DEVELOPMENT AND CHARACTERIZATION OF A NOVEL MICROFLUIDIC BIOREACTOR SYSTEM UTILIZED FOR EXAMINING HEMODYNAMIC EFFECTS ON CELLULAR RESPONSE

By

Lucas Hudson Hofmeister

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in Partial Fulfillment of the Requirements

for the Degree of

MASTER OF SCIENCE

in

Biomedical Engineering

August, 2013

Nashville, Tennessee

Approved

Professor Hak-Joon Sung

Professor Kevin Seale

To my parents, Mandy and William, who have given me every opportunity in life to succeed

and my family, Nanny, Becca, Richard, Matthew, Erich, and Charles who

have supported and loved me through all that I do.

ACKNOWLEDGMENTS

First and foremost, I would like to thank Dr. . Without his guidance, insight, and friendship this work would not be possible. I would also like to thank Dr. Jon Edd for his collaboration with our lab that made this project possible, for his oversight and guidance, and for providing the resources needed to complete this work. I would like to give a special recognition to Todd Lagus for his contribution to the bioreactor characterization and aid in designing and carrying out experiments. I would also like to thank Dr. David Bader and Dr. Elaine Shelton for their insight into the developmental mechanisms of omental mesothelium. Special recognition goes to Dr. Shelton for helping to design and performing surgeries and staining for omentum experiments. I would also like to give special thanks to Ronald Reiserer for his guidance and help troubleshooting not just during this project but in all of my scientific endeavors at Vanderbilt. I owe a very special thanks to Dr. Kevin Seale, whose friendship and guidance are a very important part of my studies. I would also like to thank Dr. John Wikswo for his support and guidance.

Additionally, I would like to thank my fellow researchers Mukesh Gupta, Xintong Wang, Angela Zachman, Shann Yu, Spencer Crowder, Sue Lee, Ricky Rath, Tim Boire, Chris Nelson, Martina Miteva, Kelsey Beavers, and Brian Evans for their help with experiments, day to day lab work, and support through the past three years. I would like to thank my undergraduate assistants, Chad Augusty, Ian Baird, and Noah Hoh their help in daily lab work and completing experiments.

Microfabrication was performed in the Vanderbilt Institute for Integrative Biosystems Research and Education (VIIBRE) cleanroom facilities. This work was supported by Vanderbilt Start-up and NSF CAREER 05-579.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Cardiovascular Tissue Engineering and Cell Sourcing

Cardiovascular disease is the number one cause of morbidity and mortality in the developed world, which establishes a huge need for therapeutic intervention. For example, the current leading technology for coronary artery disease is coronary artery bypass grafting, and 500,000 of these procedures are performed per annum in the United States alone [3]. Large vessel repair (>5mm) is generally permissive of non-cellular therapies, for example, synthetic vascular grafts such as polytetrafluoroethylene (PTFE) conduits. However, these approaches as well as arterial grafting have shown poor outcomes for small vessel therapy [2, 3]. The National Heart, Lung, and Blood Institute (NHLBI) is currently supporting around 14 million dollars of research directed at engineering small blood vessel substitutes. There are three main tissue engineering approaches for tissue engineered vascular grafts (TEV): (1) biodegradable synthetic polymer scaffolds, (2) decellularized matrices, and (3) cell sheet engineering. While significant advances have been made, there is still no widely accepted clinical treatment utilizing tissue engineered products [1, 2]. Moving forward, all tissue engineered constructs being utilized must take into account the dynamic mechanical environment in the vascular tree and how these forces effect the phenotype of all cell types used in the construct.

Cell sourcing is a significant issue for developing TEV. Human vasculature has a highly ordered structure including multiple cell types. This composition and function must be recapitulated by TEV. This means that the cell source for vascular constructs must be able to differentiate into endothelial cells, smooth muscle cells, and fibroblasts and take on the correct organization. This imposes the need for an autologous cell source capable of yielding large quantities of cells with multipotent differentiation capacity. In the past decade, approaches utilizing stem cells have been thoroughly explored. A comprehensive assessment of cell sourcing for TEVs was presented by Bajpai et. al. In spite of the advancements in stem cell therapy, there are still significant challenges associated with these approaches. For example, autologous multipotent stem cells (mesenchymal stem cells (MSCs), bone marrow derived mononuclear cells (BM-MNCs), and endothelial progenitor cells (EPCs)) have good differentiation capacity but are fairly rare (0.1-0.5% of bone marrow aspirate), often senescent in culture, and may not be active enough to repair vessels in elderly patients. Induced pluripotent stem cells can overcome some of these issues, however, they can take a long time to genetically re-program and raise significant concerns regarding their oncogenic potential and incomplete differentiation.

This thesis proposes that omental mesothelium is an alternative cell source for vascular repair by TEV. The omentum has been utilized for over a century to promote wound healing during surgical intervention in ischemic hearts, perforated gut tubes, damaged brains and spinal cords, for revascularization of lower and upper extremities, and for reconstruction of head and neck [4-12]. Importantly, the omental mesothelium plays numerous and important roles in development, but it has limited function in the adult. This makes it an ideal and abundant autologous cell source for transplantation. In development, the mesothelium originates from the pericardial cavity, undergoes epithelial to mesenchymal transformation and gives rise to cardiac fibroblasts, endothelial cells and smooth muscle of the coronary arteries, and the vasculature of the gut and lungs [13-16], [17, 18] [19] [20, 21]. In addition, recent studies by Shelton et al have demonstrated that grafting omental mesothelium over an injured carotid artery resulted in near complete recovery of the injured artery. They found that Tβ4, a naturally occurring blood protein can mobilize mesothelial graft cells to integrate into wounded blood vessels and repopulate the smooth muscle layers of the vessel, thereby significantly decreasing healing time **(Figure 1)** [22]. In spite of the promise of these cells, they still have the potential to undergo pathogenic differentiation. For example, mesothelium can be activated after injury to form painful, fibrous, vascularized and innervated serosal adhesions [15, 23, 24]. This makes it extremely important that we understand the physical and chemical cues that lead to the differentiation of this acrobatic cell type in order to purposefully drive it towards reparative phenotypes.

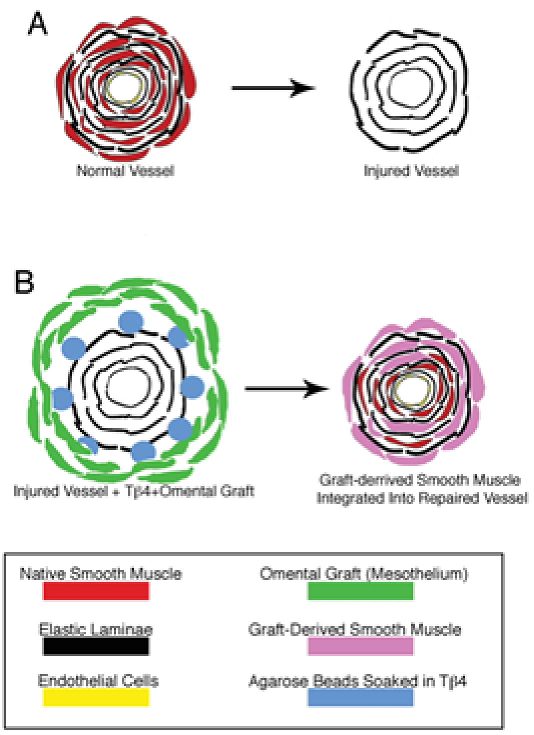

A
Normal Vessel
Injured Vessel
B
Injured Vessel + Tβ4+Omental Graft
Graft-derrived Smooth Muscle Integrated Into Repaired Vessel
Native Smooth Muscle
Omental Graft (Mesothelium)
Elastic Laminae
Graft-Derived Smooth Muscle
Endothelial Cells
Agarose Beads Soaked in Tβ4

Figure 1: Thymosin β4 reactivates adult mesothelial cells to aid in tissue repair. Thymosin β4 (Tβ4) can activate PKC and AKT signaling in adult epicardial and omental mesothelial cells. In epicardial cells, Tβ4 promotes proliferation, migration, differentiation into vasculogenic lineages, and endothelial tube formation all of which aid in repair of the myocardium following infarct. Similarly, in omental mesothelial cells, Tβ4 promotes proliferation, migration, and differentiation into vasculogenic lineages in order to accelerate the repair of injured blood vessels (Image courtesy of Shelton et. al.).

Mechanotransduction in Tissue Engineering

The effects of mechanotransduction need to be investigated for all cells involved in cardiovascular tissue engineering. It is becoming clearer that tissues that are cultured in static systems cannot fully recapitulate the function of their organ counterparts. In a few particularly appropriate examples, Huh et al demonstrate that adding mechanical stimulation to microphysiological models of the lung and gut result in recapitulation of organ level functions which have never before been observed *in vitro* [25-27]. There are also numerous bioreactors reported in the literature for investigating the formation of bone, cartilage, muscle, and cardiac tissue. Furthermore, numerous studies in recent years have demonstrated that mechanical stimuli are necessary for tissue engineering of functional tissues, and are particularly important in the case of vascular grafts [28, 29].

The human vasculature is one of the most mechanically dynamic systems in nature. During each cardiac cycle, complex patterns of pressure, flow, and shear stress propagate through the vascular tree and are sensed by endothelial cells, smooth muscle cells, and all other vascular cell types. It is well understood that these mechanical parameters are major determinants of vascular homeostasis and pathogenesis. Basic arterial hemodynamic forces are shear stress, circumferential strain, and normal stress from blood pressure. Typical human arterial values of shear stress, cyclic strain, and transmural pressure are 6 to 40 dyne•cm^{-2}, 2% to 18% at 1 Hz, and 60 to 110 mmHg, respectively [30]. Abnormal hemodynamic mechanical stimuli play a causative role in pathological vascular remodeling, including atherosclerosis, restenosis, hypertension, and stroke [31-34].

Approach

Bioreactors are an essential component of engineering cardiovascular tissues and studying relevant mechanotransduction. Cardiovascular bioreactors are uniquely complex because they require the simultaneous application of fluid shear stress and strain. In this thesis, we sought to engineer a microfluidic bioreactor system to study the effects of shear stress and strain on cultured tissues for use in vascular tissue engineering. One of the major requirements of investigating the role of mechanotransduction on cell fate is to use model systems that can supply highly controlled and repeatable mechanical stimulation to cultured tissues. There are many examples of bioreactors for studying tissue engineering and mechanotransduction in the literature, and many commercially available systems. However, there are numerous shortcomings with these systems. A key shortcoming that we wanted to address in this thesis was the effect of mechanical strain on the fluidic shear conditions in a bioreactor system. For example, when an elastomeric tube is stretched, the cross sectional dimensions change. If volumetric flow rate is held constant during the stretch, the shear conditions experienced by the cells must change. We sought to characterize this effect in our bioreactor system to better understand the mechanical environment experienced by the cells cultured in our device.

The bioreactor system described in this thesis consists of a microfluidic device for tissue culture of the cell type to be studied, two linear actuators to apply user-defined strain conditions, and a computer controlled syringe pump to apply user-defined shear conditions. We also describe novel methods based on particle image velocimetry that were used to characterize the mechanical environment in the microfluidic bioreactor.

CHAPTER II

DEVELOPMENT AND CHARACTERIZATION OF A NOVEL MICROFLUIDIC BIOREACTOR SYSTEM UTILIZED FOR EXAMINING HEMODYNAMIC EFFECTS ON CELLULAR RESPONSE

Introduction

Cell sourcing for tissue engineered approaches to vascular repair is a serious issue confronting the field of cardiovascular tissue engineering. Omental mesothelium is a promising autologous cell source for vascular repair and has been used for numerous other therapies [11]. Until recently, omental mesothelium was only thought to play a paracrine role in wound healing but there is increasing evidence that omental mesothelium can undergo divergent terminal differentiation to reparative vasculogenic cell types including: endothelial cells, fibroblasts, or vascular smooth muscle cells[15, 23, 24].

This study builds on the previous work by the Bader group that demonstrated that thymosin β4 can mobilize mesothelial graft cells to integrate into and repopulate the smooth muscle layers of arteries in an *in vivo* model of vessel damage [22]. *In vivo*, all vascular cell types are exposed to hemodynamic forces such as shear stress, circumferential strain, and normal stress from blood pressure. The role of hemodynamic forces on mesothelial potential is underexplored and their impact on cell differentiation is completely unknown. Hence, we hypothesized that smooth muscle and endothelial cell differentiation from mesothelia is regulated by hemodynamic parameters such as shear stress and strain. There is an unmet need to understand how these physicochemical parameters regulate mesothelium differentiation into these cell types.

To address this issue, we designed, developed and characterized a microfluidic bioreactor system and applied the system to study the effects of shear stress and strain on the differentiation of murine omental mesothelium. When strain is applied to a PDMS microfluidic device, the channel dimensions change. Consequently, if volumetric flow rate is kept constant through the microfluidic device, the fluid velocity and therefore wall shear change with applied strain. Therefore, in order to understand the mechanical forces that cells grown in the bioreactor were experiencing, we needed to fully characterize the bioreactor during dynamic mechanical stimulation of combined shear stress and strain. To characterize the bioreactor system, we developed a coupled numerical model of the bioreactor microfluidic dynamics under dynamic strain as well as novel characterization methods to simultaneously monitor the three dimensional fluid velocity profile, wall shear stress (WSS) and strain during mechanical stimulation.

Methods

Device Fabrication

Poly(dimethylsiloxane) (PDMS) microfluidic devices were fabricated by standard photo-lithography and soft lithography techniques [35]. Photomasks were designed using Auto-Cad and emulsion printed on Mylar (Fineline Imaging, Colorado Springs, CO). Printed photomasks were transferred to chrome on soda lime by contact printing using AZ9200 photoresist (AZ Electronic Materials, Somerville, NJ), and subsequent ceric ammonium nitrate with nitric acid etching to remove chrome from the exposed regions. To fabricate microfluidic master molds, an epoxy-based negative photoresist, SU-8 2100 (MicroChem, Newton, MA) was spun onto three inch diameter test grade silicon wafers using a spin protocol of 500 RPM for 15 seconds and 4000 RPM for 30 seconds (WS-400B-6NPP/LITE Benchtop Spin Coater, Laurell Technologies, North Wales, PA). After spinning, the wafers were soft baked at 65°C for 5 minutes (Dataplate721A, Barnstead International, Dubuque, IA) allowed to cool to room temperature, and then processed with edge bead remover (MicroChem) while spinning at 500 RPM. The photoresist coated silicon wafers were exposed at 160 J•cm^{-1} using a UV spot curing system with a 365 nm filter (Novacure 2100, EXFO, Vanier, Canada). Additional crosslinking was performed by a post exposure bake step at 95°C for 5 minutes. Wafers were cooled to room temperature and non-crosslinked SU-8 was removed by SU-8 developer (MicroChem). To improve adhesion of the photoepoxy to the wafer and remove cracks due to over-exposure, developing was followed by a 5 hour hard bake at 185°C (near the glass transition temperature of

SU-8). Finally, Photo epoxy height was measured using a profiliometer. This highly repeatable protocol yielded 110±1 µm photoepoxy thickness. X, Y resolution was limited by the mask used during photolithography and all features were well within the resolution limits of mylar masks.

All microfluidic devices were fabricated using PDMS with an initiator: base ratio of 1:10 (Dow Corning, Clarksville, TN) and cured overnight at 70 C. In order to achieve uniform and highly repeatable strain on microfluidic channels, unique microfabrication techniques needed to be developed. The major requirements for the bioreactor microfluidics were highly uniform and repeatable device thickness and integrated harnesses to apply cyclic strain to the device perpendicular to the microfluidic channel. Initial attempts at fabrication involved classical soft lithography. The major challenge with this approach was the harnessing method. As shown in figure 2 the device was assembled in PDMS and then bonded to four microscope slides by plasma bonding.

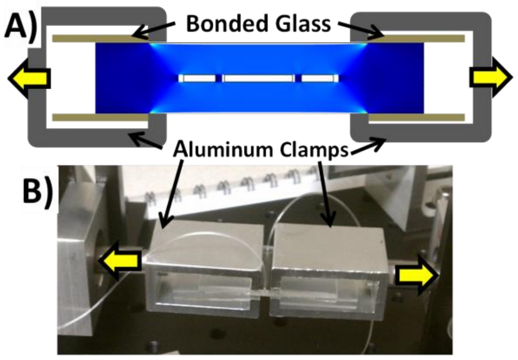


Figure 2. Glass slide approach to bioreactor harnessing. A) Shows a schematic of the device construction. B) Shows the assembled bioreactor system with glass slide harnessing strategy.

There were major issues with alignment of the glass slides to ensure they were perfectly parallel to the microfluidic channel. To overcome this issue, the glass slide harnessing methods was abandoned in favor of embedded fiber glass harnesses. 1 oz fiber glass mat (Saint Gobain Vetrotex America, Huntersville, NC) was embedded in the devices to provide structural rigidity for stretcher harnessing. Fiber glass mat was first infused with PDMS by dripping a thin layer onto the mat and curing at 70 C for 1 hour. This process allowed for easy handling of the fiber glass and reduced the production of particulates for handling in a clean room. The embedded mat was then cut to the shape of the stretcher harness using a steel rule die (Apple Steel Rule Die, Milwaukee, WI). These methods allowed us to quickly and easily fabricate stretcher harnesses with the same size, shape, and thickness. The design for the stretcher harness has one attachment point at the center of the harness (Figure 3). This eliminated possible alignment issues with the actuators and was shown to provide adequate mechanical stability for all experimental strains.

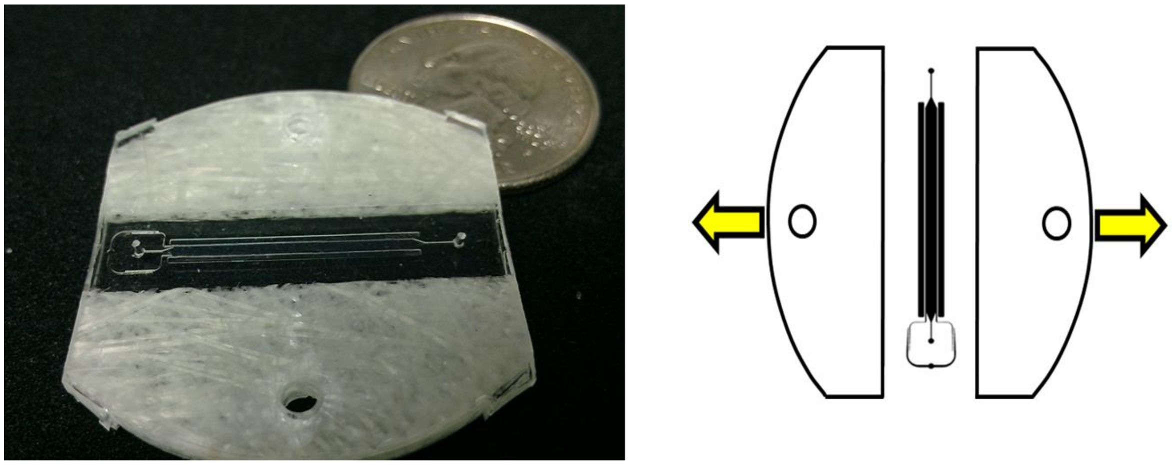


Figure 3. Microfluidic device design with integrated fiberglass harnesses. Yellow arrows indicate the direction of strain.

In order to address device thickness and uniformity, all microfluidic devices were cast using a sandwich and clamp method. To control the footprint and thickness of the device, 1/16 inch thick acrylonitrile butadiene styrene (ABS) sheet was machined to the device outer dimensions by CNC (TorMach 770, TorMach, Waunakee WI) and used as the middle layer of the sandwich. The ABS was aligned onto the photolithographic silicon master and PDMS was poured to fill the space. The device was de-gassed for 15 min at room temperature and then an optical quality glass slide was used to seal and clamp the device. The resulting device was then silanized with trimethyl(trifluoromethyl)silane (Sigma-Aldrich, St. Louis, MO) by plasma assisted chemical vapor deposition with and used to create a PDMS master mold with the exact shape and thickness required. The master mold was in turn silanized and all subsequent devices were manufactured from this device template, ensuring exact repeatability of device footprint and thickness. Fiber glass harnesses were integrated by simply placing them into the mold during fabrication. An un-patterned PDMS piece was made by the same methods and plasma bonded to the microfluidic PDMS part yielding fully sealed microfluidic devices with overall thickness of 1/8 of an inch.

