AUTOMATED COUNTING OF CELL BODIES

USING NISSL STAINED CROSS-SECTIONAL IMAGES

A Thesis

by

ASWIN CLETUS D'SOUZA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2007

Major Subject: Computer Science

AUTOMATED COUNTING OF CELL BODIES

USING NISSL STAINED CROSS-SECTIONAL IMAGES

A Thesis

by

ASWIN CLETUS D'SOUZA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	John Keyser
Committee Members,	Yoonsuck Choe
	Vinod Srinivasan
Head of Department,	Valerie E. Taylor

December 2007

Major Subject: Computer Science

ABSTRACT

Automated Counting of Cell Bodies Using Nissl Stained Cross-Sectional Images.
(December 2007)
Aswin Cletus D'Souza, B.E, Manipal Institute of Technology, India
Chair of Advisory Committee: Dr. John Keyser

Cell count is an important metric in neurological research. The loss in numbers of certain cells like neurons has been found to accompany not only the deterioration of important brain functions but disorders like clinical depression as well. Since the manual counting of cell numbers is a near impossible task considering the sizes and numbers involved, an automated approach is the obvious alternative to arrive at the cell count. In this thesis, a software application is described that automatically segments, counts, and helps visualize the various cell bodies present in a sample mouse brain, by analyzing the images produced by the Knife-Edge Scanning Microscope (KESM) at the Brain Networks Laboratory.

The process is described essentially in five stages: Image acquisition, Pre-Processing, Processing, Analysis and Refinement, and finally Visualization. Nissl staining is a staining mechanism that is used on the mouse brain sample to highlight the cell bodies of our interest present in the brain, namely neurons, granule cells and interneurons. This stained brain sample is embedded in solid plastic and imaged by the KESM, one section at a time. The volume that is digitized by this process is the data that is used for the purpose of segmentation.

While most sections of the mouse brain tend to be comprised of sparsely populated neurons and red blood cells, certain sections near the *cerebellum* exhibit a very high density and population of smaller granule cells, which are hard to segment using simpler image segmentation techniques. The problem of the sparsely populated regions is tackled using a combination of *connected component labeling* and *template*

matching, while the *watershed algorithm* is applied to the regions of very high density. Finally, the *marching cubes algorithm* is used to convert the volumetric data to a 3D polygonal representation.

Barring a few initializations, the process goes ahead with minimal manual intervention. A graphical user interface is provided to the user to view the processed data in 2D or 3D. The interface offers the freedom of rotating and zooming in/out of the 3D model, as well as viewing only cells the user is interested in analyzing. The segmentation results achieved by our automated process are compared with those obtained by manual segmentation by an independent expert.

To my family

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. John Keyser, for all the guidance and encouragement he provided me since the very beginning of my graduate program. His invaluable suggestions and expertise in the field have helped me immensely in attaining the goals I set out to achieve with my research. I am also very grateful to my committee members, Dr. Yoonsuck Choe and Dr. Vinod Srinivasan for being so supportive and giving me important feedback over the course of this thesis.

I am deeply indebted to Pei-San and Dr. Louise Abott for taking time off their busy schedules to help with my research as independent experts. Their inputs on the identification of various cells in the data set have helped in shaping a lot of my research work. I would also like to thank the members of the Brain Networks Laboratory and the Geometry and Graphics Group at TAMU, especially David Mayerich, for the many helpful suggestions I have been provided.

Finally, I would like to thank Raksh, Erwin, my parents, and my friends for all the smiles along the way that made the many months of my research a lot more pleasant and enjoyable.

This research was supported in part by the National Institutes of Health/National Institute of Neurological Disorders and Stroke grant #1R01-NS54252 (PI: Yoonsuck Choe).

NOMENCLATURE

KESM	Knife-Edge Scanning Microscope
BNL	Brain Networks Laboratory

TABLE OF CONTENTS

Page

ABSTRACT iii

DEDICATION v

ACKNOWLEDGEMENTS vi

NOMENCLATURE vii

TABLE OF CONTENTS viii

LIST OF FIGURES x

LIST OF TABLES xiv

CHAPTER

I INTRODUCTION 1

1.1 Objectives 2
1.2 Significance 2
1.3 Outline of the thesis 2

II BACKGROUND AND OVERVIEW OF STAGES 4

2.1 Basic concepts in neuroscience 4
2.2 Staining methods 5
2.3 Literature review 6
2.4 Overview of stages 10

III IMAGE ACQUISITION 12

3.1 Features of the KESM 12
3.2 Sub-systems of the KESM 13
3.3 Working principle 13
3.4 Types of images attained 14

IV PRE-PROCESSING 17

CHAPTER Page

4.1 Image normalization 17
4.2 Median filtering 17
4.3 Thresholding 19
4.4 Gradient transform 19
4.5 Distance transform 20
4.6 Gradient weighted distance transform 22

V PROCESSING 23

5.1 Choosing the algorithm 23
5.2 Algorithms for segmenting images of type A 23
5.3 Watershed algorithm for segmenting images of type B 30

VI ANALYSIS AND REFINEMENT 37

6.1 Post-Processing in images of type A 37
6.2 Post-Processing for images of type B 46

VII VISUALIZATION 50

7.1 Marching cubes 50
7.2 Graphical user interface 54

VIII RESULTS 63

8.1 Details of experiment 63
8.2 Labeling corrections 65
8.3 Type A results 66
8.4 Type B results 70

IX CONCLUSION AND FUTURE WORK 72

9.1 Conclusion 72
9.2 Future work 72

REFERENCES 74

APPENDIX 78

VITA 85

LIST OF FIGURES

FIGURE		Page
1	Neuron anatomy	4
2	Section of a Nissl-stained image	5
3	Pepper seeds on a white surface	7
4	Image after thresholding	8
5	The stages of the process	11
6	Photograph of the KESM	12
7	(a) 3D rendering of the knife edge at work (b) Labeled diagram of the internal working of the knife edge	13
8	The two kinds of image samples (a) Type A (b) Type B	14
9	Type A cells, with green outlines for possible RBC/dying cells, and red outlines highlighting the neurons	15
10	The various kinds of cells found in images of type B	16
11	(a) Image before median filtering (b) Image after median filtering	18
12	Sobel operators for (a) X-direction (b) Y-direction and (c) Z-direction	20
13	Sobel gradient transform on Nissl image (a) Before (b) After	20
14	Chamfer 2-D distance transform on sample image (a) Before (b) After	21
15	Sample connected component and the contents of the patch *P* associated with it	25
16	Illustration for step 1 and 2 of algorithm	26
17	Image and corresponding patch list during the scan, at (a) row = 0 (b) row = 3 (c) row = 13 (d) row = 18	27

FIGURE Page

18 (a) Image of a section of Nissl tissue (b) Thresholding applied (c) Eye-shaped cells highlighted in red (d) Same cell highlighted in the threshold image 28

19 (a) Kernel used for template matching (b) Original image (c) Results of template matching 29

20 Gray scale image and its digital elevation model 30

21 (a) Original image (b) Gradient image (c) Topographical representation of the gradient image 30

22 (a) Geodesic distance between two points in a sample region R (b) Geodesic influence zones 31

23 (a) Topography at immersion level $h_{\min}$ (b) X at $h_{\min}$ (c) Topography at immersion level $h_{min} + 1$ (d) X at $h_{min} + 1$ consists of two components, one that is growing from an earlier component, and the other as a new component 33

24 Sorting data structure 34

25 Watershed algorithm in progress (a) $h = 0$ (b) $h = 80$ (c) $h = 130$ 35

26 2D image of image 3 in the stack, at $h = 130$ 36

27 Illustration of hole-filling algorithm 38

28 (a) Image with visible chatter artifacts (b) Sample section of an image after image corrections (c) Noisy regions 39

29 (a) Section with a group of close cells (b) Threshold results 40

30 A mistakenly grouped cell is split using information from the preceding image. 41

31 (a) Original image (b) Thresholded image (c) 3D template matching results (d) Local maxima of the matched results 42

32 (a) Local peaks (b) Splitting of a group of cells using boolean operations 43

FIGURE Page

33 Neurons with weak features as seen in (a) are marked using standard sized patches in (b) 43

34 Cell and its respective cross-sections 44

35 Neighborhood connectivity for pixel that is (a) 4-connected (b) 8-connected (c) 26-connected 46

36 (a) C_1 and C_2 before merging (b) After (c) C_3 and C_4 before merging (c) After 47

37 (a) Watershed segmentation (b) Sparse / dense image (c) Final segmentation 49

38 (a) Initial 2D matrix (b) Representing each point by a sprite (c) Some standard 2D cases (d) Representing the set by line segments (d) Object on a finer grid 51

39 Fifteen standard cases used in 3D marching cubes [42] 51

40 Representation of a stack of 10 *type A* images, where each cell has a random color 52

41 Alternate orientation of dataset using in fig.40 53

42 Visualization of image stack of 10 *type B* images 53

43 Zoomed-in view of data set used in fig.42 54

44 Interface shown with *type A* images in (a) 2D and (b) 3D 55

45 *Type A* thresholded image at threshold value t = 140 56

46 *Type A* thresholded image at threshold value t = 133 57

47 *Type A* image shown with (a) preliminary classification: compact, noisy, corrected, prospective noise and (b) final classification: neuron/glial, rbc/dying, unidentified and noise 58

48 *Type A* images in 3D after processing with a random color assigned to each cell 59

FIGURE Page

49 *Type A* images in 3D after classification 60

50 *Type B* image in 2D during the running of the watershed algorithm 60

51 *Type B* cells classified uniquely using (a) random coloration (b) ..... coloration based on cell type: granule cells or interneurons 61

52 The 'representation' view of the labeling by the watershed algorithm..... 62

53 (a), (b), (c) Type A images provided for segmentation by independent experts (d), (e), (f) Marked images where the cell types are labeled by the colors red, blue and yellow 64

54 (a), (b), (c) Type B images provided for segmentation (d), (e), (f) Marked images where the cell types are labeled by the colors red, blue, green and yellow 64

55 (a), (b), (c) Original labeling (d), (e), (f) After 3D labeling corrections 65

56 (a), (d), (g) Cells identified by expert (b), (e), (h) All cell bodies identified by algorithm (c), (f), (i) Neurons/Glial cells identified by algorithm 66

57 (a) Original image (b) Segmentation: first pass (c) Corrections by expert during second pass 68

58 (a), (e), (i) Original images (b), (f), (j) Cells identified by expert (c), (g), (k) Segmentation results depicting cell classification (d), (h), (l) Segmentation results depicting individual cell areas 69

LIST OF TABLES

TABLE Page

1 Comparison of manual and automated results for segmentation of cell bodies 67

2 Comparison of manual and automated results for segmentation of neurons 67

3 Comparison of manual and automated results for segmentation of granular cell bodies 70

4 Comparison of manual and automated results for segmentation of interneurons 71

CHAPTER I

INTRODUCTION

The human brain has long been an area of immense fascination for researchers who believe that fully understanding its structure and functions would lead to great strides in the field of science, from biology to artificial intelligence. In this pursuit to completely understand the brain, scientists look to solve smaller and simpler problems before attempting to solve more complex ones and this is the reason the mouse brain has gained much attention from research groups from all across the scientific world.

From a medical point of view, the study of the mouse brain could help in identifying and arriving at possible treatments for neurological disorders like Parkinson's disease and clinical depression. An important metric in this study has been the count of certain cell bodies in the brain, like neurons, since the loss of neurons has been found to accompany the deterioration of many brain functions. Attaining accurate counts of these cell bodies is however not an easy task due to the microscopic size of these cells, and the vast numbers in which they are present.

Certain methods of estimation have been used over the years in laboratories, like the *hemacytometer* and the *counting chamber*, but these methods are heavily dependent on sample concentrations and the estimation formula used [1][2]. Fortunately, due to much technological advancement in the fields of microscopy and digital imaging, more accurate counting mechanisms are now possible. Detailed cross-sectional images of mouse brain tissue can be extracted using devices like the Knife-Edge Scanning Microscope (KESM), which can then be processed automatically by means of various algorithms, and visualized in 3-D.

This thesis will describe a software application that counts cells in Nissl-stained image data obtained from the KESM, and enables the user to visualize these cells in 3-D using a graphical user interface.

This thesis follows the style and format of *IEEE Transactions on Visualization and Computer Graphics*.

1.1 Objectives

The primary goal of this thesis is to describe a software alternative to neurobiologists for purposes of obtaining cell counts and visualizing the cells of interest from image data. The application described has the following features:

i. An ability to load a number of consecutive images in the stack
ii. An automated process that segments the cells of interest
iii. A provision for the user to set certain cell characteristics that might help in segmentation
iv. Ability to traverse through the image set one image at a time
v. Options to show the boundaries of the cells found, in 2-D
vi. A 3-D visualization showing the cell bodies identified
vii. Graphical user interface

1.2 Significance

Neuron loss has been shown to have a significant effect on functions like memory acquisition and illnesses like clinical depression [3] [4]. This correlation is also evident in patients suffering from various neurological disorders like Alzheimer's and Parkinson's disease. Scientists studying these disorders in mice can gain significant savings in time with the help of an automated process that supplies them readily with the cell count of the neurons in the tissues they are studying. Such a process can be used specifically by members of the Brain Networks Laboratory to study brain tissue.

1.3 Outline of the thesis

The problem of automated cell counting has been addressed before and many techniques have been put forth to solve it. In Chapter II, we provide the reader with background information needed to understand the context and the conclusions of this thesis.

The working of the Knife-Edge Scanning Microscope is briefly described in Chapter III, Image acquisition. The two prominent types of images that we shall term *type A* and *type B* are introduced here, and their differences explained. Since the cells found in these images vary so vastly in size, shape and density, the algorithms used to

tackle them vary as well. We use a combination of *connected component labeling* and *template matching* for *type A* segmentation, and a variation of the *watershed algorithm* for segmentation of *type B*.

Pre-processing of the images must take place before the application of either of the algorithms mentioned above, in order to aid in better segmentation. We introduce these techniques in Chapter IV, Pre-processing, while Chapter V, Processing, provides an in-depth description of the two algorithms for the respective types of images.

Post-processing is an important stage, as both these algorithms do not result in perfect segmentation immediately once they are run. Splitting of groups of cells into their individual constituents, merging of components that are part of the same cell, elimination of noise, and so on are important procedures that must be done before the final stage. These are discussed in detail in Chapter VI, Analysis and Refinement.

Once the cells have been segmented, a visualization of them is helpful to the neuroscientist, who can then view the dataset from any angle/zoom and study the structure and arrangement of these cell bodies. Marching cubes is used to generate the 3-D volumetric representation of this data set. We describe this algorithm as well as the other interface features in Chapter VII, Visualization.

To find out how well the results of the software application compare with those of manual segmentation, we describe the method used to validate the results in Chapter VIII. The help of an independent expert is taken for comparison.

Finally, the conclusions of the research work, and a list of possible improvements are provided in Chapter IX.

CHAPTER II

BACKGROUND AND OVERVIEW OF STAGES

In this chapter, we provide the reader with a brief description of the neuron, the staining mechanism that was used to obtain the samples, and a review of image segmentation literature.

2.1 Basic concepts in neuroscience

2.1.1 Neuron theory

The 'neuron theory' states that nerve tissue is composed of individual cells which are fully functional units. Although the morphology of various types of neurons differs in some respects, they all contain four distinct regions with differing functions: the *cell body*, the *dendrites*, the *axon*, and the *axon terminals* [5] [6] [7]. The structure of a standard neuron can be seen below in fig. 1.

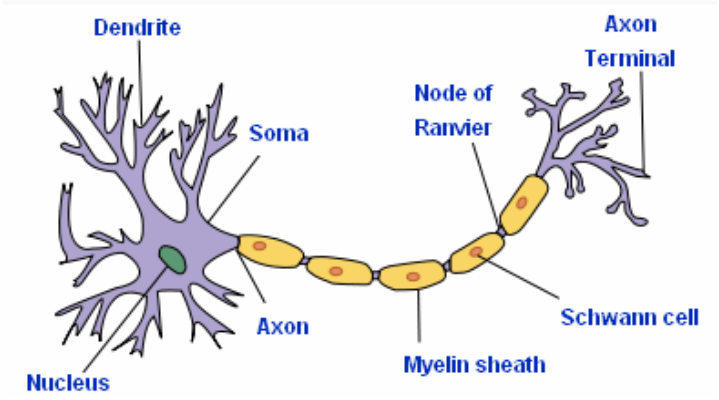


Fig. 1. Neuron anatomy.
[http://en.wikipedia.org/wiki/Image:Neuron-no_labels.png]

2.1.2 Nissl cell bodies

Named after Franz Nissl who developed a staining mechanism to identify neuronal cell bodies throughout the brain, Nissl cell bodies were the terms used to describe dense granular masses often found in nerve cells [8].

2.2 Staining methods

Staining is basically the process of making a compound of interest (DNA, proteins) more easily identifiable by enhancing its visibility through certain chemical means. The online medical dictionary defines staining as 'the use of a dye, reagent, or other material for producing coloration in tissues or microorganisms for microscopic examination' [9]. For the study of neurons, the following staining methods have been widely used:

2.1.1 Golgi staining

The Golgi technique stains neurons by using silver chromate. The detailed structure of the neuron (axons, dendrites) is made visible via this stain and has been very useful in the study of these cells. The Golgi method is essentially a stochastic technique and its exact chemical mechanism remains unclear [10].

2.1.2 Nissl staining

Nissl staining stains the Nissl bodies in cells, technically called the *endoplasmic reticulum*. The staining procedure consists of sequentially dipping the mounted brain slices in about a dozen different solutions for specified amounts of time [11]. We see a section of a Nissl-stained image below in fig. 2.

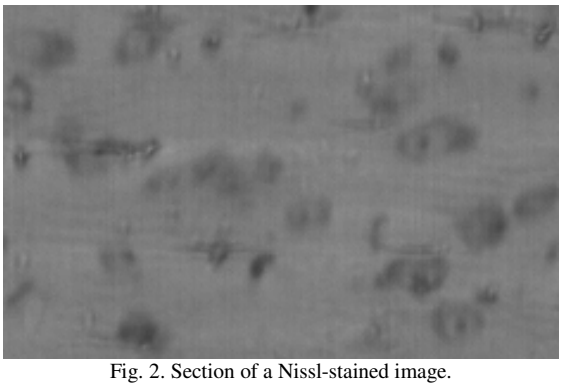


Fig. 2. Section of a Nissl-stained image.

2.3 Literature review

The application of computer image processing to cell counting and recognition has drawn much attention from image processing and cell biology communities [13]. Early work was based on simple thresholding techniques which assumed that the cells were significantly darker/lighter than background pixels [14] [15]. In the case of colored images, different colors were used to segment different objects [16]. This segmentation method has been applied to the detection of cancer cell nuclei as well [17]. Active contour detection has at times also been used in cell identification [13] [18]. In general however, most work in this area of cell segmentation could be broadly classified into two techniques: *interactive* and *automated*.

2.3.1 Interactive cell segmentation

Here, the human visual system and expert judgment is employed to manually segment and classify cell regions from the images, using a computer graphics device such as a tablet or a mouse. In some techniques, the expert goes through several 2-D images, marking the boundaries of the cell/nucleus of interest as he/she goes along [19]. The algorithm finally generates a surface that uses these boundaries to visualize the cell.

Reasoning that this process above was too tedious for the expert, work has been done in the development of several tools like *hole-filling*, *point-bridging* and *surface-dragging* that work in 3-D instead of 2-D, thus claiming to save valuable time during the segmentation process [20]. Many of these techniques employ some basic automated segmentation techniques to get started, before the manual process begins.

2.3.2 Automated segmentation

There has been a lot of work done in the field of automated segmentation, as most of the time, medical image stacks run to hundreds of images, with cell sizes being extremely small. Expecting a user to manually demarcate the cell/nucleus boundaries is not only time consuming, but near to impossible considering the tens of thousands of cells that might be present in the volume. Automated techniques will consequently be much faster, and will not burden the examiner with any of the segmentation work, other

than the initialization of parameters. Prior work in the automated effort can be broadly classified as:

i. Segmentation by threshold values
ii. Pattern recognition
iii. Gradient flow techniques

i. Segmentation by threshold values

In image analysis, there needs to be a key criterion to differentiate between objects of interest and other objects, or in other words, 'foreground pixels' and 'background pixels'.

Fig. 3. Pepper seeds on a white surface.

For instance, in fig. 3, it is easy to identify the foreground pixels as shades of brown and background pixels as white. *Thresholding* is an image processing technique for converting a grayscale or color image into a binary image based upon a defined value. If a pixel in the image has an intensity value less than the threshold value, the corresponding pixel in the resultant image is set to black. Otherwise, the resulting pixel is set to white. Image thresholding is very useful for keeping the significant part of an image and getting rid of the unimportant part or noise. This holds true under the assumption that a reasonable threshold value is chosen [21].

Formally, if intensity thresholding is applied to the pixels of the pre-processed image I, with the threshold value t, the resulting image I_r will be:

$$I_r = \begin{cases} 1 & I \geq t \\ 0 & else \end{cases}$$

One of the key problems of thresholding is that it is very sensitive to imaging variations in light and noise, and thus must be used only in situations where the foreground/background pixels are clearly identifiable [13]. Also, as can be seen in fig. 4, objects of interest that touch each other are hard to separate. Thresholding when all cells are clearly separable is a simple and straightforward way to proceed, but when closely packed cells appear in images, like in some of the cases in our Nissl stained samples, this technique is of little help as a solution.

We use this technique in the pre-processing stage for images of cells that are sparsely populated.

Fig. 4. Image after thresholding.

ii. Pattern recognition

Pattern recognition is defined as 'the study of how machines can observe the environment, learn to distinguish patterns of interest from their background, and make sound and reasonable decisions about the categories of the patterns' [22]. 'Machine Vision' is more closely related to our cell counting problem, as it is the area of study where images captured by an imaging system are analyzed using pattern recognition techniques to identify and classify objects of interest [23]. Some of the techniques are briefly described below:

- *Template matching* has long been used as a technique to help match objects of interest with the sample under supervision. Here, a template or a prototype, usually a shape that is a known representation of the object of interest, is searched for in the sample dataset and checked for how similar the two are, taking into consideration the various orientations the object could be in. A certain 'score' or 'measure' of similarity is calculated, and a threshold value of this score is used to decide the absence or presence of the object in the sample under inspection.

- *Statistical classification* basically represents each pattern in terms of a set number of features, and calculates how closely the sample matches these features. The pattern could be viewed then as a point in *d*-dimensional space, where *d* is the number of features that have been enumerated [23]. Lin et al. use a feature set consisting of *volume*, *texture*, *convexity*, *shape*, *circularity*, *area*, *mean radius* and *eccentricity* [24]. The probability distributions of the patterns belonging to each feature class must either be specified explicitly during design or learned by means of a training set.

- *Syntactic approach* uses the combination of simpler features to describe complex features. The complexities of implementing grammar and noisy patterns result in large computational needs, due to the 'curse of dimensionality' [25].

- *Neural networks* use networks and combinations of simple processors to attack the problem of feature extraction. It is closely related to statistical pattern recognition, though its advantages as a result of parallel processing are noteworthy.

iii. Gradient flow techniques

The *watershed algorithm* is an algorithm that has been widely studied and used over the last couple of decades for the purposes of image segmentation [24] [26] [27] [28] [29] [30].

The biggest advantage of this algorithm is in attacking the problem of 'connected cells', which remained unsolved when thresholding techniques were used. It places great value on the gradient of the cells, which tend to go from dark to light at the borders (or

vice-versa if cells are stained white). If these areas where the gradient changes sharply are spotted and all the areas in between are connected to each other, a segmentation of the image is formed that is successful in distinguishing individual cells in a clustered section.

The key principle behind this algorithm is to consider that the gray levels of the pixels of the image are not just color values, but 'height values', where the greater the gray-scale value, the taller the peak and vice-versa. Thus, the image represents not mere pixels, but a topographical view of crests and troughs all across the image. At this point, we start filling the scene with water, up to a height *h*, incrementally from the lowest to the highest value.

