

Fast Watershed Immersion Segmentation

Nathaniel Meierpolys

12/15/08

Abstract

We propose an implementation for a fast segmentation algorithm using the watershed transform. Using the immersion watershed technique, we divide an image into pieces at reasonable speeds even for large images, with the intention of refining the initial segmentation with a more computationally-intensive and sophisticated program. Beyond the raw watershed transform immersion process, we improve the final segmentation using gradient thresholds and basin-merging post-processing steps to reduce oversegmentation. The watershed-generated segmentation is integrated with a refinement algorithm to provide very high quality sub-pixel accuracy. This algorithm is designed to produce vertex information output for integration into other programs and goals. The method has the potential to serve as a method for splitting an image into pieces to be handled independently by a cluster of machines for a fast parallel segmentation implementation.

1. Introduction

Much of the computer vision field relies on the ability to glean meaningful information from the complex pixel information represented in an image. High-level object recognition depends on having candidate shapes to identify. The process of image segmentation is concerned with breaking an image into pieces according to some set of rules. Segmentation is important because it allows a computer to process images in a way similar to that used by the human eye and brain. In order to understand image data in a way that makes sense to human individuals, it is important for computers to have access to the same type of processed results, considering connected shapes rather than independent pixels. One particular method for segmenting an image is that of watershed transformation. This method relies on a gradient representation of the image which is an intensity map of the amount of change in both x and y -directions. Considering an image's gradient to be a grid of pixels with heights according to their intensity of change, immersion watershed works by flooding the image from minima and raising the water level until the whole image is submerged. The boundaries where waters from two minima meet define watershed lines and individual basins represent distinct polygons in the segmentation.

There has already been a fair amount of work in the area of watershed transformation segmentation in a number of contexts such as medical and astronomical image analysis. Image segmentation research has yielded some very powerful algorithms for obtaining near-ideal segmentations. This paper will begin with an overview of the methods, strengths, and limitations of current segmentation strategies. We will then give a detailed explanation of the proposed watershed transformation implementation and results of its use. We will finally give some direction for possible directions of future work in the algorithm itself as well as extensions of its use to a parallel implementation for fast and accurate segmentation.

2. Background

Much of our thought about watershed has been developed in the context of a very accurate segmentation algorithm called Eriol. The software uses cost-function minimization to develop and refine the segmentation of an image. By adding, removing, or repositioning vertices, Eriol establishes a very accurate representation of the image. Since the algorithm can make these changes by considering sub-pixel positions for vertices, using information about polygon area, edge length, and gradient in the entire segmentation, it can produce extremely accurate results. The computational cost of minimizing all of these interconnected components to very high precision becomes very computationally-intensive as image size increases. The feasibility of handling even a 500 pixel by 500 pixel image becomes rather unreasonable within a manageable amount of time for most applications.

The watershed transformation represents a contrast to the behavior of the above algorithm. Roerdink and Meijstert [1] give in-depth and expansive explanations and analysis of the primary methods of watershed segmentation. The subject is broken into two two main methods. Topological watershed segmentation involves making one pass through each pixel in a digital image. For each pixel, the algorithm examines paths through neighboring pixels to find a nearby pixel with the lowest height. This pixel will be the nearest minimum for the particular starting point. As an analogy, the image gradient can be considered to be a topographic landscape of hills and valleys. Topological watershed involves computing for each pixel the minimum pixel at which a drop of water would end up if it fell on the pixel under consideration. Once these values have been computed for every pixel in the gradient image, the algorithm groups pixels with common minima into catchment basins. This method works well to quickly break up an image. It does, however, run into some difficulty with plateaus of many adjacent pixels at the same height level. There are a number of variations on this general theme that address plateau issues or vary considerations of geodesic distance. As images become more and more noisy, the raw segmentation produced becomes significantly oversegmented.

The primary alternative to topological watershed segmentation is immersion segmentation. This method is the one used by this research project and will be described in much greater detail in subsequent sections. In general, immersion watershed transformation involves growing basins from minima in the image until two basins encounter one another to define an edge between them. The immersion method suffers from the same oversegmentation issues as the topological watershed did. Frucci and Longo [6] develop and explore a method for improving segmentation quality using markers. These markers represent automatically-detected regional minima in the image at which the segmentation can begin. Since there is a finite set of starting points for the transformation, other erroneous minima will not seed new basins where they do not belong, limiting basins divisions to important features.