Finite Element Modeling – Coupled Numerical Modeling

Numerical modeling was performed to predict the effect of microchannel strain on the fluid shear stress experienced by cells growing in the microfluidic channel. All finite element numerical modeling was conducted using Comsol Multiphysics version 4.2. In the most general situation, where the time-waveforms of mechanical strain and fluidic shear stress are to be independently controlled, it is absolutely imperative to have an accurate numerical model which captures the coupled solid and fluid physics behavior of a dynamically strained microfluidic channel. Using this type of numerical model, it is possible to create an automated control algorithm to convert desired mechanical strain and shear stress temporal profiles into separate digital control inputs for the stretcher and variable flow-rate syringe pump.

For the solid mechanics half of the problem, the use of poly(dimethylsiloxane), or PDMS (Sylgard 184; Dow Corning), as a substrate for the microfluidic bioreactor allowed for simplifying assumptions. First, well-mixed PDMS is characterized by a linear elastic and isotropic constitutive model, at least below ~40% strain [36]. Second, its relevant mechanical properties may be assumed homogeneous across any given device, where the Young's modulus (E) is about 1.8 MPa [37] and the Poisson ratio (v) is about 0.45 [38]. If we make the further assumptions that stretching is slow (neglect PDMS acceleration) and that gravitational body forces are negligible (compared to stress), we obtain a simplified set of the Navier-Cauchy equations (Equation 1) where the bulk modulus (λ) ≈ 5.6 MPa and the shear modulus (μ) ≈ 0.62 MPa are the so-called Lamé parameters. By then specifying strain boundary conditions for an otherwise free PDMS surface, as constrained by the strain harness (described in device fabrication) the spatially-varying displacement vector (u) was found numerically in a stepwise fashion for a full cycle of stretching up to 20% strain. The total strain tensor (ε) at the microchannel surface was determined from Equation 2. In order to account for possible non-linearity hyperplasticity in the behavior of the PDMS at high strain, and considering the previous assumptions, the Mooney-Rivlin neo-hookean hyperplastic constitutive model was employed when modeled in Comsol.

$$\boldsymbol{Eq. 1.} \quad (\lambda + \mu)\nabla(\nabla \cdot u) + \mu\nabla^2 u = 0 \quad where\ \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \quad and\ \mu = \frac{E}{2(1+\nu)}$$

$$\boldsymbol{Eq. 2.} \quad \epsilon = \frac{1}{2}(\nabla u + \nabla u^T)$$

To solve the fluid flow problem, we assumed that fluid pressure was insignificant compared to the rigidity of the microfluidic device, and therefore the results of the solid mechanics problem were used to set the boundary conditions for the fluid mechanics problem. This would fail to be accurate in the case where significant pressure drops occur within or downstream of the microfluidic channel. However, the bioreactor system is designed with a closed loop push-pull syringe pump so this assumption is considered accurate. We therefore combined the flow rate as a function of time as applied by the syringe pump with the static solutions to the solid mechanics problem as a sequence of deformed wall shapes corresponding with the stretcher time-history in order to achieve a complete set of time-step boundary conditions to describe a full cycle. These deformed wall shapes were supplied as boundary conditions in a stepwise manner to solve the Navier-Stokes equations. For these solutions it was assumed that the cell culture media used behaves as a homogeneous incompressible Newtonian fluid with zero buoyant forces (Equation 3) where μ is the dynamic viscosity in this case (0.6904 mPa•s for water at 37 °C), ρ is mass density (1000 kg/m^3 for normal saline at 37 °C), t is for time, and ν is the spatially-varying fluid velocity vector. The first derivative of v was used to find the shear stress at the microchannel surface during a full cycle of stretching.

$$\boldsymbol{Eq. 3.} \quad \mu\nabla^2 v - \nabla P = \rho\left(\frac{\delta v}{\delta t} + v \cdot \nabla v\right)$$

These models (Eqs. 1-3) were used to optimize device geometry i) to ensure the goal of uniform strain across the microchannel surface where cells were cultured; and ii) to determine the effect of microchannel strain on the shear stress experienced by cultured cells. These coupled numerical models were validated experimentally as described in the following section.

Bioreactor Design and Control

The microfluidic bioreactor is built around a PDMS microfluidic device with a channel 0.1 mm tall by 1 mm wide by 25.4 mm long. A low aspect ratio microfluidic channel was chosen to ensure maximum uniformity of strain across the entire culture surface. Optimization of the design was performed during finite element modeling and is discussed further in the result section.

In order to control the strain dynamics on the microfluidic device, a design integrating two opposing micro-positioning linear actuators was chosen. The opposing linear actuators allows for visualization of the microfluidic channel during dynamic stretching provided that the actuators work in concert with each other. A syringe pump was chosen to provide fluid flow for the bioreactor system. The entire bioreactor is designed around a compact optical breadboard with a footprint that fits in a standard cell culture incubator or on a microscope stage (Figure 4).

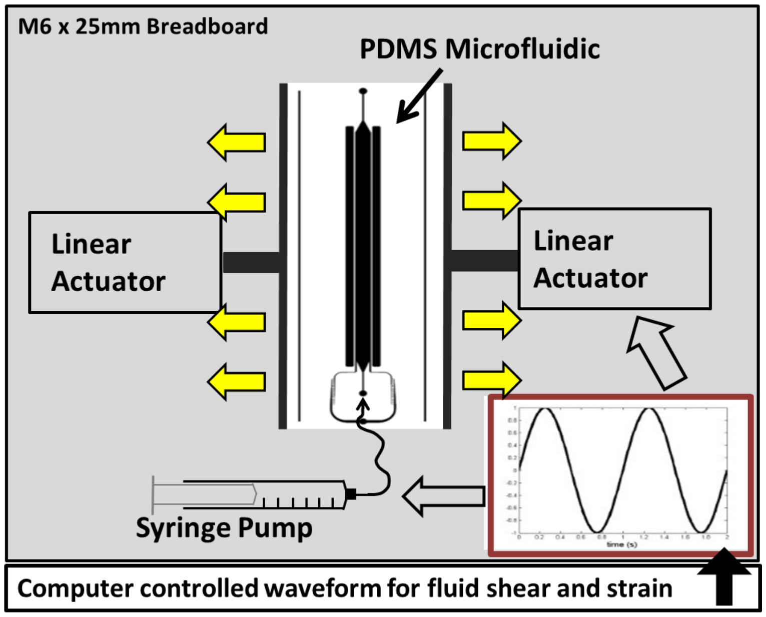


Figure 4. Top view schematic of the bioreactor system design. Yellow arrows indicate the direction of strain.

The design constraints for the linear actuators included the precision and high forces required to stretch a 1/8 inch microfluidic device repeatedly. An approximation of the forces required is shown in supplementary figure 1 (Appendix A). The actuators used in the bioreactor system are Physik Instrumente M-235.5DD high power direct drive DC actuators with a ballscrew driving mechanism

(Physik Instrumente, GmbH). The actuators have a total travel length of 50mm. Ballscrew actuators were chosen to reduce the backlash when performing cyclic motion. Direct drive DC driven linear actuators were chosen to achieve the high speeds and accelerations necessary for 1 Hz cyclic motion and produce smooth displacement curves. In addition, these actuators provide forces of over 120 N and incremental motion of 0.5 um with 0.5 um repeatability. The actuators are equipped with rotary encoders for feedback during the control process. Control of the bioreactor system is achieved through a PCI bus card integrated into the control computer and using Labview. Labview drivers provided by the company for control needed to be modified in order to control the actuators for continuous cyclic motion over extended times. A screenshot of the control software written in Labview is shown in supplementary figure 2. Before bioreactor operation, the actuators are first referenced to non-contact hall-effect limit switches and then moved into position to load the microfluidics to the system. A MATLAB program is used to write the motion profile instructions for the actuators and is capable of writing any arbitrary waveform to a set of motion instructions for the actuators. The motion profile code can be reviewed in appendix B.

Fluid flow is provided to the bioreactor system through a PHD-Ultra push-pull high precision syringe pump (Harvard Apparatus, Holliston, MA). Computer control software for the syringe pump was written using Labview (Supplementary Figure 3). To achieve a physiological arterial shear stress in our microfluidic bioreactor, a high flow rate was achieved (150 ul•min^{-1} for a wall shear stress of 1.5 dPa). The specific syringe pump used was chosen because it could supply the high flow rates necessary as well as precise control over the flow waveform.

The bioreactor was also designed to be flexible enough to use for multiple applications. For example, it is currently being utilized with a 12 well plate format PDMS well for culturing fetal rat cardiomyocytes and induced pluripotent stem cells under dynamic mechanical stimulation. In addition, the bioreactor is re-configured to double the throughput when imaging is not required for the experiment (Figure 7C).

Bioreactor Characterization

When you stretch a microfluidic channel the channel dimensions change due to the elastic properties of PDMS. When flow was supplied to the microfluidic bioreactor at a constant volumetric flow rate as defined by the syringe pump input, the shear stress experienced by the cells changed with the strain state of the bioreactor. In order to fully understand the mechanical environment experienced by cells cultured in the bioreactor device we developed characterization methods to simultaneously measure the WSS and strain during dynamic mechanical stimulation. Measurements of WSS were accomplished by monitoring the three dimensional fluid velocity profile of the channel; and WSS was calculated by taking the gradient of the downstream fluid velocity (u) at the channel walls. To measure the three dimensional fluid velocity profile, the device is tilted so that the entire cross section of the channel can be imaged in a single frame of video (Figure 5). Devices were made with small markers periodically spaced along the channel wall such that the tilt angle of the device could be measured by focusing on different markers along the length of the channel.

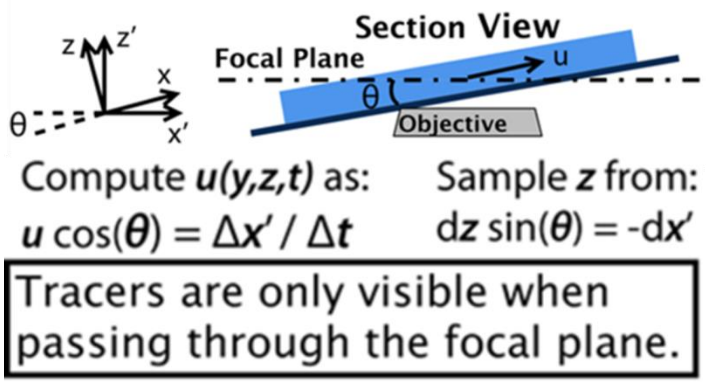


Figure 5. Bioreactor characterization schematic. The bioreactor is tilted on the microscope stage so that the entire velocity profile of the channel can be imaged in one frame. The downstream velocity (u) is then calculated using the tilt angle of the device and its location in x′ when it is in maximum focus.

After measuring the angle of the device with respect to the focal plane of the microscope objective, cyclic stretch and fluid flow were started. A flow rate of 147 µL•min^{-1} was used for characterization experiments because finite element modeling showed this flow rate would supply a physiologically relevant arterial WSS of 1.5 dPa. A physiologically relevant arterial cyclic strain of 10 % at a frequency of 1 Hz was used for all characterization experiments. Polystyrene beads (3 µm in diameter) were used as tracer particles in the fluid because they provided good contrast for light microscopy and are nearly neutrally buoyant in water. This allowed us to ignore buoyant forces when calculating the fluid streamline velocity in the channel. The devices were imaged using a Phantom v310 high speed camera at a frame rate of 3200 fps (Vision Research, Wayne, NJ). Videos were captured for 3 full cycles of strain and exported as AVI files for particle image velocimetry analysis.

Particle image velocimetry (PIV) software was developed in both Labview and MATLAB, however, final analysis was done using the MATLAB software. The software was developed by Todd Lagus in the Edd group and utilizes functions written by John C. Crocker, and was edited by Lucas Hofmeister for its final form. The tracking software is freeware available for download from http://www.physics.emory.edu/~weeks/idl/. The software consists of several different functions which are all called by the main tracking program (Appendix C). The first function, "ChannelWidthMain," (Appendix D) monitored the location of the edges of the channel in order to track strain during the experiment. The channel width function segmented each frame so that each channel wall is a single object and then measured the separation of the two objects to define channel width. The second function, "centroidmain," (Appendix E) took each frame from a tracking video and processed it by thresholding, finding particles, and assigning them shape descriptors. Thresholding was performed by Otsu's method or by manually assigning a threshold value. Next, the program identified the outlines of the particles and assigned shape descriptors. The most useful shape descriptor in this case is the eccentricity of the particles. To calculate eccentricity of the particles, the function assigned an ellipse that had the same second moments as the particle outline. Eccentricity is then defined as the ratio of the distance between the foci of the ellipse and its major axis length. A circle has an eccentricity of 0 and a line segment has an eccentricity of 1. The eccentricity measure is used to determine when each particle passes through its point of maximum focus. This information is in turn used to identify the particle's z coordinate in the microfluidic channel. This information was fed into the "track," program (Appendix F) where individual particle locations from each frame were identified and sorted into trajectories. The track software was programmed to look for particles that move a maximum displacement of 1.5 times the theoretical maximum displacement of the particles estimated from channel dimensions and average velocity approximated from volumetric flow rate. This ensured that the software was not jumping to a new particle between frames and reduced computation time. Particles were also tracked if they were not lost for more than 2 frames during the PIV video and if they existed for at least 10 frames. After identifying the trajectories of the particles, the software identified the local minimum and maximum of the eccentricity of each particle. The program then filtered the eccentricity to remove bad tracking data and then assigned each remaining particle a frame where it was in maximum focus by its minimum eccentricity (where it appears most circular). Figure 6 shows a frame of video with a distribution of particles in and out of focus. The particles were less distorted when they were in the center of the focal plane of the objective and therefore maximum focus was defined as the minimum eccentricity (=0) of each particle. Next the software excluded particles that showed large jumps in velocity. Following tracking and filtering to remove bad data, each tracked particle trajectory was assigned to a location in the channel by mapping the x' location of its maximum focus to the particle's location in the z dimension as shown in Figure 5 above. The data was then fit to a parabola for each frame to calculate wall shear stress.

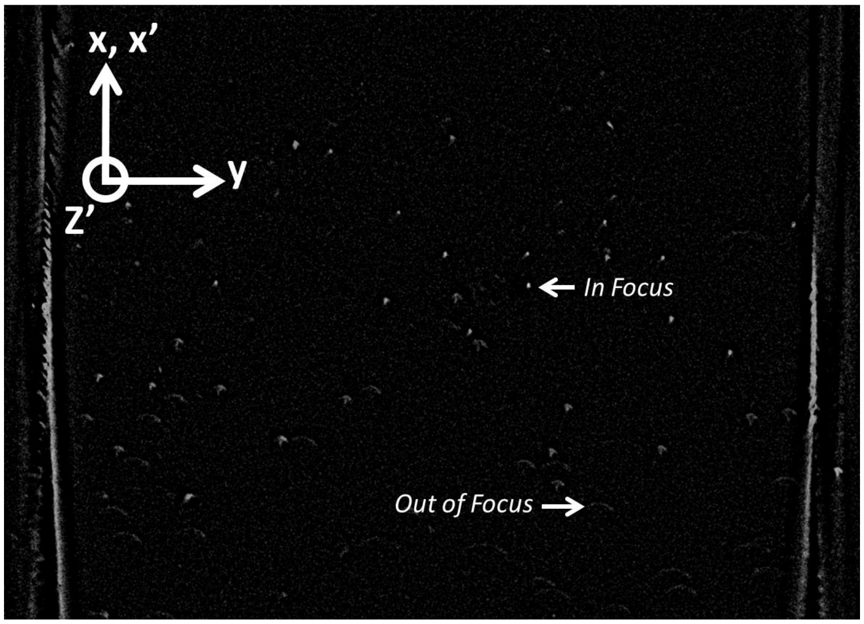


Figure 6. Representative image from PIV acquisition. The beads are round and clear when they pass through the center of the focal plane and distorted when they are out of focus. This characteristic is measured by eccentricity in the PIV software.

Strain characterization was also performed by PIV analysis. Tracking the location of the channel walls during shear stress measurements provided a baseline measure for the strain dynamics of the microfluidics. In order to obtain a more detailed picture of the strain characteristics of the bioreactor device, 10 µm fluorescent polystyrene beads were embedded in a bioreactor microfluidic. Fluorescent beads were dried out of aqueous solution and re-suspended in PDMS curing agent. Bioreactor devices were then made by the same protocol described above with microparticle containing curing agent. The devices were then imaged parallel to the focal plane of the microscope. PIV was used to track the fluorescent beads during 3 cycles of 10% cyclic strain. Bead displacements were plotted using MATLAB to visualize the strain in the device (Appendix G). Because the data points from the embedded beads were relatively sparse in the x,y plane, the data was interpolated using a Delaunay triangulation to produce a surface plot of strain for each frame of the video.

Omental Mesothelium Culture and Cell Assays

Prior to omentum experiments, human umbilical vein endothelial cells and human aortic endothelial cells were cultured in the microfluidic bioreactor system in order to develop culture conditions and test cell viability in bioreactor devices. Endothelial cells were cultured in Endothelial Growth Medium 2 (Lonza, Allendale, NJ) to confluence in 75 cm^2 flasks prior to being trypsin digested, pelleted, and seeded into bioreactor devices. Cells were cultured for 1-4 days and then stained with 5 µM Calcein AM (Sigma Aldrich, St. Louis, MO).

Omental mesothelium was isolated from adult outbred ICR, mice as previously reported [39] and cultured in supplemented medium [(10% fetal bovine serum (FBS), 1% penicillin/streptomycin, Dulbecco’s modified Eagle’s medium (DMEM)]. For each experiment, four mice were sacrificed by CO_2 asphyxiation and the stomach, spleen, and pancreas organ complex with associated omentum was isolated from the peritoneal cavity and placed in DMEM for dissection. The omentum was further separated from the associated organs and fat was trimmed from the thin transparent omental tissue. Afterwards the isolated omentum was diced with scissors, spun down at 1200 RPM for 5 minutes, and re-suspended in 10 µL media per device being seeded. Microfluidic devices were prepared by coating with 10 $\mu g \bullet mL^{-1}$ fibronectin in PBS without Ca^{2+} and Mg^{2+} at room temperature for at least 1 hour before seeding. Before seeding, the devices were washed with PBS and then culture medium. It was extremely important during these steps to maintain the devices at physiological temperature and use culture medium that was equilibrated for several hours with the atmosphere inside the culture incubator. This helped to maintain the pH of the culture medium and prevented bubble formation in the microfluidic channel. Omental cells were cultured for 48 hours prior to applying mechanical stimulation. Omental cells were exposed to 10% cyclic strain, 10% static strain, or 5 dynes/cm^2 WSS for 48 hours. Due to the nature of microfluidic cell culture, some flow rate is required to maintain the culture in the device. The low flow rate case (<0.01 dPA) was compared to static culture in microfluidics and in 4 chamber Lab-Tek glass chamber slides (Electron Microscopy Sciences, Hatfield, PA). During the culture period, the microfluidic channels were fed by either a syringe pump (PicoPlus, Harvard Apparatus, Holliston, MA) ("active pumping") or with micropipette barrier tips filled with media ("passive pumping") [40]. Using these techniques a low flow rate could be supplied to maintain cultures without having a significant effect on cell phenotype. Active pumping was used to apply shear stress conditions with flow rates determined by modeling and device characterization.

For immunohistochemistry analysis, cells were fixed with 4% paraformaldehyde in PBS for 30 min at room temperature then blocked with 10% goat serum in PBS -/- with 0.1% Tween20 for an additional 30 minutes at room temperature. Wilms tumor protein-1 (WT1) was used as a mesothelial marker to determine if omental mesothelium cells had differentiated away from their mesothelial lineage. Two markers of smooth muscle lineage were investigated. α smooth muscle actin (αSMA) was used as an early smooth muscle marker and smooth muscle myosin heavy chain (smMHC) was used as an mature smooth muscle marker. Anti-WT1 primary antibodies raised in mouse were diluted 1:50 in blocking solution (05-753, Millipore). Anti-αSMA primary antibodies raised in rabbit were diluted 1:250 in blocking solution (A2547, Sigma). Anti-smMHC antibodies raised in rabbit were diluted 1:100 in blocking solution (bt-562, Biomedical Technologies). Primary antibodies were incubated overnight at 4 C. After rinsing, secondary antibodies were diluted 1:2000 in blocking solution with 1:10000 hoechst as a nuclear counter stain and incubated for 3 hours at room temperature. Secondary antibodies used were Alexa Fluor 488 conjugated goat anti-rabbit IgG (Invitrogen, A-11008) and Alexa Fluor 568 conjugated goat anti-mouse IgG (Invitrogen, A-11004). Devices were then rinsed and stored in PBS for imaging. Imaging was performed using a Nikon inverted microscope (Nikon, Melville, NY).

