Image-Guided Microvasculature Generation in Photodegradable Hydrogels

Luke Orr

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington
2022

Committee:
Cole DeForest
Shachi Mittal

Program Authorized to Offer Degree:
Chemical Engineering

University of Washington

Abstract

Image-Guided Microvasculature Generation in Photodegradable Hydrogels

Luke Orr

Chair of the Supervisory Committee:

Cole DeForest

Department of Chemical Engineering

The complex vascular networks that exist throughout living organisms are necessary to sustain life, transporting nutrients and oxygen to and waste products from tissues in the blood. Though recent advances in biofabrication have made substantial strides, success in replicating vascular complexities at native resolution and scale in anatomically defined patterns remains limited. In this thesis, I demonstrate that multiphoton sculpting of hyper-photodegradable hydrogels through a tiled image stack-guided methodology rapidly affords perfusable microvascular networks of unprecedented size and complexity. These results have major implications for studying vasculature biology in vitro and in the creation of functional engineered tissues for in vivo transplantation.

Contents

Introduction

Vascularization is a key concept to further develop functional tissue in vitro. Capillary networks make up the complex support system within tissues.[1] These interconnected networks can have diameters as small [2] as 5-10 μm and provide necessary nutrients, oxygen, and transport red blood cells throughout organs and the entire body.[3] Ongoing efforts seek to introduce vascularity within synthetic biomaterials, in particularly hydrogels. Hydrogels are water-swollen polymer networks often used for three-dimensional (3D) cell culture. Their high water content and tissue-like elasticity recapitulates many critical aspects of native tissue, while their optical transparency enables non-invasive imaging and optical processing.[4–7]

There are currently several methods used to create vasculature in hydrogels, including those based on additive and subtractive manufacturing techniques.[3,8–16] The most commonly exploited additive manufacturing technique for vasculature formation is 3D printing, which builds material layer by layer.[9,13–15] 3D printing creates vasculature by intentionally leaving voids between layers. Additive methodologies have proved successful for generating perfusable vessels, but with resolutions (>100 μm) that do not match that of human capillaries.[3] 3D printing does provide the ability to create larger networks that span the scale of several centimeters.[15]

Subtractive manufacturing techniques create a void space for the vasculature within a bulk material by removing ("subtracting") sections of the material. One method of subtractive manufacturing is the sacrificial lattice approach.[10,17] This method utilizes a sacrificial shape that is removed once the desired material is cast around it. This has been done utilizing sugar glass. The desired vasculature is designed and made from a lattice out of sugar then the bulk material is made around it. The sacrificial lattice is then dissolved, leaving the desired network behind as

void space.[10] Both this subtractive technique and 3D printing have shown limitations in resolution of networks.[3]

A subtractive manufacturing approach utilized withing the DeForest group has created vasculature at capillary resolution utilizing multiphoton lithography to cause hydrogel degredation.[3,16] This process utilizes photodegradable crosslinkers within hydrogels. A multiphoton laser with desired wavelength for crosslinker degradation, is focused in order to carve out channels of desired shapes.[3] Recently Zheng's group produced cellularized glomerular in collagen utilizing multiphoton ablation containing 10 μm channels.[2] Arakawa et al. created cytocompatible vasculature with capillary resolution, sub 10 μm diameter channels.[16] Photoablation of collagen-based hydrogels required high laser powers and photonic dosages, limiting patterned structures to small sizes. The ablation of a single glomerular structure, as seen in Zheng's work, (~42.8 nl) took under 90 minutes to complete.[2] The use of *ortho*-nitrobenzyl (*o*NB) as a crosslinker exists, but like collagen it is not fast enough to reasonably scale to large vascular systems. A hyper photo-responsive approach is needed to quickly photopattern large-scale vascular networks with anatomically derived resolution (10 μm). This work explores one such approach by utilizing a ruthenium-based crosslinker since its two-photon photocleavage occurs[18] several orders of magnitude faster than that of *o*NB based crosslinker.[14]

Ruthenium based crosslinkers have been studied in depth by Rapp[19–23]. Rapp's research has shown that ruthenium-based hydrogels degrade rapidly in visible light[23] and have been used for multiple biological applications such as protein release.[20] RuOrange [$(Ru(bpy)_2(4\text{-azidobutyronitrile})_2[Cl_2])$, Figure 1] is a ruthenium-based crosslinker recently developed by Rapp that degrades near instantaneously when exposed to visible light. She has provided data in photolysis and rheometry to characterize this new crosslinker. Given its ease of syntheses and

rapid photodegradation, we elected to utilize RuOrange as the primary crosslinker for studies in this research.

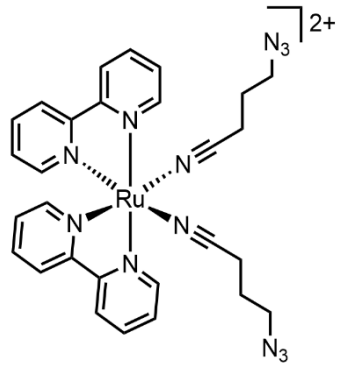


Figure 1 The structure of the crosslinker $Ru(bpy)_2(4\text{-azidobutyronitrile})_2[Cl_2]$) RuOrange

This work shows that utilizing this ruthenium-based photosensitive crosslinker can provide rapid access to large-scale microvasculature with capillary resolution for the first time at microliter volumes.

Material Synthesis and Characterization

RuOrange synthesis is derived completely from commercially available materials. A 4-azidobutyronitrile (Figure 2) is first required to be synthesized as the ligand for RuOrange. This ligand requires a reaction of 4-bromobutyronitrile (Aldrich B59802) C_4H_6BrN with Sodium Azide (Aldrich S2002) NaN_3 in Dimethyl sulfoxide (DMSO) at 55°C overnight. Diluted with water an extracted with diethyl ether, upon rotary evaporation, the light-yellow oil is the desired product.

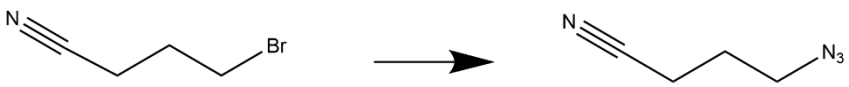


Figure 2 Synthesis of 4-Azidobutyronitirle derived from 4-bromobutyronitrile

The synthesis for RuOrange (Figure 3) was completed in the dark to avoid white light exposure to ensure the final product is not degraded during synthesis. The first reaction to synthesizing RuOrange is cis-Dichlorobis(2,2'-bipyridine)ruthenium(II) dihydrate (Strem Chemicals 44-0200) and silver hexafluorophosphate $AgPF_6$ (TCI S0981) in methanol (stored over molecular sieves) and stirred for 15 minutes at 55°C. The 4-azidobutyronitrile ligand is then added to the reaction at 55°C and stirred for approximately two hours until the color lightens from red to orange. Once a color shift is observed, the solution is then cooled to room temperature and passed through a paper filter to remove the silver chloride. The liquid is then concentrated by using rotary evaporation. Silica flash chromatography with a 1:4 acetonitrile (ACN) to dichloromethane (DCM) eluent is used for purification. The major visible yellow band is collected. The solvents are evaporated using rotary evaporation and RuOrange is than confirmed with ^1H nuclear magnetic resonance (NMR). Upon conformation of purity, the compound then passed through a chloride ion-exchange column, Amberlite IRA410 chloride. Once collected and rotary evaporation is performed, the synthesis of the RuOrange crosslinker is complete. The crosslinker can then be aliquoted with buffer and stored at -80°C until use. Aliquots at concentrations of 40mM are used within this work for gel formulations.

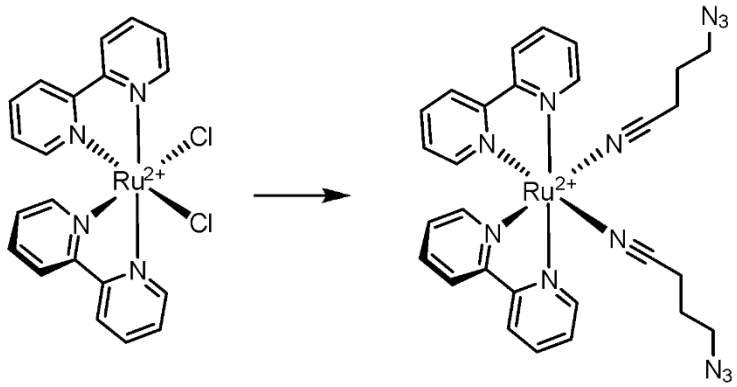


Figure 3 Synthesis of RuOrange derived from $Ru(bipyridine)_2Cl_2\,H_2O$

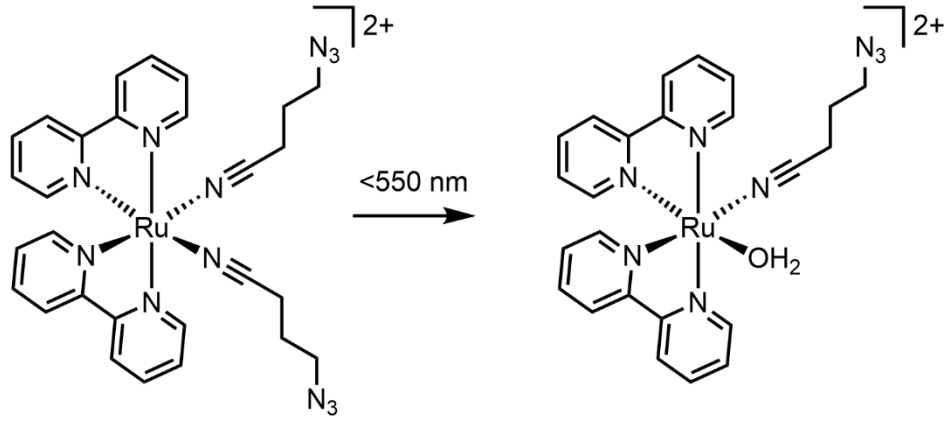


Figure 4 RuOrange Photodegradation process when exposed to wavelengths less than 550nm

RuOrange was evaluated utilizing photolysis showing how the absorbance spectrum shifts upon degradation. A 450nm laser was used to degrade the crosslinker (Figure 4) while taking UV-Vis spectrometry (Figure 5) to track the shift in absorbance while laser exposure time increased.

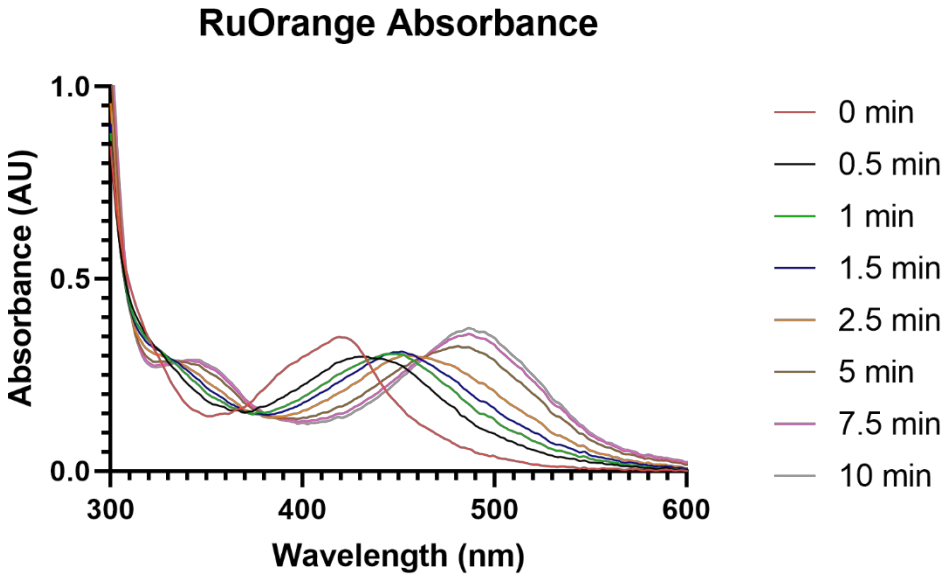


Figure 5 RuOrange photolysis after degradation with 450nm laser.

RuOrange was designed to be a hydrogel crosslinker via strained-promoted azide-alkyne cycloadditon[24] (SPAAC) step polymerization of poly(ethylene glycol) tetra-bicyclononyne (PEG-tetraBCN).[8,25–28] The polymer backbone (Figure 6) which bears four reactive strained alkynes host click reactions with the azide-modified RuOrange crosslinker. These bonds form

the hydrogel networks. When RuOrange is exposed to light, the link holding together the network is broken degrading the gel.

The PEG hydrogels within this work are formed using 4-arm PEG-BCN (PEG-tetraBCN ≈ 20 kDa), with final concentrations of 3mM PEG-BCN, 6mM of the RuOrange crosslinker and phosphate-buffered saline (PBS) as the buffer solution.

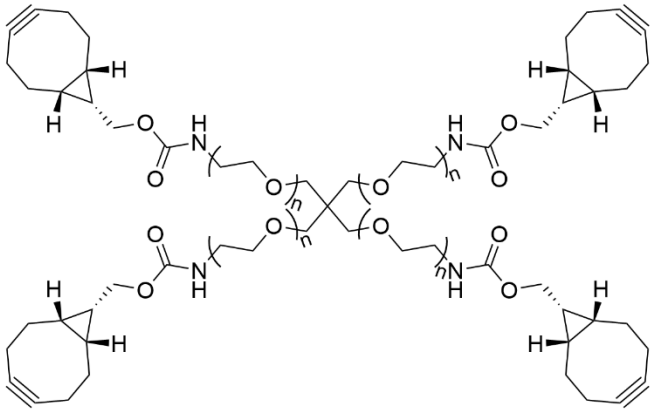


Figure 6 Structure of Polyethylene glycol tetra- bicyclononyne

The degradation of these gels is studied using in situ photorheometry. These measurements (Figure 7) were taken using an Anton Paar MCR301 with a quartz base plate allowing for light exposure at the base and a PP08 measuring plate. A 20 µL gel was cast between the 2 plates set at a working distance of 0.3 mm. The RuOrange-crosslinked gel underwent the gelation process and approached a storage moduli (G') of 5.4 kilopascals (kPa). The gel was submerged in 3 mL of 0.1M sodium NaN_3 to cap any remaining BCN groups prior to light exposure. At 8430 seconds, a 530 nm light at 10 mW/cm^2 was turned on. The result shows that the storage modulus drops instantly degrading the gel completely into liquid form.

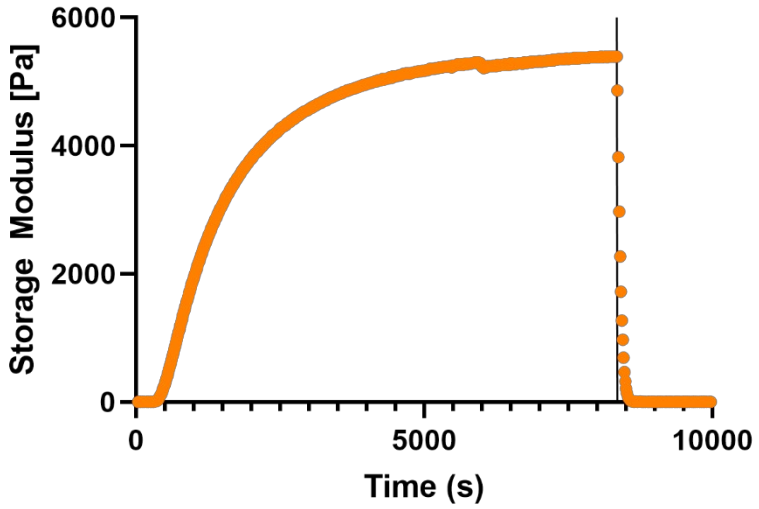


Figure 7 In situ photorheometry of RuOrange where a 530nm light at 10mW/cm^2 was activated at 8340 seconds (black vertical line)

To further display the capabilities of the RuOrange crosslinker, hydrogels were suspended with cells for viability studies. 10T1/2 Cells were encapsulated within 5 µL RuOrange hydrogels at a density of1.8 million cells per mL. As a control, triethylene glycol (TEG) diazide was used as a crosslinker. Both gels contained 1mM of azide RGD.[6] After 24 hours of incubation at 37 ℃, the gels were imaged using a Leica SP8X confocal microscope as seen in Figure 8. Using a live/dead stain (Invitrogen), the green cells signified live cells, while red were dead. Cell counting was

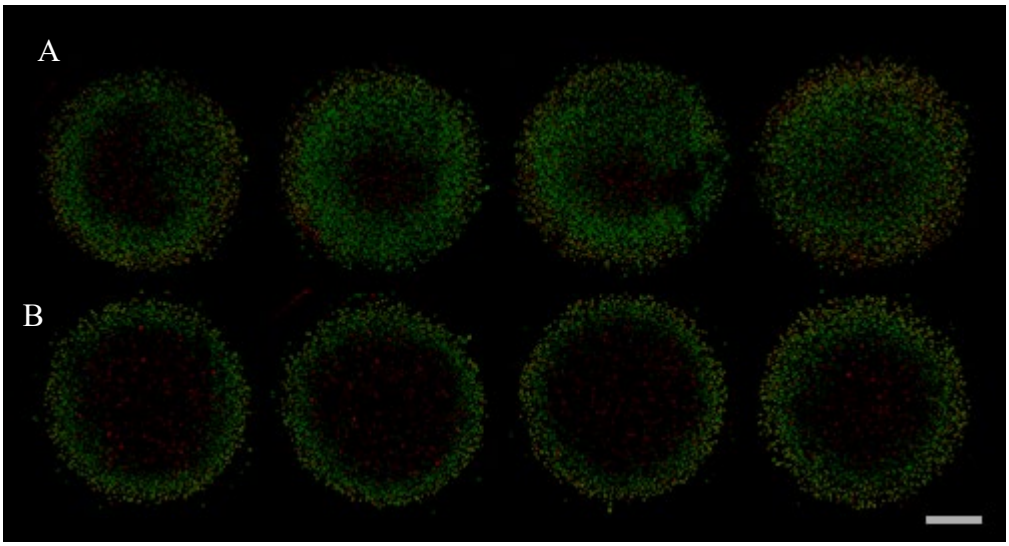


Figure 8 Cell viability study results with the top row (A) being the control formed with TEG, and the second row (B) are gels formed using RuOrange. Scale bar 1 mm.

conducted and the following results are contained in Figure 9. On average of the four samples, a viability of 90.1% was observed for the TEG control, and 89.9% for the RuOrange crosslinked gels. This study shows that RuOrange has potential in biological applications given that cells are able to survive within the hydrogel network. Further research would be required in order to optimize conditions for cell culture.