Basic gradient images can be achieved using a Sobel operator computation for a single pixel based on its nearest neighbors. Using a redundant wavelet transform consisting of higher-level analysis of curves and results from scaling described by Jung and Scharcanski [2] produces a cleaner starting point to be used for watershed segmentation. Jung and Scharcanski also suggest some robust methods for merging polygons after segmentation is completed on the basis of polygon size and strength of the edges dividing them.

3. Pre-processing

Before any watershed transformation can begin, the algorithm needs a way of representing the test image in terms of the amount of change around any particular pixel. We apply the Sobel operator to each pixel in the greyscale representation of the original image. We compute the x-component and the y-component of change separately and then find the magnitude of these two vectors. The resulting values produce a gradient image which we use as a height map.

$$\mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A}$$

Figure 3.1: Sobel operator for x-direction (G_x) and y-direction (G_y) [5]

The resulting values produce a gradient image which we use as a height map. When applying the Sobel operator, we must decide whether to use the red, green or blue channel, or the greyscale average. Since there is no guarantee that a given image can be represented better by a single red, green, or blue channel, we use the greyscale average. To maintain data integrity for as long as possible, our algorithm computes the gradient values for all three channels independently and averages the final values to give a greyscale representation. The program is designed to optionally take a more sophisticated gradient as input.

The gradient result obtained using the above methods results in a clear view of the boundaries in an image to the human eye. Considering the way the watershed transformation will work in later steps, minute variations will emerge as faint gradient values. To reduce the effects of these noisy components and better aid the segmentation in later steps, we apply a gradient threshold which replaces any gradient values below the given threshold with 0. In the current implementation, the values used for the gradient threshold are specified by the user for each image.

4. Watershed

The gradient calculations carried out during the pre-processing stage provide a height map that can be used for a watershed transform. The height map works to map a particular height to each pixel in the image. In this case, the height of each pixel represents the respective gradient value or degree of change in the x and y directions.

This implementation uses an immersion watershed technique based loosely on the Vincent-Soille algorithm described in [1]. To describe the general process of the immersion watershed transformation segmentation, it is helpful to imagine a topographical landscape of hills and valleys. Set at the very minimum of each valley is a hole or tap. The landscape is slowly lowered into a body of water such that water flows up through minima to begin flooding the landscape. When the water level raises high enough for two basins to overflow into each other, this point is defined as a dam separating them.

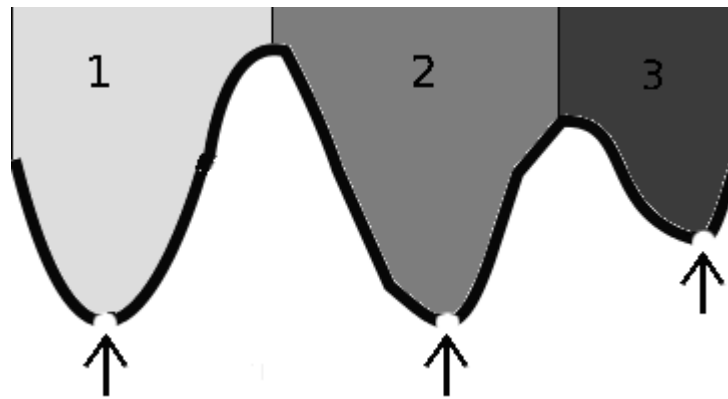


Figure 4.1: Three basins filling from sinks at minima.

When applied to an image, hills are pixels with large heights (gradient values) and valleys are marked by low heights. Our algorithm begins with minima values and proceeds by incrementally raising the “water level”. Any pixels discovered from a particular minimum starting point belong to the same basin. Whenever pixels from a basin encounter those of another basin, the point is marked as a watershed. This process continues raising the water level and exploring pixels appropriately until the entire image has been grouped into basins. Within the constraints of this algorithm, all pixels are assigned to basins. Edges are defined as those pixels where a basin first encounters its neighbor in the flooding process. The exploration process is inherently recursive, branching to explore each neighbor of the original pixel and so on until every pixel within a minimum's reach is explored.

```
//explores pixels in order of height
segment():
```

```
    sort pixels into height vectors;
```

```
    for each height
        for each pixel in heightVectors[height]
            if( !pixel.visited )
                explore( pixel );
```

```
//processes this pixel according to its label and its neighbors' labels
explore(watershedPixel *pixel):
```

```
    neighbors = getNeighbors(pixel)
```

```
    //give this pixel a label from its neighbor or a watershed if it touches two different basins
    if(pixel.label == -1)
        for each neighbor in neighbors
            if(neighbors.label != -1)
                if(pixel.label == -1)
                    pixel.label = neighbors.label;
                else if(pixel.label != neighbor.label)
                    pixel.label = 0;
```

```
//begin a new basin with this pixel if it is still unmarked
if(pixel.label == -1)
    pixel.label = ++basincounter;

//explore all unmarked neighbors
for each neighbors
    if(neighbor.label < 0 )
        explore(neighbor);
```

