

**SOLVENT ANNEALING STUDIES OF
BLOCK COPOLYMER THIN FILMS
AND THEIR POTENTIAL USE AS SURFACES
FOR BIOLOGICAL APPLICATIONS**

by

Ronald M. Lewis, III

A thesis submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Honors Degree in Chemical Engineering with Distinction

Spring 2013

© 2013 Ronald M. Lewis, III
All Rights Reserved

**SOLVENT ANNEALING STUDIES OF
BLOCK COPOLYMER THIN FILMS
AND THEIR POTENTIAL USE AS SURFACES
FOR BIOLOGICAL APPLICATIONS**

by

Ronald M. Lewis, III

Approved:

Thomas H. Epps, III, Ph.D.
Professor in charge of thesis on behalf of the Advisory Committee

Approved:

April Kloxin, Ph.D.
Committee member from the Department of Chemical and Biomolecular
Engineering

Approved:

Susan Groh, Ph.D.
Committee member from the Board of Senior Thesis Readers

Approved:

Michael Arnold, Ph.D.
Director, University Honors Program

ACKNOWLEDGMENTS

I would like to acknowledge first my thesis and research advisor, Dr. Epps. His insight, sincere comments, and guidance throughout my undergraduate career have been one of the most positive aspects to my experience at the University of Delaware. I feel that my life has benefitted greatly from my interaction with him, and that he has helped me become a better researcher and scientist. I am very grateful for the opportunity he provided me freshman year, and I know that moving forward this experience will enhance my graduate and future careers.

Next, I would like to thank my first graduate student mentor, Dr. Julie Albert. Although she graduated during my time in the group, her initial drive, encouragement, and mentorship was crucial to my understanding of block copolymers and research in general. She was always there if I had any questions or concerns with a welcoming demeanor, but was unafraid to challenge me to find a solution on my own.

I would like to acknowledge next the rest of Epps Group, especially Jonathan Seppala, Sarah Mastroianni, Ming Luo, Elizabeth Kelley, Jillian Emerson, Wei-Fan Kuan, and Matt Green for insightful and entertaining conversations. Additionally, Sarah's, Liz's, and Wei-Fan's help with techniques such as GPC, ^1H NMR, SAXS, and TEM was essential to my research, and I am very thankful for their assistance in this regard.

Over the past three years, I have come to know the members of Epps Group fairly well. Sarah has been essential to my research in a variety of ways, especially

within the past year. Her endless stories, vibrant imagination, and scientific know-how were both enjoyable and helpful in almost every way.

Although he was not in the group for very long, Jon and I became friends rather quickly and our similar way of thinking through problems meshed well. He taught me a great deal about how to be innovative in the lab, but was also always ready to talk about life and have a good laugh (even if it was about science). I am happy to have been his fan-waving, Kim wipe-handing undergrad.

Liz Kelley and I did not interact very much scientifically, primarily because we were working on totally different projects. However, her editorial advice was unparalleled, and I will certainly miss the late-afternoon coffee excursions that I occasionally attended.

As members of the thin films sub-group, I have worked rather closely with both Jill and Ming over the past two years. From reminders to clean up in lab to comments on a paper to laugh-filled conversations, they have always been there to help me with anything I needed.

Finally, I would like to thank my family and friends. Their constant support and (sometimes) helpful conversations carried me through this project, and I am thankful for that.

TABLE OF CONTENTS

LIST OF FIGURES	vii
ABSTRACT	ix
1 INTRODUCTION	1
1.1 Background.....	1
2 OVERVIEW OF TECHNIQUES.....	5
2.1 Atomic Force Microscopy	5
2.2 Flow Coating	7
3 SOLVENT ANNEALING STUDIES	9
3.1 Morphological Control using a Microfluidic Device	9
3.1.1 Microfluidic Device Fabrication	9
3.1.2 Annealing Studies.....	11
3.2 Morphological Control by Varying Solvent Removal Rate	13
3.2.1 Solvent Deswelling Rate	13
3.2.2 Through-film Morphology	15
3.3 Morphological Control through Raster Solvent Vapor Annealing (RSVA).....	17
3.3.1 Single-pass RSVA	17
3.3.2 Multi-pass and Cross-path RSVA	19
3.4 Quantitative Morphology Characterization from AFM Images	22
4 APPLICATIONS FOR STEM CELL GROWTH AND DIFFERENTIATION	24
4.1 Motivation	24
4.2 Results and Discussion	24

4.2.1	Morphology Determination	25
4.2.2	Surface Functionalization	26
4.2.3	Stem Cell Growth	28
5	CONCLUSIONS AND FUTURE WORK	30
5.1	Conclusions	30
5.2	Future Work	31
	REFERENCES	32
A	MORPHOLOGY DETERMINATION CODE	35
B	PROPERTIES OF SI POLYMER	62

LIST OF FIGURES

Figure 1.1-1 Nomenclature for block copolymer systems	1
Figure 1.1-2 Theoretical phase diagram for a diblock copolymer	2
Figure 2.2-1 Flow coating schematic	7
Figure 3.1.1-1 Microfluidic device used for annealing studies.	10
Figure 3.1.1-2 Microfluidic device used for solvent collection studies.	11
Figure 3.1.2-1 AFM images of an SIS film annealed with the device..	12
Figure 3.2.1-1 Thickness profiles for deswelling dS-I-dS films.	14
Figure 3.2.2-1 Deswell profiles and images with stopping thickness varied and rate held constant.	16
Figure 3.3.1-1 Experimental setup for raster solvent vapor annealing.....	17
Figure 3.3.1-2 Single-pass RSVA images with a larger nozzle	18
Figure 3.3.1-3 Single-pass RSVA images with a smaller nozzle.....	19
Figure 3.3.2-1 Multi-pass RSVA images	20
Figure 3.3.2-2 Cross-path RSVA images	21
Figure 3.4-1 AFM image analysis example.	22
Figure 3.4-2 Re-colored AFM image from analysis.	23
Figure 4.2.1-1 Bell jar annealed samples.	25
Figure 4.2.2-1 Pre- and post-reaction AFM images.	26
Figure 4.2.2-2 Pre- and post-reaction ¹ H NMR spectra.	27
Figure B-1 GPC trace of SI polymer.....	62

Figure B-2 Bulk morphology of SI polymer..	63
--	----

ABSTRACT

Solvent annealing represents a versatile technique for self-assembly of block copolymer thin films. The following report highlights the practical functionality of this technique by describing a series of studies undertaken to expand the understanding of this procedure. Firstly, a microfluidic device was utilized to demonstrate that, with the proper selection of solvents, a gradient in morphology could be established across a single film. Secondly, the rate of solvent removal from a film was shown to have a direct impact on the topography of the film, while maintaining a different through-film structure. Thirdly, raster solvent vapor annealing (RSVA) techniques were employed to selectively anneal a very small portion of a film. Multiple passes or cross patterns with this technique were shown to alter the morphology as well. Fourthly, a program was developed in Java to quantitatively interpret and analyze AFM images for the ratios of different morphologies. Finally, the solvent annealing procedure was applied to a biological application by initially conducting bell jar experiments to determine the morphologies of the polymer film. The film was subjected to experimental thiol-ene click chemistry reactions in order to functionalize the surface of the film. While no further experimentation occurred, a literature search was performed to evaluate future experimental parameters involving cell growth.

Chapter 1

INTRODUCTION

1.1 Background

Block copolymers consist of two or more covalently bonded homopolymer chains. The unfavorable interaction between these two chains promotes self-assembly at the nanoscale. Nanoscale self-assembly is an important facet to nanoscience studies, such as nanopatterning and micelle formation, and is why block copolymers are under study here. A block copolymer with two different types of monomers is referred to as an AB diblock copolymer. This nomenclature extrapolates to more complex systems as well, as shown in Figure 1.1-1 below.

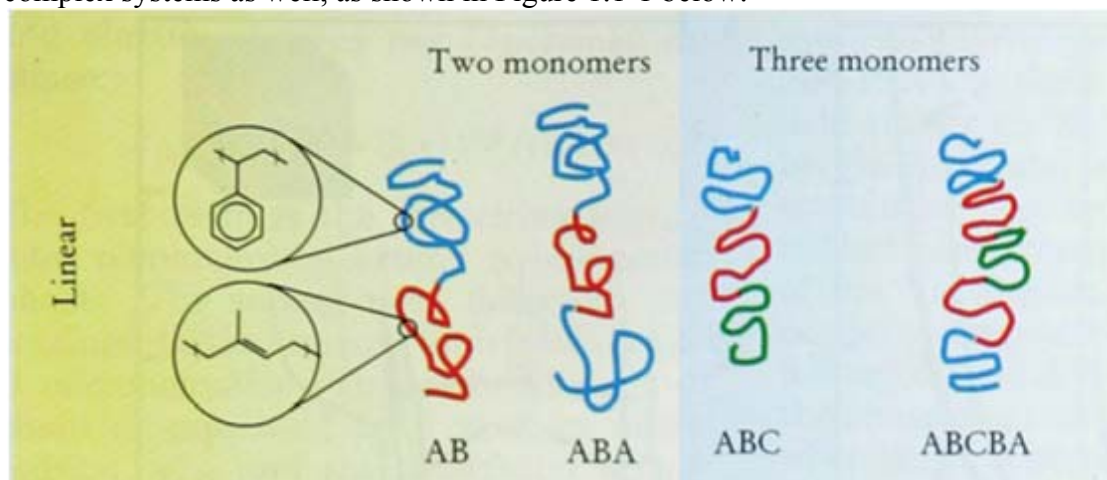


Figure 1.1-1 Nomenclature for block copolymer systems. The structures identified in the AB chain are styrene (above) and isoprene (below). Adapted from Bates, et al.¹

The interaction between these chains, characterized by the Flory-Huggins interaction parameter (χ) and the size of the chains (N) make up the segregation strength, χN . This segregation strength and the volume fraction of the polymer (f) determine the theoretical morphology of the block copolymer²⁻⁴. Figure 1.1-2 shows a phase diagram for an AB diblock copolymer, along with cartoons describing the different copolymer morphologies.

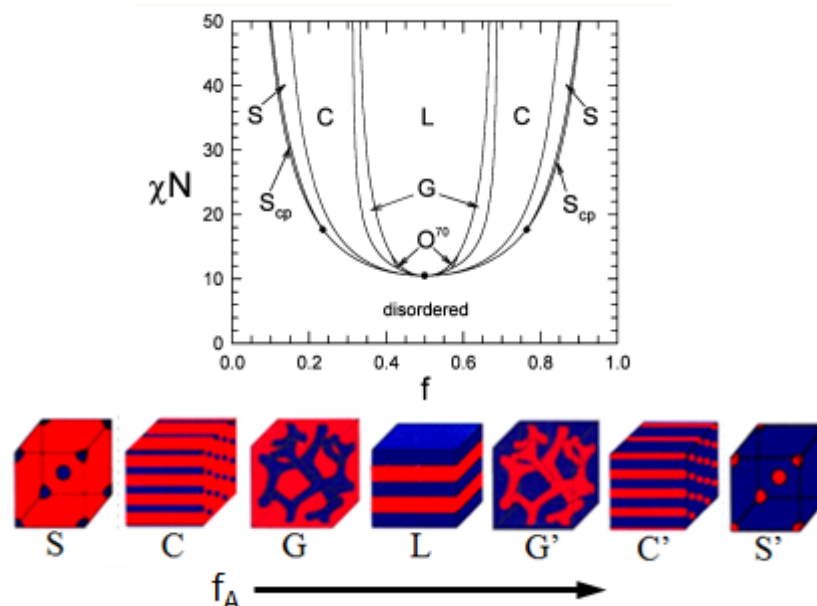


Figure 1.1-2 Theoretical phase diagram for a diblock copolymer, with the corresponding morphologies shown below. Adapted from Bates, et al.¹ and Matsen, et al.⁵

In the bulk scenario, where the characteristic dimension of the polymer system is at least six times the domain spacing of the polymer (L_o)⁶, the equilibrium morphologies of the polymer match the theoretical phase diagram moderately well⁷. However, in the thin film scenario there are additional factors contributing to the morphology, including confinement between the air and the surface and the

commensurability of the thickness with the domain spacing. These factors allow for the introduction of morphology orientation, as described by the cartoons in Figure 1.1-2. Different morphologies, orientations, and structures in block copolymers allow for their utilization in many applications, such as nanotemplating⁸⁻¹¹, organic photovoltaics^{12,13}, and drug delivery¹⁴⁻¹⁶.

Thin films are utilized in these studies for two primary reasons: a.) there are numerous parameters that can be manipulated to control the morphology of the polymer, and b.) the versatility and ease of thin film processing allows for high-throughput methods with little required material. It is important to thoroughly understand the techniques and processes used to reproducibly generate different topographies. In general, these methods are well-understood for specific polymers^{17,18}, however every polymer is different, and therefore a firm understanding of morphology is the initial focus of this study. Various methods of structural control will be discussed in the following chapter.

In general, there are two primary methods for directing the self-assembly of block copolymers: thermal annealing and solvent annealing. Solvent annealing was chosen for these studies for several reasons. First, the large inventory of solvents to choose from provides for the desired matching of solvent-block combinations. These interactions are governed primarily by the chemistry of both the polymer and the solvent. In general, compounds with similar functional groups or structure tend to interact more favorably than otherwise, and solvents can be chosen accordingly. Second, the thermal properties of the polymers utilized in these studies are not conducive for thermal annealing. In thermal annealing, it is imperative to remain at a temperature below that of the glass transition temperature of the polymer sample. If

the glass transition is overcome, the polymer will assume a hard, brittle state in which the chains cannot migrate. Solvent annealing avoids this issue by maintaining a constant temperature, an important condition in this study due to the polymer's relatively low glass transition temperature. Third, solvent in the film mitigates the confinement effects normally observed in thin film scenarios. Solvent swelling in the film increases the effective thickness of the film, creating a situation that resembles a bulk polymer scenario, where confinement and surface effects are negligible. Finally, the volume fractions, block interactions, and domain spacing of the polymer are affected, which ultimately influences the film's commensurability conditions. In solvent annealing, solvents can be chosen such that the solvent is preferential for one block or the other. In this case, when a small amount of solvent swells the film, it will preferentially enter the more favorable block, thus changing the effective volume fraction of that block and the domain spacing of the overall polymer.

