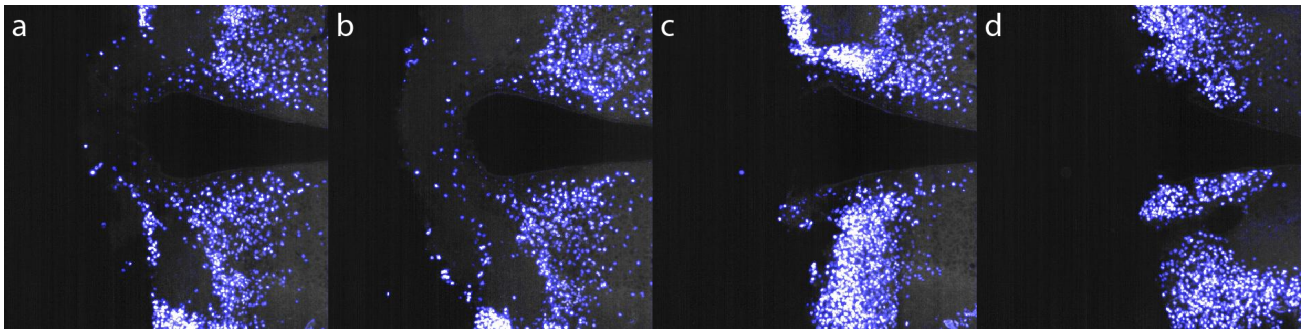
# BigStitcher: Reconstructing high-resolution image datasets of cleared and expanded samples

David Hörl[*], Fabio Rojas Rusak[*], Friedrich Preusser, Paul Tillberg, Nadine Randel, Raghhav Chhetri, Albert Cardona, Philipp Keller, Hartmann Hartz, Heinrich Leonhardt, Mathias Treier, Stephan Preibisch[#]

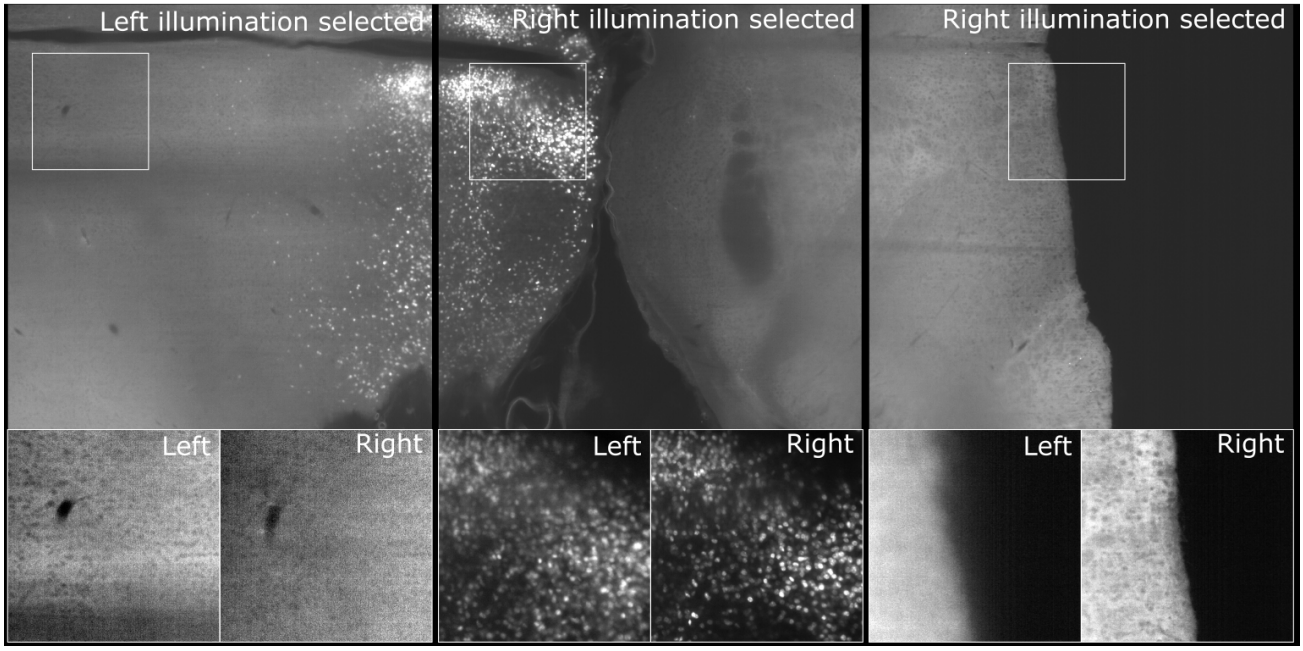
# SUPPLEMENTARY FIGURES

## SUPPLEMENTARY FIGURE 1: Fluorescence preservation in cleared samples



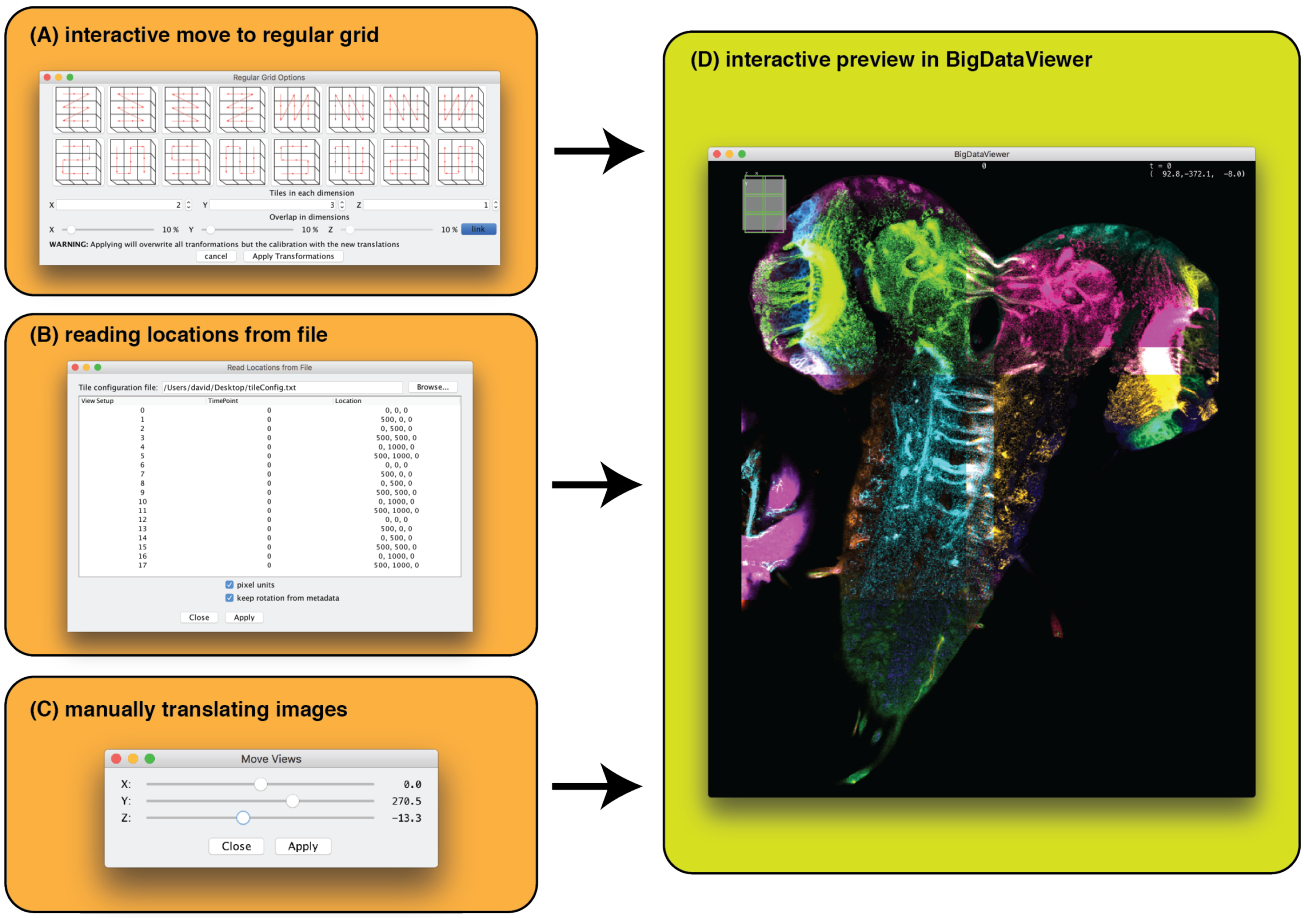

Supplementary Figure 1: *Fluorescence preservation and image analysis in cleared samples.* **(a)-(d)** Optical sections through an adult mouse hypothalamus in a CLARITY-cleared adult mouse brain expressing H2B-GFP in all bsx neurons. Fluorescence is preserved throughout the clearing procedure and can be used in advanced image analyses. The examples shows nuclear segmentation of pixel-wise classification using a Random Forest[1] model. Classification results (probability maps) are overlaid in blue onto the grayscale raw intensities.

