EasyChair Preprint

№ 4337

Image Segmentation Morphology

Arun Kumar and Pankaj Kumar Saini

October 10, 2020

IMAGE SEGMENTATION MORPHOLOGY

Arun Kumar and Pankaj Kumar Saini
Mit2020116@iiita.ac.in Mit2020117@iiita.ac.in
Indian Institute of Information Technology, Allahabad

Abstract

This paper is devoted towards the Image Segmentation with the help of Watershed Algorithm. There are many algorithms which are out there but watershed performs way better than others and helps in forming the segments in an image. The watershed principle, called topological watershed, produces correct watershed contours for grey scale image.

Keyword: **-** watershed segmentation; flooding; rain falling; computational complexity; Region growing; Region Splitting; Region Merging.

1. Introduction

In grey scale mathematical morphology, the watershed transforms, originally proposed by Digabel and Lantu´ejoul and later improved by Beucher and Lantu´ejoul, is the method of choice for image segmentation. watershed segmentation are techniques for creating meaningful clusters of pixels. Both start from "seed points" and through algorithmic means connect adjacent pixels to that seed.

In the case of watershed segmentation, you start from the local maxima of the Euclidean distance map and expand under the constraint that you cannot merge features (i.e. connect two seeds two each other.). A more generic region growing algorithm may have different means for selection of seed points (leaf nodes on a skeleton, or a fiducial mark in the image) and a different means of expanding the region that could include statistical information from one or more greyscale channels.

Which is better between them depends on two things - what you're trying to segment and how much information is contained within your images. If, for example, you're trying to segment cells in a histological specimen and you have good sample properties, watershed should work reasonably well for you. In general, watershed does ok with anything that is convex and regularly shaped, as most natural features whose shapes are given by surface energy minimization. If you have a more complex segmentation problem (neurons and plant roots come to mind) you may be better off with a region growing algorithm that can manage that complexity. The trade-off is that while region growing can be more generally applicable, it's harder to implement and easier to get wrong.

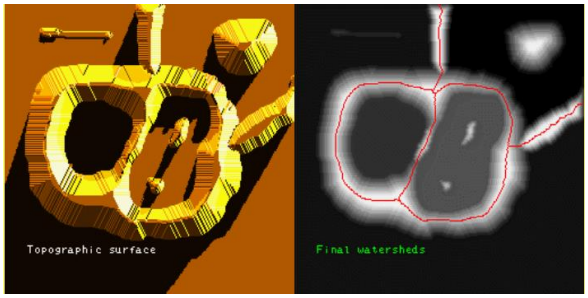

Fig 1: An example of a simple image with its watershed transform.

It depends on the object or objects you want to segment on a given background

- ➔ They are not designed for textured objects.
- ➔ For one or few objects, region growing would be more appropriate, since an initial region within the image should be provided.
- ➔ For many objects, the previous time consuming, so watershed should be more plausible.

1.1 Intuitive Notions for Watershed

The intuitive idea underlying the watershed notion comes from the field of topography: a drop of water falling on a relief follows a descending path and eventually reaches a minimum. Watershed lines are the divide lines of the domains of attraction of drops of water. This intuitive approach is not well suited to practical implementations, and can yield biased results in some cases. An alternative approach is to imagine the surface being immersed in a lake, with holes pierced in local minima. Water will fill up basins starting at these local minima, and, at points where waters coming from different basins would meet, dams are built. As a result,

the surface is partitioned into regions or basins separated by dams, called watershed lines.

1.2 Meyer's Watershed Algorithm

Starting from a greys scale image F and a set M of markers with different labels (in our case, these will be the minima of F), it expands as much as possible the set M, while preserving the number of connected components of M:

1. insert every neighbour x of every marked area in a hierarchical queue, with a priority level corresponding to the grey level F(x). Note that a point cannot be inserted twice in the queue;
2. extract a point x from the hierarchical queue, at the highest priority level, that is, the lowest grey level. If the neighbourhood Γ(x) of x contains only points with the same label, then x is marked with this label, and its neighbours that are not yet marked are put into the hierarchical queue;

Step 2 must be repeated until the hierarchical queue is empty. The watershed lines set is the complement of the set of labeled points. Let us note that this algorithm does neither label nor propagate watershed pixels, which "stop" the flooding. Thus, the watershed lines produced by Meyer's algorithm are always thinner than lines produced by other watershed algorithms.

