BigStitcher: Efficient alignment of large multi-tile and multi-view image datasets

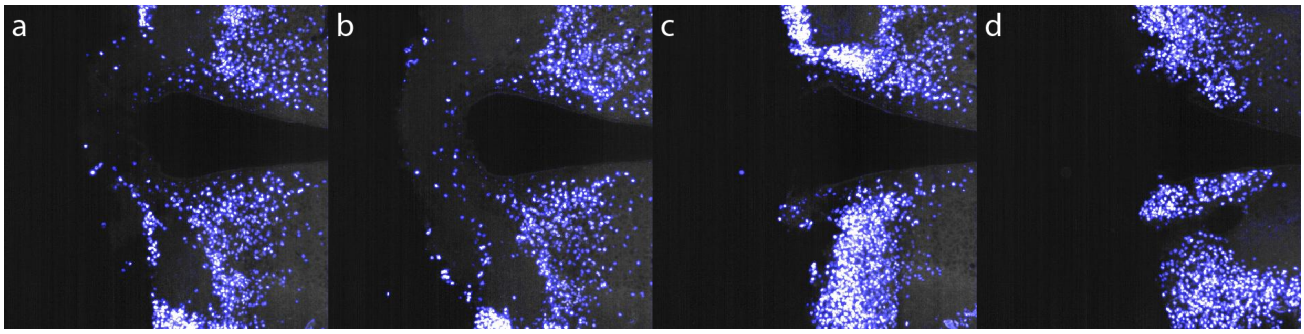
David Hörl, Fabio Rojas Rusak, Stephan Preibisch

Supplementary File	Title
Supplementary Figure 1	Downsampling error statistics
Supplementary Figure 2	Per-axis downsampling error statistics
Supplementary Figure 3	Downsampling with different SNR
Supplementary Figure 4	Example stitching
Supplementary Figure 5	Downsampling error statistics
Supplementary Figure 6	Downsampling error statistics
Supplementary Figure 7	Downsampling error statistics 1
Supplementary Figure 8	Downsampling error statistics 2
Supplementary Figure 9	Downsampling error statistics
Supplementary Figure 10	Downsampling error statistics
Supplementary Figure 11	Downsampling error statistics

Note: Supplementary Videos 1–XXX are available for download on the journal homepage.

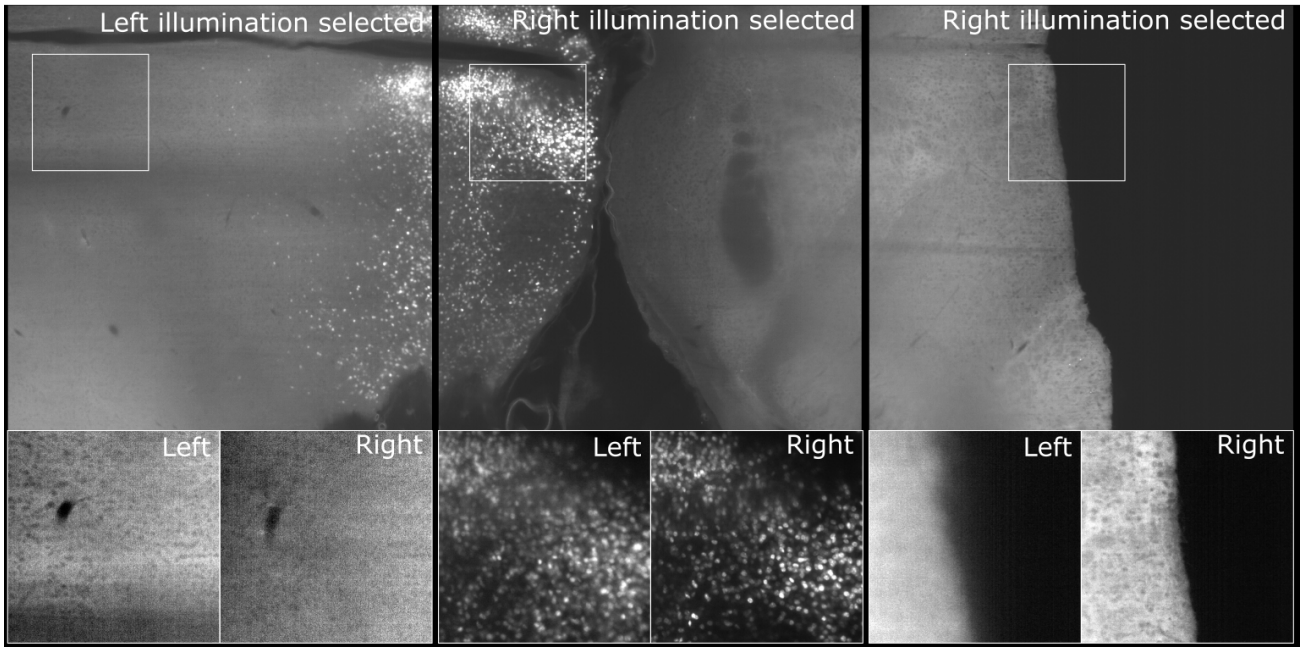
SUPPLEMENTARY FIGURES

SUPPLEMENTARY FIGURE 1: Fluorescence in cleared sample??



