BRISKit User Guide

1. INTRODUCTION

The Boundary and Region Identification Software Kit (BRISKit) is a MATLAB program that was developed to create accurate virtual human models from cross-sectional images. BRISKit can work with either color or grayscale cross-sectional images and includes several tools to aid the user in segmenting each image into different tissue regions. Such virtual human models are used in many applications, e.g., they can be used to investigate the interaction of electromagnetic fields with the human body. BRISKIT was initially developed using the Visible Human Project's datasets, but could easily be modified to work with other datasets that are made up of cross-sectional images.

2. INSTALLATION

BRISKit is a MATLAB program with a graphical user interface that allows the user to identify regions in a cross-sectional image (a 'slice'), assign tissue names to these regions, review the identified regions, and make any adjustments as needed to correct these regions and their boundaries. The following instructions configure the program to be used with the AustinMan and AustinWoman datasets. Alternative datasets can be implemented by changing the folder names.

Download the program from the website http://bit.ly/AustinMan into a main program folder in your computer, e.g., 'C:\BRISKit'. Under this program folder, create three folders called 'AustinMan', 'bodyMale', and 'bodyFemale'. The cross-sectional images (the input for BRISKit) should be stored in PNG files under the 'bodyMale' and 'bodyFemale' folders for the male and female datasets, respectively. Under each of the 'bodyMale' and 'bodyFemale' folders, create a folder called 'Masks' to store black-and-white images (see Section 3). Create three subfolders under the 'AustinMan' folder: 'BRISKit', 'SegmentsMale', and 'SegmentsFemale'. The segmented male and female slices and their corresponding '.mat' files (the output for BRISKit) will be stored under the 'SegmentsMale' and 'SegmentsFemale' folders, respectively. The 'BRISKit' folder contains the '.m' files for all the functions that are used in segmenting the slices. Once the folders are created, run MATLAB and change the current folder to 'C:\BRISKit'. The program is in the 'briskit.m' file. Execute it by typing 'briskit' in the command window and pressing enter.

3. MASKS

Masks are black-and-white images that are used to filter slices; the pixels in a mask are set to one (white) for the desired regions in the slice and zero (black) otherwise. The desired regions in a slice can be isolated by computing an 'and' operation with its mask; as a result, the pixels in the slice that correspond to a zero in the mask are blacked out while the remaining pixels are unmodified and appear on a black background. For AustinMan and AustinWoman datasets, masks were created by cropping slices, converting them to color space, filtering them, and manually adjusting the results [1]. For a more detailed explanation of how masks are created and how they are used, visit http://bit.ly/AustinMan. The masks are required as input (similar to cross-sectional images) in BRISKit and must be stored under the 'Masks' folder. The user can create new masks or can download masks for AustinMan and AustinWoman datasets from the website http://bit.ly/AustinMan.

4. STARTING A NEW SLICE

Typically, the regions identified in a previously completed slice is used as an initial guess when starting a new slice in BRISKit, i.e., the identified regions from a previously completed slice is copied onto the new slice. The program will ask four pop-up questions to determine which slice the user wants to work on and which slice will be used as the initial guess.

(i) Select whether you will work with the *Female* or *Male* slices.

(ii) Select the slice number. Typically, this is the number of the slice you will use as the initial guess when working on another slice for the first time. In this case, you should click 'No' in step (iii). Alternatively, this is the number of the slice you will actually be working on, e.g., if you want to edit a slice that you worked on previously. In this case you should click 'Yes' in step (iii). For the male slices, which have 1-mm resolution in the *z* direction, the slice number is a 4-digit number. For the female slices, which have 1/3-mm resolution in the *z* direction, add the letter 'b' at the end of the 4-digit number (each 4-digit number in the AustinWoman dataset has three slices associated with it, e.g., 1001a, 1001b, and 1001c).

(iii) You will be prompted with the question 'Use the image file corresponding to the selected slice?' Click *No* if you want to use the slice entered in step (ii) as the initial guess. Click *Yes* to edit the regions in the slice entered in step (ii).

(iv) If you clicked *No* in step (iii), enter the number of the slice that you want to work on (again, append the letter 'b' to the number for the female slices). This slice is referred to as the 'target slice' in the following.

Usually, the number entered in step (iv), the slice number, is one more than the number entered in step (ii), the segment number. Exceptions include processing the data by going upwards through the images in the dataset rather than downwards, and skipping slices, when the number entered in step (iv) should be chosen accordingly. If you would like to edit a slice with no initial guess, then first create a '.mat' file for that slice, and start the slice as explained in the previous paragraph. The '.mat' file is a file storing a matrix that identifies the material IDs of the pixels. You can create such a file by first defining a matrix variable inside MATLAB and then saving that matrix from Workspace as a '.mat' file. For AustinMan and AustinWoman datasets, you should create a 0 matrix of size 1120x2048. When starting a new slice, it is strongly recommended to use an initial guess *and* to run the *Prep Script Male or Prep Script Female* functions to improve the initial guess. Once these steps are completed, the user can begin reviewing the slice by clicking the *Review* button at the bottom of the BRISKit home screen (Fig. 1).

5. HOME MODE

The program starts at the BRISKit home screen. The buttons and the relevant functions in this screen are explained next.

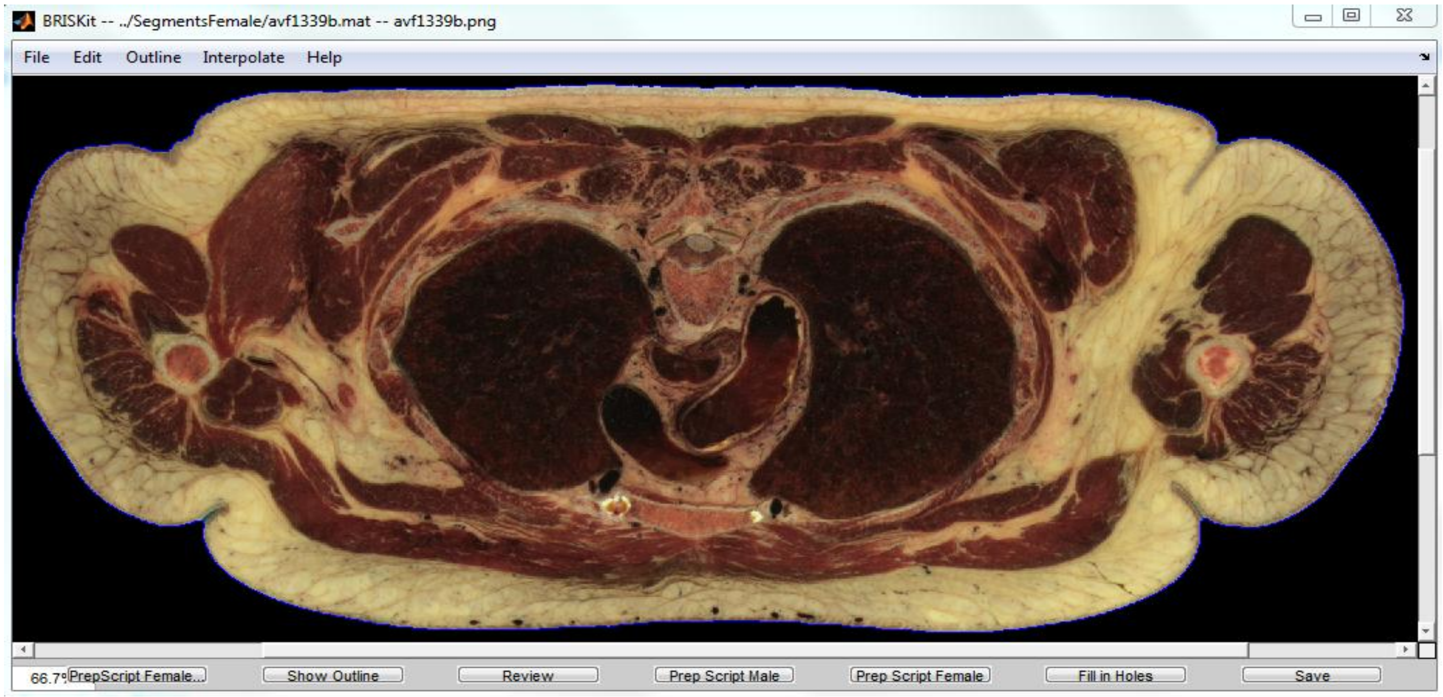

Fig. 1: BRISKit home screen.

5.A. Prep Script Male/Female

5.A.i. Objective

The main goal of this function is to automatically segment a slice as much as possible before the user begins to interactively improve it. The *Prep Script* functions leverage the already segmented slices by employing the segmentation of one of them as the initial guess and modifying this initial segmentation with the so called competition [1] approach. That is, the color of a pixel or a group of pixels is compared to the mean color values (in the L*a*b* space) of two or more regions and assigned the color of one of them. See Section 6.C for a more detailed description of the various competition functions available in BRISKit.

