Supplementary Software

End-user license agreement

1. **NIBIB** hereby grants and the **Recipient** accepts, subject to the terms and conditions of this **Agreement**, a term limited, royalty-free, nontransferable, nonexclusive internal license to use the **Software** for research and educational purposes only**,** as hereinafter defined with no provision for support, in order to conduct research and develop processes, methods, or marketable products for public use and benefit.

2. The **NIBIB** reserves the right to distribute the **Software** for research and educational purposes to others and to use it for any U.S. Government purpose. The **NIBIB** shall not be limited in future claims, publications, or distributions of the **Software** or modifications or versions thereof, or related information. No copyright to the **Software** is claimed by the United States Government in the United States under Title 17, U.S. Code.

3. The **Recipient** agrees in the use of the **Software** to comply with all the applicable **NIH** regulations and guidelines. The **Recipient** agrees that **Software** MAY NOT BE USED FOR TREATING OR DIAGNOSING HUMAN SUBJECTS. The **Recipient** agrees not to use the **Software** for research involving human subjects or clinical trials in the United States without complying with 21 C.F.R. Part 50 and 45 C.F.R. Part 46. The **Recipient** agrees not to use the **Software** for research involving human subjects or clinical trials outside of the United States without notifying the **NIBIB** in writing, of such research or trials and complying with the applicable regulations of the appropriate national control authorities. Written notification to the **NIBIB** of research involving human subjects or clinical trials outside of the United States shall be given no later than sixty (60) days prior to commencement of such research or trials.

4. THE **NIBIB** MAKES NO REPRESENTATIONS AND EXTENDS NO WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED, OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY WARRANTY THAT THE **SOFTWARE** WILL CONFORM TO SPECIFICATIONS, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR THAT THE USE OF THE **SOFTWARE** WILL NOT INFRINGE ANY PATENT, COPYRIGHT, OR TRADEMARK, OR ANY WARRANTY THAT THE DOCUMENTATION WILL CONFORM TO THE PROGRAM OR THAT **SOFTWARE** WILL BE ERROR FREE OR OTHER RIGHTS OR ANY OTHER EXPRESS OR IMPLIED WARRANTIES. In no event shall the **NIBIB** be liable for any damages, including, but not limited to direct, indirect, special or consequential damages, arising out of, resulting from, or in any way connected with the performance or breach of this license, whether or not based upon warranty, contract, tort or otherwise, whether or not injury was sustained by persons or property or otherwise, and whether or not loss was sustained from, or arose out of the any use, results or disposition of, the **Software** or services provided hereunder.

5. The **Recipient** agrees to waive any and all claims against, and to indemnify and hold harmless the U.S. Government, its employees, students, fellows, agents, consultants, contractors and subcontractors for any damage that the **Recipient** may incur from the **Recipient's** prior or future use of the **Software**, including any damages resulting from products based on results from the use thereof. The **Recipient** agrees to obtain this identical waiver of claims, indemnification and hold harmless agreement from any entities that are provided with technical data derived from the use of the **Software**.

MATLAB scripts in the folder "WBDeconvolution"

We provide 6 main MATLAB scripts for Wiener-Butterworth deconvolution (either with GPU or CPU processing) that run on MATLAB version 2016b and above with the image processing and parallel computing toolboxes. With the same scripts, we also provide an option for users to run traditional deconvolution for comparison.

1. **DeconSingleView.m**: single-view deconvolution for widefield fluorescence microscopy, confocal microscopy, instant structured illumination microscopy (iSIM), and light-sheet fluorescence microscopy data.
2. **DeconDualView.m**: alternating deconvolution for dual-view light-sheet fluorescence microscopy data.
3. **DeconQuadView.m**: additive deconvolution for quad-view light-sheet fluorescence microscopy data.
4. **DeconReflectiveDiSPIM.m**: joint deconvolution for reflective diSPIM (0.8/0.8 NA) data. Data are visualized from the perspective of one of the objective lenses.
5. **DeconReflectiveLLS.m**: joint deconvolution for reflective lattice light-sheet microscopy (0.7/1.1 NA) data. Data are visualized from the perspective of reflective coverslips.
6. **Generate_BackProjector.m**: a script that generates different back projectors.

There are 5 accessory scripts that are called after running the main software:

1. ReadTifStack.m: for loading TIFF stacks.
2. WriteTifStack.m: for writing TIFF stacks.
3. BackProjector.m: for generating back projectors based on the transpose of forward projectors.
4. ConvFFT3D.m: for performing 3D convolutions in the Fourier domain using fftn and ifftn functions.
5. IlluminationPattern.m**:** for simulating the Gaussian light-sheet pattern.

Before running the code, users should organize all input data as TIFF stacks (either 16 bit or 32 bit) as shown in **Fig. C1**. All time-series (4D) data are grouped into a single folder: Stack_0, Stack_1, Stack_2, …. etc, where the number indicates the relevant time point. For multiview data, users should organize the data as: StackA_N, StackB_N, StackC_N, StackD_N, … etc, where the capital letters A/B/C indicate the relevant view and the number N indicates the relevant time point. All views must be registered and have the same dimensions.

Users need to provide forward projectors (PSFs) of the microscope. These PSFs should be oriented from the same perspective as their corresponding views. They can be placed in the same folder as the input data or grouped into another folder, named PSFA for the first view, PSFB for the second view, and so on (**Fig. C2**). In the single-view deconvolution case, the corresponding Wiener-Butterworth back projectors will be automatically generated by calling Backprojector.m. In the multiview deconvolution cases, the corresponding Wiener-Butterworth back projectors can be generated by Generate_BackProjector.m and fed to the deconvolution scripts correspondingly. For reflective imaging, users need to generate the illumination patterns with the same size of input data according to the simulation (using IlluminationPattern.m to generate the Gaussian light-sheet) or measurement (e.g. using

beads to measure the parameters of the dithered lattice light-sheet pattern). The illumination pattern must be placed under the same folder as the PSF images and named Excitation.tif.

When running any of the first five deconvolution scripts, three dialog boxes will pop up to guide users to (1) upload the raw data (see **Fig. C1**); (2) upload the matching PSF files (see **Fig. C2**); and (3) set parameters (see **Fig. C3**) to

a) choose the deconvolution method (1 for traditional deconvolution or 2 for Wiener-Butterworth deconvolution, the default option is Wiener-Butterworth deconvolution);
b) choose CPU or GPU processing mode (0 for CPU or 1 for GPU, the default option is GPU);
c) define the iteration number (the typical number of iterations we used for Wiener-Butterworth deconvolution and traditional deconvolution are 1 and 10-20, respectively);
d) set the number of time points (in range form, e.g. 0-9) to be processed.

Deconvolved data (32-bit TIFF stacks) will be saved in the same folder that contains the input data. For time-series data, they are saved as Decon_0, Decon_1, Decon_2, …. etc, where the number indicates the relevant time point. Some intermediate files (e.g., back projectors) will be saved in the same folder that contains the PSF images. We provide an example for single-view light-sheet microscopy, including test data, a simulated PSF, and deconvolved output. Upon request, we can provide test data for other microscopy modalities.

MATLAB scripts in the folder "RegistrationFusion"

We provide 2 main MATLAB scripts that are used in performing affine registration and joint deconvolution for two volumetric views, with an Nvidia GPU card. These scripts can be run on MATLAB 2016b and above, but do not require the parallel computing toolbox. The graphics card must be supported by CUDA version 9 and above (https://developer.nvidia.com/cuda-gpus). In addition, an appropriate driver for the graphics card is required to make sure that the GPU card can be recognized by the CUDA program (users can download the drivers at http://www.nvidia.com/Download/index.aspx). Please note that the size of GPU memory determines the maximum image size that we can register and deconvolve. In our testing, we found that the GPU memory should be at least 4.4 times and 8.5 times the input image size (assuming 16-bit TIFF input) when attempting to run registration and joint deconvolution, respectively.