The basins that are formed correspond to the connected components in the image, and the points where different basins meet are called the *watershed* points, much like dams. This algorithm tends to over-segment the image, and post-processing techniques are performed to ensure a good final segmentation.

Gradient flow tracking is a novel approach to segmentation. Instead of growing the catchment basins from the minima outwards, like the watershed algorithm, this algorithm points each voxel in the direction of its gradient flow. The gradient vector field is then diffused with an elastic deformable transformation, which smoothes the gradient field by propagating gradient vectors of large magnitude to vectors with weak gradients. Thus, at the end of the procedure, we have each voxel either pointing to another voxel by means of the gradient flow vector, or being a minimum in itself. All voxels that flow to the same center are then classified together as a cell, and this automatically separates individual cells from clusters [31].

2.4 Overview of stages

In fig. 5 we show a pictorial representation of the stages that will soon be described in the chapters to follow, with each chapter describing each of the stages in the process.

Though there are essentially 5 stages: *image acquisition, pre-processing, processing, analysis and refinement,* and *visualization,* the algorithms applied in each stage to the different types of images vary. The two types, *type A* and *type B* are

introduced in section 3.4. Though the *image acquisition* stage and the *visualization* stage are identical for both types of images, the three stages in between vary vastly, as depicted in fig. 5 by the branching off of the stages in two directions.

This is because, as we will explain in the chapters to follow, the same segmentation methods do not yield the best results for both types of images.

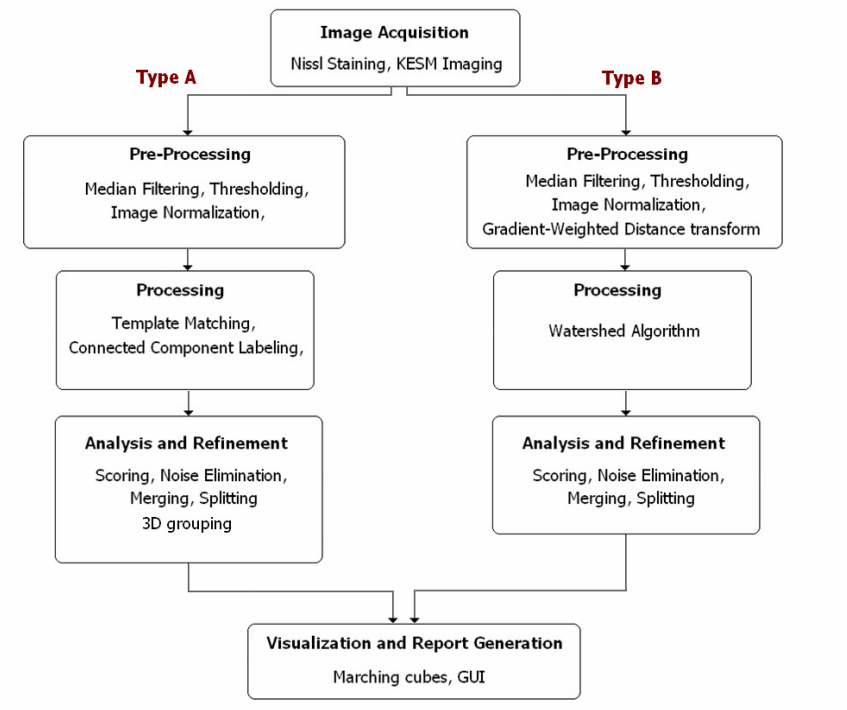


Fig.5. The stages of the process

Type A image segmentation heavily depends on the post-processing stage, since much of the 3D grouping can happen only after the components at each image level are filtered appropriately for noise and mistakenly split components. *Type B* however attains a fairly good quality of segmentation after the processing stage itself.

We start with the first stage, *image acquisition* in the next chapter.

CHAPTER III

IMAGE ACQUISITION

The Knife Edge Scanning Microscope (KESM) is an instrument at the Brain Networks Laboratory of Texas A&M University that performs the simultaneous slicing and imaging of a mouse brain (fig. 6). It was designed by Dr. Bruce McCormick, and has since played a pivotal role as the bridge between the biological world and computer science, converting stained samples of mouse brain into its volume digitized representation that can then be used for various types of study using computer algorithms.

3.1 Features of the KESM

The KESM handles the following operations:

i. Slices layers of mouse brain embedded in solid plastic in a staircase like fashion.

ii. Lights up the imaging sample using light shone through the diamond knife that is used for slicing.

ii. Captures a high resolution, high magnification image of the layer being sliced currently

iii. Stores these images that were read into a digital storage device.

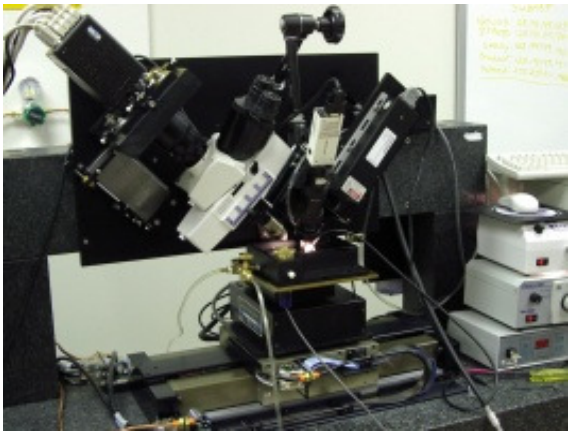
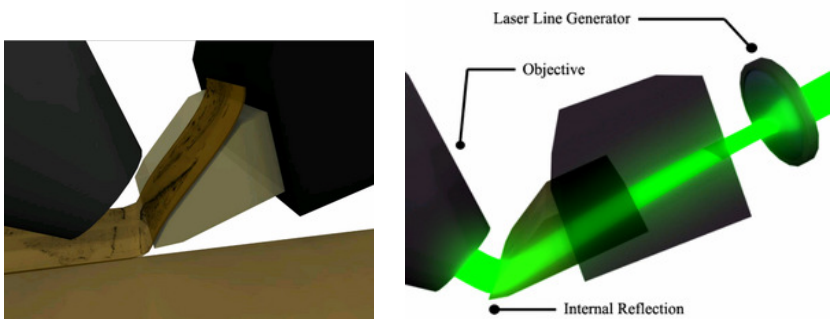


Fig. 6. Photograph of the KESM.
[http://research.cs.tamu.edu/bnl/static/galleryKesm.html]



(a) (b)

Fig. 7. (a) 3D rendering of the knife edge at work (b) Labeled diagram of the internal working of the knife edge. [http://research.cs.tamu.edu/bnl/static/galleryKesm.html]

3.2 Sub-systems of the KESM

This microscope is comprised of the following sub-systems, to enable it to provide for all the features mentioned above in 3.1. The sub-systems as mentioned in [12] are as follows:

i. The *precision positioning system* and *ultra-microtome* axis align the sample in X, Y and Z and keeps track of the position of the sample and the knife edge.

ii. The *image capture system* comprises of the microscope objective coupled with the high-resolution camera

iii. The *Image analysis and Data storage system* contains 5 clustered servers that store the information transferred over the gigabit network.

3.3 Working principle

The brain specimen which is in solidified form after being embedded in plastic is mounted using the precision positioning system. The diamond knife accurately cuts this solid block in a step-wise fashion to minimize resistance. At the same time, light is shone through the diamond knife and into the layer of tissue that is currently being sliced (fig. 7). This illuminates the region and provides better lighting conditions for the imaging system to capture the details as accurately as possible. The diamond knife thus serves dual use, one for the physical sectioning of the tissue and the other as an optical element through which light is shone during sectioning [12].

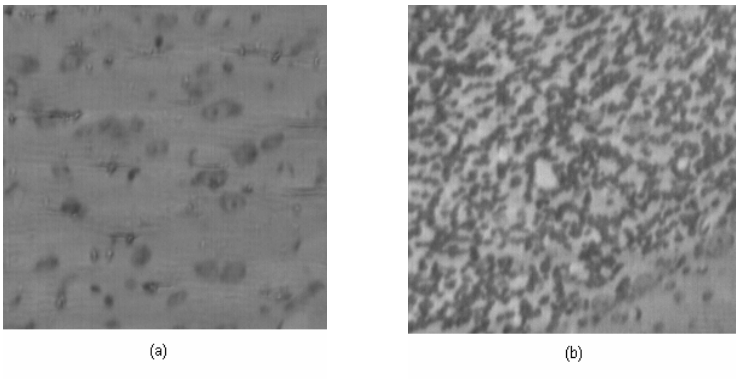


Fig. 8. The two kinds of image samples (a) Type A (b) Type B.

3.4 Types of images attained

We come across two types of cell images in our dataset. The first kind, which we call *type A* for the remainder of this thesis, is comprised of regular sized cell bodies – mainly *neurons*, *glial cells*, *red blood cells* and *dying cells*. They can be seen in fig. 8 (a). For the most part (at least 80-90% of the image space), the Nissl stained images attained comprise of regions of *type A*.

At certain portions of the brain, like the cerebellum, *granule cells* of very small size occur in very high densities in a band like fashion. We call these *type B* images and a sample can be viewed in fig.8 (b). Though the differences are obvious, they are enumerated here for clarity:

- Cells in *type A* are sparsely populated, while those in *type B* are very densely populated
- There are two main types of cells in *type A*: *red blood cells / dying cells* that are dark, small and compact in shape and *neurons / glial cells* that are larger, often hollow or exhibit a central dark spot inside a hollow ellipse, resembling an ‘eye’. There are three main types of cells in *type B*: g*ranule cells* that are tiny and compact, *interneurons* that are sparsely populated, and *purkinje cells that* are big and have a lighter texture than the rest of the cell bodies.

We analyze these two types in greater detail with the help of some magnified images, in the following sections.

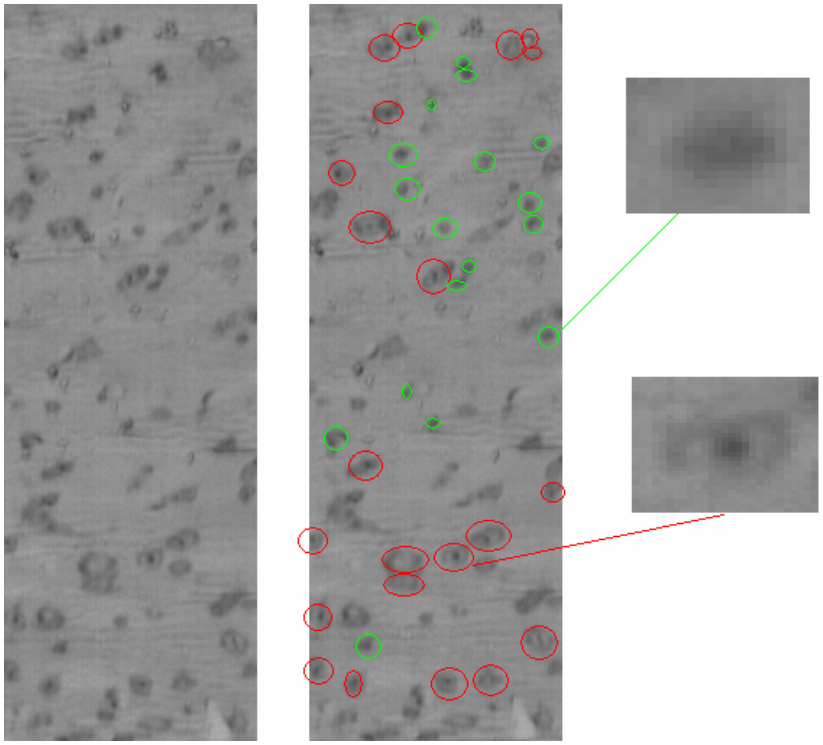


Fig. 9. Type A cells, with green outlines for possible RBC/dying cells, and red outlines highlighting the neurons.

3.4.1 Type A

The descriptions for each cell are provided below, with fig. 9 showing magnified sections of the image for better understanding:

- *Neurons / Glial cells*: are marked with red outlines in the figure above. They are identified by the central dark nucleus and an elliptical outer boundary, which makes them similar to the shape of the human eye.
- *Red blood cells / dying cells*: are darker and more compact than the rest of the cell bodies present. However, it is easy to mistake neurons for these cells because the central nucleus of the neuron is not visible in all cross-sections. Thus, the neighboring sections must be analyzed as well before classifying a cell as an RBC or a dying cell.
- *Endothelial cells:* occur rarely, but are still visible in Nissl images. They are found surrounding blood vessels.

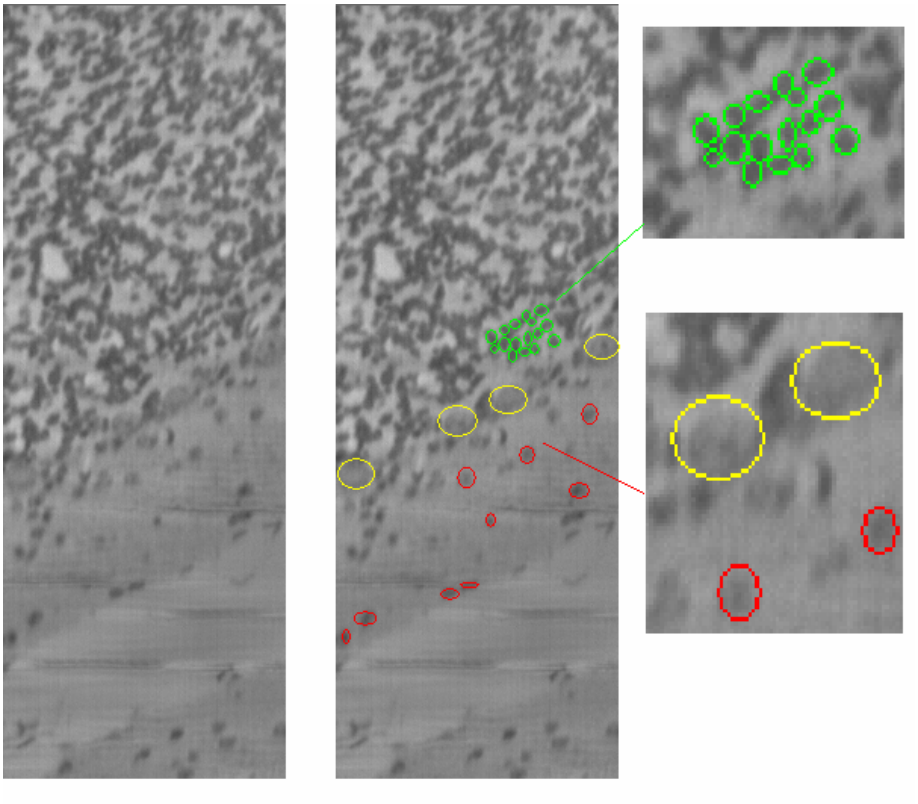


Fig. 10. The various kinds of cells found in images of type B.

3.4.2 Type B

Type B on the other hand consists of the following types, as seen in fig. 10:

- *Granular cells*: are highlighted by light green in the figure. They are densely populated and very tiny.
- *Purkinje cells*: are bigger, and exhibit neuron-like characteristics like those we find in *type A*. They are circled with yellow in fig. 10.
- *Interneurons:* are highlighted by red color. As is evident in the image, they are very sparsely populated in comparison to granular cells.
- *Endothelial cells:* are present in *type B* images again, and are found next to blood vessels.

CHAPTER IV
PRE-PROCESSING

In this chapter, we describe the various *image processing* techniques that were applied to the image stacks before the processing stage.

4.1. Image normalization

Normalization, also known as 'contrast stretching', is an image enhancement technique that 'stretches' the range of intensity values of the image. In our cross-sectional image stacks, due to variations in lighting and positioning on the KESM, the individual images often vary in their brightness and gray-scale range. This non-uniformity in range is problematic in the processing stage, where a high level of importance is placed on the gray-scale value of each pixel. Both, the *watershed* as well as the *thresholding* algorithms are based on the assumption that each image has the same brightness levels as the rest of the images in the stack.

To achieve this range stretching, we should first decide on the minimum and maximum gray scale value that the resulting image should have. Let us call these extremes R_{min} and R_{max}. Usually, they correspond to the range of values that the data element can hold, for instance for an 8-bit integer, $R_{min} = 0$ and $R_{max} = 255$. We then find the minimum and maximum of the values of the pixels in image I under processing. Let us call these P_{min} and $P_{max.}$. In order to prevent rare noisy pixels from distorting this range, the histogram of the image is analyzed to select better values for P_{min} and P_{max}. [32]

Finally, for each pixel P in the image I, the resultant pixel R is calculated as:

$$R = (P - P_{\min})\left(\frac{R_{\max} - R_{\min}}{P_{\max} - P_{\min}}\right) + R_{\min}$$

4.2 Median filtering

Median filtering is used to reduce noise in an image. While a similar technique called 'Mean filtering' is used to accomplish noise reduction as well, it suffers from the

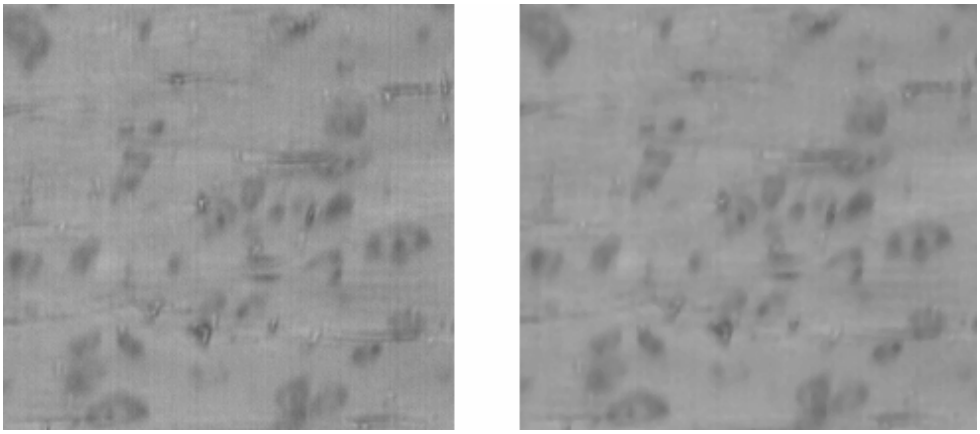


Fig. 11. (a) Image before median filtering (b) Image after Median filtering.

problem of blurring, as each pixel is replaced by the average intensity value of its neighbors. In *median filtering* though, no such average is taken, and the problem of blurring is thus avoided. This technique instead sorts the neighboring pixel values in a list, and the 'median' value, which is the value at the center of the list, is used [33].

To put it formally, if image I has dimensions $S_x \times S_y \times S_z$, $P_{i,j,k}$ is a pixel at location (i,j,k) where $0 \le i \le S_x$, $0 \le j \le S_y$ and $0 \le k \le S_z$, and $N[0..n-1]$ is an array that holds the pixel values of n neighbors of $P_{i,j,k}$ in sorted order, then the resulting median-filtered Image I_r contains at location (i,j,k),

$$R_{i,\,j,\,k} = N\lfloor n/2 \rfloor$$

For instance, in the following 3×3 sample neighborhood, the central value $P_{1,1}$ is a noisy pixel whose value needs to be suppressed.

100	97	104
102	6	104
101	94	99

Sorting the values, we have the neighborhood array N with n=9, as follows:

$$N[0..8] = \{6, 94, 97, 99, 100, 101, 102, 104, 104\}$$

The resultant pixel after median filtering will be

$$R_{1,1} = N\left[\lfloor 9/2 \rfloor\right] = N[4] = 100$$

Sample median filtering results can be seen in fig. 11.

4.3 Thresholding

As explained in section 2.3.2, *thresholding* is the process of classifying the image contents into foreground and background pixel classes, *Fg* and *Bg*. On a gray scale image, this is done by comparing each pixel to a threshold value *t*. Pixels that cross the threshold *t* are labeled *Fg* and the remaining *Bg*. Formally,

$$(x, y) \in Fg \forall I(x, y) \geq t$$
$$(x, y) \in Bg \forall I(x, y) < t$$

The resulting image after thresholding can be expressed as a boolean matrix I_r as follows

$$I_r = \begin{cases} 1 & I \in Fg \\ 0 & else \end{cases}$$

4.4 Gradient transform

The *gradient transform* serves to provide important cues about cell boundaries, which are helpful in the *watershed* algorithm when building 'dams' on the edges of the catchment basins. In the case of the Nissl images we obtained, the gradient from cell pixels (foreground) to background pixels is much higher than the corresponding intra-background or intra-foreground pixels. This important visual cue is exploited in the human cognition system as well, and is similarly applied to machine vision using this transform.

For an image function $f(x, y)$, the gradient magnitude $g(x, y)$ is computed as follows [34]:

$$g(x, y) \cong \sqrt{\Delta x^2 + \Delta y^2}$$
$$\Delta x = f(x+n, y) - f(x-n, y)$$
$$\Delta y = f(x, y+n) - f(x, y-n)$$

where $n = 1$

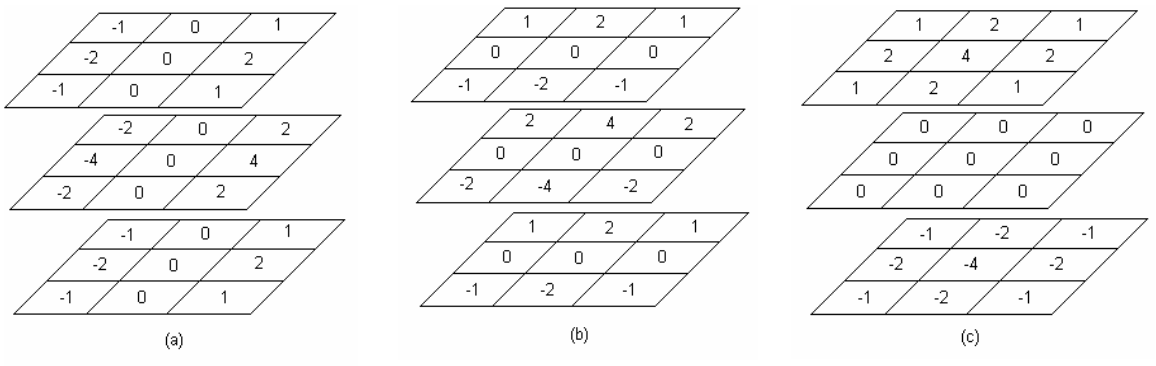


Fig. 12. Sobel Operators for (a) X-direction (b) Y-direction and (c) Z-direction.

The *Sobel* operator is used as an approximation to calculate these equations. Since we have a 3D image, 3-dimensional operators are used for each of the x, y and z directions, as shown in fig. 12.

The *Sobel* operator is less sensitive to isolated noisy variations in pixel values, since the filter calculates a local average over sets of pixels in the immediate neighborhood. Fig. 13 shows the results of a *2-D Sobel* operator on a sample portion of a Nissl image.

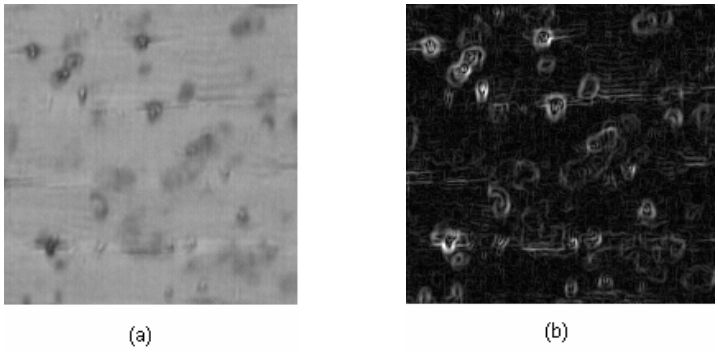


Fig. 13. Sobel Gradient Transform on Nissl image (a) Before (b) After.

4.5 Distance transform

The *distance transform* runs through the image and returns the distance of each point from its nearest boundary. Many segmentation decisions can be made with the

knowledge of how 'deep' a pixel is in a cell, and this aids the *watershed* algorithm while creating catchment basins.