Statistical Analysis

Statistical analysis was performed with one-way ANOVA followed by a Tukey test to compare experimental groups. Analyses were done with Minitab 16 software (State College, PA) or Microsoft Excel. Statistical significance was accepted within a 95% confidence limit. Results are presented as arithmetic mean ± SEM graphically.

Results

Bioreactor Design and Fabrication

The finalized form of the mechanotransduction bioreactor in both single and double stretching configurations is shown in Figure 7. An acrylic housing was added to the bioreactor design in order to further humidify the bioreactor environment. This modification was made to account for device drying due to the permeability of PDMS to water and the extremely high surface to volume ratio of the microfluidic channel.

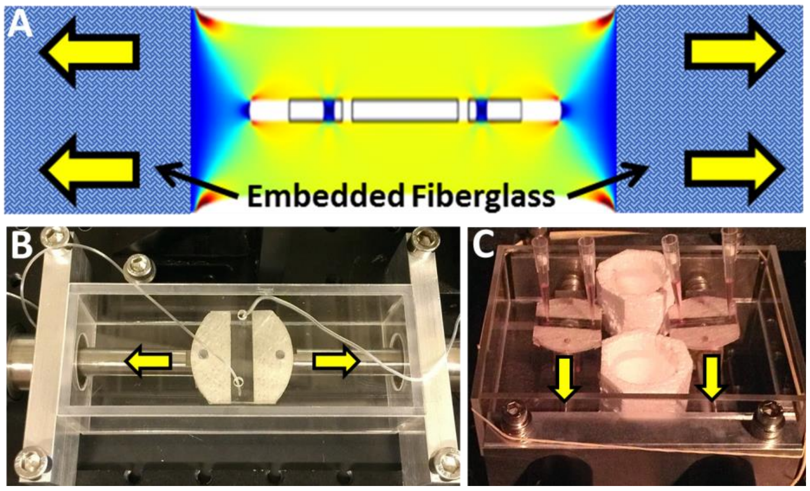


Figure 7. Mechanotransduction bioreactor assembly. The finalized mechanotransduction bioreactor is shown as A) a cross-section schematic of the microfluidic channel with embedded fiberglass strain harnesses B) Single device mode with syringe pump input and opposing actuators to allow for real-time visualization during strain and C) double device mode to increase throughput with micropipette feeding system.

Bioreactor Characterization

Numerical modeling demonstrated that a single monolithic channel in a block of PDMS will bow when strain is applied, resulting in non-uniform strain over the culture area (Figure 8A). To correct for the nonuniformity in strain, side channels were added to the device in plane with and parallel to the main channel. The addition of side channels concentrated stress outside the main channel and resulted in uniform strain over the culture surface (Fig 8B).

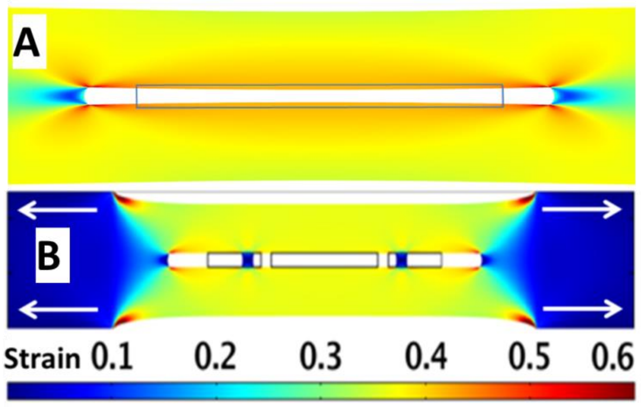


Figure 8. Neo-Hookean hyperplastic finite element model of microfluidic bioreactors shown in an x section view A) The behavior of a device with no side channels is shown. Strain is non-uniform across the channel surface and the channel walls bow inward. B) The addition of side channels corrects the bowing and results in highly uniform strain across the channel surface. Arrows indicate the direction of strain. The channel profile before stretching is shown as outlines on both images.

The modeled fluid velocity profile demonstrated uniform shear stress on >90% of the bioreactor channel (Fig 9A). Modeling also predicted that WSS in the bioreactor would change with strain and a constant volumetric flow rate (Fig 9B). The coupled numerical model predicted that flow rate would need to change by around 8.6 µL•min^{-1} to maintain a sheer stress of 1 dPa in a bioreactor device subjected 10% cyclic strain superimposed on 20% static strain. The change in flow rate was predicted to be in phase with the change in strain, and decreased strain resulted in decreased flow rate to maintain a constant shear stress. Modeling predicted a flow rate of 147 µL•min^{-1} was to achieve a physiological shear stress WSS of 1.5 dPa for an un-strained microfluidic bioreactor device.

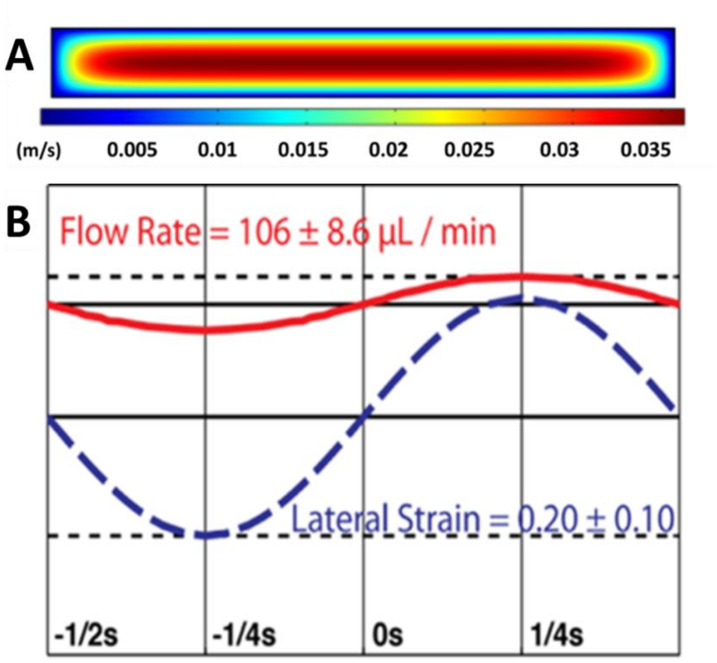


Figure 9. Finite element modeling of flow rate in a microfluidic bioreactor device. A) Shows the velocity distribution in a microfluidic device yielding a shear stress of 1.5 dPa B) Shows the relationship between strain and flow rate. A change in flow rate of around 8% of the total volumetric flow rate is required to maintain a constant shear stress under 10% cyclic strain conditions.

The three dimensional velocity profile during a single frame of PIV video confirms that the velocity profile in the channel matches that of the modeled case for no strain (Fig 10A). In addition, the dynamic PIV results show that the shear stress changes as predicted by the model in the case of 1 Hz 10% cyclic strain (Fig 10B).

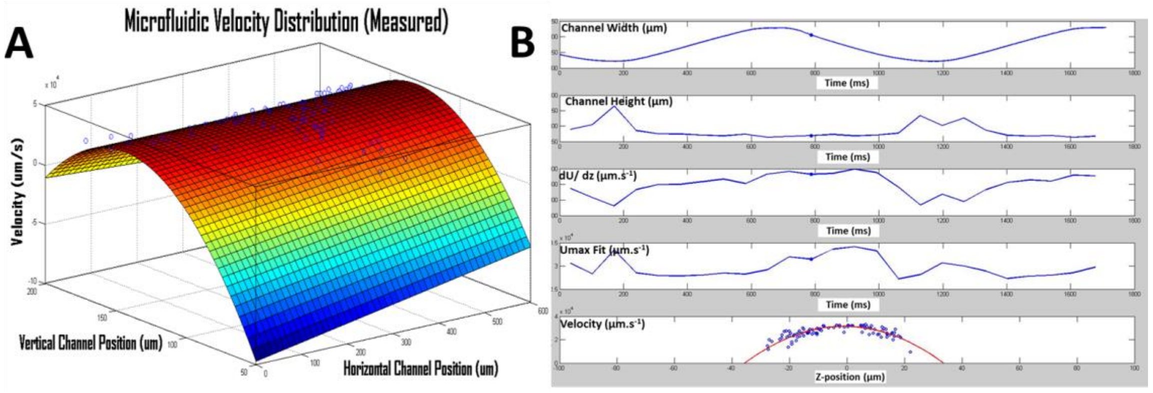


Figure 10. Bioreactor Characterization. A) Shows a parabolic fit of the measured streamline velocities in a microfluidic device during dynamic strain. B) Shows the changes in channel width, channel height, shear stress, maximum streamline velocity, and the data points used to calculate shear and maximum velocity during one frame of the PIV under 10% strain.

The strain on the microfluidic bioreactor was also found to be very uniform across the entire culture surface, as indicated by finite element modeling and PIV tracking of embedded fluorescent microparticles. This data also indicates that the opposing actuator system results in very little movement of the device during cyclic strain.

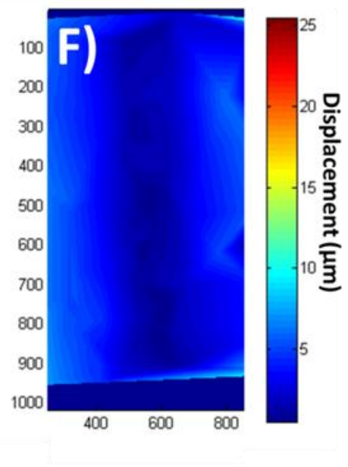


Figure 11. Strain characterization of a microfluidic bioreactor at 10% strain.

In Vitro Studies

Endothelial cells remained viable in microfluidic devices for up to a week as measured by calcein AM staining (Figure 12). Interestingly, endothelial cells were also found to adhere to all surfaces of the microfluidic bioreactor. This is confirmed by imaging a cross section of the bioreactor device with phalloidin stained endothelial cells.

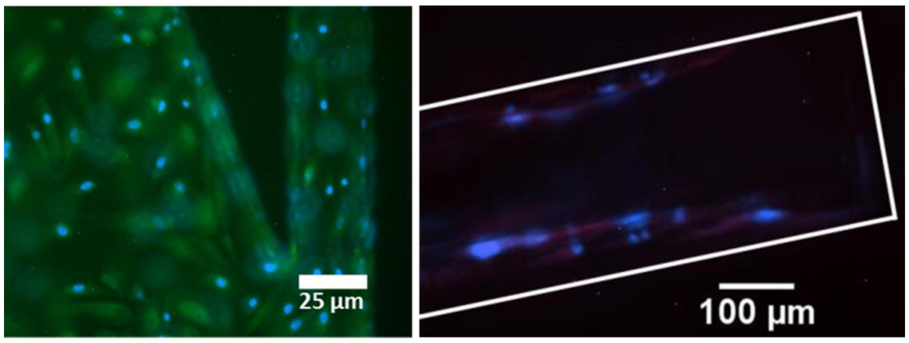


Figure 12. Endothelial cell culture in the microfluidic bioreactor system. A) Cell viability is confirmed by calcein AM staining. B) Phalloidin stained ECs can be seen adhering to all surfaces of the bioreactor device in a cross sectional view.

Exposure to WSS of 5 dynes.cm^{-2} was shown to decrease expression of alpha smooth muscle actin (αSMA, early smooth muscle marker) compared to 0.1 dynes.cm^{-2} WSS, without affecting expression of the mesothelial marker Wilm's tumor protein 1 (Wt1) (Figure 13A,B). 10% cyclic strain was found to increase the number of cells expressing the mature smooth muscle marker, smooth muscle myosin heavy chain and decrease the expression of Wt1 compared to no strain conditions and also induce alignment of αSMA fibers (Figure 13C). A comparison of low shear stress vs. static controls shows no statistical difference in expression of Wt1 or αSMA (Fig. 13D).

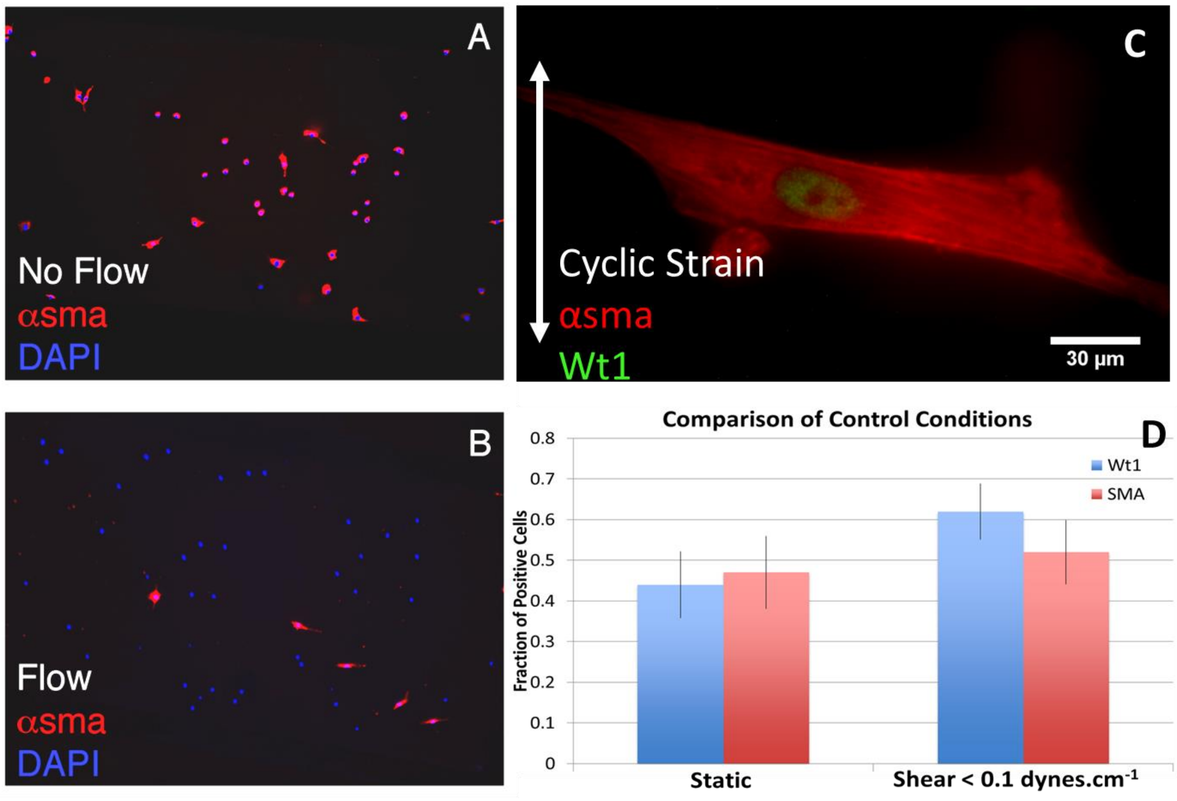


Figure 13. Effects of mechanical stimuli on omental mesothelium differentiation. A) Omental mesothelium under static WSS conditions in a microfluidic device. A majority of cells were shown to express αSMA. B) Omental mesothelium exposed to 0.5 dPa WSS was shown to significantly decrease αSMA expression compared to the static condition. C) Omental mesothelium exposed to 10% cyclic strain for 48 hours has aligned αSMA stress fibers and a low expression of WT1. D) comparison of control conditions of static culture and low shear <0.1 dPA shows no statistical difference between the low flow and static conditions.

Discussion

Understanding the regulatory role of the hemodynamic forces, WSS, and strain in vascular development from an embryonic origin is important to address a long standing issue of cell sourcing for vascular repair in the field of tissue engineering. Here we developed a bioreactor system to study how hemodynamic forces affect the differentiation of omental mesothelium.

The novel characterization techniques developed in this study allowed for the simultaneous characterization of WSS and strain, and confirmed the numerical models used to inform the device design. Side channels were added to the bioreactor design to eliminate non-uniformities in strain and bowing of channel walls. Importantly, we showed that with a constant volumetric flow rate WSS changes during stretching due to changes in channel dimensions. This analysis demonstrated that flow rate decreases with strain because the change in channel height is less significant than the change in channel width and therefore cross sectional area changes during strain. The same trends were observed from both modeling and PIV characterization, so we believe the coupled numerical model to be an accurate representation of the conditions in the microfluidic bioreactor.

We found that WSS suppressed the differentiation of omental mesothelium to smooth muscle lineages. It may follow that these cells are being pushed more in the direction of endothelial lineages, but additional experiments need to be conducted to confirm this speculation. For example, a stain for platelet endothelial cell adhesion molecule or Von Willibrand factor would elucidate differentiation to endothelial lineages.

In addition, 10% cyclic strain increased expression of early and late smooth muscle cell markers under low shear conditions, indicating that cyclic strain induces the differentiation of omental mesothelium toward smooth muscle lineages. Alignment of actin stress fibers is a key component of smooth muscle cell mechanotransduction and could represent an early transition event towards terminal differentiation of these cells to a smooth muscle lineage [41].

Conclusion

This thesis designed, developed and validated a microfluidic bioreactor system which can accurately recapitulate the major hemodynamic mechanical parameters of shear stress and strain. Furthermore, we characterized the effects of applying cyclic strain on the shear stress experienced by cultured tissues. This information will help us better understand the effects that each parameter has on the cells being studied. In addition, we developed the novel characterization techniques that allow us to visualize the three dimensional velocity profiles in a microfluidic channel during dynamic strain. This technique validated our models of bioreactor behavior, and could also prove useful for other studies involving microfluidic fluid dynamics. This study will facilitate the use of omental mesothelium as a cell source for tissue-engineered approaches to vascular repair.

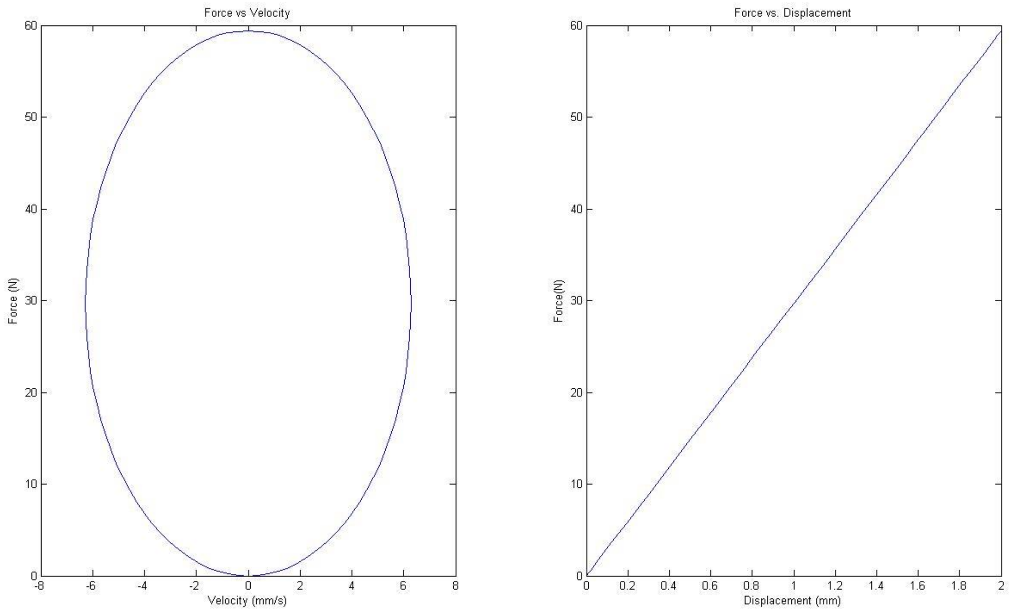
CHAPTER III

ONGOING AND FUTURE WORK

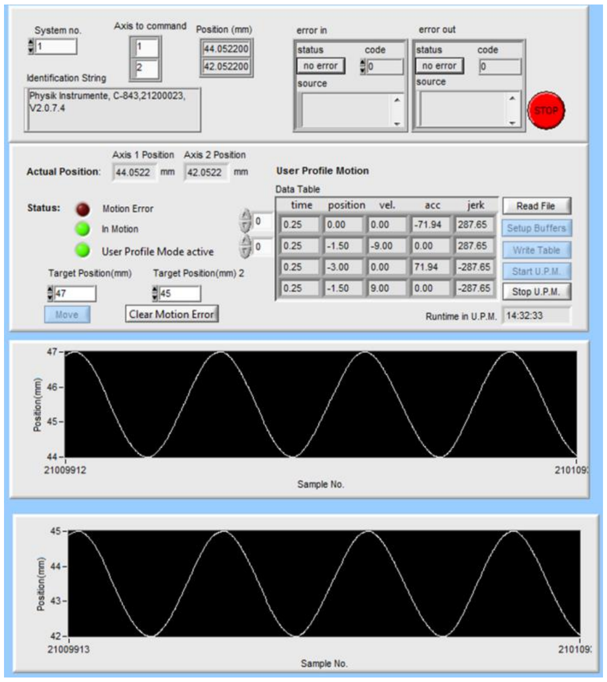
Future experiments will focus on using this information to decouple shear stress and strain by adjusting volumetric flow rate to compensate for changes in WSS in real time. We will investigate the effects of hemodynamic mechanical stimulation of the differentiation of omental mesothelium towards endothelial and fibroblast lineages.

