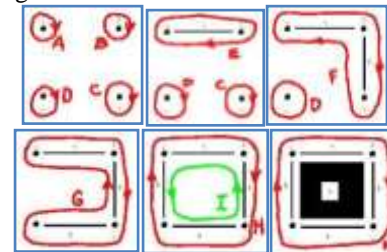


Figure 6. The stages of adding a single pixel to a blank image. 0-cycles are in red and the only 1-cycle is in green.

The construction of the image is represented by a graph. Its nodes correspond to the cycles and the arcs correspond to merging and splitting of the cycles. Each arrow is

accompanied by a number indicating which item is being added.

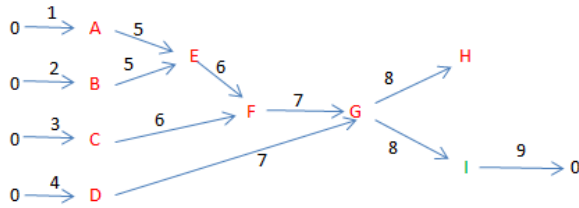


Figure 7. A graph representation of the 9 steps required to add a stand-alone pixel.

At the first stage we have 4 nodes, corresponding to the 0-cycles. Then they merge into one. This 0-cycle splits into two cycles, a 0-cycle and a 1-cycle. Finally, this 1-cycle disappears. The end result is of course just the last existing cycle, H. The graph building procedure and the graph itself will however be important in the analysis of gray scale images.

Generally, this is an acyclic directed graph. The maximum degree is 2.

Suppose N is the number of pixels in the image. Then, the total number of pixels, edges, and vertices is $O(N)$ so is the size of the graph.

7. The pseudocode of the analysis algorithm for binary images

A crucial part of the algorithm is the correspondence table T : edge \rightarrow cycle. Maintaining this table is necessary at every step.

The algorithm of image analysis is the process of adding pixels one by one while keeping track of changes in the topology.

```
//-----
ImageAnalysis with binary image I
```

```
FOR all pixels in I
  IF P is black THEN
    CALL AddPixel with P
  ENDIF
ENDIF
ENDFOR
```

```
//-----
Next is the operation of adding a pixel. It includes adding its vertices, its edges, and then the pixel itself. Adding an edge means assigning cycles to both of the directed edges – forward  $E$  and back  $-E$ . In particular, there is always a 1-cycle inside the pixel. It is “removed” as the square closes the hole.
```

```
//-----
AddPixel with pixel P
```

```
CALL AddVertex with upper right vertex of P
CALL AddVertex with upper left vertex of P
CALL AddVertex with lower right vertex of P
CALL AddVertex with lower left vertex of P
```

```
CALL AddEdge with lower edge of P
CALL AddEdge with right edge of P
```

```
CALL AddEdge with upper edge of P
CALL AddEdge with left edge of P
```

```
E = lower edge of P directed counterclockwise
CALL RemoveCycle with 1-cycle A = T(E)
//-----
```

Adding a vertex is trivial. It creates one new 0-cycle represented by a node that isn't connected to anything yet. But first you verify that the vertex isn't already present.

```
//-----
AddVertex with vertex V
```

```
IF V is present THEN
  RETURN
ENDIF
Mark V as present
Call CreateCycle with V RETURNING 0-cycle A
//-----
```

Adding an edge is the most complex step. There are three cases illustrated in Figures 8-10. Which case is determined based on the correspondence table T .

```
//-----
AddEdge with edge E
```

```
IF T(E) != NULL or T(-E) != NULL THEN
  RETURN
ENDIF
```

```
CALL NextEdge with E RETURNING edge E1
A = T(E1)
CALL NextEdge with -E RETURNING edge E2
B = T(E2)
```

```
IF A == B THEN
  CALL SplitCycle with E1, E2, and A
ELSE
  CALL MergeCycles with E1 and A, B
ENDIF
//-----
```

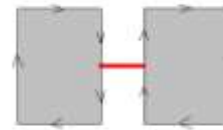


Figure 8. Case (a): the new edge connects two different 0-cycles.

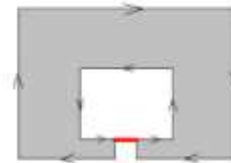


Figure 9. Case (b): the new edge connects a 0-cycle to itself.

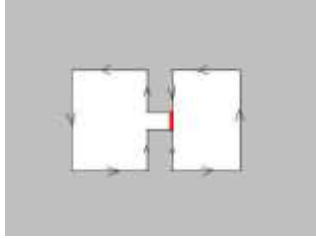


Figure 10. Case (c): the new edge connects a 1-cycle to itself.