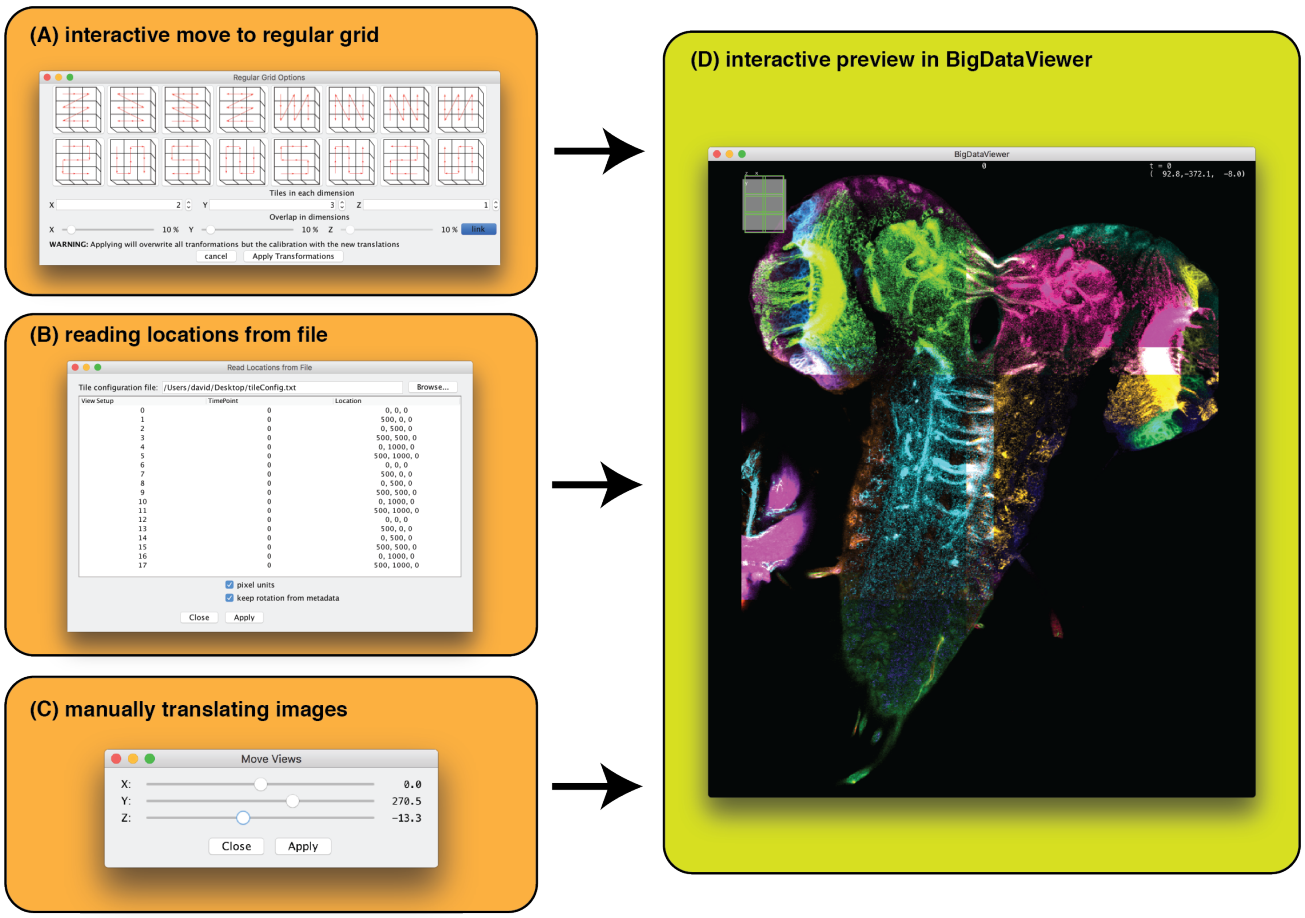
Supplementary Figure 1: *Lorem ipsum.* Dolor sit amet.