To avoid exceeding c++ limits of stack calls in recursion, we use a while loop, FIFO queue to keep to manage the breadth-first exploration process. Discovered pixels are pushed onto the queue and pixels to be processed are then popped off the other end.

The watershed transformation is incredibly desirable for its speed. The algorithm runs quickly even for large images because each pixel is processed just once and its value is dependent only on the finite number of neighboring pixels. This means that each pixel can be handled in constant time. As the image size grows, then, the overall computation required grows by the same linear factor.

5. Post-processing

Unfortunately, the speed of this process is accompanied by a sacrifice in initial quality. The resulting image is heavily oversegmented with thousands of tiny basins. This is an improvement on the original image, but there is still an incredibly amount of complexity and for many applications this would be a step backwards. To improve the segmentation and reduce the number of basins, we use three primary methods. Each of these can be turned on or off by the user. A default value of

5.1. Gradient Threshold:

The first is the application of a gradient threshold as described in pre-processing which drastically reduces the amount of low-level noise in the image, leaving only more significant basin boundaries in place.

5.2. Small Polygons:

Oversegmented images are characterized by many small polygons of little significance. These often occur in places where watershed lines are thicker and poorly defined. In these cases many small polygons mark the border of a larger and more significant polygon. We impose a second threshold that will match any polygons below a given area. The removal process analyzes each of the edges shared between this small polygon and its neighbors to determine which neighbor to merge with. Since watershed is based on the concept of water flowing over the lowest components of a dam, the small polygon is merged over the edge with lowest pixels to combine with that neighbor.

5.3. Edge Threshold:

Though many of the insignificant polygons in an image are those with small area and gradient, there are some as well which are very large and must be merged to fully represent features coherently as single polygons. We apply a threshold to each edge of each polygon in the segmentation. If the pixels comprising a particular edge are all above the given threshold, then the edge is deemed solid and kept. If, however, there are pixels below this threshold, we consider the edge to be weak in the context of watershed flooding and allow the two polygons to merge. Applying this process to each edge in the image drastically reduces oversegmentation while maintaining the most important features in an image.

5.4. Hysteresis Edge Threshold:

The above methods of processing do not suffice to fully perfect the segmentation. Images can still benefit from greater refinement in cases when larger polygons are separated by fairly low and insignificant edges. We implement a double-threshold algorithm, introduced in [5] which handles this problem. Edges are considered solid according to the application of two thresholds. As is the case with both previous thresholds, these are defined by the user on an image-by-image basis. The first threshold is a minimum. Every pixel in an edge must be above this threshold to qualify. The second threshold is a representation of height. At least 25% of an edge's pixels must be greater than the second threshold. We apply the two thresholds to each edge in each polygon to determine that edge's strength. If a given edge does not meet the threshold, the two polygons bounded by the edge are merged. This merging step addresses the issue of large objects which appear as one shape to the human eye being nevertheless split into several pieces based on tiny variations in the pixel data for an image.

6. Output

There are two methods of output. An image is generated illustrating each discovered polygon with a random color. This is primarily for the sake of the user to verify the segmentation results. More significantly, however, the polygons are represented in FIM format and stored for further processing. Polygons in FIM format consist of a series of vertex positions outlining the shape of the polygon, followed by RGB color and gradient information. The actual process of creating this representation involves imposing some additional constraints on the segmentation. For the vertex boundary description to be useful for polygon reconstruction, all pixels within a polygon must be connected by horizontal or vertical edges. Diagonal edges create ambiguous situations regarding connectivity and vertex order (Fig. 6.1).