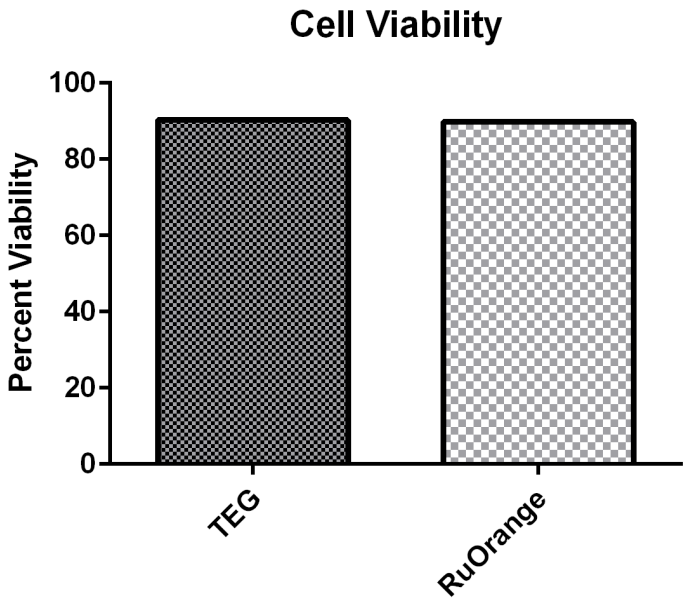


Figure 9 Cell Viability where TEG gels were used as a control. 90.1% viability observed for the control group and 89.9% for RuOrange Gels.

Experimentation and Discussion

Stack Development

Patterning techniques commonly use photomasks in order to control what areas of a sample are exposed to light. Photomasks can vary in complexity from very simple shapes to very complex vasculature networks both seen within this work. For multiphoton patterning, a virtual mask is utilized to control laser intensity at given pixel locations. These virtual stacks can be created to utilize both grayscale or binary images and can be created in near any image processing or drawing software.

Another route to develop image stacks is to slice a 3D model file such as an .stl into an image stack with user defined layer spacing. One approach used in this work was utilizing Autodesk's NetFabb software. Designed primarily for 3D printing, the program is able to export layer slices of a 3D model in a binary format, allowing for backgrounds to be white, and objects to be black. These image sequences produced can then be used as virtual masks for multiphoton patterning utilizing the ScanImage® program. An additional way to acquire photomasks is to utilize an image stack derived from other forms of microscopy allowing users to use biologically derived stacks to guide photopatterning.

Multiphoton Characterization

The RuOrange-PEGtetraBCN hydrogel degrades near instantaneously when exposed to visible light as shown in the rheometry studies. The base of RuOrange, cis-Ru(II)$(bpy)_2Cl_2$ has a relatively large two photon cross section when compared to other crosslinkers that have been used for multiphoton patterning techniques. cis-Ru(II)$(bpy)_2Cl_2$ cross section (σ_2) is 120 GM (1 GM = 10^{-50} cm^4 per second per photon[18] whereas the photodegradable *o*-nitrobenzyl (*o*NB) group has a cross section of 0.01 GM.[12]

The Thorlabs Bergamo II multiphoton confocal microscope is powered by a Coherent Chameleon Discovery NX with total power control is a femtosecond-puled laser offering dual-color through a tunable laser line offering wavelengths between 660 – 1320nm and a second fixed laser of 1040nm. The experimentation throughout this work only utilizes the variable-wavelength laser line. An acousto-optic modulator (AOM) is used to rapidly control laser intensity. A resonant-galvo-galvo scanner feeds into two GaAs Thorlab PMT detectors for simultaneous 2-color imaging. The resonant scanner allows for 8 kHz scanning (30fps at 512 x 512 pixels) and galvometric scanning providing custom ROI scanning geometries with equal

dwell time throughout all points in the field of view, 3.6 fps at 512 x 512-pixel resolution. The objective is an Olympus 25x water immersion objective with a working distance of 8 mm. The microscope is controlled by MBF Bioscience's microscope control software, ScanImage®[29] containing photopatterned illumination module custom-built by MBF Bioscience. ScanImage® is a MATLAB based program that allows users to directly interface with the software via MATLAB script or command line.

The photopatterning module built by MBF Bioscience, known as Print3D utilizes a grayscale image to calculate laser intensity for each given pixel of the desired pattern. Using the microscope's AOM, the laser intensity is able to be modified during raster scanning. The AOM operates at speeds <1 μs. In regards to the virtual mask, a white pixel or a pixel value of 255 for an 8-bit photo, corresponds to 100% of the desired laser intensity. Whereas a black pixel, or a pixel value of 0 for the same 8-bit photo corresponds to 0% laser intensity. This patterning module can utilize both the resonant scanner and the linear scanner on the Thorlabs Bergamo II allowing for a wide range of pixel dwell time control. In order to make the two-dimensional pattern three-dimensional (3D) the same processes occurs at a different z location and a separate image can be used as a virtual mask providing full spatial control for multiphoton patterning.

To show the degradation of the RuOrange crosslinker, multiple characterization experiments were conducted varying wavelength, pixel dwell time, and z-resolutions. This characterization is required to optimize the settings of the microscope to accelerate the degrading processes.

The first experiment compared wavelength to pixel dwell time. Adjusting pixel dwell time while utilizing the resonant scanner and patterning program is most conveniently done by repeating the photo-pattern in the same location. Wavelengths between 660nm and 1150nm were tested against dwell times of 88ns to 440ns which was done by repeating the pattern 1-5 time in the

given location. A photomask z stack of 10 repeated images of 512 pixels by 512 pixels where each pixel of the photomask is 0.7024 microns in both width and height was used. This mask contained an ~18 µm thick rectangle spanning the entire frame. The z stack was patterned varying the z of each pattern by 1 µm resulting in a patterning volume of 360 µm x 18 µm x 10 µm. The rectangle was then shifted 60 µm and repeated. This caused each new position to gain an additional repeated pattern. The results are seen in Figure 10.

An average of pixel values was taken over a 10 µm x 30 µm area of each patterned location utilizing Fiji's[30] measurement tool. These values confirmed that degradation occurred at multiple wavelengths. The 750 nm wavelength trial degraded near completely with 88ns of exposure. The results for the 660 and 1100 wavelengths were not graphed in figure 10 due to no significant change in pixel values for those trials.

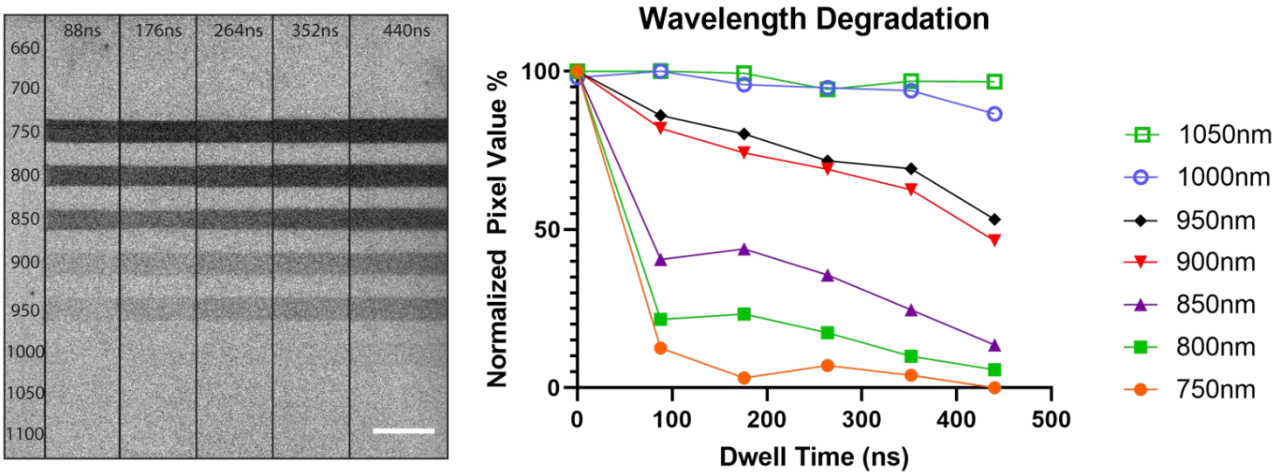


Figure 10 Degradation analysis of RuOrange hydrogel based on laser wavelength and pixel dwell time. Scale Bar 50 µm.

The second characterization was determining the resolution z resolution while varying both laser power and pixel dwell time. Utilizing a 750nm wavelength laser, a single square image stack of 10 images spaced 1 µm apart ~60 µm x 60 µm x 10 µm. This pattern was repeated in a grid fashion. Where power was varied from 20% to 100% in each column, and the number of pattern repeats (pixel dwell time) was varied in each row. Repeats were varied from 1 to 5 corresponding to dwell times of 88ns to 440 ns. The pattern was then imaged as seen in Figure 11 and processed to show the cross section of each pattern. The results shown does not include the laser power of 20% due to no significant degradation. With increasing dwell time, the expected shape does get distorted in the z dimension, though there is some distortion in the x and y directions as seen by some degraded patterns shown outside of the expected degraded areas of the gel.

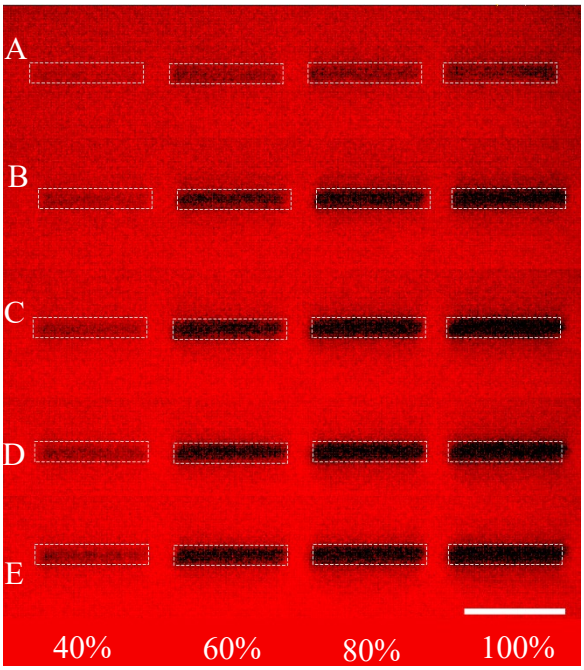


Figure 11 Characterization of RuOrange degradation where side views are compared to expected pattern shapes (white). Rows A-E correspond with increasing dwell time from 88 ns – 440 in increments of 88 ns and each column corresponds to increasing laser power. Scale bar 50 µm.

The third characterizing is the relationship between laser power and varying the spacing between each photomask in the z dimension. Here a photomask stack of 10 squares was patterned in a grid fashion varying the laser power from 20% to 100% and the z spacing between each mask starting with a spacing of 1 µm resulting in a 10 µm pattern and ending with a z spacing of 5 µm resulting in a 50 µm pattern. After patterning, an image stack was captured and processed in order to show the cross section of each pattern. Seen in the resulting image, Figure 12, the spacing of 1 and 2 µm show consistent degradation in the expected pattern range, whereas the increased z spacing >2 µm fails to show degradation in the full range of the expected pattern.

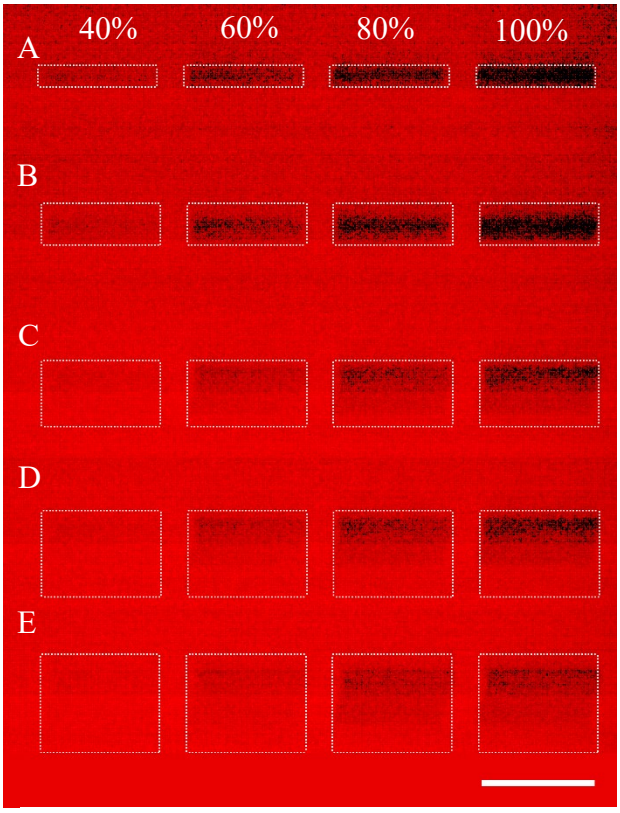


Figure 12 Characterization of RuOrange degradation where side views are compared to expected pattern shapes (white). An image stack of 10 squares was patterned increasing the z step space between each image by 1 µm for each row. Row A corresponds to a z step of 1 µm and E corresponds to 5 µm. Each column corresponds to increasing laser power. Scale bar 50 µm.

In order to show that RuOrange is degrading and forming voids versus photobleaching the Alexa 568 fluorophore used for imaging, a 2,000kDa fluorescein isothiocyanate-dextran (Sigma Aldrich FD2000s) was dissolved in PBS at a 1mg/mL concentration. A RuOrange gel was cast on an azide-treated glass slide. A rubber seal was placed around the gel creating a chamber. This chamber was filled with the dextran solution and sealed with a glass cover slip by tapping the coverslip down. This sealed chamber containing the hydrogel and dextran solution prevents the dextran from being diluted from the water bath that that gel is subjected to due to the submersible objective.

Two channels were created spanning the length of the gel. Between these two channels, several smaller channels of various sizes were created to show that the fluorescent dextran would flow through the channels created by the multiphoton patterning at scales comparable to vasculature size. A model is shown in Figure 13A. The RuOrange gel was patterned using a single repeat of the photomask image stack. The image stack consisted of 10 images and each image was patterned with a z space of 1 µm. This shows that utilizing a single repeated pattern (88ns pixel dwell time) does degrade the hydrogel and creates perfusable void spaces as seen in Figure 13B.

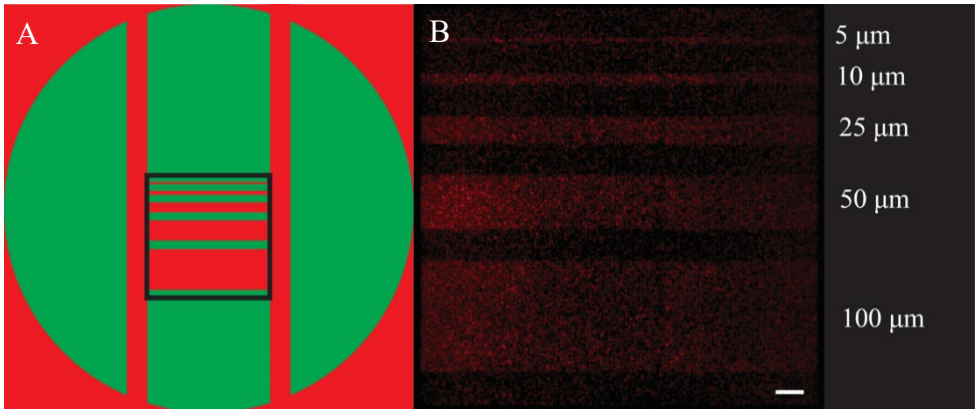


Figure 13 (A)Model of RuOrange Hydrogel with two cut channels, the photo mask of varying sized channels outlined in black (B)Dextran (red) prefused through channels in the RuOrange Hydrogel. Scale Bar 25 µm.

The speeds at which this ruthenium-based crosslinker is able to pattern is several orders of magnitude faster than other multiphoton patterning crosslinkers. For example, a pattern of 18.27 nL (360 μm x360 μm x141 μm) a single image stack that contains 141 slices that is patterned using a 360 μm x 360 μm pattern window and a z-step between each layer of 1 μm took 18,612 seconds to pattern. This was an *o*NB-derived crosslinker. The RuOrange system took 20 seconds. Both are seen in Figure 14. This is three orders of magnitude faster than the *o*NB crosslinker system. In theory, without taking into account the time it takes for the microscope to move to different locations of x, y, and z, in the time it takes to pattern a small 18.27 nL pattern of an *o*NB crosslinker system, a 17 μL pattern could be created. This enables microscale patterns to be created in the same time it would take nanoscale patterns while keeping the capillary-sized resolution.