## SUPPLEMENTARY FIGURE 2: Automatic illumination selection



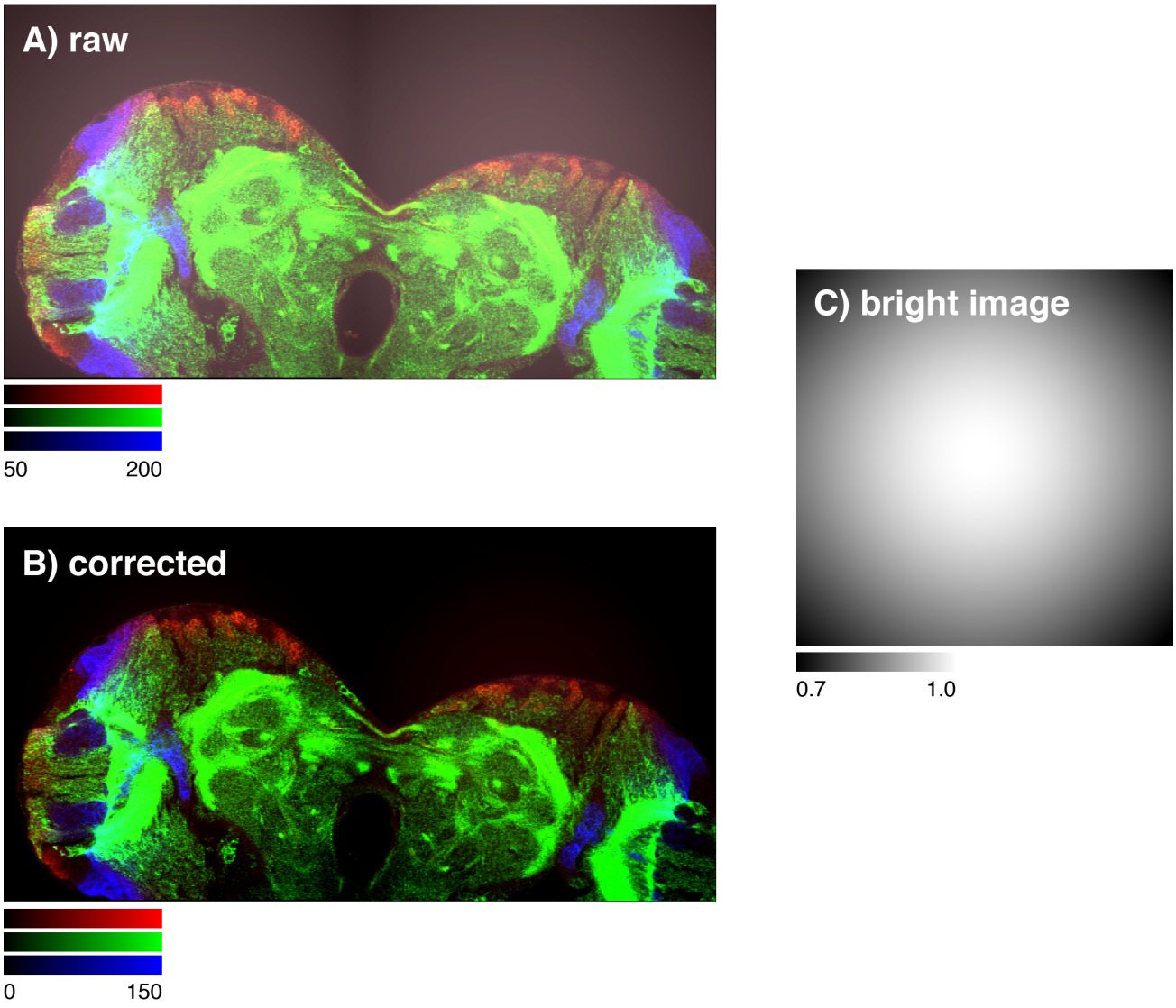

Supplementary Figure 2: *Automatic illumination selection.* Best illumination for three consecutive tiles was selected based on brightest view for each tile. Close-ups shows the specified region for both illumination directions and illustrate that this metric is usually sufficient for a correct decision of the sharper illumination direction of each tile.

## SUPPLEMENTARY FIGURE 3: Manual alignment



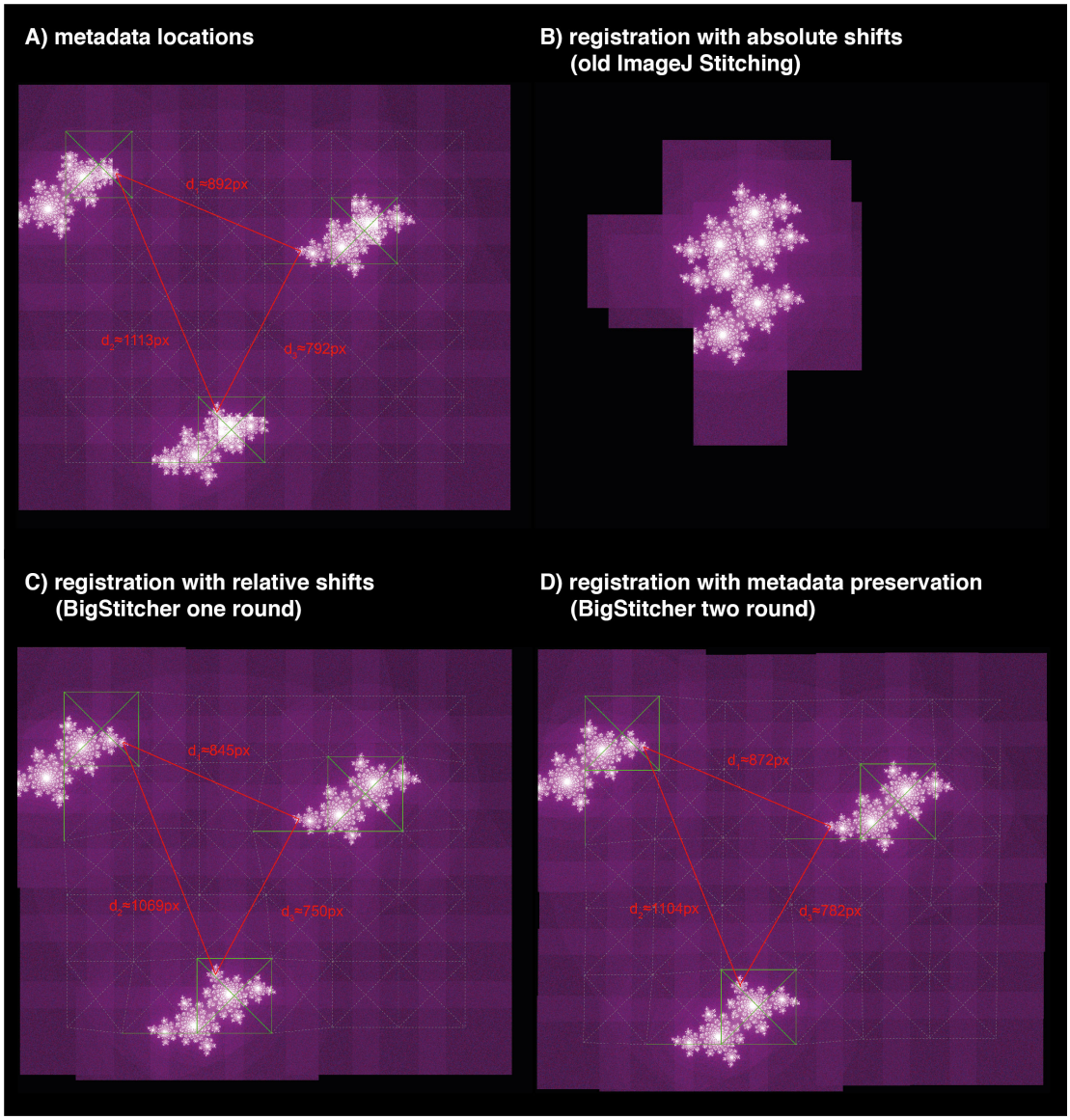

Supplementary Figure 3: *Interactive manual alignment of tiled images.* The BigStitcher GUI offers various ways of manually (pre-)aligning tiled images. Images can be moved to a regular grid with a given tile order and overlap **(A)**. Image locations can also be read from a simple *tile configuration* text file **(B)**. Furthermore, selected image(s) can be moved along axes via sliders **(C)**. All changes will be displayed in the BigDataViewer window immediately **(D)** for quick verification.

**SUPPLEMENTARY FIGURE 4: Flat-field correction**



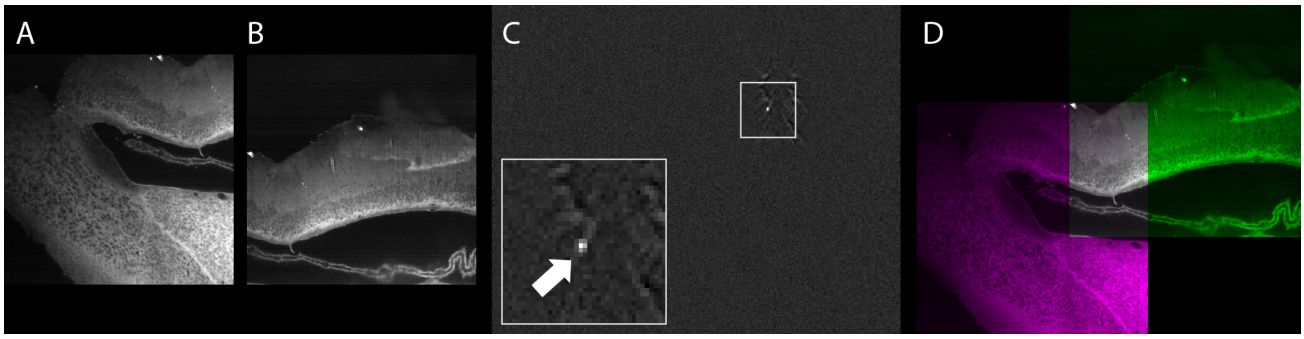

Supplementary Figure 4: *On-the-fly flat-field correction.* The BigStitcher offers correction for camera offsets, fixed pattern noise or uneven illumination. **(A)** Simulation of the effects of a constant background offset and Gaussian illumination/detection efficiency **(C)** on tiled images. By subtracting the *dark image* and modulating with the inverse relative intensity of the *bright image*, such artifacts can be corrected easily **(B)**. The correction is calculated virtually, with optional caching, to allow for immediate inspection of the results.