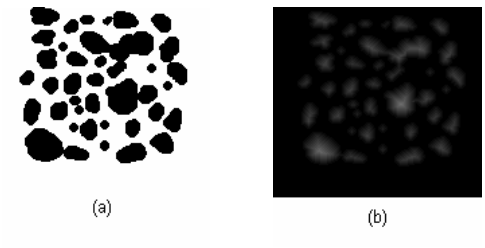


Fig. 14. Chamfer 2-D Distance transform on sample image (a) Before (b) After.

Image thresholding is first applied to the image stacks to separate the Image I into foreground and background pixel classes, Fg and Bg. This binary image is then used to calculate the *distance transformed* image I_d as follows [35],

$$I_d(x,y,z) = \begin{cases} 0 & I(x,y,z) \in \{Bg\} \\ \min\left(\left| x-x_0, y-y_0, z-z_0 \right| , \forall (x_0,y_0,z_0) \in Bg \right) & I(x,y,z) \in \{Fg\} \end{cases}$$

where the metric $| x - x_0, y - y_0, z - z_0 |$ is a distance measurement used to quantify how near or far the point (x, y, z) is from the closest background pixel. Euclidean distance, defined as

$$| x, y, z | = \sqrt{x^2 + y^2 + z^2}$$

is the best metric of distance, but due to its complexity, simpler approximations are generally used in its place. An algorithm known as the *chamfer distance transform* [36] accomplishes this in a 2-pass process to generate the distance transformed image I_d. The algorithm propagates information about boundaries from pixel to pixel, in a pre-defined direction in the first pass. Background pixels are assigned a distance of 0, and this value is incrementally passed on to the pixels that are adjacent to it. In the second pass, the same is done in the reverse direction, and we end up with I_d consisting of an approximation of the distance between each pixel and its nearest boundary. A sample 2-D distance transform can be seen in fig. 14.

If the distance between the central pixel and its neighboring pixels in a 3×3×3 filter are described by the constants d_1, d_2, d_3, d_4 and d_5, then the following is used to propagate boundary information in Pass 1, where the direction is chosen as left to right, top to bottom and back to front:

$$I_d(x,y,z) = \min \begin{pmatrix} I_d(x-1,y-1,z-1)+d_5, I_d(x,y-1,z-1)+d_4, I_d(x+1,y-1,z-1)+d_5 \\ I_d(x-1,y,z-1)+d_4, I_d(x,y,z-1)+d_3, I_d(x+1,y,z-1)+d_4 \\ I_d(x-1,y+1,z-1)+d_5, I_d(x,y+1,z-1)+d_4, I_d(x+1,y+1,z-1)+d_5 \\ I_d(x-1,y-1,z)+d_2, I_d(x,y-1,z)+d_1, I_d(x+1,y-1,z)+d_2 \\ I_d(x-1,y,z)+d_1 \end{pmatrix}$$

In pass 2, the boundary information is propagated from right to left, bottom to top and front to back, as follows:

$$I_d(x,y,z) = \min \begin{pmatrix} I_d(x+1,y,z)+d_1 \\ I_d(x-1,y+1,z)+d_2, I_d(x,y+1,z)+d_1, I_d(x+1,y+1,z)+d_2 \\ I_d(x-1,y-1,z+1)+d_5, I_d(x,y-1,z+1)+d_4, I_d(x+1,y-1,z+1)+d_5 \\ I_d(x-1,y,z+1)+d_4, I_d(x,y,z+1)+d_3, I_d(x+1,y,z+1)+d_4 \\ I_d(x-1,y+1,z+1)+d_5, I_d(x,y+1,z+1)+d_4, I_d(x+1,y+1,z+1)+d_5 \end{pmatrix}$$

4.6 Gradient weighted distance transform

Lin et al. describe a novel approach to combine the results of the *gradient transform* and the *distance transform* described above, into a 'gradient weighted distance transform' [24]. They define this new measure D' as

$$D' = D \times \exp\left(1 - \frac{G - G_{\min}}{G_{\max} - G_{\min}}\right)$$

where G is the gradient transform described in 3.3, D is the distance transform described in 3.4 and D' is the gradient weighted distance transform.

CHAPTER V

PROCESSING

This chapter mainly deals with the techniques used to process the image stack and arrive at preliminary segmentations, which are then fed to the *analysis and refinement* stage of Chapter VI to further improve the results. We first explain the reasons for choosing certain algorithms for certain types of images, and then move towards a description of these algorithms.

5.1. Choosing the algorithm

As mentioned in section 3.4, *type A* and *type B* vary vastly in the shapes, sizes and densities of their cells. Having a unified approach to solve both types would compromise on the quality of segmentation results of both types, and hence they are tackled separately using algorithms whose strengths are best suited for one or the other.

For *type A,* we would need an algorithm that can find hollow, eye-shaped cells, as well as solid cells that are sparsely populated. Thresholding followed by *connected component labeling* is a good approach to this sparsely populated dataset, as they are easily distinguishable from each other.

As for *type B,* we need an algorithm whose strength is to be able to distinguish closely grouped components. This is where the *watershed algorithm* comes into play. We look at each of these types in the sections to follow.

5.2 Algorithms for segmenting images of type A

5.2.1 Connected component labeling

A novel approach to *connected component labeling* is introduced, which differs in many ways from the more classical implementations of the algorithm [37]. The algorithm works on images that have already undergone the *threshold* operation, i.e. binary images that store at every position the values 0 or 1, depending on the absence or presence of a Foreground pixel at that point.

A. Basic definitions

Definition 1: Image-volume *I* is a 3-D array of voxels, that has dimensions $s_x \times s_y \times s_z$, along the x, y and z axis respectively, with s_z equaling the number of Images in the image stack.

Definition 2: Index *i* is a triplet (x, y, z) of integer values that gives the position of a voxel in the image volume, where $0 \le x < s_x, 0 \le y < s_y, 0 \le z < s_z$

Definition 3: The operator *ptr()* is defined to imply 'pointer to'. *ptr(i)* would thus imply 'pointer to index i'.

Definition 4: Line *l* is defined as a 4-tuple of values $(x_s, x_e, y, ptrPatch)$, where $0 \le x_s \le x_e \le s_x$, $0 \le y \le s_y$ and *ptrPatch* is the pointer to a 'patch' *P,* whose definition will be provided shortly. x_s represents the start position of the line along x-axis and x_e represents the end position of the line on a particular row y of an Image. In the algorithm, lines consist of a sequence of connected foreground pixels for a constant value of y and z.

Definition 5: A patch *P* is a group of lines that belong to the same connected component. Additional attributes describing the size and shape of the connected component are also contained in *P*. Formally, *P* can be described as a 6-tuple $(L, i_{\min}, i_{\max}, z, A, ptrBlob)$ where *L* is a list of connected lines, *z* is the image number on which the patch is contained, *ptrBlob* is a pointer to a Blob data type(defined in 4.2.3), *A* is the area of the component defined by the summation

$$A = \sum_{l \in L} \left(x_e(l) - x_s(l) + 1 \right)$$

and the indices $i_{\min}$ and $i_{\max}$ are defined as:

$$i_{\min}(x, y, z) = \begin{cases} \min\limits_{l \in L} x_s(l) \\ \min\limits_{l \in L} y_s(l) \\ z = z(P) \end{cases}$$

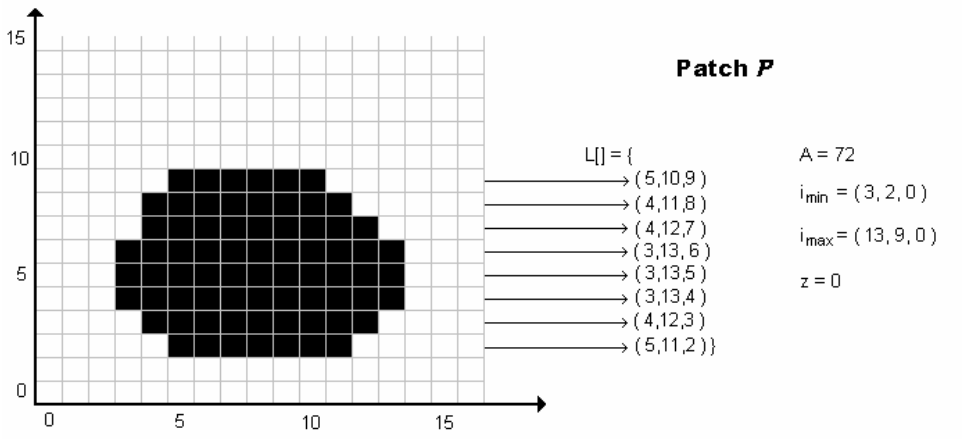


Fig. 15. Sample Connected component and the contents of the Patch *P* associated with it.

$$i_{\max}(x, y, z) = \begin{cases} \max\limits_{l \in L} x_s(l) \\ \max\limits_{l \in L} y_s(l) \\ z = z(P) \end{cases}$$

Fig.15 shows a connected component in a sample image with z value 0. The values of all the lines are shown in the *L* list, and the other attributes of the patch are shown as well.

B. The algorithm

The main advantage of the algorithm that will be described is that it collects and builds information about the connected components in the image *on the fly*. Classical implementations tend to be multi-pass algorithms, where in the first pass, the pixels are labeled, and in the second pass, equivalence classes are used to 'collect' pixels that belong to the same component.

However, a variation of the algorithm is described that takes advantage of pointers and linked-lists to progressively augment the attributes of the patches as each row of the image is processed.

For a linked list *L* of lines, we define the following operations:

- pushBack(*L*, *l*) Inserts the pointer to the line *l* at the end of the list *L*
- isEmpty(*L*) Returns *true* if list *L* is empty, *false* otherwise

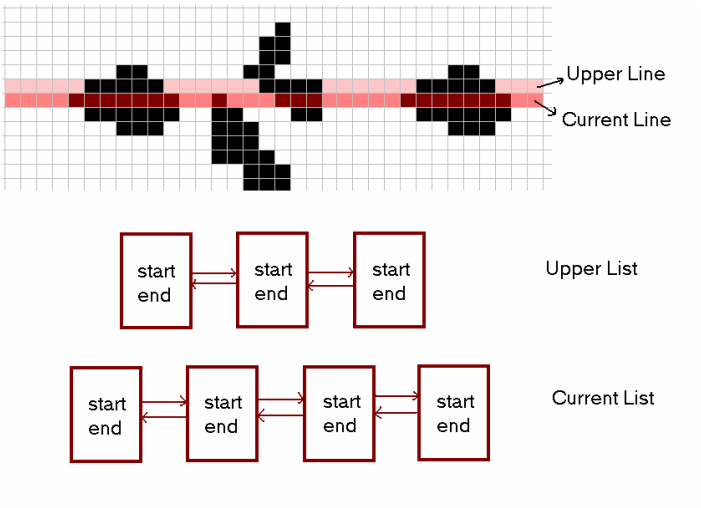


Fig. 16. Illustration for step 1 and 2 of algorithm.

- begin(L) Returns a pointer to the first line in the list L
- next(L, l) Returns a pointer to the line immediately succeeding l in L

Each row of image I is scanned from left to right. When a foreground pixel is found at position (x, y), a line l is created with starting position x_s set as x. The scanning continues until we reach a background pixel at position (x', y), or the end of the row. At this point, the end position x_e of line l is set to the value $(x'-1)$ or s_x, whichever is smaller, and the y value of l is set to the current row number.

This scanning and creation of lines is repeated till the end of the row is reached. We then compare all the new lines of this row with the lines immediately preceding it, and augment the new information to the patches that the lines point to, whenever we encounter an overlap. The algorithm is discussed in detail in the appendix (algorithm 1 and 2) but in brief is as follows:

1. Collect all lines of previous row in *UpperList*
2. Collect all lines of current row in *CurrentList* (fig. 16)
3. Check for overlap between lines of *UpperList* and *CurrentList*
4. If there is an overlap between line l of *CurrentList*, and line u of *UpperList*,
   i. if u belongs to a patch, add l to this patch

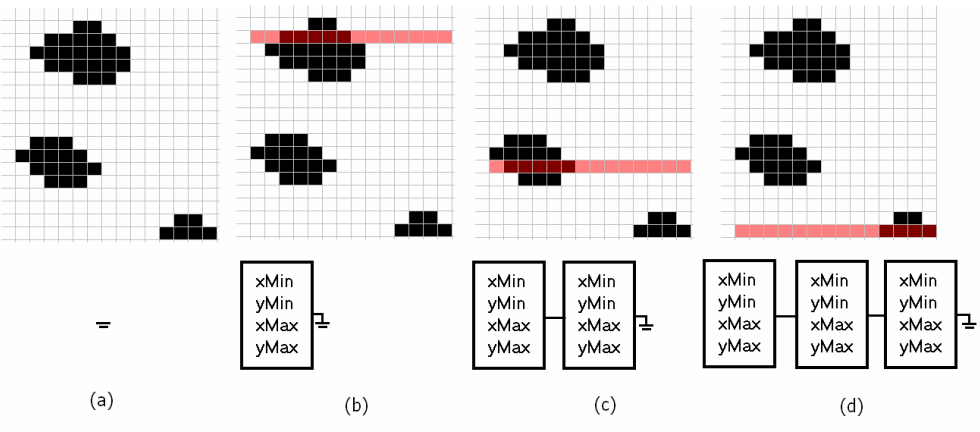


Fig. 17. Image and corresponding patch list during the scan, at (a) row = 0 (b) row = 3 (c) row = 13 (d) row = 18.

ii. else make a new patch *p*, and add *u* and *l* to patch *p*

5. For every new addition of a line to a patch,

i. update the attributes i_{min} , i_{max}, *area*

6. Update *CurrentList* and *UpperList* to their succeeding rows, and repeat all steps

At the end of the algorithm, we have a list of patches corresponding to all the connected components that were found in the image. A sample image and the resulting patch lists at various scan rows are shown in fig. 17.

5.2.2 Template matching

Thresholding and *connected-component labeling* are themselves not sufficient in attaining accurate segmentation results. This is due to the fact that the entire cell body of the neuron is not stained uniformly. Many Nissl cells are often stained more prominently near the cell boundaries, and at the center of the cell, which usually corresponds to the stained nucleus. Due to this non-uniformity in staining, thresholding often results in broken ‘eye’ shaped cells that only vaguely represent the overall elliptical shape of the cell. This problem is clearly illustrated in the images of fig.18.

Fig.18 (a) shows a section of a Nissl stained image, while fig.18 (b) shows the same image after a threshold has been applied to it. In fig.14 (c), we highlight in red two of the cells we have described above that face this problem. As is apparent in fig.14 (d),

connected component labeling results in convoluted shapes and smaller patches that do not reflect the actual shape of the cell in any way.

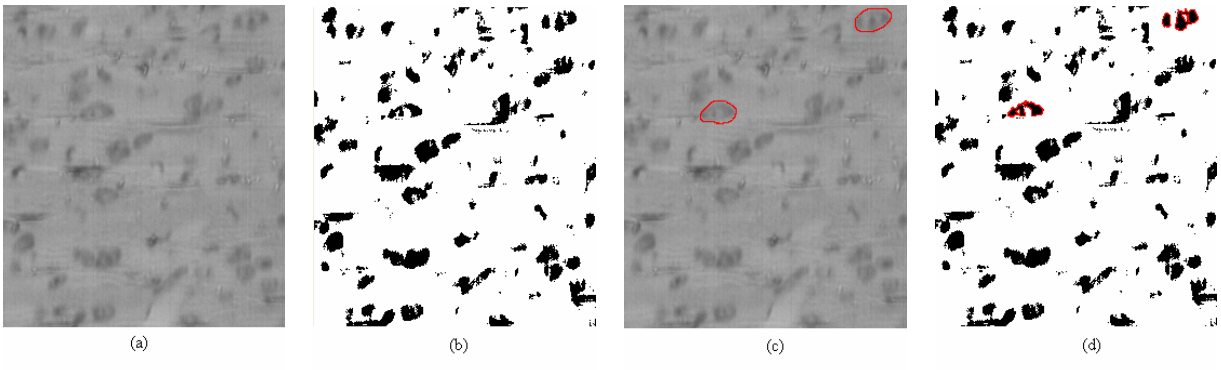


Fig. 18. (a) Image of a section of Nissl tissue (b) Thresholding applied (c) Eye-shaped cells highlighted in red (d) Same cell highlighted in the threshold image.

For the reasons showcased above, *template matching* is employed to find those positions in the image where these cells lie. Template matching is the process of comparing the image with a pre-defined sample, or *kernel* that contains the object of interest. The *kernel* moves all across the image like a moving window, comparing image pixels with *kernel* pixels at each position. The comparison is done by finding the difference between each pair of image / kernel pixels, and calculating the total sum of these differences. Points that correspond to low sums have a high probability of the occurrence of the object of interest.

Formally, a template $g(x, y, z)$ and image function $f(x, y, z)$ can be matched by running a distance metric $d(f, g)$ over the image with the template running across it like a box filter [38]. In its continuous form, the degree of matching could be calculated as

$$\int d(f, g)$$

where the distance metric is defined by

$$d(f, g) = |f - g|$$

In its discrete form, for an Image I of dimensions $s_x \times s_y \times s_z$ and a template of dimensions $t_x \times t_y \times t_z$, the following formula sums up the 'degree of matching' M, at a particular point (x, y, z) :

$$M = \sum_{t_z} \sum_{t_y} \sum_{t_x} \left| f(x, y, z) - g(x - t_x, y - t_y, z - t_z) \right|$$

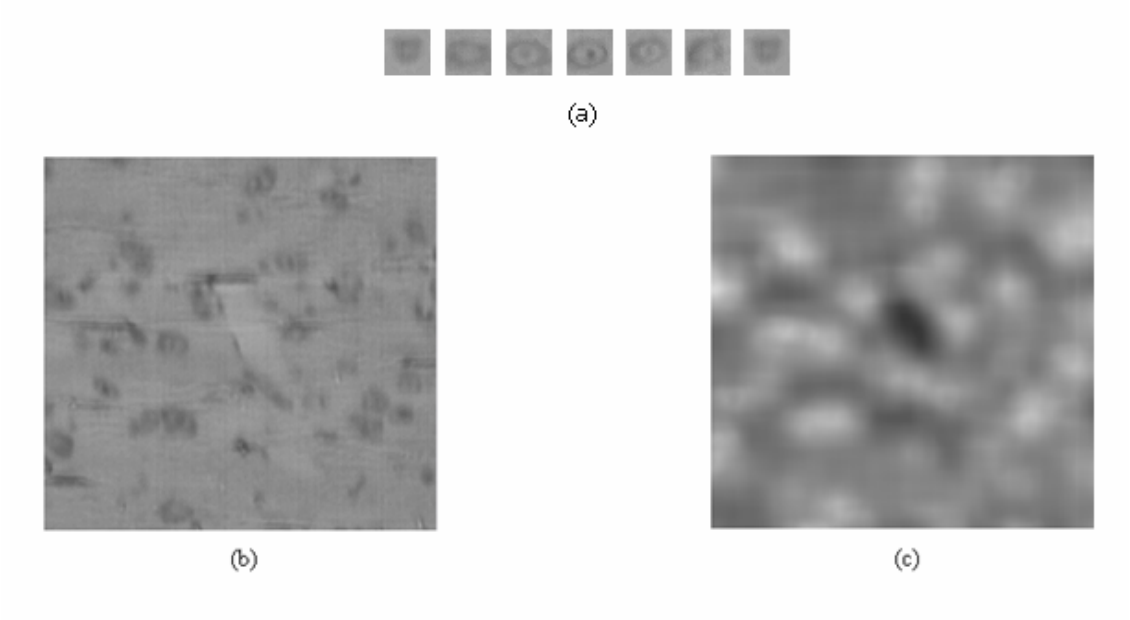


Fig. 19. (a) Kernel used for template matching (b) Original Image (c) Results of template matching.

Running this across the entire image volume, we get a new image volume I' where each point contains the 'degree of matching' with template T. Running the $25 \times 25 \times 7$ template shown in fig. 19(a) of a sample neuron over the image volume, we get template matched results such as those seen in fig. 19(c).

Once the resultant image volume I' is calculated, these results are used to make critical decisions regarding the patch segmentation. The details of how these cues from the *template matching* algorithm are used to provide better segmentation results are provided in Chapter VI.

5.3 Watershed algorithm for segmenting images of type B

5.3.1 Basic working principle

The *watershed algorithm* is a technique that makes use of image gradient cues to segment the image. It uses gray scale image data as a measure of 'elevation' rather than color value, and thus we attain a topographic representation of the data, also known as the *Digital Elevation Model* (DEM), as can be seen in fig. 20.

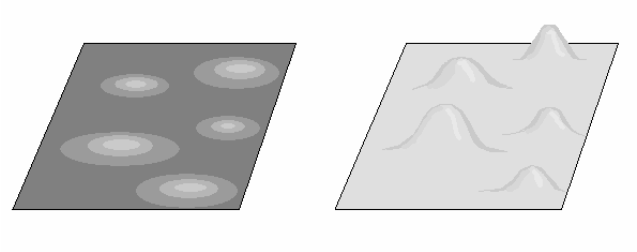


Fig. 20. Gray scale image and its Digital Elevation Model.

Now instead of using the gray scale image, if we used the gradient of the image to represent the topography, it would result in a DEM having 'walls' of separation between objects of interest, resulting from the edges that correspond to high values in the gradient.

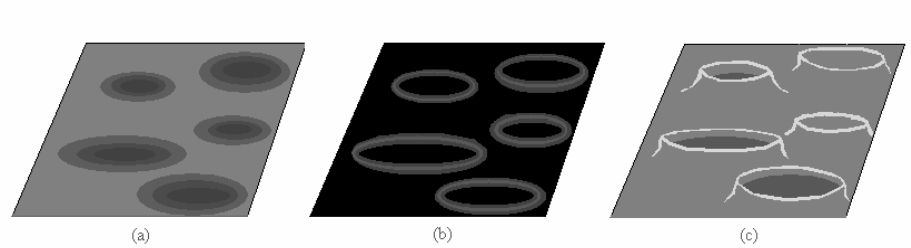


Fig. 21. (a) Original image (b) Gradient Image (c) Topographical representation of the gradient image.

This is depicted in fig. 21(b) and (c). These walls of separation are called *watersheds* and the regions they separate are called *catchment basins*. A *minimum M* of a

catchment basin is the region within the basin that is at its lowest elevation, where any drop of water that were to fall inside in the basin would eventually flow to.

If now, the DEM is filled with water, with the water level slowly increasing per iteration, the catchment basins would steadily grow in size until they reach the watersheds which would be too high to scale. At the end of the iterations, we are left with a segmentation that corresponds to catchment basins and the watersheds that separate them from each other.

5.3.2 Definitions

Definition 1: A minimum *M* of height *h* is a connected region of pixels with the value *h* from where we cannot reach a pixel of lower height without first traversing a point at a higher altitude [27]. Thus *M* can be described as a region that is significantly darker (and hence at a lower elevation) that it's neighboring pixels.

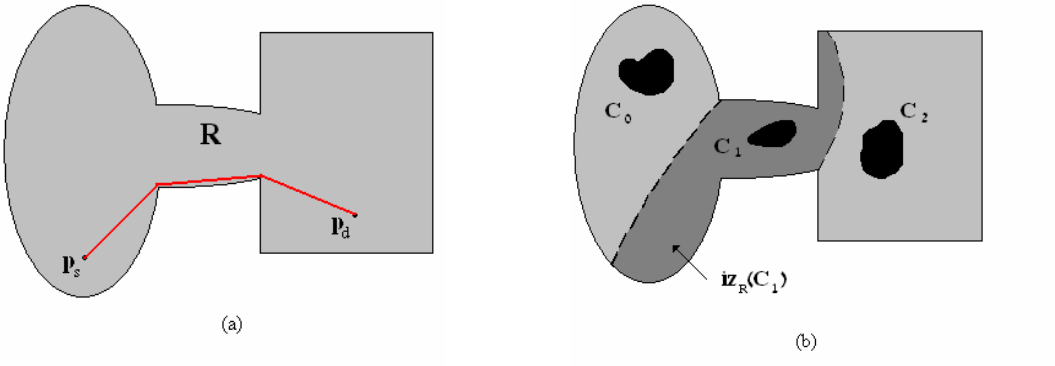


Fig. 22. (a) Geodesic distance between two points in a sample region R (b) Geodesic Influence zones.