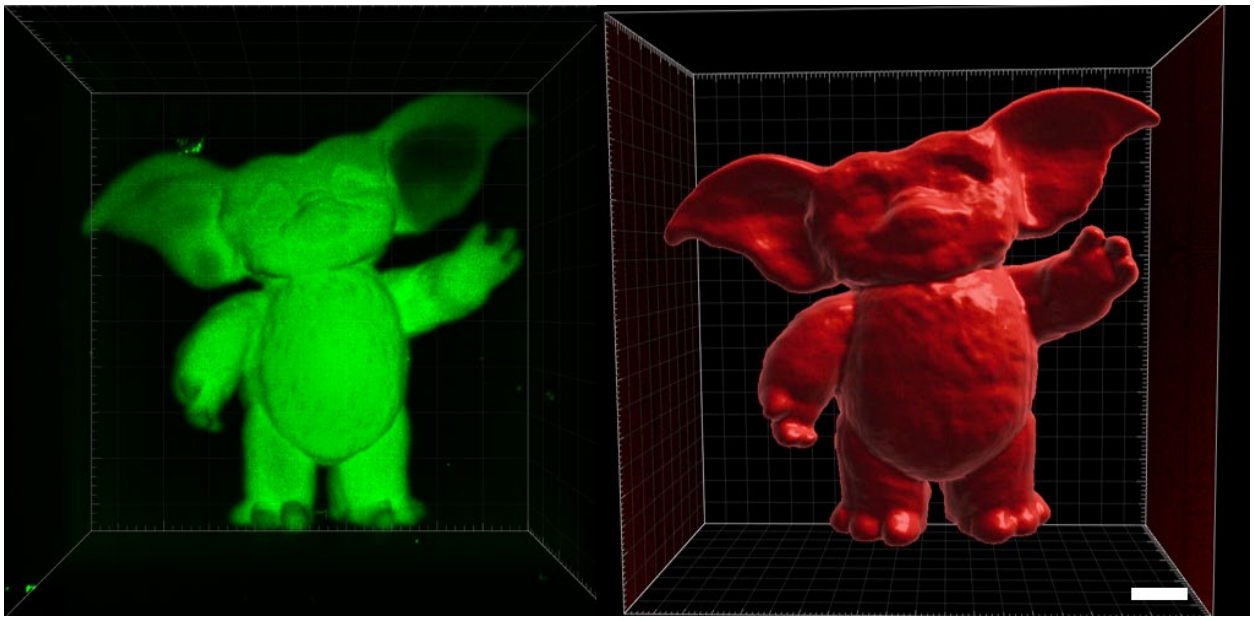


Figure 14 Comparison of chemistries, the left image utilized an oNB crossliner and took 18612 seconds to pattern. The right image, was done in 20 seconds both were patterned with the same exact image stack and size. Scale Bar 50 μm.

Automated Tile Patterning

The ability for multiphoton patterning using ScanImage® is limited to a single view frame in the "x, y" coordinates and a "z" dimension limited only to the throw of the microscope's stage movement. While utilizing the "FastZ" Piezo motor controller, the "z" dimension was limited to 450 μm. Without the use of magnification this is 719 μm x 719 μm with a maximum potential "z" height of 8000 μm using the slow setting or 450 μm in the fast setting. This is a total volume of 4.14μL and 0.23μL, respectively. A tile patterning application within MATLAB was developed in order to increase the total volume of patterning capabilities.

Utilizing the preexisting photopatterned illumination module custom-built by MBF Bioscience, a MATLAB application was created to pattern cropped volumes of a larger image stack in a user defined "x", "y", and "z" with maximum parameters dictated by the physical restraints of the instrument itself (50,000 μm in both "x" and "y" and 8,000 μm in "z"). The z dimension is limited based on the 8,000 μm focal length of the Olympus objective. The application developed titled "TileControls" is able to split image stacks in to patternable volumes and then pattern each volume in a tile fashion producing large scale patterns. The Tile Pattern application has two major components: cropping and processing image stack to be utilized as virtual masks and tile patterning the cropped volumes.

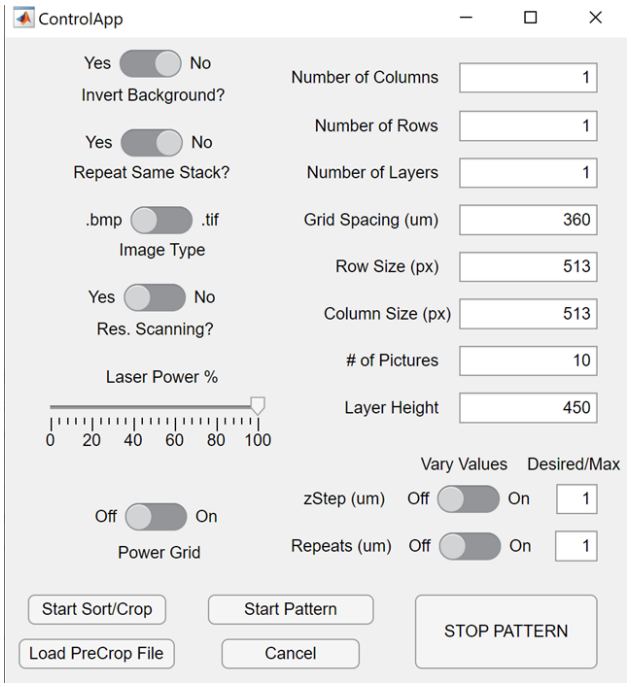


Figure 15 Tile Pattern Application's User Interface

The Tile Patterning Application (Figure 15) provides a user interface to set all required variables in order to run the application and interface with ScanImage®. In addition to image cropping and tile patterning, the application also features the ability to load a pre-cropped/sorted directory and a stop tile pattern button the ends the tile pattern function.

Cropping and Sorting

Cropping and sorting is comprised of one MATLAB script. Through the application's user interface, a user can control several parameters to how the images are processed, cropped, and sorted. The first option that a user has is to invert the background of the image. Since the Print3D widget built into ScanImage® controls the patterning based on pixel values, white utilizing full laser power and black uses no laser power, having the ability to quickly invert an image allows users to control within the application what part of the image stack is patterned and what part of the image is to remain.

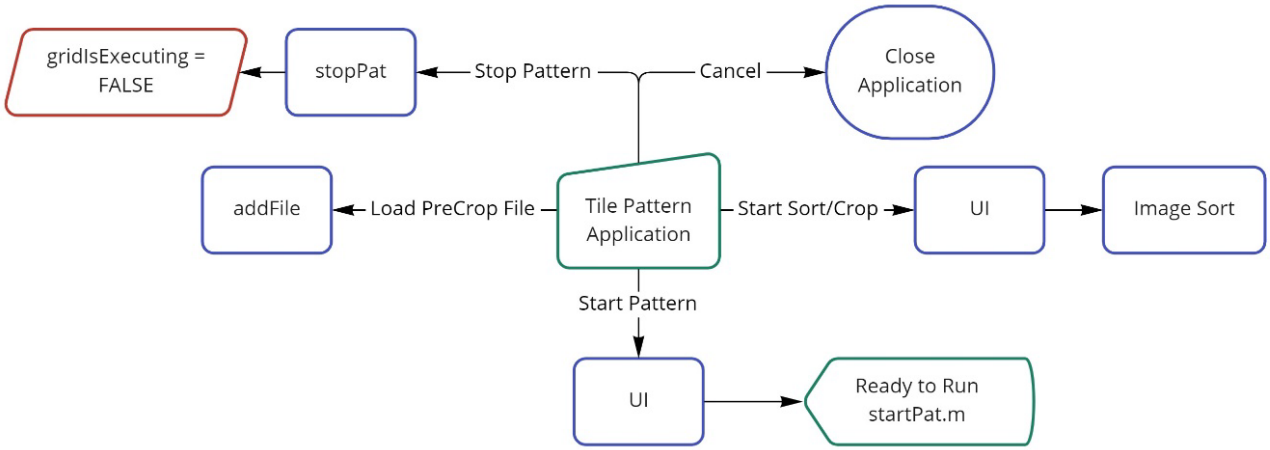


Figure 16 Tile Application flow chart. Blue boxes signify unique MATLAB scripts that are utilized together to run the app.

The second control that a user has is to tell the software if resonant scanning will be utilized. Since the Thorlabs multiphoton microscope can use both linear scanning and resonant scanning, it is important to have to pattern set up for the settings on the microscope. Utilizing the resonant scanner will mirror images across the y axis. This can be a minor inconvenience when patterning a single view frame, but when tiling if each pattern is reversed, the final pattern does not resemble the original. For this reason, the program will mirror each volume in order to accommodate the mirrored resonant scanner.

Repeating the same stack is an option for users. This allows users to tell the program to repeat the same pattern in each tile, this can be useful for characterization, multiple patterns, or grids. This option still allows users the processing features and will sort the images into the proper order to be fed into the ScanImage® program.

Users are able to select the type of format of the input images. Since the processing is slightly different for each image type, the user needs to select which file format they are attempting to upload into the software.

Users than select the desired number of images in the stack. This feature provides three options for users. The first option is to just tell the program how many images are being uploaded. The second feature is to statistically decrease the number of images in the stack. If a desired 1000-layer stack is selected, and only 500 of the images are desired, this is where users can automatically decrease the number of images. For the previous example it would only processes and sort every other image. If you desired 1/3 of the images uploaded it would only process every third image and so on. The final desired value feature is to multiply the stack. If you wanted double the number of images, it would process and sort every picture twice.

Additionally, the desired pixel width and height to crop each tile is an included variable. This will be the size of the final output images that will be patterned. Additional processing that the image cropping code conducts is to ensure each patternable volume is the requested size. The Print3D from ScanImage® will resize loaded images into a square by stretching or shrinking dimensions to ensure that the entire image is patterned. This can distort patterns and cause misalignment when attempting to tile pattern. For this reason, the cropping code will add empty pixels along the edge of a pattern to ensure it is the desired size when loaded into the Print3D widget.

Layer height is the desired height for each printable volume, based on desired z-step size and number of images. This variable is used when sorting the final cropped images into their respected folders in order to feed them into ScanImage®. A Piezo objective controller is utilized to increase the z speed of the microscope. This has a movement limit of 450μm. Layers of patterns are created in order to utilize the Piezo and fast patterning capabilities within the Print3D widget. By sorting the cropped images into layers, the usable volumes can be contained within the limits of the Piezo or other user defined sizes.

All of the cropped images are saved in the parent directory from where the user selected the image sequence. The first directory that is made is organized by layer. These layers contain all the images that will be used in each layer of the tile pattern. Within the layer directory several more directories are created. These are specific volumes that will be patterned within the layer.

Tile Pattern Automation

The tile pattern program contains several scripts in order to accomplish both interaction with ScanImage® and the images being patterned. The variables entered into the user interface are the only variables required to execute the entire tile pattern.

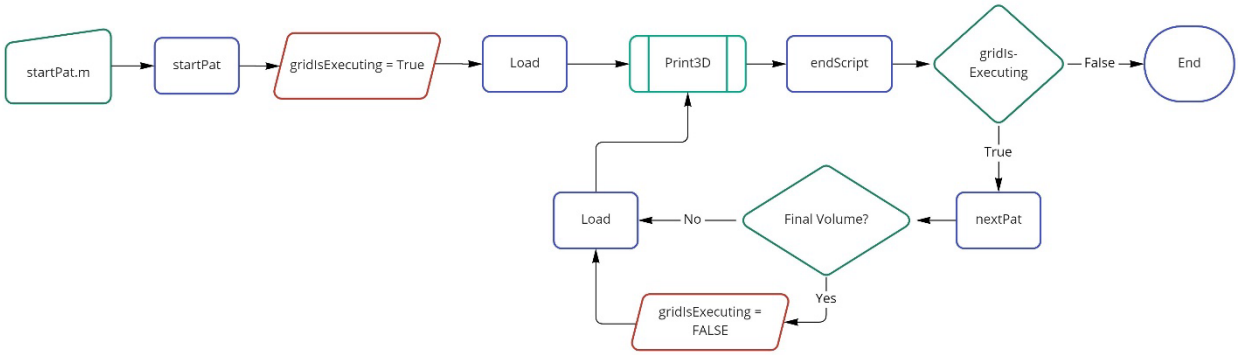


Figure 17 Tile Pattern Programing Workflow

After entering the desired variables, users select “Start Pattern” within the tile pattern application, the command line will display a message confirming the user is ready to run their pattern and if so, to run the startPat matlab script. The startPat script acquires the microscope’s stage position from ScanImage®, turns the control variable, “gridIsExecuting” to true, and then runs the load script to load the first volume from the cropped/sorted images. The load script places the images from the desired volume into an array that is compatible with the ScanImage® Print3D widget then loads it into ScanImage®. The startPat script then executes the Print3D widget patterning the desired images.

Print3D is a program built by MBF Bioscience that takes individual images to create waveform calculations. These waveforms control the laser power at each pixel location corresponding to the pixel value of the loaded image. The white pixels of an image will be corresponding to 100% laser power and the black are 0%. The Print3D widget has several speed and actuator features. The “fast” feature pre-calculates all the waveforms for a volume and then utilizes the Piezo to quickly pattern each image of a stack. This occurs so fast that the Piezo does not completely stop as it moves through the volume. The resonant scanner must be used when patterning while utilizing the fast feature. The normal speed calculates each waveform prior to patterning. The piezo and the stage motor can be used to vary the z location and either resonant or linear scanning can be used. The application also allows users to control the number of repeats of each pattern and the z-step between each image.

When the Print3D program completes, it automatically calls the end script. This script will either completely end the program or call the nextPat script depending on the state of “gridIsExecuting”. If it is true, nextPat is called, if it is false the pattern is complete.

The script nextPat first determines which volume was just patterned and moves the microscope to the next volume’s location. In the interest of time, and to limit the amount of movement of the microscope, all of the volumes in single x, y location are completed prior to moving to the next horizontal tile location. The distance the microscope moves is determined by the user in the application as the grid spacing for the x,y movement and layer height for the z movements. Once the microscope has moved to its new location the script makes two decisions. The first is if the user has requested to repeat each volume. If the user did request to repeat, the program moves forward, however, if the user decided to use unique volumes for each tile, the program then runs the load script. The load script loads the corresponding volume based on the number of volumes

already patterned. The second decision that is done is to determine if the next pattern is the last volume. This is done by keeping track of the total number of patterned volumes. If it is the final pattern, the "gridIsExecuting" variable is changed to false which will end the program after the pattern is complete. If it stays true, the program will return to the nextPat script in order to continue tiling. Throughout the process the MATLAB command line provides feedback with which the grid position and layer currently being patterned as well as a final message when the pattern is complete.

Additional features were added into the program for characterization assistance that were used for the characterization of RuOrange in the previous studies. This includes the ability to evenly vary the variables of power, repeats, and z-step. These switches take the entered value and will evenly divide the variable by the total number of volumes that will be patterned.

Tile Pattern Results

By incorporating the tile pattern program, larger image stacks are able to be autonomously patterned. To highlight this new capability an .stl file was used to produce a virtual 3D mask and then was cropped using the developed tiling application. The selected model (Figure 18D) is a dendritic vascular network derived from a mutual tree attraction algorithum.[17] The entire pattern took two volumes next to each other. The void space created using multiphoton degradation was then filled with fluorescent dextran in order to illustrate channel patency and connectivity as seen in Figure 18. At the smallest location, the channels were shown to have a diameter under 10 μm

corresponding to the size of human capillaries. This demonstration shows both the successful execution of the application and capabilities of the RuOrange hydrogel for larger scale patterns.

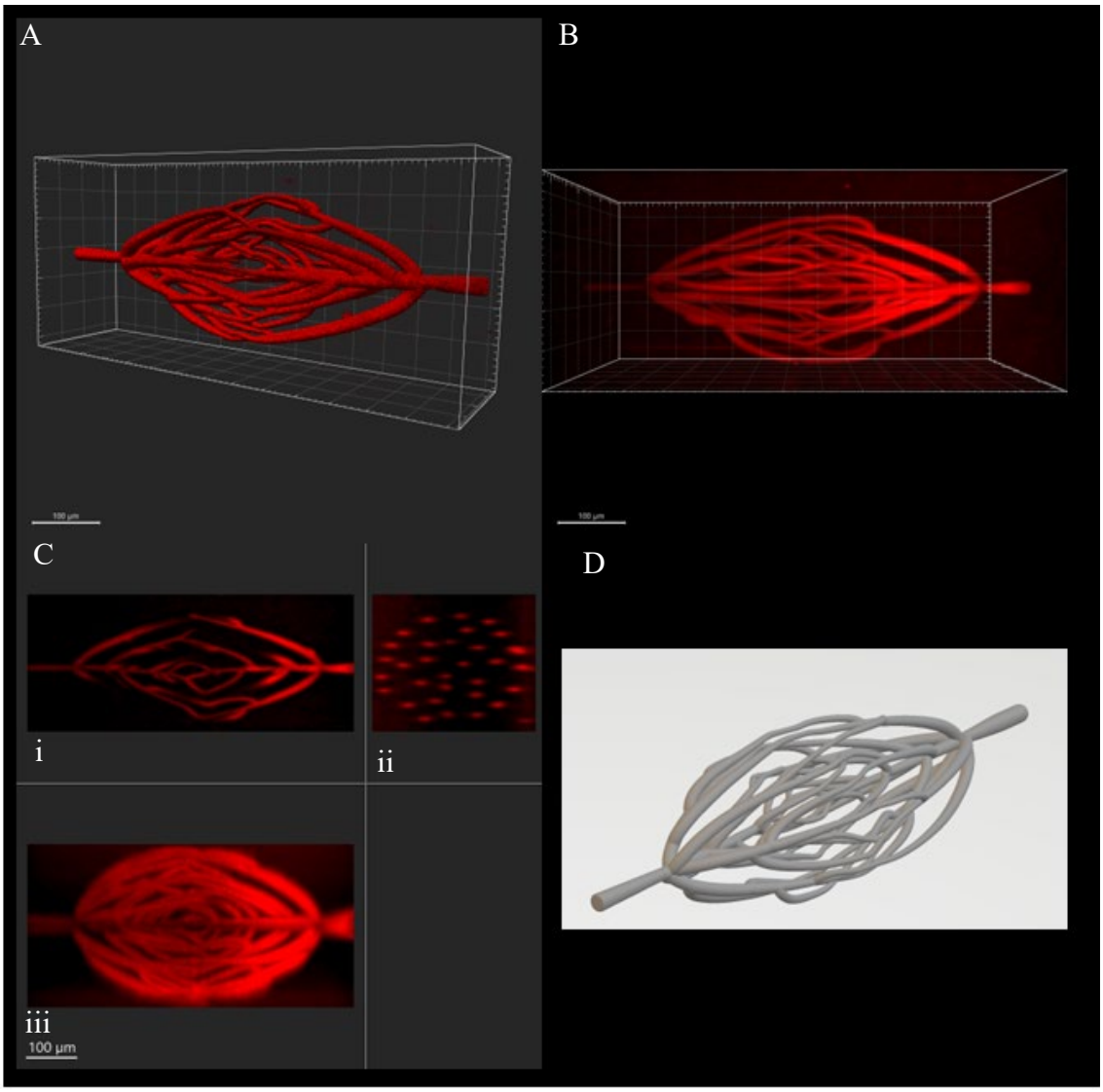


Figure 18 (A) Vasculature 3D rendering of pattern using the Imaris shading feature. (B) 3D rendering of pattern using the Imaris MIP feature. (C)(i) Central plane of the front view sectional (ii) center plane side view sectional showing <10 µm channels (iii) projection of entire stack from the top plane. (D) .stl file sliced into virtual masks. All scalebars are 100 µm

Provided through a collaboration with the Molecular Biophotonics Laboratory directed by Jonathan Liu (University of Washington, Department of Mechanical Engineering), a mouse brain with the vasculature fluorescently tagged was imaged on an open-top light-sheet microscope.[31] The image stack they provided was binarized and processed to ensure continuity of all

vasculature (Figure 19). This provided image stack was then processed and cropped in the Tile Pattern application. In order to keep the biologically derived scale, it was cropped into 180 separate volumes to be patterned using multiphoton degradation. These virtual volumes were then patterned into the RuOrange hydrogel. The 2mm x 2mm x 2mm (8 μL) pattern took a total of 77 minutes to complete which includes the calculation of waveforms used to control the multiphoton laser and all the microscope stage movements. The same parameters used to create the perfusable network was used to degrade the mouse brain vasculature. The final images were taken utilizing the Thorlabs multiphoton microscope with the hydrogel coordinated to a CY5 azide during casting. 16 image stacks were acquired and stitched together utilizing Fiji's stitching plugin [30]. The full image stack was then inverted and rendered utilizing Imaris for 3D visualization as seen in Figure 20.