**SUPPLEMENTARY FIGURE 5: Global optimization**



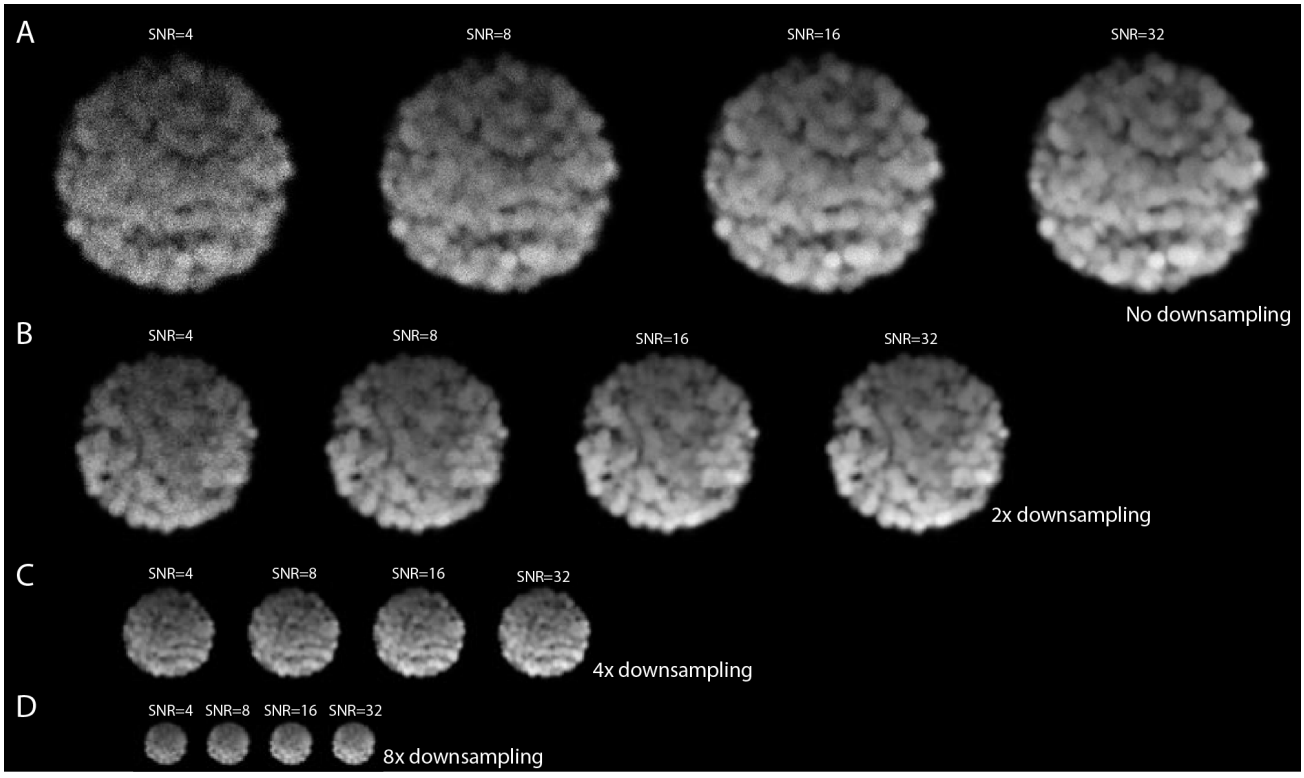

Supplementary Figure 5: *Global optimization of pairwise registration in sparse datasets connected by "empty tiles" (noise only)* . **(A)** Simulation of a tiled image dataset with sparse objects: tiled images of multiple translated Julia fractals moved to a grid according to approximate metadata (with too high overlap). Centers of images for which pairwise shifts can be determined via phase correlation are connected by green lines, whereas centers of neighboring tiles for which no meaningful shift can be calculated are linked by dashed grey lines. Manually measured distances between distinct points in the three fractals are shown in red. Performing global optimization with *absolute shifts* (as it is done BigStitcher's predecessor, the ImageJ Stitching plugin) will correctly align images within connected components of the link graph but place all fractals close to the origin **(B)**. By using *relative shifts*, BigStitcher will leave disconnected objects at their initial location while still aligning within connected components **(C)**. As registrations are not propagated between unconnected tiles, distances between neighboring objects might change. By running a second round of optimization to align connected components according to metadata shifts and applying the results to the in-component registrations, distances between neighboring objects are preserved as-good-as-possible **(D)**.

## SUPPLEMENTARY FIGURE 6: Pairwise registration by phase correlation



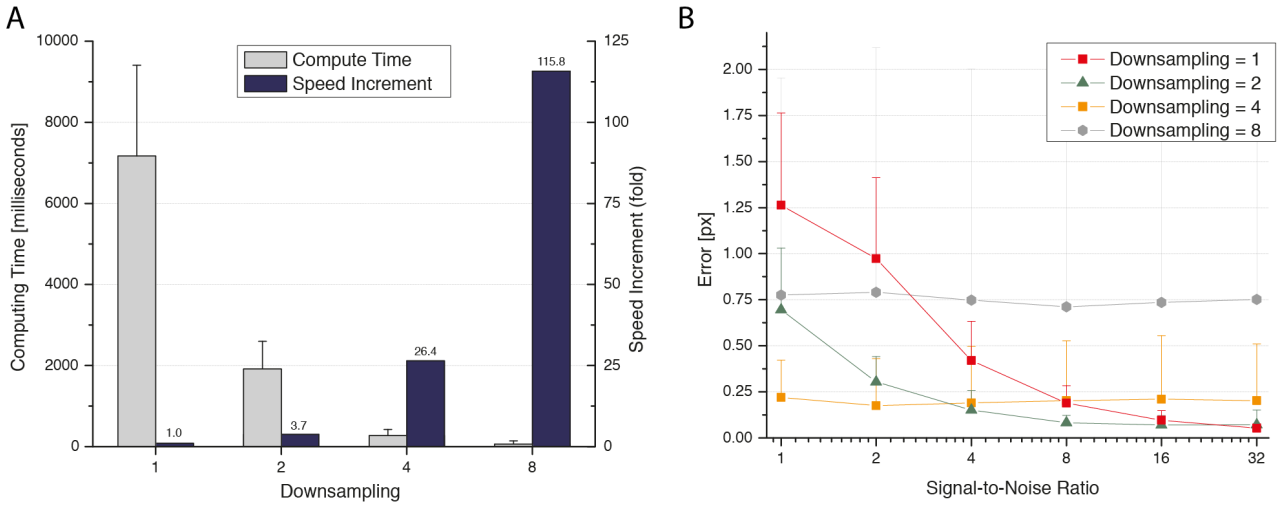

Supplementary Figure 6: *Pairwise registration by phase correlation*. **(A),(B)** Central slices of image stacks from a tiled acquisition (non-regular tiling) of an adult mouse hypothalamus slice. **(C)** Phase correlation matrix (PCM) calculated from the two images shows a single, distinct peak above nearly constant background. The peak location corresponds to the relative translation $t$ of both tiles. **(D)** Central slice through the images aligned according to $t$, as displayed in interactively during the reconstruction process.

## SUPPLEMENTARY FIGURE 7: Downsampling with different SNR



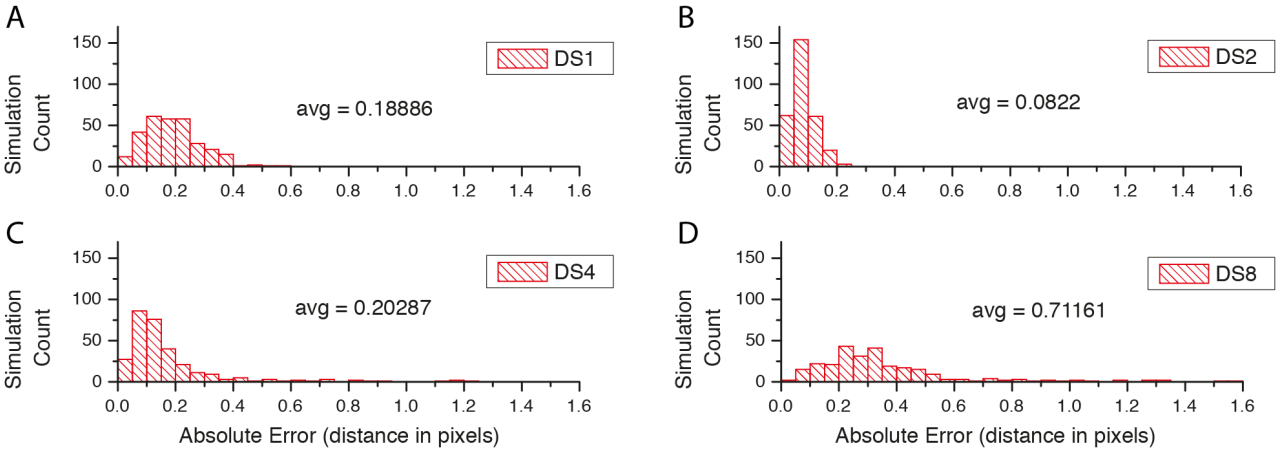

Supplementary Figure 7: *Effects of downsampling on simulated data with different SNR.* **(A)** Simulated image stacks of spheroid-like objects deteriorated by anisotropic sampling, light attenuation, convolution with an anisotropic PSF, and pixel intensity generation using Poisson processes to archive desired signal-to-noise-ratios (SNRs). A central slice through 3d volumes is shown. **(B), (C), (D)** Effects of downsampling on the simulated images. The effects of Poisson Shot Noise are gradually reduced by the blurring of increasing downsampling.