1. **Registration.m**: for performing 3D affine registration by calling Application programming interface (API) functions from a Dynamical-link Library (DLL) written in C++/CUDA.
2. **Fusion.m**: for performing both registration and joint deconvolution by calling API functions from the DLL.

The DLL along with associated dependency libraries are contained in the folder "**cudaLib**". All file paths in Registration.m and Fusion.m are preconfigured so that the two scripts are ready to run.

Before running the code, users should organize all the input data as TIFF stacks (**Fig. C1**): StackA_N, StackB_N … etc, where the capital letters A and B indicate the two perpendicular views; and the number N indicates the relevant time point. Here we will register B view to A view.

Users need to generate forward projectors and Wiener-Butterworth backward projectors according to the Online Methods and **Supplementary Note 2** (or the BackProjector.m script referenced above). The forward and backward projectors should be oriented from the same perspective as their corresponding views. These projector images can be placed in the same folder as the input data or grouped into another folder, named PSFA.tif for the forward projector of the first view, PSFA_BP.tif for the backward projector of the first view, PSFB.tif for the forward projector of the second view, and PSFB_BP.tif for the forward projector of the second view. Note that if the backward projectors are the transpose of the forward projectors, then the deconvolution is equivalent to traditional Richardson-Lucy deconvolution.

While running the Registration.m script, two dialog boxes will pop up to guide users to (1) upload the raw dual-view data (see **Fig. C1**); (2) set parameters (see **Fig. C4**) to

a) choose registration mode:
    1: translation only;
    2: rigid body;
    3: 7 degrees of freedom (translation, rotation, scaling equally in 3 dimensions);
    4: 9 degrees of freedom (translation, rotation, scaling);
    5: 12 degrees of freedom (translation, rotation, scaling, shearing);
b) set the number of time points (in range form, e.g. 0-9) to be processed.

After registration, the registered data (i.e., B view) will be saved in the same folder that contains the input data. They are saved as StackB_reg_N, where N indicates the number of time points.

While running the Fusion.m script, three dialog boxes will pop up to guide users to (1) upload the raw dual-view data (see **Fig. C1**); (2) upload the forward and backward projectors; (3) set parameters (similar to **Fig. C4** except there is an input for setting the iteration number) to

a) choose registration mode:
    1: translation only;
    2: rigid body;
    3: 7 degrees of freedom (translation, rotation, scaling equally in 3 dimensions);
    4: 9 degrees of freedom (translation, rotation, scaling);
    5: 12 degrees of freedom (translation, rotation, scaling, shearing);
b) set the iteration number (the typical number of iterations we used for Wiener-Butterworth deconvolution and traditional deconvolution is 1 and 10-20, respectively);
c) set the number of time points (in range form, e.g. 0-9) to be processed.

After processing, the registered B view data and joint deconvolution outcomes will be saved in the same folder that contains the input data. They are saved as StackB_reg_N and Decon_N, where N indicates the number of time points. Upon request, we can provide example data for registering and deconvolving diSPIM data, including test data, forward and backward projectors, registered data, and deconvolved output.

ImageJ plugin in the folder "ImageJ_diSPIMFusion"

For implementing registration and joint deconvolution of two diSPIM volumetric views, we also provide an ImageJ Macro "**diSPIMFusion_UI.ijm**" that can create a user interface to call the executable app and DLL in the "**cudaLib**" folder.

To run the plugin, the PC needs to have (1) a Windows 7 or 10 operation system; (2) either ImageJ or Fiji; (3) a graphics card supported by CUDA 9 and above; and (4) an appropriate driver for the graphics card as mentioned before.

To install the program, users should copy the "diSPIMFusion" folder to ImageJ or Fiji's main folder and install the UI macro: ImageJ (or Fiji) Plugins --> Macros --> Install..., choose the macro file "diSPIMFusion_UI.ijm". Then there will be a "dispimfusion" option listed on the Plugins--> Macros menu.

To use the program:

a) run the program: ImageJ (or Fiji) Plugins --> Macros --> dispimfusion; or users can directly open the "diSPIMFusion_UI.ijm" within ImageJ (or Fiji) and run it;
b) a pop-up dialog will show up (**Fig. C5**), then choose "Single color" or "Multiple colors" option;
c) sequentially select folders: StackA folder → StackB folder → Output folder;
for multiple colors: Main folder→ Output folder; The input images for all colors should be in the main folder and organized as in the folder convention shown in **Fig. C6**;
d) confirm and modify the parameters in the next pop-up panel (**Fig. C7**);
e) if "Customize initial transformation matrix" is selected, users are guided to choose a matrix file (a text file is provided within the example data);
f) run the registration and joint deconvolution;
g) once the processing is completed, all messages will show up in the ImageJ log window.

We provide example data for registering and deconvolving diSPIM data, including test data, forward and backward projectors and deconvolved output.

MATLAB scripts in the folder "ClearedTissueProcessing"

We provide 3 main MATLAB scripts for stitching, registration and deconvolution of large cleared tissue data, imaged with diSPIM. They can be run on MATLAB version 2016b and above and require the image processing and parallel computing toolboxes. The folder also includes two forward projectors (PSFA_FP.tif and PSFB_FP.tif), two backward projectors (PSFA_BP.tif and PSFB_BP.tif) for a diSPIM equipped with cleared tissue objective lenses (0.4/0.4 NA, Special Optics).

1. **Stitching_GUI.m**: for stitching diSPIM subvolumes (tiles) (in the subfolder "Stitching").
2. **BigData_PreProcessing.m**: for converting the stitched data (only one time point and one color) from stage scanning mode to light-sheet scanning mode, interpolating and rotating to the perspective of the coverslip.
3. **BigData_PostProcessing.m**: for registration and joint Wiener-Butterworth deconvolution of the diSPIM data (after transformation via BigData_PreProcessing.m and resaving as TIFF stacks via ImageJ).

There are 4 accessory scripts and 2 subfolders supporting the main scripts:

1. ReadTifStack.m: for loading small TIFF stacks (< 4GB)
2. WriteTifStack.m: for writing TIFF stacks (< 4GB).
3. Blend2D.m: for stitching two small TIFF stacks into one with linear blending.

4. ConvFFT3_S.m: for performing 3D convolutions in the Fourier domain using fftn and ifftn functions.
5. TIFFStack-master: a subfolder that contains libraries and functions for virtually loading big TIFF stacks (> 4GB) (http://dylan-muir.com/articles/mapped_tensor/).
6. Stitching: a subfolder that contains functions for calculating 3D translation shifts, creating weight images for each tile and writing BigTIFF stacks.

After launching the Stitching_GUI.m script (**Fig. C8**), the first step for users is to open all tile files. File names and file information including image dimensions will show up on the right-hand information box in the GUI (**Fig. C9a**). Once this information is displayed, in the second step users set the tile number and order for each dimension (**Fig. C9b**). The file names will be reorganized and displayed on the right-hand information box with the following format: for each color, left to right columns show tiles along X dimension; whereas top to bottom rows show tiles along Y dimension (**Fig. C9c**). Users can modify the order by editing the file name. Next, users have an option to preview the tiles at different z slices (**Fig. C10a**) with separate windows for image display (**Fig. C10b**) and contrast adjustment (**Fig. C10c**). The third step calculates coarse 3D translational shifts for all pairs of adjacent tiles using Fourier-based phase correlation. Users have an option to run this functionality with GPU or CPU; after this step, the overlap ratio between adjacent tiles (i.e., the ratio of overlap size over the tile size) is displayed on the information box (**Fig. C11a**). Users can skip this step if coarse overlaps between tiles are known and set as an average ratio in the box right under Step 2 (**Fig. C11a**). The next step is to compute fine, subpixel shifts using the GPU-based registration method (**Fig. C11b**). A pop-up window will guide users to locate the GPU-based registration DLL file which is put under the "**cudaLib**" folder. After this step, normalized cross-correlations (NCC) between the cropped regions are shown in the information box. Users have an option to save the overlapped regions to visually check overlapped regions before and after registration. Users can also preview the merged images at different z locations within a new image window (**Fig. C12**). Finally, stitched tiles can be saved as individual TIFF files at each z slice and color, or as a single BigTIFF z stack for each color. Saving multiple TIFF files for every z slice is implemented with multiple CPU threads, which can be 3-7x faster than saving as a single 3D TIFF stack.