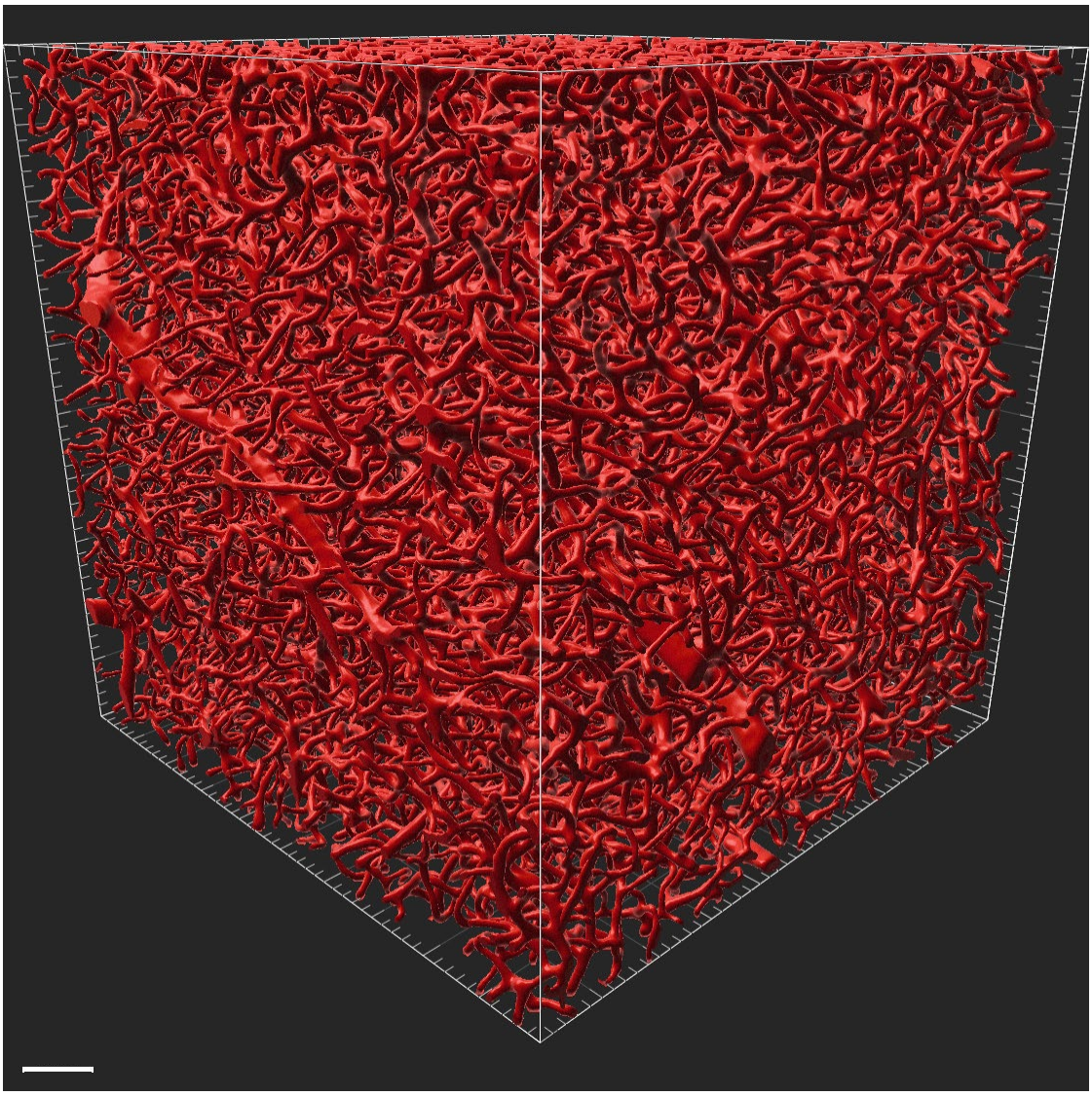


Figure 19 Virtual Mask of provided mouse brain vasculature rendered in Imaris. Scale Bar 200 µm.

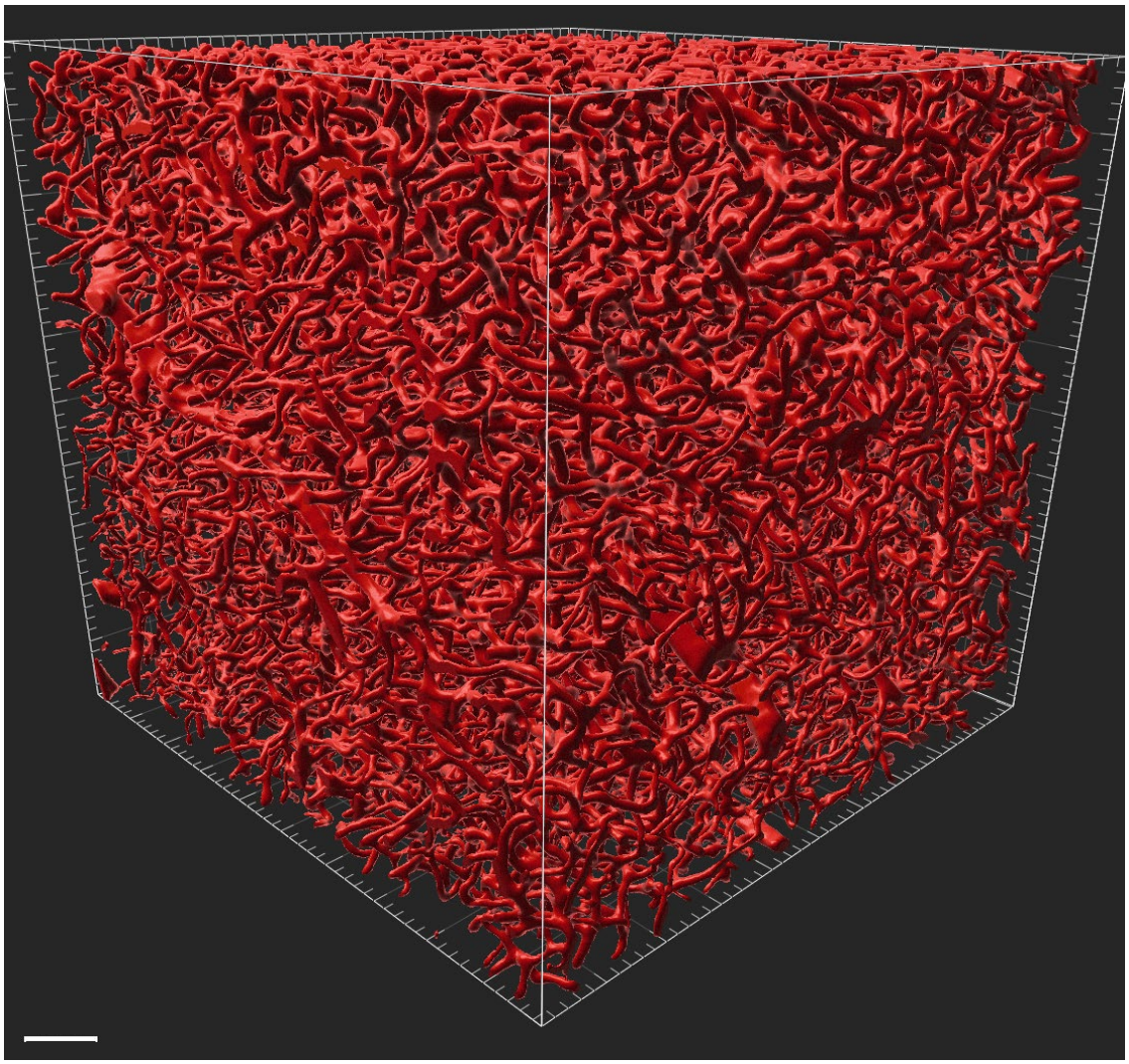


Figure 20 To scale mouse brain vasculature 2 µm x 2 µm x 2 µm patterned into RuOrange hydrogel in 77 minutes. The masks used were directly from an imaged mouse brain. Scale Bar 200 µm.

Conclusion

This work characterized and used RuOrange as a hyper-photoresponsive crosslinker, allowing for multiphoton degradation to occur at speeds currently limited by computer processing and mechanical movements of the microscope. Combining this speed of patterning at physiological resolution and the newly created tile pattern application, we have demonstrated the newfound ability to create well-defined large-scale microvascular networks with capillary-sized resolutions.

This breakthrough combination opens a new door into the biomaterial and tissue engineering fields utilizing hydrogels and proves that large scale hydrogel patterns (μL) with physiological resolution (<10 μm) are capable of being manufactured replicating anatomically derived microvasculature.

Acknowledgments

I would like to express my thanks to my advisor Prof. Cole DeForest for support throughout my research and time at the University of Washington. Given the virtual environment due to COVID-19, Cole gave me a space at UW to take pride in. He motivated me to strive for greatness in my research and degree.

Teresa Rapp was a great mentor and I could not have done this work without her crosslinkers and the knowledge she shared. Having little chemistry lab experience starting this research, having a chemist willing to teach me everything about crosslinkers, synthesis, and academic research was invaluable.

To all my labmates that provided a wonderful and supporting working environment, thank you for guiding me through the lab.

For helping and coaching me through the inner workings of ScanImage and MATLB, I would like to thank Georg Jaindl. With his expertise the Print3D function increased the labs capabilities making my work possible.

I would like to thank Dr. Nathaniel Peters at the W. M. Keck Microscopy Center for the access to IMARIS workstation allowing for the beautiful renderings of the images I was able to acquire.

I also want to thank Dr. Jonathan Liu and the molecular biophotonics lab for sharing their data enabling this work to be derived from actual specimen.

Lastly, but most importantly, I would like to thank my beautiful wife Dakota for moving across the country allowing me to attend the University of Washington. All of her love, confidence, and sacrifice made our time spent in the PNW fruitful with memories to last a lifetime. Thank you.

References:

1. Lewis, M. C., MacArthur, B. D., Malda, J., Pettet, G. & Please, C. P. Heterogeneous proliferation within engineered cartilaginous tissue: the role of oxygen tension. *Biotechnology and Bioengineering* **91**, (2005).
2. Rayner, S. G. *et al.* Multiphoton-Guided Creation of Complex Organ-Specific Microvasculature. *Advanced Healthcare Materials* **10**, (2021).
3. Arakawa, C. K., Badeau, B. A., Zheng, Y. & DeForest, C. A. Multicellular Vascularized Engineered Tissues through User-Programmable Biomaterial Photodegradation. *Advanced Materials* **29**, (2017).
4. Lutolf, M. P. & Hubbell, J. A. Synthetic biomaterials as instructive extracellular microenvironments for morphogenesis in tissue engineering. *Nature Biotechnology* vol. 23 47–55 (2005).
5. Zhang, Y. S. & Khademhosseini, A. Advances in engineering hydrogels. *Science* vol. 356 (2017).
6. Deforest, C. A. & Anseth, K. S. Advances in bioactive hydrogels to probe and direct cell fate. *Annual Review of Chemical and Biomolecular Engineering* vol. 3 421–444 (2012).
7. Ruskowitz, E. R. & Deforest, C. A. Photoresponsive biomaterials for targeted drug delivery and 4D cell culture. *Nature Reviews Materials* vol. 3 (2018).
8. Deforest, C. A. & Tirrell, D. A. A photoreversible protein-patterning approach for guiding stem cell fate in three-dimensional gels. *Nature Materials* **14**, 523–531 (2015).
9. Lee, U. N. *et al.* Layer-by-layer fabrication of 3D hydrogel structures using open microfluidics. *Lab on a Chip* **20**, 525–536 (2020).
10. Miller, J. S. *et al.* Rapid casting of patterned vascular networks for perfusable engineered three-dimensional tissues. *Nature Materials* **11**, 768–774 (2012).
11. Kinstlinger, I. S. *et al.* Perfusion and endothelialization of engineered tissues with patterned vascular networks. *Nature Protocols* vol. 16 3089–3113 (2021).
12. Zhao, H. *et al.* O-Nitrobenzyl-alt-(phenylethynyl)benzene copolymer-based nanoaggregates with highly efficient two-photon-triggered degradable properties via a FRET process. *Polymer Chemistry* **7**, 3117–3125 (2016).

13. Khalil, S. & Sun, W. Bioprinting endothelial cells with alginate for 3D tissue constructs. *Journal of Biomechanical Engineering* **131**, (2009).

14. Jia, W. *et al.* Direct 3D bioprinting of perfusable vascular constructs using a blend bioink. *Biomaterials* **106**, 58–68 (2016).

15. Lee, V. K. *et al.* Creating perfused functional vascular channels using 3D bio-printing technology. *Biomaterials* **35**, 8092–8102 (2014).

16. Arakawa, C. *et al.* Biophysical and biomolecular interactions of malaria-infected erythrocytes in engineered human capillaries. *Science Advances* **6**, eaay7243 (2020).

17. Kinstlinger, I. S. *et al.* Generation of model tissues with dendritic vascular networks via sacrificial laser-sintered carbohydrate templates. *Nature Biomedical Engineering* **4**, 916–932 (2020).

18. Nag, A., De, A. K. & Goswami, D. Two-photon cross-section measurements using an optical chopper: Z-scan and two-photon fluorescence schemes. *Journal of Physics B: Atomic, Molecular and Optical Physics* **42**, (2009).

19. Rapp, T. L. & DeForest, C. A. Targeting drug delivery with light: A highly focused approach. *Advanced Drug Delivery Reviews* vol. 171 94–107 (2021).

20. Rapp, T. L. & Dmochowski, I. J. Ruthenium-cross-linked hydrogels for rapid, visible-light protein release. in *Methods in Enzymology* vol. 624 151–166 (Academic Press Inc., 2019).

21. Rapp, T. L., Wang, Y., Delessio, M. A., Gau, M. R. & Dmochowski, I. J. Designing photolabile ruthenium polypyridyl crosslinkers for hydrogel formation and multiplexed, visible-light degradation. *RSC Advances* **9**, 4942–4947 (2019).

22. Rapp, T. L., Highley, C. B., Manor, B. C., Burdick, J. A. & Dmochowski, I. J. Ruthenium-Crosslinked Hydrogels with Rapid, Visible-Light Degradation. *Chemistry - A European Journal* **24**, 2328–2333 (2018).

23. Rapp, T. L., Highley, C. B., Manor, B. C., Burdick, J. A. & Dmochowski, I. J. Ruthenium-Crosslinked Hydrogels with Rapid, Visible-Light Degradation. *Chemistry - A European Journal* **24**, 2328–2333 (2018).

24. Agard, N. J., Prescher, J. A. & Bertozzi, C. R. A strain-promoted [3 + 2] azide-alkyne cycloaddition for covalent modification of biomolecules in living systems. *J Am Chem Soc* **126**, 15046–15047 (2004).

25. Shadish, J. A., Benuska, G. M. & DeForest, C. A. Bioactive site-specifically modified proteins for 4D patterning of gel biomaterials. *Nature Materials* **18**, 1005–1014 (2019).

26. Badeau, B. A., Comerford, M. P., Arakawa, C. K., Shadish, J. A. & Deforest, C. A. Engineered modular biomaterial logic gates for environmentally triggered therapeutic delivery. *Nature Chemistry* **10**, 251–258 (2018).

27. Deforest, C. A., Polizzotti, B. D. & Anseth, K. S. Sequential click reactions for synthesizing and patterning three-dimensional cell microenvironments. *Nature Materials* **8**, 659–664 (2009).

28. DeForest, C. A. & Anseth, K. S. Cytocompatible click-based hydrogels with dynamically tunable properties through orthogonal photoconjugation and photocleavage reactions. *Nature Chemistry* **3**, 925–931 (2011).

29. Pologruto, T. A., Sabatini, B. L. & Svoboda, K. ScanImage: Flexible software for operating laser scanning microscopes. *BioMedical Engineering OnLine* **2**, 13 (2003).

30. Schindelin, J. *et al.* Fiji: an open-source platform for biological-image analysis. *Nature Methods* **9**, 676–682 (2012).

31. Glaser, A. K. *et al.* A hybrid open-top light-sheet microscope for versatile multi-scale imaging of cleared tissues. *Nature Methods* **19**, 613–619 (2022).

Appendix A: Methodology

Synthesis

4-azidobutyronitrile

4-bromobutyronitrile (1 mL, 10 mmol) and sodium azide (1.3 g, 20 mmol) dissolved in DMSO (15 mL) and stirred over night at 55℃. Cooled to room temp, diluted with 50mL water. Extracted via diethyl either (3x 50 mL) dried over Na_2SO_4 ether removed under vacuum resulting in 4-azidobutyrnitirle. Yield 1.016 g, 91%

^{1}H NMR: ($CDCl_3$, 300 MHz) δ 3.50 (t, 2H, *J*=6.34), 2.48 (t, 2H, *J*=7.05), 1.92 (quint, 2H, *J*=12.96).