In addition to these experiments, the bioreactor system developed is flexible enough to be tailored to numerous different studies revolving around mechanotransduction. The system is already being applied by other students in the lab to study the effects of mechanical stimuli on mesenchymal stem cell differentiation to endothelial lineages as well as cardiac lineages. The system was adapted to work with a 12 well plate format PDMS devices and is being applied in this form to study the effects of mechanical stimuli on induced pluripotent stem cells.

APPENDIX A



Supplementary Figure 1. Force vs velocity and force vs. displacement approximations for the mechanotransduction bioreactor. These approximations were used to determine the type of actuator that could be used for the bioreactor system.


System no.: 1
Axis to command: 1, 2
Position (mm): 44.052200, 42.052200
Identification String: Physik Instrumente, C-843,21200023, V2.0.7.4
error in — status: no error; code: 0; source:
error out — status: no error; code: 0; source:

Actual Position: Axis 1 Position 44.0522 mm; Axis 2 Position 42.0522 mm

Status: Motion Error; In Motion; User Profile Mode active

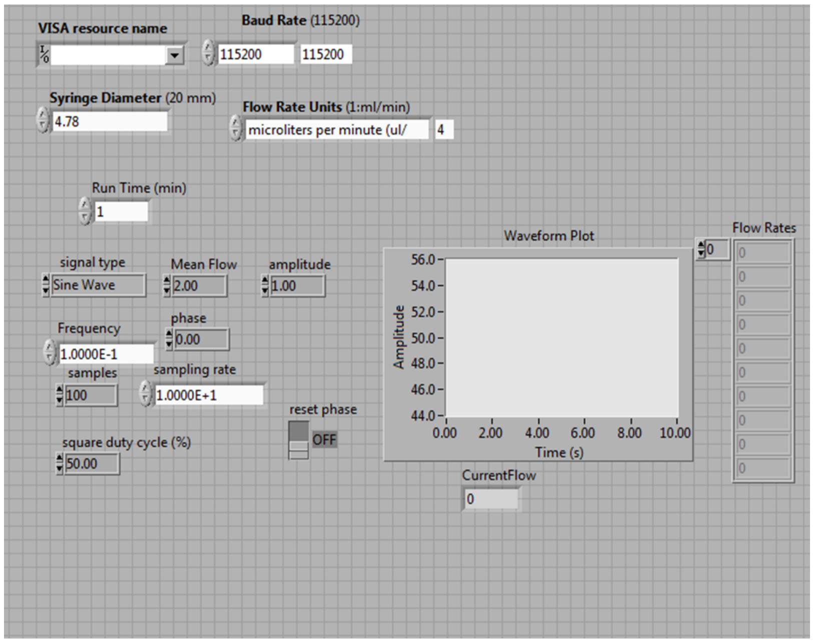
Target Position(mm): 47
Target Position(mm) 2: 45

User Profile Motion
Data Table

time	position	vel.	acc	jerk
0.25	0.00	0.00	-71.94	287.65
0.25	-1.50	-9.00	0.00	287.65
0.25	-3.00	0.00	71.94	-287.65
0.25	-1.50	9.00	0.00	-287.65

Runtime in U.P.M.: 14:32:33

Supplementary Figure 2. Actuator control software screenshot.


VISA resource name

Baud Rate (115200) 115200 115200

Syringe Diameter (20 mm) 4.78

Flow Rate Units (1:ml/min) microliters per minute (ul/ 4

Run Time (min) 1

signal type Sine Wave

Mean Flow 2.00

amplitude 1.00

Frequency 1.0000E-1

phase 0.00

samples 100

sampling rate 1.0000E+1

reset phase OFF

square duty cycle (%) 50.00

Flow Rates 0

0
0
0
0
0
0
0
0
0
0

CurrentFlow 0

Supplementary Figure 3. Syringe pump control software screenshot.

APPENDIX B

USER PROFILE CODE FOR ACTUATOR CONTROL

```
loadlibrary('GCSArrayIODLL');
% Macro to produce Spline Interpolation data for external Profile mode
% for C-843
%%
duration = 0.5; %duration in min

for MaxAmp = -0.0625
    % MaxAmp = 1;
    for FreqLow = 1
        %          FreqLow = 2;% Hz
        MaxAcc = 100000; % mm/s^2
        AmpLow = MaxAcc / (FreqLow * 2 * pi)^2; % determine maximum
amplitude from frequency and acc
        if (AmpLow>MaxAmp)
            AmpLow = MaxAmp; % reduce amplitude to 5 mm
        end
        BufLen = 4;
        %BufLen = duration*60/FreqLow*4; % length of buffer
        T_L_inCyc = 4*round(1/4/FreqLow/0.000410); %% period in cycles
        om_L = 2*pi/T_L_inCyc; % cycle frequency
        times = (0:BufLen(1))*T_L_inCyc/4; % time steps
        om2 = om_L;

        a2 = AmpLow * 10000; % amplitude in counts
        times_s = times * 0.00041;

        x = round(a2*(1-cos(times*om2))); % position at step times
        x = x / 10000; % amplitude in mm

        cs = spline(times_s,[0, x ,0]); % spline interpolation
        xinterp = linspace(times_s(1),times_s(end));
        yinterp = ppval(cs,xinterp); % for demonstration of spline
interpolation
        %         figure(1);
        plot(times_s,x,’+-‘,xinterp,yinterp);
        %         hold off;
        %%

        SplineCoefs = fliplr(cs.coefs); % for polynomial display
        Times = diff(times_s);
        Buffers = [];
        for n = 1☹BufLen(1))% build buffers matrix, multiplicate each
column with its factor
            Buffers = [Buffers;
                [Times(n),SplineCoefs(n,1),SplineCoefs(n,2),...
                SplineCoefs(n,3)*2,SplineCoefs(n,4)*6]];
        end

        % write to file

%dlmwrite(29print(‘NbuffersFor%2dHz.txt’,FreqLow),Buffers,’delimiter’,’;’,
’precision’,’%8.8e’)
        zeroit = 1;

        %%
        TableID =
calllib(‘GCSArrayIODLL’,’GCSArrayCreateNewTable’,size(Buffers,1),size(Buff
ers,2))

        Xdata = Buffers(:,2);
        if(zeroit)
            Buffers(:,2) = Xdata – Xdata(1);
        end
        ptr = libpointer(‘doublePtr’,Buffers’);
        XTData = cumtrapz(Buffers(:,1));
        [out1,out2] =
calllib(‘GCSArrayIODLL’,’GCSArraySetData’,TableID,ptr,length(Buffers(☺));
        [out1,out2] =
calllib(‘GCSArrayIODLL’,’GCSArraySetTableColumnName’,TableID,0,’Time’);
        [out1,out2] =
calllib(‘GCSArrayIODLL’,’GCSArraySetTableColumnName’,TableID,1,’Position’)
;
        [out1,out2] =
calllib(‘GCSArrayIODLL’,’GCSArraySetTableColumnName’,TableID,2,’Velocity’)
;
```

```
        [out1,out2] =
calllib('GCSArrayIODLL','GCSArraySetTableColumnName',TableID,3,'Accelerati
on');
        [out1,out2] =
calllib('GCSArrayIODLL','GCSArraySetTableColumnName',TableID,4,'Jerk');
        [out1,out2] =
calllib('GCSArrayIODLL','GCSArrayAddRemark',TableID,30print('Cosinus
Motion Profile with \nAmplitude: %fmm\n',AmpLow));
        [out1,out2] =
calllib('GCSArrayIODLL','GCSArrayAddRemark',TableID,30print('Frequency:
%fHz\n',FreqLow));
        [out1] =
calllib('GCSArrayIODLL','GCSArraySetSampleTime',TableID,1);
        %          [out1,out2,out3] =
calllib('GCSArrayIODLL','GCSArrayExportCSV',TableID,'SplineData.csv','Line
Nr');
        [out1,out2,out3] =
calllib('GCSArrayIODLL','GCSArraySave',TableID,30print('.\\CosinusFiles\\C
osinusUPX%03ddHz_%04dum.txt',FreqLow*10,round(AmpLow*1000)),'');
        [out1] = calllib('GCSArrayIODLL','GCSArrayDelete',TableID);
    end
end
```

