



12-2001

# Hierarchical Image Segmentation using The Watershed Algorithm with A Streaming Implementation

Annapoorani Gothandaraman  
*University of Tennessee - Knoxville*

---

## Recommended Citation

Gothandaraman, Annapoorani, "Hierarchical Image Segmentation using The Watershed Algorithm with A Streaming Implementation." Master's Thesis, University of Tennessee, 2001.  
[https://trace.tennessee.edu/utk\\_gradthes/1966](https://trace.tennessee.edu/utk_gradthes/1966)

To the Graduate Council:

I am submitting herewith a thesis written by Annapoorani Gothandaraman entitled "Hierarchical Image Segmentation using The Watershed Algorithm with A Streaming Implementation." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Ross T. Whitaker, Major Professor

We have read this thesis and recommend its acceptance:

Michael J. Roberts, Hairong Qi, Jens Gregor

Accepted for the Council:  
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

---

To the Graduate Council:

I am submitting herewith a thesis written by Annapoorani Gothandaraman entitled 'Hierarchical Image Segmentation using The Watershed Algorithm with A Streaming Implementation'. I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Ross T. Whitaker

Major Professor

We have read this thesis  
and recommend its acceptance:

Michael J. Roberts

Hairong Qi

Jens Gregor

Accepted for the Council:

Dr. Anne Mayhew

Vice Provost and  
Dean of Graduate Studies

(Original signatures are on file in the Graduate Student Services Office.)

# **Hierarchical Image Segmentation using The Watershed Algorithm with A Streaming Implementation**

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Annapoorani Gothandaraman

December 2001

## **Acknowledgments**

I would like to thank my parents, Mrs. Rani Gothandaraman and Mr. Gothandaraman for their continuous support and encouragement during my education. I would like to dedicate this work to them for all that they have done for me. I would also like to thank my advisor Dr. R. T. Whitaker, and Dr. J. Gregor for helping me in my research work, and other committee members, Dr. H. Qi and Dr. M. J. Roberts. I want to mention my sister Ms. Akila Gothandaraman — we have mutually derived motivation from one another to study and learn more, right from our early school days; my best friend, Mr. Rameshbabu Prabagaran, who has supported me, even in the worst of tides; many other friends who have made my stay in Knoxville, enjoyable and memorable.

## **Abstract**

We have implemented a graphical user interface (GUI) based semi-automatic hierarchical segmentation scheme, which works in three stages. In the first stage, we process the original image by filtering and threshold the gradient to reduce the level of noise. In the second stage, we compute the watershed segmentation of the image using the rainfalls simulation approach. In the third stage, we apply two region merging schemes, namely implicit region merging and seeded region merging, to the result of the watershed algorithm. Both the region merging schemes are based on the watershed depth of regions and serve to reduce the oversegmentation produced by the watershed algorithm. Implicit region merging automatically produces a hierarchy of regions. In seeded region merging, a selected seed region can be grown from the watershed result, producing a hierarchy. A meaningful segmentation can be simply chosen from the hierarchy produced.

We have also proposed and tested a streaming algorithm based on the watershed algorithm, which computes the segmentation of an image without iterative processing of adjacent blocks. We have proved that the streaming algorithm produces the same result as the serial watershed algorithm. We have also discussed the extensibility of the streaming algorithm to efficient parallel implementations.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Image Segmentation . . . . .	1
1.2	Significance of Automatic Segmentation . . . . .	2
1.3	Survey of Segmentation Methods . . . . .	2
1.3.1	Classification . . . . .	3
1.3.2	Discussion . . . . .	6
1.4	Summary of Work Done . . . . .	8
<b>2</b>	<b>Related Work</b>	<b>10</b>
2.1	Introduction . . . . .	10
2.2	Early Work on Watershed Segmentation . . . . .	12
2.3	Different Approaches to Watershed Segmentation . . . . .	13
2.3.1	Immersion or Flooding Simulation . . . . .	13
2.3.2	Rainfalling Simulation . . . . .	15
2.3.3	Morphological Watershed Segmentation . . . . .	16

2.3.4	Distributed Watershed Segmentation . . . . .	17
2.3.5	Discussion . . . . .	18
2.4	Strategies to Prevent Oversegmentation . . . . .	19
2.4.1	Pre-processing Techniques . . . . .	20
2.4.2	Post-processing Techniques . . . . .	22
2.4.3	Hierarchical Segmentation . . . . .	24
2.4.4	Discussion . . . . .	25
<b>3</b>	<b>Watershed Segmentation</b>	<b>27</b>
3.1	Introduction . . . . .	27
3.2	Pre-processing Step . . . . .	28
3.3	Watershed Segmentation Algorithm . . . . .	30
3.3.1	Definitions . . . . .	30
3.3.2	Rainfalling Simulation Algorithm . . . . .	31
3.4	Hierarchical Segmentation by Region Merging . . . . .	33
3.4.1	Implicit Region Merging . . . . .	33
3.4.2	Seeded Region Merging . . . . .	36
3.5	Watershed Segmentation Tool . . . . .	38
<b>4</b>	<b>Streaming Algorithm</b>	<b>42</b>
4.1	Introduction . . . . .	42
4.2	Details of the Streaming Algorithm . . . . .	44
4.3	Proof of Correctness . . . . .	59

4.4	Parallel Implementation . . . . .	63
4.5	Cost of Generating the Lookup Table . . . . .	66
<b>5</b>	<b>Results and Discussion</b>	<b>69</b>
5.1	Watershed Algorithm — Testing . . . . .	69
5.2	Effect of the Pre-processing Step . . . . .	73
5.3	Hierarchical Segmentation . . . . .	79
5.3.1	Implicit Region Merging . . . . .	79
5.3.2	Seeded Region Merging . . . . .	86
5.4	Segmentation of Slices of a 3D Anatomical Image . . . . .	90
5.5	Streaming Algorithm . . . . .	97
<b>6</b>	<b>Conclusion</b>	<b>110</b>
	<b>Bibliography</b>	<b>114</b>
	<b>Vita</b>	<b>124</b>

# List of Tables

5.1	Effect of the pre-processing stage in reducing oversegmentation . . . . .	79
5.2	Implicit Region Merging for the house image (See Figure 5.12): Number of regions at different levels . . . . .	82
5.3	Implicit Region Merging for the lamp image (See Figure 5.13): Number of regions at different levels . . . . .	82
5.4	Implicit Region Merging for the road image (See Figure 5.14): Number of regions at different levels . . . . .	84
5.5	Implicit Region Merging for the 3D phantom image of the head (See Figure 5.15): Number of 3D regions at different levels . . . . .	86
5.6	Implicit Region Merging for slice 1653 (See Figure 5.21) : Number of regions at different levels . . . . .	94
5.7	Implicit Region Merging for slice 1668 (See Figure 5.22) : Number of regions at different levels . . . . .	94

# List of Figures

2.1 An example of topographical surface: (a) Input $f(u, v)$ (b) Its topographical surface in 3D . . . . .	11
2.2 Working of the watershed algorithm: (a) Original 1D function $I(x)$ (b) Gradient of $I(x)$ — $f(x)$ is the input to the watershed algorithm (c) Result of watershed algorithm . . . . .	12
2.3 Immersion simulation: (a) An example 1D gradient function (b) Flooding by immersion (c) Segmented result . . . . .	14
2.4 Rainfalling simulation: (a) An example 1D gradient function (b) Rainfalling simulation (c) Segmented result . . . . .	15
2.5 An example of oversegmentation: (a) A noisy image (b) Gradient image (c) Watershed result showing oversegmentation . . . . .	20
2.6 Effect of thresholding the gradient: (a) Watershed segmentation of gradient $g(x)$ containing irrelevant minima (b) Watershed segmentation of the thresholded gradient $f(x)$ . . . . .	22
2.7 Effect of a marker function on the gradient . . . . .	23

3.1	Classification of pixels on the topographical surface: (a) Single-pixel minimum (b) Types of flat regions . . . . .	31
3.2	Watershed depth of a region. . . . .	34
3.3	Implicit region merging . . . . .	35
3.4	Seeded region merging . . . . .	37
3.5	Snapshot of the user interface for 2D images . . . . .	40
3.6	Snapshot of the user interface extended for 3D images . . . . .	41
4.1	An example of incorrect identification of a border pixel due to splitting: (a) $P$ as non-minimum pixel in the image (b) $P$ as a single-pixel minimum in block 1 . . . . .	45
4.2	Examples of the problem of split flat regions: (a) Example of a re-entering segment (b) Example of a split flat region . . . . .	46
4.3	Splitting of a 1D steepest descent path . . . . .	48
4.4	Example of splitting of a 2D steepest descent path . . . . .	49
4.5	Division of the image into blocks with overlap . . . . .	51
4.6	An example to illustrate overlap among adjacent blocks: (a) Original image (b) Splitting of the image into blocks with overlap . . . . .	52
4.7	Boundary minima are stored for (a) Pixels in the shaded rows based on the row type and (b) Pixels in the shaded columns based on the column type. (c) Example of blocks of (Row type, Column type) equal to $(F, L)$ and $(M, M)$ . . .	53

4.8	Marking pixels as <i>unknown</i> based on row and column types of blocks: (a) Pixels and flat regions spread over shaded rows are <i>unknown</i> . (b) Pixels and flat regions spread over shaded columns are <i>unknown</i> . (c) Example of a block of (Row type,Column type) equal to $(F, F)$	55
4.9	Pairwise processing of adjacent blocks for the example in Figure 4.6	56
4.10	Processing of two adjacent blocks	57
4.11	The pixel with value <b>3</b> traces to the minimum <b>0</b> carrying an <i>unknown</i> label.	61
4.12	Proof — Minimum within the block	61
4.13	Proof — Minimum in an adjacent block	61
4.14	Proof — Minimum several blocks away	62
4.15	Binary reduction of columns	64
4.16	Binary reduction of rows	65
4.17	Parallel implementation of the binary reduction scheme	67
5.1	Verification of the 2D implementation of the watershed algorithm: (a) 2D sine as gradient input ( $128 \times 128$ ) (b) Watershed segmentation	70
5.2	Verification of the 2D implementation of the watershed algorithm: (a) Image of a box ( $745 \times 734$ ) (b) Gradient image (c) Watershed segmentation	70
5.3	Verification of the 2D implementation of the watershed algorithm: (a) Image with objects of different shapes ( $236 \times 201$ ) (b) Gradient image (c) Watershed segmentation	71
5.4	A 3D phantom image of the head ( $54 \times 54 \times 50$ )	72

5.5	Gradient of the slices of the 3D phantom image . . . . .	72
5.6	Watershed segmentation of the 3D phantom image . . . . .	73
5.7	Effect of pre-processing on 2D watershed segmentation: (a) A synthetic image ( $335 \times 288$ ) (b) Gradient image (c) Watershed segmentation (d) Corrupted image ( $\sigma_n = 50\%$ of RMS gradient) (e) Gradient of (d) (f) Oversegmented watershed result (without pre-processing) (g) Watershed segmentation (with pre-processing — $\sigma_f = 4; T = 92\%$ ) . . . . .	75
5.8	Effect of pre-processing on 2D watershed segmentation: (a) A noisy and textured image ( $256 \times 240$ ) (b) Gradient image (c) Oversegmented watershed result (without pre-processing) (d) Watershed segmentation (with pre-processing — $\sigma_f = 0.5; T = 43\%$ ) . . . . .	76
5.9	Effect of pre-processing on 3D watershed segmentation: (a) Oversegmented watershed result for Figure 5.4 after addition of noise (without pre-processing) (b) Result of watershed segmentation (with pre-processing — $\sigma_f = 0.5; T = 4\%$ ) . . . . .	77
5.10	Effect of pre-processing on watershed segmentation: (a) Original image (b) Gradient image (c) Oversegmented watershed result (without pre-processing) (d) Watershed segmentation ( $\sigma_f = 0.5; T = 12\%$ ) (e) Watershed segmentation ( $\sigma_f = 0.5; T = 25\%$ ) . . . . .	78

5.11 Implicit region merging: (a) A synthetic image (b) Gradient image (c) Watershed segmentation (d) Corrupted image ( $\sigma_n = 25\%$ of RMS gradient) (e) Gradient of (d) (f) Oversegmented result (without pre-processing) . . . . .	80
5.12 Hierarchy produced by implicit region merging: (a) Watershed segmentation of corrupted image ( $\sigma_f = 2; T = 5\%$ ). Regions at levels 460, 680, 784, 786, 789 are shown in (b)–(f). . . . .	81
5.13 Hierarchy produced by implicit region merging: (a) Watershed segmentation for image in Figure 5.10(a) ( $\sigma_f = 1; T = 2\%$ ). Regions at hierarchy levels 950, 1250, 1311 are shown in (b)–(d). . . . .	83
5.14 Hierarchy produced by implicit region merging: (a) A real image — noisy and textured (b) Gradient image (c) Watershed segmentation ( $\sigma_f = 1; T = 6\%$ ). Regions at hierarchy levels 775, 1235, 1395, 1455, 1468 are shown in (d)–(h). . . . .	85
5.15 Hierarchy produced by implicit region merging: (a) Watershed segmentation of Figure 5.4 after addition of noise ( $\sigma_f = 0.5; T = 0.2566\%$ ). Regions at levels 1, 211, 292 are shown in Figures 5.15(b)–(d). . . . .	87
5.16 Hierarchy produced by seeded region merging: (a) Input image showing the seed segment (b) Oversegmented result ( $\sigma_f = 2; T = 5\%$ ) (c)&(d) Growing of the seed segment (e) Merging of the seed segment with the background in the image . . . . .	88

5.17 Hierarchy produced by seeded region merging: (b) Input image showing the seed segment (a) Watershed segmentation ( $\sigma_f = 1.5; T = 6\%$ ) (c)&(d) Growing of the seed segment . . . . .	89
5.18 Slices of a 3D medical image . . . . .	91
5.19 Slices of the gradient image . . . . .	92
5.20 Oversegmented watershed result ( $\sigma_f = 1; T = 9.63\%$ ) . . . . .	93
5.21 Result of implicit region merging: (a) Slice 1653 (b) Oversegmented watershed result. Regions at levels 12400, 12592, 13000, 13320, 13480, 13580 are shown in (c)–(h). . . . .	95
5.22 Result of implicit region merging: (a) Slice 1668 (b) Oversegmented watershed result. Regions at levels 12200, 12750, 12975, 13225, 13405, 13515 are shown in (c)–(h). . . . .	96
5.23 Result of seeded region merging: (a) Slice 1653 showing seed segment. Growing of the segment is shown in (b)–(d). (e) Slice 1663. Growing of the segment is shown in (f)–(h). . . . .	98
5.24 Result of seeded region merging: (a) Slice 1657 showing seed segment. (b) Grown segment. Growing of the segment for the slices 1660, 1663, 1666 in (c),(e),(f) are shown in (d),(f),(h) respectively. . . . .	99
5.25 Streaming algorithm: (a) Test image — letter ‘A’ (b) Splitting of the image (c) Watershed result before pairwise processing (d) Result of the streaming algorithm . . . . .	100

5.26 Streaming algorithm: (a) Test image (b) Splitting of the image (c) Watershed result before pairwise processing (d) Result of the streaming algorithm . . . .	102
5.27 Streaming algorithm: (a) Test image (b) Splitting of the image (c) Watershed result before pairwise processing (d) Result of the streaming algorithm . . . .	104
5.28 Streaming algorithm: (a) Test image (b) Splitting of the image (c) Watershed result before pairwise processing (d) Result of the streaming algorithm . . . .	105
5.29 Streaming algorithm: (a) Test image (b) Splitting of the image (c) Watershed result before pairwise processing (d) Result of the streaming algorithm . . . .	106
5.30 Streaming algorithm: (a) Test image (b) Splitting of the image (c) Watershed result before pairwise processing (d) Result of the streaming algorithm . . . .	107
5.31 Streaming algorithm: (a) Test image (b) Splitting of the image (c) Watershed result before pairwise processing (d) Result of the streaming algorithm . . . .	109

# **Chapter 1**

## **Introduction**

### **1.1 Image Segmentation**

Segmentation is the process of dividing an image into *meaningful* regions. It is often considered the first and the most important step in image analysis [20]. Segmentation finds application in a variety of fields — from medicine to defense. For instance, in medicine, it is used for image-guided surgery, surgical simulation, therapy evaluation, neuroscience studies, and diagnosis. A major area of application of segmentation methods is in solving problems related to machine vision. Some examples of these are automatic character recognition, production line quality control, automatic processing of finger prints, target recognition and tracking, and surgical robotics. A majority of these machine vision problems require partially- or fully-automatic segmentation.

## 1.2 Significance of Automatic Segmentation

Objects in images appear at different spatial resolutions. Hence building a generic automatic segmentation system may be difficult or even impossible. However, for a *given* application, automatic segmentation can be achieved with the right choice of segmentation method. Automating tasks can help improve system performance in many applications. For example, consider a problem that involves checking correctness of connections on a printed circuit board. Verifying each board manually will be slow and the results may be error-prone. Introducing automation in such an application can significantly improve system speed and produce repeatable results. Yet another instance is segmentation of medical data for diagnosis. This data could be images, 3D volumes [48] or even 4D time-series of volumes<sup>1</sup>. The complexity of such images does demand human intervention. However, manual segmentation can be highly time-consuming and error-prone. A semi-automatic segmentation method with the right use of the user's high-level knowledge can avoid problems with the manual techniques.

## 1.3 Survey of Segmentation Methods

There are many segmentation techniques in the literature for different kinds of images (intensity images, range images, images with high noise level, highly textured images, video sequences, color images, etc). However, there is no single method that works well for all image types. According to Haralick and Sapiro [23], a *good* segmentation, *qualitatively*, is one, in which the regions are uniform and homogeneous with respect to some characteristic, adjacent regions

---

<sup>1</sup>Henceforth, we will use the term *images* in general to refer to images, 3D volumes, and 4D data.

have significantly different characteristic features and boundaries of each segment are spatially accurate. Unfortunately, no standard *quantitative* performance metric has been developed yet and hence the evaluation is mostly subjective.

### 1.3.1 Classification

Instances of work in the literature on the study of segmentation methods are [18, 23, 28, 50]. Of these, the review presented by Pal and Pal [50] is very comprehensive. They discuss several existing methods under the following topics: gray-level thresholding, iterative pixel classification (methods that use relaxation, Markov random fields, and neural networks), surface-based techniques (mostly for range images), color image segmentation, edge detection, and methods based on fuzzy set theory. Most books on image processing also provide an overview of the common algorithms [20, 54]. In our study, we broadly classify the segmentation methods into four classes, namely boundary-based, pixel-based, and region-based methods, and methods that use a combination of these.

#### Boundary-based Methods

Boundary-based methods are based on the detection of discontinuities in an image. The first step, typically, is to apply an edge detector [20] to the image. Heath *et al* [25] present a study of various edge detection operators. In the ideal case, the edge operator should find points lying only on the boundaries between regions but this seldom happens due to texture and noise. Hence edge detection is usually followed by edge-point linking and boundary detection methods [54] to obtain meaningful boundaries. Some examples of this approach are [17, 31, 69]. Yu and Jain

[69] use the Hough transform [54] to detect lane boundaries. Eua-Anant and Upda [17] use particle motion in a simulated force field to obtain closed object boundaries. Jones [31] reviews the application of active contour models for detecting lines and edges in images.

### **Pixel-based Methods**

Pixel-based methods work at the level of pixels in the image, grouping them based on a pre-defined similarity criterion. Thresholding and clustering are two important pixel-based methods. In thresholding, the values of the thresholds are usually obtained by histogram analysis [20]. Depending on the nature of the threshold, the method may be local, global or adaptive and based on the number of thresholds, it can be of bi-level or multi-level type. Two examples of use of thresholding from the literature are [61, 68]. Solihin and Leedham [61] propose a novel class of global thresholding techniques for analyzing handwriting images. Yang and Yan use thresholding for a document processing system in [68]. Clustering can be viewed as a multi-dimensional extension of thresholding. It involves extracting features from an image and grouping the pixels in a higher-dimensional feature space [16]. Examples of this approach are [26, 38]. Heisele and Ritter [26] apply clustering to segment temporal sequences of range and intensity images. Lucchese and Mitra [38] propose an unsupervised segmentation scheme based on K-means clustering for color images.

### **Region-based Methods**

Region-based methods also aim at segmentation using similarities but they work at the level of regions. Region growing and merging, region splitting, split-merge [20], and the watershed

algorithm [30] are methods based on finding regions directly. In region growing, starting from seed pixels, regions are grown based on pre-defined homogeneity and stopping criteria. Region merging works by merging *similar* regions obtained from a prior segmentation of the image. Region splitting is just the opposite of merging. It starts with the entire image as a single region which is subsequently split until a stopping criterion is met. A convenient approach is to use a quadtree representation [20] in which each region is split into four and this proceeds until the homogeneity conditions are satisfied. This approach has two problems. First, it assumes a square region shape and second, it may result in a segmentation that has adjacent regions with identical properties. This can be overcome by a combination of splitting and merging.

The watershed algorithm is based on concepts from topography [30]. The input to the algorithm is a single-valued edge map, which is typically the gradient of the original image. The algorithm considers the edge map as a topographical surface where the height at each pixel location is given by the grayscale value of that pixel. Each local minimum in the surface corresponds to a *catchment basin*. The algorithm tries to group the other pixels such that in the final segmentation, most other pixels belong to a catchment basin. The boundary pixels form the *watershed lines* that separate the catchment basins. This grouping can be done using two methods — immersion simulation and rainfalls simulation. In immersion simulation [60], holes are pierced in the minima of the surface and the surface is immersed in water. The water rises and starts filling the catchment basins. When the catchment basins begin to merge, dams are built to prevent water in one basin from flowing into another. These dams represent the watershed lines which separate the different catchment basins. In rainfalls simulation [39],

drops of water fall on the surface. These drops start flowing down, under gravity, using the path of steepest descent and get collected in the catchment basins. Thus, pixels in the image can be classified under unique catchment basins. The catchment basins represent the regions in the image and the watershed lines are the region boundaries.

Some example applications of region-based methods are [1, 12, 39, 55]. Cheevasuvit *et al* [12] use the split-merge scheme for segmentation. Adams and Bischof [1] discuss a seeded region growing scheme in which high-level knowledge has been incorporated into the process through the choice of seed pixels. Rettmann *et al* [55] use the watershed algorithm for automatic segmentation of cortical sulci. Mangan and Whitaker [39] apply the watershed algorithm to segment surface meshes in 3D.

### Combination Methods

Some methods use a combination of two or more of the above discussed types. Examples are [2, 9, 10]. Beveridge [9] integrates histogram analysis with region merging techniques in his work. Borges and Aldon [10] use a fuzzy clustering algorithm in a split-merge framework. Amoroso *et al* [2] apply a combination of thresholding and neural network learning for color image analysis.

#### 1.3.2 Discussion

Edge detection is used by biological visual systems [18] and hence boundary-based methods are probably the most intuitive of all segmentation methods. However, they are highly sensitive to noise and to small variations of the boundary. Pixel-based methods are generally more

immune to noise than boundary-based methods and they produce closed boundaries directly. The thresholding scheme is simple and can be efficient but works well only on simple images with the right choice of thresholds. Similarly, the choice of features and number of classes [16] is very important in the case of clustering methods. Apart from these, one major disadvantage with pixel-based methods is that most of these techniques do not use *proximity* or *spatial adjacency* information and hence valuable information is lost. For these reasons, the applicability of pixel-based methods for segmenting complex scenes is limited. Like pixel-based techniques, region-based methods are also more tolerant of noise when compared with boundary-based methods. Unlike pixel-based methods, they are based on the adjacency of pixels in the image domain. Region-based methods are compatible with semantic methods [62] and implicitly produce a hierarchy of regions. This makes them more conducive to automating segmentation than other methods. Like some boundary detection and iterative clustering methods, region-based schemes often pose the problem of increased cost of computation and memory requirements. Some of the individual region-based methods pose additional difficulties. Region growing works well only if the initial seeds are representative of the regions of interest. The choice of the homogeneity and stopping criteria is crucial to the success of these methods and depends on the nature of the input image. These problems are overcome in the watershed algorithm. The watershed algorithm uses only an edge map as input and hence can be used to segment a variety of images. The algorithm produces the segmentation result without any user intervention. It is suitable for distributed implementation and can produce significant system optimization. The watershed algorithm produces closed boundaries and can be easily

extended to segment higher-dimensional images. Like many other methods, the watershed algorithm is sensitive to noise. However, unlike other methods, which typically produce incorrect or displaced boundaries in the presence of noise, the watershed algorithm usually produces extra boundaries. This is referred to as  *oversegmentation*, which means that apart from the *real* boundaries, the algorithm also produces spurious boundaries due to noise. This problem is related to the unstable nature of the algorithm. Even small changes in the edge map values can reroute the flow of water producing different watersheds. However, this problem can be removed by pre-processing the image to reduce noise and using a good post-merging scheme. This can make the watershed algorithm robust and if combined with the right merging scheme, it is a good choice for automatic and semi-automatic segmentation problems.

## 1.4 Summary of Work Done

We have implemented a semi-automatic segmentation scheme using a combination of the watershed algorithm and two region merging schemes. We pre-process the input image to reduce the level of noise. We use the rainfalls approach of the watershed algorithm to obtain the initial segmentation of the image. We then process the watershed result using two region merging schemes, namely implicit and seeded region merging, to reduce oversegmentation. Both the merging techniques produce a hierarchy of regions, from which a meaningful output can be easily chosen. We have also developed a graphical user interface (GUI) to view the hierarchy. We have extended the initial segmentation algorithm based on the rainfalls simulation to streaming data in 2D. The streaming algorithm proposed is also suitable for 3D images and for

parallel computation, and produces the same result as the serial watershed algorithm.

# Chapter 2

## Related Work

### 2.1 Introduction

The watershed algorithm is based on concepts from topography. The input to the algorithm is the gradient image or an equivalent graded edge map. The algorithm visualizes the edge map as a topographical surface where the grayscale value at each pixel location represents the elevation. Let  $I$  be the original image and the grayscale value at a location  $(u, v)$  of the edge map be  $f(u, v)$ . Then the corresponding surface is given by

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} u \\ v \\ f(u, v) \end{pmatrix}$$

For instance, if the edge map looks as in Figure 2.1(a), the algorithm sees this input as

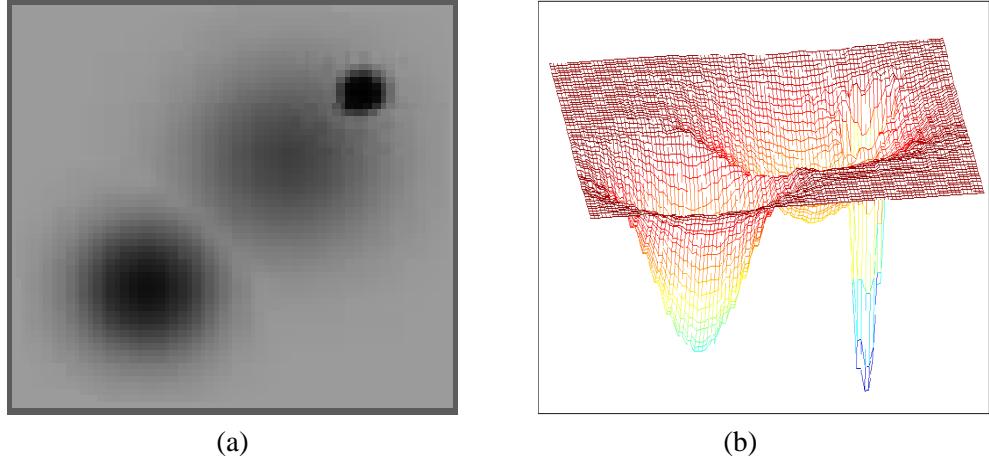


Figure 2.1: An example of topographical surface: (a) Input  $f(u, v)$  (b) Its topographical surface in 3D

the 3D topographical surface shown in Figure 2.1(b). We embed this surface in a gravity field which points in the negative  $z$  direction. A drop of water placed anywhere on this surface will follow the path of steepest descent until it reaches a minimum. This idea helps to establish an equivalence relationship among pixels that trace to the same local minimum and is used to group pixels in the image under different catchment basins. These basins are analogous to *regions*. Thus, the algorithm works by finding the minima of the surface, which correspond to the catchment basins and tries to group every other pixel under one of these basins, producing a segmented output. Consider the example in Figures 2.2(a)–(c). Figure 2.2(a) shows the original 1D function  $I(x)$ . The gradient  $f(x)$ , of this function is shown in Figure 2.2(b), which has three minima. When  $f(x)$  is used as input to the watershed algorithm, the three minima of  $f(x)$  will produce three segments in the result of the watershed algorithm. This concept extends to higher-dimensions and images containing multiple objects.

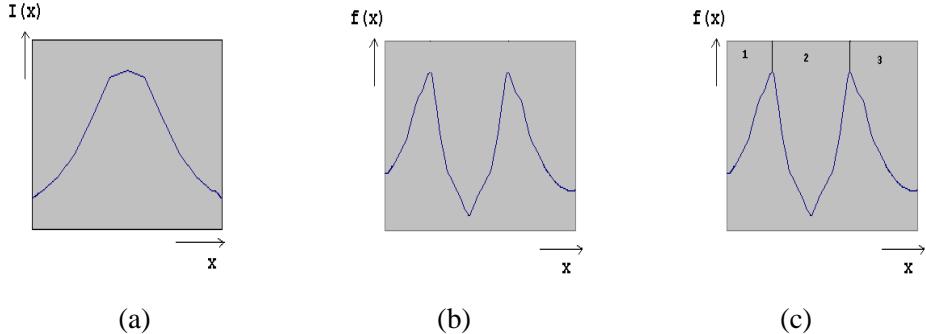


Figure 2.2: Working of the watershed algorithm: (a) Original 1D function  $I(x)$  (b) Gradient of  $I(x)$  —  $f(x)$  is the input to the watershed algorithm (c) Result of watershed algorithm

Watershed segmentation produces closed region boundaries. It is suitable for distributed computation and can easily be extended to higher dimensions. However, like many other segmentation methods, the watershed algorithm is sensitive to noise. Noise in the image can lead to false minima, each of which can create its own region in the output segmentation. This results in oversegmentation, which means that the output has too many regions. Many methods have been proposed in the literature to implement watershed segmentation and to reduce oversegmentation. In the following sections, we explore the origin of the watershed algorithm, the different approaches to implement the algorithm and the strategies used to reduce the effects of noise.