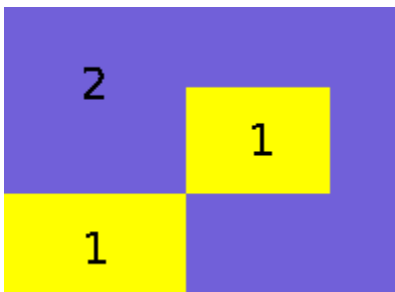


Figure 6.1: Illustration of diagonal polygon connectivity problem

This algorithm traverses the boundary of each basin identifying the necessary pixels to describe the shape in consideration. To main special situations must be negotiated to maintain correct connectivity. Region 2 completely envelopes part of region 1 and connects to itself via a single pixel. To properly represent region 2, our algorithm (approaching from below the single pixel) turns right, bounds region 1 and turns right again to avoid crossing the diagonal boundary and looping infinitely.

A second special case concerns region 1 of the example. For a polygon to be expressed easily and reconstructed, it cannot be connected by a single point. To avoid the issue, we divide any polygons matching this description into two pieces, giving the second piece a new label and separating it as a new polygon.

Finally we scan the edge vertices of each polygon for redundancy, since only two vertices are needed to represent a straight line. Removing any extra vertices along a line reduces the size of the FIM output file by a factor of two. This is especially valuable to the prospect of integrating this watershed segmentation into a broader pipeline, since the extra vertices increase the representation's complexity significantly.

7. Eriol Integration

This watershed transformation algorithm results in a fairly good but imperfect segmentation of the image, in which some edges are very good and some are poor. This initial segmentation can serve as a starting point for a more sophisticated algorithm. Such an algorithm would otherwise need to implement a top-down segmentation where edges are initially placed at each pixel boundary and incrementally removed, or bottom-up segmentation where geometry is incrementally added to an empty segmentation. The function-minimization method implemented in Eriol and described earlier represents a powerful tool for refining a segmentation until it is near-ideal. This software is very well suited to improving on the quality results of this watershed segmentation.

Since watershed segmentation represents a quick method for automatically generating a fairly good segmentation quickly, and Eriol provides a slower but high-quality segmentation solution, the combination of the two can have very positive results. The initial segmentation generated provides Eriol with a new third starting place in which much of the work is already done. The FIM file format serves as a effective means of transferring watershed-generated vertex information to Eriol for refinement. Geometry is added, removed, and repositioned from the initial segmentation to result in a much better segmentation.

8. Results

8.1. Quality Factors:

This algorithm performs fairly well with a variety of test images. Given proper thresholds, the image is successfully segmented to produce a representative set of polygons. The algorithm is not intended to be perfect. Often objects are broken into pieces. In cases of poor image quality or tiny images, the watershed algorithm sometimes mistakes important features for noise. Since the algorithm does not implement any function minimization or greedy merging of polygons, results are severely limited by

the requirement that any thresholds must be applied universally across all pixels, edges or polygons in the image. For most images, however, this watershed algorithm succeeds in presenting a much simpler geometric representation of the features in each image which can be used for later processing.