Chapter 2

OVERVIEW OF TECHNIQUES

The studies described herein utilize two major techniques: atomic force microscopy and flow coating. The purpose of this chapter is to discuss both of these methods so as to provide the reader with a basic understanding of the topics.

2.1 Atomic Force Microscopy

Atomic Force Microscopy (AFM) is essentially the use of a nanoscale probing device to image or characterize the surface of a material. An AFM consists of a tip, a laser, a photodetector, and the associated software and equipment. The tip utilized in AFM has a cantilever on one end with a protruding point structure. This point structure is what interacts with the surface of the sample, and can take the form of many different shapes depending on the desired characterization technique. The cantilever is attached to the larger section of the tip by a piezoelectric material, which can convert the small mechanical movements of the cantilever into electrical signals, and vice versa. Cantilever motion is primarily monitored by the incurred movement of a laser reflection. The laser is calibrated to reflect off of the cantilever and hit the photodetector at a specific 'zero' point. Deviations from this point are used by the software to observe changes in cantilever location.

In general, there are three different modes of AFM: contact, non-contact, and intermittent or tapping mode. Technically, contact mode is the only mode that involves physical interaction of the tip with the sample. In contact mode, the tip is

pressed to the surface of the sample, which could be either solid or liquid, and the repulsive van der Waal forces between the tip and the sample are captured. This force is recorded in an AFM force curve, which is useful for determining material properties of the sample such as hardness or elasticity. Non-contact mode provides similar results to contact mode, however the technique is performed oppositely. In this case, the tip is held above the sample and attractive van der Waal forces pull the tip towards the sample. As with contact mode, these forces can be translated to physical properties by use of a force curve.

Intermittent or tapping mode is the mode utilized by this study, and will therefore be discussed in more detail. The piezoelectric material at the base of the cantilever is used to make the cantilever oscillate at a controlled frequency by sending an electrical signal to the material that translates the resonant frequency of the cantilever. As the cantilever oscillates, it contacts the sample at its lowest point and then swings off. This mode of AFM does not provide a force curve as a result, but rather an image of the surface of the sample due to its three dimensional movement. There are three types of images that result from tapping mode AFM: height, phase, and amplitude. Height images contain mostly information about the height and roughness of the sample surface. They are attained by monitoring changes in the surface location relative to a zero point, and have a scale bar of length. Phase images are slightly less intuitive. As the cantilever oscillates, the oscillation is disturbed (either repulsively or attractively), which causes an offset in the phase of the oscillation. This disturbance is highly affected by the properties of the sample, which is an important principle for block copolymer study. The phase change or offset is what is shown in an image, with a scale bar of degrees. Because this image is highly

dependent upon the sample, it is best for seeing the morphology of block copolymer surfaces. The third image type, amplitude, is basically an image that shows the feedback from the loop that controls the system. Although it is not necessary to get good results, ideally this image is blank, indicating that the loop had minimal feedback. Because of this, amplitude images are hardly ever used for characterization. This report contains only phase images, as these are the best type to see the morphology of the surface of the sample.

2.2 Flow Coating

Flow coating is one of a few rather simple procedures used to create thin films. Figure 2.2-1 below shows a schematic of flow coating.

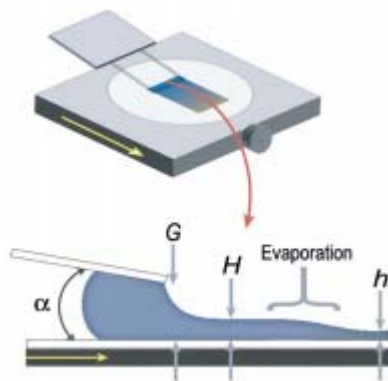


Figure 2.2-1 Flow coating cartoon. Adapted from Stafford, et al.¹⁹

As shown in the figure, a glass slide is held above the substrate at a specified height, G , and angle, α . Polymer solution is injected under the glass slide such that it wicks between the slide and the substrate. The substrate is pulled under the slide by a motorized stage at constant or varying speed to deposit a layer of solution on the substrate. The thickness of this layer, h , is determined by the stage speed, G , α , and

the weight percent of polymer in the solution. The solvent in the layer evaporates, leaving behind a polymer film of final thickness h . Flow coating parameters are generally well understood¹⁹, and the technique is fairly reproducible.

Chapter 3

SOLVENT ANNEALING STUDIES

The polymer used in each of the following studies, unless otherwise noted, is symmetric poly(styrene-*b*-isoprene-*b*-styrene) (SIS), a triblock copolymer with a number average molecular weight of 188 kg/mol, a polydispersity index (PDI) of 1.09, a bulk domain spacing of 29 nm, and block volume fractions of $f_{PS} = 0.134$, $f_{PI} = 0.732$, and $f_{PS} = 0.134$. I am an author on each of the four papers reviewed in this chapter, and the material discussed is a highlight of my (at least partial) contribution to each project.

3.1 Morphological Control using a Microfluidic Device

The content of this section is adapted from Albert, et al.¹⁸

3.1.1 Microfluidic Device Fabrication

The microfluidic device utilized in this study was made of 1H,1H-perfluoro-*n*-decyl acrylate (PFDA) and consisted of two entrance ports, a mixing tree, and seven chambers, as shown in Figure 3.1.1-1.

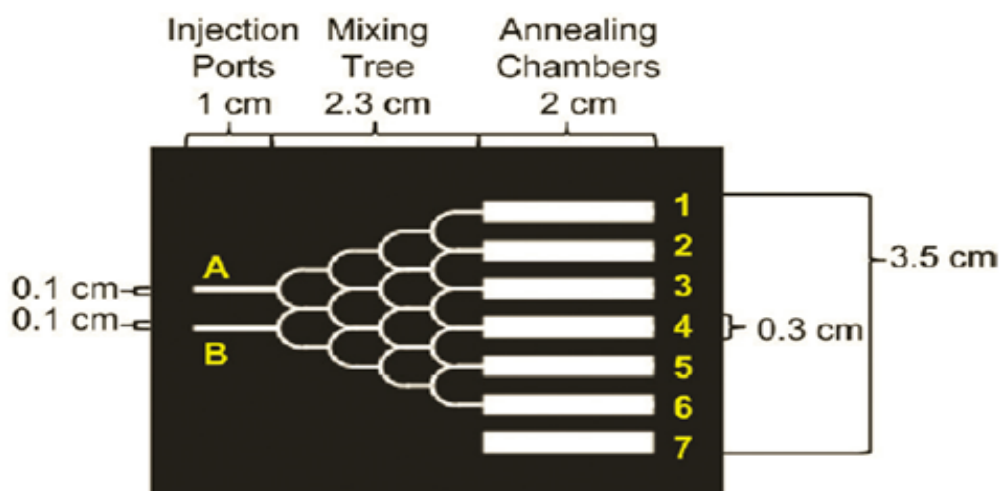


Figure 3.1.1-1 Microfluidic device used for annealing studies.

Note that the seventh chamber is not attached to the mixing tree, and therefore represents the control portion of the device. The device was fabricated by curing a PFDA and photoinitiator solution over a master template using ultraviolet radiation²⁰. PFDA was an attractive material because it is mechanically durable, it is impermeable to the solvents used in this study, and it is a soft material, which allows for an air-tight seal between individual chambers. The final device measured 75 mm x 39 mm x 6.35 mm. In addition to this device, another type of device was generated to analyze the solvent compositions in the chambers, as shown in Figure 3.1.1-2.

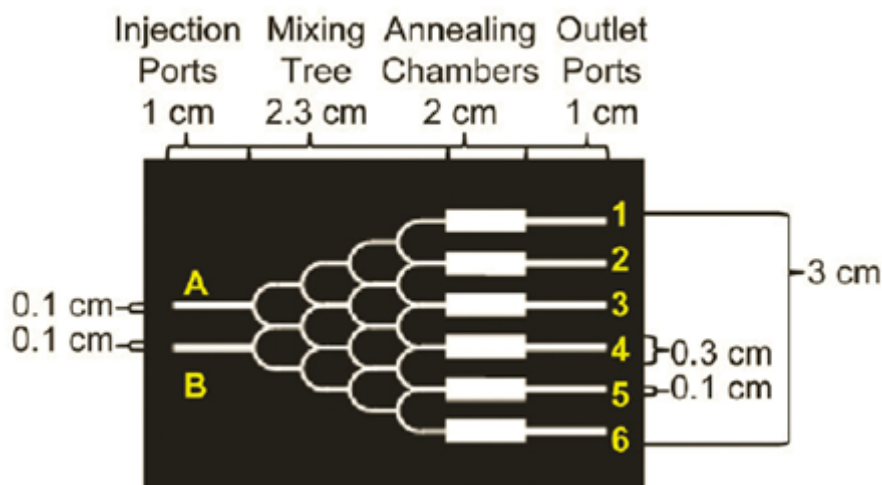


Figure 3.1.1-2 Microfluidic device used for solvent collection studies.

This device contained the same features as the previous one with the addition of exit ports to capture the solvent leaving the chambers. In terms of fabrication, it was produced in a similar manner to the annealing device, except with a different initial template.

3.1.2 Annealing Studies

Nitrogen flowing through solvent bubblers was used in this study to transport solvent to the device and polymer. Controlling the flow rate of nitrogen through the bubblers allowed for control over the flow rate of solvent and ultimately the amount of solvent in the film. Independent streams of nitrogen were flowed through two separate bubblers, one with *n*-hexane and another with tetrahydrofuran (THF), with flow rates of 10.6 ± 0.2 mL/min and 1.9 ± 0.1 mL/min, respectively. THF is a good solvent for both polymers, with a slight preference for PS, whereas *n*-hexane is preferential to PI and is a poor solvent for PS²¹. The streams were fed into the devices

using 21 gauge hypodermic needles. The devices were firmly held onto SIS thin films, which had a constant thickness of 100 ± 2 nm. Figure 3.1.2-1 shows the resulting morphologies from AFM in each of the chambers after a 10 h anneal.

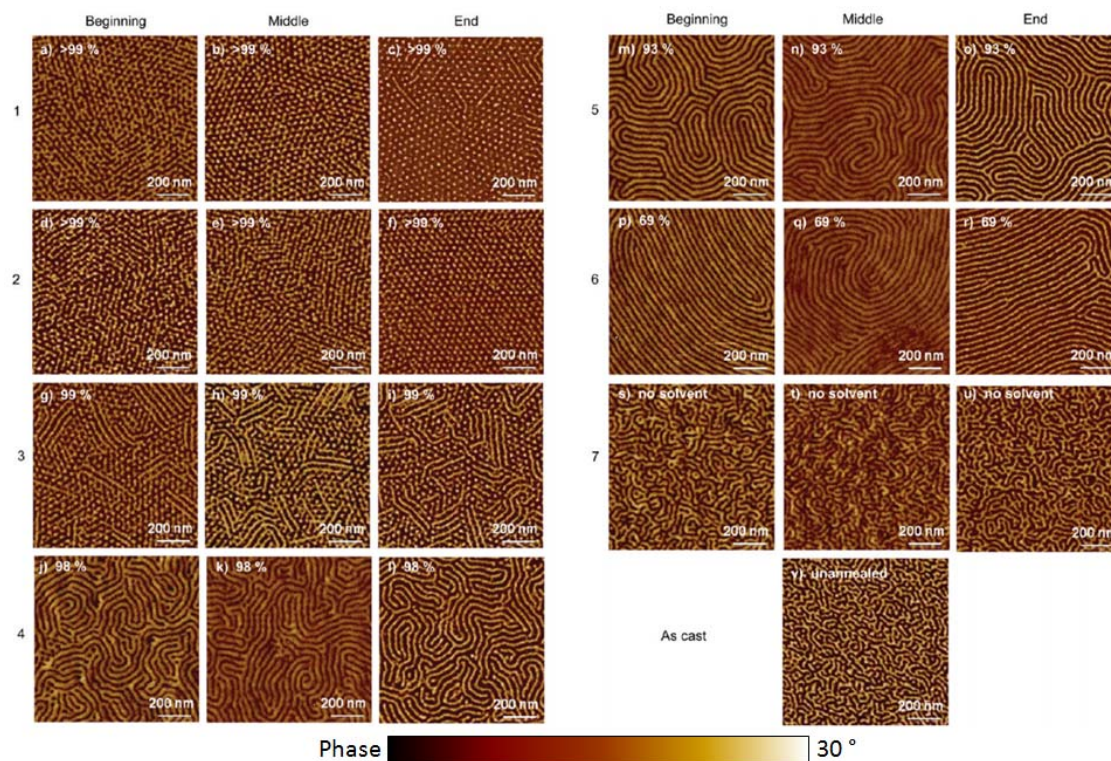


Figure 3.1.2-1 AFM images of an SIS film annealed with the device. The numbers on the left indicate the chamber number. The three columns of images correspond to the location within each chamber. Percentages in the top left of each image refer to the amount of *n*-hexane relative to THF in that chamber.

Note that the control chamber, chamber seven, resembles the film without any annealing, which confirms that the chambers were independent of each other. As the *n*-hexane content decreases in the chambers it is clear that the morphology responds by a re-orientation of the cylinders from being perpendicular to the surface to parallel

to the surface. In this way, a distinct gradient of morphology was created on the surface of the film through the use of the microfluidic device.

3.2 Morphological Control by Varying Solvent Removal Rate

The content of this chapter is adapted from Albert, et al.²² Note that in this study the polymer utilized is poly(deuterated styrene-*b*-isoprene-*b*-deuterated styrene) (dS-I-dS). It is similar to the previously described polymer, with a number average molecular weight of 160 kg/mol, a bulk domain spacing of 37 nm, and block volume fractions of $f_{PS} = 0.10$, $f_{PI} = 0.78$, and $f_{PS} = 0.12$.