Block Diagram of Watershed Based Segmentation

There are mainly three stages as indicated by figure 2 for watershed-based image segmentation approach. First stage is defined as pre-processing, second stage as watershed-based image segmentation and last stage as post-processing. Input image is first processed by the pre-processing stage, and then given to watershed-based segmentation stage. The resulting image is post processed by the final stage to get a segmented image. Pre-processing and post-processing are necessary to overcome the problem of over-segmentation in watershed-based image segmentation.

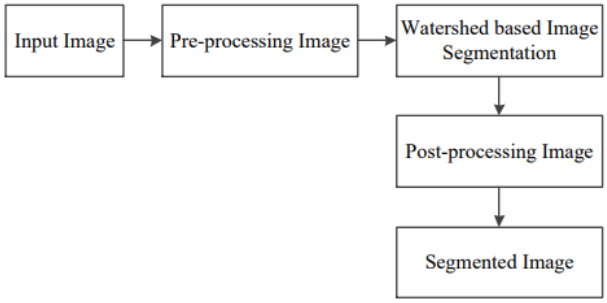

Figure 2: Block diagram of watershed-based image segmentation

Merging Basins

The decomposition of an image into regions is the basis for merging them. In the metaphorical sense of a landscape, catchment basins are merged at their watershed locations by flooding them. While some regions merge early (with low flooding level), other regions are merged later (see Fig: 3). In order to support interactive merging, Hahn and Peitgen [2003] introduced a merge tree. This tree consists of the original catchment basins as leafs and of intermediate nodes that represent merging events. A merging event is characterized by the nodes that are merged and by the flood level that is necessary for merging. As a first step, a certain amount of flooding may be applied ("pre-flooding" which may already be sufficient for segmenting the target structure [Hahn and Peitgen, 2000]).

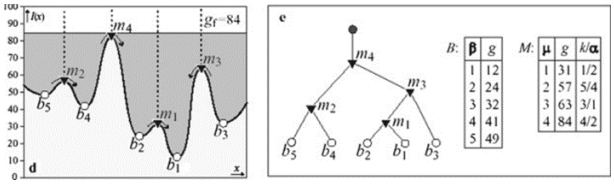

Fig: 3

Region Based Image Segmentation

- Region based segmentation is a technique for determining the region directly.
- Region growing is a simple region-based image segmentation method. It is also classified as a pixel-based image segmentation method since it involves the selection of initial seed points.

➔ Region growing
➔ Region Splitting

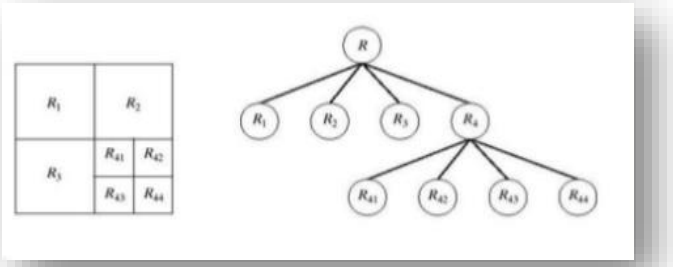

Fig:5

➔ Region Merging
➔ Split and Merge

Region growing

- Region growing is a procedure that groups pixels or sub regions into larger regions.
- The simplest of these approaches is pixel aggregation which starts with a set of seed points and from these grows regions by appending to each seed points those neighbouring pixels that have similar properties (such as gray level, texture, colour, shape).
- Region growing based techniques are better than the edge-based techniques in noisy images where edges are difficult to detect.

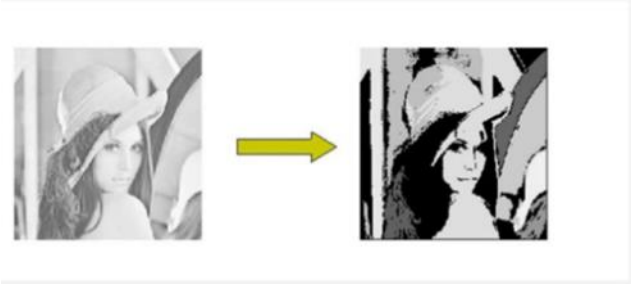

Fig: 4

Region Splitting

- Region growing start from a set of seed points.
- An alternative is to start with the whole image as a single region and subdivide the regions that do not satisfy a condition of homogeneity.