## 2.2 Early Work on Watershed Segmentation

Beucher and Lantuejoul [7] were the first to propose an algorithm based on topographical watersheds to segment images. However, even prior to this, related work had been done, specifically

in the area of topography. For instance, Cayley [11] presents the idea of viewing a topographical surface as a system of contours and slope lines in his article. Maxwell [40] discusses grouping points according to surface minima — the basic principle behind watershed segmentation. Most of this work has been done for processing digital elevation models [14]. Examples of these are [13, 52]. Peucker and Douglas [52] discuss the detection of ravines and pits on a discrete grid which is analogous to finding maxima and minima [36] for watershed segmentation. The algorithm proposed by Collins [13] uses sorted elevation data for processing, very much like the immersion scheme proposed by Vincent and Soille [66].

## 2.3 Different Approaches to Watershed Segmentation

In general, two different algorithms are used to implement watershed segmentation, namely, immersion and rainfall simulation. Each of these can be used to detect the segments in the image either directly or using morphological operators [56]. Again, the implementation can be sequential or distributed. We briefly review some of these approaches as follows.

### 2.3.1 Immersion or Flooding Simulation

The concept of immersion simulation can be described as follows. Holes are pierced at the minima of the surface and the whole surface is slowly immersed in water. The water rises in through these holes and gets collected in the catchment basins. When the water from one basin starts to pour out into an adjacent one, a dam is built to prevent this overflow. The dams or *watershed lines* separate the catchment basins from one another and correspond to the boundaries in the

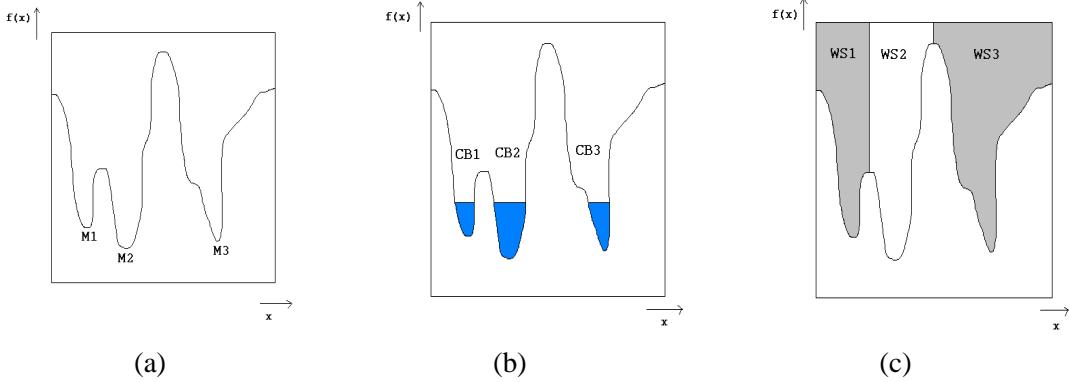


Figure 2.3: Immersion simulation: (a) An example 1D gradient function (b) Flooding by immersion (c) Segmented result

image. This process can be simulated using a computer to find the watershed segmentation.

Immersion simulation is illustrated in Figures 2.3(a)–(c). Figure 2.3(a) shows a 1D gradient function  $f(x)$  with three minima, namely,  $M_1$ ,  $M_2$ , and  $M_3$ . Water rises in and fills the corresponding catchment basins  $CB_1$ ,  $CB_2$ , and  $CB_3$ , as in Figure 2.3(b). When water in  $CB_1$  and  $CB_2$  begin to merge, a dam is built to prevent this overflow of water. Similarly, the other watershed lines are constructed. The final segmented result containing three segments,  $WS_1$ ,  $WS_2$ , and  $WS_3$ , is shown in Figure 2.3(c). Soille and Vincent [60] introduced the immersion approach. Vincent and Soille [66] also introduced an efficient immersion algorithm based on sorting of pixel values and FIFO queues. Variations on their technique were proposed by Beucher and Meyer [8]. Comparative analyses of different versions of the flooding approach were presented in [15, 22].

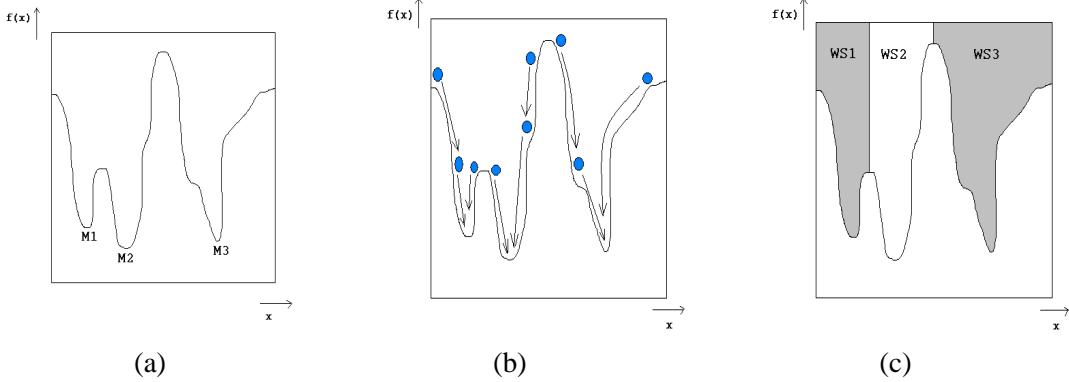


Figure 2.4: Rainfalling simulation: (a) An example 1D gradient function (b) Rainfalling simulation (c) Segmented result

### 2.3.2 Rainfalling Simulation

A conceptual description of the rainfalling simulation approach is as follows. Drops of water are allowed to fall on the surface. A drop falling on a pixel on the surface follows the path of steepest descent until it reaches a minimum. If the steepest descent paths from a set of pixels end in the same minimum, then these pixels define a catchment basin. Thus, every surface minimum can be associated with a catchment basin which corresponds to a region in the image. This idea can be used to compute the watershed segmentation of an image.

Figures 2.4(a)–(c) illustrate rainfalling simulation. The three minima of the gradient function  $f(x)$  lead to three regions in the segmented output, as in Figure 2.4(c). In the earliest work on watershed segmentation, Beucher and Lantuejoul [7] describe rainfalling simulation. The use of this approach for segmentation is also found in [5, 43, 44, 59].

### 2.3.3 Morphological Watershed Segmentation

The watershed segmentation technique has its origins in the area of mathematical morphology [56]. The algorithm borrows the idea of visualizing a nD image as a (n+1)D topographical surface from this field. Much of the work in the literature on watershed segmentation use morphological watersheds. Examples of these are [5, 7, 8, 60]. Beucher and Meyer [8] present a comprehensive review of the various morphological tools used in relation to watershed segmentation. They start with the simplest case of applying dilation and erosion to calculate the gradient image, which typically serves as the input to the watershed algorithm. They proceed to introduce more complex operators used in connection with the algorithm. This includes the application of geodesic reconstruction [8] to detect the extrema in an image and also to detect watershed lines. Beucher [5] describes the relation between watershed segmentation and morphological thinning [56].

We briefly discuss two main approaches found in the literature for segmentation using morphological watersheds as follows. The first approach works by iteratively finding the catchment basins at every elevation of the topographical surface. Let  $f(u, v)$  be the input edge map. Let  $Z_i(f)$  be a set of image points such that  $Z_i(f) = \{(u, v) : f(u, v) \leq i\}$  [8]. Thus,  $Z_i(f)$  is a cross-section of the topographical surface represented by  $f(u, v)$  at elevation  $i$ . Let us flood this surface by immersion. Assume that the flood has reached the elevation  $i$ . Flooding of  $Z_{i+1}(f)$  will occur at those pixels which lie in the geodesic zones of influence [8] of the connected components of  $Z_i(f)$  and the new minima pixels at elevation  $i + 1$ . The new minima at elevation  $i + 1$  are those which are not at a finite geodesic distance from any of the connected components

at elevation  $i$  and they are simply the difference between the sets  $Z_{i+1}(f)$ , and the geodesic reconstruction of  $Z_{i+1}(f)$  by  $Z_i(f)$ . An iterative technique based on this is used to compute the watershed segmentation of the image. A mathematical formulation for this iterative algorithm can be found in [8]. The second approach works by finding the watershed points at every elevation of the surface. Let  $W_i$  be the set of watershed points at elevation strictly less than  $i$ . Let  $I_i$  be a set of image points such that  $I_i(f) = \{(u, v) : f(u, v) < i\}$ . Assume that  $W_i$  is known. The set difference of  $I_i$  and  $W_i$  contains the catchment basins at elevation less than  $i$ . For a pixel  $(u, v)$  at elevation  $i$ , the geodesic distances from the different catchment basins are computed. If the geodesic distance is the same for two or more basins, then  $(u, v)$  is a watershed point. Thus, all watershed points at elevation  $i$  can be computed. This approach also leads to an iterative method for computing watershed segmentation [7].

### 2.3.4 Distributed Watershed Segmentation

Some applications entail very large images and it may be impossible to provide the required amount of memory or computational power on a single computer. Thus arises the need for a distributed version of the algorithm, in which blocks of the image are segmented individually and the results are combined to obtain the segmentation of the entire image. Such a distributed algorithm may also serve to improve the speed of computing watershed segmentation. Distributed watershed segmentation can be implemented either as a streaming (memory) algorithm or a parallel algorithm (time). The streaming algorithm would use a single processor while the parallel type would involve multiple processors. Designing a distributed watershed algorithm is

a challenging problem because the working of the algorithm is not strictly *local*. Many solutions to this problem have been proposed in the past, most of which discuss parallel implementation. Of these, [41, 42, 45] use immersion simulation and [43, 44] use rainfalls simulation.

### 2.3.5 Discussion

Implementations of immersion simulation found in the literature that are based on FIFO queues [8, 66] are fast versions of the watershed algorithm. For instance, Vincent and Soille [66] propose an initial sorting of pixels and storage of pixel information in queues to enable direct access to pixels in the gradient image. The watershed segmentation is computed iteratively until the queue is empty, where at every iteration two main operations are performed. First, the label of every pixel in the queue is assigned to the non-labeled neighbors of higher elevation or gray-level and these neighbors are put into the queue. Second, new minima are detected, labeled and stored. This algorithm does not perform repeated scannings of the image and hence is efficient in terms of computation speed. However, the first of the two operations discussed above is highly sequential in nature when compared with the working of rainfalls simulation. In the case of the rainfalls approach, every pixel can be traced to a minimum independent of the tracing of other pixels [43] while in the immersion example mentioned above, most pixels get their labels from a previously labeled neighbor. Hence, a distributed version of immersion simulation may need storage of a large amount of information. For a parallel implementation, this also leads to increase in the overhead for communication between processors. For this work, we have used watershed segmentation by rainfalls simulation to implement an efficient

distributed version of the algorithm.

Watershed segmentation using rainfalls simulation can be implemented using operators from morphology or by direct detection of catchment basins and watershed lines. All morphological operations used to compute watershed segmentation can be reduced to a series of operations on the image with a structuring element. There are two problems associated with the use of a structuring element — first, they tend to impart their shape to the boundaries in the image and second, it is important to choose the right shape and size of the structuring element, which in turn, depend on the application. Moreover, watershed segmentation based on morphological operators poses the problem of computational complexity. Therefore, we have used a simple and straightforward implementation of rainfalls simulation by direct detection [36, 39].

## 2.4 Strategies to Prevent Oversegmentation

Like many segmentation methods, the watershed algorithm is sensitive to noise. Noise in the image may lead to false minima, each of which creates a corresponding region in the output. This results in oversegmentation, which is illustrated in Figures 2.5(a)–(c). Figure 2.5(a) shows an image with a small amount of noise. The gradient image is shown in Figure 2.5(b). The result of the watershed algorithm is shown in Figure 2.5(c). In this output, every *real* region has been broken down into two or more regions, producing oversegmentation. Therefore, we need a good strategy to remove the spurious boundaries created due to noise and produce a meaningful segmentation.

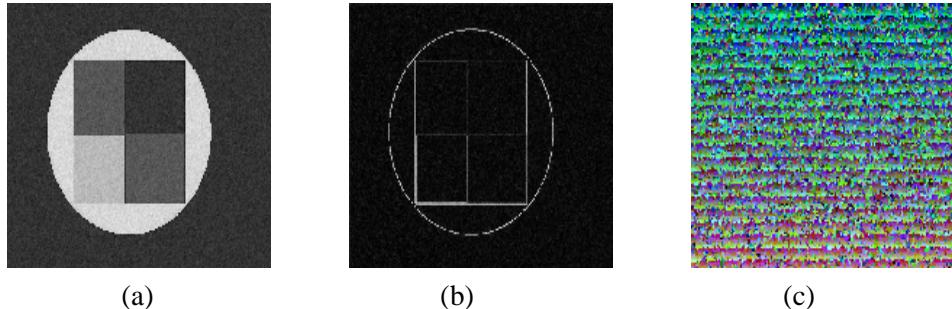


Figure 2.5: An example of oversegmentation: (a) A noisy image (b) Gradient image (c) Watershed result showing oversegmentation

Several methods have been proposed in the literature to reduce the effects of noise. They include pre-processing techniques such as filtering, thresholding, and extraction of markers, and post-processing techniques such as region merging and hierarchical segmentation.

#### 2.4.1 Pre-processing Techniques

##### **Filtering**

Filtering the original image is a common method that has been used to reduce noise and oversegmentation in connection with watershed segmentation. The filter used for smoothing can be linear or non-linear. For instance, Lee *et al* [34] use linear filtering by Gaussian blurring for the segmentation of satellite images to extract roads. Examples of work from the literature that use non-linear smoothing are [27, 49, 58]. De Smet *et al* [58] propose non-linear filtering by anisotropic diffusion. Hernandez and Barner [27] suggest median filtering while Ogor *et al* [49] use morphological opening and closing [56].

## **Thresholding**

Thresholding can be applied to the gradient image to combine several irrelevant minima into a single flat minimum, thus preventing oversegmentation. Figures 2.6(a)–(b) illustrate this. The gradient function  $g(x)$  in Figure 2.6(a) shows how spurious minima — M1,M2,M3,M4 — can result in too many insignificant segments in the output. The modified gradient  $f(x)$  in Figure 2.6(b) shows how thresholding can be used to flatten out the irrelevant minima to create a single minimum M, thus preventing oversegmentation. The choice of the threshold is crucial to this approach. A very low threshold may fail to prevent oversegmentation. On the other hand, a very high threshold may combine two or more *real* regions into a single region in the output producing an effect called *undersegmentation*. In our experience, a good threshold can be easily chosen for a *given* application. Examples of use of the thresholding approach are [4, 37]. Baccar *et al* [4] use 10% of the maximum gradient value as a threshold that works effectively on their range images. Li *et al* [37] propose thresholding of the gradient image to segment Synthetic Aperture Radar (SAR) images.

## **Use of Markers**

Markers are used to denote significant regions in the image by using the minima corresponding to these regions alone in the gradient image, which serves as input to the watershed algorithm. For instance, in the immersion approach, flooding would occur only through the minima indicated by the markers. The user's high-level knowledge can be applied for selecting or marking the sources of flooding. In the case of morphological watershed, the markers are obtained from

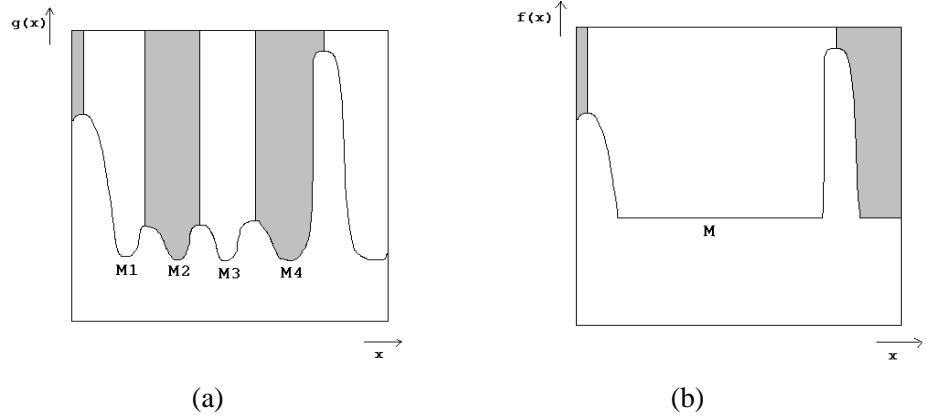


Figure 2.6: Effect of thresholding the gradient: (a) Watershed segmentation of gradient  $g(x)$  containing irrelevant minima (b) Watershed segmentation of the thresholded gradient  $f(x)$

the gradient image by a process called *homotopy modification* [8]. The effect of this process on the gradient is illustrated in Figure 2.7. The marker function samples out the significant minima and flattens out the irrelevant minima into plateaus. Instances of the use of markers are [32, 57]. Shiji and Hamada [57], and Kanai [32] use marker-based watersheds for color image segmentation.

### 2.4.2 Post-processing Techniques

#### Region Merging

Reducing oversegmentation can be posed as a region merging problem. Starting with the result of the watershed segmentation, regions can be merged based on pre-defined criteria until a meaningful segmentation is obtained. Related work found in the literature can be classified into four types based on the criteria used for merging — methods that use features of regions, those

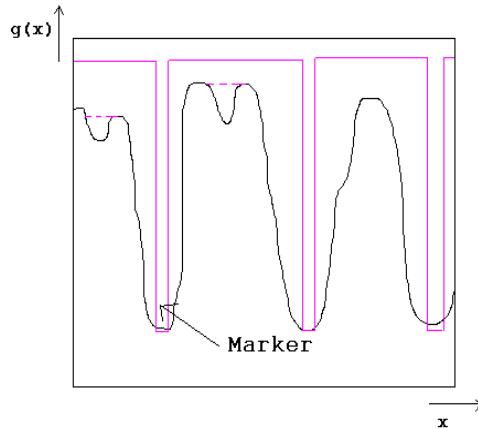


Figure 2.7: Effect of a marker function on the gradient

that characterize boundaries, those that characterize catchment basins based on the topography, and those that use a combination of two or more of the above.

Many methods have been proposed in the past that use region merging based on similarity criteria, which in turn are based on features of regions, like grayscale values, color, texture etc. Examples of such work are [24, 32]. Kanai [32] uses intensity, color and size information from regions in his method. Haris *et al* [24] pose reduction of oversegmentation as a cost-minimization problem that uses average intensities of regions. Properties of regions in the watershed result can be used to extract feature vectors and cluster *similar* regions in a higher-dimensional feature space [16]. Examples of methods that use saliency measures to characterize boundaries in the output of the watershed segmentation are [6, 46]. In Beucher's [6] mosaic image transform method, the watershed line between two catchment basins is characterized by the absolute difference between average intensities of the basins. Region merging can also be based on dynamics of contours of the image [46]. *Dynamics of contours* refers to a saliency

measure of contours in the image and was introduced by Najman and Schmitt [46]. Instances of methods that characterize catchment basins are [35, 36, 55]. The work done by Lemaréchal *et al* [35] is based on basin dynamics, which is a saliency measure of the local minima (catchment basins). Lester [36] uses watershed depth as a saliency measure [36]. Rettmann *et al* [55] use a similar measure for the segmentation of cortical sulci. Instances of combination of the above approaches are [3, 53]. Pratikakis *et al* [53] propose a saliency measure that is based on edge strength and region homogeneity. Andrade *et al* [3] suggest a merging process guided by watershed depth and area of the regions in the output of the watershed algorithm.

#### 2.4.3 Hierarchical Segmentation

Hierarchical segmentation has been widely used in the past to solve the oversegmentation problem. The user can select a meaningful segmentation from the hierarchy of regions generated. This approach also leads to a multi-resolutational watershed segmentation method by which objects at different scales can be extracted from the image. The methods in the literature that discuss hierarchical segmentation can be classified under two major categories — those based on region merging and those that use scale-space representation of images [65]. Most of the work in the literature on region merging in connection with the watershed algorithm deal with hierarchical segmentation. Examples of methods that use scale-space representation are [19, 29, 33, 34, 67]. Jackway [29] suggests convolution of the image with the Gaussian kernel to create a linear scale-space representation of the original image and application of the watershed algorithm at every scale. Gauch [19] proposes the generation of a scale-based hierarchy

by analyzing the minima at different scales. Kim [33] proposes a slightly modified version of Gauch's approach for extraction of roads from satellite images. Wright and Acton [67] use morphological operators to create a scale-space representation of the input image to produce multi-resolutional watershed segmentation.

#### 2.4.4 Discussion

Filtering does help in reducing the noise level in an image. However, filtering alone may not solve the oversegmentation problem for images that are highly noisy and textured. Specifically, Gaussian blurring and morphological filtering have their own disadvantages. Gaussian blurring wipes out structures below a particular scale and this may result in the loss of significant information. This problem can be overcome by the use of edge-preserving non-linear filtering methods like anisotropic diffusion [51]. Morphological filtering works well only with the right choice of the structuring element and this depends on the application. Like filtering, thresholding also may not be a good solution to the problem of oversegmentation if used alone but with a good threshold value, it can serve as a useful pre-processing stage. Unlike filtering or thresholding, the marker-based approach can be used by itself to prevent oversegmentation. Another advantage of marker-based watershed is that the user can employ his high-level knowledge to bring out a meaningful segmentation. However, in some cases [6], marker extraction needs the output of the watershed segmentation and the modified gradient has to be processed again by the watershed algorithm, thus increasing the computation cost. In addition, marker selection for complex images may be a tedious process. The difficulty in using region merging schemes

that use criteria based on region properties is that these techniques are highly dependent on the nature of the input image. We prefer a region merging algorithm that works closely with the concepts behind the watershed algorithm itself so that the method would be generic and can be used on a variety of image types. Such a merging scheme can be one that is based on characterization of catchment basins or watershed lines.

Even with a good region merging scheme, it may be hard to determine a stopping criterion that would always produce a meaningful segmentation. Therefore, hierarchical segmentation, which involves user's high-level knowledge in choosing a meaningful segmentation, is a more effective strategy. Of the hierarchical segmentation methods discussed, those based on scale-space representation may not preserve edges. This includes methods that involve Gaussian blurring and those that use morphological operators. Thus, a region merging scheme based on characterization of basins or watershed lines to produce a hierarchy of regions can solve the oversegmentation problem.

In our work, we have used watershed depth as the saliency measure [36] to characterize catchment basins. The region merging scheme based on this measure can be visualized as an immersion of the segmented image in water. At every level in the hierarchy, catchment basins containing weak edges are merged. This continues until only one region is left, which is the entire image itself. Thus, a hierarchy of regions is generated and a meaningful segmentation can be chosen from the hierarchy. This method, by itself, serves to solve the problem of oversegmentation and is also consistent with our goal to produce partially- or fully-automatic segmentation.

## **Chapter 3**

# **Watershed Segmentation**

### **3.1 Introduction**

We have implemented semi-automatic hierarchical image segmentation using the watershed algorithm. As a pre-processing step, we filter the original image, and threshold the edge map to reduce the number of irrelevant minima. We then obtain the watershed segmentation by an implementation of the rainfall simulation approach. As a solution to the oversegmentation problem, we have used a region merging post-processing stage, in which regions in the segmented result are merged based on watershed depth [36]. This step produces hierarchical segmentation and the user can select a meaningful result from the hierarchy of regions generated. In the rest of this chapter, we present the details of the algorithms and their implementation.

## 3.2 Pre-processing Step

We lowpass filter the original image using the Gaussian function to reduce the level of noise.

We use the gradient of the image as the edge map that serves as input to the algorithm. We compute the gradient image for 2D and 3D, grayscale and color images as described below.

Let  $I$  denote the original image and  $g$  denote the gradient image. If  $I$  is gray-valued, we use Equation 3.1 to compute  $g$ . There are methods in the literature that discuss computation of gradient of color images. For instance, Zenzo [70] suggests using the magnitude of the gradient along the direction of maximum change. Finding the gradient of color images, however, is not a focus of this work. Hence, we have simply used the RMS value of the gradients of the color components  $I_R$ ,  $I_G$ , and  $I_B$  in the  $x$  and  $y$  directions to compute  $g$ . This is done by using Equations 3.2 and 3.3 in Equation 3.1.

$$g = \sqrt{\left(\frac{\delta I}{\delta x}\right)^2 + \left(\frac{\delta I}{\delta y}\right)^2} \quad (3.1)$$

$$\frac{\delta I}{\delta x} = \sqrt{\frac{(\frac{\delta I_R}{\delta x})^2 + (\frac{\delta I_G}{\delta x})^2 + (\frac{\delta I_B}{\delta x})^2}{3}} \quad (3.2)$$

$$\frac{\delta I}{\delta y} = \sqrt{\frac{(\frac{\delta I_R}{\delta y})^2 + (\frac{\delta I_G}{\delta y})^2 + (\frac{\delta I_B}{\delta y})^2}{3}} \quad (3.3)$$

We compute the gradient of a 3D grayscale image using Equation 3.4. To find the gradient of a 3D color image, we use Equations 3.5, 3.6 and 3.7 in Equation 3.4.

$$g = \sqrt{\left(\frac{\delta I}{\delta x}\right)^2 + \left(\frac{\delta I}{\delta y}\right)^2 + \left(\frac{\delta I}{\delta z}\right)^2} \quad (3.4)$$

$$\frac{\delta I}{\delta x} = \sqrt{\frac{(\frac{\delta I_R}{\delta x})^2 + (\frac{\delta I_G}{\delta x})^2 + (\frac{\delta I_B}{\delta x})^2}{3}} \quad (3.5)$$

$$\frac{\delta I}{\delta y} = \sqrt{\frac{(\frac{\delta I_R}{\delta y})^2 + (\frac{\delta I_G}{\delta y})^2 + (\frac{\delta I_B}{\delta y})^2}{3}} \quad (3.6)$$

$$\frac{\delta I}{\delta z} = \sqrt{\frac{(\frac{\delta I_R}{\delta z})^2 + (\frac{\delta I_G}{\delta z})^2 + (\frac{\delta I_B}{\delta z})^2}{3}} \quad (3.7)$$

We subject the gradient image  $g$  to thresholding. If  $T$  denotes the threshold as a % of the RMS gradient, we obtain the modified gradient value  $f$  for every pixel  $p$  using Equation 3.8.

$$f(p) = \max(g(p), T) \quad (3.8)$$

The thresholding step helps to flatten out irrelevant minima whose gradient values are less than  $T$ , forming flat regions of value  $T$ . This pre-processing stage is not required but is highly recommended to reduce the amount of oversegmentation in the output of the watershed algorithm.

### 3.3 Watershed Segmentation Algorithm

We have implemented watershed segmentation using rainfalls simulation [36, 39]. To describe our implementation, we first define terms that are required to understand the working of the algorithm. We then discuss in detail our discrete implementation of watershed segmentation by rainfalls simulation.

#### 3.3.1 Definitions

Depending on the values of its neighbors, every pixel on the topographical surface could be a *single-pixel minimum* or part of a *flat region*. We define these as follows.

##### Single-Pixel Minimum

A pixel whose value is strictly less than the value at all neighboring pixels is a single-pixel minimum. By neighboring pixels, we refer to the 8-neighbors of a pixel in 2D and 26-neighbors in 3D [20]. For instance, in Figure 3.1(a),  $P$  is a single-pixel minimum.

##### Flat Region

A pixel is said to be part of a flat region if its value is equal to the value of at least one of its neighboring pixels. Flat regions can be classified into three types, namely, flat maxima, flat plateaus, and flat minima. If the values of pixels in the neighborhood of a flat region are strictly less than the values inside the flat region, the flat region is called a flat maximum. If the values of pixels in the neighborhood of a flat region are strictly greater than the values inside the region,

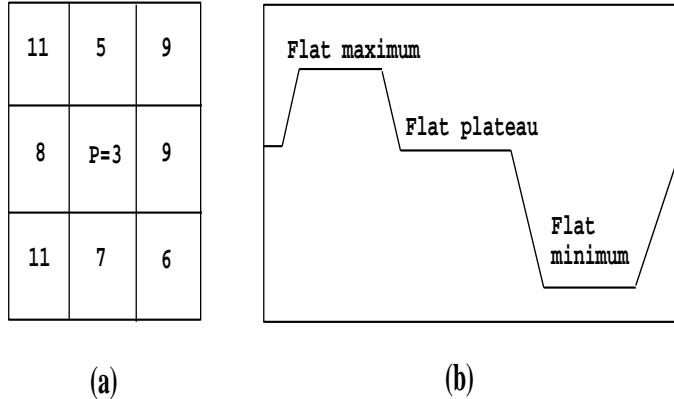


Figure 3.1: Classification of pixels on the topographical surface: (a) Single-pixel minimum (b) Types of flat regions

the flat region is termed a flat minimum. If the values of pixels in the neighborhood are such that some are greater than and some are less than the values inside the region, the flat region is a flat plateau. These three types are illustrated in Figure 3.1(b).

### 3.3.2 Rainfalling Simulation Algorithm

Watershed segmentation by rainfalling simulation is obtained using the following two major steps:

#### Searching

We construct a one-pixel wide wall around the surface and set the height of this wall to a value higher than the maximum value in the image. This step is only for programming convenience.

It does not overwrite on the image and does not prevent the detection of minima along the boundary. In a single width-first scan of the image, we detect and label single-pixel minima and flat regions, and also find the pixel with the least value along the boundary of each of the flat regions<sup>1</sup>. In finding the boundary minimum of a flat region, we may come across more than one boundary pixel with the same value. In such a case, we choose to use the minimum that is detected first while scanning the image. At the end of this step, the single-pixel minima and flat regions in the image carry unique labels.