3.2.1 Solvent Deswelling Rate

For this study, nitrogen was flowed through two solvent bubblers in series filled with chloroform. This saturated vapor was transported into a sealed chamber containing a dS-I-dS thin film, which had a constant thickness of 112 ± 2 nm. The film was initially swelled to 208 ± 3 nm and was allowed to equilibrate for 2 hours. After 2 hours, the solvent was removed at various rates (0.5, 0.1 and 0.05 nm/min) by incrementally decreasing the nitrogen flow rate through the bubblers. Thickness profiles are shown in Figure 3.2.1-1 a.).

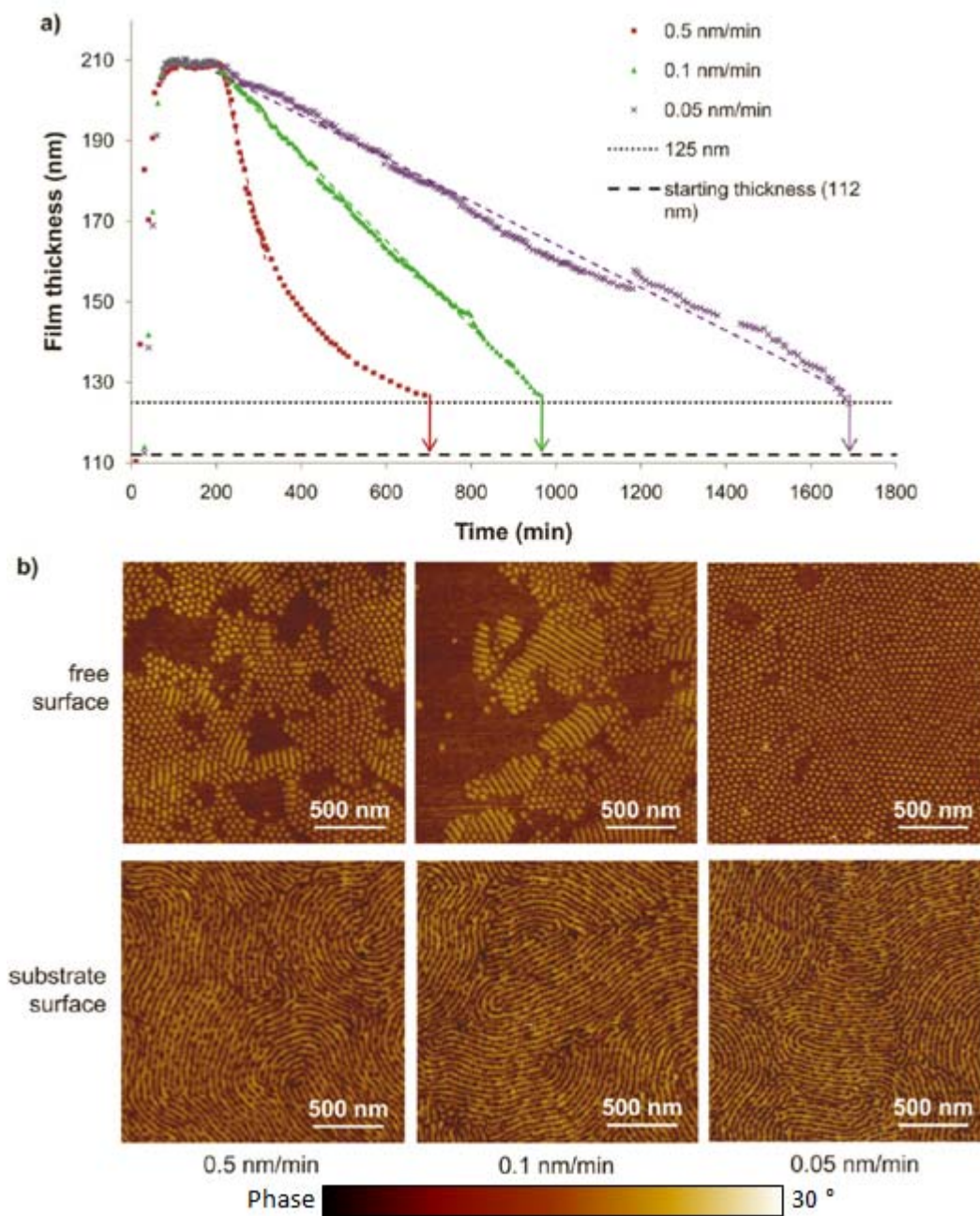


Figure 3.2.1-1 a.) Thickness profiles for deswelling dS-I-dS films, and b.) AFM images of the free and substrate surfaces of the film for the three corresponding deswell rates.

From the AFM images in Figure 3.2.1-1 b.), it is clear that the topography is a function of the solvent removal rate, with an increasing proportion of perpendicular cylinders as the solvent removal rate decreases. Additionally, the substrate side of the film remains fairly constant from sample to sample, which points to the presence of a hybrid morphology through the film.

3.2.2 Through-film Morphology

In the case of the 0.1 nm/min removal rate of solvent, further work was performed to determine the structural transition observed in the previous section. To do this, films were similarly annealed and then subjected to subsequent ultraviolet ozone etches to partially remove sections of film. Figure 3.2.2-1 shows the deswelling profiles for three films along with AFM images describing the through-film morphology of each film.

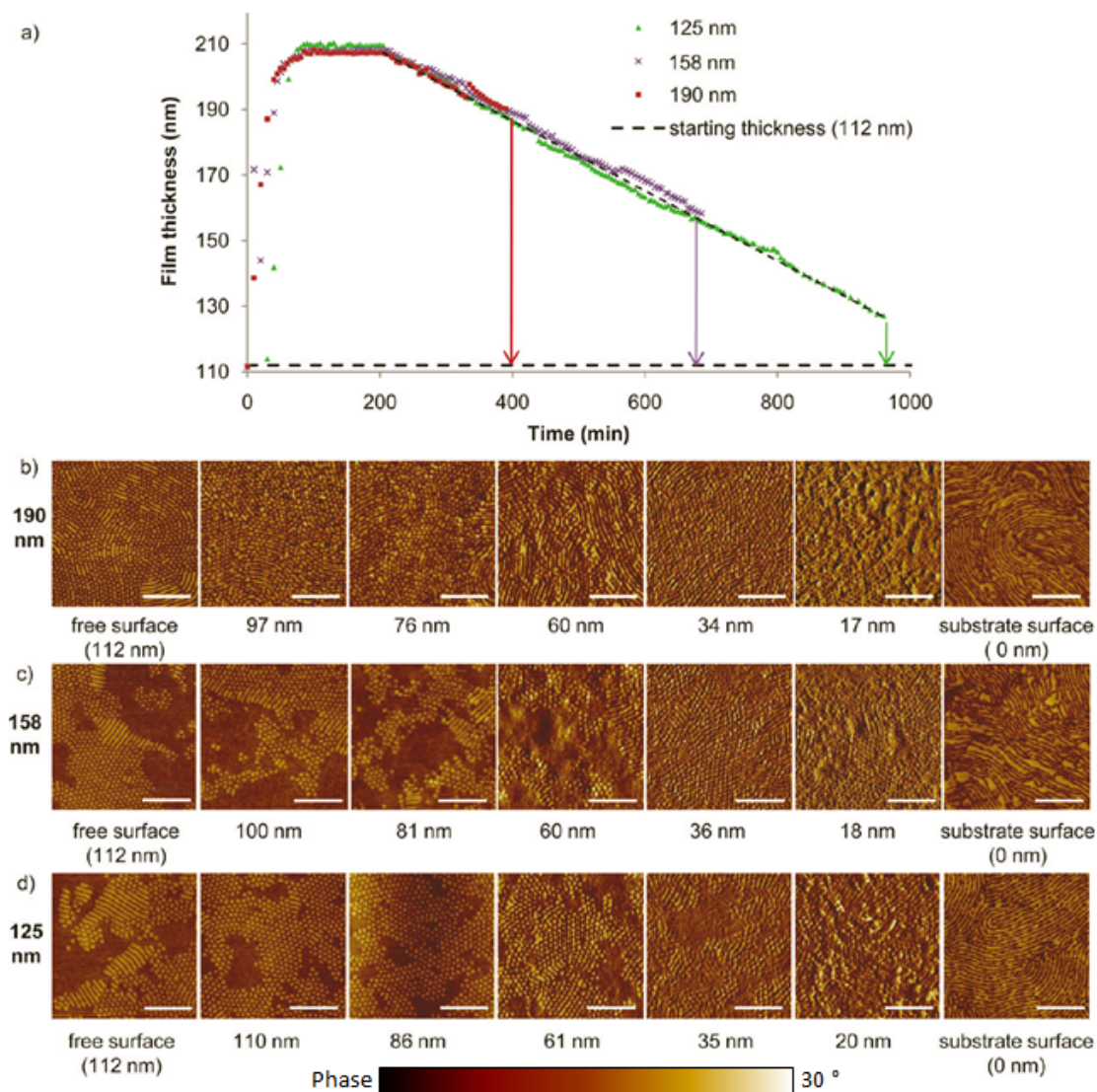


Figure 3.2.2-1 a.) Deswell profiles with stopping thickness varied and rate held constant, and b-d.) AFM images corresponding to the three stopping points showing the through film morphology. All scale bars are 500 nm.

It appears that the transition from perpendicular cylinders to parallel cylinders takes place primarily in the last domain spacing of the polymer. At thicknesses greater than one domain spacing, the morphology resembles that of the topography of the

film. In general, there are two routes for this re-orientation of morphology from parallel to perpendicular cylinders: a.) the parallel cylinders ‘flip’ into a perpendicular conformation or b.) the parallel cylinders break down and reorganize into perpendicular cylinders. Literature indicates that the latter is more common²³, and, based upon the AFM images, it is most likely the case for this system as well.

3.3 Morphological Control through Raster Solvent Vapor Annealing (RSVA)

The content of this section is adapted from Seppala, et al.²⁴

3.3.1 Single-pass RSVA

Similarly to the solvent removal studies, in this study nitrogen was flowed through two solvent bubblers in series filled with THF. The THF-saturated vapor then flowed through a hypodermic nozzle onto a constant thickness SIS thin film. The thickness of the film was initially 100 nm (3.7 domain spacings). A motorized stage was utilized to precisely move the sample under the nozzle, allowing for slow, zone annealing effects. A diagram of the experimental setup is shown in Figure 2.3.1-1.

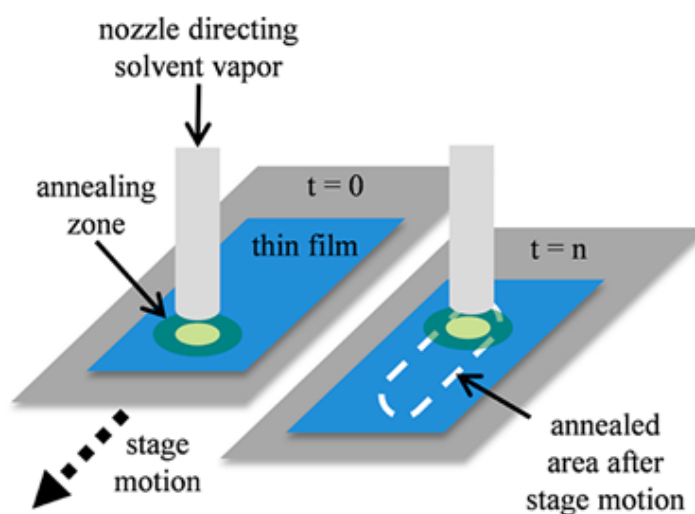


Figure 3.3.1-1 Experimental setup for raster solvent vapor annealing.

For the first set of experiments, a nozzle with an internal diameter (i.d.) of 0.514 mm and a nitrogen flow rate of 30 mL/min were used to observe the effects of raster zone annealing. AFM images of the resulting experiments are shown in Figure 3.3.1-2.

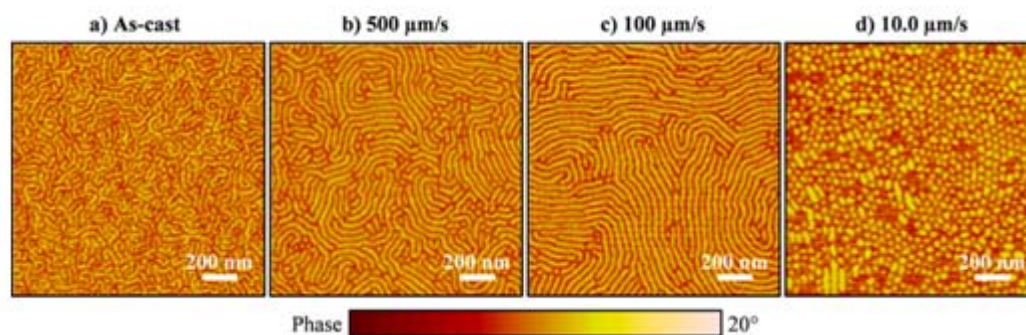


Figure 3.3.1-2 AFM images from anneals conducted with a 0.514 mm i.d. nozzle and a flow rate of 30 mL/min for three different speeds

As the stage speed decreases and therefore the annealing time increases, the morphology transitions from parallel to perpendicular cylinders. Because the area annealed on the film was relatively large for these experiments, additional experiments were required to attain a better understanding of smaller zone anneals. For these new experiments, a smaller nozzle, with an i.d. of 0.210 mm, was chosen in combination with a lower nitrogen flow rate, 5 mL/min, to maintain a constant solvent flux. Figure 3.3.1-3 contains AFM images from these experiments.