## SUPPLEMENTARY FIGURE 8: Downsampling statistics 1



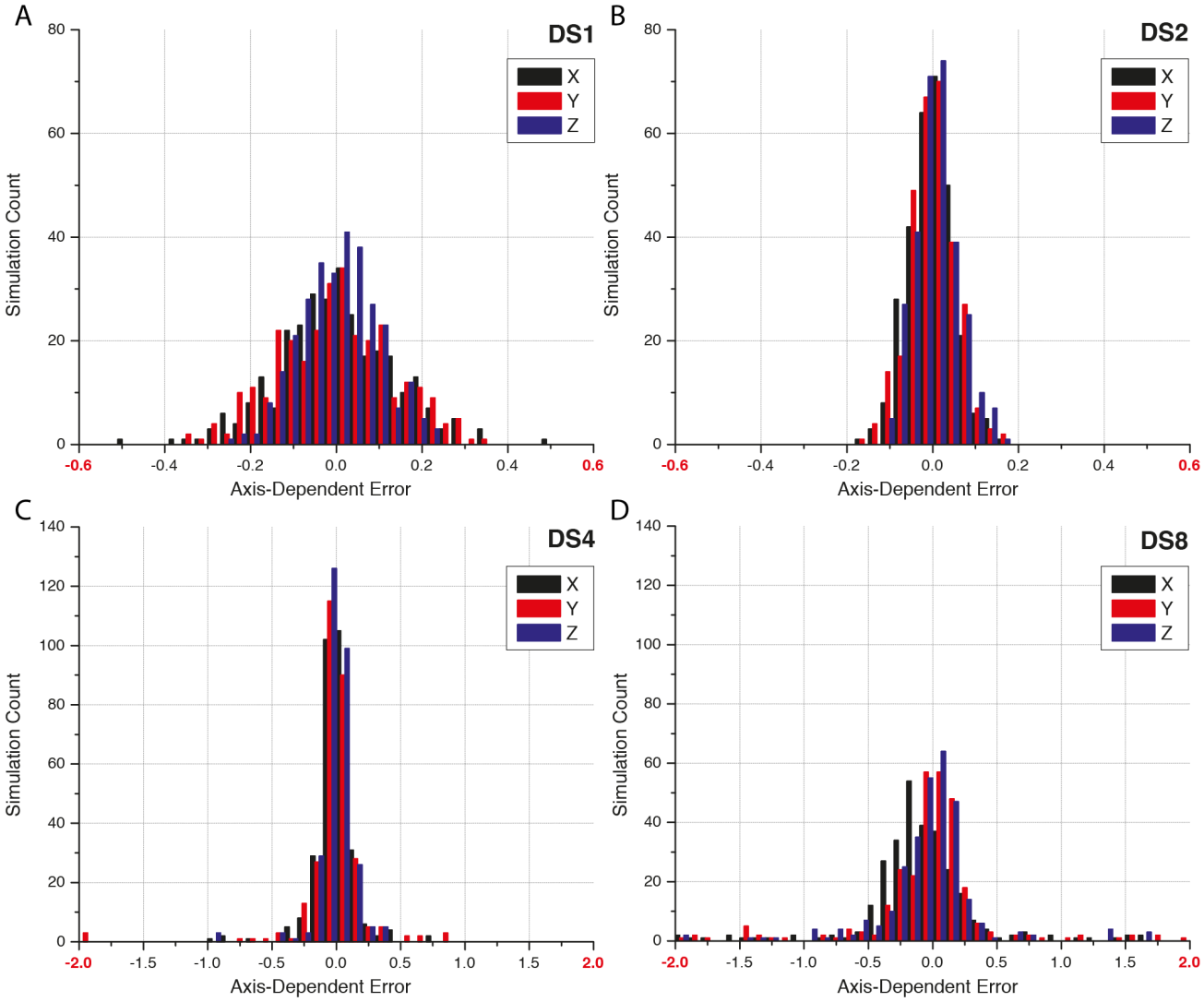

Supplementary Figure 8: *Processing times and overall errors.* **(A)** Processing times for sub-pixel precise identification of overlap between simulation spheroids. With increasing downsampling, the computation time drops significantly. **(B)** Average errors including their standard deviation for all combinations of SNR and downsampling. **(A,B)** All errors are in units of the input images (no downsampling). For each combination of SNR and downsampling 300 independent simulations were run to compute the values.

## SUPPLEMENTARY FIGURE 9: Downsampling statistics 2



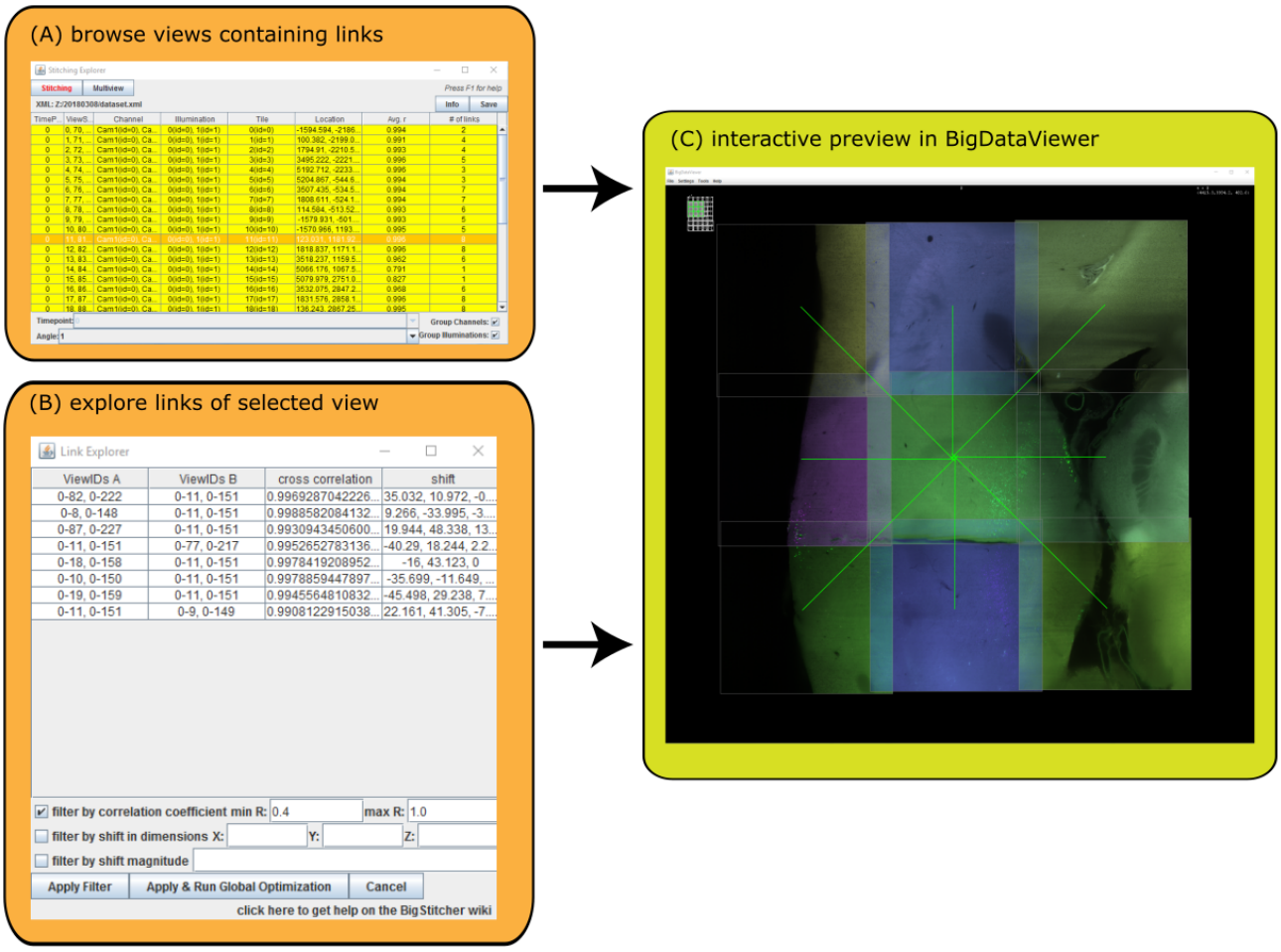

Supplementary Figure 9: *Errors for different downsamplings at SNR=8.* **(A-D)** Histograms showing the distributions of error of the simulations. Errors initially decrease due to the smoothing effect of the downsampling. All errors are in pixel units of the original resolution (DS1). Each histogram consists of 300 independent simulations.

**SUPPLEMENTARY FIGURE 10: Downsampling statistics 3**



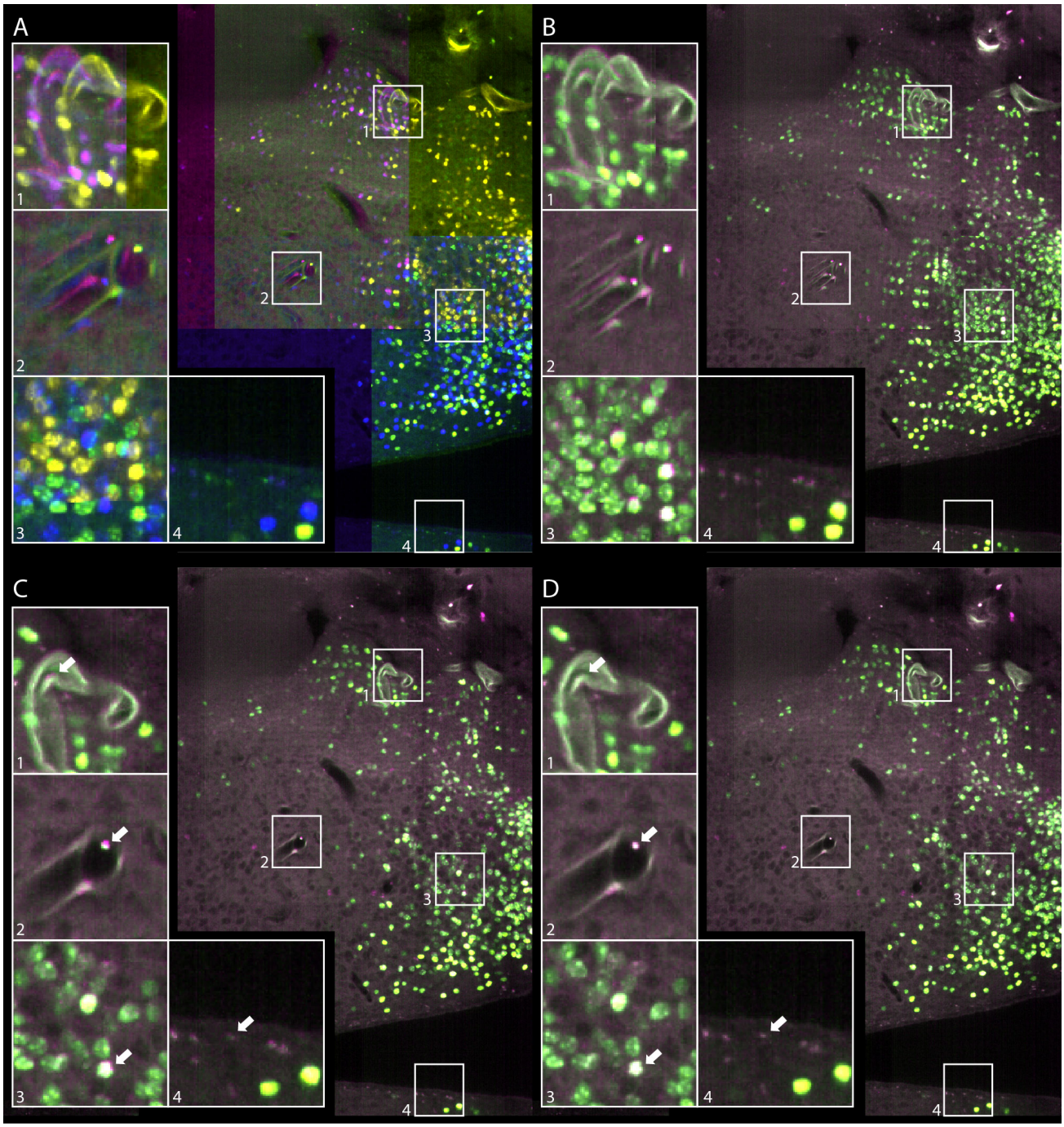

Supplementary Figure 10: *Absolute distance errors at SNR=8.* **(A-D)** Histograms showing the absolute distances between computed and know shift between two simulated spheroids, split by dimension. It illustrates a normal distribution of the error made during the pairwise phase correlation. All errors are in pixel units of the original resolution (DS1). Each histogram consists of 300 independent simulations.