5.A.ii. Algorithm Description

Each of the *Prep Script* functions takes the segments in the slice indicated in step (ii) when opening BRISKit as an initial guess. Using these segments, it executes the *CompeteSelect Smooth* function on the target slice for each pair of tissues specified by the user. There is also a non-smooth version of this function and it is called 'competeSelect'. The difference between these two functions is the same as the difference between the functions 'localCompete' (6.C) and 'localCompeteSmooth' (6.D). The tissue pairs are specified by entering their material IDs as arguments to the *CompeteSelect Smooth* function and are evaluated ('competed') sequentially in the *Prep Script* algorithm. Thus, strictly speaking, the order in which the pairs are listed can affect the outcome; in practice, this effect is minimal and it is safe to order them randomly. For the male dataset, which has a resolution of $1/3\times1/3\times1$ mm^3, *Prep Script Male* function should be used: It only runs with the previous slice and propagates it down 1 mm. For the female dataset, which has a resolution of $1/3\times1/3\times1/3$ mm^3, the *Prep Script Female* function should be used: It loops over three slices in the order 'c', 'a', and 'b', runs the *CompeteSelect Smooth* function three times (once for each slice), and uses the segmentation of each intermediate slice as the input for the next one.

5.A.iii. Implementation

This program should be run every time a new slice is started: Click on the *Prep Script Male* or *Female* button and wait. There are two steps that must be performed beforehand for this script to work in the most effective way possible: (i) Identify the tissue pairs that will be competed against each other. After identifying the tissue pairs, edit the 'Methods.m' file to call the 'competeSelectSmooth.m' function with the desired material IDs entered as its arguments. This file is in the same directory as the other m-files such as 'briskit.m'. Choose tissue pairs such that the two materials have distinct color values. It is not recommended to use this script for tissue pairs that have similar colors, such as fat and tendon. Moreover, this script does not give good results when applied to tissues that have a wide range of colors or more than one distinct color in themselves; e.g., bone marrow, which contains both yellowish and pink colors. This is because it uses a single color value to represent tissues throughout the whole slice when evaluating the pixels. (ii) Determine the number of iterations that the function should execute for every tissue pair and enter it as an argument to the 'competeSelectSmooth' function inside the 'Methods.m' file. Previous experience indicates that this function works best when the number of iterations is around 3. It is not recommended to use more than 5 iterations since that would increase the amount of wait time significantly. Using 1 or 2 iterations may also give satisfactory results.

5.A.iv. CompeteSelect Smooth

The *CompeteSelect Smooth* function is called inside the *Prep Script* functions. It uses an iterative algorithm; at each iteration, only the pixels along the boundary of a region are allowed to change their material IDs. This implies that the boundary can move one pixel per iteration. Ideally, the boundary approaches closer to the best boundary at each iteration. The function is quite similar to the *Local Compete Smooth* function. The main difference is that the color value assigned to each tissue for competition is the average color value of the whole tissue in the entire slice instead of the color value obtained from a localized user-specified region. Therefore, this function is global whereas the *Local Compete*

Smooth function is local. This function is not recommended to be used outside of the *Prep Script* functions and is not available to the user from the BRISKit home screen.

5.A.v. BoundaryAdjust

The *BoundaryAdjust* function is called inside the *Prep Script* functions and the *adjustSize* function. It uses the mask of the target slice to adjust (expand or shrink) the boundary of the initial guess, which can be too small or too large compared to the target slice. The method is implemented as follows: First, the initial guess is used to find “boundary pixels”; these are the pixels that bound the pixels identified by the initial guess as having a nonzero material ID (they are found by using the MATLAB function “bwperim”). Second, the 4 neighboring pixels that are above, below, to the left, and to the right of each boundary pixel are identified. Third, all neighboring pixels of each boundary pixel are evaluated to determine whether they are inside or outside target slice’s mask: If the material ID of a neighboring pixel is 0 while its mask value is 1, then the neighboring pixel’s material ID is replaced with that of the boundary pixel. Then, the boundary pixels are updated and the above steps are repeated until the boundary pixels do not change. Finally, the matrix storing the material IDs is multiplied element-wise by the matrix storing the mask values such that the pixels that have 0 mask values are cut out (assigned 0 as material ID), thereby shrinking the initial guess wherever it is larger than the mask.

5.B. Prep Script Female Reverse

This is the same script as *Prep Script Female*, except that it allows the user to go in the opposite direction after completing a slice, i.e., the slices are processed upward (or backward). This function should be used instead of *Prep Script Female* if the user segments the slices starting from bottom and going upward, e.g., if the user wants to start from the heart and go up to the brain.

5.C. Show Outline/Remove Outline

The *Show Outline/Remove Outline* button appears the bottom of BRISKit when it is in the Home mode (the default mode when BRISKit is run [Fig. 1]). *Show Outline* displays the green outline of the regions on top of the background image (Fig. 2). *Remove Outline* will remove the outlines from the image.

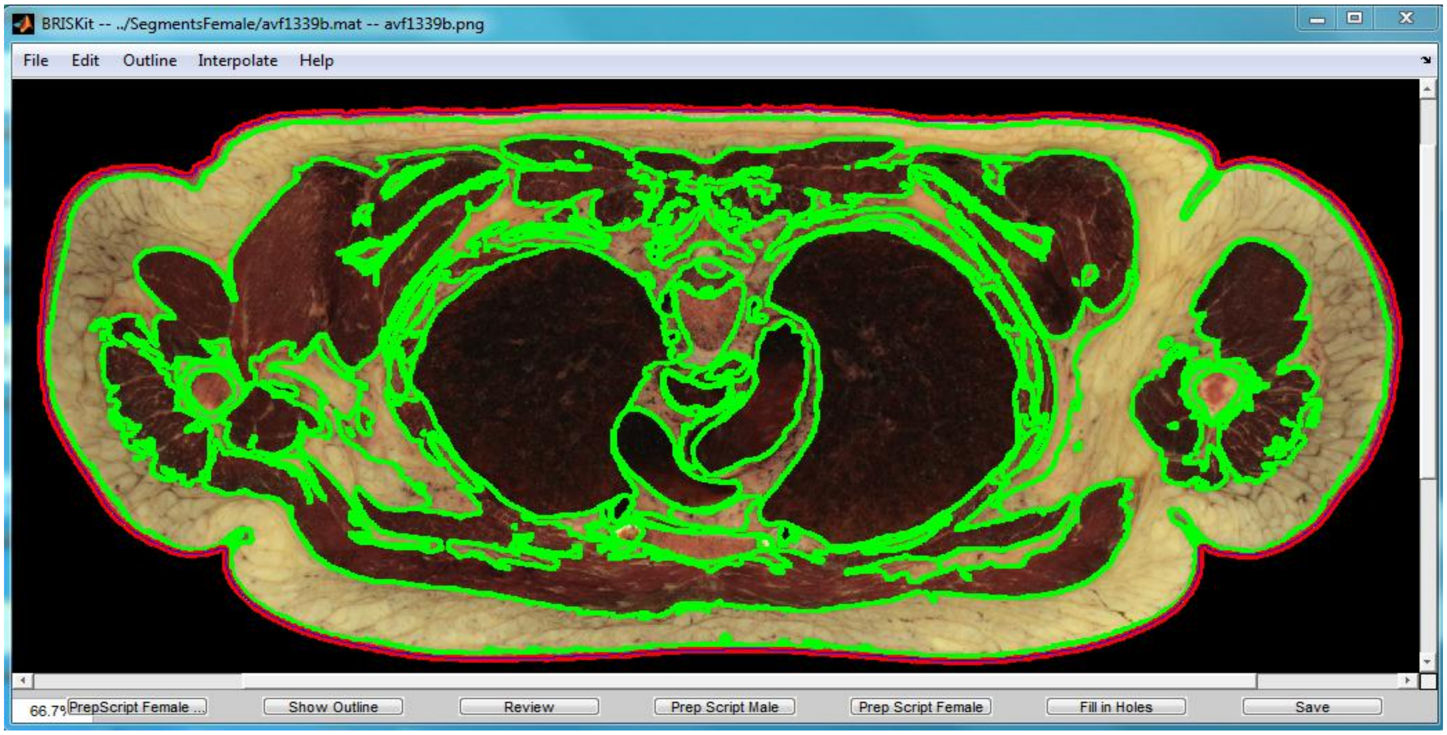

Fig. 2: The display after clicking the *Show Outline* button.