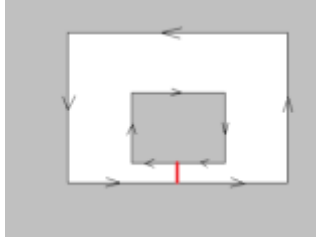


Figure 11. Case (d): the new edge connects a 1-cycle and a 0-cycle.

A 0-cycle can merge with either 0- or 1-cycle.

```
//-----
MergeCycles with cycles A, B and edge E

CALL CreateCycle with E RETURNING 0-cycle C
CALL MarkEdges with E and C
Add arcs from A, B to C to the graph
0-Betti number --
//-----
```

Either a 0- or a 1-cycle can split.

```
//-----
SplitCycle with edges E1, E2 and cycle A

IF A is a 1-cycle THEN
  CALL CreateCycle with E1 RETURNING 0-cycle C
  CALL CreateCycle with E2 RETURNING 0-cycle D
ENDIF

IF A is a 0-cycle THEN
  CALL CreateCycle with E1 RETURNING 0-cycle C
  CALL CreateCycle with E2 RETURNING 1-cycle D
ENDIF
```

```
CALL MarkEdges with E1 and C
CALL MarkEdges with E2 and D
Add arcs from A to C, D to the graph
1-Betti number ++
//-----
```

Creating a cycle means adding a new node to the graph.

```
//-----
CreateCycle with edge E

Create node A in the graph
T(E) = A
RETURN A
//-----
```

Removing a cycle means assigning NULL to all of its edges.

```
//-----
RemoveCycle with cycle A

FOR all edges E in I
  IF T(E) == A THEN
    T(E) = NULL
  ENDIF
```

```
ENDFOR
//-----
Given an edge of a cycle, one can find the next edge of
the cycle.
//-----
NextEdge with edge E

Start points of edges E1, E2, E3, E4 = end point
of E
Direction of E1 = direction of E + 90 degrees
Direction of E2 = direction of E
Direction of E3 = direction of E - 90 degrees
Direction of E4 = - direction of E

FOR edge G = E1, E2, E3, E4
  IF T(G) != NULL
    RETURN G
  ENDIF
ENDFOR
//-----
```

Next function goes around a given cycle and assigns it to the edges.

```
//-----
MarkEdges with edge E and cycle A

CALL NextEdge with edge E RETURNING edge G
WHILE G != E
  T(G) = A
  CALL NextEdge with edge G RETURNING edge G
ENDWHILE
//-----
```

8. The topology of a gray scale image

A gray scale image is simply a table filled with numbers indicating the gray level. One can also think of it as a function of two variables (the gray level function) defined on a rectangular grid.

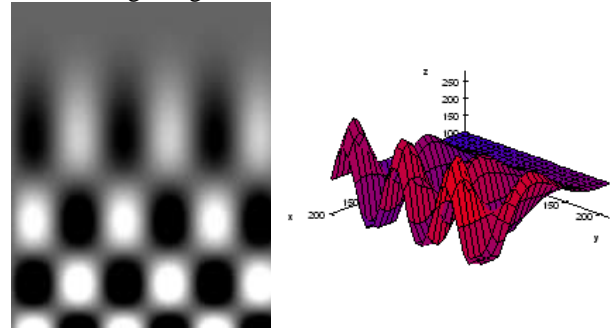


Figure 12. A gray scale image and its gray level function.

In binary images objects are either black areas surrounded by white background or white areas surrounded by black background. Similarly, our initial assumption about gray scale images will be that objects are either darker areas surrounded by lighter background or lighter areas surrounded by darker background. In other words, the *dark objects* are the connected sub-level sets and the *light objects* are the connected supra-level sets of the gray level function. The boundaries of the objects are the level curves. Since all of these sets are connected

collections of pixels they will be represented as 0- and 1-cycles.

At this stage the image analysis does not have unambiguous results.



Figure 13. The blurred version of the image in Figure 2.

It may seem clear that in the above image there are two objects and the first one has two holes. However, where the boundaries of these objects are located depends on the chosen threshold or thresholds. The choice of these thresholds will affect the topology of the image, as illustrated below.



Figure 14. Each choice of thresholds produces a different topology.

The chosen thresholds will affect the measurements of the objects will also be different. As a result some of the objects may or may not be discarded as noise.