## Tracing

In the tracing step, we first trace the non-minimum and non-flat pixels to their respective minima using the path of steepest descent. For such a pixel  $P$ , we start tracing by finding the minimum of  $P$  in its 8-neighborhood (26-neighborhood in 3D). In this process, we may come across more than one pixel in the neighborhood of  $P$  carrying the same value. If the equal-valued pixels are connected, then we have reached a flat region (plateau or minimum) and we terminate the tracing of  $P$  at the flat region. If the equal-valued pixels are not connected, then we use the one which is detected first. We then move to the minimum in the 8-neighborhood of  $P$  and continue this tracing as described above until a single-pixel minimum or flat region is reached. We label all the pixels along this path of steepest descent using the label of the single-pixel minimum or flat region where the tracing ends. Similarly, we trace flat maxima and plateaus to their respective minima. For each of the flat maxima and plateaus, we begin the tracing at the boundary minimum pixel. Here again, in each trace, we assign the label of the single-pixel

---

<sup>1</sup>Henceforth, we refer to the minimum along the boundary of a region as *boundary minimum*.

minimum or flat region reached to all the pixels along the path of steepest descent. We then adjust the labels in the image so that the labels range from 0 to  $R - 1$  where  $R$  is the number of regions in the segmented output. At the end of this step, every *segment* in the image is labeled with a unique number.

## 3.4 Hierarchical Segmentation by Region Merging

The watershed algorithm typically produces oversegmentation in the presence of noise. We propose hierarchical segmentation by region merging as a solution to the oversegmentation problem. The region merging algorithm is based on characterization of the catchment basins in the output of the watershed algorithm using watershed depth [36]. For a given region in the segmented result, watershed depth is defined as the difference in value between the boundary minimum of the region and the local minimum of the region. For instance, consider the depth  $d$  shown in Figure 3.2. This measure can be used to continuously merge regions, until all the regions are merged into one, thus producing a hierarchy. Based on the way in which the watershed depth measure is used for merging, the region merging scheme can be classified into two types — implicit region merging and seeded region merging.

### 3.4.1 Implicit Region Merging

The process of implicit region merging can be visualized as an immersion of the segmented image in water in such a way that every region is flooded simultaneously. Shallow regions merge first followed by deeper ones until all the regions are merged into a single region. The

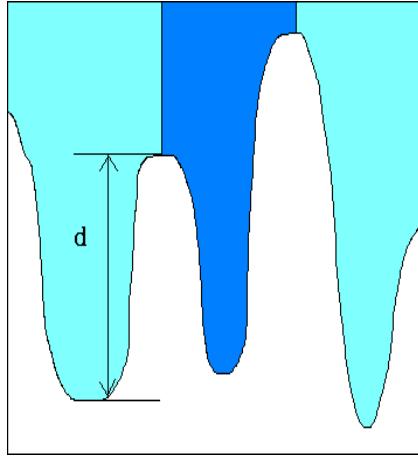


Figure 3.2: Watershed depth of a region.

concept of implicit region merging is illustrated in Figure 3.3. Figure 3.3(a) shows the segments in the output of the watershed algorithm and the initial watershed depth for each region. The region with the lowest watershed depth of 4 has weak edges compared to the other regions in the image and hence may have been created by an irrelevant minimum. Therefore, it is merged with the neighbor corresponding to the watershed depth, forming a new region. Here, a *neighbor* of a region  $r$  refers to a set of pixels which form a region in the result of watershed segmentation and are spatially adjacent to  $r$ . The watershed depth measure is updated for this new region and its neighbors. The result of this is the image shown in Figure 3.3(b). The region with the lowest watershed depth of 5 in this image is merged with its corresponding neighbor. The above process occurs iteratively until all the regions are merged. The hierarchy of regions generated is shown in Figure 3.3. Thus, implicit merging automatically produces a hierarchy of regions based on watershed depth.

Consider the set of regions  $C = \{c_1, \dots, c_R\}$  in the result of the watershed algorithm. Let

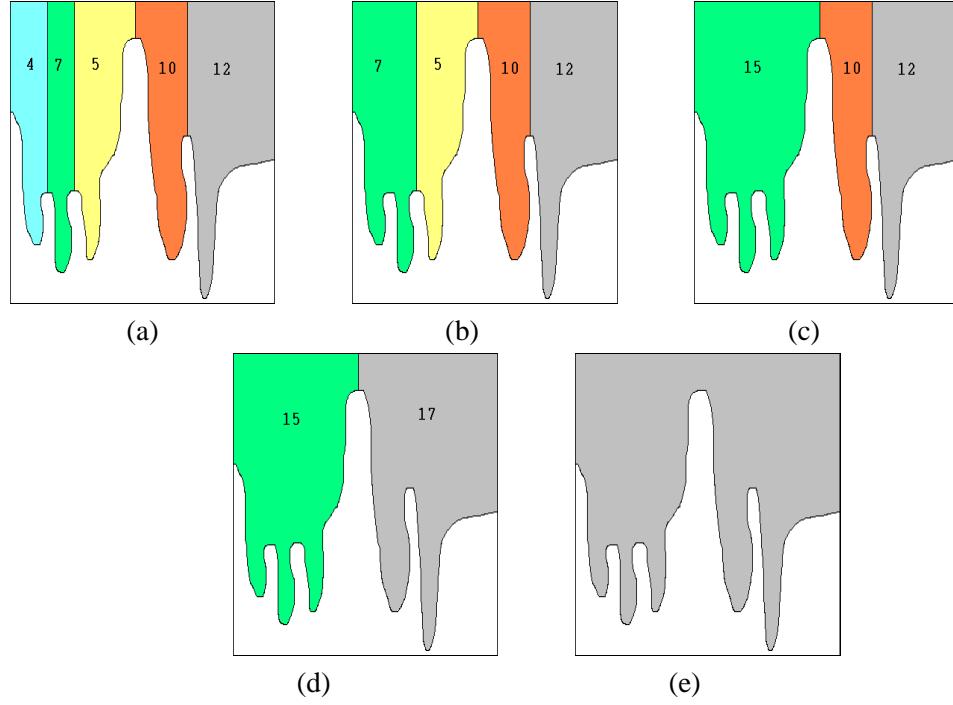


Figure 3.3: Implicit region merging

the set of neighbors of a region  $c_i$  be  $N_i = \{c_{i,1}, \dots, c_{i,n_i}\}$ . Let  $m_{i,j}$  denote the difference between the minimum of region  $c_i$  along the boundary with  $c_{i,j}$ , and the local minimum of  $c_i$ . Let  $m_i = m_{i,d}$  denote the watershed depth of  $c_i$  and  $c_{i,d}$  denote the corresponding neighbor, where  $m_{i,d} = \min\{m_{i,1}, \dots, m_{i,n_i}\}$ . Let  $N$  denote the number of regions in the image. The implicit merging algorithm is as follows.

1. Initialize  $N = R$ .
2. Find the region  $c_i$  for which the value of  $m_i$  is the smallest.
3. Merge  $c_i$  with  $c_{i,d}$  and update the boundary minima of  $c_i \cup c_{i,d}$  and its neighbors, and the local minimum of  $c_i \cup c_{i,d}$ . Decrement  $N$  by 1.

4. If  $N = 1$ , the merging is complete. If not, go to step 5.

5. Recompute  $m_i$  for the regions. Go to step 2.

The result of the above algorithm is a hierarchy of regions. The lowest level in the hierarchy contains the regions from the result of watershed segmentation and the highest level contains one region which is the entire image itself. Typically, merging of two neighboring regions at one level corresponds to the next higher level in the hierarchy. However, at any level in the above merging process, we may come across more than one region with the smallest value of watershed depth. In this case, the merging of these regions with their corresponding neighbors will occur simultaneously.

### 3.4.2 Seeded Region Merging

In the case of seeded region merging, the seed region which initiates the merging is chosen by the user. The concept of seeded region merging is illustrated in Figure 3.4. Figure 3.4(a) shows the regions from the result of watershed segmentation and the initial watershed depths of these regions. The region in Figure 3.4(a) with initial watershed depth of 5 is chosen as the seed. This seed region is merged with the neighbor corresponding to the watershed depth forming a new region. The watershed depth of this new region is updated. We again merge this new region with the neighbor corresponding to the watershed depth. The above occurs iteratively until the seed is merged with all other regions in the image. The hierarchy generated is shown in Figure 3.4. The hierarchy produced by seeded region merging represents a growing of the seed from the watershed result.

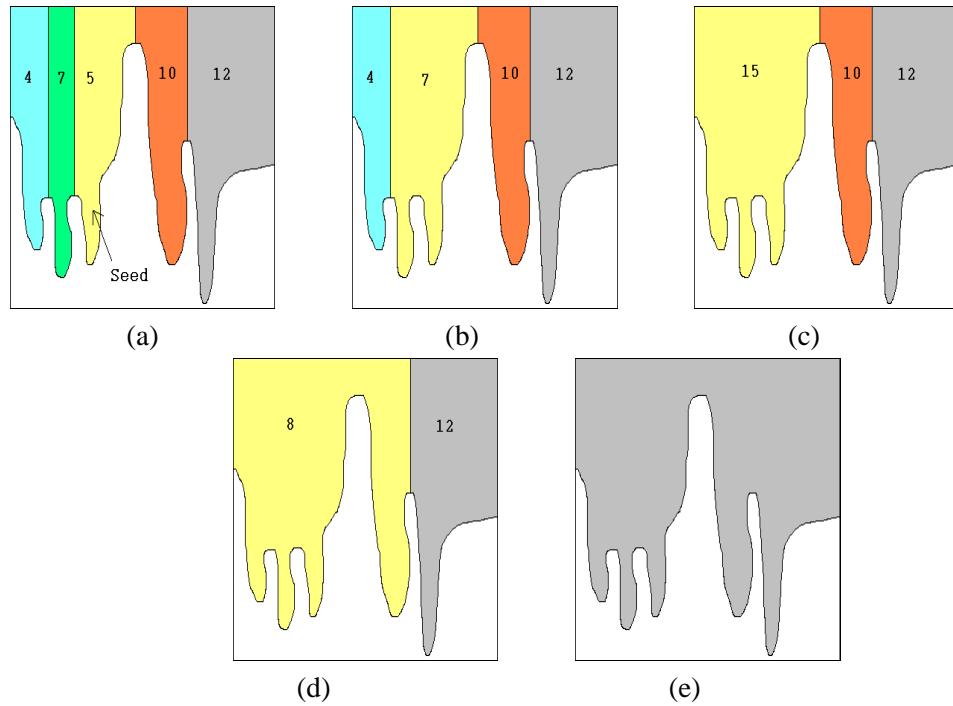


Figure 3.4: Seeded region merging

Seeded region merging is different from implicit region merging in that, in the implicit case, the seed is implicit and at every level of merging, it is the region with the lowest watershed depth. In seeded merging, the seed at every level is the same and is the region chosen by the user. Therefore, the hierarchy of regions generated by seeded merging may be different from the one generated by implicit merging. For instance, compare the hierarchies in Figures 3.3 and 3.4.

Consider the seed region  $c_s$  chosen by the user. Let the set of its neighbors be  $N_s = \{c_{s,1}, \dots, c_{s,n_s}\}$ . Let  $b_{s,j}$  be the minimum of region  $c_s$  along the boundary with  $c_{s,j}$ . Note that the neighbor corresponding to the watershed depth of  $c_s$  is the one corresponding to the smallest

value of  $b_{s,j}$ . Let the smallest of the boundary minima be  $b_{s,d} = \min\{b_{s,1}, \dots, b_{s,n_s}\}$  and let  $c_{s,d}$  be the corresponding neighbor. Let  $N$  denote the number of regions in the image. The seeded merging algorithm is as follows.

1. Initialize  $N = R$ .
2. Find the neighbor  $c_{s,d}$  corresponding to the smallest value  $b_{s,d}$ .
3. Merge  $c_{s,d}$  with  $c_s$  and update the boundary minima of  $c_{s,d} \cup c_s$ . This step results in a new set of  $b_{s,j}$  values. Decrement  $N$  by 1.
4. If  $N = 1$ , then the merging is complete. If not, go to step 2.

The result of this merging, again, is a hierarchy of regions. The lowest level in the hierarchy contains the regions in the result of the watershed algorithm and the highest level contains a single region, which is the entire image. At any level in the hierarchy, merging of the seed region  $c_s$  with the neighbor  $c_{s,d}$  corresponding to its watershed depth produces the next higher level in the hierarchy. However, we may come across more than one neighbor  $c_{s,d}$  for the same value of watershed depth. In such a case, all these neighbors are merged with  $c_s$  simultaneously.

### 3.5 Watershed Segmentation Tool

We have developed a graphical user interface (GUI) for ease in viewing the hierarchy. We have interfaced the programs that implement watershed segmentation and the region merging algorithms with windowing functions in the toolkit FLTK [63]. Figure 3.5 shows a snapshot of

the tool for 2D images. The functions available in the user interface for 2D images are described below.

1. *Input image*: This reads in a gray-valued or color image in 2D from file and displays it.
2. *Gradient*: This computes the gradient of grayscale and color images using derivative functions in VISPack as discussed in Section 3.2.
3. *WS*: This filters the image, lets the user select an appropriate threshold for the gradient image and computes the watershed segmentation. The *gauss* function in VISPack has been used for filtering. The default threshold is equal to 10% of the RMS value in the gradient image. A grayscale-to-color mapping function has been used to show different segments in the output using different colors.
4. *Merge*: Merging can be done using one of the two methods described earlier. Both the merging algorithms use the heap sort technique. The hierarchy of regions can be viewed and the image at any level in the hierarchy can be saved to a file.
5. *WriteHR*: This saves the entire hierarchy to two files. The result of watershed segmentation is saved in a *.ws* file and the list of region pairs merged at every level in the hierarchy is stored in a *.hr* file. The entire hierarchy can be reconstructed using these two files.
6. *ReadHR*: This allows the user to read in and view a hierarchy which has been saved as a combination of the *.ws* and *.hr* files.

We have also extended the user interface for 3D images. Figure 3.6 shows a snapshot of the tool for 3D images. The additional features available are:

1. *Load*: This reads in a set of slices in 3D from files and displays them.
2. *Gradient*: This computes the gradient of 3D grayscale and color images as discussed in Section 3.2.

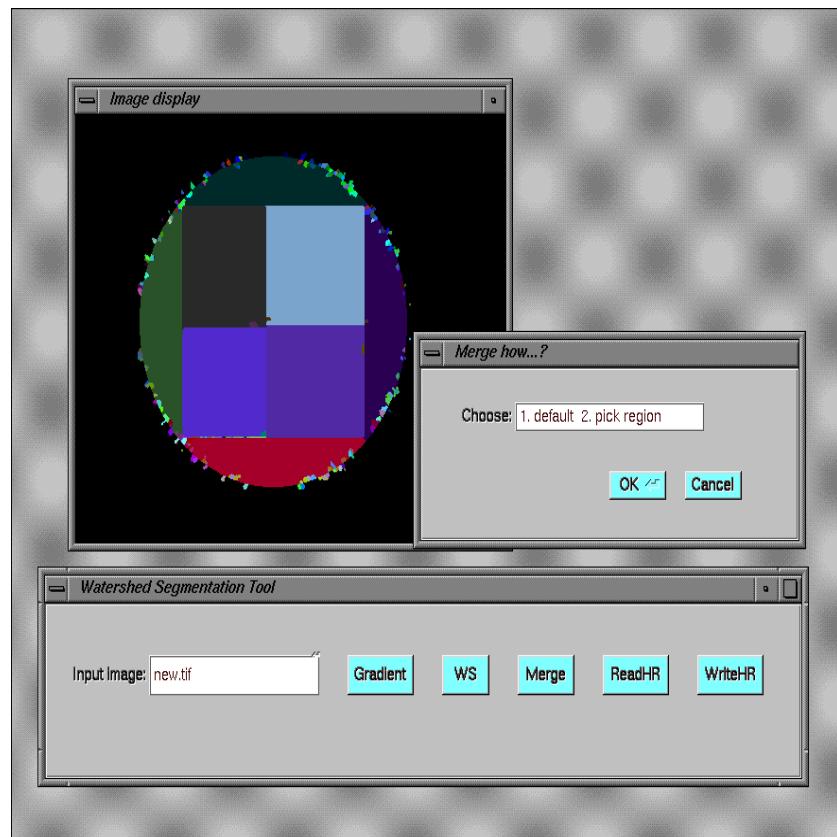


Figure 3.5: Snapshot of the user interface for 2D images

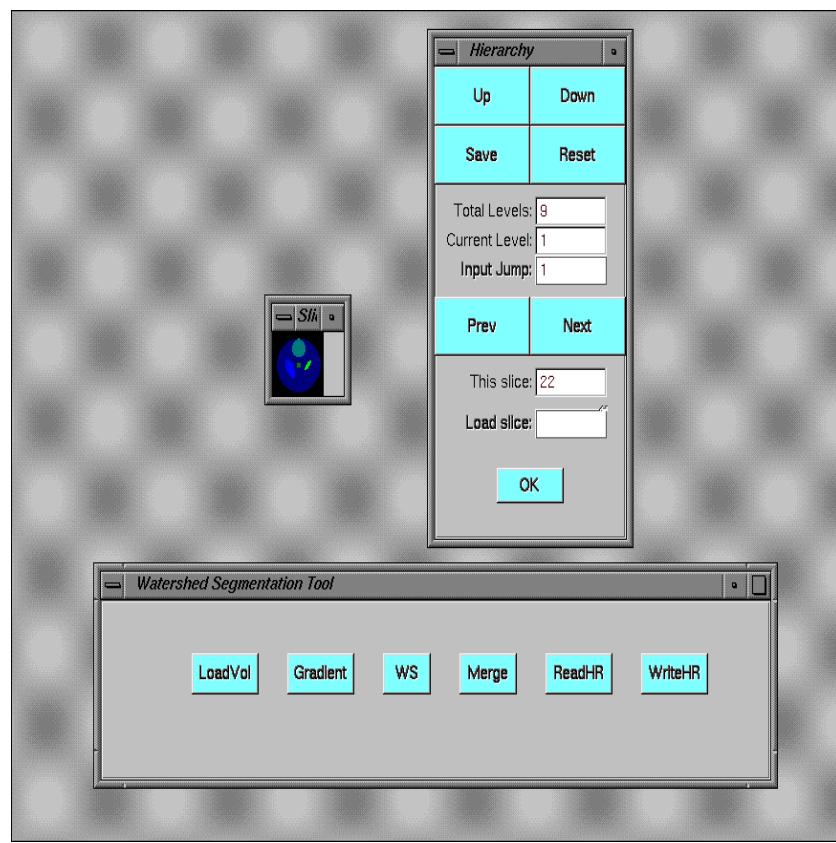


Figure 3.6: Snapshot of the user interface extended for 3D images

## **Chapter 4**

# **Streaming Algorithm**

### **4.1 Introduction**

In some applications, the amount of data to be segmented may be so large that it is difficult to provide the required amount of memory on a single computer. Thus, the need arises for a distributed implementation of the segmentation algorithm. The distributed algorithm can be implemented either as a streaming algorithm or as a parallel algorithm. In a distributed segmentation scheme, the image is divided into blocks and the segmentation of each block is computed individually. The result of segmentation from the blocks are then used to produce the segmentation of the entire image. In the case of a streaming algorithm, all of the blocks are processed by a single processor while in a parallel algorithm, blocks are processed by multiple processors. Designing a distributed version of the watershed algorithm is a challenging problem because the searching and tracing steps described in Section 3.3 cannot be *completed* for any one block due to the lack of information about the neighbors of the border pixels of the blocks. Most of

the work in the literature that discusses distributed versions of the watershed algorithm deals with parallel implementation. Examples of such work are [42, 43, 45]. Moga *et al* [42] propose a parallel implementation based on Meyer’s immersion algorithm [8]. They divide the image into blocks with overlap and distribute them to different processors. One master processor supervises the communication among other processors. In their approach, the pixels in each block are first classified as minima and non-minima. The processors then communicate with one another to compare this local classification in each block and reclassify pixels as minima and non-minima. The minima in each block are then labeled. The processors communicate again to make sure that the *same* set of pixels classified as minima are assigned the same label in different blocks. Flooding is done within each block followed by communication among the adjacent processors to continue the flooding across the boundaries. Moga *et al* [43] propose a parallel algorithm based on rainfalls simulation. In rainfalls simulation, the tracing of every pixel using the steepest descent path to a minimum can be done independent of the tracing of other pixels. Hence they have described this approach as being better suited for distributed implementation than the immersion approach. Here again, the image is divided into blocks with overlapping borders. Labeling and tracing of pixels occur within each block. The blocks are then allowed to communicate iteratively to exchange labels until there are no unresolved paths. Moga *et al* [45] pose the problem of distributed computation of watershed segmentation as a connected-component labeling problem and have implemented a parallel algorithm based on the immersion approach. They propose the use of local connectivity graphs of labels to keep track of connected components (split flat regions and steepest descent paths) in adjacent blocks.

A master processor then combines these local graphs to produce a global graph from which the final segmentation is generated.

The above-mentioned work relies on iterative communication among adjacent blocks to obtain the final watershed segmentation. In our work, we have designed and tested a streaming algorithm based on rainfalls simulation which produces watershed segmentation without the need to iterate. In the rest of this chapter, we discuss the details of this streaming algorithm and the extensibility of the algorithm to efficient parallel implementations.

## 4.2 Details of the Streaming Algorithm

In the streaming algorithm, blocks of the image are segmented individually by a single processor and the results from the blocks are used to obtain the segmentation of the entire image. Problems arise while computing the watershed segmentation of each block because the searching and tracing steps described in Section 3.3 cannot be completed *locally* within a block. Two major problems are as follows:

1. Incorrect identification of the topography of border pixels in a block: The splitting of the image may result in incorrect identification of the nature of the topographical surface at the border pixels of the blocks. For instance, consider pixel  $P$  in Figures 4.1(a)–(b).  $P$  in the image in Figure 4.1(a) is not a single-pixel minimum. However, when the image is split into two and segmented individually,  $P$  will be classified as a single-pixel minimum in block 1 as in Figure 4.1(b). This is because complete information about the neighbors of  $P$  is not available in block 1. Hence steepest descent paths that trace to  $P$  will be

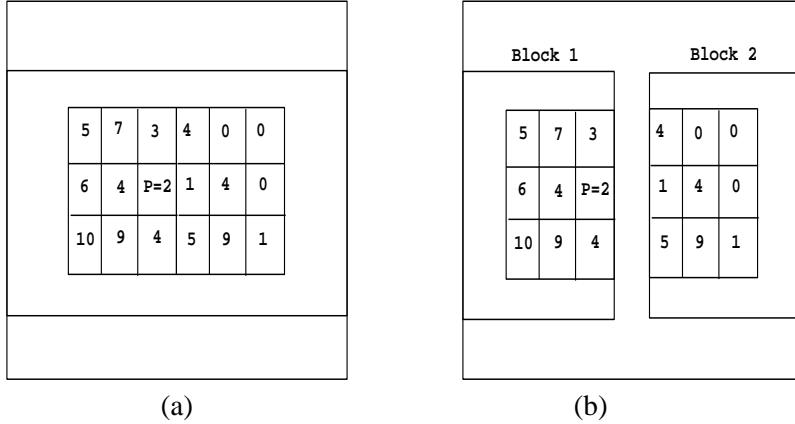


Figure 4.1: An example of incorrect identification of a border pixel due to splitting: (a)  $P$  as non-minimum pixel in the image (b)  $P$  as a single-pixel minimum in block 1

*incomplete*. Yet another instance of this problem is the mis-classification of a flat plateau as a flat maximum or minimum within a block.

2. Splitting of flat regions along the border in a block: Flat regions may be split into two or more sub-regions in different blocks. This may occur if segments<sup>1</sup> in one block re-enter the same block or other blocks [45]. Figure 4.2(a) shows an example of a re-entering segment. Splitting of flat regions causes the region to have different boundary minima in the different blocks. A simple example is illustrated in Figure 4.2(b). Consider the flat region with value 4 in Figure 4.2(b). This has been split into three sub-regions in blocks 1, 2, and 4. The sub-region in block 1 appears to be a flat basin. The sub-region in block 2 has a boundary minimum at  $P = 0$ . The sub-region in block 4 has a boundary minimum at  $Q = 2$ . We know that the tracing of flat regions begins at the boundary minimum

---

<sup>1</sup>In the rest of this chapter, we will use the term *regions* to refer only to flat regions on the topographical surface. We will use the term *segments* to refer to segments as in the result of watershed segmentation.

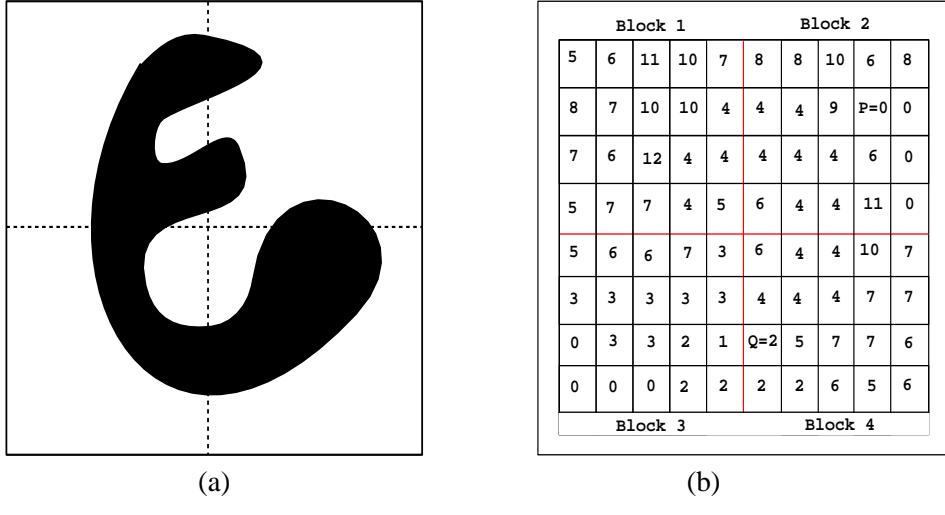


Figure 4.2: Examples of the problem of split flat regions: (a) Example of a re-entering segment  
(b) Example of a split flat region

pixel. Hence, for this example, the same flat region will be traced to different minima in blocks 1, 2, and 4. To prevent this, it is important to identify the equivalence of these sub-regions and compare the local boundary minima to identify  $P$  as the correct global boundary minimum of the flat region.

Both the problems discussed above affect the tracing of pixels to the *correct* minima. This results in the assignment of incorrect labels to pixels, and hence in an incorrect segmentation. Consider the 1D steepest descent path shown in Figure 4.3(a). The path consists of the pixels  $\{x_1, \dots, x_7\}$ . Let  $P$  be the label of  $x_3$  and  $Q$  be the label of  $x_7$ . In the figure, all the pixels trace to  $x_7$  and hence get the label  $Q$ . It is important to understand that wherever the tracing begins in this path, the minimum reached is the same, and is the pixel  $x_7$ . If the path  $\{x_1, \dots, x_7\}$  is split into two parts with pixel  $x_3$  occurring at the border of the parts, two steepest descent paths

are obtained — one that consists of  $\{x_1, x_2, x_3\}$ , which get label  $P$ , and another that consists of  $\{x_4, \dots, x_7\}$ , which get label  $Q$ . To connect these two paths, we only have to identify that  $x_3$  traces to the minimum  $x_4$  in its neighborhood, and hence labels  $P$  and  $Q$  are equivalent. To achieve this, we split the path in such a way that pixels  $x_3$  and  $x_4$  are present in both parts. In the first part, we trace  $x_1$  and  $x_2$  to  $x_3$ , and assign them the label  $P$ . In the second, we trace  $\{x_4, x_5, x_6\}$  to  $x_7$  and assign them the label  $Q$ . In both parts, we do not begin any tracing at  $x_3$  because it is a border pixel. We simply *note* that  $x_4$  is the minimum in the neighborhood of  $x_3$  in the two parts. This information can be used to complete the tracing that stopped at the border pixel  $x_3$ . In this process, we assign the label of  $x_4$ , which is  $Q$ , to  $x_3$  and all the pixels traced to  $x_3$ . It is clear that to continue the tracing which stopped at the border pixel  $x_3$ , we only need the local minimum in the neighborhood of  $x_3$  in both parts.