After the two diSPIM views arestitched, users should organize the two stitched data as TIFF stacks (similar to **Fig. C1**) before running BigData_PreProcessing.m script: StackA_StageMode.tif and StackB_StageMode.tif, where the capital letters A and B indicates the two perpendicular views. While running BigData_PreProcessing.m, a dialog box will pop up to guide users to (1) indicate the folder that contains the raw view data; (2) set the acquisition step size used in stage scanning. After running the script, the two views are downsampled by a factor of 5, and automatically saved as StackA_LightsheetMode_5x.tif and StackB_LightsheetMode_5x.tif, respectively. Moreover, a series of 2D images (with full resolution) along the XZ view (perpendicular to the coverslip) will be saved under two new folders StackA_LightsheetMode and StackB_LightsheetMode. Users need to load these in ImageJ and resave as BigTIFF stacks as StackA_LightsheetMode.tif and StackB_LightsheetMode.tif under the same folder that contains the raw data.

Before running BigData_PostProcessing.m script, users should deposit the attached PSFs (PSFA_FP.tif, PSFA_BP.tif, PSFB_FP.tif, and PSFB_BP.tif) in the same folder that contains the raw data. As previously described, authors can generate their own forward projectors and Wiener-Butterworth backward

projectors according to the **Online Methods** and **Supplementary Note 2**. Please note that these projectors should be oriented from the perspective of the coverslip (i.e., rotated from the objective collection view to the coverslip view). While running BigData_PostProcessing.m, dialog boxes will pop up to guide users to indicate the folder that contains all the data and PSFs. After running the script, the deconvolution results (a series of XY images, 16 bit TIFF format) will be saved in a separate folder "Joint Decon". Users can resave them as a big TIFF stack by loading them in ImageJ. Upon request, we can provide a small cleared tissue dataset (~20 GB) for testing this postprocessing pipeline (i.e., coarse registration, cropping, fine registration, joint deconvolution and stitching in a single MATLAB script).

Python scripts in folder "DLResolutionRecovery"

We provide 2 Python scripts for running a deep learning model with TensorFlow to recover resolution and contrast via deconvolution. These scripts are designed for single-view input training, single-view input validation, dual-view input training, and dual-view input validation. They can be run on Python 3.5.2 with Tensorflow framework version 1.4.0 in either Windows 7 (or later) or Ubuntu 16.04.4 LTS (or later) operating systems. We have tested the software using Windows 7 with two Nvidia GPU Tesla K40c cards (12 GB memory each) and Ubuntu 16.04.4 LTS with two Nvidia GeForce GTX 1080 Ti GPU cards (11 GB memory each). Users may refer https://www.tensorflow.org/install to correctly install Tensorflow packages.

1. **Single_Input_DL.py**: for single-view input training and testing.
2. **Dual_Input_DL.py**: for dual-view input training and testing.

Before running the script for single-view input training and testing, users should create five folders (**Fig. C13**): one for 'ground_truth' data, the second for 'input' data, the third for 'test' data, the fourth for 'output' data (for saving recovery results either through the training process or validation), and the last one for 'model' (for saving or loading the training outcome model - model.ckpt). For running the script for dual-view input training and testing, users need to create two additional folders 'input1' and 'input2' for two input datasets. All data should be organized as TIFF stacks (either 16 bit or 32 bit).

After organizing the data into the folders, users need to load the python scripts, and confirm or modify the parameters in the code (**Fig. C14**): (1) setup the root directory of the above folders; (2) choose 'training' or 'testing' mode (the default value is set as 'testing'); (3) set iteration parameters including training iteration number (the default value is set as 12000), training batch size (i.e., the number of data volumes will be simultaneously loaded for training at each iteration, the default value is set as 1), test number (i.e., the number of testing data volumes), and test batch size (the number of data points that will be simultaneously loaded for validation at each iteration, the default value is set as 1); (4) set the image size of the input stack; (5) check the GPU setting.

Users can check the training progress when running the scripts, including the values of MSE (mean square error), SSIM (structural similarity index) and run time that will be displayed in the Python GUI every 100 iterations; the training output images that will be automatically saved every 200 iterations in the 'output' folder; and the training model and parameters that will be saved every 2000 iterations in the 'model' folder.

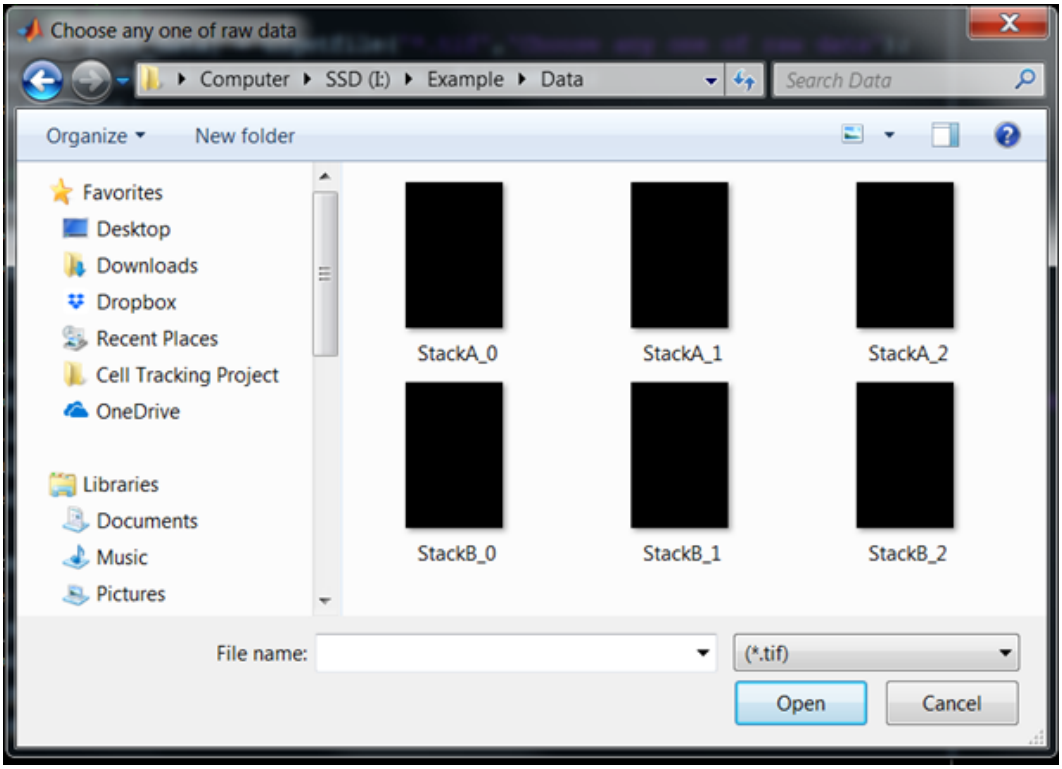


Fig. C1. Time series data are grouped into a single folder, and named StackA_0, StackA_1, StackA_2, …. etc for the first view; StackB_0, StackB_1, StackB_2, …. etc, for the second view; StackC_0, StackC_1, StackC_2, …. etc, for the third view; and so on. Here the number indicates the relevant time point. Upon running one of the main scripts, this dialog box will pop up to guide users as to where to locate the files.