5.D. Save

Save will store the identified regions in hard disk. This will merge the newly identified regions with the previously saved regions, create one single matrix, and store it under the ‘SegmentsMale’ or ‘SegmentsFemale’ folder in the ‘.mat’ file that corresponds to the image number that was loaded.

5.E. Fill in Holes

Fill in Holes will fill in pixels that have not been assigned any material ID or that have a different material ID than the surrounding pixels. That is, if all 4 immediate neighbors of the pixel have the same material ID, its material ID is changed to that of the surrounding material. It is recommended to use this function in the final step when processing slices.

6. REVIEW MODE

The Review mode is used to review the tissues in the slice one at a time. This allows the user to ensure the accuracy of the boundaries and correct the regions as necessary. Typically, it is used after executing a *Prep Script* function. Enter the Review mode by selecting the *Review* button at the bottom of the home screen (Fig. 3). In this mode, the *Prev* and *Next* buttons are used to switch to the previous and the next material, respectively. The button between the *Prev* and *Next* buttons shows the material ID of the tissue that is currently being reviewed. Clicking on this button allows the user to select a material from a list to quickly jump to a new material. The *Undo* button is used to undo the last operation performed. The semi-automatic functions used for segmenting the slice are listed under the *Interpolate* menu. The ones that are relevant to the Review mode are *Local Compete* (6.C), *Local Compete Smooth* (6.D), *Local Compete Smooth Vertical/Horizontal* (6.E/6.F), *Select Shift* (6.G), and *Expand/Contract* (6.H).

When you are finished reviewing the slice, use ‘File > Save’. This command, just like the *Save* (5.D) button in the Home mode saves a matrix, which stores the material IDs of the pixels, in a ‘.mat’ file inside the ‘SegmentsMale’ or ‘SegmentsFemale’ folder. After the matrix is saved, you will be prompted to load a new image. Select *Yes* to start the load process for a new slice, which is the same as starting a new slice. Select *No* to keep working on the current image.

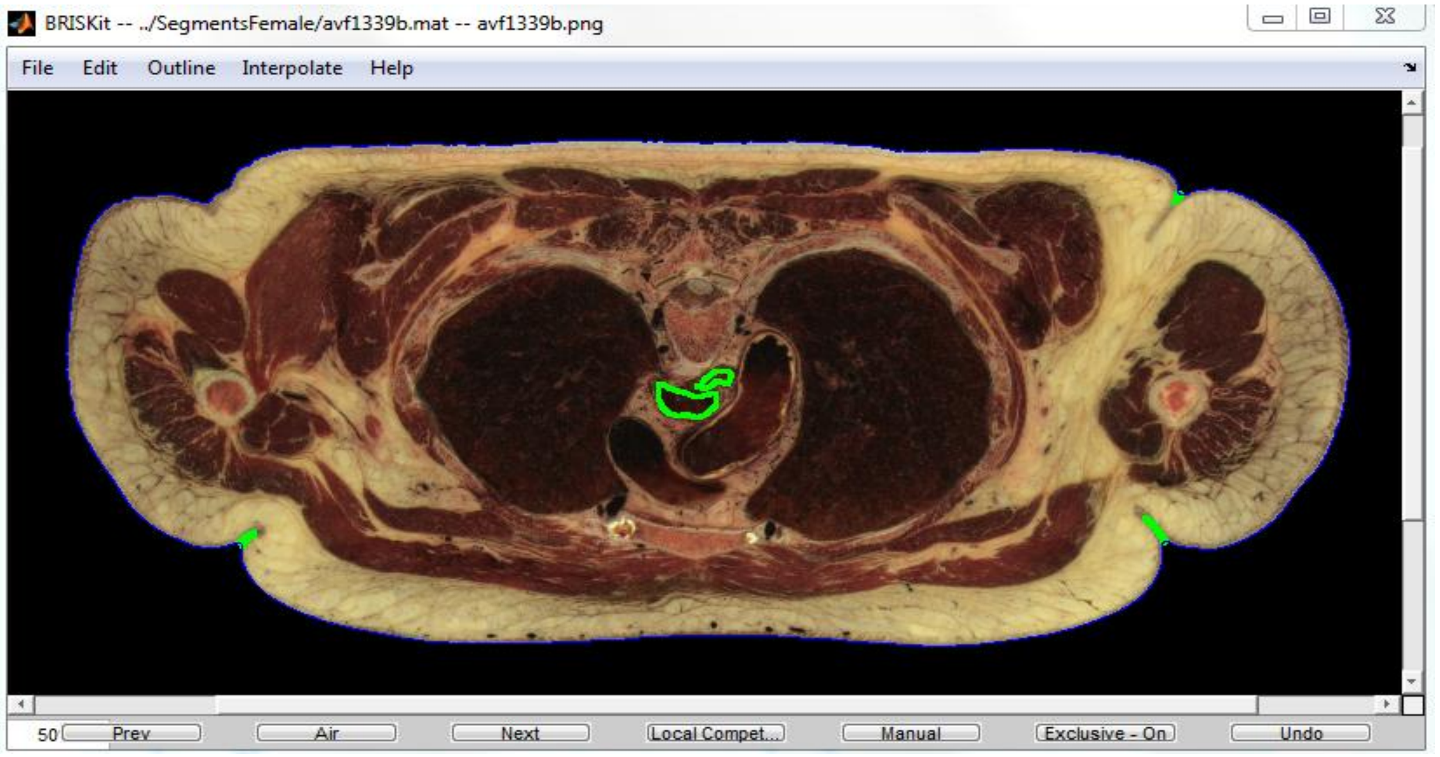

Fig.3: BRISKit review screen.

6.A. Manual

The *Manual* button lets the user to draw regions (Fig. 4) with the mouse and assign a material ID to the pixels in the selected regions. The region will automatically close when the user stops drawing (letting go of the left mouse button). When you complete drawing a region, double click inside of it to let MATLAB know that you are finished. You will then be prompted as to whether you want to continue drawing regions of the same type of material. Selecting *Yes* will let you draw additional regions, which allows you to label different regions with the same material ID without having to click *Manual* multiple times. Once you have finished drawing all of the regions for a material, select *No* from the pop-up window and you will be prompted for the material ID to be assigned to the pixels in these regions. Finally, the outline of the identified regions (including recent changes made using *Manual*) are displayed (Fig. 5).

6.B. Exclusive On/Off

The *Exclusive On/Off* button changes the behavior of the *Manual* function. The default state is *Exclusive – On*, which allows changes only to the pixels that have the same material ID as the tissue that is currently being worked on (indicated by the button between the *Prev* and *Next* buttons in Review mode). For example, if muscle is the current tissue, then drawing a region in fat and assigning the pixels in this region a tissue other than muscle will not change anything (since the pixels in the drawn region are not muscle pixels) but assigning them muscle will convert the pixel material IDs from fat to muscle. Conversely, if the pixels in the selected region are already in muscle, they can be converted to any other tissue by choosing the desired tissue from the pop-up window in the *Manual* function. The *Exclusive – On* state allows the user to cut away from the current tissue without having to be very precise after the cursor has left the current tissue's region. In the *Exclusive – Off* state, the pixels in the drawn region are assigned the selected tissue regardless of their current material IDs, e.g., both fat and muscle tissues in the above example can be assigned to a new tissue in this state. Thus, the user should be very careful drawing regions in the *Exclusive-off* state. The *Exclusive – Off* state may help save time by assigning to the drawn region a tissue that is different from the current tissue, provided that the part of the region that is outside the current tissue is drawn carefully enough.