The above involves a 1D path split across two blocks. However, this idea can be extended to the case of 2D (and 3D), and splitting across multiple blocks. Consider a specific example of a 2D steepest descent path shown in the image in Figure 4.4(a). We split the image into two blocks as in Figure 4.4(b). The flat region of value 1 is split into two sub-regions, causing the steepest descent path also to split into two and hence the tracing to be incomplete. To solve this problem, we split the blocks in such a way that the two blocks have an overlap of two columns. This is shown in Figure 4.4(c). We note that the sub-region in block 1 is itself a minimum and hence store the value 1 as its boundary minimum. We also note that the sub-region in block 2 has a boundary minimum of 0. By comparing the boundary minima in the two blocks, we identify that the flat region of value 1 is not a minimum as classified in block

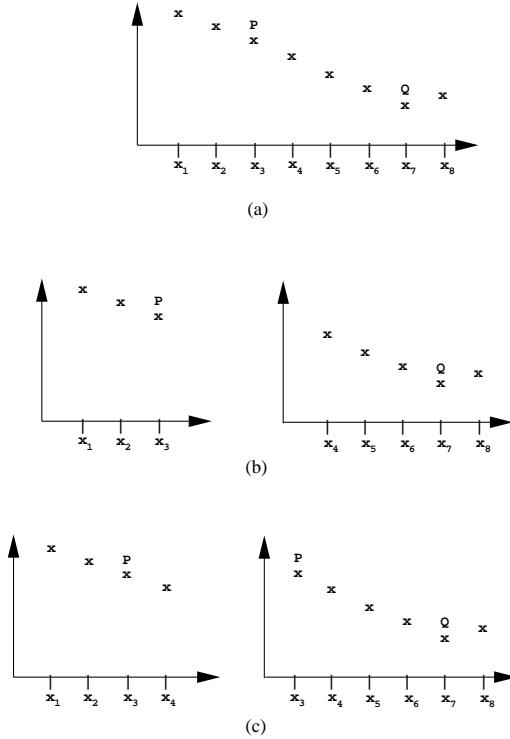


Figure 4.3: Splitting of a 1D steepest descent path

1, but that it traces to the minimum 0 in block 2. Therefore, the correct label of the path in block 1 is the label of the path in block 2. Note that the path in either block is a steepest descent path and it is sufficient to compare the boundary minima of the sub-regions in the two blocks to solve the problem of incomplete tracing. This fact can be exploited in computing distributed watershed segmentation without the need for iterative communication among adjacent blocks. In generating the watershed segmentation of each block, the steepest descent paths can be made to stop at the border pixels of the blocks. Splitting of the blocks can be done as shown in the Figure 4.4(c). The boundary minima of the border pixels can be stored for each block and they

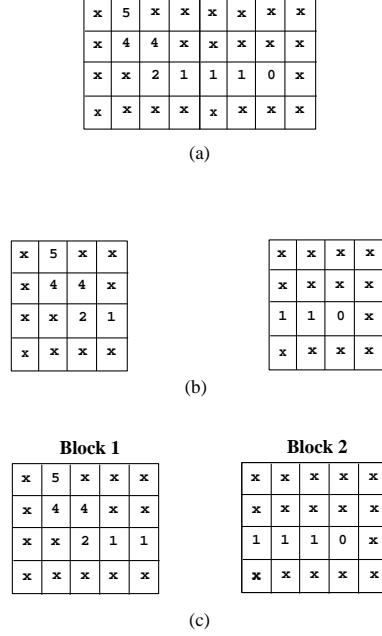


Figure 4.4: Example of splitting of a 2D steepest descent path

can be used to complete the tracing that was stopped at the borders of the blocks. We have designed a streaming algorithm to test this approach. This algorithm works in three stages. In the first stage, we compute the watershed segmentation of each block separately by a modified watershed algorithm. In the second stage, we process adjacent blocks, pairwise, to exchange boundary minima information about the border pixels. Using the result of this stage, we re-label pixels in the blocks in the third stage. At the end of these three stages, we can simply combine the segmentation of the blocks to get the segmentation of the whole image. The details of the implementation used for testing are as follows:

- Stage 1:
  1. We assume that the pre-processing steps, namely, filtering and thresholding have been completed and the resulting edge map is the input to the algorithm.
  2. We divide the input image into blocks such that the matrix of these blocks is of size  $X \times Y$ . Depending on the row positions in the matrix, we assign the blocks *row types*  $F$ ,  $M$ , and  $L$  which denote first, middle, and last respectively. Similarly, depending on the column positions in the matrix, we assign the blocks *column types*  $F$ ,  $M$ , and  $L$ . This is illustrated in Figure 4.5. The division into blocks is such that every pair of adjacent blocks have two columns (or two rows) overlapping. Consider Figure 4.6(a). The division of this image into 9 blocks with overlap is shown in Figure 4.6(b). Note that in this case,  $X = Y = 3$ .
  3. We compute the watershed segmentation of *each* block using a modified watershed algorithm, discussed as follows:
    - (a) We complete the searching step as discussed in Section 3.3.
    - (b) Depending on the row and column types of the block, we store the boundary minima of selected border pixels. The boundary minimum for a non-flat, non-minimum pixel, refers to the minimum in its 8-neighborhood. The boundary minimum for a pixel or flat region which is a minimum inside the block would be the pixel or flat region itself. Let  $p$  be the number of rows in a block and  $q$  be the number of columns. The rows and columns for which the boundary minima are stored, depending on the row and column types of the block, are

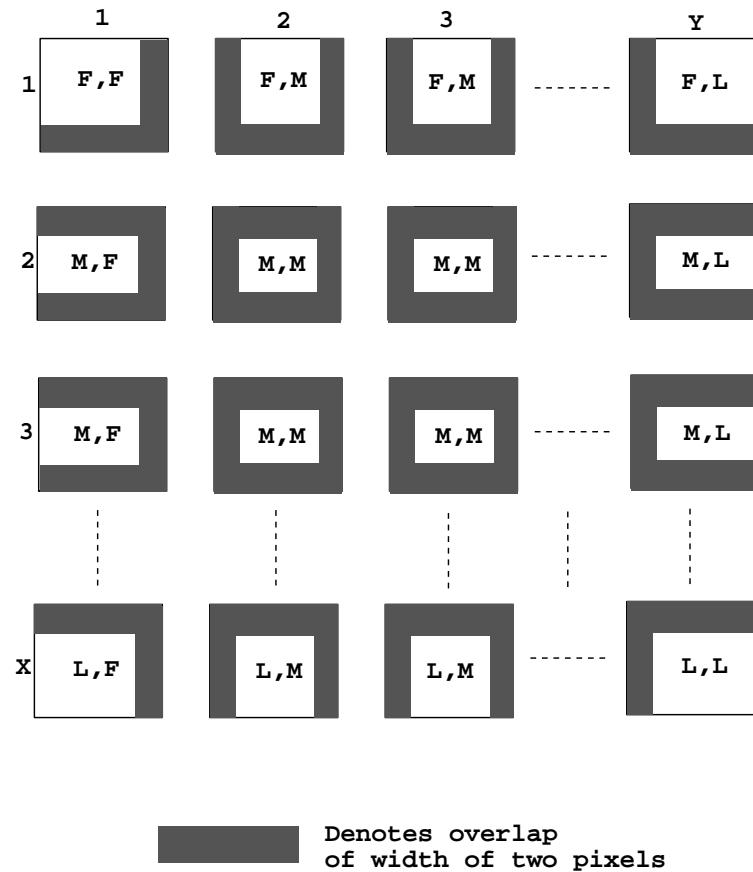
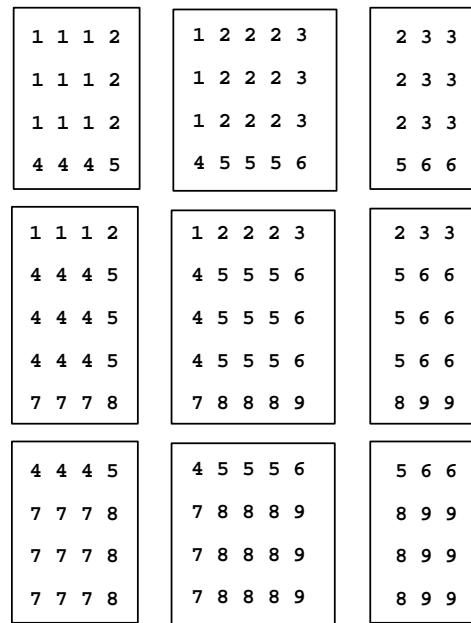


Figure 4.5: Division of the image into blocks with overlap

1	1	1		2	2	2		3	3
1	1	1		2	2	2		3	3
1	1	1		2	2	2		3	3
4	4	4		5	5	5		6	6
4	4	4		5	5	5		6	6
4	4	4		5	5	5		6	6
7	7	7		8	8	8		9	9
7	7	7		8	8	8		9	9
7	7	7		8	8	8		9	9

(a)



(b)

Figure 4.6: An example to illustrate overlap among adjacent blocks: (a) Original image (b) Splitting of the image into blocks with overlap

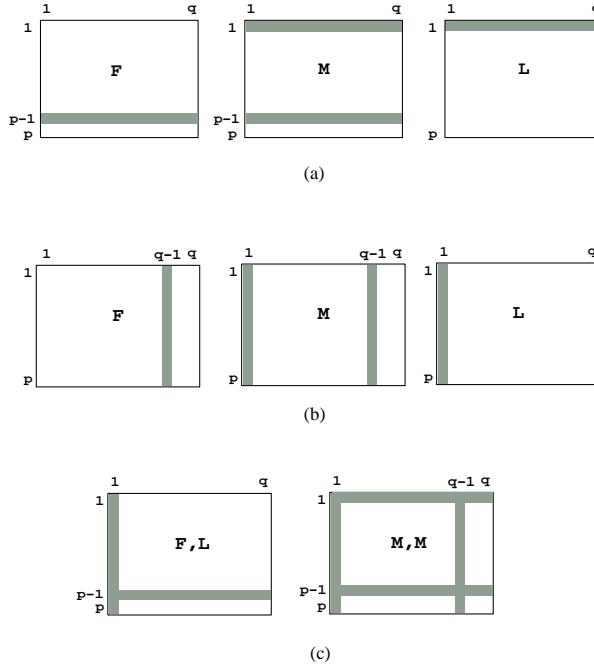


Figure 4.7: Boundary minima are stored for (a) Pixels in the shaded rows based on the row type and (b) Pixels in the shaded columns based on the column type. (c) Example of blocks of (Row type, Column type) equal to  $(F, L)$  and  $(M, M)$

shown in Figures 4.7(a)–(b). Specific examples of blocks of type  $(F, L)$  and  $(M, M)$  are shown in Figure 4.7(c).

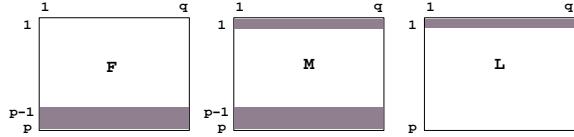
- (c) Depending on the row and column types of the blocks, we mark selected pixels along the border using negative labels, as shown in Figures 4.8(a)–(b). Marking with negative pixels is done to indicate that these pixels are *unknown*. This means that enough information about the neighbors of these pixels is not available to complete the next (tracing) step. An example block of type  $(F, F)$  is shown in Figure 4.8(c). In this figure,  $m$  is a single-pixel minimum and  $f$  is a

flat region detected during the searching step. The other pixels are non-minima and non-flat pixels. For this block, we mark pixels in the shaded area as *unknowns* using unique negative numbers. The pixel at location (2, 3) is also *unknown* because it is part of the flat region of value  $f$  that spreads across rows 3 and 4, and columns 4 and 5. Note that the two-pixel wide overlap is useful in keeping track of split flat regions.

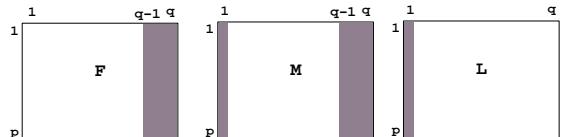
- (d) In the tracing step, we trace all pixels except the ones marked *unknown*. If a steepest descent path leads to an *unknown*, the tracing stops there. No tracing begins at any *unknown* pixel.
4. We compute the segmentation of all the blocks in the manner described above. We adjust the labels in the watershed result of all the blocks such that no two blocks use the same label.

- Stage 2:

1. We then process adjacent blocks, pairwise, to exchange boundary minima of pixels and solve the problem of incomplete tracing at the border of the blocks. For instance, pairwise processing of blocks for the example in Figure 4.6 is done as shown in Figure 4.9.
2. For each pair of adjacent blocks, we handle the *unknown* border pixels as follows. Consider the blocks  $a$  and  $b$  shown in Figure 4.10.  $a_1$  and  $a_2$  denote the last two columns in  $a$ , and  $b_1$  and  $b_2$  denote the first two columns in  $b$ . Note that the pixels in  $a_1$  and  $a_2$  are the same as those in  $b_1$  and  $b_2$  respectively, and that they are *unknown*



(a)



(b)

	1	2	3	4	5
1	x	x	x	x	x
2	x	m	f	f	f
3	x	x	x	f	f
4	x	x	f	x	x

F,F block

(c)

Figure 4.8: Marking pixels as *unknown* based on row and column types of blocks: (a) Pixels and flat regions spread over shaded rows are *unknown*. (b) Pixels and flat regions spread over shaded columns are *unknown*. (c) Example of a block of (Row type,Column type) equal to  $(F, F)$

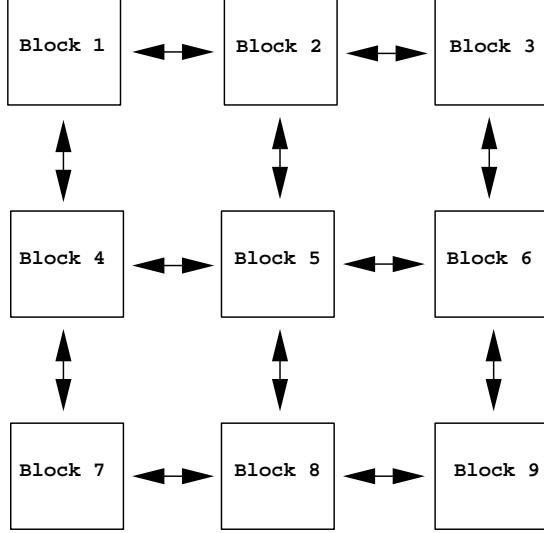


Figure 4.9: Pairwise processing of adjacent blocks for the example in Figure 4.6

pixels. Consider one such pixel  $P_a$  in  $a$  and  $P_b$  in  $b$ . Let  $L_{P_a}$  and  $L_{P_b}$  be the labels of  $P_a$  and  $P_b$  respectively. Let  $m_1$  be the boundary minimum of  $P_a$  and  $L_{m_1}$  be its label. Let  $m_2$  be the boundary minimum of  $P_b$  and  $L_{m_2}$  be its label. Let us refer to the occurrence of  $P_a$  (or  $P_b$ ) in the original image as  $P$ . Let the label of  $P$  be  $L_P$  and its boundary minimum be  $m$ . We process pixels  $P_a$  and  $P_b$  as described below:

- (a) We first map the labels of  $P_a$  and  $P_b$  equivalent. This is done by storing the pair  $(L_{P_a}, L_{P_b})$  in a map of equivalent labels called  $M$ .
- (b) We then compare the minima  $m_1$  and  $m_2$  to solve the incomplete tracing problem at the borders of blocks  $a$  and  $b$ . There are four cases to be considered based on the minima  $m_1$ ,  $m_2$ , and  $m$ .
  - i.  $m_1 = m_2$ ,  $m_1 \leq m$ , and  $m_1$  lies in  $a_2$ : This means that tracing from pixel  $P$  continues in the block  $b$ . Therefore,  $L_P = L_{m_2}$  and  $m = m_2$ .

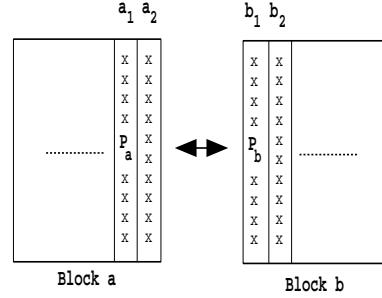


Figure 4.10: Processing of two adjacent blocks

- ii.  $m_1 = m_2$ ,  $m_1 \leq m$ , and  $m_1$  does not lie in  $a_2$ : This could mean that the boundary minimum of  $P_a$  and  $P_b$  is the same and lies in  $a_1$  (or  $b_1$ ). For this, we can make the assignments  $L_P = L_{m_1}$  and  $m = m_1$ . This case could also mean that there are two different boundary minima with the same value. Note, however, that such cases are genuine ambiguities in the data and can also arise in the serial watershed algorithm. In our algorithm, we choose to make the assignments  $L_P = L_{m_1}$  and  $m = m_1$ .
- iii.  $m_1 < m_2$  and  $m_1 \leq m$ : This indicates the tracing from  $P$  continues in block  $a$ . Therefore,  $L_P = L_{m_1}$  and  $m = m_1$ .
- iv.  $m_2 < m_1$  and  $m_2 \leq m$ : This indicates the tracing from  $P$  continues in block  $b$ . Therefore,  $L_P = L_{m_2}$  and  $m = m_2$ .

The above processing scheme has two advantages. First, it serves to identify the equivalence of occurrences of the *same* pixel  $P$  in different blocks by storing the

equivalence of their labels. This helps to detect the equivalence of shared flat regions and re-entering segments. Second, pairwise processing of adjacent blocks for comparison of boundary minima helps to locate the *actual* boundary minimum of  $P$ , which will be the same as the one detected by the serial watershed algorithm.

3. We process all the pairs of adjacent blocks using the scheme described above.

- Stage 3:

From the result of Stage 2, we know the label of the correct minima of the *unknown* pixels. We add this information to the map  $M$ . This is done by storing the label of one of the occurrences of every *unknown* pixel, and the label of the correct minimum as equivalent. For instance, consider a pixel  $P$  in the original image that occurs as the border pixel  $P_a$  in one block,  $P_b$  in a second block, and  $P_c$  in a third block. For this pixel  $P$ , it is sufficient to store the pair  $(L_{P_a}, L_P)$  in  $M$ . We then use  $M$  to derive a lookup table  $L$  which gives the final label for every *unknown* pixel. Let  $N_M$  be the size of  $M$  and the  $m$ th element in  $M$  be of the form  $(a_m, b_m)$ . We use the following procedure to generate  $L$ .

for  $i = 1$  to  $N_M$

if  $L[a_m]$  is not set and  $a_m$  is *unknown*

$L[a_m] \leftarrow a_m$

if  $L[b_m]$  is not set and  $b_m$  is *unknown*

$L[b_m] \leftarrow b_m$

$a \leftarrow L[a_m]; b \leftarrow L[b_m]$

```

if  $a$  is equal to  $b$ 

    continue with the next element in  $M$ 

for every  $l$  in  $L$ 

    if  $L[l]$  equals  $a$ 

         $L[l] \leftarrow b$ 

```

We use the table  $L$  to re-label the *unknown* pixels in the blocks. At the end of this step, each block represents a portion of the segmentation of the entire image. Therefore, the blocks can be simply put together to obtain the watershed segmentation of the entire image.

### 4.3 Proof of Correctness

In the streaming algorithm described above, every pair of adjacent blocks is processed only once. This is consistent with the general concept discussed earlier in this chapter, that distributed computation of watershed segmentation can be achieved without iterative communication among adjacent blocks. To show this, we first discuss some properties of the streaming algorithm as follows:

1. The tracing of pixels within a block is *complete* except at the border. There are two kinds of pixels in the watershed result of a block, namely, *unknown* and *known*. As discussed earlier, pixels may be traced to *unknown* pixels but no tracing begins at these pixels. The

*known* pixels are those that trace to local minima within the block. The watershed result of each block is incomplete because the tracing is stopped at the *unknown* pixels.

2. A pixel can be considered as a flat region of size 1. Hence, we refer to pixels as flat regions or simply *regions*. To start tracing at a region, we need to find the boundary minimum of the region. Consider one such region. In the serial watershed algorithm, tracing for this region would begin at the boundary minimum. Splitting of the image into blocks causes the region to have different boundary minima in different blocks. It is necessary to compare the local boundary minima in the blocks to obtain the global boundary minimum and hence the correct steepest path out of this region. In our streaming algorithm, this is achieved through pairwise processing of adjacent blocks.
3. The final segmentation result could contain *unknown* labels also. The presence of *unknown* labels does not mean that the segmentation is incorrect or incomplete. *Unknown* labels in the final segmentation are caused by minima which occur along the border of the blocks. A simple example of this case is shown in the Figure 4.11. The pixel with value **3** is traced to the pixel **0**, which is a minimum pixel carrying an *unknown* label.

We can prove that the streaming algorithm produces the same result as the serial watershed algorithm by method of induction. Consider  $P$  in Figure 4.12. In this figure,  $P$  is traced to a minimum  $M$  within the block. We know that the paths within a block are steepest descent paths.  $M$  is not along the border and therefore, the tracing of  $P$  to  $M$  is correct. Consider the case in Figure 4.13. Here,  $P$  is traced to a minimum  $M$  in an adjacent block. The paths within the blocks are steepest descent paths. Therefore, the tracing of  $P$  to  $M$  will be correct if the

6	5	3	3
5	4	2	1
5	3 → 0	0	0
5	4	2	2
7	6	4	3

3	3	4	5
2	1	2	4
0	0	0	5
2	2	4	5
4	3	5	6

Figure 4.11: The pixel with value **3** traces to the minimum **0** carrying an *unknown* label.

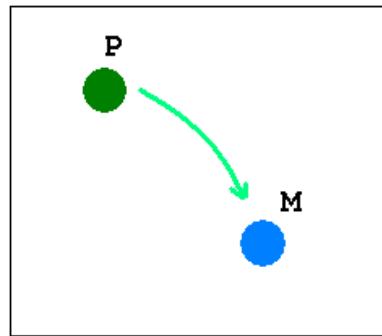


Figure 4.12: Proof — Minimum within the block

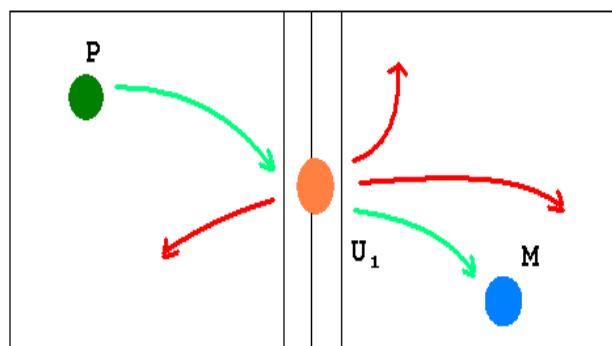


Figure 4.13: Proof — Minimum in an adjacent block

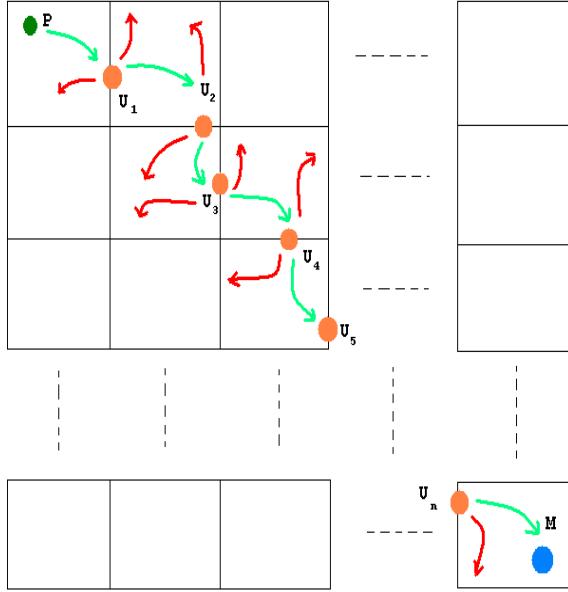


Figure 4.14: Proof — Minimum several blocks away

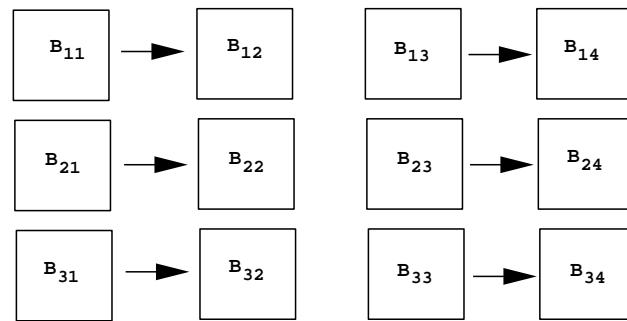
path  $PU_1$ , that starts from  $P$  and ends in  $U_1$  is connected with the *correct* path in the other block. We know that the streaming algorithm uses the boundary minimum of  $U_1$  to continue its tracing. Hence, the two steepest descent paths,  $PU_1$  and  $U_1M$  connected through the boundary minimum of  $U_1$  form a single steepest descent path. Thus, the tracing of  $P$  to  $M$  is correct.

Consider the case in Figure 4.14 where  $P$  is traced to a minimum  $M$  several blocks away. From the previous two cases, we know that paths  $PU_1$  and  $PU_1U_2$  are steepest descent paths. Then by induction,  $PU_1U_2U_3$  is also a steepest descent path. Proceeding thus, the different paths connected through the boundary minima of  $U_1, U_2, \dots, U_n$  form a single steepest descent path  $PU_1U_2\dots U_nM$ . Hence, in this case also, the tracing of  $P$  to  $M$  is correct. Thus, our streaming algorithm produces the same result as the serial watershed segmentation without iterative processing of adjacent blocks.

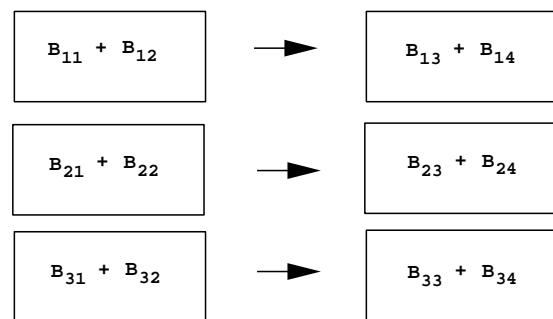
## 4.4 Parallel Implementation

An efficient parallel algorithm can be implemented using the above described streaming algorithm and a binary-tree-like reduction method. We use the example in Figures 4.15 and 4.16 to describe the binary reduction scheme. Figure 4.15(a) shows the division of the image into a matrix of blocks of size  $3 \times 4$ . Let  $B_{ij}$  denote a block in the  $i$ th row and  $j$ th column of this matrix. The pairwise processing method described in Section 4.2 can be used as a scheme for communication among pairs of adjacent blocks. This can be combined with a binary reduction of columns of blocks, followed by reduction of rows. The reduction of columns is illustrated in Figures 4.15(a)–(c). Along row 1, block  $B_{11}$  communicates with block  $B_{12}$ , and block  $B_{13}$  communicates with block  $B_{14}$ . In the next step, the combination of blocks  $B_{11}$  and  $B_{12}$  communicates with the combination of  $B_{13}$  and  $B_{14}$ , resulting in a single block along row 1. This occurs for all other rows until each row has a single block as in Figure 4.15(c). Thus, binary reduction of columns is complete. Binary reduction of rows occurs in a similar way and this is shown in Figures 4.16(a)–(b). In general, let the size of the matrix of the blocks be  $X \times Y$ , where  $X$  is the number of rows, and  $Y$  is the number of columns. Consider the implementation of the above binary reduction scheme on a single processor. The cost of completing the reduction of columns for each row is  $Y - 1$ . The cost for  $X$  such rows is  $X(Y - 1)$ . The cost of completing the reduction of rows is  $X - 1$ . Therefore, the total cost for communication among blocks is  $X(Y - 1) + X - 1 \approx O(XY)$ . If  $X = Y = N$ , this is  $\approx O(N^2)$ .

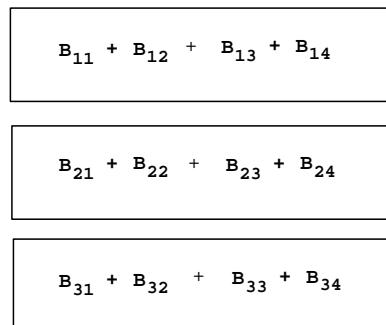
Consider Figure 4.15(a) again. In row 1, the communication between  $B_{11}$  and  $B_{12}$ , and  $B_{13}$  and  $B_{14}$  can occur at the same time, and simultaneously as the communication between



(a)

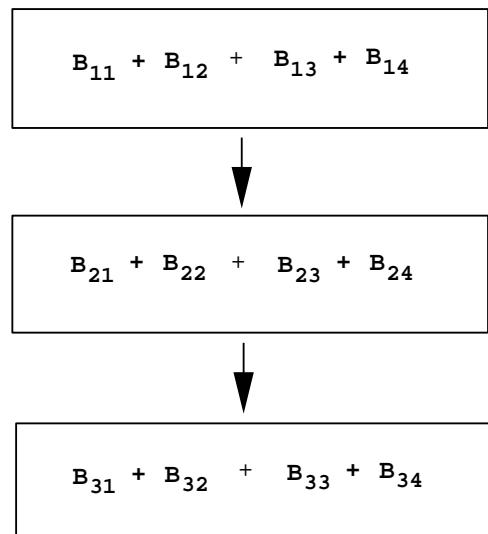


(b)

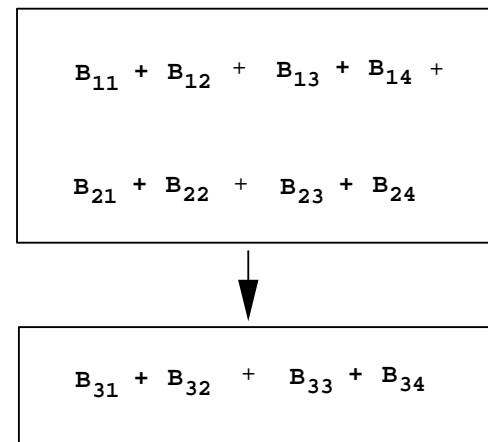


(c)

Figure 4.15: Binary reduction of columns



(a)



(b)

Figure 4.16: Binary reduction of rows

$B_{21}$  and  $B_{22}$ ,  $B_{23}$  and  $B_{24}$ ,  $B_{31}$  and  $B_{32}$ ,  $B_{33}$  and  $B_{34}$ . Similarly the communication between blocks shown in Figure 4.15(b) can also occur simultaneously. This holds for reduction of rows too. Thus, the binary reduction scheme can be implemented as a parallel algorithm. For the example under consideration, this is illustrated in the Figure 4.17. If  $X \times Y$  is the size of the matrix of the blocks, the cost of completing the reduction of columns for a single row is  $\log_2 Y$ . The cost for  $X$  such rows is  $X \log_2 Y$ . The cost of completing the reduction of rows is  $\log_2 X$ . The total cost for communication among blocks is  $X \log_2 Y + \log_2 X$ . If  $X = Y = N$ , this is  $\approx O(N \log N)$ . The cost in this case is much less than in the earlier case where a single processor is used. Thus an efficient parallel implementation can be achieved using a combination of the streaming algorithm and the binary reduction method.

## 4.5 Cost of Generating the Lookup Table

The cost of computation includes the cost of computing the watershed segmentation on all processors, the overhead for communication among block pairs, the cost of generating the lookup table  $L$ , and the cost of re-labeling pixels in the blocks. The cost of computing the watershed segmentation and the cost of the final re-labeling step are both proportional to the size of the image. The communication overhead depends on the size of the matrix of blocks, as discussed in Section 4.4. In this section, we analyze the cost of generating the lookup table  $L$ .