Definition 2: The neighborhood $N(p)$ of a pixel p at index position $i(x,y,z)$ is defined by $N(p) = p(x+i, y+j, z+k) \forall i, j, k \in \{-1,0,1\}$. Since the algorithm is in 3-D, we have 26 neighbors for each point in the volume in the general case, with lesser number of neighbors along the edges and corners of the volume.

Definition 3: Each catchment basin *CB* is associated with a minimum *M* and denoted by *CB*(*M*). It is defined as the set of all points in the image volume that are

topographically nearer to M than any other minimum in the volume [29]. Thus, a water droplet falling at any point in $CB(M)$ will eventually flow into the minima at M.

Definition 4: If the image volume I has value $I(x,y,z)$ at index (x,y,z), then the range of gray scale values that I contains is given by $0 \le h_{\min} \le I(x,y,z) \le h_{\max} \le 255$ where $h_{\min}$ and $h_{\max}$ are the minimum and maximum values, respectively. The *catchment basin* at height h, $CB_h(M)$, is defined in [27] as

$$CB_h(M) = \{p \in CB(M), I(p) \le h\}$$

and the threshold image $T_h(I)$ at height h is similarly defined as

$$T_h(I) = \{p \in I, I(p) \le h\}$$

Definition 5: The *geodesic distance* between two points in a region R is the length of the shortest path between the two points such that every point on this path lies within R. Formally, if p_s and p_d represent the start and destination points, the geodesic distance $gd_R(p_s,p_d)$ is given by

$$gd_R(p_s, p_d) = \min(|p_s, p_i, p_j, \ldots p_d|) \qquad p_s, p_i, p_j, \ldots p_d \in R$$

A sample region R and the geodesic path between two points within it can be seen in fig.22 (a).

Definition 6: Vincent and Soille's definitions of *geodesic influence zones* and *SKIZ* are widely used across watershed transform literature as the standard terms to theoretically describe this segmentation technique [27]. Here, we provide the formal definitions that they introduced and proceed to explain the immersion procedure.

If R_i is a subset of region R, the *geodesic distance* between the point p_s and region R_i is:

$$gd_R(p_s,R_i) = \min(gd_R(p_s, p_i)) \forall p_i \in R_i$$

If region R consists of several connected components like C_i, the *geodesic influence zone* of C_i within R, $iz_R(C_i)$ is defined as:

$$iz_R(C_i) = \{p \in R, \forall j \in [1,k]/\{i\}, gd_R(p, C_i) < gd_R(p, C_j)\}$$

It is the locus of all points within R that are closer to C_i than any other component C_j . Fig. 19(b) shows three connected components, C_0, C_1 and C_2 and highlights the geodesic influence zone of C_1 in dark gray. The union of all the influence zones of C is defined by:

$$IZ_R(C) = \bigcup_{i \in |1;k|} iz_R(C_i)$$

Now, the boundary points between the influence zones of two or more regions do not belong to the influence zones of any component in C and these points are collectively called the *Skeleton by Influence Zones* or $SKIZ_R(C)$, given by

$$SKIZ_R(C) = A / IZ_R(C)$$

5.3.3 Recursive relation

Beucher and C. Lantuéjoul presented an algorithm that used *immersion* as an analogy to the watershed transformation [39]. This immersion technique has since been a popular approach, and is defined as a recursive relation which we shall describe shortly.

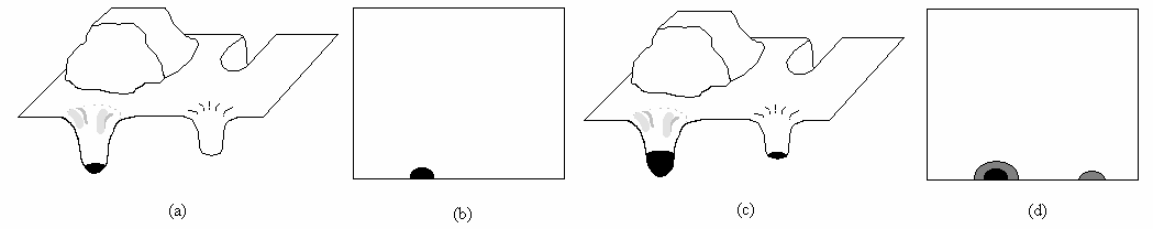


Fig. 23. (a) Topography at Immersion level h_{min} (b) X at h_{min} (c) Topography at Immersion level h_{min} + 1 (d) X at h_{min} + 1 consists of two components, one that is growing from an earlier component, and the other as a new component.

If X_h is the set of all points in the image volume at a height equal to or lower than h, then $X_h \subseteq T_{h+1}(I)$. If C is one of connected components of $T_{h+1}(I)$, then C is either an extension of a component in X_h or is an altogether new component. This observation is depicted in fig. 23 at height levels h_{min} and h_{min} + 1. The gray value in fig. 23(d) represents the new pixels arising from raising the immersion level by 1. The recursion is formally described in [29] as follows:

$$X_{h_{\min}} = T_{h_{\min}}(I)$$
$$X_{h+1} = MIN_{h+1} \cup IZ_{T_{h+1}(I)} \qquad h \in [h_{\min}, h_{\max})$$

5.3.4 Implementation

The implementation of the *watershed algorithm* takes advantage of many standard data structures like arrays and linked lists. The image volume is represented by a 3-D array of 8-bit integers, while storage of positions during the course of the algorithm is done using lists of indices. The terms ‘voxel’ and its representation ‘index’ are used interchangeably in this text.

Since the immersion procedure happens h_{max} - h_{min} times, with each iteration involving going through every pixel in the volume to see if its value is less than or equal to *h*, a more efficient way was introduced [27]. All the voxels are instead sorted in increasing order of their heights, and stored in a data structure from where they can be retrieved readily by querying the data structure with the height *h*. Once this is done, the ‘flooding’ step is where the recursion defined in 4.3.3 is implemented.

A. Sorting the image volume

Since we use 8-bit integers to store the values of the voxels, we know that the minimum and maximum values for a voxel are 0 and 255 respectively.

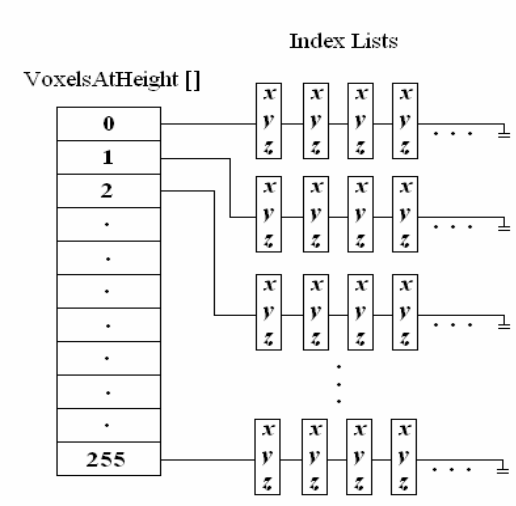


Fig. 24. Sorting data structure.

Thus, we initialize an array *VoxelsAtHeight*[256] that stores a pointer to an index at each array location. The index pointer at *VoxelsAtHeight*[*h*] points to the head of a linked list that contains all the indexes of voxels that have a height value *h.* This data structure is illustrated in fig.24. The pseudo-code for sorting by storage is available in algorithm 3 of the Appendix.

B. Immersion

By *immersion*, we mean the increasing of the 'water level' *h* in the topography, thus 'immersing' the voxels that are below this level. Starting the process with h_{min}, the algorithm continues to iteratively increase the water level, while keeping track of catchment basins, geodesic influence zones, and watershed voxels along the way. The immersion stage is basically an implementation of the recursive relation described in sections 5.3.2 and 5.3.3. Access to all pixels at the *h* and *h*+1 levels are easy with the data structure defined in 5.3.4 A.

Calculating geodesic influence zones is done using a *distance matrix* which updates distance information as and when the immersion takes place. A *label matrix* is also used to keep track of the various labels used. We maintain a running queue *Q* that is used to traverse all the pixels that are to be analyzed in the current iteration.

The algorithm takes care of both cases of voxels of height *h*+1:

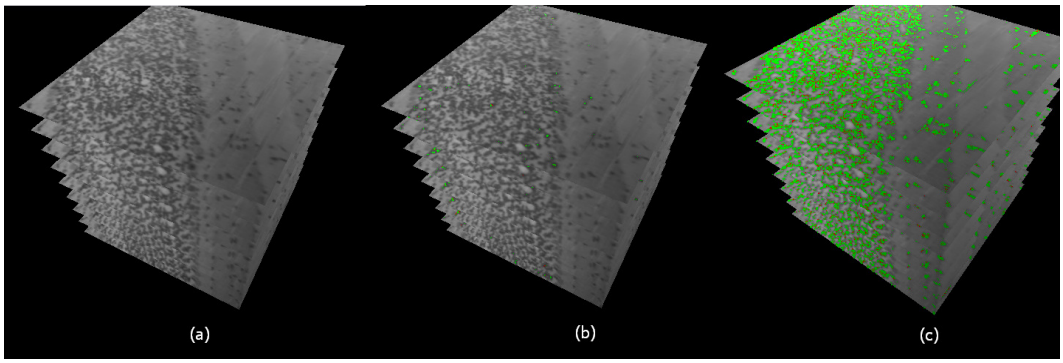


Fig. 25. Watershed algorithm in progress (a) *h* = 0 (b) *h* = *80* (c) *h* = *130*.

(a) Voxels that are extensions of catchment basins identified at height *h*

(b) Voxels that form an altogether new minimum.

In fig. 25, we see the watershed algorithm in progress, with the light green color representing the 'water' that is slowly filling up the image volume. A scaled down and slightly modified version of the 'fast watershed' algorithm described in [27] is presented in algorithm 4 and 5 of the Appendix.

Below in fig. 26, we see the water filled image in 2-D at *h = 135*

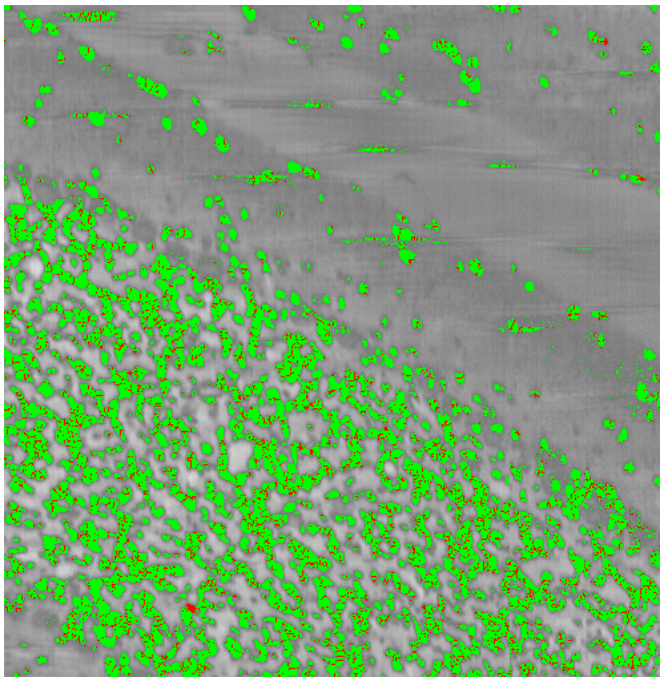


Fig. 26. 2D image of image 3 in the stack, at *h = 130*.

CHAPTER VI

ANALYSIS AND REFINEMENT

Post processing is an important stage in the filtering out of noise and segmenting the cells in the best way possible. As stated before, images of *type A* use the results from *template matching* to help further segment the results of *connected component labeling*, and the images of *type B* analyze the 3-D catchment basins that were formed to eliminate noise and split/combine cells.

6.1 Post-processing in images of type A

6.1.1 Hole-filling

Many of the components that we end up with after the processing of *type A* Images have 'holes' in them. This is either due to the inherent nature of the staining of the cell as we discussed earlier in 5.2.2, or simply because a few of the pixels inside the cell fell short of the threshold value that was chosen to separate background and foreground pixels. For these reasons, an appropriate hole-filling algorithm is used, that is illustrated in fig. 27. For detailed pseudo-code, the reader is referred to algorithm 6 in the Appendix. In its simplest form, the algorithm works as follows:

i. Create a boolean matrix whose dimensions are based on the min and max values of the patch
ii. Fill this matrix with data from the patch, as in fig. 27(a)
iii. Complement this matrix (fig. 27(b))
iv. Identify those patches that are touching the boundaries of the matrix.
v. Delete this patches and their corresponding regions from the matrix (fig. 27(c))
vi. Complement the new image (fig. 27(d))
vii. Use this patch as a replacement for the original patch. (fig. 27(e))

6.1.2 Scoring

To decide if a component fits a certain amount of 'similarity' to a standard cell, few measures have been defined to help in scoring each component. The attributes we take into consideration during scoring are as follows:

Fig. 27. Illustration of hole-filling algorithm.

A. Size

Though cross-sectional areas of a cell vary due to the shape in 3-D, it is fairly straight-forward identifying components that are too big, by comparing them to thresholds attained either by manual observation or a training data set. The 'size' of a component is defined by the number of pixels it contains. Since as defined in 6.2.1, each component is represented by a patch *P* consisting of a list of lines *l*, the number of pixels *n* is given by:

$$n = \sum_{l \in L(P)} \left(x_e(l) - x_s(l) \right)$$

B. Compactness

Compactness is a region-based shape descriptor that is given as [40]

$Cp = \rho / A$

where ρ is the perimeter of the region.

C. Circularity

The circularity measure is similar to the compactness measure, except that it is calculated by the following formula [41]:

$$Cr = \frac{4\pi \times A}{\rho^2}$$

where a value of 1 indicates a perfect circle, and deviation towards 0 implies an ellipse with progressively larger eccentricities.

D. Height-to-width ratio

The height-to-width ratio of a component represented by a patch *P* is given by the following:

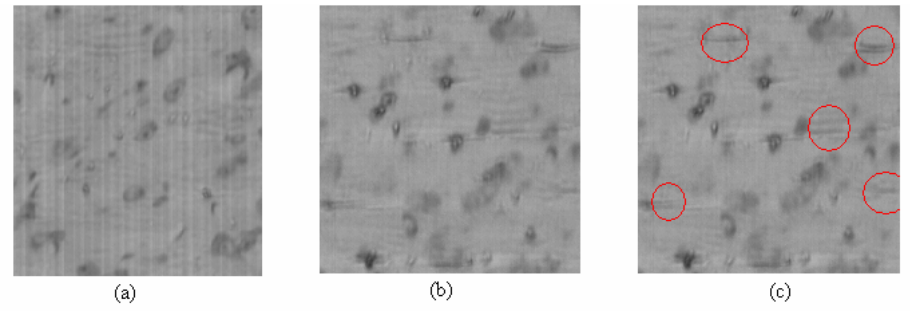


Fig. 28. (a) Image with visible chatter artifacts (b) Sample section of an image after image corrections (c) Noisy regions.

$$HTWR = \frac{y(i_{\max}(P)) - y(i_{\min}(P))}{x(i_{\max}(P)) - x(i_{\min}(P))}$$

The *HTWR* measure is a good way to spot noise, which we shall discuss below.

6.1.3 Segmentation refinement

Now that the components have been 'scored' according to the metrics described above, we can proceed to derive inferences from them to help in the post-processing.

A. Noise elimination

There are two predominant types of noise found in Nissl images. Chatter noise, and Non-Chatter noise

- *Chatter:* is the term used to express the physical phenomenon of the vibration of the sectioning knife that occurs during the slicing of the mouse brain by the KESM. Due to this chatter, there are often imaging artifacts that are left behind in the image, which generally correspond to alternating lines of low and high intensities(fig 28.a).

Though a fair amount of image processing was done on the image set well before processing for an automated cell count, some artifacts are too prominent to ignore and often mistakenly get classified as foreground pixels. However, many of these components that arise out of mistaken segmentation of chatter artifacts can be successfully eliminated using the scores we mentioned earlier. Horizontal chatter that can be seen in fig 28.b is often eliminated after image processing, but some artifacts still remain (fig 28.c).

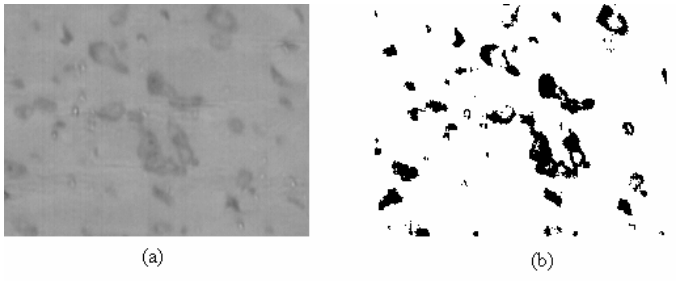


Fig. 29. (a) Section with a group of close cells (b) Threshold results.

The 'height/width' ratio helps identify components like these which are either too long compared to their height or too tall compared to their breadths. Since chatter usually occurs in the horizontal and vertical directions only, this measure is often very helpful and fairly accurate in determining noise components.

Thus if either of the following conditions for patch *P* is satisfied,

$$HTWR(P) < HTWR_{\min}$$
$$HTWR(P) > HTWR_{\max}$$

P is classified as noise and taken off from the *PatchList*.

- *Non-chatter noise:* Other noise that occurs in the image volume is eliminated fairly easily during the 3D Grouping process described in 6.1.4. Random noisy elements that occur in an image most often do not have a corresponding noisy element at the same location in the immediately preceding or succeeding images

and hence are not grouped together. Thus, the inherent nature of the algorithm itself eliminates noise by preventing its propagation in 3D.

B. Incorrectly grouped cells

Though it occurs relatively infrequently when compared to images of *type B*, cells that are close to each other sometimes end up being grouped together as part of the same component due to the threshold level that was chosen for that image. At these points, some 3D cues are made use of as well as the results of the template matching algorithm to break the large component into its constituent cells. Fig. 29 illustrates this problem.

Identifying the groups: Since standard Nissl cells are not boundless in their sizes, it is fairly straightforward to identify which of the components arising after labeling comprise of multiple cells.

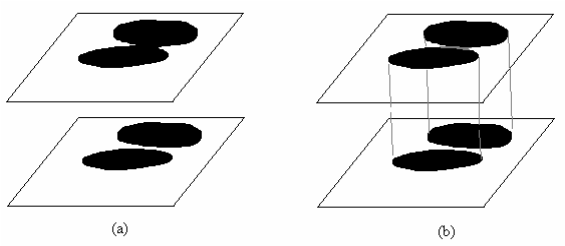


Fig. 30. A mistakenly grouped cell is split using information from the preceding image.

If the size of the patch P is n and n_{max} is the size of the largest cross-section of a standard cell, then if $n>n_{max}$, patch P is classified as a component that might possibly contain more than one cell. The methods used to help separate them are described below.

Using patch list comparisons: One option for tackling this problem is by comparing the identified patch P with patches that it overlaps in the immediately preceding or succeeding image. In fig 30(a), we see that the upper cross-section consists of a component that is clearly a combination of two cells. We use this information for splitting P into separate cells, and thus perform the split by performing binary AND operations between the two images.

Template matching: The above patch list comparisons described often do not work very well because of various reasons, namely

- Cells that are mistakenly grouped in the current image are likely to be mistakenly grouped in the preceding / succeeding image as well. This is because the distance between two cross-sections in the image stack is negligible.
- A component that in reality represents a single cell, but is incorrectly broken into two due to bad thresholding might further influence more components to incorrectly split up.

For these reasons, the results of the *template matching* algorithm are invoked to try and get better segmentation results and split the groups appropriately. In fig.31 (a) we see a standard section from a Nissl stained image, followed by its corresponding threshold image in (b).

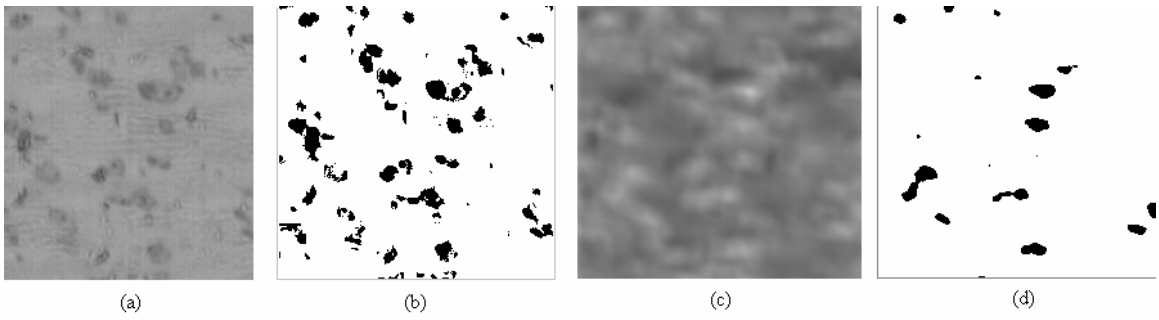


Fig. 31. (a) Original image (b) Thresholded image (c) 3D Template matching results (d) Local maxima of the matched results.

The results from the 3D *template matching* algorithm are seen in fig. 31(c). Using a threshold that classifies based on the top 5% of the gray scale range of the matched image, we arrive at an image consisting of connected components that correspond to the peaks of the matching, in fig.31 (d). Connected component labeling is run over this image, and *local peaks* in the components are collected. These local peaks correspond to *a high probability of occurrence of a cell,* not just by local 2D pattern matching but by taking into account data from about 7 consecutive images at a time. Hence, a local peak very

strongly suggests the existence of a neuron / glial cell at this point. Local peaks for the template matched patches are shown as red dots in fig. 32(a), and blue dots in fig. 32(b).

To split a large cell, we see if it contains more than one local peak within it. If it does, we use simple boolean operations like AND and MINUS with a solid circle of standard size and arrive at the results, as shown below in fig. 32(b).

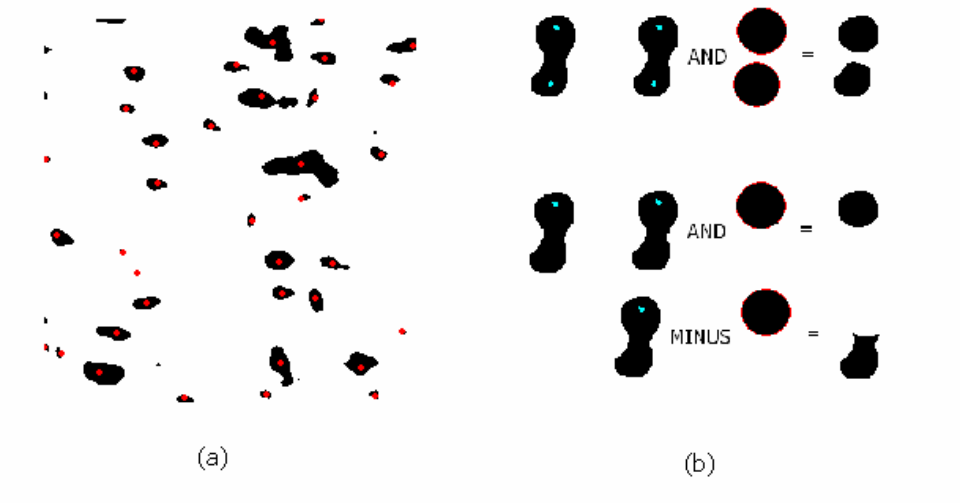


Fig. 32. (a) Local peaks (b) Splitting of a group of cells using boolean operations.