APPENDIX C

MAIN TRACKING PROGRAM CODE

```
%function [Frame_vel,Xmat_dim, Ymat_dim, channel_width,
channel_centerline,
channeldim_avg]=Image_process_lucas_avi(fname1,fname2);
%% Introduction
%Image processing example following matlab tutorial
%TPL 5/12/11
% %% Clear Variables
    clear all
    close all
%% Initialize Global Variables
    global num;
    global numframes;
    global timestep;
%% GUI file selection
%numfiles=input('Number of File Pairs (1): ');
numfiles=1; %number of file pairs
% the first file in the pair should be an unprocessed image for
determining
% the c hannel width in each frame
% The second file should be a processed image with white particles on
black
% background from ImageJ
for i=1:numfiles
    %file_temp=input(strcat('Enter Filename of Cropped and Processed Image
for Slice',num2str(i),' : '),'s');
    %display(strcat('Enter Filename of Cropped and Processed Image for
Slice',num2str(i),' after pause: '));

    [filename,pathname] = uigetfile('*.avi',strcat('Enter Filename of
Original Image Stack'));
```

```
    fullname=fullfile(pathname, filename);
    fnames{I,1}=fullname;
    [filename,pathname] = uigetfile('*.avi',strcat('Enter Filename of
Processed Image Stack'));
    fullname=fullfile(pathname, filename);
    fnames{I,2}=fullname;
end
% test_wall=input(strcat('Do all frames show channel walls? (Y/N)'),'s');
% if test_wall == 'Y' | test_wall == 'y'
%     [filename,pathname] = uigetfile('*.avi',strcat('Enter Filename of
Original Image Stack'));
%     fullname=fullfile(pathname, filename);
%     fnames{I,3}=fullname;
% end
%% Specify file name
%     fname1='C:\Users\VUBioreactor\Desktop\ForTodd\12_29_Tilted
Experiments\Result of 3200fps_20x_opto1_147ul_tilt3_subtract_2.avi';
%     fname2='C:\Users\VUBioreactor\Desktop\ForTodd\12_29_Tilted
Experiments\3200fps_20x_opto1_147ul_tilt3_orig.avi';
%     mov1=mmreader(fname1);
%     mov2=mmreader(fname2);
    mov1=mmreader(fnames{1,1}); %original movie
    mov2=mmreader(fnames{1,2}); %processed movie
    numframes=mov1.NumberOfFrames;
    startframe=1;
    %numframes=2; %limit number of frames for diagnostics
    num=1; %start particle counter at 1
% User 'inputs'
    framerate=3200; %frames per second
    flowrate=147; %uL/min
    channelheight=110; %um %put in minimum values to minimize area
    channelwidth=1000; %um estimated – measured using video
    diam=3; %nominal particle size, um
    scale=1.21; %um/pixel
    rho=1000; %kg/m3, fluid density
    mu=0.001; %N-s/m2
    %cropdims=[220 0 979 800]; % x start y start x length y height % Crop
unneeded portions of frames

% Calculations based on inputs
    timestep=1/framerate;

%estimating velocity for downstream displacement of particles from frame
to
%frame for threshold calculation
% Note that pdisp is very important in selecting particles
    Axsec=channelheight*channelwidth; %um^2 (estimated)
    u_mean=(flowrate*1e9*(1/60))/Axsec; %um/s
    u_max=1.5*u_mean; %um/s (estimated)
    pdisp=1.5*u_max*timestep*(1/scale); %particle displacement (absolute)
in pixels/frame
    %pdisp=20;
    % Create a time vector
    t=(0:timestep☹numframes-1)*timestep)';
    f=(1:1:numframes)';
%% Loop for Wall Coordinates
```

```
H=waitbar(0,'Obtaining Wall Locations');
for frame=startframe:numframes
%for frame=600:50:5000
    f_orig = read(mov1, frame);
    f_orig = rgb2gray(f_orig);
    %Measure channel dimensions in pixels
    channeldim(frame,☺=channelwidthmain_min(f_orig,frame);
    % take data points only where left and right wall measurements
    % obtained
    % index_real=find(isnan(channeldim(:,1)+channeldim(:,2))~=1);
    waitbar(frame/(numframes),H);
    %pause
    %frame
end
close(H);

%% Smooth, scale, and calculate channel width
smooth_region=250;
for i=1:size(channeldim,2)

channeldim_smoothed(:,i)=smooth(channeldim(1:numframes,i),smooth_region);
end
% channel_width=(channeldim(:,3)-channeldim(:,1))*scale;
% channel_centerline=(channeldim(:,3)+channeldim(:,1))/2*scale;
% channeldim_scaled=channeldim*scale;
channel_width=(channeldim_smoothed(:,3)-channeldim_smoothed(:,1))*scale;
channel_centerline=(channeldim_smoothed(:,3)+channeldim_smoothed(:,1))/2*s
cale;
channeldim_scaled=channeldim_smoothed*scale;
%% Set frame crop dimensions to crop out walls for particle counting
wall_offset=25; % cut off centroid measurement by wall_offset number of
pixels from the wall
x_crop=[(wall_offset+channeldim(:,1)),(channeldim(:,3)-channeldim(:,1))-
2*wall_offset];

%% Do Loop for particle centroids
%profile on

H=waitbar(0,'Obtaining Centroid Measurements');
for frame=startframe:numframes
% for frame=1:5    % Extract frame
    f_subtract = read(mov2, frame); %read selected frame
    f_subtract = rgb2gray(f_subtract); %drop color matrices
    cropdims=[x_crop(frame,1) 1 x_crop(frame,2), 800];
    f_subtract = imcrop(f_subtract, cropdims); %crop frame



    %Call Centroid Function
    offset=x_crop(frame,1)-1; %offset in x direction for pixel coordinates
in original frame
    [C{frame,1}]=centroidmain(f_subtract,frame,offset);

    %Frame counter for updating
    frame
    waitbar(frame/(numframes),H);
```

```
end
close(H);

%profile report
save('loopone.mat')
%%  convert centroid cell array to number array
    for frame=startframe:numframes
        Ccorr1{frame,1}=cell2mat(C{frame});
    end
    Ccorr2=cell2mat(Ccorr1);
%% Call particle counting function – Based on relative reference frame
    param.mem=2; %maximum time steps for a "lost" particle
    param.good=10; %minimum frames a particle must exist to be included in
tracked particle output
    param.dim=2; %first two cols of Ccorr2 represent centroid data
    param.quiet=0; %set to 1 to turn off output text
    results=track(Ccorr2,pdisp,param); %2nd input represents maximum pixel
displacement between frames – factor may need adjustment if "track" has
difficulty
%% Replace all non-values with NaNs and populate position matrices

   unique=max(results(:,9)); %read number of unique tracked particles
   Xmat=sparse(zeros(unique,numframes)); %creates matrix for x position –
column number is the frame, row number is the particle number
   Ymat=Xmat;pixel_intensity=Ymat;eccentricity=Ymat;
   H=waitbar(0,'Arranging Tracking Data');
    for part=1:size(results,1)
        Xmat(results(part,9),results(part,8))=results(part,1); % x
position at row = particle number, column = frame
        Ymat(results(part,9),results(part,8))=results(part,2); % y
position at row = particle number, column = frame
        %Nmat(results(part,9),results(part,8))=results(part,7); % particle
number from centroid tracker at row = particle number, column = frame
        pixel_intensity(results(part,9),results(part,8))=results(part,6);
%weighted centroid at row = particle number, column = frame
        eccentricity(results(part,9),results(part,8))=results(part,5);
%particle shape at row = particle number, column = frame
        waitbar(part/(size(results,1)),H);
    end
    close(H);
    %% scale values
    Xmat_dim=Xmat*scale; %convert x position to um
    Ymat_dim=Ymat*scale; %convert y position to um


% %% Particle velocities based on position vectors- central in time
%     Xmatrixvel=NaN(size(Xmat_dim)); %initialize matrix with NaNs
%     Ymatrixvel=Xmatrixvel;
%     Velmag=Xmatrixvel;
%     for j=1:size(Xmat_dim,1)
%         for k=2:numframes-1
%             if isnan(Xmat_dim(j,k))==0 && isnan(Xmat_dim(j,k+1))==0 &&
isnan(Xmat_dim(j,k-1))==0
```

```
%                    Xmatrixvel(j,k)=(Xmat_dim(j,k+1)-Xmat_dim(j,k-
1))/(2*timestep);
%                    Ymatrixvel(j,k)=-(Ymat_dim(j,k+1)-Ymat_dim(j,k-
1))/(2*timestep);
%                    Velmag(j,k)=sqrt(Xmatrixvel(j,k)^2+Ymatrixvel(j,k)^2);
%               end
%           end
%       end
%% Particle velocities based on position vectors — forward in time
Xmatrixvel=sparse(zeros(unique,numframes)); %initialize matrix with NaNs
Ymatrixvel=Xmatrixvel;
Velmag=Xmatrixvel;
H=waitbar(0,'Calculating Particle Velocities');
for j=1:size(Xmat_dim,1)
    for k=1:numframes-1
        %if isnan(Xmat_dim(j,k))==0 && isnan(Xmat_dim(j,k+1))==0
        if isfinite(1/Xmat_dim(j,k))==1 && isfinite(1/Xmat_dim(j,k+1))==1
%calculate if particle exists for the frame
            Xmatrixvel(j,k)=(Xmat_dim(j,k+1)-Xmat_dim(j,k))/(timestep);
%calculate x velocity in um/s
            Ymatrixvel(j,k)=-(Ymat_dim(j,k+1)-Ymat_dim(j,k))/(timestep);
%calculate y velocity in um/s
            Velmag(j,k)=sqrt(Xmatrixvel(j,k)^2+Ymatrixvel(j,k)^2);
%calculate velocity magnitude in um/s
        end
    end
    waitbar(j/(size(Xmat_dim,1)),H);
end
close(H);
%% Particle Acceleration Calculation based on particle positions — central
in time
Xmatrixacc=sparse(zeros(unique,numframes)); %initialize matrix with NaNs
Ymatrixacc=Xmatrixacc;
Accmag=Xmatrixvel;
H=waitbar(0,'Calculating Particle Accelerations');
for j=1:size(Xmat_dim,1)
    for k=2:numframes-1
        %if isnan(Xmat_dim(j,k))==0 && isnan(Xmat_dim(j,k+1))==0 &&
isnan(Xmat_dim(j,k-1))==0
        if isfinite(1/Xmat_dim(j,k))==1 && isfinite(1/Xmat_dim(j,k+1))==1
&& isfinite(1/Xmat_dim(j,k-1))==1 %calculate if particle exists for the
frame
        Xmatrixacc(j,k)=(Xmat_dim(j,k+1)+Xmat_dim(j,k-1)-
2*Xmat_dim(j,k))/(timestep^2); %calculate x acceleration in um/s2
        Ymatrixacc(j,k)=(Ymat_dim(j,k+1)+Ymat_dim(j,k-1)-
2*Ymat_dim(j,k))/(timestep^2);%calculate y acceleration in um/s2
        Accmag(j,k)=sqrt(Xmatrixacc(j,k)^2+Ymatrixacc(j,k)^2); %calculate
acceleration magnitude in um/s2
        end
    end
    waitbar(j/(size(Xmat_dim,1)),H);
end
close(H);
%% Number individual particles
Part_num=cell(size(Xmat,1),1);
H=waitbar(0,'Indexing Particle Data');
for part=1:size(Xmat,1)
```

```
    for frame=1:numframes
        if isfinite(1/Xmatrixvel(part,frame))==1;
            %particle number, particle frame, x pixel pos in frame, y
pixel
            %pos in frame,relative x velocity (scaled), relative y
velocity
            %(scaled), relative x accel (scaled), relative y accel
(scaled)
            % [ particle number, frame number, time, Xposition in
frame(um),
            % Yposition in frame(um), Xvelocity (um/s), Yvelocity (um/s),
            % Xacceleration (um/s2),Yacceleration (um/s2), pixel
intensity, ecceentricity]

            Part_num{part,1}=[Part_num{part,1};[part, frame, t(frame),
full(Xmat_dim(part,frame)),full(Ymat_dim(part,frame)),
full(Xmatrixvel(part,frame)),full(Ymatrixvel(part,frame)),full(Xmatrixacc(
part,frame)),full(Ymatrixacc(part,frame)),full(pixel_intensity(part,frame)
),full(eccentricity(part,frame))]];
        else
        end
    end
    waitbar(part/(size(Xmat,1)),H);
end
close(H);

%% Smooth data
H=waitbar(0,'Smoothing Data');
smooth_region_vel=20; %moving average filter region for velocity
smooth_region_int=20; %moving average filter region for intensity
smooth_region_ecc=15; %moving average filter region for eccentricity
tic
for particle=1:size(Part_num,1);
    if size(Part_num{particle,:},1)>2;

Part_num{particle,:}(:,12)=smooth(Part_num{particle,:}(:,7),smooth_region_
vel); % smoothed velocity profile

Part_num{particle,:}(:,13)=smooth(Part_num{particle,:}(:,10),smooth_region
_int); % smoothed intensity profile

Part_num{particle,:}(:,14)=smooth(Part_num{particle,:}(:,11),smooth_region
_ecc); % smoothed eccentricity profile
        waitbar(particle/(size(Part_num,1)),H);
    end
    %particle
end
toc
close(H);
%% Test for particles to keep based on eccentricity threshold
particle_index_1=zeros(0,4); %initialize index matrix – 1st column:
particle number, 2nd column: frame at which particle is in focus
ecc_thresh_1=0.75; %min eccentricity threshold for double dips
ecc_thresh=0.55; %minimum eccentricity threshold
peak_thresh=50; %peak separation threshold, frames
H=waitbar(0,'Filtering Data on Eccentricity');
```

```
for particle=1:size(Part_num,1)
    %particle
    if size(Part_num{particle,:},1)>2 &&
min(Part_num{particle,1}(:,14))<ecc_thresh %check for minimum eccentricity
        ecci_min=[];
        [ecc_min,ecci_min]=findpeaks(-Part_num{particle,1}(:,14));
%localextrema for eccentricity
        ecc_min=-1*ecc_min;
            if min(ecc_min)<ecc_thresh %discard edge minima
                ecci_min=ecci_min(find(ecc_min<ecc_thresh_1)); %discard
local minima with high eccentricity (keep the particles indexed)
                ecc_min=Part_num{particle,1}(ecci_min,14);
                [int_max,inti_max]=max(Part_num{particle,1}(:,13));
                if size(ecci_min,1)>1 && size(ecci_min,2)==1 %check for
multiple minima
                    ecc_min=min(Part_num{particle,1}(ecci_min,14));
%update ecc_min value
                    ecci_min=(find(Part_num{particle,1}(:,14)==ecc_min));
                end
                if size(ecci_min,1)==1 && size(ecci_min,2)==1
                    if ecci_min(1)>5 &&
ecci_min(1)<(size(Part_num{particle,1},1)-5) && inti_max(1)>5 &&
inti_max(1)<(size(Part_num{particle,1},1)-5)

peak_separation=abs(Part_num{particle,1}(ecci_min,2)-
Part_num{particle,1}(inti_max,2));
                        if peak_separation<=peak_thresh

particle_index_1=[particle_index_1;particle,Part_num{particle,1}(ecci_min,
2),ecci_min,inti_max]; %particle number, frame number, index within cell
array
                        end
                    end
                end
            end
        waitbar(particle/(size(Part_num,1)),H);
    end

end
close(H);
%% Filter out large changes in velocity data
n_hood=10; %filter neighborhood
remove_index=zeros(0,2);
H=waitbar(0,'Removing Bad Data');
for i=1:size(particle_index_1,1);
    particle=particle_index_1(I,1);
    ecci_min=particle_index_1(I,3);
    inti_max=particle_index_1(I,4);
    %particle
    temp1=(Part_num{particle,1}(max(1,ecci_min-
n_hood):min(size(Part_num{particle,1},1),ecci_min+n_hood),7)); %raw data,
test for velocity changes near eccentricity min
    temp2=(Part_num{particle,1}(max(1,ecci_min-
n_hood):min(size(Part_num{particle,1},1),ecci_min+n_hood),12)); %smoothed
data
    spread_pct_raw=abs(max(temp1)-min(temp1))/nanmean(temp1);
    spread_pct_filt=abs(max(temp2)-min(temp2))/nanmean(temp2);
    if spread_pct_raw>0.15 && spread_pct_filt>0.10
        remove_index=[remove_index;particle,i];
    else %test for velocity changes near intensity peak
        temp1=(Part_num{particle,1}(max(1,inti_max-
n_hood):min(size(Part_num{particle,1},1),inti_max+n_hood),7)); %raw data,
test for velocity changes near eccentricity min
        temp2=(Part_num{particle,1}(max(1,inti_max-
n_hood):min(size(Part_num{particle,1},1),inti_max+n_hood),12)); %smoothed
data
        spread_pct_raw=abs(max(temp1)-min(temp1))/nanmean(temp1);
        spread_pct_filt=abs(max(temp2)-min(temp2))/nanmean(temp2);
        if spread_pct_raw>0.15 && spread_pct_filt>0.10
            remove_index=[remove_index;particle,i];
        end
    end
    waitbar(particle/(size(Part_num,1)),H);
end
% remove the data rows in the particle index
particle_index_2=particle_index_1;
particle_index_2(remove_index(:,2),☺=[];
close(H);

%% assign final data filter
particle_index=particle_index_2;

% %% Plot intensity,velocity, and eccentricity as a check
%
% particle=particle_index(1,1);
% figure;
% subplot(1,2,1);
%
[haxes_1,hline1_1,hline2_1]=plotyy(Part_num{particle,1}(:,2),[Part_num{par
ticle,1}(:,10),1e2*Part_num{particle,1}(:,11)],Part_num{particle,1}(:,2),P
art_num{particle,1}(:,7));
% h1=title(strcat({'Raw Data, Particle '},num2str(particle)));
% axes(haxes_1(1));
% ylabel({'Pixel Intensity'});
% set(haxes_1(1),'ytickmode','auto','ylim',[0,200]);
% xlim('auto')
% xlabel('Frame')
% axes(haxes_1(2));
% ylabel({'Velocity';'(um/s)'});
% set(haxes_1(2),'ytickmode','auto','ylim',[0,10e4]);
% xlim('auto')
% xlabel('Frame')
%
% subplot(1,2,2);
%
[haxes_2,hline1_2,hline2_2]=plotyy(Part_num{particle,1}(:,2),[Part_num{par
ticle,1}(:,13),1e2*Part_num{particle,1}(:,14)],Part_num{particle,1}(:,2),P
art_num{particle,1}(:,12));
% h2=title(strcat({'Smoothed Data, Particle '},num2str(particle)));
% axes(haxes_2(1));
% ylabel({'Pixel Intensity'});
% set(haxes_2(1),'ytickmode','auto','ylim',[0,200]);
% xlim('auto')
% xlabel('Frame')
% axes(haxes_2(2));
% ylabel({'Velocity';'(um/s)'});
% set(haxes_2(2),'ytickmode','auto','ylim',[0,10e4]);
% xlim('auto')
% xlabel('Frame')
% pause
%
% for i=2:size(particle_index,1);
%     particle=particle_index(I,1);
%     set(hline1_1(1), 'Xdata', Part_num{particle,1}(:,2), 'Ydata',
Part_num{particle,1}(:,10)); %intensity
%     set(hline1_1(2), 'Xdata', Part_num{particle,1}(:,2), 'Ydata',
1e2*Part_num{particle,1}(:,11)); %eccentricity
%     set(hline2_1(1), 'Xdata', Part_num{particle,1}(:,2), 'Ydata',
Part_num{particle,1}(:,7)); %velocity
%     set(hline1_2(1), 'Xdata', Part_num{particle,1}(:,2), 'Ydata',
Part_num{particle,1}(:,13)); %smoothed intensity
%     set(hline1_2(2), 'Xdata', Part_num{particle,1}(:,2), 'Ydata',
1e2*Part_num{particle,1}(:,14)); %smoothed eccentricity
%     set(hline2_2(1), 'Xdata', Part_num{particle,1}(:,2), 'Ydata',
Part_num{particle,1}(:,12)); %smoothed velocity
%     axes(haxes_1(1));
%     set(haxes_1(1),'ytickmode','auto','ylim',[0,200]);
%     xlim('auto')
%     axes(haxes_1(2));
%     set(haxes_1(2),'ytickmode','auto','ylim',[0,10e4]);
%     xlim('auto')
%     set(h1,'String',(strcat({'Raw Data, Particle '},num2str(particle))))
%     axes(haxes_2(1));
%     set(haxes_2(1),'ytickmode','auto','ylim',[0,200]);
%     xlim('auto')
%     axes(haxes_2(2));
%     set(haxes_2(2),'ytickmode','auto','ylim',[0,10e4]);
%     xlim('auto')
%     set(h2,'String',(strcat({'Smoothed Data, Particle
'},num2str(particle))))
%     pause
% end
%% Plot the selected particles at each particular captured frame

%% Sort out data
%part_select=[36,52,65,66,68,94,98,121,122,123,126,161,163,165,173,191,192
,193,196,217,218,220,251,254,255];
%part_select=[19,26,27,39,74,75,76,79,81,82,87,89,90,93,95,98,102,103,104,
109,111,112,119,125,129,133,139,143,158,158,160,163,166,168,170,172,173,17
4,175,178,179,180,181,187,188,193,230,232,234,235,236,238,240,242,244,247]
;

theta=17;
part_select=particle_index(:,1); %particle number
frame_select=particle_index(:,2); %frame number
ind_select=particle_index(:,3); %particle index within Part_num cell
matrix
vel_data=zeros(size(part_select,1),8);
for  i=1:size(part_select,1);
```

```
    vel_data(I,1)=part_select(i); % particle number
    vel_data(I,2)=Part_num{part_select(i),:}(ind_select(i),2); %frame of
measurement
    vel_data(I,3)=Part_num{part_select(i),:}(ind_select(i),3); %time of
measurement
    vel_data(I,4)=Part_num{part_select(i),:}(ind_select(i),4); %x
coordinate
    vel_data(I,5)=(Part_num{part_select(i),:}(ind_select(i),4)-
channeldim_scaled(vel_data(I,2),1))/channel_width(vel_data(I,2),1);
%dimensionless x coordinate where 0=left wall, 0.5=centerline, 1=right
wall
    vel_data(I,6)=Part_num{part_select(i),:}(ind_select(i),5)/cosd(theta);
%convert y* coordinate to y coordinates
    % calculate the z=0 plane (z centerline) based on wall "in focus" data
    x_star_1=0;
    x_star_2=channeldim_scaled(vel_data(I,2),3)-
channeldim_scaled(vel_data(I,2),1);
    y_star_1=channeldim_scaled(vel_data(I,2),2);
    y_star_2=channeldim_scaled(vel_data(I,2),4);
    x_star=vel_data(I,4)-channeldim_scaled(vel_data(I,2),1);
    %x_star=0.5*x_star_2; %comment this out for a non-static, linear fit
case
    y_star_0=(y_star_2-y_star_1)/(x_star_2-x_star_1)*x_star+y_star_1; %z=0
centerline in terms of y*
    y_star=Part_num{part_select(i),:}(ind_select(i),5);



    %dist=((y_star_2-y_star_1)/(x_star_2-x_star_1)*x_star-
y_star+y_star_1)/(sqrt(((y_star_2-y_star_1)/(x_star_2-x_star_1))^2+1));
    dist=Part_num{part_select(i),:}(ind_select(i),5)-y_star_0;
    vel_data(I,7)=(dist)*sind(theta); %z-coordinate

vel_data(I,8)=Part_num{part_select(i),:}(ind_select(i),12)/cosd(theta);
%streamwise (y) velocity (smoothed)

end

% Plot 2d velocity profile data (all times)
iptsetpref('ImshowBorder','tight');
f1=figure('visible','on');
velplot=scatter(vel_data(:,7),vel_data(:,8)),ylim([0 7e4]),xlim([-400
400]);
% Plot 2d velocity profile data (all times)
iptsetpref('ImshowBorder','tight');
f1=figure('visible','on');
velplot=scatter(vel_data(:,5),vel_data(:,8)),ylim([0 4e4]),xlim([0 1]);
% Plot 3d velocity profile (all times)
iptsetpref('ImshowBorder','tight');
f2=figure('visible','on');
scatter3(vel_data(:,4),vel_data(:,7),vel_data(:,8));

%% Sort rows based on time
%iptsetpref('ImshowBorder','tight');
%f3=figure('visible','on');
vel_data_timesort=sortrows(vel_data,3); %sort rows based on time
% eliminate data too near lateral walls
```

```
x_min=0.15;
x_max=0.85;
vel_data_timesort=vel_data_timesort(find(vel_data_timesort(:,5)>x_min &
vel_data_timesort(:,5)<x_max),☺;
% velplot=scatter(vel_data_timesort(:,7),vel_data_timesort(:,8)),ylim([0
7e4]),xlim([-500 500]);

% %% Plot particle number overlay with velocity (all kept particles, tif
output)
% H=waitbar(0,'Creating Check Plots');
% framesample=50;
% for frame=250:350%numframes
%     frame
%     f_orig = read(mov1, frame);
%     f_orig = rgb2gray(f_orig);
%     %Re-plot new particle numbers
%     framesample=50; %interval between file outputs
%     f_orig=imadjust(f_orig); %increase contrast for output
%     fig=figure('visible','off');
%     axes1 =
axes('Parent',fig,'Ygrid','off','Xgrid','off','LineWidth',1,...
%     'FontSize',16);
%     imshow(f_orig);
%
%     hold on
%     for particle=1:size(Part_num,1)
%         if size(Part_num{particle,:},1)~=0
%             for f=1:size(Part_num{particle,1}(:,2),1)
%                 if Part_num{particle,1}(f,2)==frame %check to see if the
frame matches
%                     cx=Part_num{particle,1}(f,4)/scale; %re-scale to
pixel coordinates
%                     cy=Part_num{particle,1}(f,5)/scale; %re-scale to
pixel coordinates
%                     plot1=plot(cx,cy,'Parent',axes1,'MarkerEdgeColor',[0
0 1],'Marker','o','MarkerSize',12,'LineWidth',2,'LineStyle','none');
% %                     scatter(cx,cy,'blue','o','filled')
%                     hold on
%                     text(cx+20, cy, 40print('%d', particle)); %plot
particle number in overlay
%                     hold on
%                     text(75,25,40print('%c','Frame='));
%                     text(150,25,40print('%d',frame));
%                     hold on
% %
scatter(channeldim(frame,1),channeldim(frame,2),'red','o','filled')
% %
scatter(channeldim(frame,3),channeldim(frame,4),'red','o','filled')
%
plot2=plot(channeldim(frame,1),channeldim(frame,2),'Parent',axes1,'MarkerE
dgeColor',[1 0 0],'Marker','o','LineWidth',4,'LineStyle','none');
%
plot3=plot(channeldim(frame,3),channeldim(frame,4),'Parent',axes1,'MarkerE
dgeColor',[1 0 0],'Marker','o','LineWidth',4,'LineStyle','none');
```

```
%
plot4=plot([channeldim(frame,1),channeldim(frame,3)],[channeldim(frame,2),
channeldim(frame,4)],'Color',[1 0 0],'LineStyle','—','LineWidth',2);
%                  end
%              end
%          end
%      end
%      fignum=hardcopy(fig,'-Dzbuffer','-r100'); %scale to save memory
%      %ff=ff(:,:,1); %Grayscale only
%      imwrite(fignum, 'numbers.TIFF', 'writemode', 'append');
%      %frame %display current frame
%      hold off
%      waitbar(frame/(numframes),H);
% end
% close(H);
%% Static curve fitting
z_fit=[-100:1:100];
channel_z=zeros(1,3);
vel_fit_coeff(1,☺=polyfit(vel_data_timesort(:,7),vel_data_timesort(:,8),2)
;
vel_fit(1,☺=polyval(vel_fit_coeff(1,☺,z_fit);
channel_z(1,1:2)=roots(vel_fit_coeff(1,☺);
channel_z(:,3)=abs(channel_z(:,1)-channel_z(:,2));

figure;
subplot(2,1,1); %plot static channel width
axis([0 t(numframes,1)*1000 800 1200]);
plot(t*1000, channel_width, 'LineWidth', 2);
xlabel('time (millisecond)'); ylabel('Channel Width (microns)');

subplot(2,1,2); %plot static velocity profile
hh3(1) =
plot(vel_data_timesort(:,7),vel_data_timesort(:,8),'MarkerEdgeColor',[0 0
1],'Marker','o','MarkerSize',5,'LineWidth',2,'LineStyle','none');
hh3(2) = line(z_fit,vel_fit(1,☺,'LineWidth',2,'Color',[1 0 0])
axis([-100 100 0 4e4])
xlabel('z-position (microns)'); ylabel('Velocity (micron/s)');
% static shear stress calculation
for i=1:size(vel_fit_coeff,1)
    du_dz(I,☺=2*vel_fit_coeff(I,1)*channel_z(I,1:2)+vel_fit_coeff(I,2);
%micron/s over microns = [1/s] units
end


%% Plot and analyze dynamic Time dependencies
% determine time range for time binning
%t_min=(vel_data_timesort(1,3));

f_min=1;
f_max=numframes;
t_min=0;
t_max=t(f_max,1);
n_bins=25; %number of bins
bin_sizet=(t_max-t_min)/n_bins;
bin_edget=[t_min:bin_sizet:t_max]';
```

```
%[a,bin_index]=histc(vel_data_timesort(:,3),bin_edget);
%create time bin index
t_bin=zeros(n_bins,1);
for i=1:n_bins
    t_bin(I,1)=(bin_edget(I,1)+bin_edget(i+1,1))/2;
    %f_bin(I,1)=(bin_edgef(I,1)+bin_edgef(i+1,1))/2;
    tdiff=abs(t-t_bin(I,1));
    [idx idx]=min(tdiff);
    f_bin(I,1)=f(idx);

end

channel_z=zeros(size(t_bin,1),3);
temp_index=cell(size(t_bin,1),1);
Frame_plot=cell(size(t_bin,1),1);
vel_fit_coeff=zeros(size(t_bin,1),3);
z_fit=[-100:1:100];

for count=1:size(Frame_plot,1)

temp_index{count,1}=find(vel_data_timesort(:,3)>(bin_edget(count))&vel_dat
a_timesort(:,3)<(bin_edget(count+1)));
    Frame_plot{count,1}=vel_data_timesort(temp_index{count,1},☺;
    % fit to parabola


vel_fit_coeff(count,☺=polyfit(vel_data_timesort(temp_index{count,1},7),vel
_data_timesort(temp_index{count,1},8),2);
    vel_fit(count,☺=polyval(vel_fit_coeff(count,☺,z_fit);
    %count
    channel_z(count,1:2)=roots(vel_fit_coeff(count,☺);
end
u_max_fit=max(vel_fit,[],2);
%infer height from velocity curve fit
channel_z(find(imag(channel_z)~=0))=NaN;
channel_z(:,3)=abs(channel_z(:,1)-channel_z(:,2));
for i=1:size(vel_fit_coeff,1)
    du_dz(I,☺=2*vel_fit_coeff(I,1)*channel_z(I,1:2)+vel_fit_coeff(I,2);
%micron/s over microns = [1/s] units
end

%% Plot time dependent velocity data
mov(1:length(t_bin)) = struct('cdata', [], 'colormap', []);
count=1;
screen_size = get(0, 'ScreenSize');
ptime=1; % pause time between frames
f1=figure;
set(f1, 'Position', [0 0 screen_size(3) screen_size(4) ] );
subplot(5,1,1); %plot animated channel width
axis([0 t(numframes,1)*1000 800 1200]);
plot(t*1000, channel_width, 'LineWidth', 2);
hh1 = line(t_bin(1)*1000, channel_width(f_bin(1),1), 'Marker', '.',
'MarkerSize', 20, 'Color', 'b');
xlabel('time (millisecond)'); ylabel('Channel Width (microns)');
tt=title(strcat({'Time = '},num2str(round(t_bin(1)*1000)), {' ms'}));
```

```
subplot(5,1,2); %plot animated channel height (calculated)
plot(t_bin*1000, channel_z(:,3), 'LineWidth', 2);
xlabel('time (millisecond)'); ylabel('Channel Height (microns)');
hh2 = line(t_bin(1)*1000, channel_z(1,3), 'Marker', '.', 'MarkerSize', 20,
'Color', 'b');

subplot(5,1,3); %plot animated du_dz
plot(t_bin*1000,abs(du_dz(:,1)), 'LineWidth', 2);
xlabel('time (millisecond)'); ylabel('du/dz (1/s)');
hh3 = line(t_bin(1)*1000, abs(du_dz(1,1)), 'Marker', '.', 'MarkerSize',
20, 'Color', 'b');

subplot(5,1,4); %plot animated umax_fit
plot(t_bin*1000,u_max_fit, 'LineWidth', 2);
xlabel('time (millisecond)'); ylabel('u_max (fit) (micron/s)');
hh4 = line(t_bin(1)*1000, u_max_fit(1,1), 'Marker', '.', 'MarkerSize', 20,
'Color', 'b');

subplot(5,1,5); %plot animated velocity profile
hh5(1) =
plot(Frame_plot{1,1}(:,7),Frame_plot{1,1}(:,8),'MarkerEdgeColor',[0 0
1],'Marker','o','MarkerSize',5,'LineWidth',2,'LineStyle','none');
hh5(2) = line(z_fit,vel_fit(1,☺,'LineWidth',2,'Color',[1 0 0]);
axis([-100 100 0 4e4])
xlabel('z-position (microns)'); ylabel('Velocity (micron/s)');
%pause (ptime)
mov(count) = getframe(gcf);
for count=2:size(Frame_plot,1)
   % Update Xdata and Ydata
   set(hh1, 'Xdata', t_bin(count,1)*1000, 'Ydata',
channel_width(f_bin(count),1));
   set(hh2, 'Xdata', t_bin(count,1)*1000, 'Ydata', channel_z((count),3));
   set(hh3, 'Xdata', t_bin(count,1)*1000, 'Ydata', abs(du_dz(count,1)));
   set(hh4, 'Xdata', t_bin(count,1)*1000, 'Ydata', u_max_fit(count,1));
   set(hh5(1), 'Xdata', Frame_plot{count,1}(:,7), 'Ydata',
Frame_plot{count,1}(:,8));
   set(hh5(2), 'Xdata', z_fit, 'Ydata', vel_fit(count,☺);
   set(tt,'String',(strcat({'Time =
'},num2str(round(t_bin(count,1)*1000)), {' ms'})))
   drawnow

   %pause (ptime)
   mov(count) = getframe(gcf);
end

movie2avi(mov, 'dynamic.avi','fps',1);

clear mov
%static shear stress calculation

%% end function call
%end
```