Let  $N_M$  be the size of the map  $M$  and  $N_U$ , the number of *unknowns*, be the size of the lookup table  $L$ . Consider a 2D image of size  $N \times N$ , which is divided into a matrix of blocks of size  $X \times Y$ . The worst case of  $N_U$  occurs when every border pixel gets a unique *unknown*

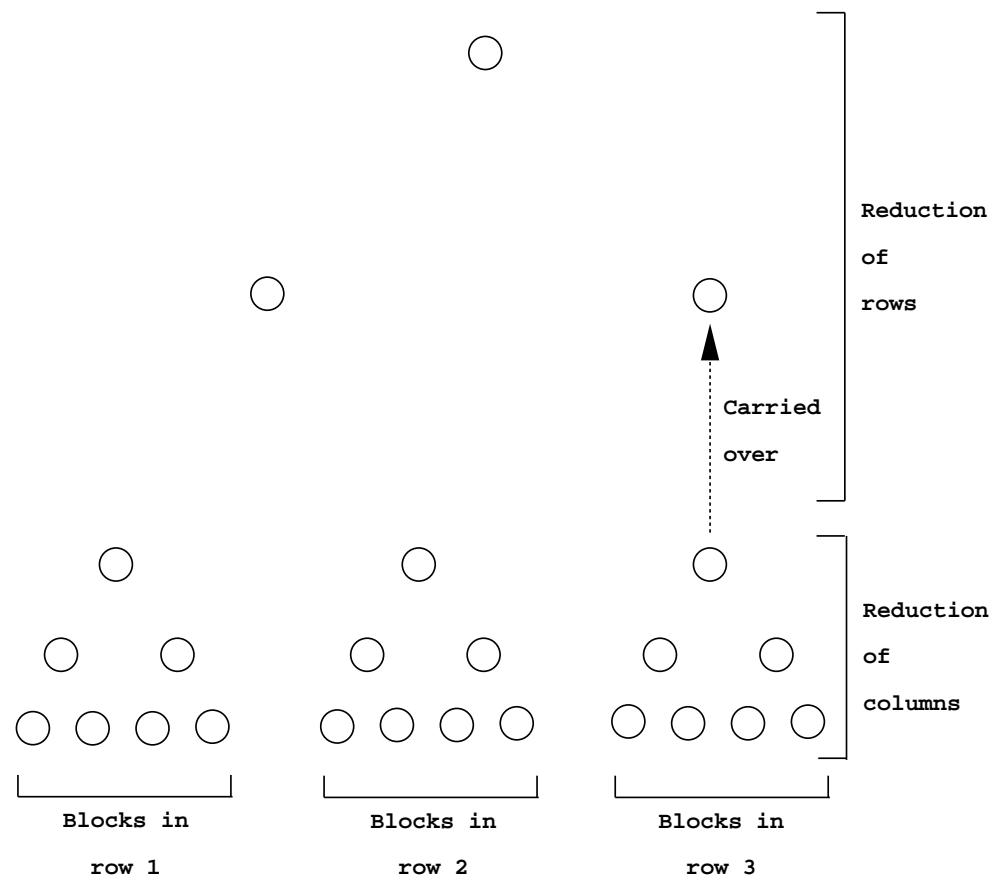


Figure 4.17: Parallel implementation of the binary reduction scheme

label. Considering the overlap of the blocks, we can show that  $N_U = 2N[(X - 1) + (Y - 1)] + 4(X - 1)(Y - 1)$ . For constant  $X$  and  $Y$ , we can reduce this expression to the form  $N_U = k_1N + k_2$  where  $k_1$  and  $k_2$  are constants. Therefore,  $N_U$  is  $\approx O(N)$ .  $N_M$  is the sum of  $N_E$ , the number of pairs of equivalent labels and  $N_C$ , the number of entries of the correct minimum for each set of equivalent *unknowns*. We can show that the  $N_E = N[(X - 1) + (Y - 1)] + 4(X - 1)(Y - 1)$ . The value of  $N_C$  depends on the nature of the input edge map. In the worst case, when every border pixel gets a unique *unknown* label,  $N_C = N[(X - 1) + (Y - 1)] + (X - 1)(Y - 1)$ . Adding and simplifying, we can show that  $N_M$  is of the form  $N_M = k_1N + k_3$  where  $k_3$  is another constant. Therefore,  $N_M$  is  $\approx O(N_U)$ . Therefore, the worst case cost of generating the lookup table is  $= O(N_M N_U) \approx O(N_U^2) \approx O(N^2)$ . For a 2D input image, the *unknown* pixels are in 1D. For a nD image, the *unknown* pixels will be in (n-1)D. Therefore, for a nD image, which is of length  $N$  along each dimension, the cost is  $O(N^{2(n-1)})$ .

# Chapter 5

## Results and Discussion

### 5.1 Watershed Algorithm — Testing

We begin by verifying the implementation of the 2D watershed algorithm with some simple test images. Figure 5.1(a) shows a 2D sine image of size  $128 \times 128$  that we have used directly as the gradient input to the watershed algorithm. This image,  $f(i, j)$ , is calculated using Equation 5.1.

$$f(i, j) = 5 \sin\left(\frac{\pi i}{20}\right) \sin\left(\frac{\pi j}{20}\right) \quad (5.1)$$

The pixels in the sine image  $f$  represent a topographical surface that contains flat plateaus, single-pixel minima, single-pixel maxima, and other non-minima pixels. The image serves to verify that the watershed algorithm detects and traces such pixels correctly. The resulting segmentation is shown in 5.1(b). We have used different colors to represent different *segments*

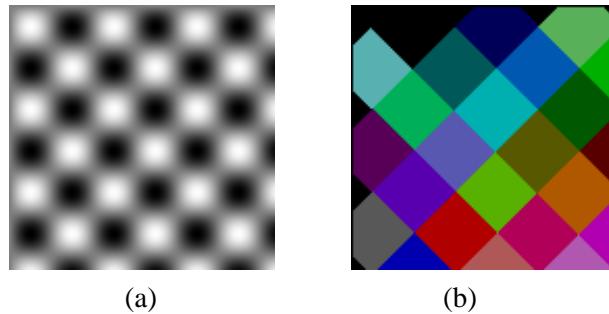


Figure 5.1: Verification of the 2D implementation of the watershed algorithm: (a) 2D sine as gradient input ( $128 \times 128$ ) (b) Watershed segmentation

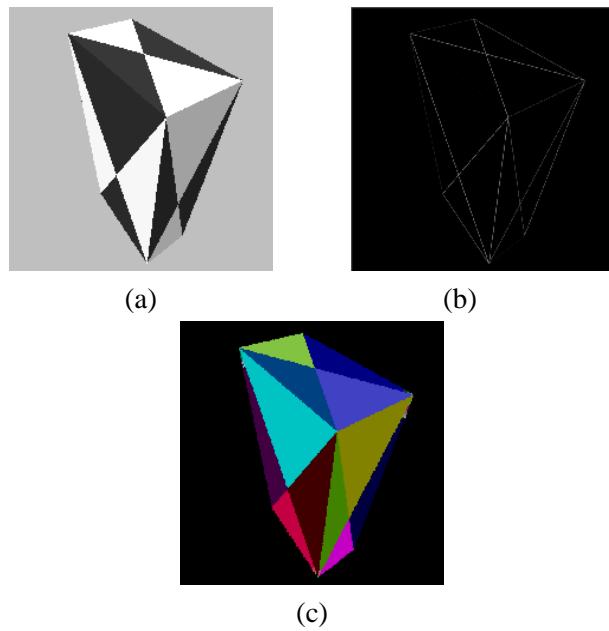


Figure 5.2: Verification of the 2D implementation of the watershed algorithm: (a) Image of a box ( $745 \times 734$ ) (b) Gradient image (c) Watershed segmentation

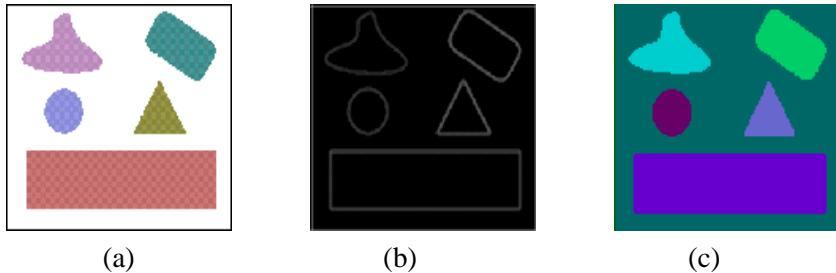


Figure 5.3: Verification of the 2D implementation of the watershed algorithm: (a) Image with objects of different shapes ( $236 \times 201$ ) (b) Gradient image (c) Watershed segmentation

in the output.

Figure 5.2(a) is a synthetic image containing flat regions and verifies the ability of the algorithm to detect flat segments. Figure 5.3(a) is another synthetic image that contains objects of different shapes. This is used to check that the algorithm identifies boundaries of objects with various shapes correctly. The outputs of the watershed algorithm for these images are shown in 5.2(c) and 5.3(c) respectively.

We have tested the implementation of the 3D watershed algorithm using a 3D phantom image of the head whose slices are shown in Figure 5.4. The slices are 50 in number, each of which is of size  $54 \times 54$ . Slices of the 3D gradient image are shown in Figure 5.5. The result of the watershed algorithm is shown in Figure 5.6. The result indicates that the algorithm is able to detect segments embedded in 3D.

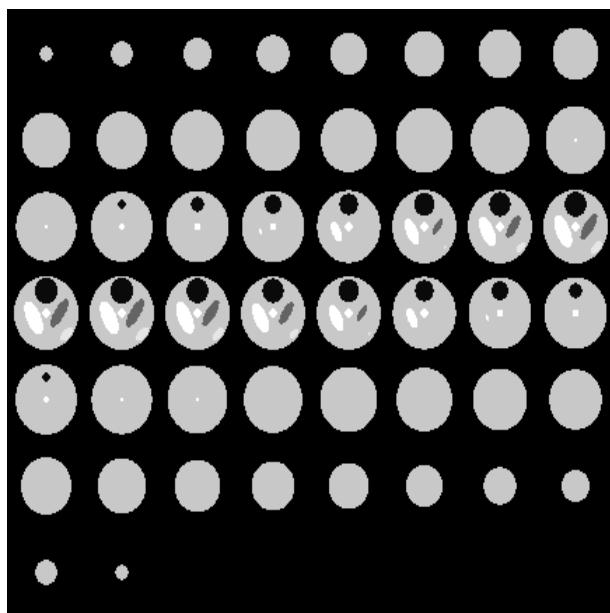


Figure 5.4: A 3D phantom image of the head ( $54 \times 54 \times 50$ )

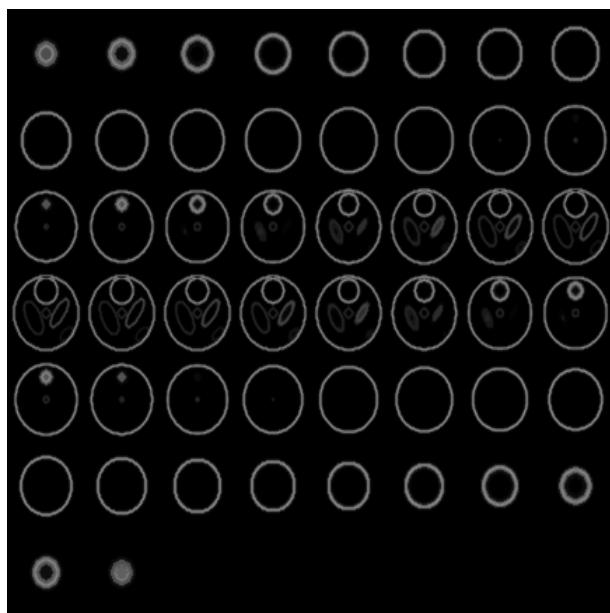


Figure 5.5: Gradient of the slices of the 3D phantom image

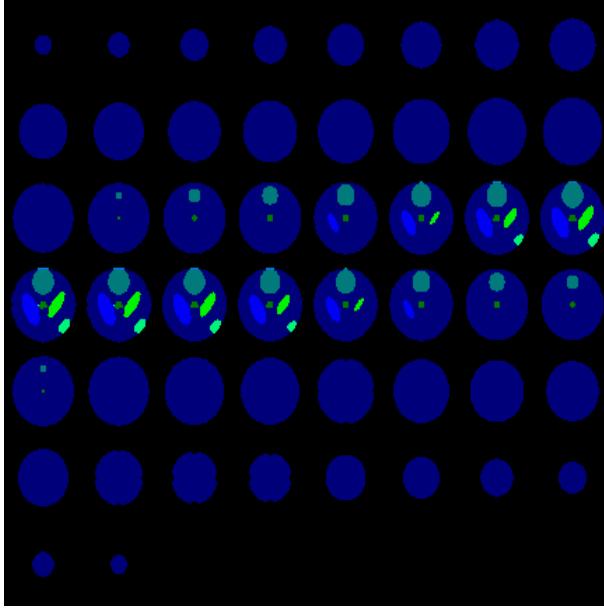


Figure 5.6: Watershed segmentation of the 3D phantom image

## 5.2 Effect of the Pre-processing Step

We have used some synthetic and real images to show the effect of the pre-processing step.

Consider the synthetic image in Figure 5.7(a). Its gradient is shown in Figure 5.7(b). Its watershed segmentation is shown in Figure 5.7(c), which has 6 regions. Gaussian noise is then added to the image in Figure 5.7(a) using the *noiseGauss* function in VISPack with input standard deviation  $\sigma_n = 4$ , which is 50% of RMS of gradient. Watershed segmentation of the corrupted image, shown in Figure 5.7(f), has 9873 regions. This is due to the irrelevant minima created by noise. We subject the corrupted image to filtering using the *gauss* function in VISPack, with input standard deviation  $\sigma_f = 4$ , and then threshold the gradient at  $T = 92\%$  of the RMS value. The effect of this step on the watershed result is indicated in Figure 5.7(g). The pre-processing step removes the irrelevant minima, producing a clean segmentation.

Figure 5.8(a) shows a real image of strips of different colors on a table. This image contains a high level of noise and texture. Therefore, the watershed algorithm produces an oversegmented result as shown in Figure 5.8(c), which has 1499 regions. The original image is subject to filtering with  $\sigma_f = 0.5$  and the gradient is thresholded at  $T = 43\%$  of the RMS value. This pre-processing helps to reduce oversegmentation. The result of the watershed algorithm for the pre-processed image is shown in Figure 5.8(d), in which the number of regions has been reduced to 354.

We have used the 3D phantom image in Figure 5.4 to show the effect of the pre-processing step on the result of the 3D watershed algorithm. Each of the 50 slices is corrupted with Gaussian noise using the *noiseGauss* function with input standard deviation  $\sigma_n = 0.1$ , which is 0.3% of RMS gradient. Watershed segmentation of the corrupted image leads to oversegmentation producing the result in Figure 5.9(a), which contains 5333 regions. We process the corrupted image by filtering with  $\sigma_f = 0.5$  and threshold the gradient at  $T = 4\%$  of the RMS value. This removes the oversegmentation caused due to noise and produces the result in Figure 5.9(b).

The effect of the pre-processing step in reducing the number of irrelevant regions in the output of the watershed algorithm can be observed from Table 5.1. In this table,  $N_{wo}$  denotes the number of regions in the segmentation without pre-processing, and  $N_w$  denotes the number of regions in the segmentation with pre-processing. As seen from the table different thresholds work well for different images. Note, however, that the pre-processing step did not completely remove the oversegmentation in the case of the image in Figure 5.8. For some images, it may be even difficult to find a threshold that reduces oversegmentation without wiping out significant

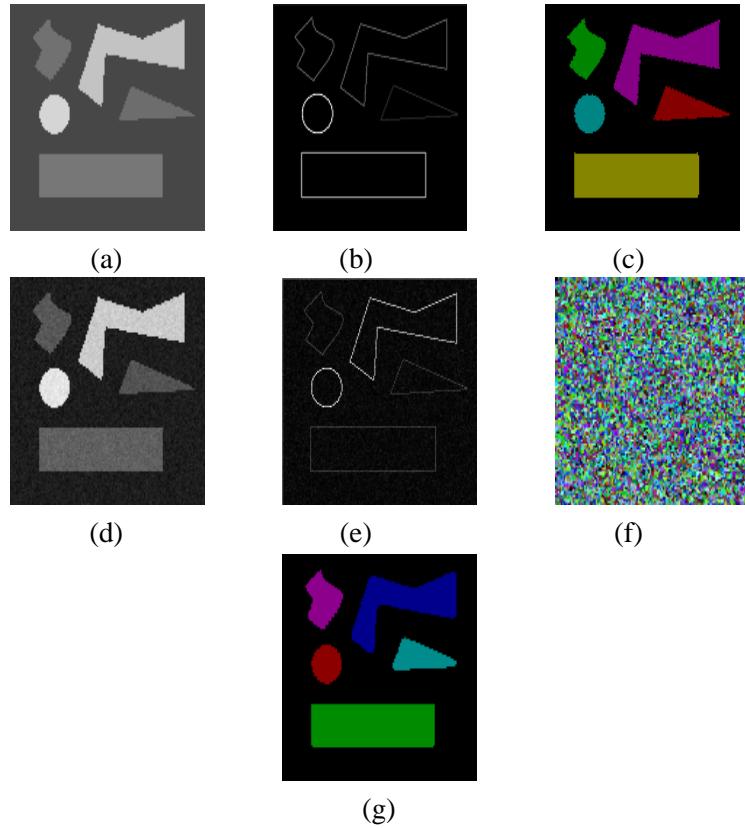


Figure 5.7: Effect of pre-processing on 2D watershed segmentation: (a) A synthetic image ( $335 \times 288$ ) (b) Gradient image (c) Watershed segmentation (d) Corrupted image ( $\sigma_n = 50\%$  of RMS gradient) (e) Gradient of (d) (f) Oversegmented watershed result (without pre-processing) (g) Watershed segmentation (with pre-processing —  $\sigma_f = 4$ ;  $T = 92\%$ )

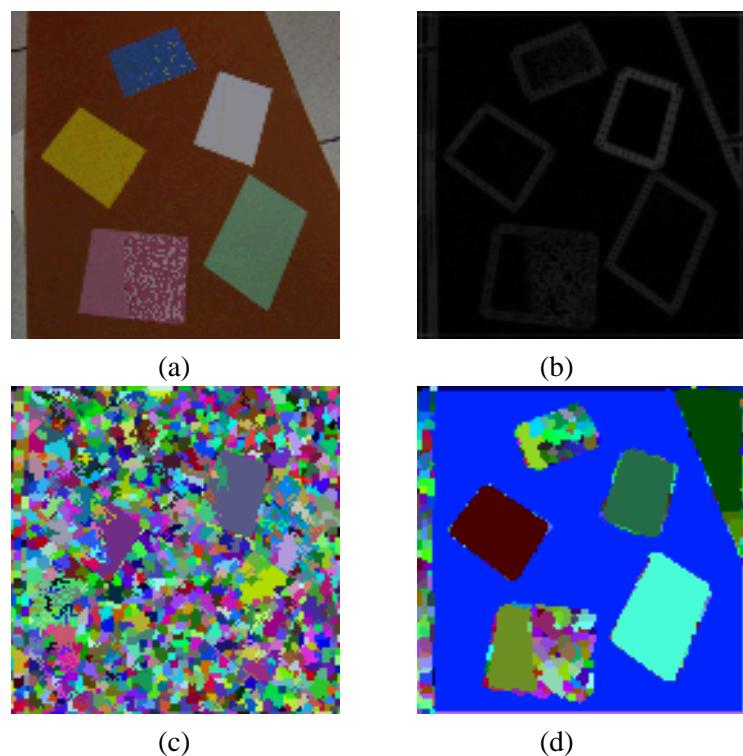


Figure 5.8: Effect of pre-processing on 2D watershed segmentation: (a) A noisy and textured image ( $256 \times 240$ ) (b) Gradient image (c) Oversegmented watershed result (without pre-processing) (d) Watershed segmentation (with pre-processing —  $\sigma_f = 0.5$ ;  $T = 43\%$ )

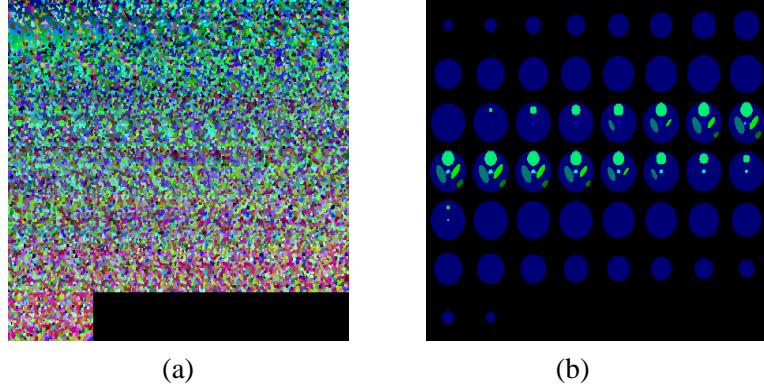


Figure 5.9: Effect of pre-processing on 3D watershed segmentation: (a) Oversegmented watershed result for Figure 5.4 after addition of noise (without pre-processing) (b) Result of watershed segmentation (with pre-processing —  $\sigma_f = 0.5$ ;  $T = 4\%$ )

minima. Such a case is illustrated in Figure 5.10. Figure 5.10(a) is a real image containing noise. Watershed segmentation of the image produces the oversegmented result shown in Figure 5.10(c), which has 5686 regions. We filter the original image with  $\sigma_f = 0.5$  and threshold the gradient at  $T = 12\%$  of the RMS value. The result of the watershed algorithm still shows oversegmentation with 1632 regions, as in Figure 5.10(d). If we increase the threshold  $T$  to 25% of the RMS value to further reduce oversegmentation, it results in the merging of *different* segments in the output, as in Figure 5.10(e). Such problems are overcome by the application of the proposed region merging post-processing schemes.

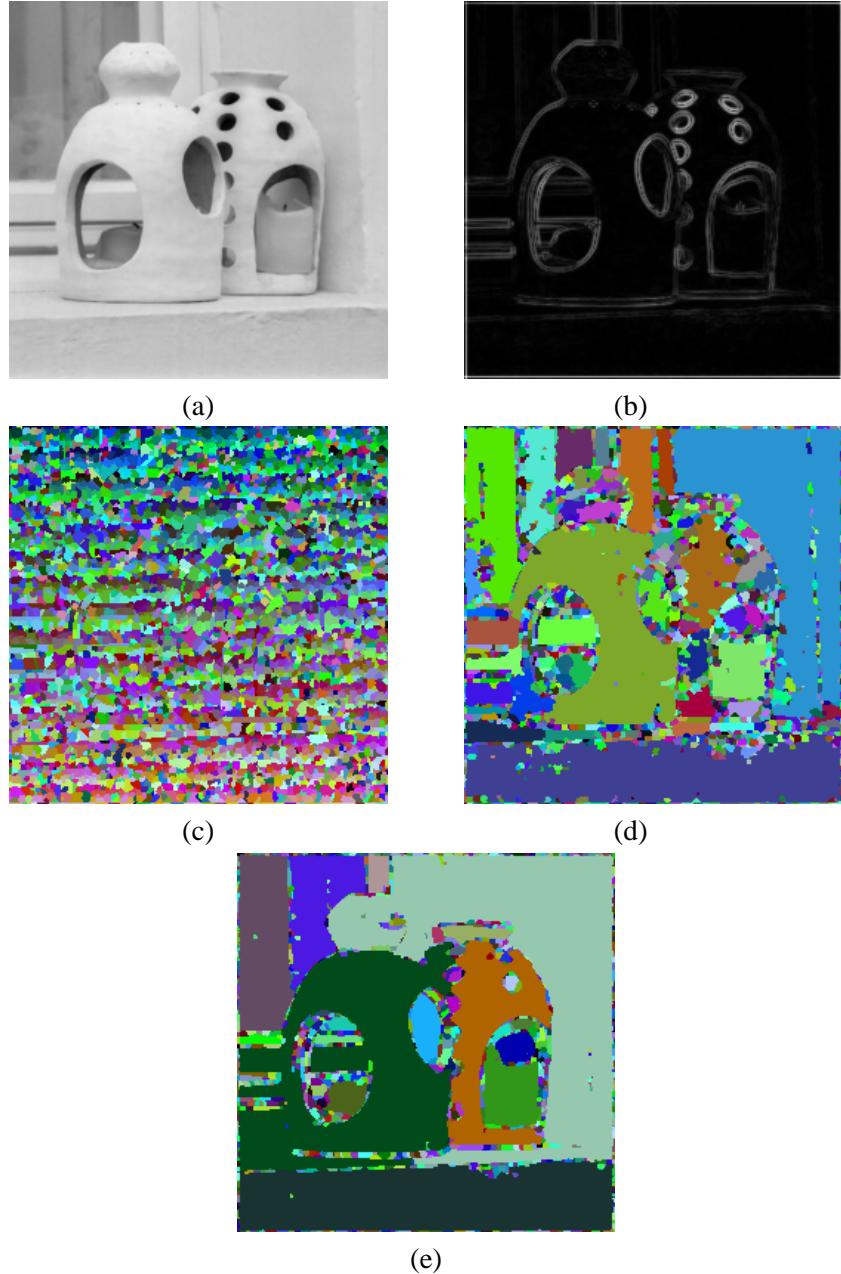


Figure 5.10: Effect of pre-processing on watershed segmentation: (a) Original image (b) Gradient image (c) Oversegmented watershed result (without pre-processing) (d) Watershed segmentation ( $\sigma_f = 0.5$ ;  $T = 12\%$ ) (e) Watershed segmentation ( $\sigma_f = 0.5$ ;  $T = 25\%$ )

Table 5.1: Effect of the pre-processing stage in reducing oversegmentation

Image	$\sigma_f$	T	$N_{wo}$	$N_w$
Figure 5.7	4	92	9873	6
Figure 5.8	0.5	43	1499	354
Figure 5.9	0.5	4	5333	9
Figure 5.10	0.5	12	5686	1632
Figure 5.10	0.5	25	5686	958

## 5.3 Hierarchical Segmentation

### 5.3.1 Implicit Region Merging

We have used synthetic and real images to demonstrate the role of the implicit region merging scheme in reducing oversegmentation and producing a hierarchy from which a meaningful segmentation can be simply chosen. Consider the synthetic house image shown in Figure 5.11(a). The gradient of this image is shown in Figure 5.11(b), and the watershed segmentation is shown in Figure 5.11(c). The synthetic image is corrupted by adding Gaussian noise to the RGB components using the *noiseGauss* function with  $\sigma_n = 2.3$  which is 25% of the RMS gradient. Watershed segmentation of the corrupted image without pre-processing produces the oversegmented result in Figure 5.11(f), which has 10,870 regions. To show the working of the merging scheme, we process the input image by filtering with  $\sigma_f = 2$  and threshold the gradient at  $T = 5\%$  of the RMS value. This pre-processing reduces the number of regions in the watershed output, shown in Figure 5.12(a), to 794. We apply implicit merging on the regions in this image to produce a hierarchy. Regions at selected levels from the hierarchy are shown in Figures 5.12(b)–(f). The number of regions at these levels are shown in Table 5.2. It is clear that the

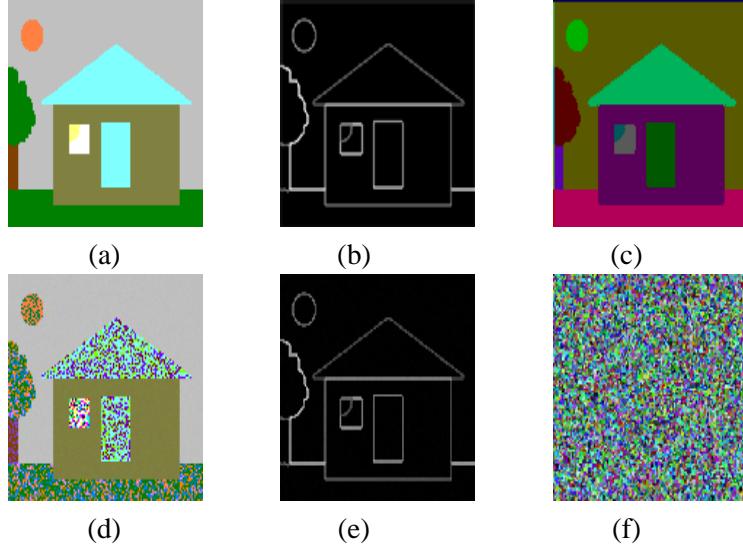


Figure 5.11: Implicit region merging: (a) A synthetic image (b) Gradient image (c) Watershed segmentation (d) Corrupted image ( $\sigma_n = 25\%$  of RMS gradient) (e) Gradient of (d) (d) Oversegmented result (without pre-processing)

number of regions reduces as we move up the hierarchy. The regions at level 784<sup>1</sup>, shown in Figure 5.12(d), represent a meaningful segmentation.

Consider the working of the implicit merging scheme for the lamp image in Figure 5.10(a). The image is filtered with  $\sigma_f = 1$  and the gradient is thresholded using  $T = 2\%$  of the RMS value. Implicit merging is applied to the watershed result shown in Figure 5.13(a). Selected levels of the resulting hierarchy are shown in Figure 5.13 and the number of regions at these levels are listed in Table 5.3. A combination of the pre-processing stage and implicit merging on regions from the watershed result effectively reduces oversegmentation of the image.

The application of the implicit merging scheme in the case of another real image is shown

---

<sup>1</sup>For convenience, the levels in the hierarchy are labeled from 0, which corresponds to the initial watershed result, to a maximum number that corresponds to the level with a single region.

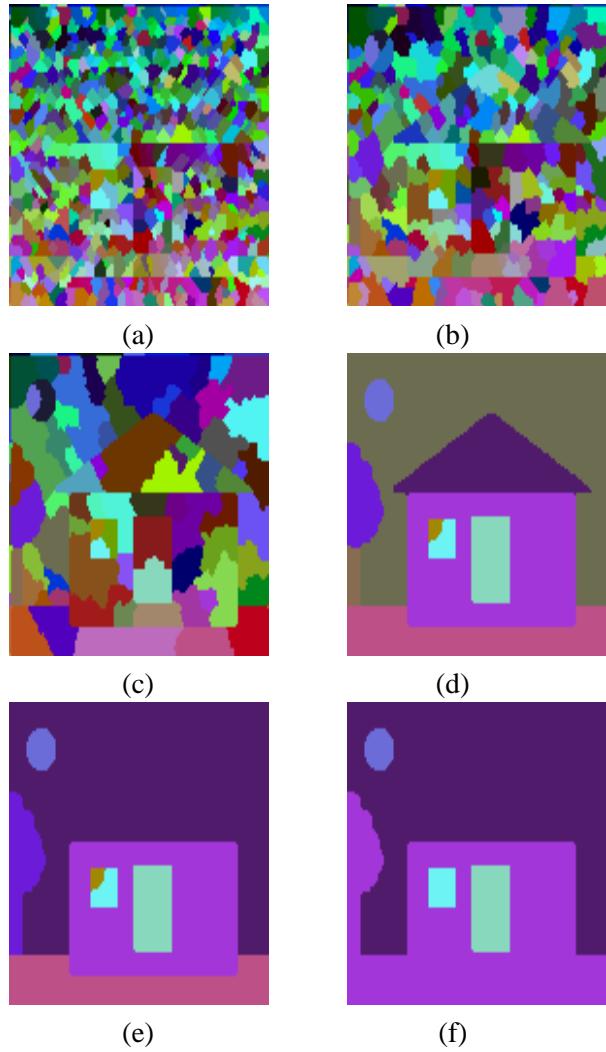


Figure 5.12: Hierarchy produced by implicit region merging: (a) Watershed segmentation of corrupted image ( $\sigma_f = 2; T = 5\%$ ). Regions at levels 460, 680, 784, 786, 789 are shown in (b)–(f).