SUPPLEMENTARY FIGURE 2: Illumination selection



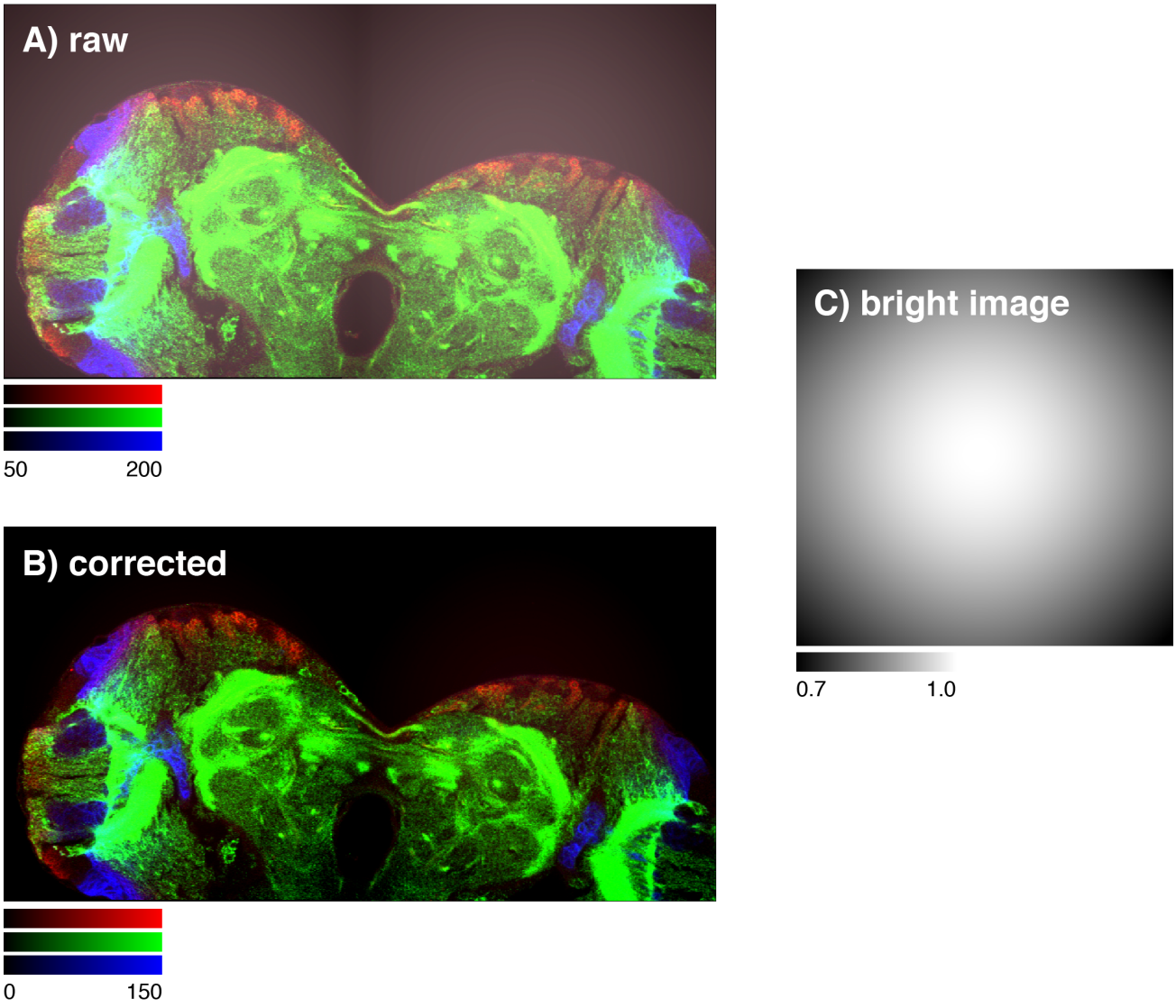
Supplementary Figure 2: *Lorem ipsum.* Dolor sit amet.

SUPPLEMENTARY FIGURE 3: Manual alignment



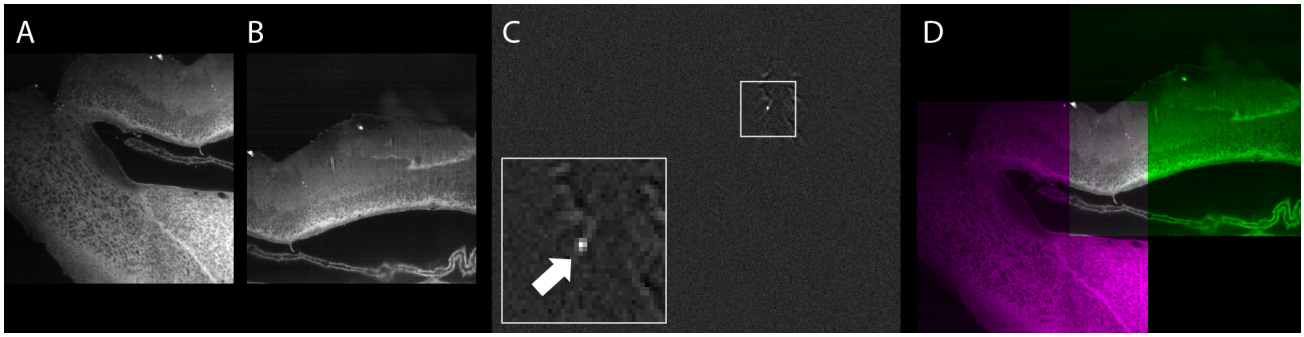
Supplementary Figure 3: *Interactive manual alignment of tiled images.* The BigStitcher GUI offers various ways of manually (pre-)aligning tiled images. Images can be moved to a regular grid with a given tile order and overlap **(A)**. Image locations can also be read from a simple *tile configuration* 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: Flatfield correction