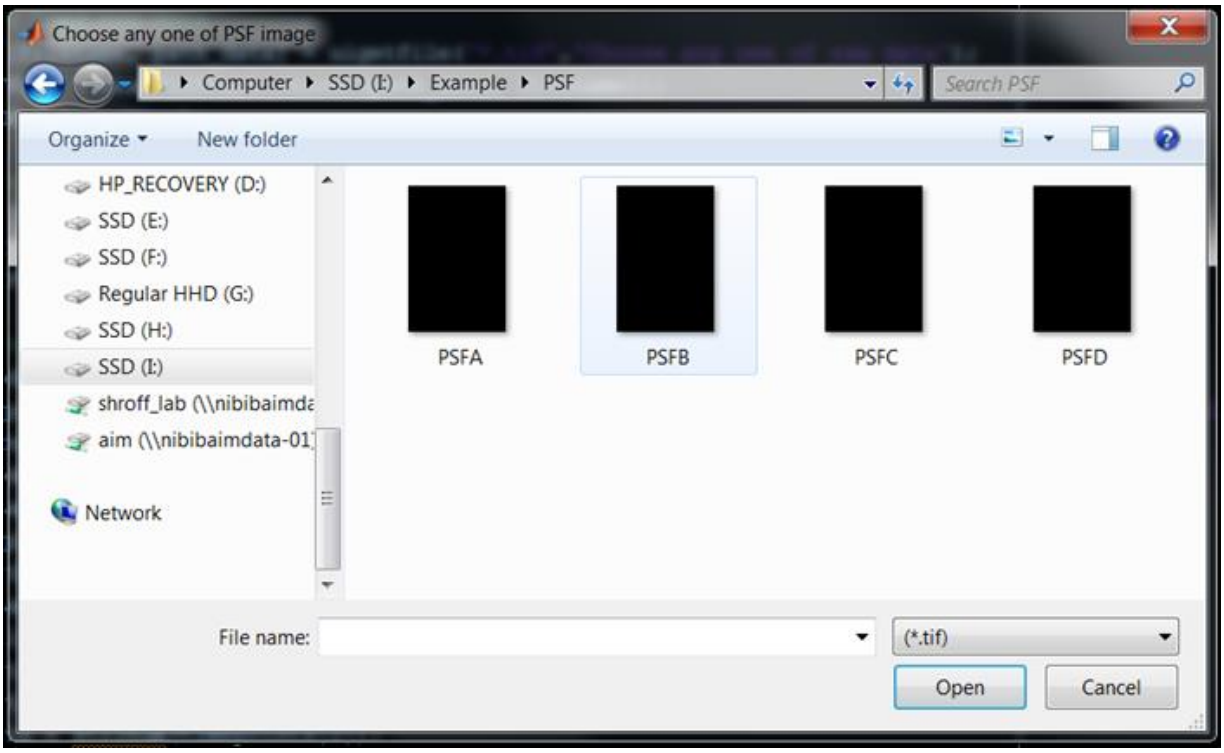


Fig. C2. PSF images are grouped into a single folder, and named PSFA, PSFB, PSFC, etc, the capital letter A, B, C… corresponding to the number of views. Upon running one of the main scripts, this dialog box will pop up to guide users as to where to locate the folders (choose any one of the PSF images).

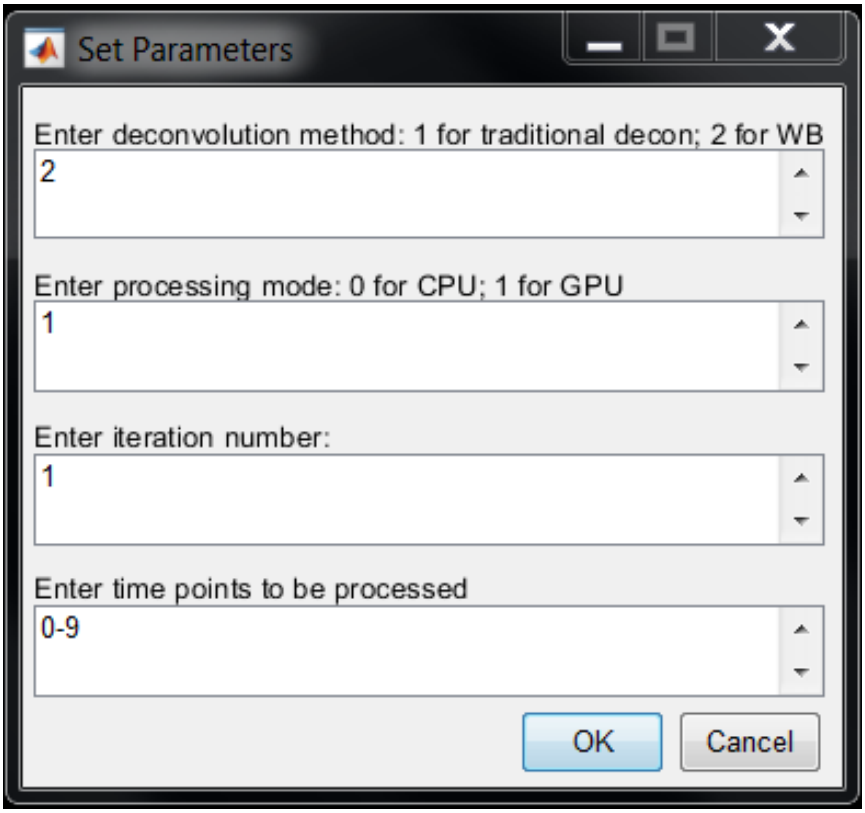


Fig. C3. The dialog box for parameter setting in SingleViewDecon.m, including the deconvolution method (1 for traditional deconvolution deconvolution or 2 for Wiener-Butterworth), processing mode (0 for CPU or 1 for GPU), iteration number and time points to be deconvolved.

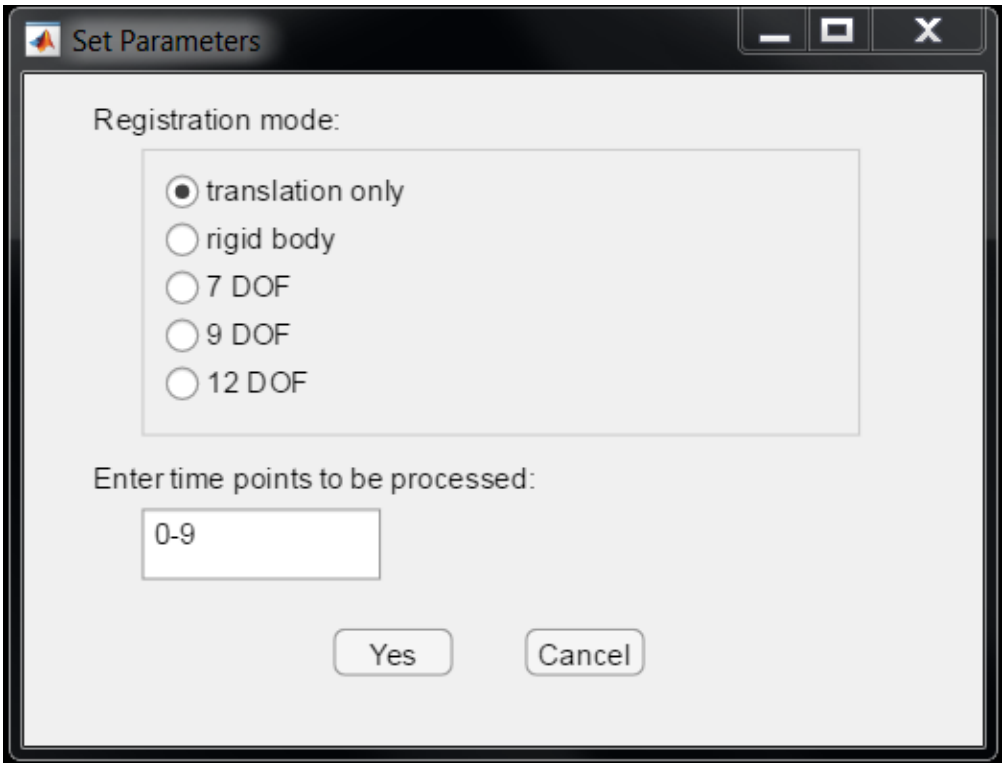


Fig. C4. The dialog box for parameter setting in Registration.m, including the registration mode and time points to be registered.