The maximal number of dark (light) objects is equal to the number of local minima (maxima) of the gray scale function.

Our approach to dealing with this ambiguity is to collect all possible cycles and then filter them based on size, contrast, etc.

The acquisition of all possible cycles is carried out via image thresholding. Given a number T , thresholding is the process of replacing all the pixels with gray level lower than or equal to T with black leaving the rest white. This creates a binary image that we call the *frame* corresponding to T . As gray runs from 0 to 255, we have a sequence of 256 frames. Observe that as you move from frame to the next, more black pixels appear.

9. The graph representation of the topology of a gray scale image

The collection of all possible cycles is organized in a graph, the *topology graph* of the gray scale image. The collections of cycles of frames from 0 to 255 form a sequence. They are arranged in layers within the graph.

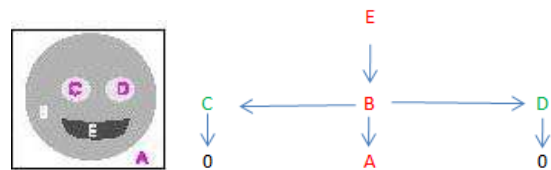


Figure 15. The topology graph for the image in Figure 1. Here the objects are: mouth E (black), head B (gray), eyes C and D, entire image A (white).

The arcs in the graph indicate inclusion. Suppose we change the threshold from 0 to 255. Then, any of the minima of the gray scale function will produce a sequence of growing dark objects, up to some threshold. At the same time, any of the maxima will produce a sequence of shrinking light objects, starting at some threshold. There are also three other kinds of “topological events” (see Figures 8-11):

1. dark objects merge,
2. a dark object form a hole (light object) inside,
3. light objects split.

One can form the topology graph from the two “inclusion trees” – the one for the dark objects and the other for the light. The latter is turned upside down and attached to the former wherever a topological event of the second type happens. The topology graph may be a tree (Figure 15), but generally it isn't. Consider, for example, the negative of the image in Figure 15, below.

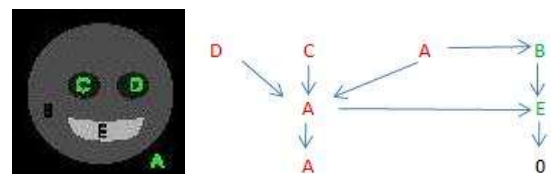


Figure 16. An image and its topology graph, which isn't a tree.

As the threshold grows, new pixels are being added and the cycles in the image appear and disappear. Components merge, holes split, etc. and the topology keeps changing. The information about these changes is recorded in the topology graph. Each node in the graph represents a cycle, and the arcs represent merging and splitting of the cycles.

As a matter of convenience instead of the topology graph we build the *augmented topology graph*, or simply the augmented graph, of the image. The growing threshold creates a partial order on the set of pixels. In that order the pixels are added to the image. The procedure of building this graph with nodes representing cycles is exactly the same as the one presented in Section 6. During this process the 256 frames will appear but between them there will be auxiliary stages. The frames will generate the *principal cycles* and the rest are the *auxiliary cycles*. The graph breaks into layers: auxiliary nodes, principal nodes of the 0th frame, auxiliary nodes, principal nodes of the 1st frame, etc. The topology graph can be extracted from the augmented graph by removing all auxiliary nodes and adding arcs between principal nodes accordingly. Unlike the topology graph, the augmented graph is dependent on the order of pixels (within frames).

Suppose the image consists of two pixels, black and gray, on white background. Then the construction of the augmented graph starts with adding the black pixel. The graph at this stage is given in Figure 55. It ends with a single principal cycle, H. The rest are auxiliary.

Next, if the gray pixel isn't adjacent to the black, the second stage looks exactly the same as the first. It contains a single frame cycle, H'. In this case the augmented graph consists of these two disconnected parts and the topology graph is simply: H, H'.

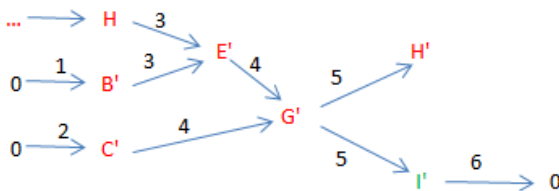


Figure 17. The augmented topology graph: adding the second (adjacent to the first) pixel to the image. This graph is attached to the graph in Figure 7.