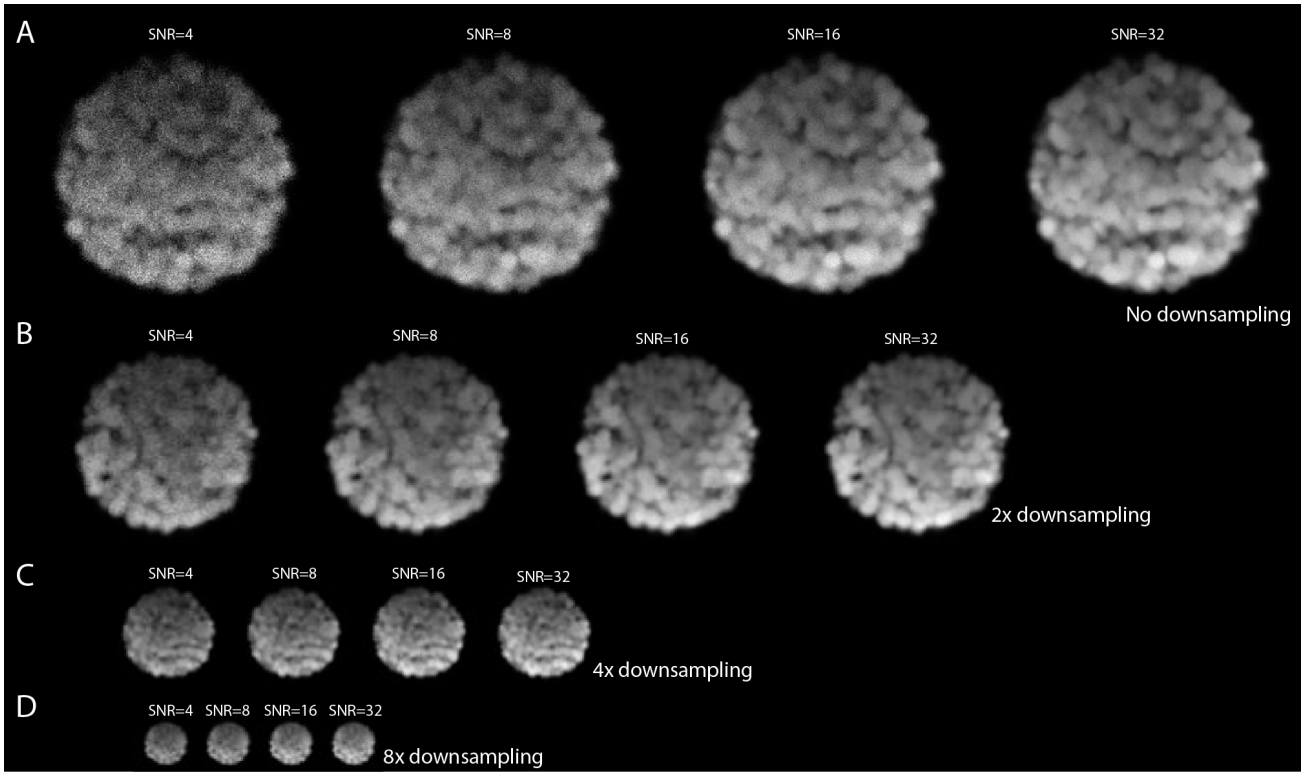
Supplementary Figure 4: *On-the-fly flat-field correction.* The BigStitcher offers correction for camera offsets, fixed pattern noise or uneven illumination. **(A):** Simulation of the effects of a constant background offset and Gaussian illumination/detection efficiency **(C)** on tiled images. By subtracting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

SUPPLEMENTARY FIGURE 5: Pairwise registration by phase correlation



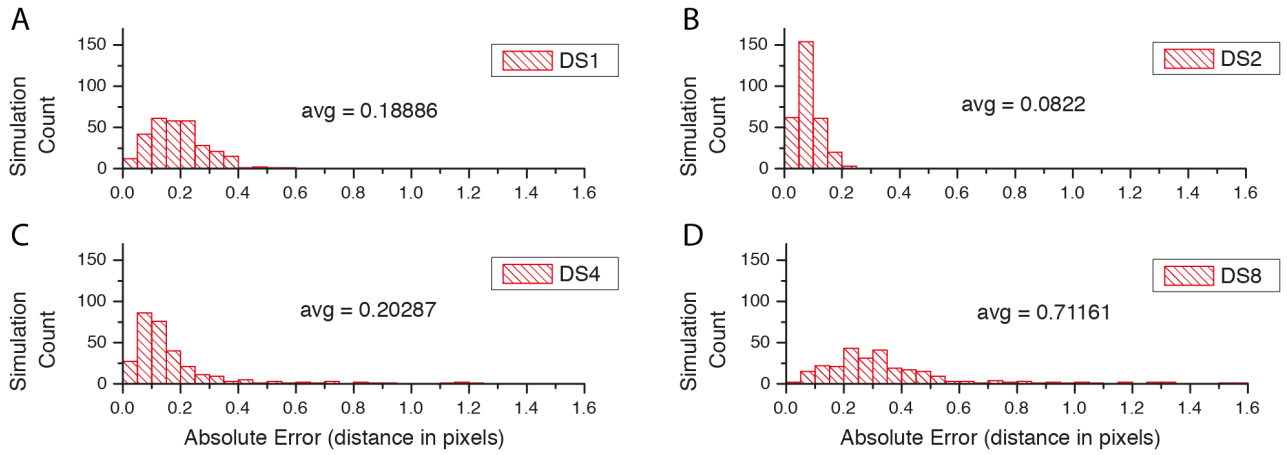
Supplementary Figure 5: *Lorem ipsum.* Dolor sit amet.