Table 5.2: Implicit Region Merging for the house image (See Figure 5.12): Number of regions at different levels

Level	Number of regions
0	794
460	334
680	114
784	10
786	8
789	5

Table 5.3: Implicit Region Merging for the lamp image (See Figure 5.13): Number of regions at different levels

Level	Number of regions
0	1383
950	433
1250	133
1311	72

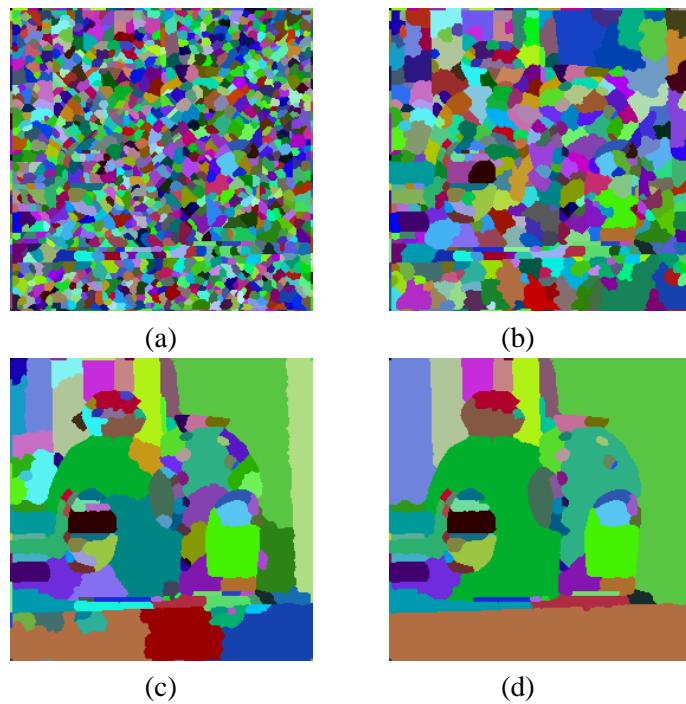


Figure 5.13: Hierarchy produced by implicit region merging: (a) Watershed segmentation for image in Figure 5.10(a) ( $\sigma_f = 1; T = 2\%$ ). Regions at hierarchy levels 950, 1250, 1311 are shown in (b)–(d).

Table 5.4: Implicit Region Merging for the road image (See Figure 5.14): Number of regions at different levels

Level	Number of regions
0	1480
775	705
1235	245
1395	85
1455	25
1468	12

in Figure 5.14. The image in 5.14(a) is noisy and textured. The image is filtered with  $\sigma_f = 1$  and the gradient is thresholded at  $T = 6\%$  of the RMS value. The watershed output is shown in Figure 5.14(c), which contains 1480 segments and is heavily oversegmented. Selected levels from the hierarchy produced by implicit merging are shown in Figures 5.14(d)–(h). The number of regions at these levels are shown in Table 5.4. The implicit merging scheme clearly helps to reduce oversegmentation. The regions at level 1468, shown in Figure 5.14(h) represent a meaningful segmentation.

To show the working of implicit merging on 3D images, we add noise to the image in Figure 5.4 using the *noiseGauss* function with  $\sigma_n = 0.1$ , which is 0.3% of the RMS gradient. We process the corrupted image by filtering with  $\sigma_f = 0.5$  and threshold the gradient at  $T = 0.2566\%$  of the RMS value. Watershed segmentation of this image produces the result in Figure 5.15(a), which contains 864 regions. Selected levels from the hierarchy produced by implicit merging are shown in Figures 5.15(b)–(d). The number of 3D regions at these levels are indicated in Table 5.5. The region merging scheme does reduce oversegmentation and produces a meaningful

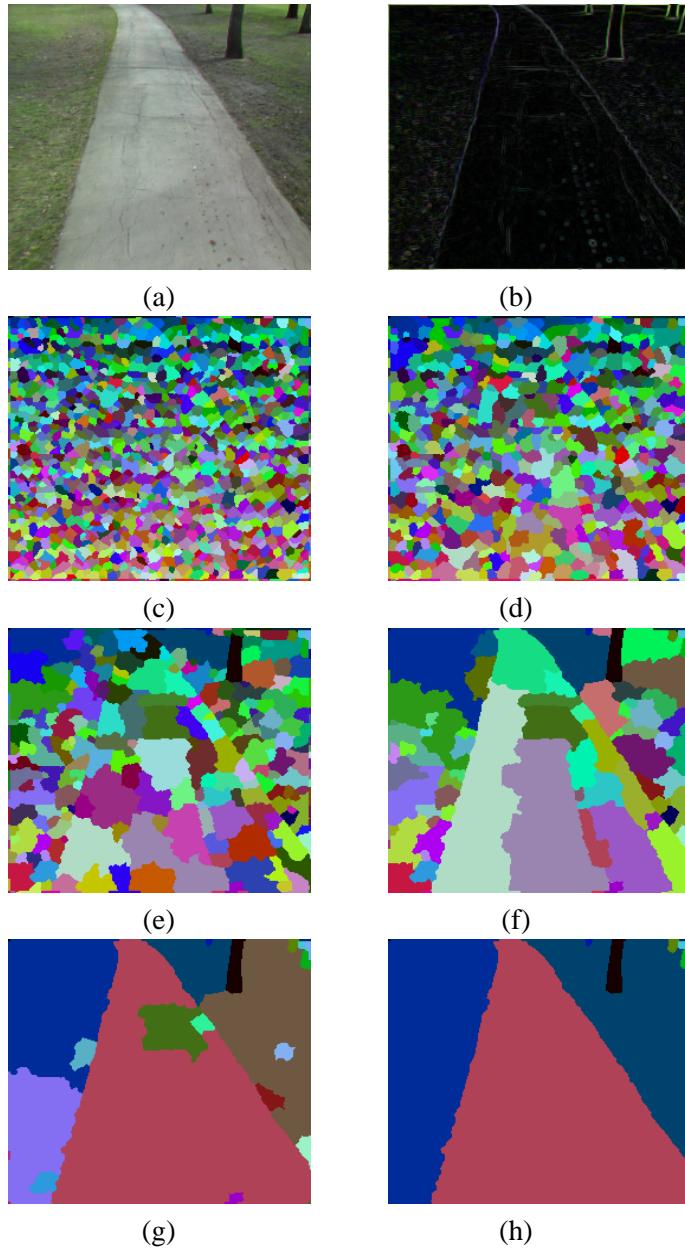


Figure 5.14: Hierarchy produced by implicit region merging: (a) A real image — noisy and textured (b) Gradient image (c) Watershed segmentation ( $\sigma_f = 1; T = 6\%$ ). Regions at hierarchy levels 775, 1235, 1395, 1455, 1468 are shown in (d)–(h).

Table 5.5: Implicit Region Merging for the 3D phantom image of the head (See Figure 5.15): Number of 3D regions at different levels

Level	Number of 3D regions
0	864
1	302
211	91
292	10

segmentation at level 292. The segmented slices at this level are shown in 5.15(d).

### 5.3.2 Seeded Region Merging

In the case of seeded region merging, the user picks a *segment* of interest from the original image to grow this segment from the oversegmented result and also study its merging with adjacent segments. Consider the corrupted house image in Figure 5.11(d) and the oversegmented watershed result in Figure 5.11(f). To start the seeded merging process, we choose the sun in the original image as the seed. The seed is indicated in Figure 5.16(a). Selected levels from the hierarchy produced by seeded region merging are shown in Figures 5.16(b)–(e). The growing of the seed segment is complete in the Figure 5.16(d). Higher levels of the hierarchy show the merging of this segment with others. For instance, consider Figure 5.16(e), where the seed segment merges with the background leaving an outline of the tree, the road, and the house.

Consider the application of seeded region merging for the road image in Figure 5.17(a). We process this image by filtering with  $\sigma_f = 1.5$  and threshold the gradient image at  $T = 6\%$  of the RMS value. The watershed segmentation result is shown in Figure 5.17(b), which is an oversegmented result. To start the merging process, we select the road in the image as the

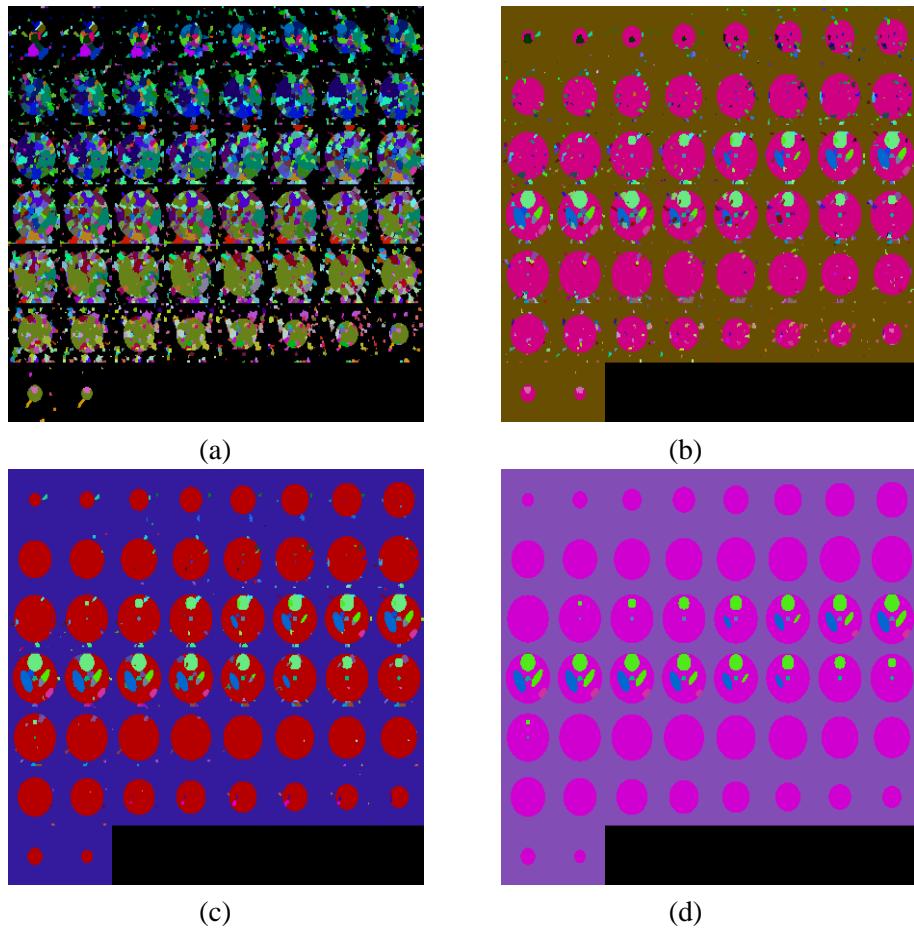


Figure 5.15: Hierarchy produced by implicit region merging: (a) Watershed segmentation of Figure 5.4 after addition of noise ( $\sigma_f = 0.5$ ;  $T = 0.2566\%$ ). Regions at levels 1, 211, 292 are shown in Figures 5.15(b)–(d).

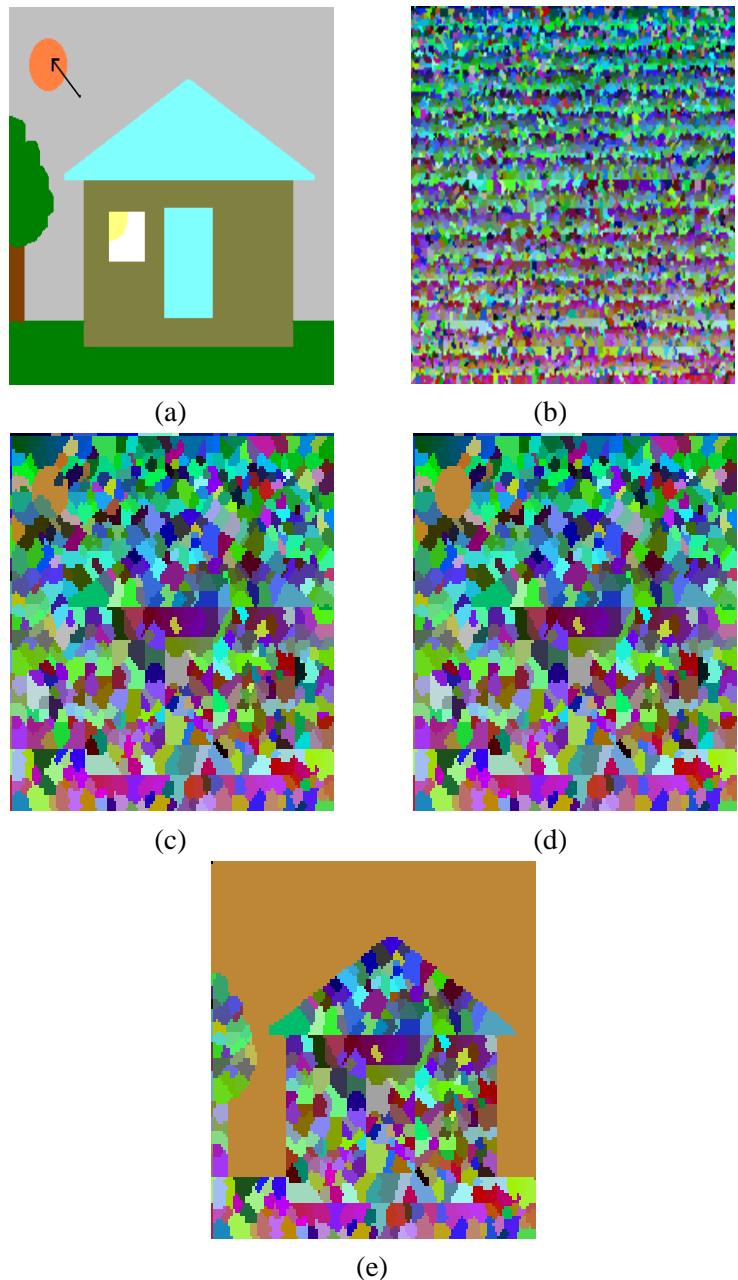


Figure 5.16: Hierarchy produced by seeded region merging: (a) Input image showing the seed segment (b) Oversegmented result ( $\sigma_f = 2$ ;  $T = 5\%$ ) (c)&(d) Growing of the seed segment (e) Merging of the seed segment with the background in the image

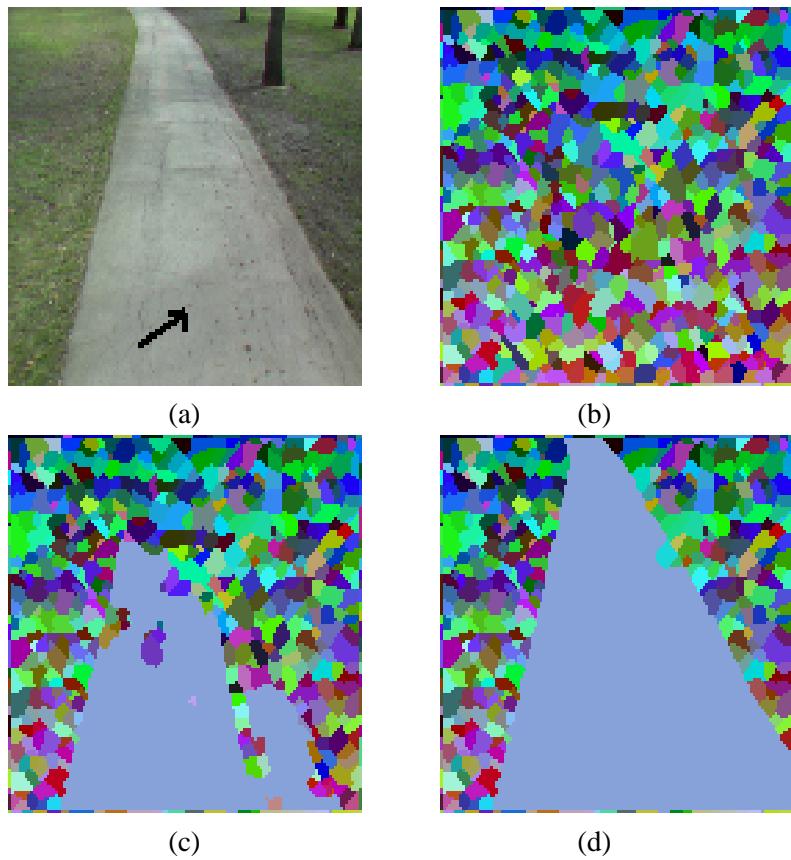


Figure 5.17: Hierarchy produced by seeded region merging: (b) Input image showing the seed segment (a) Watershed segmentation ( $\sigma_f = 1.5$ ;  $T = 6\%$ ) (c)&(d) Growing of the seed segment

seed, as indicated in 5.17(a). We have shown two selected levels from the hierarchy in Figures 5.17(c)–(d). The growing of the selected seed, the road in this case, is complete in the Figure 5.17(d).

## 5.4 Segmentation of Slices of a 3D Anatomical Image

We have used our implementation of the watershed algorithm and the region merging schemes to segment slices of a 3D anatomical image obtained from the datasets of the Visible Human Project of the National Library of Medicine [47]. Selected 16 slices, numbered 1653–1668, from the dataset obtained from the male cadaver are shown in Figure 5.18. We see that the images are complex and highly textured. The slices are filtered with the *gauss* function in VISPack with input standard deviation  $\sigma_f = 1$ . The slices of the gradient image are shown in Figure 5.19. The gradient is thresholded at  $T = 9.63\%$  of the RMS value. The slices from the result of the watershed segmentation are shown in Figure 5.20. This output image contains 13,898 regions and is heavily oversegmented. Therefore, we apply the region merging schemes to obtain more meaningful results.

The regions in the initial watershed segmentation, in Figure 5.20 are subject to merging using the implicit merging scheme. Selected levels of the hierarchy produced by implicit merging for slices 1653 and 1668 are shown in Figures 5.21 and 5.22. The number of regions at these levels for each of the two slices are shown in Tables 5.6 and 5.7. From these results, we see that the merging scheme reduces the oversegmentation and produces a hierarchy from which the user can choose a meaningful result.

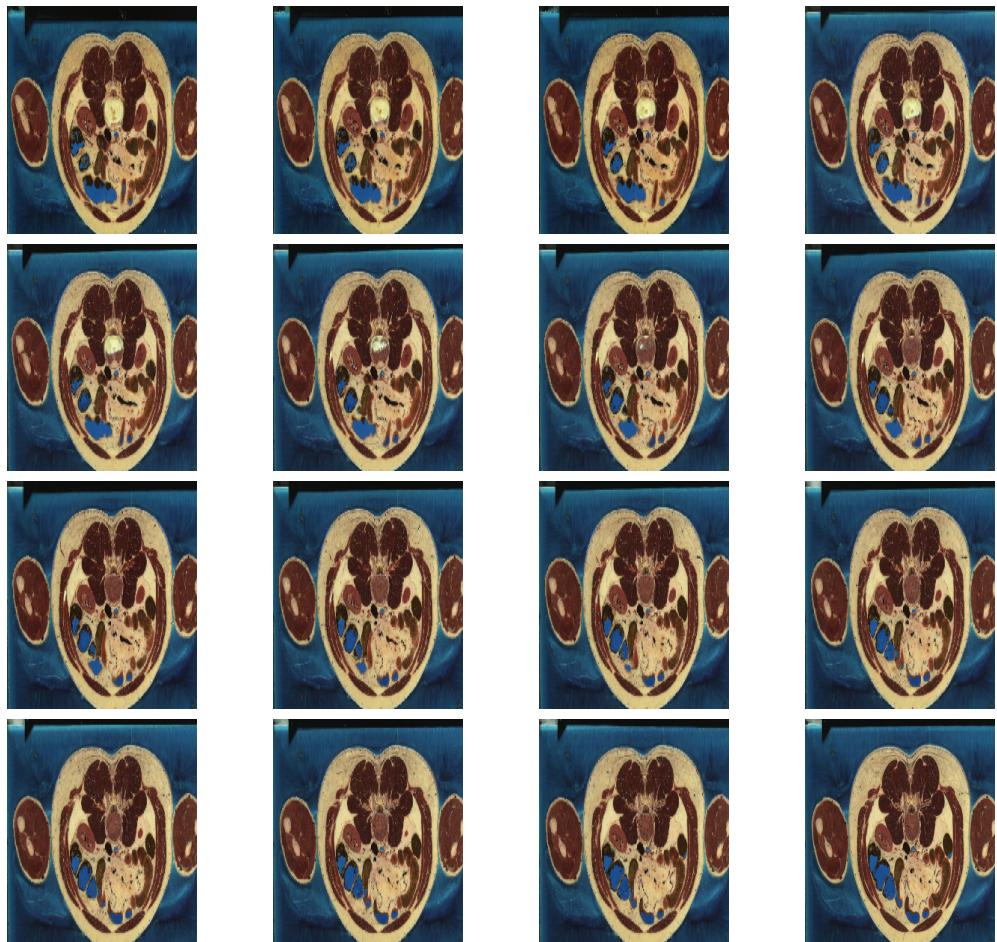


Figure 5.18: Slices of a 3D medical image

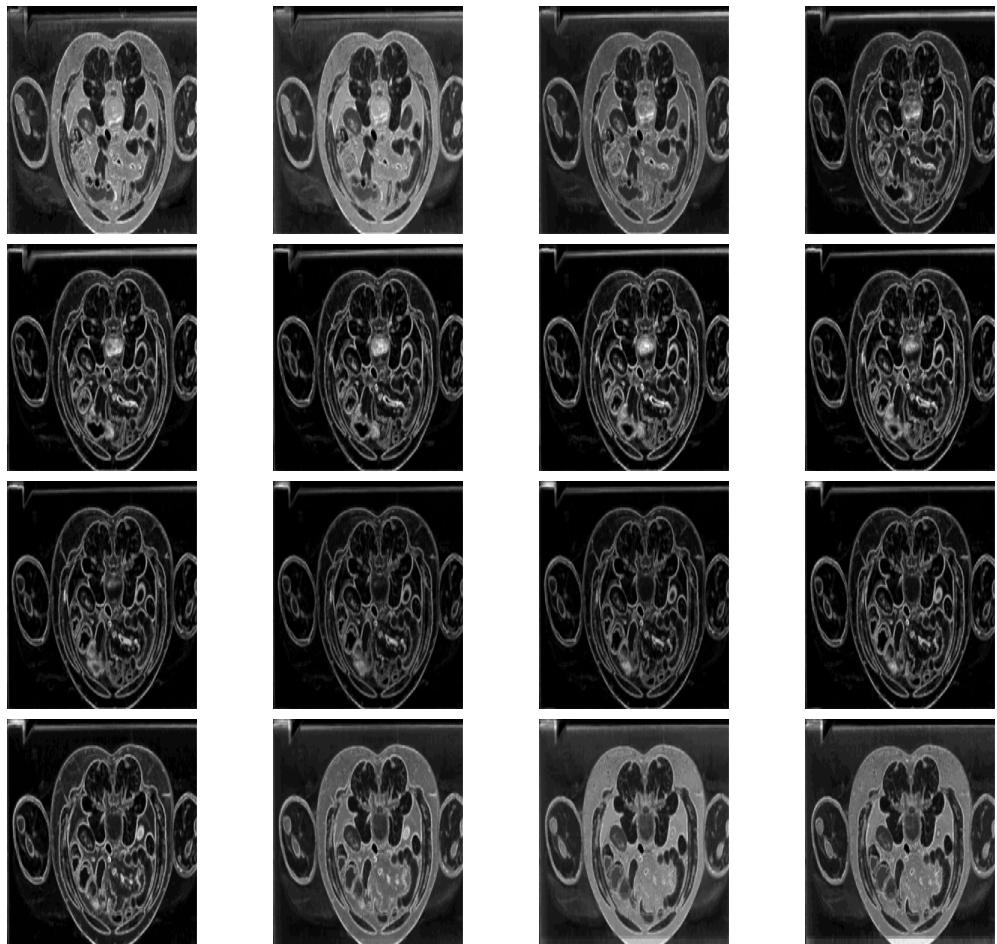


Figure 5.19: Slices of the gradient image

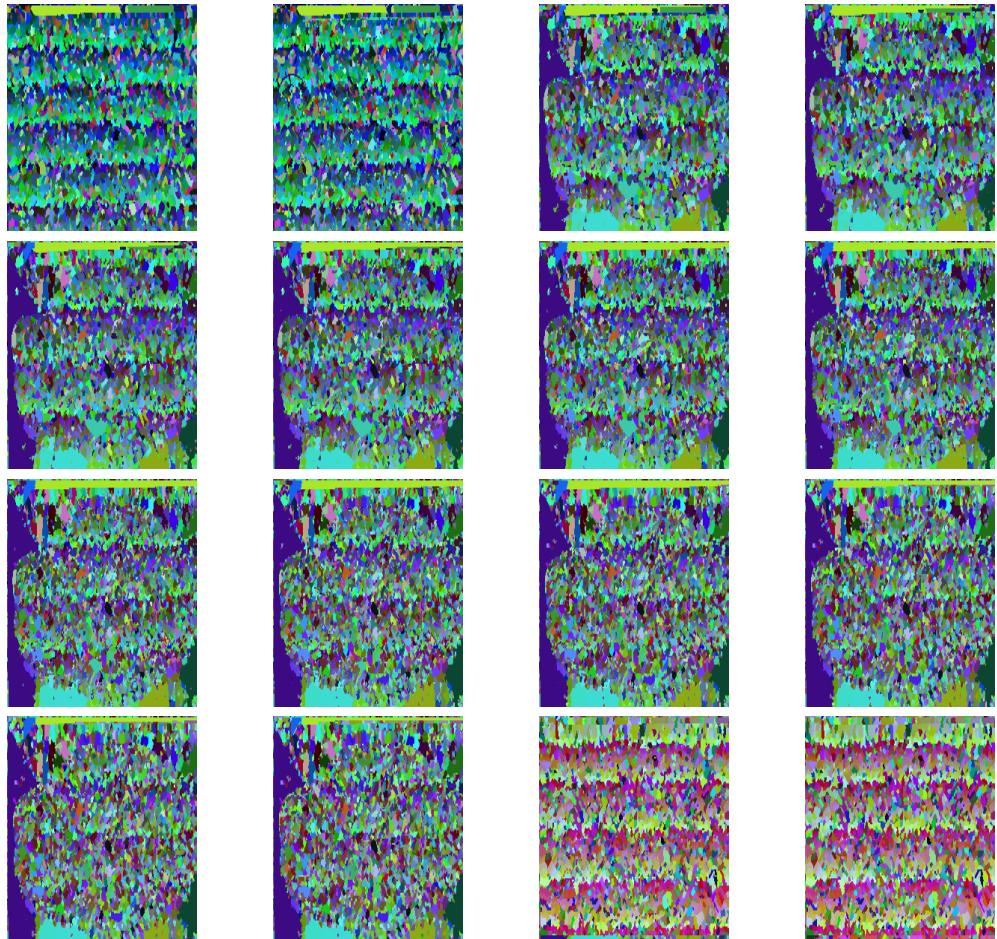


Figure 5.20: Oversegmented watershed result ( $\sigma_f = 1$ ;  $T = 9.63\%$ )

Table 5.6: Implicit Region Merging for slice 1653 (See Figure 5.21) : Number of regions at different levels

Level	Number of regions
0	13898
12400	1275
12592	1083
13000	674
13320	354
13480	194
13580	94

Table 5.7: Implicit Region Merging for slice 1668 (See Figure 5.22) : Number of regions at different levels

Level	Number of regions
0	13898
12200	1475
12750	925
12975	699
13225	449
13405	269
13515	159

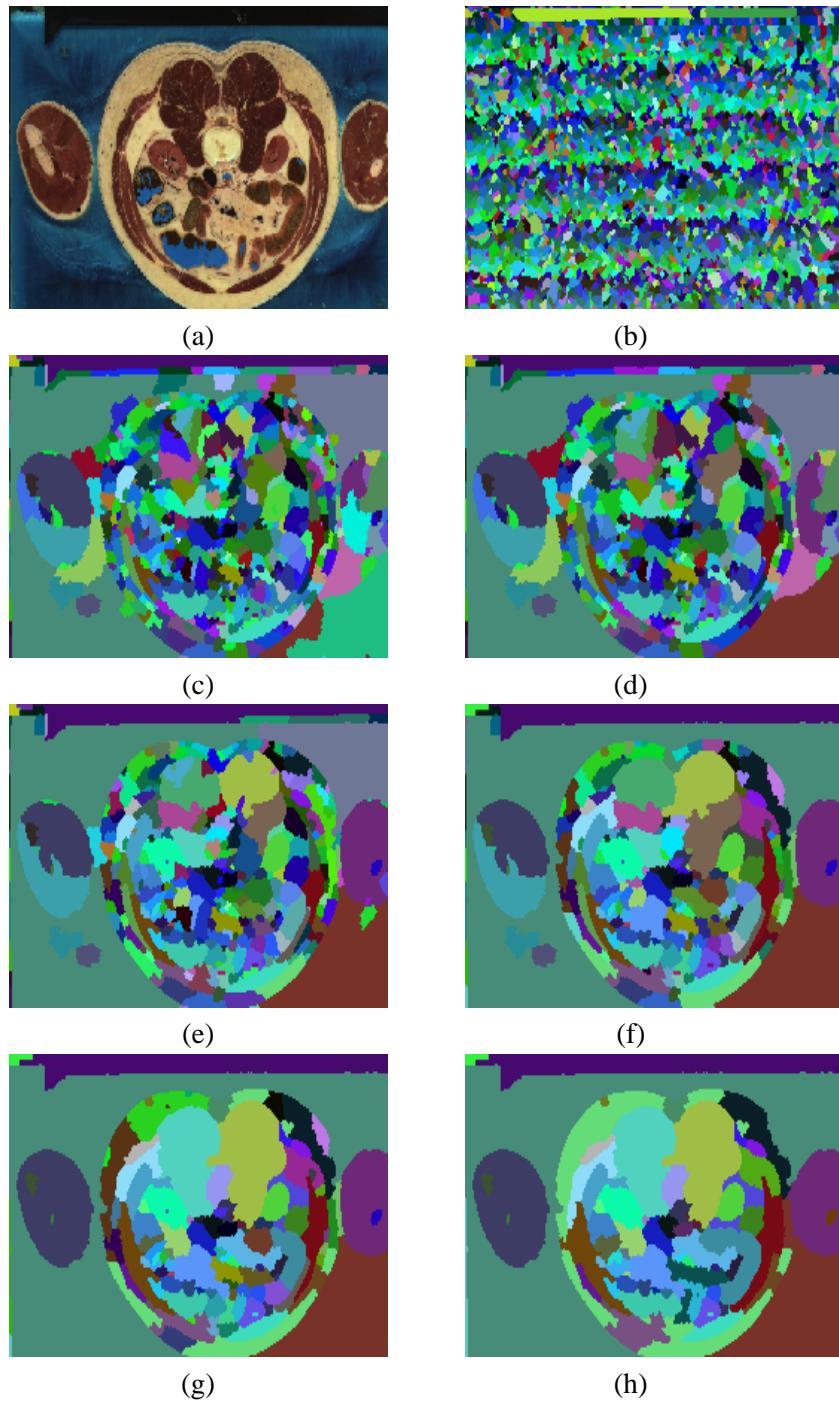


Figure 5.21: Result of implicit region merging: (a) Slice 1653 (b) Oversegmented watershed result. Regions at levels 12400, 12592, 13000, 13320, 13480, 13580 are shown in (c)–(h).