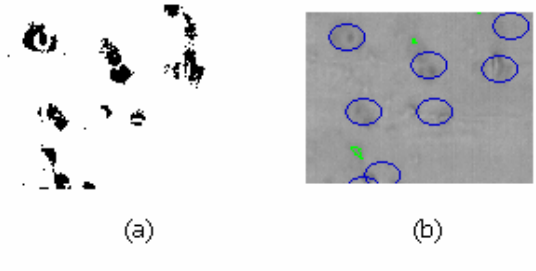


Fig. 33. Neurons with weak features as seen in (a) are marked using standard sized patches in (b).

C. Incorrectly split cells

Just as we have incorrectly grouped cells, we have neurons and glial cells that are incorrectly split into many pieces due to bad thresholding, as discussed in section 5.2.2.

This is where the process of *patch replacement* is introduced. Just like in the case of splitting grouped cells, we analyze the data we have obtained using *template matching*. If there is a local peak at a point where there are many small patches (most likely pieces of the same cell), *all* these pieces are replaced by a cell of standard size. Fig. 33 shows some neurons that corresponded to bad thresholded components being successfully marked by cells of standard size (blue outlines in fig. 33 (b))

For a formal description of the process discussed in the last two sections, the reader is referred to algorithm 7 in the Appendix.

6.1.4 Three dimensional grouping

Now that all the low scoring patches in each image have been modified, replaced or refined by some means, we can begin to combine all these patches in 3D without having to worry about erroneous 2D patches propagating in 3D. A cell in 3-D usually spans 5-6 images, reaching its largest size at the central images and diminishing in size as we move further away due to its generally spherical shape (fig. 34).

The bounding box of a patch P is the rectangle whose top-left corner and bottom-right corner is defined by the co-ordinates of its minimum and maximum index positions, $i_{min}$and i_{max} respectively. Using the bounding box for patch comparisons, we can easily eliminate patches that do not overlap with each other.

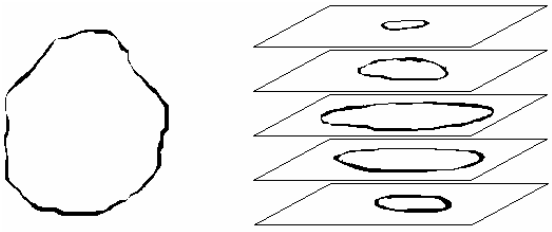


Fig. 34. Cell and its respective cross-sections.

Definition 1. A *blob B* is defined as a representation of a 3-D cell, consisting of a collection of patches attached to each other. Formally, it is a 4-tuple (L, i_{min}, i_{max}, S) where L is a list of patches, and S, i_{min}, i_{max} are defined as follows:

$$S = \sum_{p \in L} A(p)$$

$$i_{min}(x, y, z) = \begin{cases} \min_{p \in L} x\left(i_{min}(p)\right) \\ \min_{p \in L} y\left(i_{min}(p)\right) \\ \min_{p \in L} z\left(p\right) \end{cases}$$

$$i_{max}(x, y, z) = \begin{cases} \max_{p \in L} x\left(i_{max}(p)\right) \\ \max_{p \in L} y\left(i_{max}(p)\right) \\ \max_{p \in L} z\left(p\right) \end{cases}$$

For a linked list L of patches, the following operations are defined:

- pushBack(L, p) Inserts the pointer to the patch p at the end of the list L
- isEmpty(L) Returns *true* if list L is empty, *false* otherwise
- begin(L) Returns a pointer to the first patch in the list L
- next(L, p) Returns a pointer to the patch succeeding p in L

A formal algorithm for the 3D Grouping process is available in Appendix, but in brief, it is as follows:

i. Compare a pair of patches in the upper patch list and current patch list, by checking if their bounding boxes overlap. If they do not, compare next pair.

ii. If the bounding boxes *do* overlap, check if they really overlap by using the line information of both patches

iii. If this is true as well, add the current overlapping patch to the blob being pointed to by the upper patch

This algorithm is very similar to the 2D case with lines and patches. Once 3D grouping is carried out using the algorithms described in algorithms 8, 9 and 10 of the

Appendix, we are left with a list *BlobList* which contains all the blobs present in the volume.

6.2 Post-processing for images of type B

The *watershed algorithm* has a known problem of over-segmentation in the case of large foreground objects. This is because every local minimum ends up with a catchment basin of its own that is separated from other minima by watershed lines. In the case of large objects, more than one minimum often occurs within the same object, thus resulting in multiple segments per object though there ideally should be just one.

In the case of the Nissl data set however, over-segmentation is not much of a problem since the objects are extremely tiny (25-30 pixels in 2D). The minima and their catchment basins seem to represent the granular cells uniquely, i.e. one segment per cell. However, there are still noisy segments that need to be addressed. Just as in Section 6.1, we introduce certain metrics to score the segments to decide if the segments are on track. The scoring as described by Lin et. al is described as follows:

6.2.1 Scoring

A. Size

The total number of voxels that form the component is considered the 'size' or 'volume' *V*.

B. Uniformity of intensity

Since the granular cells exhibit uniformity of intensity when compared to the cells of *type A* which due to their 'eye' shape exhibit vast variations in intensity, this measure

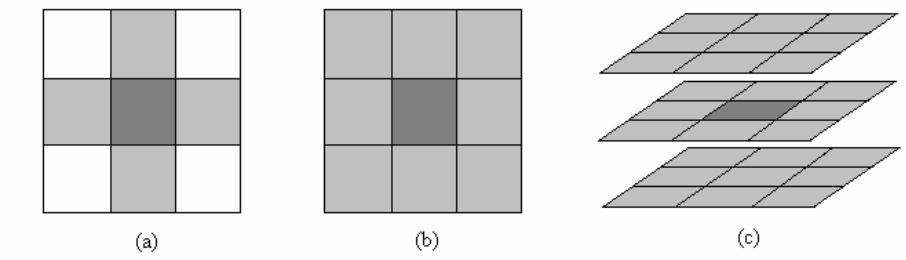


Fig. 35. Neighborhood connectivity for pixel that is (a) (a) 4-connected (b) 8-Connected (c) 26-Connected.

is a good metric for scoring *type B* cells. We use the standard deviation equation,

$$U = \sqrt{\frac{1}{n-1}\sum_{i=0}^{n-1}(v_i - v_{avg})^2}$$

where v_{avg} is the average intensity of the cell, and U is the uniformity of intensity.

C. Shape

Similar to the circularity measure defined for 2D in section 5.1.2, the shape measure is defined for a 3D object as:

$$S = \frac{|B|^3}{64\pi \times V^2}$$

where B is the number of boundary voxels in the image, and V is the volume defined in 6.2.1.A. A boundary voxel is defined as any voxel that is not completely surrounded by other foreground voxels. Since we use a 26-connected grid for our algorithm, we use the same to decide if the voxel is a boundary one or not (fig 35.c).

6.2.2 Merging

Often times a single cell gets broken up into different segments due to the formation of multiple minima and the growth of their respective catchments basins within this cell. To identify and rectify this incorrect splitting, we use the scoring mechanisms described in section 5.2.1 as well as some new merging techniques described below.

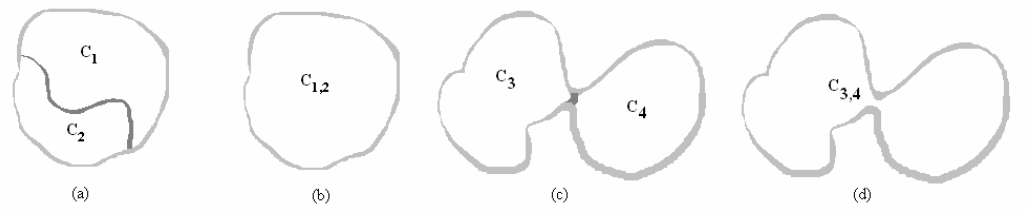


Fig. 36. (a) C_1 and C_2 before merging (b) After (c) C_3 and C_4 before merging (c) After.

The basic principle behind the decision of whether or not to merge is to compare the score of the merged segment with that of the pre-merged segments and see if the

score is significantly higher. If it is, then the watersheds separating the segments are broken, and the segments are combined. If not, they are left as they are. In fig.36, we see an example of cell segments where merging helps (fig.(a),(b)) and where it fails to make things better(fig (c),(d)).

The scoring metrics described above can be combined linearly, with a slightly higher priority given to the size of the segment. Overall score of a segment resulting out of the watershed algorithm is given by:

$$Score(C) = aV(C) + bU(C) + cS(C)$$

If cell segments C_1 and C_2 are candidates for the merging, and if $C_{1,2}$ is the combined cell segment, merging happens when

$$\frac{Score(C_{1,2})}{Score(C_1)} \times \frac{Score(C_{1,2})}{Score(C_2)} \geq \varphi$$

where φ is a pre-defined threshold for merging decisions.

6.2.3 Classification

The *watershed algorithm* described in the last few sections is very helpful in identifying granule cells. However, *interneurons* and *purkinje* cells also get segmented using this algorithm. Classifying certain cells as granule cells and interneurons is done by analyzing the features that differentiate the two.

As mentioned in 3.4.2, interneurons are found in regions where the cells are very sparsely populated, while granule cells occur in regions of very dense concentration. We can use this fact to differentiate between the two.

Fig. 37. (a) Watershed segmentation (b) Sparse / dense image (c) Final segmentation.

A kernel of size 40×40 is used to run through each image, and generate a boolean matrix that is called *DenseOrSparse*. If the number of pixels that cross a certain darkness value in this kernel window is more than a pre-defined threshold value, the algorithm outputs ‘dense’ to the boolean matrix, else it outputs ‘sparse’.

When classifying the segments, the *DenseOrSparse* boolean matrix is referred to decide how to label them. Purkinje cells are not identified using this matrix, but granule cells and interneurons have been consistenly identified. The shades of red in fig. 37(c) correspond to granule cells, while the shades of cyan correspond to interneurons.

CHAPTER VII

VISUALIZATION

This chapter describes not only the algorithm that is used to convert the volumetric data set to a 3D polygonal model for viewing, but also the various features that the graphical user interface (GUI) provides the user.

7.1 Marching cubes

The Marching Cubes algorithm is used for converting the volumetric representation to a 3D Model [42]. This is briefly described in the following sections.

7.1.1 Basic principle

Rendering what is essentially a matrix of points where each point is either 'in' or 'out' of the foreground object of interest is not as straightforward as rendering a 'pixel' or a 'dot' at each foreground point. This might work when the object is at a distance from the observer and the rendering of the dots covers all the 'holes' between each pair of dots, but if we zoom into a rendering of such a kind, the flaws in the rendering become obvious.

If we were to analyze this in 2-D for simplicity, and the 'foreground object' was represented by the dots in fig. 38(a), a straight-forward but unrealistic rendering would be attained by setting up 'sprites' at each position where a foreground pixel is seen, like fig. 38(b). We could improve on this by instead analyzing each 'square' of 4 positions each in the matrix, and looking at which of the corners of the square are classified as foreground, and which as background. Since the number of possible combinations of foreground-background pixels in each square is very limited, we can use this combination to look up a table of standard cases (fig. 38(c)), from where we can derive lines that cut across edges of the square.

Using these standard cases, we can depict the foreground object as a more solid combination of line segments that form a continuous 'boundary' for the object, enclosing the entire foreground object inside this boundary (fig. 38(d)). The essence of this algorithm is that a complicated boundary for this object is built *incrementally*, one square

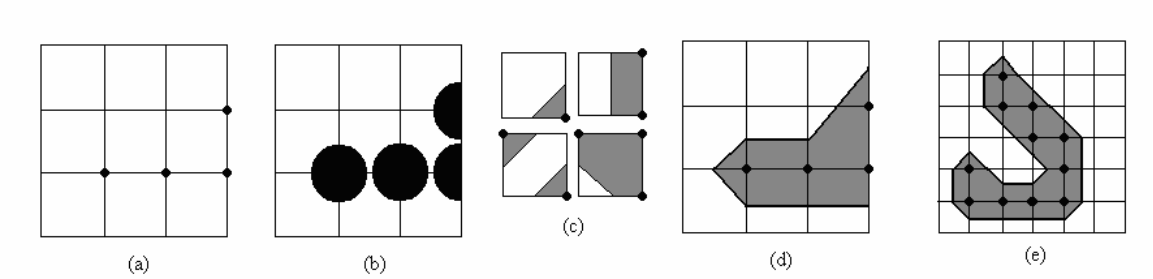


Fig. 38. (a) Initial 2D matrix (b) Representing each point by a sprite (c) Some standard 2D cases (d) Representing the set by line segments (d) Object on a finer grid.

at a time, without having to worry about the overall object in its entirety at any point. The finer the grid, the smoother the boundary looks eventually, as in fig. 29 (e). To extend the case to 3D, we have the cases as enumerated in fig. 39, with the volume being traversed one cube at a time, rather than squares. The end product of this operation is a smooth boundary for the object, which is much less jagged than a sprite representation.

The modifications we made on the algorithm, as well as the images that were created are shown in the sections to follow. For a more formal description of the algorithm, the reader is referred to algorithm 11 in the Appendix.

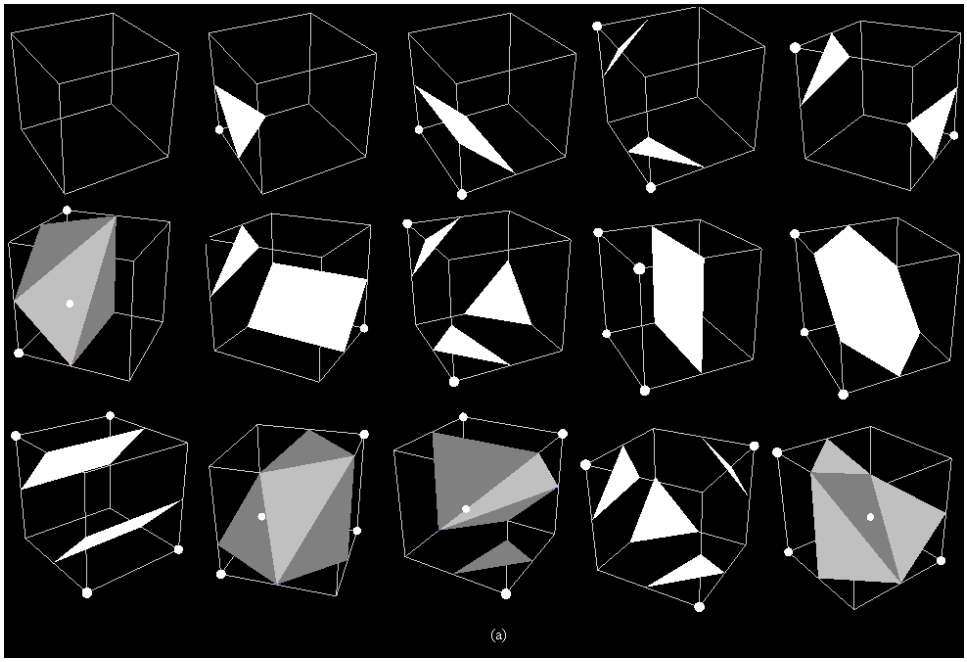


Fig. 39. Fifteen standard cases used in 3D marching cubes [42].

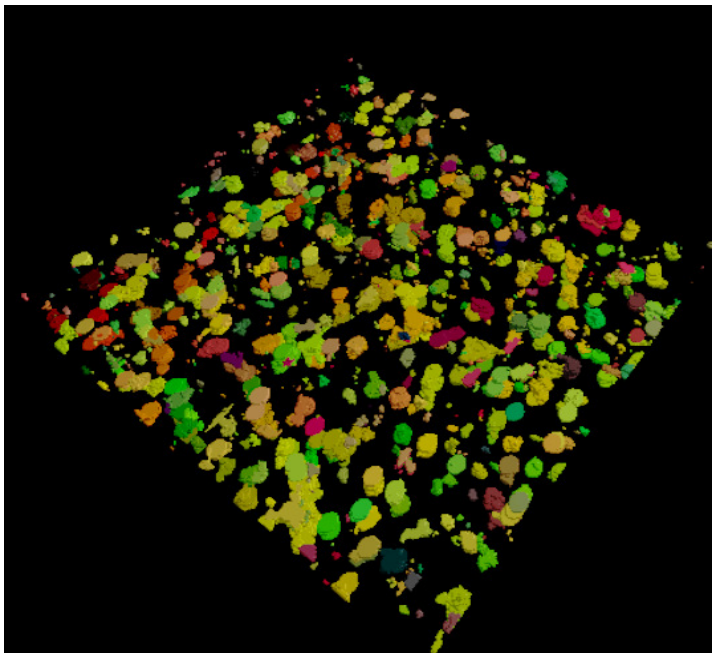


Fig. 40. Representation of a stack of 10 *type A* images, where each cell has a random color.

7.1.2 Modifications

Since the volumetric dataset is anisotropic, i.e. distances along the z axis are much larger than distances along the x and the y axes, certain modifications were made to the resulting heights of the rendering. Apart from these, the implementation was a standard one as described in [42].

7.1.3 Visualization results

In fig. 40 and fig. 41, we see the results attained by using marching cubes on images of *type A*. Here, to help the user identify the distinction between cells, random colors were used. We see the rendering of images of *type B* in fig. 42, with a zoomed in view available in fig. 43.

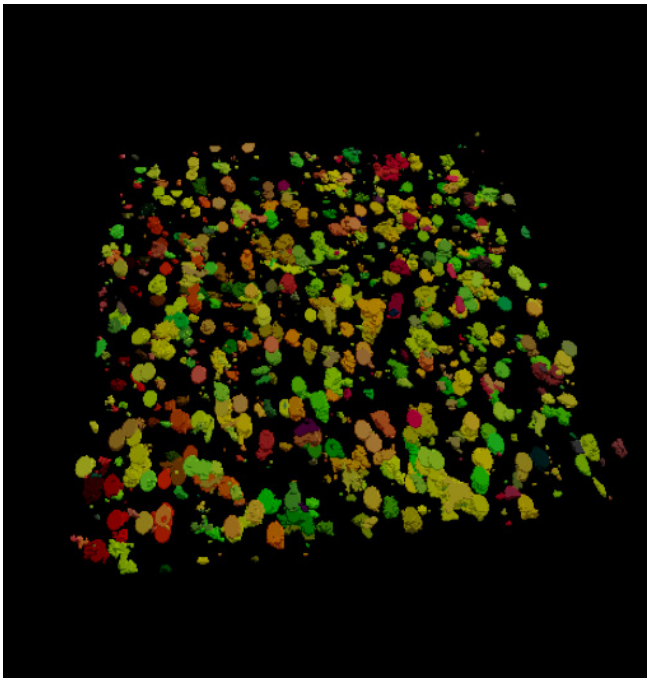


Fig. 41. Alternate orientation of dataset using in fig.40.

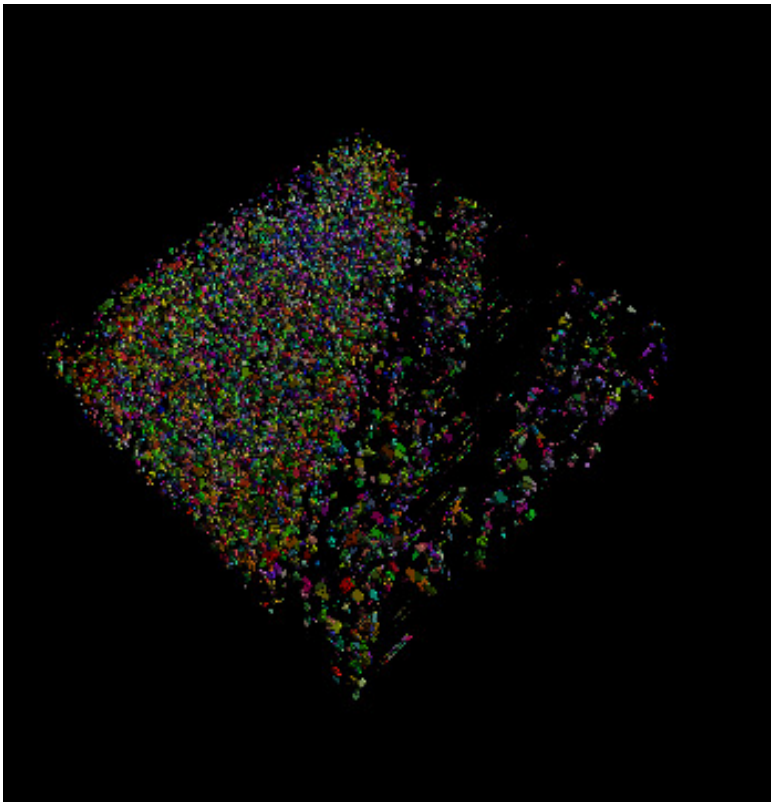


Fig. 42. Visualization of image stack of 10 *type B* images.

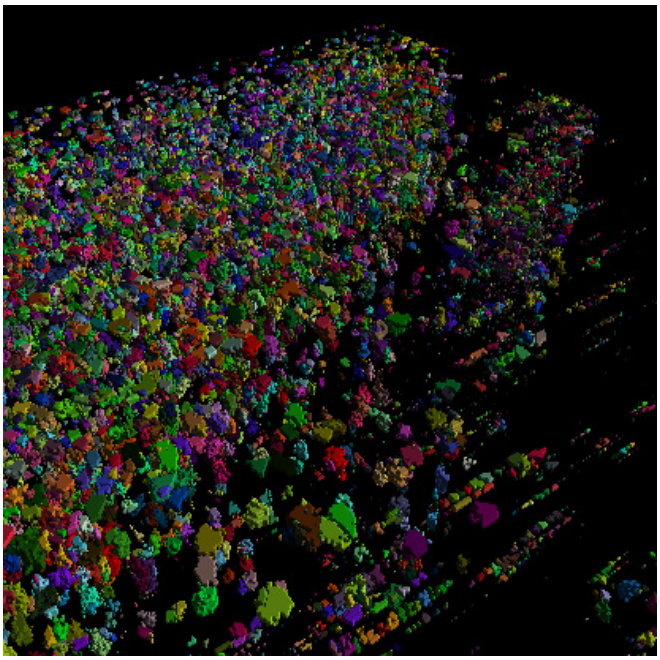


Fig. 43. Zoomed-in view of data set used in fig.42.

7.2 Graphical user interface

The interface for this application was developed using the OpenGL User Interface Library (GLUI), which is an open source library in C++ that can be downloaded and used by programmers. It provides an easy to use, intuitive and flexible platform from where the user can examine the image set and analyze the resulting segmentation, both in 2D and in 3D. Here are some of the main features of the interface.

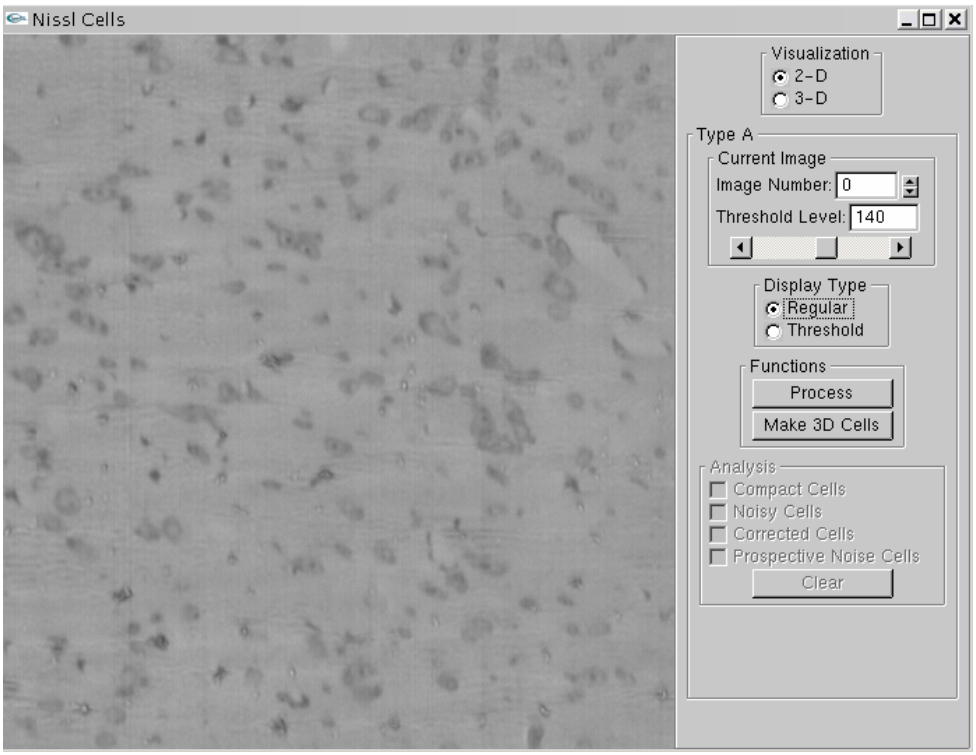
7.2.1 Choosing segmentation type

Though we can use certain image cues like average intensity to decide if a certain image is of *type A* or *type B*, we leave this task to the user to tell the application what images in general it is going to be analyzing. This prevents algorithms of one type from unsuccessfully trying to segment images from the other type.