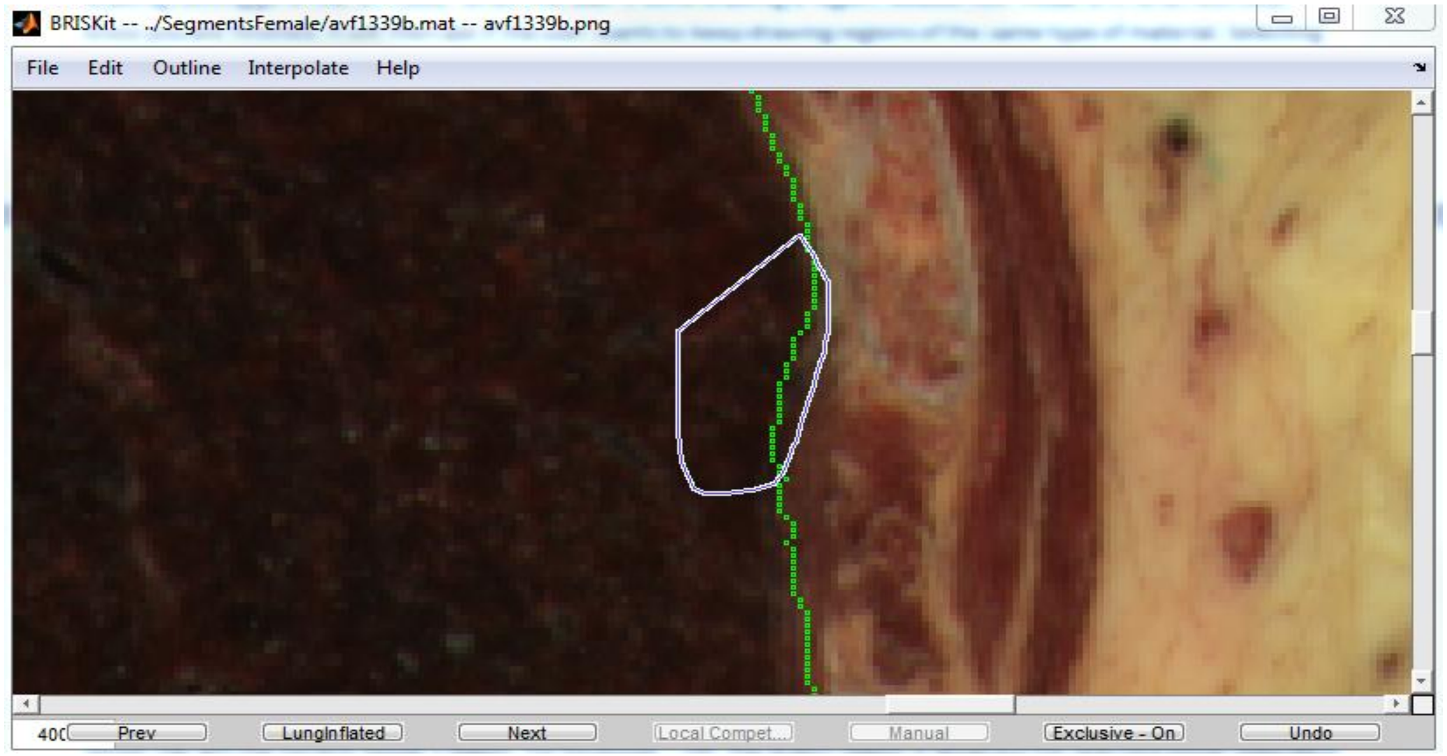

Fig. 4: Drawing regions using the *Manual* function.

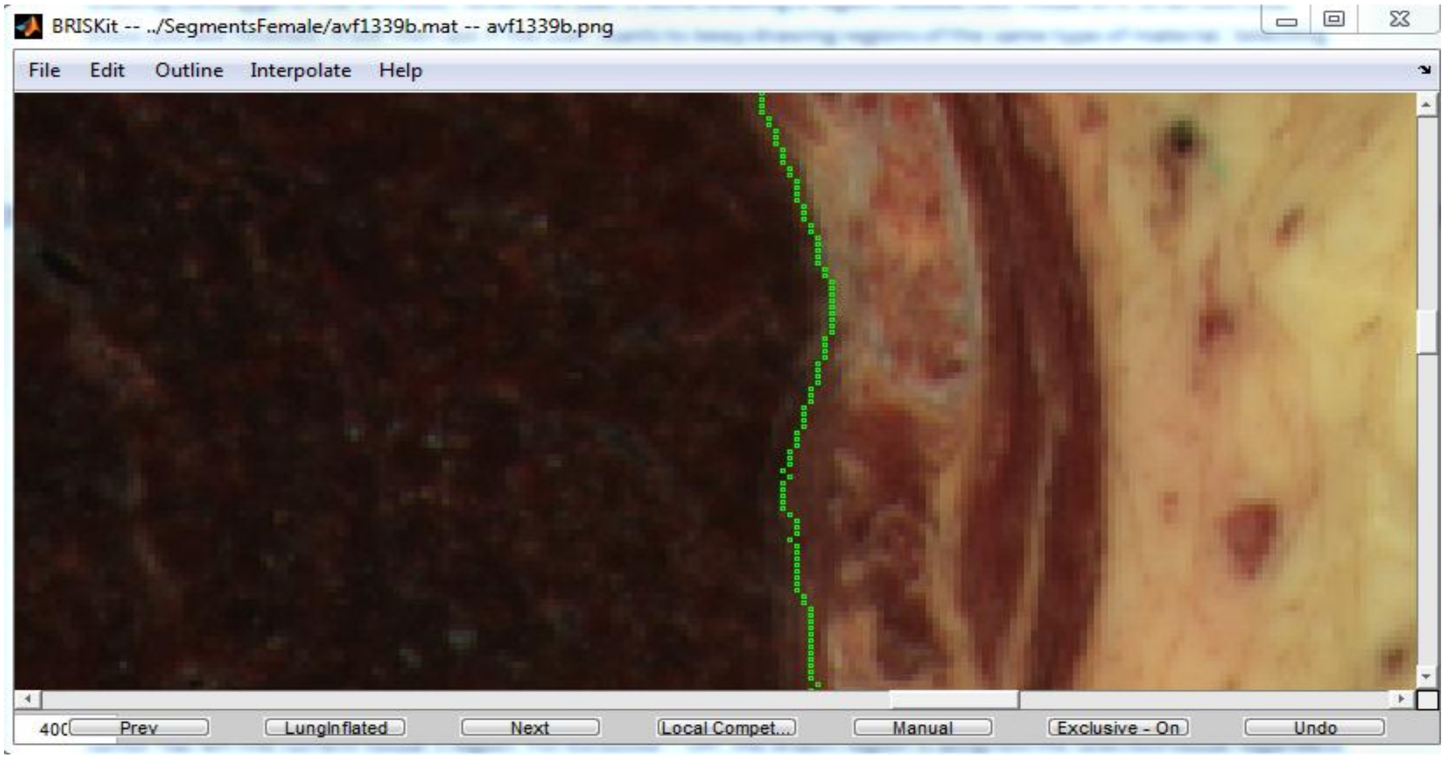

Fig. 5: Final display after the *Manual* function is executed.

6.C. Local Compete

6.C.i. Objective

The *Local Compete* function is used to semi-automatically correct a distorted boundary between two or more tissues. The method was created to give the user more control over the competition algorithm. It lets the user identify a region where the competition is executed, i.e., only the pixels in a local region are changed and the rest of the slice remains unchanged. Moreover, the user also selects the color(s) assigned to each region during the competition. Then, the competition is executed until convergence.

6.C.ii. Algorithm Description

Given a local region identified by the user, the rectangle that tightly encloses the competition region is constructed. Then, the boundaries of two (or more) tissues, which are identified by the user, are modified using the competition algorithm in this rectangle. The competition algorithm starts by identifying the pixels immediately outside the first tissue inside the rectangular region. Then, for each such pixel, it finds the L*a*b color value and calculates the Euclidean distance between this value and the color assigned to the different tissues. Comparing the distances, the pixel under consideration is assigned the material ID corresponding to the tissue with the shortest distance. This procedure is repeated for each of the tissues that were selected. The algorithm then loops back to the first tissue and continues iterating through all of the tissues until convergence is reached (when none of the pixels change tissues between two iterations). The algorithm allows multiple colors to be assigned to each tissue; in this case, the algorithm compares the pixels' color to each of these values during the competition process. Note that the local competition region may not be rectangular; thus, there will be pixels in the enclosing rectangular region that will not be changed. This is implemented of by 'locking' these pixels and storing the relevant information in a separate matrix. Finally, the boundaries are updated using the *getBoundaries* function so that the recently made changes appear when the regions are outlined. This function takes the coordinates of the updated boundaries and applies the changes made by outlining the new boundaries.