If the gray pixel is adjacent to the black, the augmented graph is connected. There are 6 steps. Initially only H, the 0-cycle created during the first stage, is present. Then, two new vertices are added creating two new 0-cycles, B' and C'. Then two edges are added and these three cycles merge. When the third edge is added, a 1-cycle appears. Adding the square removes this cycle. Only one 0-cycle is left, H' (a two pixel dark object). The topology graph is simply $H \rightarrow H'$.

Now, H' contains H. Therefore only one of them should be counted. Which one? This is for the user to decide. He may be interested in larger objects (H') or objects with higher contrast from the surroundings (H).

Let's consider another simple example. The visual inspection of the image in Figure 17 reveals that it contains two dark objects. This information can also be acquired from the analysis of the frames of the image, as follows. The number of objects in each frame is: 1, 2, and 1. However, to avoid double-counting, we will only count an object if it does not have an ancestor in the previous frame. Then the total count of objects in the image is: $1 + 1 + 0 = 2$. This is the maximum possible number. The user may decide that one or both of these objects are noise. In the latter case there is only one object – the whole square.



Figure 18: A gray scale image and its frames.

Some of the principal cycles will have to be discarded by the user. The remaining ones will represent the simplified topology of the image. We will call them *active cycles*.

10. The pseudocode of the analysis algorithm for gray scale images

The algorithm of image analysis is the process of adding pixels one by one while keeping track of changes in the topology.

The analysis of the topology of a gray scale image consists of two stages:

1. The creation of the augmented graph, and
2. The filtering of the principal cycles and creation of the list of active cycles.

The operation of adding a pixel and all functions it calls are exactly the same as in the case of a binary image. The only difference is a list of all principal cycles is maintained.

```

//-----
ImageAnalysis with gray scale image I
FOR all thresholds T = 0, ..., 255
  FOR all pixels P in I
    IF the value of P <= T THEN
      CALL AddPixel with P
    ENDF
  ENDFOR
  Add all cycles with no descendants to the list
  of principal cycles
ENDFOR

FOR all principal cycles C
  IF C is a 0-cycle,
    Evaluate(C) == 1 and
    C does not have an ancestor D with
    Evaluate(D) == 1 THEN
    Add C to the list of active cycles
  ENDF
  IF C is a 1-cycle,

```

```

    Evaluate(C) == 1 and
    C does not have a descendent D with
    Evaluate(D) == 1 THEN
    Add C to the list of active cycles
  ENDIF
ENDFOR
//-----
  Cycles can be evaluated based on their measurements
  and other characteristics: area, integrated optical density,
  contrast, etc. The most typical is the area, as below.
//-----
  Evaluate with cycle C

  IF the area enclosed by C < MinArea THEN
    RETURN 0
  ENDIF
  RETURN 1
//-----

```

11. Implementation

The complexity of the algorithm of building the augmented graph is $O(N^2)$, number N of pixels in the image. This conclusion is based on the following facts:

- Each pixel is processed separately.
- For each of the N pixels a new object may be created and you may have to run around it to mark its edges.
- If this object is thin and fills the image, its perimeter is proportional to N .

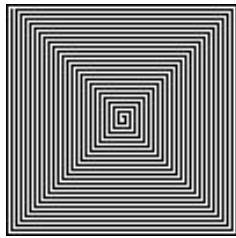


Figure 19. The complexity for this image will be $O(N^2)$.

This situation however may seem unusual, such as the image of the spiral below. Images of maps or microchips may fall into this category. Image of cells or other particles do not.

Adding a pixel creates up to 9 nodes in the augmented graph. Hence the graph size and the memory usage is $O(N)$.

After the graph is built, it is analyzed to extract objects from the image. It requires following the list of all principal cycles. This list is $O(N)$, above. For each such cycle, you may have to visit some of its ancestors or descendents. To reach them you have to visit some auxiliary nodes as well. But once again the total number of nodes is $O(N)$. Therefore the complexity of the entire algorithm is $O(N^2)$.

A variety of images was analyzed and, as the diagram indicates, the processing time appears to depend linearly on the number N of pixels in the image. It takes, roughly,

40 seconds for each million pixels in the image. The testing was done on HP Pavilion laptop with Intel Core 2 Dual CPU T7500 2.2GHz.