SUPPLEMENTARY FIGURE 6: Downsampling with different SNR



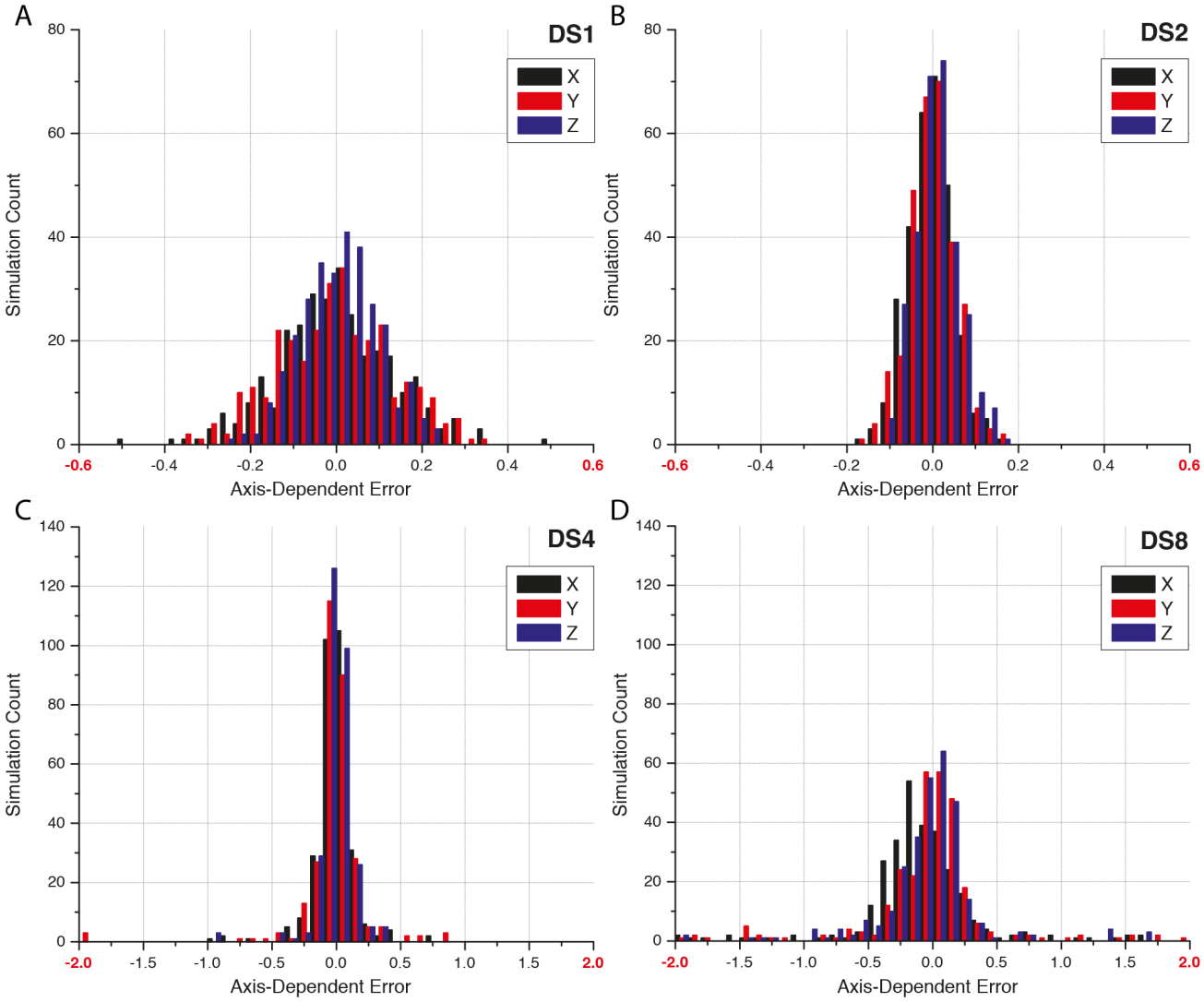
Supplementary Figure 6: *Lorem ipsum.* Dolor sit amet.