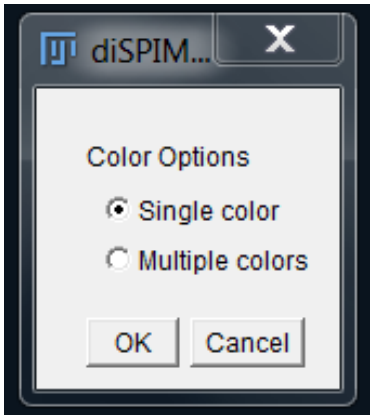


Fig. C5. The dialog box for choosing colors in the ImageJ Macro “diSPIMFusion_UI.ijm”.

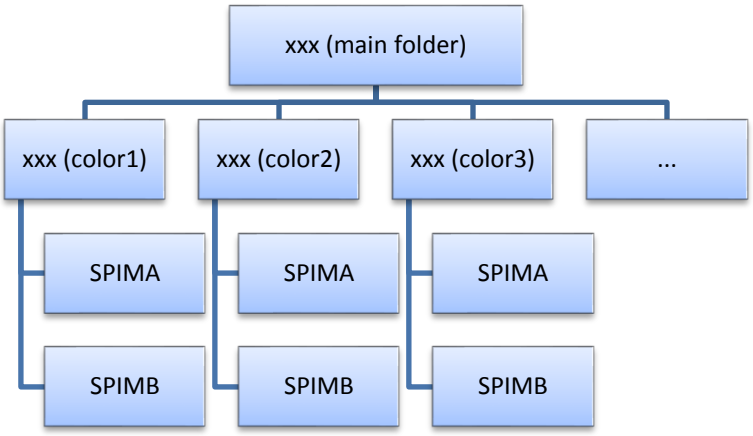


Fig. C6. Folder convention for organizing multicolor datasets when using the ImageJ Macro "diSPIMFusion_UI.ijm". The "xxx ( )" indicates the name for the folders.

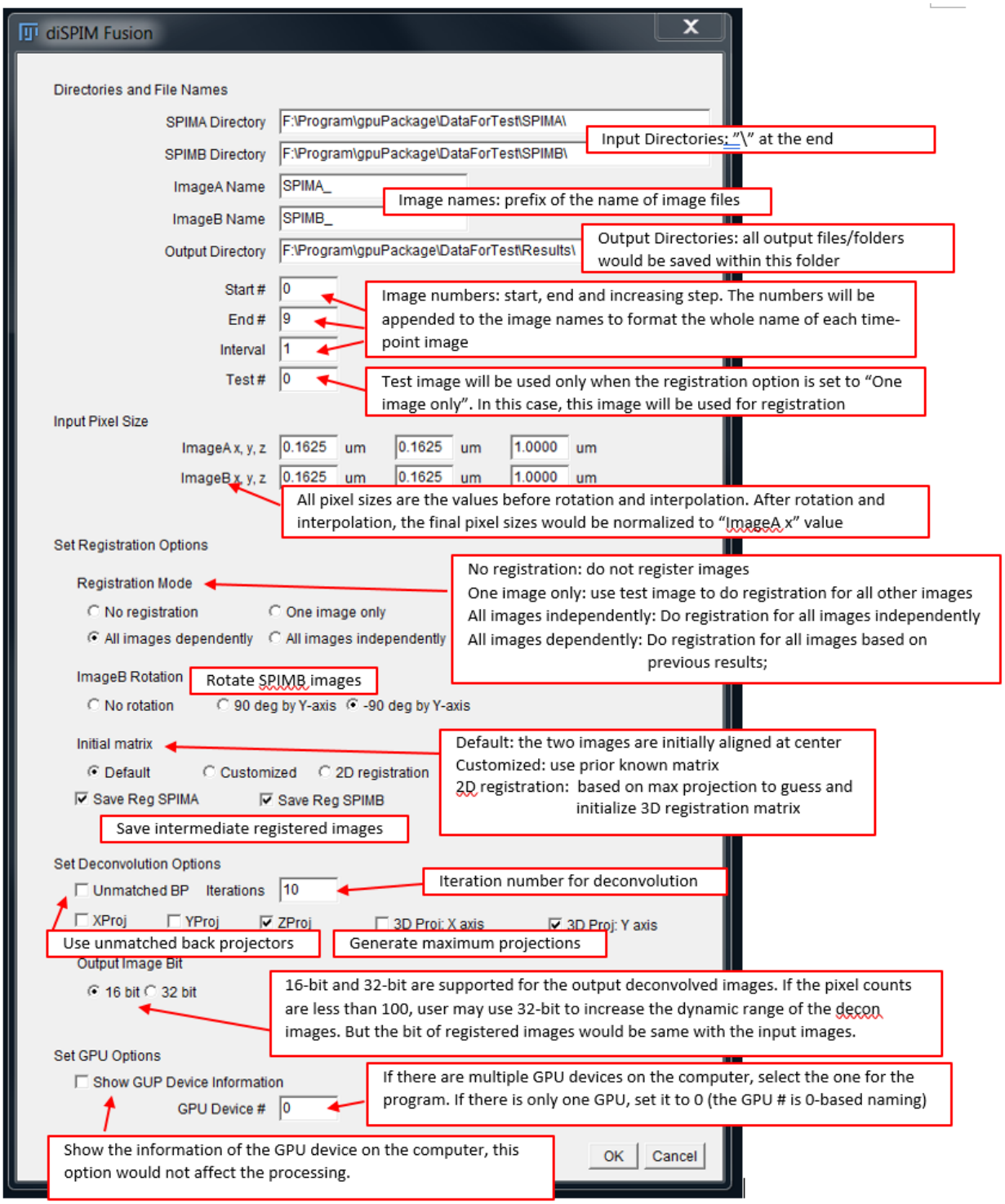


Fig. C7. The dialog box for parameter settings in the ImageJ Macro "diSPIMFusion_UI.ijm"