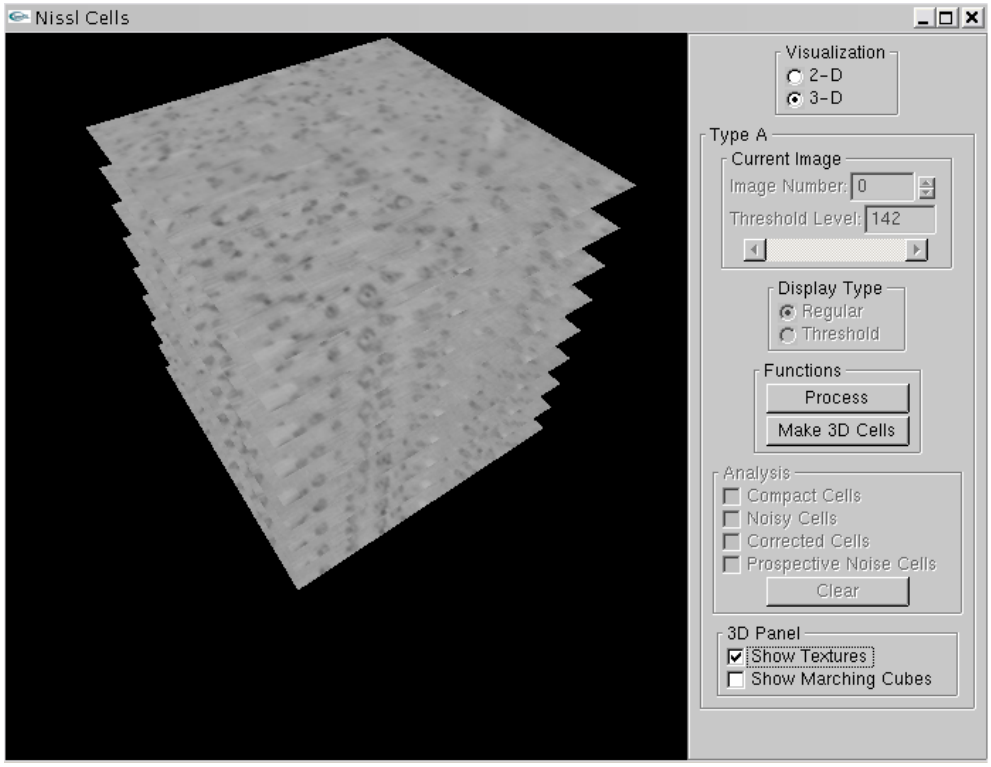
7.2.2 Type A

i. Choosing between 2D and 3D

While visualizing in 3D can be helpful in looking at the overall topology of the cells, 2D is helpful in analyzing the boundaries segmented in each image in detail. The 2D and 3D representations are shown in fig. 44.



(a)



(b)

Fig. 44. Interface shown for *type A* images in (a) 2D and (b) 3D.

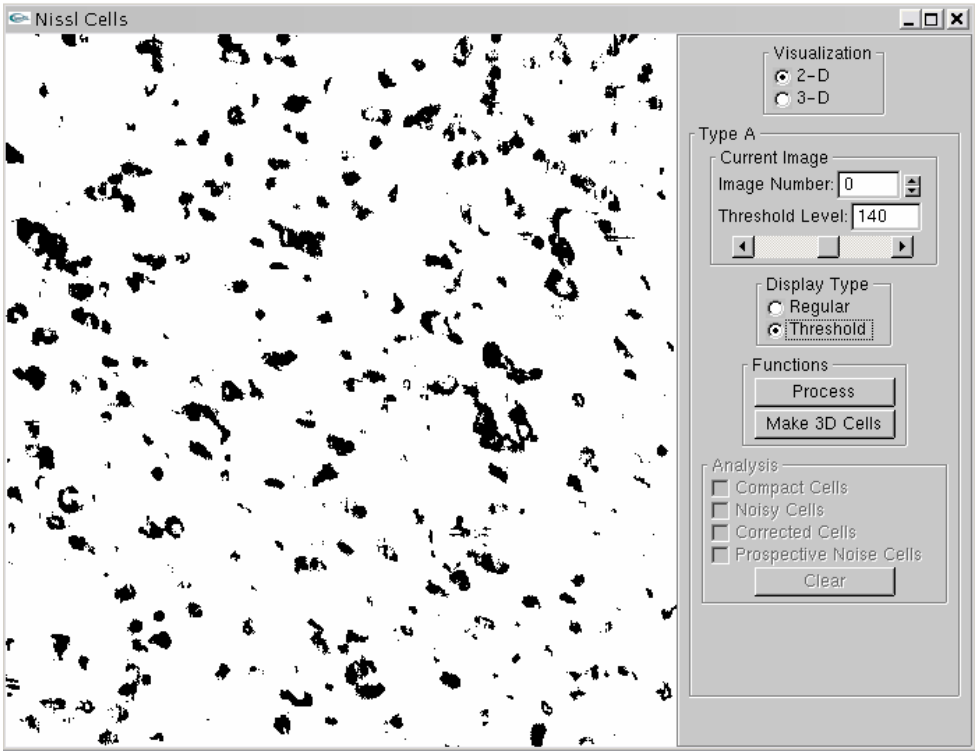


Fig. 45. *Type A* thresholded image at threshold value t = 140.

ii. *Traversing the image stack*

Spinner controls are used to shift through the images in the stack, one by one. This helps the user track the segmentation of certain cells as he moves through the stack. We see this spinner control in fig. 44 and fig.45 in the 'current image' panel.

iii. *Threshold control*

A slider is used to change the threshold value of the current image on the fly, in case the user feels that the automated threshold value chosen was too low or too high. *Type A* images with their threshold value at 140 and 133 are shown in fig. 45 and fig. 46 respectively.

iv. *Display type panel*

To switch between the regular image (fig.44) and the thresholded image (fig 45), the 'display type' panel is equipped with a radio button to enable the user to make this choice. Option to view *template matching* results are also provided after processing takes place.

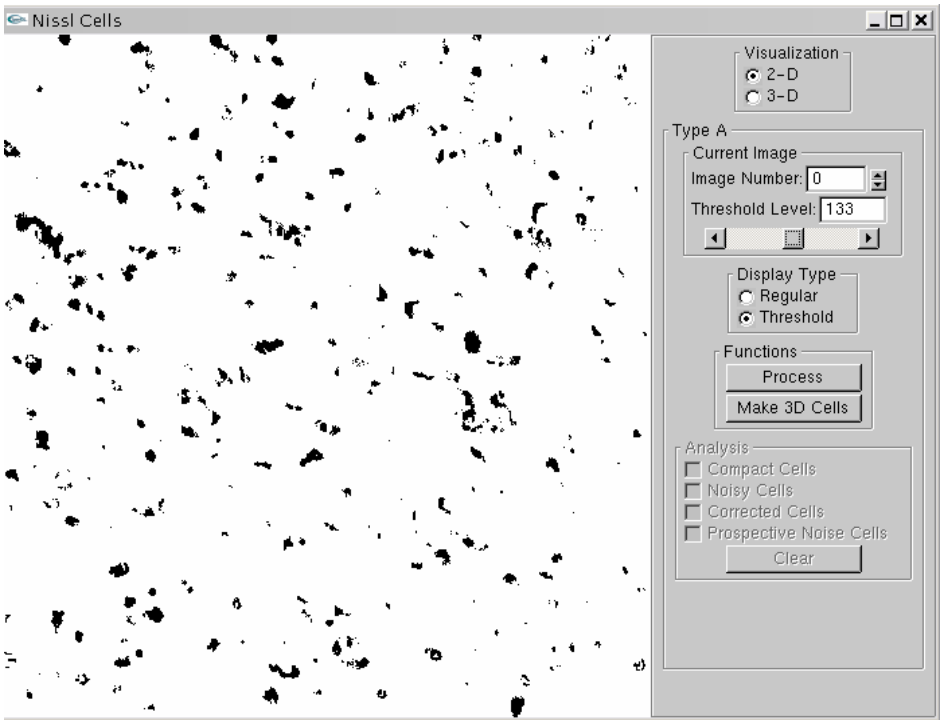


Fig. 46. *Type A* thresholded image at threshold value t = 133.

v. *Functions panel*

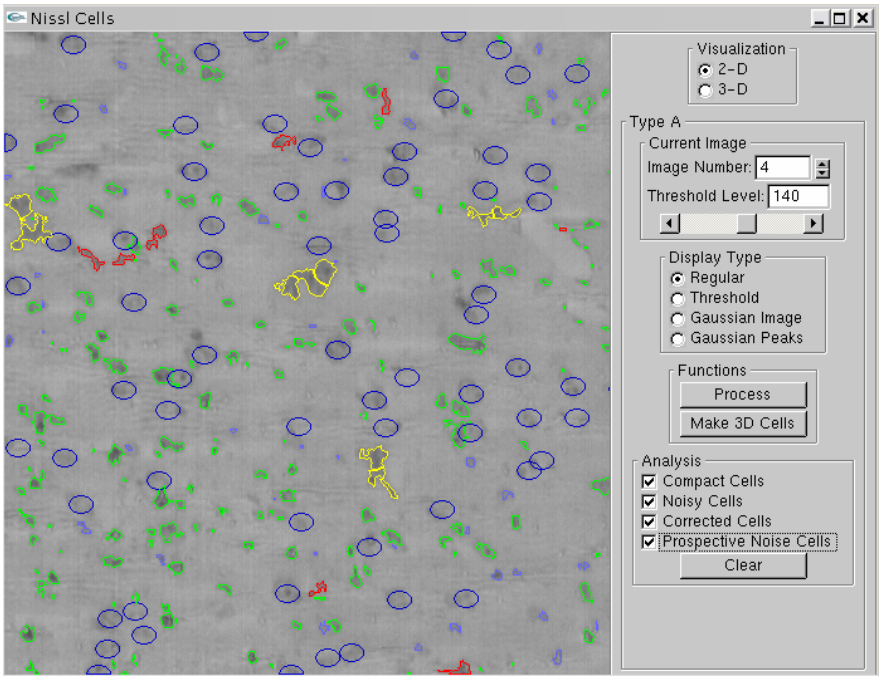
At what point the algorithm segments the images, converts to a 3D representation and so on are in the hands of the user. These buttons are present in the 'functions' panel in last few images.

vi. *Analysis panel*

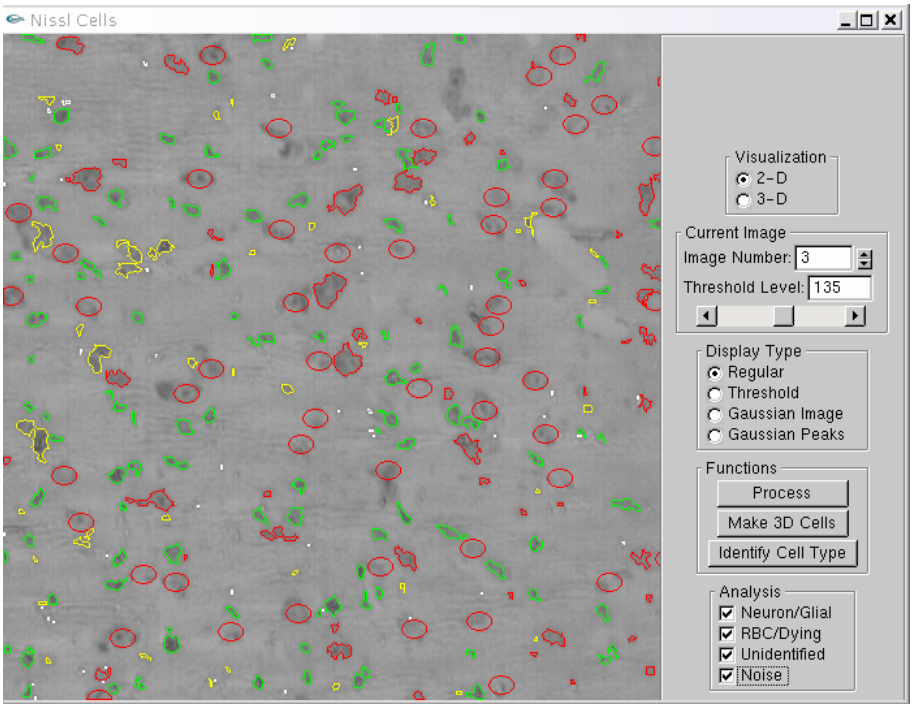
This panel is 'activated' only after the *process* operation has taken place. The algorithm analyzes the stack using the *type A* algorithms, and the user is then free to analyze the results. The preliminary classification before 3D grouping is: *compact*, *noisy*, *corrected* (replaced), and *prospective noise* cells. We see the boundaries of these respective types outlined in fig. 47(a), where *compact* cells are outlined in green, *noisy* in red and yellow, *corrected* in dark blue and *prospective noise* in light blue.

After 3D grouping, more useful information is now available. We are able to classify as *neurons* / *glial*, *red blood /dying* cells, *noisy* cells and *unidentified* cells,

as seen in fig. 47(b), where *neurons* are outlined in red, *red blood cells* in green, *unidentified* in yellow and *noise* in white.



(a)



(b)

Fig. 47. *Type A* image shown with (a) preliminary classification: compact, noisy, corrected, prospective noise and (b) final classification: neuron/glial, rbc/dying, unidentified and noise.

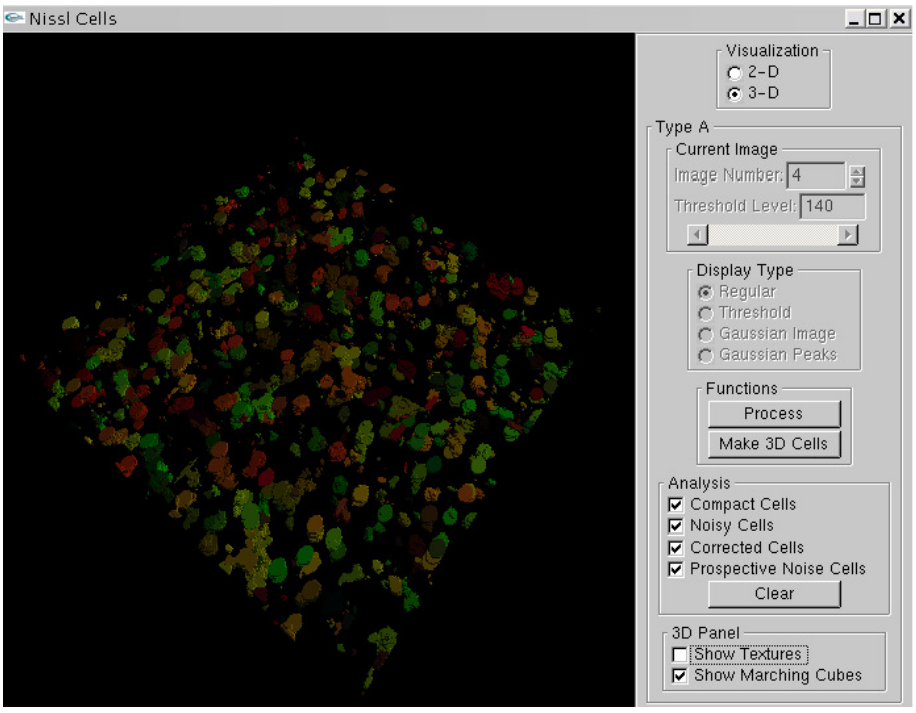


Fig. 48. *Type A* images in 3D after processing with a random color assigned to each cell.

vii. *3D panel*

The options of showing the image textures / marching cubes representation is available to the user in the 3D panel, as seen in fig. 48. We see the classification in 3D in fig. 49, where green represents *neurons*, red represent *red blood cells* and *blue* represents unidentified cells.

7.2.3 Type B

Most of the features are similar to *type A,* and so we only highlight those interface features here that are unique to *type B* segmentation.

i. *Display panel*

We are able to view not only the watershed growing through the image volume, but also see unique labels with random coloration, classified cells with a set color scheme, and various other representations in 3D. We see some of these features in fig. 50 and fig. 51.

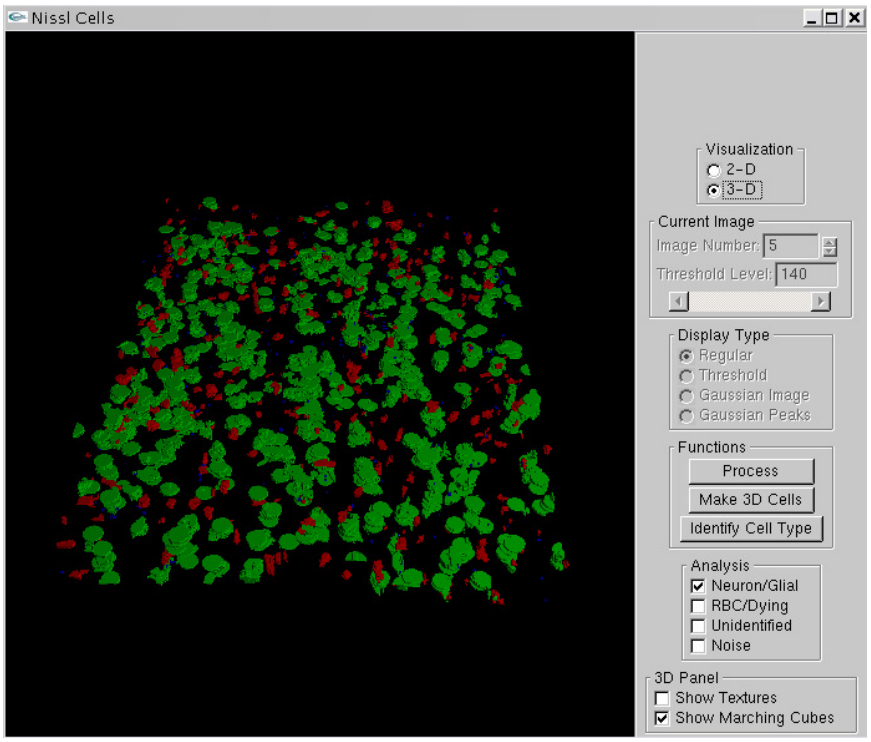


Fig. 49. *Type A* images in 3D after classification.

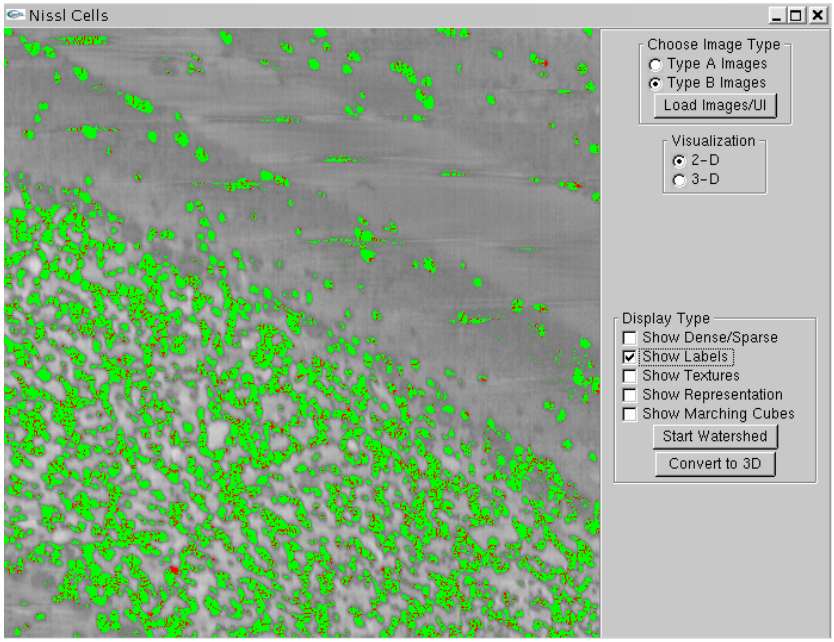
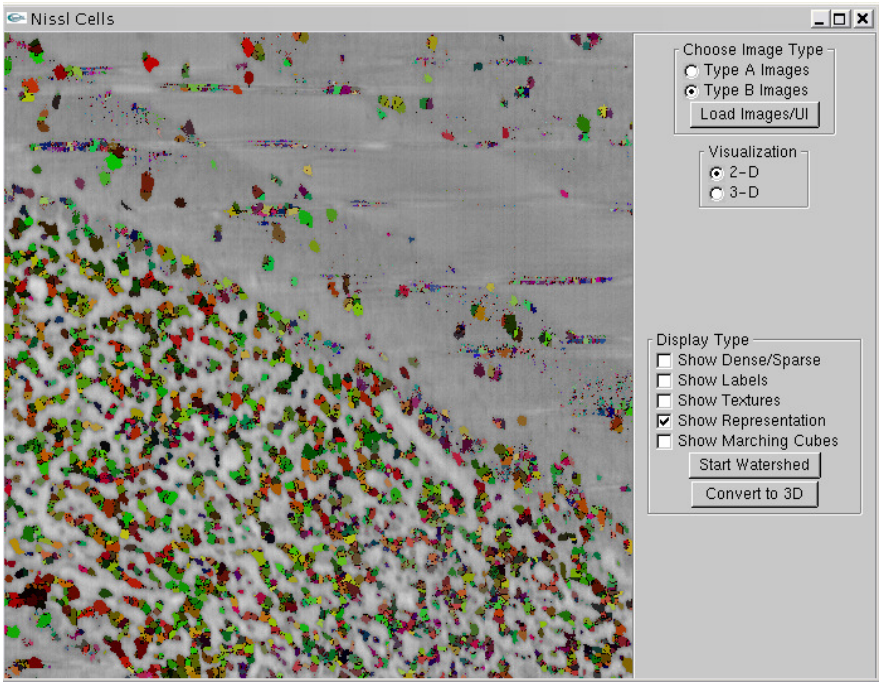
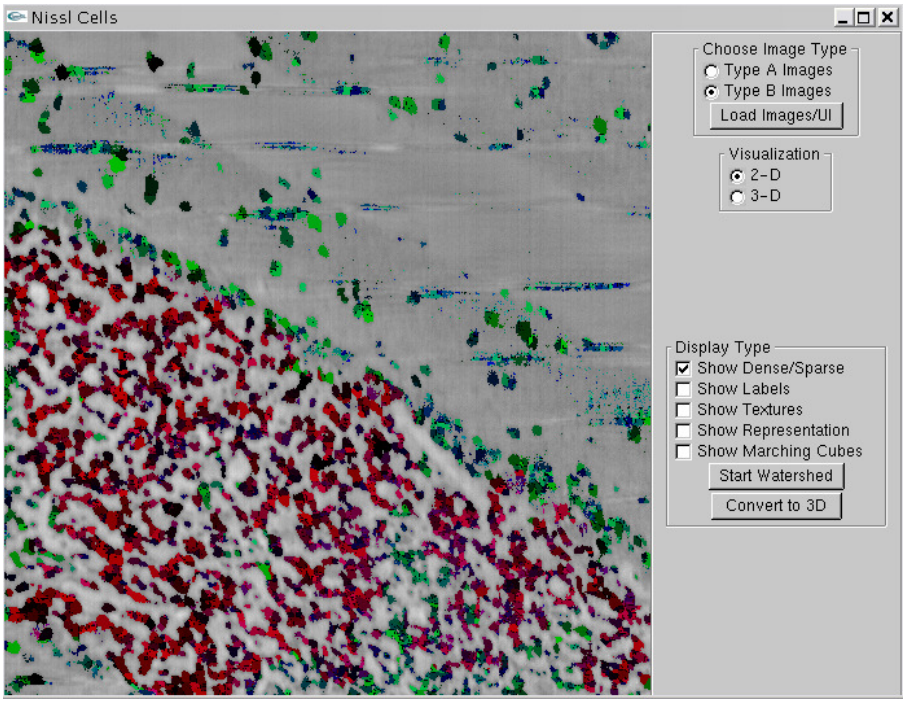


Fig. 50. *Type B* image in 2D during the running of the watershed algorithm.