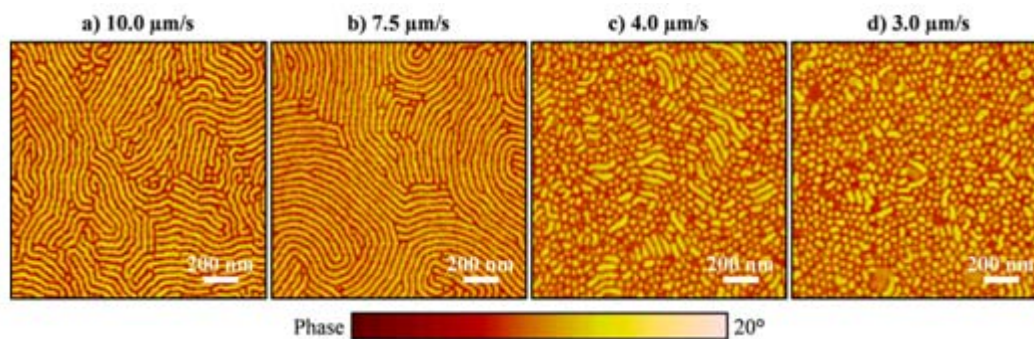


Figure 3.3.1-3 AFM images from anneals conducted with a 0.210 mm i.d. nozzle and a flow rate of 5 mL/min for three different speeds

Once again, there is a transition from parallel to perpendicular cylinders as the stage speed decreases. However, from comparing the two nozzle sizes, it is clear that the smaller nozzle required slower stage speeds overall to achieve the same morphologies as the larger nozzle. This is because, although the solvent flux through the nozzle is constant, the annealing zone on the film is much smaller, which implies that the stage speed should be reduced to allow for higher solvent-film permeation. Additionally, the overall amount of solvent flowing to the film is smaller, and therefore fluctuations in the air pressure that act as a disturbance to the system are amplified.

3.3.2 Multi-pass and Cross-path RSVA

These sets of experiments involve a similar setup to the previous ones, except that multiple passes or runs were performed on the same sample. For the multi-pass anneals, shown in Figure 3.3.2-1, the parameters of each pass were held constant from the single-pass anneals with the 0.514 mm i.d. nozzle.

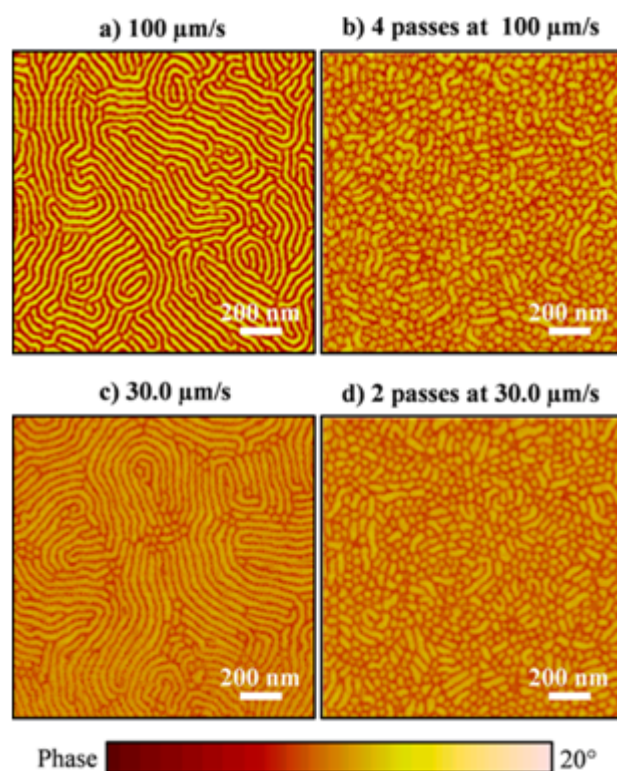


Figure 3.3.2-1 AFM images for the multi-pass experiments. The speed and number of passes are indicated above the images.

In between passes, the films were dried under vacuum, which implies that each additional pass was equivalent to subsequent anneals. As the number of passes is increased, the morphology transitions from parallel to perpendicular oriented cylinders.

For the cross-path RSVA studies, a single pass was performed, followed by another anneal such that the two paths intersected with each other. An image of the annealed film and the morphologies at different locations along the paths are shown in Figure 3.3.2-2. The annealed areas are indicated by the dark lines in the optical image, which are depressions from solvent altering the thickness of the film.

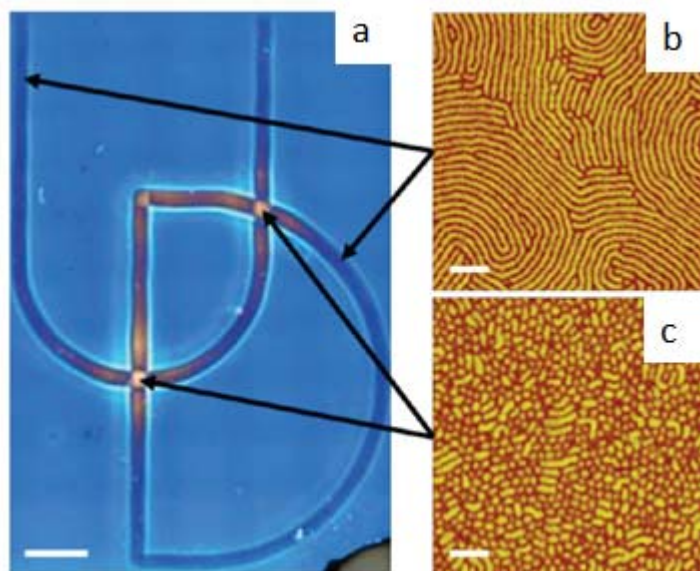


Figure 3.3.2-2 a.) Optical image of the cross-path experiment. Image is a combination of several smaller images taken through an optical microscope. Scale bar corresponds to 1 mm. b,c.) AFM images of the locations indicated, with scale bars corresponding to 200 nm. Adapted from Mastroianni, et al.²⁵

In the locations where there is only one pass of the nozzle, parallel cylinders are the primary morphology of the polymer. However, when the film was annealed twice in the crossed locations, the morphology changed to perpendicular cylinders. This is a similar result to the multi-pass experiments. For both the multi-pass and the cross-path experiments, the addition of subsequent anneals re-orient the cylinders through a mechanism similar to that of the previously mentioned solvent removal studies. This change is primarily induced by the need for the polymer to attain commensurability through multiple swells and deswells.

3.4 Quantitative Morphology Characterization from AFM Images

Data are an important aspect to the scientific method; however, the inability to fully understand and interpret the data renders all experiments useless. With this in mind, an effort was made to extract more information from AFM images in terms of mixed morphological characterization. More specifically, a code was developed in Java (see Appendix A) to accurately quantify the fraction of the image that contains perpendicular cylinders versus parallel cylinders. This was accomplished by first converting the AFM phase images to discrete binary, black and white images. White pixels represent structure, and black pixels represent matrix. From here, the image was loaded into the java program, which considered the color of each pixel, either black or white, and grouped the white pixels by nearest neighbor analysis. The size of each grouping was determined by counting the number of pixels in the group, and a size distribution was generated. A sample size distribution is shown in Figure 3.4-1 along with the corresponding image.

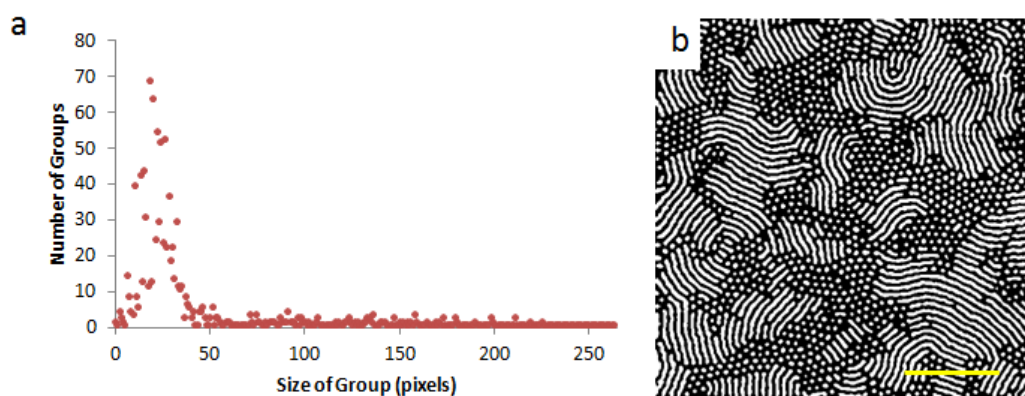


Figure 3.4-1 a.) Size distribution of the white groups in the sample AFM image shown in b.) The scale bar represents 500 nm. AFM image courtesy of Ming Luo.

Since the domain spacing of a block copolymer is constant, groups that represent perpendicular cylinders are all lumped together towards the beginning of the distribution, whereas parallel cylinders tend to be much larger and polydisperse in size. The total peak interval can be approximated as twice the location of the peak, which means that groups that are of size less than twice the peak location can be considered perpendicular. A simple re-coloring of the image (Figure 3.4-2) indicates that the program operates with relatively high accuracy. This analysis was particularly useful in the work done by Luo, et al.²⁶

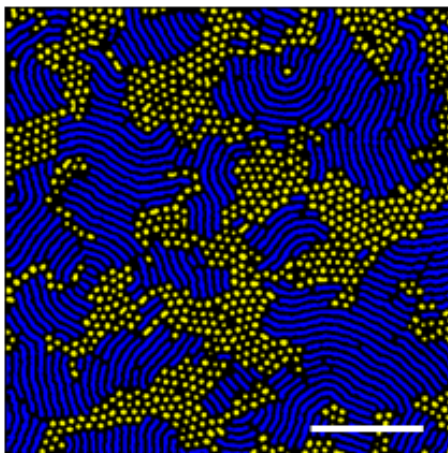


Figure 3.4-2 The image from Figure 2.4-1 re-colored to show regions of perpendicular and parallel cylinders. The scale bar represents 500 nm.

Chapter 4

APPLICATIONS FOR STEM CELL GROWTH AND DIFFERENTIATION

4.1 Motivation

For many years, stem cells have been a primary target of research for tissue growth and repair in the field of chemical and biomolecular engineering. It is known that the fate of stem cells is partially determined by the environment in which they grow^{27,28}. For example, human mesenchymal stem cells (MSC)s cultured on substrates that mimic the mechanical properties of bone, muscle, and brain differentiate into the respective cell types of these tissues, simply based upon their structural environment²⁹. This encourages the notion that artificial surfaces can be fabricated to imitate the conditions inside the body. Understanding how nanoscale structural organization of the cell microenvironment influences cell function and fate is of emerging interest³⁰. Block copolymers are an attractive platform to mimic the nanoscale organization of native tissue because of their ability to phase separate at the nanoscale into well-defined and periodic structures. The ultimate goal is to show that this phase separation can be used for stem cell differentiation.

4.2 Results and Discussion

In general, there are three stages to the scope of this project: a.) define polymer morphologies and processes needed to attain them, b.) functionalize the surface with peptides or other growth factors, and c.) grow stem cells on the artificial surface. The polymer utilized to conduct this study is a poly(styrene-*b*-isoprene) (SI)

diblock copolymer, with a number average molecular weight of 141.1 kg/mol, a PDI of 1.16, volume fractions of $f_{PS} = 0.70$ and $f_{PI} = 0.30$, and a bulk domain spacing of roughly 77 nm. Appendix B contains characterization data for the polymer.

4.2.1 Morphology Determination

Thin films were flow coated onto silicon wafers from 5 wt% solutions in THF. Solvent anneals were carried out by placing the films into a sealed chamber with a 10 mL beaker of the selected solvent for a predetermined amount of time. This type of annealing will be referred to as ‘bell jar’ annealing moving forward. Various solvents were chosen for testing based upon previous work.^{18,22} Figure 4.2.1-1 shows results from a series of different bell jar anneals.

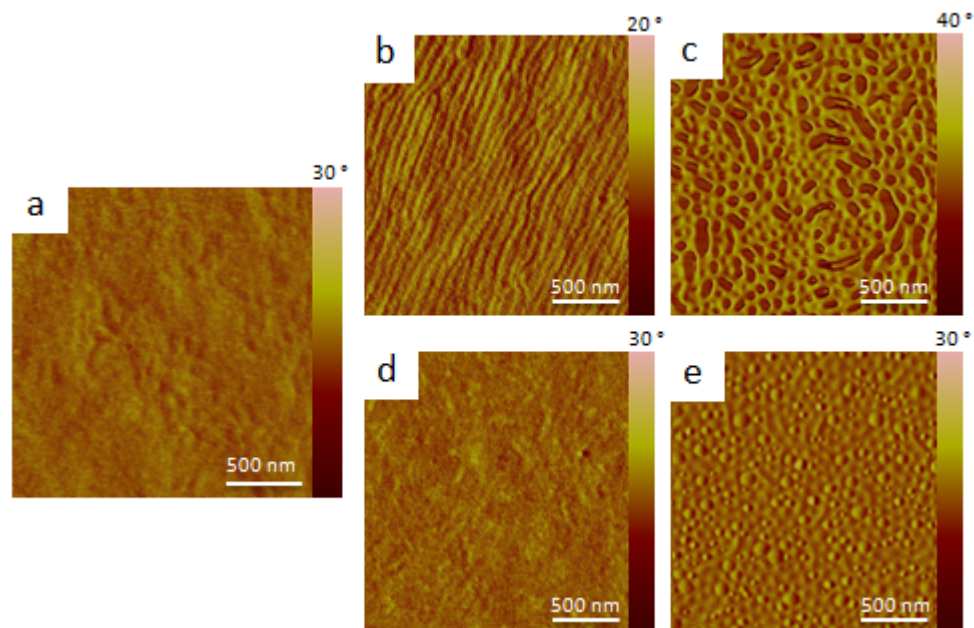


Figure 4.2.1-1 a.) As cast AFM image of SI. Images of samples bell jar annealed with b.) toluene, c.) *n*-hexane, d.) chloroform, and e.) acetone.

Toluene anneals have shown promising results to attain long range parallel cylinders. Ideally, conditions will be found to create well-ordered, long range perpendicular cylinders. Annealing with acetone creates structures resembling cylinders; however they are variable in size and not at all ordered. Further experimentation is required to determine the parameters necessary for perpendicular cylinder morphology.

4.2.2 Surface Functionalization

The methods used in this section roughly follow those outlined by Hawker, et al.³¹ Prior to functionalizing with peptides or functional groups, experiments were first conducted with acetylcysteine as the attaching molecule for proof of concept. Thiol-ene click chemistry was utilized to attach the acetylcysteine to the surface via the pendant isoprene double bonds. A bath of isopropanol was first heated to roughly 70 °C, to which the film was added with 2 mg of azobisisobutyronitrile and 20 mg of acetylcysteine. The reaction was allowed to proceed for 1 hour. Figure 4.2.2-1 contains AFM images showing the pre- and post-reaction film surface.