**SUPPLEMENTARY FIGURE 11: Interactive inspection and curation of pairwise links**



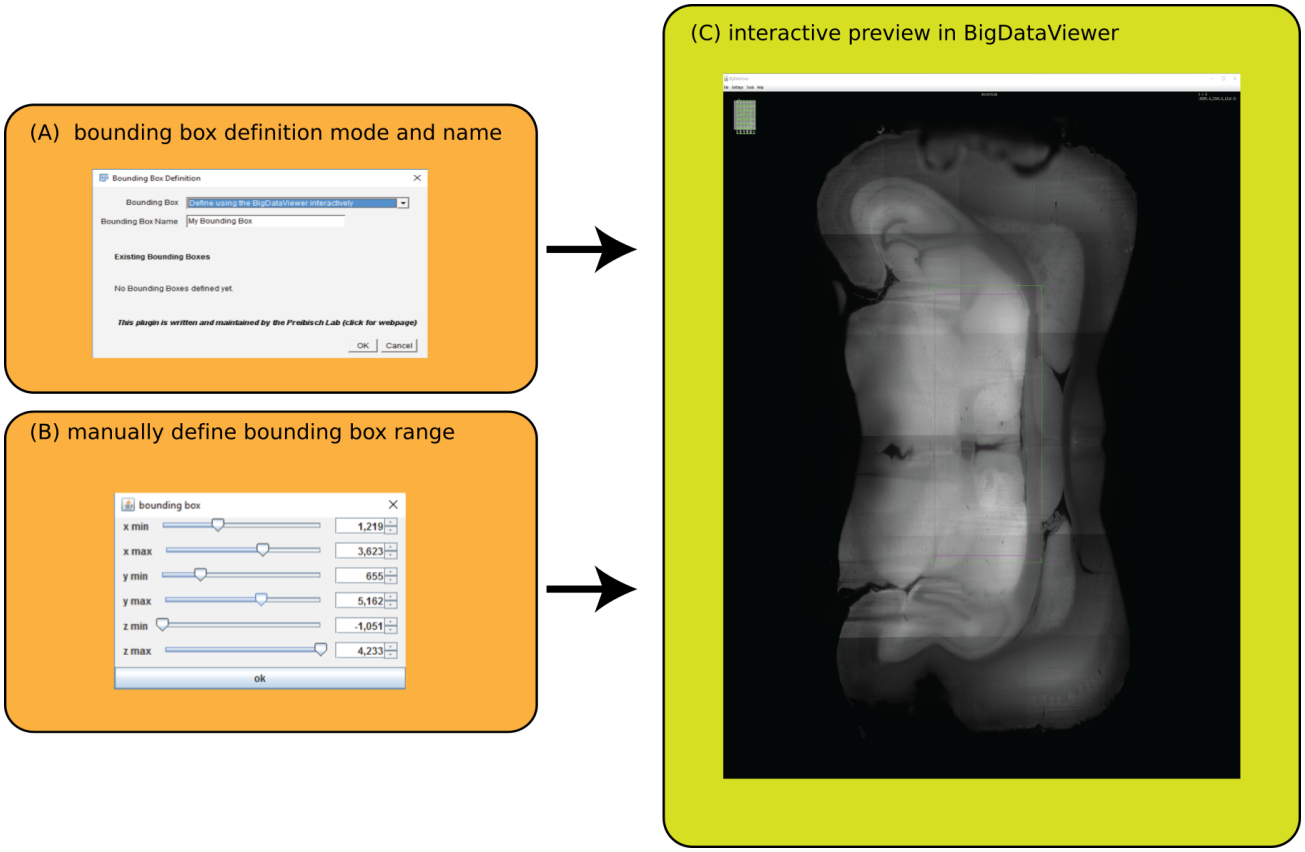

Supplementary Figure 11: *Interactive visualization of links in the link explorer.* The BigStitcher GUI offers the exploring and modifying calculated links between corresponding tiles in the link explorer menu. **(A)** Tiles containing links are displayed in yellow and can be selected. **(B)** Display corresponding tiles of the selected view. Single links can be removed manually or through available filtering options. **(C)** Corresponding links of the selected view are displayed in real-time in the BigDataViewer.

**SUPPLEMENTARY FIGURE 12: Affine refinement via ICP**



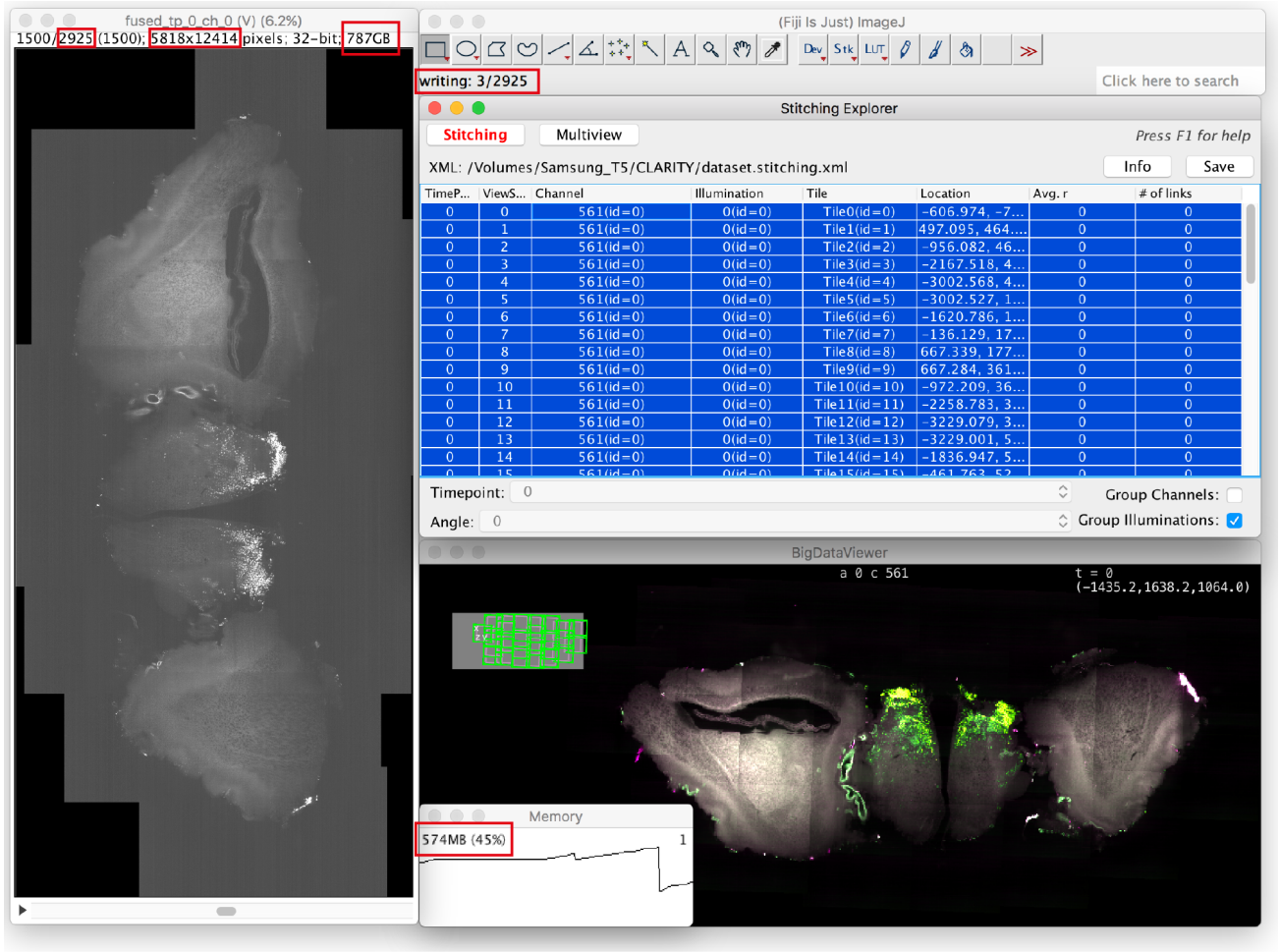

Supplementary Figure 12: *Illustration of different steps for multi-tile alignment* **(A)** Four overlapping image tiles randomly color-coded illustrate the typical error when only the metadata as read from the microscope is applied. **(B)** Shows the same four image tiles as in (A), but without random color coding. **(C)** Quality of the registration after applying the phase-correlation based stitching with downsampling 4 and two-round global optimization. **(D)** Result after applying the automatic ICP refinement for tile alignment and chromatic aberration correction. **(A-D)** Insets highlight specific areas to better appreciate quality differences.