SUPPLEMENTARY FIGURE 7: Downsampling statistics 1



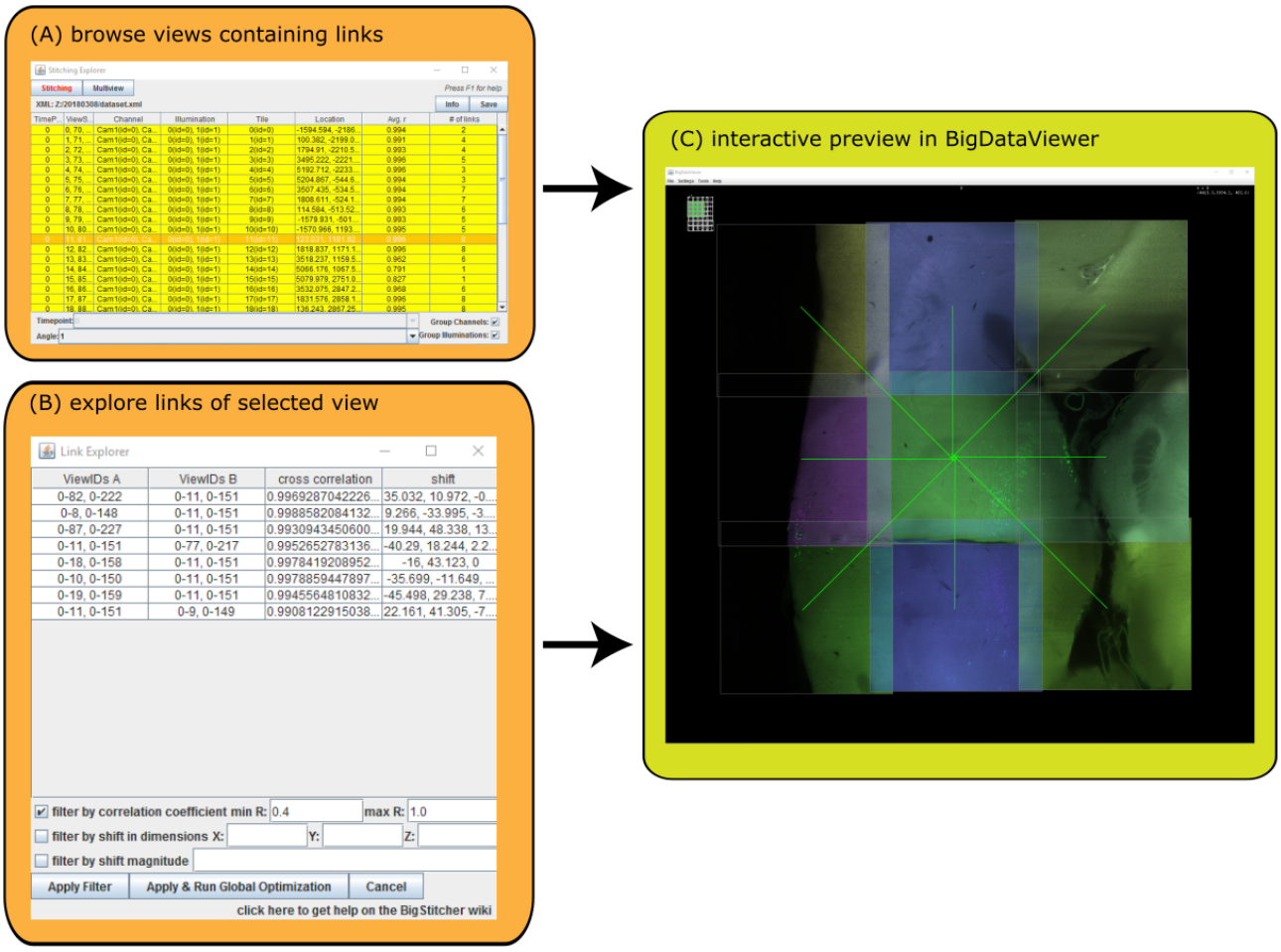
Supplementary Figure 7: *Lorem ipsum.* Dolor sit amet.

SUPPLEMENTARY FIGURE 8: Downsampling statistics 2



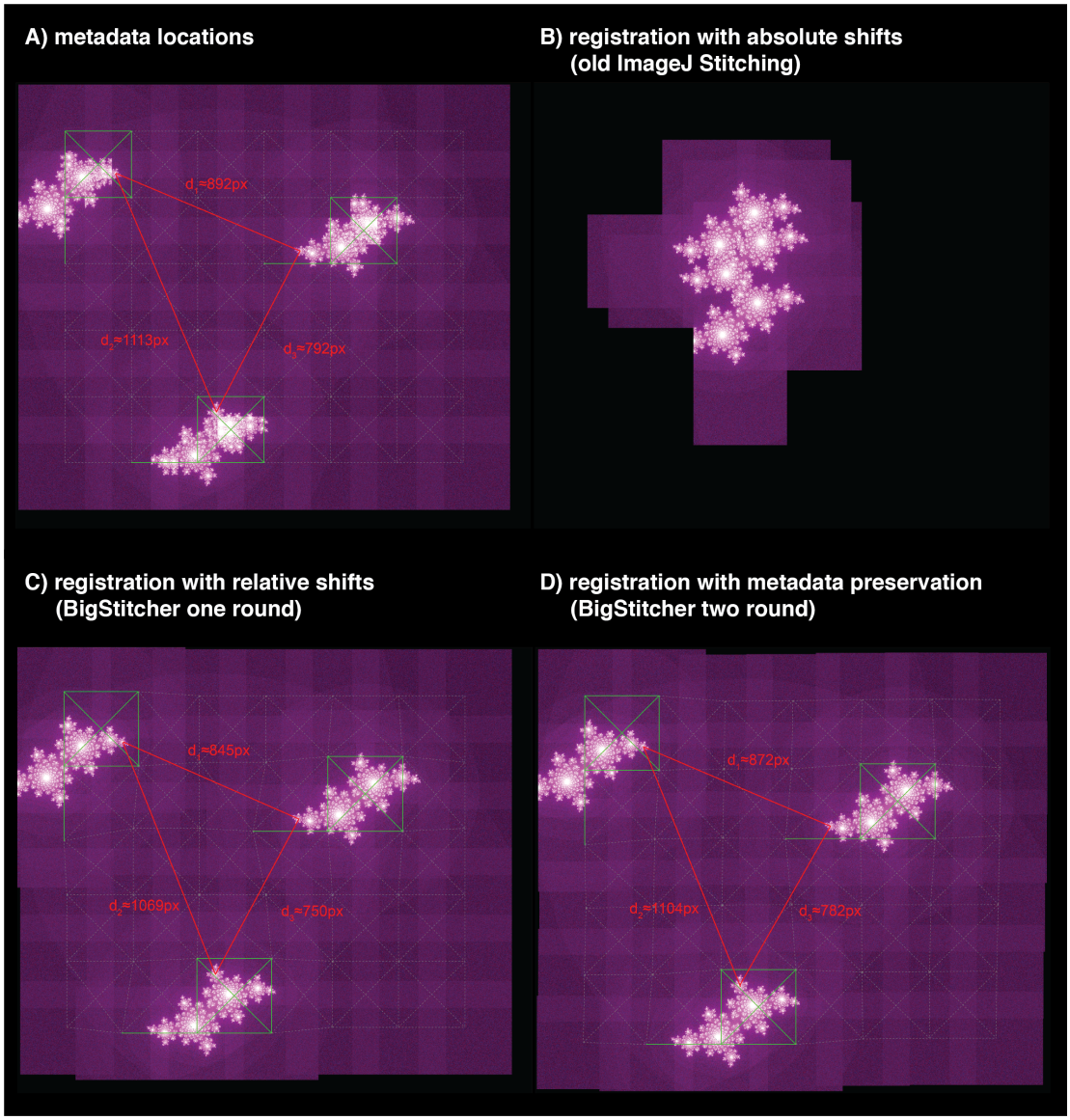
Supplementary Figure 8: *Lorem ipsum.* Dolor sit amet.

