Parismi can be run in interactive mode, as an ImageJ plugin, or in batch mode.

Interactive mode

Open `Parismi/IJ/ij.jar` and select `A0PipelineManager` from the plugin menu. To load a pre-existing pipeline (which can be a good way of exploring Parismi's functionality), either click the "Load table" button and select an `.xml` file from our dataset release, or drag such a file and drop it onto the Parismi control panel.

Overall pipeline structure

Each step in the pipeline corresponds to one row in the GUI table. Table columns include the following: - *Primary image input*. A right click in the field provides a popup menu to assist with input selection. The input can be a file on disk (specified with a full path name, in which `~/` or `~username/` can be used as shorthand for user home directories, or a path name relative to the current working directory), an image opened through ImageJ and specified by its name, or the output of another plugin. In the latter case, the reference to the other plugin is given as number specifying the row offset (i.e. -1 to specify the output of the preceding plugin), or as a number prefixed by `$` to specify an absolute row in the table. Parismi can natively read TIFFs with 32-bit or 64-bit offsets and LSM files. It can transparently decompress GNU zipped files that it recognizes with the `.gz` extension. Large image files can be opened as "virtual stacks", which means their contents are initially not fully loaded into memory (this is done by default for files over 2 GB in size; this behavior can be manually adjusted with the "Use virtual stacks to open files" checkbox). Note that some (but not all) plugins will trigger a read of the full contents when they are given such an input.

- *Channels*. For multiple channel images, a subset of channels to which to apply the plugin can be specified by selecting from the list provided.
- *Plugin name*. A right click in the field provides a hierarchical popup menu of all plugins that were loaded (note that deprecated plugin can be loaded from pre-existing `.xml` files but are not offered for loading through the GUI). Hovering over a plugin name in the popup menu displays a tooltip that gives a brief overview of the plugin.
- *Plugin parameters*. These parameters are plugin-specific and appear once a plugin has been selected. For numerical values or ranges, the range of values covered by the sliders can be manually adjusted (the range is saved along with the selected values to facilitate adjustment of reloaded tables). Values can be adjusted by typing in a new number, by using the sliders, or by using the mouse scrollwheel while over the slider. Any change leads by default to coalesced live updates of the pipeline (this behavior can be adjusted; see below). Hovering over parameter names displays a tooltip that offers a brief description of the parameter.
- *Auxiliary inputs*. Some plugins require multiple inputs (e.g. a set of active contour seeds and an image on which to run active contours). The inputs are specified in the "Auxiliary inputs" column, following the same format as the primary image input column.
- *Auxiliary outputs*. For plugins that have multiple outputs, this column provides a list of names, which Parismi will use to match inputs and outputs of plugins that do not follow the one input - one output model.
- *Show*. GUI views of plugin outputs are created by default for most plugins run in GUI mode. This behavior can be adjusted on a plugin-by-plugin basis using the "Show" checkbox. Note that all the outputs of a plugin can be brought to the foreground by selecting the plugin in the table and clicking the "Bring outputs to front" button.
- *Autorange*. When this checkbox is selected, the output image display range is automatically adjusted to match the minimal and maximal pixel values.
- *On*. If this checkbox is not selected, the plugin is skipped when the pipeline is run.
- *C+*. This option will be documented at a later time and can be left to its default value.
- The *progress bar* shows progression of the plugin as it computes its output. Note that some plugins provide more fine-grained progress reports than others.

GUI views for Parismi data structures

GUI views of plugin outputs include an extended ImageJ image window for image outputs, and a table view for cells.

- *Image windows* are based on a standard ImageJ display, to which a set of controls and user interaction functionalities are added. The user can perform *de novo* cell annotations, or edits of pre-existing annotations that can be derived from automatic detection, previous user edits, or any combination thereof. The annotations include cell position, as well as any combination of user-defined labels, e.g. to denote manually-determined cell types. The annotations are overlaid with the image; their graphical display parameters (thickness, diameter, transparency, and color) can be manually adjusted to provide the most useful display for the particular kind of image that is used. The position of annotations can be adjusted by dragging. All annotations in a rectangular region can be deleted in one step by selecting delete mode and dragging the mouse from one corner of the rectangle to the opposite corner. Hovering over an annotation for a user-definable delay produces a table listing all information for that annotation (the table can be preserved by clicking on it; if not it disappears when moving the mouse away from the annotation). Displayed sections can be "thickened" by projecting on an interval centered around the slice in view. The image display can be adjusted in the following ways:
    - The image can be switched back and forth between the "Composite" mode (in which multiple channels are drawn together, with a different color for each) or the "Multiple channel" model with the "Make composite" button in the main Parismi window.
    - Orthogonal views (on which annotations also appear and can be manipulated) can be toggled on or off with the "Orthogonal" button.
    - By default the scrollwheel changes the displayed z slice (in the main xy view), or the displayed x or y slices (in the yz and xy orthogonal views, respectively). For fine stepping through slices, one can press the x, y, or z keys.
    - The contrast can be adjusted by using the mouse scrollwheel while pressing shift
    - One can scroll through the channels by using the mouse scrollwheel while pressing option
    - Option-click is a shortcut to add new cells without manually selecting the corresponding tool. Shift-click is a shortcut to delete.
- *Cell tables*. These tables give a spreadsheet-like presentation of all the fields in cell objects, which can be the raw cell detections (made by the cell detector and/or by users), or the detections on which various plugins have been run to add new fields (e.g. fluorescence contents). Some features are as follows:
    - Interactive histograms are displayed for each column. A range can be selected by clicking in these histograms, which will cause filtering of the displayed cells.
    - Cells can be sorted on the value of any field by clicking on the column header.
    - A scatterplot of two columns can be obtained by selecting cells in any two columns and clicking "Scatter plot from selected columns". In the resulting graph window, the x and y axes can be switched an a trendline with user-adjustable local averaging window can be displayed. If cells were selected from a single column, a histogram is displayed instead and standard statistics reported.
    - Basic spreadsheet-like computing functionality is included, which is not fully implemented and will be documented at a later time.
    - The results can be exported to a tab-delimited text file (this is for user convenience, and is of course not required for batch operation).
    - Note that the segmentation coordinates, which normally comprise a very large set of numbers, are hidden from the table view. Segmentations can be displayed from cell/seed objects by running e.g. the `DisplaySolidSegmentation` plugin, which produces an image output in which the individual cells can be colored following a user-selected field.