**SUPPLEMENTARY FIGURE 13: Bounding Box definition**



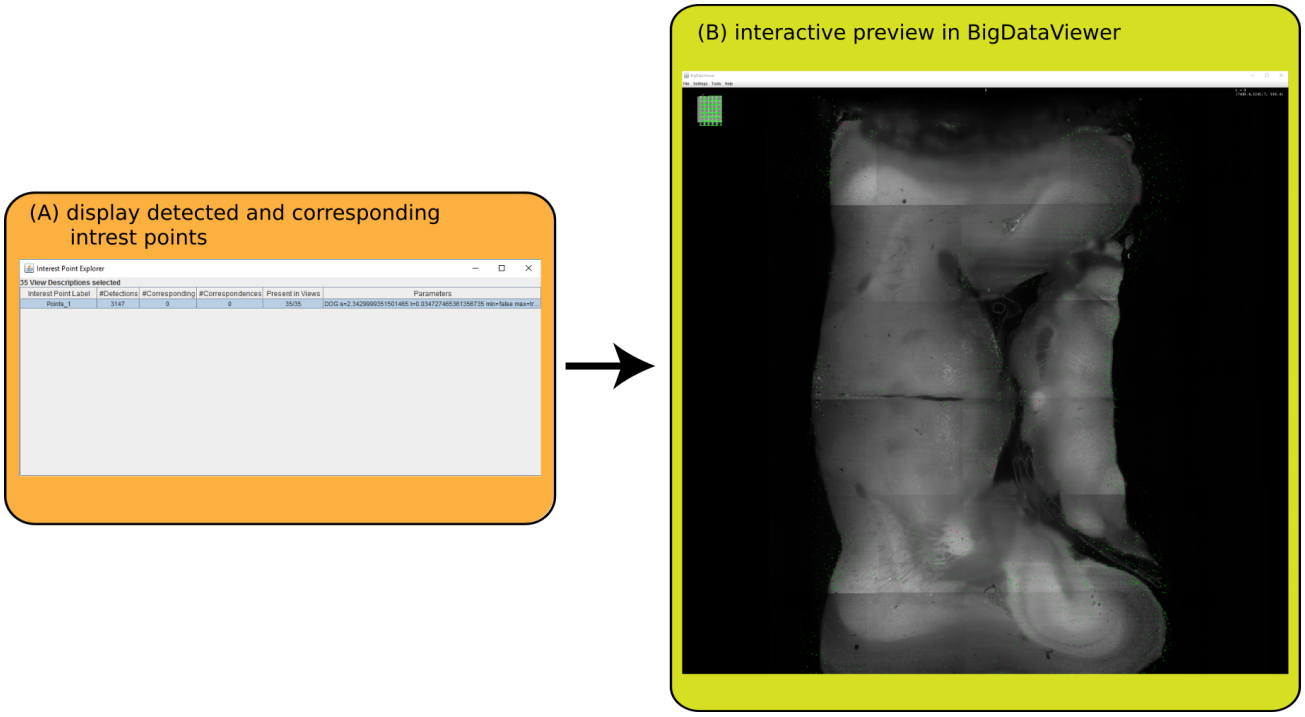

Supplementary Figure 13: *Interactive definition of bounding boxes.* The BigStitcher GUI offers the possibility of defining or modifying regions of interest via the creation of bounding boxes. **(A)** Choose the method used to define a new bounding box. In this case the interactive mode is selected. **(B)** manually define the bounding box range **(C)** Preview the size of the specified bounding box in the BigDataViewer in real-time.

## SUPPLEMENTARY FIGURE 14: Virtual Fusion of large image



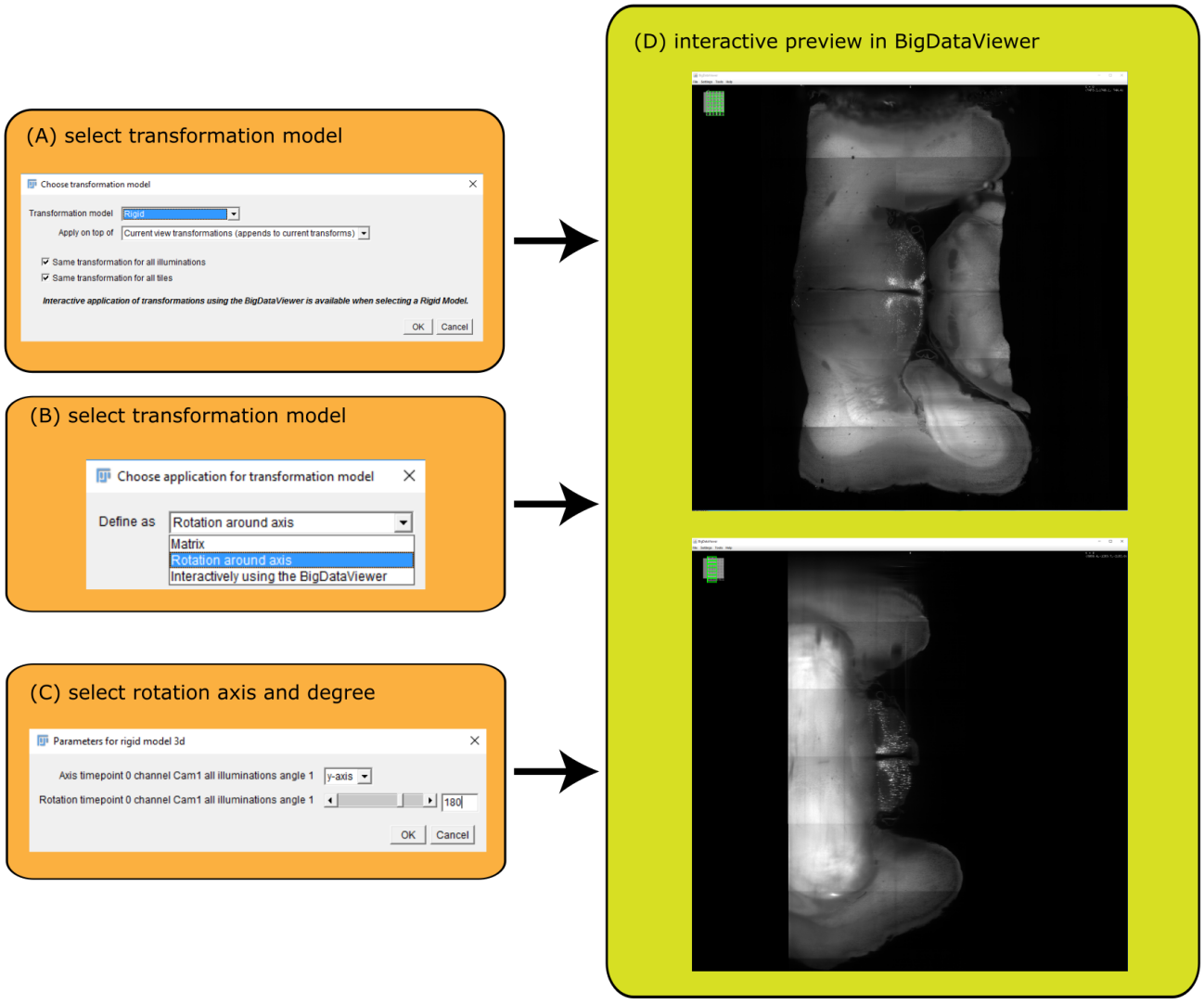

Supplementary Figure 14: *Virtual Fusion.* Screenshot of a Fiji instance running with **1.25GB of RAM** successfully fusing and saving a **787GB** volume 5818×12414×2925 pixels in size. This is achieved through **virtual fusion** combined with virtual, cached loading of blocked, multi-resolution input images. Red boxes highlight memory consumption, size, and progress. During the fusion process, the BigStitcher and BigDataViewer are interactively accessible.

**SUPPLEMENTARY FIGURE 15: Interest point visualization**



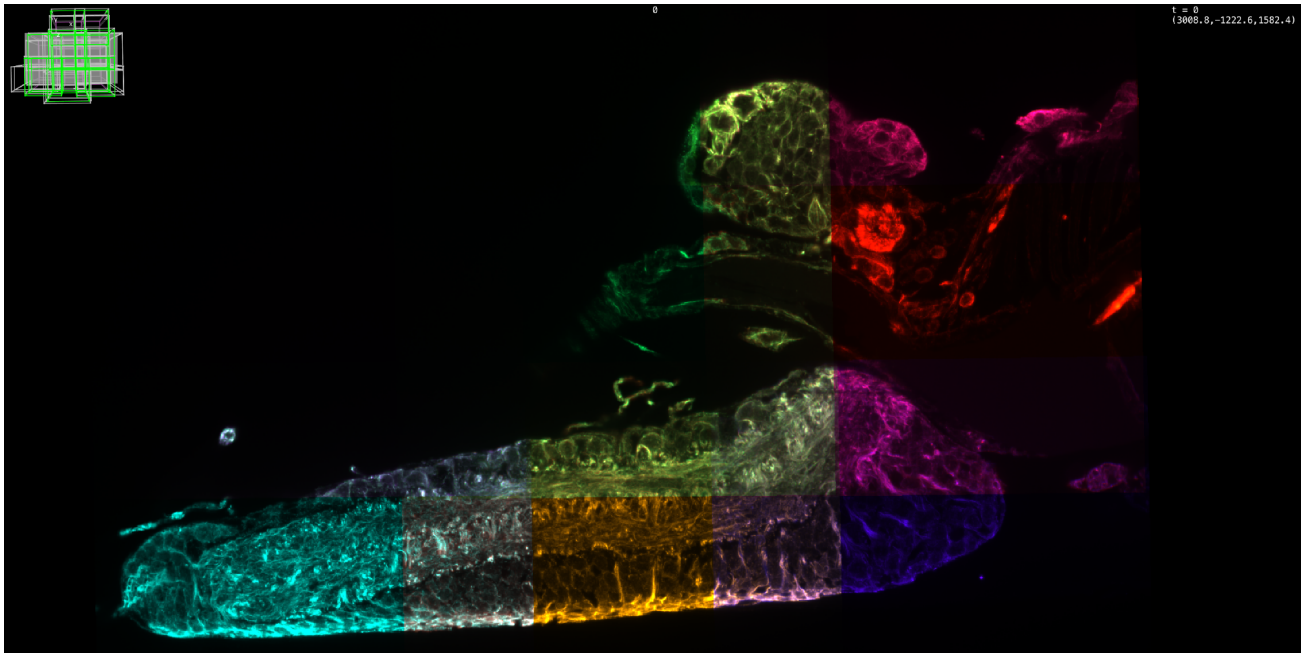

Supplementary Figure 15: *Interactive visualization of interest points.* The interest points explorer allows the visualization of interest points and corresponding interest points between views. **(A)** select desired interest points for visualization. **(B)** preview the interest points overlaid in the BigDataViewer.

**SUPPLEMENTARY FIGURE 16: Manual Transformation of multi-view datasets**




Supplementary Figure 16: *Interactive transformation of views.* Different transformation models can be applied to one or more views and simultaneously visualized in the BigDataViewer. **(A)** Choose transformation model grouping. **(B)** Further define the transformation model. In this case a rotation around the axis is selected. **(C)** Select rotation axis and angle. **(D)** Visualize rotation of the view in the BigDataViewer.

**SUPPLEMENTARY FIGURE 17: Expansion Microscopy Stitching**




Supplementary Figure 17: *Stitching of a single view of the expansion microscopy sample* All stitched tiles of one view of the expanded Drosophila central nervous system. Each tile is randomly colored to illustrate the overlap regions in between the stacks.

# SUPPLEMENTARY NOTE

## 1. Data Import

### `SpimData` data format

We internally represent our image data and metadata using an extended version of the `SpimData` data format of BigDataViewer.[2] Each image stack is defined by a (`ViewSetup`, `TimePoint`)-combination. We extend the format by giving each `ViewSetup` the following *attributes*: `Channel` to represent color channels, `Illumination` to represent illumination directions, `Angle` to represent multi-view acquisition angles and finally `Tile`, representing (local) x,y points in a multipoint acquisition.