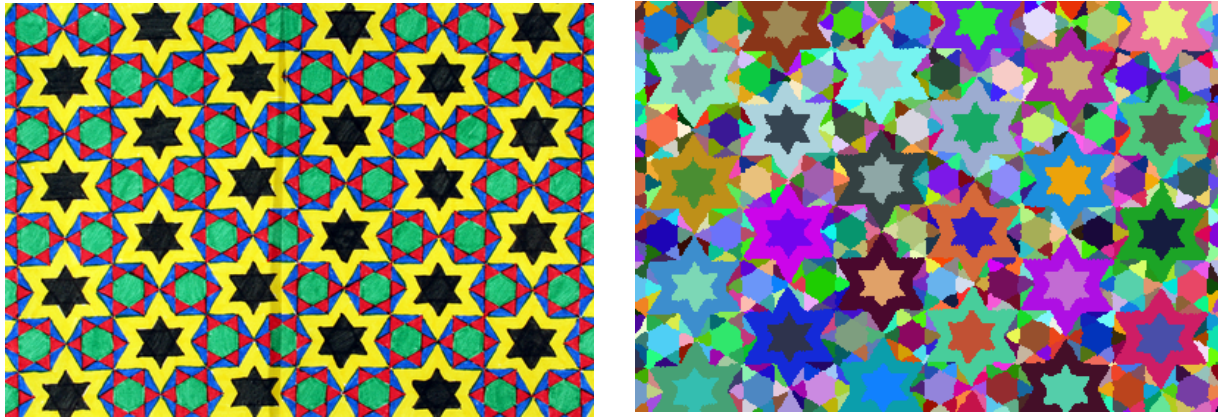


Figure 8.1: a) Original image and b) segmented result, random colors

8.2. Time Factors:

As noted throughout its description, the primary strength of this algorithm is its speed. Whereas many alternative solutions grow according to $O(n^2)$, $O(n^3)$, or steeper relationships with large number of pixels n , this process is linear. A typical megapixel image can be segmented and refined in 10-20 seconds. This speed opens the door to more broad applications of segmentation for larger images, or those with smaller images but with more strict speed requirements.

8.3. Eriol Integration

Combining the speed of the watershed-segmentation method with the quality of Eriol, we achieve a very fast new strategy for high-quality image segmentation at reasonable speeds. The sample image in Figure 7.1 with resolution of 400 pixels by 287 pixels is divided into approximately 800 polygons in approximately a matter of minutes. The new integration of these two segmentation methods improves on Eriol's original segmentation since the original segmentation provides a good enough starting point to require only that Eriol remove unnecessary vertices and not add any. The significant speed increase will presumably allow current segmentation applications to expand in scale and an entirely new set of uses to become more feasible.

Figure 8.2: Results of integration with Eriol: vertices marked in white, edges marked in red.

9. Future Work

9.1. Automatic Thresholding

Presently, this algorithm relies on user-input for the threshold levels for pre and post-processing steps. There are many cases where a reliance on a user to drive the segmentation process restricts its

usefulness. Since no single set of thresholds will work for all images, an automatic method for determining threshold values is desirable. A statistical study of gradient, polygon size, and edge values in an image would potentially yield a set of thresholds that automatically scale to fit a particular image. This addition would provide a valuable feature since it would allow a user to achieve a good segmentation without requiring any advanced knowledge of the processes involved.

9.2. Parallelization

Though the individual watershed algorithm is quick, its refinement with more sophisticated quality-focused algorithms takes time for large image. Given a cluster of many machines, it may be possible to achieve much better overall running time. The greatest difficulty with segmenting large images using quality-focused methods is the need to evaluate elements dependent on the entire image at each step of the process. A fast watershed initial segmentation could serve as a means of dividing up an image into smaller pieces that do not depend heavily on the rest of the image. Each of these pieces could be segmented independently on separate machines to provide the best local segmentation. Using information about the quality of each local segmentation, a head node could combine the individual pieces to yield a coherent final segmentation of the overall image.

10. Conclusion

The watershed transformation-based segmentation presented here has proved to be very quick compared to alternative methods of segmentation. The segmentation it produces succeeds in identifying much of the geometry in the image but suffers from a dependence on a single set of thresholds across the entire image. Variations from image to image in contrast and complexity make it difficult to establish a universal set of parameters for the optimal segmentation. Nevertheless, the resulting segmentation provides an effective starting place for a variety of applications in the field of computer vision.

11. Acknowledgments

This project has received considerable assistance from a number of contributors. We would like to recognize Mike Krahulec for his direct work on the development of this algorithm. We would also like to thank Olaf Hall-Holt as faculty advisor on the project and fellow researchers in the 2008 Capstone class for their continued support and feedback.

References

- [1] Roerdink, Jos B.T.M. and Meijster, A. The Watershed Transform: Definitions, Algorithms, and Parallelization Strategies. *Fundamenta Informaticae 41* (2001), 187-228. IOS Press.
- [2] Jung, C.R. and Scharcanski, J. Robust Watershed Segmentation Using the Wavelet Transform. *Computer Graphics and Image Processing, 2002. Proceedings. XV Brazilian Symposium on.* (2002), 131-137.

- [3] Wikipedia: Sobel Operator: http://en.wikipedia.org/wiki/Sobel_operator
- [4] Najman, L. and Couprie M. Watershed algorithms and contrast preservation. *Lecture Notes in Computer Science* (2003).
- [5] Frucci, M. and Longo, J. Watershed transform and the segmentation of astronomical images. *Proceedings of Astronomical Data Analysis III* (2004).