(a)



(b)

Fig. 51. *Type B* cells classified uniquely using (a) random coloration (b) coloration based on cell type: granule cells or interneurons.

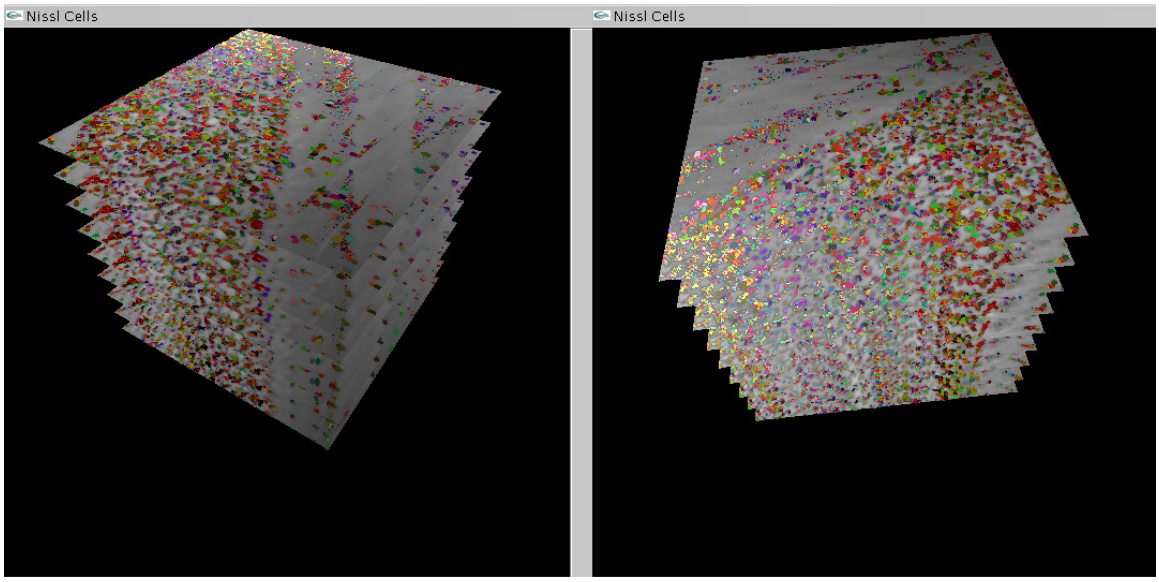


Fig. 52. The ‘representation’ view of the labeling by the watershed algorithm.

We also see in fig. 52 the ‘representation’ option in 3D, which is basically a unique coloration applied to each cell and superimposed on the originally image textures.

ii. *Functions*

The user is given the control to ‘start watershed’ algorithm and to ‘convert to 3D’ for purposes of 3D viewing.

CHAPTER VIII

RESULTS

The procedures described in this thesis lead to segmentation results that are very comparable with those arrived at by human cognition. The segmentation results of the independent experts are first described, and then a comparison and explanation of the results achieved by the techniques discussed in this thesis is provided.

8.1 Details of experiment

Since the best way of comparing the results from automated segmentation is by comparing them with those arrived at by human cognition, experts in the field of neuroscience were asked to mark the cell bodies they found in the image sets that were provided. They were given 3 each of *type A* and *type B* images. The *type A* images were of size 256×256 px, while those of the latter were 128×128 px.

The experts proceeded to mark with different colors, the various cell bodies they were able to identify in the individual sections. In *type A*, neurons and glial cells were marked by a red dot, red blood cells and cells that were dying were marked by yellow dots and endothelial cells by blue dots. In *type B*, granule cells were marked by red dots, purkinje cells by green dots, Interneurons by yellow dots and endothelial cells by blue dots.

In fig. 53, we see the *type A* images that were supplied in their raw form before segmentation, as well as the results after the expert identified the cells in each image. Similarly we see the raw and labeled *type B* images in fig. 54. Each image was labeled by what they look like to the observer in that *one* image alone. For instance, a solid dark region is labeled as a ‘red blood cell’ *even though* it might actually be a neuron on further investigation of the preceding and succeeding image. For this reason, certain labeling corrections were made to the set, which we describe in the section 8.2.

Cells of *type B* that are closely packed are labeled granule cells, while those that occur sparsely in the section are labeled Interneurons, as discussed previously in section 6.2.3.

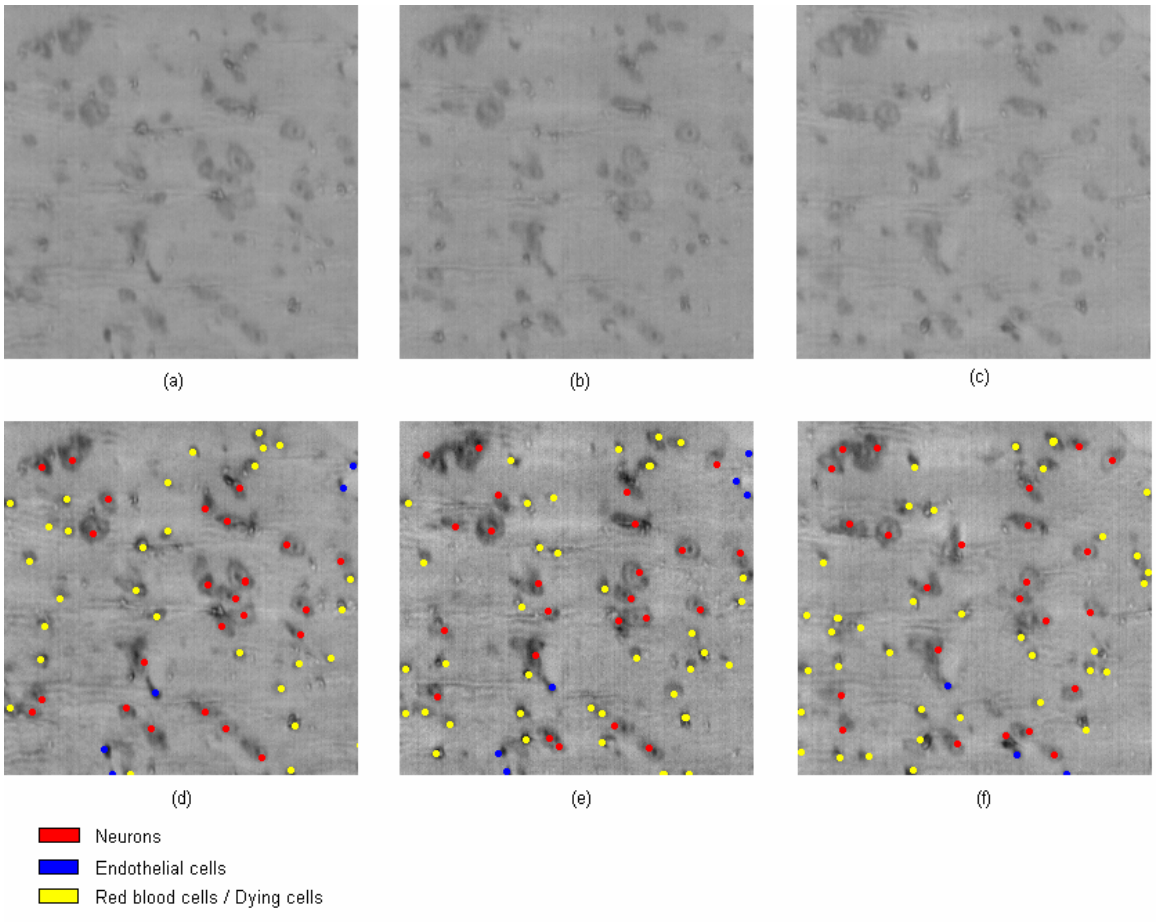


Fig. 53. (a), (b), (c) Type A images provided for segmentation by independent experts
(d), (e), (f) Marked images where the cell types are labeled by the colors red, blue and yellow.

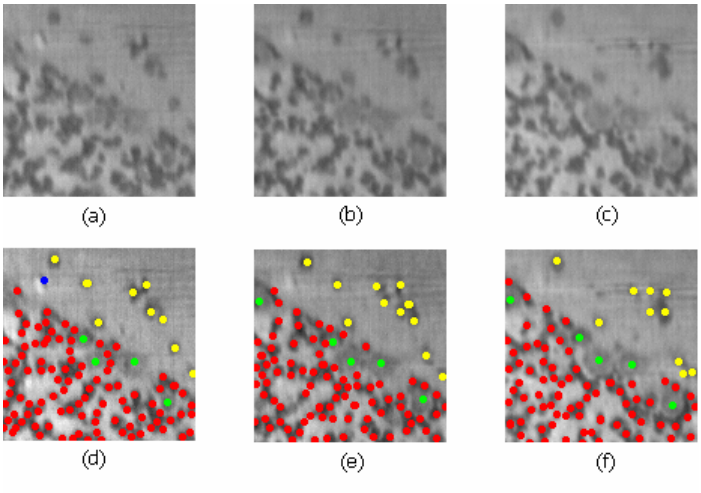


Fig. 54. (a), (b), (c) Type B images provided for segmentation
(d), (e), (f) Marked images where the cell types are labeled by the colors red, blue, green and yellow.

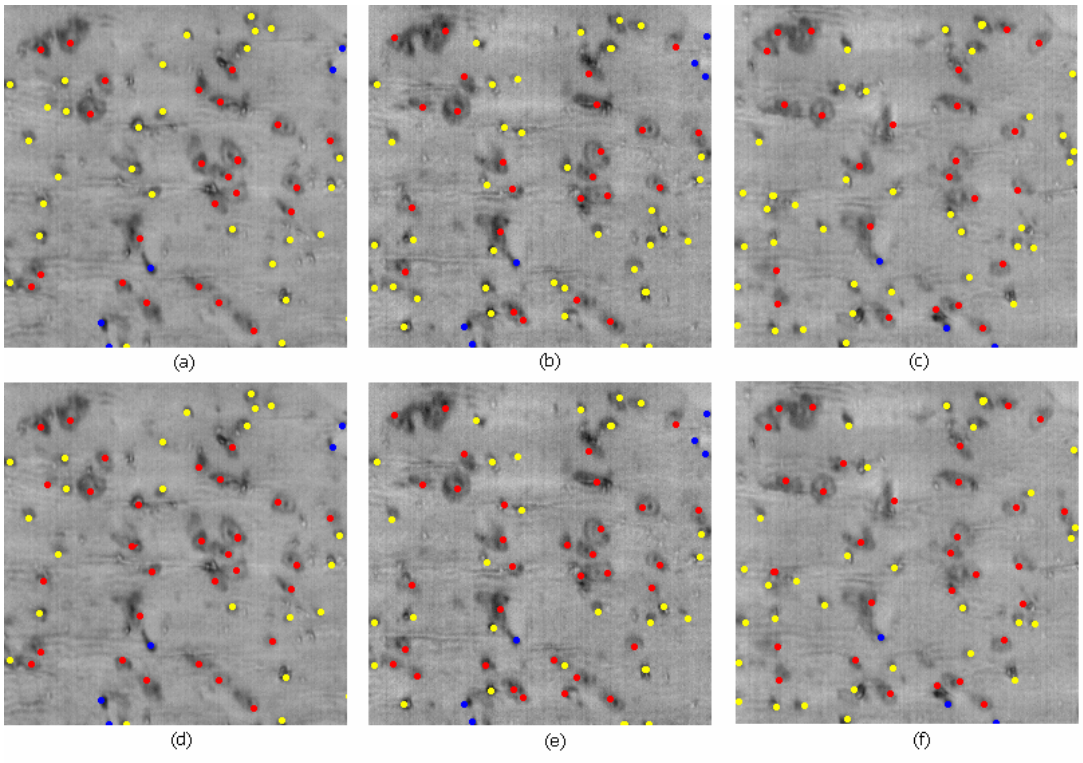


Fig. 55. (a), (b), (c) Original labeling (d), (e), (f) After 3D labeling corrections.

8.2 Labeling corrections

For comparison between our results and those of the expert, a 3D consolidation process has to take place, because the automated process is the result of a 3D combination and not merely 2D segmentation. Thus, the results of the experts are combined in 3D using certain rules that were agreed upon, as described below.

In *type A,* any cell that is classified as a 'neuron' in even one of the images must be labeled as a neuron in *all* the remaining images as well. This is because the key characteristics of the neuron (central dark nucleus and elliptical outer body) might be seen only in a few images that contain it. *Thus, only a yellow dot (non-neuron) can be corrected to a red dot (neuron), and not vice versa.* No corrections are made to endothelial cells. We see these corrections in fig. 55. Only a handful of cells change their label during this correction process.

No corrections were made to *type B* cells as all cells had the same label across the different images.

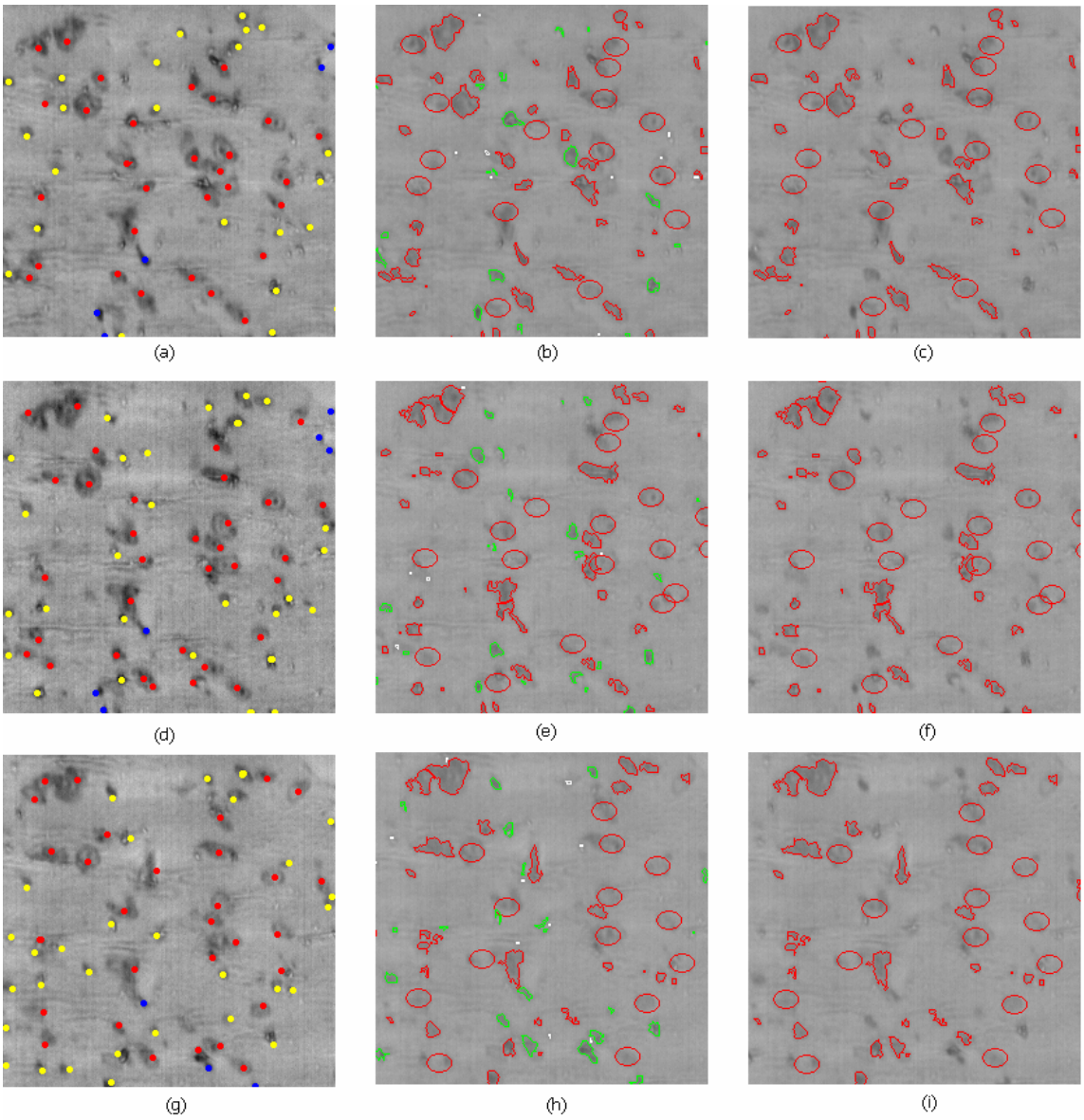


Fig. 56. (a), (d), (g) Cells identified by expert (b), (e), (h) All cell bodies identified by algorithm (c), (f), (i) Neurons/Glial cells identified by algorithm.

8.3 Type A results

In fig. 56, we see the results attained by human cognition compared with those of the algorithm. The color scheme for the expert's results is the same as in fig. 53 while the color scheme for automated results is as follows: Red outlines indicate neurons and glial

cells, white outlines indicated tiny noisy elements, and green is used for red blood cells / dying cells.

TABLE 1

Comparison of manual and automated results for segmentation of cell bodies

Img	Manual $num(\mathrm{cb_m})$	Automated $num(\mathrm{cb_a})$	1-1 Correspondence $num(\mathrm{cb_{corr}})$	Percentage identified $\frac{num(cb_{corr})}{num(cb_m)}\times 100$	Percentage extra / missed by expert $\frac{num(cb_a)-num(cb_{corr})}{num(cb_a)}\times 100$
1	57	62	55	96.5 %	11.3 %
2	66	65	60	90.9 %	8.3 %
3	63	59	57	90.5 %	3.4 %
3D	72	68	62	86.1 %	8.8 %

TABLE 2

Comparison of manual and automated results for segmentation of neurons

Img	Manual $num(\mathrm{n_m})$	Automated $num(\mathrm{n_a})$	1-1 Correspondence $num(\mathrm{n_{corr}})$	Percentage identified $\frac{num(n_{corr})}{num(n_m)}\times 100$	Percentage extra / missed by expert $\frac{num(n_a)-num(n_{corr})}{num(n_a)}\times 100$
1	30	40	25	83.3 %	37.5 %
2	32	45	27	84.4 %	40 %
3	31	40	26	83.9%	35 %
3D	33	45	28	84.8%	37.7 %

Since the focus is on identifying neurons, we have a separate column of images in fig. 56 showing only the outlines of neurons that were identified by our algorithm.

We tabulate these results in Table 1 and Table 2. We find that the ability of our algorithm to find general cell bodies in the image set is very strong. Examining the numbers in the '1-1 Correspondence' column, we see that a high percentage (86 %) of those cell bodies identified by the expert have also been successfully identified by our automated process. This result is encouraging, considering that many cell bodies,

especially neurons, tend to get split up into multiple pieces due to their non-uniform staining during the thresholding process. This shows that the patch replacement using template matching has been overall a successful approach to this problem of non-uniform staining of cells. In the case of further classifying these cell bodies into neurons and non-neurons, the results are promising as well. The percentage of neurons found by the expert that the algorithm was also able to identify is reasonably high as well (85%).

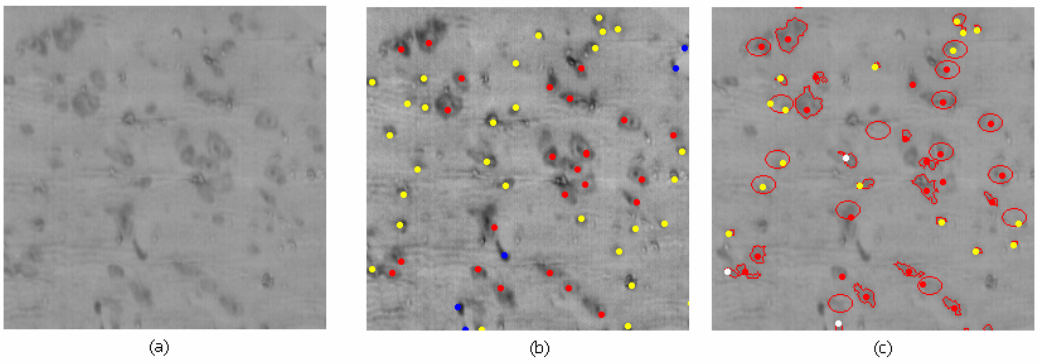


Fig. 57. (a) Original image (b) Segmentation: first pass (c) Corrections by expert during second pass.

However, there are many neurons that are ‘extra’ in the automated segmentation. Now, these extra neurons could be either of the two following cases:

- Cells that were incorrectly marked as neurons by our algorithm.
- Cells that were actually neurons but were missed by the human observer when segmenting the first time.

To find out which of these categories the extra neurons fell into, the automated results were shown to the expert for a second look, to confirm whether each cell was indeed incorrectly marked, or a real neuron. The results of the expert did not change much from that of the first time, though there were a few cells that the observer conceded to have missed during the first time. In fig. 57 we see one such sample, where the 3 white dots in fig. 57 (c) correspond to cells that the algorithm helped the observer find. In this sample, we find that out of the 18 extra cells, 3 were conceded to be actually neurons,

while the rest of the 15 were thought to be red blood cells that were incorrectly marked as neurons.

This excess of cells classified as neurons can be explained due to the following:

- The automated experiments were conducted using an image stack of 20 images with the kernel comprising of 7 cross-sections, out of which the subset corresponding to the images handed to the expert were extracted from this stack and compared. It is thus possible that the algorithm saw neurons in other sections (beyond those given to the expert) that then propagated its labeling into this smaller subset.

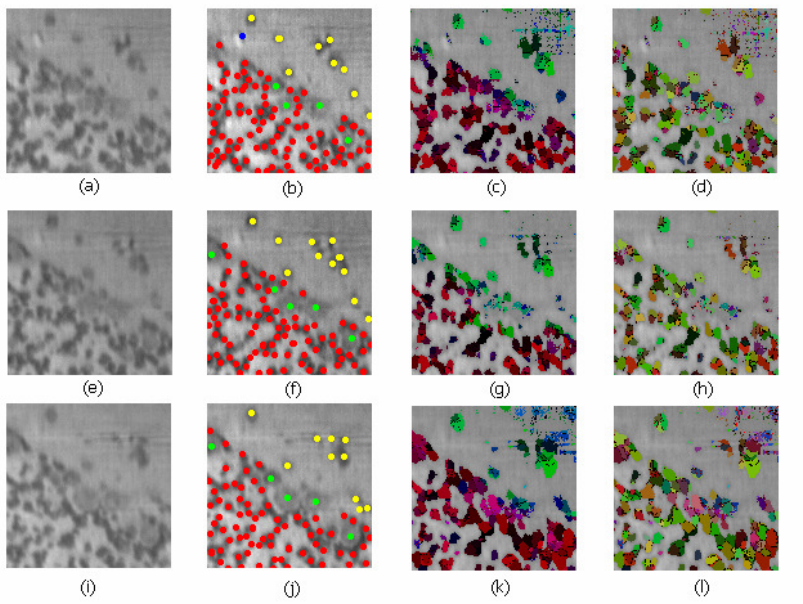


Fig. 58. (a), (e), (i) Original images (b), (f), (j) Cells identified by expert
(c), (g), (k) Segmentation results depicting cell classification (d), (h), (l) Segmentation results depicting individual cell areas.