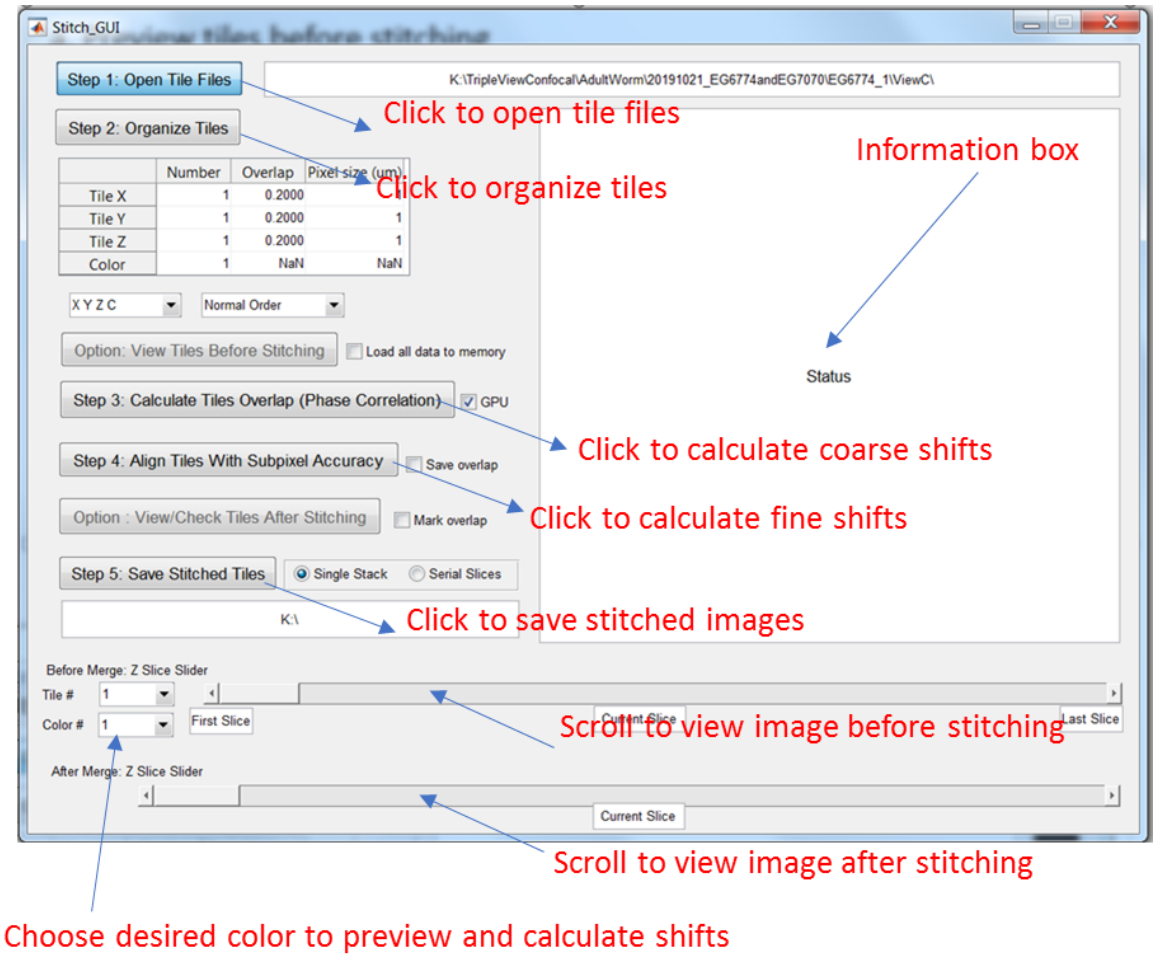
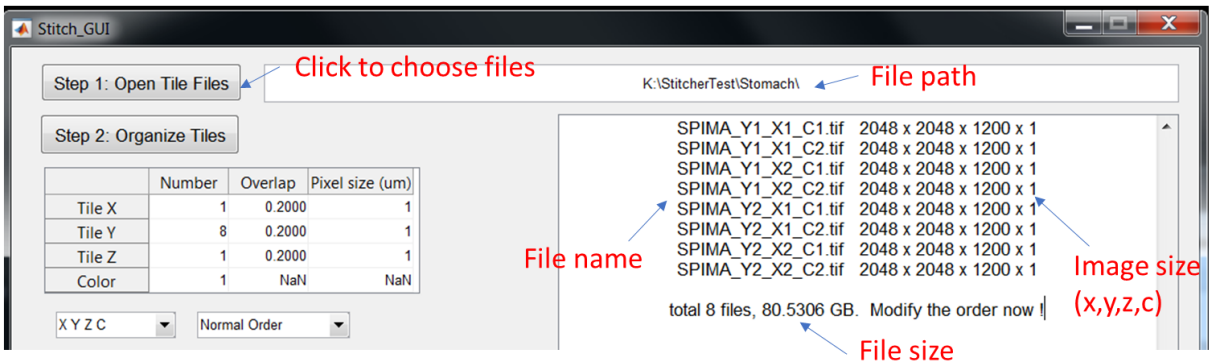
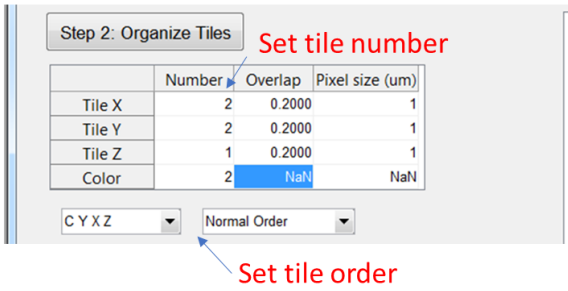


Fig. C8. GUI for stitching diSPIM tiles. Uses five main steps: open files, organize tiles, calculate coarse shifts using phase correlation method, compute subpixel shifts, and save stitched images. The GUI also provides options to preview tiles before and after stitching. After running each step results will be displayed and updated in the right-hand information box.

a. Open files



b. Organize tiles



c. Tile order

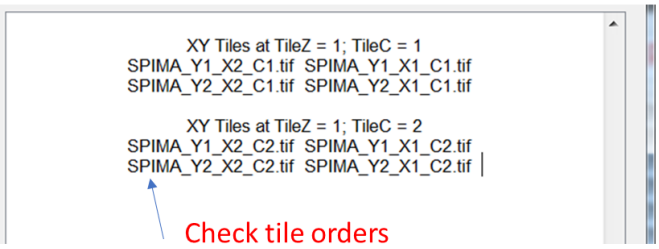
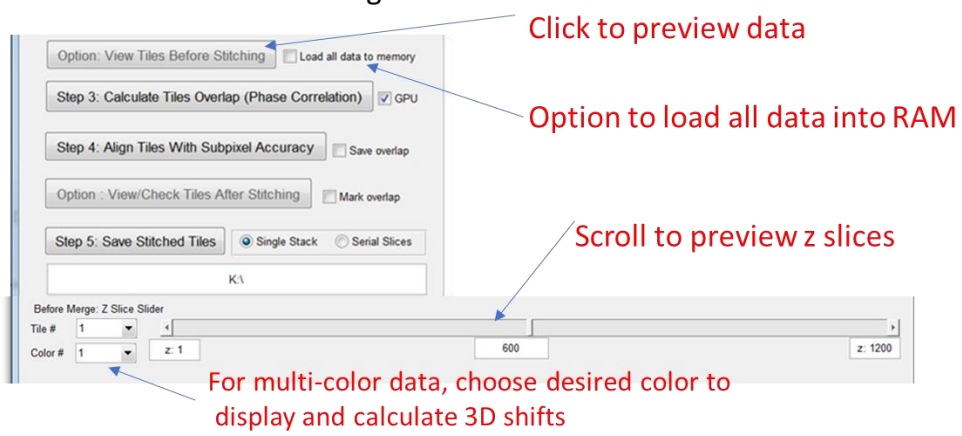
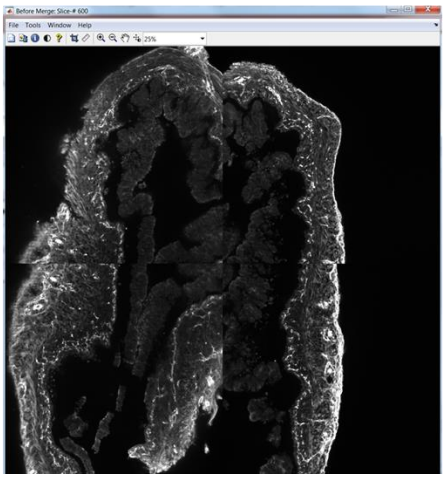


Fig. C9. First two steps in stitching GUI. (a) After clicking 'Step 1: Open Tile Files', users are instructed to load all files for stitching, and the information box will show the image information; (b) After setting the tile number and orders in the left-hand table, then clicking 'Step 2: Organize Tiles', the file names will be re-organized. (c) Tiles along the X dimension are listed in the information box from left to right columns, whereas tiles along the Y dimension are listed from top to bottom rows. Users can modify the file name in the information box if necessary.