APPENDIX D

“CHANNEL WIDTH MAIN”

```
function [channeldim]=channelwidthmain(frame1,frame);
%Todd Lagus
%Channel width measurement using minimum pixel values for finding the
focused wall
%location. This code assumes that the streamwise direction is vertical in
the image. Create two images (one for left wall, one for right wall) and
%find the minima and location in each.

global numframes
%% Find Minimum and maximum pixel values
%ymax;
xwidth=size(frame1,2);
%yheight=size(frame1,1);
yheight=size(frame1,1);
% %% sobel edge filter on original frame
% filt1=fspecial('sobel');
% filt1=filt1';
% frame1_filt=imfilter(frame1,filt1,'replicate');
frame1_filt=frame1;
%% Thresholding
%thresh=graythresh(channel); %automatically find threshold using Otsu's
%method
thresh= 70; %manual threshold value
channel=im2bw(frame1,thresh/255);
%imshow(channel)
%% Remove Noise
channel=1-channel;
channel=bwareaopen(channel, 200); %removes objects less than pixel area
specified
%imshow(channel)
%% Find ballpark wall coordinate
[B,L]=bwboundaries(channel, 8,'noholes');

while size(B,1)~=2 %dilate thresholded walls until each wall section is
one object
    channel=imdilate(channel,strel('disk',4));
    [B,L]=bwboundaries(channel, 8,'noholes');
end

stats=regionprops(L,'Centroid');
%% Create a region of interest to determine acutal wall coordinates
tol_x=15;
tol_y=100;
left=[min(B{1,1}(:,2)),min(B{1,1}(:,1));max(B{1,1}(:,2)),max(B{1,1}(:,1))]
;% left wall [xmin ymin; xmax ymax]
right=[min(B{2,1}(:,2)),min(B{2,1}(:,1));max(B{2,1}(:,2)),max(B{2,1}(:,1))
];% right wall [xmin ymin; xmax ymax]
framecrop_left=[(left(1,1)-tol_x) (left(1,2)-tol_y) (left(2,1)-
left(1,1)+2*tol_x) (left(2,2)-left(1,2)+2*tol_y)];
framecrop_right=[(right(1,1)-tol_x) (right(1,2)-tol_y) (right(2,1)-
right(1,1)+2*tol_x) (right(2,2)-right(1,2)+2*tol_y)];

% filt_left=fspecial('average',[20 5]);
% filt_right=fspecial('average',[20 5]);
% frame_left=imfilter(frame_left,filt_left,'replicate');
% frame_right=imfilter(frame_right,filt_right,'replicate');
filt2=fspecial('average',[100,5]);
frame2_filt=imfilter(frame1_filt,filt2,'replicate');
frame_left=imcrop(frame2_filt,framecrop_left);
frame_right=imcrop(frame2_filt,framecrop_right);%imshow(frame_right);
%% Find x-y wall coordinate using minimum pixel value

[min_left,x_left]=min(min(frame_left,[],1));
[min_left,y_left]=min(min(frame_left,[],2));
[min_right,x_right]=min(min(frame_right,[],1));
[min_right,y_right]=min(min(frame_right,[],2));
x_left=x_left+(framecrop_left(1,1)-1);
y_left=y_left+(framecrop_left(1,2)-1);
x_right=x_right+(framecrop_right(1,1)-1);
y_right=y_right+(framecrop_right(1,2)-1);

channeldim=[x_left,y_left,x_right,y_right];

%% Average Channel
h1=subplot(1,2,1);
imshow(frame1)
title(strcat({'Un-processed Frame '},num2str(frame), {' of '},
num2str(numframes)))
hold on
scatter(x_left,y_left,'blue','o','filled');
hold on
scatter(x_right,y_right,'red','o','filled');
hold on
rectangle('position',framecrop_left,'EdgeColor','red');
hold on
rectangle('position',framecrop_right,'EdgeColor','red');
hold off
h2=subplot(1,2,2);
%imshow(channel)
imshow(frame2_filt)
title(strcat({'Processed Frame '},num2str(frame), {' of '},
num2str(numframes)))
end
```

APPENDIX E

"CENTROIDMAIN"

```
function [A]=centroidmain(cframe,frame,x_offset)
%% Introduction
%TPL 5/12/11
%Recall Global Variables
global num;
global numframes;
    %% Thresholding for particle centers
    %thresh=graythresh(channel); %automatically find threshold using
Otsu's
    %method
    thresh=30; %manual threshold value
    cframe_binary=im2bw(cframe,thresh/255);
%     imshow(cframe_binary)
%     %% Fill Holes 1
%     cframe_binary=imfill(cframe_binary,'holes');
%     %imshow(cframe_binary)
% subplot(1,2,1), imshow(cframe_binary)
% subplot(1,2,2), imshow(cframe)
    %% Remove Noise
    cframe_binary=bwareaopen(cframe_binary, 10); %removes objects less
than pixel area specified
    %imshow(cframe_binary)
    %% filter large particles
    cframe_binary=cframe_binary-bwareaopen(cframe_binary, 100);
   %imshow(cframe_binary);
%     %% Watershed segmentation of "double" particles
%     D = -bwdist(~cframe_binary);
%     W=watershed(D); %label matrix W
%     cframe_binary(W == 0) = 0;
%     %imshow(cframe_binary) % Segmented image D (above)
%         %% Remove Noise
%     cframe_binary=bwareaopen(cframe_binary, 10); %removes objects less
than pixel area specified
%     %imshow(cframe_binary)
%% imclose
cframe_binary=imclose(cframe_binary,strel('disk',5));
 %% Remove Noise
    cframe_binary=bwareaopen(cframe_binary, 10); %removes objects less
than pixel area specified
%% Animated display (optional, disable this on a slow computer)
h1=subplot(1,2,1);
imshow(cframe)
title(strcat({'Pre-processed Frame '},num2str(frame), {' of '},
num2str(numframes)))
h2=subplot(1,2,2);
imshow(cframe_binary)
title(strcat({'Processed Frame '},num2str(frame), {' of '},
num2str(numframes)))
%% Find Particles
    [B,L]=bwboundaries(cframe_binary, 8,'noholes'); %noholes - only finds
parent and child boundaries
    s=regionprops(L,'all');
    numparticles=max(L(:));

    %% Find Centroids
    centers=[s.Centroid];
    c=zeros(size(B,1),2);
    for j=1:length(centers)/2
        c(j,1)=centers(2*j-1);
        c(j,2)=centers(2*j);
    end
    %% Calculate Intensity
    average_pixel_values=zeros(size(B,1),1);
    cmassk=zeros(size(B,1),2);
    for k = 1:numel(s)
        idx = s(k).PixelIdxList;
        pixel_values = double(cframe(idx));
        sum_pixel_values = sum(pixel_values);
        average_pixel_values(k)=sum_pixel_values/size(pixel_values,1);
        x = s(k).PixelList(:, 1);
        y = s(k).PixelList(:, 2);
        cmassk(k,1) = sum(x .* pixel_values) / sum_pixel_values;
        cmassk(k,2) = sum(y .* pixel_values) / sum_pixel_values;
    end
    %%
    A=cell(size(B,1),1);
    shapes=[s.Eccentricity];
    areas=[s.Area];
    if length(average_pixel_values)==0
        A{1,1}=[0 0 0 0 0 frame]
        else
        for j=1:length(average_pixel_values)
        A{j,1}=[cmassk(j,1)+x_offset cmassk(j,2) c(j,1)+x_offset c(j,2)
shapes(j) average_pixel_values(j) num frame]; %added x_offset to obtain
the proper coordinates
                num=num+1;
        end
    end
%     %% Diagnostic-Display "keeper" particles over original image
%     keepers=[1:length(average_pixel_values)];
%     iptsetpref('ImshowBorder','tight');
%     f=figure('visible','on');
%     imshow(cframe_orig);
%     hold on
%    for n=1:length(keepers)
%         outline=B{keepers(n)};
%         line(outline(:,2),outline(:,1),'Color','r','LineWidth',2)
%         hold on
%         cx=A{n,1}(1);
%         cy=A{n,1}(2);
%         cn=A{n,1}(7);
%         %scatter(cmassk(n,1),cmassk(n,2),'blue','o','filled')
%         hold on
%         %scatter(cx,cy,'red','o','filled')
%         text(cx+5, cy, sprintf('%d', cn)); %display particle number
%         %text(cx+5, cy, sprintf('%d', shapes(n))); %display eccentricity
%         hold on
%
%    end
%     pause
%     %% display non-keeper particles
%     for n=1:length(nokeepers)
%         nooutline=B{nokeepers(n)};
%         line(nooutline(:,2),nooutline(:,1),'Color','g','LineWidth',2)
%         hold on
% %         nopoint=c(nokeepers(n),:);
% %         scatter(nopoint(1),nopoint(2),'green','o','filled')
% %         hold on
% %         text(nopoint(1)+20, nopoint(2), sprintf('%d', nopoint(1)));
% %         hold on
%     end
%     %% Write to file to check numbers
%     ff=hardcopy(f,'-Dzbuffer');%,'-r100');
%     %ff=ff(:,:,1); %Grayscale only
%     imwrite(ff, 'myFile.TIFF', 'writemode', 'append');
%     %frame %display current frame
%    hold off
end
```