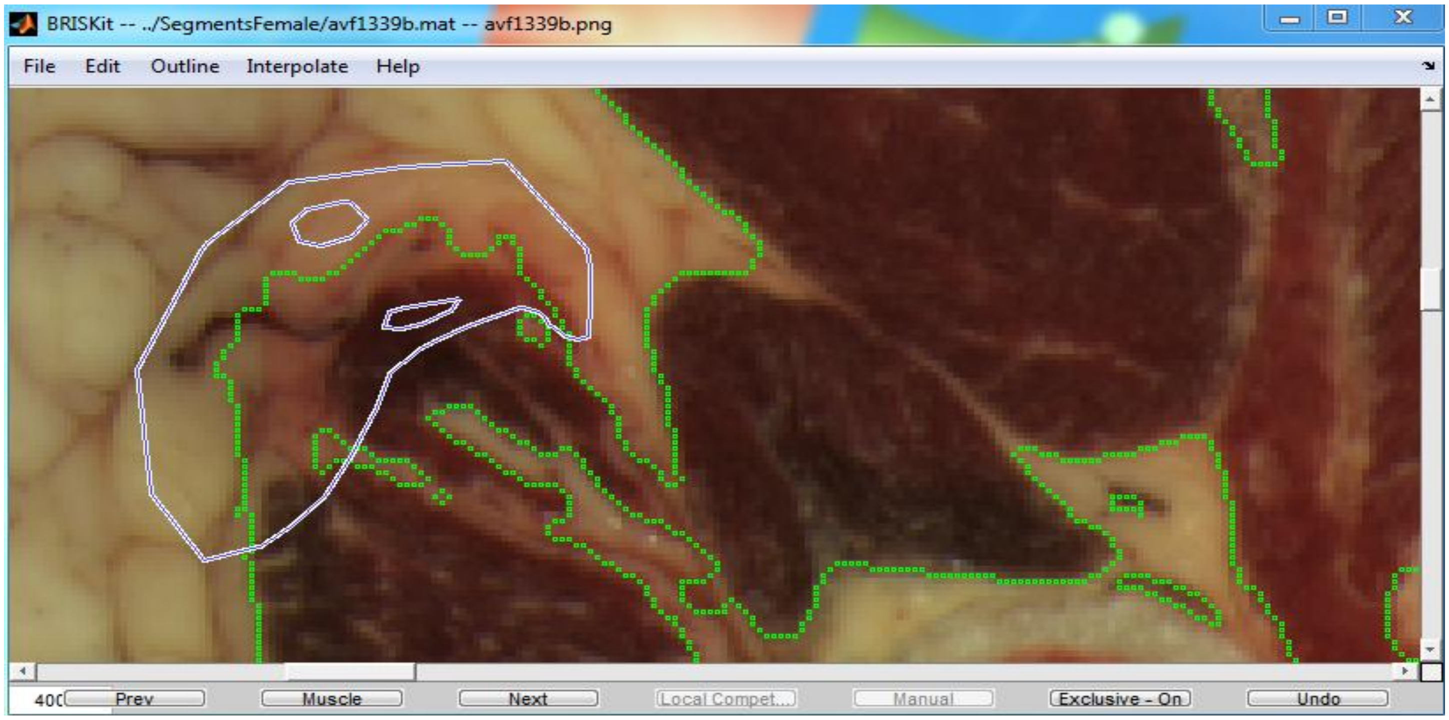

Fig. 6: Selecting regions using the *Local Compete* function.

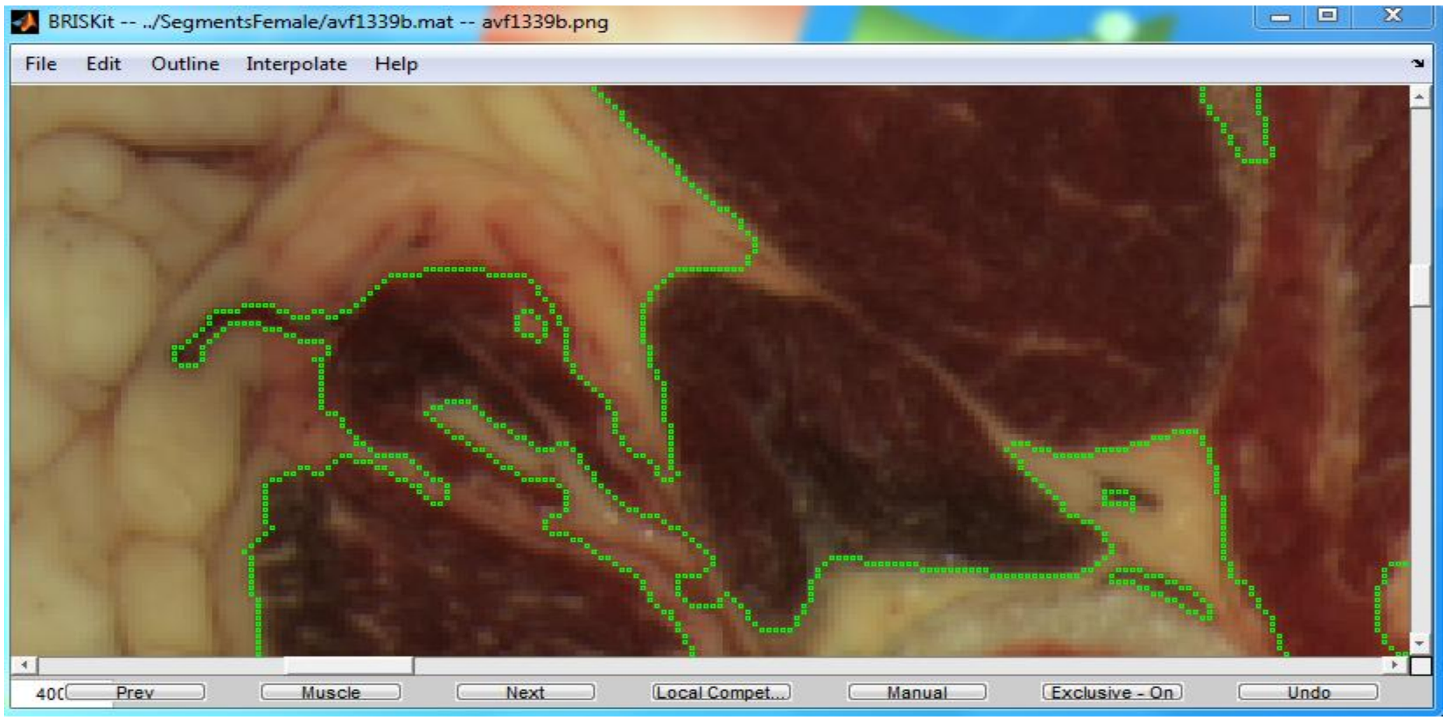

Fig. 7: Final display after the *Local Compete* function is executed.

6.C.iii. Implementation

First, select the region where the competition will be performed. Second, select two or more smaller regions, whose average colors (in the L*a*b* color space) will represent the color for each tissue during the competition algorithm. These smaller regions must be within the rectangle that tightly encloses the competition region. The smaller regions will also determine which tissues that will compete (by using the material ID of the first point for each region), i.e., pixels in tissues that are in the competition region but are not selected during the second step are not changed. Several regions can be selected for the same material if desired. After selecting regions (Fig. 6), click *No* in the pop-up window to start the competition and observe the results (Fig. 7).

6.C.iv. Suggested Use

This function is most effective for regions that mostly contain relatively smooth boundaries. It is also better if it is used on boundaries with a large color difference in the L*a*b* color space, such as the boundary between muscle and fat.