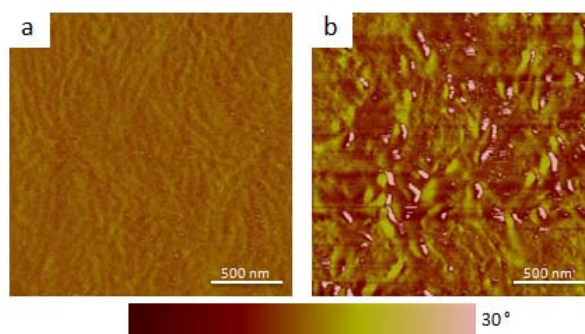


Figure 4.2.2-1 a.) Pre-reaction AFM image of the film, and b.) post-reaction AFM image of the film.

It appears that the reaction conditions have caused the roughness of the film to increase, however the underlying morphology still seems to be intact. The ^1H NMR results in Figure 4.2.2-2 indicates that the number of pendant double bonds on the isoprene reduced slightly from a peak ratio of 0.251 to 0.218, which coincides with the notion that the reaction proceeded successfully.

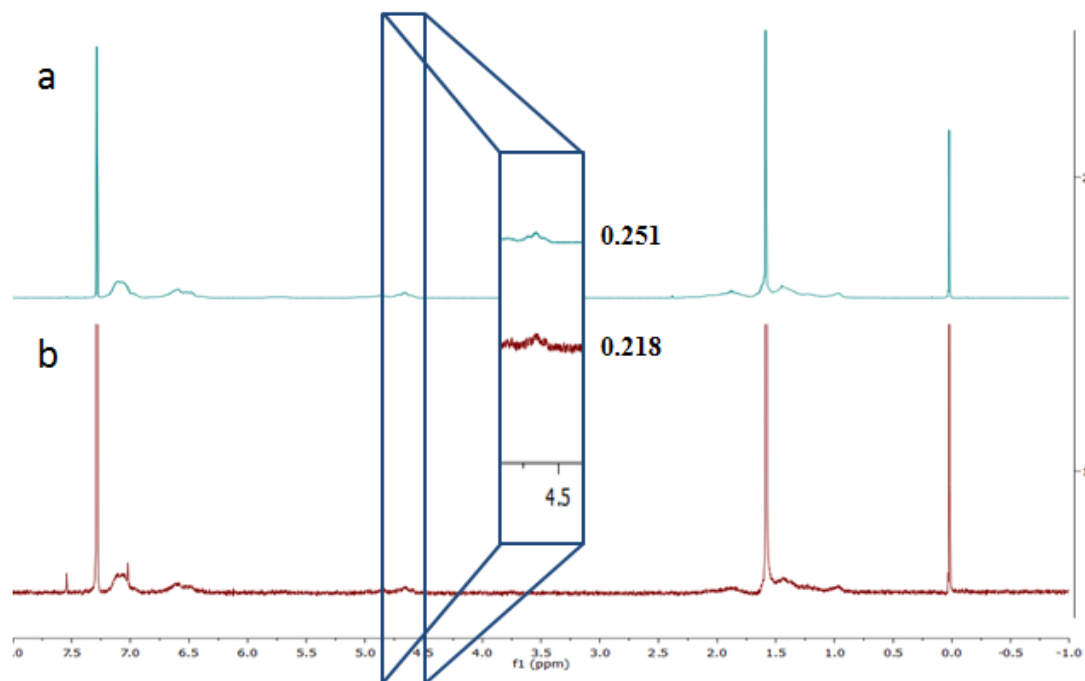


Figure 4.2.2-2 a.) ^1H NMR spectra for the polymer before the reaction, and b.) ^1H NMR spectra after the reaction. The inset shows the peaks that represent hydrogen from the isoprene pendant double bond, and the numbers next to the peaks indicate the ratio of these isoprene peaks to the styrene peaks.

However, the intensity of the post-reaction ^1H NMR was very low, and therefore the results are questionable.

4.2.3 Stem Cell Growth

While stem cells were not actually grown in this study, a background search through the literature was performed to determine some of the future experimental parameters. It is well understood that several variables affect the growth of stem cells^{32,33}; however it is crucial to evaluate which of these variables can actually be manipulated within this study. For this study, this long list reduces to essentially two controlling factors: the morphology of the polymer and the growth hormone or peptide attached to the film. In a study performed by Campos, et al., stem cells were grown on an ordered film of poly(styrene-*b*-ethylene oxide), which was pre-functionalized with CRGDS peptides. They observed a drastic change in the formation of focal adhesion sites and the morphologies of the cells from the use of an ordered substrate.

Similar results are expected from this study, with the addition of a reproducible method to create several cell morphologies or types. In the Campos paper, the underlying morphology of the block copolymer was hexagonally packed perpendicular cylinders. In this study, both perpendicular and parallel cylinders will be studied, adding to the results of Campos. The primary benefit of performing surface functionalization after polymer alignment comes from a materials and time conservation standpoint. Using the method outlined by Campos, in order to test multiple ligands or functional groups, a new polymer would have to be made each time. This is a laborious and costly procedure that can be avoided by functionalizing the polymer after alignment. A single aligned polymer can be divided into ten different pieces, and each piece can be tested differently. Eventually, an even higher throughput method may be developed to functionalize a single film with many different groups at the same time.

Experimentally, the different morphologies of the polymer could be tested with any combination of growth factors, such as activin³⁴ or a transforming growth factor³⁵, or ligands, as utilized by Campos. Additionally, the cells could be mechanically loaded using a loading device or fluid flow through the cells. The high number of parameters in this study allows for many experimental conditions and results.

Chapter 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

This work focuses primarily on the application of solvent annealing methods to block copolymer thin films, with an extended investigation of their use as biomimetic materials. A microfluidic device was shown to produce a morphological gradient across a thin film by evenly mixing two different solvents together. In the annealing process, the rate at which solvent exits the film results in a change in the structure of the film. In the case of SIS films annealed with chloroform, as the rate decreased, structure shifted from mostly parallel cylinders to perpendicular cylinders on the surface. Additionally, a Java program was developed to quantify the proportions of perpendicular and parallel cylinders in an AFM image. This was done by classifying the structures as either perpendicular or parallel and then calculating a percentage based on area fractions. Finally, bell jar annealing methods were employed to alter the morphology of SI films to investigate their use for biological applications. A technique was explored to attach peptides or growth factors to the surface of the film using click chemistry, however further characterization is required to confirm the outcome of the technique. Previous work shows that this is a promising venue of research, and it should be pursued moving forward.

5.2 Future Work

Experiments with the microfluidic device could be performed with different solvents or flow rates to capture various gradients of morphology. Additionally, the device could be redesigned to have a larger mixing tree or larger chambers to allow for solvent swelling gradients within the film. Solvent removal rates could be decreased to increase the ratio of perpendicular cylinders on the surface of the film. While quantification of the ratio of morphologies in AFM images was accomplished, a new program could be developed to evaluate the number of grains and the grain sizes of surface structures.

For the work done with applications to stem cells, conditions will be determined to generate long range perpendicular cylinders on the surface of the film in addition to the already attained parallel cylinders. Once a stable method for thiol-ene click chemistry is produced, different peptides and growth factors will be attached to the film to create a functional biomimetic material. The extent of reaction for the thiol-ene click chemistries can be quantified using ^1H NMR or FTIR. Finally, stem cells will be grown on this new substrate and their differentiation will be observed. Mechanical loading will also be implemented to affect the morphology and growth of the stem cells. These experiments should provide conditions for which the cells can reproducibly differentiate into various types of progenitor cells.

REFERENCES

1. Bates, Frank S., Glenn H. Frederickson. *Physics Today*. **1999**, 52(2), 32-38.
2. Bates, Frank S. *Science*. **1991**, 251(4996), 898-905.
3. Matsen, M. W., Frank S. Bates. *Macromolecules*. **1996**, 29(4), 1091-1098.
4. Matsen, M. W., R. B. Thompson. *Journal of Chemical Physics*. **1999**, 111 (15), 7139-7146.
5. Matsen, M. W. *Macromolecules*. **2012**, 45, 2161-2165.
6. Ji, Shengxiang, Chi-Chun Liu, Wen Liao, Alyssa L. Fenske, Gordon S. W. Craig, Paul F. Nealey. *Macromolecules*. **2011**, 44, 4291-4300.
7. Khandpur, Ashish K., Stephen Forster, Frank S. Bates, Ian W. Hamley, Anthony J. Ryan, Wim Bras, Kristoffer Almdal, Kell Mortenson. *Macromolecules*. **1995**, 28, 8796-8806.
8. Segalman, Rachel A. *Materials Science and Engineering Review*. **2005**, 48, 191-226.
9. Cheng, Joy Y., Charles T. Rettner, Daniel P. Sanders, Ho-Choel Kim, William D. Hinsberg. *Advanced Materials*. **2008**, 20, 3155-3158.
10. Park, Miri, Christopher Harrison, Paul M. Chaikin, Richard A. Register, Douglass H. Adamson. *Science*. **1997**, 276, 1401-1404.
11. Ruiz, Ricardo, Huiman Kang, Francois A. Detcheverry, Elizabeth Dobisz, Dan S. Kercher, Thomas R. Albrecht, Juan J. de Pablo, Paul F. Nealey. *Science*. **2008**, 321, 936-939.
12. Darling, Seth B. *Energy and Environmental Science*. **2009**, 2(12), 1221-1328.
13. Vohra, Varun, Mariano Campoy-Quiles, Miquel Garriga, Hideyuki Murata. *Journal of Materials Chemistry*. **2012**, 1-9.
14. Jeong, Byeongmoon, You Han Bae, Doo Sung Lee, Sung Wan Kim. *Nature*. **1997**, 388, 860-862.
15. Rosler, Annette, Guido W. M. Vandermeulan, Harm-Anton Klok. *Advanced Drug Delivery Reviews*. **2012**, 64, 270-279.
16. Adams, Monica L., Afsaneh Lavasanifar, Glen S. Kwon. *Journal of Pharmaceutical Sciences*. **2003**, 92(7), 1343-1355.

17. Pujari, Saswati, Michael A. Keaton, Paul M. Chaikin, Richard A. Register. *Soft Matter*. **2012**, 8, 5358-5363.
18. Albert, Julie N. L., Timothy D. Bogart, Ronald L. Lewis, Kathryn L. Beers, Michael J. Fasolka, J. Brian Hutchison, Bryan D. Vogt, Thomas H. Epps, III. *NanoLetters*. **2011**, 11, 1351-1357.
19. Stafford, Christopher M., Kristen E. Roskov, Thomas H. Epps, III, Michael J. Fasolka. *Rev. Sci. Instr.* **2006**, 77, 1-7.
20. Bogart, Timothy D. Design and Fabrication of a Solvent Resistant Microfluidic Device for Use in Combinatorial Solvent Annealing Studies of Block Copolymer Thin Films. B.Che. Thesis, University of Delaware, May 2010.
21. *Polymer Handbook*, 4th ed.; John Wiley & Sons, Inc: Hoboken 1999; Vol. 2.
22. Albert, Julie N. L., Wen-Shiue Young, Ronald L. Lewis, III, Timothy D. Bogart, Jasmine R. Smith, Thomas H. Epps. **2012**, 6(1), 459-466.
23. Paik, Marvin Y., Joan K. Bosworth, Detlef-M. Smilges, Evan L. Schwartz, Xavier Andre, Christopher K. Ober. *Macromolecules*. **2010**, 43(9), 4253-4260.
24. Seppala, Jonathan E., Ronald L. Lewis, III, Thomas H. Epps, III. *ACS Nano*. **2012**, 6(11), 9855-9862.
25. Mastroianni, Sarah E., Thomas H. Epps, III. *Langmuir*. **2013**, 29(12), 3864-3878.
26. Luo, Ming, Jonathan E. Seppala, Julie N. L. Albert, Ronald L. Lewis, III, Nikhila Mahadevapuram, Gila E. Stein, Thomas H. Epps, III. *Macromolecules*. **2013**, 46, 1803-1811.
27. Lutolf, Matthias P., Penney M. Gilbert, Helen M. Blau. *Nature*. **2009**, 462, 433-441.
28. Pittenger, Mark F., Alastair M. Mackay, Stephen C. Beck, Rama K. Jaiswal, Robin Douglas, Joseph D. Mosca, Mark A. Moorman, Donald W. Simonetti, Stewart Craig, Danial R. Marshak. *Science*. **1999**, 284, 143-147.
29. Engler, Adam J., Shamik Sen, H. Lee Sweeney, Dennis E. Discher. *Cell*. **2006**, 126, 677-689.
30. McNamara, Laura E., Rebecca J. McMurray, Manus J. P. Biggs, Fahsai Kantawong, Richard O. C. Oreffo, Matthew J. Dalby. *Journal of Tissue Engineering*. **2010**, 1, 1-13.
31. Campos, Luis M., Kato L. Killops, Ryosuke Sakai, Jos M. J. Paulusse, Denis Damiron, Eric Drockenmuller, Benjamin W. Messmore, Craig J. Hawker. *Macromolecules*. **2008**, 41, 7063-7070.
32. Tse, Justin R., Adam J. Engler. *PLOS ONE*. **2011**, 6(1), 1-9.

33. Woodruff, Maria Ann, Peter Jones, David Farrar, David M. Grant, Colin A. Scotchford. *Journal of Molecular Histology*. **2007**, 38, 491-499.
34. Schuldiner, Maya, Ofra Yanuka, Joseph Itskovitz-Eldor, Douglas A. Melton, Nissim Benvenisty. *PNAS*. **2000**, 97(21), 11307-11312.
35. Choi, Sunyoung, Tae-Jun Cho, Soon-Keun Kwon, Gene Lee, Jaejin Cho. *Int. Jour. Oral Sci.* **2013**, 5, 7-13.