APPENDIX F

TRACKING PROGRAM

```
function tracks = track(xyzs,maxdisp,param)

%;
% ; see http://glinda.lrsm.upenn.edu/~weeks/idl
% ;   for more information
% ;
% ;+
% ; NAME:
% ; track
% ; PURPOSE:
% ; Constructs n-dimensional trajectories from a scrambled list of
% ; particle coordinates determined at discrete times (e.g. in
% ; consecutive video frames).
% ; CATEGORY:
% ; Image Processing
% ; CALLING SEQUENCE:
% ; result = track( positionlist, maxdisp, param )
% ;  set all keywords in the space below
% ; INPUTS:
% ; positionlist: an array listing the scrambled coordinates and data
% ;     of the different particles at different times, such that:
% ;  positionlist(0:d-1,*): contains the d coordinates and
% ;     data for all the particles, at the different times. must be
positve
% ;  positionlist(d,*): contains the time t that the position
% ;     was determined, must be integers (e.g. frame number.  These values
must
% ;                be monotonically increasing and uniformly gridded in
time.
% ; maxdisp: an estimate of the maximum distance that a particle
% ;     would move in a single time interval.(see Restrictions)
%  OPTIONAL INPUT:
%   param:  a structure containing a few tracking parameters that are
%       needed for many applications.  If param is not included in the
%       function call, then default values are used.  If you set one value
%       make sure you set them all:
% ;         param.mem: this is the number of time steps that a particle
can be
% ;             'lost' and then recovered again.  If the particle
reappears
% ;             after this number of frames has elapsed, it will be
% ;             tracked as a new particle. The default setting is zero.
% ;             this is useful if particles occasionally 'drop out' of
% ;             the data.
% ;         param.dim: if the user would like to unscramble non-coordinate
data
% ;             for the particles (e.g. apparent radius of gyration for
% ;             the particle images), then positionlist should
% ;             contain the position data in positionlist(0:param.dim-1,*)
% ;             and the extra data in positionlist(param.dim:d-1,*). It is
then
% ;             necessary to set dim equal to the dimensionality of the
% ;             coordinate data to so that the track knows to ignore the
% ;             non-coordinate data in the construction of the
% ;             trajectories. The default value is two.
% ;         param.good: set this keyword to eliminate all trajectories
with
% ;             fewer than param.good valid positions.  This is useful
% ;             for eliminating very short, mostly 'lost' trajectories
% ;             due to blinking 'noise' particles in the data stream.
%;          param.quiet: set this keyword to 1 if you don't want any text
% ; OUTPUTS:
% ; result:  a list containing the original data rows sorted
% ;     into a series of trajectories.  To the original input
% ;     data structure there is appended an additional column
% ;     containing a unique 'id number' for each identified
% ;     particle trajectory.  The result array is sorted so
% ;     rows with corresponding id numbers are in contiguous
% ;     blocks, with the time variable a monotonically
% ;     increasing function inside each block.  For example:
% ;
% ;     For the input data structure (positionlist):
% ;         (x)      (y)      (t)
% ;     pos = 3.60000      5.00000      0.00000
% ;           15.1000      22.6000      0.00000
% ;           4.10000      5.50000      1.00000
% ;           15.9000      20.7000      2.00000
% ;           6.20000      4.30000      2.00000
% ;
% ;     IDL> res = track(pos,5,mem=2)
% ;
% ;     track will return the result 'res'
% ;         (x)      (y)      (t)          (id)
% ;     res = 3.60000      5.00000      0.00000      0.00000
% ;           4.10000      5.50000      1.00000      0.00000
% ;           6.20000      4.30000      2.00000      0.00000
% ;           15.1000      22.6000      0.00000      1.00000
% ;           15.9000      20.7000      2.00000      1.00000
% ;
% ;     NB: for t=1 in the example above, one particle temporarily
% ;     vanished.  As a result, the trajectory id=1 has one time
% ;     missing, i.e. particle loss can cause time gaps to occur
% ;     in the corresponding trajectory list. In contrast:
% ;
% ;     IDL> res = track(pos,5)
% ;
% ;     track will return the result 'res'
% ;         (x)      (y)      (t)          (id)
% ;     res = 15.1000      22.6000      0.00000      0.00000
% ;                   3.60000      5.00000      0.00000      1.00000
% ;               4.10000      5.50000      1.00000      1.00000
% ;                6.20000      4.30000      2.00000      1.00000
% ;                15.9000      20.7000      2.00000      2.00000
% ;
% ;     where the reappeared 'particle' will be labelled as new
% ;     rather than as a continuation of an old particle since
% ;     mem=0.  It is up to the user to decide what setting of
% ;     'mem' will yeild the highest fidelity .
% ;
% ; SIDE EFFECTS:
% ; Produces informational messages.  Can be memory intensive for
% ; extremely large data sets.
% ; RESTRICTIONS:
% ; maxdisp should be set to a value somewhat less than the mean
% ; spacing between the particles. As maxdisp approaches the mean
% ; spacing the runtime will increase significantly. The function
% ; will produce an error message: "Excessive Combinatorics!" if
% ; the run time would be too long, and the user should respond
% ; by re-executing the function with a smaller value of maxdisp.
% ; Obviously, if the particles being tracked are frequently moving
% ; as much as their mean separation in a single time step, this
% ; function will not return acceptable trajectories.
% ; PROCEDURE:
% ; Given the positions for n particles at time t(i), and m possible
% ; new positions at time t(i+1), this function considers all possible
% ; identifications of the n old positions with the m new positions,
% ; and chooses that identification which results in the minimal total
% ; squared displacement. Those identifications which don't associate
% ; a new position within maxdisp of an old position ( particle loss )
% ; penalize the total squared displacement by maxdisp^2. For non-
% ; interacting Brownian particles with the same diffusivity, this
% ; algorithm will produce the most probable set of identifications
% ; ( provided maxdisp >> RMS displacement between frames ).
% ; In practice it works reasonably well for systems with oscillatory,
% ; ballistic, correlated and random hopping motion, so long as single
% ; time step displacements are reasonably small.  NB: multidimensional
% ; functionality is intended to facilitate tracking when additional
% ; information regarding target identity is available (e.g. size or
% ; color).  At present, this information should be rescaled by the
% ; user to have a comparable or smaller (measurement) variance than
% ; the spatial displacements.
% ;
% ; MODIFICATION HISTORY:
% ;  2/93 Written by John C. Crocker, University of Chicago (JFI).
% ;  7/93 JCC fixed bug causing particle loss and improved performance
% ;     for large numbers of (>100) particles.
% ; 11/93 JCC improved speed and memory performance for large
% ;     numbers of (>1000) particles (added subnetwork code).
% ;  3/94 JCC optimized run time for trivial bonds and d<7. (Added
% ;     d-dimensional raster metric code.)
% ;  8/94 JCC added functionality to unscramble non-position data
% ;     along with position data.
% ;  9/94 JCC rewrote subnetwork code and wrote new, more efficient
% ;     permutation code.
% ;  5/95 JCC debugged subnetwork and excessive combinatorics code.
% ; 12/95 JCC added memory keyword, and enabled the tracking of
% ;     newly appeared particles.
% ;  3/96 JCC made inipos a keyword, and disabled the adding of 'new'
% ;     particles when inipos was set.
% ;  3/97 JCC added 'add' keyword, since Chicago users didn't like
% ;     having particle addition be the default.
% ;  9/97 JCC added 'goodenough' keyword to improve memory efficiency
% ;     when using the 'add' keyword and to filter out bad tracks.
% ;       10/97 JCC streamlined data structure to speed runtime for >200
% ;               timesteps.  Changed 'quiet' keyword to 'verbose'. Made
% ;               time labelling more flexible (uniform and sorted is ok).
% ;  9/98 JCC switched trajectory data structure to a 'list' form,
% ;     resolving memory issue for large, noisy datasets.
% ;  2/99 JCC added Eric Weeks's 'uberize' code to post-facto
% ;     rationalize the particle id numbers, removed 'add' keyword.
% ;  1/05 Transmuted to MATLAB by D. Blair
% ;  5/05  ERD Added the param structure to simplify calling.
%    6/05  ERD Added quiet to param structure
%    7/05  DLB Fixed slight bug in trivial bond code
%    3/07  DLB Fixed bug with max disp pointed out by Helene Delanoe-Ayari
%
% ; This code 'track.pro' is copyright 1999, by John C. Crocker.
% ; It should be considered 'freeware'- and may be distributed freely
% ; (outside of the military-industrial complex) in its original form
% ; when properly attributed.
% ;
% ;-

dd = length(xyzs(1,:));

%use default parameters if none given
if nargin==2
    %default values
    memory_b=0; % if mem is not needed set to zero
    goodenough = 0;  % if goodenough is not wanted set to zero
    dim = dd - 1;
    quiet=0;
else
    memory_b    =   param.mem;
    goodenough  =   param.good;
    dim         =   param.dim;
    quiet       =   param.quiet;
end


% checking the input time vector
t = xyzs(:,dd);
st = circshift(t,1);
st = t(2:end) - st(2:end);
if  sum(st(find(st < 0))) ~= 0
    disp('The time vectors is not in order')
    return
end
info = 1;

w = find(st > 0);
z = length(w);
z = z +1;
if isempty(w)
```

```
    disp('All positions are at the same time... go back!')
    return
end

% partitioning the data with unique times

%res = unq(t);
% implanting unq directly
    indices = find(t ~= circshift(t,-1));
        count = length(indices);
        if count > 0
            res = indices;
        else
            res = length(t) -1;
        end
 %%%%%%%%%%%%%%%%%%%%%%%

res = [1,res',length(t)];
ngood = res(2) - res(1) + 1;
eyes = 1:ngood;
pos = xyzs(eyes,1:dim);
istart = 2;
n = ngood;

zspan = 50;
if n > 200
    zspan = 20;
end
if n > 500
    zspan = 10;
end
resx = zeros(zspan,n) - 1;

bigresx = zeros(z,n) - 1;
mem = zeros(n,1);
%  whos resx
%  whos bigresx
uniqid = 1:n;
maxid = n;
olist = [0.,0.];

if goodenough > 0
    dumphash = zeros(n,1);
    nvalid = ones(n,1);
end

%  whos eyes;
resx(1,:) = eyes;
% setting up constants
maxdisq = maxdisp^2;

% John calls this the setup for "fancy code" ???
notnsqrd = (sqrt(n*ngood) > 200) & (dim < 7);
notnsqrd = notnsqrd(1);
```

```
if notnsqrd
    %;   construct the vertices of a 3x3x3... d-dimensional hypercube

    cube = zeros(3^dim,dim);


    for d=0:dim-1,
        numb = 0;
        for j=0:(3^d):(3^dim)-1,
            cube(j+1:j+(3^(d)),d+1) = numb;
            numb = mod(numb+1,3);
        end
    end

    %   calculate a blocksize which may be greater than maxdisp, but which
    %   keeps nblocks reasonably small.

    volume = 1;
    for d = 0:dim-1
        minn = min(xyzs(w,d+1));
        maxx = max(xyzs(w,d+1));
        volume = volume * (maxx-minn);
    end
    volume;
    blocksize = max( [maxdisp,((volume)/(20*ngood))^(1.0/dim)] );
end
%   Start the main loop over the frames.
for i=istart:z
    ispan = mod(i-1,zspan)+1;
    %disp(ispan)
    % get new particle positions
    m = res(i+1) - res(i);
    res(i);
    eyes = 1:m;
    eyes = eyes + res(i);

    if m > 0

        xyi = xyzs(eyes,1:dim);
        found = zeros(m,1);

        % THE TRIVIAL BOND CODE BEGINS

        if notnsqrd
            %Use the raster metric code to do trivial bonds

            % construct "s", a one dimensional parameterization of the
space
            % which consists of the d-dimensional raster scan of the
volume.)

            abi = fix(xyi./blocksize);
            abpos = fix(pos./blocksize);
            si = zeros(m,1);
            spos = zeros(n,1);
            dimm = zeros(dim,1);
            coff = 1.;

            for j=1:dim
                minn = min([abi(:,j);abpos(:,j)]);
                maxx = max([abi(:,j);abpos(:,j)]);
                abi(:,j) = abi(:,j) - minn;
                abpos(:,j) = abpos(:,j) - minn;
                dimm(j) = maxx-minn + 1;
                si = si + abi(:,j).*coff;
                spos = spos + abpos(:,j).*coff;
                coff = dimm(j).*coff;
            end
            nblocks = coff;
            % trim down (intersect) the hypercube if its too big to fit in
the
            % particle volume. (i.e. if dimm(j) lt 3)

            cub = cube;
            deg = find( dimm < 3);
            if ~isempty(deg)
                for j = 0:length(deg)-1
                    cub = cub(find(cub(:,deg(j+1)) < dimm(deg(j+1))),:);
                end
            end

            % calculate the "s" coordinates of hypercube (with a corner @
the origin)
            scube = zeros(length(cub(:,1)),1);
            coff = 1;
            for j=1:dim
                scube = scube + cub(:,j).*coff;
                coff = coff*dimm(j);
            end

            % shift the hypercube "s" coordinates to be centered around
the origin

            coff = 1;
            for j=1:dim
                if dimm(j) > 3
                    scube = scube - coff;
                end
                coff = dimm(j).* coff;
            end
            scube = mod((scube + nblocks),nblocks);
            % get the sorting for the particles by their "s" positions.
            [ed,isort] = sort(si);

            % make a hash table which will allow us to know which new
particles
            % are at a given si.
            strt = zeros(nblocks,1) -1;
            fnsh = zeros(nblocks,1);
            h = find(si == 0);
            lh = length(h);
            if lh > 0

            si(h) = 1;
            end

            for j=1:m
                if strt(si(isort(j))) == -1
                    strt(si(isort(j))) = j;
                    fnsh(si(isort(j))) = j;
                else
                    fnsh(si(isort(j))) = j;
                end
            end
            if lh > 0
            si(h) = 0;
            end
            coltot = zeros(m,1);
            rowtot = zeros(n,1);
            which1 = zeros(n,1);
            for j=1:n


                map = fix(-1);

                scub_spos = scube + spos(j);
                s = mod(scub_spos,nblocks);
                whzero = find(s == 0 );
                if ~isempty(whzero)
                    nfk = find(s ~=0);
                    s = s(nfk);
                end

                w = find(strt(s) ~= -1);

                ngood = length(w);
                ltmax=0;
                if ngood ~= 0

                    s = s(w);
                    for k=1:ngood
                        map = [map;isort( strt(s(k)):fnsh(s(k)))];
                    end
                    map = map(2:end);
%                     if length(map) == 2
%                         if (map(1) - map(2)) == 0
%                             map = unique(map);
%                          end
%                     end
                    %   map = map(umap);
                    %end
                    % find those trival bonds
                    distq = zeros(length(map),1);
                    for d=1:dim
                        distq = distq + (xyi(map,d) - pos(j,d)).^2;
                    end
                    ltmax = distq < maxdisq;

                    rowtot(j) = sum(ltmax);

                    if rowtot(j) >= 1
                        w = find(ltmax == 1);
                        coltot( map(w) ) = coltot( map(w)) +1;
                        which1(j) = map( w(1) );
                    end
                end

            end


            ntrk = fix(n - sum(rowtot == 0));

            w = find( rowtot == 1);
            ngood = length(w);


            if ngood ~= 0
                ww = find(coltot( which1(w) ) == 1);
                ngood = length(ww);
                if ngood ~= 0
                     %disp(size(w(ww)))
                    resx(ispan,w(ww)) = eyes( which1(w(ww)));
                    found( which1( w(ww))) = 1;
                    rowtot( w(ww)) = 0;
                    coltot( which1(w(ww))) = 0;
                end
            end

            labely = find( rowtot > 0);
            ngood = length(labely);
            if ngood ~= 0
                labelx = find( coltot > 0);

                nontrivial = 1;
            else
                nontrivial = 0;
            end

        else

            %   or: Use simple N^2 time routine to calculate trivial bonds

            % let's try a nice, loopless way!
            % don't bother tracking perm. lost guys.
            wh = find( pos(:,1) >= 0);
            ntrack = length(wh);
            if ntrack == 0
                'There are no valid particles to track idiot!'
                break
            end
```

```
            xmat = zeros(ntrack,m);
            count = 0;
            for kk=1:ntrack
                for ll=1:m
                    xmat(kk,ll) = count;
                    count = count+1;
                end
            end
            count = 0;
            for kk=1:m
                for ll=1:ntrack
                    ymat(kk,ll) = count;
                    count = count+1;
                end
            end

            xmat = (mod(xmat,m) + 1);
            ymat = (mod(ymat,ntrack) +1)';
            [lenxn,lenxm] = size(xmat);
%            whos ymat
%            whos xmat
%            disp(m)

            for d=1:dim
                x = xyi(:,d);
                y = pos(wh,d);
                xm = x(xmat);
                ym = y(ymat(1:lenxn,1:lenxm));
                if size(xm) ~= size(ym)
                    xm = xm';
                end

                if d == 1
                    dq = (xm -ym).^2;
                    %dq = (x(xmat)-y(ymat(1:lenxn,1:lenxm))).^2;
                else
                    dq = dq + (xm-ym).^2;
                    %dq = dq + (x(xmat)-y(ymat(1:lenxn,1:lenxm)) ).^2;
                end
            end

            ltmax = dq < maxdisq;

            % figure out which trivial bonds go with which

            rowtot = zeros(n,1);
            rowtot(wh) = sum(ltmax,2);


            if ntrack > 1
                coltot = sum(ltmax,1);
            else
                coltot = ltmax;
            end
            which1 = zeros(n,1);
```

```
    for j=1:ntrack
        [mx, w] = max(ltmax(j,:));
        which1(wh(j)) = w;
    end

    ntrk = fix( n - sum(rowtot == 0));
    w= find( rowtot == 1) ;
    ngood = length(w);
    if ngood ~= 0
        ww = find(coltot(which1(w)) == 1);
        ngood = length(ww);
        if ngood ~= 0
            resx( ispan, w(ww) ) = eyes( which1( w(ww)));
            found(which1( w(ww))) = 1;
            rowtot(w(ww)) = 0;
            coltot(which1(w(ww))) = 0;
        end
    end

    labely = find( rowtot > 0);
    ngood = length(labely);

    if ngood ~= 0
        labelx = find( coltot > 0);
        nontrivial = 1;
    else
        nontrivial = 0;
    end
end

%THE TRIVIAL BOND CODE ENDS

if nontrivial

    xdim = length(labelx);
    ydim = length(labely);

    %  make a list of the non-trivial bonds

    bonds = zeros(1,2);
    bondlen = 0;

    for j=1:ydim
        distq = zeros(xdim,1);

        for d=1:dim
            %distq
            distq = distq + (xyi(labelx,d) - pos(labely(j),d)).^2;
            %distq
        end

        w= find(distq <  maxdisq)' - 1;
        ngood = length(w);
        newb = [w;(zeros(1,ngood)+j)];
```

```
            bonds = [bonds;newb'];

            bondlen = [ bondlen;distq( w + 1) ];

        end
        bonds = bonds(2:end,:);

        bondlen = bondlen(2:end);
        numbonds = length(bonds(:,1));
        mbonds = bonds;
        max([xdim,ydim]);


        if max([xdim,ydim]) < 4
            nclust = 1;
            maxsz = 0;
            mxsz = xdim;
            mysz = ydim;
            bmap = zeros(length(bonds(:,1)+1),1) - 1;

        else


            %   THE SUBNETWORK CODE BEGINS
            lista = zeros(numbonds,1);
            listb = zeros(numbonds,1);
            nclust = 0;
            maxsz = 0;
            thru = xdim;

            while thru ~= 0
                %  the following code extracts connected
                %   sub-networks of the non-trivial
                %   bonds.  NB: lista/b can have redundant entries due
to
                %   multiple-connected subnetworks


                w = find(bonds(:,2) >= 0);
   %             size(w)

                lista(1) = bonds(w(1),2);
                listb(1) = bonds(w(1),1);
                bonds(w(1),:) = -(nclust+1);
                bonds;
                adda = 1;
                addb = 1;
                donea = 0;
                doneb = 0;
                if (donea ~= adda) | (doneb ~= addb)
                    true = 0;
                else
```

```
true = 1;
end

while ~true

    if (donea ~= adda)
        w = find(bonds(:,2) == lista(donea+1));
        ngood = length(w);
        if ngood ~= 0
            listb(addb+1:addb+ngood,1) = bonds(w,1);
            bonds(w,:) = -(nclust+1);
            addb = addb+ngood;
        end
        donea = donea+1;
    end
    if (doneb ~= addb)
        w = find(bonds(:,1) == listb(doneb+1));
        ngood = length(w);
        if ngood ~= 0
            lista(adda+1:adda+ngood,1) = bonds(w,2);
            bonds(w,:) = -(nclust+1);
            adda = adda+ngood;
        end
        doneb = doneb+1;
    end
  if (donea ~= adda) | (doneb ~= addb)
      true = 0;
  else
      true = 1;
  end
end

[pp,pqx] = sort(listb(1:doneb));
%unx =  unq(listb(1:doneb),pqx);
%implanting unq directly
    arr = listb(1:doneb);
    q = arr(pqx);
    indices = find(q ~= circshift(q,-1));
    count = length(indices);
    if count > 0
        unx = pqx(indices);
    else
        unx = length(q) -1;
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

xsz = length(unx);
[pp,pqy] = sort(lista(1:donea));
%uny =  unq(lista(1:donea),pqy);
%implanting unq directly
    arr = lista(1:donea);
    q = arr(pqy);
    indices = find(q ~= circshift(q,-1));
    count = length(indices);
    if count > 0
        uny = pqy(indices);
```

```
            else
                uny = length(q) -1;
            end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%




        ysz = length(uny);
        if xsz*ysz > maxsz
            maxsz = xsz*ysz;
            mxsz = xsz;
            mysz = ysz;
        end


        thru = thru -xsz;
        nclust = nclust + 1;
    end
    bmap = bonds(:,2);
end
% THE SUBNETWORK CODE ENDS
% put verbose in for Jaci

%   THE PERMUTATION CODE BEGINS

for nc =1:nclust
    w = find( bmap == -1*(nc));

    nbonds = length(w);
    bonds = mbonds(w,:);
    lensq = bondlen(w);
    [pq,st] = sort( bonds(:,1));
    %un = unq(bonds(:,1),st);
       %implanting unq directly
            arr = bonds(:,1);
            q = arr(st);
            indices = find(q ~= circshift(q,-1));
            count = length(indices);
            if count > 0
                un = st(indices);
            else
                un = length(q) -1;
            end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    uold = bonds(un,1);

    nold = length(uold);

    %un = unq(bonds(:,2));

    %implanting unq directly
```

```
                indices = find(bonds(:,2) ~= circshift(bonds(:,2),-1));
                count = length(indices);
                    if count > 0
                        un = indices;
                    else
                        un = length(bonds(:,2)) -1;
                    end
                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

                unew = bonds(un,2);
                nnew = length(unew);

                if nnew > 5
                    rnsteps = 1;
                    for ii =1:nnew
                        rnsteps = rnsteps * length( find(bonds(:,2) == ...
                            unew(ii)));
                        if rnsteps > 5.e+4
                            disp('Warning: difficult combinatorics
encountered.')
                        end
                        if rnsteps > 2.e+5
                            disp(['Excessive Combinitorics you FOOL LOOK
WHAT YOU HAVE' ...
                                    ' DONE TO ME!!!'])
                            return
                        end
                    end
                end
                st = zeros(nnew,1);
                fi = zeros(nnew,1);
                h = zeros(nbonds,1);
                ok = ones(nold,1);
                nlost = (nnew - nold) > 0;


                for ii=1:nold
                    h(find(bonds(:,1) == uold(ii))) = ii;
                end
                st(1) = 1 ;
                fi(nnew) = nbonds; % check this later
                if nnew > 1
                    sb = bonds(:,2);
                    sbr = circshift(sb,1);
                    sbl = circshift(sb,-1);
                    st(2:end) = find( sb(2:end) ~= sbr(2:end)) + 1;
                    fi(1:nnew-1) = find( sb(1:nbonds-1) ~= sbl(1:nbonds-
1));
                end
%                if i-1 == 13
%                    hi
%                end
                checkflag = 0;
                while checkflag ~= 2

                    pt = st -1;
                    lost = zeros(nnew,1);
                    who = 0;
                    losttot = 0;
                    mndisq = nnew*maxdisq;


                    while who ~= -1

                        if pt(who+1) ~= fi(who+1)


                            w = find( ok( h( pt( who+1 )+1:fi( who+1 ) ) )
); % check this -1

                            ngood = length(w);
                            if ngood > 0
                                if pt(who+1) ~= st(who+1)-1
                                    ok(h(pt(who+1))) = 1;
                                end
                                pt(who+1) = pt(who+1) + w(1);
                                ok(h(pt(who+1))) = 0;
                                if who == nnew -1
                                    ww = find( lost == 0);
                                    dsq = sum(lensq(pt(ww))) +
losttot*maxdisq;


                                    if dsq < mndisq
                                        minbonds = pt(ww);
                                        mndisq = dsq;
                                    end
                                else
                                    who = who+1;
                                end
                            else
                                if ~lost(who+1) & (losttot ~= nlost)
                                    lost(who+1) = 1;
                                    losttot = losttot + 1;
                                    if pt(who+1) ~= st(who+1) -1;
                                        ok(h(pt(who+1))) = 1;
                                    end
                                    if who == nnew-1
                                        ww = find( lost == 0);
                                        dsq = sum(lensq(pt(ww))) +
losttot*maxdisq;

                                        if dsq < mndisq
                                            minbonds = pt(ww);
                                            mndisq = dsq;
                                        end
                                    else
                                       who = who + 1;
                                    end

                                else
                                    if pt(who+1) ~= (st(who+1) -1)
                                        ok(h(pt(who+1))) = 1;
                                    end
                                    pt(who+1) = st(who+1) -1;
                                    if lost(who+1)
                                        lost(who+1) = 0;
                                        losttot = losttot -1;
                                    end
                                    who = who -1;
                                end
                            end
                        else
                            if ~lost(who+1) & (losttot ~= nlost)
                                lost(who+1) = 1;
                                losttot = losttot + 1;
                                if pt(who+1) ~= st(who+1)-1
                                    ok(h(pt(who+1))) = 1;
                                end
                                if who == nnew -1
                                    ww = find( lost == 0);
                                    dsq = sum(lensq(pt(ww))) +
losttot*maxdisq;

                                    if dsq < mndisq
                                        minbonds = pt(ww);
                                        mndisq = dsq;
                                    end
                                else
                                    who = who + 1;
                                end
                            else
                                if pt(who+1) ~= st(who+1) -1
                                    ok(h(pt(who+1))) = 1;
                                end
                                pt(who+1) = st(who+1) -1;
                                if lost(who+1)
                                    lost(who+1) = 0;
                                    losttot = losttot -1;
                                end
                                who = who -1;
                            end
                        end
                    end

                    checkflag = checkflag + 1;
                    if checkflag == 1
                        plost = min([fix(mndisq/maxdisq) , (nnew -1)]);
                        if plost > nlost
                            nlost = plost;
                        else
                            checkflag = 2;
                        end
                    end

                end
                %   update resx using the minimum bond configuration

                resx(ispan,labely(bonds(minbonds,2))) =
eyes(labelx(bonds(minbonds,1)+1));
```

```
                found(labelx(bonds(minbonds,1)+1)) = 1;

            end

            %   THE PERMUTATION CODE ENDS
        end

        w = find(resx(ispan,:) >= 0);
        nww = length(w);

        if nww > 0
            pos(w,:) = xyzs( resx(ispan,w) , 1:dim);
            if goodenough > 0
                nvalid(w) = nvalid(w) + 1;
            end
        end  %go back and add goodenough keyword thing
        newguys = find(found == 0);

       nnew = length(newguys);

        if (nnew > 0) % & another keyword to workout inipos
            newarr = zeros(zspan,nnew) -1;
            resx = [resx,newarr];

            resx(ispan,n+1:end) = eyes(newguys);
            pos = [[pos];[xyzs(eyes(newguys),1:dim)]];
            nmem = zeros(nnew,1);
            mem = [mem;nmem];
            nun = 1:nnew;
            uniqid = [uniqid,((nun) + maxid)];
            maxid = maxid + nnew;
            if goodenough > 0
                dumphash = [dumphash;zeros(1,nnew)'];
                nvalid = [nvalid;zeros(1,nnew)'+1];
            end
            % put in goodenough
            n = n + nnew;

        end

    else
        ' Warning- No positions found for t='
    end
    w = find( resx(ispan,:) ~= -1);
    nok = length(w);
    if nok ~= 0
        mem(w) =0;
    end

    mem = mem + (resx(ispan,:)' == -1);
    wlost = find(mem == memory_b+1);
    nlost =length(wlost);

    if nlost > 0
        pos(wlost,:) = -maxdisp;
        if goodenough > 0
            wdump = find(nvalid(wlost) < goodenough);
            ndump = length(wdump);
            if ndump > 0
                dumphash(wlost(wdump)) = 1;
            end
        end
        % put in goodenough keyword stuff if
    end
    if (ispan == zspan) | (i == z)
        nold = length(bigresx(1,:));
        nnew = n-nold;
        if nnew > 0
            newarr = zeros(z,nnew) -1;
            bigresx = [bigresx,newarr];
        end
        if goodenough > 0
            if (sum(dumphash)) > 0
                wkeep = find(dumphash == 0);
                nkeep = length(wkeep);
                resx = resx(:,wkeep);
                bigresx = bigresx(:,wkeep);
                pos = pos(wkeep,:);
                mem = mem(wkeep);
                uniqid = uniqid(wkeep);
                nvalid = nvalid(wkeep);
                n = nkeep;
                dumphash = zeros(nkeep,1);
            end
        end

        % again goodenough keyword
        if quiet~=1
            disp(strcat(num2str(i), ' of ' ,num2str(z), ' done.  Tracking
',num2str(ntrk),' particles  ', num2str(n),' tracks total'));
        end
        bigresx(i-(ispan)+1:i,:) = resx(1:ispan,:);
        resx = zeros(zspan,n) - 1;


        wpull = find(pos(:,1) == -maxdisp);
        npull = length(wpull);

        if npull > 0
            lillist = zeros(1,2);
            for ipull=1:npull
                wpull2 = find(bigresx(:,wpull(ipull)) ~= -1);
                npull2 = length(wpull2);
                thing =
[bigresx(wpull2,wpull(ipull)),zeros(npull2,1)+uniqid(wpull(ipull))];
                lillist = [lillist;thing];

            end
            olist = [[olist];[lillist(2:end,:)]];

        end
```

```
            wkeep = find(pos(:,1) >= 0);
            nkeep = length(wkeep);
            if nkeep == 0
                    'Were going to crash now, no particles....'
            end
            resx = resx(:,wkeep);
            bigresx = bigresx(:,wkeep);
            pos = pos(wkeep,:);
            mem = mem(wkeep);
            uniqid = uniqid(wkeep);
            n = nkeep;
            dumphash = zeros(nkeep,1);
            if goodenough > 0
                nvalid = nvalid(wkeep);
            end
        end

    end

    if goodenough > 0
        nvalid = sum(bigresx >= 0 ,1);
        wkeep = find(nvalid >= goodenough);
        nkeep = length(wkeep);
        if nkeep == 0
            for i=1:10
            disp('You are not going any further, check your params and data')
            end
            disp('the code broke at line 1045')
            return
        end
        if nkeep < n
            bigresx = bigresx(:,wkeep);
            n = nkeep;
            uniqid = uniqid(wkeep);
            pos = pos(wkeep,:);
        end
    end


    wpull = find( pos(:,1) ~= -2*maxdisp);
    npull = length(wpull);
    if npull > 0
        lillist = zeros(1,2);
        for ipull=1:npull
            wpull2 = find(bigresx(:,wpull(ipull)) ~= -1);
            npull2 = length(wpull2);
            thing =
    [bigresx(wpull2,wpull(ipull)),zeros(npull2,1)+uniqid(wpull(ipull))];
            lillist = [lillist;thing];
        end
        olist = [olist;lillist(2:end,:)];
    end
```

```
olist = olist(2:end,:);
%bigresx = 0;
%resx = 0;

nolist = length(olist(:,1));
res = zeros(nolist,dd+1);
for j=1:dd
    res(:,j) = xyzs(olist(:,1),j);
end
res(:,dd+1) = olist(:,2);

% this is uberize included for simplicity of a single monolithic code

ndat=length(res(1,:));
newtracks=res;


%u=unq(newtracks(:,ndat));

% inserting unq
indices = find(newtracks(:,ndat) ~= circshift(newtracks(:,ndat),-1));
        count = length(indices);
        if count > 0
            u = indices;
        else
            u = length(newtracks(:,ndat)) -1;
        end


ntracks=length(u);
u=[0;u];
for i=2:ntracks+1
    newtracks(u(i-1)+1:u(i),ndat) = i-1;
end

% end of uberize code

tracks = newtracks;
```