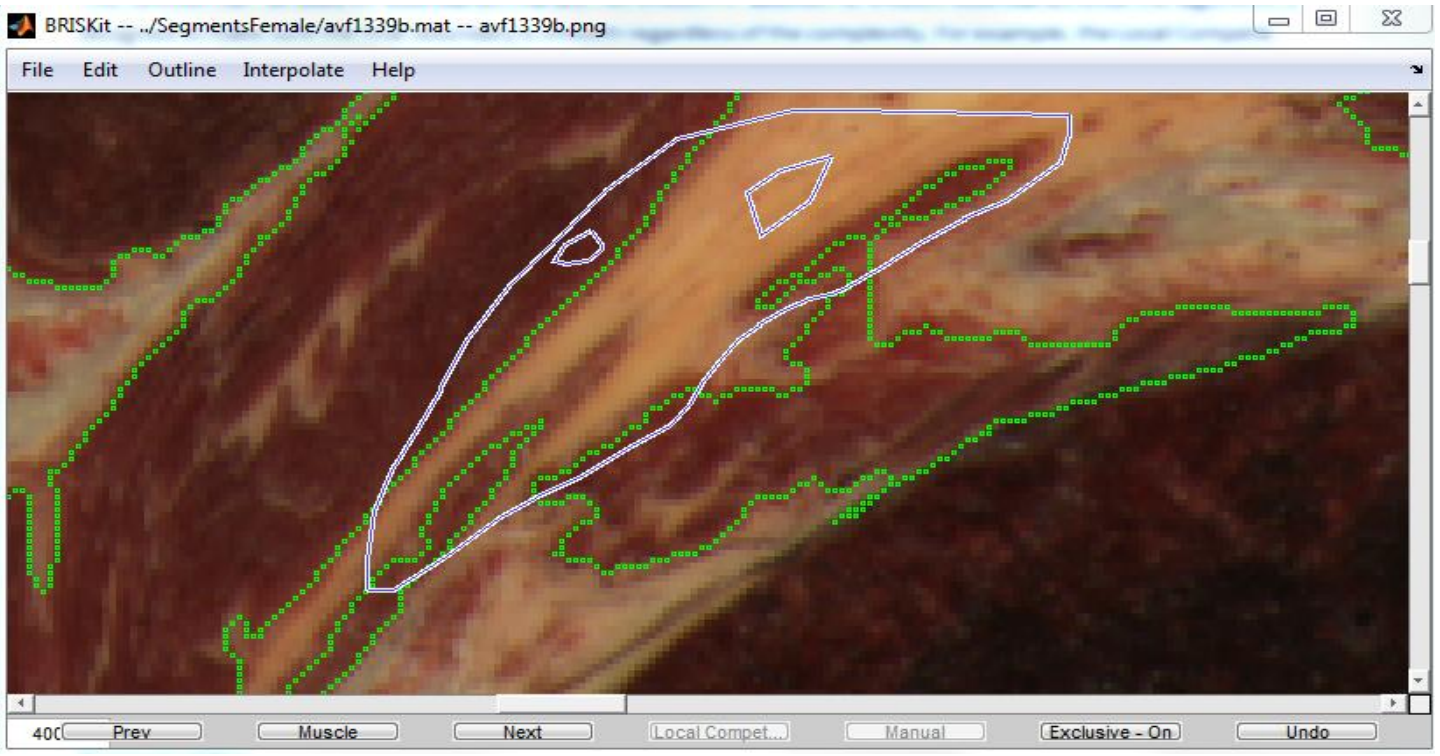

Fig. 8: Selecting regions using the *Local Compete Smooth* function.

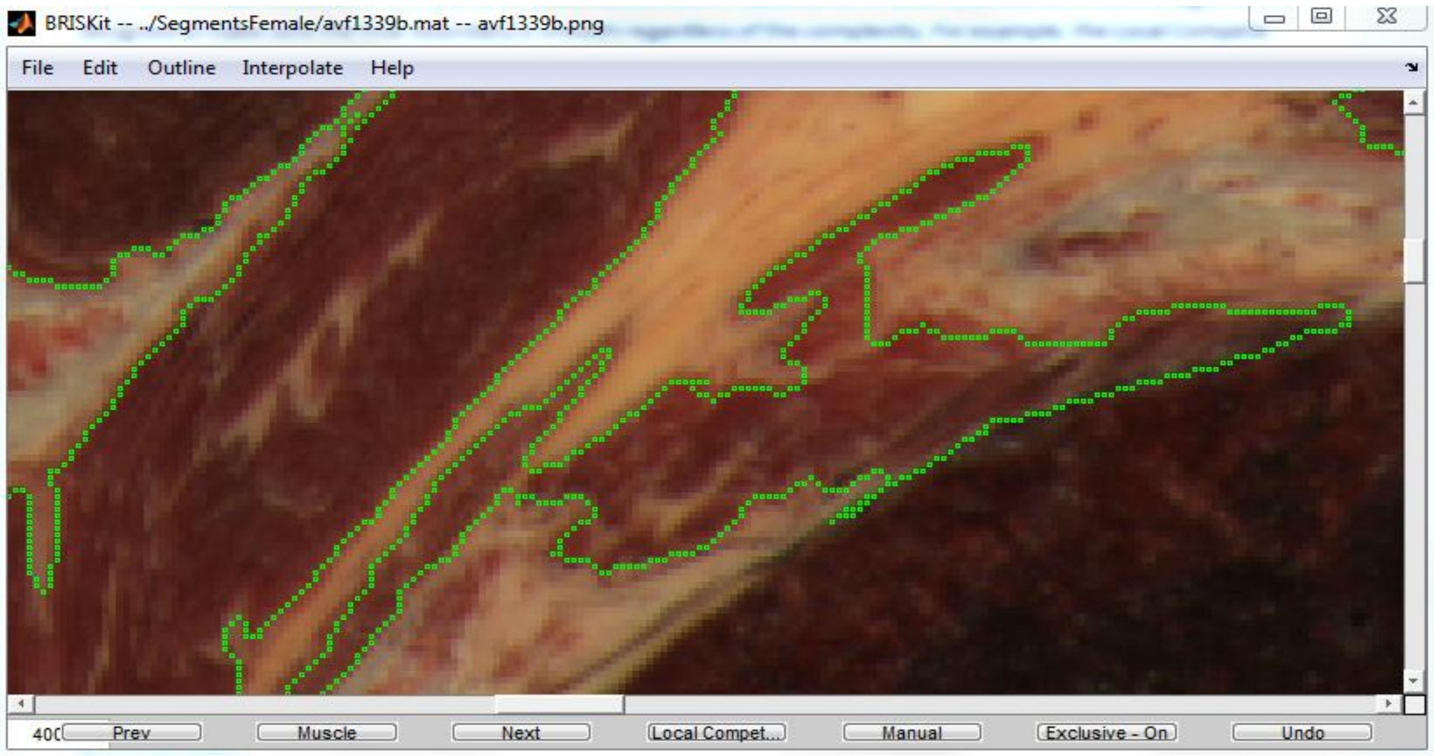

Fig. 9: Final display after the *Local Compete Smooth* function is executed.

6.D. Local Compete Smooth

6.D.i. Objective

While the *Local Compete* algorithm compares a pixel's color value to the color values assigned to the different tissues, the *Local Compete Smooth* algorithm uses the weighted color from a small region of pixels instead of just the pixel's color value. This results in smoother boundaries due to the averaging effect.

6.D.ii. Algorithm Description

The algorithm is similar to that of the *Local Compete* function. The main difference is that it uses a windowing approach to end up with smoother boundaries. Specifically, a 5×5 matrix called 'weights' is used, which assigns weights to pixels according to their proximity to the center pixel. The entries of this matrix are

$$\frac{1}{80}\times\begin{bmatrix}1 & 2 & 3 & 2 & 1\\ 2 & 4 & 6 & 4 & 2\\ 3 & 6 & 9 & 6 & 3\\ 2 & 4 & 6 & 4 & 2\\ 1 & 2 & 3 & 2 & 1\end{bmatrix}$$

When evaluating whether a pixel should be assigned a different color during the competition, the color used is the weighted color from the 25 pixels centered around the pixel of interest. The neighbors of the pixel of interest are still used even if they are outside the competition region as long as the pixel of interest is inside the competition region. Thus, the surrounding pixels will affect the tissue that each pixel ends up being assigned to.

6.D.iii. Implementation

The algorithm is implemented the same (Fig. 8) as the *Local Compete* function, except a weighting matrix is used.

6.D.iv. Suggested Use

This function is best used for regions in which the user would like to have smooth boundaries where some level of inaccuracy can be tolerated. Unlike the *Local Compete* function, this function does not result in jagged boundaries; thus, the boundary found is generally more smooth but can be less accurate compared to the *Local Compete* function. For example, the *Local Compete Smooth* function works well with the muscle-tendon boundary (Fig. 9).

6.E. Local Compete Smooth Vertical/Horizontal

6.E.i. Objective

The *Local Compete Smooth Vertical* and *Local Compete Smooth Horizontal* functions are designed to identify smooth boundaries for thin vertical and horizontal regions of tissues, respectively

6.E.ii. Algorithm Description

The algorithms are identical to the *Local Compete Smooth* algorithm except a 5×1 or a 1x5 weighting matrix is used instead of a 5×5 one. The entries of the 1x5 matrix are

$$\frac{1}{9}\times\begin{bmatrix}1 & 2 & 3 & 2 & 1\end{bmatrix}$$

and the 5×1 matrix is its transpose.

6.E.iii. Implementation

The implementations are identical to the *Local Compete Smooth* function except for the weighting matrix.

6.E.iv. Suggested Use

This functions work best when used on boundaries that are oriented vertically or horizontally.

6.F. Select Shift

6.F.i. Objective

The *Select Shift* function is used to move tissue regions around in a slice.

6.F.ii. Algorithm Description

First, the coordinates of the region of pixels selected by the user are found and stored in two vectors, a row-index vector and a column-index vector. The entries of these vectors are integer numbers that identify the location of each of the pixels in the selected region, i.e., a row index and a column index are used to identify a pixel. Then, an input is received from the keyboard; depending on the input, all of the entries of the row-index vector are decreased or increased by 1 or all of the entries of the column-index vector are decreased or increased by 1. Once the horizontal or vertical shifts are completed, the original set of pixels (which are stored at the beginning of the algorithm) are assigned a material ID determined by the user. Then, the pixels identified by the row- and column-index vectors are assigned the material ID of the current tissue. Finally, the boundaries are updated and displayed.

6.F.iii. Implementation

Select the region to be shifted, and shift it one row or column at a time using the arrow keys (Fig. 10). When the shift is complete, press ‘q’ and a pop-up window request a material that will be assigned to the pixels that are no longer assigned a material ID because of the shifting.