Running pipelines

- A full run of the pipeline can be initiated by clicking the "Run" button.
- A run of the pipeline starting from one particular point can be initiated by selecting a row in the table and clicking the "Update step" button. If the "Update pipeline upon param change" checkbox is selected, all plugins following the selected one will also be run.
- Any change to a plugin parameter will by default cause the plugin to recompute its output (this behavior can be adjusted with the "Update current step upon param change" checkbox), which will by default in turn cause the next plugins to be also re-run. If a second parameter change occurs before the re-computing triggered by a first change has completed, the update for the second change can either be put in a coalescing queue or cancel the re-computation triggered by the first change.
- A pipeline run can be interrupted at any time by clicking the "Stop all updates" button. Note that plugins differ in how quickly they respond to that button.

Semi-automated functionality

Pipelines often comprise steps for which there is no need for user interaction, and steps at which it is desirable for a user to review the results and adjust them if need be. Parismi can be paused at key steps using the `Pause` plugin; execution can be resumed after user edits by selecting the appropriate row in the pipeline table and clicking "Run". To avoid users spending time moving from one dataset to another, one can use the "Open next and run" button. This button closes all current plugin outputs, and goes through all input and output file names to increment any numbers that occur between braces. If datasets are structured in numbered directories, naming files following e.g. `~username/datasets/{1}/input_1.tiff'` will make it possible to iterate through all the directories without manual setup at each dataset change. Note that more sophisticated batch functionality, which gives more flexibility in terms of file naming and directory structure, can be achieved using the `BatchOpenV2` plugin.

File formats

The file formats that Parismi can read and write are as follows:

- *Images*. Parismi can natively read 32-bit-offset Tagged Image File Format (i.e. "regular" TIFF) or 64-bit-offset TIFF (i.e. BigTIFF); the latter removes the 2GB or 4GB (depending on implementation) limit on file size. Parismi also natively reads the proprietary Zeiss LSM file format (which is a TIFF variant). Note that other file types can be opened through ImageJ, in particular with the LOCI Bio-Formats plugin, which can read a broad array of file types. By default Parismi writes BigTIFF with ImageJ metadata to specify channels, although for ease of browsing of output images (since BigTIFF is not always supported) it can also rely on ImageJ to output regular TIFF images, in which channels can be displayed as an RGB overlay.
- *Cell annotations and segmentations*. All data pertaining to automatic cell detections, user detections and edits, annotations, segmentations, and quantification results are stored in a unified format. Parismi relies on Google's `Protobuf` file format, with fixed fields including cell position and x, y, and z coordinates of all pixels in the segmentation mask, as well as optional fields specifying annotations and analysis results. Parismi reads and writes corresponding files with the `.proto` extension. To facilitate data exchange with other programs, Protobuf objects can be exported to tab-delimited text files (without the segmentation mask) either manually or through a plugin. Segmentation masks can also be exported as a single TIFF in which each cell receives a pixel value matching its index. Such a file can easily be reused in another program such as Matlab.

Notes on saved pipelines

We normally include in each of our pipelines a plugin to save the pipeline itself in the dataset directory in which it was run. This allows any interactive user edits that were made to plugin parameters to be permanently recorded. As plugins evolve, we often find that we need to add new parameters. To maximize backward compatibility, the programmer can either transparently specify default values to be given to plugin parameters that did not exist in earlier pipelines, or specify that an error should be thrown so manual review must occur.

Batch mode

Batch runs can be performed with `java -jar A0PipeLine_Manager.jar batch path_to_pipeline.xml`. This matches the semi-automated functionality described above, except that pauses for user review are skipped, and no GUI windows are open (for a command-line run without batch functionality, replace `batch` by `singleRun`). When using long pipelines with adjustments that may need to be made at select steps, it may be a good idea to split the pipelines in smaller pieces and use `make` to handle updates. This is what we have done for the datasets we have released.