a. Preview tiles before stitching



b. Image window



c. Image tool for adjusting contrast

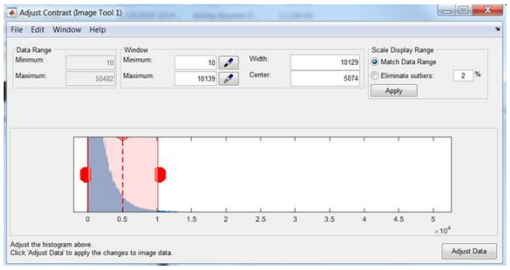
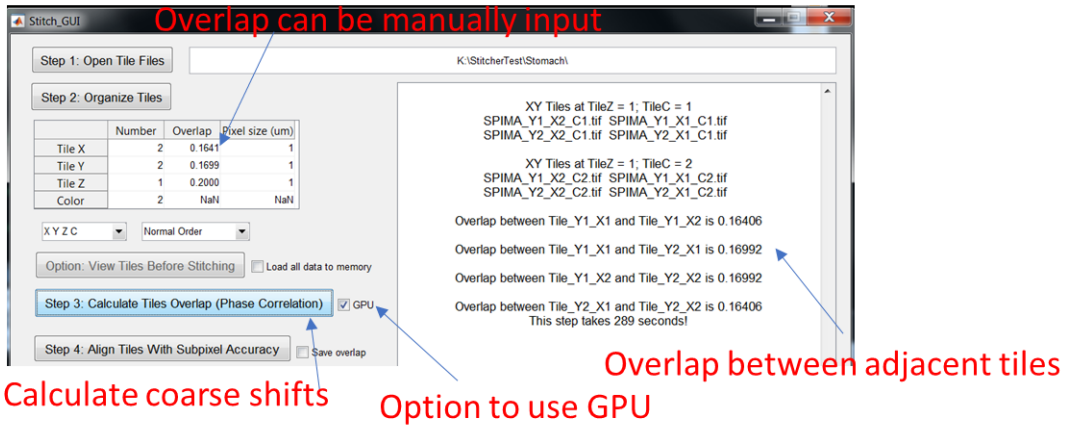


Fig. C10. GUI for previewing tiles before stitching. By clicking the button, 'Option: View Tiles Before Stitching' (a), an image window (b) and an image tool for contrast adjustment (c) will pop up. Users can scroll the horizontal bar to preview lateral slices at different z locations. In multicolor datasets, users have to select the desired color to display and use for calculating 3D shifts for subsequent steps.

a. Calculate coarse shifts



b. Compute fine shifts

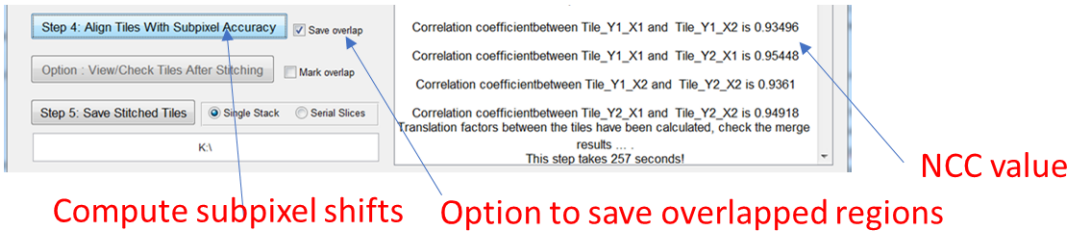
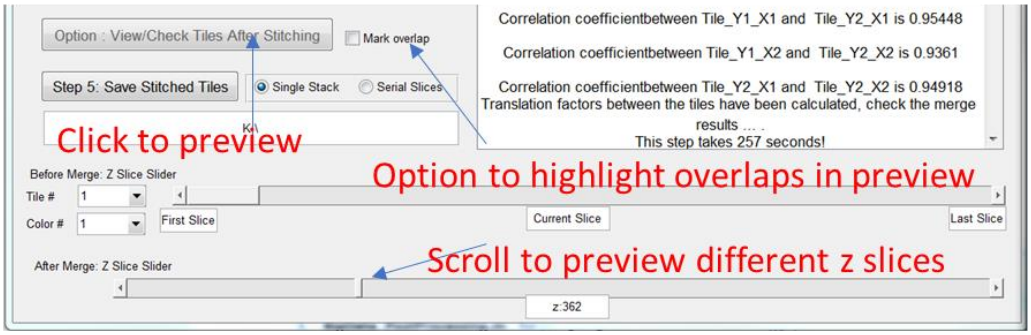


Fig. C11. GUI for calculation of 3D shifts. (a) Upon clicking 'Step 3: Calculate Tiles Overlap (Phase Correlation)', the coarse 3D shifts will be calculated. Once complete, the overlap ratio between adjacent tiles will be displayed in the information box. (b) After clicking 'Step 4: Align Tiles With Subpixel Accuracy', fine registration of the overlapped regions will be performed, and the normalized cross-correlation (NCC) between the cropped regions will be displayed in the information box.

a. Preview images after stitching



b. Image window for stitching

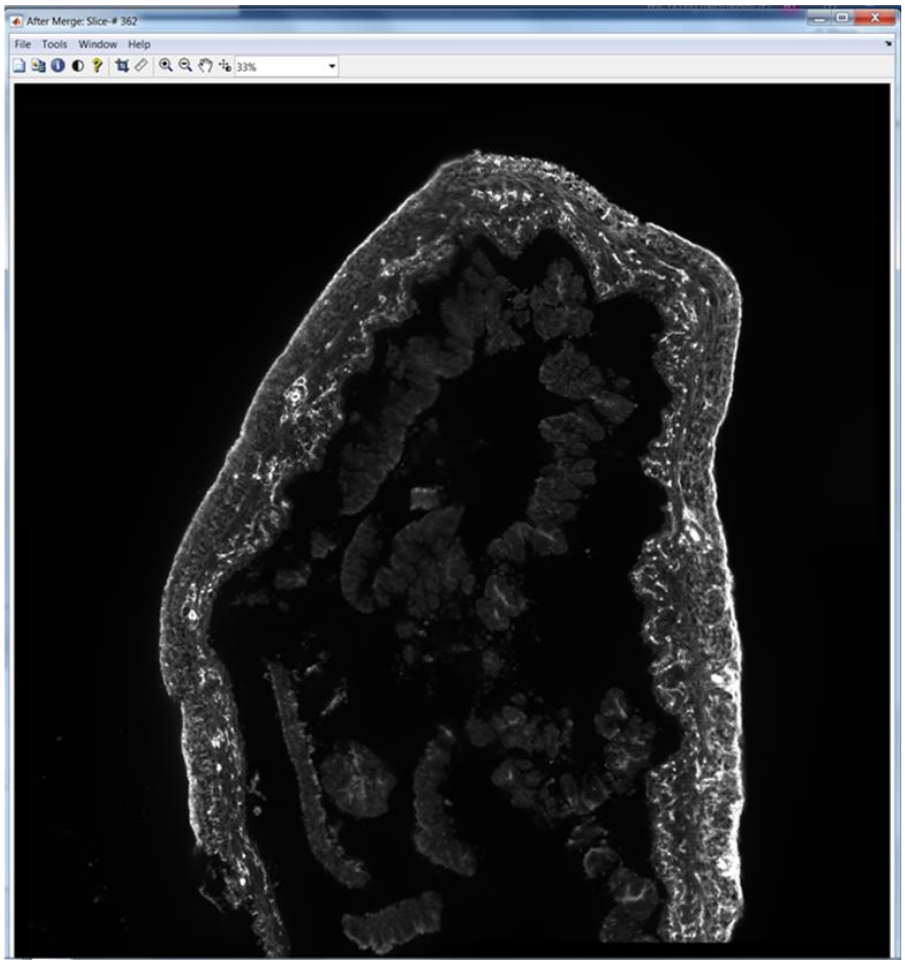


Fig. C12. GUI for previewing tiles before stitching. By clicking the button – 'Option: View/Check Tiles After Stitching' (a), an image window (b) will popup. Users can use the horizontal scroll bar to preview lateral slices at different z locations.