6.F.iv. Suggested Use

This function is best used when there is a tissue that shifts and does not change its shape very much between the slices. An example would be some of the bones (Fig. 11) or bone-like structures constituting the rib cage. If the tissue has drastic changes in its shape, other algorithms should be used to correctly segment the region.

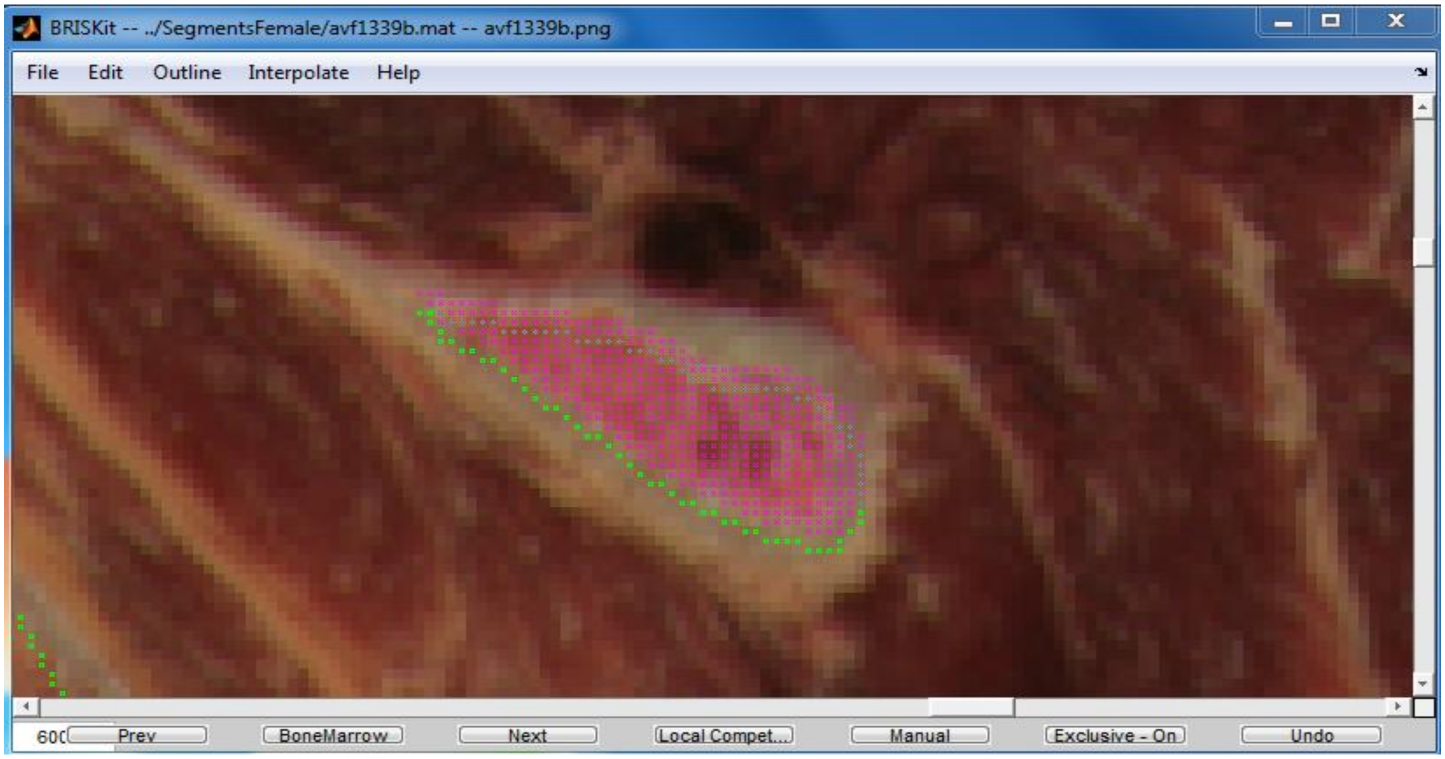

Fig. 10: Moving regions using the *Select Shift* function.

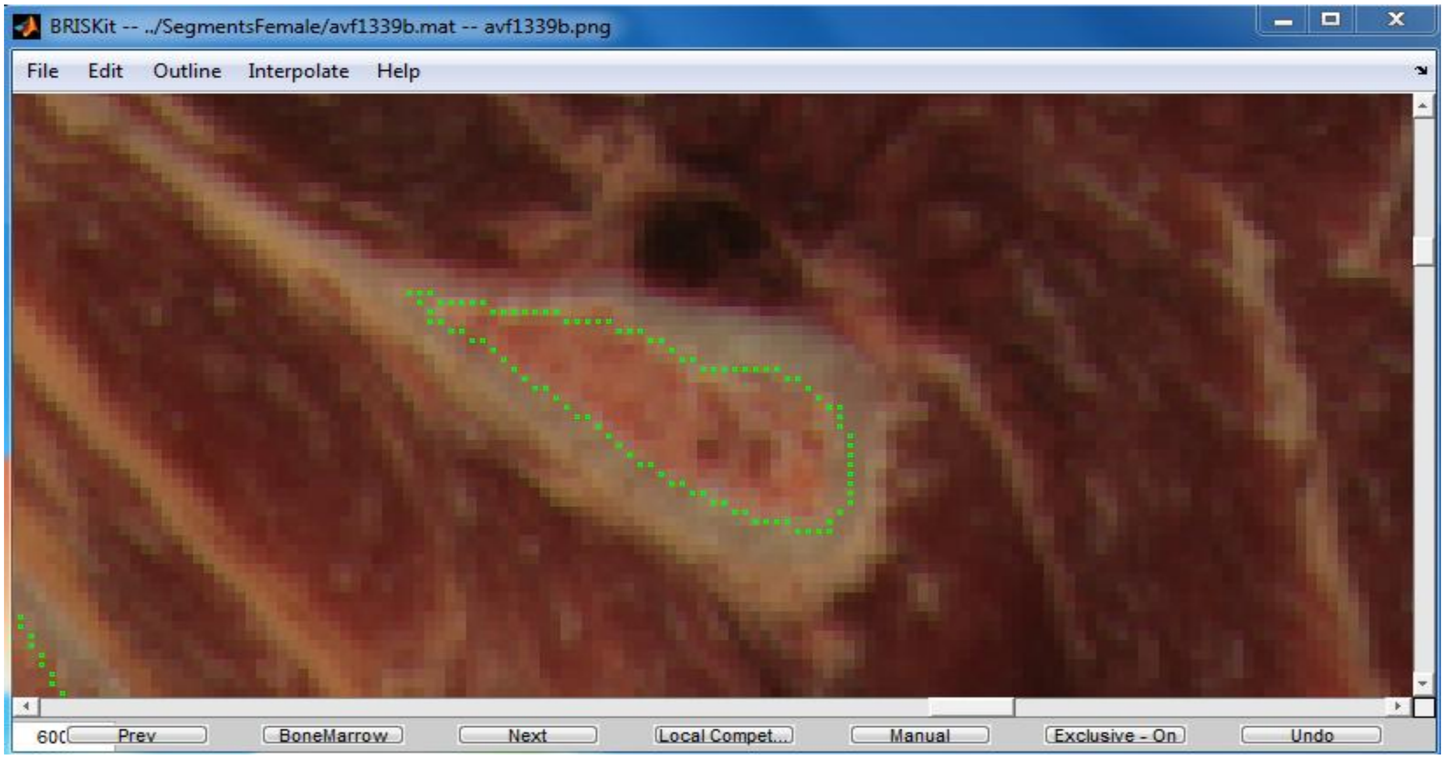

Fig. 11: Final display after the operation of *Select Shift* function is executed.

6.G. Expand/Contract

6.G.i. Objective

The *Expand/Contract* function is designed to give the user the ability to quickly expand/contract the selected regions.

6.G.ii. Algorithm Description

First, the coordinates of the pixels in the region selected by the user are found. Then, the boundaries are moved by taking into account only two tissues at a time. The boundary movement is in the form of either inward or outward shifting. If the boundary is shared with multiple tissues, then the algorithm iterates through the different tissues, allowing the user to expand or contract each pair differently, i.e., the user can shift the boundary between one pair of tissues inward and between another pair of tissues outward, and the amount of shift can also be different. Next, the algorithm waits for an input, i.e., the 'up,' 'down,' or 'right' arrow keys. If 'up' is pressed, the material ID of the current tissue is assigned to the pixels just outside the boundary. If 'down' is pressed, the material ID of the tissue just outside the boundary is assigned to the pixels along the boundary. If 'right' is pressed, the algorithm skips to the next tissue. After iterating through all of the tissues, the boundaries are updated.