APPENDIX G

STRAIN TRACKING PROGRAM

```
%% Microchannel striain from bead tracking
%copyright Lucas Hofmeister 2011 all rights reserved
clear all
close all
clc
FrameWidth = 1280+20; %video resolution in pixels
FrameHeight = 980+20; %videwo resolution in pixels

folder = 'E:\2011_10_20_StrainTracking\Attempt1'; %path on laptop
```

```
%folder = 'K:\2011_10_24_StrainTracking'; %path on desktop

fileName = 'ActuatorMove1001';
B = load([folder,'\',fileName]);
%Bdist = load([folder,'\',fileName,'_Dist']);
%Binput = load([folder,'\',fileName,'_PointsIn']);
[num,time] = size(B);

%this will round the x and y points in the feature position data to the
%nearest pixel

xdata = round(B(:,1:2:time-3))';
ydata = round(B(:,2:2:time-2))';

[Gxx,Gxy] = gradient(xdata); %Gxy is the x gradient along each column
(pixels/frame)
[Gyx,Gyy] = gradient(ydata); %Gyy is the y gradient along each column
(pixels/frame)


% StrainY = zeros(FrameHeight,FrameWidth);

[row,col] = size(xdata);

%this section pulls out all of the frames where there is a change in
%position of the beads. I have to do this because the files saved as
100fps
%but are actually sampled less than that
MeanGx = mean(Gxy')';
indices = find(MeanGx~=0);
rows = indices(1:2:length(indices));
% MeanGy = mean(Gyy')';
% rows = find(MeanGy~=0);

%jason's method

% CompRow = find(Gxy~=0); %find the nonzero points in the x gradient data
% holder = diff(CompRow);
% ind_holder = find(holder<0);
% rows = CompRow(1:ind_holder(1));
%may need to do this for Gyy too

%vid = zeros(FrameHeight,FrameWidth,length(rows)); %this would initialize
%vid, but the size is actually different than the resolution
n=1;
r = 1;
c = 1;
for r = 1:1:length(rows)    %:14:row-15;
    StrainX = zeros(FrameHeight,FrameWidth);
%     StrainY = zeros(FrameHeight,FrameWidth);
    for c = 1:1:col;
        StrainX(abs(ydata(rows(r),c)),abs(xdata(rows(r),c))) =
Gxy(rows(r),c);
%         StrainY(abs(ydata(rows(r),c)),abs(xdata(rows(r),c))) =
Gyy(rows(r),c);
```

```
    end


%this section contains the parameters for performing the dilation
% MN = [10,100]; %shape parameters for rectangular dilation
% SE = strel('ball', 5,0,0); %grayscale dilation using a ball shape with
radius 50 and height 1
%SE = strel('rectangle', MN); %grayscale dilation using a rectangle

% StrainX_dilate =  imdilate(abs(StrainX), SE ,'same');
% figure
% imagesc(StrainX_dilate)

%SX = StrainX_dilate;
SX = abs(StrainX);

% this section prepares variables for TriScatteredInterp
NNZ = nonzeros(SX); %nonzero values of the dilated displacement
[I,J] = find(SX ~= 0); %locations of nonzero values I=row J=column

%V = SX(find(SX~=0)); %an alternative to the 'nonzeros' function

I2 = 1:size(SX,1);
J2 = 1:size(SX,2);
[I2,J2] = meshgrid(I2,J2); %this creates a grid to use for the final
interpolation

X = [I,J];
Z  = TriScatteredInterp(X,abs(NNZ), 'linear');
Strain = Z(I2',J2');
figure
imagesc(Strain)
% montage(Strain)

% to make an image stack for a video
vid(:,:,n) = Strain;
n=n+1;
% clear StrainX StrainY Strain StrainX_dilate Z NNZ SX
end

[Fx Fy] = gradient(vid(:,:,2));
[row col frames] = size(Fx);

row = 1:1:row;
col = 1:1:col;
figure
imagesc(vid(:,:,2))
hold on
quiver(row,col, Fx, Fy)
```

REFERENCES

1. JAKLENEC, A., ET AL., *PROGRESS IN THE TISSUE ENGINEERING AND STEM CELL INDUSTRY "ARE WE THERE YET?".* TISSUE ENGINEERING. PART B, REVIEWS, 2012. **18**(3): P. 155-66.
2. LUNDBERG, M.S., *CARDIOVASCULAR TISSUE ENGINEERING RESEARCH SUPPORT AT THE NATIONAL HEART, LUNG, AND BLOOD INSTITUTE.* CIRCULATION RESEARCH, 2013. **112**(8): P. 1097-103.
3. BAJPAI, V.K. AND S.T. ANDREADIS, *STEM CELL SOURCES FOR VASCULAR TISSUE ENGINEERING AND REGENERATION.* TISSUE ENGINEERING. PART B, REVIEWS, 2012. **18**(5): P. 405-25.
4. ASAI, S., Y. KAMEI, AND S. TORII, *ONE-STAGE RECONSTRUCTION OF INFECTED CRANIAL DEFECTS USING A TITANIUM MESH PLATE ENCLOSED IN AN OMENTAL FLAP.* ANNALS OF PLASTIC SURGERY, 2004. **52**(2): P. 144-7.
5. ATHANASSIADI, K., ET AL., *OMENTAL TRANSPOSITION: THE FINAL SOLUTION FOR MAJOR STERNAL WOUND INFECTION.* ASIAN CARDIOVASCULAR & THORACIC ANNALS, 2007. **15**(3): P. 200-3.
6. BAYLES, S.W. AND R.E. HAYDEN, *GASTRO-OMENTAL FREE FLAP RECONSTRUCTION OF THE HEAD AND NECK.* ARCHIVES OF FACIAL PLASTIC SURGERY : OFFICIAL PUBLICATION FOR THE AMERICAN ACADEMY OF FACIAL PLASTIC AND RECONSTRUCTIVE SURGERY, INC. AND THE INTERNATIONAL FEDERATION OF FACIAL PLASTIC SURGERY SOCIETIES, 2008. **10**(4): P. 255-9.
7. FALAGAS, M.E. AND E.S. ROSMARAKIS, *RECURRENT POST-STERNOTOMY MEDIASTINITIS.* THE JOURNAL OF INFECTION, 2006. **52**(5): P. E151-4.
8. GOLDSMITH, H.S., W.F. CHEN, AND S.W. DUCKETT, *BRAIN VASCULARIZATION BY INTACT OMENTUM.* ARCHIVES OF SURGERY, 1973. **106**(5): P. 695-8.
9. MALONEY, C.T., JR., ET AL., *FREE OMENTAL TISSUE TRANSFER FOR EXTREMITY COVERAGE AND REVASCULARIZATION.* PLASTIC AND RECONSTRUCTIVE SURGERY, 2003. **111**(6): P. 1899-904.
10. PATEL, R.S. AND R.W. GILBERT, *UTILITY OF THE GASTRO-OMENTAL FREE FLAP IN HEAD AND NECK RECONSTRUCTION.* CURRENT OPINION IN OTOLARYNGOLOGY & HEAD AND NECK SURGERY, 2009. **17**(4): P. 258-62.
11. SHAH, S., ET AL., *CELLULAR BASIS OF TISSUE REGENERATION BY OMENTUM.* PLOS ONE, 2012. **7**(6): P. E38368.
12. VATANSEV, C., ET AL., *OMENTAL TRANSPOSITION DECREASES ISCHEMIC BRAIN DAMAGE EXAMINED IN A NEW ISCHEMIA MODEL.* EUROPEAN SURGICAL RESEARCH. EUROPAISCHE CHIRURGISCHE FORSCHUNG. RECHERCHES CHIRURGICALES EUROPEENNES, 2003. **35**(4): P. 388-94.
13. WILM, B., ET AL., *THE SEROSAL MESOTHELIUM IS A MAJOR SOURCE OF SMOOTH MUSCLE CELLS OF THE GUT VASCULATURE.* DEVELOPMENT, 2005. **132**(23): P. 5317-28.
14. QUE, J., ET AL., *MESOTHELIUM CONTRIBUTES TO VASCULAR SMOOTH MUSCLE AND MESENCHYME DURING LUNG DEVELOPMENT.* PROC NATL ACAD SCI U S A, 2008. **105**(43): P. 16626-30.
15. MORIMOTO, M., ET AL., *CANONICAL NOTCH SIGNALING IN THE DEVELOPING LUNG IS REQUIRED FOR DETERMINATION OF ARTERIAL SMOOTH MUSCLE CELLS AND SELECTION OF CLARA VERSUS CILIATED CELL FATE.* JOURNAL OF CELL SCIENCE, 2010. **123**(PT 2): P. 213-24.
16. MANNER, J., ET AL., *THE ORIGIN, FORMATION AND DEVELOPMENTAL SIGNIFICANCE OF THE EPICARDIUM: A REVIEW.* CELLS, TISSUES, ORGANS, 2001. **169**(2): P. 89-103.

17. VIRAGH, S., ET AL., *EARLY DEVELOPMENT OF QUAIL HEART EPICARDIUM AND ASSOCIATED VASCULAR AND GLANDULAR STRUCTURES.* ANAT EMBRYOL (BERL), 1993. **188**(4): P. 381-93.
18. VIRAGH, S. AND C.E. CHALLICE, *THE ORIGIN OF THE EPICARDIUM AND THE EMBRYONIC MYOCARDIAL CIRCULATION IN THE MOUSE.* ANAT REC, 1981. **201**(1): P. 157-68.
19. DETTMAN, R.W., ET AL., *COMMON EPICARDIAL ORIGIN OF CORONARY VASCULAR SMOOTH MUSCLE, PERIVASCULAR FIBROBLASTS, AND INTERMYOCARDIAL FIBROBLASTS IN THE AVIAN HEART.* DEV BIOL, 1998. **193**(2): P. 169-81.
20. MANNER, J., ET AL., *THE ORIGIN, FORMATION AND DEVELOPMENTAL SIGNIFICANCE OF THE EPICARDIUM: A REVIEW.* CELLS TISSUES ORGANS, 2001. **169**(2): P. 89-103.
21. MANNER, J., *EXPERIMENTAL STUDY ON THE FORMATION OF THE EPICARDIUM IN CHICK EMBRYOS.* ANAT EMBRYOL (BERL), 1993. **187**(3): P. 281-9.
22. SHELTON, E.L., ET AL., *OMENTAL GRAFTING: A CELL-BASED THERAPY FOR BLOOD VESSEL REPAIR.* JOURNAL OF TISSUE ENGINEERING AND REGENERATIVE MEDICINE, 2012.
23. HERRICK, S.E., ET AL., *HUMAN PERITONEAL ADHESIONS ARE HIGHLY CELLULAR, INNERVATED, AND VASCULARIZED.* J PATHOL, 2000. **192**(1): P. 67-72.
24. QUE, J., ET AL., *MESOTHELIUM CONTRIBUTES TO VASCULAR SMOOTH MUSCLE AND MESENCHYME DURING LUNG DEVELOPMENT.* PROCEEDINGS OF THE NATIONAL ACADEMY OF SCIENCES OF THE UNITED STATES OF AMERICA, 2008. **105**(43): P. 16626-30.
25. HUH, D., ET AL., *A HUMAN DISEASE MODEL OF DRUG TOXICITY-INDUCED PULMONARY EDEMA IN A LUNG-ON-A-CHIP MICRODEVICE.* SCIENCE TRANSLATIONAL MEDICINE, 2012. **4**(159): P. 159RA147.
26. HUH, D., ET AL., *RECONSTITUTING ORGAN-LEVEL LUNG FUNCTIONS ON A CHIP.* SCIENCE, 2010. **328**(5986): P. 1662-8.
27. HUH, D., ET AL., *MICROENGINEERED PHYSIOLOGICAL BIOMIMICRY: ORGANS-ON-CHIPS.* LAB ON A CHIP, 2012. **12**(12): P. 2156-64.
28. SYEDAIN, Z.H., ET AL., *IMPLANTABLE ARTERIAL GRAFTS FROM HUMAN FIBROBLASTS AND FIBRIN USING A MULTI-GRAFT PULSED FLOW-STRETCH BIOREACTOR WITH NONINVASIVE STRENGTH MONITORING.* BIOMATERIALS, 2011. **32**(3): P. 714-22.
29. NIKLASON, L.E., ET AL., *FUNCTIONAL ARTERIES GROWN IN VITRO.* SCIENCE, 1999. **284**(5413): P. 489-93.
30. PATRICK, C.W., JR. AND L.V. MCINTIRE, *SHEAR STRESS AND CYCLIC STRAIN MODULATION OF GENE EXPRESSION IN VASCULAR ENDOTHELIAL CELLS.* BLOOD PURIF, 1995. **13**(3-4): P. 112-24.
31. HARRISON, D.G., ET AL., *INFLAMMATION, IMMUNITY, AND HYPERTENSION.* HYPERTENSION. **57**(2): P. 132-40.
32. GALIS, Z.S. AND J.J. KHATRI, *MATRIX METALLOPROTEINASES IN VASCULAR REMODELING AND ATHEROGENESIS: THE GOOD, THE BAD, AND THE UGLY.* CIRC RES, 2002. **90**(3): P. 251-62.
33. MARCHESI, C., P. PARADIS, AND E.L. SCHIFFRIN, *ROLE OF THE RENIN-ANGIOTENSIN SYSTEM IN VASCULAR INFLAMMATION.* TRENDS PHARMACOL SCI, 2008. **29**(7): P. 367-74.
34. SAVOIA, C. AND E.L. SCHIFFRIN, *INFLAMMATION IN HYPERTENSION.* CURR OPIN NEPHROL HYPERTENS, 2006. **15**(2): P. 152-8.
35. MCDONALD, J.C. AND G.M. WHITESIDES, *POLY(DIMETHYLSILOXANE) AS A MATERIAL FOR FABRICATING MICROFLUIDIC DEVICES.* ACCOUNTS OF CHEMICAL RESEARCH, 2002. **35**(7): P. 491-499.
36. SCHNEIDER, F., ET AL., *MECHANICAL PROPERTIES OF SILICONES FOR MEMS.* JOURNAL OF MICROMECHANICS AND MICROENGINEERING, 2008. **18**(6): P. -.

37. CHOI, K.M. AND J.A. ROGERS, *A PHOTOCURABLE POLY(DIMETHYLSILOXANE) CHEMISTRY DESIGNED FOR SOFT LITHOGRAPHIC MOLDING AND PRINTING IN THE NANOMETER REGIME.* JOURNAL OF THE AMERICAN CHEMICAL SOCIETY, 2003. **125**(14): P. 4060-4061.

38. STUDER, V., ET AL., *SCALING PROPERTIES OF A LOW-ACTUATION PRESSURE MICROFLUIDIC VALVE.* JOURNAL OF APPLIED PHYSICS, 2004. **95**(1): P. 393-398.

39. KAWAGUCHI, M., D.M. BADER, AND B. WILM, *SEROSAL MESOTHELIUM RETAINS VASCULOGENIC POTENTIAL.* DEVELOPMENTAL DYNAMICS : AN OFFICIAL PUBLICATION OF THE AMERICAN ASSOCIATION OF ANATOMISTS, 2007. **236**(11): P. 2973-9.

40. HSU, Y.H., ET AL., *A MICROFLUIDIC PLATFORM FOR GENERATING LARGE-SCALE NEARLY IDENTICAL HUMAN MICROPHYSIOLOGICAL VASCULARIZED TISSUE ARRAYS.* LAB ON A CHIP, 2013.

41. MATSUMOTO, T. AND K. NAGAYAMA, *TENSILE PROPERTIES OF VASCULAR SMOOTH MUSCLE CELLS: BRIDGING VASCULAR AND CELLULAR BIOMECHANICS.* JOURNAL OF BIOMECHANICS, 2012. **45**(5): P. 745-55.