In addition to those attributes, we store detected interest points, bounding boxes (named sub-volumes in which we can *fuse* or deconvolve images), point spread functions for deconvolution and pairwise registrations (that have yet to be used in global optimization) for each (`ViewSetup`, `TimePoint`) *view*. For each image stack, we also store its *registration* (i.e. the transformation from pixel to world coordinates) as a list of affine transform matrices. The registration steps described below will typically prepend another transformation matrix to this list. Finally, the `SpimData` is associated with an `ImgLoader` object that can make image pixel data available as an ImgLib2 `RandomAccessibleInterval` given a (`ViewSetup`, `TimePoint`) *view id*.

The `SpimData` data structure can be saved as an XML *project file*, allowing users to manually edit it with any text editor. We automatically save previous versions of the project file to provide the user with the ability to un-do registration steps.

### Import of data

BigStitcher imposes little constraints on the format and naming of raw data files. Using the Bioformats[3] library, we support a large variety of image file formats, from ”conventional” TIFF stacks to vendor-specific formats. The assignment of attributes to the image stacks in the raw data can be done automatically or with minimal interaction from the users. We offer the possibility to immediately compute multi-resolution pyramids from the images and saving them as chunks to HDF5 files. Furthermore, making use of imglib2-cache, we support virtual loading of image planes from the raw files with chaching of already loaded planes.

## 2. Flat-field correction

Flat-field correction is the process of correcting for image artifacts due to uneven illumination or detection efficiency or fixed-pattern noise. Aside from being visually unpleasing, especially in tiled acquisitions, these artifacts can also effect image registration and downstream quantitative image analyses. We therefore offer simple on-the-fly correction for a *dark image* (which might be nonzero due to e.g. camera offset) and a *bright image* (representing uneven ilumination or detection efficiency across the field-of-view). We calculate corrected pixel intensities $C$ from a raw image $R$ and bright and dark images $B$ and $D$ as:

$$C_x = \frac{(R_x - D_{x'}) * \overline{(B - D)}}{(B_{x'} - D_{x'})} \tag{1}$$

The correction images can either have the same dimensionality as the raw images, in which case $x' = x$, or have lower dimensionality (e.g. when using 2D correction images on a 3D image stack), in which case $x' = (x_1 \dots x_n)$ with $n$ being the dimensionality of the correction images. If a dark image is not provided by the user, we assume it to have constant intensity of 0 (corresponding to no background offset). Likewise, if no bright image is provided, we assume it to have constant intensity of 1 (uniform illumination and detection efficiency).

We implemented the flat-field correction as a wrapper around an `ImgLoader`, calculating corrected pixel intensity values on-the-fly (with optional caching) every time an image is loaded. That way, the corrected

images are available for all other processing steps such as intensity-based registration, interest point detection or image fusion, but it is still possible to actviate or de-activate the correction or change bright or dark images after the initial flat-field correction. A separate (bright, dark)-correction image pair can be set for every image in the dataset by modifying the XML project file, while in the GUI we offer user-friendly assignment of correction images to every (channel, illumination direction)-pair.

## 3. Pairwise shift calculation

In BigStitcher, we currently offer three ways of calculating shifts between a pair of images: the Fourier-based *phase correlation* algorithm, the Gradient-descent-based *Lucas-Kanade* algorithm, both intensity-based methods, as well as interest point-based alignment.

### Phase correlation

By default, we calculate pairwise translational shifts of two images $I_1$ and $I_2$ using phase correlation as explained in the **Online Methods** section.

### Lucas-Kanade

In addition to the default phase correlation-based pairwise shift calculation, we offer registration via an ImgLib2 implementation of the *inverse compositional* formulation of the gradient descent-based Lucas-Kande optical flow algorithm.[4] While the algorithm is applicable to a variety of transformation models, we currently stick to estimating a translation vector $t$. If the pairwise registration converges, we calculate the cross correlation of the overlapping portions of the images as a quality metric for the pairwise registration.

### Intensity-based Registration of grouped images

In many use cases, one might want to align not single images but groups of images, e.g. all channels of a tile, in the pairwise registration step. For this, we implemented a flexible framework for the registration of grouped images.

Each attribute of the images can be set to be an *axis of application*, an *axis of comparison* or an *axis of grouping*. The registration will proceed by first splitting the images by the application attributes, i.e. grouping all images that have the same value for these attributes. In each each group, the images are then split by the comparison attributes and finally, the remaining image groups (that differ only in the grouping attributes) are combined into one image stack by either averaging all images for each grouping attribute or picking the image with a specific instance of the attribute.

In a typical application, the stitching of tiled datasets, we would, for example, start by *applying* the registration to all (`Angle`, `TimePoint`)-combinations individually, *comparing* by `Tiles` and finally *grouping* by `Illumination` and `Channel` for each tile, e.g. by averaging illumination directions and picking a specific channel.

### Intensity-based registration of images with pre-registrations

The two images $I_1$ and $I_2$ can have arbitrary pre-registrations, i.e. pixel coordinates $x_{px}$ are mapped to world coordinates $x_w$ via the affine transforms $x_{w,I_1} = A_{I_1}x_{px,I_1} + b_{I_1}$ and $x_{w,I_2} = A_{I_2}x_{px,I_2} + b_{I_2}$. Depending on the values of $A_{I_1}$ and $A_{I_2}$, we consider two cases: If they are equal, i.e. the pre-registrations differ only by a translation, we perform the shift calculation on the raw pixel data of the overlapping volume to get a shift vector $t$ for $I_2$ in pixel coordinates. The transformation in world coordinates is then given by $R\left(\begin{smallmatrix} I & \\ 0\ \cdots\ 0 & t \\ & 1 \end{smallmatrix}\right)R^{-1}$ with $R = \left(\begin{smallmatrix} A_{I_2} & \\ 0\ \cdots\ 0 & b_{I_2} \\ & 1 \end{smallmatrix}\right)$. If the pre-registrations differ in more than just translation, we create virtually transformed images of the smallest rectangular bounding box enclosing the overlapping volume and use them as input to the registration. As the virtual input images are already in world coordinates in this case, the resulting transformation matrix for $I_2$ is simply $\left(\begin{smallmatrix} I & \\ 0\ \cdots\ 0 & t \\ & 1 \end{smallmatrix}\right)$

### Interest-point based

For interest-point based pairwise registration, we detect local extrema in either Difference-of-Gaussian or Difference-of-mean filtered images, optionally followed by subpixel refinement of the detections via a quadratic fit. If we are registering a pair of image *groups*, the interest points of each image in the group are pooled, with optional replacement of point clusters within a user-defined radius by their center.

For each image, we apply the current (affine) registrations to the pixel-coordinate interest points and then determine *candidate point matches* via descriptor matching. We then perform model-based outlier removal via the RANSAC algorithm, yielding a set of *inlier point pairs*, $C_{inliers}$, and an optimal translation $t$ for $I_2$, minimizing $\sum_{(ip_1,ip_2)\in C_{inliers}} ||ip_1 - ip_2 - t||^2$

## 4. Global optimization

### Estimation of globally optimal transformations

The pairwise registration step results in *links* between image (groups) $V$ (note that sice we do not use the actual image *content* in the global optimization, we will refer to the images by their integer *id* in this section: $V \subset \mathbb{N}$). The links can be either in the form of pairwise transformations $T^p$ (such that coordinates $x$ from two images $V_i$ and $V_j$ can be transformed according to $T^p_{ij}(x_j) = x_i$) or *point correspondences* $PM$ from which such transformations can be estimated. The pairwise registrations thus form a *link graph* $(V, C)$ with edges $C = \{(i, j) \in V \times V | T^p_{ij} \in T^p\}$ between image pairs for which we could determine pairwise transformations. Simply traversing a spanning tree of the link graph and propagating the pairwise transformations can lead to the compounding of pairwise registration errors, even if the traversal is done along a *minimal* spanning tree determined according to some quality metric $q_{ij}$, e.g. cross-correlation, of the pairwise registrations.

We thus make use of an algorithm for globally optimal registration by iterative minimization of square displacement of point correspondences[5] for reaching a reasonable consensus in this case. This point match-based framework allows for flexible groping and fixing of images, is applicable to, among others, time series-, chromatic channel- or view-registration and can easily be adapted to incorporate the pairwise transformations from e.g. phase correlation. The algorithm is agnostic of the transformation model (e.g. translation, affine transform,...), with the only requirement being that the model parameters can be estimated by a least-squares fit from point correspondences.

We determine the globally optimal registrations $R$ given the image (groups) $V$, pairwise links $C$, pairwise $n$-dimensional point matches $PM$ with $PM_{ij} \subset \mathbb{R}^n \times \mathbb{R}^n$ and a set of fixed views $F \subseteq V$ by minimizing:

$$\underset{R\setminus\{R_i|V_i\in F\}}{\arg\min} \sum_{(i,j)\in C} \left( \sum_{(x_k,y_k)\in PM_{ij}} ||R_i(x_k) - R_j(y_k)||^2 \right) \tag{2}$$

Note that for all fixed views, the registration will be constrained to be the indentity transformation $I$: $\forall V_i \in F : R_i = I$.

### Global optimization given pairwise transfomations

The intensity-based pairwise shift calculations do not directly give us the point correspondences we need for the global optimization step, instead the results are pairwise transformations $T^p$ in the form of affine transform matrices. We can, however, easily construct point correspondences by taking a set of points and transforming them with the *inverse* transform (the only requirement being that the $n$-dimensional points do not all lie in a subspace of lower dimensionality of $\mathbb{R}^n$).

Using the 3-dimensional pairwise transformations $T^p$ ($T^p_{ij}(x_j) = x_i$) between two image (groups) $V_i$ and $V_j$ given their existing registrations $R^{meta}$, we use the 8-point approximate bounding box of their overlapping region $BB_{ij}$ to construct the point correspondences: $PM_{ij} = \{\big(bb_k, (T^p_{ij})^{-1}(bb_k)\big) | bb_k \in BB_{ij}\}$. We can then determine the gobally optimal registrations $R$ by performing the minimization described above (2).