6.G.iii. Implementation

Select a region whose boundaries will be expanded or contracted (Fig. 12). Then, use the 'up' and 'down' arrow keys to 'expand' and 'contract' the region by shifting its boundary one pixel outward or inward, respectively. Here, inward refers to the direction from the tissue just outside the current tissue toward the current tissue. Likewise, outward refers to the direction from the current tissue toward the tissue that is just outside the current tissue. After the expansion/ contraction is completed, press the 'right' arrow key and the boundaries will be updated (Fig. 13).

6.G.iv. Suggested Use

This function is useful for long boundaries that need a little push/pull to be corrected, e.g., the skin/fat boundary.

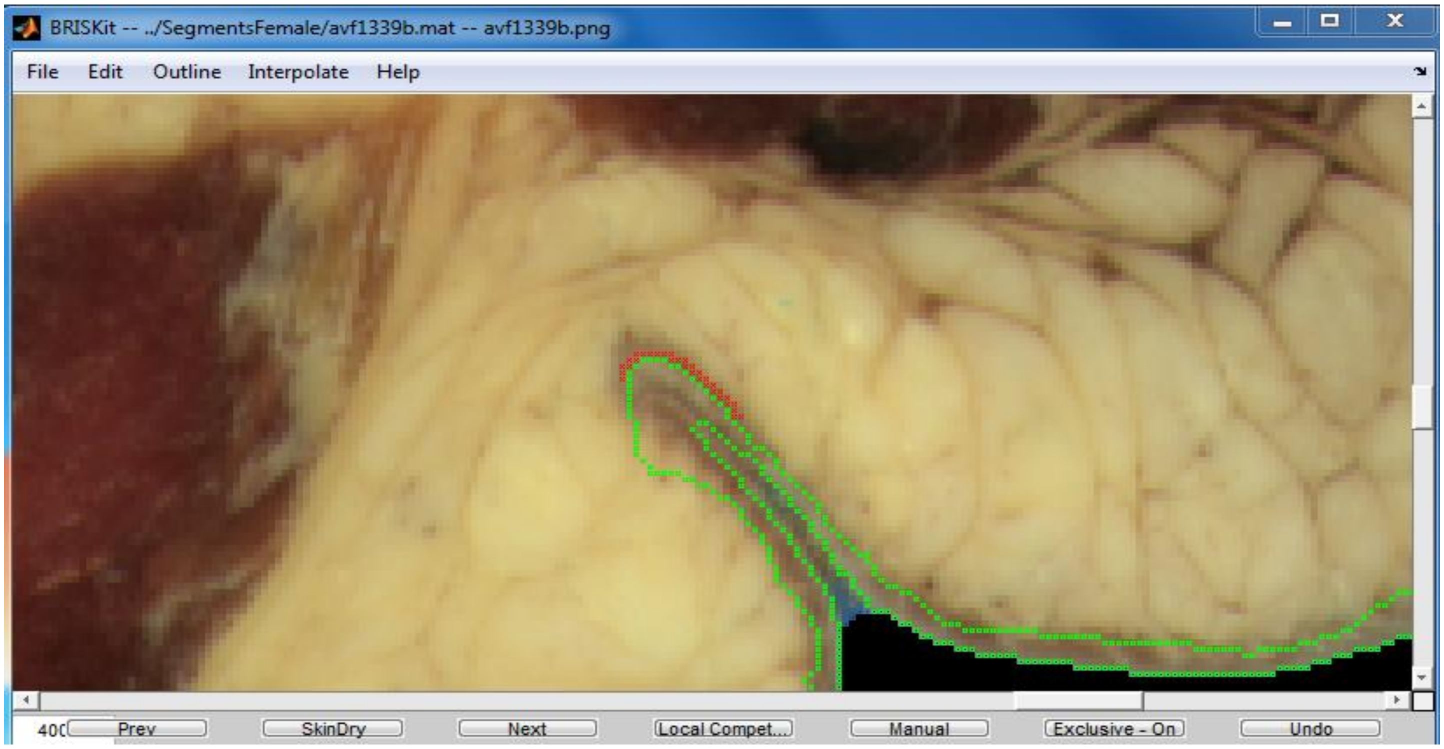

Fig. 12: The display after selecting the region using the *Expand/Contract* function.

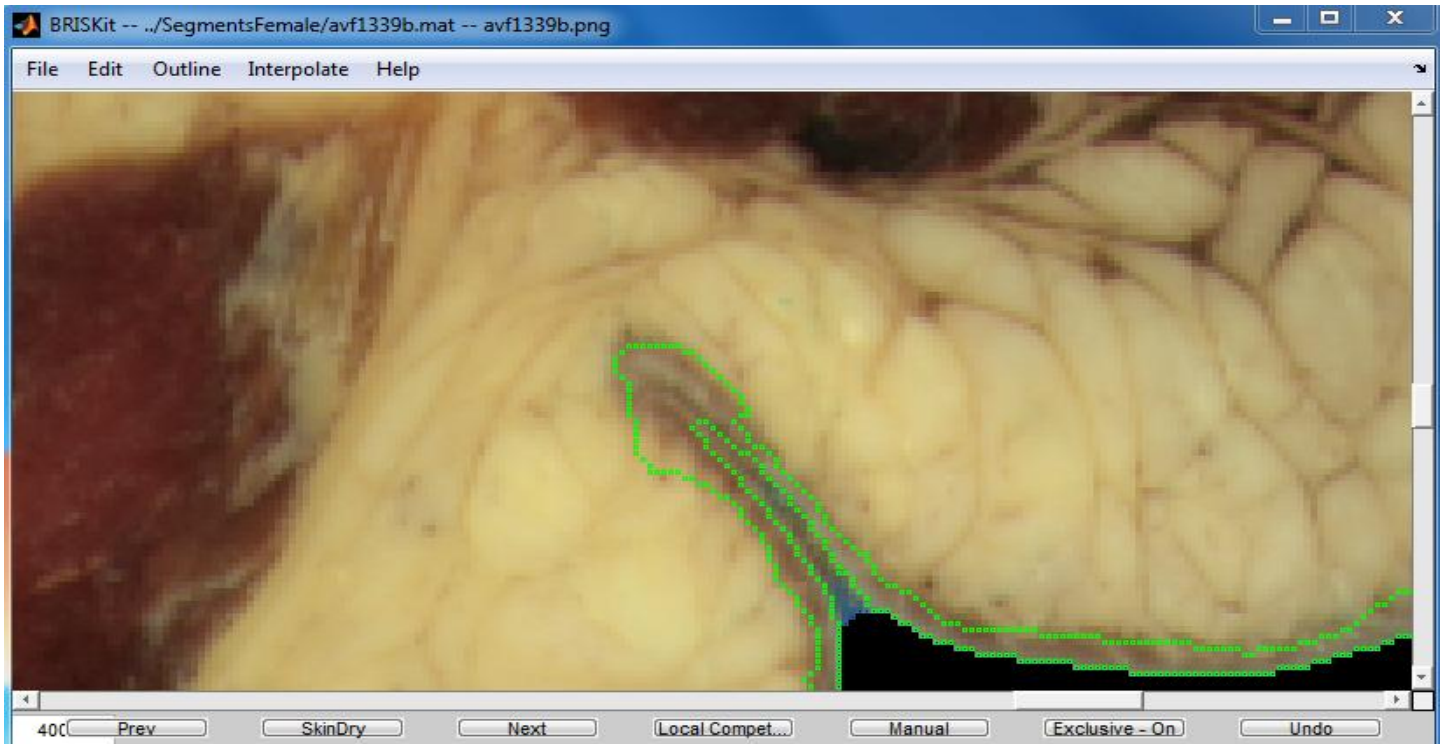

Fig. 13: Final display after the *Expand/Contract* function is executed.

6.H. AdjustSize

This function is identical to the *BoundaryAdjust* function (5.A.V). It is useful only if there is a need to start a slice for which no initial guesses exist or if there is a big gap between the slice that you want to work on and the slice that you have available as an initial guess.

7. OPEN PROBLEMS

Segmentation of cross-sectional images to identify tissues is a large scale problem that is hard to fully automate without sacrificing accuracy. Even using the functionality provided in BRISKit, processing a slice to improve the boundaries can require up to 45 mins-1 hour of human effort. This time can be reduced by developing even more sophisticated algorithms to segment boundaries that are difficult to segment automatically, such as the bone marrow / bone cortical boundary. Additional functionality and new algorithms can be added to BRISKit to address such boundaries. It should be mentioned that any such automatic or semi-automatic algorithm should outperform the easy to use and accurate (but repetitive and time consuming) alternative provided by the *Manual* function.

8. REFERENCES

[1] J. Massey, "Creating AustinMan: An Electromagnetic Voxel Model of the Visible Human." B.S. thesis, Dept. ECE, University of Texas at Austin, Austin, TX, 2011. Available at http://bit.ly/AustinMan .