RuOrange

$Ru(bipyridine)_2Cl_2$ $2H_2O$ (100 mg, 0.19 mmol) and silver hexafluorophosphate (106 mg, 0.43 mmol) dissolved in methanol (stored over sieves, 30mL) and stirred at 55 ℃ for 15 minutes. 4-azidobutyronitrile (100 µL, 1 mmol) was added and stirred for two hours until the color changed from red to orange. The reaction was cooled to room temperature and filtered to remove silver chloride. Then the solvent was evaporated using a rotary evaporator. RuOrange was purified using silica flash chromatography with a 1:4 acetonitrile to dichloromethane as eluent. Collection of the major yellow band.

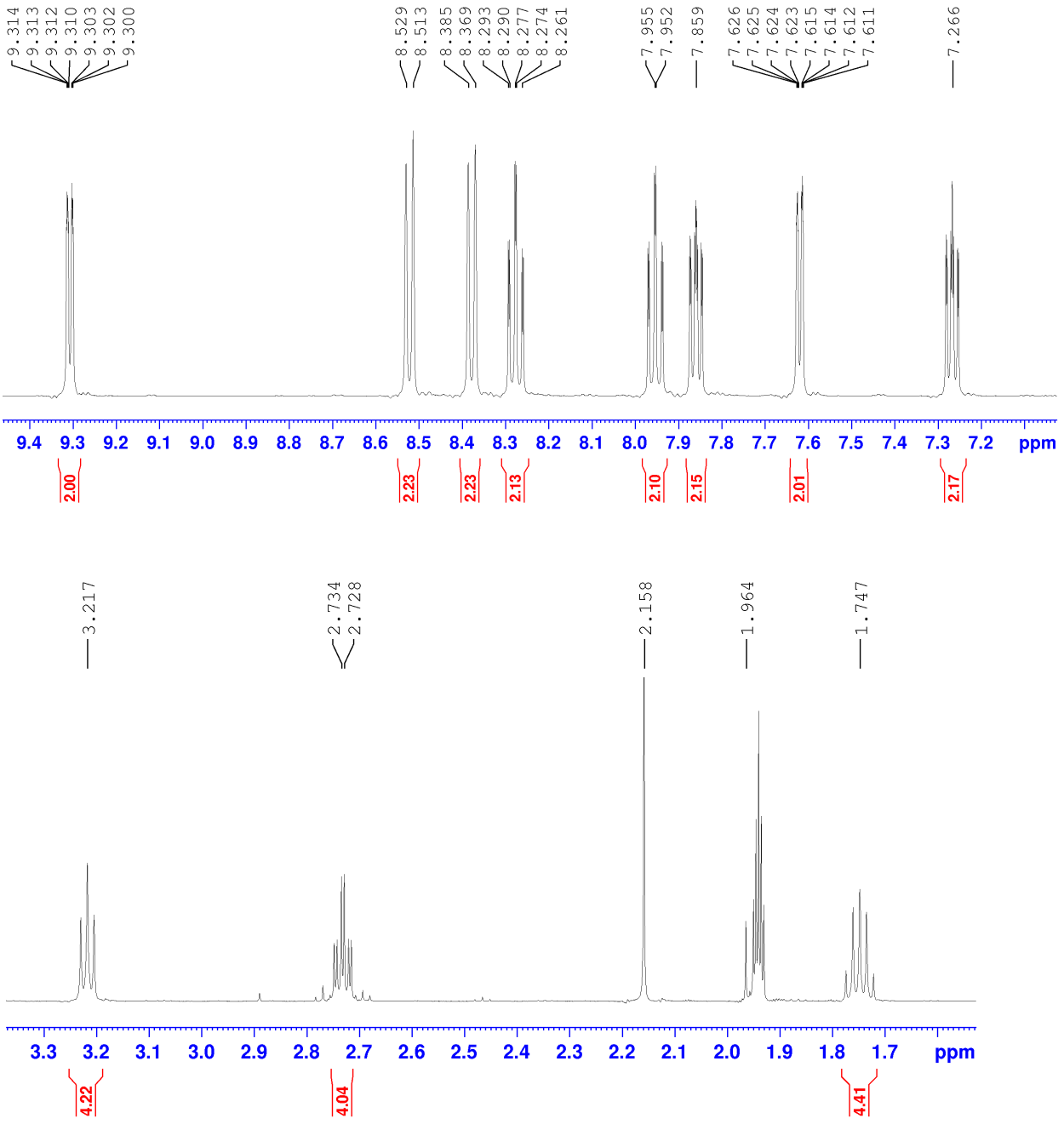


Figure 21 HNMR spectrum for RuOrange

^{1}H NMR (500 MHz, CD_3CN): δ 1.75 (m, 4H, J = 6.52), 2.73 (m, 4H, J = 4.68), 3.22 (t, 4H, J = 6.30), 7.27 (m, 2H, J = 2.46), 7.62 (m, 2H, J = 1.10), 7.86 (m, 2H, J = 2.08), 7.95 (m, 2H, J = 3.46), 8.28 (m, 2H, J = 3.43), 8.38 (d, 2H, J = 8.04), 8.52 (d, 1H, J = 8.07), 9.31 (d, 2H, J = 1.08)

Hydrogel Design

Hydrogels were formed using PEG-BCN at a final concentration of 3mM. PEG-BCN was fluorescently stained by SPAAC with azide Alexa Fluro 568 or azide CY5 fluorophore. These were coordinated at a 1:200 fluorophore to BCN groups. The fluorophore coordinated PEG-BCN and buffer were added together at room temperature and left for 30 minutes. The crosslinker was added prior to casting.

Gels were cast between a Rain-X treated glass slide, and an azide-treated glass slide. The slides were held apart with a 1 mm thick rubber spacer. Gelation occurred for 30 minutes. Upon slide separation the gel was equilibrated in PBS overnight.

Compound	Stock Concentration (mM)	Final Concentration (mM)	Volume added for 30 µL gel (µL)
PEG-BCN	10	3	12
Cross Linker	40	6	6
PBS			12

Cell Viability

Hydrogels for cell culture included azide RGD (N_3-GRGDS) at a concentration of 1mM.

Compound	Stock Concentration (mM)	Final Concentration (mM)	Volume added for 5 µL gel (µL)
PEG-BCN	10	3	1.5
Cross Linker	40	5	0.63
RGD	40	1	.13
Buffer			2.74

Patterning Hydrogels

With the exception of the *o*NB crosslinked gel, all patterns utilized the Print3D widget in “Fast” mode with the use of pre-calculating waveforms for each volume the “fast z” piezo controller

and the resonant scanner. oNB crosslinked gel utilized the “slow” linear scanning patterning feature. Each pattern was done at a 2x magnification and 100% laser intensity.

The wavelength study utilized the virtual mask of a single white bar. The mask stack consisted of 10 images and were patterned at 1 µm layer spacing. A single repeat at 88 ns pixel dwell time at a resolution of 512 pixels x 512 pixels. These settings were used varying only the location for each pattern. The tiling pattern application was used for each row’s pattern shifting the mask 60 µm in the positive x direction. The wavelength was adjusted for each respected row, repeating the same process.

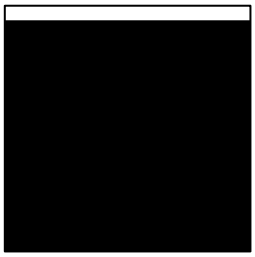


Figure 22 Photomask for the wavelength study.

The varying repeats and z-step pattern utilized the same virtual mask stack, consisting of 10 of the below masks. Each mask patterned using the same microscope settings, 750 nm wavelength, 88 ns dwell time at a 512 pixel x 512 pixel resolution. The power grid feature built within the Tile application was used to vary laser intensity and shift the pattern 60 µm. Either repeats or z step was adjusted for each row prior to executing the rows pattern.

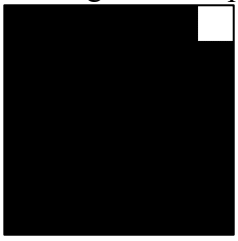


Figure 23 Photomask for the repeat and z step characterization study.

The dextran studies were conducted with the gel submerged in the 1mg/ml dextran solution. A rubber gasket of 2mm was placed around the gel. The gasket formed a chamber of which was filled with the fluorescent dextran solution. This chamber was sealed with a coverslip and then submerged in PBS for patterning. The mask for the varying chamber size consisted of 10 images and patterned using 88ns dwell time at 512x512 pixel resolution at 100% laser intensity at 750nm and a single repeat.

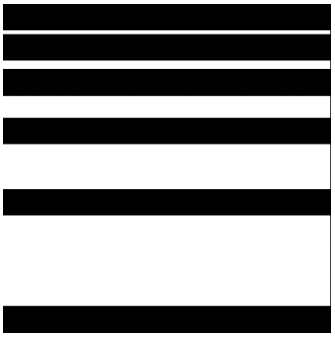


Figure 24 Photomask for the dextran perfusion with varing channel sizes ranging from 5 µm to 100 µm.

The gremlin Gizmo was patterned in both an *o*NB gel and a RuOrange gel. The same photo mask derived from the "Gizmo" model by Multiverse3DDesigns licensed under Creative Commons licenses and freely available through Thingiverse. The .stl file was sliced in Autodesk Netfabb and used for both patterns.

The *o*NB hydrogel used for time comparison The hydrogel used was a 20 kDa PEG-tetraBCN at (4mM) coordinated with AlexaFluor-N_3-594, NPPOC-NHO-TEG-N_3 (1mM) TEG-diazide (7.5mM).

4 µL of PEG-tetraBCN (10mM stock) was reacted with 1 µL of NPPOC-NHO-TEG-N_3 (10mM stock) at RT for 30 minutes. 3.125 µl of PBS was added at RT and let sit for 30 minutes. 1.875ul

of TEG-diazide(40 mM stock) was added at RT. The gel was then cast between a Rain-X treated slide and an azide treated glass slide. After 30 minutes the Rain-X slide was removed and the gel attached to the azide slide were soaked in PBS overnight prior to patterning.

Post Patterning: The patterned gel was left overnight in 5 mL of PBS with FAM -BA (10nM) and Aniline(100mM). After the overnight bath, the gel was washed with PBS prior to imaging.

Patterning occurred with a 760nm laser at 100% power using 64 repeats at a 3200 ns pixel dwell time and a zstep of 1 µm.

For the RuOrange hydrogel patterning 750 nm laser at 100% power was used in the “fast” resonant patterning mode. Using 1 repeat at 88 ns pixel dwell time and a z step of 1 µm.

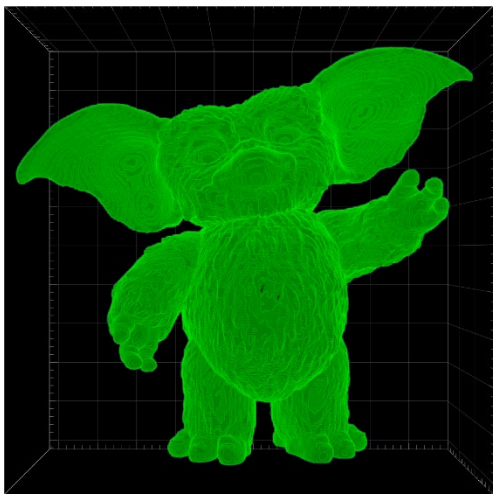


Figure 25 3D rendering of the .stl file used for the Gizmo mask generation.

Dendritic vascular network utilized the same physical set up and patterning technique as the dextran studies. Where the gel is submerged in the 1mg/mL fluorescent dextran solution. A rubber gasket of 2mm was placed around the gel. The gasket formed a chamber of which was filled with the dextran solution. This chamber was sealed with a coverslip and then submerged in PBS for patterning. The mask was created by slicing the .stl file into 1 μm steps. The developed tile pattern application was used to crop the image stack into 2 patternable volumes. Each was patterned using 88ns dwell time at 512x512 pixel resolution at 100% laser intensity at 750nm and a single repeat utilizing the tile pattern feature. To show resolution a virtual measuring device was used in Imaris. Set as sphere with a diameter of 10 μm the <10 μm channels can clearly be seen.

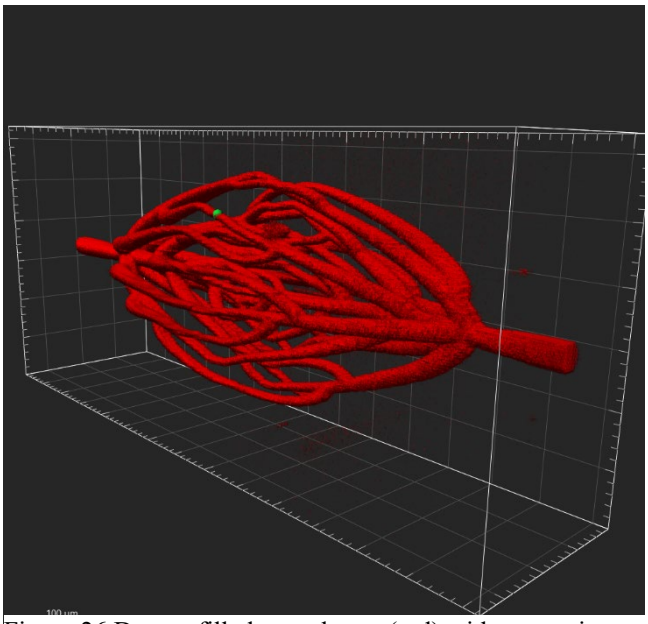


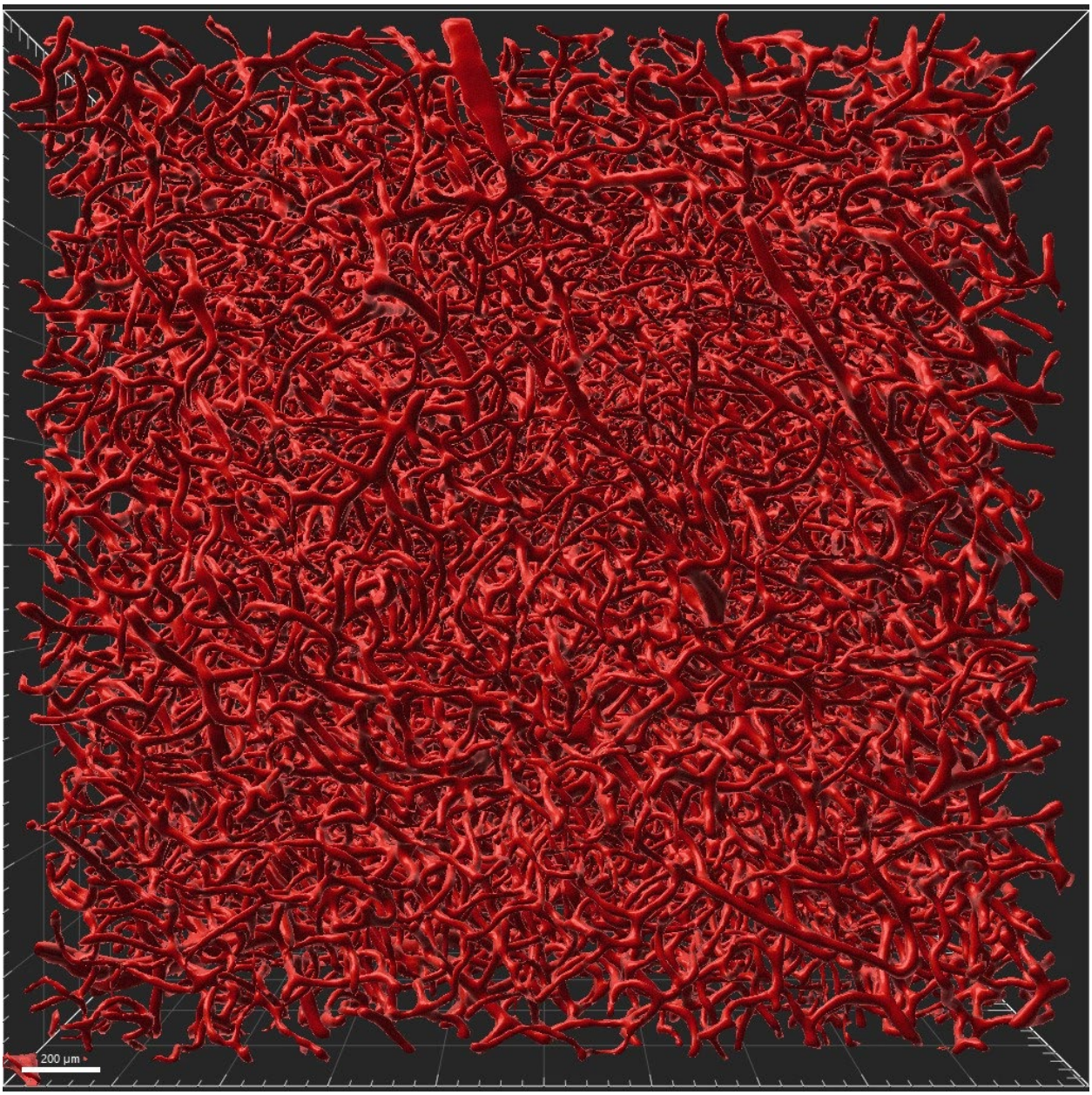
Figure 26 Dextan filled vasculature (red) with measuring feature of 10 μm diameter (green).