SUPPLEMENTARY FIGURE 9: Interactive inspection and curation of pairwise links



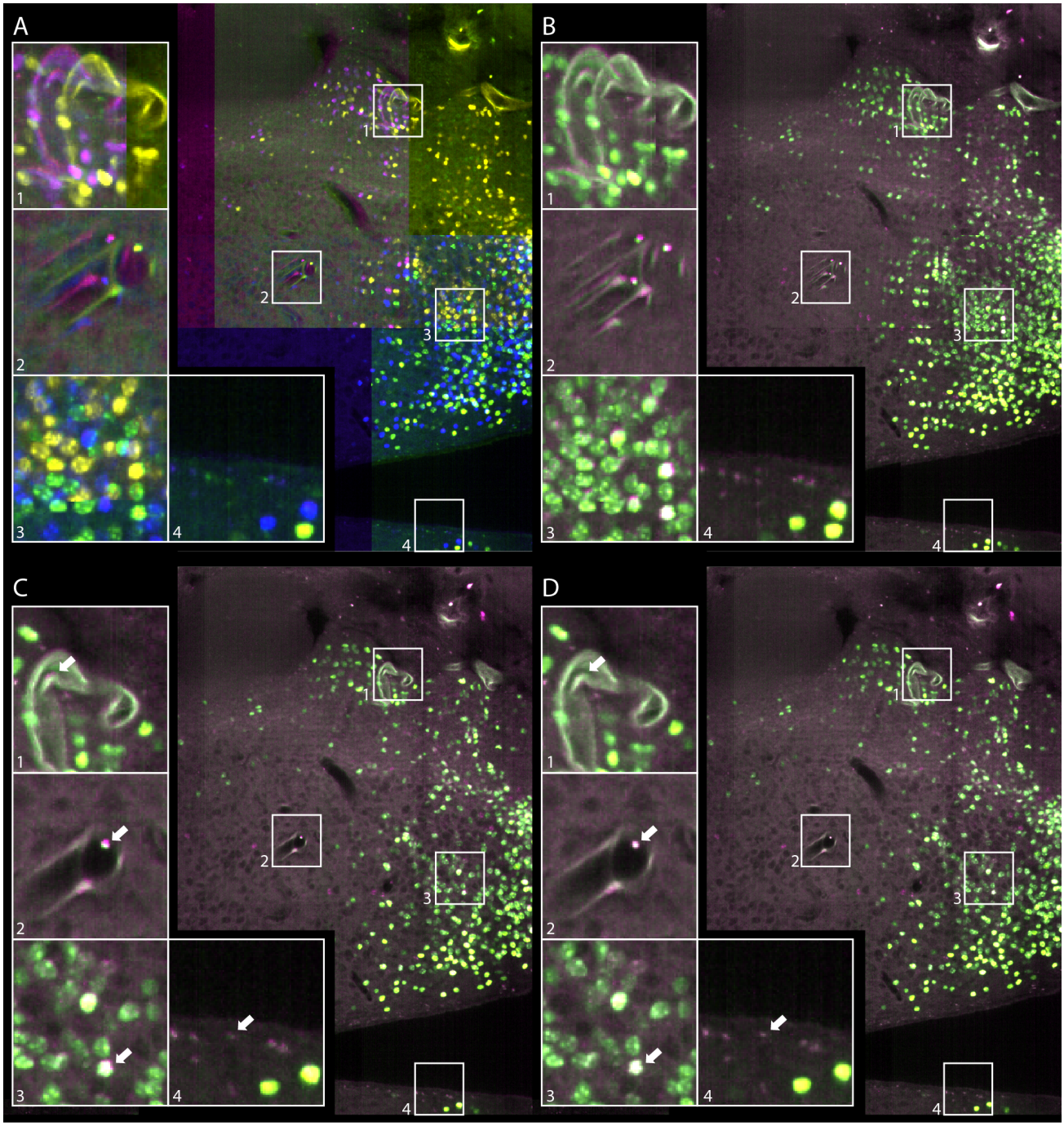
Supplementary Figure 9: *Lorem ipsum.* Dolor sit amet.