Appendix A

MORPHOLOGY DETERMINATION CODE

```
package AFMAnalysis;

import java.util.ArrayList;

public class ScaledBlob {

    int blobID = -1;
    ArrayList<Pixel> pixels = new ArrayList<Pixel>();
    boolean inGrain = false;
}

package AFMAnalysis;
import java.awt.Color;

public class Pixel {

    boolean isPerpendicular;
    boolean isParallel;
    int pBlobID;
    Color color;
    Pixel next;
    int x, y;
    boolean isLine = false;
    boolean isEdge = false;
    int grainID = -1;

    public Pixel(boolean perp, boolean par, int blob, Color c, Pixel p, int x, int y){

        isPerpendicular = perp;
        isParallel = par;
        pBlobID = blob;
        color = c;
        next = p;
        this.x = x;
        this.y = y;
    }
}

package AFMAnalysis;
//Taken from www.vogella.de
```

```

public class Quicksort {
    private int[] numbers;
    private int number;

    public void sort(int[] values) {
        if (values == null || values.length == 0) {
            return;
        }
        this.numbers = values;
        number = values.length;
        quicksort(0, number - 1);
    }

    private void quicksort(int low, int high) {
        int i = low, j = high;
        int pivot = numbers[low + (high - low) / 2];

        while (i <= j) {
            while (numbers[i] < pivot) {
                i++;
            }
            while (numbers[j] > pivot) {
                j--;
            }
            if (i <= j) {
                exchange(i, j);
                i++;
                j--;
            }
        }
        if (low < j)
            quicksort(low, j);
        if (i < high)
            quicksort(i, high);
    }

    private void exchange(int i, int j) {
        int temp = numbers[i];
        numbers[i] = numbers[j];
        numbers[j] = temp;
    }
}

package AFMAAnalysis;
import java.awt.Color;
import java.awt.image.BufferedImage;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.text.DecimalFormat;
import java.util.ArrayList;

```

```

import javax.imageio.ImageIO;

public class Control {

    BufferedImage image = null;
    Pixel[][] pixels;
    ArrayList<Blob> blobArray = new ArrayList<Blob>();
    int currID = 0;

    public Control(File file){

        try{
            image = ImageIO.read(file);
        } catch(Exception e){
            System.out.println(e);
        }
        pixels = generatePixels(image);
    }

    public void createBlobs(){

        for(int count = 2; count < this.image.getHeight()-1; count = count + 1){
            for(int count1 = 2; count1 < this.image.getWidth()-1; count1 =
count1 + 1){

                this.setBlobID(count1, count);

            }
        }
    }

    public Pixel[][] generatePixels(BufferedImage image){

        int argbInt;
        Pixel[][] pixels = new Pixel[image.getWidth()][image.getHeight()];
        System.out.println(image.getHeight());
        for(int count = 0; count < image.getHeight(); count = count + 1){
            for(int count1 = 0; count1 < image.getWidth(); count1 = count1 +
1){

                argbInt = image.getRGB(count1, count);
                pixels[count1][count] = createPixel(argbInt, count1,
count);

            }
        }
        return pixels;
    }

    public Pixel createPixel(int argbInt, int x, int y){

        int red = (argbInt >> 16) & 0xff;
        int green = (argbInt >> 8) & 0xff;
        int blue = (argbInt) & 0xff;
        Color color = new Color(red, green, blue);
    }
}

```

```

        Pixel pixel = new Pixel(false, false, -1, color, null, x, y);
        return pixel;
    }

    public void setBlobID(int x, int y){

        //For each pixel: see if it is white.
        //If it is, then see if the others around it are also white
        //If any of them are, take their blobID
        //If they aren't, then move on
        //If they don't have an ID, create a new one for that pixel.
        if (this.pixels[x][y].color.getRed() > 125 &&
            this.pixels[x][y].color.getGreen() > 125 &&
            this.pixels[x][y].color.getBlue() > 125){

            this.checkAroundPixel(x,y);

        }
    }

    public Pixel[] smallerRange(int x,int y){

        Pixel[] tempPixels = new Pixel[4];
        tempPixels[0] = this.pixels[x-1][y-1];
        tempPixels[1] = this.pixels[x-1][y];
        tempPixels[2] = this.pixels[x][y-1];
        tempPixels[3] = this.pixels[x+1][y-1];
        return tempPixels;
    }

    public Pixel[] largerRange(int x,int y){

        Pixel[] tempPixels = new Pixel[12];
        tempPixels[0] = this.pixels[x-1][y-1];
        tempPixels[1] = this.pixels[x-1][y];
        tempPixels[2] = this.pixels[x][y-1];
        tempPixels[3] = this.pixels[x+1][y-1];
        tempPixels[4] = this.pixels[x-2][y];
        tempPixels[5] = this.pixels[x-2][y-1];
        tempPixels[6] = this.pixels[x-2][y-2];
        tempPixels[7] = this.pixels[x-1][y-2];
        tempPixels[8] = this.pixels[x][y-2];
        tempPixels[9] = this.pixels[x+1][y-2];
        tempPixels[10] = this.pixels[x+2][y-2];
        tempPixels[11] = this.pixels[x+2][y-1];
        return tempPixels;
    }

    public void checkAroundPixel(int x, int y){

        ArrayList<Pixel> reducedIDs = null;
        Pixel[] tempPixels = smallerRange(x,y);
        ArrayList<Pixel> IDs = new ArrayList<Pixel>();
    }

```

```

        for(int count = 0; count < tempPixels.length; count = count + 1){
            if(tempPixels[count].pBlobID != -1){
                IDs.add(tempPixels[count]);
                //System.out.println("Found another white pixel with an
ID.");
            }
        }
        if(IDs.size() != 0){
            reducedIDs = this.removeDuplicates(IDs);
            //System.out.println("Found at least 1 other white pixel. Reduced.");
        }
        else{
            reducedIDs = IDs;
            //System.out.println("There weren't any other white pixels with
IDs");
        }
        if(reducedIDs.size() == 0){
            this.blobArray.add(new Blob(this.pixels[x][y], this.currID, 0));
            this.pixels[x][y].pBlobID = this.currID;
            this.currID = this.currID + 1;
        }
        else if(reducedIDs.size() == 1){
            this.pixels[x][y].pBlobID = reducedIDs.get(0).pBlobID;
            this.goToEnd(this.blobArray.get(reducedIDs.get(0).pBlobID)).next
= this.pixels[x][y];
            //System.out.println("Adopted an ID from a previous pixel, and
attached to it's blob.");
        }
        else{
            int tempID = 0;
            if(reducedIDs.get(1).pBlobID < reducedIDs.get(0).pBlobID){
                //Take the first pixel, get its ID
                //Set the current pixels ID to that one
                this.pixels[x][y].pBlobID = reducedIDs.get(1).pBlobID;
                //Get the second pixels ID
                //Find its blob
                //set the first pixel.next equal to the second pixels blobs
head
                this.goToEnd(this.blobArray.get(reducedIDs.get(1).pBlobID)).next =
this.blobArray.get(reducedIDs.get(0).pBlobID).head;
                //reassign all of that blobs IDs
                tempID = reducedIDs.get(0).pBlobID;
                //reassign blobID of blob
                //delete old blob
                this.blobArray.remove(tempID);

                this.reassignBlobID(this.blobArray.get(reducedIDs.get(1).pBlobID),
reducedIDs.get(1).pBlobID);
                for(int count = tempID; count < this.blobArray.size();
count++){

```



```

                                this.reassignBlobID(this.blobArray.get(count),
this.blobArray.get(count).head.pBlobID - 1);
                                }
                                //get that blobs end and set its .next to the current

                                this.goToEnd(this.blobArray.get(reducedIDs.get(1).pBlobID)).next = this.pixels[x][y];
                                }
                                else{
                                    //Take the first pixel, get its ID
                                    //Set the current pixels ID to that one
                                    this.pixels[x][y].pBlobID = reducedIDs.get(0).pBlobID;
                                    //Get the second pixels ID
                                    //Find its blob
                                    //set the first pixel.next equal to the second pixels blobs
head
                                this.goToEnd(this.blobArray.get(reducedIDs.get(0).pBlobID)).next =
this.blobArray.get(reducedIDs.get(1).pBlobID).head;
                                    //reassign all of that blobs IDs
                                    tempID = reducedIDs.get(1).pBlobID;
                                    //reassign blobID of blob
                                    //delete old blob
                                    this.blobArray.remove(tempID);

                                    this.reassignBlobID(this.blobArray.get(reducedIDs.get(0).pBlobID),
reducedIDs.get(0).pBlobID);
                                for(int count = tempID; count < this.blobArray.size();
count++){
                                    this.reassignBlobID(this.blobArray.get(count),
this.blobArray.get(count).head.pBlobID - 1);
                                    }
                                    //get that blobs end and set its .next to the current

                                    this.goToEnd(this.blobArray.get(reducedIDs.get(0).pBlobID)).next = this.pixels[x][y];
                                    }
                                    //decrement currID
                                    this.currID--;
                                    //System.out.println("Encountered two blobs, eliminated one of
them.");
                                }
                                }

    public void generatePicture(String name){

        BufferedImage newImage = new BufferedImage(this.pixels.length,
this.pixels.length, BufferedImage.TYPE_INT_RGB);
        ArrayList<Integer> colors = new ArrayList<Integer>();
        colors.add(Color.RED.getRGB());
        colors.add(Color.BLUE.getRGB());
        colors.add(Color.GREEN.getRGB());
        colors.add(Color.YELLOW.getRGB());
        colors.add(Color.ORANGE.getRGB());

```

```

        for(int count = 0; count < this.pixels.length; count = count + 1){
            for(int count1 = 0; count1 < this.pixels.length; count1 = count1 +
1){
                if(this.pixels[count1][count].pBlobID != 0){
                    newImage.setRGB(count1, count,
colors.get(this.pixels[count1][count].pBlobID%5));
                }
                else{
                    newImage.setRGB(count1, count,
Color.BLACK.getRGB());
                }
            }
        }
        try {
            File rendered = new File("C:\\Users\\Ron\\Desktop\\"+ name +
".png");
            ImageIO.write(newImage, "png", rendered);
        } catch (IOException e) {
            System.out.println(e);
        }
    }

    public void generateFinalPicture(String name, File location, boolean perp, boolean
par){

        BufferedImage newImage = new BufferedImage(this.pixels.length,
this.pixels.length, BufferedImage.TYPE_INT_RGB);

        for(int count = 0; count < this.pixels.length; count = count + 1){
            for(int count1 = 0; count1 < this.pixels.length; count1 = count1 +
1){
                if(this.pixels[count1][count].isPerpendicular == true &&
perp){
                    newImage.setRGB(count1, count,
Color.YELLOW.getRGB());
                }
                else if(this.pixels[count1][count].isParallel == true &&
par){
                    newImage.setRGB(count1, count,
Color.BLUE.getRGB());
                }
                else{
                    newImage.setRGB(count1, count,
Color.BLACK.getRGB());
                }
            }
        }
        try {
            File rendered = new File(location.getAbsolutePath(), name +
".png");
            ImageIO.write(newImage, "png", rendered);
        } catch (IOException e) {

```

```

        System.out.println(e);
    }
}

public ArrayList<Pixel> removeDuplicates(ArrayList<Pixel> data){

    ArrayList<Pixel> reducedData = new ArrayList<Pixel>();
    reducedData.add(data.get(0));
    boolean searching;
    for(int count = 0; count < data.size(); count = count + 1){
        searching = true;
        for(int count1 = 0; count1 < reducedData.size(); count1 = count1 +
1){
            if((data.get(count).pBlobID !=
reducedData.get(count1).pBlobID) && searching){
                reducedData.add(data.get(count));
                searching = false;
            }
        }
    }

    return reducedData;
}

public Pixel goToEnd(Blob blob){

    Pixel curr = blob.head;
    while(curr.next != null){
        curr = curr.next;
    }
    return curr;
}

public ArrayList<Blob> reduceBlobs(int minPixels){

    ArrayList<Blob> reducedArrayList = new ArrayList<Blob>();
    for(int count = 0; count < this.blobArray.size(); count = count + 1){
        int length = this.getLength(this.blobArray.get(count));
        if(length >= minPixels){
            reducedArrayList.add(this.blobArray.get(count));
        }
        else{
            this.reassignBlobID(this.blobArray.get(count), 0);
        }
    }
    return reducedArrayList;
}

public int getLength(Blob b){

    Pixel curr = b.head;
    int counter = 1;

```

```

        while(curr != null){
            counter = counter + 1;
            curr = curr.next;
        }
        return counter;
    }

    public void reassignBlobID(Blob b, int desired){

        Pixel curr = b.head;
        while(curr.next != null){
            curr.pBlobID = desired;
            curr = curr.next;
        }
        curr.pBlobID = desired;
        b.blobID = desired;

    }

    double roundTwoDecimals(double d) {
        DecimalFormat twoDForm = new DecimalFormat("#.###");
        return Double.valueOf(twoDForm.format(d));
    }

    public void getRadialDistribution(Solver solver, File location, String name){

        solver.actualRadialDistribution(blobArray, 500, 12);
        double[] values = solver.getRValues();
        double[] theoreticalRValues = solver.getTheoreticalRValues();
        double[] theoreticalFreqs = solver.getTheoreticalFreqs();
        double[] freqs = solver.getFreqs();
        double[] movingAverage = solver.movingAverage(2);
        System.out.println(values.length);
        double[] idealValues = solver.idealRadialDistribution(values.length);
        try {
            File file = new File(location.getAbsolutePath(), name + "1.txt");
            BufferedWriter out = new BufferedWriter(new FileWriter(file));
            out.write("Count, Actual r, g(r), Moving Average, Theoretical r,

g(r)");

            out.newLine();
            for(int count = 0; count < values.length; count++){
                if(theoreticalRValues.length > count){
                    out.write(count + ", " + values[count] + ", " +
freqs[count] + ", " + movingAverage[count] + ", " + theoreticalRValues[count] + ", " +
theoreticalFreqs[count]);
                }
                else{
                    out.write(count + ", " + values[count] + ", " +
freqs[count] + ", " + movingAverage[count]);
                }
                out.newLine();
            }
        }
    }