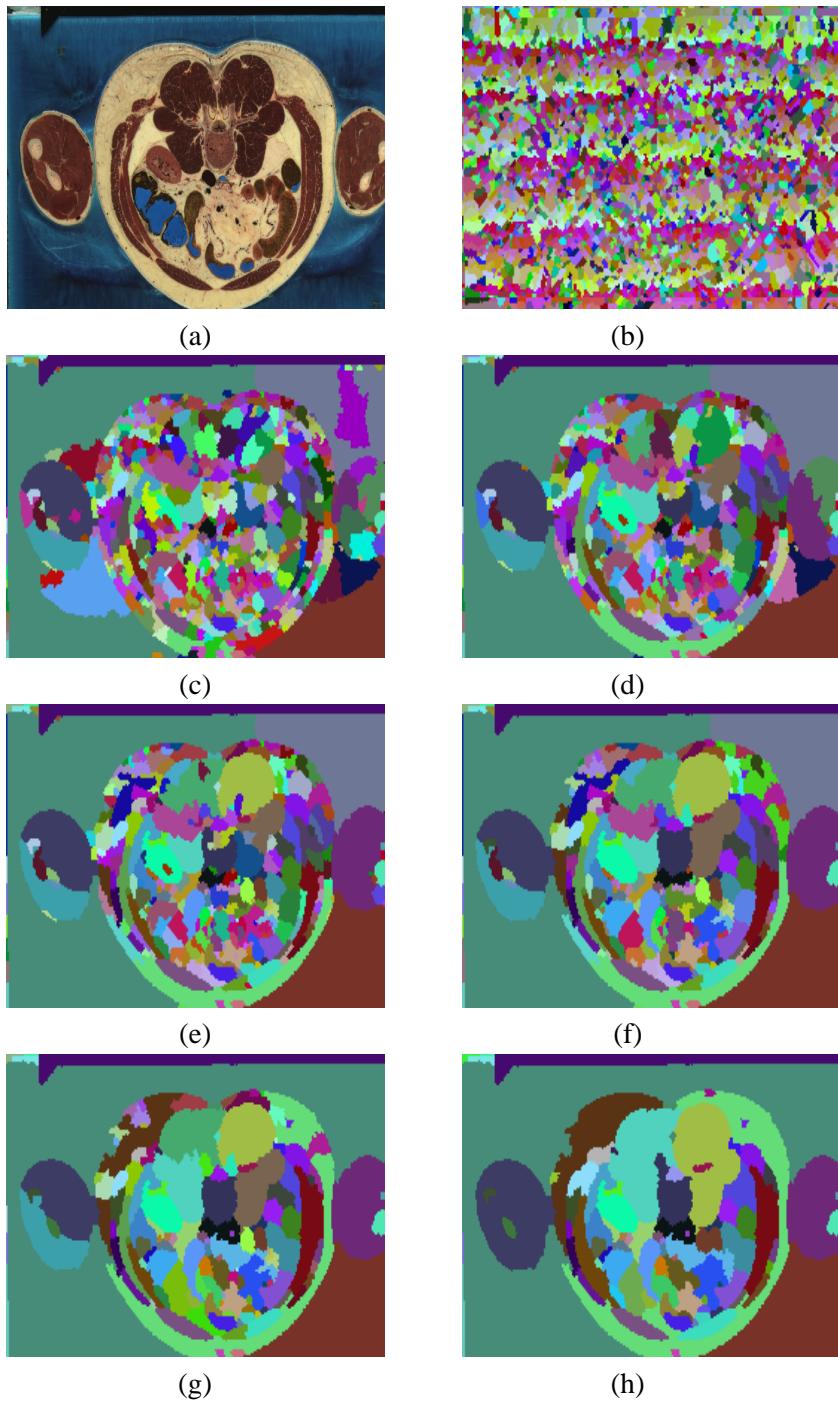


Figure 5.22: Result of implicit region merging: (a) Slice 1668 (b) Oversegmented watershed result. Regions at levels 12200, 12750, 12975, 13225, 13405, 13515 are shown in (c)–(h).

Specific segments can be grown from the watershed result of the slices using the seeded merging scheme. We here show the use of seeded merging to grow two different segments in the 3D image. Consider the seed segment indicated in Figure 5.23(a). The growing of this segment in slice 1653 is shown in Figures 5.23(b)–(d). The growing of the same seed segment for slice 1663 is shown in Figure 5.23(f)–(h). Consider another seed segment indicated in Figure 5.24(a). Growing of this segment in slices 1657, 1660, 1663, and 1666 are shown in Figures 5.24(b),(d),(f)&(h) respectively.

## 5.5 Streaming Algorithm

We have tested our implementation of the streaming algorithm using images that represent several special cases, particularly the cases of split flat regions and re-entering segments. The first test image contains a single object against a background. This is an image of the letter ‘A’ and mainly tests the algorithm for correctness in the presence of split flat regions. We split the image in Figure 5.25(a) into overlapping blocks. Let  $X \times Y$  denote the size of the matrix of blocks. Therefore in this case,  $X = Y = 2$ . The blocks are shown in Figure 5.25(b). The splitting causes the segment ‘A’ to be split into four segments in the four different blocks. The watershed segmentation of the individual blocks are shown in Figure 5.25(c), in which segments carrying different labels are represented by different colors. In these images, the four parts of the letter ‘A’ are identified as four different segments against different backgrounds. The watershed result after pairwise processing of adjacent blocks is shown in 5.25(d). From the result, we see that the algorithm is able to identify the equivalence of the split portions of the letter ‘A’ and its

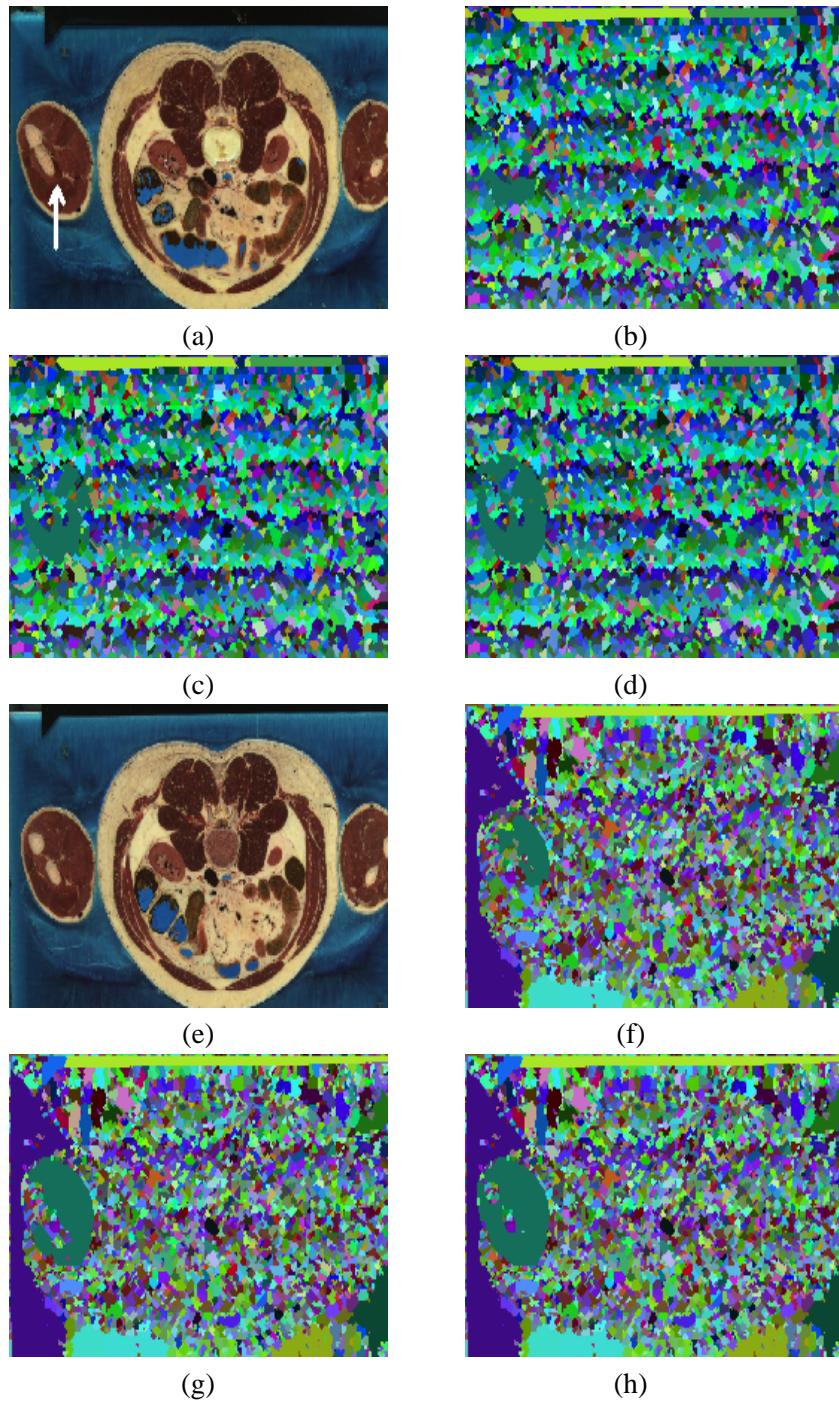


Figure 5.23: Result of seeded region merging: (a) Slice 1653 showing seed segment. Growing of the segment is shown in (b)–(d). (e) Slice 1663. Growing of the segment is shown in (f)–(h).

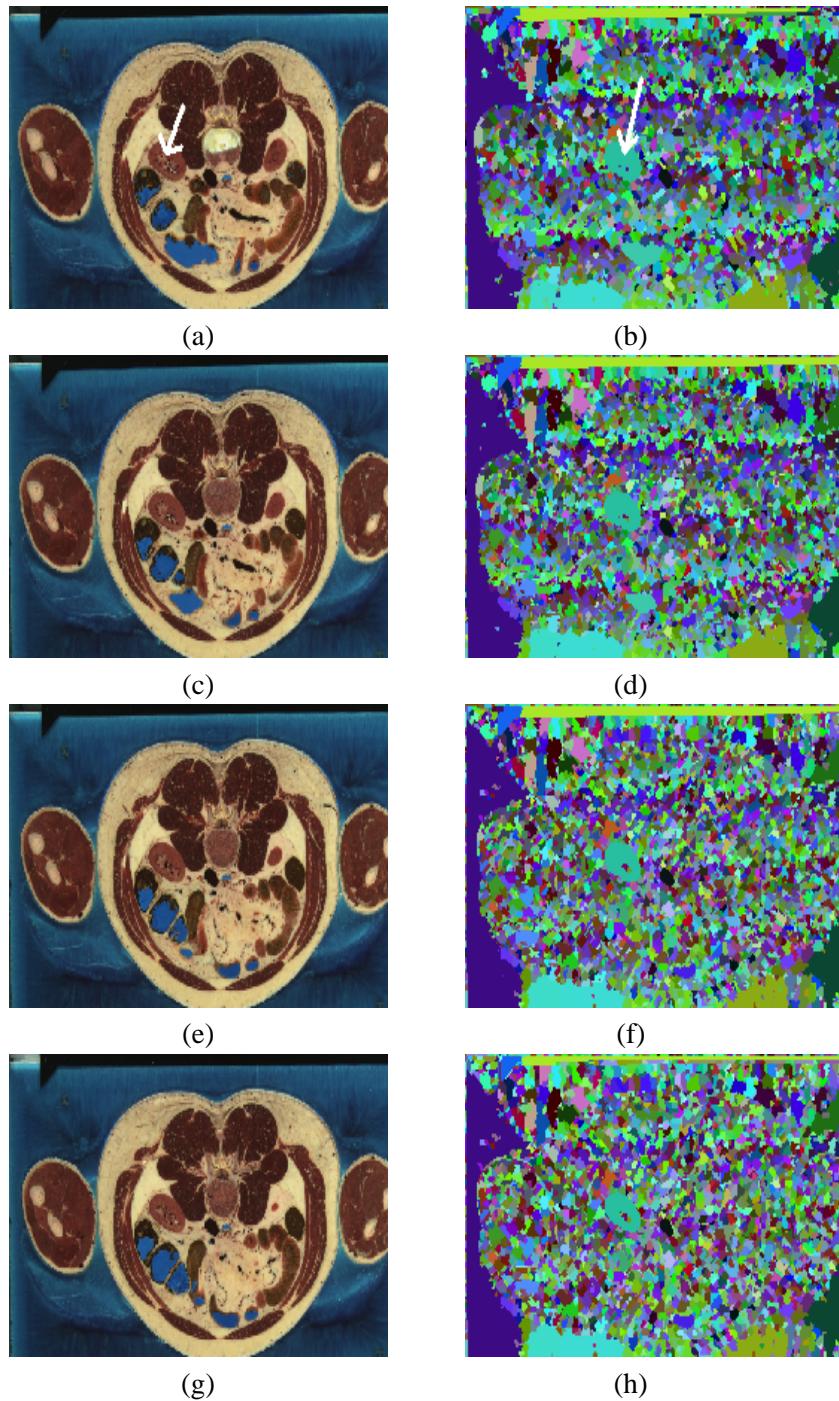


Figure 5.24: Result of seeded region merging: (a) Slice 1657 showing seed segment. (b) Grown segment. Growing of the segment for the slices 1660, 1663, 1666 in (c),(e),(f) are shown in (d),(f),(h) respectively.

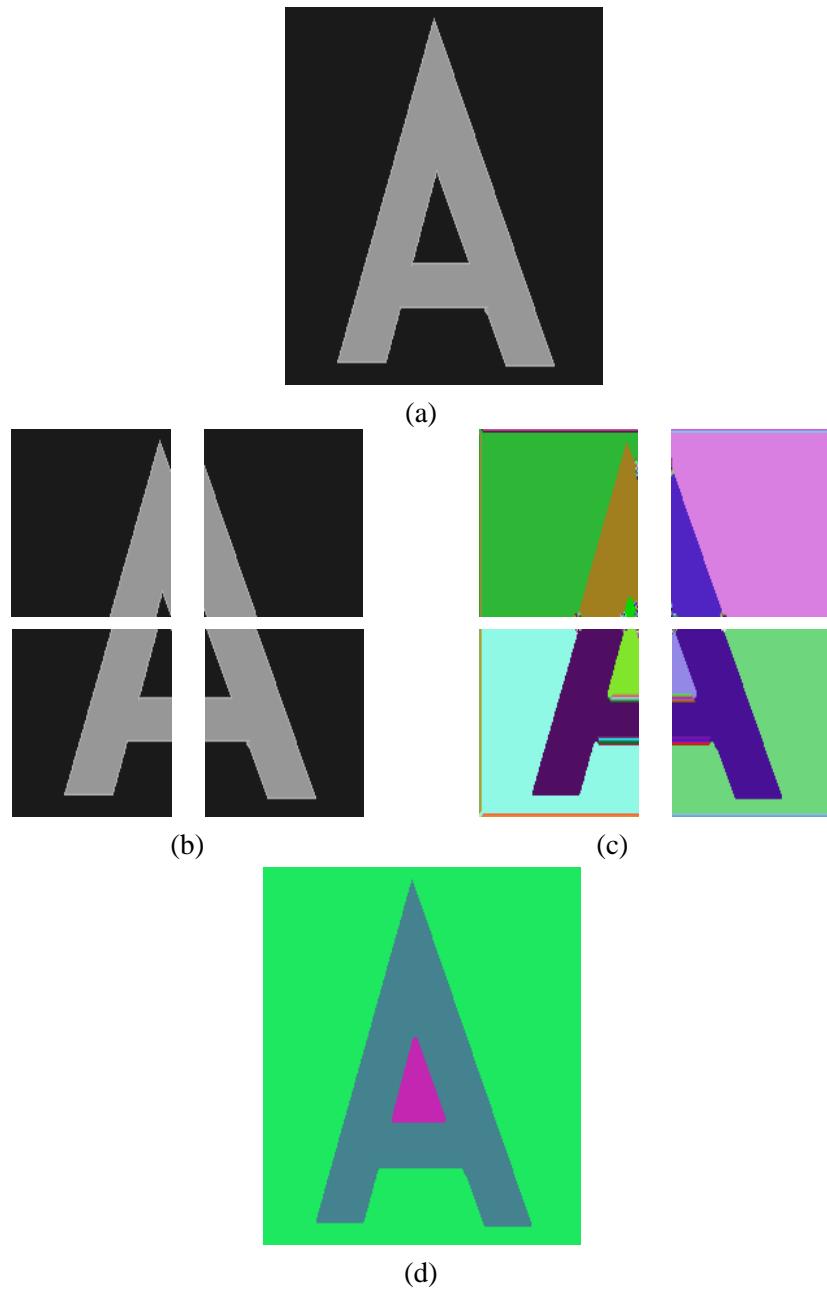


Figure 5.25: Streaming algorithm: (a) Test image — letter ‘A’ (b) Splitting of the image (c) Watershed result before pairwise processing (d) Result of the streaming algorithm

background and produces a segmentation which is meaningful for the given image.

The next set consists of the image of the letter ‘Z’, a logo image and two other images and has been used specifically to test the working of the streaming algorithm for the case of re-entering segments. The first of this set, the image of the letter ‘Z’ in Figure 5.26(a), is split into blocks as shown in Figure 5.26(b). In this case,  $X = 2$  and  $Y = 3$ . The letter ‘Z’ is split into different segments across the different blocks. Here again, the splitting gives rise to the case of split flat regions. Note the additional special case of a re-entering segment in the block at position (1,2) in the matrix of blocks. Although the two segments in this block are parts of the letter ‘Z’, the watershed result of this block, shown in Figure 5.26(c) identifies them separately because they are not connected. After processing of pairs of adjacent blocks in Figure 5.26(c) is complete, the segmentation in Figure 5.26(d) is produced. This segmentation is meaningful for the given image. This result shows that the algorithm works well even in the presence of re-entering segments in a block.

The logo image shown in Figure 5.27(a) is split into blocks as shown in the Figure 5.27(b), such that  $X = Y = 2$ . Note that the blocks in positions (1,1) and (2,2) in the matrix in Figure 5.27(b) pose the problem of re-entering segments. The watershed segmentation of the individual blocks are shown in Figure 5.27(c). Note that parts of the same segment have been identified differently in the blocks at positions (1,1) and (2,2) in the matrix. This is because the different parts of the same segment are not connected. Pairwise processing of blocks helps to identify the equivalence of these re-entering segments and other split flat regions to produce a meaningful segmentation, which is shown in Figure 5.27(d).

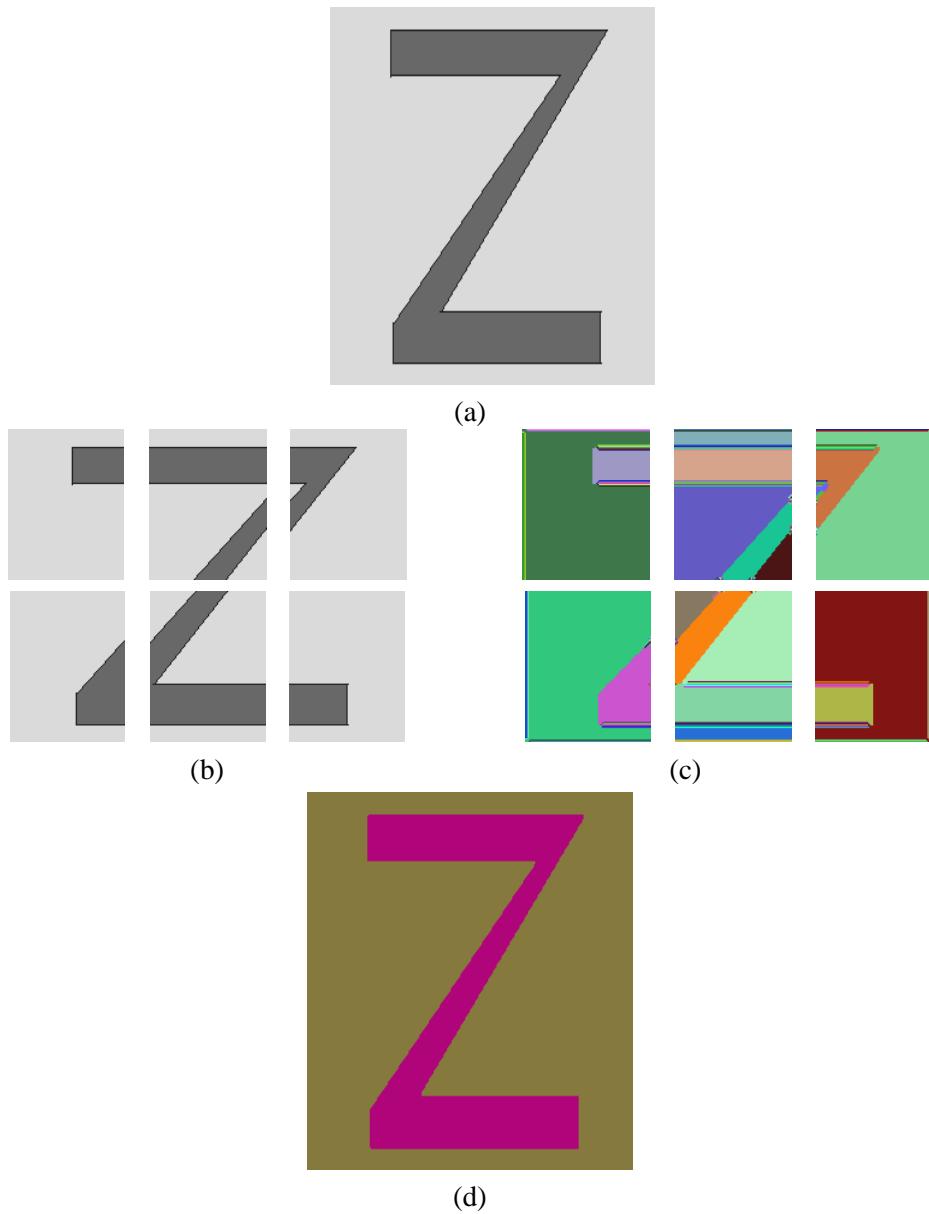


Figure 5.26: Streaming algorithm: (a) Test image (b) Splitting of the image (c) Watershed result before pairwise processing (d) Result of the streaming algorithm

The test image in Figure 5.28(a) is split into blocks such that  $X = Y = 2$ . The splitting is shown in Figure 5.28(b). Note that the block at position (1,2) in the matrix of blocks contains a segment that re-enters the same block twice. The output of the watershed algorithm for the blocks is shown in 5.28(c). As expected, the watershed result for the block at position (1,2) considers the same re-entering segment as three different segments. However, the pairwise processing scheme serves to identify the equivalence of parts of the re-entering segment and produces a meaningful segmentation, as shown in Figure 5.28(d).

Consider the splitting of the segment in the image in Figure 5.29(a). Here again,  $X = Y = 2$  and the overlapping blocks are shown in Figure 5.29(b). Re-entering segments are present in blocks at positions (1,1), (1,2), and (2,2) in the matrix of blocks. Parts of the same re-entering segment are identified as separate segments in the watershed result of the blocks, as shown in Figure 5.29(c). Our pairwise processing scheme helps to identify the equivalence of these parts and produces a meaningful result, which is shown in Figure 5.29(d).

The next test image contains multiple objects against a background and is shown in Figure 5.30(a). The blocks produced by splitting, their individual segmentations, and the result of the streaming algorithm are shown in 5.30(b)–(d) respectively. Note that the block at position (1,2) in the matrix of blocks in Figure 5.30(b) contains a re-entering segment.

From our experiments, we can see that the algorithm works well for all our test images, including the ones whose splitting presents the special cases of split flat regions and re-entering segments. We have used the streaming algorithm to segment slice 1653 from the medical data set. The splitting of the gradient of this slice, the individual segmentations of the blocks before

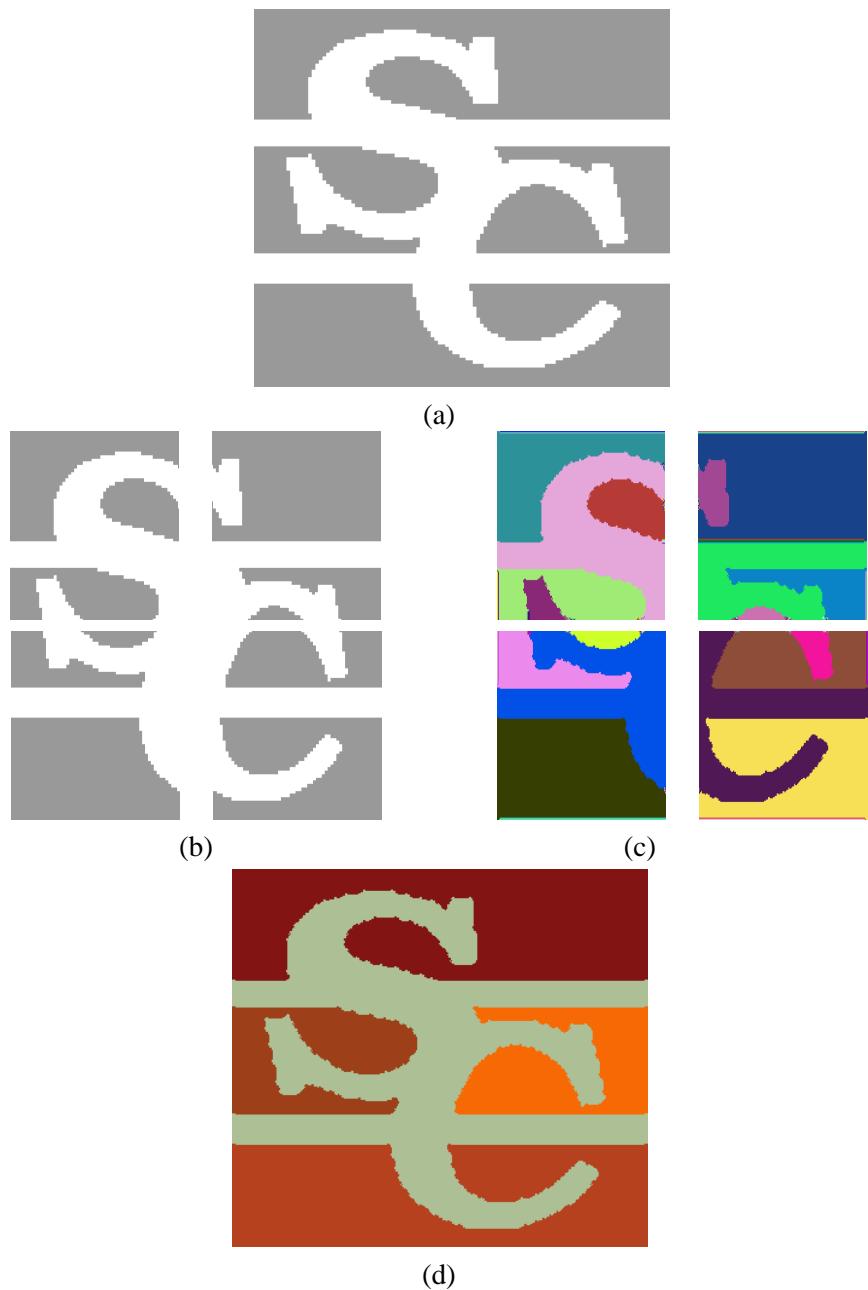


Figure 5.27: Streaming algorithm: (a) Test image (b) Splitting of the image (c) Watershed result before pairwise processing (d) Result of the streaming algorithm