SUPPLEMENTARY FIGURE 10: Global optimization



Supplementary Figure 10: *Global optimization of pairwise registration in sparse datasets.* **(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-well-as-possible **(D)**.

SUPPLEMENTARY FIGURE 11: Affine reffinement via ICP



Supplementary Figure 11: *Lorem ipsum.* Dolor sit amet.

SUPPLEMENTARY TABLES

SUPPLEMENTARY NOTE 1: SUPPLEMENTARY METHODS

SUPPLEMENTARY NOTE 1.1: `SpimData` data format

We internally represent our image data and metadata using an extended version of the `SpimData` data format of BigDataViewer.[1] 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.

SUPPLEMENTARY NOTE 1.2: Import of data

BigStitcher imposes little constraints on the format and naming of raw data files. Using the Bioformats**?** 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.

SUPPLEMENTARY NOTE 1.3: 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.

SUPPLEMENTARY NOTE 1.4: 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.

SUPPLEMENTARY NOTE 1.4.1: Phase correlation

By default, we calculate pairwise translational shifts using an ImgLib2 re-implementation of the Fourier-based *phase correlation* algorithm.[2] In noiseless images, the method produces a phase correlation matrix Q containing a single δ-impulse at the location corresponding to the shift between the two images. Real images might contain multiple peaks, so we localize the n highest peaks in Q. By detecting peaks with subpixel accuracy using a quadratic fit.[3] Aside from allowing subpixel-accurate registration, we can also use this to counteract the effects of downsampling, allowing us to perform registration of similar quality to full-resolution with significant performance gains.

Due to the periodic nature of the Fourier shift theorem, each peak in the PCM can actually correspond to 2^d possible shifts in d dimensions. We therefore test each of these candidate shifts by calculating the cross-correlation between the images with $I2$ shifted according to the candidate shift (optionally with interpolation in the case of sub-pixel shifts). In the end, we keep the shift vector t corresponding the highest cross correlation as the final result (applying downsampling correction, if necessary).

SUPPLEMENTARY NOTE 1.4.2: 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.

SUPPLEMENTARY NOTE 1.4.3: 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.

SUPPLEMENTARY NOTE 1.4.4: 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 $R\left(\begin{smallmatrix} I & \\ 0\ \cdots\ 0 & 1 \end{smallmatrix}\right) R^{-1}$ with $R = \left(\begin{smallmatrix} A_{I_2} & b_{I_2} \\ 0\ \cdots\ 0 & 1 \end{smallmatrix}\right)$. 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 $\left(\begin{smallmatrix} I & t \\ 0\ \cdots\ 0 & 1 \end{smallmatrix}\right)$

SUPPLEMENTARY NOTE 1.4.5: 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 $\sum_{(ip_1,ip_2)\in C_{inliers}} ||ip_1 - ip_2 - t||^2$

SUPPLEMENTARY NOTE 1.5: Global optimization

SUPPLEMENTARY NOTE 1.5.1: 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: $\forall V_i \in F : R_i = I$.

SUPPLEMENTARY NOTE 1.5.2: 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).

SUPPLEMENTARY NOTE 1.5.3: 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).

SUPPLEMENTARY NOTE 1.5.4: 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: $R_i \leftarrow R^{cc}_{CC(i)} R_i^{strong}$.

SUPPLEMENTARY NOTE 1.6: 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 (1.4.5). 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 (1.5), optionally with iterative link removal and a second round to preserve metadata.

SUPPLEMENTARY NOTE 1.7: 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.

SUPPLEMENTARY NOTE 1.8: MultiView Deconvolution

In addition to image fusion, we offer deconvolution of the fused volume using a MultiView formulation of the iterative Richardson-Lucy deconvolution algorithm with Tikhonov regularization and various optimizations. The points spread functions necessary for deconvolution can be extracted from interest points detected in the images (e.g. when subdiffraction fluorescent beads were incorporated with the sample) or supplied as TIFF-stacks by the user. We offer GPU acceleration of the deconvolution on CUDA-capable Nvidia GPUs.

SUPPLEMENTARY NOTE 2: 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.

SUPPLEMENTARY NOTE 3: LINKS TO THE CURRENT SOURCE CODES

BigStitcher can be seen as an extension to the multiview-reconstruction project, but over time, many changes have been pushed upstream and the two projects have evolved in parallel. Both projects 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 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 (2) will be hosted on the same wiki page (http://imagej.net/BigStitcher#Documentation).

REFERENCES

1. 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.
2. S. Preibisch, S. Saalfeld, and P. Tomancak, "Globally optimal stitching of tiled 3d microscopic image acquisitions," *Bioinformatics* **25**(11), pp. 1463–1465, 2009.
3. D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision* **60**(2), pp. 91–110, 2004.
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.