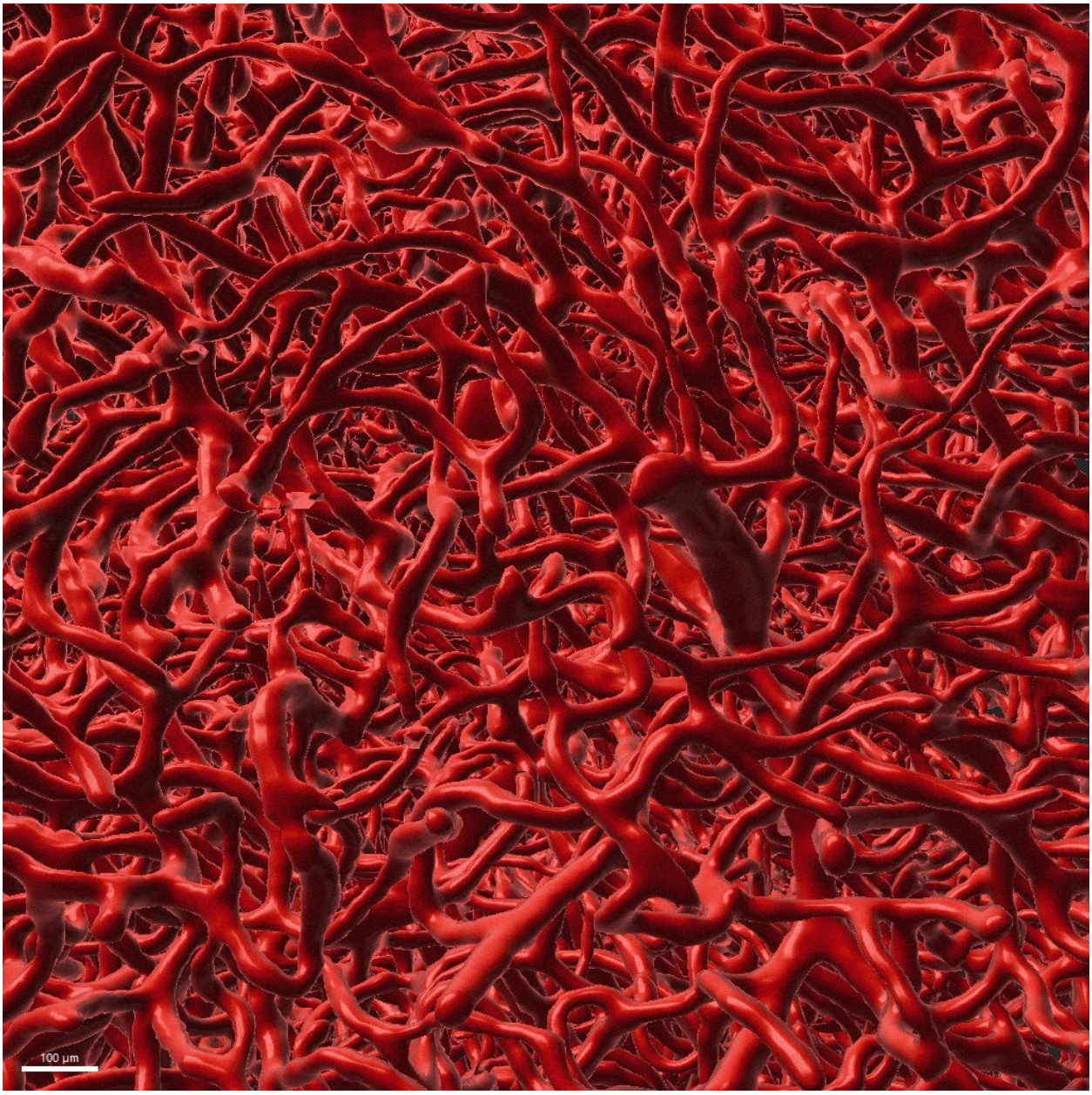
The mouse brain vasculature was patterned using the tile patterning application. The cropping required usable volumes to be 181 pixels by 181 pixels for keeping the scale while patterning.

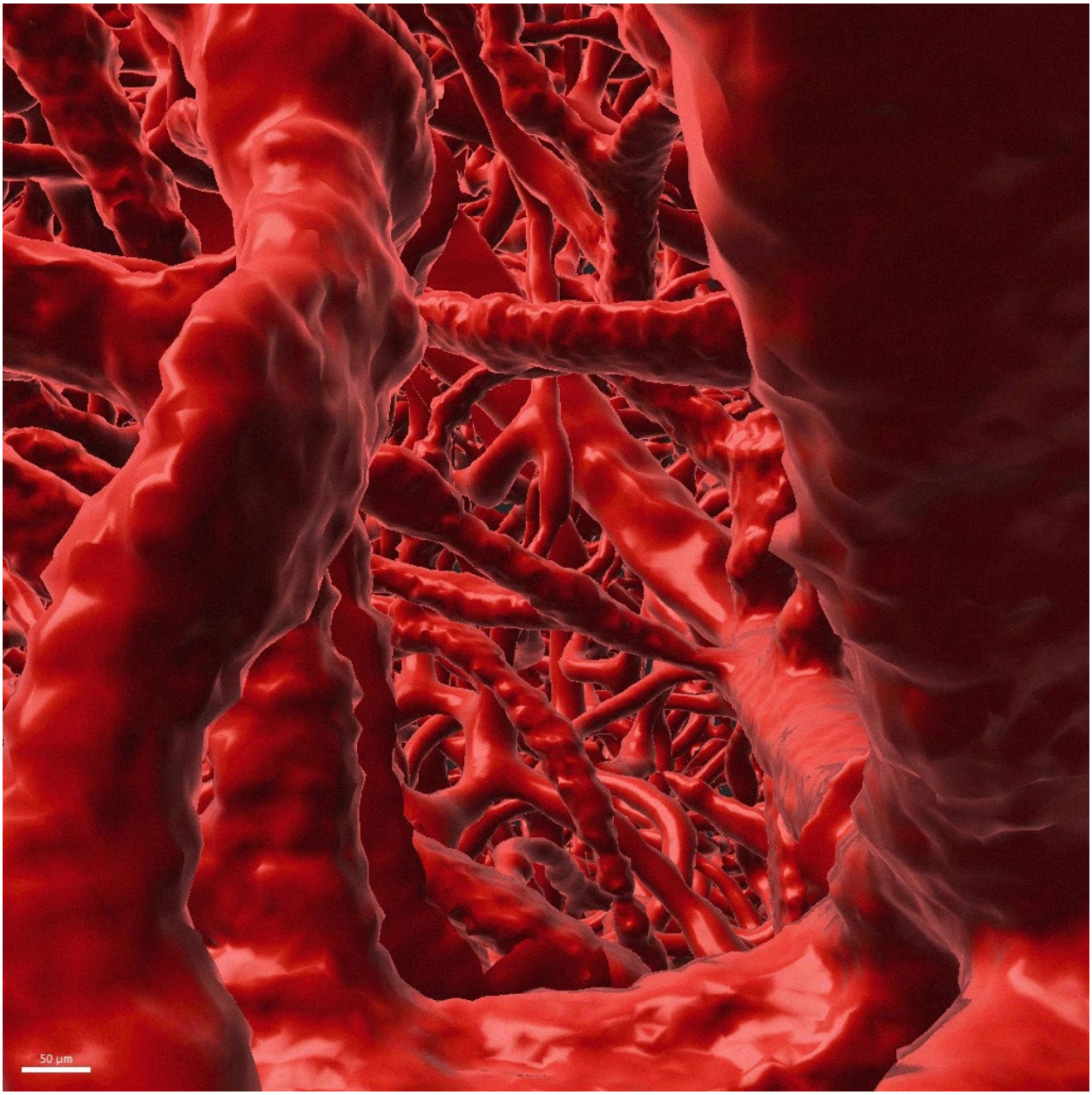
This allowed for patterning to take place at a 256 x 256 pixel resolution and a dwell time of 175ns. 100% laser intensity at 750nm and a single repeat utilizing during patterning. The tile application was used for both cropping and patterning all 180 volumes.

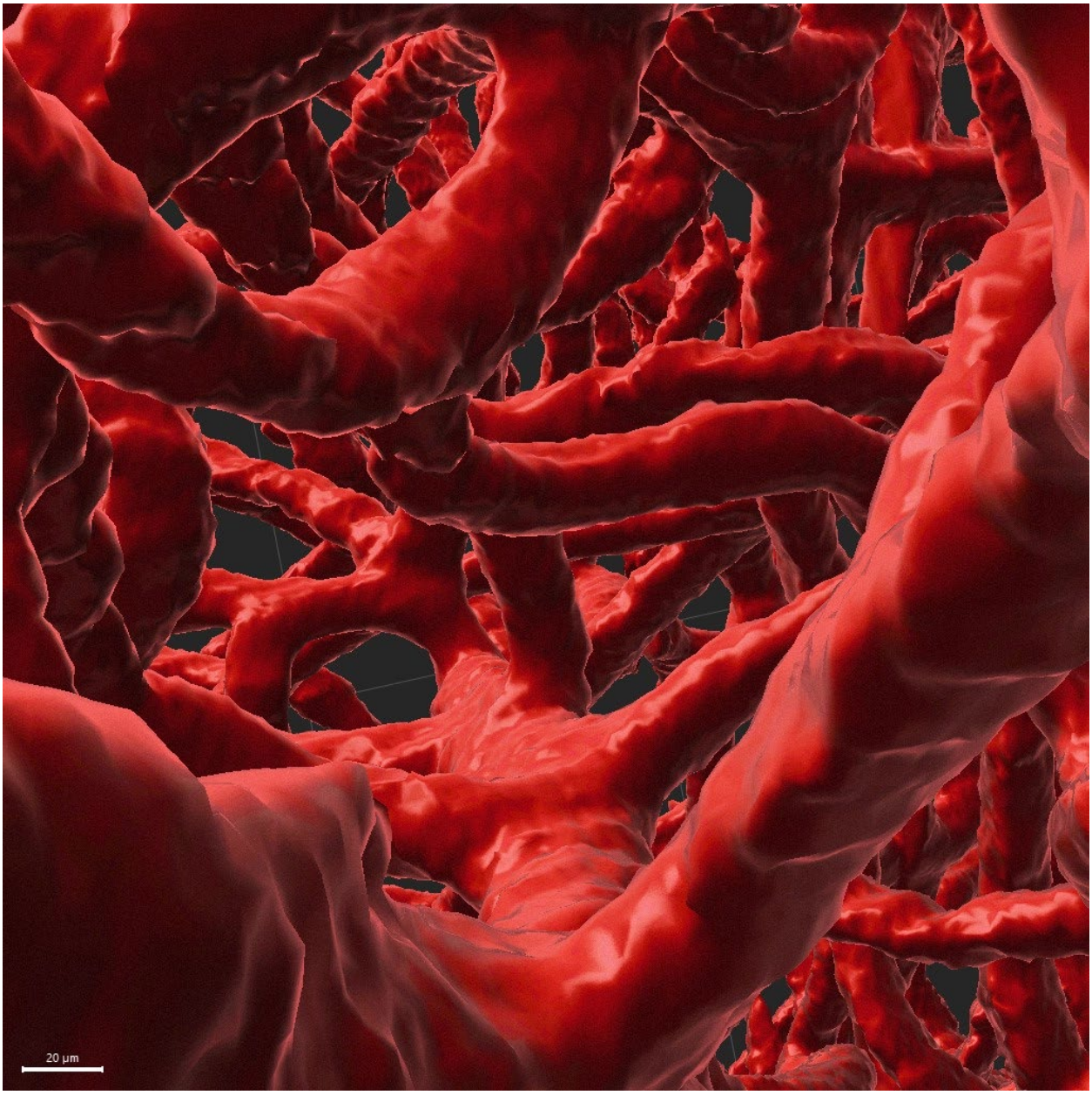
Appendix B: Microvasculature Detail

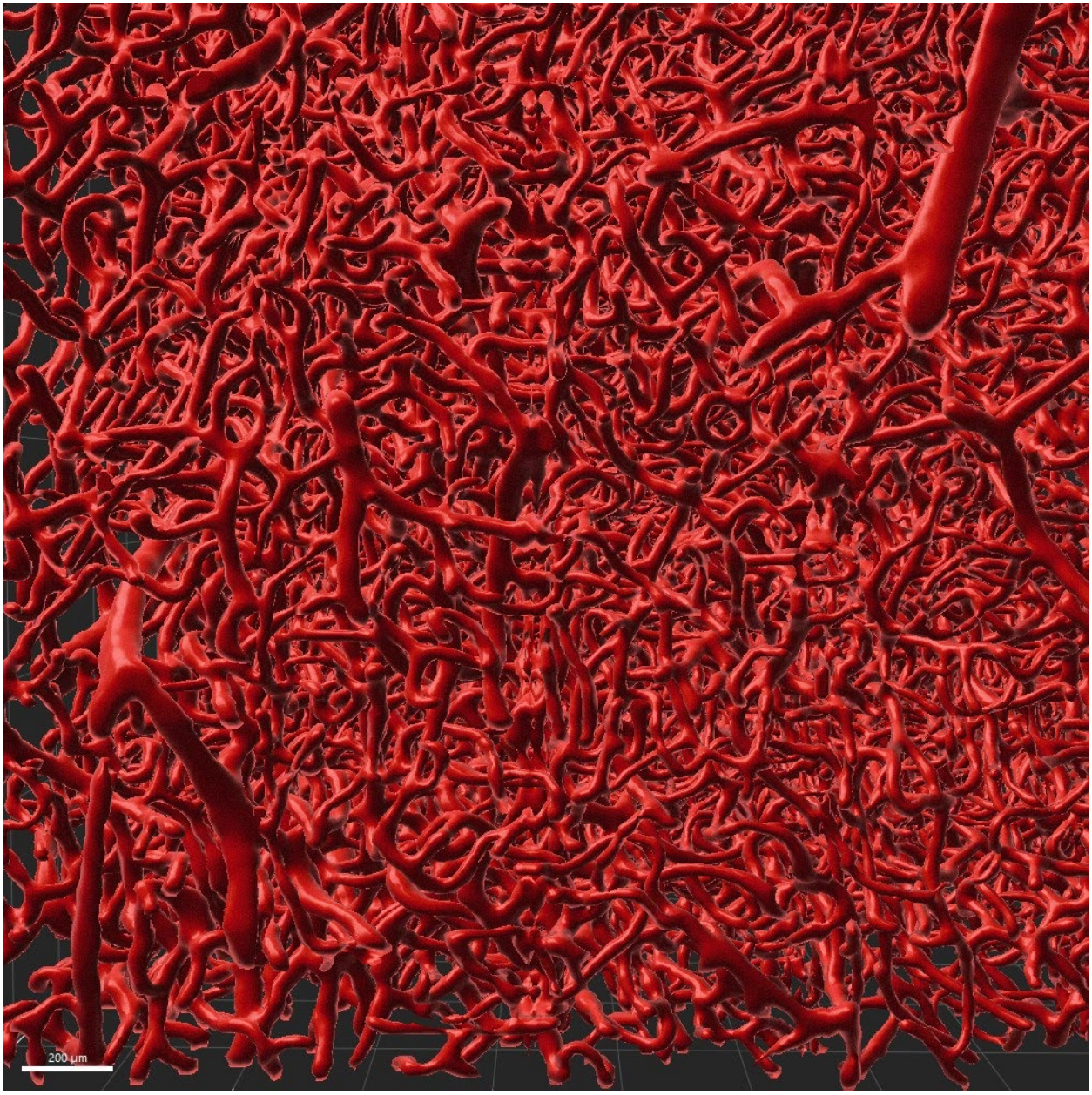
Multiple photos were taken from within the patterned brain microvasculature using Imaris.




100 µm


50 µm


20 µm


200 µm

Appendix C: MATLAB Code

UI.m

```
global nLayers layerHeight spacing zstep rowsize columnsize laserpower;
global nCols nRows repeat colorIdx ressonantScanIdx desired bmp powergrid;
global overlap repeats zGrid repGrid;
%%%%DEFAULTS
%%% PHOTO CROPPING %%%

ressonantScanIdx = true;
rowsize = 513; %How large should the images be cropped?
columnsize = 513; %How large should the iamges be cropped?
desired = 10; %desired number of images
layerHeight = 450; %Max Height for Piezo, can differ if req
overlap = 0; %number of pixles to overlap
colorIdx = true;% Leave as true for image to remain the same, flase switches color
repeat = false; %Do you want to repeat the same image file
bmp = true; %use true for .bmp photos false for .tif

%%% Microscope Settings%%%
nCols = 1; %Number of Cols for the Pattern This will not print the full pattern
unless it matches
nRows = 1; %Number of Rows for the pattern This will not print the full pattern
unless it matches
nLayers = 1;%Number of Layers for the pattern
spacing = 360; %360 res zoom x2
zstep = 1; %Used for Calculating the Volume Size
repeats = 1;
laserpower = 100;%desired laser power setting during printing
powergrid = false;
zGrid = false;
repGrid = false;

load("appvars.mat");
```

startPat.m

```
%This script initilizes the tile patterning

%*****************Warnings************************************
%There are currently no safeties to prevent crashing/running the limits of
%any direction including the z direciton. Build yourself a buffer, please double
check
%grid size before starting

global yyGrid xxGrid orriginalPosition gridIdx layerIdx startPosition;
global gridIsExecuting powergrid hBeams nCols laserpower nRows files fastz;
global spacing hPrint repeats zstep zGrid repGrid;
UI();

[xxGrid,yyGrid] = ndgrid(0:spacing:spacing*(nCols-1),0:spacing:spacing*(nRows-1));
%sets up the grid based on user variables
startPosition = hSI.hMotors.samplePosition; %gets current motor position, this
includes slow z position.
orriginalPosition = hSI.hMotors.samplePosition; %gets current motor position, this
includes slow z position.
gridIdx = 1; %initializes grid Index
layerIdx = 1; %Initializes layer index
hResourceStore = dabs.resources.ResourceStore(); %calls for ScanImage functions
fastz = hResourceStore.filterByName('Piezo'); %sets fastz equal to a ScanImage
function
hSI.hFastZ.move(fastz, 0); %Sets the fast Z, PIEZO, to Zero as part of intilaziation
hPrint = hResourceStore.filterByName('Print 3D'); %Again finds the ScanImage function
hBeams = hSI.hBeams;
hPrint.numRepeats = repeats;
hPrint.zStep_um = zstep;
gridIsExecuting = true; %%**********DO NOT TOUCH*****************
disp('Executing grid position 1, layer 1');%Prints to let user know first grid is
printing
Load();%this should call the Load Script to creat a image array from the load script
function, based on the grididx and layeridx
disp(append('Loading of cell ',num2str(gridIdx),' Layer ',num2str(layerIdx),'
complete')); %display to user at when loading is complete.
hPrint.loadImageStack(files);
if powergrid == true
  hBeams.powers(4) =((laserpower / (nCols*nRows)) * gridIdx);
else
  hBeams.powers(4) = laserpower;
end
if zGrid == true
  hPrint.zStep_um =((zstep / (nCols*nRows)) * gridIdx);
else
  hPrint.zStep_um = zstep;
end
if repGrid == true
  hPrint.numRepeats =((repeats / (nCols*nRows)) * gridIdx);
else
  hPrint.numRepeats = repeats;
end
hPrint.start();
```