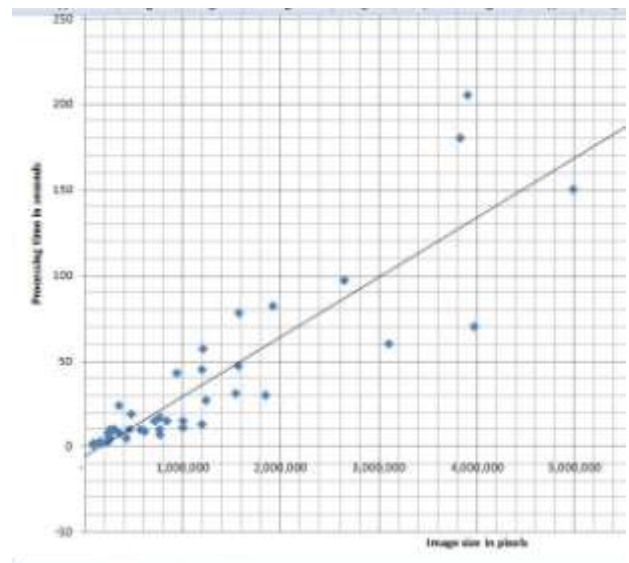


Figure 20. The processing time as a function of the image size.

12. Experiments

During processing, the areas, perimeters, and contrasts of the objects are also computed. These parameters are used filter objects. The user indicates ranges of parameters of the features he considers to be irrelevant or noise.

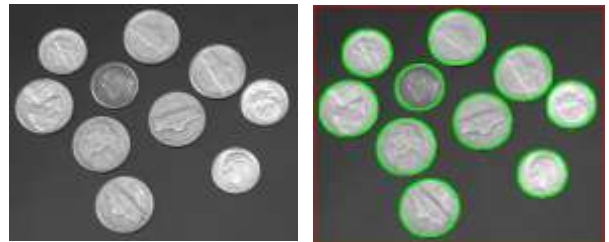


Figure 21: The coins are captured.

The settings may be chosen based on *a priori* knowledge about the image. For example the image in Figure 21 is 300×246 . To capture the coins and ignore the noise and the small features depicted on the coins, one sets the lower limit for the area at 1000 pixels.

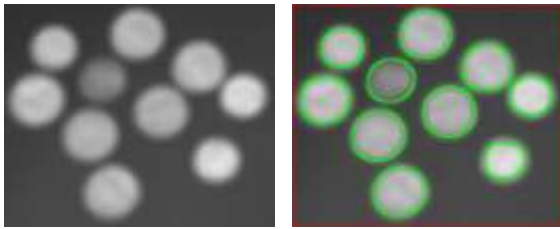


Figure 22: The coins are captured in the blurred version.

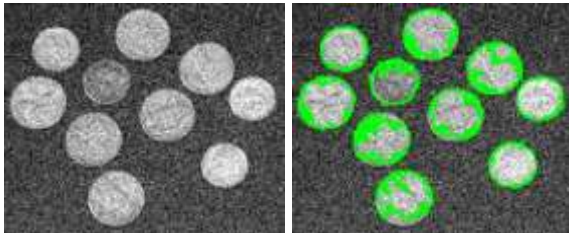


Figure 23: The coins are captured in the version with Gaussian noise.



Figure 24: The image of a fingerprint and its rotated version.

The analysis is not significantly affected by rotations. The output for the original 640×480 fingerprint in Figure 24 is 3121 dark and 1635 light objects. For the rotated version, it is 2969-1617. By limiting the analysis to objects with area above 50 pixels, the results are improved to 265-125 and 259-124, respectively.



Figure 25 An MRI of breast tissue and its version with all low contrast objects removed.

Stretching the image does not affect the count of objects. Shrinking makes objects merge. If the goal, however, is to count and analyze larger features, limited shrinking of the image does not affect the outcome. The count is also stable under noise and blurring.

The method works best with images that represent something 2-dimensional. It fails when the third dimension is essential. It is the case when the images contain:

- occluded objects,
- transparent objects,
- objects well lit on one side and dark on the other,
- X-rays.

Another limitation of the method is that clustered cells or other objects can't be separated unless there is variation of intensity.

13. Conclusions and further research

The method proposed in this paper is well grounded in mathematics and produces meaningful results for various gray scale images. Its speed and memory requirements make it practical for every day use on today's personal computers.

The main issue still to be addressed is finding better ways to filter the data in the topology graph. Another is the issue of merging objects based on similarity and proximity rather than the hierarchy of the topology graph.

A modification of the algorithm applies to color images. It is possible to threshold an RGB image so that a binary image (frame) is created for each combination of red, green, and blue. The cycles from these frames form the topology graph of the image.