- Low value of threshold might have been chosen during the template matching phase, resulting in many more local maxima forming and influencing cells to be classified as neurons / glial cells.

- Due to the high priority given to neurons in the 3D classification, this error is propagated across images and we get many more cells identified as neurons than those identified by a human observer.

8.4 Type B results

In fig. 58, we see the results attained for *type B*. The first column shows original images, the second the results of the human observer, the third shows the automated results highlighting the classification, i.e. granule cells(shades of maroon) and interneurons (shades of green and blue), while the final column randomly colors each cell to aid in the process of counting.

Purkinje cells and endothelial cells, which occur at the boundary of the densely populated granule cells and on blood vessels respectively, are *not* segmented by the automated algorithm and are consequently ignored in the counting process. The results are tabulated in Table 3 and Table 4.

TABLE 3
Comparison of manual and automated results for segmentation of granular cells

Img	Manual $num(\mathrm{gr_m})$	Automated $num(\mathrm{gr_a})$	1-1 Correspondence $num(\mathrm{gr_{corr}})$	Percentage identified $\frac{num(gr_{\mathrm{corr}})}{num(gr_{\mathrm{m}})}\times 100$	Percentage extra / missed by expert $\frac{num(gr_a)-num(gr_{corr})}{num(gr_a)}\times 100$
1	98	109	86	87.7 %	21.1 %
2	83	101	75	90.3 %	25.7 %
3	73	85	62	84.9 %	27.05 %
3D	112	134	103	91.9 %	23.1 %

Similar to *type A*, we have a very high percentage of cells segmented by the human observer that have also been spotted by the automated process, but many extra as well. The watershed algorithm works well and can be seen by the numbers of 1-1

correspondence in granule cells. Interneurons are spotted by the dense/sparse matrix discussed earlier.

TABLE 4

Comparison of manual and automated results for segmentation of interneurons

Img	Manual $num(\text{in}_m)$	Automated $num(\text{in}_a)$	1-1 Correspondence $num(\text{in}_{corr})$	Percentage identified $\frac{num(in_{corr})}{num(in_m)} \times 100$	Percentage extra / missed by expert $\frac{num(in_a) - num(in_{corr})}{num(in_a)} \times 100$
1	9	15	9	100 %	40 %
2	11	31	11	100 %	64.5 %
3	10	17	10	100 %	41.1 %
3D	13	33	13	100%	60.6 %

The problem of extra cells occurs again, which can either be incorrectly marked cells or cells that the observer failed to see. However, ending up with extra segments is a known drawback of the watershed algorithm, and so the results were not re-verified for confirmation. These extra segments can be eliminated by some changes in the post-processing method and some tweaking of the threshold data used to classify noise.

CHAPTER IX
CONCLUSION AND FUTURE WORK

9.1 Conclusion

A software application was described that could serve as an alternative to the manual approach by neuroscientists to segment, count and visualize various types of cells present in the Nissl stained images obtained from the KESM. While most of the noise was eliminated from the images and most of the cell bodies were successfully identified, the classification process still had some false positives when identifying neurons and granule cells. The watershed algorithm is one of the only ways to approach the problem of granule cells due to their size and density, and this was successfully implemented to separate out these cells.

There is still room for better classification results, and we discuss this in the 'future work' section.

9.2 Future work

9.2.1 Automatic identification of *type A* and *type B*

While using an average intensity threshold was indeed considered to decide if the image being analyzed was of *type A* or *type B*, this was not implemented since it was likely to be prone to error. More complex analysis to decide on the type is needed for this classification.

9.2.2 Using faster techniques for matching

The 3D template matching that is currently implemented uses a 3D kernel that steps through the volume one at a time. This is computationally very costly, and can be improved by using faster techniques like *Fast Fourier Transforms* (FFT).

9.2.3 Smoothing on marching cubes visualization

The smoothing operation could serve to make the 3D polygonal representation using marching cubes more appealing visually.

9.2.4 More accurate classification techniques

The problem of false positives could be further alleviated using better post-processing techniques for this classification.

9.2.5 Better comparison method for validation

As described in section 8.3, there were certain issues with the set analyzed by the experts and those analyzed by the automated algorithms. A more consistent comparison method could be used in future.

REFERENCES

[1] Lars K. Nielsen, Gordon K. Smythl, and Paul F. Greenfield, "Hemacytometer Cell Count Distributions: Implications of Non-Poisson Behavior," *Biotechnol. Prog*., vol.7, pp. 560-563, 1991.

[2] David R Caprette, "Counting cells using a Microscope Counting Chamber," http://www.ruf.rice.edu/~bioslabs/methods/microscopy/cellcounting.html, Mar 2007.

[3] E. Pauli, M. Hildebrandt, J. Romstöck, H. Stefan, and I. Blümcke, "Deficient memory acquisition in temporal lobe epilepsy is predicted by hippocampal granule cell loss," *Neurology*, vol. 67, pp. 1383 – 1389, Oct 2006.

[4] Barry L. Jacobs, Henriette van Praag, Fred H. Gage, "Depression and the birth and death of brain cells," http://www.biopsychiatry.com/newbraincell/, July 2000.

[5] Francisco López-Muñoz, Jesús Boya and Cecilio Alamo, "Neuron theory, the cornerstone of neuroscience on the centenary of the Nobel Prize award to Santiago Ramón y Cajal," *Brain Research Bulletin,* vol. 70, no.s 4-6, pp. 391-405, 16 October 2006.

[6] Harald Fodstad, "The Neuron Theory," *Proceedings of the 13th Meeting of the World Society for Stereotactic and Functional Neurosurgery,* vol.77, pp. 20-24, September 2001.

[7] Harvey Lodish, Arnold Berk, Lawrence S. Zipursky, Paul Matsudaira, David Baltimore and James Darnell, http://www.ncbi.nlm.nih.gov/books/bv.fcgi?rid=mcb.section.6108 *Molecular Cell Biology,* 2000.

[8] Alain Beaudet and Alain Rambourg, "Tri-dimensional Structure of Nissl Bodies: A Stereoscopic Study in Ventral Horn Cells of Rat Spinal Cord," *The Anatomical Record,* vol.207, pp. 539-546.

[9] "Staining," http://cancerweb.ncl.ac.uk/cgi-bin/omd?staining, May 2007.

[10] Strausfeld, N. J., I. Vilinsky, and L. C. Hansen , "Golgi Impregnations, Introduction," http://web.neurobio.arizona.edu/Flybrain/html/atlas/golgi/index.html, Apr 2007.

[11] Daniel S. Barth, "Neuroscience Methods, Background," http://psych.colorado.edu/~dbarth/PDFs/4052/4052%20Manual%20Chapters/Histology%20I.pdf.

[12] Bruce H. McCormick, "Development of the Brain Tissue Scanner," *Brain Networks Lab Technical Report,* March 2002.

[13] Ying-Lun Fok, Joseph C. K. Chan, and Roland T. Chin, "Automated Analysis of Nerve-Cell Images Using Active Contour Models," *IEEE Transactions on Medical Imaging*, vol. 15, June 1996.

[14] Elmoataz, A.; Revenu, M.; Porquet, C, "Segmentation and Classification of various types of cells in Cytological Images," *International Conference on Image Processing and its Applications, 1992*, pp.385-388, April 1992.

[15] Mussio, P.; Pietrogrande, M.; Bottoni, P.; Dell'Oca, M.; Arosio, E.; Sartirana, E.; Finanzon, M.R.; Dioguardi, N., "Automatic cell count in digital images of liver tissue sections," *Proceedings of the Fourth Annual IEEE Symposium*, *Computer-Based Medical Systems, 1991*, pp.153-160, May 1991.

[16] D. Comaniciu and P. Meer, "Robust Analysis of Feature Spaces: Color Image Segmentation," *IEEE Conf. on Comp. Vis. and Pattern Recognition*, pp. 750-755, 1997.

[17] Constantinos G. Loukas, George D. Wilson, Borivoj Vojnovic, Alf Linney, "An Image Analysis-based Approach for Automated Counting of Cancer Cell Nuclei in Tissue Sections," *Cytometry*, vol. 55A, pp.30-42, September 2003.

[18] Amini, A.A.; Weymouth, T.E.; Jain, R.C, "Using Dynamic Programming for Solving Variational Problems in Vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol.12, pp.855-867, Sep 1990.

[19] Stephen J. Lockett, Damir Sudar, Curtis T. Thompson, Dan Pinkel, Joe W. Gray, "Efficient, Interactive, and Three-dimensional Segmentation of Cell Nuclei in Thick Tissue Sections," *Cytometry*, vol.31, no. 4, pp.275-286, Dec 1998.

[20] Yan Kang, Klaus Engelke and Willi A. Kalender**,** "Interactive 3D Editing Tools for Image Segmentation," *Medical Image Analysis*, vol.8, pp. 35-46, Mar 2004.

[21] Bill Green, "Histogram, Thresholding and Image Centroid Tutorial," http://www.pages.drexel.edu/~weg22/hist_thresh_cent.html, Apr 2002.

[22] Anil K. Jain, Robert P.W. Duin, and Jianchang Mao, "Statistical Pattern Recognition: A Review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol.22, January 2000.

[23] Sergios Theodoridis and Konstantinos Koutroumbas, *Pattern Recognition, Second Edition,* Academic Press*,* February 2003.

[24] Gang Lin, Umesh Adiga, Kathy Olson, John Guzowski, Carol Barnes and Badrinath Roysam, "A Hybrid 3D Watershed Algorithm Incorporating Gradient Cues and Object Models for Automatic Segmentation of Nuclei in Confocal Image Stacks," *Cytometry* , vol. 56A, pp. 23-36, 2003.

[25] Perlovsky, L.I., "Conundrum of combinatorial complexity," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, pp. 666-670, June 1998.

[26] Zheng Lin, Jesse Jin, Hugues Talbot, "Unseeded region growing for 3D image segmentation," *Selected Papers from Pan-Sydney Workshop on Visual Information Processing*, 2002.

[27] Vincent L. and Soille P., "Watersheds in digital spaces: an efficient algorithm based on immersion simulations," *IEEE Transactions on Pattern Analysis and Machine Intelligence* vol.13, pp. 583 – 598, June 1991.

[28] S. Beucher, "The watershed transformation applied to image segmentation," *Conference on Signal and Image Processing in Microscopy and Microanalysis*, pp. 299 - 314, September 1991.

[29] J. B. T. M. Roerdink and A. Meijster, "The Watershed Transform: Definitions, Algorithms and Parallelization Techniques," *Mathematical Morphology,* pp.187-228, 1999.

[30] Paul R.Hill, C Nishan Canagarajah, David R Bull, "Image Segmentation Using a Texture Gradient Based Watershed Transform," *IEEE Transactions of Image Processing* , vol. 12, December 2003.

[31] Gang Li, Tianming Liu, Jingxin Nie, Lei Guo, Wong, S.T.C., "Segmentation of Touching cells using gradient flow tracking," *4th IEEE International Symposium on Biomedical Imaging: From Nano to Macro,* vol. 12, pp.77-80, April 2007.

[32] Robert Fisher, Simon Perkins, Ashley Walker and Erik Wolfart, http://homepages.inf.ed.ac.uk/rbf/HIPR2/stretch.htm, "Contrast Stretching," July 2007.

[33] Bob Fisher, Simon Perkins, Ashley Walker and Erik Wolfart, http://www.cee.hw.ac.uk/hipr/html/median.html, "Median Filtering," July 2007.

[34] "Edges: Gradient edge detection," http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MARBLE/low/edges/gradient.htm, July 2007.

[35] Donald G. Bailey, "An Efficient Euclidean Distance Transform," *Combinatorial Image Analysis,* vol. 3322, pp. 394-408, 2004.

[36] R. Klette, "Algorithms for Picture Analysis," Lecture 08, available at www.citr.auckland.ac.nz/~rklette/Books/MK2004/Algorithms.htm, February 2005.

[37] Robert Fisher, Simon Perkins, Ashley Walker and Erik Wolfart, "Connected components labeling," http://homepages.inf.ed.ac.uk/rbf/HIPR2/label.htm, July 2007.

[38] Rudolf K. Bock, “Template Matching,” http://rkb.home.cern.ch/rkb/AN16pp/node283.html#SECTION0002830000000000000000, April 1998.

[39] S. Beucher and C.Lantuéjoul, “Use of Watershed in Contour Detection,” *Proc. Int. Workshop Image Processing, Real-Time Edge and Motion Detection/Estimation*, pp. 12-21, September 1979.

[40] Lee, S.C. Yimmg Wang Lee, E.T., “Compactness measure of digital shapes,” *Region 5 Conference: Annual Technical and Leadership Workshop*, pp. 103-105, April 2004.

[41] D.P. Huijsmans, “Microscopy, Modeling Visualization Mathematical Morphology,” http://www.liacs.nl/~fverbeek/courses/ia2007/IA2007-lecture10.pdf, April 2007.

[42] William E. Lorensen, Harvey E. Clin, “Marching Cubes: A High Resolution 3D Surface Construction Algorithm,” *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 163 – 169, 1987.

APPENDIX

Pseudo-code for various algorithms referenced in the thesis are provided here

1. Scan

Initialization:

/* upperList and currentList hold Lines, while patchList holds Patches */

$upperList \leftarrow NULL$

$currentList \leftarrow NULL$

$patchList \leftarrow NULL$

Scanning:

For $y \leftarrow 0$ to s_y

$upperList \leftarrow currentList$

$currentList \leftarrow NULL$

For $x \leftarrow 0$ to s_x

if $I(x, y) \in Fg$

Initialize l

$x_s(l) \leftarrow x$

while $I(x, y) \in Fg$ and $x \leq s_x$

$x \leftarrow x + 1$

$x_e(l) \leftarrow x$

$y(l) \leftarrow y$

pushBack (*currentList*, l)

if $y > 0$ and isEmpty(*currentList*) = *false* and isEmpty(*upperList*) = false

augmentResults (*upperList*, *currentList*)

2. Augment results(*upperList*, *currentList*)

For each line *ul* in *upperList* and *cl* in *currentList*{

if $x_s(ul) > x_e(cl)$

Advance *cl* to the next line in *currentList*

else if $x_e(ul) < x_s(cl)$

Advance *ul* to the next line in *upperList*

else

/* Case 1 - *cl* overlaps with multiple lines from *upperList* */

while (isOverlap(*cl*,*ul*) = *true* and isOverlap(*cl*, next(*ul*)) = *true*)

Combine $ptrPatch(ul)$ with $ptrPatch(next(ul))$

Advance *ul* to the next line in *upperList*

/* Case 2 – *cl* overlaps with a single line from *upperList* */

if $ptrPatch(ul) = NULL$

initialize new Patch *P*

$pushBack(L(P), ul)$

$pushBack(L(P), cl)$

$ptrPatch(ul) \leftarrow ptr(P)$

$ptrPatch(cl) \leftarrow ptr(P)$

fill all attributes of P based on values of *ul* and *cl*

$pushBack(patchList, P)$

else

Initialize patch $P \leftarrow ptrPatch(ul)$

$pushBack(L(P), cl)$

$ptrPatch(cl) \leftarrow P$

Adjust the attributes of *P* based on the newly added *cl*

Advance *cl* to the next line in *currentList*

3. Sort by storage

for $i \leftarrow 0$ to s_z

for $j \leftarrow 0$ to s_y

for $k \leftarrow 0$ to s_x

Initialize index *ind*(*x*,*y*,*z*) to (k,j,i)

if $VoxelsAtHeight[I(k,j,i)]=NULL$

$VoxelsAtHeight[I(k,j,i)] \leftarrow ind$

else

Insert *ind* at the end of the List pointed to by
VoxelsAtHeight[*I*(*k*,*j*,*i*)]

4. Is marked (voxel v(x,y,z))

if *Labels*[*z*][*y*][*x*]>0 or *Labels*[*z*][*y*][*x*]=WSHED
return true
return false

5. Fast watersheds

Sort all the voxels in the Image Volume *I* using the *sortByStorage*( ) algorithm

for $h \leftarrow h_{\min}$ to $h_{\max}$

for each voxel *v* in the list *VoxelsAtHeight*[h]
set *Labels(v)* $\leftarrow$ MASK
if $\exists v' \in N(v)$ such that $isMarked(v') = true$
set *Distance(v)* $\leftarrow 1$
pushBack(*Q*, *ind*)

set *curDist* $\leftarrow 1$
pushBack(*dummyVoxel*)
while(*true*)
Initialize *v* $\leftarrow$ *begin*(*Q*)
if *v* = *dummyVoxel*
if *isEmpty*(*Q*) = *true*
Break
else
pushBack(dummyVoxel)
curDist $\leftarrow$ *curDist* + 1
ind $\leftarrow$ begin(Q)

For every voxel v' in $N(v)$

```
                    if Distance(v')<curDist and isMarked(v')
                        if v' is named
                            if Labels(v) = MASK or WSHED
                                Labels(v) = Labels(v')
                            else if Labels(v) ≠ Labels(v')
                                Labels(v) = WSHED
                        else if Labels(v) = MASK
                            Labels(v) = WSHED
                    else if Labels(v')=MASK and Distance(v')=0
                        set Distance(v') to curDist + 1
                        pushBack(Q, v')

    //New minima
    for every voxel v in VoxelsAtHeight[h]
        set Distance(v) ←0
        if Labels(v)=MASK
            Assign new label l at voxel v
            pushBack(v)
            while isEmpty(Q) = false
                Initialize v' ←begin(Q)
                for every neighbouring voxel v" in N(v')
                    if Labels(v")=MASK
                        pushBack(v")
                        set Labels(v") ←l
```

6. Hole fill(patch P)

Initialize a Boolean matrix *Img* of dimensions $rows \times cols$, where

$$cols = \left(x(i_{\max}(P)) - x(i_{\min}(P)) + 1 \right)$$
$$rows = \left(y(i_{\max}(P)) - y(i_{\min}(P)) + 1 \right)$$

Initialize every index in Img to *false*

for each $l \in L(P)$

for each $x \in [x_s(l), x_e(l)]$

set $\text{Img}[y(l) - y(i_{\min}(P))][x(l) - x(i_{\min}(P))] = true$

//Complement the *Img* matrix

for each (x, y) where $0 \le x < cols, 0 \le y < rows$

$\text{Img}[y][x] = \overline{\text{Img}[y][x]}$

//Run Connected-Component labeling on the resulting *Img*, clearing the pixels of all //patches that do not touch any boundary

newPatchList $\leftarrow$ *returnPatchList*(Img[][])

for each $nP \in newPatchList$

if $x(i_{\min}(nP)) \neq x(i_{\min}(P))$ *and* $x(i_{\max}(nP)) \neq x(i_{\max}(P))$

and $y(i_{\min}(nP)) \neq y(i_{\min}(P))$ *and* $y(i_{\max}(nP)) \neq y(i_{\max}(P))$

for each $l \in L(nP)$

for each $x \in [x_s(l), x_e(l)]$

set $\text{Img}[y(l) + y(i_{\min}(P))][x(l) + x(i_{\min}(P))] = false$

//Complement Image *Img* again

for each (x, y) where $0 \le x < cols, 0 \le y < rows$

$\text{Img}[y][x] = \overline{\text{Img}[y][x]}$

//Resulting patch *P* is one without any holes (fig. 22 e)

7. Patch refinement (*PatchList*)

/* This algorithm inputs a list of Patches in the current image and corrects by means of modification or replacement, all Patches that do not satisfy certain score requirements */

for each patch $P \in PatchList$

if $HWR(P) < HWR_{\min}$ or $HWR(P) > HWR_{\max}$ or $n(P) < n_{\min}$

Classify *P* as noise. Remove *P* from *PatchList*

if $Cp(P) < Cp_{\min}$ or $Cr(P) < Cr_{\min}$

Analyze Convolved Image I_c and get local maxima near *P*

if $\exists (x, y, z) \in (i_{\min}(P), i_{\max}(P))$ such that $I_c(x, y, z) \ge Threshold$

Modify or Replace patch *P* so its attributes are within acceptable limits

8. Bounding-box overlap (*patch* P_1, *patch* P_2)

if $y_s(i_{min}(P_1)) > y_e(i_{max}(P_2))$ or $y_s(i_{min}(P_2)) > y_e(i_{max}(P_1))$

    Return *false*

else if $x_s(i_{min}(P_1)) > x_e(i_{max}(P_2))$ or $x_s(i_{min}(P_2)) > x_e(i_{max}(P_1))$

    Return *false*

else

    Return *true*

9. Patch overlap (*patch* P_1, *patch* P_2)

Initialize Boolean matrix M of size

$$\left(x(i_{max}(P_1)) - x(i_{min}(P_1)) + 1\right) \times \left(y(i_{max}(P_1)) - y(i_{min}(P_1)) + 1\right)$$

for each line $l \in L(P_1)$

    for $i \leftarrow x_s(l)$ *to* $x_e(l)$

        set $M[y][i] \leftarrow true$

for each line $l \in L(P_2)$

    for $i \leftarrow x_s(l)$ *to* $x_e(l)$

        if $M[y][i] = true$

            Return *true*

Return *false*

10. Group in 3D (*list*[] Patches)

```
for i ← 0 to s_z - 1
    Initialize patch curPatch ← begin(Patches[i])
    Initialize patch upPatch ← begin(Patches[i+1])

    while y(i_min(upPatch)) < y(i_max(curPatch)) and upPatch<>end(Patches[i+1])
        if isBoundingBoxOverlap( curPatch, upPatch ) = false
            Advance upPatch to next patch in Patches[i+1]
        else
            if isPatchOverlap(curPatch, upPatch) = false
                Advance upPatch to next patch in Patches[i+1]
            else
                /* Patch is overlapping. Add to the Blob list */
                if ptrBlob(curPatch)<>NULL
                    ptrBlob(upPatch) ← ptrBlob(curPatch)
```

pushBack (*L*(*ptrBlob*(*upPatch*)), *upPatch*)
Adjust attributes of *ptrBlob*(*upPatch*) with new data
else
Create a new Blob *B*
pushBack (*L*(*B*), *curPatch*)
pushBack (*L*(*B*), *upPatch*)
ptrBlob(*curPatch*) ← *B*
ptrBlob(*upPatch*) ← *B*
Fill attributes of *B*
Add *B* to *BlobList*
Advance *upPatch* to next patch in *Patches*[*i*+1]

Advance *upPatch* to next patch in *Patches*[*i*+1]

11. Marching cubes()

for each image I_k in the data set, where $k \in (0, s_z)$

Read image I_k and next image I_{k+1}

for each position (x, y, z) in I_k

Read in the values of the four neighboring voxels of (x, y, z) in I_k:

$$v_3 = (x+1, y+1, z)$$
$$v_2 = (x, y+1, z)$$
$$v_1 = (x+1, y, z)$$
$$v_0 = (x, y, z)$$

Read the values of the corresponding voxels in I_{k+1}:

$$v_7 = (x+1, y+1, z+1)$$
$$v_6 = (x, y+1, z+1)$$
$$v_5 = (x+1, y, z+1)$$
$$v_4 = (x, y, z+1)$$

Calculate the table index *TI* as described using the values of the corner voxels of the cube as values for $(i_7, i_6, i_5, i_4, i_3, i_2, i_1, i_0)$

Use the calculated *TI* to extract information about edge intersections and corresponding triangles from the table

Calculate the normal for the triangles so that lighting calculations are accurate

VITA

Aswin Cletus D'Souza received his Bachelor of Engineering (B.E.) in Computer Science & Engineering from Manipal Institute of Technology in 2004. He entered the Master of Science (MS) program in the department of Computer Science at Texas A&M University in August 2005 and will be receiving his degree in December 2007. He is planning to start his professional career in the industry upon graduation. He can be contacted either at aswindsouza@yahoo.com or his current residence: 6014 Winsome Lane, #209, Houston, TX 77057.