```

```

        out.close();
    } catch (IOException e){
        e.printStackTrace();
    }
}
}
package AFMAnalysis;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.lang.Math;

public class Solver {

    int[] counts;
    int[] frequencies;
    int[] sizes;
    double[] rValues;
    double[] pairFunction;
    double[] theoreticalFreqs;
    double[] theoreticalRValues;
    int parallel;
    int perpendicular;
    int threshold;
    double averagePerpSize;

    public void countPixels(ArrayList<Blob> blobArray){

        this.counts = new int[blobArray.size()];
        Blob currentBlob = null;
        Pixel current = null;
        int counter;
        for(int count = 0; count < blobArray.size(); count = count + 1){
            currentBlob = blobArray.get(count);
            current = currentBlob.head;
            counter = 0;
            while(current != null){
                counter++;
                current = current.next;
            }
            this.counts[count] = counter;
            currentBlob.size = counter;
        }
        Quicksort sorter = new Quicksort();
        sorter.sort(this.counts);
    }

    public void findFrequencies(){

```

```

        int[] tempSizes = new int[this.counts.length];
        int[] tempFrequencies = new int[this.counts.length];
        int index = 0;
        int counter = 0;
        int frequency = 0;
        boolean searching;
        while(index < this.counts.length){
            searching = true;
            tempSizes[counter] = this.counts[index];
            index++;
            frequency = 1;
            while(index < this.counts.length && searching == true){
                if(this.counts[index] == this.counts[index-1]){
                    index++;
                    frequency++;
                }
                else{
                    searching = false;
                }
            }
            tempFrequencies[counter] = frequency;
            counter++;
        }
        this.frequencies = new int[counter];
        this.sizes = new int[counter];
        for(int count = 0; count < counter; count = count + 1){
            this.frequencies[count] = tempFrequencies[count];
            this.sizes[count] = tempSizes[count];
        }
    }

    public void solve(ArrayList<Blob> blobArray, File location, String fileName){

        this.countPixels(blobArray);
        this.findFrequencies();
        this.labelBlobs(blobArray);
        File file = new File(location.getAbsolutePath(), fileName + ".txt");
        if(!file.exists()){
            try {
                file.createNewFile();
            } catch (IOException e) {
                System.out.println("Cannot create file.");
            }
        }
        try {
            BufferedWriter output = new BufferedWriter(new FileWriter(file));
            for(int index = 0; index < this.sizes.length; index = index + 1){
                output.write(index + ", " + this.sizes[index] + ", " +
this.frequencies[index]);

                output.newLine();
            }
        }
    }

```

```

        output.close();
    } catch (IOException e) {
        System.out.println("Can't find the file.");
    }
}

public int determineThreshold(){

    int threshold = 0;
    int temp;
    int firstFreq = 0;
    int firstIndex = 0;
    int secondFreq = 0;
    int secondIndex = 0;
    int thirdFreq = 0;
    int thirdIndex = 0;
    int fourthFreq = 0;
    int fourthIndex = 0;
    int fifthFreq = 0;
    int fifthIndex = 0;
    int[] closest = new int[5];

    for (int count = 0; count < this.frequencies.length; count++){

        temp = this.frequencies[count];
        if (temp > firstFreq){
            fifthIndex = fourthIndex;
            fifthFreq = fourthFreq;
            fourthIndex = thirdIndex;
            fourthFreq = thirdFreq;
            thirdIndex = secondIndex;
            thirdFreq = secondFreq;
            secondIndex = firstIndex;
            secondFreq = firstFreq;
            firstIndex = count;
            firstFreq = temp;
        }
        else if(temp > secondFreq){
            fifthIndex = fourthIndex;
            fifthFreq = fourthFreq;
            fourthIndex = thirdIndex;
            fourthFreq = thirdFreq;
            thirdIndex = secondIndex;
            thirdFreq = secondFreq;
            secondIndex = count;
            secondFreq = temp;
        }
        else if(temp > thirdFreq){
            fifthIndex = fourthIndex;
            fifthFreq = fourthFreq;
            fourthIndex = thirdIndex;
            fourthFreq = thirdFreq;

```

```

        thirdIndex = count;
        thirdFreq = temp;
    }
    else if(temp > fourthFreq){
        fifthIndex = fourthIndex;
        fifthFreq = fourthFreq;
        fourthIndex = count;
        fourthFreq = temp;
    }
    else if(temp > fifthFreq){
        fifthIndex = count;
        fifthFreq = temp;
    }
}

closest[0] = firstIndex;
closest[1] = secondIndex;
closest[2] = thirdIndex;
closest[3] = fourthIndex;
closest[4] = fifthIndex;

int counter = 0;
ArrayList<Integer> consecs = new ArrayList<Integer>();
consecs.add(this.sizes[firstIndex]);

for(int count1 = 0; count1 < closest.length-1; count1++){
    for(int count = count1+1; count < closest.length; count++){

        if(Math.abs(this.sizes[closest[count1]]-
this.sizes[closest[count]]) <= 5){

            counter++;
            if(!consecs.contains(this.sizes[closest[count1]])){
                consecs.add(this.sizes[closest[count1]]);
            }
            if(!consecs.contains(this.sizes[closest[count]])){
                consecs.add(this.sizes[closest[count]]);
            }
        }
    }
}
/*
for(int count = 0; count < closest.length; count++){
    System.out.println(this.sizes[closest[count]]);
}

System.out.println(counter);*/

if(counter >= 2 && this.frequencies[closest[0]] > 5 &&
this.frequencies[closest[1]] > 5){
    int sum = 0;
    for(int count = 0; count < consecs.size(); count++){
        sum = sum + consecs.get(count);
    }
}

```



```

        }
        threshold = sum*2/consecs.size();
    }

    return threshold;
}

public void labelBlobs(ArrayList<Blob> blobArray){

    this.parallel = 0;
    this.perpendicular = 0;
    threshold = determineThreshold();
    // System.out.println(threshold);
    for(int index = 0; index < blobArray.size(); index = index + 1){
        Pixel current = blobArray.get(index).head;
        if(blobArray.get(index).size <= threshold){
            while(current != null){
                if(current.isPerpendicular == false &&
current.isParallel == false){

                    this.perpendicular++;
                    current.isPerpendicular = true;
                }
                current = current.next;
            }
        }
        else{
            while(current != null){
                if(current.isPerpendicular == false &&
current.isParallel == false){

                    this.parallel++;
                    current.isParallel = true;
                }
                current = current.next;
            }
        }
    }
}

public double[][] determineCenters(ArrayList<Blob> blobArray){

    int counter;
    int sumX;
    int sumY;
    Pixel current;
    int numPerp = 0;
    for(int count = 0; count < blobArray.size(); count++){
        if(blobArray.get(count).head.isPerpendicular){
            numPerp++;
        }
    }
    double[][] averages = new double[numPerp][2];
    int index = 0;

```

```

        int masterCounter = 0;
        for(int count = 0; count < blobArray.size(); count++){
            current = blobArray.get(count).head;
            counter = 0;
            sumX = 0;
            sumY = 0;
            if(current.isPerpendicular){
                while(current.next != null){
                    counter++;
                    sumX+=current.x;
                    sumY+=current.y;
                    current = current.next;
                }
                masterCounter+=counter;
                averages[index][0] = (double)sumX/counter;
                averages[index][1] = (double)sumY/counter;
                index++;
            }
        }
        averagePerpSize = (double)masterCounter/numPerp;
        return averages;
    }

    public void actualRadialDistribution(ArrayList<Blob> blobArray, int size, int
numPeaks){

        double[][] averages = determineCenters(blobArray);
        ArrayList<Double> distribution = new ArrayList<Double>();
        double distance, x1, y1, x2, y2;
        for(int count = 0; count < averages.length; count++){
            for(int count1 = count + 1; count1 < averages.length; count1++){
                x1 = averages[count][0];
                x2 = averages[count1][0];
                y1 = averages[count][1];
                y2 = averages[count1][1];
                distance = Math.sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
                distribution.add(distance);
            }
        }

        double average = 0;
        double min = distribution.get(0);

        for(int count = 0; count < distribution.size(); count++){
            if(distribution.get(count) < min){
                min = distribution.get(count);
            }
            average += distribution.get(count)/distribution.size();
        }

        System.out.println(average);
    }

```

```

double increment = (average-min)/size;

pairFunction = new double[size];
int[] freqs = new int[size];
rValues = new double[size];
double current = min;

for(int count = 1; count <= size; count++){
    for(int i = 0; i < distribution.size(); i++){
        if(distribution.get(i) <= (current+increment) &&
distribution.get(i) > current){
            freqs[count-1] = freqs[count-1] + 1;
        }
        current = current + increment;
        rValues[count-1] = current;
    }

pairFunction[0] = freqs[0]/(Math.PI*rValues[0]*rValues[0]);

double criticalRValue = 0;
double criticalGValue = 0;
boolean searching = true;

for(int count = 1; count < rValues.length; count++){
    pairFunction[count] =
freqs[count]/(Math.PI*(rValues[count]*rValues[count]-rValues[count-1]*rValues[count-1]));
    if((pairFunction[count] - pairFunction[count-1]) < 0 &&
searching){
        criticalRValue = rValues[count-1];
        criticalGValue = pairFunction[count-1];
        searching = false;
    }
}

double[] peakValues = getTheoreticalPeakValues(numPeaks);
theoreticalRValues = new double[peakValues.length];
theoreticalFreqs = new double[peakValues.length];

theoreticalFreqs[0] = criticalGValue;

for(int count = 0; count < peakValues.length; count++){
    theoreticalRValues[count] =
Math.sqrt(peakValues[count])*criticalRValue;
}

for(int count = 1; count < theoreticalFreqs.length; count++){
    int currentIndex = 0;
    while(rValues[currentIndex] < theoreticalRValues[count]){
        currentIndex++;
    }
}

```

```

        theoreticalFreqs[count] =
(pairFunction[currentIndex]+pairFunction[currentIndex-1])/2;
    }

    }

    public double[] getFreqs(){
        return pairFunction;
    }

    public double[] getTheoreticalFreqs(){
        return theoreticalFreqs;
    }

    public double[] getTheoreticalRValues(){
        return theoreticalRValues;
    }

    public double[] getRValues(){
        return rValues;
    }

    public double[] getTheoreticalPeakValues(int size){

        ArrayList<Integer> n = new ArrayList<Integer>();
        ArrayList<Integer> d = new ArrayList<Integer>();

        int m = 2;

        n.add(1);
        d.add(1);
        int count = 1;
        int dValue = 2;

        while(count < size){
            //Fill n values
            for(int i = 0; i < 2*m; i++){
                if((i+1)%2 != 0){
                    n.add(i);
                }
            }
            for(int i = 0; i < 2*m; i++){
                if((i+1)%2 == 0){
                    n.add(i);
                }
            }

            //Fill d values
            for(int i = 0; i < m; i++){
                d.add(dValue);
            }
            dValue++;
        }
    }

```

```

        for(int i = m; i < 2*m; i++){
            d.add(dValue);
        }
        dValue++;
        count = count + 2*m;
        m++;
    }

    double[] x = new double[n.size()];

    for(int index = 0; index < x.length; index++){
        double numerator = (double)n.get(index)/2;
        double denominator = (double)d.get(index)*(Math.sqrt(3)/2);
        x[index] = numerator/denominator;
    }

    double[] values = new double[x.length];

    for(int index = 0; index < values.length; index++){
        values[index] = 3*d.get(index)*d.get(index)*(1 +
x[index]*x[index])/4;
        //System.out.print(values[index] + ", ");
    }

    return values;
}

public double[] movingAverage(int period){

    double[] movingAverage = new double[pairFunction.length];
    double sum;

    for(int count = 0; count < period-1; count++){
        movingAverage[count] = 0;
    }

    for(int count = period-1; count < movingAverage.length; count++){
        sum = 0;
        for(int index = 0; index < period; index++){
            sum = sum + pairFunction[count - index];
        }
        movingAverage[count] = sum/period;
    }

    return movingAverage;
}

public double[] idealRadialDistribution(int size){

    double[] values = getTheoreticalPeakValues(size);

    double averageDistance = 4*Math.sqrt(averagePerpSize/Math.PI);

```

```

        //System.out.println(averageDistance);

        double[] idealValues = new double[values.length];
        for(int index = 0; index < idealValues.length; index++){
            double distance = averageDistance*Math.sqrt(values[index]);
            idealValues[index] = distance/(Math.PI*distance*distance);
            //System.out.print(idealValues[index] + ", ");
        }

        return idealValues;
    }
}
package AFMAnalysis;

import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Rectangle;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.util.ArrayList;

import javax.swing.JFileChooser;
import javax.swing.JRadioButton;
import javax.swing.JCheckBox;
import javax.swing.JTextField;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.event.ChangeEvent;

public class AFMAnalysis extends JFrame{

    private JFrame mainFrame = null; // @jve:decl-index=0:visual-constraint="16,-13"
    private JPanel mainPane = null;
    private JCheckBox perpCheckBox = null;
    private JCheckBox parCheckBox = null;
    private JTextField currentImageText = null;
    private JButton currentImageButton = null;
    private JLabel currentImageLabel = null;
    private JLabel finalImageLocationLabel = null;
    private JButton finalImageLocationBrowse = null;
    private JTextField finalImageLocationText = null;
    private JLabel finalImageNameLabel = null;
    private JTextField finalImageNameText = null;
    private JButton runButton = null;
    private Control control = null;
    private File startingImage = null;
    private File finalImageLocation = null;