Region Merging

- Region Merging is the opposite of Region Splitting.
- Start with small regions (eg.2*2 or 4*4 regions) and merge the regions that have similar characteristics (such as gray level, variance).
- Typically, splitting and merging approaches are used iteratively.

Split and Merge

- Splitting and Merging proceed simultaneously step by step.

5**. The pseudo-code of the Algorithm**

1: Input : f , Output : l
2: v[p] ← 0, l[p] ← 0, New label ← 0, Scan Step2 ← 1, Scan Step3 ← 1 // Initialization
3: Scan from top left to bottom right : step1(p)
4: while Scan Step2 = 1 do
5: Scan image from top left to bottom right : step2(p)
6: if v[p] is not changed then
7: Scan Step2 ← 0
8: else
9: Scan image from bottom right to top left : step2(p)
10: if v[p] is not changed then
11: Scan Step2 ← 0
12: end if
13: end if
14: end while
15: while Scan Step3 = 1 do
16: Scan image from top left to bottom right : step3(p)
17: if l[p] is not changed then
18: Scan Step3 ← 0
19: else
20: Scan image from bottom right to top left : step3(p)
21: if l[p] is not changed then
22: Scan Step3 ← 0
23: end if
24: end if
25: end while
26: function step1(p)
27: if v[p] 6= 1 then

28: for each n of p // n is neighbour pixel of p
29: if f[n] < f(p) then v[p] ← 1
30: end if
31: end if
32: end function

6. **Time Complexity**

In this paper, we have overviewed and measured the expended computational resources of the watershed segmentation algorithms. Despite the fact that most of them implement the same algorithms with O(n) time complexity relative to the number of image elements, empirical testing shows a large difference in execution time, since it also depends on many other factors, such as implementation language, applied software optimizations, etc. Furthermore, most libraries showed similar results in the peak memory consumption. We identified two libraries that showed the best results in our tests. Thus, for the best performance of the watershed segmentation, one should try different implementations from modern versions of libraries and choose the one that consumes the least computing resources, despite the fact that most of them contain implementations of the same algorithms. With this testing, one can already achieve a significant increase in performance.

7. **Application in Real World**

The watershed transform has been successfully applied to a variety of segmentation tasks. Hahn and Peitgen [2000] extracted the brain with a single watershed transform from MRI data. Its generally using finding tumors , veins, lung lobes in CT data, finding targets in satellite aerial images, finding peoples in surveillance images, summarizing thumb impression and videos many more.

8**. Conclusion**

As we know Image segmentation means division of an image into meaningful structures. It is process of extracting and representing information from the image to group pixels together with region of similarity in other words “to divide an image into parts that have a strong correlation with objects or areas of the real world contained in the image”. All the objects of the original image can be identified in segmented image with their boundaries. We have seen that there were several algorithms which works same as watershed algorithm and there are many variances in performance of other types of watershed algorithm but topological watershed is far better and efficient algorithm than others. These segmentation methods differ from their computation complexity and segmentation quality.

Computational complexity is one of the important parameters to take into consideration when real time image segmentation is required. The best approach should be less computational complexity, less input parameter dependency, minimum segmentation time and provide efficient segmentation output for real time applications.

Watershed based image segmentation algorithms are less computational complex and provide very good segmentation results. It is possible to implement in the hardware using pipelined or parallel architecture for real time applications because of the independent mathematical computations flow of the algorithms.[3]

9. **References**

[1]https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_watershed.html#:~:text=The%20watershed%20is%20a%20classical,a%20local%20topography%20(elevation).

[2] https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_watershed/py_watershed.html.