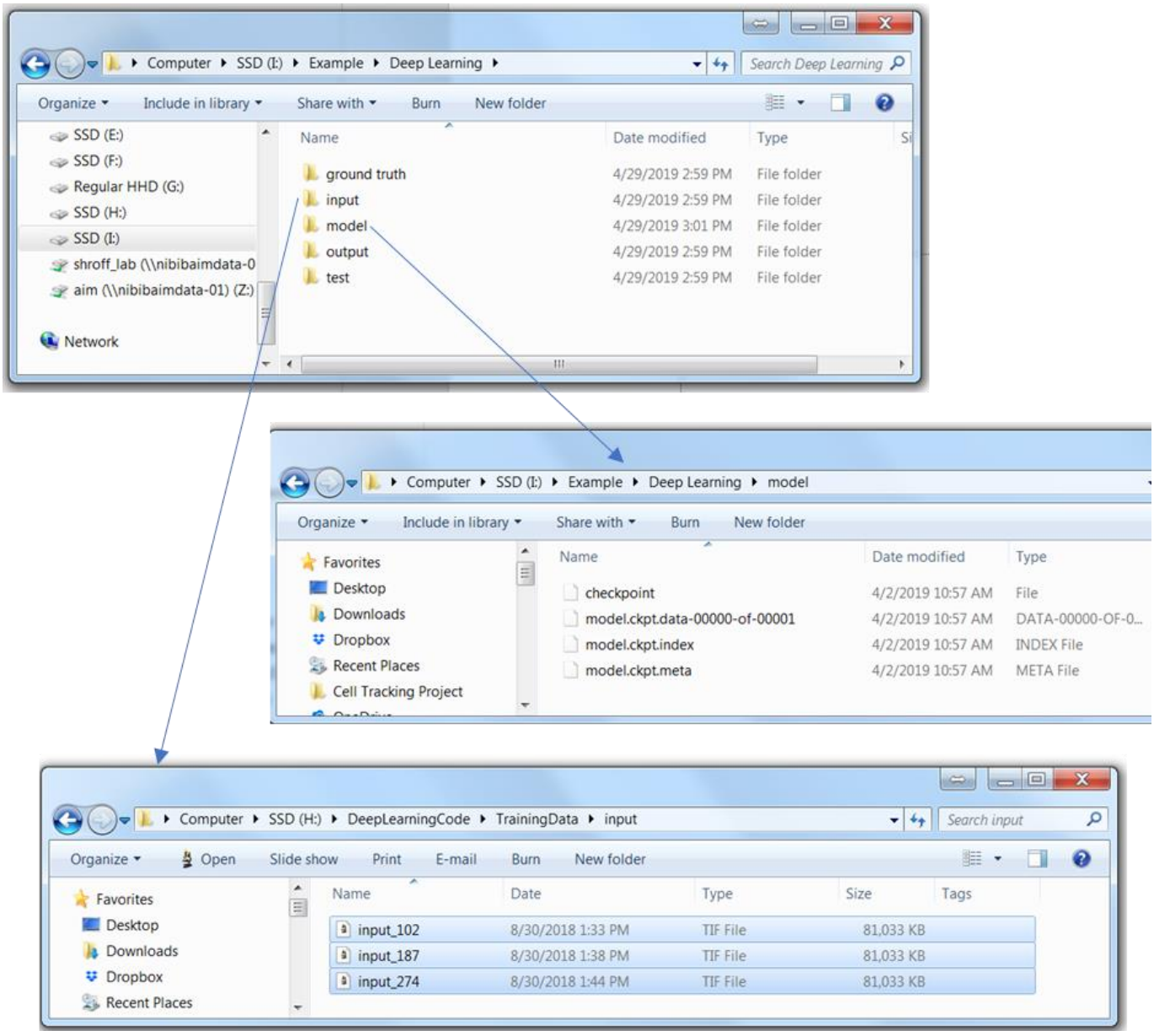


Fig. C13. Data and model for single-view input deep learning are organized into five folders: 'ground_truth', 'input', 'test', output' and 'model'.

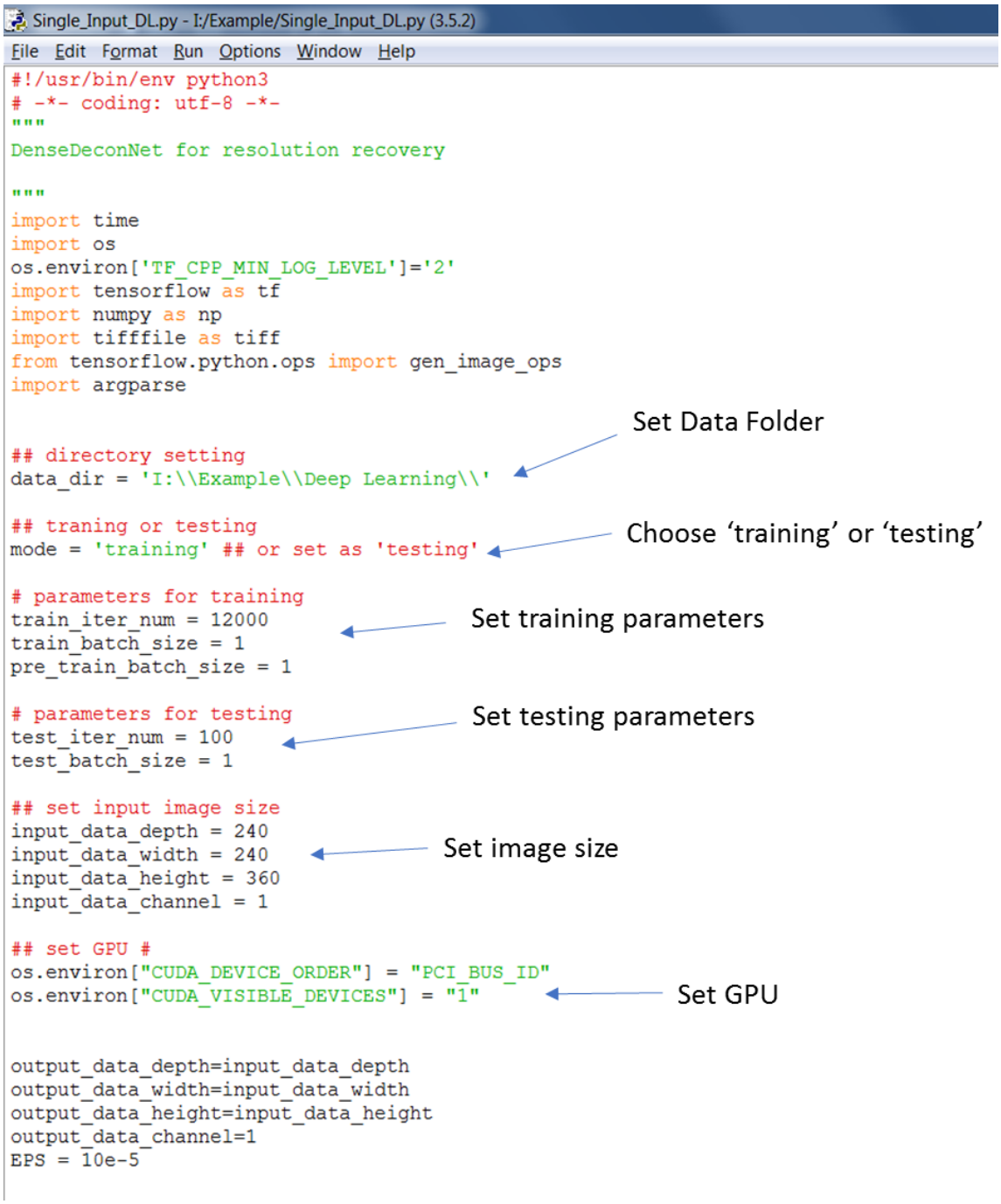


Fig. C14. Description of Python script - Single_Input_DL.py. Users need to check or modify the parameters: root directory of the data folders, performing 'training' or 'testing', training and testing parameters, image size and GPU settings.