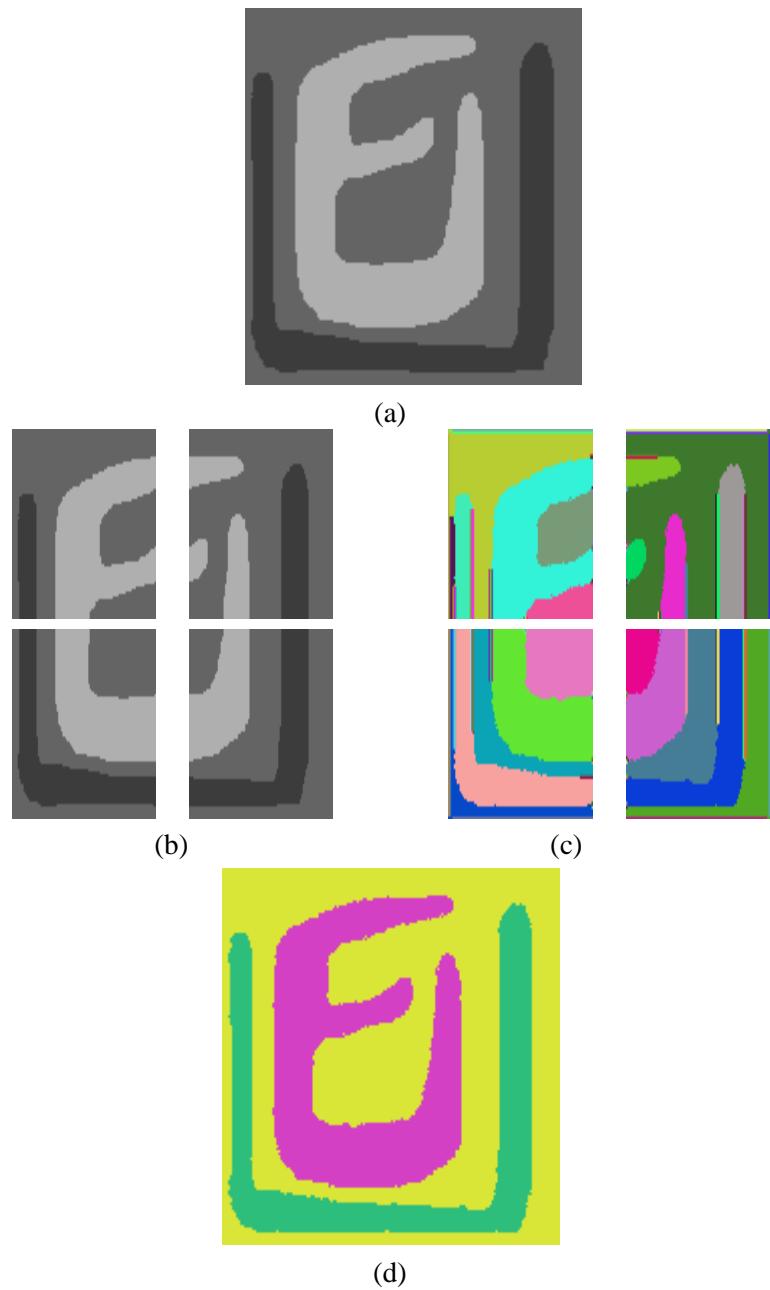


Figure 5.28: Streaming algorithm: (a) Test image (b) Splitting of the image (c) Watershed result before pairwise processing (d) Result of the streaming algorithm

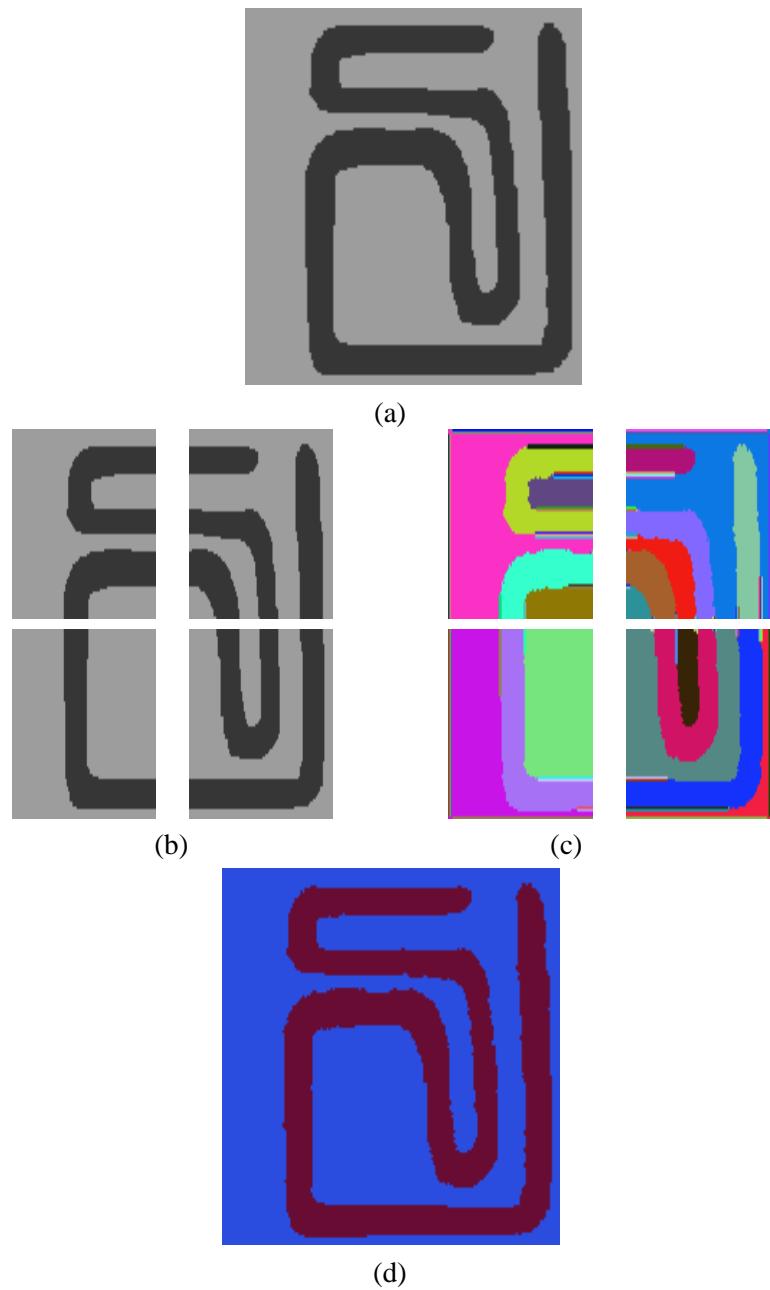


Figure 5.29: Streaming algorithm: (a) Test image (b) Splitting of the image (c) Watershed result before pairwise processing (d) Result of the streaming algorithm

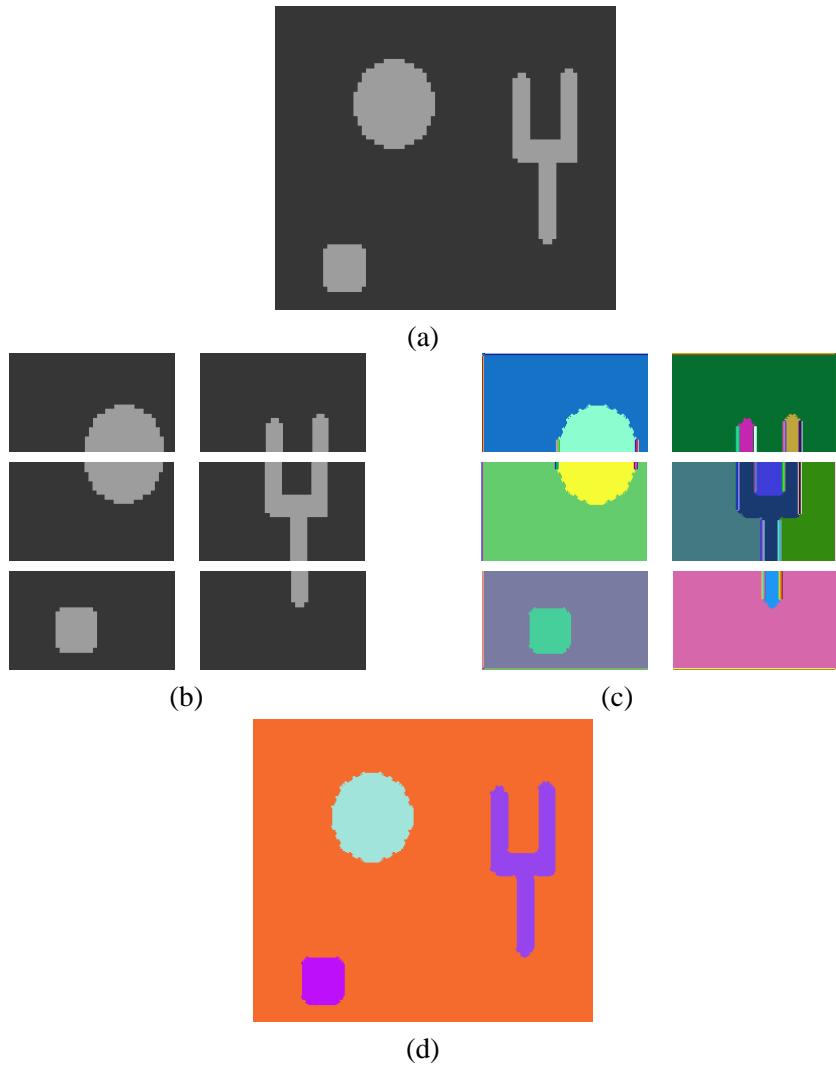


Figure 5.30: Streaming algorithm: (a) Test image (b) Splitting of the image (c) Watershed result before pairwise processing (d) Result of the streaming algorithm

pairwise processing, and the result of the streaming algorithm are shown in Figure 5.31(b)–(d).

The final result represents a meaningful segmentation.

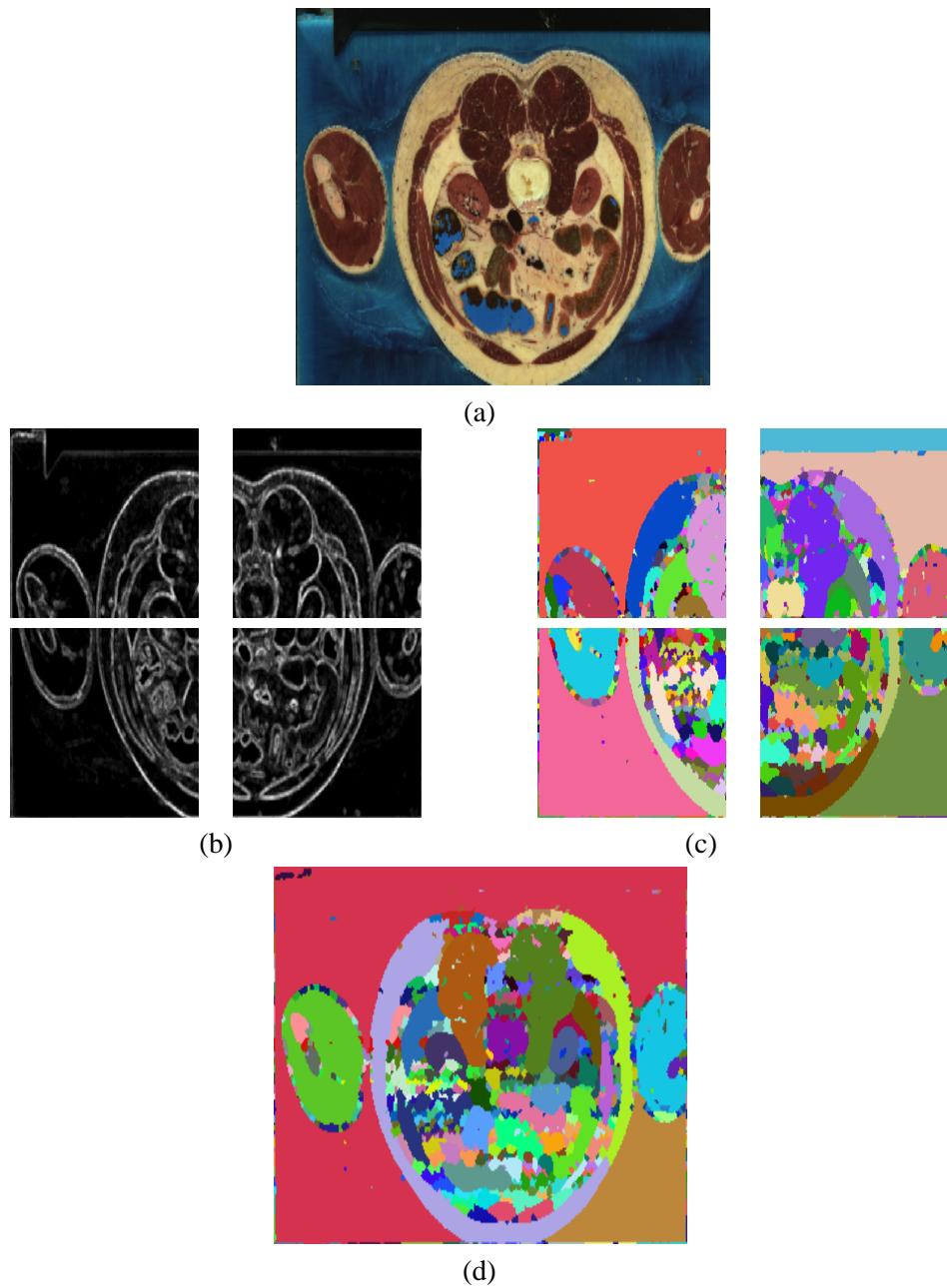


Figure 5.31: Streaming algorithm: (a) Test image (b) Splitting of the image (c) Watershed result before pairwise processing (d) Result of the streaming algorithm

# **Chapter 6**

## **Conclusion**

We have implemented a semi-automatic segmentation scheme based on the watershed algorithm. This scheme involves three stages: pre-processing of the image to reduce the effects of noise, segmentation using the watershed algorithm, and post-processing by region merging. We have also tested a streaming algorithm for segmentation based on the watershed algorithm, in which blocks of an image are segmented individually and the results from these blocks are used to obtain the segmentation of the entire image. In this chapter, we briefly discuss the advantages of our scheme and follow this with a discussion of the disadvantages, and scope for future work.

For the semi-automatic segmentation scheme, we pre-process the image by filtering with the linear Gaussian filter. This is followed by thresholding of the edge map at a chosen percentage of the RMS value. This step serves to reduce the effects of noise and hence the number of regions in the watershed result, to which the merging schemes are applied.

We have implemented rainfalls simulation for watershed segmentation in 2D and 3D. In

the rainfaling approach, the tracing of every pixel in the image can be done independently of tracing of other pixels in the image. This makes the implementation easily extensible for a streaming or parallel implementation. The searching step occurs in a single width-first scan of the image. We have used an 8-neighborhood in our implementation (26-neighborhood in 3D), which causes significant reduction in the number of minima detected in the searching step and thereby, considerably reduces oversegmentation. The implementation of the algorithm allows for the occurrence of minima along the image boundaries. In addition, our watershed implementation produces a segmentation with zero-width watershed lines. This means that every pixel is uniquely assigned to a region.

Although the pre-processing step serves to reduce the number of regions in the output of the watershed algorithm, it does not help in removing oversegmentation to produce a meaningful result. We have implemented two region merging schemes to generate a hierarchy of regions from which meaningful segmentations can be easily chosen. The implicit region merging scheme automatically produces a hierarchy using the watershed depth of regions. The user can simply choose the required segmentation from this hierarchy. Using the seeded region merging scheme, the user can select a specific segment from the image as the seed, grow this segment from the watershed result, and study its merging with adjacent segments. The data structures used to represent regions are independent of dimension.

The only free parameters involved in our semi-automatic segmentation scheme are the standard deviation input to the *gauss* function in VISPack and the threshold parameter, as a % of the RMS value. However, for a particular application, these parameters can be tuned to produce the

required segmentation. Thus, for a specific application, our segmentation scheme can be made almost fully automatic.

The graphical user interface to our implementation of the semi-automatic segmentation scheme facilitates execution of the watershed segmentation for user-chosen values of the standard deviation input to the Gaussian filter and the threshold, and viewing of the hierarchy. The hierarchy of regions can be saved to disk as a combination of two files in the `.ws` and `.hr` formats. The entire hierarchy can be regenerated from these two files.

We have validated the streaming algorithm through testing using several images. Our streaming algorithm can compute the segmentation of the image without iterative processing of adjacent blocks. It can also be extended to work on 3D images and when combined with a binary reduction scheme, it can lead to an efficient parallel implementation.

A number of improvements can be made on our existing implementations, thus leaving a good scope for future research. We discuss some of these as follows.

1. The linear Gaussian filter, depending on the input standard deviation, may smooth out edges in the image. Hence, filtering of the image can be done using edge-preserving methods. One such scheme is the anisotropic diffusion method proposed by Perona and Malik [51] and is used in connection with watershed segmentation in [21].
2. The result of the watershed algorithm is highly dependent on the input edge map. The input edge map can be chosen depending on the problem at hand to achieve good segmentation. For instance, Mangan and Whitaker [39] use curvature values as input to the watershed algorithm. Zenzo [70] discusses computation of gradient for multi-spectral

images by finding the direction of maximum change.

3. Although the rainfalls watershed algorithm carries advantages, it has some problems that many discrete implementations of the watershed algorithm have. A typical example of such problems is the presence of two equal-valued minima at different locations along the boundary of a flat region. Depending on the nature of the input image, such problems may result in pixel level inaccuracies. Thus, obtaining a segmentation which is accurate at the pixel level and also at the sub-pixel level [64] is a good area for further research.
4. The implementation of the algorithms can be further optimized with regard to speed and memory requirements. Specifically, the streaming algorithm can be used to implement an efficient parallel algorithm by combining it with a scheme for binary reduction of blocks.
5. In the streaming algorithm, the size of the blocks can be reduced by using an an overlap of one column (or row). The scheme for pairwise processing of blocks can be modified accordingly and the working of the algorithm can be tested.
6. The user interface for 3D images can be extended for 3D viewing of results of the watershed algorithm using OpenGL related libraries.

# **Bibliography**

# Bibliography

- [1] R. Adams and L. Bischof. Seeded region growing. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16(6):641–647, June 1994.
- [2] C. Amoroso, E. Ardizzone, V. Morreale, and P. Storniolo. A new technique for color image segmentation. In *Proceedings of the Tenth International Conference on Image Analysis and Processing (ICIAP)*, 1999.
- [3] M. C. Andrade, G. Bertrand, and A. A. Araújo. Segmentation of microscopic images by flooding simulation: A catchment basins merging algorithm. In *Nonlinear Image Processing, Proc. SPIE*, volume 3026, pages 164–175, 1997.
- [4] M. Baccar, L. A. Gee, R. C. Gonzalez, and M. A. Abidi. Segmentation of range images via data fusion and morphological watersheds. *Pattern Recognition*, 29(10):1671–1685, 1996.
- [5] S. Beucher. Watersheds of functions and picture segmentation. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1982.

- [6] S. Beucher. Watershed, hierarchical segmentation and waterfall algorithm. *Mathematical Morphology and Its Applications to Image Processing*, pages 69–76, 1994.
- [7] S. Beucher and C. Lantuejoul. Use of watersheds in contour detection. In *International Workshop on image processing: Real-time edge and motion detection/estimation*, September 1979.
- [8] S. Beucher and F. Meyer. The morphological approach to segmentation: The watershed transformation. *Mathematical Morphology in Image Processing*, pages 433–481, 1993.
- [9] J. R. Beveridge, J. Griffith, R. R. Kohler, A. R. Hanson, and E. M. Riseman. Segmenting images using localized histograms and region merging. *International Journal of Computer Vision*, 2:311–347, 1989.
- [10] G. A. Borges and M. J. Aldon. A split-and-merge segmentation algorithm for line extraction in 2D range images. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*, 2000.
- [11] A. Cayley. On contour and slope lines. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, XVIII:264–268, 1859.
- [12] F. Cheevasuvit, H. Maitre, and D. Vidal-Madjar. A robust method for picture segmentation based on split-and-merge procedure. *Computer Vision, Graphics and Image Processing*, 34(3):268–281, June 1986.
- [13] S. H. Collins. Terrain Parameters directly from a digital terrain model. *The Canadian Surveyor*, 29(5):507–518, December 1975.

- [14] Digital Elevation Models. <http://grid2.cr.usgs.gov/dem/what.html>.
- [15] B. P. Dobrin, T. Viero, and M. Gabbouj. Fast watershed algorithms: analysis and extensions. In *Nonlinear Image Processing, Proc. SPIE*, volume 2180, pages 209–220, 1994.
- [16] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, Inc., 1973.
- [17] N. Eua-Anant and L. Upda. Boundary extraction algorithm based on particle motion in a vector image field. In *Proceedings of the International Conference on Image Processing (ICIP)*, volume II, 1997.
- [18] K. S. Fu and J. K. Mui. A survey on image segmentation. *Pattern Recognition*, 13:3–16, 1981.
- [19] J. M. Gauch. Image segmentation and analysis via multiscale gradient watershed hierarchies. *IEEE Trans. on Image Processing*, 8(1), 1999.
- [20] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison Wesley Longman, Inc., 1999.
- [21] R.T. Whitaker J. Gregor and P.F. Chen. Indoor scene reconstruction from sets of noisy range images. In *Proc. of the Second International Conference on 3-D Imaging and Modeling (3DIM)*, pages 348–357, 1999.

- [22] D. Hagyard, M. Razaz, and P. Atkin. Analysis of watershed algorithms for greyscale images. In *Proceedings of the International Conference in Image Processing (ICIP)*, pages 41–44, 1996.
- [23] R. M. Haralick and L. G. Shapiro. Survey - Image segmentation techniques. *Computer Vision, Graphics and Image Processing*, 29:100–132, 1985.
- [24] K. H. Haris, S. N. Efstratiadis, N. Maglaveras, and A. K. Katsaggelos. Hybrid image segmentation using watersheds and fast region merging. *IEEE Trans. on Image Processing*, 7(12), December 1998.
- [25] M. Heath, S. Sarkar, T. Sanocki, and K. Bowyer. Comparison of edge detectors: A methodology and initial study. In *Proceedings of conference on Computer Vision and Pattern Recognition (CVPR)*, 1996.
- [26] B. Heisele and W. Ritter. Segmentation of range and intensity image sequences by clustering. In *Proceedings of the International Conference on Information Intelligence and Systems (ICIIS)*, 1999.
- [27] S. E. Hernandez and K. E. Barner. Joint region merging criteria for watershed-based image segmentation. In *Proceedings of the International Conference in Image Processing (ICIP)*, 2000.
- [28] A. Hoover, G. Jean-Baptiste, X. Jiang, P. J. Flynn, H. Bunke, D. B. Goldgof, K. Bowyer, D. W. Eggert, A. Fitzgibbon, and R. B. Fisher. An experimental comparison of range im-

- age segmentation algorithms. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 18(7), July 1996.
- [29] P. Jackway. Gradient watersheds in morphological scale-space. *IEEE Trans. on Image Processing*, 5(6):913–921, 1996.
- [30] J. Ji. Region-based segmentation - Watershed segmentation. Tutorial, <http://ice.hansung.ac.kr/jun/CV/Segmentation3.html#watershed>.
- [31] T. D. Jones. *Improving the precision of leg ulcer area measurement with active contour models*. PhD thesis, School of Computing, University of Glamorgan, Wales, UK, May 1999.
- [32] Y. Kanai. Image segmentation using intensity and color information. In *Proceedings of the Visual Communications and Image Processing*, pages 709–720, 1998.
- [33] D. Kim. Multi-resolutonal watershed segmentation with user-guided grouping. In *Proc. of SPIE Conference on Image Processing*, volume 3338, pages 1087–1095, 1998.
- [34] H. Y. Lee, W. Park, H. K. Lee, and T. Kim. Towards knowledge-based extraction of roads from 1m-resolution satellite images. In *Proceedings of the 4th IEEE Southwest Symposium on Image Analysis and Interpretation*, 2000.
- [35] C. Lemaréchal, R. Fjørtoft, P. Marthon, E. Cubero-Castan, and A. Lopès. SAR image segmentation by morphological methods. In *EUROPTO Conference on SAR Image Analysis, Modeling, and Techniques, Proc. SPIE*, volume 3497, pages 111–121, 1998.

- [36] E. Lester. Feature extraction, image segmentation and surface fitting: the development of a 3D scene reconstruction system. Master's thesis, Department of Electrical Engineering, The University of Knoxville, Tennessee, Knoxville, TN, 1998.
- [37] W. Li, B. Bénié, D. C. He, S. Wang, D. Ziou, and H. J. Gwyn. Watershed-based hierarchical SAR image segmentation. *International Journal of Remote Sensing*, 20(17):3377–3390, 1999.
- [38] L. Lucchese and S. K. Mitra. Unsupervised segmentation of color images based on K-means clustering in the chromaticity plane. In *Proceedings of the IEEE Workshop on Content-Based Access of Image and Video Libraries*, June 1999.
- [39] A. P. Mangan and R. T. Whitaker. Partitioning 3D surface meshes using watershed segmentation. *IEEE Trans. on Visualization and Computer Graphics*, 5(4), October/December 1999.
- [40] J. C. Maxwell. On contour-lines and measurement of heights. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, XL:421–427, 1870.
- [41] A. Moga and M. Gabbouj. A parallel marker based watershed transformation. In *Proceedings of the International Conference in Image Processing (ICIP)*, volume II, pages 137–140, 1996.
- [42] A. Moga, T. Viero, B. P. Dobrin, and M. Gabbouj. Implementation of a distributed watershed algorithm. *Mathematical Morphology and Its Applications to Image Processing*, pages 281–288, 1994.

- [43] A. N. Moga, B. Cramariuc, and M. Gabbouj. An efficient watershed segmentation algorithm suitable for parallel implementation. In *Proceedings of the International Conference in Image Processing (ICIP)*, volume II, pages 101–104, 1995.
- [44] A. N. Moga, B. Cramariuc, and M. Gabbouj. A parallel watershed algorithm based on rainfall simulation. In *Proceedings of the European Conference on Circuit Theory and Design (ECCTD)*, volume I, pages 339–342, 1995.
- [45] A. N. Moga and M. Gabbouj. Parallel image component labeling with watershed transformation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(5):441–450, May 1997.
- [46] L. Najman and M. Schmitt. Geodesic saliency of watershed contours and hierarchical segmentation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 18(12):1163–1173, December 1996.
- [47] National Institute of Health National Library of Medicine. Visible Human Project.  
[http://www.nlm.nih.gov/research/visible/getting\\_data.html](http://www.nlm.nih.gov/research/visible/getting_data.html).
- [48] N. Nikolaidis and I. Pitas. *3D image processing algorithms*. John Wiley & Sons, Inc., 2001.
- [49] B. Ogor, V. Haese-Coat, and K. Kpalma. A cooperation of mathematical morphology and region growing for remote sensing image segmentation. In *Image and signal processing for remote sensing, Proc. SPIE*, volume 2579, pages 375–386, 1995.

- [50] N. R. Pal and S. K. Pal. A review on image segmentation techniques. *Pattern Recognition*, 26(9):1277–1294, 1993.
- [51] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 12(7):629–639, 1990.
- [52] T. K. Peucker and D. H. Douglas. Detection of surface-specific points by local parallel processing of discrete terrain elevation data. *Computer Graphics and Image Processing*, (4):375–387, 1975.
- [53] I. E. Pratikakis, H. Sahli, and J. Cornelis. Hierarchy determination of the gradient watershed adjacent groups. In *Proc.of the 10th Scandinavian Conference on Image Analysis*, pages 685–692, 1997.
- [54] W. K. Pratt. *Digital Image Processing*. John Wiley & Sons, Inc., second edition, 1991.
- [55] M. E. Rettmann, X. Han, and J. L. Prince. Watersheds on the cortical surface for automated sulcal segmentation. In *Proceedings of the IEEE Workshop on Mathematical Methods in Biomedical Image Analysis(MMBIA'00)*, 2000.
- [56] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, Inc, 1982.
- [57] A. Shiji and N. Hamada. Color image segmentation method using watershed algorithm and contour information. In *Proceedings of the International Conference in Image Processing (ICIP)*, 1999.

- [58] P. D. Smet. Activity driven non-linear diffusion for color image watershed segmentation. *Journal of Electronic Imaging*, 8(3):270–278, July 1999.
- [59] P. D. Smet and R. Pires. Implementation and analysis of an optimized rainfallsing watershed algorithm. In B. Vasudev, T. R. Hsing, A. G. Tescher, and R. L. Stevenson, editors, *Image and Video Communications and Processing, Proc. SPIE*, volume 3974, pages 759–766, 2000.
- [60] P. Soille and L. Vincent. Determining watersheds in digital pictures via flooding simulations. In *Visual Communications and Image Processing, Proc. SPIE*, volume 1360, 1990.
- [61] Y. Solihin and C. G. Leedham. Integral ratio: A new class of global thresholding techniques for handwriting images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 21(8):761–767, August 1999.
- [62] M. Sonka. Image understanding: semantic image segmentation and understanding. Tutorial, <http://www.icaen.uiowa.edu/dip/LECTURE/Understanding6.html>.
- [63] B. Spitzak. Fast Light Tool Kit (FLTK). <http://www.fltk.org/>.
- [64] C. Steger. Subpixel-precise extraction of watersheds. In *Proceedings of the 7th International Conference on Computer Vision (ICCV)*, volume II, pages 884–890, 1999.
- [65] Bart M. ter Haar Romeny. Multiscale geometric image analysis: scale-space theory. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 1999. Tutorial.

- [66] L. Vincent and P. Soille. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 13(6), June 1991.
- [67] A. S. Wright and S. Acton. Watershed pyramids for edge detection. In *Proceedings of the International Conference in Image Processing (ICIP)*, volume II, 1997.
- [68] Y. Yang and H. Yan. A robust document processing system combining image segmentation with content-based document compression. In *Proceedings of the International Conference on Pattern Recognition(ICPR)*, 2000.
- [69] B. Yu and A. Jain. Lane boundary detection using a multiresolution Hough transform. In *Proceedings of the International Conference on Image Processing (ICIP)*, volume II, 1997.
- [70] S. D. Zenzo. A note on the gradient of a multi-image. *Computer Vision, Graphics, and Image Processing*, 33:116–125, 1986.

## **Vita**

Annapoorani Gothandaraman was born in the city of Coimbatore, located in Tamil Nadu, India, in 1978. She did her initial schooling in Sri Rukmani Kannan Vidyalaya, in a small urban town called Kovaipudur. When her family moved to Coimbatore, she continued her education in Sri Nehru Vidyalaya and completed higher secondary education in May 1995. She continued her studies in Anna University, Madras and completed the Bachelors program in Electronics and Communication Engineering in May 1999. She then enrolled in the Masters program in the Department of Electrical Engineering, University of Tennessee, Knoxville in August 1999. She is currently working as a Graduate Research Assistant in The Vision Group laboratory under the supervision of Dr. R. T. Whitaker. She will complete her graduate study in December 2001 with a specialization in image processing.