### Global optimization with iterative link dropping

Once the global optimization terminates due to convergence or exceeding of the maximium number of iterations, we can calculate the *error* of the individual images as the average displacement of all interest points in an image to their point matches:

$$e_i = \frac{\sum_{\{j:(i,j)\in C\}} \sum_{(x_k,y_k)\in PM_{ij}} ||R_i(x_k) - R_j(y_k)||}{\sum_{\{j:(i,j)\in C\}} |PM_{ij}|} \tag{3}$$

If the link graph $(V, C^n)$ contains links with contradicting point correspondences, stopping after one round of global optimization might leave us with unsatisfying results. In the *iterative* version of the global optimization, we therefore check that both the average error of all images and the ratio of maximal and average error fall below a user-defined threshold. If these condiditions are not yet met, we will proceed to iteratively remove disagreeing links from the link graph and repeat the global optimization. To do this, we first determine the link with the highest error by maximizing:

$$c_{worst} = \arg\max_{(i,j)} \max_{(x_k,y_k)\in PM_{ij}} \left( (1-q_{ij})^2 \sqrt{d_{ijk}} \log_{10} \left( \max \left( deg(i), \deg(j) \right) \right) \right) \tag{4}$$

with $d_{ijk}$ denoting the distance of the $k$'th point match of the link $(i,j)$, $d_{ijk} = ||R_i(x_k) - R_j(y_k)||$, $\deg(i)$ denoting the degree (number of neighbors) of an image $V_i$ in the link graph and $q_{ij}$ being a *quality metric* $\in (0,1)$ of the link, e.g. 0-truncated cross correlation. We then remove the worst link from the links ($C^{n+1} \leftarrow C^n \setminus c_{worst}$) and repeat the optimization step 2 with the new link graph $(V, C^{n+1})$. The whole process is repeated until the errors fall below a user-defined threshold (in the worst case, links will be dropped until we end up with *spanning trees* of the connected components in the link graph).

### Two-round global optimization using metadata

If some cases, the link graph might contain multiple connected components, e.g. in datasets from screening applications, where the actual sample only occupies isolated "islands" and most images contain only background. In this case, we can only realibly determine pairwise transformations within the connected components and align images within the components in the global optimization step. We might, however, have resonable registrations $R^{meta}$ from metadata and wish to keep as closely as possible to those if we do not have *stong* links.

For this, we offer a *two-round* version of the global optimization. In the first round, we determine registrations $R^{strong}$ as described above, using the graph of *strong* links, i.e. links that are backed by pairwise transformations. In the second round, we determine the connected components in the $(V, C^{strong})$ graph and a mapping $CC : \mathbb{N} \to \mathbb{N}$ from image (group) indices to connected component indices as well as *weak links* $C^{weak} = \{(i,j) \in V \times V | CC(i) \neq CC(j)\}$ between images in different components. We then determine transformations $R^{cc}$ for each connected component not containing a fixed image by minimizing:

$$\arg\min_{R^{cc}\setminus\{r_i^{cc}\in R^{cc}|CC_i\cap F\neq\emptyset\}} \sum_{(i,j)\in C^{weak}} \sum_{bb_k\in BB_{ij}} ||R^{cc}_{CC(i)}\left(R_i^{strong}(bb_k)\right) - R^{cc}_{CC(j)}\left(R_j^{strong}(bb_k)\right)||^2 \tag{5}$$

Note that we use the corners $bb_k$ of the bounding box $BB_{ij}$ of the overlapping volume of two images $V_i$ and $V_j$ as the point correspondences. The overlap is determined accoring to the metadata transformations $R^{meta}$ and we essentially try to "un-do" the registrations of the first round as well as possible (while keeping the registrations *within* the connected components). The final transformations $R$ are the concatenation of the registrations within the connected components with the relative transformations of the connected components: $R_i \leftarrow R^{cc}_{CC(i)} R_i^{strong}$.

## 5. MultiView Registration

For MultiView registration, e.g. registration of angles or time series stablilization, we first detect interest points in the individual images as described above (3). Images may be grouped (and are by default if we are, e.g. registering tiled acquisitions from multiple angles for which we already aligned the tiles via an intensity-based method) according to their attributes, by pooling their interest points and optionally merging clusters. For registering time-series data, we offer four strategies. First, we can treat time points individually, registering only images within a time point. We can also perform interest point matching between different time points, either comparing all-to-all, all to a user-defined *reference* time point or all time points within a defined range to each other.

Pairwise point correspondences can either be established by descriptor matching followed by RANSAC outlier removal, a modified version of the iterative closest point (ICP) algorithm or by simply matching the center of mass of the point clouds of both images (note that in this case the registration will be constrained to be a translation). Using the link graph $(V, C)$ and pairwise point correspondences $P_{ij}$ established thus, we calculate the final registration by performing global optimization as described above (4), optionally with iterative link removal and a second round to preserve metadata.

## 6. Image Fusion

We *fuse* multiple images by performing a weighted average of the raw images $I^{raw}$ transformed by their registrations $R$. Each raw image $I_i^{raw}$ has a set weight images $W_i$. For example, we allow the user to weigh the images with a cosine-shaped fade-out, deemphasizing the artifact-prone border regions of the individual images, as well as by the approximate local entropy, to emphasize images with sharper structures in overlapping regions. Since the raw images will be evaluated at non-integer coordinates, we offer the choice between nearest-neighbour and linear interpolation. Downsampling can easily be achieved by prepending a scaling transformation to each of the registrations $R$. The intensity of the fused volume at a coordinate $x$ is given by:

$$I^{fused}(x) = \frac{\sum_{I_i^{raw} \in I^{raw}} \left( I_i^{raw}\left(R_i^{-1}(x)\right) * \prod_{w_j \in W_i} w_j\left(R_i^{-1}(x)\right)\right)}{\sum_{I_i^{raw} \in I^{raw}} \left( \prod_{w_j \in W_i} w_j\left(R_i^{-1}(x)\right)\right)} \tag{6}$$

In practice, we evalute $I^{fused}$ only at integer coordinates of a user-defined *bounding box*. We implemented the image fusion to perform all calculations virtually on-the-fly, with caching of previously computed planes using imglib2-cache. This allows the quick inspection of fusion results as well as creation and planewise saving of images that might exceed the RAM available to the user.

## 7. Brightness and Contrast adjustment

Even after correcting for fixed-pattern noise (2), differences in brightness and contrast between images, e.g. due to bleaching, might persist and be visible in the fused images. To correct for this, we estimate optimal linear transforms of pixel intensities in adjacent images[6] to achieve uniform brightness and contrast in the whole dataset. We minimize the intensity difference of all pixels in the overlapping volume $O_{AB}$ of two images $I_A, I_B$ (with corresponding coordinates $(x_A, x_B)$ according to the current registrations):

$$\underset{\alpha,\beta}{\arg\min} \sum_{I_A \in I} \left( \sum_{I_B \in I \setminus I_A} \left( \sum_{(x_A, x_B) \in O_{AB}} \left( I_B(x_B) - \left[ \alpha^{I_A} I_A(x_A) + \beta^{I_A} \right] \right)^2 \right) \right) \tag{7}$$

Since this is equal to one-dimensional point correspondence registration, we can make use of the same iterative optimization algorithm used for image registration (4). To reduce influence of noise and computational costs, we use (precomputed) downsampled versions of the images for the optimization. A problem with unconstrained optimization is the possibility of convergence to the trivial solution of setting all pixel intensities to zero.

We therefore formulate the linear transform $I(x) * \alpha + \beta$ as a weighted average between a linear transform, an additive transform and the identity transform:

$$\alpha I(x) + \beta = \lambda_1 * (\alpha' I(x) + \beta_1) + \lambda_2 * (I(x) + \beta_2) + \lambda_3 * I(x) \tag{8}$$

with user-definable regularization parameters $\lambda_1, \lambda_2, \lambda_3 : \lambda_1 + \lambda_2 + \lambda_3 = 1$. By using nonzero $\lambda_2, \lambda_3$, we can constrain the optimization to not converge to the trivial solution.

## 8. BigStitcher User Guide

The BigStitcher comes with extensive documentation that is hosted on the ImageJ wiki. The current version of the continuously updated user guide can be found at https://imagej.net/BigStitcher#Documentation.

## 9. Links to the current source codes

The BigStitcher is distributed over two projects. Both are licensed under the GPL(v2) and the source code is freely available on GitHub, at https://github.com/PreibischLab/BigStitcher and https://github.com/PreibischLab/multiview-reconstruction, respectively.

The CUDA code for accelerated interest point detection and devonvolution is available from https://github.com/StephanPreibisch/SeparableConvolutionCUDALib and https://github.com/StephanPreibisch/FourierConvolutionCUDALib, respectively.

Newer versions will be hosted using GitHub, and release announcements will be done via Twitter (https://twitter.com/preibischs), on the GitHub page (https://github.com/PreibischLab/BigStitcher), and on the ImageJ wiki (http://imagej.net/BigStitcher). Releases are and will be provided to end users via the Fiji update mechanism. The current version of the user guide (8) will be hosted on the same wiki page (http://imagej.net/BigStitcher#Documentation).

## REFERENCES


1. I. Arganda-Carreras, V. Kaynig, C. Rueden, K. W. Eliceiri, J. Schindelin, A. Cardona, and H. Sebastian Seung, "Trainable Weka Segmentation: a machine learning tool for microscopy pixel classification," *Bioinformatics* **33**(15), pp. 2424–2426, 2017.
2. T. Pietzsch, S. Saalfeld, S. Preibisch, and P. Tomancak, "BigDataViewer: visualization and processing for large image data sets," *Nature methods* **12**(6), p. 481, 2015.
3. M. Linkert, C. T. Rueden, C. Allan, J.-M. Burel, W. Moore, A. Patterson, B. Loranger, J. Moore, C. Neves, D. MacDonald, *et al.*, "Metadata matters: access to image data in the real world," *The Journal of cell biology* **189**(5), pp. 777–782, 2010.
4. S. Baker and I. Matthews, "Lucas-Kanade 20 years on: A unifying framework," *International journal of computer vision* **56**(3), pp. 221–255, 2004.
5. S. Saalfeld, A. Cardona, V. Hartenstein, and P. Tomančák, "As-rigid-as-possible mosaicking and serial section registration of large ssTEM datasets," *Bioinformatics* **26**(12), pp. i57–i63, 2010.
6. C. Blasse, S. Saalfeld, R. Etournay, A. Sagner, S. Eaton, and E. W. Myers, "PreMosa: extracting 2D surfaces from 3D microscopy mosaics," *Bioinformatics* **33**(16), pp. 2563–2569, 2017.