stopPat.m

```
%Ends the tile pattern after current pattern is complete
global gridIsExecuting
gridIsExecuting = false; %**********This turns off the aquisition in
endScript!*********
disp('USER REQUESTED STOP'); %Tells the user the project is complete
hBeams.powers(4) = 0; %Turns off the laser
```

nextPat.m

```
%Calls the next grid of the tile pattern

global xxGrid yyGrid orriginalPosition gridIdx laserpower layerIdx;
global layerHeight nLayers startPosition gridIsExecuting hPrint;
global files repeat powergrid zGrid repGrid zstep repeats;

layerIdx = layerIdx+1; %adds another index to track which print is running

if layerIdx>nLayers %if statement if the layer is completed
  gridIdx = gridIdx+1; %if layer is compolete the next grid needs to be started
  disp('Next grid'); %Displays next layer to tell user it made it this far
  baselayer = [0 0 ((nLayers-1)*layerHeight)];
  startPosition = (startPosition - baselayer);

  layerIdx = 1; %resets the grid index to reprint the grid patern
else
  disp('Next Layer');
  nextlayer = [0 0 layerHeight]; %defines what a layer height is based on layer
number
  startPosition = (startPosition + nextlayer); %creates a new start position with the
new layer

end %ends the if statement

if gridIdx>numel(xxGrid) %if statement for if the grids are complete.
  gridIsExecuting = false; %*******This turns off the aquisition in endScript!
  disp('All done'); %Tells the user the project is complete
  hBeams.powers(4) = 0;
  hSI.hMotors.moveSample(orriginalPosition); %Motors will return to the orginal
positon. (First layer)
  hSI.hFastZ.move(fastz, 0);
end

if gridIsExecuting == true
  nextGridPosition = [xxGrid(gridIdx) yyGrid(gridIdx) 0]; % changing the zero here to
create a 3D layer (next grid positon)
  nextSamplePosition = startPosition + nextGridPosition; %Creates a new positoin
based on the index
  hSI.hMotors.moveSample(nextSamplePosition); %move the motors to the next position
  fprintf('Executing grid position %d layer %d\n',gridIdx, layerIdx); %Tells user
what grid positon is printing
  if repeat == false %Repeat True calls load, False runs the same pattern
    Load();
  else
  end
  disp(append('Loading of cell ',num2str(gridIdx),' Layer ',num2str(layerIdx),'
complete'));
  hPrint.loadImageStack(files);
  if powergrid == true
    hBeams.powers(4) =((laserpower / (nCols*nRows)) * gridIdx);
  else
    hBeams.powers(4) = laserpower;
  end
```

```
  if zGrid == true
    hPrint.zStep_um =((zstep / (nCols*nRows)) * gridIdx);
  else
    hPrint.zStep_um = zstep;
  end
  if repGrid == true
    hPrint.numRepeats =((repeats / (nCols*nRows)) * gridIdx);
  else
    hPrint.numRepeats = repeats;
  end
  hPrint.start();

end

%***************Comments************************************
```

Load.m

```
%This function is utilized for loading corresponding stacks for tile pat
global location gridIdx files layerIdx nImglayer numberofimages

layerIDX = layerIdx -1; %creates the counting variable for the layer counting, it is
-1 to account for the remander file
disp(append('Loading of cell ',num2str(gridIdx), ' Layer ',num2str(layerIdx)));
%Display for user to see what file is loading
files = {}; %creates a blank cell array for file loacitons.

fullarray = {}; %need an additional array for counting
  for j = (nImglayer * layerIDX)+2: ((nImglayer * layerIDX)+ nImglayer) %counts the
number of files that would be in each folder img x -> y
    if j < numberofimages+2 %ensures there are not ghost images created after all the
images are counted, the +2 accounts for the img # 0
      if bmp == true
        pic = sprintf('%d.bmp',(j-1));
      else
        pic = sprintf('%d.tif',(j-1));
      end
      folder = append('Layer ',num2str(layerIdx),'\',num2str(gridIdx),'\'); %folder
at which the file is located based on global layerIdx and gridIdx
      fullarray{j} = fullfile(location,folder,pic); %creats a single array of all of
the image files
      empties = find(cellfun(@isempty,fullarray)); % identify the empty cells
      fullarray(empties)= []; %deletes the emptie cells to creat a array of only
populated cells, the files we actually have in the folder
      files = fullarray; %sets the array equal to the global value of an array.
    end
  end
```

addFile.m

```
% addFile is utilized to select a preexisting cropped folder. This scritp
% is initiated by selecting Load PreCrop File
global location numberofimages zstep desired nImglayer layerHeight;
UI(); %Runs UI to get required variables

[location] = uigetdir(); %Opens the file browser
numberofimages = desired; %Necessary for running Load
nImglayer = ceil(layerHeight / zstep); %Redefnes for Load
disp(append('Complete, Reading From ',location)); %User Interaction
```

ImageSort.m

```
%  This Image Sort Code not only sorts desired images into proper folders
%  for utilizaiton of the grid pattern program
%  but it also will crop the images depeding on desired row/column size and
% overlap. Utilize the UI for defining variables and is currently capable
% .tif and bmp
global rowsize columnsize numberofimages location nImglayer zstep desired repeat bmp
layerHeight ressonantScanIdx colorIdx;
UI();
rSize = rowsize;
cSize = columnsize;
[file,location] = uigetfile('*.*',...
  'Select One or More Files', ...
  'MultiSelect', 'on');
if iscell(file) == 0
  file = cellstr(file);
else
end
[m,n] = size(file);
count = (n./desired);
if n >= desired
  count = (n/desired);
  sorted = file(:,1:count:end);
else
  repcount = ceil(desired/n);
  file_rep = repelem(file,1,repcount);
  [m1,n1] = size(file_rep);
  if n1 > desired
    count = n1/desired;
    sorted = file_rep(:,1:count:end);
  else
    sorted = file_rep;
  end
end
numberofimages = length(sorted);
folder = fullfile(location);
vertvol = zstep * numberofimages; %Calculates the vertical space the stack will take
up
nLayers = (vertvol/layerHeight); %calculates the number of layers required based on
the 450 limit of the piezo (rounds up accounting for any remainders)
fullLayers = floor(nLayers);
NumberofLayers = ceil(nLayers);
nLayers = NumberofLayers;
nImglayer = ceil(layerHeight / zstep);
layerIDX = 0; %sets the local counter to zero
y=0;
f = waitbar(y/numberofimages,'Image Crop and Save');
for j = 1:nLayers
  if exist(fullfile(location, append('Layer ',sprintf('%d',j)),'dir')) %not sure why
but this keeps errors from being thrown should if ~exist but exist avoids the error
    mkdir(fullfile(location, append('Layer ',sprintf('%d',j))));
  end
  for y = ((nImglayer * layerIDX)+1: (nImglayer * layerIDX)+ nImglayer)
  rgbImage = imread(strcat(location,cell2mat(sorted(1,y))));
```

```
  [rows, columns, numberOfColorBands] = size(rgbImage);
      if repeat == true
       rSize = (rows+1);
       cSize = (columns+1);
      end
      blockSizeR = rSize; % Rows in block.
      blockSizeC = cSize; % Columns in block.
      wholeBlockRows = floor(rows / blockSizeR);
      blockVectorR = [blockSizeR * ones(1, wholeBlockRows), rem(rows, blockSizeR)];
      wholeBlockCols = floor(columns / blockSizeC);
      blockVectorC = [blockSizeC * ones(1, wholeBlockCols), rem(columns,
blockSizeC)];
      if numberOfColorBands > 1
        ca = mat2cell(rgbImage, blockVectorR, blockVectorC, numberOfColorBands);
      else
        ca = mat2cell(rgbImage, blockVectorR, blockVectorC);
      end
      cropIndex = 1;
      numPlotsR = size(ca, 1);
      numPlotsC = size(ca, 2);
      for r = 1: numPlotsR
        for c = 1:numPlotsC
          rgbBlock = ca{r,c};
          if bmp == true
            filename = sprintf('%d.bmp',y);
          else
            filename = sprintf('%d.tif',y);
          end
          [rowsB, columnsB, numberOfColorBandsB] = size(rgbBlock);
          if ~exist(fullfile(location, append('Layer ',sprintf('%d',j),'\',
sprintf('%d',cropIndex))))
          mkdir(fullfile(location, append('Layer ',sprintf('%d',j),'\',
sprintf('%d',cropIndex))));
          end
          %Invert the color if desired
          if colorIdx == false
            bw = imcomplement(rgbBlock);
          else
            bw = rgbBlock;
          end
          % Fullfill the image to creat a sqare
          if rowsB < rSize || columnsB < cSize
            bw(rSize,cSize,:) = 0;
          end
          % Flips the individual cells for resscaning
          if ressonantScanIdx == true
            fy = flip(bw, 2);
          else
            fy = bw;
          end
          %Writes the files into the correct location
          imwrite(fy, fullfile(location, append('Layer ',sprintf('%d',j),'\'),
sprintf('%d',cropIndex),filename));
          cropIndex = cropIndex + 1;
```

```
          waitbar(y/(numberofimages-1),f);
        end
      end
  if y > numberofimages-1
    break
  end
  end
    layerIDX = layerIDX + 1;
end
close(f);
disp(append('Complete, saved at ',location));
```

endScript.m

```
% This funciton runs after every Print3D run and will determine if the
% program ends or continues to the next pattern in the grid.
global oldPowerFractions;
global oldShutters;
global gridIsExecuting;
global fastz;
hSI.hShutters.shuttersTransitionAll(false);

hSI.hBeams.powerFractions = oldPowerFractions;
hResourceStore = dabs.resources.ResourceStore();

if ~isempty(gridIsExecuting) && gridIsExecuting==true
  hResourceStore = dabs.resources.ResourceStore();
  fastz = hResourceStore.filterByName('Piezo');
  hSI.hFastZ.move(fastz, 0); %Sets the fast Z, PIEZO to Zero
  nextPat();
else
  hSI.hScan2D.hShutters = oldShutters;
end
```