Other multi-parameter images come from combined MRI and PET scans in medical imaging or from a spectrum of sensors (optical, thermal, etc.) in satellite or surveillance imaging.

The ability to remove pixels on top adding pixels turns the algorithm into a motion tracking program for binary images.

References

- [1] N. Akkiraju, H. Edelsbrunner, M. Facello, P. Fu, E. P. Mucke, and C. Varela, Alpha shapes: definition and software. *Proc. Internat. Comput. Geom. Software Workshop*, ed. N. Amenta, Rpt. GCG 80, Geometry Center, Minneapolis, Minnesota, 1995.
- [2] Z. Aktouf, G. Bertrand, and L. Perroton. A 3D-hole closing algorithm. *Lectures Notes in Computer Science*, Vol. 1176, pp 36-47. Springer Verlag, 1996.
- [3] S. Angenent, E. Pichon, and A. Tannenbaum. Mathematical Methods in Medical Image Processing. *Bulletin of the American Mathematical Society*, 43, pp. 365-396, July 2006.
- [4] C. Arcelli, G. Sanniti di Baja, and S. Svensson. Computing and analysing convex deficiencies to characterise 3D complex objects. *Image and Vision Computing*, 23(2):203-211, Feb. 2005.
- [5] *Alpha Shapes*, Biogeometry Project, Duke University, biogeometry.duke.edu/software/alphashapes/.

- [6] J. A. Bangham, J. R. Hidalgo, R. Harvey, and G. C. Cawley, The Segmentation of Images via Scale-Space Trees, *British Machine Vision Conference*, 1998.
- [7] G. Bredon, *Topology and Geometry*. Springer Verlag, 1993.
- [8] H. Carr, J. Snoeyink, and U. Axen, Computing Contour Trees in All Dimensions, *Computational Geometry*, 2003.
- [9] *CGAL*, Computational Geometry Algorithms Library, cgal.org.
- [10] *CHomP*, Computational Homology Project, Georgia Tech, www.math.gatech.edu/~chomp/.
- [11] J. Cox, D. Karron, and N. Ferdous, Digital Morse Theory for Scalar Volume Data. *Journal of Mathematical Imaging and Vision*. Vol. 18, 2003, pp. 95-117.
- [12] S. Derdar, M. Allili, and D. Ziou, Image matching using algebraic topology, *Proceedings of SPIE*, Volume 6066 Vision Geometry XIV, Longin Jan Latecki, David M. Mount, Angela Y. Wu, Editors, 60660J (Jan. 15, 2006).
- [13] H. Edelsbrunner, D. Letscher, and A. Zomorodian, Topological persistence and simplification. *Discrete Comput. Geom.* 28 (2002), pp. 511-533.
- [14] R. González and P. Real, Towards digital cohomology, *Lecture Notes in Computer Science*, v. 2886, pp. 92-101, 2003.
- [15] T. Kaczynski, K. Mischaikow, and M. Mrozek, *Computational Homology*, Appl. Math. Sci. Vol. 157, Springer Verlag, NY 2004.
- [16] R. Klette and A. Rosenfeld, *Digital Geometry: Geometric Methods for Digital Image Analysis*, Morgan Kaufmann Series in Computer Graphics, 2004.
- [17] S. Kundu, Peak-tree: a new approach to image simplification, *Proc. SPIE Vision Geometry VIII*, Vol. 3811, p. 284-294, 1999.
- [18] G. Lohmann, *Volumetric Image Analysis*, John Wiley & Sons, 1998.
- [19] R. Malgouyres and M. More, On the Computational Complexity of Reachability in 2D Binary Images and Some Basic Problems of 2D Digital Topology, *Theoretical Computer Science*, Vol. 283/1, no 1, pp 67-108, 2002.
- [20] P. Monasse and F. Guichard. Fast computation of a contrast invariant image representation. *IEEE Transactions on Image Processing*, 9(5):860-872, 2000.
- [21] *PLEX*, Topological Methods in Scientific Computing, Statistics and Computer Science, Stanford University, math.stanford.edu/comptop/program.
- [22] P. Salembier and J. Serra, Flat zones filtering, connected operators, and filters by reconstruction, *IEEE Transactions on Image Processing*, Vol. 4, n. 8, August, 1995, pp. 1153-1160.
- [23] *Simplicial Homology for GAP*, University of Delaware, www.cis.udel.edu/~dumas/Homology/.
- [24] A. J. Zomorodian, *Topology for Computing*, Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, 2005.