```

```

private String finalImageName = null;
private boolean perp = false;
private boolean par = false;
private JLabel resultsLabel = null;
private JTextField perpPercentOutput = null;
private JLabel perpPercentLabel = null;
private JTextField parPercentOutput = null;
private JLabel parPercentLabel = null;

/**
 * This method initializes mainFrame
 *
 * @return javax.swing.JFrame
 */
private JFrame getMainFrame() {
    if (mainFrame == null) {
        mainFrame = new JFrame();
        mainFrame.setSize(new Dimension(330, 513));
        mainFrame.setTitle("AFMAnalysis");
        mainFrame.setContentPane(getMainPane());
    }
    return mainFrame;
}

/**
 * This method initializes mainPane
 *
 * @return javax.swing.JPanel
 */
private JPanel getMainPane() {
    if (mainPane == null) {
        parPercentLabel = new JLabel();
        parPercentLabel.setBounds(new Rectangle(178, 377, 96, 25));
        parPercentLabel.setText("Percent Parallel");
        perpPercentLabel = new JLabel();
        perpPercentLabel.setBounds(new Rectangle(14, 377, 131, 25));
        perpPercentLabel.setText("Percent Perpendicular");
        resultsLabel = new JLabel();
        resultsLabel.setBounds(new Rectangle(133, 337, 47, 20));
        resultsLabel.setText("Results");
        finalImageNameLabel = new JLabel();
        finalImageNameLabel.setBounds(new Rectangle(106, 258, 102,
26));

        finalImageNameLabel.setText("Final Image Name");
        finalImageLocationLabel = new JLabel();
        finalImageLocationLabel.setBounds(new Rectangle(95, 171, 117,
26));

        finalImageLocationLabel.setText("Final Image Location");
        currentImageLabel = new JLabel();
        currentImageLabel.setBounds(new Rectangle(56, 73, 196, 28));
        currentImageLabel.setText("Starting Image Name and Location");
        mainPane = new JPanel();

```

```

        mainPane.setLayout(null);
        mainPane.add(getPerpCheckBox(), null);
        mainPane.add(getParCheckBox(), null);
        mainPane.add(getCurrentImageText(), null);
        mainPane.add(getCurrentImageButton(), null);
        mainPane.add(currentImageLabel, null);
        mainPane.add(finalImageLocationLabel, null);
        mainPane.add(getFinalImageLocationBrowse(), null);
        mainPane.add(getFinalImageLocationText(), null);
        mainPane.add(finalImageNameLabel, null);
        mainPane.add(getFinalImageNameText(), null);
        mainPane.add(getRunButton(), null);
        mainPane.add(resultsLabel, null);
        mainPane.add(getPerpPercentOutput(), null);
        mainPane.add(perpPercentLabel, null);
        mainPane.add(getParPercentOutput(), null);
        mainPane.add(parPercentLabel, null);
    }
    return mainPane;
}

/**
 * This method initializes perpCheckBox
 *
 * @return javax.swing.JCheckBox
 */
private JCheckBox getPerpCheckBox() {
    if (perpCheckBox == null) {
        perpCheckBox = new JCheckBox();
        perpCheckBox.setBounds(new Rectangle(15, 36, 107, 22));
        perpCheckBox.setText("Perpendicular");
    }
    return perpCheckBox;
}

/**
 * This method initializes parCheckBox
 *
 * @return javax.swing.JCheckBox
 */
private JCheckBox getParCheckBox() {
    if (parCheckBox == null) {
        parCheckBox = new JCheckBox();
        parCheckBox.setBounds(new Rectangle(149, 36, 75, 22));
        parCheckBox.setText("Parallel");
    }
    return parCheckBox;
}

/**
 * This method initializes currentImageText
 *

```



```

    * @return javax.swing.JTextField
    */
private JTextField getCurrentImageText() {
    if (currentImageText == null) {
        currentImageText = new JTextField();
        currentImageText.setBounds(new Rectangle(13, 126, 185, 24));
        currentImageText.setText("Input required here");
    }
    return currentImageText;
}

/**
 * This method initializes currentImageButton
 *
 * @return javax.swing.JButton
 */
private JButton getCurrentImageButton() {
    if (currentImageButton == null) {
        currentImageButton = new JButton();
        currentImageButton.setBounds(new Rectangle(216, 123, 82, 32));
        currentImageButton.setText("Browse");
        currentImageButton.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent
e) {
        JFileChooser fileChooser = new JFileChooser();

        fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
        fileChooser.setDialogTitle("Starting Image Name
and Location");

        fileChooser.showOpenDialog(mainFrame);
        startingImage = fileChooser.getSelectedFile();

        currentImageText.setText(startingImage.getPath());
    }
});
    }
    return currentImageButton;
}

/**
 * This method initializes finalImageLocationBrowse
 *
 * @return javax.swing.JButton
 */
private JButton getFinalImageLocationBrowse() {
    if (finalImageLocationBrowse == null) {
        finalImageLocationBrowse = new JButton();
        finalImageLocationBrowse.setBounds(new Rectangle(216, 213, 82,
32));

        finalImageLocationBrowse.setText("Browse");
    }
}

```

```

        finalImageLocationBrowse.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent
e) {
                JFileChooser fileChooser = new JFileChooser();

                fileChooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
                fileChooser.setDialogTitle("Final Image
Location");

                fileChooser.showOpenDialog(mainFrame);
                finalImageLocation =
fileChooser.getSelectedFile();

                finalImageLocationText.setText(finalImageLocation.getPath());
            }
        });
    }
    return finalImageLocationBrowse;
}

/**
 * This method initializes finalImageLocationText
 *
 * @return javax.swing.JTextField
 */
private JTextField getFinalImageLocationText() {
    if (finalImageLocationText == null) {
        finalImageLocationText = new JTextField();
        finalImageLocationText.setBounds(new Rectangle(13, 215, 185,
24));

        finalImageLocationText.setText("Input required here");
    }
    return finalImageLocationText;
}

/**
 * This method initializes finalImageNameText
 *
 * @return javax.swing.JTextField
 */
private JTextField getFinalImageNameText() {
    if (finalImageNameText == null) {
        finalImageNameText = new JTextField();
        finalImageNameText.setBounds(new Rectangle(65, 298, 185, 24));
        finalImageNameText.setText("Input file name here");
    }
    return finalImageNameText;
}

/**
 * This method initializes runButton
 *

```

```

        * @return javax.swing.JButton
        */
private JButton getRunButton() {
    if (runButton == null) {
        runButton = new JButton();
        runButton.setBounds(new Rectangle(240, 15, 59, 42));
        runButton.setText("Run!");
        runButton.addActionListener(new java.awt.event.ActionListener()
{
    public void actionPerformed(java.awt.event.ActionEvent
e) {

        if(perpCheckBox.isSelected()){
            perp = true;
        }
        else{
            perp = false;
        }
        if(parCheckBox.isSelected()){
            par = true;
        }
        else{
            par = false;
        }
        JFileChooser openChooser = new JFileChooser();

        openChooser.setDialogType(JFileChooser.OPEN_DIALOG);
        openChooser.showOpenDialog(mainPane);
        openChooser.setDialogTitle("Open File");
        startingImage = openChooser.getSelectedFile();
        JFileChooser saveChooser = new JFileChooser();

        saveChooser.setDialogType(JFileChooser.SAVE_DIALOG);
        saveChooser.showSaveDialog(mainPane);
        saveChooser.setDialogTitle("Save Location");
        finalImageLocation =
saveChooser.getCurrentDirectory();
        finalImageName =
saveChooser.getName(saveChooser.getSelectedFile());
        control = new Control(startingImage);
        control.createBlobs();
        control.blobArray = control.reduceBlobs(5);
        Solver solver = new Solver();
        solver.solve(control.blobArray,
finalImageLocation, finalImageName);

        control.generateFinalPicture(finalImageName,
finalImageLocation, perp, par);

        GrainDetector g = new
GrainDetector(control.pixels, control.blobArray);
        //g.generateFinalScaledPicture(finalImageName,
finalImageLocation, perp, par);

        System.out.println("Percent Parallel:");
    }
}
}

```

```

double percentPar =
control.roundTwoDecimals(((double)solver.parallel*100/(solver.parallel+solver.perpendicular));
System.out.println(percentPar);
System.out.println("Percent Perpendicular:");
double percentPerp =
control.roundTwoDecimals(((double)solver.perpendicular*100/(solver.parallel+solver.perpendicular));
System.out.println(percentPerp);
perpPercentOutput.setText(percentPerp + " ");
parPercentOutput.setText(percentPar + " ");
control.getRadialDistribution(solver,

finalImageLocation, finalImageName);

System.out.println("Done!");
//Comment the following out to run properly until

this is finished

finalImageName);

finalImageName);

finalImageName);

finalImageName);

long start = System.currentTimeMillis();
g.createGrains();
long end = System.currentTimeMillis();
System.out.println("That took " +

(double)(end-start)/60000 + " minutes.");

System.out.println(g.grains.size());
System.out.println("Reducing");
g.reduceGrains();
for(int count = 0; count < g.grains.size();

count++){

System.out.println(g.grains.get(count).scaledBlobs.size());

System.out.println(g.grains.get(count).scaledBlobs.get(0).pixels.get(0).x

+ ", " +

g.grains.get(count).scaledBlobs.get(0).pixels.get(0).y);

}
System.out.println(g.grains.size());
g.drawGrains(finalImageLocation,

finalImageName);

for(int count = 0; count < g.grains.size();

count++){

System.out.println(count + ", "

+ g.grains.get(count).scaledBlobs.size());

}
System.out.println(g.lines.get(0).slope);

```

```

System.out.println(g.lines.get(0).pixels.get(0).x + ", " + g.lines.get(0).pixels.get(0).y);
                        System.out.println(g.lines.get(1).slope);

System.out.println(g.lines.get(1).pixels.get(0).x + ", " + g.lines.get(1).pixels.get(0).y);
                        System.out.println(g.lines.get(2).slope);

System.out.println(g.lines.get(2).pixels.get(0).x + ", " + g.lines.get(2).pixels.get(0).y);
                        System.out.println(g.lines.get(3).slope);

System.out.println(g.lines.get(3).pixels.get(0).x + ", " + g.lines.get(3).pixels.get(0).y);
                        System.out.println(g.lines.get(4).slope);

System.out.println(g.lines.get(4).pixels.get(0).x + ", " + g.lines.get(4).pixels.get(0).y);*/
//}
    }
    });
}
return runButton;
}

/**
 * This method initializes perpPercentOutput
 *
 * @return javax.swing.JTextField
 */
private JTextField getPerpPercentOutput() {
    if (perpPercentOutput == null) {
        perpPercentOutput = new JTextField();
        perpPercentOutput.setBounds(new Rectangle(44, 421, 82, 21));
        perpPercentOutput.setText("No Input Here");
    }
    return perpPercentOutput;
}

/**
 * This method initializes parPercentOutput
 *
 * @return javax.swing.JTextField
 */
private JTextField getParPercentOutput() {
    if (parPercentOutput == null) {
        parPercentOutput = new JTextField();
        parPercentOutput.setBounds(new Rectangle(184, 421, 82, 21));
        parPercentOutput.setText("No Input Here");
    }
    return parPercentOutput;
}

/**
 * @param args
 */

```

```
public static void main(String[] args) {  
  
    AFMAnalysis main = new AFMAnalysis();  
    JFrame frame = main.getMainFrame();  
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    frame.setVisible(true);  
  
}  
  
} // @jve:decl-index=0:visual-constraint="355,-8"
```

Appendix B
PROPERTIES OF SI POLYMER

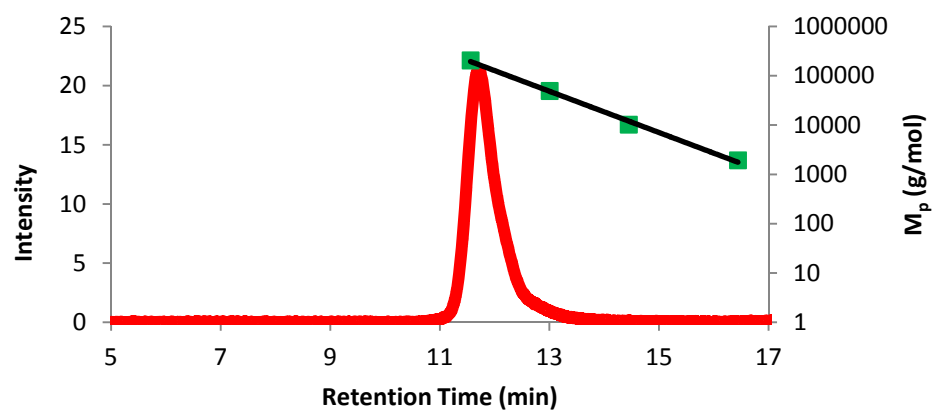


Figure B-1 GPC trace of SI polymer. Green points and trendline indicate calibration for molecular weight.

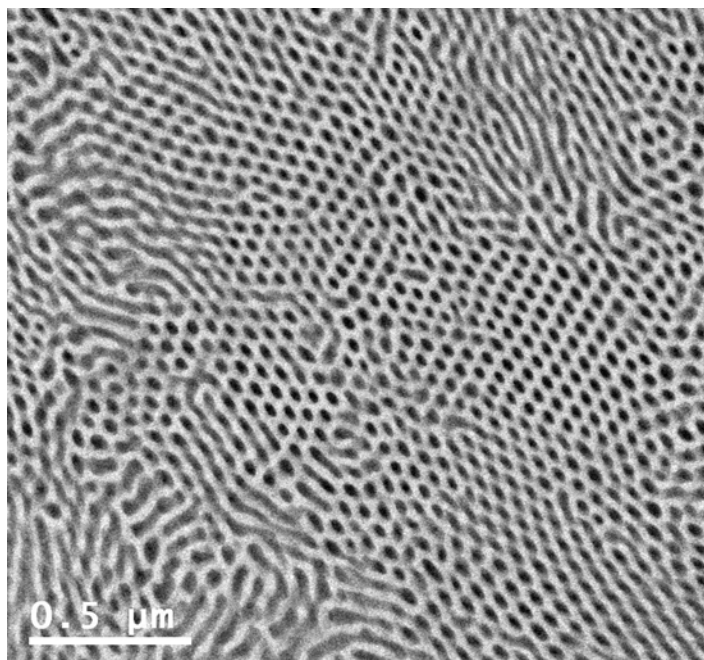


Figure B-2 Bulk morphology of SI polymer. The sample was cut using cryo microtoming, and the image was taken using transmission electron microscopy. Sample annealed for 8 hours at 180 °C.