[3]. Institute of Parallel and Distributed Systems Department of Parallel Systems Universit¨atsstraße 38 D–70569 Stuttgart

[4] https://youtu.be/mPJTOcEJOhY

[5] https://youtu.be/SifvvNVpMKM

[6] National Research Nuclear University MEPhI, Kashirskoye sh. 31, 115409 Moscow, Russia; ilia.safonov@gmail.com

[7] Laurent Najman and Michel Couprie
Laboratoire A2SI, Groupe ESIEE
Cit´e Descartes, BP99
93162 Noisy-le-Grand Cedex France
{l.najman,m.couprie}@esiee.fr
http://www.esiee.fr/˜coupriem/Sdi/

APPENDIX

Algorithm 1 The pseudo-code of the algorithm

1: Input : f , Output : l
2: v[p] ← 0, l[p] ← 0, New label ← 0, Scan Step2 ← 1, Scan Step
3 ← 1 // Initialization 3: Scan from top left to bottom right : step1(p)
4: while Scan Step2 = 1 do
5: Scan image from top left to bottom right : step2(p)
6: if v[p] is not changed then
7: Scan Step2 ← 0
8: else
9: Scan image from bottom right to top left : step2(p)
10: if v[p] is not changed then
11: Scan Step2 ← 0
12: end if
13: end if
14: end while
15: while Scan Step3 = 1 do
16: Scan image from top left to bottom right : step3(p)
17: if l[p] is not changed then
18: Scan Step3 ← 0
19: else
20: Scan image from bottom right to top left : step3(p)
21: if l[p] is not changed then
22: Scan Step3 ← 0
23: end if
24: end if
25: end while
26: function step1(p)
27: if v[p] 6= 1 then
28: for each n of p // n is neighbour pixel of p
29: if f[n] < f(p) then v[p] ← 1
30: end if
31: end if
32: end function
33: function step2(p)
34: if v[p]6= 1 then
35: min ← VMAX, for each n of p // n is neighbour pixel of p
36: if f(n) = f(p) and v[n] > 0 and v[n] < min then min ← v[n]
37: end if
38: if min 6= VMAX and v[p] 6= (min+1) then v[p] ← min+1
39: end if
40: end if
41: end function

42: function step3(p)
43: lmin ← LMAX, fmin ← f(p)
44: if v[p] = 0 then
45: for each n of p
46: if f(n) = f(p) and l[n] > 0 and l[n] < lmin then lmin ← l[n]
47: end if
48: if lmin = LMAX and l[p] = 0 then lmin ← New label + 1
49: end if
50: else if v[p] = 1 then
51: for each n of p
52: if f(n) < fmin then fmin ← f[n]
53: end if
54: for each n of p
55: if f(n) = fmin and l[n] > 0 and l[n] < lmin then lmin ← l[n]
56: end if
57: else
58: for each n of p
59: if f(n) = f(p) and v[n] = v[p] − 1 and l[n] > 0 and l[n] < lmin then
60: lmin ← l[n]
61: end if
62: end if
63: if lmin 6= LMAX and l(n) 6= lmin then l[p] ← lmin
64: end if
65: end function

Implementation in python

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import ndimage as ndi

from skimage.segmentation import watershed
from skimage.feature import peak_local_max



# Generate an initial image with two overlapping circles
x, y = np.indices((80, 80))
x1, y1, x2, y2 = 28, 28, 44, 52
```

```
r1, r2 = 16, 20
mask_circle1 = (x - x1)**2 + (y - y1)**2 < r1**2
mask_circle2 = (x - x2)**2 + (y - y2)**2 < r2**2
image = np.logical_or(mask_circle1, mask_circle2)

# Now we want to separate the two objects in image
# Generate the markers as local maxima of the distance to the background
distance = ndi.distance_transform_edt(image)
local_maxi = peak_local_max(distance, indices=False, footprint=np.ones((3, 3
)),
                            labels=image)
markers = ndi.label(local_maxi)[0]
labels = watershed(-distance, markers, mask=image)

fig, axes = plt.subplots(ncols=3, figsize=(9, 3), sharex=True, sharey=True)
ax = axes.ravel()

ax[0].imshow(image, cmap=plt.cm.gray)
ax[0].set_title('Overlapping objects')
ax[1].imshow(-distance, cmap=plt.cm.gray)
ax[1].set_title('Distances')
ax[2].imshow(labels, cmap=plt.cm.nipy_spectral)
ax[2].set_title('Separated objects')

for a in ax:
    a.set_axis_off()

fig.tight_layout()
plt.show()
```