TileControls.mlapp

```
classdef TileControls < matlab.apps.AppBase


  % Properties that correspond to app components
  properties (Access = public)
    ControlAppUIFigure       matlab.ui.Figure
    DesiredMaxLabel        matlab.ui.control.Label
    VaryValuesLabel        matlab.ui.control.Label
    RepeatGrid           matlab.ui.control.Switch
    zStepGrid           matlab.ui.control.Switch
    RepeatsumEditField       matlab.ui.control.NumericEditField
    RepeatsumEditFieldLabel    matlab.ui.control.Label
    PowerGridSwitch        matlab.ui.control.Switch
    PowerGridSwitchLabel      matlab.ui.control.Label
    LoadPreCropFileButton     matlab.ui.control.Button
    STOPPATTERNButton       matlab.ui.control.Button
    ResScanningSwitch       matlab.ui.control.Switch
    ResScanningSwitchLabel     matlab.ui.control.Label
    ImageTypeSwitch        matlab.ui.control.Switch
    ImageTypeSwitchLabel      matlab.ui.control.Label
    RepeatSameStackSwitch     matlab.ui.control.Switch
    RepeatSameStackSwitchLabel   matlab.ui.control.Label
    InvertBackgroundSwitch     matlab.ui.control.Switch
    InvertBackgroundSwitchLabel  matlab.ui.control.Label
    zStepumEditField        matlab.ui.control.NumericEditField
    zStepumEditFieldLabel     matlab.ui.control.Label
    GridSpacingumEditField     matlab.ui.control.NumericEditField
    GridSpacingumEditFieldLabel  matlab.ui.control.Label
    NumberofLayersEditField    matlab.ui.control.NumericEditField
    NumberofLayersEditFieldLabel  matlab.ui.control.Label
    NumberofRowsEditField     matlab.ui.control.NumericEditField
    NumberofRowsEditFieldLabel   matlab.ui.control.Label
    NumberofColumnsEditField    matlab.ui.control.NumericEditField
    NumberofColumnsEditFieldLabel matlab.ui.control.Label
    LayerHeightEditField      matlab.ui.control.NumericEditField
    LayerHeightEditFieldLabel   matlab.ui.control.Label
    DesiredPicturesEditField    matlab.ui.control.NumericEditField
    ofPicturesLabel        matlab.ui.control.Label
    ColumnSizeEditField      matlab.ui.control.NumericEditField
    ColumnSizepxEditFieldLabel   matlab.ui.control.Label
    RowSizeEditField        matlab.ui.control.NumericEditField
    RowSizepixLabel        matlab.ui.control.Label
    LaserPowerSlider        matlab.ui.control.Slider
    LaserPowerLabel        matlab.ui.control.Label
    CancelButton          matlab.ui.control.Button
    StartPatternButton       matlab.ui.control.Button
    StartSortCropButton      matlab.ui.control.Button
  end
```

```
  properties (Access = private)

  end

  properties (Access = public)
    ressonantScanIdx = false;
    nLayers; %Number of Layers Blocks to be Patterned
    layerHeight %Height of each layer (max of 450 for fast patterning)
    spacing %Spacing between each grid, adjust for magnificaiton
    zstep %distance between each z slice(um)
    rowsize %Pixels in each row.
    columnsize %Pixels in each column.
    laserpower %Power of laser while patterning
    nCols %Number of Columns in the pattern
    nRows %Number of rows in the the pattern
    repeat %Does the same image stack repeat? True no crop, false crop
    colorIdx %Does the background need to be inverted?true for image to remain the
same, flase switches color
    scanidx %This mirrors each crop for res.scanning.
    desired %The desired number of images to be patterned.
    bmp %Raw file type, bmp or TIF, use true for .bmp photos flase for .tif
    overlap % Overlap of the croped images (pixels) % Description
    Welcome = 'Welcome'

 % Description
  end

  methods (Access = public)

  end


% Callbacks that handle component events
methods (Access = private)


% Code that executes after component creation
function startupFcn(app)
Welcome = 1;
save('appvars.mat','Welcome');
end


% Button pushed function: CancelButton
function CancelButtonPushed(app, event)
UI();
closereq();
end


% Button pushed function: StartPatternButton
```

```
function StartPatternButtonPushed(app, event)
UI();
disp('Ensure settings are correct on the app then Run startPat.m')
end


% Button pushed function: StartSortCropButton
function StartSortCropButtonPushed(app, event)
ImageSort();
end


% Value changed function: ResScanningSwitch
function ResScanningSwitchValueChanged(app, event)
value = app.ResScanningSwitch.Value;
if strcmp(value,'Yes')
ressonantScanIdx = true;
else
ressonantScanIdx = false;
end
save('appvars.mat','ressonantScanIdx','-append');
end


% Value changed function: ImageTypeSwitch
function ImageTypeSwitchValueChanged(app, event)
value = app.ImageTypeSwitch.Value;
if strcmp(value, '.bmp')
bmp = true;
else
bmp = false;
end
save('appvars.mat','bmp','-append');
end


% Value changed function: InvertBackgroundSwitch
function InvertBackgroundSwitchValueChanged(app, event)
value = app.InvertBackgroundSwitch.Value;
if strcmp(value,'Yes')
colorIdx = false;
else
colorIdx = true;
end
save('appvars.mat','colorIdx','-append');
end


% Value changed function: RepeatSameStackSwitch
function RepeatSameStackSwitchValueChanged(app, event)
value = app.RepeatSameStackSwitch.Value;
if strcmp(value, 'Yes')
```

```
repeat = true;
else
repeat = false;
end
save('appvars.mat',"repeat",'-append');
end


% Value changed function: RowSizeEditField
function RowSizeEditFieldValueChanged(app, event)
value = app.RowSizeEditField.Value;
rowsize = value;
save('appvars.mat',"rowsize",'-append');
end


% Value changed function: ColumnSizeEditField
function ColumnSizeEditFieldValueChanged(app, event)
value = app.ColumnSizeEditField.Value;
columnsize = value;
save('appvars.mat',"columnsize",'-append');
end


% Value changed function: DesiredPicturesEditField
function DesiredPicturesEditFieldValueChanged(app, event)
value = app.DesiredPicturesEditField.Value;
desired = value;
save('appvars.mat','desired','-append');
end


% Value changed function: NumberofColumnsEditField
function NumberofColumnsEditFieldValueChanged(app, event)
value = app.NumberofColumnsEditField.Value;
nCols = value;
save('appvars.mat','nCols','-append');
end


% Value changed function: NumberofRowsEditField
function NumberofRowsEditFieldValueChanged(app, event)
value = app.NumberofRowsEditField.Value;
nRows = value;
save('appvars.mat',"nRows",'-append');
end


% Value changed function: NumberofLayersEditField
function NumberofLayersEditFieldValueChanged(app, event)
value = app.NumberofLayersEditField.Value;
nLayers = value;
```

```
save('appvars.mat',"nLayers",'-append');
end


% Value changed function: GridSpacingumEditField
function GridSpacingumEditFieldValueChanged(app, event)
value = app.GridSpacingumEditField.Value;
spacing = value;
save('appvars.mat',"spacing",'-append');
end


% Value changed function: zStepumEditField
function zStepumEditFieldValueChanged(app, event)
value = app.zStepumEditField.Value;
zstep = value;
save('appvars.mat',"zstep",'-append');
end


% Value changed function: LaserPowerSlider
function LaserPowerSliderValueChanged(app, event)
value = app.LaserPowerSlider.Value;
laserpower = value;
save('appvars.mat',"laserpower",'-append');
end


% Callback function
function CropOverlapSliderValueChanged(app, event)
value = app.CropOverlapSlider.Value;
overlap = value;
save('appvars.mat',"overlap",'-append');
end


% Button pushed function: STOPPATTERNButton
function STOPPATTERNButtonPushed(app, event)
stopPat();
end


% Button pushed function: LoadPreCropFileButton
function LoadPreCropFileButtonPushed(app, event)
addFile();
end


% Value changed function: PowerGridSwitch
function PowerGridSwitchValueChanged(app, event)
value = app.PowerGridSwitch.Value;
if strcmp(value,'Off')
```

```
powergrid = false;
else
powergrid = true;
end
save('appvars.mat','powergrid','-append');
end


% Value changed function: LayerHeightEditField
function LayerHeightEditFieldValueChanged(app, event)
value = app.LayerHeightEditField.Value;
layerHeight = value;
save('appvars.mat',"layerHeight",'-append');
end


% Value changed function: RepeatsumEditField
function RepeatsumEditFieldValueChanged(app, event)
value = app.RepeatsumEditField.Value;
repeats = value;
save('appvars.mat',"repeats",'-append');
end


% Value changed function: zStepGrid
function zStepGridValueChanged(app, event)
value = app.zStepGrid.Value;
if strcmp(value,'Off')
zGrid = false;
else
zGrid = true;
end
save('appvars.mat','zGrid','-append');
end


% Value changed function: RepeatGrid
function RepeatGridValueChanged(app, event)
value = app.zStepGrid.Value;
if strcmp(value,'Off')
repGrid = false;
else
repGrid = true;
end
save('appvars.mat','repGrid','-append');
end
end


% Component initialization
methods (Access = private)
```

```
% Create UIFigure and components
function createComponents(app)


% Create ControlAppUIFigure and hide until all components are created
app.ControlAppUIFigure = uifigure('Visible', 'off');
app.ControlAppUIFigure.Position = [100 100 451 467];
app.ControlAppUIFigure.Name = 'ControlApp';


% Create StartSortCropButton
app.StartSortCropButton = uibutton(app.ControlAppUIFigure, 'push');
app.StartSortCropButton.ButtonPushedFcn = createCallbackFcn(app,
@StartSortCropButtonPushed, true);
app.StartSortCropButton.Position = [18 44 100 22];
app.StartSortCropButton.Text = 'Start Sort/Crop';


% Create StartPatternButton
app.StartPatternButton = uibutton(app.ControlAppUIFigure, 'push');
app.StartPatternButton.ButtonPushedFcn = createCallbackFcn(app,
@StartPatternButtonPushed, true);
app.StartPatternButton.Position = [148 44 120 22];
app.StartPatternButton.Text = 'Start Pattern';


% Create CancelButton
app.CancelButton = uibutton(app.ControlAppUIFigure, 'push');
app.CancelButton.ButtonPushedFcn = createCallbackFcn(app, @CancelButtonPushed, true);
app.CancelButton.Position = [158 12 100 22];
app.CancelButton.Text = 'Cancel';


% Create LaserPowerLabel
app.LaserPowerLabel = uilabel(app.ControlAppUIFigure);
app.LaserPowerLabel.HorizontalAlignment = 'right';
app.LaserPowerLabel.Position = [69 212 87 22];
app.LaserPowerLabel.Text = 'Laser Power %';


% Create LaserPowerSlider
app.LaserPowerSlider = uislider(app.ControlAppUIFigure);
app.LaserPowerSlider.ValueChangedFcn = createCallbackFcn(app,
@LaserPowerSliderValueChanged, true);
app.LaserPowerSlider.Tooltip = {'Select laser power for patterning. '};
app.LaserPowerSlider.Position = [34 200 150 3];
app.LaserPowerSlider.Value = 100;


% Create RowSizepixLabel
app.RowSizepixLabel = uilabel(app.ControlAppUIFigure);
app.RowSizepixLabel.HorizontalAlignment = 'right';
```

```
app.RowSizepixLabel.Position = [235 292 80 22];
app.RowSizepixLabel.Text = 'Row Size (px)';


% Create RowSizeEditField
app.RowSizeEditField = uieditfield(app.ControlAppUIFigure, 'numeric');
app.RowSizeEditField.ValueChangedFcn = createCallbackFcn(app,
@RowSizeEditFieldValueChanged, true);
app.RowSizeEditField.Tooltip = {'Desired Cropped size, recommed +1 for rounding
error. '};
app.RowSizeEditField.Position = [330 292 100 22];
app.RowSizeEditField.Value = 513;


% Create ColumnSizepxEditFieldLabel
app.ColumnSizepxEditFieldLabel = uilabel(app.ControlAppUIFigure);
app.ColumnSizepxEditFieldLabel.HorizontalAlignment = 'right';
app.ColumnSizepxEditFieldLabel.Position = [227 258 97 22];
app.ColumnSizepxEditFieldLabel.Text = 'Column Size (px)';


% Create ColumnSizeEditField
app.ColumnSizeEditField = uieditfield(app.ControlAppUIFigure, 'numeric');
app.ColumnSizeEditField.ValueChangedFcn = createCallbackFcn(app,
@ColumnSizeEditFieldValueChanged, true);
app.ColumnSizeEditField.Tooltip = {'Desired Cropped size, recommed +1 for rounding
error. '};
app.ColumnSizeEditField.Position = [330 258 100 22];
app.ColumnSizeEditField.Value = 513;


% Create ofPicturesLabel
app.ofPicturesLabel = uilabel(app.ControlAppUIFigure);
app.ofPicturesLabel.HorizontalAlignment = 'right';
app.ofPicturesLabel.Position = [243 224 72 22];
app.ofPicturesLabel.Text = '# of Pictures';


% Create DesiredPicturesEditField
app.DesiredPicturesEditField = uieditfield(app.ControlAppUIFigure, 'numeric');
app.DesiredPicturesEditField.ValueChangedFcn = createCallbackFcn(app,
@DesiredPicturesEditFieldValueChanged, true);
app.DesiredPicturesEditField.Tooltip = {'Total number of pictures being cropped or
total desired number of pictures. '};
app.DesiredPicturesEditField.Position = [330 224 100 22];
app.DesiredPicturesEditField.Value = 10;


% Create LayerHeightEditFieldLabel
app.LayerHeightEditFieldLabel = uilabel(app.ControlAppUIFigure);
app.LayerHeightEditFieldLabel.HorizontalAlignment = 'right';
app.LayerHeightEditFieldLabel.Position = [242 190 73 22];
```

```
app.LayerHeightEditFieldLabel.Text = 'Layer Height';


% Create LayerHeightEditField
app.LayerHeightEditField = uieditfield(app.ControlAppUIFigure, 'numeric');
app.LayerHeightEditField.Limits = [1 Inf];
app.LayerHeightEditField.ValueChangedFcn = createCallbackFcn(app,
@LayerHeightEditFieldValueChanged, true);
app.LayerHeightEditField.Tooltip = {'Max is 450 when using fastZ.'};
app.LayerHeightEditField.Position = [330 190 100 22];
app.LayerHeightEditField.Value = 450;


% Create NumberofColumnsEditFieldLabel
app.NumberofColumnsEditFieldLabel = uilabel(app.ControlAppUIFigure);
app.NumberofColumnsEditFieldLabel.HorizontalAlignment = 'right';
app.NumberofColumnsEditFieldLabel.Position = [203 429 112 22];
app.NumberofColumnsEditFieldLabel.Text = 'Number of Columns';


% Create NumberofColumnsEditField
app.NumberofColumnsEditField = uieditfield(app.ControlAppUIFigure, 'numeric');
app.NumberofColumnsEditField.ValueChangedFcn = createCallbackFcn(app,
@NumberofColumnsEditFieldValueChanged, true);
app.NumberofColumnsEditField.Position = [330 429 100 22];
app.NumberofColumnsEditField.Value = 1;


% Create NumberofRowsEditFieldLabel
app.NumberofRowsEditFieldLabel = uilabel(app.ControlAppUIFigure);
app.NumberofRowsEditFieldLabel.HorizontalAlignment = 'right';
app.NumberofRowsEditFieldLabel.Position = [220 394 95 22];
app.NumberofRowsEditFieldLabel.Text = 'Number of Rows';


% Create NumberofRowsEditField
app.NumberofRowsEditField = uieditfield(app.ControlAppUIFigure, 'numeric');
app.NumberofRowsEditField.ValueChangedFcn = createCallbackFcn(app,
@NumberofRowsEditFieldValueChanged, true);
app.NumberofRowsEditField.Position = [330 394 100 22];
app.NumberofRowsEditField.Value = 1;


% Create NumberofLayersEditFieldLabel
app.NumberofLayersEditFieldLabel = uilabel(app.ControlAppUIFigure);
app.NumberofLayersEditFieldLabel.HorizontalAlignment = 'right';
app.NumberofLayersEditFieldLabel.Position = [214 360 101 22];
app.NumberofLayersEditFieldLabel.Text = {'Number of Layers'; ''};


% Create NumberofLayersEditField
app.NumberofLayersEditField = uieditfield(app.ControlAppUIFigure, 'numeric');
```

```
app.NumberofLayersEditField.ValueChangedFcn = createCallbackFcn(app,
@NumberofLayersEditFieldValueChanged, true);
app.NumberofLayersEditField.Position = [330 360 100 22];
app.NumberofLayersEditField.Value = 1;


% Create GridSpacingumEditFieldLabel
app.GridSpacingumEditFieldLabel = uilabel(app.ControlAppUIFigure);
app.GridSpacingumEditFieldLabel.HorizontalAlignment = 'right';
app.GridSpacingumEditFieldLabel.Position = [212 326 103 22];
app.GridSpacingumEditFieldLabel.Text = 'Grid Spacing (um)';


% Create GridSpacingumEditField
app.GridSpacingumEditField = uieditfield(app.ControlAppUIFigure, 'numeric');
app.GridSpacingumEditField.ValueChangedFcn = createCallbackFcn(app,
@GridSpacingumEditFieldValueChanged, true);
app.GridSpacingumEditField.Tooltip = {'360 for res x2'};
app.GridSpacingumEditField.Position = [330 326 100 22];
app.GridSpacingumEditField.Value = 360;


% Create zStepumEditFieldLabel
app.zStepumEditFieldLabel = uilabel(app.ControlAppUIFigure);
app.zStepumEditFieldLabel.HorizontalAlignment = 'right';
app.zStepumEditFieldLabel.Position = [212 124 64 22];
app.zStepumEditFieldLabel.Text = 'zStep (um)';


% Create zStepumEditField
app.zStepumEditField = uieditfield(app.ControlAppUIFigure, 'numeric');
app.zStepumEditField.ValueChangedFcn = createCallbackFcn(app,
@zStepumEditFieldValueChanged, true);
app.zStepumEditField.Tooltip = {'Required for calculations, must also change the
widget''s value. '};
app.zStepumEditField.Position = [400 124 30 22];
app.zStepumEditField.Value = 1;


% Create InvertBackgroundSwitchLabel
app.InvertBackgroundSwitchLabel = uilabel(app.ControlAppUIFigure);
app.InvertBackgroundSwitchLabel.HorizontalAlignment = 'center';
app.InvertBackgroundSwitchLabel.Position = [52 416 112 22];
app.InvertBackgroundSwitchLabel.Text = 'Invert Background?';


% Create InvertBackgroundSwitch
app.InvertBackgroundSwitch = uiswitch(app.ControlAppUIFigure, 'slider');
app.InvertBackgroundSwitch.Items = {'Yes', 'No'};
app.InvertBackgroundSwitch.ValueChangedFcn = createCallbackFcn(app,
@InvertBackgroundSwitchValueChanged, true);
```

```
app.InvertBackgroundSwitch.Tooltip = {'Yes, inverte the image. No, leaves it the
same. '};
app.InvertBackgroundSwitch.Position = [84 440 45 20];
app.InvertBackgroundSwitch.Value = 'No';


% Create RepeatSameStackSwitchLabel
app.RepeatSameStackSwitchLabel = uilabel(app.ControlAppUIFigure);
app.RepeatSameStackSwitchLabel.HorizontalAlignment = 'center';
app.RepeatSameStackSwitchLabel.Position = [48 360 119 22];
app.RepeatSameStackSwitchLabel.Text = 'Repeat Same Stack?';


% Create RepeatSameStackSwitch
app.RepeatSameStackSwitch = uiswitch(app.ControlAppUIFigure, 'slider');
app.RepeatSameStackSwitch.Items = {'Yes', 'No'};
app.RepeatSameStackSwitch.ValueChangedFcn = createCallbackFcn(app,
@RepeatSameStackSwitchValueChanged, true);
app.RepeatSameStackSwitch.Tooltip = {'Repeat the same image stack for each grid
location.'};
app.RepeatSameStackSwitch.Position = [85 383 45 20];
app.RepeatSameStackSwitch.Value = 'No';


% Create ImageTypeSwitchLabel
app.ImageTypeSwitchLabel = uilabel(app.ControlAppUIFigure);
app.ImageTypeSwitchLabel.HorizontalAlignment = 'center';
app.ImageTypeSwitchLabel.Position = [81 304 68 22];
app.ImageTypeSwitchLabel.Text = 'Image Type';


% Create ImageTypeSwitch
app.ImageTypeSwitch = uiswitch(app.ControlAppUIFigure, 'slider');
app.ImageTypeSwitch.Items = {'.bmp', '.tif'};
app.ImageTypeSwitch.ValueChangedFcn = createCallbackFcn(app,
@ImageTypeSwitchValueChanged, true);
app.ImageTypeSwitch.Tooltip = {'Select the image stack file type.'};
app.ImageTypeSwitch.Position = [92 327 45 20];
app.ImageTypeSwitch.Value = '.bmp';


% Create ResScanningSwitchLabel
app.ResScanningSwitchLabel = uilabel(app.ControlAppUIFigure);
app.ResScanningSwitchLabel.HorizontalAlignment = 'center';
app.ResScanningSwitchLabel.Position = [65 247 90 22];
app.ResScanningSwitchLabel.Text = 'Res. Scanning?';


% Create ResScanningSwitch
app.ResScanningSwitch = uiswitch(app.ControlAppUIFigure, 'slider');
app.ResScanningSwitch.Items = {'Yes', 'No'};
```

```
app.ResScanningSwitch.ValueChangedFcn = createCallbackFcn(app,
@ResScanningSwitchValueChanged, true);
app.ResScanningSwitch.Tooltip = {'Res Scannign switch will flip images to make them
appear in the correct direction when patterning.'};
app.ResScanningSwitch.Position = [87 271 45 20];
app.ResScanningSwitch.Value = 'Yes';


% Create STOPPATTERNButton
app.STOPPATTERNButton = uibutton(app.ControlAppUIFigure, 'push');
app.STOPPATTERNButton.ButtonPushedFcn = createCallbackFcn(app,
@STOPPATTERNButtonPushed, true);
app.STOPPATTERNButton.Position = [298 12 132 54];
app.STOPPATTERNButton.Text = 'STOP PATTERN';


% Create LoadPreCropFileButton
app.LoadPreCropFileButton = uibutton(app.ControlAppUIFigure, 'push');
app.LoadPreCropFileButton.ButtonPushedFcn = createCallbackFcn(app,
@LoadPreCropFileButtonPushed, true);
app.LoadPreCropFileButton.Position = [11 12 113 22];
app.LoadPreCropFileButton.Text = 'Load PreCrop File';


% Create PowerGridSwitchLabel
app.PowerGridSwitchLabel = uilabel(app.ControlAppUIFigure);
app.PowerGridSwitchLabel.HorizontalAlignment = 'center';
app.PowerGridSwitchLabel.Position = [80 85 66 22];
app.PowerGridSwitchLabel.Text = 'Power Grid';


% Create PowerGridSwitch
app.PowerGridSwitch = uiswitch(app.ControlAppUIFigure, 'slider');
app.PowerGridSwitch.ValueChangedFcn = createCallbackFcn(app,
@PowerGridSwitchValueChanged, true);
app.PowerGridSwitch.Tooltip = {'Power grid will vary the power for each grid loaction
evenly. '};
app.PowerGridSwitch.Position = [88 112 45 20];


% Create RepeatsumEditFieldLabel
app.RepeatsumEditFieldLabel = uilabel(app.ControlAppUIFigure);
app.RepeatsumEditFieldLabel.HorizontalAlignment = 'right';
app.RepeatsumEditFieldLabel.Position = [203 90 78 22];
app.RepeatsumEditFieldLabel.Text = 'Repeats (um)';


% Create RepeatsumEditField
app.RepeatsumEditField = uieditfield(app.ControlAppUIFigure, 'numeric');
app.RepeatsumEditField.ValueChangedFcn = createCallbackFcn(app,
@RepeatsumEditFieldValueChanged, true);
```

```
app.RepeatsumEditField.Tooltip = {'Required for calculations, must also change the
widget''s value. '};
app.RepeatsumEditField.Position = [399 90 31 22];
app.RepeatsumEditField.Value = 1;


% Create zStepGrid
app.zStepGrid = uiswitch(app.ControlAppUIFigure, 'slider');
app.zStepGrid.ValueChangedFcn = createCallbackFcn(app, @zStepGridValueChanged, true);
app.zStepGrid.Tooltip = {'Evenly vary the zStep for each grid loaciton.'};
app.zStepGrid.Position = [314 125 45 20];


% Create RepeatGrid
app.RepeatGrid = uiswitch(app.ControlAppUIFigure, 'slider');
app.RepeatGrid.ValueChangedFcn = createCallbackFcn(app, @RepeatGridValueChanged,
true);
app.RepeatGrid.Tooltip = {'Evenly vary the number of repeats for each grid
loaciton.'};
app.RepeatGrid.Position = [315 91 45 20];


% Create VaryValuesLabel
app.VaryValuesLabel = uilabel(app.ControlAppUIFigure);
app.VaryValuesLabel.Position = [302 149 68 22];
app.VaryValuesLabel.Text = 'Vary Values';


% Create DesiredMaxLabel
app.DesiredMaxLabel = uilabel(app.ControlAppUIFigure);
app.DesiredMaxLabel.Position = [379 149 73 22];
app.DesiredMaxLabel.Text = 'Desired/Max';


% Show the figure after all components are created
app.ControlAppUIFigure.Visible = 'on';
end
end


% App creation and deletion
methods (Access = public)


% Construct app
function app = TileControls


% Create UIFigure and components
createComponents(app)
```

```
% Register the app with App Designer
registerApp(app, app.ControlAppUIFigure)


% Execute the startup function
runStartupFcn(app, @startupFcn)


if nargout == 0
clear app
end
end


% Code that executes before app deletion
function delete(app)


% Delete UIFigure when app is deleted
delete(app.ControlAppUIFigure)
end
end
end
```